Analyse des données LIDAR issues de la base nuScenes. Qualification (semi-)automatique de l'analyse des interactions entre véhicules

Tiffany Darini

Travail de fin d'études – Mémoire de master

pour le diplôme d'ingénieur de l'École nationale des travaux publics de l'État

Année 2020-2021

Voie d'approfondissement :
Ingénierie de la mobilité

Soutenu le 03 septembre 2021

Devant le jury de TFE composé de :

- Président : Christine Buisson
- Expert : Fouad Baouche
- Tuteurs : Pierre-Antoine Laharotte et Andres Ladino

Devant le jury de master composé de :

- Président : Christine Buisson
- Tuteurs : Pierre-Antoine Laharotte et Andres Ladino
- Expert : Fouad Baouche
- Rapporteur : Delphine Lejri

Par

Tiffany DARINI

Analyse des données LIDAR issues de la base nuScenes : qualification (semi-)automatique de l'analyse des interactions entre véhicules

Organisme d'accueil :

Laboratoire d'Ingénierie Circulation Transport (LICIT)

Notice analytique

AUTEUR			
Nom	DARINI		
Prénom	Tiffany		
ORGANISME D'ACCUEIL			
Nom de l'organisme et localité	Laboratoire d'Ingénierie Circulation Transport (LICIT) 25 Avenue François Mitterrand – Case 24, 69675 Bron cedex		
Nom des Tuteurs	Pierre-Antoine Laharotte Andres Ladino		
ANALYSE DU TFE			
Titre (français)	Analyse des données LIDAR issues de la base nuScenes : qualification (semi-)automatique de l'analyse des interactions entre véhicules		
Titre (anglais)	Analysis of LIDAR data from the nuScenes database: (semi-)automatic qualification of vehicle interaction analysis		
Résumé (français)	Les véhicules autonomes représentent aujourd'hui une potentielle et nouvelle source d'informations dans la compréhension et l'étude du trafic, que ce soit en temps réel ou à posteriori. Ces données très riches dans le détail de l'interaction et de plus en plus nombreuses sont pourtant peu exploitées dans le domaine du trafic routier. Pour le moment, elles sont principalement utilisées dans la problématique de perception de l'environnement par le véhicule autonome. Les travaux menés au cours de cette étude consistent à déterminer les informations que peuvent apporter les bases de données de véhicules autonomes et ce à partir de la base nuScenes. NuScenes a été créée à partir d'expériences réalisées grâce à des véhicules autonomes en conditions réelles de circulation. Ce travail vise, dans un premier temps, à caractériser les interactions entre deux véhicules telles que restituées par le passage d'un véhicule autonome, puis, dans un second temps, à regrouper de façon non supervisée les interactions observées et à les comparer à des catégories identifiées manuellement. L'objectif sous-jacent est, à termes, de parvenir à identifier automatiquement les interactions entre deux véhicules se référant à un comportement de type véhicule-suiveur et de quantifier les différences de comportements entre conduite humaine et automatisée.		
Résumé (anglais)	Nowadays, autonomous vehicles represent a potential new source of information in the understanding and study of traffic, whether in real time or after the fact. This data, which is very rich in interaction details and increasingly numerous, is nevertheless little exploited in the field of road traffic. For the moment, they are mainly used in the problem of the perception of the environment by the autonomous vehicle. The work made during this study consists in determining the information an autonomous vehicle database can provide based on the nuScenes database. NuScenes was created based on experiments with autonomous vehicles in real traffic conditions. This work aims, firstly, to characterize the interactions between two vehicles as rendered by the passage of an autonomous vehicle and, secondly, to group the observed interactions in an unsupervised way and to compare them to manually identified categories. The underlying objective is, in the long term, to be able to automatically identify the interactions between two vehicles referring to a vehicle-follower type of behaviour and to quantify the differences in behaviour between human and automated driving.		
Mots-clés (français)	Véhicules autonomes, Base de données, Interactions, Regroupement, Classification		
Mots-clés (anglais)	Autonomous vehicles, Database, Interactions, Clustering, Classification		
Termes géographique (français)	Boston, Singapour		
COLLATION			
	Nb de pages	Nb d'annexes (nb de pages)	Nb de réf. biblio
	86	11 (12 pages)	21

Remerciements

Je tiens pour commencer à remercier mes tuteurs Pierre-Antoine Laharotte et Andres Ladino, tous deux chercheurs au Laboratoire d'Ingénierie Circulation Transport (LICIT), pour leur accompagnement tout au long du stage. Ils ont su me guider pendant toute cette période de recherche. Ils ont aussi été d'une grande aide sur certains points techniques et sur l'identification de points d'avancement délicats. Je les remercie aussi pour leur disponibilité malgré les difficultés liées au télétravail et pour leurs précieux conseils pour la rédaction du mémoire.

Je remercie ensuite les membres du LICIT pour leur chaleureux accueil sur le site de Bron les jours où il était possible de venir.

J'adresse également mes remerciements aux membres du jury Christine Buisson, Fouad Baouche et Delphine Lejri pour le temps qu'ils auront consacré à l'évaluation de mon travail.

Pour finir, je remercie mes camarades de la D113 avec lesquels j'ai travaillé à l'ENTPE les jours de télétravail. Ils ont été d'un grand soutien autant moral que technique et ont permis de rendre ces journées plus agréables.

Table des matières

Table des illustrations

Introduction

Depuis quelques années un nouveau mode de transport se développe : les véhicules autonomes. Ces véhicules qui ne nécessitent pas d'intervention humaine pour circuler commencent petit à petit à entrer dans notre quotidien. Ce développement s'est fait en plusieurs étapes : cela a commencé par une automatisation progressive de l'automobile à partir de capteurs et de systèmes d'aides à la conduite tels que l'aide au stationnement ou le régulateur de vitesse. [A] Cela ayant pour but d'améliorer la perception du véhicule. Les étapes suivantes ont nécessité et nécessitent encore de multiples démarches puisque le véhicule autonome suscite des contraintes technologiques, des contraintes de réglementation et de sécurité mais aussi des contraintes d'acceptation par les futurs utilisateurs.

Les véhicules autonomes font maintenant partie intégrante du développement de nos mobilités puisqu'ils font l'objet d'expérimentations multiples partout dans le monde. En France, la loi « PACTE » de 2018 fournit un cadre pour leur développement : l'objectif est de permettre, d'ici 2022, la circulation de véhicules autonomes de niveau 4 c'est-à-dire avec une autonomie élevée et permettant l'inattention du conducteur. [B] De plus, des orientations stratégiques ont été fixées : elles donnent un cadre législatif et réglementaire, les enjeux liés aux véhicules autonomes notamment pour des aspects de sécurité et de cyber-sécurité…

Leur développement pourrait permettre d'améliorer certains services de mobilité notamment les transports collectifs. En effet, les bus pourraient être remplacés par des bus autonomes ce qui permettrait de faire circuler les bus avec des fréquences plus élevées sur de plus grandes amplitudes horaires et dans des zones de dessertes à faible densité. Cela pourrait inciter davantage de personnes à abandonner leurs véhicules personnels pour se tourner vers les transports collectifs qui seront des véhicules moins polluants et donc cela pourrait entraîner une diminution des émissions de $C0_2$. Ce phénomène pourrait également, au contraire créer une augmentation des émissions de $C0_2$ en créant une hausse des mobilités. [A]

Si les véhicules autonomes possèdent de grands avantages sociologiques et environnementaux, ils peuvent aussi être potentiellement des sources de données, et c'est cette perspective que nous allons explorer. Ils sont équipés de capteurs embarqués qui donnent accès à de nombreuses données notamment sur les trajectoires des véhicules, leur environnement… Le véhicule autonome constitue lui-même un capteur en temps réel qui peut permettre de mesurer des conditions de trafic très précises. Il apporte de nouvelles informations avec un grand niveau de précision sur l'interaction avec son

environnement, des informations auxquelles nous n'avions pas accès jusqu'ici. En effet, les mesures de trafic faites jusqu'à présent reposaient en majorité sur des boucles de comptage or pour réaliser ce type de mesure, il est nécessaire de déployer des infrastructures dédiées et d'investir souvent en tout point du réseau. Il est donc utile de s'intéresser davantage à ces données et de les étudier dans la perspective d'améliorer nos connaissances sur le trafic en temps réel.

C'est dans ce contexte que plusieurs entreprises ont décidé de créer des bases de données obtenues à l'aide de véhicules autonomes. [1] [2] [3] Ces bases de données sont créées à partir d'expérimentations qui ont pour premier objectif de favoriser le développement de l'automatisation des fonctionnalités de conduite mais qui par la même occasion permettent la création de nombreuses données. Elles contiennent des informations sur le mouvement du véhicule autonome, sur son environnement et donc aussi des données sur des véhicules pilotés par des humains. Elles sont nombreuses et se distinguent par les types de capteurs utilisés et leur façon de créer les bases de données. Parmi elles, on peut citer la base de données KITTI créée en 2013 [1]. Elle a été réalisée à partir d'un véhicule équipé de caméras, d'un laser et d'un système GPS ce qui permet d'obtenir des données en images, des positions… Les objets observés dans les images ont été annotés et placés dans des boîtes 3D… Plus récemment, les bases de données Argoverse [2] et Waymo Open Dataset [3] ont été publiées. Argoverse est une version supérieure de la base KITTI puisque les données ont été récupérées avec les mêmes types de capteurs couplés à un LIDAR (Light Detection and Ranging), permettant d'obtenir des informations plus précises. De plus, cette base de données produit également des informations sur les lieux où circulent les véhicules : quel type de route, quelle est la limite de vitesse sur le lieu de circulation… Quant à Waymo Open Dataset, la production de la base de données est semblable à celle de Argoverse : mêmes types de capteurs et de données recueillies. Ils ont tenté d'obtenir des données dans différents types d'environnement pour pouvoir généraliser leurs résultats à plusieurs endroits.

On peut donc s'interroger :

Peut-on utiliser les données issues de capteurs embarqués pour affiner la compréhension et la modélisation du trafic ?

Nous proposons d'utiliser les données de la base nuScenes pour créer un processus permettant d'identifier automatiquement les interactions entre véhicules. Ce document présente les résultats des recherches faites sur ce sujet et s'articule en quatre grandes parties. La première partie présente la base de données nuScenes et le début de son exploration. La seconde partie met en évidence les méthodes que nous avons choisies pour caractériser les interactions des véhicules. La troisième partie montre les pistes que nous avons explorées pour classifier ces interactions et enfin la quatrième partie revient sur les résultats obtenus et sur les pistes à explorer par la suite.

Partie 1 : Le véhicule autonome et la base de données nuScenes

I. La base de données nuScenes

1. Création de la base

La base de données nuScenes a été créée en 2020 par des ingénieurs de Motional, société spécialisée dans la recherche et l'innovation sur les véhicules autonomes. Elle est inspirée de la base de données KITTI mais nuScenes contient sept fois plus d'annotation d'objets.[4]

Cette base contient 1000 scènes de 20 secondes, extraites des expériences réalisées dans les villes de Boston et Singapour à l'aide de Renault Zoé équipées de plusieurs capteurs embarqués. Plusieurs types de capteurs sont utilisés afin d'obtenir un maximum d'informations :

- Les caméras : elles permettent d'observer les bords des objets, leurs couleurs et l'éclairement afin de les localiser sur le plan de l'image. Elles permettent également de faire la distinction entre chaque catégorie d'objet présent dans l'environnement du véhicule. Cependant il est difficile de faire de la localisation 3D à partir de caméras, d'où l'utilisation d'autres technologies.
- Les radars : ils atteignent entre 200m et 300m autour du véhicule et permettent d'obtenir des données de distance.
- Le lidar : il permet également de mesurer des distances pour des objets situés à des distances de 50 à 150m du véhicule autonome. Le lidar fonctionne de la même façon qu'un radar mais il utilise la lumière et non les ondes radio. Afin de ne pas perdre la localisation du véhicule en utilisant des données GPS, on récupère la localisation du véhicule autonome à partir des informations lidars.

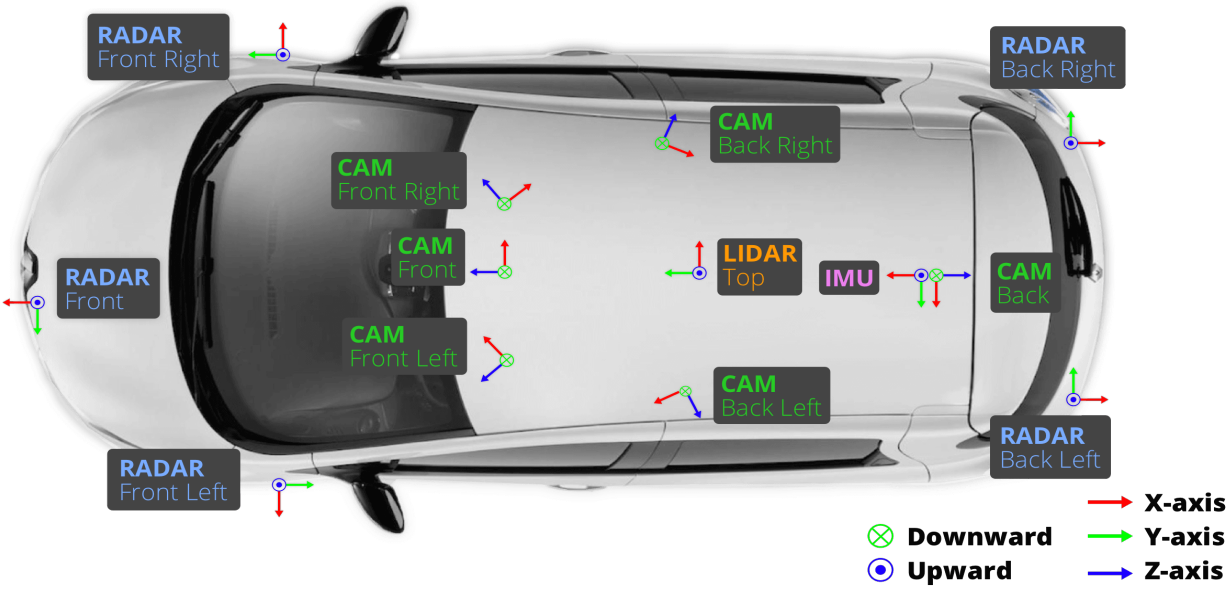


Figure 1 - Renault Zoé utilisée pour tourner les scènes

(Source : nuscenes.org)

Avec ces trois sources de données qui ont chacune des forces différentes, comme on peut le voir sur la figure 2, on obtient une description très précise de l'environnement du véhicule autonome. L'usage de données multi-sources est essentiel pour assurer une qualité d'information en continu quelles que soient les conditions météorologiques.

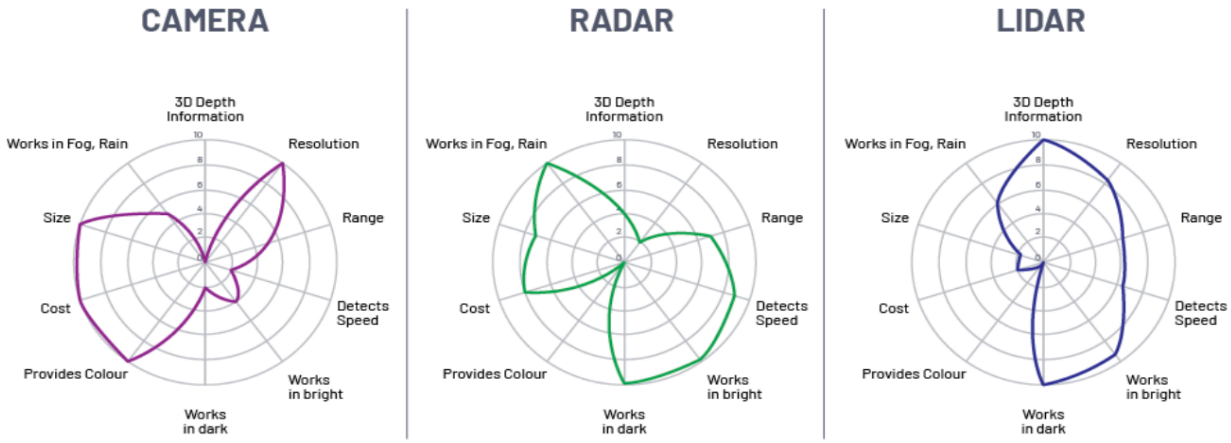


Figure 2 – Comparaison des trois sources de données

2. Réalisation des expérimentations et extraction des données

L'élaboration de la base de données s'est faite en plusieurs étapes

Tout d'abord il a fallu définir les trajets à réaliser : les images sont réalisées dans deux villes avec des trafics denses. Ils ont tenté d'utiliser des itinéraires permettant de parcourir un maximum de lieux différents en termes de végétations, de bâtiments, de marquage au sol… De plus, les itinéraires réalisés sont les chemins idéalisés qu'un véhicule autonome devrait prendre, en supposant qu'il n'y a pas d'obstacle.

Après la réalisation de ces scènes, il a fallu sélectionner celles qui étaient intéressantes. Ils en ont sélectionné 1000 manuellement, celle-ci ayant des éléments particuliers : des objets rares, des accidents, des manœuvres, des conditions météorologiques particulières… Toutes ces scènes ont ensuite été annotées tout en protégeant la vie privée des individus visibles sur les vidéos.

NuScenes prend en charge plusieurs tâches dont notamment la détection et le suivi. La qualité et la précision de la détection sont mesurées à l'aide d'indicateurs. La tâche de suivi consiste à suivre chaque objet détecté dans une scène. Les objets détectés sont placés dans des boîtes 3D dont on connaît la position du centre au cours de la période de détection, sa rotation…Ils sont classés à la main dans différentes catégories sémantiques, il y en a 23 au total :

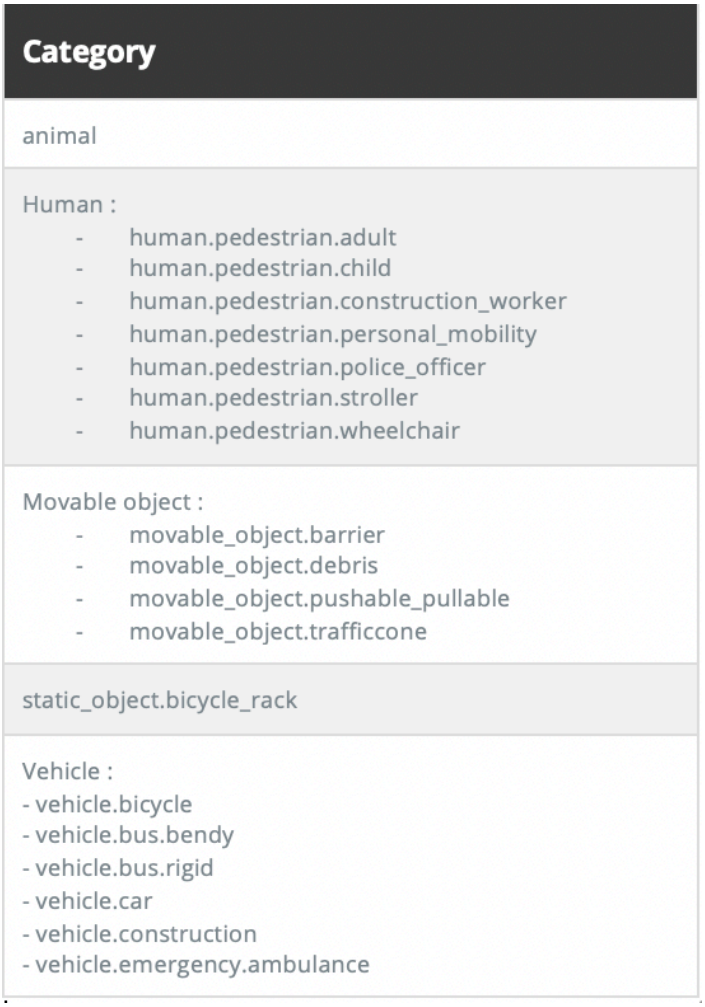


Figure 3 – Catégories sémantiques de la base NuScenes

(Données extraites de nuscenes.org)

La base nuScenes présente également depuis juillet 2020, une extension appelée nuScenes-lidarseg dans laquelle il y a 32 catégories sémantiques, les 9 rajoutées concernant plutôt l'environnement du véhicule c'est-à-dire les trottoirs, la végétation…

L'ensemble de ces données est contenu dans la base.

3. La structure de la base

La base contient de nombreuses informations, il a donc fallu les ordonner d'une certaine façon pour pouvoir facilement les utiliser. Les concepteurs ont choisi de classer les informations dans des tables, chacune d'entre elles ayant une clé qui permet de croiser des informations entre les tables. Une clé permet d'identifier de façon unique un enregistrement dans chacune des tables. Il peut être difficile de passer d'une table à une autre puisqu'elles ne sont pas toutes liées entre elles. C'est une base de données type SQL.

Les auteurs ont choisi de diviser les tables en 4 parties, chacune d'entre elle comprenant plusieurs tables :

- 'Vehicle' : cela concerne les données sur la carte, le véhicule autonome et les capteurs utilisés…
- 'Extraction' : cette partie est liée aux données la scène et les positions du véhicule autonome. Chaque scène est découpée en échantillons de 0,5 secondes, on retrouve ces informations dans cette partie.
- 'Annotation' : Les objets étant annotés et placés dans des catégories sémantiques, c'est dans cette partie que se trouvent ces données.
- 'Taxonomy' : on retrouve ici des informations sur l'état des objets, leur bonne détection en termes de visibilité…

Ci-dessous le schéma explicatif des tables de la base, les flèches représentent les tables qui sont liées entre elles à l'aide de clés :

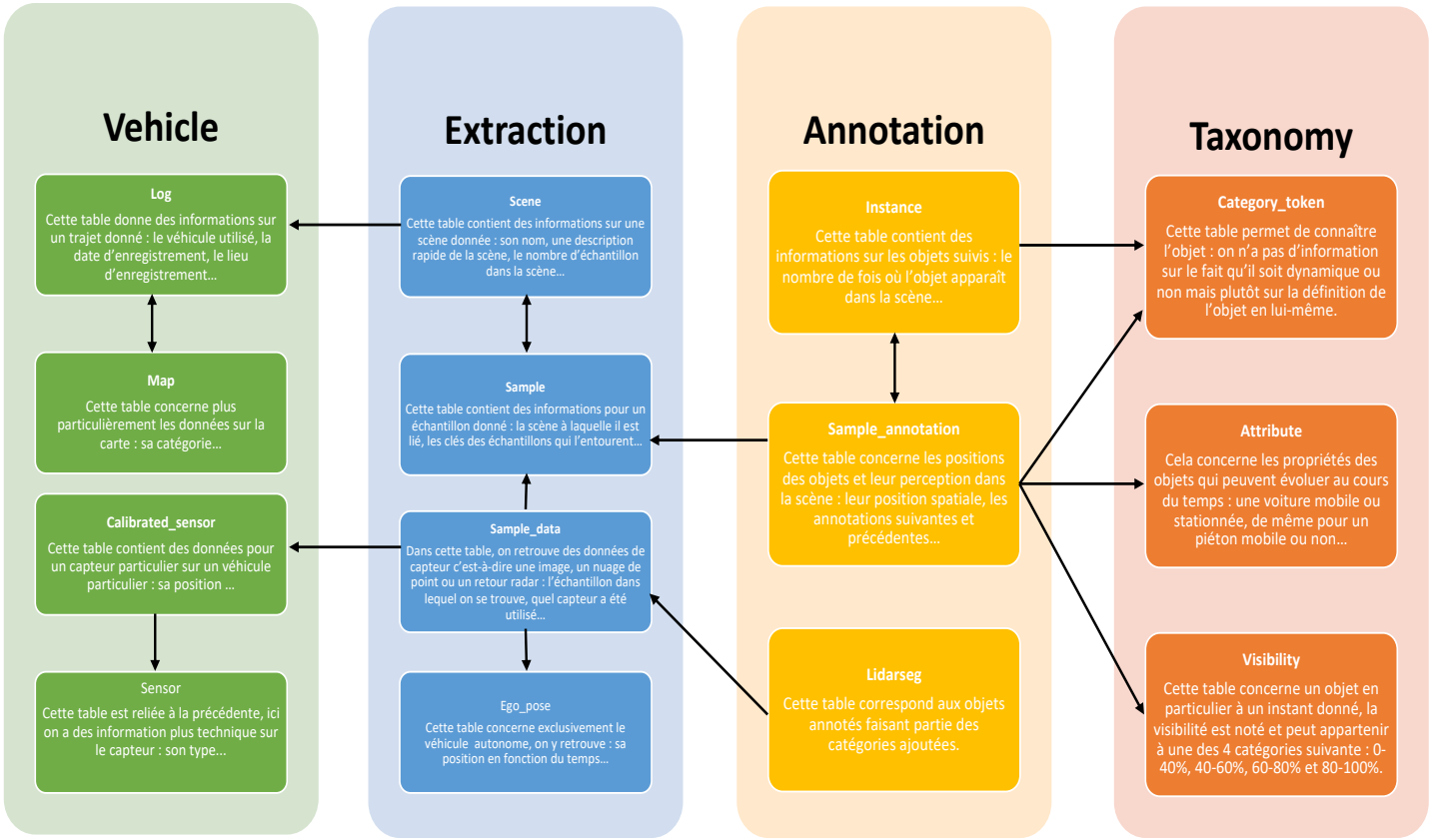


Figure 4 - Structure de la base

II. Exploration de la base

La base de données étant très grande, j'ai commencé par travailler sur un sous-ensemble de 10 scènes proposé par nuScenes et nommé 'Mini'.

1. Trajectoires du véhicule ego

Grâce aux données LIDAR, nous avons à notre disposition, pour une scène donnée, la position du véhicule autonome, aussi appelé véhicule ego, au cours du temps. On commence donc par s'intéresser à la trajectoire du véhicule ego pour une scène donnée. Connaissant la position du véhicule au cours du temps, on peut également tracer sa trajectoire dans un diagramme espace-temps, celui-ci nous sera utile par la suite pour étudier le profil de vitesses du véhicule.

Dans les tables nuScenes, on retrouve également des fonctions préétablies permettant de réaliser différentes actions. Une des fonctions, 'get_poses', permet en particulier de récupérer la liste des éléments de la table 'ego_pose' correspondant à une scène donnée, c'est-à-dire toutes les positions du véhicule ego au cours du temps dans la scène choisie. On se sert de cette fonction pour tracer les diagrammes suivants :

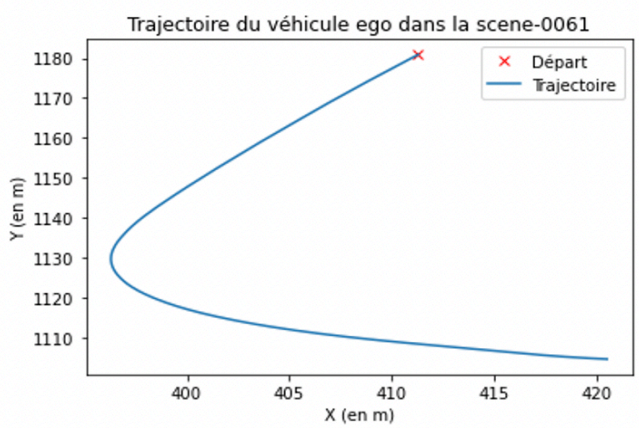


Figure 5 - Trajectoire du véhicule ego

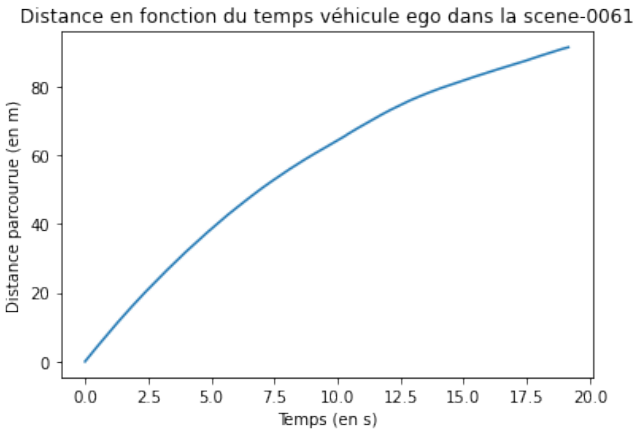


Figure 6 - Trajectoire du véhicule ego dans le diagramme espace-temps

Les informations à disposition concernent la trajectoire des véhicules, mais sont également contextualisées. Il est ainsi possible de tracer la trajectoire en fonction du type de lieu traversé par le véhicule. L'analyse de la Figure 8 met en évidence l'interaction entre le contexte et la trajectoire adoptée par le véhicule. On remarque que dans cette scène, le véhicule a circulé sur un passage piéton, cependant sa vitesse n'a pas diminué fortement, il ne s'est donc pas arrêté pour laisser passer un piéton.

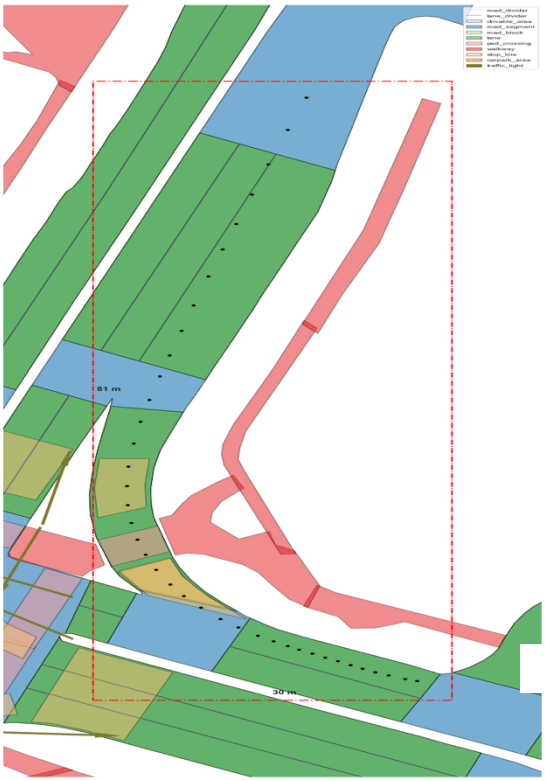


Figure 8 - Trajectoire du véhicule ego sur la carte

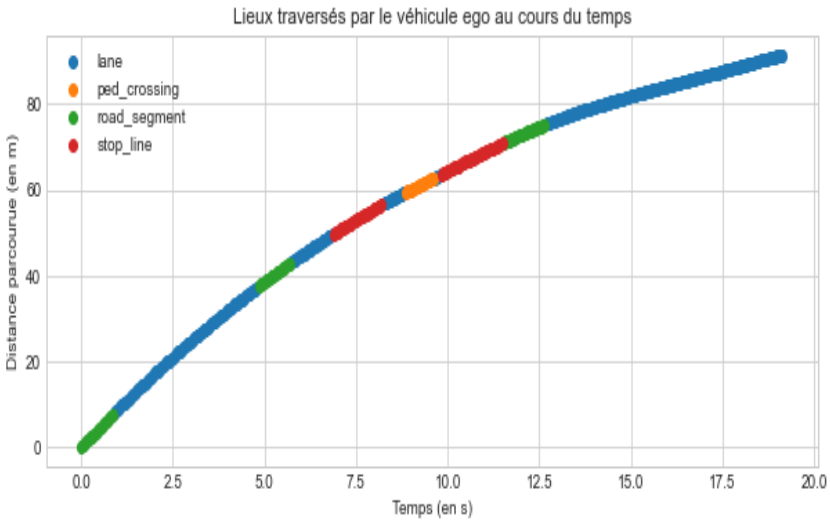


Figure 7 - Lieux traversés par le véhicule dans la scène

2. Trajectoires des éléments qui entourent le véhicule

L'interaction avec les objets qui entourent le véhicule autonome est un facteur très important à étudier. En effet, l'environnement du véhicule et son interaction avec celui-ci peut impacter le comportement du véhicule autonome et par conséquent sa trajectoire. J'ai commencé par tracer les trajectoires de ces objets.

2.1 Dans l'espace

Dans les tables, on ne retrouve pas directement la position des éléments d'une scène donnée, il faut passer entre plusieurs tables pour les récupérer. Voici le cheminement :

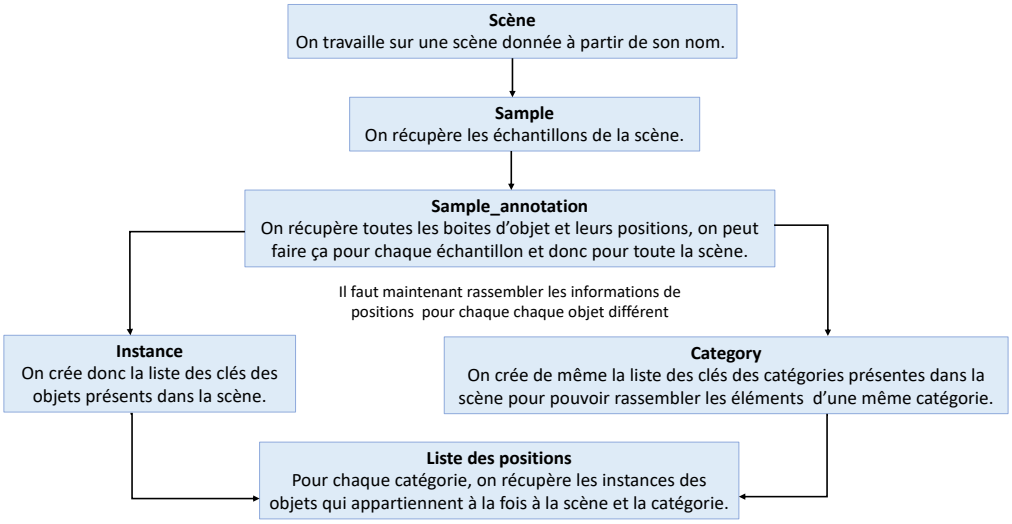


Figure 9 - Obtenir la position des objets d'une scènes

À partir de ce cheminement, on obtient pour chaque catégorie, une liste contenant n listes, n étant le nombre d'objet de la catégorie et chaque liste correspond aux positions d'un objet de la catégorie au cours du temps dans la scène.

Certains objets d'une scène sont détectés plusieurs fois par les capteurs à des positions très proches : ils sont immobiles mais la précision des capteurs entraîne de légères différences sur chaque mesure de position. J'ai donc défini une fonction qui permet de séparer les objets d'une catégorie qui sont mobiles et ceux qui sont immobiles (cf Annexe 1). On considère qu'un objet est immobile lorsque la distance totale qu'il parcourt pendant son temps de détection est inférieure à deux mètres. Cette fonction est utile pour représenter la trajectoire de l'ensemble des éléments de la scène dans l'espace puisque grâce à cela, on représente les objets immobiles par une unique croix (cf Annexe 2). On obtient des diagrammes tels que celui-ci :

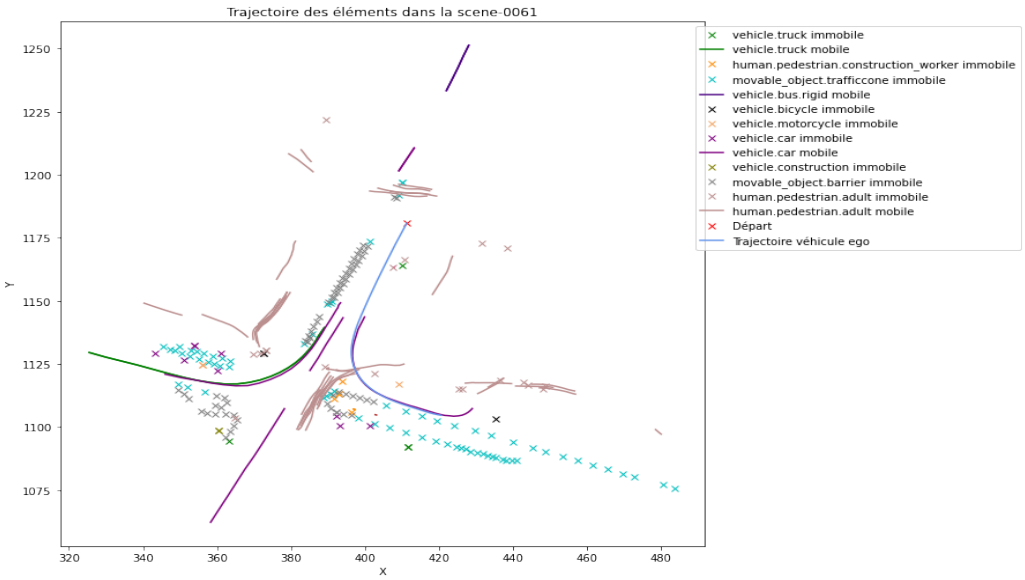


Figure 10 - Trajectoire des éléments de la scène 61

Le diagramme de la figure 10 permet de donner une première interprétation de la situation de trafic. L'objectif est de mieux comprendre cette situation de trafic via l'analyse des interactions entre les agents participants dans la scène. Dans ce cas, il s'agit d'un carrefour à sens unique dans lequel le véhicule égo poursuit un autre véhicule, des interactions entre d'autres véhicules sont observées avec un haut niveau de précision.

2.2 Dans le diagramme espace-temps

On peut également tracer la trajectoire des objets de la scène dans le diagramme espace-temps. Pour ce diagramme, j'ai choisi de tracer uniquement les trajectoires des véhicules mobiles car dans cette étude nous ne nous intéresserons pas aux interactions avec les autres éléments pouvant modifier le comportement des véhicules comme les piétons.

Pour tracer ces trajectoires, il faut bien à faire attention au décalage dans le temps : certains véhicules ne sont pas repérés dès le début de la scène, leur trajectoire doit donc débuter à l'instant où ils sont détectés et non à t=0 ce qui correspond au départ du véhicule ego.
De plus, il faut également être attentif au décalage dans l'espace : si le véhicule ego suit un véhicule, la trajectoire du véhicule suivi doit démarrer plus tard dans l'espace et non pas à 0. Pour prendre en compte ce phénomène, j'ai défini une fonction qui permet de savoir si un véhicule suit ou non le véhicule ego (cf Annexe 3), dans le cas où les deux véhicules circulent dans le même sens. Pour cela on définit deux vecteurs :

- $\overrightarrow{V1}$: le vecteur reliant le premier point de la trajectoire du véhicule et le premier point de la trajectoire du véhicule ego
- $\overrightarrow{V2}$: le vecteur reliant le premier point de la trajectoire du véhicule et le second point de sa trajectoire

On calcule le produit scalaire entre les deux et on obtient 3 cas de figure :

- Le produit scalaire est positif : les deux vecteurs sont dans le même sens donc le véhicule suit le véhicule ego
- Le produit scalaire est négatif : les deux vecteurs sont en sens inverse donc le véhicule ego suit le véhicule
- Le produit scalaire est nul : les vecteurs sont orthogonaux donc on démarre au même endroit dans l'espace.

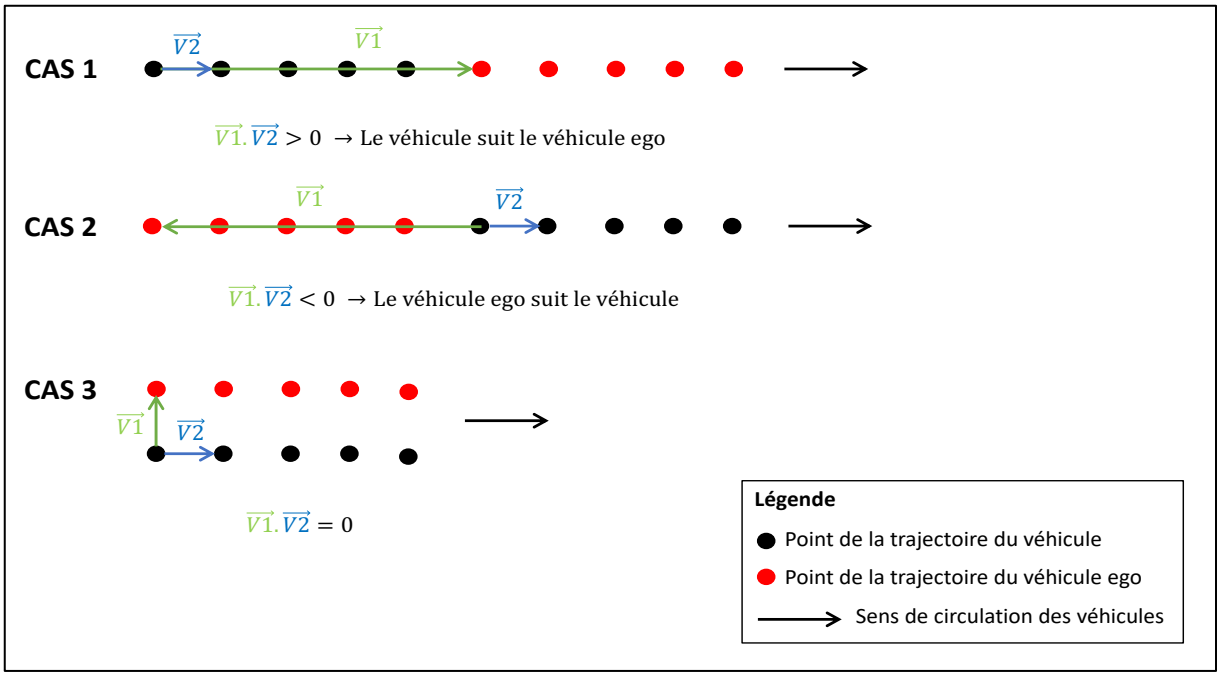


Figure 11 - Principe de l'algorithme suiveur

Les diagrammes obtenus sont illustrés en figures 12 et 13 :

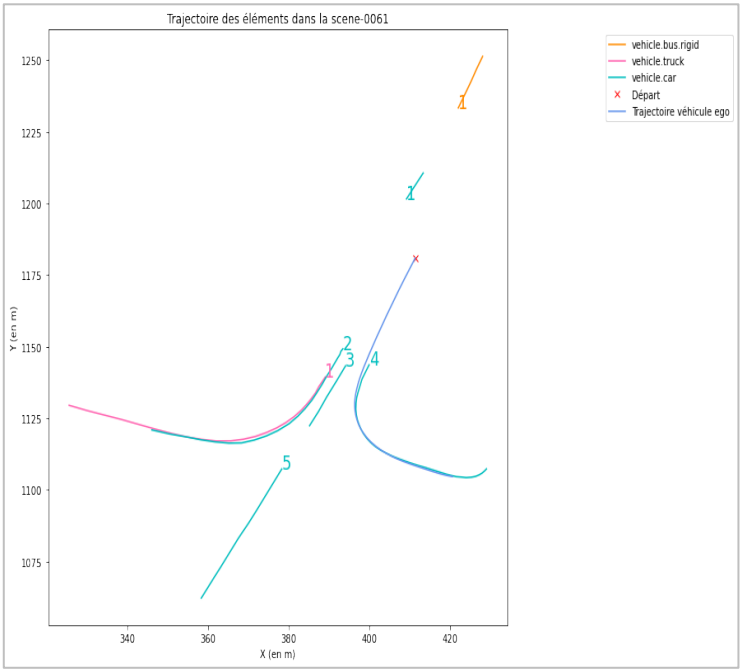


Figure 13 - Trajectoire dans l'espace des véhicules de la scène 61

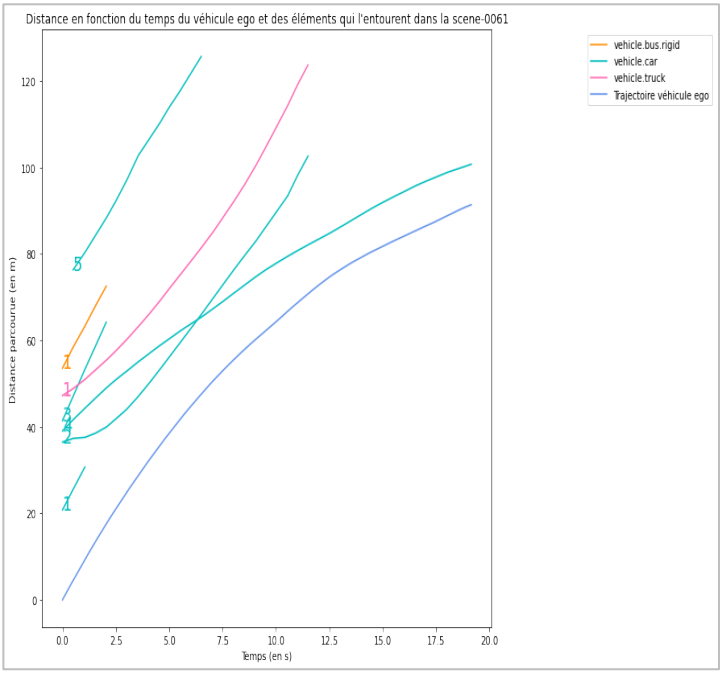


Figure 12 -Trajectoire dans le diagramme espace-temps des véhicules de la scène 61

On remarque que certains véhicules ont quasiment les mêmes trajectoires comme le véhicule ego et le 'vehicle.car 4' et, le 'vehicle.car 2' et le 'vehicle.truck 1'. Leurs trajectoires diffèrent par des temps d'arrivée ou de départ légèrement décalés dans le temps. Visuellement, il est relativement aisé dans certains cas d'identifier les trajectoires similaires et de comprendre comment interagissent les véhicules entre eux. Le problème est

d’automatiser cette détection et d’identifier automatiquement le comportement des véhicules les uns par rapport aux autres. En reconnaissant par exemple les véhicules se suivant entre eux, on peut ensuite appliquer les méthodes usuelles de théorie du trafic pour produire des indicateurs de trafic. On peut également s’intéresser à des problématiques d’accidentologie en créant des indicateurs relatifs aux presqu’accidents et/ou définir un risque d’accident au cours du temps t.

Le but de notre étude va donc être de trouver une façon d’automatiser la caractérisation des interactions entre les véhicules à partir des données obtenues grâce aux véhicules autonomes. Pour cela, il faut commencer par déterminer des indicateurs permettant de caractériser ces interactions.

Partie 2 : Caractérisation des interactions

L'objectif de cette partie est d'identifier les critères et caractéristiques qui nous seront utiles pour comparer les couples de trajectoires entre eux. On souhaite déterminer le type d'interaction entre chaque couple de véhicules c'est-à-dire savoir si les véhicules se suivent ou non, s'ils circulent dans le même sens, si leurs trajectoires risquent de se croiser à une intersection… Des critères basés sur l'orientation des trajectoires mais aussi sur leur similarité seront donc calculés.

III. Similarité des trajectoires

Un des premiers critères sur lequel on peut se baser est lié à la similarité des trajectoires : comme vu précédemment, certaines trajectoires sont presque identiques, il faut créer un indicateur qui retranscrive cette similarité. De nombreuses techniques existent pour réaliser ce travail.

1. Les méthodes existantes

Avec l'essor des données sur les positions d'objets mouvants notamment au travers des GPS, l'étude de la similarité des trajectoires est devenue un problème commun pour lequel de nombreuses méthodes ont été créées. [5]

Il existe deux types de mesures de similarité [5] :

- La similarité spatiale : on recherche des trajectoires ayant des formes géométriques similaires sans prendre en compte le temps.
- La similarité spatio-temporelle : on prend à la fois en compte la dimension spatiale et la dimension temporelle.

1.1 Mesure de similarité spatiale

Parmi les mesures de similarité spatiale on peut citer les plus connues : la distance Euclidienne, la distance d'Hausdorff, la distance de Fréchet…

La distance Euclidienne :

Elle consiste à mesurer la distance entre le i^e point de chacune des trajectoires. Cette méthode semble peu adaptée à notre cas puisqu'il peut y avoir un décalage temporel entre deux trajectoires même si elles sont identiques et avec cette méthode elles ne seront donc pas considérées comme étant similaires.

La distance d'Hausdorff :

On considère deux trajectoires P et Q. On prend un point i de P et on cherche le point le plus proche de i appartenant à Q. On réitère cette action pour tous les points de P et on détermine le maximum parmi ces distances, noté maxP. On fait de même pour les points de Q et le résultat obtenu est noté maxQ. La distance d'Hausdorff sera alors le maximum entre maxP et maxQ. Cette méthode ne semble pas adaptée au cas de trajectoires car dans la résolution on se concentre uniquement sur l'emplacement des points et non sur leur ordre, ce qui est important dans le cas d'une trajectoire. La dimension temporelle doit être prise en compte.[6]

La distance de Fréchet :

Pour le calcul de cette distance, on peut faire l'analogie à une personne qui promène son chien en laisse : la distance de Fréchet représente la longueur minimale de laisse pour que chacun puisse réaliser cette trajectoire.

De nombreuses autres méthodes de similarité spatiale existent mais dans notre cas, une méthode de similarité spatio-temporelle sera plus adaptée.

1.2 Mesure de similarité spatio-temporelle

Les mesures de similarité spatio-temporelle sont aussi nombreuses, il y a par exemple : Dynamic Time Warping (DTW), Edit distance with real penalty (ERP), Edit distance on real sequences (EDR)…

Dynamic Time Warping (DTW) :

Cette méthode est initialement utilisée pour comparer deux séries temporelles. Contrairement à la distance euclidienne, on ne compare pas les deux séries temporelles linéairement dans le temps, on peut « déformer » le temps. Le but est d'obtenir un alignement optimal des deux séries de données. Une trajectoire constitue une série temporelle mais avec deux données spatiales, cette méthode peut donc être utilisée. D'autant plus qu'elle est avantageuse car on peut l'utiliser bien que les deux trajectoires soient décalées dans le temps et que leurs longueurs soient différentes. Je présenterai plus précisément cette méthode dans la suite du rapport.

Edit distance with real penalty (ERP) :

Cette méthode est basée sur la 'Edit distance'. La 'Edit distance' est un moyen de quantifier la similarité entre deux chaînes de caractères en calculant le nombre d'opérations requises pour transformer une des chaînes de caractère en l'autre. Dans cette méthode, on utilise la norme L1 comme métrique et on cherche à faire correspondre chaque point des trajectoires normalisées

Edit distance on real sequences (EDR) :

Cette méthode est semblable à la précédente et utilise également la 'Edit distance'. La différence réside dans la normalisation de la trajectoire.

Plusieurs de ces méthodes peuvent être utilisées dans notre cas, il faut donc les comparer pour savoir laquelle est la plus efficace.

1.3 Comparaison des méthodes

Dans l'article 'An effectiveness study on trajectory similarity measures' [7], on compare 6 mesures de similarités sur des données réelles de trajectoires de taxi : la distance euclidienne, DTW sous deux formes, ERP, EDR et Longest common subsequence (LCSS). La méthode Longest common subsequence est une mesure utilisée pour déterminer la similarité entre des chaînes de caractères. Pour des trajectoires, on détermine la plus longue sous-séquence commune de points entre deux trajectoires.

Pour cela, les auteurs considèrent une trajectoire donnée et créent plusieurs trajectoires à partir de celle-ci. Il y a trois types de transformation :

- Ré-échantillonnage : on ajoute ou on supprime des points aléatoirement de la trajectoire.
- Décalage des points : on ne change pas le nombre de point, on modifie uniquement leur positionnement. On peut soit décaler les points indépendamment les uns des autres, soit les décaler tous de la même façon, dans la même direction.
- Ajout de bruit : on ajoute des points placés hors de la trajectoire.

Ces transformations sont contrôlées par deux paramètres : le taux de points modifiés et un seuil définissant une distance limite à laquelle un point peut être déplacé par rapport à sa position initiale.

On calcule pour chaque méthode la similarité entre la trajectoire initiale et les trajectoires modifiées : la trajectoire avec le plus petit degré de transformation devrait être la plus similaire.

Les résultats montrent qu'il n'y a pas de méthode qui permette d'avoir de bons résultats quelle que soit la modification faite : certaines sont sensibles aux bruits, d'autres aux ré-échantillonnage… Chaque méthode a ses faiblesses mais également ses forces.

Dans notre cas d'étude, j'ai donc choisi d'utiliser la méthode DTW pour évaluer la similarité entre les trajectoires des véhicules des scènes. C'est une méthode extrêmement populaire et adaptée à de nombreux problèmes.

2 Dynamic Time Warping (DTW)

La DTW est une méthode récursive [8] : dans le cas de trajectoires, on construit une matrice des distances contenant la distance euclidienne entre chaque point des deux trajectoires. Ci-dessous la matrice correspondante pour le calcul entre un véhicule et le véhicule autonome :

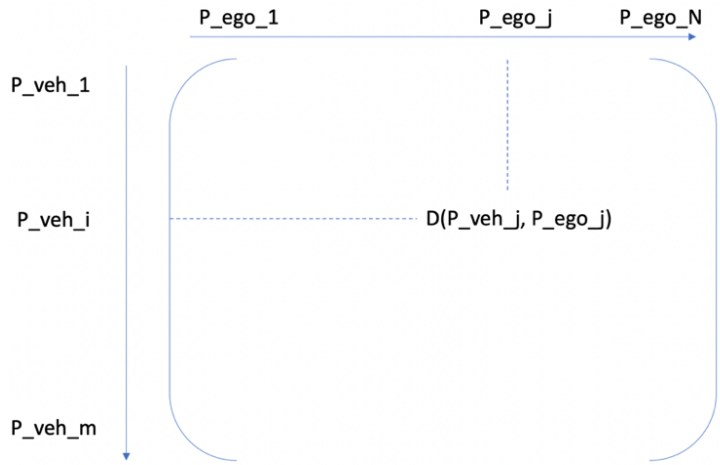


Figure 14 - Matrice des distances

Le but de l'algorithme est ensuite de déterminer un chemin optimal dans la matrice des distances, appelé 'warping path'.

Ce chemin doit respecter certaines règles :

- Le départ du chemin est le point (0,0) et l'arrivée est le point (m,N).
- Le but est de déterminer le chemin pour lequel la somme des distances sera minimale.
- On ne peut pas revenir en arrière dans le chemin : les indices des points doivent être croissants ou constants.

Pour déterminer l'ensemble des points qui appartiennent au chemin on utilise la formule de récurrence suivante :

$$\gamma(i,j) = d + \min\big(d(i,j+1) + d(i+1,j) + d(i+1,j+1)\big)$$

Avec :

- $\gamma(i,j)$: la distance cumulative obtenue lorsque que l'on atteint le point (i,j)
- d : la distance calculée en amont de ce point
- $\min\big(d(i,j+1) + d(i+1,j) + d(i+1,j+1)\big)$: la distance minimale avec les points qui entourent le point (i,j), cela permet de passer au point suivant

Cette méthode présente parfois des ‘singularités’ c’est-à-dire des points qui sont reliés à tous les autres points dans l’alignement des deux séries temporelles. [9] Pour remédier à ce problème, une nouvelle solution est proposée : la Derivative Dynamic Time Warping (DDTW). [10] Cette nouvelle méthode consiste à procéder de la même façon mais avec une matrice des distances différente : on construit une matrice contenant la racine carrée de la différence des dérivées instantanées des points de chaque série temporelle. Si l’on doit utiliser cette méthode dans notre cas, il suffit d’utiliser les vitesses puisque la dérivée d’un point d’une trajectoire correspond à sa vitesse.

3 Application aux données des trajectoires nuScenes

3.1 DTW normalisée

On applique donc cette méthode pour les dix scènes de notre échantillon : on calcule la DTW pour chaque couple de véhicules dans une scène donnée. On choisit de comparer autant les véhicules au véhicule ego que les véhicules entre eux. On décide également, pour plus de cohérence, de calculer la matrice des distances et donc la DTW, uniquement sur les points où les véhicules interagissent dans le temps. (cf Annexe 5) Cela signifie que pour un véhicule qui est uniquement détecté par le véhicule ego entre 0 et 6 secondes, on conservera uniquement les points de la trajectoire du véhicule ego compris dans cet intervalle de temps. Si on calcule la DTW pour deux véhicules non autonomes dont les intervalles de détections sont respectivement [t1_debut ; t1_fin] et [t2_debut ; t2_fin], l’intervalle de temps utilisé pour calculer la matrice de distance sera donc :

$$[max(t1\_debut\ ;\ t2\_debut)\ ;\ min(t1\_fin;\ t2\_fin)]$$

Enfin, pour comparer les DTW des différents couples, ces distances doivent être normalisées car certains véhicules sont détectés pendant toute la scène alors que pour d’autres le temps de détection est très faible. Le package utilisé sur python possède une option de normalisation qui consiste simplement à diviser la DTW par la somme des dimensions de la matrice distance.

Prenons l’exemple d’un couple dans la scène 61 : le ‘vehicle.car2’ et le véhicule ego. On obtient le chemin optimal ci-dessous, visualisé dans la matrice des distances et l’alignement suivant :

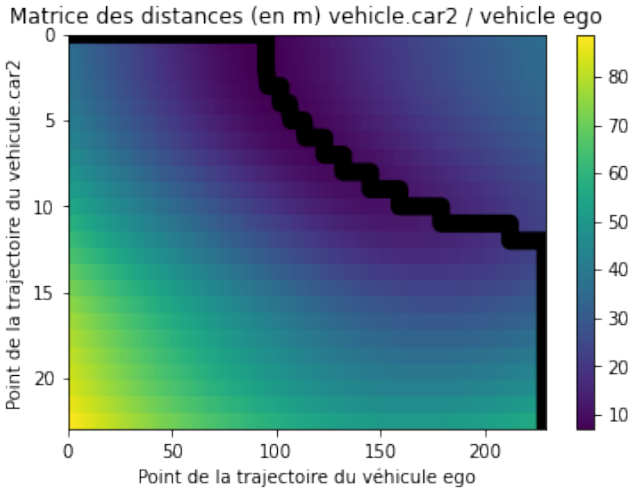


Figure 16 – Chemin optimal de minimisation de distance entre trajectoires

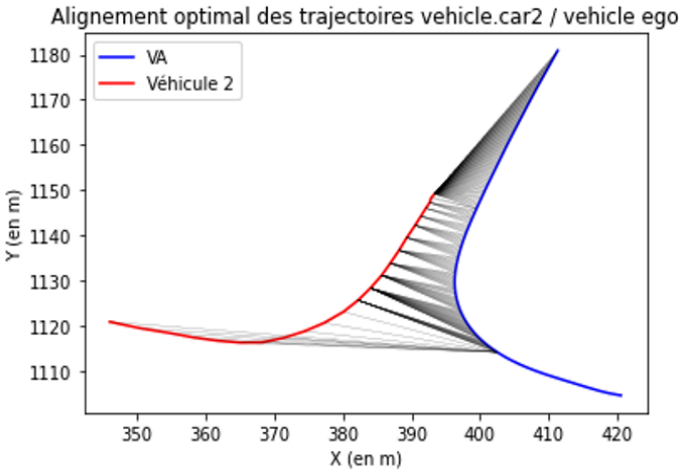


Figure 15 – Alignement obtenu avec DTW

On observe que le chemin optimal obtenu passe bien par les points où la distance entre les points des deux trajectoires est minimale. Les traits verticaux et horizontaux correspondent au décalage temporel entre les deux trajectoires. Les points d'extrémité concentrent le poids de la distance.

Cet indicateur semble donner de bons résultats, on peut donc le conserver pour comparer nos couples de trajectoires : il fournit une distance normalisée entre deux trajectoires données.

3.2 Calcul de DTW sur une fenêtre glissante

Les trajectoires de deux véhicules peuvent être très similaires pendant une période puis les véhicules peuvent aller dans des directions totalement opposées comme c'est le cas pour le véhicule ego et le 'vehicule.car2' représentés sur la figure 13 ci-dessus.

Pour prendre en compte ce phénomène, il est possible de calculer la DTW sur une fenêtre glissante : au lieu de calculer une matrice de distance sur l'intégralité du temps d'interaction de deux véhicules, on décide de calculer la matrice de distance sur des intervalles de temps donnés et on calcule la DTW correspondant à ces matrices. On choisit de calculer ces matrices sur des intervalles de 1 seconde : [0 ; 1], [1 ; 2] … (cf Annexe 6)

On définit alors une fonction nommée 'dtw_glissante' qui, pour une scène donnée, renvoie une base de données avec en index les intervalles de temps choisis et en colonne, les véhicules de la scène et la valeur de la DTW normalisée obtenue sur ces intervalles pour une comparaison avec le véhicule ego. Lorsque les véhicules n'interagissent pas pendant un intervalle de temps, on complète la base de données par un 'Nan'. Pour la scène 61, on obtient le résultat suivant :

	vehicle.truck 1	vehicle.bus.rigid 1	vehicle.car 1	vehicle.car 2	vehicle.car 3	vehicle.car 4	vehicle.car 5
[0, 1]	40.799903	55.787752	24.142933	30.496933	35.404644	32.984968	72.845804
[1, 2]	36.055663	73.237145	40.586937	23.331444	38.301036	29.549825	68.295421
[2, 3]	32.986513	88.573954	NaN	18.449771	41.083774	26.569462	68.405882
[3, 4]	31.024287	NaN	NaN	15.947720	NaN	23.462511	70.101673
[4, 5]	30.573171	NaN	NaN	15.506647	NaN	20.642470	72.894331
[5, 6]	33.314116	NaN	NaN	18.349097	NaN	19.167695	76.961201
[6, 7]	34.661234	NaN	NaN	19.956462	NaN	16.683242	78.957768
[7, 8]	35.264882	NaN	NaN	20.535479	NaN	13.643041	NaN
[8, 9]	40.640636	NaN	NaN	25.524104	NaN	12.589354	NaN
[9, 10]	48.562025	NaN	NaN	31.787413	NaN	11.977253	NaN
[10, 11]	59.728710	NaN	NaN	40.472456	NaN	10.951940	NaN
[11, 12]	74.851973	NaN	NaN	54.255603	NaN	10.783121	NaN
[12, 13]	NaN	NaN	NaN	NaN	NaN	8.582556	NaN
[13, 14]	NaN	NaN	NaN	NaN	NaN	8.507734	NaN
[14, 15]	NaN	NaN	NaN	NaN	NaN	8.740337	NaN
[15, 16]	NaN	NaN	NaN	NaN	NaN	8.821031	NaN
[16, 17]	NaN	NaN	NaN	NaN	NaN	8.950762	NaN
[17, 18]	NaN	NaN	NaN	NaN	NaN	8.877451	NaN
[18, 19]	NaN	NaN	NaN	NaN	NaN	8.282291	NaN
[19, 20]	NaN	NaN	NaN	NaN	NaN	6.696696	NaN

Figure 17 - Résultat de la fonction 'dtw_glissante' pour la scène 61

Observons plus précisément l'évolution de la DTW au cours du temps pour chaque véhicule :

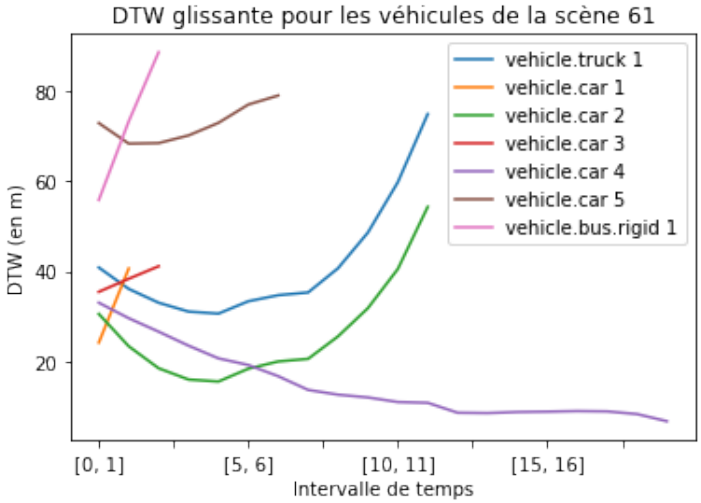


Figure 18 - Évolution de la DTW par intervalle de temps pour les véhicules de la scène 61

On remarque bien que la DTW n'est pas constante au cours du temps : ici tous les véhicules s'éloignent du véhicule ego sauf le 'vehicle.car4' qui s'en rapproche de plus en plus. D'où la nécessité de prendre en compte cette évolution pour ne pas omettre l'interaction dynamique entre les véhicules.

On prendra alors comme indicateur, les valeurs minimales, maximales et moyennes de la DTW glissante pour avoir un aperçu de la variation de distance entre les trajectoires au cours du temps.

IV. Autres indicateurs

Nous avons choisi de travailler avec d'autres indicateurs pour caractériser une interaction entre deux véhicules, ils sont définis en plusieurs catégories : d'autres indicateurs de distance, des indicateurs de temps et de retard, des indicateurs d'orientation et de sens et des indicateurs de vitesse. Étudions plus précisément ces indicateurs.

1. Les indicateurs de distance

En plus de la DTW, il peut être intéressant d'utiliser directement les valeurs de la matrice distance en conservant le minimum, le maximum et la valeur moyenne dans cette matrice. Il faudra uniquement vérifier par la suite que ces indicateurs ne sont pas corrélés avec la DTW.

2. Les indicateurs de temps d'interaction et de retard

Comme vu précédemment, le véhicule autonome ne détecte pas tous les véhicules de son environnement pendant la totalité de la scène. On peut donc calculer un temps d'interaction entre deux véhicules. Cependant, il faut également prendre en compte le fait que l'on calcule également les interactions entre véhicules non autonomes : deux véhicules peuvent être détectés pendant un cours instant par le véhicule autonome et pour autant interagir ensemble pendant toute cette période. L'indicateur finale pour deux véhicules est donc le suivant :

$$T = \frac{\left[max(t1_{debut}\,;\, t2_{debut}) \,-\, min\left(t1_{fin};\, t2_{fin}\right)\right]}{\max\left(t1_{fin} - t1_{debut}\,;\;\, t2_{fin} - t2_{debut}\right)}$$

Il est sans unité, il correspond à un « pourcentage » d'interaction en temps.

De plus, deux trajectoires peuvent être identiques sur une portion mais avec un retard temporel : on peut calculer cet indicateur à l'aide des chemins optimaux calculés à la l'aide de la DTW. Ce retard est visible sur les paliers horizontaux et verticaux que l'on observe dans nos chemins optimaux, comme l'on peut l'observer sur la figure 15 : ces paliers signifient qu'un point de la trajectoire d'un des véhicules est associé à plusieurs points de la trajectoire de l'autre véhicule. Si on regarde de plus près l'alignement optimal de notre exemple dans la partie précédente, on observe que le premier point de la trajectoire du 'vehicule.car2' est associé à de nombreux points de la trajectoire du véhicule ego. Passée cette tendance, les deux trajectoires sont bien alignées comme on peut le voir sur la figure 19 :

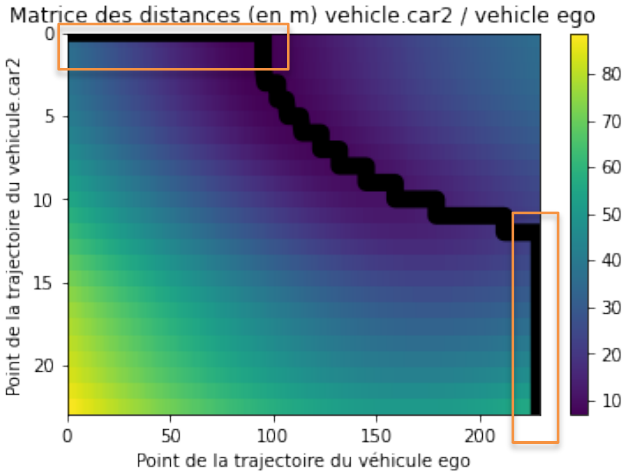


Figure 19 - Observation des paliers de retard entre deux trajectoires

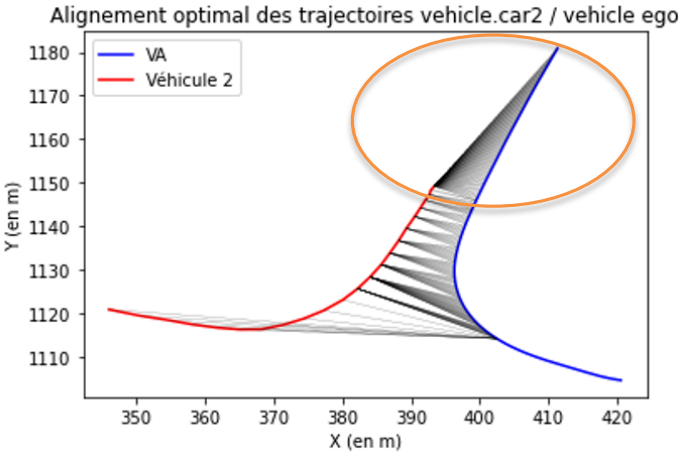


Figure 20 - Observation du retard entre deux trajectoires

3. Les indicateurs d'orientation et de sens

L'orientation instantanée des véhicules semble être un indicateur pertinent pour caractériser les trajectoires se suivant ou en conflit. Pour ce faire, on propose de s'appuyer sur deux indicateurs : la colinéarité et l'orthogonalité instantanée.

3.1 Indicateur de colinéarité

Pour construire cet indicateur, on commence par créer une fonction 'Matrice_produit_scalaire' (cf Annexe 7), celle-ci prend en entrée les coordonnées des trajectoires de deux véhicules et renvoie en sortie une matrice contenant le cosinus de l'angle du produit scalaire entre chaque vecteur vitesse des deux trajectoires.

La façon de procéder est la suivante :

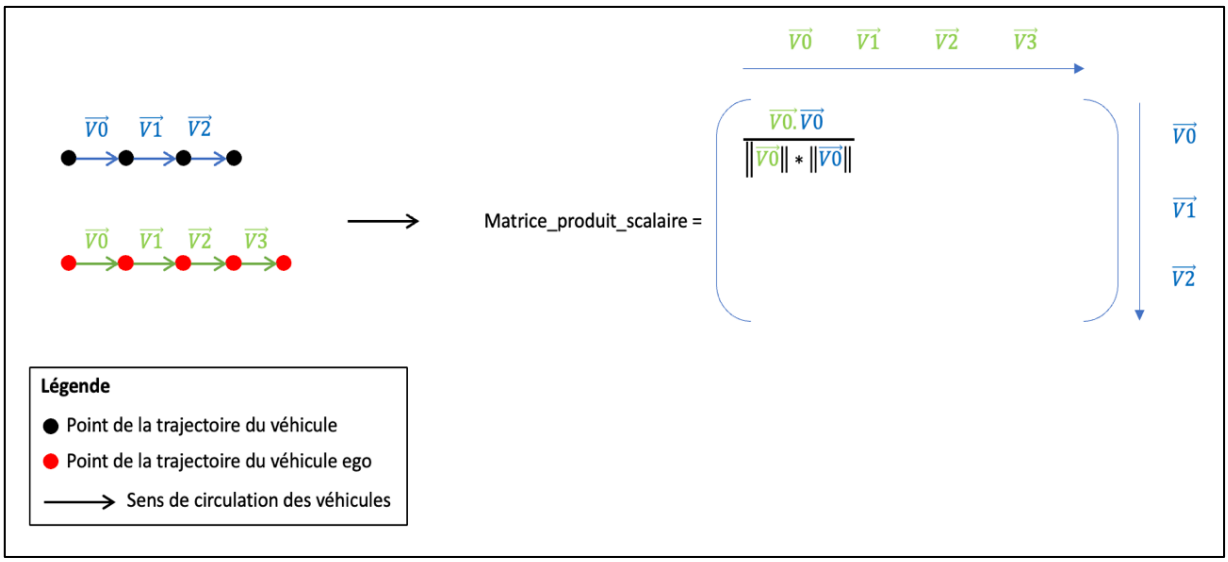


Figure 21 - Calcul de la matrice produit scalaire

Ce calcul s'applique également si on travaille sur l'interaction entre deux véhicules non autonomes.

On conserve ensuite uniquement les valeurs des cosinus pour les points correspondant au chemin optimal obtenu grâce à la DTW. En effet, du point de vue du trafic, les indicateurs calculés ont plus de sens si on compense les écarts temporels entre les deux véhicules. On peut alors calculer nos indicateurs finaux c'est-à-dire : les indicateurs de colinéarité moyen, minimal et maximal. De cette manière, si deux véhicules se suivent, ils sont en principe localement colinéaires et l'indicateur de colinéarité moyen est proche de 1.

3.2 Indicateur d'orthogonalité

On procède de la même façon que pour la matrice de colinéarité, cette fois-ci avec la fonction 'Matrice_produit_vectoriel' qui renvoie une matrice contenant la norme du produit vectoriel de chaque vecteur vitesses des deux trajectoires, c'est-à-dire le sinus de l'angle du produit vectoriel entre ces deux vecteurs.

La méthode est la suivante :

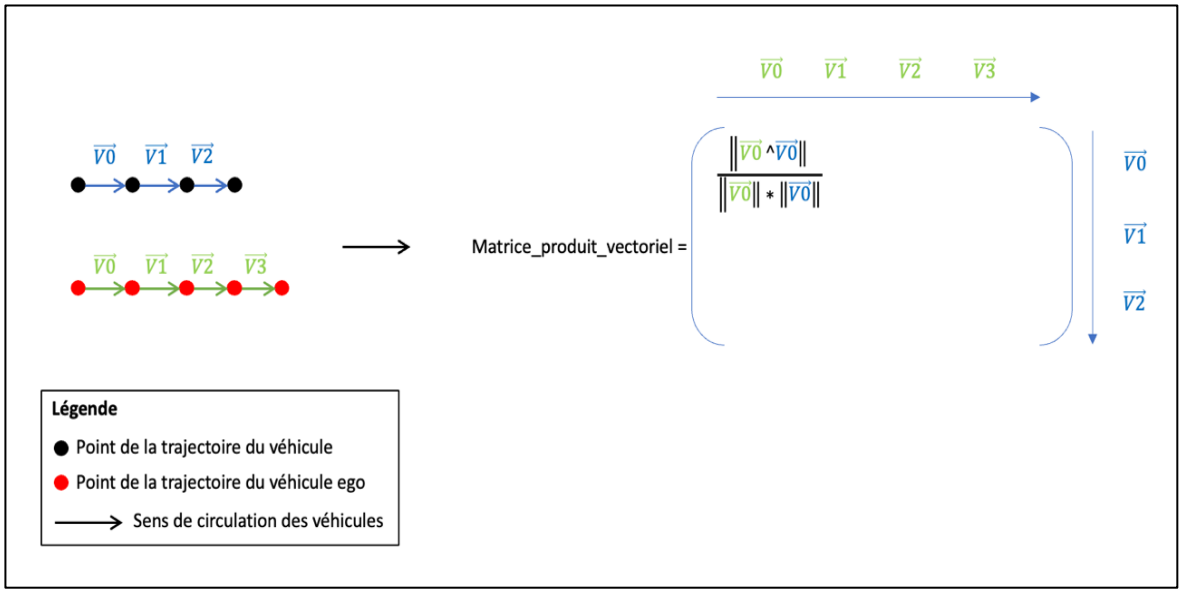


Figure 22 - Calcul de la matrice produit vectoriel

De même, ce calcul s'applique également si on travaille sur l'interaction entre deux véhicules non autonomes.

On conserve ensuite uniquement les valeurs des sinus pour les points appartenant au chemin optimal obtenu grâce à la DTW et on peut calculer nos indicateurs finaux c'est-à-dire : les indicateurs d'orthogonalité moyen, minimal et maximal.

3.3 Indicateur de sens

Pour s'assurer de bien distinguer les véhicules qui circulent dans le même sens ou non, on crée un indicateur catégorique du sens. Celui-ci se base sur l'indicateur de colinéarité moyen :

- Si l'indicateur de colinéarité moyen est positif : on considère que les véhicules sont dans le même sens et cet indicateur prend la valeur 1.
- Sinon : on considère que les véhicules circulent en sens inverse ou que leurs trajectoires sont orthogonales, l'indicateur prend alors la valeur 0.

4. Les indicateurs de vitesse

Pour finir, nous avons choisi de travailler avec un dernier indicateur : les vitesses des véhicules. On crée une fonction 'vitesse_moyenne' qui prend en entrée les coordonnées des trajectoires de deux véhicules et renvoie en sortie les listes des vitesses instantanées sur chaque point de chaque trajectoire. Ces vitesses sont les mêmes que celles prises en compte pour les calculs des indicateurs d'orthogonalité et de colinéarité. Pour nos deux indicateurs finaux : vitesse moyenne véhicule 1 et vitesse moyenne véhicule 2, on prend la valeur moyenne de chacune de ces listes.

Le schéma ci-dessous illustre le processus réalisé pour calculer ces indicateurs pour un couple de véhicules dont on connaît les coordonnées au cours du temps :

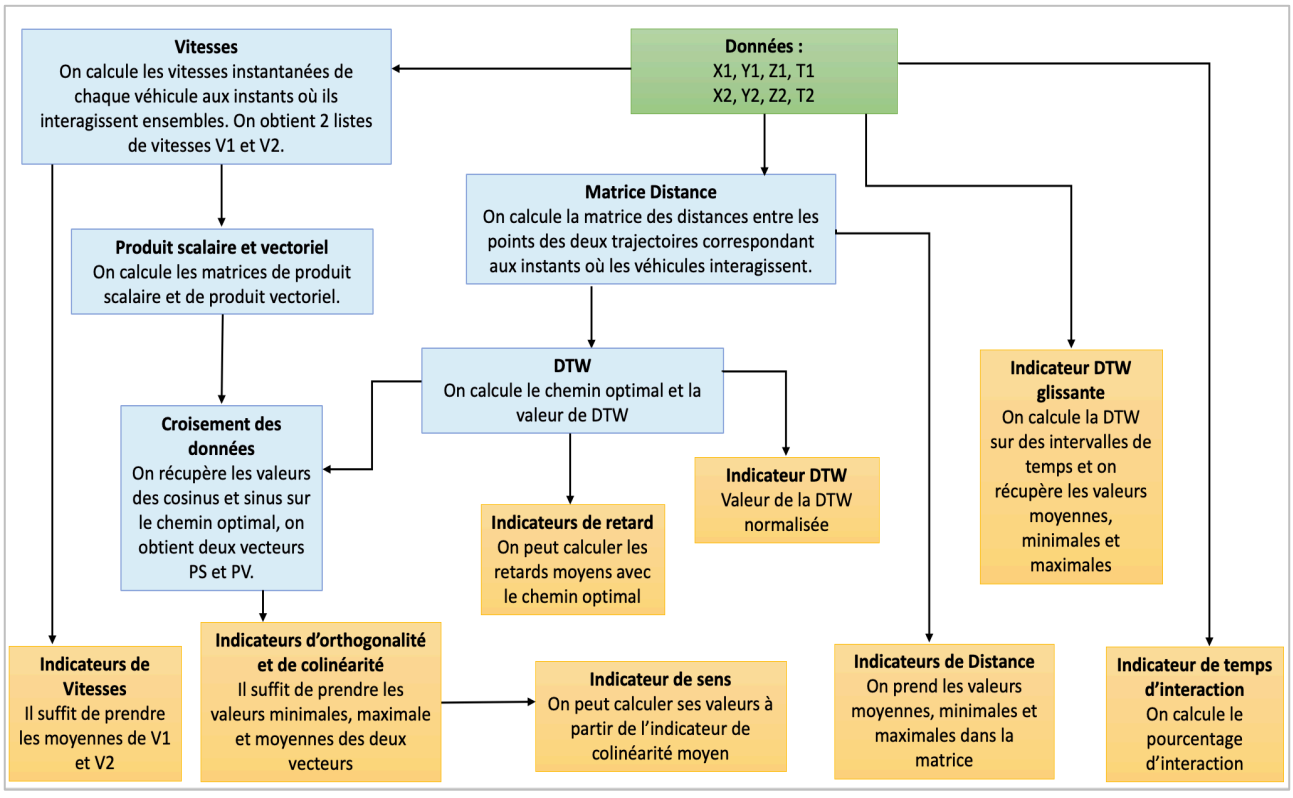


Figure 23 - Processus de calcul des indicateurs

V. La base de données d'indicateurs

Les 19 indicateurs retenus sont donc les suivants :

- Vitesse moyenne véhicule 1
- Vitesse moyenne véhicule 2
- DTW
- DTW glissante moyenne, minimale et maximale
- Durée d'interaction
- Retard moyen (Timelag horizontal et vertical)
- Indicateur d'orthogonalité moyen, minimal et maximal
- Indicateur de colinéarité moyen, minimal et maximal
- Sens
- Distance moyenne, minimale et maximale

On peut donc créer une base de données contenant les valeurs de ces indicateurs pour tous les couples de véhicules dans les dix scènes étudiées.

On effectue ce travail grâce à deux fonctions :

- 'indicateur_similarite' : fonction qui permet de calculer, pour une scène donnée, les valeurs des indicateurs calculés pour tous les véhicules de la scène par rapport au véhicule ego. (cf Annexe 8)
- 'indicateur_similarite_entre_veh' : cette fonction étend le calcul des indicateurs entre tous les véhicules non autonomes.

Il suffit d'appliquer chacune des fonctions à chaque scène et de concaténer les résultats pour obtenir la base de données finale comportant sur chaque ligne un couple dans une scène donnée et les valeurs des indicateurs correspondant.

Pour s'assurer que certaines variables ne sont pas corrélées entre elles, on trace les valeurs des indicateurs les uns par rapport aux autres :

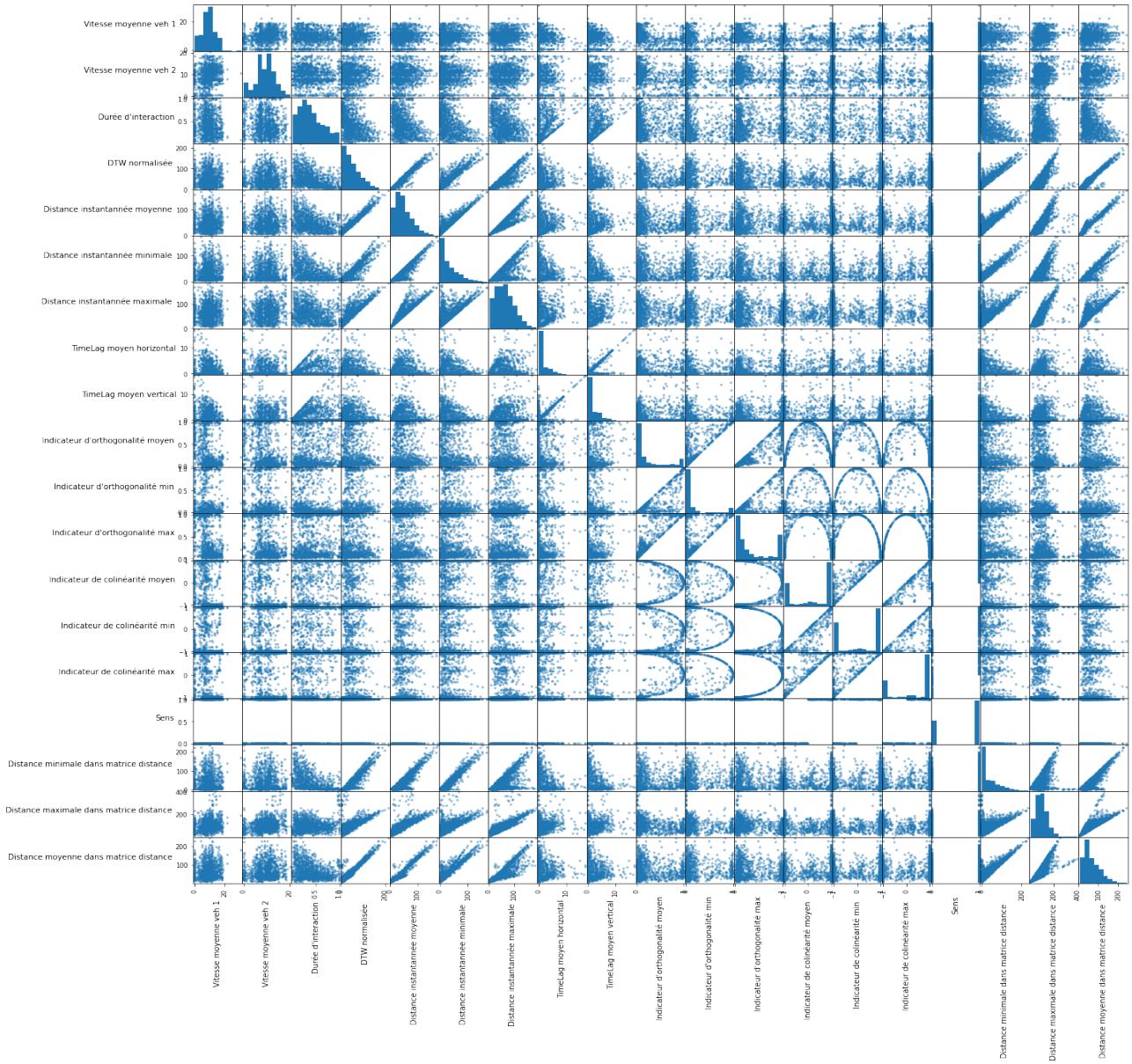


Figure 24 - Répartition des indicateurs les uns par rapport aux autres

Plus le nuage de point est dispersé, plus faible est la corrélation. On n'observe pas de forte corrélation entre certains indicateurs hormis pour les indicateurs où on a calculé des valeurs moyennes, minimales et maximales et également pour les valeurs des DTW et des distances dans les matrices. On décide tout de même de conserver l'ensemble des indicateurs pour le moment afin de favoriser les moments statistiques des données pour mieux développer la classification.

VI. Interactions longitudinales : le Car following

Avant de commencer à classifier les interactions, intéressons-nous à un type d'interaction dont on connaît les caractéristiques et qui servira par la suite de référence pour qualifier la qualité de la classification.

1. Les caractéristiques du Car following

Le car following fait référence à un modèle de déplacement longitudinal. Les modèles de car following sont microscopiques [11] : on considère le trafic comme une entité discrète. Dans ce type de modèle, on caractérise la relation entre vitesse désirée du véhicule et sa distance inter-véhiculaire avec le véhicule qui le précède.

Certaines études ont tenté d'analyser le comportement de véhicules dans un peloton. C'est le cas des chercheurs et ingénieurs qui ont écrit "On some experimental features of car-following behavior and how to model them " [12]. Ils ont réalisé des expériences pour des pelotons de 25 véhicules et de 3 véhicules. Leurs retours montrent que même si la vitesse moyenne des véhicules du peloton est la même, les distances inter-véhiculaires peuvent être significativement différentes et que c'est à des vitesses plus élevées que ces distances ont tendance à être plus faibles.

Le comportement des conducteurs est donc très différent, ce n'est donc pas la distance inter-véhiculaire qui nous permettra d'identifier ce type de comportement mais plutôt des critères basés sur la similarité des trajectoires. On se basera notamment sur le critère de « pourcentage » de temps d'interaction et sur des critères visuels tels que l'absence de véhicule interposé entre les véhicules, la circulation sur une même voie… Intéressons-nous au car following dans la base de données nuScenes.

2. Le Car following dans la base de données

Pour étudier le car following dans les scènes de la base, il faut extraire les couples de trajectoires qui correspondent à ce type d'interaction. Il faut en amont les identifier, cette classification manuelle nous permettra de mieux comprendre ce type d'interaction avant de passer à une classification automatisée. Pour cela, on trace pour chaque scène les trajectoires des véhicules avec leur couleur qui évolue selon le temps de détection avec la fonction nommée 'traj_ego_veh'.

On obtient le résultat suivant pour la scène 61 par exemple :

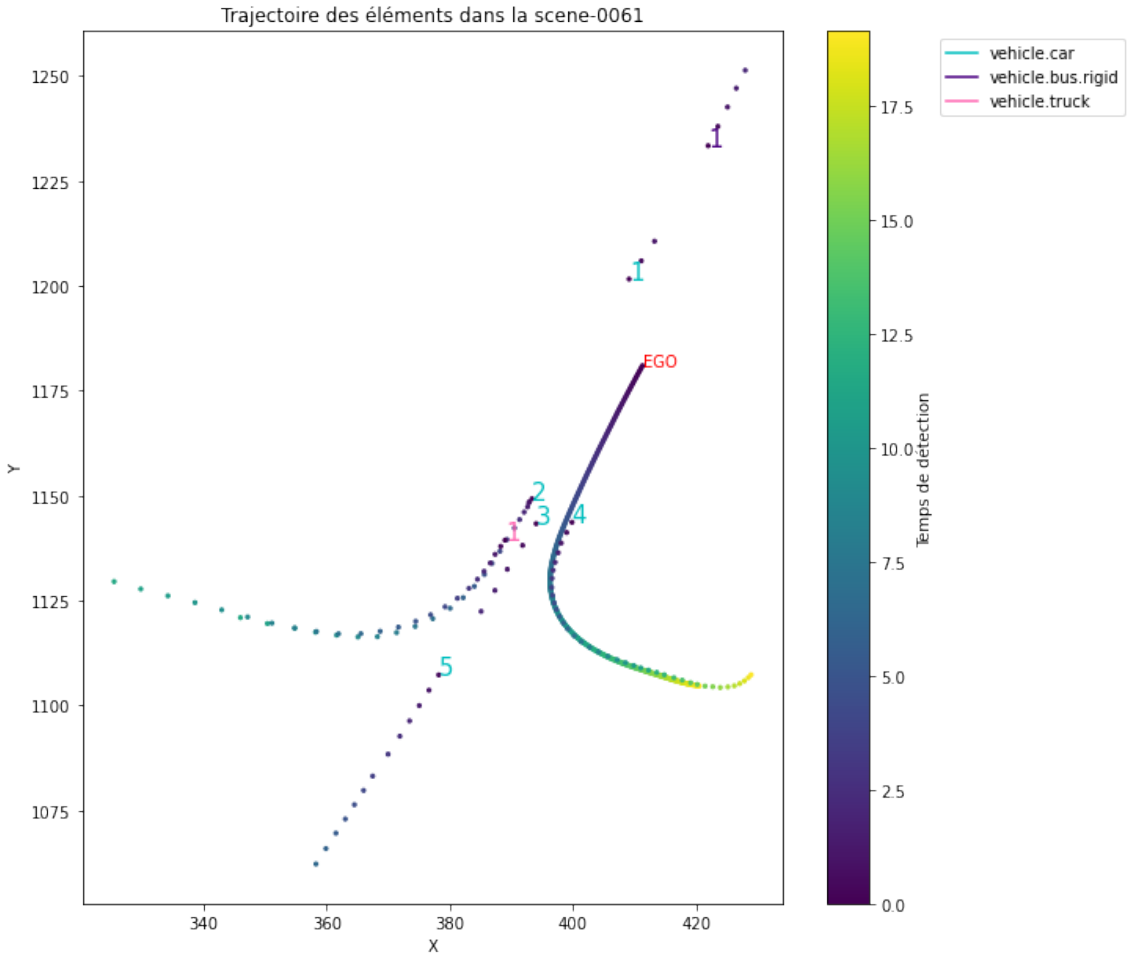


Figure 25 - Évolution des trajectoires au cours du temps dans la scène 61

On remarque ici que les couples ‘vehicle.car2 / vehicle.truck1’ et ‘vehicle.car4 / ego’ sont en car following : en effet ils ont les mêmes trajectoires avec une distance inter-véhiculaire qui semble constante dans le temps.

On procède de la même façon pour les autres scènes et on extrait une trentaine de couples dont l’interaction correspond à du car following. On peut alors étudier les caractéristiques de ces couples : sont-ils semblables et peut-on en dégager un individu moyen représentatif de ce type d’interaction ?

On trace la répartition des indicateurs les uns par rapport aux autres pour les véhicules sélectionnés :

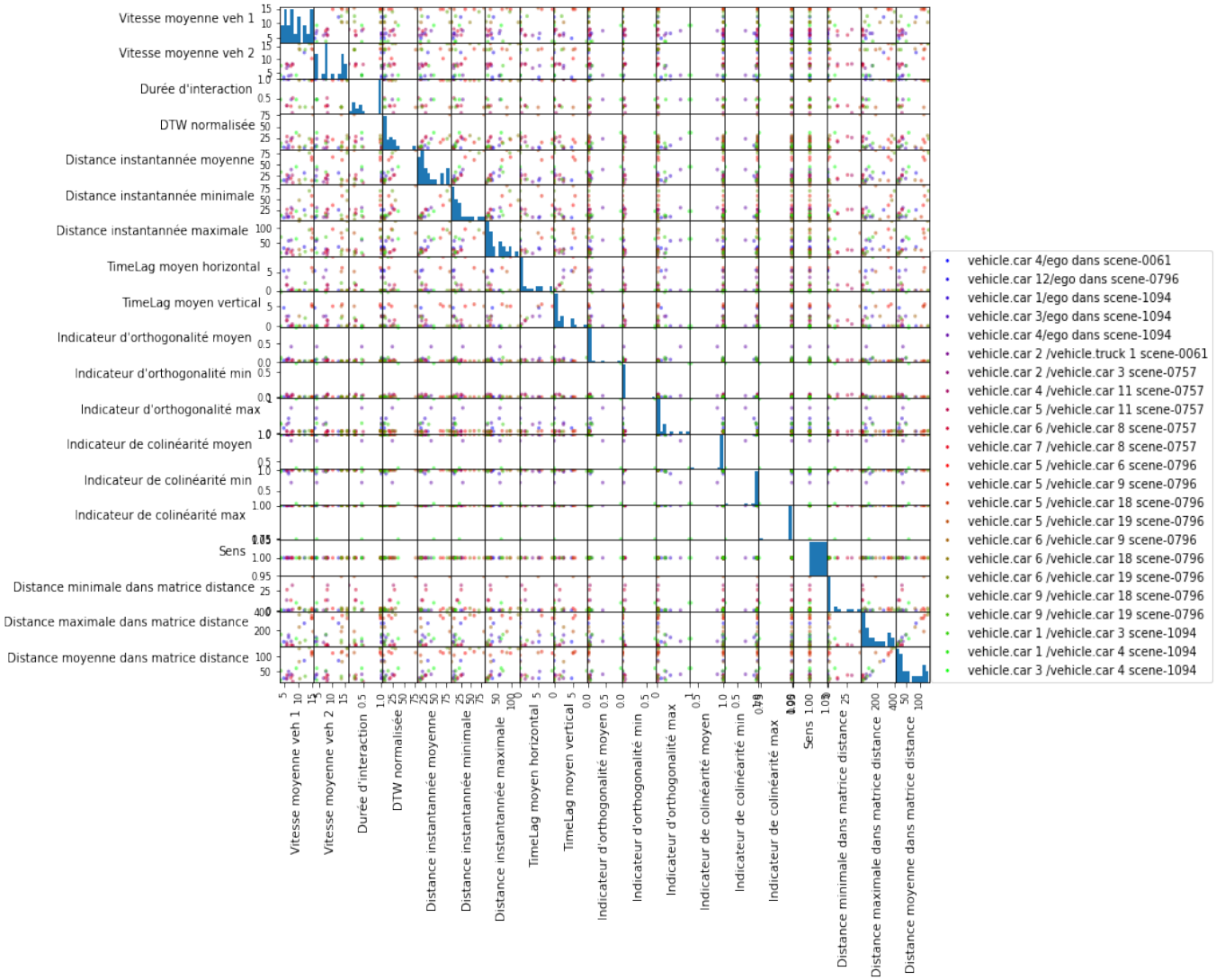


Figure 26 - Répartition des indicateurs les uns par rapport aux autres pour la classe car following

On remarque que les indicateurs de colinéarité, orthogonalité et sens sont très semblables, ce n'est cependant pas le cas pour les indicateurs de vitesse, de distance et de retard. Il est donc possible que dans le clustering, ces véhicules ne forment pas une classe unique.

Intéressons-nous maintenant à des indicateurs de trafic : on travaille sur la distance inter-véhiculaire au cours du temps.

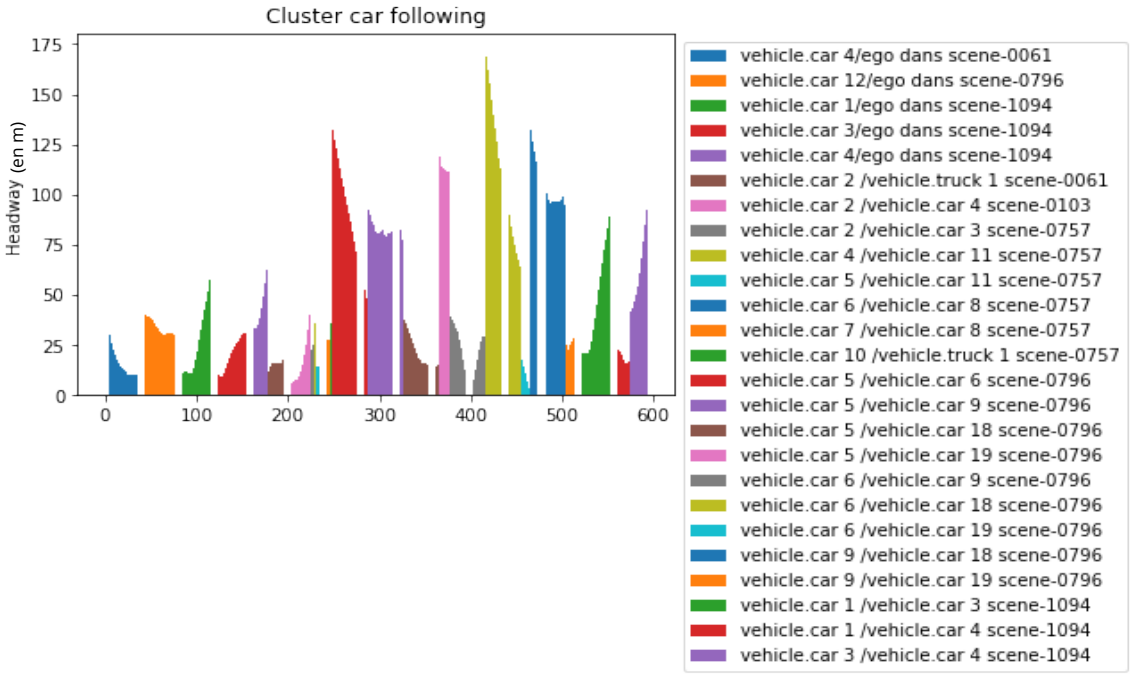


Figure 27 - Distance inter-véhiculaire pour la classe car-following

On n'observe pas de constance pour cette variable, ni au niveau de l'amplitude, ni au niveau du temps. C'est ce qui a été décrit dans les articles cités précédemment. On remarque tout de même que lorsque le véhicule ego est suiveur, les distances inter-véhiculaires sont plutôt dans la tranche basse. On ne dispose pas de suffisamment d'échantillon pour procéder à un test statistique, mais cela reste un élément à confirmer.

On a vu que les indicateurs liés à l'orientation sont extrêmement représentatifs de l'interaction de type car-following. Reste alors à observer la distribution des autres paramètres vis-à-vis de l'ensemble des interactions recensées. Pour ce faire, nous proposons d'adopter des approches automatiques ou semi-automatiques.

Partie 3 : Classification des interactions

VII. Introduction aux approches usuellement appliquées pour catégoriser les trajectoires

Comme nous l'avons vu précédemment, il peut être difficile d'identifier manuellement des trajectoires ayant les mêmes caractéristiques. Le regroupement de trajectoire est un travail qui a déjà été traité de nombreuses fois et ce pour divers objectifs :

- Regrouper les trajectoires similaires, présentant les mêmes caractéristiques afin d'analyser des habitudes de mobilités.[13]
- Identifier les « bruits », c'est-à-dire les trajectoires anormales ou pour lesquelles les données sont inexactes.[14]
- Déterminer les axes sur lesquels le trafic est plus dense, où les flux sont plus conséquents. [15]

Plusieurs techniques ont été proposées, la majorité d'entre elles se basent sur des méthodes de clustering déjà existantes. Le clustering est un processus qui consiste à regrouper un ensemble d'objets en quelques classes, appelées « clusters », d'objets similaires. Intéressons-nous à ces méthodes.

1. Les méthodes de clustering

L'ensemble des méthodes de clustering peut être divisé en trois groupes [16] :

- Les algorithmes non-supervisés : l'apprentissage non-supervisé correspond à l'ensemble des algorithmes pour lesquels les données d'entrée ne fournissent pas d'exemple de résultats, c'est-à-dire que les données sont non étiquetées.
- Les algorithmes supervisés : l'apprentissage supervisé, au contraire de l'apprentissage non-supervisé, se base sur des données étiquetées. On détermine l'étiquette de nouvelles données à partir de prédictions basées sur les données dont l'étiquette est déjà connue.
- Les algorithme semi-supervisés : ils sont entre les deux catégories précédentes puisque l'on a des données étiquetées mais elles sont peu nombreuses.

De plus, il existe différents types de méthodes de clustering parmi les trois groupes cités ci-dessus. [14]

1.1 Les méthodes basées sur la partition

Ce sont des méthodes pour lesquelles l'unique paramètre, c'est-à-dire le nombre de clusters k est déterminé à l'avance. Chaque cluster est non vide et chaque individu appartient à un unique cluster.

On peut citer la méthode k-means. [13] C'est une méthode itérative qui consiste à fixer k centroïdes, c'est-à-dire fixer le centre d'un cluster, et à associer les individus restants au centroïde qui leurs est le plus proche afin de former des clusters. Avec les clusters obtenus, on recalcule les valeurs des centroides et on réitère ces opérations jusqu'à obtenir des centroides fixes, c'est-à-dire jusqu'à ce que l'algorithme ait convergé. Il existe aussi la méthode fuzzy-c-means (FCM), elle fonctionne de la même façon que k-means mais dans ce cas, un individu peut appartenir à plusieurs clusters.

Ce type de méthode est principalement limité par la nécessité de connaître le nombre de clusters dès le départ alors que ce n'est souvent pas le cas et également par le fait qu'il possède un coût de mémoire important.

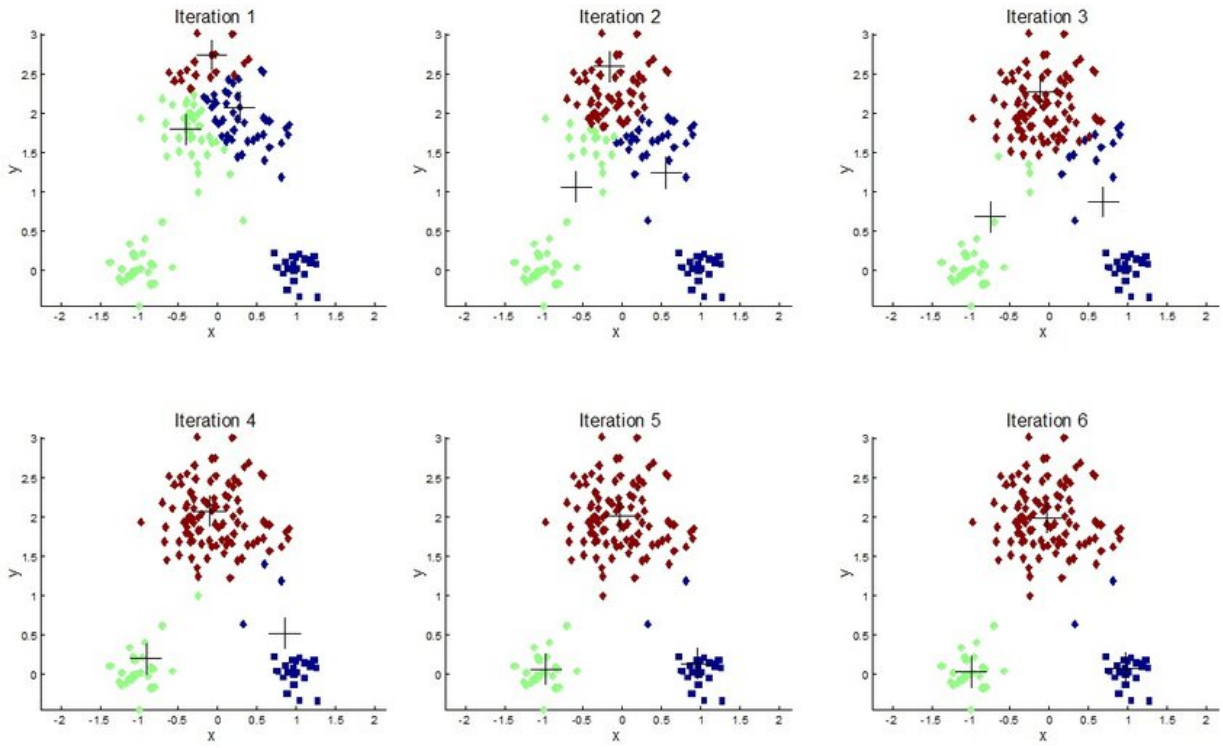


Figure 28 - Exemple de fonctionnement de k-means)

(Sources : https://www.researchgate.net/figure/Exemple-de-deroulement-de-lalgorithme-K-means_fig18_337840545)

1.2 Les méthodes basées sur la hiérarchie

Avec ces méthodes, on classe les données selon un système de hiérarchie : par exemple si on voulait classer un groupe d'êtres humains, on pourrait faire deux groupes, les hommes et les femmes. Avec ce type de méthode, on peut construire des dendrogrammes. Ces méthodes s'appuient sur une mesure de dissimilarité définie entre deux individus et entre deux classes. Ces deux mesures prennent différentes formes, notamment pour la dissimilarité entre 2 classes : saut maximum, saut minimum, lien moyen…

On peut citer comme exemple la classification ascendante hiérarchique (CAH). [14] C'est une méthode ascendante puisqu'on part des individus seuls puis on les regroupe selon plusieurs catégories pour former des clusters. Il existe aussi la méthode descendante qui consiste à partir d'un seul cluster qu'on va diviser petit à petit.

Dans l'article « Similarity based vehicle trajectory clustering and anomaly detection", les auteurs utilisent un clustering hiérarchique pour classer les trajectoires de véhicules obtenues à partir de vidéos de trafic réel.[17] Ils se basent sur les coordonnées spatiales des trajectoires pour réaliser le clustering et puisque les données sont complexes, ils réalisent deux clusterings pour avoir des résultats plus fins. La figure 24 illustre un exemple de résultat :

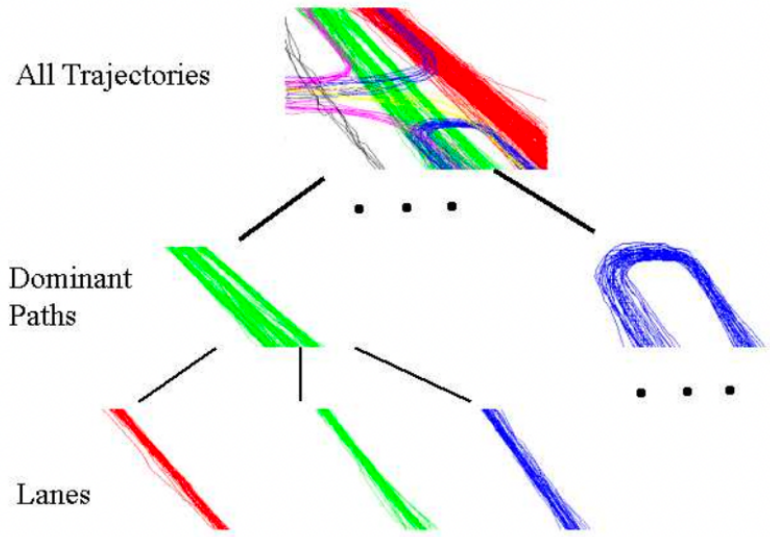


Figure 29 - Exemple de clustering hiérarchique [17]

Ce sont des méthodes simples mais le nombre de calculs peut devenir très important avec un grand nombre d'individus.

1.3 Les méthodes basées sur la densité

Dans ce type de méthode, on ne détermine pas à l'avance le nombre de clusters. On définit un seuil caractérisant la densité minimale permettant d'obtenir un cluster.

DBSCAN (« Density-Based Spatial Clustering of Application with Noise ») est une méthode très connue sur le sujet. [18] [16] Cet algorithme de clustering fonctionne avec deux paramètres d'entrée : epsilon et MinPts. Epsilon représente une distance : si on considère un ensemble d'objet contenant les objets p et q, on considère que p appartient à l'epsilon-voisinage de q si la distance entre ces deux objets est inférieure à epsilon. Avec la valeur d'epsilon, on peut définir l'epsilon-voisinage de chaque objet et également trois catégories d'objets :

- Les « core point » ou points cœur : ils correspondent aux points dont le nombre de voisin dans l'epsilon-voisinage est supérieur à MinPts.
- Les « border point » ou points frontières : ce sont les points pour lesquels le nombre de voisin dans l'epsilon-voisinage n'est pas nul mais inférieur à MinPts.
- Les « outliers » ou points aberrants : ce sont les points qui n'ont aucun voisin dans l'epsilon-voisinage.

Les clusters résultent des points cœur : si un point p est un point cœur alors il forme un cluster avec tous les objets de son epsilon-voisinage et si de plus un autre point de son voisinage est aussi un point cœur alors le cluster est formé de l'union du voisinage de p et de q. Tous les points aberrants sont rassemblés dans un cluster noté « cluster -1 ». Ci-dessous un exemple illustrant la façon de procéder :

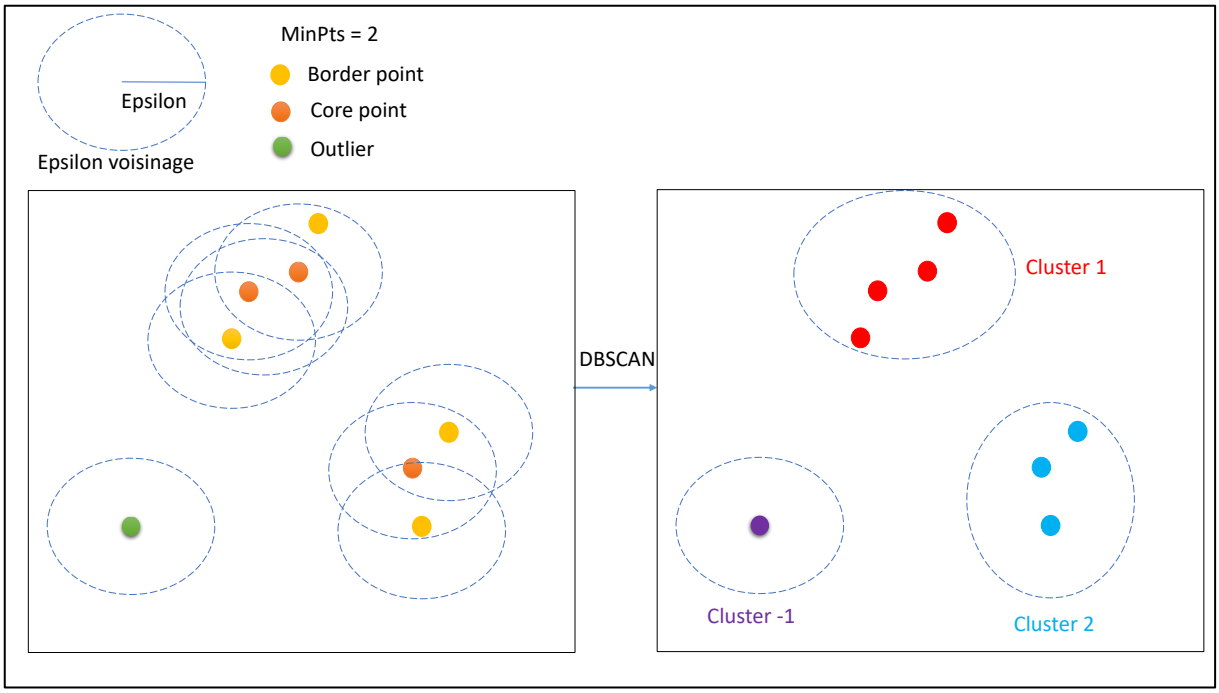


Figure 30 - Exemple DBSCAN

1.4 Les méthodes « Grid based methods »

Pour ce type de méthode, on ne s'intéresse pas au nombre d'individus mais plutôt à leur emplacement : on découpe l'espace des individus en une grille, on calcule la densité de chaque case, on peut alors trier les cellules selon leurs densités et identifier les clusters.

La méthode « Statistical Information Grid » (STING) fait partie de ce groupe et consiste à faire ce clustering en utilisant une grille à cellules rectangulaires. [14]

1.5 Les méthodes « Model based methods»

Ces méthodes utilisent des distributions statistiques pour créer les clusters : on utilise des fonctions de densité pour représenter la distribution spatiale des points. L'algorithme COBWEB fait partie de ces méthodes. [14]

Ces méthodes de clustering peuvent être appliquées au cas du regroupement de trajectoires de plusieurs façons.

2. Le clustering de trajectoires

Le clustering de trajectoires nécessite au préalable d'utiliser une mesure de distance entre les trajectoires, nous en avons cité quelques-unes dans la partie 2.III « Similarité des trajectoires ». Celle-ci peut ensuite être utilisée de diverses manières.

Le clustering de trajectoires est particulier puisqu'une trajectoire présente une dimension spatiale et une dimension temporelle. Il existe 5 catégories de clustering d'objets mouvants. [14]

2.1 Regroupement basé sur l'espace

Pour ce type de regroupement, on s'occupe très peu de la dimension temporelle, ce sont les informations spatiales qui nous intéressent davantage.

C'est ce qu'ont tenté de faire Jim Kim et Hani S. Mahmassani dans l'article « Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories ». [19] Leur but était de déterminer les trajectoires les plus usuelles sur des données de la ville de New York. Ils ont considéré des trajectoires avec des longueurs différentes et des points repérés à des instants différents c'est pourquoi la dimension temporelle était peu prise en compte. Leur objectif était de ne pas classer ensemble des véhicules effectuant le même trajet mais en sens inverse ou des véhicules qui ont la même trajectoire mais dans des lieux différents. Ils ont procédé en quatre étapes :

- Étape 1 : Avec la mesure de distance Longest common subsequence (LCS), ils ont créé une matrice D contenant la distance entre chaque trajectoire de leur base de données.
- Étape 2 : À partir de cette matrice, ils ont réalisé un clustering avec l'algorithme DBSCAN et en ont déduit dans chaque cluster, le trafic sur les arcs identifiés.
- Étape 3 : Dans chaque cluster, ils ont tenté d'identifier, des trajectoires représentatives à l'aide d'un clustering hiérarchique et de la LCS qui permet de voir si des trajectoires se chevauchent et peuvent donc fusionner.
- Étape 4 : Classer de nouvelles trajectoires. Pour cela, il suffit de calculer la LCS entre ces nouvelles trajectoires et les trajectoires représentatives des clusters. Les nouvelles trajectoires appartiendront au cluster dont la LCS est la plus faible.

Leur étude permet de déterminer les zones de trafic dense à New York.

2.2 Regroupement en fonction du temps

Dans ce cas, on prend bien en compte l'aspect temporel des trajectoires. C'est bénéfique car par exemple dans notre cas, les véhicules ne sont pas forcément détectés pendant toute la scène par le véhicule autonome or certains véhicules peuvent avoir la même trajectoire mais avec un décalage temporel. Cet aspect sera donc à prendre en compte dans nos futures exploitations.

L'idéal reste tout de même de prendre en compte les dimensions temporelles et spatiales pour réaliser le clustering.

2.3 Regroupement en découpant les trajectoires

Les données que l'on possède sur les trajectoires sont souvent très précises et on obtient alors de longues trajectoires qui passent par plusieurs axes et qui ont des longueurs différentes d'une trajectoire à une autre. Suivant l'objectif du clustering, comme déterminer les flux par intersection, ce type de données n'est pas adapté : il faut diviser les trajectoires en segment. Certaines méthodes existent pour ça, comme l'algorithme « Traclus » qui consiste à découper les trajectoires en sous-trajectoires suivant des changements de direction puis à faire un clustering sur cet ensemble à l'aide de DBSCAN. [15]

2.4 Regroupement sémantique de trajectoires

Dans ce type de clustering, on ne se contente pas des informations temporelles et spatiales, on ajoute des informations géographiques : cela permet de savoir ou un véhicule s'est arrêté et donc s'il y a des points stratégiques dans le lieu où circule le véhicule, des feux tricolores… Il faut cependant disposer de beaucoup d'informations sur l'environnement du véhicule pour pouvoir faire ce type de classement. [14]

2.5 Regroupement en fonction du réseau routier

Les véhicules circulent sur un réseau routier, leurs trajectoires sont donc contraintes, c'est un aspect qui est souvent peu pris en compte pourtant on pourrait obtenir de meilleurs résultats. C'est ce qui est proposé dans ce type de clustering et ce qui a été en particulier fait dans l'article « NEAT : Road network aware trajectory clustering ». [15] Le but de NEAT est de créer des sous-trajectoires par rapport à la structure du réseau qui impose certains mouvements : des mouvements continus, des changements de direction… C'est pour cela qu'ils commencent par diviser les trajectoires par intersection et réalise ensuite le clustering qui permet d'obtenir les flux par intersection.

Dans notre cas, on ne peut pas utiliser ce type de modèle car on souhaite classer les interactions et on travaille sur des scènes filmées dans plusieurs lieux donc avec des réseaux différents.

Nous venons de voir plusieurs possibilités pour regrouper des trajectoires. Dans la plupart des articles, le clustering est surtout utilisé pour obtenir des zones de trafic dense. L'idée de notre travail est plutôt de classer les trajectoires selon la façon dont elles interagissent avec les autres véhicules. On ne souhaite pas comparer des véhicules un à un sur un axe mais plutôt comparer l'interaction entre des couples de véhicules. L'idée est de classer ensemble même des couples de véhicules qui ne circulent pas sur le même réseau, puisque certains sont à Boston et d'autres à Singapour, du moment qu'ils ont le même type d'interaction. Nous allons donc utiliser la base de données faites sur nos couples et présentée dans la partie précédente du rapport et utiliser plusieurs méthodes de clustering présentées dans cette partie pour les rassembler en clusters.

VIII. Application à la base de données nuScenes

Pour réaliser la classification des interactions, nous avons commencé par tester plusieurs algorithmes classiques de clustering, DBSCAN et k-means et nous avons comparé les résultats obtenus pour savoir lequel était le plus adapté aux résultats attendus.

1. DBSCAN

On travaille à partir de la base de données d'indicateurs construite en amont (Partie 2.V). Dans les parties précédentes, nous avions tenté d'observer la corrélation des variables en les traçant les unes par rapport aux autres. On va maintenant vérifier ces résultats en réalisant une analyse en composantes principales (ACP).

1.1 ACP

Une analyse en composantes principales permet de visualiser des données par des graphiques simples : on procède à une réduction du jeu de données en le résumant. Son objectif est de résumer l'information par un nombre réduit d'indicateurs.

Pour réaliser l'ACP, on commence par normaliser notre base de données afin de donner la même importance à toutes les variables. On conservera les données ainsi pour faire nos clusters par la suite. On regarde ensuite le pourcentage cumulé de variance expliquée afin de connaître le nombre de composantes à conserver :

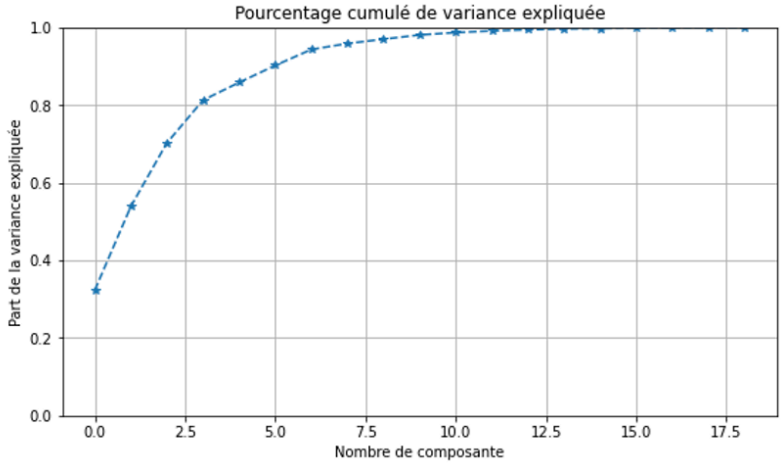


Figure 31 - Détermination du nombre de composantes dans l'ACP

On peut voir qu'avec seulement quatre variables on explique plus de 80% de la variance, on peut donc conserver uniquement ces quatre composantes.

Il faut maintenant projeter la base de données sur ces quatre dimensions et analyser leur signification. On représente les cercles de corrélations pour les différentes composantes deux à deux :

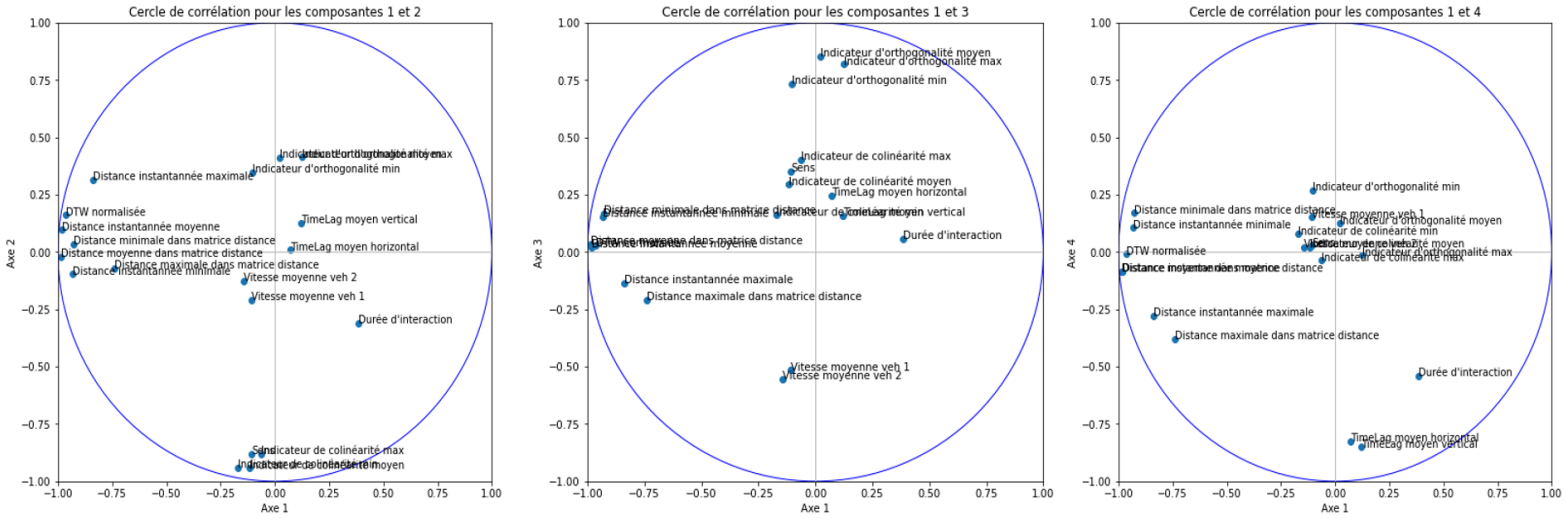


Figure 32 - Interprétation des dimensions retenues

Cela nous permet d'en déduire les significations suivantes pour les axes :

- Axe 1 : cet axe est expliqué par des distances
- Axe 2 : il oppose les indicateurs d'orthogonalité et de colinéarité, il se rapporte donc à la direction des véhicules
- Axe 3 : il oppose vitesse et orthogonalité, il se rapporte donc aux vitesses des véhicules
- Axe 4 : il est relatif au timelag, c'est-à-dire le retard des véhicules qui est très peu corrélé aux autres variables

Quatre variables semblent donc jouer un rôle prédominant dans cette base de données : la distance, la direction, la vitesse et le retard.

Avec la base de données projetée sur ces quatre axes, on utilise l'algorithme DBSCAN. Pour rappel de la partie 3.VII.1.3, deux paramètres sont à déterminer pour appliquer l'algorithme : epsilon et MinPts. On teste l'algorithme avec plusieurs valeurs pour le couple (epsilon ; MinPts) :

- (1 ; 2) : dans ce cas, on obtient 14 clusters mais 90% des couples d'individus se trouvent dans le cluster 0.
- (1.5 ; 2) : dans ce cas, on obtient seulement 6 clusters et 95% des couples d'individus se trouvent dans le cluster 0.

En testant d'autres valeurs et en modifiant MinPts, on fait toujours le même constat : un cluster contient la majorité des individus, on ne peut donc pas identifier de comportement particulier.

On décide alors de travailler avec la base construite au départ, en conservant tous les indicateurs et en les normalisant.

1.2 Détermination des paramètres optimaux de DBSCAN

Pour appliquer l'algorithme DBSCAN, il faut au préalable avoir déterminé la valeur des paramètres epsilon et MinPts. On cherche leurs valeurs optimales et on commence par epsilon. On détermine, pour chaque individu, la distance à laquelle se situe son plus proche voisin. On trace ces valeurs en les ordonnant par ordre croissant et on obtient le graphique suivant :

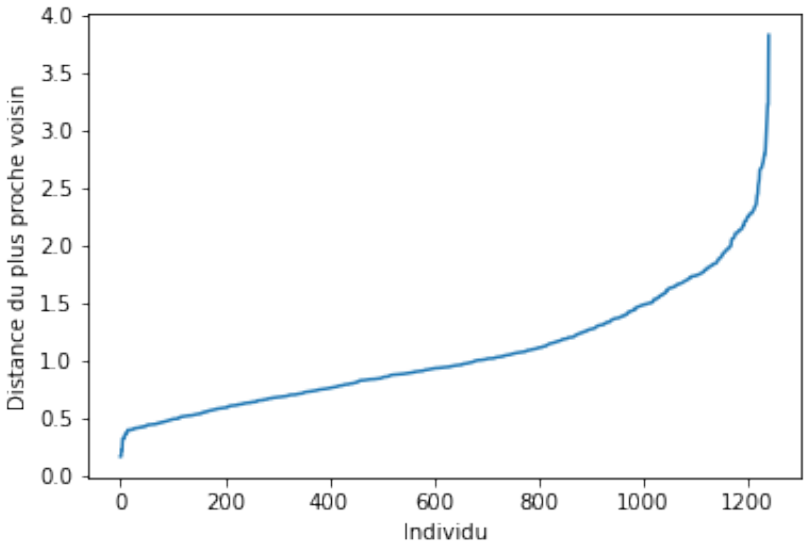


Figure 33 - Détermination de la valeur d'epsilon

On prend une valeur d'epsilon environ égale à la distance pour laquelle on voit un point d'inflexion, c'est-à-dire une distance pour laquelle la majorité des individus auront au moins un voisin. Epsilon vaut donc 1.5.

On teste ensuite plusieurs valeurs n de MinPts :

- Si on prend n = 2 : on obtient 68 clusters
- Si on prend n = 3 : on obtient seulement 31 clusters mais si on regarde plus précisément la répartition des éléments dans les clusters, on remarque que les clusters sont les mêmes et les éléments des clusters en moins ont été placés dans le cluster -1, c'est-à-dire qu'ils sont inclassables.

On fait de même avec la valeur 2 pour epsilon, on obtient beaucoup moins de clusters mais une très grande partie des individus sont placés dans un même cluster, il est donc difficile d'interpréter les classes obtenues.

Les paramètres retenus sont donc epsilon = 1.5 et MinPts = 2.

1.3 Les clusters finaux

1.3.1 Visualisation des clusters

Afin de comprendre et de pouvoir analyser les clusters, on crée une fonction « cluster_dbscan_total » qui trace pour chaque cluster, les trajectoires des individus appartenant au cluster par scène. (cf Annexe 9) Pour plus de visibilité, on crée également une fonction « cluster_dbscan_total_cluster » (cf Annexe 10) qui renvoie pour un cluster donné, le tracé de la trajectoire de couples pour 3 scènes parmi le cluster, on obtient le résultat suivant pour le cluster 0 :

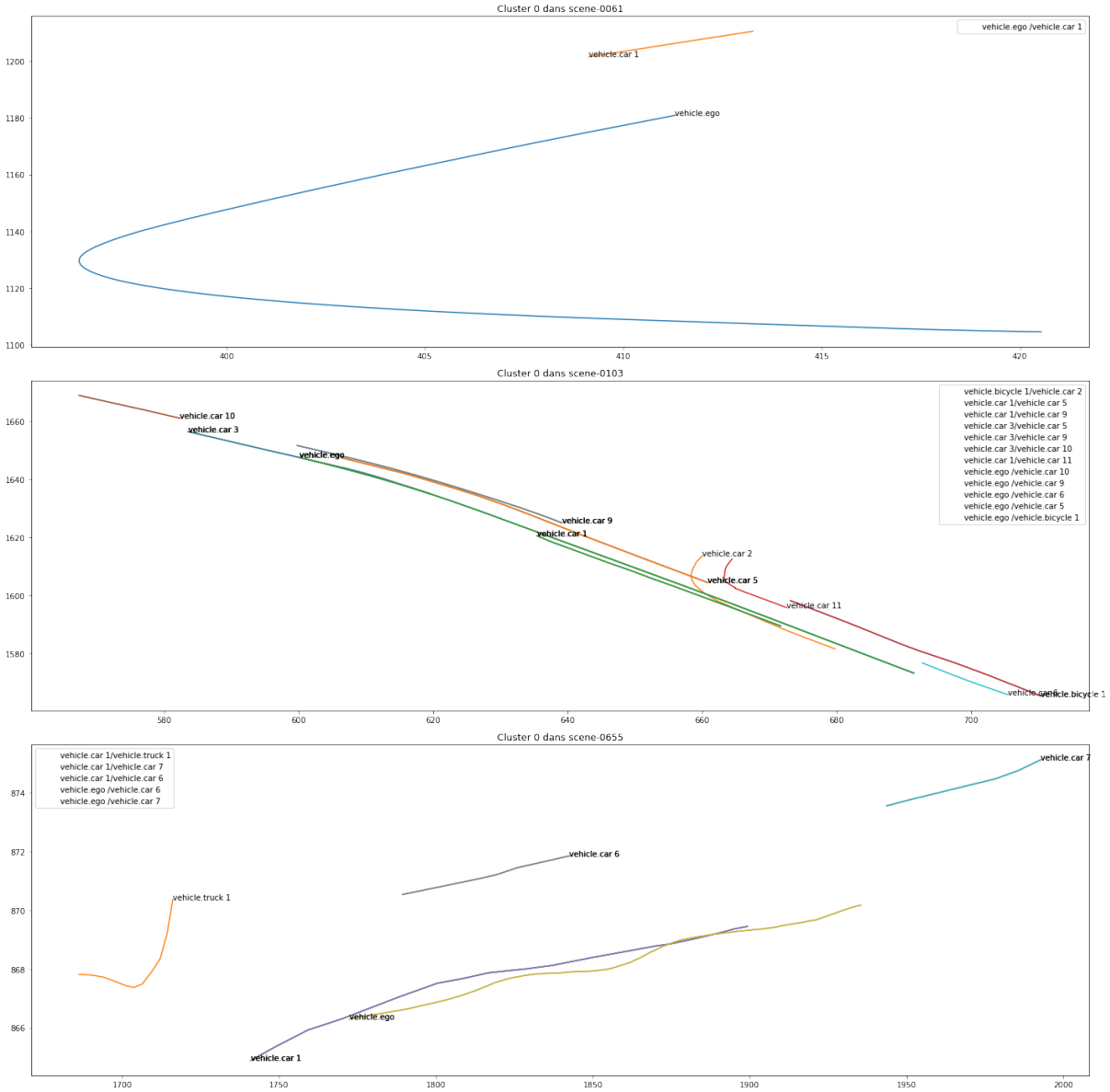


Figure 34 - Visualisation des clusters DBSCAN

Ces visuels seront utiles pour la suite pour interpréter les interactions observées dans chaque cluster.

1.3.2 Les centroïdes des clusters

Pour décrire chaque cluster, on construit des centroïdes : ce sont des individus moyens, représentatifs de leur cluster respectif. Les caractéristiques du centroïde d'un cluster donné sont calculées en faisant la moyenne des indicateurs des individus présents dans le cluster.

On souhaite comparer les clusters entre eux, on va donc comparer les centroïdes deux à deux. Pour cela on commence par calculer la distance inter-cluster : celle-ci est du même ordre de grandeur entre chaque cluster, il n'y a pas de cluster isolé des autres, avec des caractéristiques très spécifiques.

On veut également vérifier la cohérence des clusters, on va donc calculer la moyenne des distances des éléments d'un cluster au centroïde de ce cluster ainsi que la variance de cette distance grâce à la formule : $s^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$.

On obtient les résultats statistiques suivant pour nos 68 clusters :

	Distance intra-cluster moyenne	Variance
count	68.000000	6.800000e+01
mean	0.834821	1.030795e-01
std	0.636222	3.015279e-01
min	0.219973	0.000000e+00
25%	0.558568	7.318534e-32
50%	0.697944	3.081488e-31
75%	0.883214	8.086898e-02
max	4.523113	1.757898e+00

On remarque que pour la majorité des clusters, la distance intra-cluster moyenne est faible et la variance de la distance intra-cluster est très faible également. Le classement effectué par DBSCAN semble donc cohérent puisque les couples d'un même cluster sont proches dans l'espace, ils ont donc des caractéristiques semblables.

1.3.3 Indicateurs de trafic dans les clusters

Pour analyser chaque cluster et ses caractéristiques, on peut également se pencher sur les indicateurs de trafic correspondant aux couples d'un cluster.

On calcule pour chaque couple, dans chaque cluster, la distance parcourue par chaque véhicule pendant le temps où ils interagissent, on peut ensuite tracer l'évolution de ces distances par cluster. Voici un exemple pour le cluster 12, l'évolution de la distance parcourue par le deuxième véhicule de chaque couple :

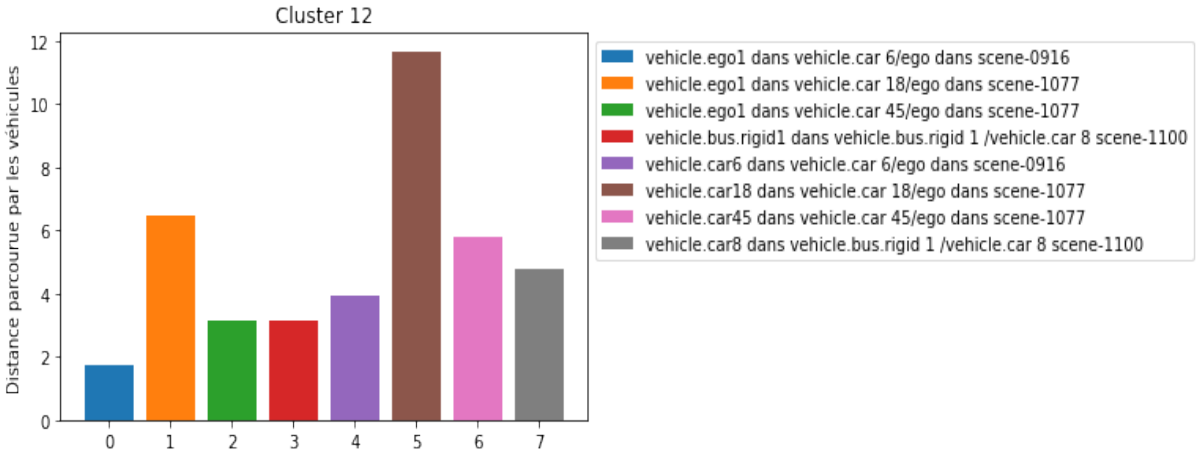


Figure 35 - Distance parcourue par les véhicules du cluster 12

Sur cet exemple, on n'observe pas de « tendance » pour les distances parcourues. C'est également le cas pour les autres clusters. La distance parcourue ne peut donc pas être utilisée pour identifier les éléments appartenant à un même cluster.

On peut également s'intéresser à l'évolution de la distance inter-véhiculaire au cours du temps pour chaque couple de véhicule d'un même cluster et tracer ensuite cette évolution :

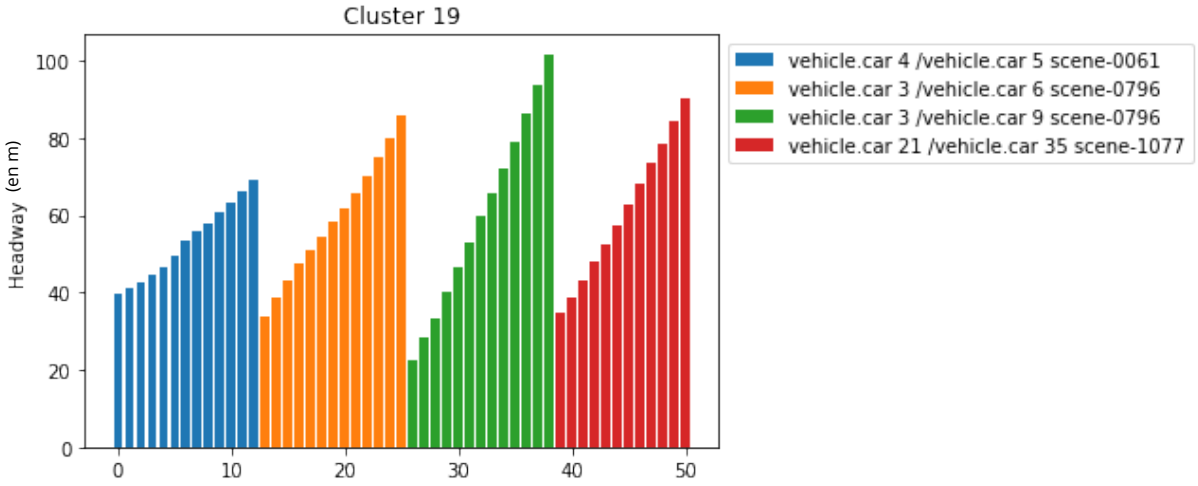


Figure 36 - Évolution de la distance inter-véhiculaire dans le cluster 19

On peut cette fois-ci identifier une tendance dans la distance inter-véhiculaire similaire pour tous les couples d'un même cluster. C'est le cas dans presque tous les clusters dont on dispose. C'est donc un indicateur important pour identifier les éléments appartenant à un même cluster.

On peut également utiliser la distance inter-véhiculaire h pour tracer l'évolution de la vitesse de chaque véhicule en fonction de h. On obtient le résultat suivant pour le cluster 19 :

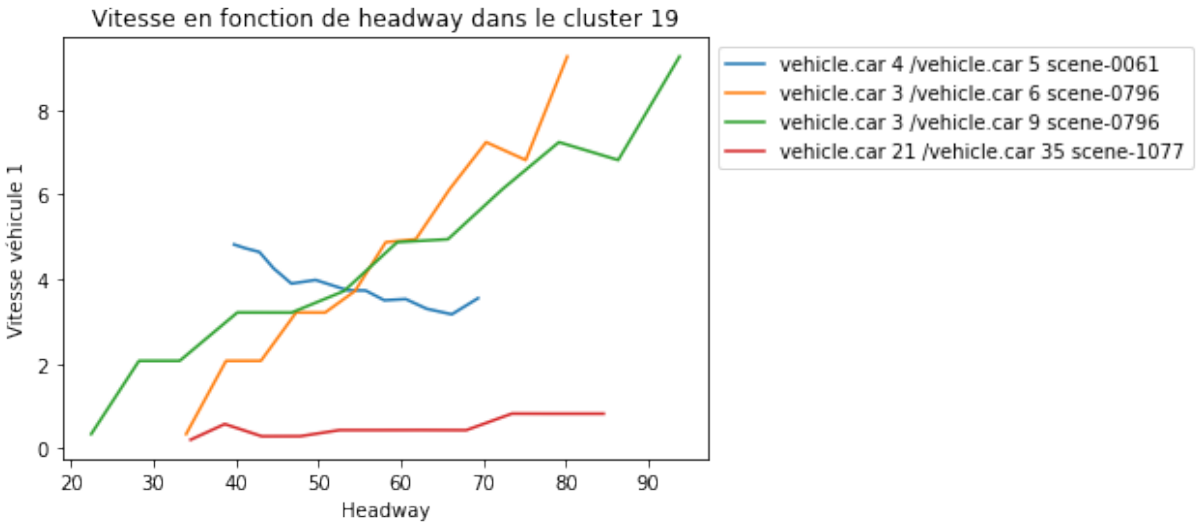


Figure 37 - Évolution de la vitesse en fonction de la distance inter-véhiculaire dans le cluster 19

Comme pour la distance parcourue, on n'observe pas de tendance par cluster, il est donc difficile de se servir de cet indicateur pour interpréter un cluster.

1.3.4 Indicateurs de trafic dans les clusters

Avec l'ensemble de ces indicateurs, on peut interpréter chacun des clusters un à un et identifier des comportements connus :

Type d'interaction	Clusters DBSCAN concernés
Car following	7, 8, 10, 40, 42, 60
Même sens sans car following	9, 13, 14, 19, 30, 31, 32, 34, 36, 38, 44, 45, 47, 51, 55, 61, 66
Véhicules en sens inverse	12, 15, 16, 17, 18, 24, 25, 37, 39, 41, 43, 46, 49, 50, 52, 53, 57, 62, 64, 65
Beaucoup de couple dans le cluster	0, 2, 11
Trajectoires qui se ressemblent, dans le même sens	1
Véhicule à l'arrêt	3, 59, 63
Trajectoires orthogonales	4, 5, 6, 20, 21, 22, 23, 27, 29, 56
Routes qui vont se croiser	35
Non classable	26, 28, 33, 48, 54, 58, -1

Comme on peut le voir avec les catégories formées, les contraintes de sens sont bien respectées dans un même cluster.

Manuellement, on identifie moins de clusters qu'avec DBSCAN. De plus, on pensait par exemple obtenir un seul cluster dédié au car following et pourtant ce n'est pas le cas, essayons d'utiliser une autre méthode de clustering afin de comparer les résultats.

2. K-means

DBSCAN fait partie des méthodes basées sur la densité, on choisit de travailler maintenant avec un autre type de méthode, une méthode basée sur la partition : k-means.

2.1 Détermination du nombre de cluster optimal

Comme nous l'avons vu précédemment, k-means nécessite un unique argument : le nombre de clusters k. Il existe une technique pour déterminer la valeur optimale de ce paramètre : la « elbow method ». Cette méthode consiste à réaliser le clustering pour plusieurs valeurs de k et pour chaque valeur, on calcule le score de distorsion, c'est-à-dire la somme des distances carrées de chaque point à son centroïde. On trace ensuite l'évolution de la distorsion suivant la valeur de k et on peut en déduire le k optimal :

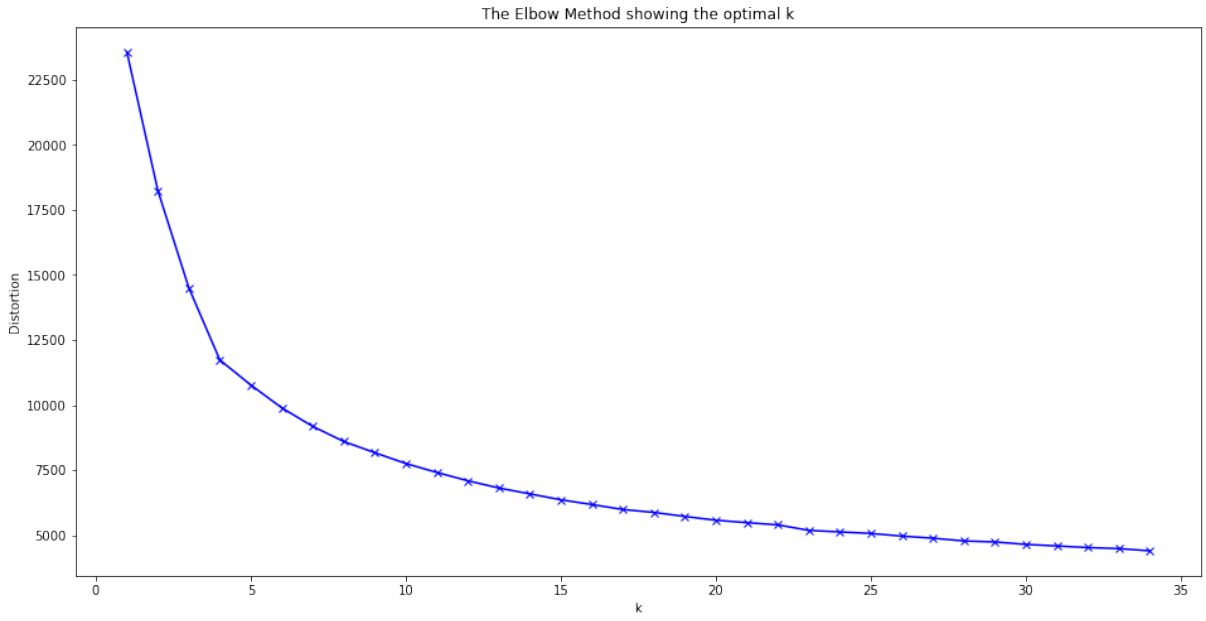


Figure 38 - Détermination du nombre de cluster optimal avec k-means

On choisit alors k = 20.

2.2 Comparaison avec DBSCAN

Le nombre de clusters avec la méthode k-means est beaucoup moins important que le nombre de clusters avec la méthode DBSCAN. De plus, les couples sont beaucoup plus répartis dans les clusters : contrairement à DBSCAN, il n'y a pas de cluster avec peu de couples.

On réalise les mêmes analyses que pour DBSCAN et on remarque que contrairement aux clusters de DBSCAN, à l'intérieur d'un même cluster, on retrouve des véhicules dans des sens différents. On ne voit de récurrence dans aucun indicateur de trafic pour un cluster donné. De plus, le nombre important d'individus par classe rend l'interprétation difficile.

Pour comparer d'avantage les résultats de k-means et DBSCAN, on regarde pour chaque cluster i de DBSCAN, à quel cluster de k-means appartiennent les individus du cluster i : sont-ils dans plusieurs clusters ?

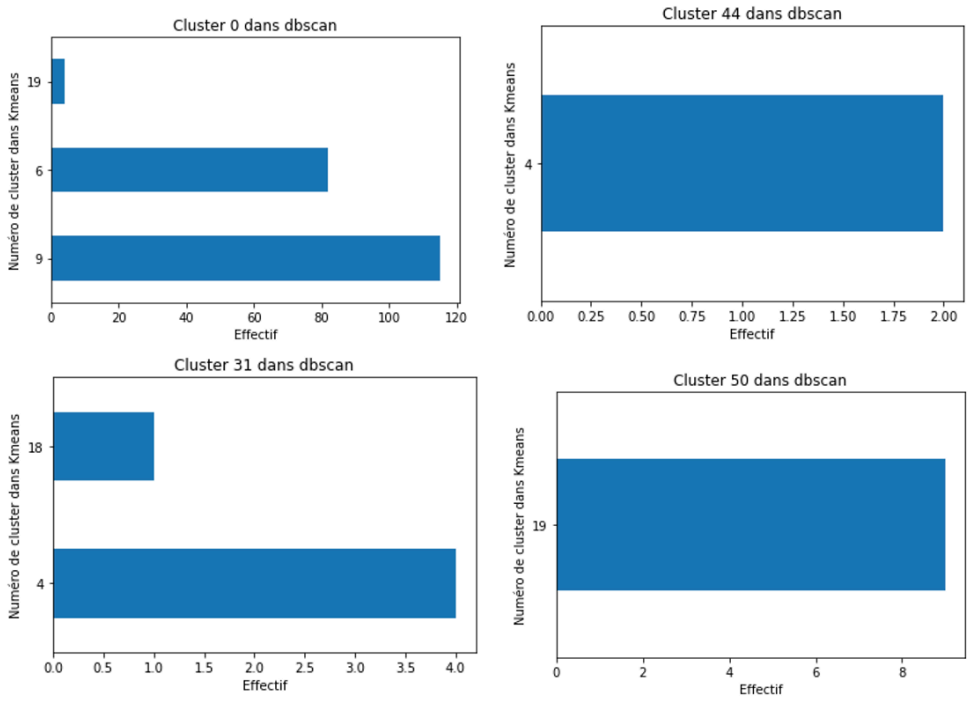


Figure 39 - Comparaison clusters k-means/DBSCAN

Ces graphiques signifient que :

- Pour le cluster 0 de DBSCAN : près de 110 individus de ce cluster sont dans le cluster 9 de k-means, 80 dans le cluster 6 et 5 dans le cluster 19
- Pour le cluster 31 : 4 individus de ce cluster sont dans le cluster 4 de k-means et 1 dans le cluster 18
- Pour le cluster 44 : tous les individus de ce cluster sont dans le cluster 4 de k-means
- Pour le cluster 50 : tous les individus de ce cluster sont dans le cluster 19 de k-means

La majorité des clusters DBSCAN sont inclus dans un seul cluster k-means. Cependant l'inclusion n'est pas forcément cohérente : après analyse, nous avons vu que le cluster 44, le

cluster 50 correspondaient à des véhicules circulant en sens inverse, pourtant ils ne font pas partie du même cluster dans k-means. De plus, les clusters 44 et 31 de DBSCAN ont des éléments dans le cluster 4 de k-means alors que le cluster 31 correspond à des véhicules dans le même sens.

L'algorithme k-means ne permet donc pas d'obtenir les résultats attendus et il a de moins bons résultats que l'algorithme DBSCAN. Testons alors une nouvelle méthode de clustering.

3. Ward

Nous avons choisi de travailler avec une classification hiérarchique ascendante : la méthode de Ward.

3.1 Détermination du nombre de cluster

Pour ce type de méthode, on peut déterminer le nombre de clusters en traçant le dendrogramme et en définissant un niveau de coupure du dendrogramme. Pour notre base de données, on obtient le dendrogramme suivant :

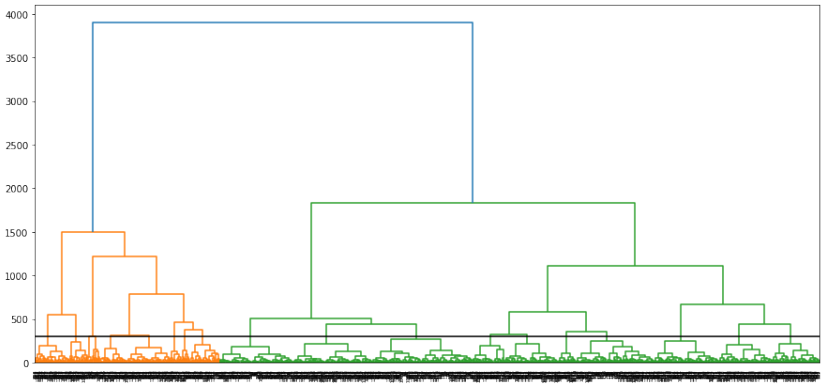


Figure 40 - Dendrogramme avec la méthode de ward

Le niveau de coupure choisi donne 18 clusters, ce qui est très proche du nombre de cluster optimal obtenu avec k-means mais beaucoup plus faible que celui obtenu avec DBSCAN.

3.2 Comparaison avec DBSCAN

Nous avons vu que les résultats produits avec l'algorithme DBSCAN étaient meilleurs que ceux avec l'algorithme k-means. On compare donc les résultats de Ward avec ceux de DBSCAN.

Dans ce cas, comme pour k-means, il est beaucoup plus difficile d'interpréter les résultats. Les couples sont répartis plus équitablement entre chaque cluster, ce qui fait que l'on a parfois dans les clusters des véhicules dans différents sens, difficiles à classer. De plus les résultats sont moins cohérents au niveau du sens, des valeurs des indicateurs…

Pour ce qui est des indicateurs de trafic on n'observe pas non plus de comportement similaire à l'intérieur d'un même cluster comme on peut le voir pour le cluster 11 :

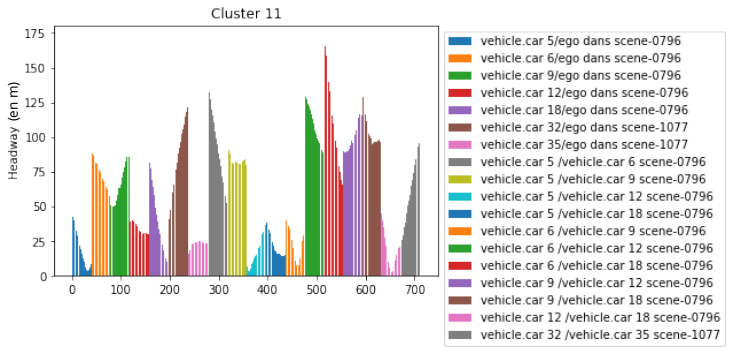


Figure 41 - Évolution de la distance inter-véhiculaire dans le cluster 11 Ward

On voit que la distance inter-véhiculaire des véhicules de cluster peut autant diminuer qu'augmenter au cours du temps, elle peut également être très faible comme élevée. Cela ne permet pas d'identifier des comportements connus.

On regarde pour chaque cluster i de DBSCAN, à quel cluster de ward appartiennent les individus du cluster i :

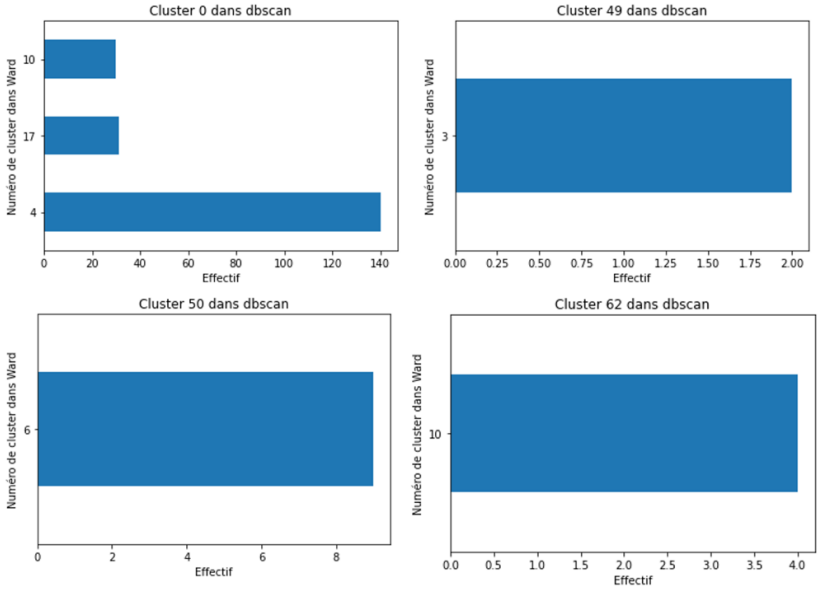


Figure 42 - Comparaison clusters Ward/DBSCAN

Comme pour k-means, la majorité des clusters DBSCAN sont inclus dans un seul cluster ward mais des éléments qui devraient être dans le même clusters ne le sont pas comme le cluster 50 dans DBSCAN et le cluster 49.

Face à ce constat et aux analyses faites auparavant, on conserve les résultats de DBSCAN comme étant les plus représentatifs de ce qu'on attendait du clustering. Il faut maintenant tenter d'aller vers une automatisation de la compréhension des clusters obtenus, on décide alors pour commencer de créer un clustering manuel des couples de trajectoire puis un clustering semi-automatique.

IX. Classification des clusters obtenus

1. Clustering manuel

Avec les clusters de DBSCAN, nous avions créé un tableau en associant chaque cluster à un type d'interaction. On part donc de ces résultats et on crée un nouveau clustering avec ce tableau comme suit :

Type d'interaction	Clusters DBSCAN concernés	Numéro de cluster manuel
Car following	7, 8, 10, 40, 42, 60	0
Même sens sans car following	9, 13, 14, 19, 30, 31, 32, 34, 36, 38, 44, 45, 47, 51, 55, 61, 66	1
Véhicules en sens inverse	12, 15, 16, 17, 18, 24, 25, 37, 39, 41, 43, 46, 49, 50, 52, 53, 57, 62, 64, 65	2
Beaucoup de couple dans le cluster	0, 2, 11	3
Trajectoires qui se ressemblent, dans le même sens	1	4
Véhicule à l'arrêt	3, 59, 63	5
Trajectoires orthogonales	4, 5, 6, 20, 21, 22, 23, 27, 29, 56	6
Routes qui vont se croiser	35	7
Non classable	26, 28, 33, 48, 54, 58, -1	8

On peut maintenant comparer ces nouvelles classes aux classes DBSCAN en s'intéressant aux centroïdes de chaque méthode. On calcule la distance entre les centroïdes des deux méthodes deux à deux et on représente cette distance dans une matrice dont les couleurs évoluent suivant la distance entre deux clusters.

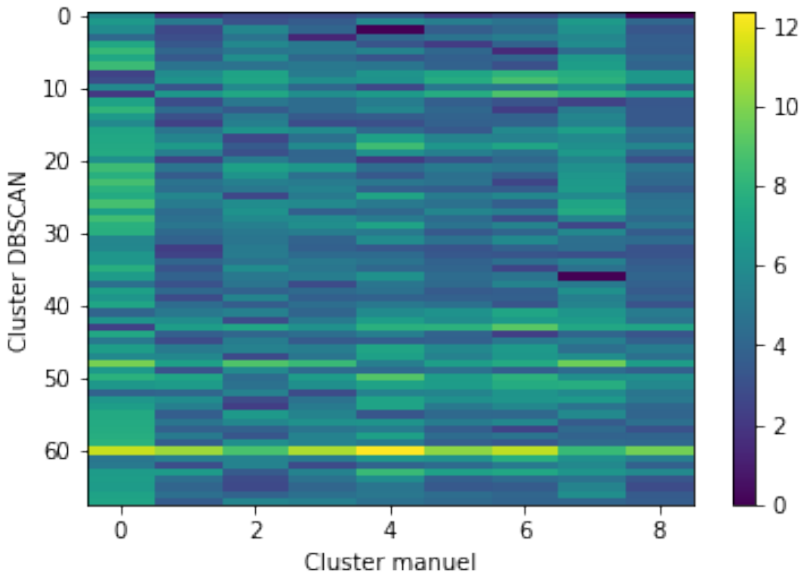


Figure 43 - Distance centroïdes manuel/DBSCAN

On associe alors chaque cluster DBSCAN à un cluster manuel : un cluster DBSCAN *i* est associé au cluster manuel dont le centroïde est le plus proche du centroïde *i*. Sur les 68 clusters DBSCAN, 32, soit 47% des clusters, sont mal associés par rapport au classement initialement établi. Cela montre que les classes créées manuellement par des rassemblements de classe DBSCAN ne sont pas correctes, leurs caractéristiques doivent diverger sur certains points et il est nécessaire de mettre des conditions moins subjectives pour déterminer leur signification. C'est ce que l'on a fait avec un clustering semi-automatique.

2. Clustering semi-automatique

2.1 Construction des clusters

Lors de l'ACP, nous avons vu que certains critères prédominent dans la base, nous allons donc utiliser ces critères pour classer les couples de trajectoires ensemble.

On travaille avec quatre catégories pour lesquelles on donne plusieurs valeurs possibles :

- Orthogonalité, trois valeurs sont possibles :
  - Orthogonale
  - Colinéaire
  - Variable
- Sens, quatre valeurs possibles :
  - Même sens
  - Sens inverse
  - Orthogonale
  - Variable

- Distance, trois valeurs possibles :
    - Petite distance
    - Moyenne distance
    - Grande distance
- Retard, quatre valeurs possibles :
    - Aucun retard
    - Retard faible
    - Retard moyen
    - Retard élevé

Pour fixer ces valeurs pour chaque couple de véhicules, on utilise les règles suivantes (cf Annexe 11) :

- Tout d'abord, on considère que des véhicules interagissent peu si leur « pourcentage d'interaction » en temps est inférieur à 10%, soit au maximum 2 secondes. Si c'est le cas, on met dans chaque catégorie la valeur « Inclassable ».
- Pour l'orthogonalité :
    - Si l'indicateur d'orthogonalité moyen est supérieur à 0.8 : la variable prend la valeur « Orthogonal »
    - Si l'indicateur d'orthogonalité est compris entre 0.2 et 0.8 non inclus : la variable prend la valeur « Variable »
    - Sinon la variable prend la valeur « Colinéaire »
- Pour le sens :
    - S'il vaut 1 : on sait que les véhicules sont dans le même sens, la variable prend donc la valeur « Même sens »
    - S'il vaut 0 : les véhicules circulent en sens inverse ou ont des trajectoires orthogonales, il suffit de regarder la valeur du critère d'orthogonalité pour trancher entre les deux
    - S'il est compris entre 0 et 1 non inclus : la valeur est « même sens »
- Pour la distance, on définit des intervalles à partir des valeurs de DTW dans toute la base :
    - Si la distance appartient au quantile 0.3 : la distance est faible
    - SI la distance est comprise dans le quantile 0.7 et supérieure au quantile 0.3 : la distance est moyenne
    - Si la distance est supérieure au quantile 0.7 : la distance est élevée
- On procède de la même façon pour le retard

On concatène ces quatre critères en un seul, on obtient une valeur pour chaque couple de notre base de données. Les valeurs uniques correspondent aux classes de ce nouveau clustering, il y en a 54 ce qui est proche du nombre de clusters obtenus avec DBSCAN.

2.2 Comparaison avec DBSCAN

On regarde pour chaque cluster de DBSCAN, la répartition de ces éléments par couple de véhicule :

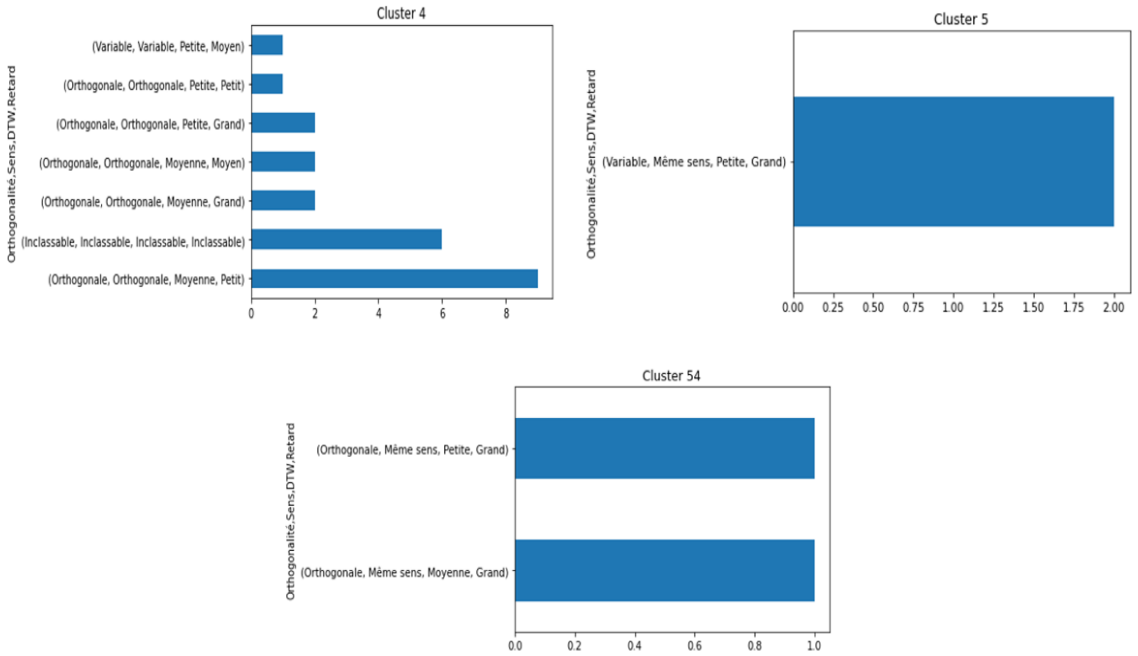


Figure 44 - Comparaison clusters semi-automatique/DBSCAN

Pour la majorité des clusters, il y a un unique quadruplet qui correspond aux couples et lorsque ce n'est pas le cas, les variabilités sont faibles.

On peut maintenant s'intéresser aux centroïdes de ces nouvelles classes et les comparer à ceux de DBSCAN. On calcule la distance entre les centroïdes des deux méthodes deux à deux et on associe chaque cluster DBSCAN à un cluster semi-automatique : un cluster DBSCAN *i* est associé au cluster semi-automatique dont le centroïde est le plus proche du centroïde *i*. Ci-dessous l'illustration de ce travail pour les premiers clusters DBSCAN :

	Cluster Dbscan	Cluster semi-automatique le plus proche	Classe correspondante (Orthogonalité, Sens, DTW, Retard)
0	-1	Cluster 17	Variable Variable Moyenne Grand
1	0	Cluster 0	Colinéaire Sens inverse Moyenne Moyen
2	1	Cluster 2	Variable Même sens Petite Grand
3	2	Cluster 7	Colinéaire Même sens Moyenne Petit
4	3	Cluster 20	Orthogonale Même sens Petite Grand
5	4	Cluster 9	Orthogonale Orthogonale Moyenne Petit
6	5	Cluster 2	Variable Même sens Petite Grand
7	6	Cluster 10	Variable Variable Petite Moyen
8	7	Cluster 4	Colinéaire Même sens Petite Grand
9	8	Cluster 5	Colinéaire Même sens Moyenne Grand
10	9	Cluster 35	Variable Même sens Petite Petit
11	10	Cluster 4	Colinéaire Même sens Petite Grand

Figure 45 - Correspondance des centroides DBSCAN/semi-automatique

Si on compare les caractéristiques associées à chaque cluster aux analyses faites manuellement sur les clusters DBSCAN, on se rend compte que les résultats sont cohérents. Cette méthode permet donc de nommer les caractéristiques d'un cluster automatiquement. Cela permet de faciliter la compréhension d'un cluster et les interactions auxquelles il correspond.

Ces résultats sont encourageants mais aucune des méthodes de clustering que nous avons utilisées nous a permis d'obtenir les résultats que l'on attendait. Cela est peut-être dû au fait que l'on agrège trop les informations sur les 20 secondes. En effet en 20 secondes un individu peut changer de comportement : il peut suivre un véhicule pendant 10 secondes puis tourner à une intersection pendant les 10 secondes restantes alors que le véhicule suivi continue son chemin tout droit. On peut tenter de désagréger les informations en passant à des descriptions plus fines : c'est ce qui est initié avec l'usage de la méthode « Latent Dirichlet Allocation (LDA) » introduite par la suite.

X. Piste d'exploration : la méthode Latent Dirichlet Allocation (LDA)

1. Le principe

L'idée de base est qu'on possède un corpus de plusieurs documents, chacun ayant des sujets propres, par exemple la science, la politique ou le sport ou même plusieurs sujets à la fois. On connaît le nombre de sujets dans le corpus mais on ne connaît pas le sujet de chacun de ces documents et on souhaiterait les classer par thème, c'est là qu'intervient l'algorithme LDA. [20][21]

On considère que chaque document est un « sac de mot » et ces mots appartiennent au vocabulaire des sujets abordés, cependant l'algorithme ne connaît pas la signification de ces mots, ni la signification des sujets. L'algorithme connaît uniquement le nombre de sujets dans notre corpus et il possède également, pour chaque sujet, le vocabulaire du thème et donc les mots les plus probablement associés à ce thème.

Pour un document donné, on associe chaque mot du document à un thème et on a alors la proportion de mot appartenant à chaque thème dans le document. Voici un exemple d'association de mots d'un texte à des thèmes donnés, extrait de [21] :

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

Figure 46 - Fonctionnement de LDA

On peut alors déterminer à quel thème appartient plus probablement ce document.

Cette méthode peut être utilisée dans d'autres domaines, notamment les transports. En 2014, elle a été utilisée dans l'article « Spatio temporal clustering analysis of segment to segment matrices : application to a large urban network » pour déterminer l'évolution du volume de trafic au cours du temps à l'aide de données Bluetooth dans la ville de Brisbane. D'après les auteurs : « L'idée principale derrière l'utilisation de modèles thématiques tels que LDA pour découvrir les modèles de mobilité sous-jacents est que l'utilisation d'un mode de transport peut être résumée par un ensemble fini de profils de demande, ou routines, encodés dans des modèles origine-destination typiques. ».

Dans notre cas, on sait que le nombre de types d'interactions est fini, on peut donc utiliser cette méthode pour les classer.

2. Application au clustering d'interactions

On a choisi d'utiliser cette méthode pour moins agréger les informations que l'on a sur chaque couple de véhicules. Il faut donc une dataframe avec des données au cours du temps pour chaque couple. Le nombre de points de chaque trajectoire n'étant pas identique, on va travailler avec les coordonnées de l'alignement optimal donné par la DTW.

De plus pour chacun de ces points, on va calculer 5 critères :

- La distance entre les deux points
- L'indicateur d'orthogonalité entre ces deux points : valeur absolue du sinus de l'angle entre les deux vecteurs vitesse instantanée à ces points
- L'indicateur de colinéarité entre ces deux points : valeur du cosinus de l'angle entre les deux vecteurs vitesse instantanée à ces points
- La vitesse du premier véhicule
- La vitesse du second

Pour chaque couple de véhicule, on obtient donc plusieurs quintuplés d'indicateurs à des instants différents. On obtient alors une base de données de la forme suivante :

	Couple de trajectoire	Distance	Sinus	Cosinus	Vitesse véhicule 1	Vitesse véhicule 2
0	vehicle.bus.rigid 1/ego dans scene-0061	53.479254	0.020871	-0.999782	9.135392	9.736836
1	vehicle.bus.rigid 1/ego dans scene-0061	53.928887	0.015877	-0.999874	9.113071	9.736836
2	vehicle.bus.rigid 1/ego dans scene-0061	54.377268	0.021460	-0.999770	9.104178	9.736836
3	vehicle.bus.rigid 1/ego dans scene-0061	54.830140	0.023996	-0.999712	9.067166	9.736836
4	vehicle.bus.rigid 1/ego dans scene-0061	55.285332	0.030690	-0.999529	9.101840	9.736836
...	...	...	...	...	...	...
38021	vehicle.motorcycle 1 /vehicle.motorcycle 2 sce...	70.524364	0.999047	0.043643	6.531769	11.687041
38022	vehicle.motorcycle 1 /vehicle.motorcycle 2 sce...	73.354761	0.999894	0.014555	6.531769	11.685370
38023	vehicle.motorcycle 1 /vehicle.motorcycle 2 sce...	76.099012	0.999894	0.014555	6.532070	11.685370
38024	vehicle.motorcycle 1 /vehicle.motorcycle 2 sce...	78.882958	0.999995	0.003060	6.530783	11.685370
38025	vehicle.motorcycle 1 /vehicle.motorcycle 2 sce...	81.720409	0.999995	0.003060	6.530587	11.685370

Figure 47 - Dataframe avec des indicateurs au cours du temps pour chaque couple d'individus

Pour le modèle LDA, il faut deux éléments : un corpus de document et des mots. Le corpus de document correspond ici à l'ensemble des couples d'individus. Il faut maintenant créer des mots et pour cela on va créer des classes basées sur les 5 critères utilisés.

On créer plusieurs classes pour chaque critères, basées sur les distributions observées dans la dataframe précédente :

- Pour la distance, on choisit les intervalles suivants :
  - [0,10]
  - [10,20]
  - [20,30]
  - [30,40]
  - [40,50]
  - Plus de 50
- Pour l'indicateur d'orthogonalité, on choisit les intervalles suivants :
  - [0, 0.25]
  - [0.25, 0.5]
  - [0.5, 0.75]
  - [0.75, 1]
- Pour l'indicateur de colinéarité, on choisit les intervalles suivants :
  - [-1,-0.5]
  - [-0.5, 0]
  - [0, 0.5]
  - [0.5, 1]
- Pour la vitesse du véhicule 1 :
  - [0,5]
  - [5,10]
  - [10,15]
  - [15,20]
  - [20,25]
  - [25,30]
  - [30,35]
- Pour la vitesse du véhicule 2 :
  - [0,5]
  - [5,10]
  - [10,15]
  - [15,20]
  - [20,25]
  - [25,30]

Dans la base de données précédente, on remplace les vraies valeurs des indicateurs par les intervalles auxquels ils appartiennent. On concatène ces 5 intervalles et les valeurs uniques de ces concaténations vont représenter les mots de notre modèle LDA, il y en a 640.

On crée alors une nouvelle base de données avec en colonne les mots et en ligne les individus et la fréquence d'apparition dans chaque 'mot' pour l'individu correspondant. La somme de chaque ligne vaut donc 1. La base de données est de la forme suivante :

tout	(50, 10000]; (0.0, 0.25]; (-1.0, -0.5]; (5, 10]; (5, 10]	(20, 30]; (0.0, 0.25]; (-1.0, -0.5]; (5, 10]; (5, 10]	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (0, 5]	(10, 20]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (0, 5]	(20, 30]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (0, 5]	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (5, 10]	(10, 20]; (0.5, 0.75]; (0.5, 1.0]; (0, 5]; (5, 10]	(0, 10]; (0.25, 0.5]; (0.5, 1.0]; (5, 10]; (5, 10]	(10, 20]; (0.75, 1.0]; (0.0, 0.5]; (0, 5]; (5, 10]	(10, 20]; (0.75, 1.0]; (0.5, 1.0]; (0, 5]; (5, 10]	...	(0, 10]; (0.75, 1.0]; (0.5, 1.0]; (10, 15]; (5, 10]	(0, 10]; (0.75, 1.0]; (0.0, 0.5]; (10, 15]; (0, 5]	(10, 20]; (0.75, 1.0]; (-0.5, 0.0]; (10, 15]; (0, 5]	(10, 20]; (0.75, 1.0]; (-0.5, 0.0]; (15, 35]; (0, 5]	(10, 20]; (0.5, 0.75]; (-1.0, -0.5]; (10, 15]; (0, 5]
vehicle.bus.rigid 1/ego dans scene-0061	1.0	0.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 1/ego dans scene-0061	0.0	1.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 2/ego dans scene-0061	0.0	0.0	0.197479	0.12605	0.105042	0.084034	0.084034	0.079832	0.071429	0.071429	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 3/ego dans scene-0061	0.0	0.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 4/ego dans scene-0061	0.0	0.0	0.125984	0.00000	0.000000	0.089239	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
vehicle.car 11 /vehicle.motorcycle 1 scene-1100	0.0	0.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 11 /vehicle.motorcycle 2 scene-1100	0.0	0.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0
vehicle.car 12 /vehicle.motorcycle 1 scene-1100	0.0	0.0	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0

Figure 48 - Extrait de la base de données finale LDA

On peut alors appliquer à cette base de données, le module 'LatentDirichletAllocation' et déterminer le nombre de sujet c'est-à-dire de classe d'interaction ici. On trouve 75 classes, ce qui est proche de nos 68 clusters DBSCAN. Intéressons-nous plus particulièrement à ces classes.

3. Analyse des clusters

3.1 Les topics et 'mots' associés

L'algorithme LDA nous donne pour chaque individu la probabilité d'appartenir à chacun des 'Topics' soit des clusters. On associe alors chaque individu au topic auquel il a la plus grande probabilité d'appartenir. On regarde alors la répartition du nombre de couple par topic :

```
count    75.000000
mean     17.520000
std      16.847776
min       1.000000
25%       7.000000
50%      12.000000
75%      21.500000
max      70.000000
Name: Nombre de couples dans le topic, dtype: float64
```

Figure 49 - Répartition du nombre d'individu par topic

Il y a au total 1341 couples d'individus et au maximum, un topic contient 70 couples soit 5% des couples, les individus sont donc bien répartis dans les clusters, ce qui peut laisser penser que l'interprétation des clusters sera plus facile.

On s'intéresse maintenant à la signification de chaque topic. LDA nous donne pour chaque topic, le poids de chaque 'mot', ici quintuplé d'indicateurs, dans le topic.

On représente une matrice qui en abscisse correspond aux mots et en ordonnées aux topics, ses couleurs évolue en fonction du poids des mots dans le topic :

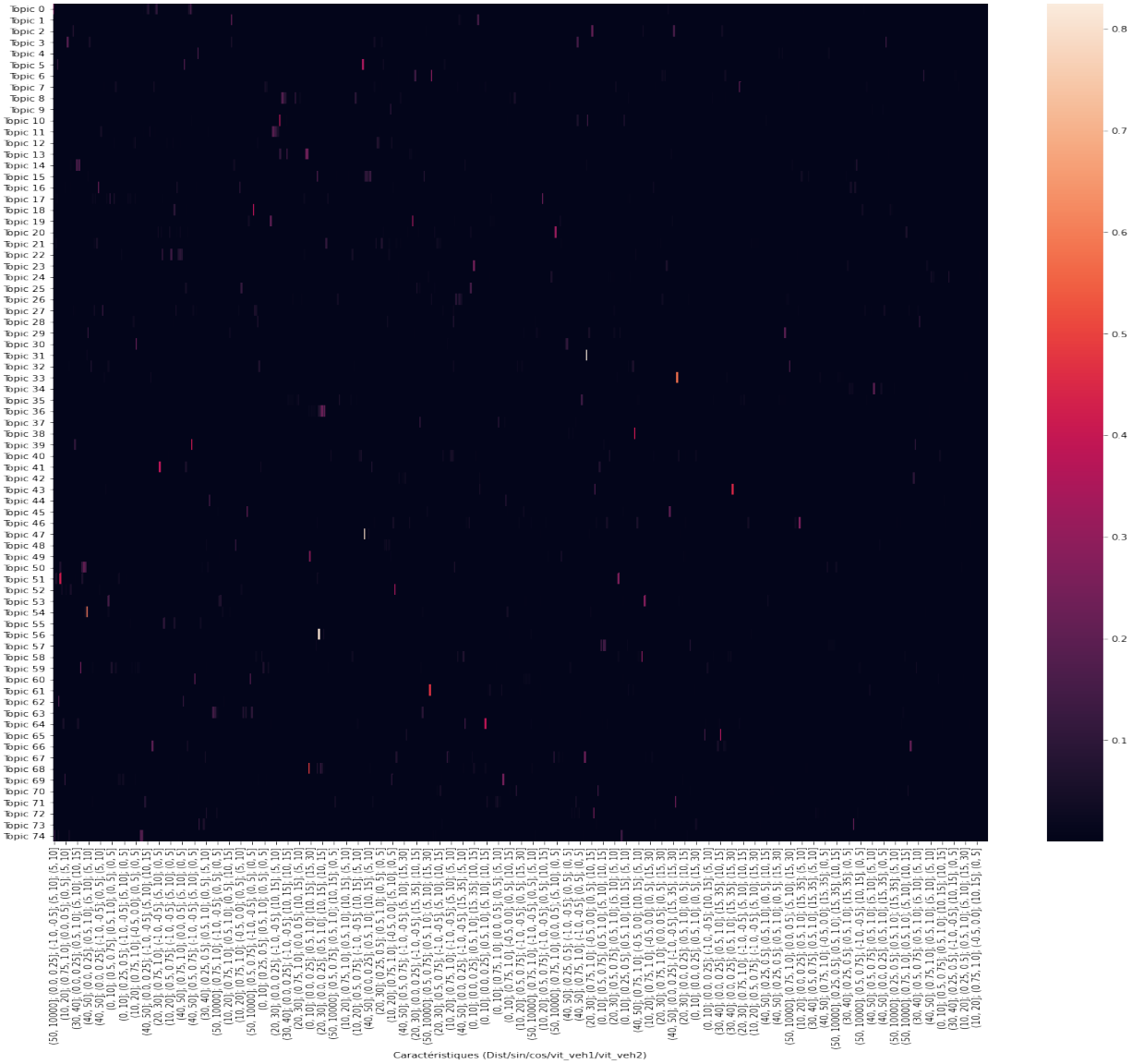


Figure 50 – Répartition du poids des 'mots' dans les topics

On voit que pour la majorité des topics, c'est souvent un ou deux 'mots' qui prédominent dans la constitution du topic.

On associe alors à chaque topic, le mot qui prédomine le plus dans sa constitution. On remarque que chaque topic est associé à un mot différent. Les classes sont donc bien distinctes, même si certains mots sont proches.

3.2 Le car following dans les topics LDA

Intéressons-nous de nouveau à la classe car following pour évaluer la qualité des clusters LDA : pour la trentaine de couples que l'on avait identifiée comme faisant partie de la classe car following, regardons s'ils ont été placés dans le ou les mêmes topics et les caractéristiques de ces topics.

La répartition du nombre d'individus faisant parti de la classe car following par topics est la suivante :

```
Topic dominant
49                5
59                5
50                4
68                4
51                2
19                1
22                1
43                1
71                1
dtype: int64
```

Figure 51 - Répartition des couples dans les clusters

On peut voir que cinq clusters prédominent. Regardons leurs caractéristiques pour savoir s'ils sont comparables et se distinguent uniquement par un critère. Ci-dessous, le 'mot' auquel correspond les topics :

	Catégorie dominante
Topic 19	(50, 10000]; (0.0, 0.25]; (-1.0, -0.5]; (15, 35]; (10, 15]
Topic 22	(20, 30]; (0.75, 1.0]; (0.0, 0.5]; (5, 10]; (0, 5]
Topic 43	(50, 10000]; (0.0, 0.25]; (0.5, 1.0]; (15, 35]; (5, 10]
Topic 49	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (10, 15]; (15, 30]
Topic 50	(20, 30]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (5, 10]
Topic 51	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (5, 10]; (5, 10]
Topic 59	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (0, 5]; (0, 5]
Topic 68	(0, 10]; (0.0, 0.25]; (0.5, 1.0]; (10, 15]; (10, 15]
Topic 71	(50, 10000]; (0.25, 0.5]; (0.5, 1.0]; (10, 15]; (5, 10]

Figure 52 - Correspondance topic/'mot'

On peut voir que les topics 49, 51, 59 et 68 sont très proches, leur différence réside uniquement dans les vitesses des véhicules. Ce sont les clusters dans lesquels les individus du groupe car following sont les plus nombreux, il y a donc de la cohérence dans la répartition des individus.

C'est un résultat encourageant qui nécessiterait davantage de recherches. Pour poursuivre les analyses il faudrait regarder le mot associé à chaque topic et le comparer à celui des autres classes et observer ceux qui sont proches. Il faudrait également, comme pour DBSCAN regarder les caractéristiques des individus d'un même topic pour identifier le type d'interaction auquel il correspond. Cependant, l'analyse avec autant de mot n'est pas aisée, il faudrait certainement réduire le nombre de mot ou développer une méthode d'analyse plus automatisée et plus fine.

Partie 4 : Recommandations, conclusions et perspectives

Ce travail sur la base nuScenes nous a ainsi montré que ce type de base de données regorge d'informations exploitables et utiles pour la compréhension du trafic. En effet, nous avons vu qu'il était possible d'extraire toutes les trajectoires des éléments qui entourent le véhicule autonome, même celles des piétons et également la position d'objets fixes dans l'environnement comme des cônes de chantier. L'accès à ces données est un premier pas vers leur utilisation et leur nouvelle exploitation. Pour ce qui est de l'exploitation, nous avons tenté de mettre à profit ces données pour classifier des interactions et ainsi permettre de comprendre la dynamique du trafic en temps réel.

Pour la classification, le travail a commencé par la création d'indicateurs sur des couples d'individus, le but étant d'en créer un maximum pour décrire le plus précisément possible l'interaction. Un des indicateurs qui est apparu évident dès le début est la similarité entre deux trajectoires. Nous avons déterminé cela à l'aide de la DTW qui effectue un alignement optimal de deux trajectoires et calcule la somme de la distance euclidienne entre les couples de points successifs de cet alignement.

L'objectif était ensuite d'utiliser ces indicateurs pour classer les interactions. Le clustering de trajectoire est un sujet qui a déjà été traité et qui se base effectivement en partie sur une mesure de similarité. Cependant, notre travail se distingue par le fait que la majorité des clustering de trajectoires travaille sur des individus et non sur des couples d'individus.

Le clustering d'interaction n'ayant jamais été réalisé, nous avons testé plusieurs types de méthodes pour voir laquelle donnerait les meilleurs résultats. L'interprétation des clusters a parfois été compliquée et ce par le manque de cohérence de certains regroupements. Les méthodes DBSCAN, k-means et Ward n'ont pas réellement porté leurs fruits, on ne parvient pas à obtenir des associations similaires à celles obtenues de façon manuelle, même si DBSCAN semblait donner des résultats plus similaires à ceux attendus. On peut expliquer ce manque de cohérence par le fait que l'on a résumé à chaque fois au maximum 20 secondes d'interaction par des indicateurs uniques alors que pendant cette période le mouvement de deux véhicules peut complètement changer : ils peuvent se suivre un temps puis partir chacun dans des directions opposées. Pour remédier à ce problème, nous avons pensé à conserver davantage d'informations sur chaque interaction à l'aide de la méthode LDA. L'analyse des résultats produits par cette méthode n'est pas terminée. Dans ce document, nous présentons uniquement le début de l'implémentation de cette méthode, il faut encore l'explorer davantage en analysant plus les topics obtenus et leur cohérence. Ce travail ne sera pas simple puisque le nombre de mots est très élevé : il semble que ce que l'on gagne en précision de classification avec cette méthode, on le perd en facilité d'interprétation. Il faudra donc trouver un moyen d'automatiser les analyses si cela est possible.

Si cette méthode fonctionne bien, la suite pourrait être de donner automatiquement une signification à chaque cluster. Ce travail pourrait se baser sur un travail fait manuellement sur les résultats que nous avons déjà. La cohérence des résultats pourra ensuite être vérifiée en travaillant sur les autres scènes de la base nuScenes.

Bibliographie

[1] A. Geiger, P. Lenz, C. Stiller, et R. Urtasun, « Vision meets robotics: The KITTI dataset », *Int. J. Robot. Res.*, vol. 32, n° 11, p. 1231-1237, sept. 2013, doi: 10.1177/0278364913491297.

[2] M.-F. Chang *et al.*, « Argoverse: 3D Tracking and Forecasting With Rich Maps », in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, juin 2019, p. 8740-8749. doi: 10.1109/CVPR.2019.00895.

[3] P. Sun *et al.*, « Scalability in Perception for Autonomous Driving: Waymo Open Dataset », *ArXiv191204838 Cs Stat*, mai 2020, Consulté le: août 13, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1912.04838

[4] H. Caesar *et al.*, « nuScenes: A multimodal dataset for autonomous driving », *ArXiv190311027 Cs Stat*, mai 2020, Consulté le: avr. 01, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1903.11027

[5] N. Magdy, M. A. Sakr, T. Mostafa, et K. El-Bahnasy, « Review on trajectory similarity measures », in *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, Cairo, Abbassia, Egypt, déc. 2015, p. 613-619. doi: 10.1109/IntelCIS.2015.7397286.

[6] S. Sankararaman, P. K. Agarwal, T. Mølhave, et A. P. Boedihardjo, « Computing Similarity between a Pair of Trajectories », *ArXiv13031585 Cs*, mars 2013, Consulté le: avr. 23, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1303.1585

[7] H. Wang, H. Su, K. Zheng, S. Sadiq, et X. Zhou, « An Effectiveness Study on Trajectory Similarity Measures », *Database Technol.*, vol. 137, p. 10, 2013.

[8] E. J. Keogh et M. J. Pazzani, « Scaling up Dynamic Time Warping for Datamining Applications », p. 5.

[9] D. Folgado, M. Barandas, R. Matias, R. Martins, M. Carvalho, et H. Gamboa, « Time Alignment Measurement for Time Series », *Pattern Recognit.*, vol. 81, p. 268-279, sept. 2018, doi: 10.1016/j.patcog.2018.04.003.

[10] E. J. Keogh et M. J. Pazzani, « Derivative Dynamic Time Warping », in *Proceedings of the 2001 SIAM International Conference on Data Mining*, avr. 2001, p. 1-11. doi: 10.1137/1.9781611972719.1.

[11] H. Rakha, C. C. Pecker, et H. B. B. Cybis, « Calibration Procedure for Gipps Car-Following Model », *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1999, n° 1, p. 115-127, janv. 2007, doi: 10.3141/1999-13.

[12] R. Jiang, M.-B. Hu, H. M. Zhang, Z.-Y. Gao, B. Jia, et Q.-S. Wu, « On some experimental features of car-following behavior and how to model them », *Transp. Res. Part B Methodol.*, vol. 80, p. 338-354, oct. 2015, doi: 10.1016/j.trb.2015.08.003.

[13] M. Y. Choong, L. Angeline, R. K. Yin Chin, K. Beng Yeo, et K. T. Kin Teo, « Modeling of vehicle trajectory clustering based on LCSS for traffic pattern extraction », in *2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, Kota Kinabalu, oct. 2017, p. 74-79. doi: 10.1109/I2CACIS.2017.8239036.

[14] G. Yuan, P. Sun, J. Zhao, D. Li, et C. Wang, « A review of moving object trajectory clustering algorithms », *Artif. Intell. Rev.*, vol. 47, n° 1, p. 123-144, janv. 2017, doi: 10.1007/s10462-016-9477-7.

[15] B. Han, L. Liu, et E. Omiecinski, « NEAT: Road Network Aware Trajectory Clustering », in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, Macau, China, juin 2012, p.

142-151. doi: 10.1109/ICDCS.2012.31.

[16] J. Bian, D. Tian, Y. Tang, et D. Tao, « A survey on trajectory clustering analysis », *ArXiv180206971 Cs*, févr. 2018, Consulté le: août 18, 2021. [En ligne]. Disponible sur: http://arxiv.org/abs/1802.06971

[17] Zhouyu Fu, Weiming Hu, et Tieniu Tan, « Similarity based vehicle trajectory clustering and anomaly detection », in *IEEE International Conference on Image Processing 2005*, Genova, Italy, 2005, p. II-602. doi: 10.1109/ICIP.2005.1530127.

[18] M. Ester, H.-P. Kriegel, et X. Xu, « A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise », p. 6.

[19] J. Kim et H. S. Mahmassani, « Spatial and Temporal Characterization of Travel Patterns in a Traffic Network Using Vehicle Trajectories », *Transp. Res. Procedia*, vol. 9, p. 164-184, 2015, doi: 10.1016/j.trpro.2015.07.010.

[20] E. Come, N. A. Randriamanamihaga, L. Oukhellou, et P. Aknin, « Spatio-temporal Analysis of Dynamic Origin-Destination Data Using Latent Dirichlet Allocation: Application to Vélib' Bike Sharing System of Paris », p. 20.

[21] Blei, D. M., Ng, A. Y., & Jordan, M. I, « Latent dirichlet allocation », the *Journal of machine Learning research*, 3, p. 993-1022, 2003

Sitographie

[A] IFSTTAR, Regards croisés sur le véhicule autonome [en ligne]. Disponible sur : https://www.ifsttar.fr/fileadmin/redaction/dossiers-thematiques/Mobilites/vehicule_autonome/10169_MOB_VA_FR_interactif.pdf (Consulté le 16/09/2021)

[B] Actu-Juridique.fr, Pleins phares sur les évolutions du cadre légal et réglementaire des véhicules autonomes [en ligne]. Disponible sur : https://www.actu-juridique.fr/transports/pleins-phares-sur-les-evolutions-du-cadre-legal-et-reglementaire-des-vehicules-autonomes/ (Consulté le 16/09/2021)

ANNEXES

Annexe 1 : Fonction immobile/mobile

```
# On crée une fonction qui renvoie la liste des rangs dans les listes X,Y des objets immobiles de
# cette liste et donc d'une même catégorie, ainsi que la liste des objets mobiles

def immobile(X,Y):
    Immo = []
    Mobile = []
    n = len(X) # Nombre d'élément dans la catégorie
    for i in range(0,n-1):
        X_coord = X[i]
        Y_coord = Y[i]
        if len(X_coord)==1 :
            Immo.append(i)
        else :
            d = 0
            for k in range(0,len(X_coord)-1):
                d+= sqrt((X_coord[k+1]-X_coord[k])**2+(Y_coord[k+1]-Y_coord[k])**2)
            if d<2 : # Si la distance totale parcourue par l'objet est inférieure à 2m -> objet immobile
                Immo.append(i)
            else :
                Mobile.append(i)
    return Immo,Mobile
```

Annexe 2 : Tracer la trajectoire du véhicule ego et de son environnement

In [ ]:

```
def traj_ego_environment(scene_name):

    scene_token = nusc.field2token('scene','name',scene_name)[0]
    sample_token = nusc.field2token('sample','scene_token',scene_token)# Ensemble des token des échantillons de la scene

    # Pour tracer la trajectoire du véhicule ego
    Liste_ego_poses = get_poses(scene_token)
    X_coord =[]
    Y_coord =[]
    Z_coord =[]
    for k in range(0,len(Liste_ego_poses)):
        X_coord.append(Liste_ego_poses[k]['translation'][0])
        Y_coord.append(Liste_ego_poses[k]['translation'][1])
        Z_coord.append(Liste_ego_poses[k]['translation'][2])
    # Lieu de départ du véhicule ego
    X1 = X_coord[0]
    Y1 = Y_coord[0]

    # Ensemble des sample_annotation de la scène :
    sample_annotation_token = []
    for k in sample_token :
        L = nusc.field2token('sample_annotation','sample_token',k)
        sample_annotation_token = sample_annotation_token + L

    # On créé la liste des token des instances présents dans les sample_annotation
    L_instance_token = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_instance_token.append(exemple['instance_token'])
    # Puisque pour un même objet, on peut avoir plusieurs annotations, on supprime les doublons de L
    L_instance_token = list(set(L_instance_token))

    # Liste des catégories d'objet dans la scène
    L_category_name = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_category_name.append(exemple['category_name'])
    L_category_name = list(set(L_category_name)) #on supprime les doublons

    # On crée une liste de 23 couleurs car il y a au maximum 23 catégories dans une scène
    c = ['g','darkorange','firebrick','c','indigo','k','sandybrown','purple','olive',\
         'hotpink','grey','rosybrown','chartreuse','magenta','gold','lavender','aqua','darkkhaki',\
         'lightgrey','pink','palegreen','navy','violet']

    # Boucle sur chaque catégorie trouvée
    for j in range(0,len(L_category_name)):
        couleur_catego = c[j]
        catego_name = L_category_name[j]
        catego_token = nusc.field2token('category','name',catego_name)
        category = nusc.get('category',catego_token[0])

        # Ensemble des instances appartenant à la fois à la scène et à la catégorie
        L_instance_category = nusc.field2token('instance','category_token',catego_token[0]) # instance dans la catégorie
        L_instance_category = [x for x in L_instance_category if x in L_instance_token]# et dans la scène
        L_instance_category

        # Il peut y avoir plusieurs éléments de la catégorie dans la scène, ici chaque ligne de L représente
        # les token des sample_annotation pour un élément de cette catégorie.

        L=[]
        for k in L_instance_category :
            instance_category = nusc.get('instance',k)
            nbr_annotations = instance_category['nbr_annotations']
            if nbr_annotations == 1:
                L.append([instance_category['first_annotation_token']])
            else :
                H = []
                first_token = instance_category['first_annotation_token']
                last_token = instance_category['last_annotation_token']
                current_token = first_token

                s = 0
                while s!= nbr_annotations :
                    current_ann = nusc.get('sample_annotation', current_token)
                    H.append(current_ann['token'])
                    current_token = current_ann['next'] #On passe au suivant
                    s+=1
                L.append(H)

        # Maintenant qu'on a les token de tous les sample_annotation pour tous les éléments d'une même catégorie,
        # on va pouvoir récupérer leurs positions et tracer leurs trajectoires

        X = []
        Y = []
        Z = []
        for k in L :
            x = []
            y = []
            z = []
            for j in k :
                sample_annotation = nusc.get('sample_annotation',j)
                x.append(sample_annotation['translation'][0])
                y.append(sample_annotation['translation'][1])
                z.append(sample_annotation['translation'][2])
            X.append(x)
            Y.append(y)
            Z.append(z)

        fig = plt.figure(1, figsize=(10, 10))

        # On crée les listes des objets immobiles et mobiles de la catégorie
        Immo, Mobile = immobile(X,Y)
        if len(Immo)>0 :
            plt.plot(X[Immo[0]][0],Y[Immo[0]][0],'x',label=catego_name +' immobile',color=couleur_catego)
        if len(Mobile)>0 :
            plt.plot(X[Mobile[0]],Y[Mobile[0]],label=catego_name +' mobile',color=couleur_catego)
        for i in range(0,len(X)):
            if i in Immo :
                plt.plot(X[i][0],Y[i][0],'x',color=couleur_catego)
            else :
                plt.plot(X[i],Y[i],color=couleur_catego)

    plt.plot(X1,Y1,'x',label='Départ',color='r')
    plt.plot(X_coord,Y_coord,label ='Trajectoire véhicule ego',color='cornflowerblue')
    plt.legend(bbox_to_anchor=(1.5, 1.0),loc='best')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Trajectoire des éléments dans la '+ scene_name)
    plt.show()
```

Annexe 3 : Fonction suiveur

```python
import numpy as np

def suiveur(X_ego,Y_ego,X_veh,Y_veh):
    V1 = np.array([X_ego[0]-X_veh[0],Y_ego[0]-Y_veh[0]])
    V2 = np.array([X_veh[1]-X_veh[0],Y_veh[1]-Y_veh[0]])
    PC = np.vdot(V1,V2)
    if PC >0 :
        return 0 # Le véhicule suit le véhicule ego
    elif PC<0 :
        return 1 # Le véhicule ego suit le véhicule
    else :
        return 2 # On est parallèle au départ

# La fonction permet de savoir si un véhicule donné dans la scène suit ou non le véhicule ego à partir
# des coordonnées des deux véhicules dans l'espace. La fonction renvoie :
# - 0 si le véhicule suit le véhicule ego
# - 1 si le vehicule est suivit pas le véhicule égo
# - 2 si les véhicules sont parallèles au départ
```

Annexe 5 : Matrice des distances dans le temps

```python
import numpy as np
from math import sqrt

# Fonction qui renvoie la matrice des distances entre
# les pts qui interagissent dans le temps

def M_distance_time(X_ego,Y_ego,T_ego,X_veh,Y_veh,T_veh):

    debut = max(T_ego[0],T_veh[0]) # Temps début detection vehicule 1
    fin = min(T_ego[len(T_ego)-1],T_veh[len(T_veh)-1]) # Temps fin detection vehicule 1

    # On va créer la liste des indices des élements du veh 1 qui sont en même temps que le veh 2
    L_indice_ego = []
    for k in range(0,len(T_ego)):
        if debut<=T_ego[k]<=fin:
            L_indice_ego.append(k)

    # Pareil pour le veh
    L_indice_veh = []
    for k in range(0,len(T_veh)):
        if debut<=T_veh[k]<=fin:
            L_indice_veh.append(k)

    M = np.zeros((len(L_indice_veh),len(L_indice_ego)))
    for i in range(0,len(L_indice_veh)):
        for j in range(0,len(L_indice_ego)):
            M[i,j] = sqrt((X_veh[L_indice_veh[i]]-X_ego[L_indice_ego[j]])**2+(Y_veh[L_indice_veh[i]]-Y_ego[L_indice_ego[j]])**2)

    return M
```

Annexe 6 : DTW sur fenêtre glissante

In [ ]:

```
## Fonction qui renvoie une dataframe avec les valeurs de dtw calculés sur des intervalles de temps d'une seconde :

#imports nécessaires
import matplotlib.pyplot as plt
from math import sqrt
import datetime
from datetime import timedelta
from dtw import *
import pandas as pd

def dtw_glissante(scene_name):

    # On crée une dataframe
    df = pd.DataFrame()
    L_intervalle = [[k,k+1] for k in range(0,20)]
    df['index']= L_intervalle
    df.set_index('index',inplace=True)
    df.index.name = None

    scene_token = nusc.field2token('scene','name',scene_name)[0]
    sample_token = nusc.field2token('sample','scene_token',scene_token)# Ensemble des token des échantillons de la scene

    # Pour tracer la trajectoire du véhicule ego
    Liste_ego_poses = get_poses(scene_token)
    X_coord =[]
    Y_coord =[]
    Z_coord =[]
    T_timestamp = [] # liste des timestamps
    for k in range(0,len(Liste_ego_poses)):
        X_coord.append(Liste_ego_poses[k]['translation'][0])
        Y_coord.append(Liste_ego_poses[k]['translation'][1])
        Z_coord.append(Liste_ego_poses[k]['translation'][2])
        T_timestamp.append(Liste_ego_poses[k]['timestamp'])

    # Ensemble des sample_annotation de la scène :
    sample_annotation_token = []
    for k in sample_token :
        L = nusc.field2token('sample_annotation','sample_token',k)
        sample_annotation_token = sample_annotation_token + L

    # On créé la liste des token des instances présents dans les sample_annotation
    L_instance_token = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_instance_token.append(exemple['instance_token'])
    # Puisque pour un même objet, on peut avoir plusieurs annotations, on supprime les doublons de L
    L_instance_token = list(set(L_instance_token))

    # Liste des catégories d'objet dans la scène
    L_category_name = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_category_name.append(exemple['category_name'])
    L_category_name = list(set(L_category_name)) #on supprime les doublons

    L_vehicule = ['vehicle.bicycle','vehicle.bus.bendy','vehicle.bus.rigid','vehicle.car','vehicle.construction','vehicle.emergency.ambulance','vehicle.emergency.police

    L_category_name = [k for k in L_category_name if k in L_vehicule]

    # Boucle sur chaque catégorie trouvée
    for j in range(0,len(L_category_name)):
        catego_name = L_category_name[j]
        catego_token = nusc.field2token('category','name',catego_name)
        category = nusc.get('category',catego_token[0])

        # Ensemble des instances appartenant à la fois à la scène et à la catégorie
        L_instance_category = nusc.field2token('instance','category_token',catego_token[0]) # instance dans la catégorie
        L_instance_category = [x for x in L_instance_category if x in L_instance_token]# et dans la scène
        L_instance_category

        # Il peut y avoir plusieurs éléments de la catégorie dans la scène, ici chaque ligne de L représente
        # les token des sample_annotation pour un élément de cette catégorie.

        L=[]
        for k in L_instance_category :
            instance_category = nusc.get('instance',k)
            nbr_annotations = instance_category['nbr_annotations']
            if nbr_annotations == 1:
                L.append([instance_category['first_annotation_token']])
            else :
                H = []
                first_token = instance_category['first_annotation_token']
                last_token = instance_category['last_annotation_token']
                current_token = first_token

                s = 0
                while s!= nbr_annotations :
                    current_ann = nusc.get('sample_annotation', current_token)
                    H.append(current_ann['token'])
                    current_token = current_ann['next'] #On passe au suivant
                    s+=1
                L.append(H)

        # Maintenant qu'on a les token de tous les sample_annotation pour tous les éléments d'une même catégorie,
        # on va pouvoir récupérer leurs positions et tracer leurs trajectoires

        X = []
        Y = []
        Z = []
        T_time = []
        for k in L :
            x = []
            y = []
            z = []
            t=[]
            for j in k :
                sample_annotation = nusc.get('sample_annotation',j)
                x.append(sample_annotation['translation'][0])
                y.append(sample_annotation['translation'][1])
                z.append(sample_annotation['translation'][2])
                t.append(nusc.get('sample',sample_annotation['sample_token'])['timestamp'])
            X.append(x)
            Y.append(y)
            Z.append(z)
            T_time.append(t)

        # On crée les listes des objets immobiles et mobiles de la catégorie
        Immo, Mobile = immobile(X,Y)

        # On conserve uniquement les objets mobiles
        X_mobile=[X[i] for i in Mobile]
        Y_mobile=[Y[i] for i in Mobile]
        Z_mobile=[Z[i] for i in Mobile]
        T_mobile = [T_time[i] for i in Mobile]

        if len(X_mobile)>0:

            for i in range(0,len(X_mobile)):

                L_df = []
                # On fait une boucle sur les intervalles de temps
                for k in range(0,20):
                    current_intervalle = [k,k+1]
                    # On crée les listes des coordonnées qu'on va garder ie qui sont dans l'intervalle de temps ci-dessus
                    Lx_pt_veh = []
                    Ly_pt_veh = []
                    Lt_pt_veh = []
                    Lx_pt_ego = []
                    Ly_pt_ego = []
                    Lt_pt_ego = []

                    # On remplit avec les pts du véhicule dont le timestamp est dans l'intervalle
                    for s in range(len(X_mobile[i])):
                        tveh = (datetime.datetime.fromtimestamp(T_mobile[i][s]/1e6)-datetime.datetime.fromtimestamp(T_timestamp[0]/ 1e6)).total_seconds()
                        if tveh <= current_intervalle[1] and tveh >= current_intervalle[0] :
                            Lx_pt_veh.append(X_mobile[i][s])
                            Ly_pt_veh.append(Y_mobile[i][s])
                            Lt_pt_veh.append(T_mobile[i][s])



                    # On remplit avec les pts du véhicule ego dont le timestamp est dans l'intervalle
                    for s in range(len(X_coord)):
                        tego = (datetime.datetime.fromtimestamp(T_timestamp[s]/1e6)-datetime.datetime.fromtimestamp(T_timestamp[0]/ 1e6)).total_seconds()
                        if tego <= current_intervalle[1] and tego >= current_intervalle[0]:
                            Lx_pt_ego.append(X_coord[s])
                            Ly_pt_ego.append(Y_coord[s])
                            Lt_pt_ego.append(T_timestamp[s])


                    if len(Lx_pt_veh)==0:
                        L_df.append(np.nan)
                    else :
                        #M_distance = M_distance_time(Lx_pt_ego,Ly_pt_ego,Lt_pt_ego,Lx_pt_veh,Ly_pt_veh,Lt_pt_veh)
                        M_distance = Matrice_distance(Lx_pt_ego,Ly_pt_ego,Lx_pt_veh,Ly_pt_veh)
                        alignment = dtw(M_distance)
                        distance = alignment.normalizedDistance
                        L_df.append(distance)


                nom = catego_name +' '+str(i+1)
                df[nom]= L_df

    return df
```

Annexe 7 : Matrice produit scalaire

```
def Matrice_produit_scalaire(X_coord,Y_coord,T_timestamp,X_veh,Y_veh,T_veh):


    debut = max(T_timestamp[0],T_veh[0]) # Temps début detection vehicule 1
    fin = min(T_timestamp[len(T_timestamp)-1],T_veh[len(T_veh)-1]) # Temps fin detection vehicule 1

    # On va créer la liste des indices des élements du veh 1 qui sont en même temps que le veh 2
    L_indice_ego = []
    for k in range(0,len(T_timestamp)):
        if debut<=T_timestamp[k]<=fin:
            L_indice_ego.append(k)

    # Pareil pour le veh
    L_indice_veh = []
    for k in range(0,len(T_veh)):
        if debut<=T_veh[k]<=fin:
            L_indice_veh.append(k)

    # On crée la liste des vecteurs pour le véhicule ego avec les points qu'on peut garder
    V_ego = []
    for k in range(0,len(L_indice_ego)-1):
        t2 = datetime.datetime.fromtimestamp(T_timestamp[L_indice_ego[k+1]]/ 1e6)
        t1 = datetime.datetime.fromtimestamp(T_timestamp[L_indice_ego[k]]/ 1e6)
        t = (t2-t1).total_seconds()
        v = np.array([X_coord[L_indice_ego[k+1]]-X_coord[L_indice_ego[k]],Y_coord[L_indice_ego[k+1]]-Y_coord[L_indice_ego[k]]])/t
        # Le vecteur a pour norme la vitesse à l'instant k
        V_ego.append(v)


    # On crée la liste des vecteurs pour le veh avec les points qu'on peut garder
    V = []
    for k in range(0,len(L_indice_veh)-1):
        t2 = datetime.datetime.fromtimestamp(T_veh[L_indice_veh[k+1]]/ 1e6)
        t1 = datetime.datetime.fromtimestamp(T_veh[L_indice_veh[k]]/ 1e6)
        t = (t2-t1).total_seconds()
        v = np.array([X_veh[L_indice_veh[k+1]]-X_veh[L_indice_veh[k]],Y_veh[L_indice_veh[k+1]]-Y_veh[L_indice_veh[k]]])/t
        V.append(v)

    # On crée la matrice des produits scalaires :
    M = np.zeros((len(V),len(V_ego)))
    for i in range(0,len(V)):
        for j in range(0,len(V_ego)):
            M[i,j] = np.dot(V[i],V_ego[j])/(np.linalg.norm(V[i])*np.linalg.norm(V_ego[j]))
    return M
```

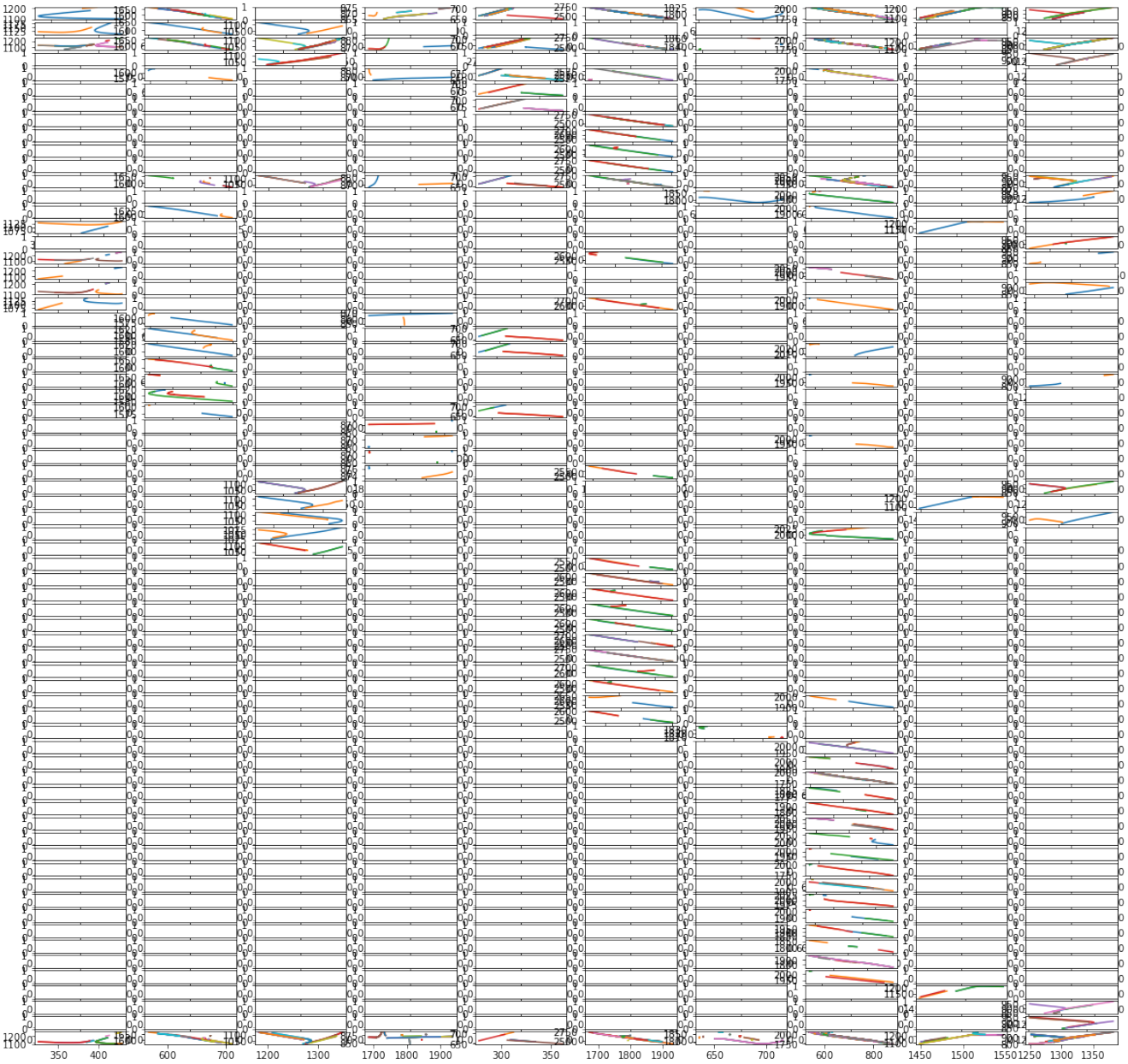
Annexe 8 : Calcul des indicateurs entre le véhicule ego et tous les autres véhicules de la scène

In [ ]:

```
#Importe nécessaires
import matplotlib.pyplot as plt
from math import sqrt
import datetime
from datetime import timedelta
from dtw import *
import pandas as pd
from statistics import mean

def indicateur_similarite(scene_name):

    # On crée une dataframe vide
    df = pd.DataFrame()
    # Ligne de la dataframe
    L_ligne = ['scene','Catégorie Véhicule 1','Numéro Véhicule 1','Catégorie Véhicule 2',\
               'Numéro Véhicule 2','Vitesse moyenne veh 1','Vitesse moyenne veh 2',\
               'Durée d'interaction','DTW normalisée','Distance instantannée moyenne',\
               'Distance instantannée minimale','Distance instantannée maximale','TimeLag moyen horizontal',\
               'TimeLag moyen vertical','Indicateur d'orthogonalité moyen','Indicateur d'orthogonalité min',\
               'Indicateur d'orthogonalité max','Indicateur de colinéarité moyen',\
               'Indicateur de colinéarité min','Indicateur de colinéarité max','Sens',\
               'Distance minimale dans matrice distance','Distance maximale dans matrice distance',\
               'Distance moyenne dans matrice distance']
    # Les lignes en plus :  Time log moyen'
    # On met ça en index
    df['index']= L_ligne
    df.set_index('index',inplace=True)
    df.index.name = None

    # Dataframe des fenêtre glissantes
    df_glissante = dtw_glissante(scene_name)

    scene_token = nusc.field2token('scene','name',scene_name)[0]
    sample_token = nusc.field2token('sample','scene_token',scene_token)# Ensemble des token des échantillons de la scene

    # Pour tracer la trajectoire du véhicule ego
    Liste_ego_poses = get_poses(scene_token)
    X_coord =[]
    Y_coord =[]
    Z_coord =[]
    T_timestamp = [] # liste des timestamps
    for k in range(0,len(Liste_ego_poses)):
        X_coord.append(Liste_ego_poses[k]['translation'][0])
        Y_coord.append(Liste_ego_poses[k]['translation'][1])
        Z_coord.append(Liste_ego_poses[k]['translation'][2])
        T_timestamp.append(Liste_ego_poses[k]['timestamp'])

    # Ensemble des sample_annotation de la scène :
    sample_annotation_token = []
    for k in sample_token :
        L = nusc.field2token('sample_annotation','sample_token',k)
        sample_annotation_token = sample_annotation_token + L

    # On créé la liste des token des instances présents dans les sample_annotation
    L_instance_token = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_instance_token.append(exemple['instance_token'])
    # Puisque pour un même objet, on peut avoir plusieurs annotations, on supprime les doublons de L
    L_instance_token = list(set(L_instance_token))

    # Liste des catégories d'objet dans la scène
    L_category_name = []
    for k in sample_annotation_token :
        exemple = nusc.get('sample_annotation', k)
        L_category_name.append(exemple['category_name'])
    L_category_name = list(set(L_category_name)) #On supprime les doublons

    L_vehicule = ['vehicle.bicycle','vehicle.bus.bendy','vehicle.bus.rigid','vehicle.car','vehicle.construction','vehicle.emergency.ambulance','vehicle.emergency.police

    L_category_name = [k for k in L_category_name if k in L_vehicule]
    L_category_name.sort()

    # Boucle sur chaque catégorie trouvée
    for j in range(0,len(L_category_name)):
        catego_name = L_category_name[j]
        catego_token = nusc.field2token('category','name',catego_name)
        category = nusc.get('category',catego_token[0])

        # Ensemble des instances appartenant à la fois à la scène et à la catégorie
        L_instance_category = nusc.field2token('instance','category_token',catego_token[0]) # instance dans la catégorie
        L_instance_category = [x for x in L_instance_category if x in L_instance_token]# et dans la scène
        L_instance_category

        # Il peut y avoir plusieurs éléments de la catégorie dans la scène, ici chaque ligne de L représente
        # les token des sample_annotation pour un élément de cette catégorie.

        L=[]
        for k in L_instance_category :
            instance_category = nusc.get('instance',k)
            nbr_annotations = instance_category['nbr_annotations']
            if nbr_annotations == 1:
                L.append([instance_category['first_annotation_token']])
            else :
                H = []
                first_token = instance_category['first_annotation_token']
                last_token = instance_category['last_annotation_token']
                current_token = first_token

                a = 0
                while a!= nbr_annotations :
                    current_ann = nusc.get('sample_annotation', current_token)
                    H.append(current_ann['token'])
                    current_token = current_ann['next'] #On passe au suivant
                    a+=1
                L.append(H)

        # Maintenant qu'on a les token de tous les sample_annotation pour tous les éléments d'une même catégorie,
        # on va pouvoir récupérer leurs positions et tracer leurs trajectoires

        X = []
        Y = []
        Z = []
        T_time = []
        for k in L :
            x = []
            y = []
            z = []
            t=[]
            for j in k :
                sample_annotation = nusc.get('sample_annotation',j)
                x.append(sample_annotation['translation'][0])
                y.append(sample_annotation['translation'][1])
                z.append(sample_annotation['translation'][2])
                t.append(nusc.get('sample',sample_annotation['sample_token'])['timestamp'])
            X.append(x)
            Y.append(y)
            Z.append(z)
            T_time.append(t)

        # On crée les listes des objets immobiles et mobiles de la catégorie
        Immo, Mobile = immobile(X,Y)

        # On conserve uniquement les objets mobiles
        X_mobile=[X[i] for i in Mobile]
        Y_mobile=[Y[i] for i in Mobile]
        Z_mobile=[Z[i] for i in Mobile]
        T_mobile = [T_time[i] for i in Mobile]

        for i in range(0,len(X_mobile)):
            L_objet = []
            # nom du véhicule
            nom = catego_name +' '+str(i+1)

            ## Scene
            L_objet.append(scene_name)

            ## Catégorie Véhicule 1
            L_objet.append('vehicle.ego')

            ## Numéro véhicule 1
            L_objet.append(1)

            ## Catégoie Véhicule 2
            L_objet.append(catego_name)

            ## Catégoie Véhicule 2
            L_objet.append(i+1)

            ## Vitesses moyennes
            v_moy_ego, v_moy_veh = vitesse_moyenne(X_coord,Y_coord,T_timestamp,X_mobile[i],Y_mobile[i],T_mobile[i])
            L_objet.append(v_moy_ego)
            L_objet.append(v_moy_veh)

            ## Durée d'interaction
            debut = max(T_mobile[i][0],T_timestamp[0])
            fin = min(T_mobile[i][-1],T_timestamp[-1])
            temps_ego = (datetime.datetime.fromtimestamp(T_timestamp[-1]/1e6)-datetime.datetime.fromtimestamp(T_timestamp[0]/ 1e6)).total_seconds()
            t_total = (datetime.datetime.fromtimestamp(fin/1e6)-datetime.datetime.fromtimestamp(debut/ 1e6)).total_seconds()
            L_objet.append(t_total/temps_ego)
            #L_objet.append(t_total)

            ## DTW normalisée
            #M_distance = Matrice_distance(X_coord,Y_coord,X_mobile[i],Y_mobile[i])
            M_distance = M_distance_time(X_coord,Y_coord,T_timestamp,X_mobile[i],Y_mobile[i],T_mobile[i])
            alignment = dtw(M_distance)
            distance = alignment.normalizedDistance
            L_objet.append(distance)

            Chemin = [(alignment.index1[k],alignment.index2[k]) for k in range(0,len(alignment.index1))]

            ## Distance instannée moyenne
            L_objet.append(df_glissante[nom].mean())

            ## Distance instannée minimale
            L_objet.append(df_glissante[nom].min())

            ## Distance instannée maximale
            L_objet.append(df_glissante[nom].max())

            ## Time log moyen horizontal
            recurrence_0 = list(alignment.index1).count(0)
            indice_max_ego = alignment.index2[recurrence_0-1]
            t_horizontal = (datetime.datetime.fromtimestamp(T_timestamp[indice_max_ego]/1e6)-datetime.datetime.fromtimestamp(T_timestamp[0]/ 1e6)).total_seconds()
            L_objet.append(t_horizontal)

            ## Time log moyen vertical
            N = max(alignment.index2)
            recurrence_N = list(alignment.index2).count(N)
            indice_min_veh = alignment.index1[len(alignment.index1)-recurrence_N]
            t_vertical = (datetime.datetime.fromtimestamp(T_mobile[i][max(alignment.index1)]/1e6)-datetime.datetime.fromtimestamp(T_mobile[i][indice_min_veh]/ 1e6)).tot
            L_objet.append(t_vertical)

            ## Indicateur d'orthogonalité moyen : sin
            M_pv = Matrice_produit_vectoriel(X_coord,Y_coord,T_timestamp,X_mobile[i],Y_mobile[i],T_mobile[i])
            L_pv = [M_pv[i,j] for(i,j) in Chemin[:len(Chemin)] if i<max(alignment.index1) and j<max(alignment.index2)]
            L_objet.append(mean(L_pv))

            ## Indicateur d'orthogonalité min
            L_objet.append(min(L_pv))

            ## Indicateur d'orthogonalité max
            L_objet.append(max(L_pv))

            ## Indicateur de colinéarité moyen : cos
            M_ps = Matrice_produit_scalaire(X_coord,Y_coord,T_timestamp,X_mobile[i],Y_mobile[i],T_mobile[i])
            L_ps = [M_ps[i,j] for(i,j) in Chemin[:len(Chemin)] if i<max(alignment.index1) and j<max(alignment.index2)]
            L_objet.append(mean(L_ps))

            ## Indicateur de colinéarité min
            L_objet.append(min(L_ps))

            ## Indicateur de colinéarité max
            L_objet.append(max(L_ps))

            ## Sens
            if mean(L_ps)>0:
                L_objet.append(1)
            else :
                L_objet.append(0)

            ## Distance min dans la matrice
            L_objet.append(M_distance.min())

            ## Distance max dans la matrice
            L_objet.append(M_distance.max())

            ## Distance moyenne dans la matrice
            L_objet.append(M_distance.mean())

            nom = catego_name +' '+str(i+1)+'/ego dans '+scene_name
            df[nom]=L_objet
    return df.transpose()
```

Annexe 9 : Résultats des clusters dbscan sur les 10 scènes

Chaque colonne correspond à une scène et chaque ligne correspond à un cluster.



Annexe 10 : Tracé des trajectoires dans un cluster donné

In [ ]:
```
#On trace le résultat du clustering par cluster

def cluster_dbscan_total_cluster(df,epsilon,nb_voisin,num_cluster):

    # ON supprime les Nan de la df
    df = df.dropna()

    # On normalise les indicateurs
    df[list(df.columns[5:])] = StandardScaler().fit_transform(df[list(df.columns[5:])])

    labels = cluster (df[list(df.columns[5:])],epsilon,nb_voisin)
    df['clusters']=labels

    # On garde uniquement la scène voulue
    df = df[df['clusters']==num_cluster]
    liste_scene = list(df['scene'].unique())

    # Liste des labels uniques et dans l'ordre
    labels_unique = sorted(list(set(df['clusters'].values)))

    nb_scene = len(df['scene'].unique())


    fig, axs = plt.subplots(3) # Ici on aura 10 scènes pour commencer
    #fig.suptitle("Tracé des clusters obtenus avec l'algorithme DSCAN")
    plt.gcf().set_size_inches(20, 20)
    #plt.gcf().subplots_adjust(left = 0.3, bottom = 0.3,right = 0.7, top = 0.9, wspace = 0, hspace = 0.5)

    counter = -1
    for s in labels_unique:

        # On s'occupe d'un cluster
        df_new = df[df['clusters']==s]

        # On classe les lignes par scènes
        df_new = df_new.sort_values(by=['scene'])
        # On cherche les scènes dans le cluster
        scene_in_cluster = df_new['scene'].unique()

        for h in range(0,len(scene_in_cluster[:3])) :
            # ON s'occupe d'une scene

            # On récupère la place de cette scène dans l'ordre
            num_scene = liste_scene.index(scene_in_cluster[h])

            df_bis = df_new[df_new['scene']==scene_in_cluster[h]]


            nb_ligne = len(df_bis)

            Texte_legend = 'Couple de trajectoires \n'

            for j in range(0,nb_ligne):
                current_ligne = df_bis.iloc[j]

                scene_name = current_ligne['scene']
                veh1 = current_ligne['Catégorie Véhicule 1'] # catégorie du véhicule 1
                veh2 = current_ligne['Catégorie Véhicule 2']# ici on a le nom de la catégorie
                num_veh2 = current_ligne['Numéro Véhicule 2']-1 # Numéro du véhicule 2

                scene_token = nusc.field2token('scene','name',scene_name)[0]
                sample_token = nusc.field2token('sample','scene_token',scene_token)# Ensemble des token des échantillons de la scene

                # Ensemble des sample_annotation de la scène :

                sample_annotation_token = []
                for k in sample_token :
                    L = nusc.field2token('sample_annotation','sample_token',k)
                    sample_annotation_token = sample_annotation_token + L

                # On créé la liste des token des instances présents dans les sample_annotation
                L_instance_token = []
                for k in sample_annotation_token :
                    exemple = nusc.get('sample_annotation', k)
                    L_instance_token.append(exemple['instance_token'])
                # Puisque pour un même objet, on peut avoir plusieurs annotations, on supprime les doublons de L
                L_instance_token = list(set(L_instance_token))

                ## Véhicule 1
                if veh1 == 'vehicle.ego':
                    num_veh1_legend = ''
                    Liste_ego_poses = get_poses(scene_token)
                    X1 =[]
                    Y1 =[]
                    Z1 =[]
                    T1 = [] # liste des timestamps
                    for k in range(0,len(Liste_ego_poses)):
                        X1.append(Liste_ego_poses[k]['translation'][0])
                        Y1.append(Liste_ego_poses[k]['translation'][1])
                        Z1.append(Liste_ego_poses[k]['translation'][2])
                        T1.append(Liste_ego_poses[k]['timestamp'])
                else :
                    num_veh1 = current_ligne['Numéro Véhicule 1']-1 #numéro du véhicule dans la liste de tous les véh de cette catégo
                    num_veh1_legend = str(num_veh1+1)

                    catego_name = veh1
                    catego_token = nusc.field2token('category','name',catego_name)
                    category = nusc.get('category',catego_token[0])

                    # Ensemble des instances appartenant à la fois à la scène et à la catégorie
                    L_instance_category = nusc.field2token('instance','category_token',catego_token[0]) # instance dans la catégorie
                    L_instance_category = [x for x in L_instance_category if x in L_instance_token]# et dans la scène

                    L=[]
                    for k in L_instance_category :
                        instance_category = nusc.get('instance',k)
                        nbr_annotations = instance_category['nbr_annotations']
                        if nbr_annotations == 1:
                            L.append([instance_category['first_annotation_token']])
                        else :
                            H = []
                            first_token = instance_category['first_annotation_token']
                            last_token = instance_category['last_annotation_token']
                            current_token = first_token

                            i = 0
                            while i!= nbr_annotations :
                                current_ann = nusc.get('sample_annotation', current_token)
                                H.append(current_ann['token'])
                                current_token = current_ann['next'] #On passe au suivant
                                i+=1
                            L.append(H)
                    # Il peut y avoir plusieurs éléments de la catégorie dans la scène, ici chaque ligne de L représente
                    # les token des sample_annotation pour un élément de cette catégorie.

                    # Maintenant qu'on a les token de tous les sample_annotation pour tous les éléments d'une même catégorie,
                    # on va pouvoir récupérer leurs positions

                    X = []
                    Y = []
                    Z = []
                    T_time = []
                    for k in L :
                        x = []
                        y = []
                        z = []
                        t = []
                        for i in k :
                            sample_annotation = nusc.get('sample_annotation',i)
                            x.append(sample_annotation['translation'][0])
                            y.append(sample_annotation['translation'][1])
                            z.append(sample_annotation['translation'][2])
                            t.append(nusc.get('sample',sample_annotation['sample_token'])['timestamp'])
                        X.append(x)
                        Y.append(y)
                        Z.append(z)
                        T_time.append(t)

                    Immo, Mobile = immobile(X,Y)

                    # On conserve uniquement les objets mobiles
                    X_mobile=[X[i] for i in Mobile]
                    Y_mobile=[Y[i] for i in Mobile]
                    Z_mobile=[Z[i] for i in Mobile]
                    T_mobile = [T_time[i] for i in Mobile]

                    X1 =X_mobile[num_veh1]
                    Y1 =Y_mobile[num_veh1]
                    Z1 =Z_mobile[num_veh1]
                    T1 =T_mobile[num_veh1]

                catego_name = veh2
                catego_token = nusc.field2token('category','name',catego_name)
                category = nusc.get('category',catego_token[0])

                # Ensemble des instances appartenant à la fois à la scène et à la catégorie
                L_instance_category = nusc.field2token('instance','category_token',catego_token[0]) # instance dans la catégorie
                L_instance_category = [x for x in L_instance_category if x in L_instance_token]# et dans la scène

                L=[]
                for k in L_instance_category :
                    instance_category = nusc.get('instance',k)
                    nbr_annotations = instance_category['nbr_annotations']
                    if nbr_annotations == 1:
                        L.append([instance_category['first_annotation_token']])
                    else :
                        H = []
                        first_token = instance_category['first_annotation_token']
                        last_token = instance_category['last_annotation_token']
                        current_token = first_token

                        i = 0
                        while i!= nbr_annotations :
                            current_ann = nusc.get('sample_annotation', current_token)
                            H.append(current_ann['token'])
                            current_token = current_ann['next'] #On passe au suivant
                            i+=1
                        L.append(H)
                # Il peut y avoir plusieurs éléments de la catégorie dans la scène, ici chaque ligne de L représente
                # les token des sample_annotation pour un élément de cette catégorie.

                # Maintenant qu'on a les token de tous les sample_annotation pour tous les éléments d'une même catégorie,
                # on va pouvoir récupérer leurs positions

                X = []
                Y = []
                Z = []
                T_time = []
                for k in L :
                    x = []
                    y = []
                    z = []
                    t = []
                    for i in k :
                        sample_annotation = nusc.get('sample_annotation',i)
                        x.append(sample_annotation['translation'][0])
                        y.append(sample_annotation['translation'][1])
                        z.append(sample_annotation['translation'][2])
                        t.append(nusc.get('sample',sample_annotation['sample_token'])['timestamp'])
                    X.append(x)
                    Y.append(y)
                    Z.append(z)
                    T_time.append(t)

                Immo, Mobile = immobile(X,Y)

                # On conserve uniquement les objets mobiles
                X_mobile=[X[i] for i in Mobile]
                Y_mobile=[Y[i] for i in Mobile]
                Z_mobile=[Z[i] for i in Mobile]
                T_mobile = [T_time[i] for i in Mobile]

                X2 =X_mobile[num_veh2]
                Y2 =Y_mobile[num_veh2]
                Z2 =Z_mobile[num_veh2]
                T2 =T_mobile[num_veh2]

                # print(counter)

                axs[h].plot(X1,Y1)
                axs[h].text(X1[0], Y1[0],veh1 + ' '+num_veh1_legend)#,fontsize=8)
                axs[h].plot(X2,Y2)
                axs[h].text(X2[0], Y2[0],veh2 + ' '+str(num_veh2+1))#,fontsize=8)
                axs[h].set_title('Cluster '+str(s)+' dans '+ scene_name)
                axs[h].plot([],[],'w',label = veh1 + ' '+num_veh1_legend +'/'+veh2 + ' '+str(num_veh2+1))
                axs[h].legend(loc='best')

    plt.tight_layout()
    plt.show()
```

Annexe 11 : Classification semi-automatique

In [ ]:

```
# Fonction qui renvoit pour chaque élément une classe selon différents critères

def labelling_auto_dbscan(df,epsilon,nb_voisin):

    df_sortante = pd.DataFrame() # La df qu'on va renvoyer

    # ON supprime les Nan de la df
    df = df.dropna()

    # Les individus
    L_indiv = list(df.index)
    df_sortante['Individus'] = L_indiv

    # Les clusters
    df_bis = df.copy()
    df_bis[list(df_bis.columns[5:])] = StandardScaler().fit_transform(df_bis[list(df_bis.columns[5:])])
    labels = cluster (df_bis[list(df_bis.columns[5:])],epsilon,nb_voisin)
    df_sortante['Cluster Dbscan'] = labels

    # Préparation des liste pour la df_sortante
    L_ortho = []
    L_sens = []
    L_distance = []
    L_retard = []

    # Quantile pour la DTW
    quantile_30_DTW = df['DTW normalisée'].quantile(0.3)
    quantile_70_DTW = df['DTW normalisée'].quantile(0.7)
    # Quantile pour timelag
    quantile_30_retard = df['TimeLag moyen vertical'].quantile(0.3)
    quantile_70_retard = df['TimeLag moyen vertical'].quantile(0.7)

    # Boucle sur chaque indiv
    for i in range(0,len(L_indiv)):

        Ligne = df.iloc[i:i+1]

        if Ligne["Durée d'interaction"][0]<=0.1:
            L_ortho.append('Inclassable')
            L_sens.append('Inclassable')
            L_distance.append('Inclassable')
            L_retard.append('Inclassable')

        else :
            # Critère d'orthogonalité
            if Ligne["Indicateur d'orthogonalité moyen"][0]>=0.8 :
                L_ortho.append('Orthogonale')
            elif 0.2 < Ligne["Indicateur d'orthogonalité moyen"][0]<0.8 :
                L_ortho.append('Variable')
            else :
                L_ortho.append('Colinéaire')

            # Critère de sens
            if Ligne['Sens'][0]==1:
                L_sens.append('Même sens')
            else :
                if L_ortho[-1]=='Orthogonale':
                    L_sens.append('Orthogonale')
                elif L_ortho[-1]=='Variable':
                    L_sens.append('Variable')
                else :
                    L_sens.append('Sens inverse')

            # Critère de distance
            if Ligne["DTW normalisée"][0]<= quantile_30_DTW:
                L_distance.append('Petite')
            elif quantile_30_DTW < Ligne["DTW normalisée"][0] <= quantile_70_DTW:
                L_distance.append('Moyenne')
            else :
                L_distance.append('Grande')

            #Critère de retard
            if Ligne["TimeLag moyen vertical"][0]<= quantile_30_retard:
                L_retard.append('Petit')
            elif quantile_30_retard < Ligne["TimeLag moyen vertical"][0] <= quantile_70_retard:
                L_retard.append('Moyen')
            else :
                L_retard.append('Grand')

    df_sortante['Orthogonalité'] = L_ortho
    df_sortante['Sens'] = L_sens
    df_sortante['DTW'] = L_distance
    df_sortante['Retard'] = L_retard
    df_sortante['Concatenation'] = [L_ortho[k]+' '+L_sens[k]+' '+L_distance[k]+' '+L_retard[k] for k in range(len(L_ortho))]

    return df_sortante
```