Filtrage de mesures de trajectoires de véhicules

Sébastien Plantier

Travail de fin d'études

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

Année 2019-2020

Voie d'approfondissement :
Transport – Ingénierie de la mobilité

Soutenu le mercredi 8 juillet 2020

Devant le jury composé de :

- Président du Jury : Pierre-Antoine LAHAROTTE
- Tuteurs : Christine BUISSON et Pierre LEMAIRE
- Expert : Serge MIGUET

Par

Sébastien PLANTIER

Filtrage de mesures de trajectoires de véhicules

Organisme d'accueil :
Laboratoire d'ingénierie Circulation Transports (LICIT)

Notice analytique

AUTEUR	
Nom	PLANTIER
Prénom	Sébastien
ORGANISME D'ACCUEIL	
Nom de l'organisme et Localité	Laboratoire d'Ingénierie Circulation Transport – ENTPE – Vaulx-en-Velin, France
Nom du Tuteur	Christine BUISSON - Pierre LEMAIRE
ANALYSE DU TFE	
Titre (français)	Filtrage de mesures de trajectoires de véhicules
Titre (anglais)	Vehicules trajectories filtering
Résumé (français)	Le comportement individuel des conducteurs à l'échelle microscopique reste mal connu. S'il existe des modèles, les données permettant de les valider restent rares. Pour combler cette lacune, une des techniques utilisées a été de reconstituer les trajectoires des véhicules à partir d'images vidéo. Ces données vidéos sont difficiles à exploiter à cause des diverses sources d'erreurs pouvant les impacter. Il s'agit de problèmes lors de l'enregistrement, de la résolution des caméras ou encore de la difficulté à bien identifier les véhicules sur toutes les images. Toutes ces imprécisions doivent être filtrées pour pouvoir permettre une exploitation plus détaillée des données. Buisson, Villegas et Rivoirard proposèrent, en 2016, une méthode de filtrage basée sur le passage en coordonnées polaires. Cette méthode avait été testée, avec succès, uniquement sur une seule source de données, MOCoPo. Le présent travail a consisté à son application sur diverses sources de données vidéos (NGSIM, pNEUMA) mais aussi sur des données provenant de GPS et LIDAR. Les résultats obtenus montrent que si les données sont régulières dans le temps et que les hypothèses sur le rayon et l'angle polaire sont vérifiées, la méthode est applicable à toutes sources de données de trajectoires de véhicules.
Résumé (anglais)	Drivers individual behavior on a microscopic scale stay relatively unknown. Existing models lack corroboration data and cannot be completely trusted. In order of providing proof of accuracy, datasets based on traffic video recording were collected. These video datasets are heavily impacted by measurement errors. Recording missing parts, camera resolution or post-processing treatment (mainly vehicle identification between successive frames) are most of the error's sources. Literature proposes a wide variety of solution from classical data treatment such as Kalman or Butterworth filters to adding extra physical constraints. In 2016, Buisson, Villegas and Rivoirard suggested a new methodology based on polar coordinates and tested it with good results on the MOCoPo dataset. The present work continues this way and applies it on four others dataset with again good results provided that the trajectories stay in hypothesises constraints (which are relatively low: temporal continuity, no circuit).
Mots-clés (français, 5 maxi)	Trajectoires - Données vidéos - lissage polaire – Drone - Algorithme
Mots-clés (anglais, 5 maxi)	Trajectories – Video dataset – polar filtering – Drone - Algorithm
Termes géographique (français)	Grenoble, San Fransisco, Athènes

COLLATION			
	Nb de pages	Nb d'annexes (nb de pages)	Nb de réf. biblio.
	90	1 (10)	12

Sommaire

Table des illustrations

Table des tableaux

Introduction

L'automobiliste est un « objet » d'étude paradoxal. Par exemple, il est à la fois convaincu de la nécessité de lutter contre le changement climatique, mais refuse la mise en place d'une taxe carbone sur les carburants. Il se sent concerné par la sécurité routière, mais il est contre l'abaissement des vitesses. Il est pour la création de nouveaux axes routiers, mais pas dans son voisinage. Ces paradoxes, somme toute humains, sont bien sûr caricaturaux. Il n'y a pas deux automobilistes semblables et donc pas deux qui aient le même comportement. C'est d'autant plus logique quand on sait que conduire est l'activité quotidienne la plus complexe que l'on demande à notre cerveau. Si à première vue, ces questions comportementales sont éloignées du filtrage de trajectoires dont il va être question dans ce rapport, il ne faut pas oublier que toutes ces trajectoires sont issues du comportement de ces conducteurs.

Dans la suite, on va faire beaucoup d'analyses et notamment statistiques pour déterminer si nos résultats sont réalistes ou non. Généralement, on va disposer d'un échantillon de conducteurs suffisant pour que cette analyse ait du sens. Cette analyse doit être faite en tenant compte de ce que la trajectoire reflète des choix effectués par un être humain. Le choix, par exemple, de la vitesse à laquelle il va rouler, mais aussi du temps qu'il va mettre pour atteindre cette vitesse. La plupart des conducteurs vont appliquer une accélération progressive qui va les amener à la vitesse désirée dans une sensation de confort. Pourtant, nous avons tous déjà été témoin de véhicules accélérant beaucoup plus fortement, slalomant entre les autres voitures, sans doute à la recherche de sensations fortes ou d'un gain en temps de parcours.

Ces « fous » du volant vont clairement sortir comme des anomalies dans les trajectoires que l'on va étudier. Et c'est bien là un des enjeux de ce travail : appliquer un lissage qui élimine les défauts générés par la manière d'enregistrer les trajectoires sans supprimer les comportements extrêmes que peuvent avoir certains automobilistes. La méthode de lissage par le passage aux coordonnées polaires, qui va être présenté dans ce rapport, a cet avantage de ne pas chercher à contraindre les résultats issus des observations de terrain. On ne va pas définir ce qu'est le comportement normal d'un conducteur puis obliger les trajectoires à suivre ce modèle. Au contraire, l'objectif sera bien de préserver au mieux la diversité des comportements observés tout en obtenant des résultats statistiques cohérents avec nos connaissances actuelles.

Le premier chapitre va repositionner le problème que l'on essaye de résoudre et qui vient d'être brièvement introduit. Le deuxième chapitre va lister les différentes sources de données utilisées et leurs défauts majeurs, c'est-à-dire ceux que l'on va chercher à corriger. Le troisième chapitre sera dédié à la méthode et plus particulièrement à l'algorithme utilisé. Enfin, le quatrième chapitre présentera les différents résultats obtenus.

Chapitre 1
Positionnement du problème

1. Du besoin de données de trajectoires individuelles

Dans « The Future of Traffic Flow Theory » en 1963, Haight faisait remarquer à quel point les modèles de trafic sont avant tout issus d'une mathématisation du réel. La remarque reste toujours d'actualité en particulier pour les modèles microscopiques. Cette approche dogmatique n'est pas due à une plus grande présence de théoriciens dans le milieu de la théorie du trafic, mais à une réelle difficulté pour obtenir des données expérimentales fiables. Si les techniques de recueil de données de trafic macroscopiques sont bien maitrisées et servent à l'exploitation quotidienne des réseaux par les gestionnaires, les techniques pour l'échelle microscopique restent à la limite de notre technologie. Il peut paraître étonnant qu'il soit plus facile d'observer des objets stellaires à des milliers d'années-lumière de la terre ou le comportement de particules subatomiques que les véhicules pris dans les bouchons quotidiens. Les solutions existantes reposent soit sur des enregistrements vidéo, soit sur des radars ou soit sur des relevés GPS.

À la difficulté d'obtenir des données précises avec ces techniques, il faut ajouter que les situations d'intérêts sont également restreintes. Les questions que soulèvent les modèles microscopiques de trafic sont généralement liées aux changements de voies, pour un dépassement ou une insertion. Ces phénomènes sont à la fois locaux et fugaces, ce qui amplifie le besoin d'une grande précision des données recueillies. Cette connaissance précise des trajectoires des véhicules est indispensable pour que la théorie progresse. Car si les modèles existants reposent essentiellement sur les mathématiques et la physique, le conducteur, pour le moment toujours humain, du véhicule ajoute une composante non-déterministe. Or ce n'est qu'en étudiant le comportement d'un grand nombre de chauffeur que ces modèles pourront prendre en compte cette composante d'une manière statistiquement représentative.

2. Brève présentation des bases de données vidéo de trajectoires existantes

Une des premières idées pour combler cette lacune a été d'enregistrer, sur vidéo, la circulation à un endroit judicieusement choisi. L'idéal afin de pouvoir suivre au mieux les véhicules est d'avoir une vue zénithale pour éviter tout masquage. Les premiers essais ont reposé sur le placement de caméras sur des points hauts, comme des mats, des ponts ou des immeubles. L'une des premières sources de données ainsi réalisée est NGSIM, acronyme de Next Generation SIMulation. Réalisé en 2005, le projet a consisté en l'enregistrement par 7 caméras (figure 1) de l'autoroute I-80 au niveau de Emeryville, à proximité de San Francisco en Californie. Des enregistrements sur US-101 et le boulevard Lankershim sont venus compléter la base de données. Le principal apport du projet vient de l'algorithme de traitement des images qui permet de reconnaitre les véhicules sur chaque image puis de reconstituer les trajectoires.

Figure 1 - Placement des caméras utilisées lors de NGSIM - Source : Projet NGSIM - https://www.fhwa.dot.gov/publications/research/operations/its/06135/index.cfm.

Mais il n'y a pas que de points fixes que l'on peut filmer le trafic. Par exemple, une expérimentation a été conduite par l'université de Delft, au Pays-Bas, en utilisant un hélicoptère. En faisant du vol stationnaire, on obtient des images du même type que de celle avec un point haut. Mais cette technique peut être appliquée là où l'on ne dispose pas d'infrastructure en hauteur (et même plutôt dans ces conditions, il n'est pas toujours aisé de faire voler un hélicoptère entre les gratte-ciels). Cette méthode sera reprise en France, pour obtenir des images de différents sites dans l'agglomération grenobloise et notamment du convergent A41-N87 au niveau de Meylan (38) représenté sur la figure 2 suivante :

Figure 2 - Zone de convergence filmée lors du projet MOCoPo - Source : Site du projet MOCoPo - https://mocopo.ifsttar.fr/the-project/task-1/.

Enfin, les dernières technologies peuvent être aussi utilisées pour obtenir des données de trajectoires. En effet, les drones présentent des caractéristiques proches de l'hélicoptère même s'ils volent moins haut, et sont équipés nativement de caméras de bonne qualité. L'étude aurait dû pouvoir utiliser des données recueillies ainsi, mais le confinement a rendu impossible ces enregistrements. Une autre source de données disponible, utilisant cette fois non pas un mais plusieurs drones en essaim, a été trouvée. Cette méthode a été utilisée dans le cadre du projet pNEUMA à Athènes. Contrairement à toutes les autres bases de données mentionnées, il s'agit cette fois d'enregistrements urbains. Les dynamiques et trajectoires seront donc différentes des autres expérimentations. Les bases de données utilisées dans l'étude seront présentées plus en détail dans le chapitre 2.

3. Ce que l'on cherche à observer

L'objectif principal des modèles microscopiques est de pouvoir anticiper les comportements des conducteurs. La principale donnée dont ont besoin les chercheurs est la trajectoire du véhicule. Celle-ci représente le parcours du véhicule dans le plan classiquement nommé (x,y) (voir figure 3).Pour ce faire, il faut pouvoir « suivre » le véhicule ce qui a logiquement conduit à l'utilisation de la vidéo pour enregistrer ces données.

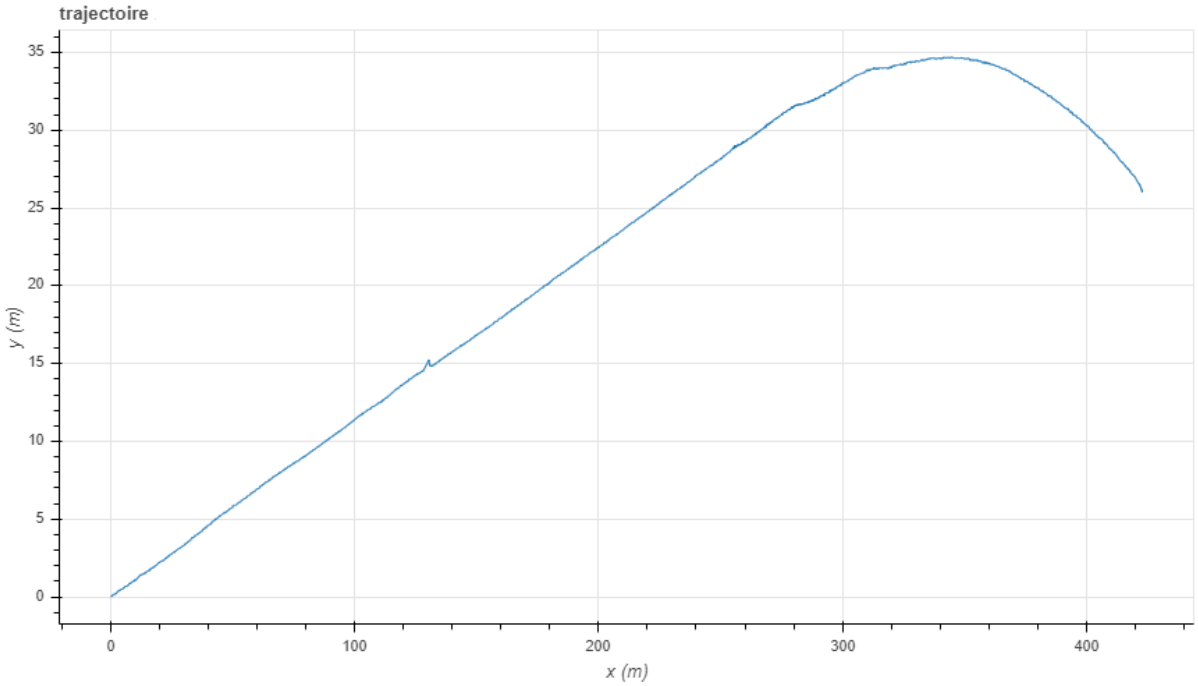

Figure 3 - Trajectoire du centre de gravité d'un véhicule dans le plan (x,y) issue des données MOCoPo.

Mais l'accès à la trajectoire d'un véhicule permet également de connaître d'autres informations. Il est facile de calculer la distance parcourue à partir des coordonnées de chaque point de la trajectoire. De cette distance, on peut par dérivation obtenir la vitesse instantanée du véhicule. La vitesse, au contraire des trajectoires est mieux documentée. Le comportement vis-à-vis de la vitesse des automobilistes à fait l'objet d'études, notamment dans le domaine de la sécurité routière. Cela fait de la vitesse un premier moyen de contrôle de la qualité des trajectoires obtenues. L'accélération, qui est la dérivée de la vitesse, est moins bien connue. Les plages théoriquement admissibles ainsi que les capacités d'une voiture pilotée par un être humain à accélérer ont été déterminées dans des conditions de « laboratoire », mais les données terrains contenant des mesures d'accélération restent rares. Néanmoins, pour estimer la validité d'un recueil de données, l'accélération reste une variable primordiale, à la fois simple à interpréter et avec des variations marquées qui facilitent la détection des anomalies.

En effet, n'importe quel automobiliste sait que son accélération a tendance être limitée, on ne s'amuse pas à appuyer brutalement sur la pédale, car cela rend la conduite inconfortable en plus de nous rendre moins prévisible pour les autres. Un conducteur « normal » évite les à-coups sauf en situation d'urgence. D'ailleurs, on peut également mesurer les à-coups, c'est ce que les anglosaxons appelle le « jerk » et il s'agit de la dérivée troisième de la distance. Ces dérivations successives sont illustrées sur la figure 4. Elles ont été obtenues à partir de données lissées afin de bien illustrer le procédé de dérivation. Ce ne serait que rarement le cas avec des trajectoires brutes. C'est pourquoi quand les données présentent de très forts changements de l'accélération ou des variations très rapides du jerk, on peut les soupçonner d'être erronées.

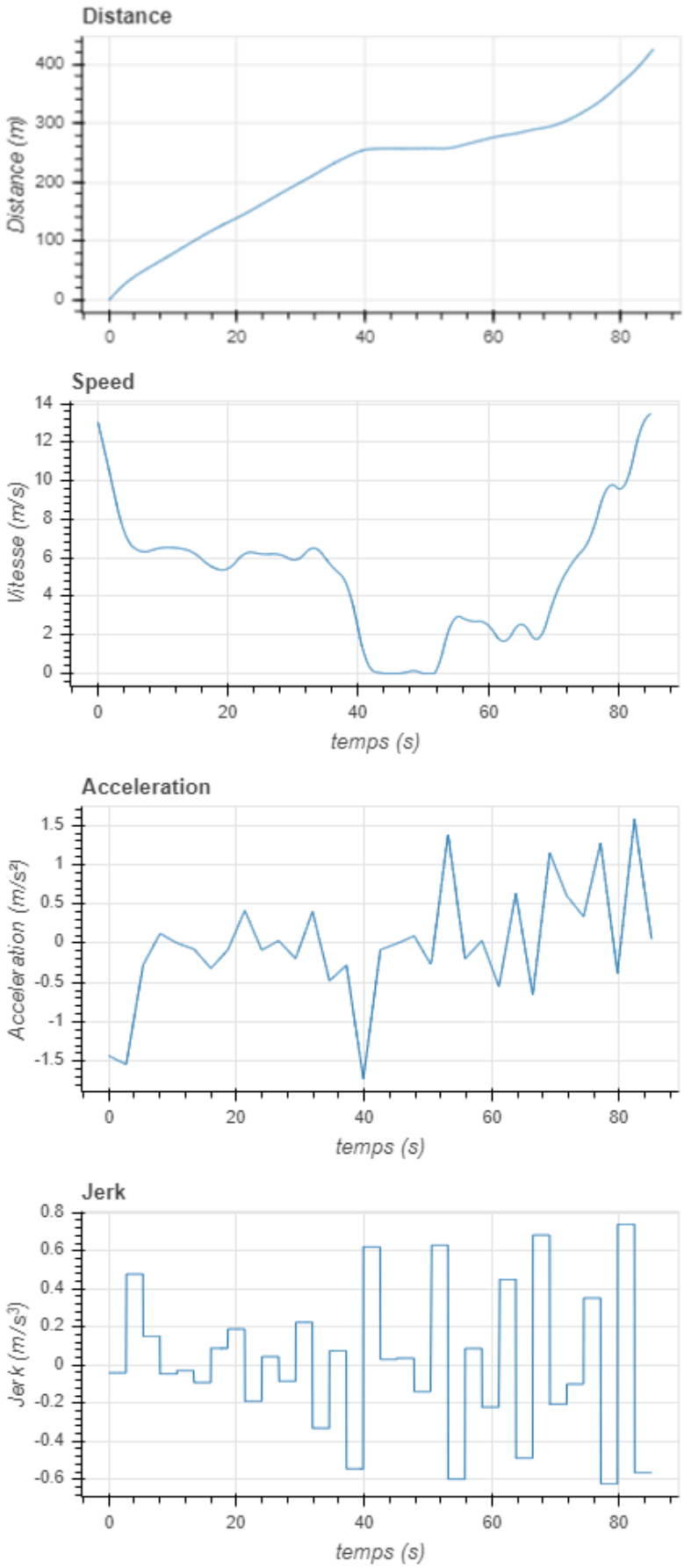

Figure 4 – Dérivation successive de la distance pour obtenir la vitesse, l'accélération et le jerk (à-coup).

4. Difficulté d'exploitation des données vidéos

Les raisons expliquant qu'une trajectoire recueillie par vidéo soit fausse sont nombreuses. La première d'entre elles vient directement de la qualité des enregistrements. Cette qualité est évidemment en progrès, les drones récents disposent par exemple de caméra 4K[1], mais les données de 2005 souffrent de ce que l'on peut qualifier de « bouillie de pixel ». D'ailleurs ces derniers sont également une des causes d'erreur et ce quelle que soit la qualité de la caméra. La détection d'un véhicule repose sur l'identification, image par image, des pixels qui le composent. Sa taille est donc définie par cette enveloppe de pixel, on parle de « blob », qui n'est pas stable d'une image à l'autre. Cette taille ne peut pas être inférieure à un pixel, et sa position dans l'image est également mesurée en pixel. Il est alors évident que plus la définition de la caméra est bonne et plus il sera facile à la fois de reconnaître le véhicule mais aussi de le localiser précisément.

Pour reconstituer les trajectoires, on se base sur la position non pas de tous les pixels constituant le blob du véhicule, mais sur son centre de gravité. La position de ce centre de gravité peut varier d'une image à l'autre selon comment le blob est détecté. Ce phénomène a tendance à créer des micromouvements sur la trajectoire. Ces mouvements parasites sont faciles à détecter puisqu'ils engendrent à chaque fois des variations de la trajectoire. De même, quand une trajectoire issue de données vidéo présente trop de changements rapides de valeurs d'accélération, on peut être sûr que cela vient de problèmes de consistance de la détection plus que de réelles impulsions du conducteur. Par exemple, sur la figure 5 ci-après, le conducteur aurait accéléré et décéléré plusieurs fois à des valeurs supérieures à 3 m/s² en 25 secondes. Or des accélérations de cette amplitude ne sont généralement observées qu'uniquement pour des insertions, des sorties d'autoroute ou des freinages d'urgence (-5 m/s²). Sur une section droite comme celle sur laquelle les enregistrements ont été réalisés, seuls des freinages d'urgence peuvent engendrer des valeurs aussi fortes de l'accélération. De plus, on peut constater que les accélérations ont été bornées pour éviter justement d'atteindre des valeurs irréalistes. Le seuillage appliqué force les accélérations à rester dans une plage de $\pm$ 3,7 m/s². Les conséquences de ce traitement seront également très visibles sur la figure 43.

[1] La 4K est à l'origine un format d'image dont la définition est normalement de 4 096 x 2 160 pixels (donc un peu plus de 4 000 pixels pour la largeur de l'image d'où son nom). Ce terme a ensuite été repris pour d'autres formats d'image (de largeur supérieure à 3 840 pixels), principalement pour des questions de marketing.

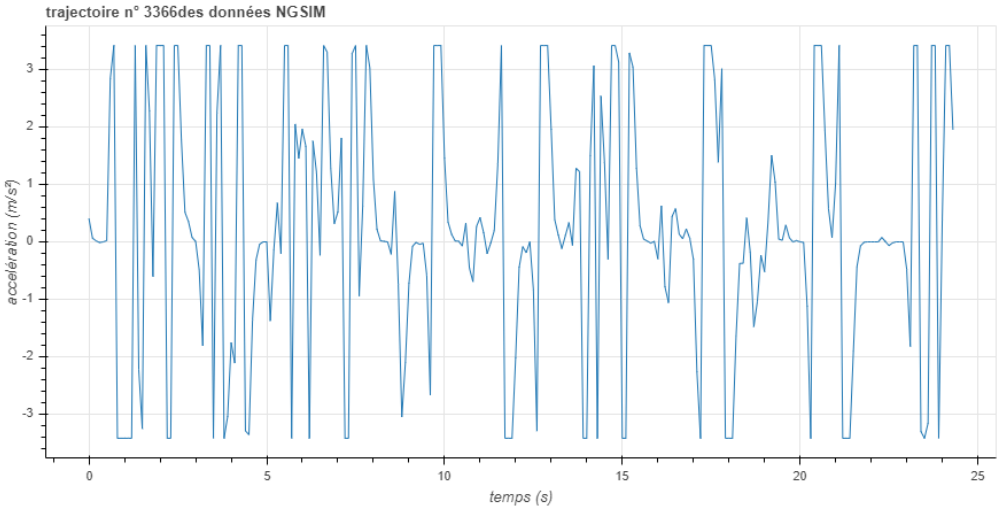

Figure 5 - Accélération issue des données NGSIM. Ces accélérations ne peuvent refléter le comportement d'un quelconque conducteur.

Cette erreur n'est pas la seule commise lors du traitement des images. Les méthodes de reconnaissance des véhicules ne sont pas infaillibles (figure 6). Selon les angles des prises de vues, divers phénomènes de masquage peuvent impacter leur précision. Pour les images zénithales, c'est-à-dire, au-dessus du trafic, les ombres, notamment des poids-lourds, peuvent changer l'aspect des véhicules voire être considérés comme des véhicules. Pour les autres angles de vue, les éléments de l'environnement de la route, comme les bâtiments, les portiques routiers ou les arbres, peuvent également masquer totalement ou partiellement les véhicules.

Les trajectoires sont alors morcelées et il est très difficile de les reconstruire. La seule manière réellement efficace repose sur une reconstruction par un être humain qui revisionne l'intégralité des images correspondantes. Etant donné la quantité d'images qui peut être faite, en général 24 images par seconde parfois plus, ce travail est extrêmement chronophage. Enfin, la taille de la route étudiée sur les images est variable. Dû à l'éloignement nécessaire pour avoir un bon angle de vue, la route peut n'être qu'une faible partie de l'image enregistrée. Ce problème est évidement amplifié si la caméra est de mauvaise qualité, la chaussée pouvant ne représenter plus qu'une poignée de pixel sur laquelle il sera très difficile de reconnaître les véhicules.

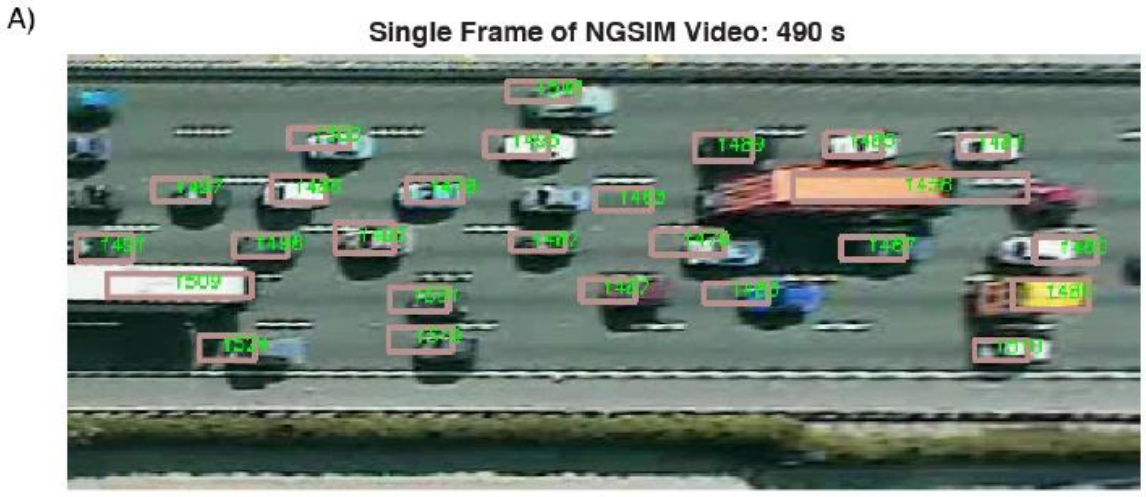

Figure 6 - Illustration des erreurs de détection sur les données NGSIM provenant de (Coifman and Li, 2017). Les blobs reconnus par la détection de véhicule sont représentés en rose et associé à un identifiant de véhicule en vert. Au milieu de l'image, le blob 1463 n'a aucun pixel en commun avec le véhicule qu'il est censé détecter.

5. Traitement des données issues de la reconnaissance d'image

Comme il n'est pas raisonnablement envisageable de redresser les trajectoires par des corrections manuelles, le choix le plus logique est d'appliquer des méthodes numériques pour les filtrer. Même si les méthodes de reconnaissance d'image ont également beaucoup progressé entre le début des années 2000 et maintenant, il y a toujours du bruit sur les trajectoires issues d'enregistrement vidéo. Ce bruit est notamment dû à l'usage de pixel comme unité graphique de base et même si sa taille diminue, la précision de la reconnaissance d'image ne peut pas être inférieure à cette unité. Le filtrage du bruit dans des données expérimentales est un sujet récurrent et vastement documenté. Il existe de nombreuses méthodes pour le réduire, mais il est important de ne pas oublier que chaque méthode à un objectif précis, souvent en lien avec les données sur lesquelles elle est appliquée. Les données NGSIM présentées précédemment on fait l'objet d'un grand nombre de tests avec une grande diversité des approches. Cela s'explique par leur ancienneté, leur libre-accès, leur volume important mais également par leur fort bruitage. A peu près toutes les méthodes classiques ont été appliquées sur ces données :

- Filtre passe-bas, passe-haut pour atténuer les points extremums.
- Filtre Butterwoth pour ne garder que les parties de trajectoires dans une bande passante admissible.
- Moyenne mobile, (Ossen and Hoogendoorn, 2008), qui consiste à calculer de manière continue la moyenne de la série à chaque ajout de point et à éliminer ceux qui s'éloignent trop manifestement de la tendance générale.
- Filtre de Kalman (Punzo et al., 2005), qui lui est un filtre fréquemment utilisé pour redresser des trajectoires (notamment celles des navettes spatiales). Ce filtre peut être utilisé pour lisser une série de données, mais son fonctionnement implique une connaissance d'un modèle physique auquel est soumis l'objet étudié. En effet, le filtre à partir de la donnée précédente, fait une prédiction sur la prochaine valeur. Puis il compare cette valeur à celle observée pour mettre à jour sa prédiction. Cela a le double avantage d'à la fois corriger les valeurs théoriques mais aussi de mitiger les erreurs de mesure.
- Lissage par une fonction que ce soit avec un filtre de Savitsky-Golay (Brockfeld et al., 2004), par une technique de régression locale ou par l'utilisation de fonction polynomiale par morceaux (spline). Souvent ces techniques sont couplées avec l'une des méthodes de filtrage précédente.

Ces différentes méthodes ont fourni des résultats plus ou moins probants selon ce que leurs auteurs cherchaient à obtenir. Dans tous les cas, les résultats étaient meilleurs que les données originales, mais toutes conservaient une proportion élevée de données irréalistes. Toutes ces tentatives infructueuses ont fait avancer la connaissance à la fois des données NGSIM mais également des données de trajectoires issues de vidéos d'une manière générale. Elles ont abouti à un questionnement légitime sur la manière d'évaluer les résultats obtenus. Ces bases de données comportent généralement plusieurs milliers de trajectoires et illustrer de bons résultats sur une seule n'a pas grand sens. Les analyses statistiques présentées par (Marczak and Buisson, 2012) sont un bien meilleur moyen de savoir si le procédé numérique employé a bien limité la distorsion dans les données ou s'il a été surcalibré sur une unique trajectoire. Des méthodes plus sophistiquées que la simple application de filtre ont également vu le jour. Parmi ces dernières, on peut en citer deux en particulier (Montanino and Punzo, 2015) et (Buisson et al., 2016).

6. Filtrage physique

La méthode présentée dans (Montanino and Punzo, 2015) a été appliquée aux données de NGSIM. L'idée derrière la méthode est de ne laisser aucune donnée qui soit physiquement impossible. Leur méthode est opérée sur les données positionnelles (x et y) et comprend 4 étapes :

- La première étape consiste à supprimer les données qui sont extrêmement biaisées. Les points dont les accélérations sont supérieures à 5 m/s² et les décélérations inférieures à -8 m/s² sont remplacés par des points reconstruits en appliquant une trajectoire curvilinéaire basé sur 10 points de référence avant et après l'extremum. Il est à noter que même si la détection est basée sur l'accélération la reconstruction concerne bien les données positionnelles.
- La deuxième étape a pour but d'éliminer le bruit. Cette étape se déroule par l'application d'un filtre passe-bas pour lisser les vitesses. Il est important que la première étape ait eu lieu pour améliorer la performance du filtre et éviter la perte de données. Le filtre utilisé est un filtre de Butterworh avec une fréquence d'écrêtage de 1 Hz.
- La troisième étape consiste en la reconstruction locale des points dont les caractéristiques cinématiques sont impossibles pour les véhicules. La prise en compte des variables de trafic se traduit par l'utilisation de paires de véhicules, c'est-à-dire la création d'un couple leader-suiveur. Ce couple permet d'imposer une distance entre les véhicules physiquement réalisable, cela permet également d'imposer une contrainte supplémentaire au véhicule suiveur. Cependant, dû au phénomène de changement de voie, il est parfois très difficile de déterminer qui est le leader de qui est le suiveur entre les véhicules. Or il est ainsi primordial de pouvoir, pour chaque pas de temps, identifier qui est le leader de qui. Les auteurs présentent un algorithme qui permet de réaliser des arbres définissant le leader et le suiveur de chaque véhicule pendant son temps de présence dans la section d'expérimentation. Une fois que cette hiérarchie a été reconstituée, il est alors possible de définir des règles pour déterminer une trajectoire physiquement compatible avec le trafic. La méthode consiste à encadrer l'accélération et la décélération maximale que peuvent réaliser les véhicules tout en vérifiant qu'ils maintiennent une distance suffisante avec le véhicule leader. À partir d'un point de divergence, les auteurs peuvent ainsi créer une fenêtre de valeurs admissibles pour la vitesse. On cherche ensuite le point de la trajectoire suivante qui appartient à la fenêtre. Entre ces deux points, la vitesse est reconstituée en utilisant une fonction curviligne continue par la résolution d'un problème non-linéaire formulé par les auteurs. Cette résolution ne présente pas forcément un seul optimum, il peut donc varier néanmoins les auteurs assurent que les paramètres adoptés permettent d'obtenir des résultats robustes.
- La quatrième, et dernière étape, consiste en l'application du même filtre que celui de la deuxième l'étape. Dû à des paramètres utilisés dans l'étape précédente, il est possible qu'il y ait certaines discontinuités sur les vitesses que ce filtre va corriger.

La figure 7, page suivante, montre le résultat de l'application successive de ces différentes étapes.

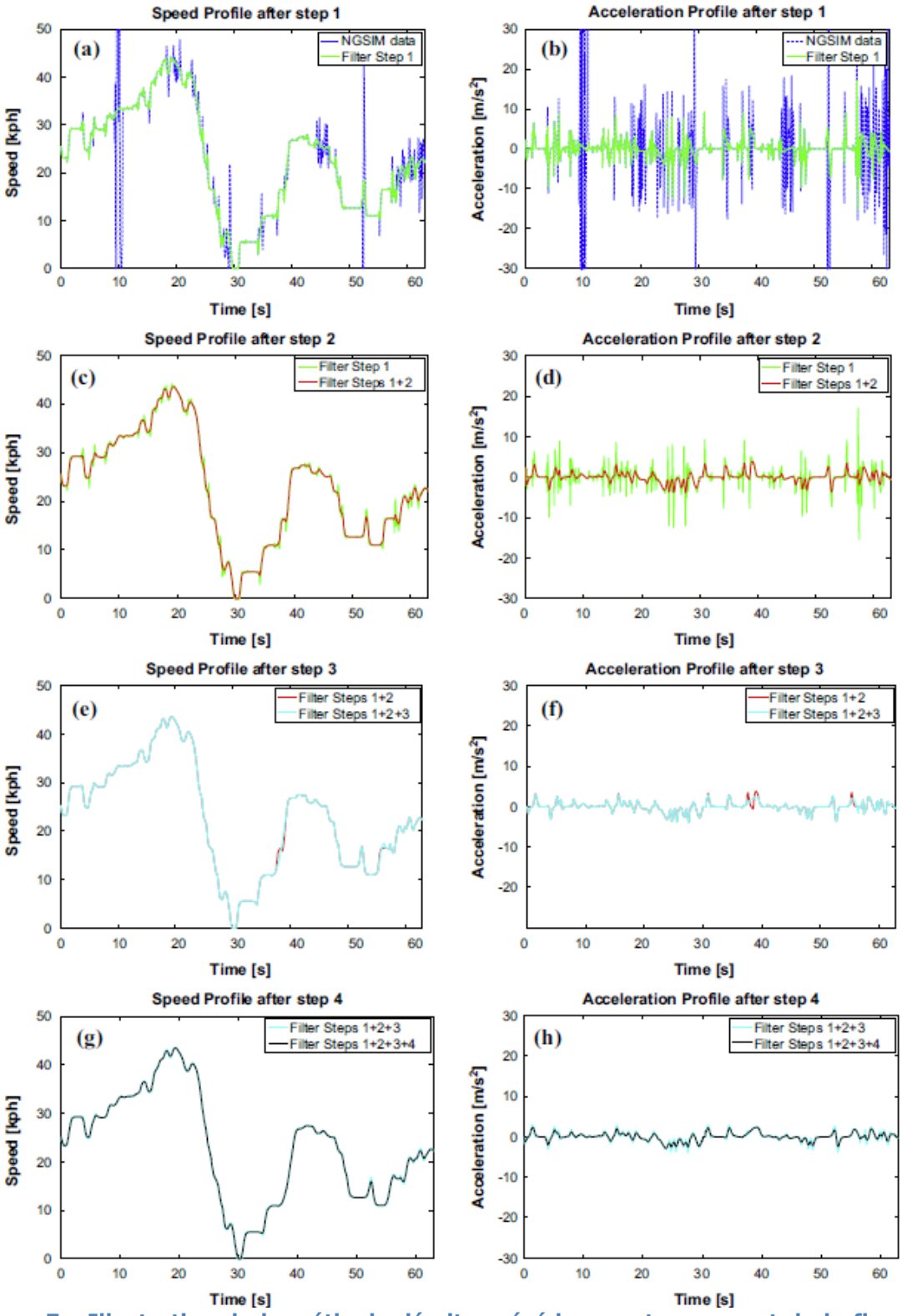

Figure 7 - Illustration de la méthode décrite précédemment provenant de la figure 3 page 87 de (Montanino and Punzo, 2015).

L'ensemble forme une méthode astucieuse pour corriger les défauts des données NGSIM. C'est d'ailleurs son principal défaut : elle a été conçue uniquement pour traiter ces données. A titre d'exemple, il est impossible de l'appliquer sur les données pNEUMA car ces données ne contiennent aucune indication sur qui est le leader ou le suiveur d'un véhicule. Ce qui est logique car elles concernent un réseau urbain dense, et les véhicules détectés, même à des instants très proches, peuvent ne pas être du tout sur les mêmes artères.

7. Filtrage polaire

La méthode présentée par (Buisson et al., 2016) a été appliqué sur les données MOCoPo. Ces dernières sont plus récentes que les données NGSIM, mais présentent malgré tout des défauts similaires. Comme on l'a vu dans le paragraphe précédent, en lissant les données positionnelles on peut créer des discontinuités, ce qui rendait nécessaire la quatrième étape. Ce problème est connu depuis longtemps par les physiciens, et il vient de la double relation qu'entretiennent les variables de position entre elles et avec le temps. En effet, la trajectoire est bien une fonction de x et de y mais x et y sont également des fonctions du temps. Cela implique que la trajectoire est également une fonction du temps, mais aussi que les erreurs entachant x et y sont liées. Une technique classique pour contourner ce problème est le passage au coordonnées polaires (figure 8).

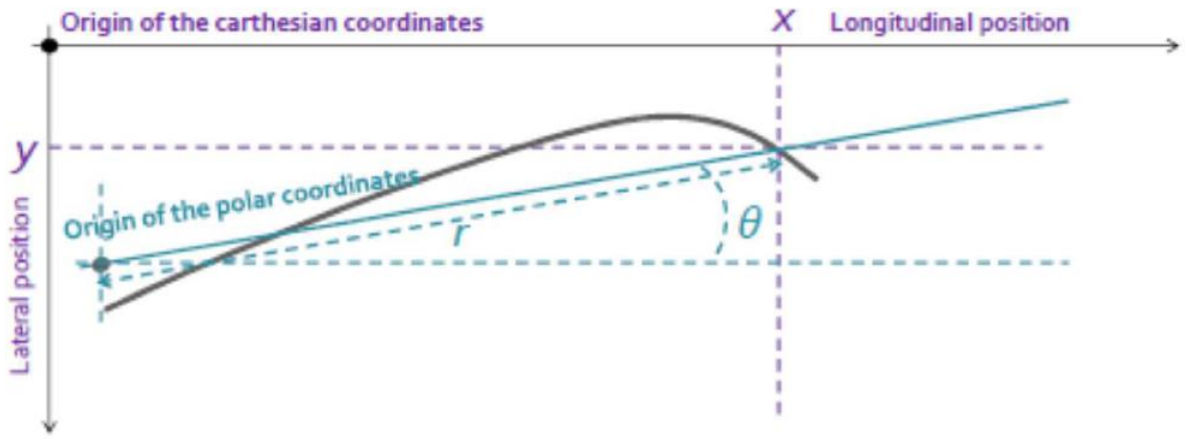

Figure 8 - Passage en coordonnées polaires issue de (Buisson et al., 2016), figure 2 page 7.

Dans ce système de coordonnées, les points sont définis par une distance par rapport à l'origine du repère appelé r et un angle mesuré dans le sens trigonométrique entre le point et l'axe des abscisses noté θ. L'avantage de ce système de coordonnées, c'est que pour la trajectoire d'un véhicule, r est assez proche de la distance parcourue. Comme nous l'avons vu précédemment, c'est de cette distance parcourue que l'on peut déduire les autres caractéristiques du véhicule comme la vitesse, l'accélération et le jerk. Lisser la distance parcourue est un bon moyen de supprimer les accélérations irréalistes qui parsèment les données recueillies. L'angle est lui plus lié aux caractéristiques de la route qu'au comportement individuel de l'automobiliste. Soit dit autrement, les variations de θ sont plus liées à la courbure de la route qu'a une embardée volontaire du chauffeur. Toutes ces raisons font du filtrage et du lissage en coordonnée polaire, une bonne approche du problème des données de trafic recueilli par vidéo. Cependant, pour pouvoir comparer la trajectoire filtrée avec la trajectoire originale, il faut pouvoir repasser en coordonnées cartésiennes ce qui n'est pas toujours facile. Les résultats présentés par (Buisson et al., 2016) montrent que l'application de cette technique sur les données MOCoPo permet d'obtenir des accélérations beaucoup plus réalistes et ce sans contraindre les données à rentrer dans la plage de valeur acceptable comme l'avaient fait Montanino et Punzo.

8. Problématique

L'arrivée des drones permet d'obtenir des images du trafic de bonne qualité et de manière plus simple qu'avec un hélicoptère ou à partir d'un immeuble. Ces données vidéos étant intrinsèquement bruitées, il est nécessaire de disposer d'une méthode pour les lisser. C'est pourquoi l'ambitieuse problématique soulevée par le présent travail est de **définir une méthode universelle de filtrage des données de trajectoires des véhicules**. Mais, on ne va pas repartir de zéro. La méthode de filtrage polaire semble prometteuse et au contraire de la méthode physique son principe est normalement adaptable à tout type de données. De plus, le code source développé par L. Rivoirard durant son master à l'ENTPE (Rivoirard, 2015) est disponible ce qui permettra un gain notable de temps. Cet objectif est complété par les questions suivantes :

- La méthode a fait ses preuves sur les données MOCoPo mais que se passe-t-il si on l'applique aux données NGSIM, pNEUMA ou aux échantillons de données drones disponibles ?

- Est-elle généralisable aux données de trajectoire qui ne proviennent pas d'image vidéo comme des traces GPS ?

Afin de restreindre ce questionnement fort vaste, il est bon de poser quelques hypothèses :

- On considère que la sélection des bonnes trajectoires dans le fourmillement que peut être une base de données comme NGSIM ou MOCoPo n'est pas l'objectif principal du présent travail. C'est pourquoi, on supposera que les travaux antérieurs de détermination des trajectoires pertinentes à traiter sont valides, même s'ils n'ont pas été faits dans l'optique de l'utilisation d'un filtrage en coordonnées polaires.

- On va supposer que les conversions de coordonnées GPS en coordonnées planes peuvent être approximées par les formules basiques suivantes :

$$x = R_{terrestre} \times \cos(latitude) \times \cos(longitude)$$

$$y = R_{terrestre} \times \cos(latitude) \times \sin(longitude)$$

$$avec\ R_{terrestre} \cong 6\,371\,009\ m\ , longitude\ et\ latitude\ en\ dégré\ décimaux$$

C'est bien sûr faux, mais le but du travail ne porte pas sur ce type de changement de repère.

Enfin, afin de rendre la diffusion de la méthode libre de droit, l'algorithme originel étant écrit en Matlab, on va devoir le transposer en Python en essayant de reproduire au mieux les résultats précédents.

Chapitre 2
Présentation des sources de données

1. Introduction

Dans ce chapitre, nous allons parcourir les différentes bases de données qui ont été utilisées pendant l'étude. Étant donné l'ambition universaliste présentée dans le chapitre précédent, il était important de pouvoir tester la méthode sur le plus de données possibles. Finalement, pendant les trois mois de travail, la méthode a été expérimentée sur 4 sources de données vidéos et également sur 2 sources de données non-vidéos. L'une provient de données GPS et l'autre de données LIDAR. Chacune de ces sources de données est bien-sûr différente des autres, mais toutes contiennent assez d'information pour pouvoir appliquer la méthode. Ce minimum d'informations requis est le suivant :

- Un identifiant unique pour chaque véhicule, ou un moyen d'en créer un.
- Une base de temps, dans l'idéal un horodatage (« timestamp ») ou un chronométrage en secondes.
- Enfin, des données positionnelles, dans l'idéal localisées en coordonnées cartésiennes (x et y) métrique, mais tous les repères ou unités peuvent convenir s'ils sont suffisamment documentés.

Les collections de données présentées ci-après, ne sont donc pas les seules qui puissent être utilisées. La seule limite est le temps, et bien sûr, la disponibilité des données. C'est pourquoi pouvoir disposer de 6 sources de données différentes est déjà une bonne base pour l'évaluation de la méthode. Dans la suite de ce chapitre, on va commencer par présenter les données issues d'enregistrement vidéo puis les autres sources. Enfin, le dernier paragraphe présentera un tableau des difficultés attendues pour l'exploitation de chacune de ces sources.

2. Données vidéo - NGSIM : la plus étudiée des bases de données

NGSIM, acronyme pour Next Generation SIMulation, est une base de données provenant d'une expérimentation américaine. Elle résulte d'une collaboration entre l'université de Berkeley et la Federal HighWay Adminstration (FHWA) de l'United State Departement of Transport (US DOT), le ministère des transports américain. L'objectif est à la fois de mieux comprendre le comportement des automobilistes pendant les périodes de congestion, mais également de fournir des données et des méthodes pour leur analyse. À ce titre, un logiciel est spécialement conçu par une société privée, Cambridge Systematics, pour faire de la reconnaissance d'image. 15 ans après leur création, les données NGSIM sont fréquemment utilisées dans des études ce qui démontre que cette ambition, de se poser comme un nouveau standard, a réussi. Les données regroupées sous ce terme de « données NGSIM » consistent en réalité en 4 jeux distincts. Deux jeux concernent les autoroutes : l'un enregistré sur l'I-80 (le plus vieux) à San Francisco et l'autre sur une autre autoroute de Californie l'US-101. Les deux dernières bases de données ont été enregistrées sur les artères urbaines de Lankershim et Peachtree de San Francisco.

La méthodologie pour les 4 sources de données est la même. En utilisant un point haut déjà existant, c'est-à-dire un immeuble, on ajoute à son sommet des caméras montées sur mat (comme illustré à la figure 1). L'avantage de cette méthode c'est que l'on évite que l'angle de vue de la caméra soit masqué. L'inconvénient c'est qu'à cette distance les éléments filmés sont très loin. Il faut donc des caméras avec une bonne résolution pour obtenir des images exploitables. À l'époque, il est évident qu'une caméra seule ne peut pas filmer une zone suffisamment longue pour qu'elle soit intéressante et que les images restent exploitables. Pour contourner cette difficulté, les expérimentateurs ont recours à plusieurs caméras qui leur permettent de filmer ainsi une plus grande zone sans devoir composer avec une résolution trop faible. Mais cette technique a également un inconvénient majeur, c'est qu'il faut alors pouvoir « recoller » les images de chaque caméra pour permettre le suivi de véhicule. Ceci n'est pas neutre comme nous le verrons après.

Parmi les données NGSIM, le seul jeu de donnée qui a été testé est celui de l'I-80. Le site de l'étude est illustré sur la figure 9 :

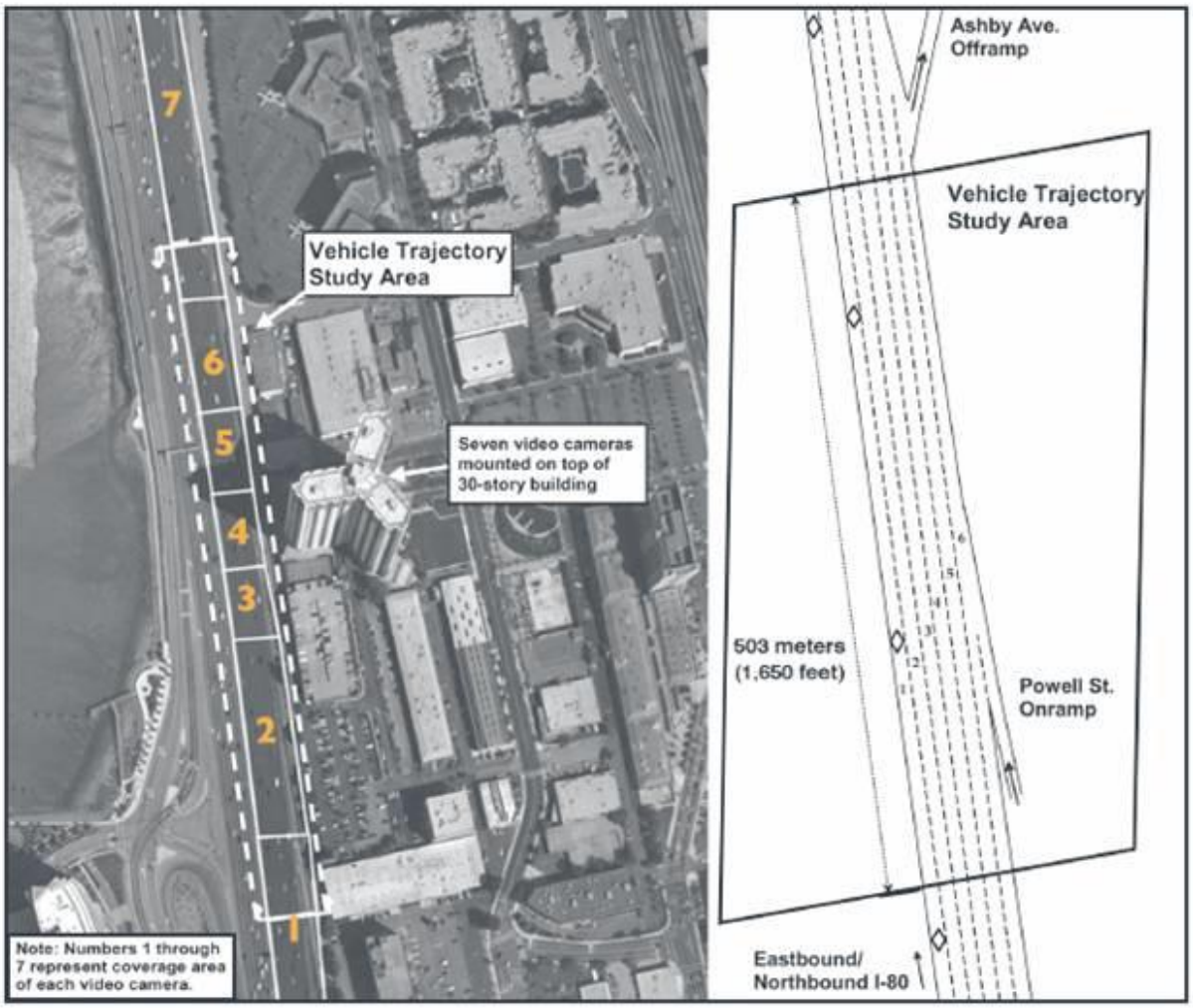

Figure 9 - Site de l'expérimentation sur l'Interstate 80 (I-80). Source : FHWA.

Il est donc assez peu étendu : 503 m. L'enregistrement a été fait le 07 avril 2005 sur trois périodes de temps (16h – 16h15, 17h-17h15 et 17h15-17h30) soit un peu moins d'une heure. Comme l'un des objectifs est de comparer les résultats de la méthode de filtrage polaire avec la méthode de filtrage physique, on va réduire encore le nombre de données en n'utilisant que les données du premier quart d'heure. Finalement, tout cela n'a rien de très impressionnant, mais il ne faut pas oublier qu'il s'agit d'une autoroute américaine à l'heure de pointe. Du fait qu'il y ait 6 voies de circulation, les données contiennent malgré tout 2037 trajectoires. Pour chacune d'entre elles, on dispose des 25 informations suivantes :

Nom de la rubrique	Commentaire
Vehicle_ID	Identifiant du véhicule
Frame_ID	Identifiant de l'image
Total_Frames	Nombre total d'image lié à ce véhicule
Global_Time	Horodatage
Local_X, Local_Y	Coordonnées de l'avant du véhicule par rapport à une origine située au début du tronçon étudié
Global_X, Global_Y	Coordonnées par rapport à une carte aplatie de l'état de Californie.
v_length, v_Width	Longueur et largeur du véhicule (moyenne sur toutes les images)
v_Class	Classe du véhicule (voiture, poids-lourd, deux-roues)
v_Vel, v_Acc	Vitesse et accélération instantanées du véhicule
Lane_ID	Numéro de la voie de circulation (1 à 6)
O_Zone, D_Zone	Origine et destination du véhicule
Int_ID, Section_ID	Passage à une intersection, position dans le sectionnement (uniquement pour les données urbaines).
Direction, Movement	Direction et mouvement du véhicule (uniquement pour les données urbaines).
Preceding, Following	Numéro (ID) du véhicule précédant et suivant le véhicule actuel.
Space_Headway, Time_Headway	Ecart en distance et en temps entre le point central de l'avant du véhicule et le point central de l'avant du véhicule qui le précède.
Location	Nom de la rue traversée (dans notre cas I-80)

Tableau 1 – Détail des rubriques fournies dans les données NGSIM.

C'est bien plus d'information que ce dont on a besoin, la seule difficulté est de choisir si on travaille avec les Local ou Global X et Y. Dans notre cas, il semble plus simple d'utiliser le repérage local car comme on le verra dans la partie suivante, ils sont plus proches de ce qui avait été fait pour les données MOCoPo.

2.1 La qualité des données NGSIM

Pour illustrer l'un des problèmes des données NGSIM, le plus simple est de représenter une des trajectoires contenues dans les données comme ci-dessus sur la figure 10 :

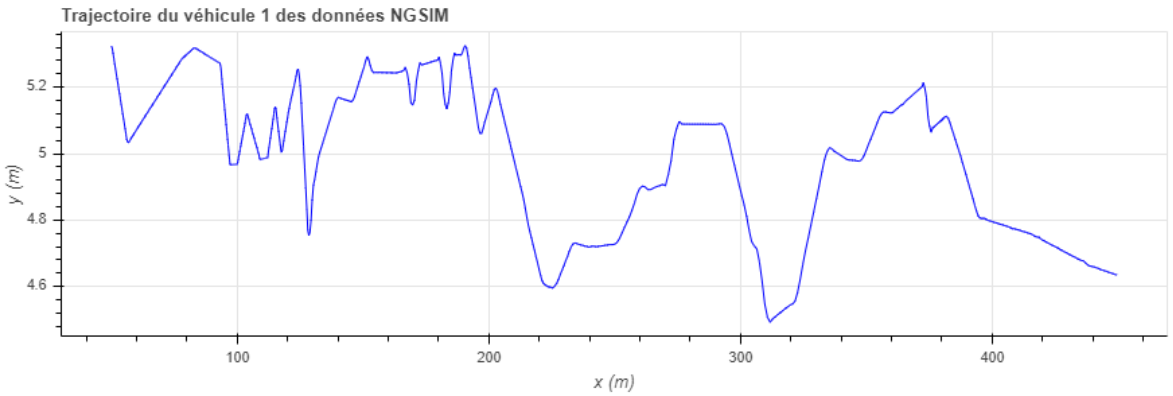

Figure 10 - illustration d'une trajectoire NGSIM : celle du premier véhicule issu du premier quart d'heure de la base de l'I-80.

Cette trajectoire peut ne pas sembler problématique mais si on regarde l'axe des ordonnées, on se rend compte que le véhicule n'a en fait pas quitté sa voie de circulation. Pour mémoire, une voie de circulation sur autoroute a une largeur de 3,5 mètres. Ces variations sont celles du point central de l'avant du véhicule. Et visiblement alors que ce dernier est resté sur la même voie de circulation, le conducteur a fait pas mal de zig-zag sur les 500 m du tronçon. Évidemment, ces écarts ne sont pas dus au conducteur, mais bien à des erreurs dans les données. Les différentes méthodes de traitement présenté dans le chapitre 1, ont en grande partie essayé de réduire ces erreurs. Ces différents auteurs ont produit des analyses plus ou moins détaillées des origines de ces erreurs. Et le moins qu'on puisse dire c'est qu'elles sont nombreuses.

Par exemple, l'un des défauts présents dans les données NGSIM, provient du recollage des images qui n'est pas toujours très réussi. Pour certaines caméras, il y a des recoupements qui engendrent des décalages dans le suivi de la trajectoire. Sur d'autres caméras, la taille de l'autoroute n'est pas la même sur toutes les images, ce qui rend le collage délicat. Cela peut sembler surprenant au premier abord, mais la lecture de (Coifman and Li, 2017) permet de mieux appréhender la situation. D'après eux, l'une des plus grandes sources d'erreur vient de la reconnaissance des trajectoires. Ce traitement n'a pas été réalisé sur les images haute résolution enregistrées par les caméras mais sur des images réduites à une résolution de 640x480 pixels. Sur ces images, l'autoroute ne représente plus que 160x480 pixels et les véhicules à peine 12x12 pixels ce qui est insuffisant pour pouvoir les détecter de manière efficace. Le papier détaille également les problèmes de la caméra 6 qui ne voit que l'arrière des véhicules, alors que l'on a vu dans les en-têtes des données que le point de détection était situé sur l'avant des véhicules. Cette erreur participe également au décrochement que l'on voit sur la figure 10. Montanino et Punzo, en 2015, avaient également mis en évidence des problèmes lorsque l'on changeait de trajectoire. Néanmoins, ils avaient considéré que les données étaient redressables par l'emploi d'un procédé mathématique, alors que Coifman lui suggère que les données sont beaucoup trop entachées d'erreurs pour pouvoir être exploitées telles quelles.

Pour résumer, l'intérêt des données NGSIM provient du fait qu'elles ont été largement étudiées, mais elles sont beaucoup trop erronées pour pouvoir être corrigées uniquement par un traitement mathématique. Néanmoins, dans le cadre de cette étude, il est indispensable de les traiter pour pouvoir comparer les résultats avec ceux de la littérature. A ce titre, les données lissées de Montanino et Punzo, (Montanino and Punzo, 2015), mises à disposition avec leur article seront également utilisées comme point de comparaison dans le chapitre 4.

3. Données vidéo – MOCoPo : la base exploitée originellement par l'algorithme

MOCoPo pour Measing and mOdelling, traffic Congestion and Pollution, est un projet financé par le ministère (MEDDE) dans le cadre du Programme de REcherche et d'Innovation dans les Transports terrestre (PREDIT). Il s'agit d'une association entre différents laboratoires de l'Université Gustave Eiffel (LICIT, LTE, LEPSIS, IM et MACS), l'INRIA, le CNRS, Atmo Rhône-Alpes, le CEREMA, le CEREA et la DIR-CE (gestionnaire de l'infrastructure). Le but du projet est plus large que de simplement recueillir des données de trajectoires. Il comprend également une composante modélisation macro et microscopique du trafic, mais aussi de la pollution qu'il génère. La partie qui nous intéresse se trouve dans le package nommée « Task 1 ».

Les sites d'études se trouvent tous localisés sur le périphérique de Grenoble, la RN87. Trois types de situations ont été étudiés : un convergent, une section courante et un échangeur. Comme pour les données NGSIM, on ne va pas étudier toutes les données mais uniquement le convergent déjà illustrée sur la figure 2. Les observations ont été réalisées par trois caméras montées sur un hélicoptère. Ce dernier vole à une distance de 500 m au-dessus de la route étudiée comme illustré sur la figure 11 :

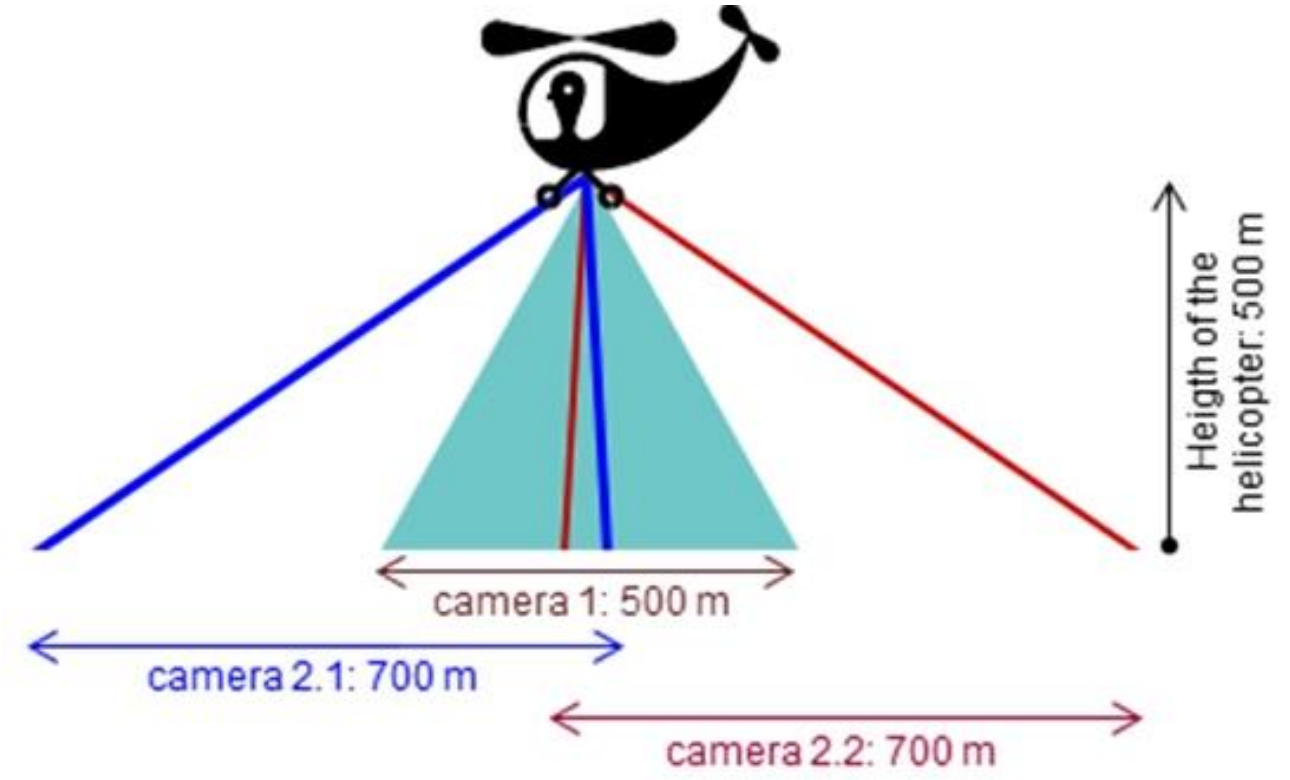

Figure 11 - Principe du recueil de données par hélicoptère. Source : Projet MOCoPo - https://mocopo.ifsttar.fr/the-project/task-1/

Pour les trajectoires, seule la caméra 1 a de l'importance, les deux autres servant pour faire de la détection d'origine-destination pour une autre partie du projet. Le fait qu'il n'y ait qu'une seule camera va évidemment supprimer les problèmes de recollage d'image qui avait impacté les données NGSIM. En tout, une dizaine de films ont été réalisés, mais seul le film 4 a été utilisé dans ces travaux. Ce film a été réalisé le 16/09/2011 de 7h58 à 8h58 donc là encore, une heure de donnée. Ce film a ensuite été passé à un logiciel de reconnaissance d'image pour établir les trajectoires. Le résultat est une base de données de 24214 trajectoires. Elles se présentent avec les en-têtes suivants :

Nom de la rubrique	Commentaire
Id	L'identifiant du véhicule
Horodatage	L'heure d'enregistrement de l'image (en seconde depuis le 1er janvier 1970)
Coordonnée des points de l'octogone	La détection du véhicule se fait par la création d'un octogone qui entoure
Coordonnées du centre de gravité	Les coordonnées du centre de gravité de l'octogone calculé depuis un point de référence placé au bord de la route.
Taille du blob	La taille de l'octogone. Cette taille change sur chaque image.

Tableau 2 – Détail de l'en-tête des données MOCoPo.

Ces informations sur le blob vont être bien utiles pour la suite pour pouvoir trier les trajectoires.

3.1 Les difficultés liées à l'exploitation des données MOCoPo

Contrairement aux données NGSIM, les données MOCoPo sont « brutes ». Cela signifie qu'elles n'ont pas été retraitées à la sortie de la reconnaissance d'image. Cet état peut être illustré sur la figure 12 suivante :

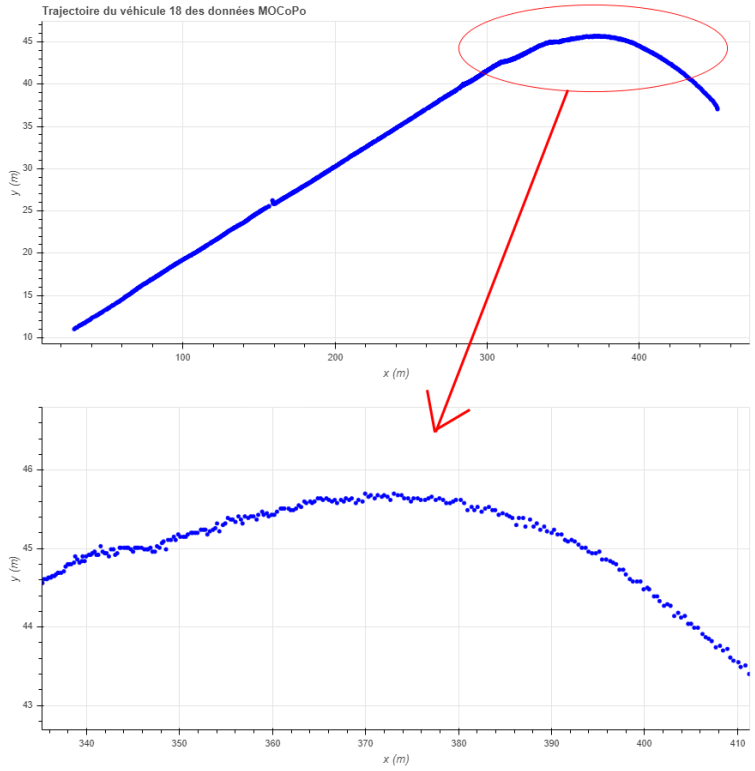

Figure 12 - Trajectoire d'un véhicule des données MOCoPo avec un zoom pour illustrer le phénomène de dispersion des points.

Les données sont plus un « nuage de points » qu'une belle trajectoire uniforme. Cela vient directement du manque de constance et de précision dans la détection des véhicules. D'instinct, on a tendance à penser que la « vraie » trajectoire passe au « milieu » de ces points. C'est exactement ce que l'on va essayer de faire et la méthode sera expliquée dans le prochain chapitre. Cependant, ce n'est pas le seul problème dans

le jeu de données. Le fait d'utiliser un hélicoptère pose des problèmes de stabilité de la trajectoire, car ce dernier ne peut pas rester parfaitement immobile au-dessus du trafic. De même pour des raisons diverses, le logiciel de traitement d'image n'a pas associé les bons blobs aux bons véhicules. Il en résulte des trajectoires comme celle de la figure 13 :

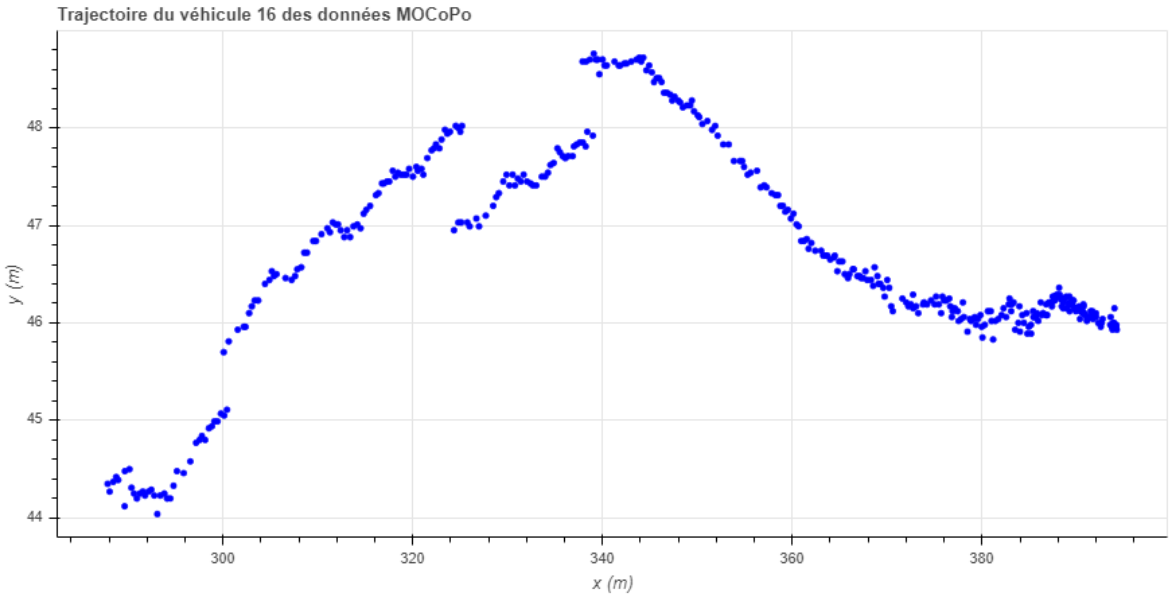

Figure 13 - Erreur de détection du véhicule sur la partie centrale de la trajectoire.

Ici, on peut supposer qu'il s'agit d'un masquage par un poids lourd sur l'autre voie. Cela peut être également un véhicule noir qui sont plus souvent confondus soit avec la chaussée, soit avec les ombres. C'est la principale difficulté d'usage de ce jeu de données. Il faut donc savoir comment séparer les « mauvaises » trajectoires des bonnes. En effet, malgré toutes les qualités que peut avoir un algorithme de lissage des données, il n'est pas possible de passer d'une trajectoire fausse à une trajectoire plausible. Heureusement, une sélection des trajectoires exploitables a déjà été faite par Rivoirard (Rivoirard, 2015, p39) ce qui réduit le nombre de trajectoire à 638.

En somme, les données MOCoPo ont l'avantage d'être celles sur lesquels l'algorithme a été formulé à l'origine. Les données ont été prises avec une seule caméra et on dispose de la sortie brute du logiciel de reconnaissance des véhicules. L'inconvénient, c'est qu'il faut filtrer les trajectoires extraites pour ne garder que celle qui sont pertinentes.

4. Données vidéo - Recueil avec un drone, des échantillons de données uniquement.

Ces données auraient dû être le gros apport de ce travail. Malheureusement, les jours de vol prévu sont tombés pendant le confinement lié à l'épidémie de Coronavirus. Des données avaient déjà été acquises avant le début du confinement, mais il s'agissait de test pour valider l'intérêt de l'utilisation de drone plus que de vraies situations d'enregistrement du trafic. Le principal défaut de l'utilisation des drones « légers » du commerce est leur autonomie qui est comprise entre 15 et 30 minutes. Pour contourner ce problème, l'idée était d'utiliser un drone « lourd », c'est-à-dire alimenté en continu par un câble comme illustré sur la figure 14 suivante :

Figure 14 - A gauche drone Phantom 4 de DJI - source : DJI. A droite drone Orion de Elistair - source : Elistair.

Les premiers tests avaient montré que l'utilisation de deux drones légers qui font des rotations était beaucoup plus simple que le drone lourd. De plus le câble d'alimentation du drone lourd sert aussi à transférer les images, or la hauteur à laquelle celui-ci vole fait que le câble se comporte comme une antenne. Les images enregistrées sont alors parasitées par les signaux électromagnétiques ce qui dégrade la qualité des images.

L'avantage des drones sur l'hélicoptère, c'est d'abord leur coût. Entre les caméras utilisées pour MOCoPo en 2011 et celles disponibles actuellement, les résolutions ont encore progressé. Ainsi, les images enregistrées par les drones sont en format 4k : 4096 x 2160 pixels contre 2448 x 2050 pixels pour la caméra de l'hélicoptère de MOCoPo. Ces images sont bien plus faciles à utiliser pour la détection des véhicules. L'inconvénient, c'est que pour des raisons de réglementation notamment, le survol des routes par des drones est strictement encadré. On ne peut pas avoir une position zénithale comme avec l'hélicoptère, mais on se retrouve avec des images plus proches de celles prises en point haut. Dans les essais, les drones volaient à 30 m de hauteur, sur le côté du trafic. Cette position permet d'éviter une partie du masquage par la végétation de bord de voie. Le site choisi se trouve sur la rocade est (RN 346) à Meyzieu dans la périphérie de Lyon (69).

Dans cette étude, on ne présentera que les résultats sur des morceaux de trajectoires isolées issues de ces données. L'analyse complète fera sans doute l'objet d'un article dans les années à venir.

5. Données vidéo – pNEUMA : les résultats d'un essaim de drone en milieu urbain

L'expérimentation pNEUMA (New Era of Urban traffic Monitoring with Aerial footage) est la démultiplication de celle du paragraphe précédent. Elle a été orchestrée par Emmanouil Barmpounakis, Nikolas Geroliminis de l'École Polytechnique Fédérale de Lausanne (EPFL). Réalisée le 30 octobre 2018, elle consiste en l'utilisation simultanée de 10 drones pour parcourir une zone d'1,3 km² dans le centre-ville d'Athènes, Grèce. Comme illustré sur la figure 15, ci-après, le principe est de faire voler les drones au-dessus des rues pour obtenir énormément d'image du trafic (59 heures d'enregistrement cumulées pour une durée réelle d'observation de l'ordre de 2 heures).

Figure 15 - Zone de l'expérimentation pNEUMA. Les zones colorées sont celles survolées par les drones. Les points marqués d'un H sont les deux points d'envol et de rechargement des drones. Source : (Barmpounakis and Geroliminis, 2020).

Les drones utilisés sont tous des drones légers du même type que le Phantom 4 représenté sur la figure 14. Ils ont donc les mêmes limitations en termes d'autonomie. Les enregistrements se font donc de la manière suivante :

- décollage du drone vers sa zone de couverture,
- enregistrement d'images pour une période de 25 minutes environ,
- retour à la zone de décollage pour changement des batteries et ainsi de suite.

En tout, ce processus a été répété 5 fois. Le fait d'avoir un essaim de drones signifie qu'il faut une bonne coordination entre les pilotes. Par exemple, il faut que tous les drones soient en place pour commencer l'enregistrement. Pour le traitement de toutes

ces données vidéos, les chercheurs ont fait appel à une société : DataFromSky[2]. Leur technique de reconnaissance des trajectoires à partir d'image serait basée sur des avancées en intelligence artificielle et en apprentissage automatisé. Néanmoins, leur site ne donne pas plus d'information sur leur méthode. À la sortie du logiciel de traitement d'image, on dispose d'un jeu de 2276 trajectoires avec l'en-tête suivante :

Nom de la rubrique	Commentaire
track_id	Identifiant du véhicule
type	Au choix parmi les options suivantes : voiture, taxi, bus, utilitaire, poids lourd, deux-roues motorisé.
traveled_distance	Distance parcourue (au total sur l'ensemble de la trajectoire)
avg_speed	Vitesse moyenne (sur l'ensemble de la trajectoire)
lat	Latitude en degré selon le référentiel WGS 84
Lon	Longitude en degré selon le référentiel WGS 84
Speed	Vitesse instantanée en km/h
tan_accel	Accélération tangentielle instantanée en m/s²
lat_accel	Accélération latéral instantanée en m/s²
time	Horodatage en millisecondes compté depuis le début de l'enregistrement vidéo.

Tableau 3 – Informations disponibles dans la base de données pNEUMA.

À la différence des autres jeux de données, les trajectoires sont présentées, de manière surprenante en ligne. Il faut donc les « transposer » pour les rendre utilisables. De même, comme les données proviennent de plusieurs drones, il n'est plus possible de travailler dans un repère local propre à l'image. Pour contourner cette difficulté, toutes les positions ont été converties dans le système de coordonnées du GPS, le WSG 84. Cependant, un véhicule peut bien être tracé d'un drone à l'autre comme illustré sur la figure 16 suivante :

[2] https://datafromsky.com/

Figure 16 - Illustration du suivi d'un véhicule par plusieurs drones. Source : (Barmpounakis and Geroliminis, 2020).

Si on trace l'une des trajectoires contenues dans la base de données, on obtient quelque chose de proche des données MOCoPo comme illustré sur la figure 17 :

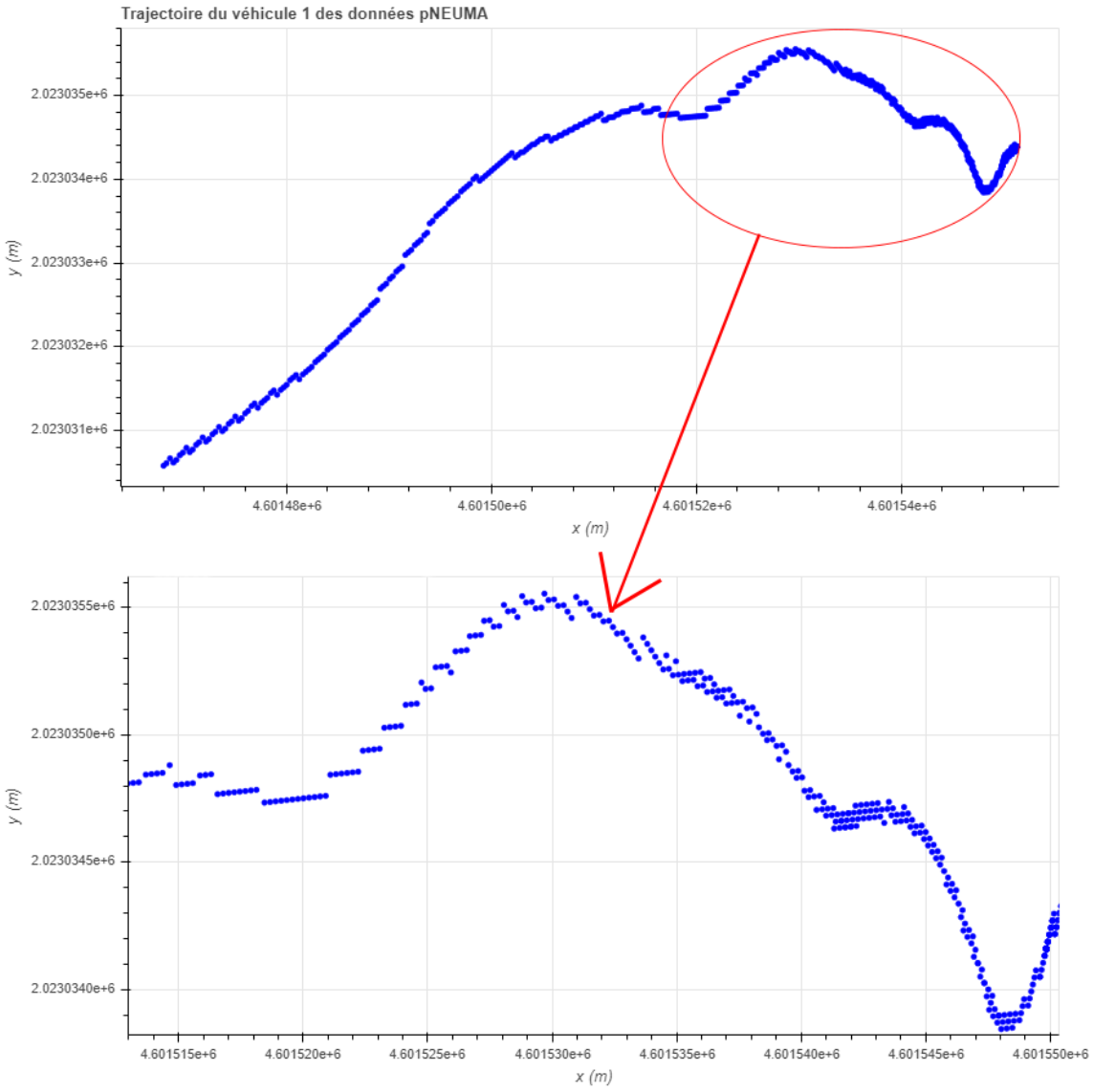

Figure 17 - 1ère trajectoire brute contenue dans les données pNEUMA. Sur le zoom, on retrouve un nuage de point comme celui des données MOCoPo.

Ces données sont très récentes, et elles n'ont pas encore été pleinement exploitées par la communauté scientifique. Elles présentent certainement un certain nombre d'erreurs comme à chaque fois que l'on a recours à la combinaison : enregistrement d'image – reconnaissance de trajectoire. Néanmoins, le but de l'étude n'est pas de chercher ces erreurs et encore moins d'en déterminer la cause. Cependant, du fait qu'il s'agisse de données de milieu urbain, un certain nombre de cas de figure sont présent dans ces données qui n'avait évidemment pas été pris en compte lors de la création de la méthode de filtrage polaire, conçue pour une autoroute péri-urbaine. Parmi ces cas de figure, on trouve tout d'abord un grand nombre d'arrêts aux feux tricolores. Ces arrêts vont créer une densification de point à des endroits précis de la trajectoire qui peuvent perturber par la suite leur traitement. L'autre cas problématique, vient du fait qu'il s'agit d'un réseau maillé. Les véhicules ont la possibilité de faire des trajectoires « circulaires » ou tout du moins des circuits comme illustrée sur la figure 18 suivante :

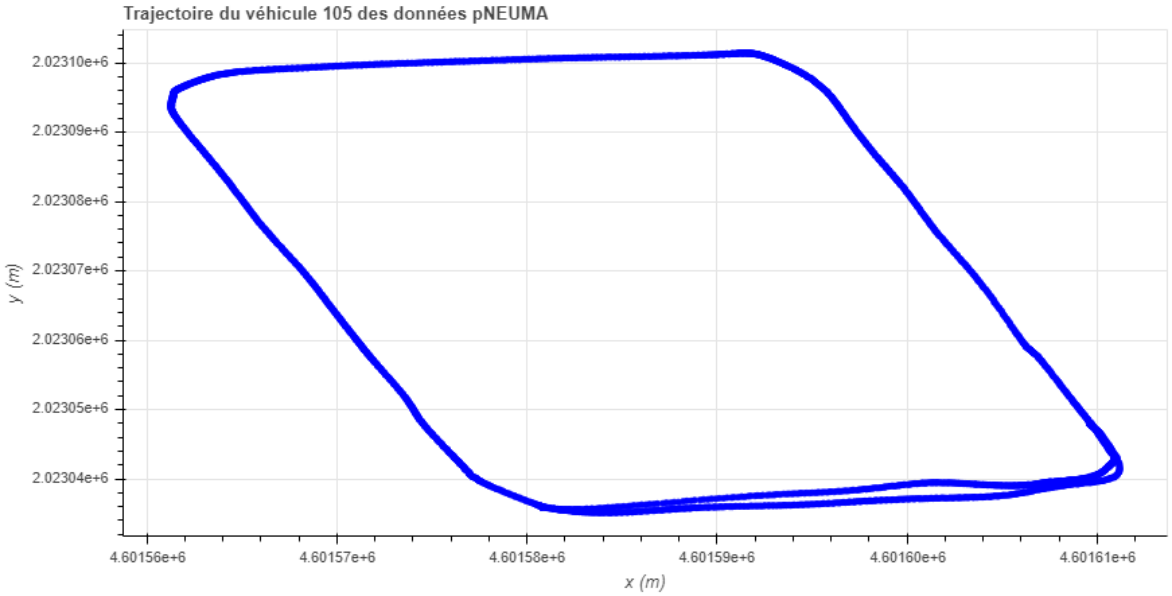

Figure 18 - Trajectoire "circulaire" extraite des données pNEUMA.

Or dans ces cas, la distinction faite entre les variables polaires r et θ, n'est plus valable. Cela ne veut pas dire que le filtrage polaire ne fonctionnera pas mais comme ces variables seront fortement liées, il ne sera plus possible de repasser facilement dans le système de coordonnées cartésiennes.

Pour synthétiser, ces données devraient être utilisables avec nos algorithmes, mais il ne faut pas perdre de vue que ces données ont été enregistrées dans un milieu urbain. Les trajectoires des véhicules dans une vaste zone en ville seront très différentes de celles sur une petite portion d'autoroute. Enfin, le fait que les coordonnées soient enregistrées dans le système GPS signifie qu'il va falloir faire une projection ce qui va irrémédiablement entraîner des approximations sur les valeurs.

6. Données GPS et LIDAR

Passons maintenant à des données qui ne proviennent pas d'enregistrement vidéo. L'autre moyen généralement employé pour obtenir des trajectoires est le suivi par GPS. Malgré cela, cette technologie n'a quasiment pas été utilisée dans la recherche sur le comportement des conducteurs. En effet, jusqu'à récemment, le taux de pénétration des GPS était bien trop faible. À cela il faut ajouter que tous les GPS ne dépendent pas des mêmes opérateurs ce qui réduit encore le nombre d'équipement disponible. Avec l'utilisation des smartphones pour se repérer et se guider à la place de machine dédiée et l'incorporation de GPS directement dans les véhicules, le recueil de données GPS a beaucoup progressé.

Actuellement, peu de données sont librement accessibles, notamment pour des questions de droit. Les bases de données disponibles connues suivent par exemple des taxis, ce qui crée forcément une forte distorsion à la fois sur les origines-destinations des trajets, mais également sur les comportements de conduite. Enfin, les trajets suivis sont soit uniquement en ville comme (Yang et al., 2019), soit sur de très grandes distances. Il est assez rare de disposer de jeux de données GPS contenant assez de trajectoires pour pouvoir étudier un point particulier d'un réseau autoroutier (convergent, divergent voir même section courante). Enfin, les erreurs commises par un GPS sont très différentes de celles des logiciels de reconnaissance d'image. Le GPS assure une remontée à pas temporel constant des informations, on n'a donc plus la problématique de masquage qui entraîne une interruption momentanée du suivi de la trajectoire. Cependant, la précision d'un GPS standard est de l'ordre de 5 mètres. Si c'est suffisant pour faire du guidage d'itinéraire, pour les questions de trajectoires cela peut créer des discontinuités. Il existe des GPS bien plus précis, mais il ne s'agit pas de produit grand public, donc il y a peu de données avec de telles machines.

La sélection d'un jeu de données GPS pour pouvoir tester l'algorithme n'a donc pas été facile. L'objectif est de voir si le lissage par les coordonnées polaires va permettre une amélioration des trajectoires, pas de chercher des trajectoires qui pourraient correspondre dans un grand ensemble de trajectoires. Le choix s'est porté sur des données GPS contenues dans une étude de Coifman, (Coifman et al., 2016), qui porte sur un autre moyen de recueil de trajectoires. Avec le même objectif d'analyser le comportement des conducteurs, Coifman et ses co-auteurs ont équipés un véhicule avec à la fois un GPS de précision (DGPS) et des radars (les LIDARS). Ce véhicule est illustré sur la figure 19 suivante :

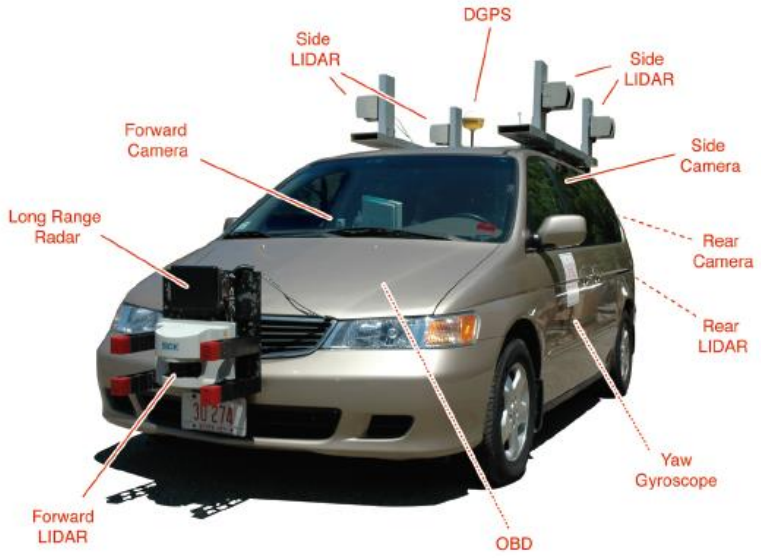

Figure 19 - Véhicule de recueil de données utilisé dans l'étude. Source : (Coifman et al., 2016).

Les LIDAR permettent de recueillir les trajectoires des véhicules à proximité du véhicule instrumenté. Comme pour les données vidéos, elles doivent ensuite être retraitées pour pouvoir être exploitées. On va donc exploiter deux bases de données : les positions GPS du véhicule de collecte de données et les trajectoires LIDAR. Ces données ont été recueillies pendant 2 heures, en effectuant un trajet aller-retour de 28 miles. Le véhicule a circulé sur les autoroutes I-70, I-71, I-270, I-670 à proximité de Colombus aux Etats-Unis (figure 20) :

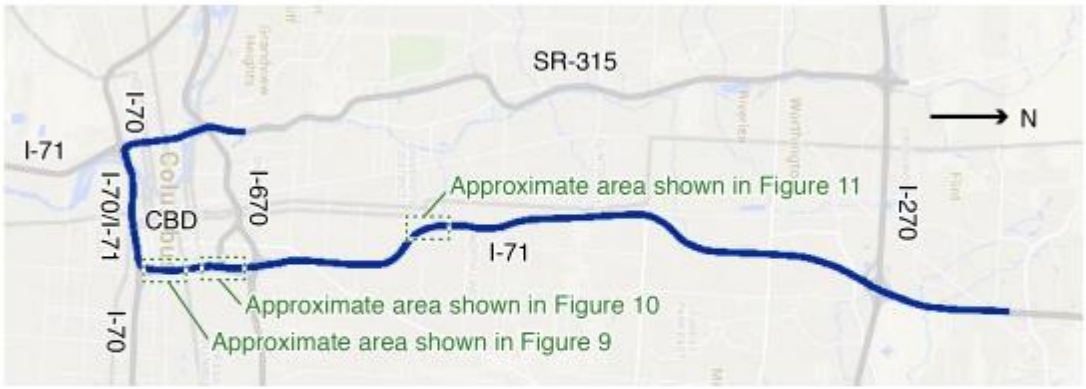

Figure 20 - Parcours du véhicule équipé pour le recueil de trajectoire via LIDAR. Source : (Coifman et al., 2016).

Ces données sont fournies avec un fichier PDF qui explique ce que contiennent ces bases de données :

Numéro de la colonne	Commentaire
1	Horodatage (secondes)
2	Heure complète telle que donnée par le GPS
3, 4	Position globale (m) (repéré par rapport au Nord et à l'Est)
5	Orientation du véhicule (degré)
6, 7	Distance longitudinale et latérale par rapport à la voie de circulation en m.
8, 9	Vitesse longitudinale et latérale en mph.
10, 11	Accélération longitudinale et latérale en mphps.
12	Numéro de la voie

Tableau 4 – En-tête des données du véhicule équipé (Probe Vehicle).

Numéro de la colonne	Commentaire
1	Identifiant du véhicule détecté
2	Horodatage (secondes)
3, 4	Valeurs minimales de x et y de détection en m. (détection au plus proche)
5, 6	Valeurs maximales de x et y de détection en m. (détection au plus loin).
7, 8	Position globale (m) (repéré par rapport au Nord et à l'Est)
9, 10	Distance longitudinale et latérale par rapport à la voie de circulation en m.
11	Vitesse longitudinale du véhicule suivi en mph.
12	Accélération longitudinale en mphps.
13	Détecteur d'occlusion (1 ou 0). Indique si le véhicule est masqué par le véhicule de recueil de donnée.
14	Numéro de la voie

Tableau 5 – En-têtes des données pour les véhicules suivis par les LIDARs.

Il est intéressant de noter que l'on dispose de deux repérages pour les données du véhicule équipé, mais dont les mesures sont données en mètre. Dans l'étude, les positions utilisées sont les positions globales car elles sont communes aux deux bases de données. De plus, ces données contiennent la vitesse et l'accélération, même si elles sont décomposées. Dans notre cas, la composante qui nous intéresse est la composante longitudinale, car elle orientée dans le sens de la route. C'est donc bien cette dernière qui va traduire les accélérations « vers l'avant » des véhicules. La deuxième composante pourrait être intéressante pour analyser les changements de voie, mais ce n'est pas notre objectif.

Ces données recueillies, par Coifman, contiennent à la fois des données GPS et des données LIDAR. Elles sont bien documentées et elles ont été enregistrées avec un but proche des usages de la vidéo. Ainsi à la différence des nombreuses données GPS qui servent à faire du suivi d'itinéraire ou de l'origine-destination, on dispose ici d'un jeu de données précises sur autoroute, dont le parcours est connu. Enfin, elles sont localisées dans un repère local donc en coordonnées planes ce qui va faciliter leur exploitation.

7. Conclusion

Pour conclure, le chapitre 4 va présenter les résultats de la méthode de filtrage polaire sur ces 6 jeux de données. A partir des spécificités de chacune de ces bases, on peut déjà s'attendre à certaines difficultés pour appliquer la méthode. Ceci est résumé dans le tableau suivant :

Base de données	Inconvénient
MOCoPo	Le principal inconvénient de cette base de données vient de la diverse qualité des trajectoires contenues. La sélection étant déjà effectuée, elle ne devrait pas poser de problème.
NGSIM	Ces données sont proches de celle de MOCoPo. Le problème principal vient des nombreuses erreurs intrinsèquement présentes dans la base. Enfin, ces données ont été traitées et elles ne sont pas brutes.
Drone	Cette base de données ne contient que des morceaux de trajectoires (traklets). En l'état, elle n'est pas réellement exploitable.
pNEUMA	Les données de cette base ont été enregistrées dans un milieu urbain. Les trajectoires n'ont pas la même forme que celle des autres bases. Enfin, les coordonnées sont au format GPS ce qui va entraîner une distorsion et un accroissement des erreurs commises.
GPS	Les données présentent deux parcours assez longs. Elles ont, de plus, été intensivement traitées et sont donc peu bruitées. Il est toujours délicat de lisser des données déjà débruitées.
LIDAR	Les données LIDAR sont beaucoup plus courtes tant au niveau des distances que des temps d'enregistrement. Le moindre défaut risque de créer une grosse divergence dans le lissage.

Tableau 6 – Difficultés d'exploitation des bases de données.

Dans le prochain chapitre, on va présenter comment on passe de la méthode de lissage, c'est-à-dire le filtrage polaire, à un algorithme efficace en Python.

Chapitre 3
L'algorithme

1. Introduction

L'algorithme présenté dans ce travail n'est pas une création ex-nihilo. Il reprend toutes les étapes proposées par (Buisson et al., 2016). Néanmoins, sa programmation est très différente de celle réalisée à l'époque, et ce, pour deux raisons :

- Premièrement, l'algorithme de (Buisson et al., 2016) était codé sous Matlab alors que le présent algorithme est codé en Python. Si la majeure partie du code Matlab est transposable, quasiment tel quel en Python, certaines fonctions ne sont pas totalement identiques. Ces différences sont parfois liées à des approches opposées sur l'appel, le paramétrage et la sortie de la fonction. Dans le même ordre d'idée, Matlab compte ces indices en partant de 1 là ou Python commence à 0. Ce décalage d'indice, sans être foncièrement gênant, entraîne néanmoins des adaptations du code notamment dans l'ordre d'exécution des commandes.

- Deuxièmement, si l'algorithme d'origine a bien été conçu pour traiter par le passage en coordonnées polaires, de n'importe quel jeu de donnée, le code a été écrit avant tout pour traiter les données MOCoPo. Pour lui permettre d'être employé sur tous les jeux de données présentés dans le deuxième chapitre, il faut bien sûr l'adapter. Cette adaptation passe par la redéfinition des processus de filtrage des données ou du retour en coordonnées cartésiennes à partir du calcul de la distance polaire.

Dans la suite de ce chapitre, on va commencer par présenter le principe de fonctionnement de lissage en utilisant une spline, puis on détaillera la méthode polaire qui l'utilise.

2. Principes du filtrage par spline

La méthode qui a recours aux splines pour effectuer les lissages sur ces variables. Les splines sont des fonctions mathématiques définies par morceaux. Chaque morceau est constitué par un polynôme dont l'ordre maximal défini lors de la création de la fonction, le plus souvent 3. La particularité des splines provient de le leur fonctionnement basé sur des nœuds. À la base, les splines servaient à la réalisation de tracés en dessin technique où il s'agissait de faire passer une courbe continue par un certain nombre de points prédéfinis comme illustré sur la figure 21 suivante :

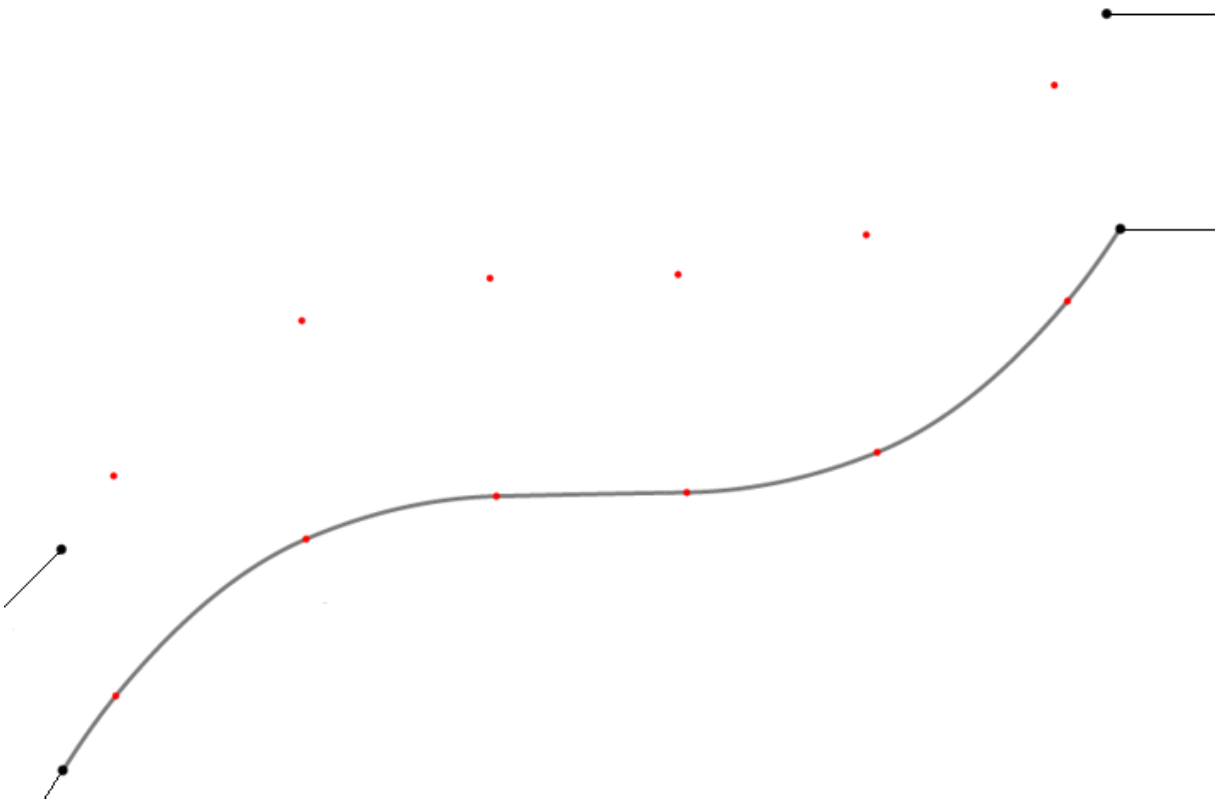

Figure 21 - Illustration de l'utilisation des splines en dessin technique. La pièce à dessiner est matérialisée par le tracer noir. Le dessinateur sait que pour connecter les deux extrimités (points noirs), il doit passer par un certain nombre de positions intermédaires matérialisées par les points rouges. La technique, avant l'informatisation du dessin technique, consistait à planter un clou dans chacune des positions intermédiaire puis de faire passer une règle en bois souple (cerce, représenté en gris) entre les deux positions connues et les clous. La déformation du bois permettait de tracer une courbe élégante qui remplit toutes les conditions.

Le principe mathématique reste le même et les nœuds restent des points cruciaux.

$$Soit\ m\ noeuds\ de\ coordonnées\ (x_1, y_1), \dots, (x_m, y_m).$$

Pour définir la spline, il faut m-1 polynômes P de degré k (1 ≤k ≤5 en règle générale).

$$Avec\ P = a_0 x^k + a_1 x^{k-1} + \cdots + a_k$$

On doit donc définir k+1 coefficients pour m-1 polynômes, soit un total de (k+1)(m-1) inconnues. Il faut donc ajouter des contraintes pour pouvoir résoudre le problème. Ce sont les nœuds qui vont imposer une série de contraintes « aux limites » pour chacun des polynômes. En effet, les polynômes à droite et à gauche d'un nœud doivent avoir la même valeur à son niveau. Comme chaque polynôme est compris entre deux nœuds, cela ajoute 2(m-1) conditions. Cette condition est celle qui s'applique sur le polynôme lui-même et donc elle est généralement notée C0. La condition C1 est la même que la C0 mais elle s'applique sur la dérivée première des polynômes. Deux polynômes consécutifs ont donc la même dérivée première au niveau de leur nœud commun. Cela ajoute (m-2) contraintes supplémentaires. On peut ainsi continuer à ajouter des

conditions tant que les polynômes sont dérivables. Pour nos polynômes de degré k, on peut donc descendre jusqu'à la contrainte C(k-1). On arrive donc à 2(m-1) +(k-1)(m-2).

$$Nombre\ d'inconnues : (k+1)(m-1) = km - k + m - 1.$$

$$Nombre\ de\ contraintes : 2(m-1) + (k-1)(m-2) = km + m - 2k.$$

$$Nombre\ d'inconnue\ restantes : km - k + m - 1 - km - m + 2k = k - 1.$$

Pour fixer ces k-1 inconnues restantes, on a recours à des conditions sur les extrémités. En partant de la plus grande dérivée (la k-1^{ième}) on impose, par exemple, sa nullité aux deux extrémités ce qui nous ajoute 2 contraintes. On remonte ainsi la succession des dérivées jusqu'à disposer des k-1 contraintes manquantes.

Le filtrage par les splines est principalement utilisé avec les données temporelles. Ces dernières doivent être continues, mais la mesure expérimentale ne peut être que ponctuelle. Les splines permette donc d'interpoler les valeurs prises par la courbe entre les points de mesure. C'est pourquoi le fonctionnement général des techniques empiriques de lissage par spline consiste à faire passer une courbe au travers du nuage de points des valeurs expérimentales en minimisant son écart avec l'ensemble de ces points. Ainsi, on va limiter au maximum l'erreur commise sur les portions où il y a le moins de points de mesures disponibles. Cependant, il est à noter que les splines sont très mauvaises pour gérer les « trous » dans les données. Comme on l'a vu dans le paragraphe précédent, cette absence de données ne gêne pas pour calculer la spline, mais elle ne reposera que sur un seul polynôme. Or, ces polynômes sont pertinents quand l'écart entre deux nœuds reste faible. Si on doit prolonger une fonction d'ordre 3, il y a de grandes chances que l'on arrive dans la partie fortement courbée de la fonction comme illustré sur la figure 22. Et cela va rendre ardue la jonction au prochain nœud avec le polynôme suivant.

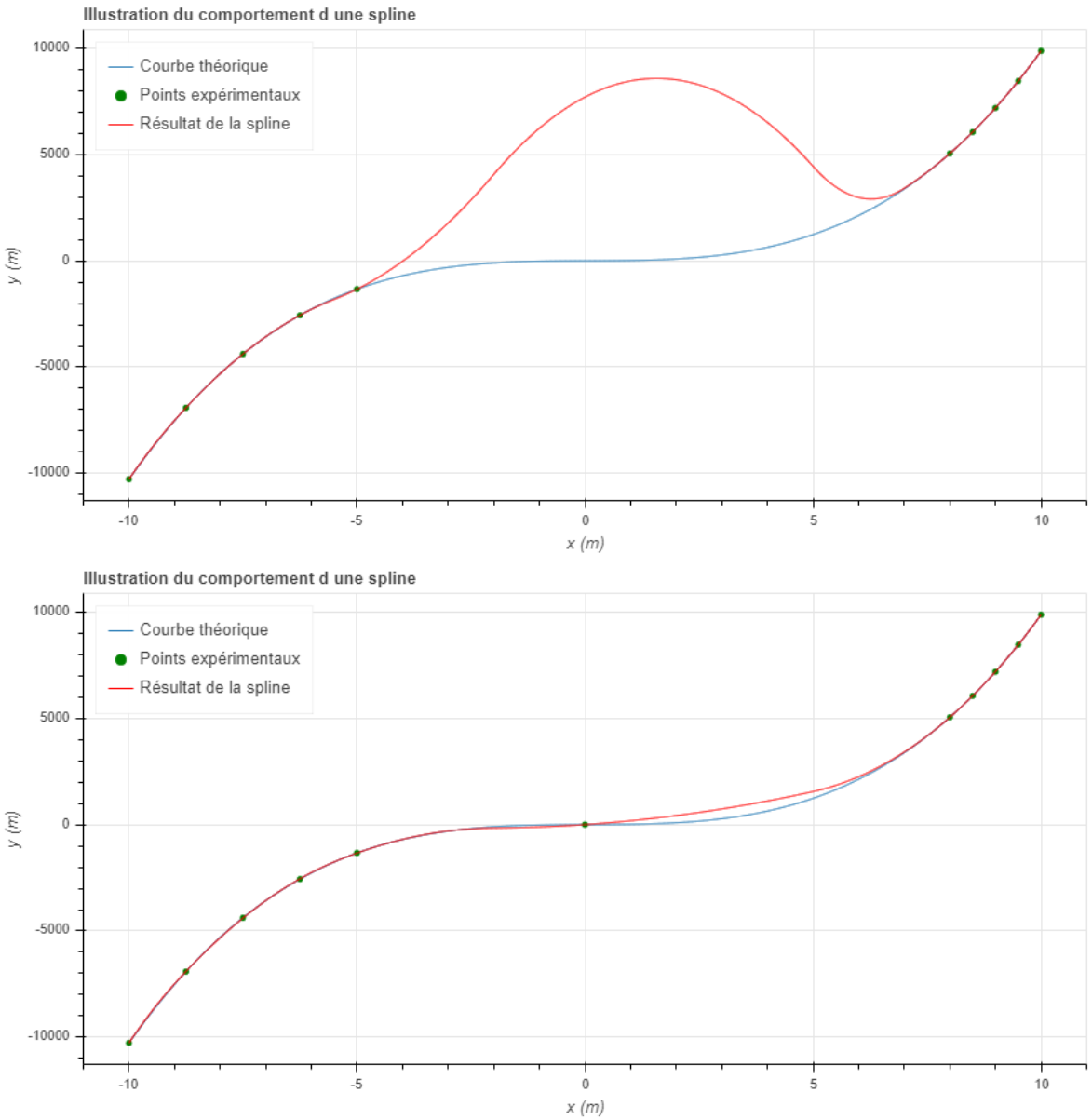

Figure 22 - Illustration de l'effet d'un trou de données sur le comportement des splines. Pour une raison quelconque, on ne dispose pas de point mesure (en vert) entre -5 et 8 mètres, mais on connait la loi théorique (en bleu). Cette absence oblige l'algorithme à continuer à travailler avec un seul polynôme entre -5 et 8. Si on dispose d'un point de mesure dans ce trou, ici le point (0,0), on peut réduire fortement la divergence puisque cette fois, on a deux polynômes sur ce même intervalle [-5,8].

2.1 Création des splines

Pour pouvoir définir correctement une spline, il faut donc fournir aux algorithmes des données avec le moins d'écart possible entre les valeurs de la variable de référence. Du fait que l'on utilise un langage de programmation, on doit respecter des contraintes supplémentaires, liées à la définition mathématique des splines. Si on définit les variables suivantes :

k *le degré de la spline,*

t *le vecteur des noeuds,*

n *le nombre total de points dans les données d'origine,*

m *le nombre de noeuds*,

et x *la variable* à *interpoler, on doit respecter les conditions suivantes* :

Premièrement, Il faut s'assurer de disposer d'un assez grand nombre de points et de nœuds par rapport au degré de la spline :

$$k + 1 \leq m - k - 1 \leq n$$

Que les valeurs de la variable de référence soient strictement croissantes :

$$x(1) \leq x(2) \leq \cdots \leq x(n)$$

Que les nœuds soient également strictement croissant :

$$t(1) \leq t(2) \leq \cdots \leq t(n)$$

Enfin, les nœuds doivent également respecter la condition de Schoenberg-Whitney à savoir :

$$t[i] < x[i] < t[i + k + 1] \; \forall \mathrm{i} \in [0, \mathrm{n} - \mathrm{k} - 2] \;\; condition\ de\ Schoenberg - Whitney$$

Cette condition impose que les nœuds « encadrent » les points à interpoler. De plus cette condition, nous dit que si elle est respectée, alors la spline interpolée est unique.

La librairie qui a été utilisée pour créer les splines est Scipy et son module interpolate. Le choix s'est porté sur cette dernière car son fonctionnement est très proche de celle de Matlab. Malheureusement, elles diffèrent au niveau de la philosophie d'application des splines. Dans Matlab, les splines sont écrites pour faire du lissage. La définition des nœuds est assez libre pour permettre à l'utilisateur de paramétrer sa spline au plus près de ses données. À l'inverse, dans Python, les fonctions de création de spline génèrent automatiquement des fonctions qui passent par tous les points donnés par l'utilisateur. Il ne s'agit plus de faire un lissage, mais plutôt d'extrapoler la fonction correspondant aux points fournis. Cette utilisation peut-être particulièrement pratique pour estimer de manière fiable la dérivée d'une fonction créée à partir d'une série de données propres. Néanmoins, cette différence de philosophie est gênante pour pouvoir traduire le code Matlab en code Python. En Matlab, il suffit de fournir les deux séries x et y et le nombre de nœuds que l'on veut pour qu'il réalise une optimisation du placement des nœuds. Python va lui considérer que chaque point fourni est un nœud et on ne peut pas spécifier le nombre de nœuds que l'on désire. Si on veut moins de nœuds, il faut lui spécifier manuellement où sont ces nœuds et ne pas oublier de vérifier les conditions précédemment citées.

Le problème, c'est que le fonctionnement de l'algorithme sous Matlab repose sur le fait d'ajouter des nœuds. En effet, le principe est de créer de manière itérative des splines avec de plus en plus de nœuds. On évalue alors, pour chaque point crée par la spline, sa distance avec les points d'origines. On continue d'ajouter des nœuds à la spline tant que la distance avec les points d'origines est pour un certain pourcentage réglable à l'appel de la fonction supérieur à une distance cible. Pour illustrer cela, pour la création d'une spline sur le rayon polaire, on continue d'ajouter des nœuds tant qu'il y a plus de 5 % des points qui sont éloignés de plus de 0,7 m.

```
function[s_prime,spline1,Np,knots]=spline_fonction(tps,s,deg,pourcentage_acceptable,deltaPos )
%Fonction qui spline la variable s en fonction de tps
% deg = degré du polynôme
% On fixe le nb point du départ
    Np=10;
    MM=pourcentage_acceptable/100*size(tps,1)+1;
    % On cherche quand on a moins de 5% des points ou dessus de l'erreur de
    % position et on s'arrête si on place plus de point que 1 par seconde
    while(MM>pourcentage acceptable/100*size(tps,1) & Np<max(tps)-min(tps) )
        % Calcul des splines sur les points non optimisés
        spline1 = spap2(Np,deg,tps,s);
        % Optimisation des points
        knots = newknt(spline1);
        % Nouveau calcul de la spline
        spline1 = spap2(knots,deg,tps,s);
        % Calcul de la position en fonction du pas de temps choisi
        pos20 = fnval(spline1,tps);
        % On compte le nombre de point où ça dépasse l'erreur
        MM=sum(abs(pos20(10:end-10)-s(10:end-10))>deltaPos/2);
        Np=Np+1;
    end

   % On définit les paramètres optimaux
    Np = Np-1;
    spline1 = spap2(Np,deg,tps,s);
    knots = newknt(spline1);
    spline1 = spap2(knots,deg,tps,s);

 % On l'applique sur la base de temps
    s_prime=fnval(spline1,tps)';
end
```

Cela n'est pas possible avec Python. La première spline que va créer Python contiendra tous les points d'origine et la distance entre les points de la spline et ces points originaux sera nulle (figure 23). Ainsi, il ne s'agira pas d'un lissage, mais juste de la création d'une fonction passant par l'intégralité des points originaux.

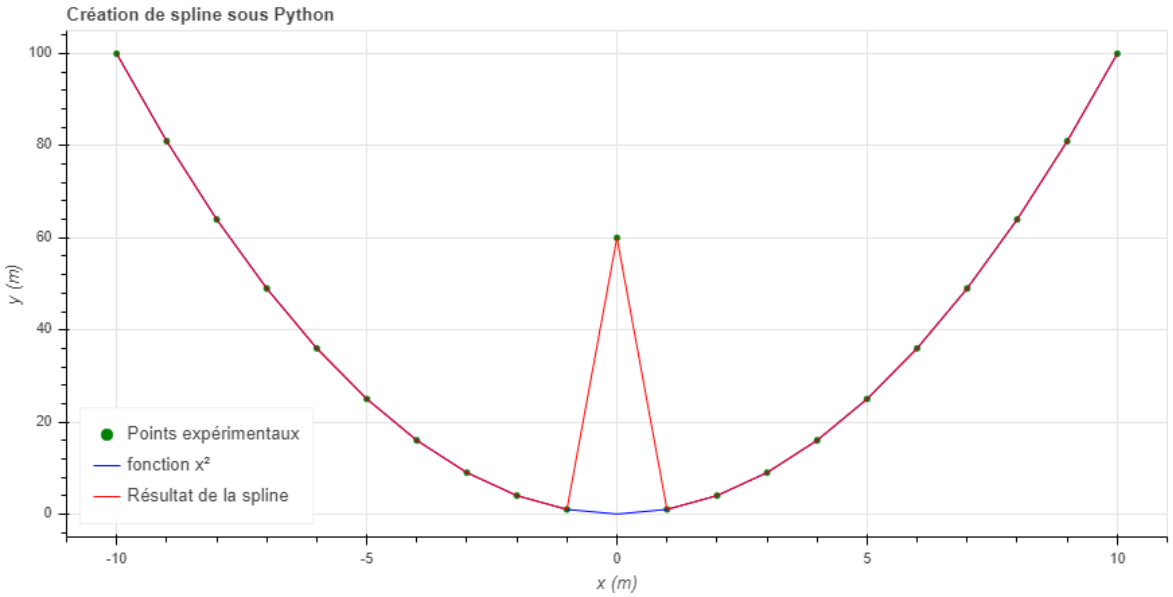

Figure 23 - Résultat de la création d'une spline sur des points suivant la fonction x². Le point 0 a été volontairement faussé, mais là ou Matlab aurait ignoré ce point, visiblement faux, Python considère qu'il s'agit d'un nœud et force donc la spline à passer par ce dernier.

Pour recréer un comportement similaire à Matlab sur Python, il va falloir modifier la structure de l'algorithme.

2.2 Spline basée sur la distribution linéaire des nœuds

Pour contourner cette limitation, il va falloir définir « manuellement » les nœuds plutôt que de laisser le langage de programmation les déterminer. La solution, la plus simple, est de distribuer linéairement les nœuds sur la plage des abscisses (figure 24).

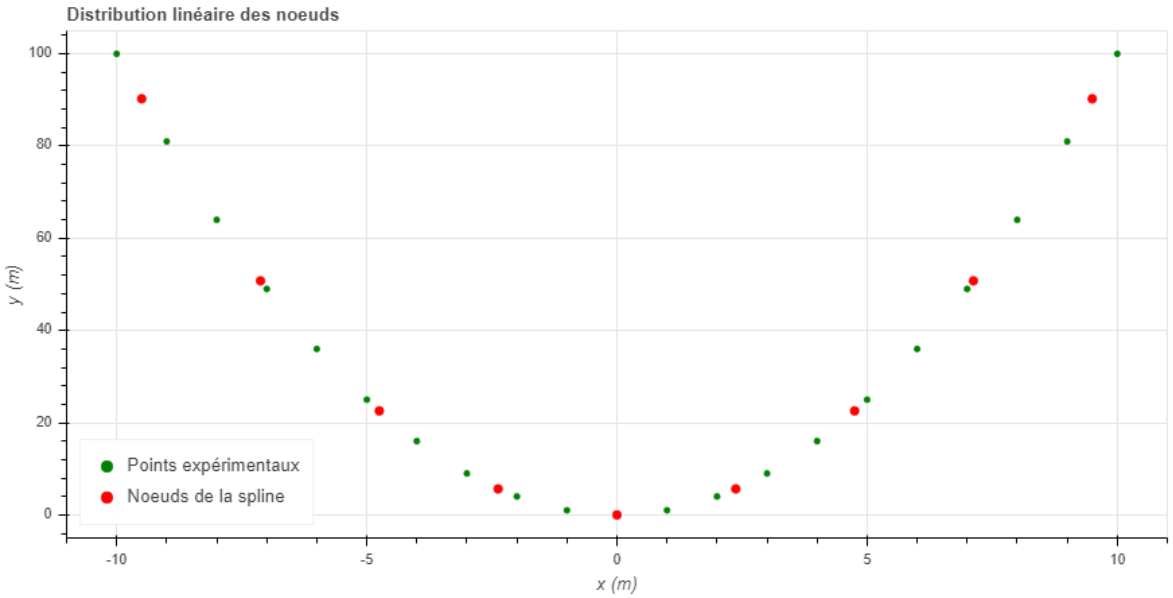

Figure 24 - Distribution linéaire des nœuds. Ici, on a distribué 9 nœuds sur l'intervalle [-9.5,9.5] soit un nœud tous les 2,375 mètres.

L'intervalle entre deux nœuds est fixé par le nombre de nœuds désirés. Par exemple, si la plage des abscisses va de 0 à 10 et que l'on veut un seul nœud, on va le placer au

point 5. Si on veut deux nœuds, on va les placer en 3 et 6, etc. Ainsi, on est sûr que les nœuds respectent les conditions de Schoenberg-Whitney et que l'écart entre les nœuds ne soit pas trop important pour éviter une trop grande dérive des polynômes. Cette méthode présente une particularité qui n'était pas prévu dans la fonction d'origine. Dans la fonction Matlab, l'ajout de nœuds est considéré comme toujours améliorant la qualité de la spline. Or, dans sa transcription en Python, peut-être à cause de la méthode de création des nœuds intermédiaires, ce n'est pas le cas. C'est pourquoi, on ajoute une fonction mémoire de manière à stocker le meilleur résultat. On arrive alors à la réécriture suivante de la fonction :

```
#Spline function avec mémoire
def Spline_function(x,y,deg,acceptable_percentage,deltaPos):
    ''' spline_function(x,s,deg,acceptable_percentage,deltaPos)
    Create a spline from x,y based on a knot linear distribution
    return spline value, spline function, Number of knots, knots position
    '''
    #setting the number of correct point to be attending
    MM = acceptable_percentage / 100 * len(x) + 1
    #creating the memory core, A is a temporary array, all value are high enough to be ignore if they are not fulfill
    A = np.ones((int(x.max()-x.min()),2)) * acceptable_percentage / 100 * len(x) * 10000
    Mem = pd.DataFrame(A,columns=['Np','MM'])
    #setting condition for loop interruption
Complete = 0
    #main loop : each time we add a knots in the spline until we get a spline matching the condition
    for Np in range(1,int(x.max()-x.min())):
        #setting the knots
        knot_offset = (x.values[-2] - x.values[2]) / (Np+1)
        knots = np.linspace(x.values[2] + knot_offset, x.values[-2] - knot_offset, Np)
        #LSQUnivariateSpline is not a very stable function so we catch the numerous error it can produce
        try :
            #Creating the spline function with the previous knots
            spline = interpolate.LSQUnivariateSpline(x,y,knots,bbox=[x.min()-1e-9,x.max()+1e-9],k=deg)
            #Calculating distance, equivalent to L2 norm, between the new y and the old one
            #but without the tenth first and last points
```

```
            MM = np.abs(spline(x)[10:-10] - y[10:-10])
            #Getting the number of point where distance is bigger than deltaPos/2 because we use absolute distance
            MM = len(MM[MM > deltaPos / 2])
            #we stock the value of knots'number and the number of wrong point
            Mem.Np[Np] = Np
            Mem.MM[Np] = MM
        #ValueError is raise when the knots are not well defined in this case we make the memory value with a high value
        except ValueError:
            MM = acceptable_percentage / 100 * len(x) * 10000
        #if we match the condition set in the call of the function we stop the loop for calculation gain
        if MM < acceptable_percentage / 100 * len(x):
            #set Complete to 1 and breaking the loop
            Complete = 1
            break
    #if the condition are met before the end of the loop we use the last spline created
    #instead of finding the best in the memory core
    if Complete == 1 :
        s_prime = spline(x)
  knots = spline.get_knots()
    #if the loop ended, we search the best attempt in the memory core. Due to the non-linear comportement of the spline creation
    #the last isn't always the best try.
    else :
        #first we must assure that there is only one best attempt
        if len(Mem.Np.values[Mem.MM == Mem.MM.min()] ) == 1:
            Np = int(Mem.Np.values[Mem.MM == Mem.MM.min()])
        #if there is more than one best attempt we take the one with the lesser number of knots
        else :
            Np_array = Mem.Np.values[Mem.MM == Mem.MM.min()]
            Np = int(Np_array.min())
```

```
        #we recreate the spline from memory core informations
        knot_offset = (x.values[-2] - x.values[2])/(Np+1)
        knots = np.linspace(x.values[2] + knot_offset, x.values[-2] - knot_offset, Np)
        spline = interpolate.LSQUnivariateSpline(x,s,knots,bbox=[x.min()-1e-9,x.max()+1e-9],k=deg)
        s_prime = spline(x)
        knots = spline.get_knots()
        #returning the calculated value, spline function,knots number and position (mainly for control purpose)
    return(s_prime,spline,Np,knots)
```

2.3 Spline basée sur les points tournants

Cet algorithme fonctionne bien, mais quand il y a beaucoup de variations de la fonction d'origine, la distribution des nœuds a du mal à refléter ces variations. Une des solutions qui aurait pu être employée aurait été d'utiliser des nœuds de Tchebychev. Ces nœuds sont efficaces pour supprimer les phénomènes de Runge, c'est-à-dire des oscillations en début et fin de courbe d'interpolation. Néanmoins, les splines ne sont normalement pas sujettes à ce phénomène. C'est pourquoi, pour améliorer le placement des nœuds et les mettre à des endroits plus stratégiques de la courbe d'origine, on va utiliser la notion de « point tournant ». Un point tournant est :

« Un point où la dérivée change de signe. Un point tournant peut être un maximum ou un minimum local. Si la fonction est dérivable en ce point, alors un point tournant est un point stationnaire ; la réciproque est fausse. » source wikipédia : https://fr.wikipedia.org/wiki/Point_stationnaire

Ce que l'on peut illustrer ainsi (figure 25) :

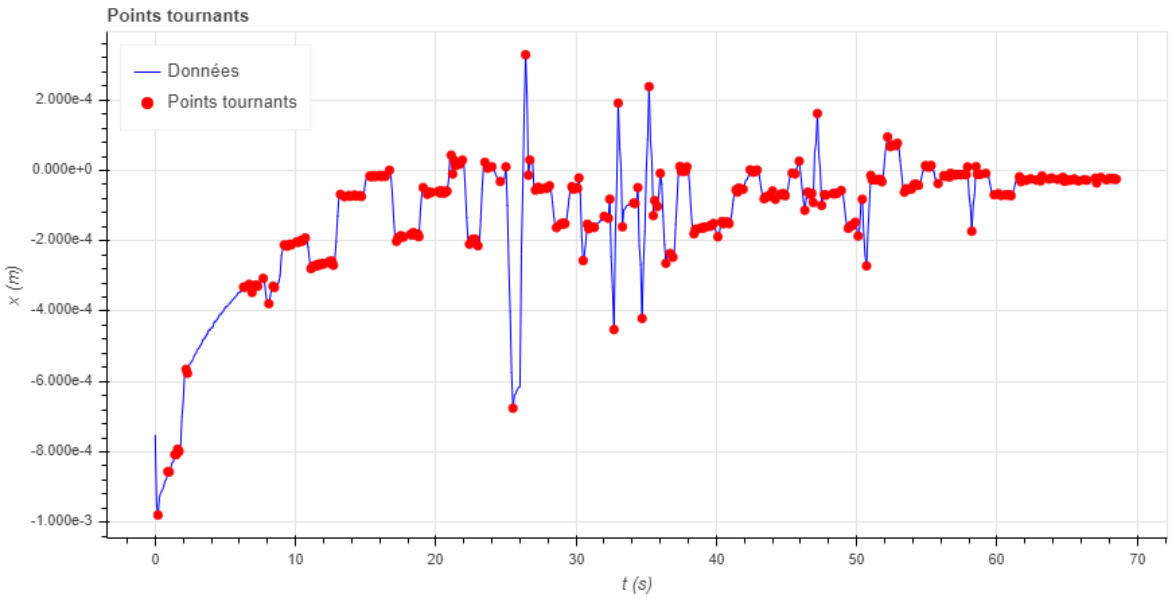

Figure 25 - illustration des points tournants (en rouge) sur un échantillon de données issues des données NGSIM.

La fonction va donc d'abord localiser ces points. Il se peut qu'il y ait trop de points tournants sur certains segments, ou que les données comprennent des micro-variations trop fréquentes, voir les deux problèmes en simultané. L'objectif est bien de trouver les points tournants majeurs qui vont permettre de placer au mieux les futurs nœuds de la spline. On peut donc éliminer une partie des points qui n'ont pas d'intérêt, comme ceux en bout de la figure 25. Pour remédier à cela, on ajoute un paramètre de pente pour éliminer les petites variations (figure 26). Tant que l'écart de pente reste dans des valeurs inférieures à ce paramètre, on considère que la fonction est « constante ».

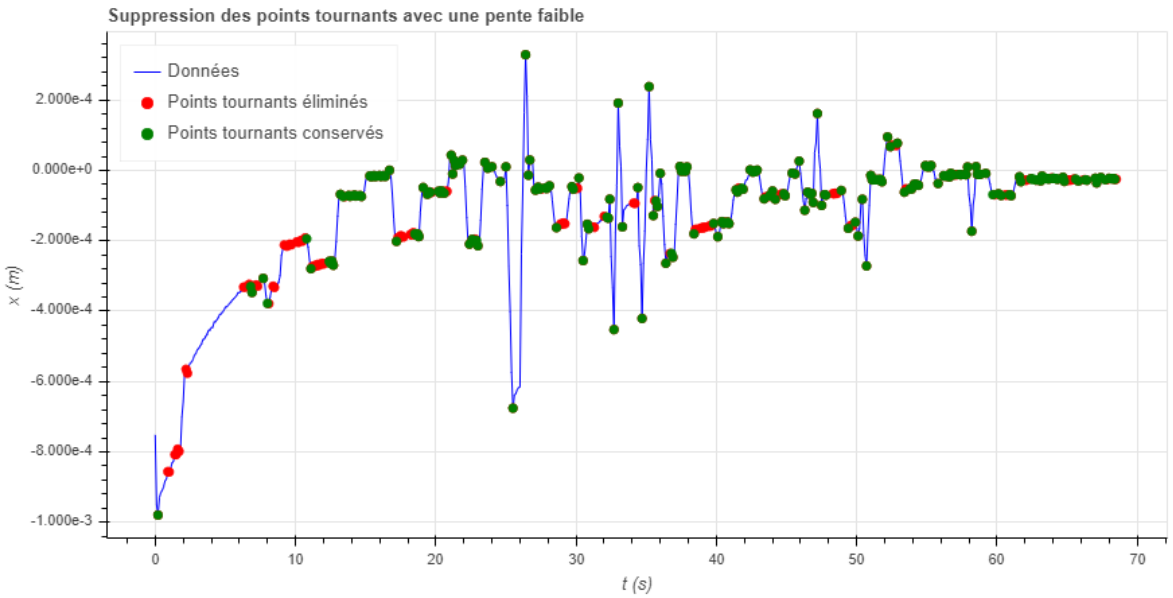

Figure 26 - Illustration du procédé de suppression des nœuds correspondant à des faibles valeurs de pente.

Ensuite, il peut encore rester des points tournants intéressants, mais en trop grande densité sur certaines parties de la courbe comme sur l'intervalle [50,60] de la figure XX. Si on garde ces concentrations de points pour les nœuds, cela va « déséquilibrer » la spline. La correction qui va être introduite va consister à ne garder que les points tournants coïncidant aux plus grandes variations (extremum locaux). On ajoute un paramètre pour définir la distance minimale entre deux points tournants conservés (figure 27).

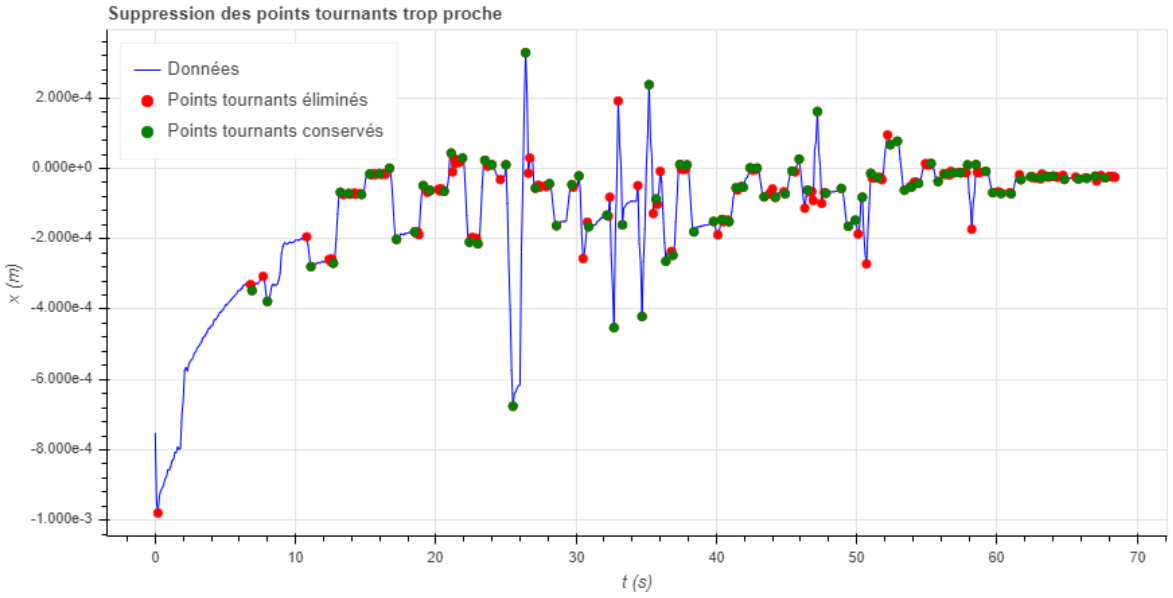

Figure 27 - illustration du résultat après suppression des nœuds trop proche qui nuisent à l'équilibre de la spline.

La fonction transforme ensuite les points tournants sélectionnés en nœuds qui sont passés à la fonction de création de spline. À la différence de la fonction précédente, le processus n'est pas itératif, on considère que si la sélection des points tournants a été faite correctement, on obtient une spline dont le lissage peut être visuellement qualifié de satisfaisant (figure 28).

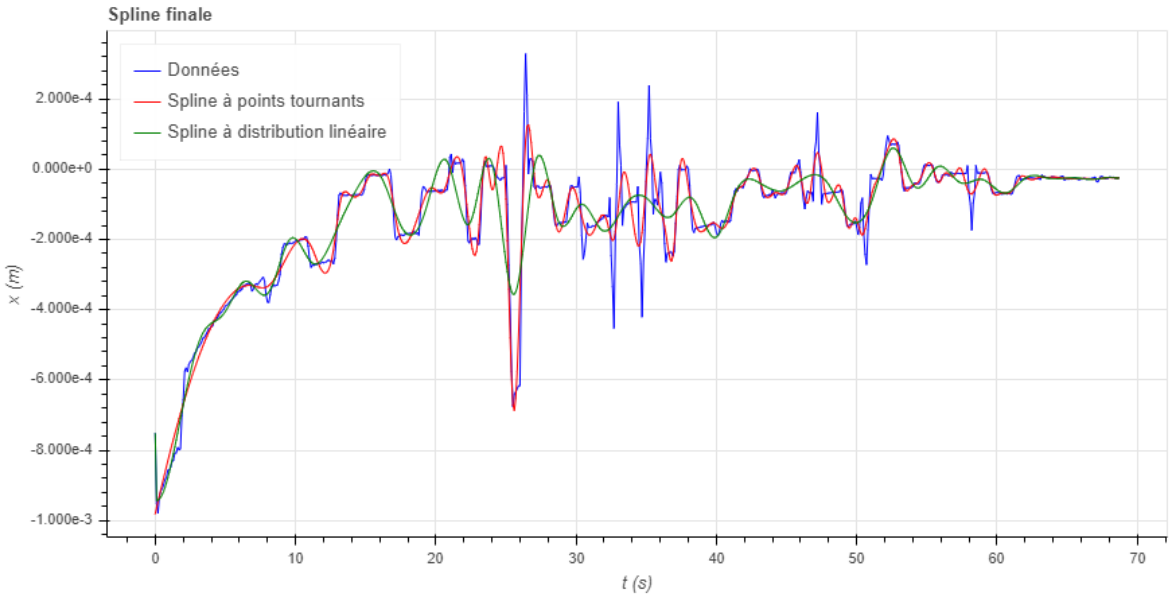

Figure 28 - Résultat du lissage avec la spline à point tournant (en rouge). Les données d'origine sont représentées en bleu et le résultat d'une spline linéaire en vert pour comparaison.

3. Concept de l'algorithme

L'algorithme utilisé dans cette étude est basé sur la méthode polaire présentée succinctement au chapitre 1. L'idée sur laquelle repose cette méthode vient de la difficulté que l'on peut avoir à travailler directement sur les coordonnées cartésiennes. En effet, quand on veut traiter des données planes, les erreurs commises sur l'abscisse et l'ordonnée ne sont pas indépendantes. Dans le cas de données vidéos, les positions des points de référence (centres de gravité ou points centraux avant des véhicules) sont identifiées à partir d'une reconnaissance de pixels. Cela permet de créer le blob avec les multiples erreurs qui ont été présentées au chapitre 2. Ces erreurs impactent généralement les deux coordonnées, il n'y a pas de raison qu'il y ait plus d'erreurs suivant x que y. Cela vient du fait que les pixels les plus problématiques sont généralement dans les coins. De plus, x et y sont liés par la distance parcourue par le véhicule. Ainsi faire des corrections d'abord sur x puis sur y n'est pas équivalent à faire des corrections d'abord sur y puis x. Et dans les deux cas, la distance parcourue aura varié.

C'est pourquoi les méthodes de traitement présentés dans le chapitre 1, qui sont appliquées sur les coordonnées cartésiennes, sont largement basées sur l'application de filtres. Ainsi, le but est plus d'éliminer les plus mauvais points que de recaler réellement la trajectoire. Un procédé généralement employé en cinématique pour restreindre ce problème de « coordonnées liées » est de passer en coordonnées polaires. En effet, r et θ sur des portions « droites », ils sont quasi-indépendants et peuvent donc être filtrés en ne commettant pas l'inévitable erreur que l'on commet sur x et y. Cependant, dès que les trajectoires vont devenir circulaire, comme ça peut être le cas dans le données pNEUMA (figure 18), cette quasi-indépendance n'est plus de mise. Il sera donc à confirmer que la méthode peut s'adapter à de telles données.

Le filtrage sur r permet de rectifier la position des points par rapport à l'origine de la trajectoire pour éviter les phénomènes d'ondulation parasite. Le filtrage sur θ permet, lui, de corriger les légères variations de la direction du véhicule. Convertir les coordonnées polaires lissées en coordonnées cartésiennes ne présente pas de difficulté, mais la méthode ne consiste pas qu'en un simple changement de repère. Elle permet aussi de lisser les valeurs exagérées de vitesse, d'accélération et de jerk qui peuvent persister après le lissage en polaire. C'est pourquoi, on se sert des coordonnées polaires propres pour calculer la distance « polaire », l'interpoler puis revenir aux coordonnées cartésiennes pour constater le résultat de la méthode. C'est cette dernière étape de retour aux coordonnées cartésiennes à partir de la distance polaire filtrée qui est problématique. Cela sera illustré dans la partie correspondante un peu plus loin dans ce chapitre.

4. Structure de l'algorithme

La structure générale de l'algorithme est toujours celle présentée dans l'article de (Buisson et al., 2016) et illustré sur la figure 29 suivante :

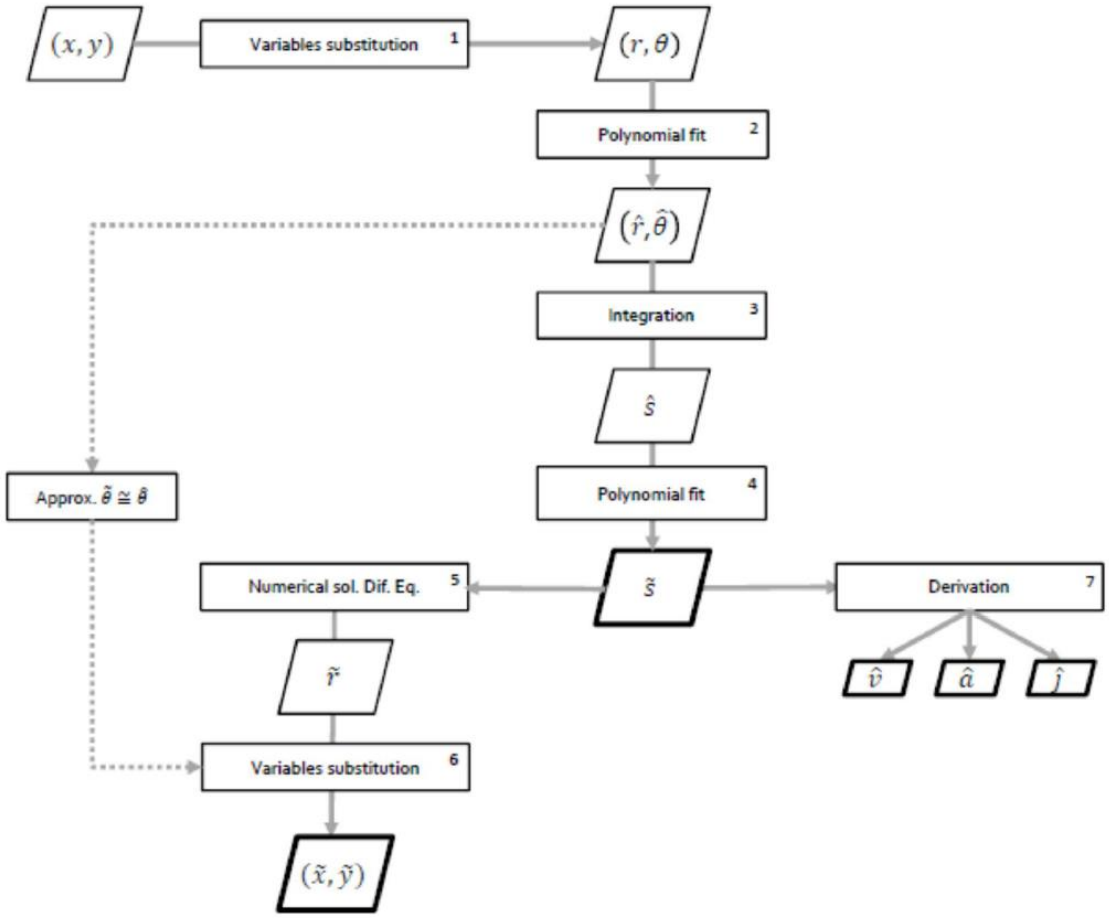

Figure 29 - Schéma du fonction de l'algorithme - source (Buisson et al., 2016).

La boite 1 consiste à passer des coordonnées cartésiennes aux coordonnées polaires.

La boite 2 effectue un lissage de valeur de r et θ ce qui permet à la fois de supprimer une partie des points aberrants, mais également de créer une fonction pour les deux coordonnées polaires.

Dans la boite 3, on dérive les fonctions interpolées de r et θ : $\hat{r}$ et $\hat{\theta}$. Ensuite, on les réalise alors une intégration suivant l'équation suivante pour obtenir la distance « polaire » (qui est équivalente à l'abscisse curviligne) :

$$\hat{s}(t) = \int_0^t \left[\left(\frac{d\hat{r}}{dt} \right)^2 + \left(\hat{r} \times \frac{d\hat{\theta}}{dt} \right)^2 \right] dt \quad (1)$$

On peut maintenant, dans la boite 4, faire un nouveau lissage sur la distance polaire que l'on vient de calculer afin d'avoir une fonction propre ($\tilde{s}$) pour les dérivations successives de la boite 7.

La boite 5 est la seule qui va connaître un grand changement. La résolution de l'équation différentielle a été remplacée par une résolution géométrique du problème de placement des points connaissant $\tilde{s}$.

A partir de ce résultat, on peut revenir aux coordonnées cartésiennes, ce qui est fait dans la boite 6.

Enfin, dans la boite 7, on calcule la nouvelle vitesse ($\hat{v}$), accélération ($\hat{a}$) et le jerk ($\hat{j}$) par dérivation successive de $\tilde{s}$.

Le fonctionnement détaillé de chacune de ces étapes va être exploré dans la suite de ce chapitre, le code complet utilisé pour les données MOCoPo est disponible en annexe.

4.1 Passage aux coordonnées polaires

Pour effectuer un passage aux coordonnées polaires, on pourrait utiliser les formules de conversion bien connue. C'est ce qui était fait dans la version Matlab, en utilisant les formules suivantes :

$$r = \sqrt{(x - x_0)^2 + (y - y_0)^2}$$

$$\theta = \tan^{-1}\left(\frac{(x - x_0)}{(y - y_0)}\right)$$

avec x_0 et y_0 les origines de la trajectoire.

Néanmoins, ces formules posent deux problèmes. Tout d'abord, il faut définir l'origine de la trajectoire, ce qui n'a rien d'évident. Le code Matlab résolvait ce problème en créant une spline pour interpoler le point d'intersection de la trajectoire avec l'axe des ordonnées. Ensuite, cette méthode ne permet pas de réaliser l'ambition de généralisation de l'usage de l'algorithme avec tous les types de données. En effet, la formule utilisée pour le calcul de θ n'est valable que si on travaille avec x et y positif. Il faut donc utiliser une formule plus universelle. Il aurait été possible d'utiliser la formulation avec *atan2,* mais le plus simple reste de passer par l'écriture complexe. Ce faisant, on laisse à Python le soin de choisir de lui-même la valeur de l'angle. La nouvelle formulation du changement de coordonnées est donc la suivante :

$$z = x + i * y$$

$$r = |z| \quad \text{et} \quad \theta = \arg(z)$$

Maintenant, il ne faut pas oublier que l'on travaille sur des vecteurs ce qui donne un code légèrement différent (recours à la librairie numpy, abrégé np) :

```
#Creating the complex number z from x and y

z = x + 1j * y

#Computing r and th from z

r = np.abs(z)

th = np.angle(z)
```

Maintenant que les variables ont été substituées, on va pouvoir les filtrer.

4.2 Application de ces splines « à points tournants » sur les variables de substitutions

Pour cette étape, on veut effectuer un lissage pour supprimer les bagotements[3] liés à l'origine des données. Mais il est important de ne pas trop déformer les courbes originales surtout pour θ. En effet, les variations sont par nature assez faible, et si la spline ne les suit pas, on va se retrouver avec une forte différence quand on va repasser

[3] Oscillation erratique entre plusieurs valeurs ou états.

aux coordonnées cartésiennes. C'est pourquoi, on va utiliser des splines basées sur les points tournants pour faire ces deux lissages (figures 30 et 31). Ainsi, on s'assure de ne pas créer une trop grande distorsion avec les données originales. C'est d'autant plus important que l'on intègre dans l'étape suivante.

```
#Creating R and thêta spline with turning point based knots
interpR = TPSpline(t,r,deg_R,slope_R,MaxKnotsR)
interpTh = TPSpline(t,th,deg_Th,slope_Th,MaxKnotsTh)
#Calculing new r and thêta from spline function
rop = interpR(t)
thop = interpTh(t)
```

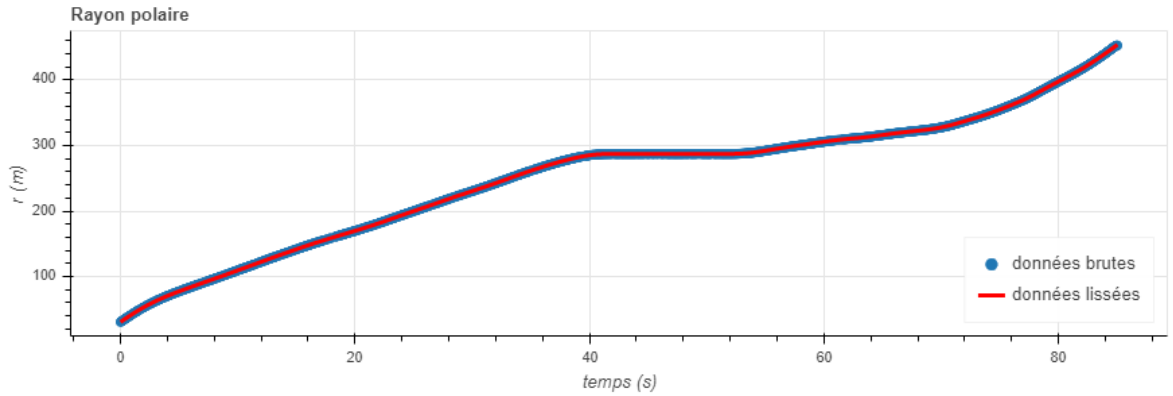

Figure 30 - Résultat de l'application du lissage sur le rayon polaire. Données issues de la base MOCoPo.

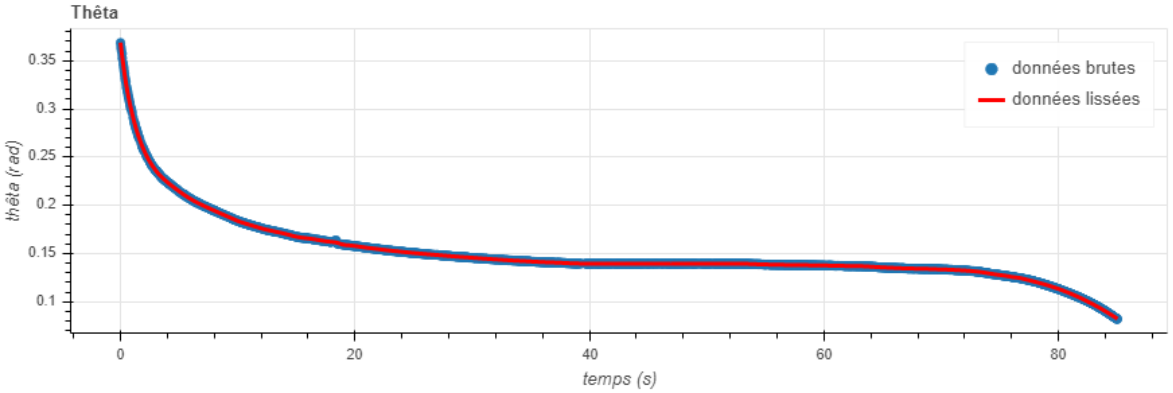

Figure 31 - Résultat de l'application du lissage sur thêta. Données issues de la base MOCoPo.

4.3 Calcul de la distance polaire

Maintenant, on va calculer la distance polaire. La distance polaire est particulièrement utile pour le filtrage de trajectoire, car c'est à partir de cette dernière que l'on va pouvoir obtenir la vitesse, l'accélération et le jerk. Pour ce faire, on intègre la relation entre *s*, *r* et θ :

$$\left(\frac{d\hat{s}}{dt}\right)^2 = \left(\frac{d\hat{r}}{dt}\right)^2 + \hat{r}^2 \times \left(\frac{d\hat{\theta}}{dt}\right)^2 \quad (2)$$

$$\hat{s}(t) = \int_0^t \left[\left(\frac{d\hat{r}}{dt} \right)^2 + \left(\hat{r} \times \frac{d\hat{\theta}}{dt} \right)^2 \right] dt \quad (3)$$

D'un point de vue informatique, cette intégration ne pose aucun souci. Les fonctions python étant équivalentes à leur homologue Matlab, il s'agit juste d'une « traduction ».

```
#Calculing polar distance derivative from the new r and thêta and their derivatives

ds = np.sqrt(interpR(t,1)**2+(interpTh(t,1)*rop)**2)

#integration of ds for getting polar distance s

s = integrate.cumtrapz(ds,t,initial=0)
```

4.4 Lissage de la distance polaire

Comme le but du passage en coordonnées polaires est aussi de lisser les changements trop fréquents de valeur d'accélération et de jerk, on va appliquer une spline sur la distance polaire. Cette spline sera une spline « linéaire », car le but est d'avoir des nœuds bien distribués sur toute la durée. Ainsi, on s'assure de ne pas avoir une trop grande différence entre la distance totale originale et celle lissée. Si le lissage raccourcit trop la distance totale originale, les résultats ne seront pas exploitables quand on va repasser en coordonnées cartésiennes. Selon les données, on fait varier les paramètres de la fonction de création de spline pour obtenir le meilleur résultat possible. Du point de vue du code, il s'agit d'un simple appel à la fonction de création de la spline.

```
#Making a linear knots based spline for s

[s_prime,interpS,Np,knots] = Spline_function(t,s,deg_S,acceptable_percentage_S,deltaS)
```

4.5 Intégration de l'équation différentielle pour passer de la distance polaire au rayon

La méthode programmée en Matlab reposait sur l'intégration de l'équation différentielle suivante issue de l'équation (2) :

$$\frac{d\tilde{r}}{dt} = \sqrt{\left(\frac{d\tilde{s}}{dt} \right)^2 - \tilde{r}^2 \left(\frac{d\tilde{\theta}}{dt} \right)^2} \quad (4)$$

Le problème de cette méthode, c'est que l'on ne connaît que $\tilde{s}$, on doit donc faire une hypothèse sur θ. On va donc considérer que le lissage effectué sur $\hat{s}$ n'a que peu fait varier θ et donc que $\tilde{\theta} \cong \hat{\theta}$. Néanmoins, la résolution de cette équation différentielle n'a rien de triviale. La méthode implémentée en Matlab fonctionne bien sur les données MOCoPo, mais n'a pas été conçue pour résoudre tous les cas possibles. Notamment, elle ne considère qu'une seule solution de l'équation différentielle, or celle-ci étant de degré 2, elle admet systématiquement deux solutions (parfois confondues). Si on veut prendre en compte ces deux solutions, la méthode écrite en Matlab n'est pas adaptée, et il faut résoudre l'équation d'une autre manière.

À partir $\tilde{s}$, on peut connaître la distance entre chacun des points de la trajectoire. En travaillant dans le plan complexe, cette distance peut se voir comme le rayon d'un cercle centré sur le point précédent. Pour continuer, on a besoin de poser une hypothèse :

$$Hypothèse : le\ lissage\ de\ \hat{s}\ en\ \tilde{s}\ n'a\ eut\ que\ peu\ d'effet\ sur\ \theta. On\ pose\ donc\ que\ \tilde{\theta} \cong \hat{\theta}$$

Cette hypothèse reste acceptable parce que s à un comportement plus proche de r que de θ. Maintenant, on peut tracer la droite qui passe par les points ayant pour argument $\tilde{\theta}$. Pour faciliter cette résolution, on peut effectuer un changement d'origine du repère initial. En utilisant le premier point de la trajectoire comme origine de ce nouveau repère, l'équation de la droite que l'on cherche devient tout simplement :

$$\theta = \tilde{\theta}$$

L'intersection de cette droite avec le cercle précédent permet de connaître les deux positions acceptables pour le prochain point (figure 32). Pour déterminer lesquels de ces deux points est le bon, on va simplement calculer leur distance au point d'origine (exprimé dans le plan complexe).

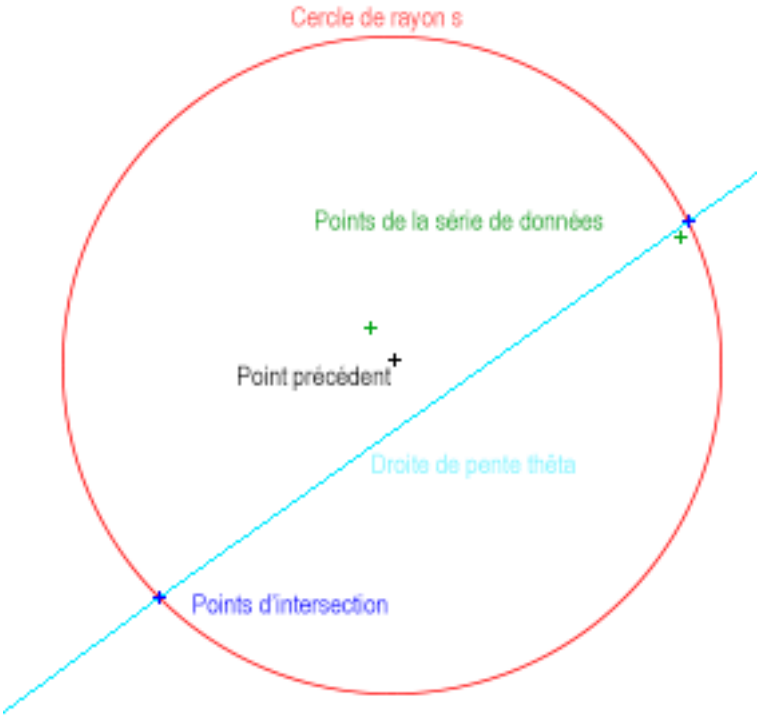

Figure 32 - Illustration de la représentation géométrique utilisée pour déterminer les coordonnées des points en connaissant $\tilde{s}$.

Le point le plus près est considéré comme bon, c'est-à-dire que l'on fait l'hypothèse suivante :

$$Hypothèse : le\ lissage\ de\ \hat{s}\ n'a\ pas\ entraîné\ de\ distortion\ majeure\ dans\ la\ continuité\ du\ placement\ des\ points.$$

C'est une hypothèse forte, et si elle semble assez raisonnable, il n'est pas forcément évident qu'elle soit toujours vérifiée. La détermination du bon point s'effectue de la manière suivante :

$$Soit\ z_1\ et\ z_2\ les\ deux\ points\ d'intersection\ entre\ le\ cercle\ et\ la\ droite.$$

$$\hat{z} = min(|z - z_1|, |z - z_2|)$$

Comme on a besoin du point précédent pour déterminer la position du point suivant, on procède de proche en proche en partant du premier point qui maintenant correspond à l'origine du plan complexe. La résolution présentée n'est pas optimale. C'est clairement le point faible de l'algorithme. Le lissage de la distance polaire permet de s'assurer que les valeurs de l'accélération et du jerk seront plus réalistes mais une fois ce lissage fait, il va être très difficile de pouvoir retrouver les modifications qu'il a engendrées sur la trajectoire.

```
#Complex solving method
#Declaring list for saving data
X = []
Y = []
zr = []
#initialisating first value
zr.append(cmath.rect(r.iloc[0],th[0]))
X.append(zr[0].real)
Y.append(zr[0].imag)
#We parse through the s vector
for i in range(1,len(s)) :
    #setting last r as new depart point
    r0 = cmath.polar(zr[i - 1])[0]
    #setting the distance from s
    Rayon = s[ i ] - s[i - 1]
    #calculating the two possibles solutions
    zr1 = cmath.rect((r0 + rayon),thop[ i ])
    zr2 = cmath.rect((r0 - rayon),thop[ i ])
    #calulating distance from the original data
    dist1 = np.abs(z.iloc[i]-zr1)
    dist2 = np.abs(z.iloc[i]-zr2)
    #we keep the smallest distance
    if dist1 <= dist2 :
        zr.append(zr1)
    else :
        zr.append(zr2)
```

4.6 Retour aux coordonnées cartésiennes

Cette partie contrairement à la partie précédente ne présente aucune difficulté. Connaissant les valeurs de $\hat{z}$, on réapplique la formule suivante :

$$X = \mathrm{Re}(\hat{z})$$

$$Y = \mathrm{Im}(\hat{z})$$

```
#Getting back from complex to cartesian
X.append(zr[i].real)
Y.append(zr[i].imag)
```

Ce qui ne présente aucune difficulté et nous permet d'arriver au résultat suivant illustré sur la figure 33 :

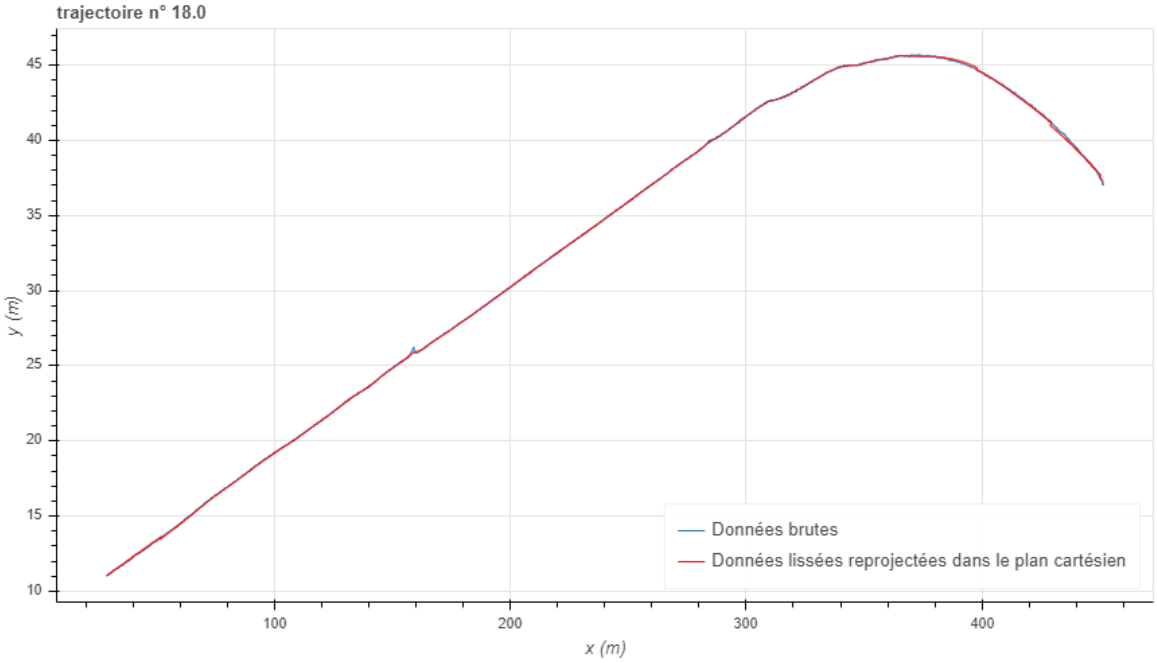

Figure 33 - Résultat de la reprojection dans le plan cartésien pour comparer à la trajectoire d'origine. Données provenant de MOCoPo.

4.7 Dérivations successives de la distance polaire

La distance polaire ne sert pas qu'à revenir aux coordonnées cartésiennes. La distance polaire étant normalement égale à la distance « classique », on peut donc en déduire la vitesse, l'accélération et le jerk. Il suffit juste de dériver. La vitesse est la dérivée première, l'accélération, la dérivée seconde et le jerk, la dérivée troisième. Vu que l'on a créé une spline pour faire le lissage de cette distance, on dispose directement d'une fonction. La dérivation d'une fonction ne présentant aucune difficulté en Python, on obtient rapidement les valeurs recherchées (figure 34).

```
#Getting Distance, Velocity, Acceleration and Jerk from the s spline function and its derivative
Distance = interpS(t)
Velocity = interpS(t,1)
#numeric calculation can lead to near 0 negative velocity that are corrected here
Velocity [Velocity < 0] = 0
Acceleration = interpS(t,2)
Jerk = interpS(t,3)
```

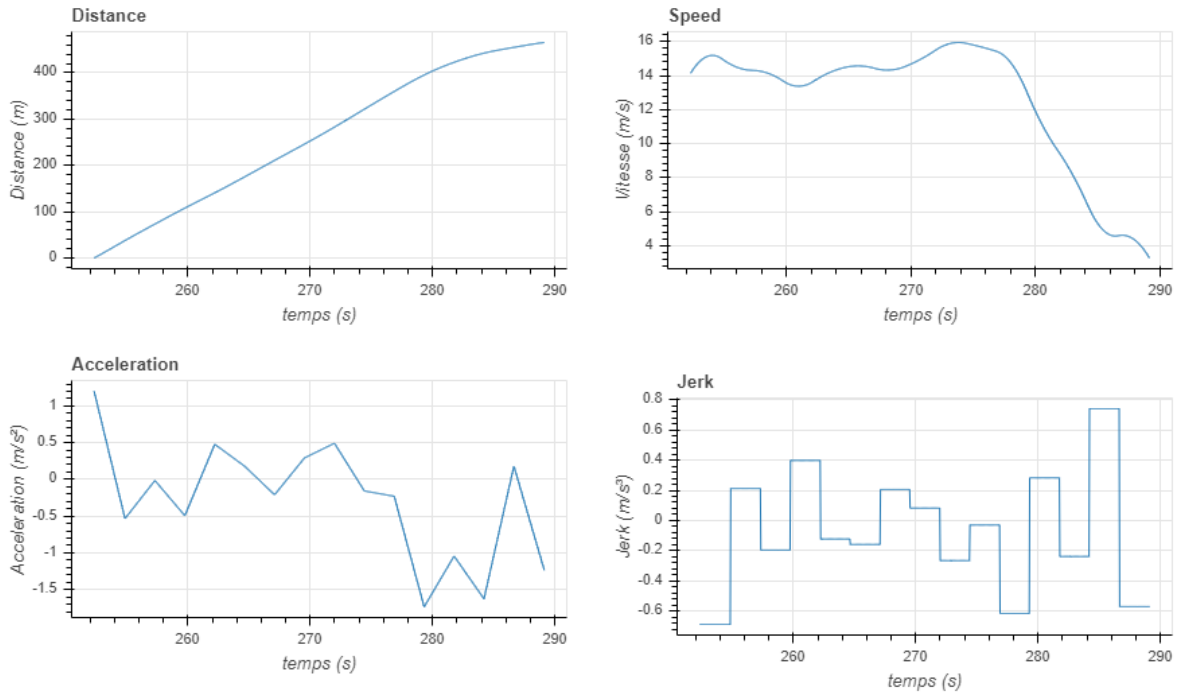

Figure 34 - Résultat de la dérivation successive de la distance. Données d'illustration provenant de MOCoPo.

5. Conclusion

Les changements à la fois d'objectif et de langage de programmation ont conduit à la réalisation d'un certain nombre de changement dans l'algorithme. Toutefois, l'esprit même de la méthode de filtrage polaire a été conservé. Cependant, la phase de retour dans le plan cartésien reste délicate. La méthode employée pour éviter la résolution délicate de l'équation différentielle (4) ne donne pas toujours de bons résultats. Cela ne veut pourtant pas dire que le lissage effectué sur la distance polaire a été trop violent et qu'il a complétement déformé la trajectoire d'origine. Pourtant, il est difficile de justifier le contraire si le retour dans le plan cartésien est trop imprécis (figure 35).

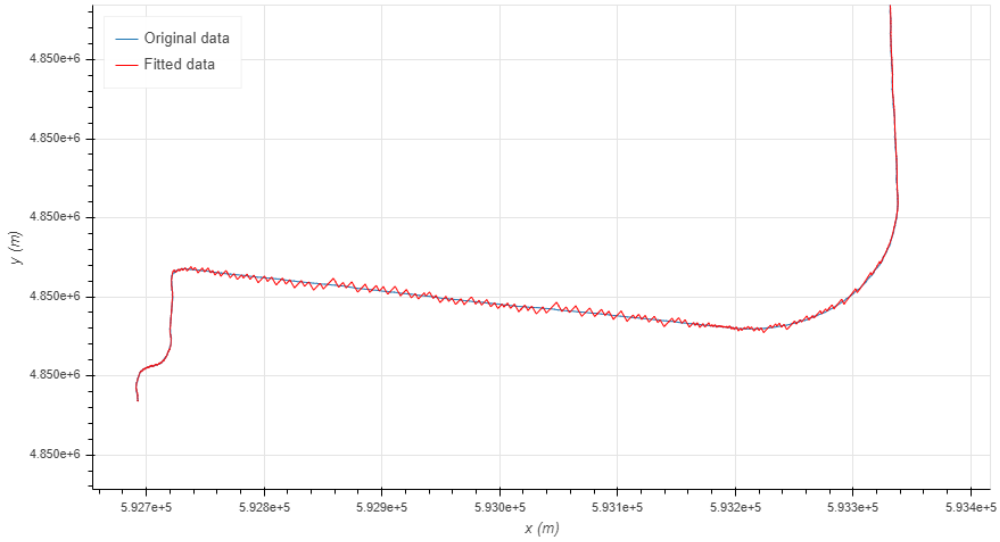

Figure 35 - illustration des erreurs commises par « la résolution de l'équation différentielle ».

Actuellement, et jusqu'à ce qu'une nouvelle méthode de résolution de l'équation différentielle soit implémentée, il s'agit de la plus grande faiblesse du procédé. Cet avertissement ayant été donné, nous allons pouvoir passer à son application dans le chapitre suivant.

Chapitre 4
Résultats de l'application de la méthode

1. Introduction

Pour savoir si le traitement des données a été efficace, il faut définir des indicateurs. Dans (Marczak and Buisson, 2012), on peut trouver des pistes indiquant quels sont les indicateurs intéressants et la raison de leur pertinence. Ceux qui ont été choisis dans cette étude sont les suivants :

- La distribution des accélérations : La distribution des accélérations doit être centrée sur 0 avec peu de valeurs qui doivent être supérieures à ± 2 m/s², et aucune au-delà de ± 3 m/s².

- La distribution des changements de signe du jerk : Le jerk ne doit que rarement changer de signe en moins d'une seconde. En effet, en prenant en compte les temps de réaction humain et machine, il est difficile de descendre en-dessous de cette seconde.

- La distribution des temps entre les changements de valeur du jerk : Dans le même d'ordre d'idée, le jerk ne doit pas trop changer en moins d'une seconde ni prendre trop de valeurs différentes.

Ce chapitre va permettre de faire des comparaisons entre les données brutes et les données traitées avec l'algorithme sur ces trois critères. Néanmoins, on ne dispose jamais de la valeur du jerk dans les données brutes et ce quelle que soit la source de données. Cependant, certaines sources contiennent les valeurs d'accélération et ce sont donc ces dernières qui seront utilisées. Pour les données pour lesquelles on ne dispose d'aucune autre information que les coordonnées cartésiennes, on devra calculer l'accélération et le jerk à partir de la distance. Ce calcul reste moins fiable qu'une mesure directe, et sera réalisé par l'application des formules suivantes :

$$Distance : d = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} = \sqrt{\Delta x^2 + \Delta y^2}$$

$$vitesse\ v : v = \frac{\Delta d}{\Delta t}$$

$$accélération\ a : a = \frac{\Delta v}{\Delta t}$$

$$jerk\ j : j = \frac{\Delta a}{\Delta t}$$

Les premières données à comparer sont les données issues de MOCoPo puis celle de NGSIM, pNEUMA, GPS, LIDAR et enfin drone. Le chapitre se conclura sur une discussion de ces résultats.

2. Résultat sur les données MOCoPo

Les données MOCoPo ont déjà été traitées par (Buisson et al., 2016) mais avec le code écrit en Matlab. Comme le changement de langage de programmation n'a pas été totalement neutre et que certaines parties ont été modifiées pour rendre le code plus universel, il n'est pas inutile de vérifier que les résultats sont toujours corrects. L'application de l'algorithme, sur les données qui avaient été sélectionnées, montre un premier résultat différent. En effet, certaines des trajectoires possèdent des « trous » temporels. Les données MOCoPo ne sont pas spécialement régulières d'un point de vue temporel, ce qui n'est pas l'idéal pour l'application de spline, mais sur une partie des trajectoires, ces écarts sont trop importants. Cela crée des accélérations et changement de jerk complétement irréalistes comme on peut le voir sur la figure 36 suivante :

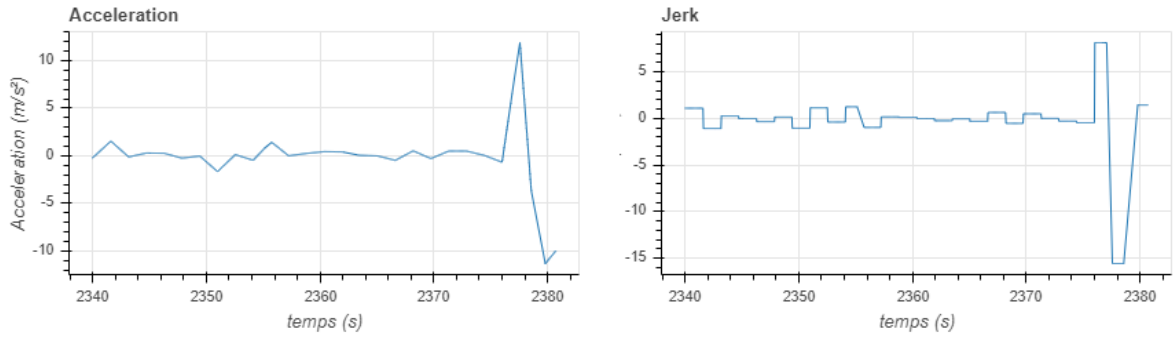

Figure 36 - Conséquence des erreurs dues à des sauts temporels.

Les trajectoires présentant des trous supérieurs à 1 seconde sont écartées, ce qui réduit le nombre de trajectoires à 453.On peut maintenant construire les indicateurs présentés en introduction.

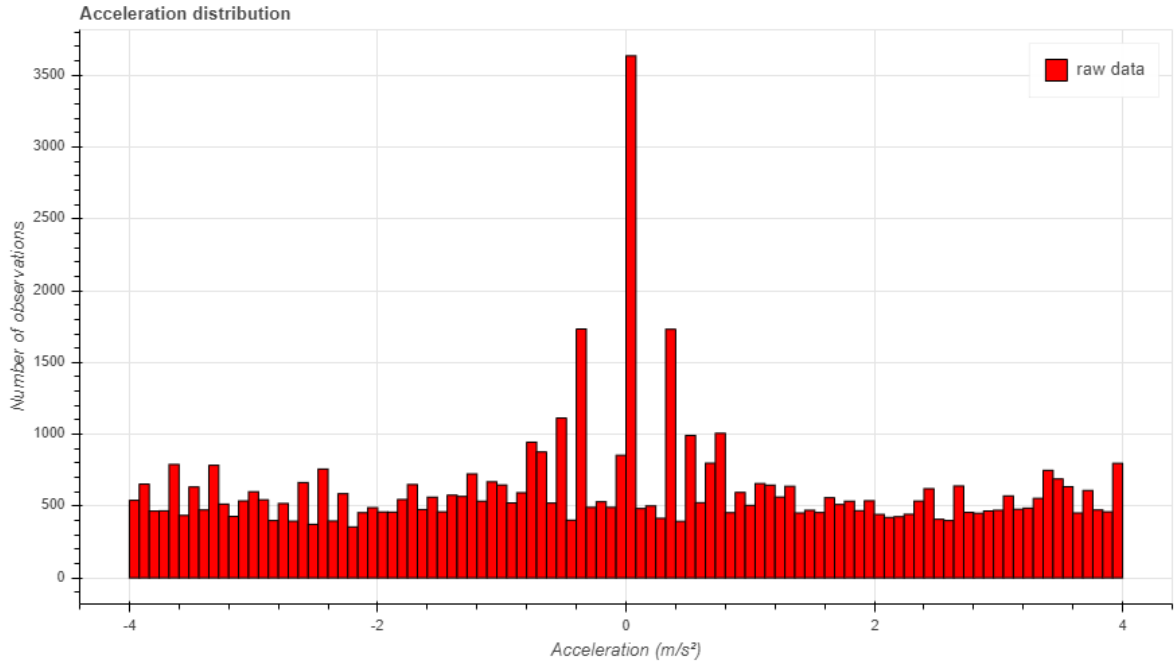

Figure 37 - Distribution des accélérations - Données MOCoPo originales.

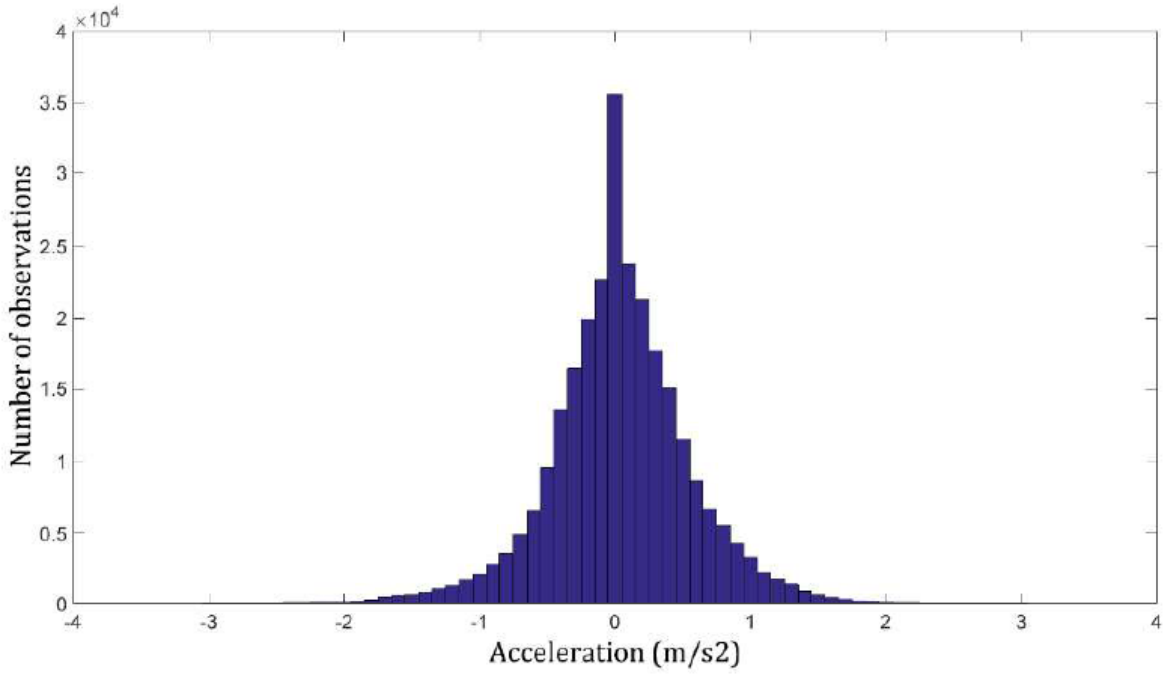

Figure 38 - Distribution des accélérations - Données MOCoPo de (Buisson et al., 2016).

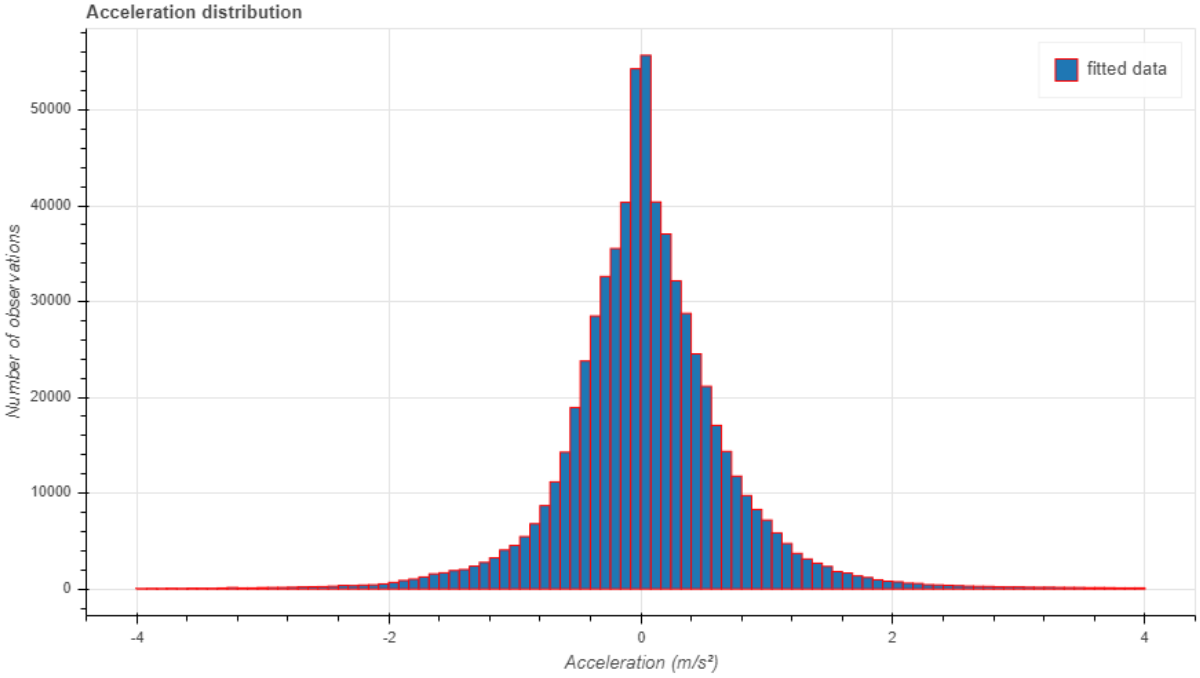

Figure 39 - Distribution des accélérations - Données MOCoPo lissées par l'algorithme.

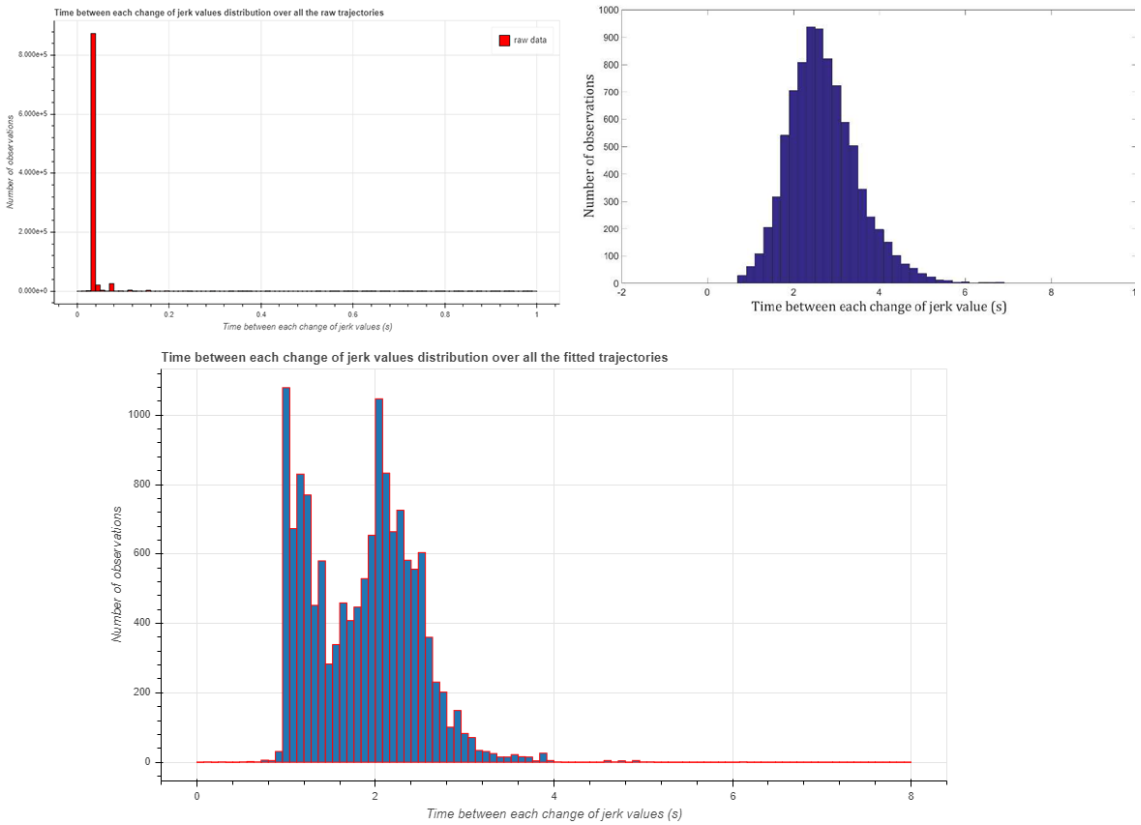

Figure 40 - Comparaison des distributions de temps entre les changements de valeur du jerk - Données MOCoPo. Figure en haut à droite extraite de (Buisson et al., 2016).

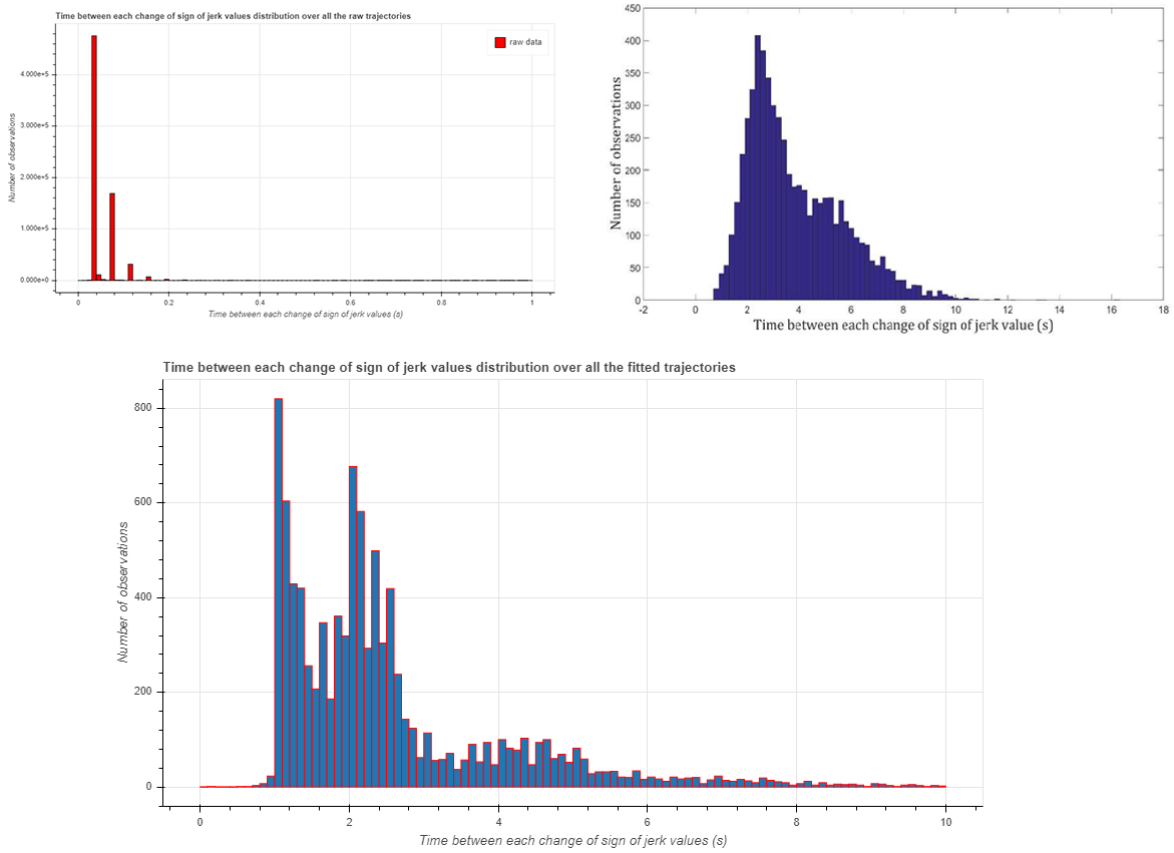

Figure 41 - Comparaison des distributions de temps entre les changements de signe du jerk - Données MOCoPo. Figure en haut à droite extraite de (Buisson et al., 2016).

Les distributions sont différentes entre celles de cette étude et celles de l'article original (figures 37, 38, 39, 40 et 41). Ce n'est pas surprenant, car le fonctionnement de l'algorithme a été assez profondément modifié. Néanmoins, ces résultats restent bons, ce qui tend à prouver que la méthode de filtrage polaire est assez robuste. Quand on compare les résultats de l'algorithme avec les données originales, il y a bien un lissage efficace qui améliore la qualité des trajectoires. Au niveau des distributions du jerk, on retrouve des pics après 2 secondes dans les résultats des deux versions de l'algorithme. Mais le changement dans la création des nœuds entraîne également l'apparition d'un pic à proximité d'une seconde. À la différence de l'algorithme en Matlab, la nouvelle version augmente le nombre des nœuds ce qui tend à créer de nouveaux pics. Néanmoins, l'ajout de ces nœuds est obligatoire pour travailler avec l'ensemble des sources de données.

Si on regarde une unique trajectoire pour voir l'effet de l'application de l'algorithme, on obtient de bons résultats comme le suivant illustré sur la figure 42 :

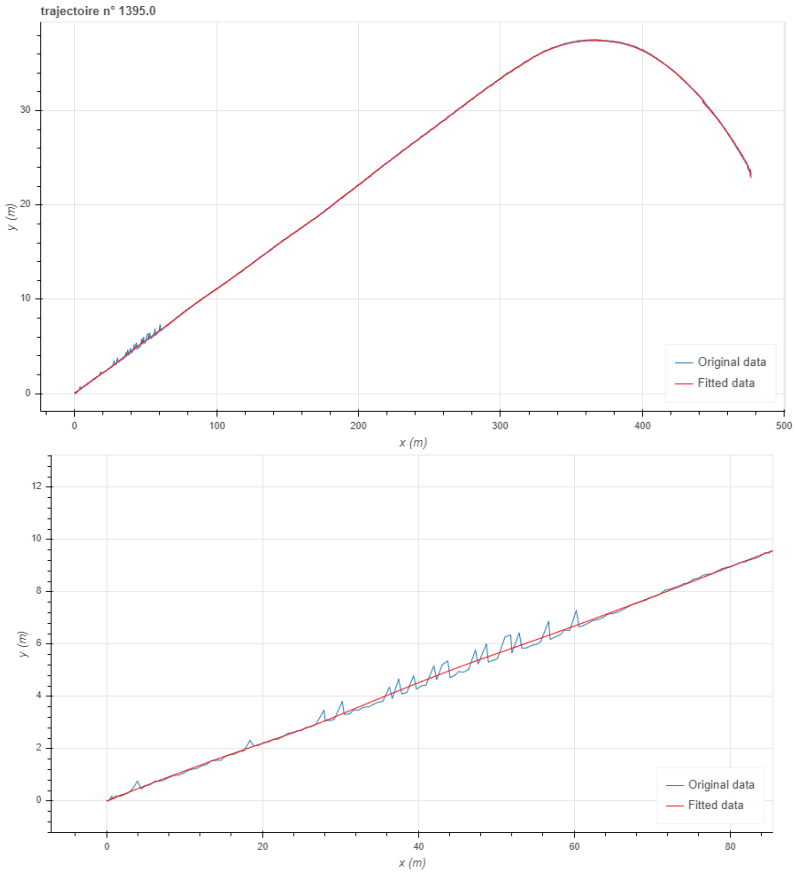

Figure 42 - Illustration du lissage effectué sur une trajectoire issue des données MOCoPo.

La courbe lissée passe bien au milieu des points de données. Finalement, même si les résultats obtenus ne sont pas identiques à ceux de (Buisson et al., 2016), ils restent acceptables. Avec cette validation, il faut maintenant traiter des données qui n'ont pas encore été lissées par cette méthode pour voir si les résultats seront aussi bons.

3. Résultat sur les données NGSIM

Les données NGSIM ont été étudiées par de nombreuses études, mais on va limiter la comparaison aux données originales et aux résultats de (Montanino and Punzo, 2015). Premièrement, on va s'intéresser à la distribution des accélérations. La distribution d'origine est vraiment médiocre, avec des pics à $\pm$ 3,7 m/s² comme illustré sur la figure 43 suivante :

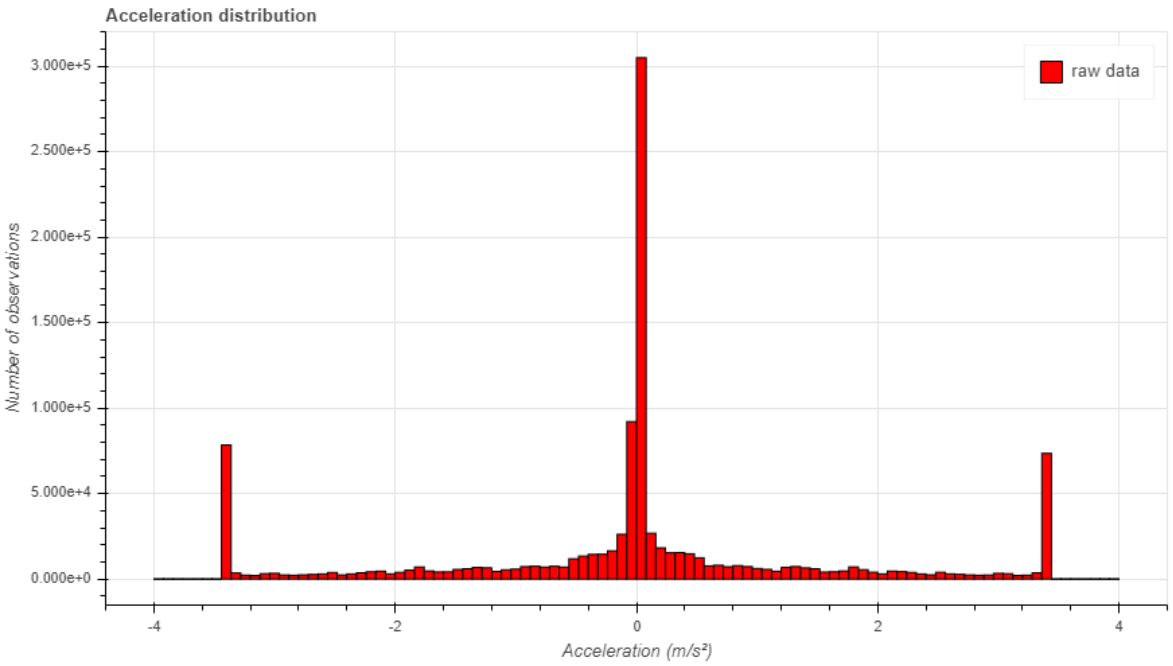

Figure 43 - Distribution des valeurs de l'accélération - Données NGSIM brutes

Ces pics sont dus à des contraintes imposées lors de la reconnaissance des trajectoires pour limiter le nombre de valeurs aberrantes. Le travail de Montanino et Punto avait permis d'obtenir une distribution beaucoup plus cohérente. Leur résultat est représenté sur la figure 44 suivante :

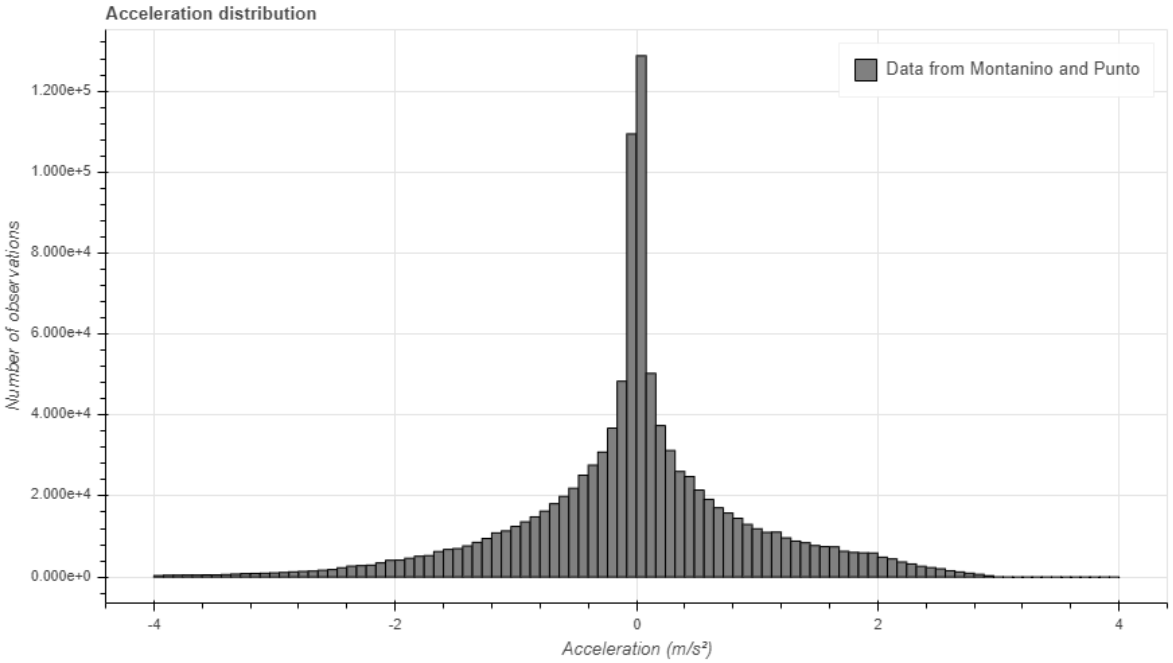

Figure 44 - Distribution des valeurs de l'accélération - Données de (Montanino et Punto, 2015)

Etant donné que leur méthode repose sur la suppression des données physiquement impossibles, il est logique que leur distribution des accélérations soit conforme avec ce

que l'on en sait. Comme sur les données originales, la distribution présente une asymétrie assez marquée. Sans être choquante, elle est malgré tout questionnable, car il est difficile de trouver une raison logique à ce phénomène.

L'application de la méthode de filtrage polaire permet également d'obtenir une distribution moins erronée.

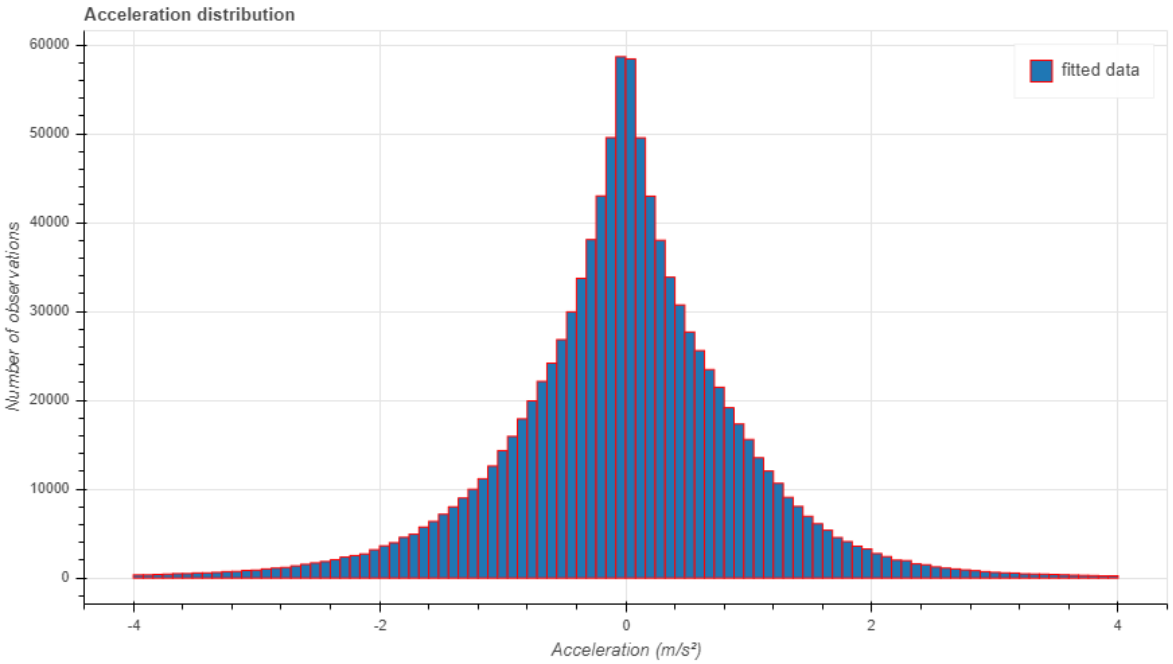

Figure 45 - Distribution des valeurs de l'accélération - Données NGSIM lissées

Au contraire du traitement de Montanino et Punto, la distribution obtenue est plus progressive et symétrique. Cette distribution est à première vue plus mauvaise que la précédente. Mais c'est en réalité au niveau de l'intensité du traitement qu'il faut aller chercher l'explication. Les données de la figure 45, sont brutes de sortie d'algorithme. Contrairement aux données italiennes, elles n'ont pas ensuite été vérifiées et redressées manuellement au besoin, quitte à revisionner les images. C'est aussi en grande partie ce post-traitement qui est à l'origine de leurs très bons résultats. Mais dans notre cas, il s'agit juste du résultat de la méthode sur des données de qualité assez mauvaises. Le résultat ne peut qu'être inférieur à celui de Montanino et Punto. En partant uniquement des données et sans aller les corriger avec les images, il n'est pas possible d'obtenir d'un résultat de très grande qualité. Finalement, à la vue de la qualité des données initiales ce résultat est déjà satisfaisant.

On va maintenant s'intéresser à la distribution des changements et des changements de signe du jerk. Les données originales et celles de Montanino et Punto ne comprennent aucune donnée sur le jerk, il faudra donc les extrapoler à partir des accélérations.

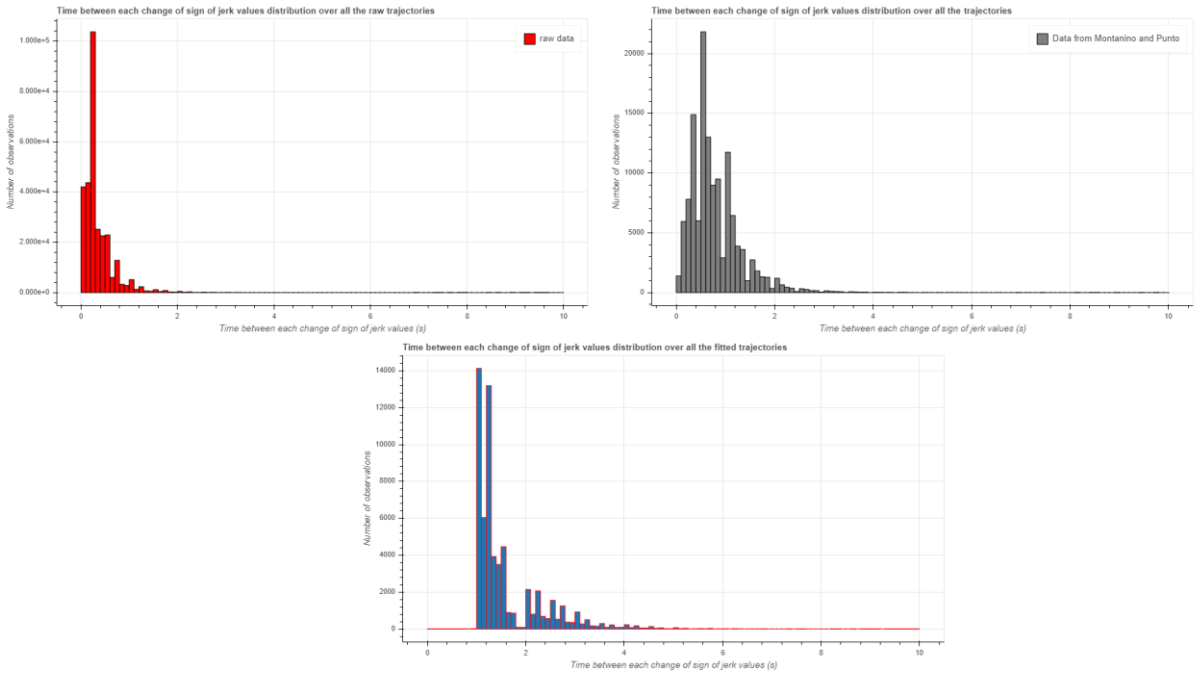

Figure 46 - Comparaison des distributions de temps entre les changements de valeur du jerk - Données NGSIM.

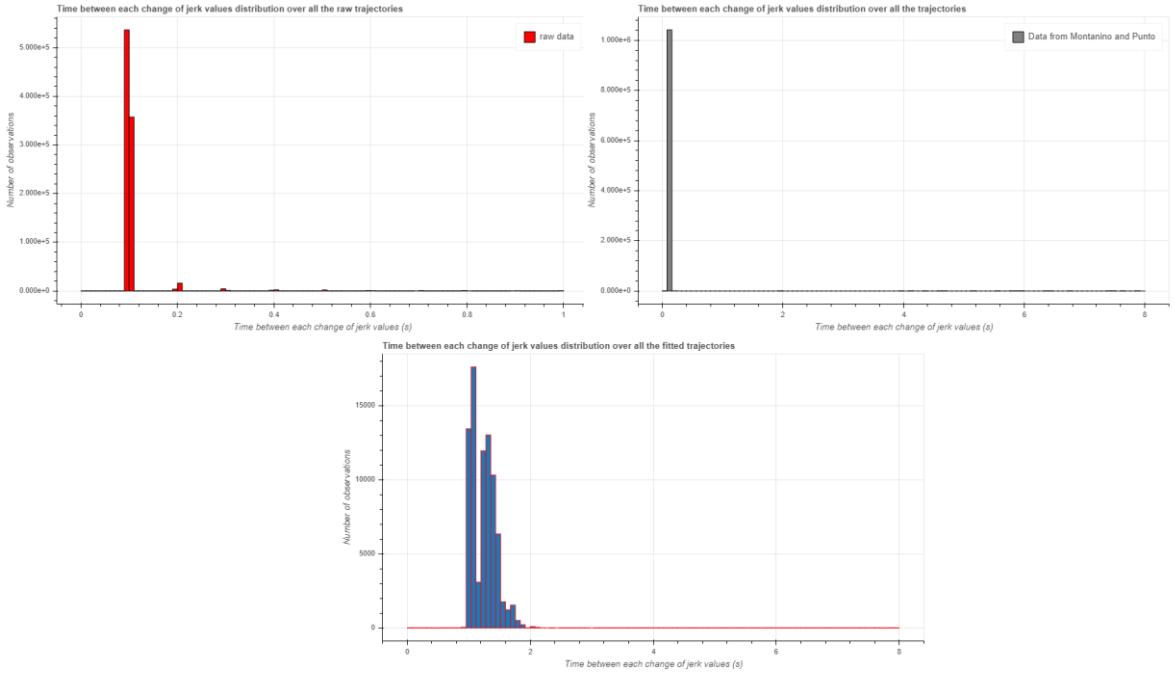

Figure 47 - Comparaison des distributions de temps entre les changements de signe du jerk - Données NGSIM.

Comme on peut le voir sur les figures 46 et 47, le travail de Montanino et Punto restant limité à la correction des valeurs impossibles de l'accélération, il ne corrige pas les distributions de jerk. On peut donc en conclure qu'il ne suffit pas de faire un redressement (même de très bonne qualité) des accélérations pour obtenir des valeurs cohérentes de jerk. Sans un lissage performant des accélérations, les changements de jerk beaucoup trop fréquents présents dans les données originales ne peuvent pas être corrigés. Finalement, si on se base sur ces indicateurs, la méthode polaire est donc supérieure à la méthode physique.

4. Résultat sur les données pNEUMA

Les données pNEUMA,(Barmpounakis and Geroliminis, 2020), ont la particularité d'être des données urbaines. Il n'est donc pas évident que les comportements observés soient les mêmes que dans les autres bases de données. Par exemple, si on trace la distribution des accélérations présentes dans les données brutes, on obtient la figure 48 suivante :

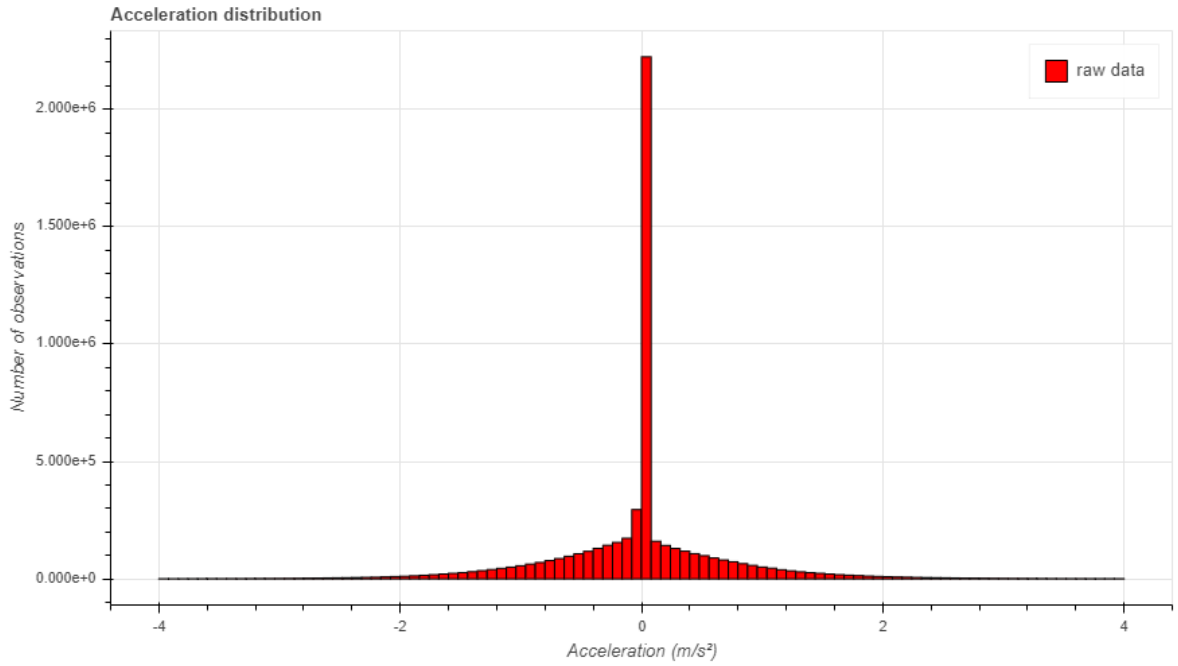

Figure 48 - Distribution des valeurs de l'accélération - Données pNEUMA brutes, nombre pair de colonne verticale.

Si on change le nombre de colonnes verticales pour obtenir un nombre impair (figure 49), on a l'explication de la forme très particulière observée précédemment : il s'agit d'un nombre significatif de zéros dus aux arrêts aux feux car il s'agit de données urbaines !

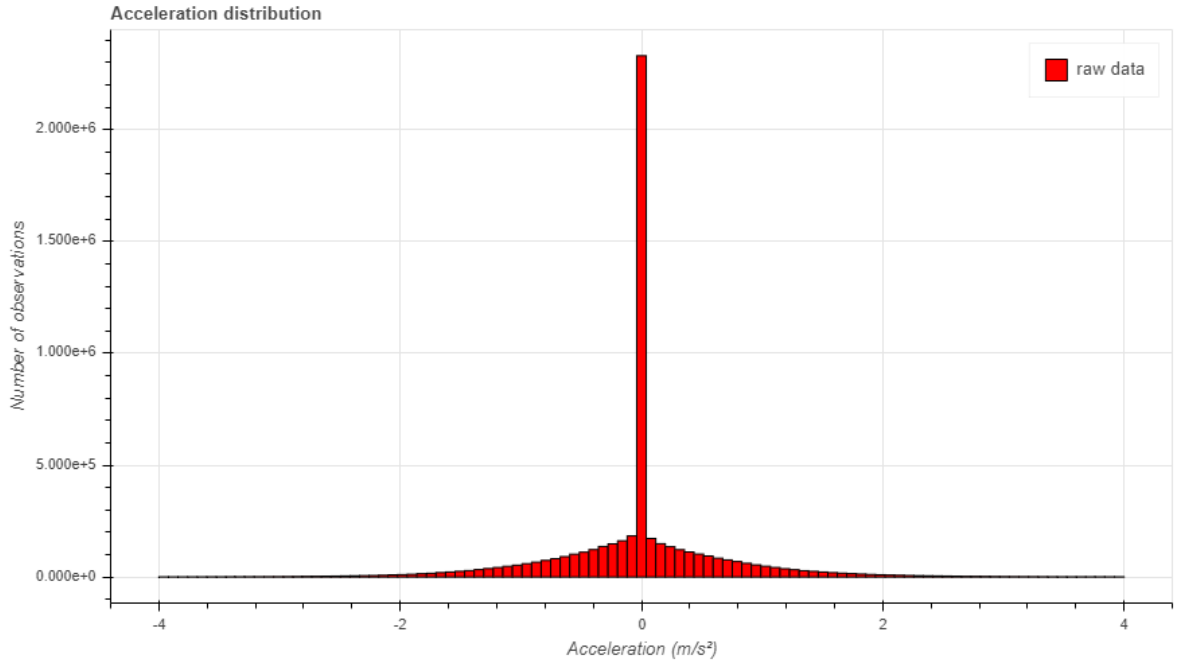

Figure 49 - Distribution des valeurs de l'accélération - Données pNEUMA brutes, nombre impair de colonne verticale.

Pour faire la comparaison, on utilise également un nombre impair de colonnes dans les résultats de l'algorithme (figure 50) :

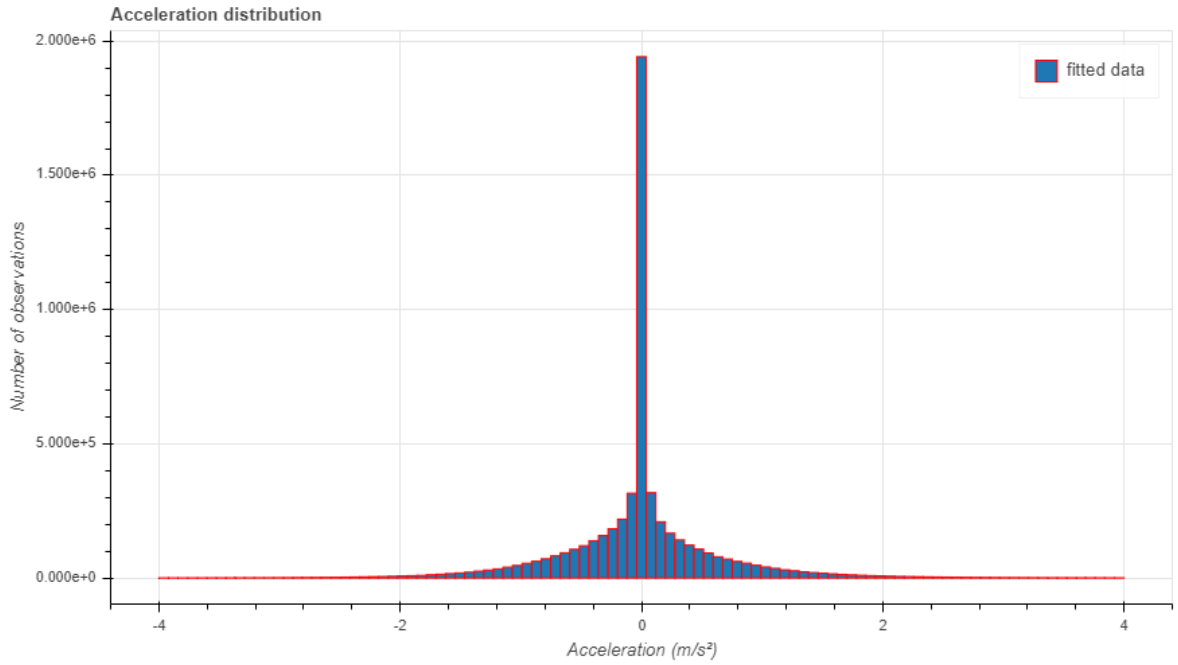

Figure 50 - Distribution des valeurs de l'accélération - Données pNEUMA lissées.

La distribution originale des accélérations est proche de celle obtenue avec le lissage polaire. Les valeurs proches de 0 sont un peu plus nombreuses dans la version lissée, mais dans les deux cas les résultats peuvent être considérés comme totalement plausibles. Il est intéressant de noter que c'est la première base de données qui comporte des accélérations brutes cohérentes. C'est aussi la plus récente et cela laisse entrevoir que les améliorations tant au niveau des cameras que des techniques de reconnaissance de véhicule vont permettre de travailler sur des données de bien meilleures qualités. Pour la distribution des jerks cependant, les données brutes sont moins bonnes comme le montre la figure 51 :

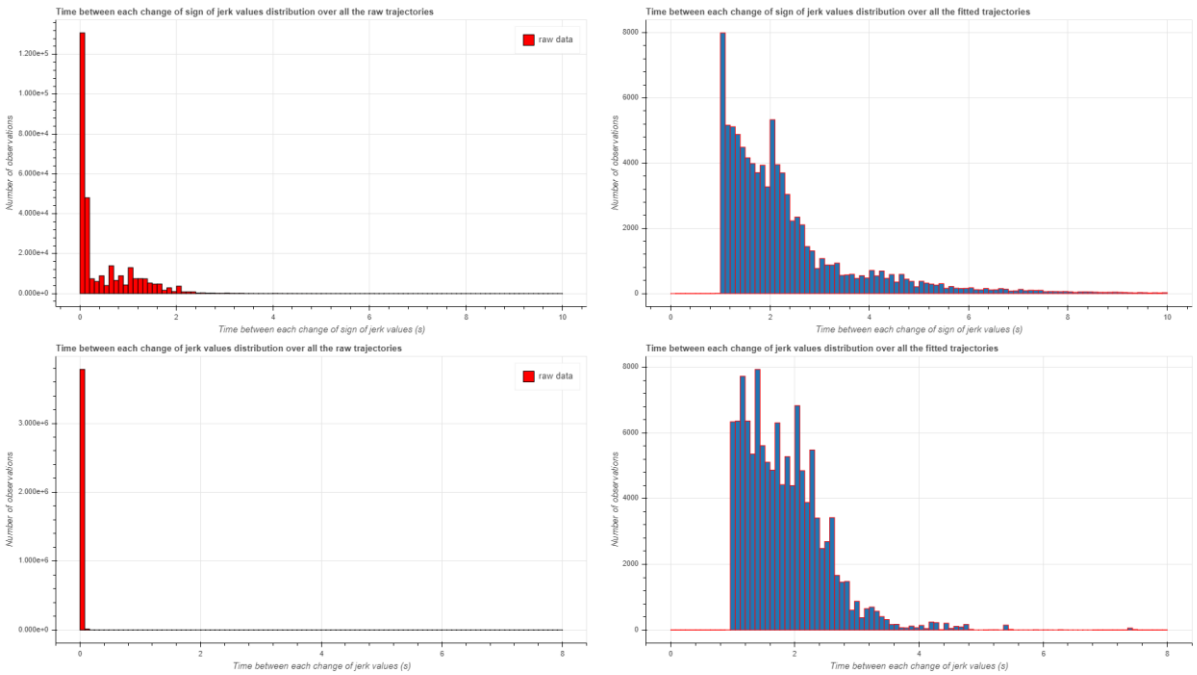

Figure 51 - Comparaison des distributions de temps avant changement de valeur et de signe du jerk – Données pNEUMA.

On retrouve le même problème que pour les données NGSIM traitées par Montanino et Punto. Ce n'est pas parce que la distribution des accélérations est cohérente que les changements de jerk se font avec des temps réalistes. Malgré leurs qualités intrinsèques, ces données ont bien besoin d'être lissées pour pouvoir être exploitées. Le lissage fonctionne bien, mais le retour des coordonnées polaires vers les coordonnées cartésiennes n'est clairement pas de très bon (figure 52). Si ce point venait à être amélioré, alors la méthode utilisée pourrait être une très bonne solution pour le filtrage de ces données. Car la base de données exploitées n'est en réalité qu'un petit échantillon des données totales qui sont toujours en cours de traitement par les auteurs du papier.

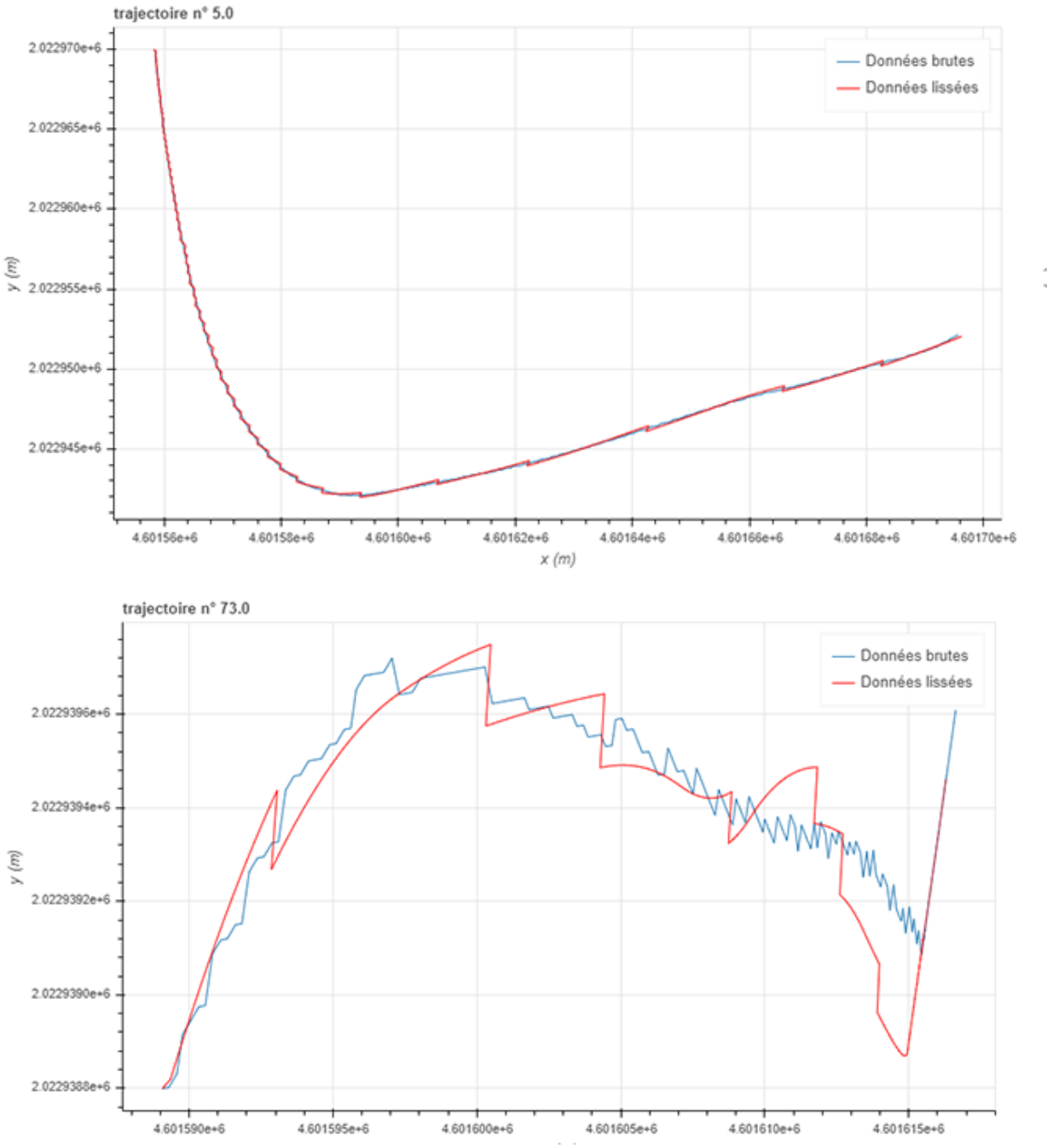

Figure 52 - Illustrations des défauts dans le retour à la trajectoire avec la méthode actuelle.

5. Résultat sur les données GPS

Les données GPS ne concernent finalement qu'un seul véhicule, mais qui a effectué 4 parcours. Ces parcours étant séparés par des parties non enregistrées (demi-tour, entrée-sortie d'autoroute), on a pu créer virtuellement 4 véhicules, un pour chaque parcours. Les données contiennent déjà des enregistrements des accélérations dont la distribution est représentée sur la figure 53 :

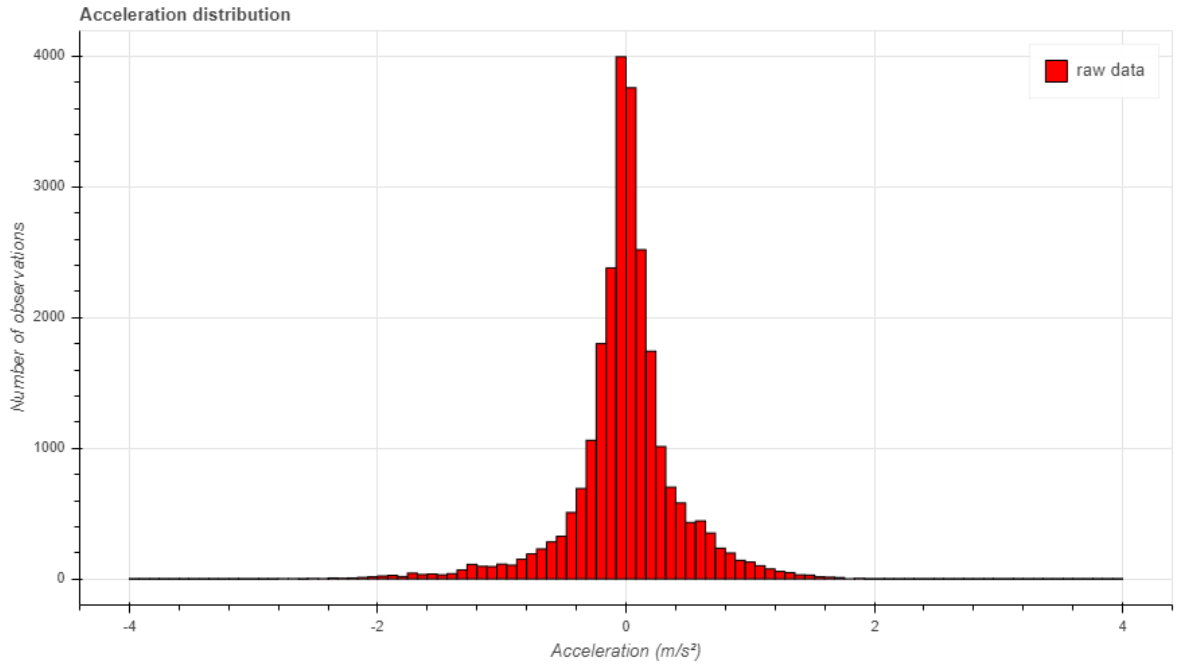

Figure 53 - Distribution des valeurs de l'accélération - Données GPS « brutes ».

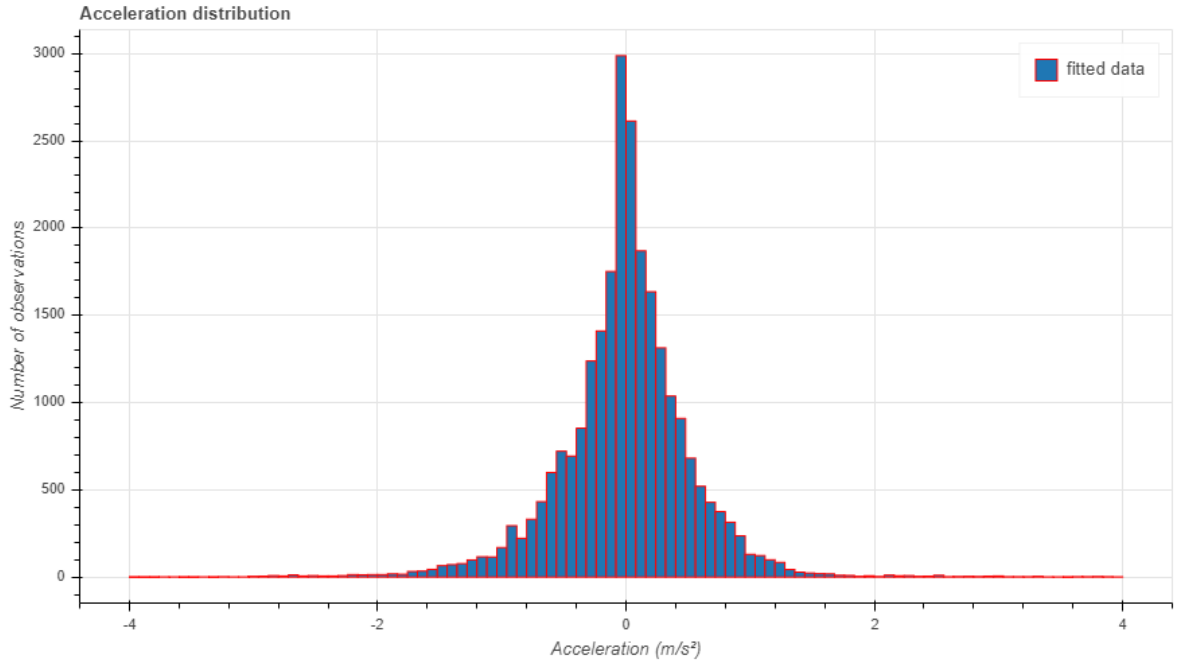

Figure 54- Distribution des valeurs de l'accélération - Données GPS lissées.

La distribution originale est sans conteste meilleure que la distribution lissée. Malgré de nombreux et infructueux essais en faisant varier les différents paramètres, il n'a pas été possible d'obtenir une distribution d'une qualité équivalente. L'article associé aux données, (Coifman et al., 2016), étant relativement évasif sur la manière d'obtenir ces données, on a donc contacté ces auteurs. D'après Coifman, ces données auraient été traitées de manière intensive pour obtenir un résultat aussi bon. Il nous a fait parvenir

les données brutes utilisées pour obtenir ce résultat. Ces données ne comprennent pas de valeur d'accélération, mais uniquement des vitesses. À partir de ces vitesses, on obtient une distribution des accélérations bien différente (figure 55) :

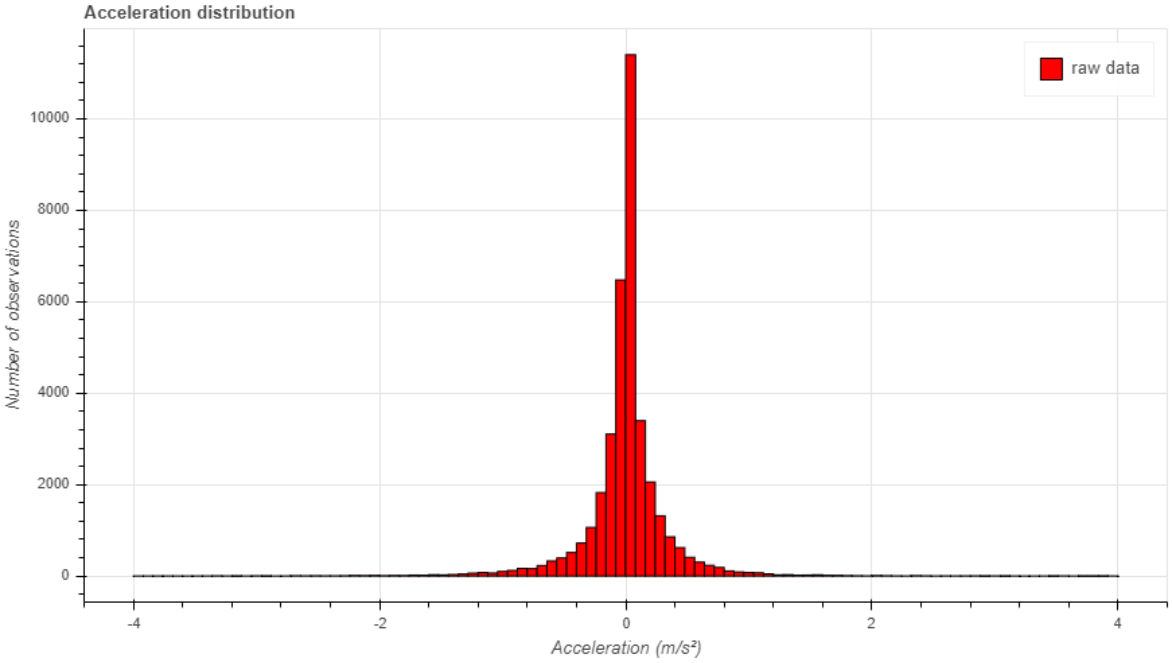

Figure 55- Distribution des valeurs de l'accélération - Données GPS brutes (fournies par Coifman).

Comme pour les données de Montanino et Punto, l'algorithme issu de la méthode polaire ne peut pas rivaliser avec des traitements intenses appliqués sur l'accélération. Mais une nouvelle fois, quand on s'intéresse au jerk, ces traitements se révèlent insuffisants pour corriger également les changements trop rapides (figure 56).

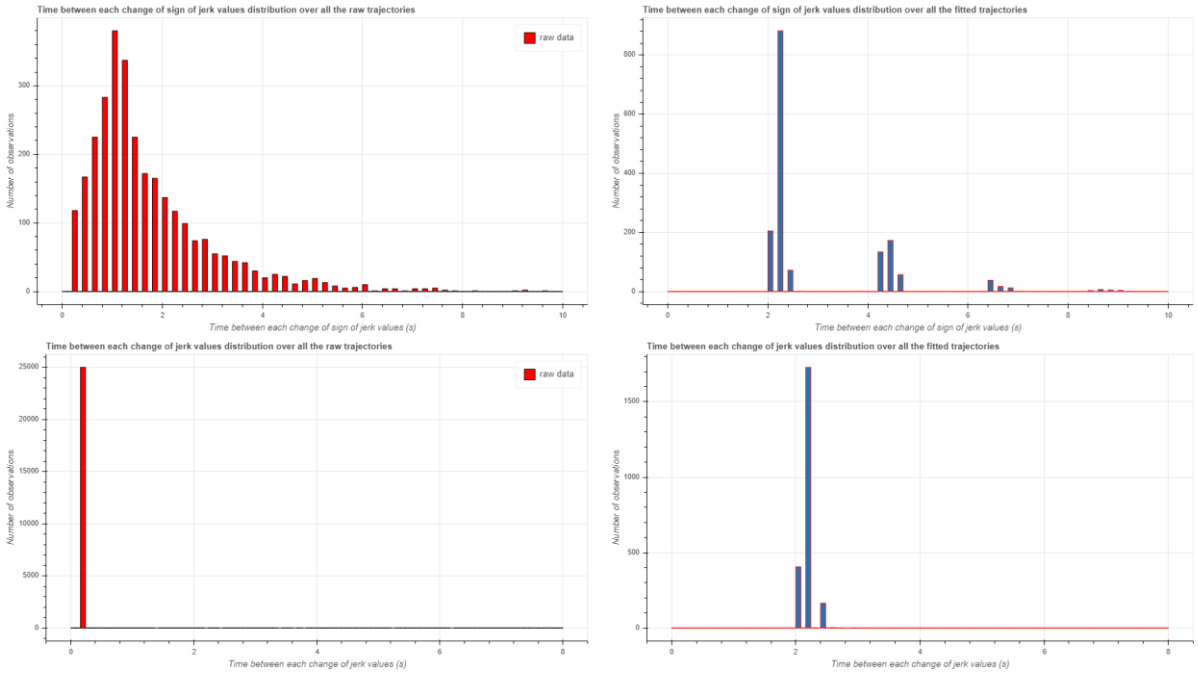

Figure 56 - Comparaison des distributions de temps avant changement de valeur et de signe du jerk – données GPS.

L’analyse des trajectoires montre que les résultats (principalement du retour en coordonnées cartésiennes) ne sont pas satisfaisants (figure 57). La cause identifiée vient de la longueur des trajectoires étudiées. Alors que dans les sources de données précédentes les trajectoires ne dépassaient que rarement 500 m de long, ici les trajectoires font un peu plus de 20 km ! Sur des trajectoires aussi longues, il est probable qu’il y ait des courbures plus ou moins importantes qui viennent remettre en question l’hypothèse de relative séparation entre r et θ.

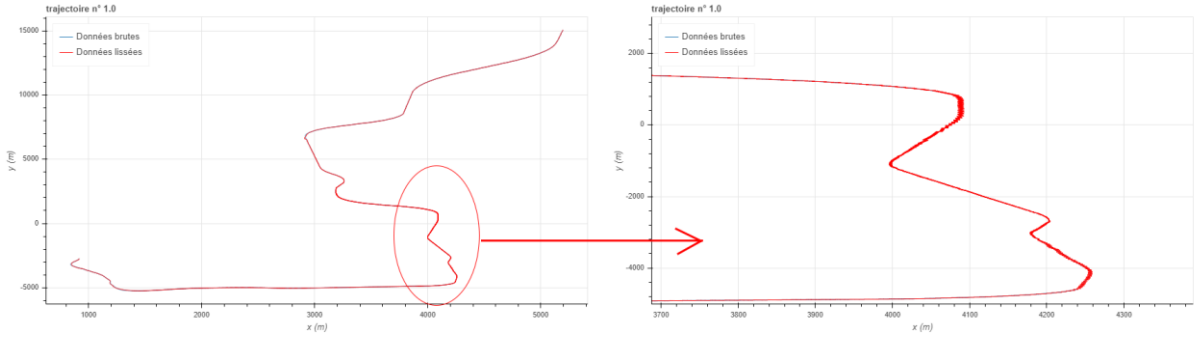

Figure 57 - Illustration des défauts de retour à la trajectoire avec les données GPS. Les zigzags sont dus à la méthode utilisée pour repasser aux coordonnées cartésiennes.

6. Résultat sur les données LIDAR

Alors que le principal problème du traitement des données GPS était lié à la grande distance des parcours, les données LIDAR présentent le profil inverse. Il s'agit généralement de portions de trajectoire, détectées sur des temps très courts. En ne prenant que les trajectoires qui sont supérieures à 7 secondes, on réduit la base à 381 trajectoires. Et sur ces 381 trajectoires, une partie présente de gros « trous » temporels, c'est-à-dire d'une taille supérieure à la seconde, ce qui vient encore réduire la base exploitable à 272 trajectoires. C'est d'ailleurs le principal défaut de ces données, qui plus encore que les données vidéos souffrent d'un grand manque de continuité dans la détection (figure 58).

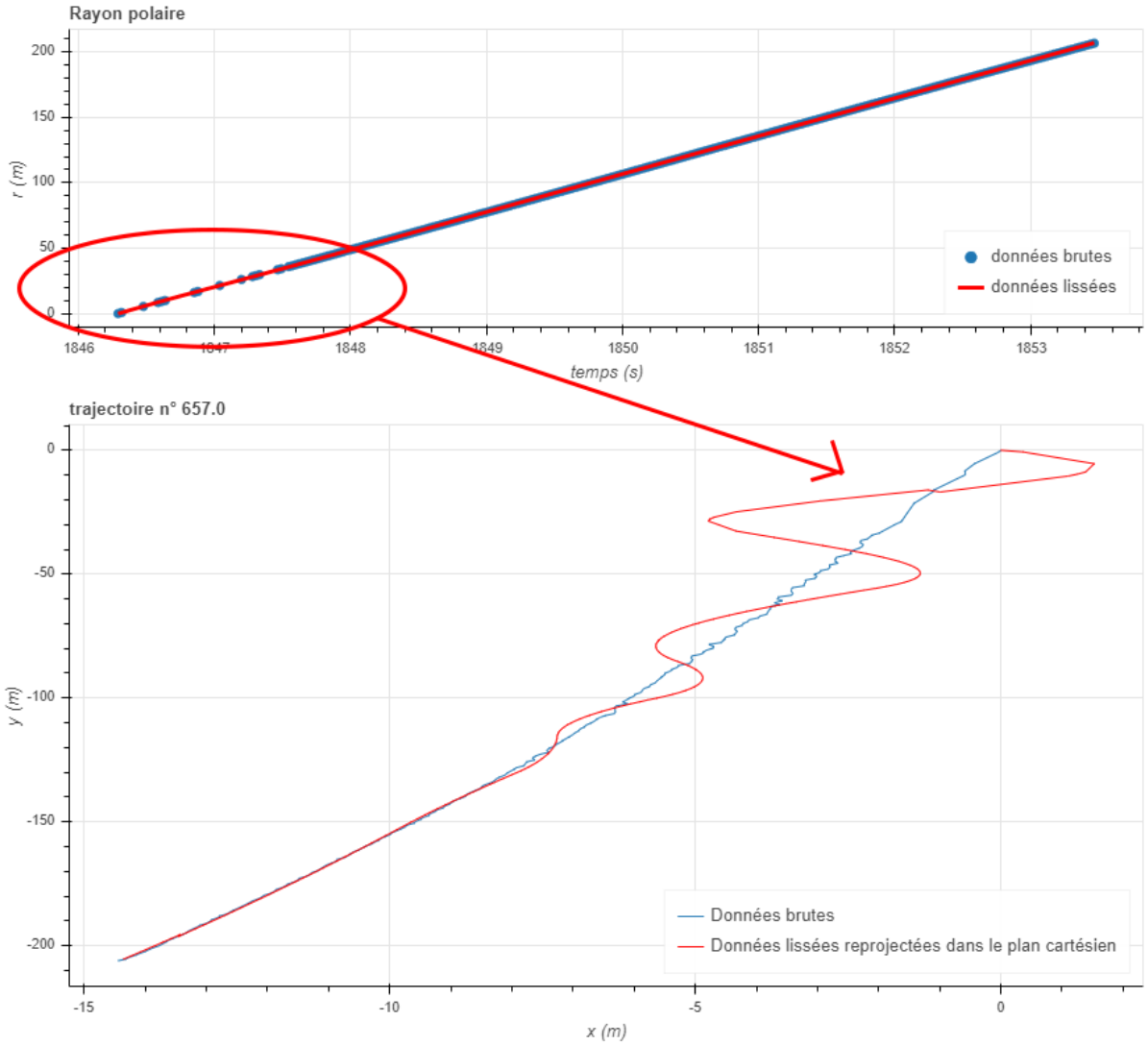

Figure 58 - Illustration des trous temporels présents dans les données LIDAR.

Néanmoins, on arrive à produire les indicateurs habituels pour une analyse. La distribution des accélérations fournies dans les données originales est plutôt bonne et conforme avec ce que l'on attend (figure 59). À l'inverse, les résultats de l'algorithme sont franchement moyens, et l'absence de continuité temporelle dans les données a clairement diminué les performances de lissage des splines (figure 60).

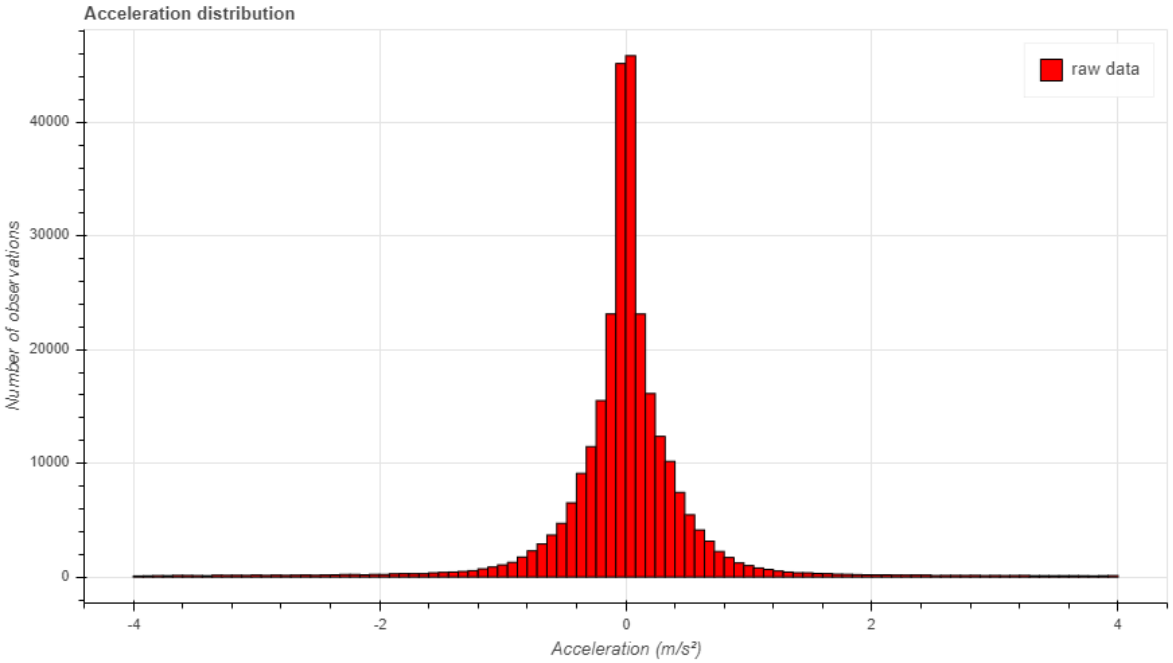

Figure 59 - Distribution des valeurs de l'accélération - Données LIDAR brutes.

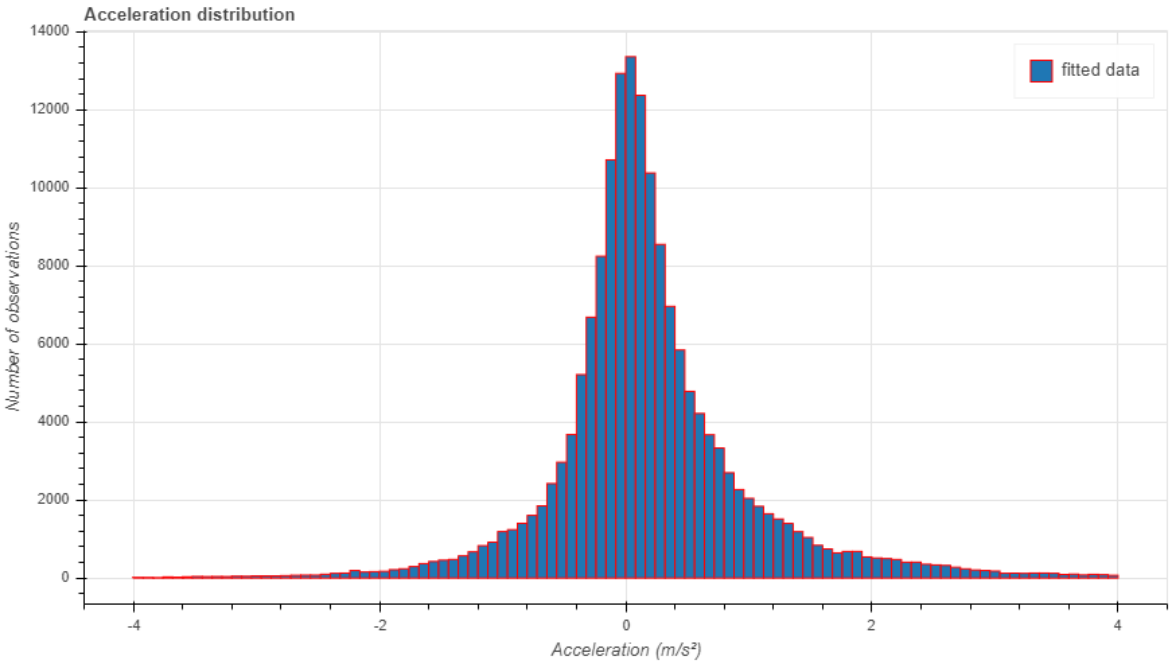

Figure 60 - Distribution des valeurs de l'accélération - Données LIDAR brutes.

En revanche, le lissage a clairement amélioré les distributions du jerk, mais il reste des points avec des valeurs inférieures à 1 seconde (figure 61). De cela, on peut déduire que malgré les vertus universalistes que l'on a essayé d'implémenter dans l'algorithme, la qualité du jeu de données initial reste primordiale. Et la présence de trop grandes plages de non-détection est la principale qualité que doit avoir un jeu de donnés pour pouvoir être exploitable.

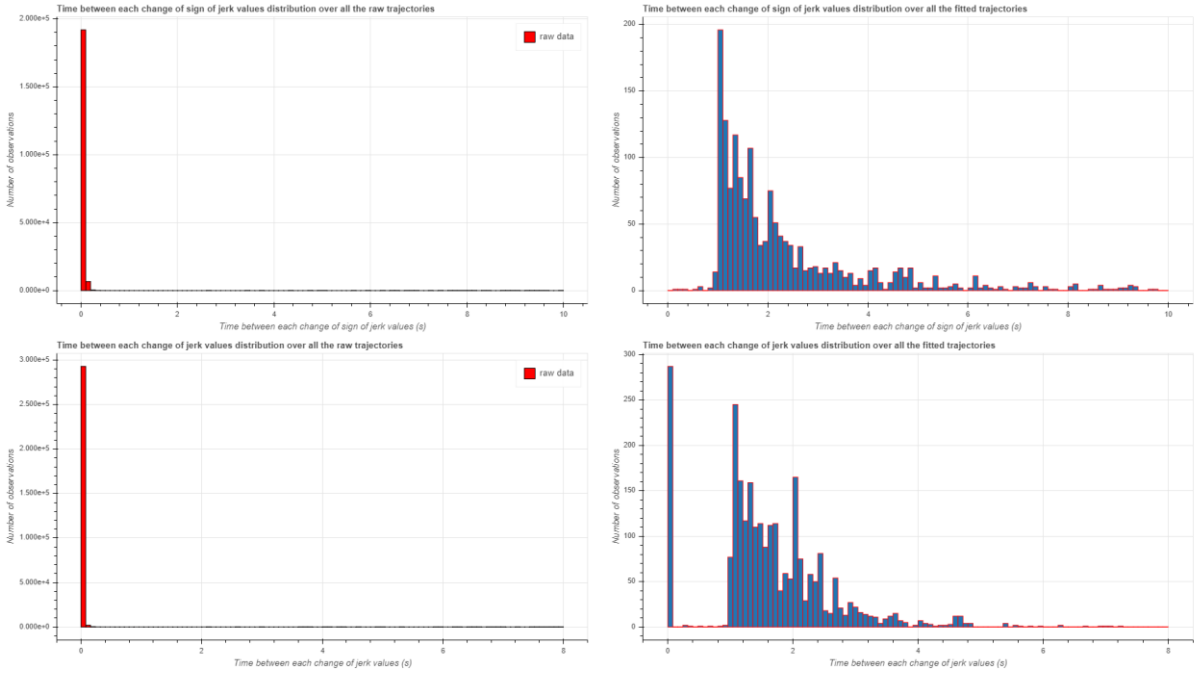

Figure 61 - Comparaison des distributions de temps avant changement de valeur et de signe du jerk – Données LIDAR.

7. Résultat sur les données Drones

Etant donné que l'on ne dispose que de morceaux de trajectoires (traklet), on ne va pas faire une analyse de distribution, mais juste présenter une trajectoire pour illustrer les premiers résultats. Ces résultats sont prometteurs et la méthode devrait permettre de les traiter pour obtenir des résultats de qualité. Si on prend l'une des tracklets les plus longues, on obtient le résultat de la figure 62 suivante :

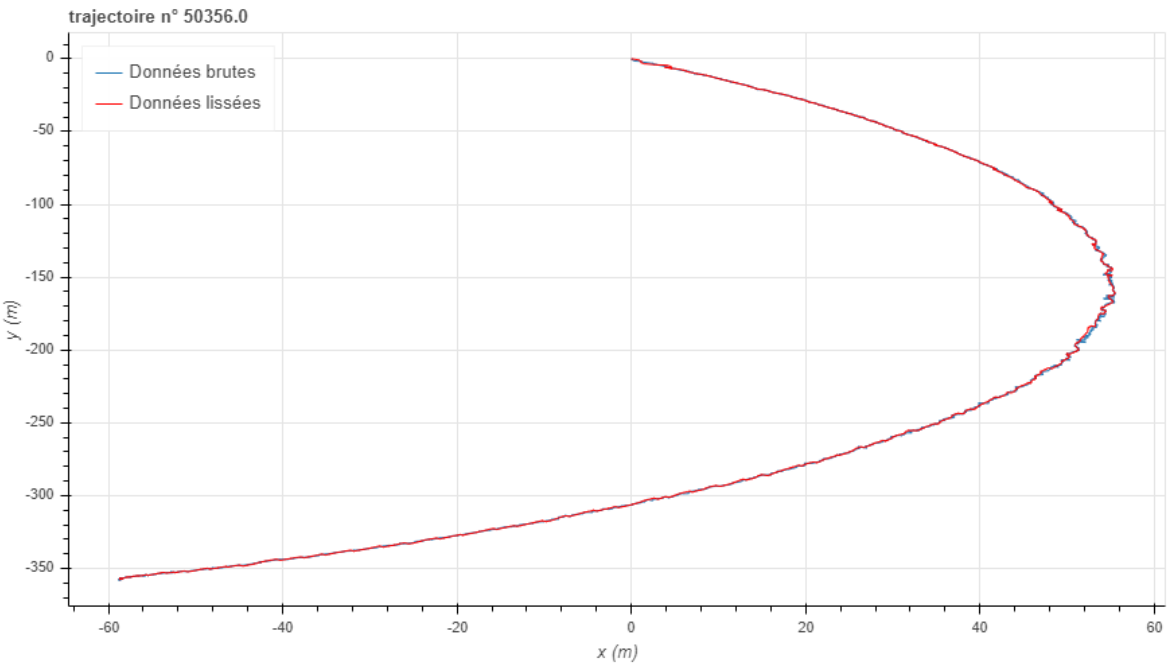

Figure 62 – Illustration d'une des premières trajectoires extraites des données de l'expérimentation avec le drone.

8. Discussion des résultats

L'une des questions que posait ce travail était de savoir si l'algorithme est applicable à toutes sources de données de trajectoires. Au vu des résultats, on serait tenté de répondre plutôt oui, mais néanmoins avec quelques réserves :

- Premièrement, les données doivent être le plus régulières possible d'un point de vue temporel. Le fait d'utiliser des splines pour le lissage impose une distribution autant harmonieuse que dense de données. Les méthodes de lissage par interpolation sont généralement assez peu adaptées à ces problèmes et dans le cas de données très irrégulières, un traitement par des filtres donnera sans doute de meilleurs résultats.

- Deuxièmement, la méthode a été pensée pour traiter des trajectoires de courte étendue tant spatiale que temporelle, provenant d'autoroutes. La particularité des autoroutes vient de leur rayon de courbure qui est très faible. Dans ces conditions, l'hypothèse que $r$est proche de la distance parcourue et θ proche du rayon de courbure et reste relativement faible, est bien vérifiée. Quand on prend des données comme celle de pNEUMA ou GPS, cette hypothèse n'est plus vérifiée. Le résultat de l'algorithme devient clairement moins bon, sans être complétement faux, mais surtout le retour en coordonnées cartésiennes devient très compliqué.

- Enfin, c'est bien ce retour en coordonnées cartésiennes qui est problématique. Le lissage de la distance calculée permet d'obtenir des valeurs réalistes de vitesse, d'accélération et de jerk, mais il est délicat d'estimer dans quelle mesure on s'est éloigné de la trajectoire originale. Tant que l'on est sûr que l'hypothèse sur θ est vérifiée, les approximations faites dans le retour aux coordonnées cartésiennes ne sont pas très gênantes. Mais quand on sort de cette zone, la résolution n'est plus satisfaisante et on se retrouve avec un apport de bruit plutôt qu'un lissage, ce qui est paradoxal.

L'intégralité de ce travail aurait aussi pu être réalisée avec une autre méthode de traitement des données. La méthode de Montanino et Punzo aurait également pu être appliquée. Néanmoins, cette dernière est beaucoup plus consommatrice de temps et nécessite d'avoir les images vidéo pour recaler les trajectoires des véhicules au comportement trop étrange. Cependant, elle est appliquée directement sur les coordonnées cartésiennes ce qui évite la problématique de retour à ces coordonnées. Des méthodes complexes de filtre auraient pu être envisagées, mais cela n'aurait pas beaucoup contribué au sujet. En effet, par leur simplicité de mise en œuvre et leur utilisation dans tous les domaines pour redresser des séries de données, elles sont quasi-systématiquement employées pour faire du traitement de données. Si, par exemple, les récentes données issues de pNEUMA n'ont pas encore été traitées avec des filtres, ce sera certainement le cas dans les années à venir.

Pour conclure, la méthode utilisée dans cette étude est l'une des seules à pouvoir obtenir des données cohérentes à la fois en termes d'accélération et de jerk. Cet avantage devrait inciter à continuer le présent travail pour remédier aux derniers défauts et arriver à une méthode intéressante pour traiter des données de trajectoires de toute provenance. Enfin, la méthode est assez peu coûteuse en temps de calcul et il est envisageable que cette méthode puisse être utilisé pour faire du lissage en temps réel. Maintenant que les sources de données de trajectoires sont en train de se démocratiser, pouvoir analyser voire prédire les comportements des usagers intéressera fortement à la fois les gestionnaires d'infrastructures, mais aussi les développeurs de véhicules autonomes.

Conclusion

La méthode de lissage polaire utilisé dans ce rapport est bien une méthode qui peut être utilisée pour le filtrage de tout type de données de trajectoires de véhicules. Néanmoins, il faut quand même respecter deux conditions : avoir une bonne continuité temporelle des données et vérifier que le rayon polaire est quasi-assimilable à la distance et l'angle à la courbure de la route. Toutefois, Si ces deux conditions ne sont pas réunies, cela ne veut pas pour autant dire que la méthode n'est pas applicable.

Le fait d'utiliser des splines pour le lissage impose la condition sur le temps. Mais, en découpant les trajectoires, il est possible de supprimer les discontinuités. Cela étant, il est évident qu'il faut que les opérations que l'on veuille mener à la suite du lissage ne nécessitent pas des trajectoires complètes. Si on n'est intéressé que par une partie de la trajectoire, par exemple lors d'un changement de voie, et si sur cette portion, cette dernière ne présente pas d'écarts temporels trop importants alors la méthode peut permettre d'obtenir une trajectoire propre.

Cette technique peut également être utilisée pour réduire l'étendue spatiale des trajectoires, ce qui permet généralement de vérifier la condition sur r et θ. Le travail à faire se déplace de la méthode de filtrage vers la méthode de sélection des données. Dans l'ensemble du rapport, il y a une supposition sous-jacente qui est que l'intégralité des données fournies doit servir. En effet, plus le volume de données est grand et plus il y aura de matière à analyser. Le choix de sélectionner minutieusement les données pour ne garder que celles qui sont intéressantes aurait également été légitime. Mais il n'aurait pas démontré la grande robustesse de la méthode.

Finalement, ce n'est pas une grande surprise que d'arriver à la conclusion que la méthode polaire, dont les prérequis sont très limités, est déployable à grande échelle pour lisser des trajectoires de véhicules. Ce type de changement de repère pour limiter l'interdépendance des variables est utilisé depuis longtemps dans les problèmes de physique (problème à deux corps par exemple). Mais les trajectoires étudiées ne sont pas des objets de physique théorique, il s'agit du résultat de comportements humains. Il était donc bon de montrer que cet ajout stochastique ne perturbait pas la méthode et que ces résultats étaient cohérents avec des données venant de France, des Etats-Unis ou de Grèce.

Pour conclure, il reste des points à améliorer dans la méthode et qui n'ont pas pu être complétement résolus dans ce travail. Il s'agit particulièrement du retour en coordonnées cartésiennes qui est délicat. Il est sans doute possible de faire des ajouts supplémentaires pour garantir le bon fonctionnement de la méthode. Il existe sans doute des méthodes plus performantes pour définir les nœuds de spline qu'une simple distribution linéaire, ou des points singuliers, plus important que ceux issus de la recherche de point tournant. Ces améliorations sont d'autant plus importantes que comme on l'a vu sur les données les plus récentes, celle de pNEUMA, la qualité des images et de la détection des véhicules a beaucoup progressé. Il faut donc continuer à améliorer les processus de lissage de ces données pour obtenir des trajectoires toujours plus exploitables, mais la méthode de filtrage polaire est un bon point de départ pour cela.

Bibliographie

Barmpounakis, E., Geroliminis, N., 2020. On the new era of urban traffic monitoring with massive drone data: The pNEUMA large-scale field experiment. Transportation Research Part C: Emerging Technologies 111, 50–71. https://doi.org/10.1016/j.trc.2019.11.023

Brockfeld, E., Wagner, P., Kuehne, R., 2004. Calibration and Validation of Microscopic Traffic Flow Models 15.

Buisson, C., Villegas, D., Rivoirard, L., 2016. Using Polar Coordinates to Filter Trajectories Data without Adding Extra Physical Constraints.

Coifman, B., Li, L., 2017. A critical evaluation of the Next Generation Simulation (NGSIM) vehicle trajectory dataset. Transportation Research Part B: Methodological 105, 362–377. https://doi.org/10.1016/j.trb.2017.09.018

Coifman, B., Wu, M., Redmill, K., Thornton, D.A., 2016. Collecting ambient vehicle trajectories from an instrumented probe vehicle. Transportation Research Part C: Emerging Technologies 72, 254–271. https://doi.org/10.1016/j.trc.2016.09.001

Haight, F., 1963. The future of trafic flow theory, Traffic Quarterly, Volume 17, Issue 4

Marczak, F., Buisson, C., 2012. A new filtering method for trajectories measurement errors and its comparison with existing methods 21.

Montanino, M., Punzo, V., 2015. Trajectory data reconstruction and simulation-based validation against macroscopic traffic patterns. Transportation Research Part B: Methodological 80, 82–106. https://doi.org/10.1016/j.trb.2015.06.010

Ossen, S., Hoogendoorn, S., 2008. Validity of Trajectory-Based Calibration Approach of Car-Following Models in Presence of Measurement Errors. Transportation Research Record 2088, 117–125. https://doi.org/10.3141/2088-13

Punzo, V., Formisano, D.J., Torrieri, V., 2005. Nonstationary Kalman Filter for Estimation of Accurate and Consistent Car-Following Data. Transportation Research Record 1934, 2–12. https://doi.org/10.1177/0361198105193400101

Rivoirard, L., 2015. Préparer le déploiement de véhicules autonomes en qualifiant la variabilité des comportements de conduite réels observés en congestion 110.

Yang, Y., Yuan, Z., Fu, X., Wang, Y., Sun, D., 2019. Optimization Model of Taxi Fleet Size Based on GPS Tracking Data. Sustainability 11, 731. https://doi.org/10.3390/su11030731

Annexes

Polar Filtering Algorithm

libraries loading

In [1]:

```
import pandas as pd
import numpy as np
import datetime as dt
from scipy import integrate, interpolate
import math
import cmath
from ipywidgets import interact
from bokeh.io import push_notebook, show, output_notebook
from bokeh.plotting import figure, show ,output_notebook
from bokeh.layouts import row ,column
from bokeh.layouts import layout
```

Function definition

In [2]:

```
#Spline function avec mémoire
def Spline_function(x,y,deg,acceptable_percentage,deltaPos):
    ''' spline_function(x,s,deg,acceptable_percentage,deltaPos)
    Create a spline from x,y based on a knot linear distribution
    return spline value, spline function, Number of knots, knots position
    '''
    #setting the number of correct point to be attending
    MM=acceptable_percentage/100*len(x)+1
    #creating the memory core, A is a temporary array, all value are high
enough to be ignore if they are not fulfill
    A=np.ones((int(x.max()-x.min()),2))*acceptable_percentage/100*len(x)*
10000
    Mem=pd.DataFrame(A,columns=['Np','MM'])
    #setting condition for loop interruption
    Complete=0
    #main loop : each time we add a knots in the spline until we get a sp
line matching the condition
    for Np in range(1,int(x.max()-x.min())):
        #setting the knots
        knot_offset = (x.values[-2] - x.values[2])/(Np+1)
        knots = np.linspace(x.values[2]+knot_offset, x.values[-2]-knot_of
fset, Np)
        #LSQUnivariateSpline is not a very stable function so we catch th
e numerous error it can produce
        try:
            #Creating the spline function with the previous knots
            spline1=interpolate.LSQUnivariateSpline(x,y,knots,bbox=[x.min
()-1e-9,x.max()+1e-9],k=deg)
```

```
            #Calculating distance, equivalent to L2 norm, between the new
y and the old one
            #but without the tenth first and last points
            MM=np.abs(spline1(x)[10:-10]-y[10:-10])
            #Getting the number of point where distance is bigger than de
ltaPos/2 because we use absolute distance
            MM=len(MM[MM>deltaPos/2])
            #we stock the value of knots'number and the number of wrong p
oint
            Mem.Np[Np]=Np
            Mem.MM[Np]=MM
        #ValueError is raise when the knots are not well defined in this
case we make the memory value with a high value
        except ValueError:
            MM=acceptable_percentage/100*len(x)*10000
        #if we match the condition set in the call of the function we sto
p the loop for calculation gain
        if MM<acceptable_percentage/100*len(x):
            #set Complete to 1 and breaking the loop
            Complete=1
            break
    #if the condition are met before the end of the loop we use the last
spline created
    #instead of finding the best in the memory core
    if Complete==1 :
        s_prime=spline1(x)
        knots = spline1.get_knots()
    #if the loop ended, we search the best attempt in the memory core. Du
e to the non-linear comportement of the spline creation
    #the last isn't always the best try.
    else :
        #first we must assure that there is only one best attempt
        if len(Mem.Np.values[Mem.MM==Mem.MM.min()])==1:
            Np=int(Mem.Np.values[Mem.MM==Mem.MM.min()])
        #if there is more than one best attempt we take the one with the
lesser number of knots
        else :
            Np_array=Mem.Np.values[Mem.MM==Mem.MM.min()]
            Np=int(Np_array.min())
        #we recreate the spline from memory core informations
        knot_offset = (x.values[-2] - x.values[2])/(Np+1)
        knots = np.linspace(x.values[2]+knot_offset, x.values[-2]-knot_of
fset, Np)
        spline1=interpolate.LSQUnivariateSpline(x,s,knots,bbox=[x.min()-1
e-9,x.max()+1e-9],k=deg)
        s_prime=spline1(x)
        knots = spline1.get_knots()
        #returning the calculated value, spline function,knots number and
position (mainly for control purpose)
    return(s_prime,spline1,Np,knots)
```

In [3]:

```
def TPSpline(x,y,k,slope_tolerance,max_knots_by_time):
    #First we calculate y[i]-y[i+1] for all y
    a=np.concatenate((np.diff(y),[np.diff(y)[-1]]))
    #Creating a temporay dataframe to record variation
    df=pd.DataFrame(np.array([x,y,a]).T,columns=['x','y','a'])
    #Duplicate the difference column a with shifting index by 1
    df['a1']=df.a.shift(1)
    #first row create by shift(1) is NaN so we put it at 0
    df.loc[0,'a1']=0
    #we tolerate a small variation
    df['tolerance']=df.a1*slope_tolerance
    #Creating a new column 'state' for recording the state of the functio
n (Constant,Increasing,Decreasing)
    df['state']='CONSTANT'
    #we determine current state for each row of df based on 'a'
    df.loc[df.a<(df.a1-df.tolerance),'state']='INCREASING'
    df.loc[df.a>(df.a1+df.tolerance),'state']='DECREASING'
    #Constant state have no interest for finding turning point so we remo
ve then
    df=df[df.state!='CONSTANT']
    #Some data can be only constant, in that case x vector is used for kn
ots
    if len(df)==0:
        knots=x[2:-2]
    else:
        #Once again we shifting state for futher comparison
        df['state1']=df.state.shift(1)
        #first row is still NaN so we put it at 'CONSTANT'
        df.loc[df.index[0],'state1']='CONSTANT'
        #Creating a new column for determinig turning point (TP)
        df['TP']=0
        #If state change between two row we get a turning point
        df.loc[df.state!=df.state1,'TP']=1
        #we create a temporary turning point df
        tpt=df[df.TP==1]
        #we calculate in new column 'b' the difference in x for each row
        #for controling respect of minimal interval between knot
        tpt=tpt.reset_index(drop=True)
        tpt['b']=np.concatenate((np.diff(tpt.x),[np.diff(tpt.x)[-1]]))
        #creating control vector of node without minimal distance
        c=tpt.b[tpt.b<max_knots_by_time]
        #while vector c exist we suppress tpt.b min value than recaclculi
ng distance between row
        while len(c)>0 :
            tpt=tpt[tpt.b>tpt.b.min()]
            tpt['b']=np.concatenate((np.diff(tpt.x),[np.diff(tpt.x)[-1]])
)
            c=tpt.b[tpt.b<max_knots_by_time]
```

```
        #When c is empty, all the remaining point of tpt can be used as k
not
        #but for avoiding problems (Schoenberg-Withney condition) we cut
the fist and the last
        knots=tpt.x[1:-1]
    #We create the spline based on initial data and knots
    F = interpolate.LSQUnivariateSpline(x,y,knots,k=k,bbox=[x.min()-1e-12
,x.max()+1e-12],ext=2)
    #Returning the function
    return(F)
```

Loading and formating data

The following example is based on MOCoPo data's. The algorithm will work if is provided with a data frame called **Trajectories** containing at least the following columns : **id, time, x, y without any capital letter** *(no iD, Time, X or Y)*. **Time must be in seconds**. x and y in meters.Others columns may be used for data selection, for example, can be keep without problems and columns order doesn't matter. This is the only part that can be futher automatically done and must be set by the user with his knowledge of his dataset organisation.

In [4]:

```
%%time
#Mocopo's data
#access path
Mocopo=pd.read_csv('../Données sources/MOCoPo_Project_365_2011_09_16_Film
4.csv',sep=',',decimal='.',header=None)
#displaying the result for control
Mocopo.head()
Wall time: 10.2 s
```

In [5]:

```
%%time
#Creating a raw extraction for safety and verification if needed (not pre
sent in the code)
Trajectories_raw=Mocopo[[0,1,10,11,12]]
#Creating the working dataframe
Trajectories=Trajectories_raw.rename(columns={0:"id",1:"time",10:"x",11:"
y",12:"blob_size"})
#Setting the first timestamp as the new 0 of the dataframe.
#Absolute time sampling is not needed in the code, we're going to work on
relative time
Trajectories.time=Trajectories.time-Trajectories.time.iloc[0]
#displaying the result for control
Trajectories.head()
Wall time: 545 ms
```

In [6]:

```
#Parameters for selection
minLen=300 #Minimal number of observation
infCord=50 #x's data must have a starting point before this one
supCord=425 #x's data must have a endind point after this one
#Selecting data with consistent blob size along the trajectory
meanBlob=350 #Mean size of the blob
```

```
stdBlob=120 #standard deviation of the mean size of the blob
```

In [7]:

```
%%time
#During the furthermore section, Selected will be a temporary dataframe f
or applying conditions set for selection
#We group the data by id and we count the number of time the id is presen
t in the dataframe
Selected=Trajectories.id.groupby(Trajectories.id).count()
#We keep only ids with a number of data greater than minLen
Selected=Selected[Selected>minLen]
#Now we keep ids that match the previous condition
Trajectories=Trajectories[Trajectories.id.isin(Selected.index)]
#We group the data by id and we calculate the mean for all the column but
only size have an interest
Selected=Trajectories.groupby(Trajectories.id).mean()
#We keep only the id with a mean size of the blob lesser than meanBlob
Selected=Selected[Selected.blob_size<meanBlob]
#Again, we keep ids that match the previous condition
Trajectories=Trajectories[Trajectories.id.isin(Selected.index)]
#We group the data by id and we calculate the standard deviation for all
the column but only size have an interest
Selected=Trajectories.groupby(Trajectories.id).std()
#We keep only ids with a blob'size standard deviation lesser than stdBlob
Selected=Selected[Selected.blob_size<stdBlob]
#Once more, we keep ids that match the previous condition
Trajectories=Trajectories[Trajectories.id.isin(Selected.index)]
#We duplicate the x column, by this way we can apply the border condition
simultaneously
Trajectories.insert(3,"x1",Trajectories.x.copy(),True)
#For each id, we keep only the first value in the x column and the last v
alue in the x1 column
Selected=Trajectories.groupby(Trajectories.id).agg({"x":lambda row: row.i
loc[0],"x1":lambda row: row.iloc[-1]})
#We filter the x column, only keeping the one that are lesser than infCor
d
Selected=Selected[Selected.x<infCord]
#We filter, now, the x1 column, only keeping the one that are greater tha
n supCord
Selected=Selected[Selected.x1>supCord]
#finally, we keed ids matching this last double condition
Trajectories=Trajectories[Trajectories.id.isin(Selected.index)]
#we drop the temporary column (duplicate of x)
Trajectories=Trajectories.drop('x1',axis=1)
#displaying the result for control
Trajectories.head()
Wall time: 2.38 s
```

Algorithm

In [8]:

```
# Setting the parameter for function call
deg_R=2 #spline'degre
slope_R=1e-6 #slope tolerance
MaxKnotsR=0.5 #Max Knots per second

deg_Th=2 #spline'degre
slope_Th=1e-6 #slope tolerance
MaxKnotsTh=0.5 #Max Knots per second

deg_S=3 #spline'degre
acceptable_percentage_S=5 #percentage of outlet tolerate
deltaS=0.3 #maximum distance between spline and original data
```

In [9]:

```
%%time
#setting j at 0 for tracking trajectories remove by time filter
j=0
#Creating Group for browsing
Trajectories_Group=Trajectories.groupby('id')
#Creating result dataframe
Trajectories_Fit=pd.DataFrame()
#For each unique id we apply polar filtering
for id,x in Trajectories_Group:
    Current=Trajectories[Trajectories.id==id]
    #print(id)

    #filtrage temporel (à mettre ailleurs)
    Current=Current.copy()
    Current['time1']=Current['time']
    Current.time1=Current.time1.shift(1)
    Current['Change']=Current.time-Current.time1
    if len(Current[Current.Change>1])>0:
        j=j+1
        continue

    #setting x,y and t value from the extract dataset
    x=Current.x
    y=Current.y
    t=Current.time

    #init 0,0 origin
    x=x-x.iloc[0]
    y=y-y.iloc[0]

    #creating the complex number z from x and y
    z=x+1j*y
    #Computing r and th from z
    r=np.abs(z)
    th=np.angle(z)
```

```
    #Creating R and Thêta spline with turning point based knots
    interpR=TPSpline(t,r,deg_R,slope_R,MaxKnotsR)
    interpTh=TPSpline(t,th,deg_Th,slope_Th,MaxKnotsTh)
    #interpR=interpolate.Akima1DInterpolator(t,r)
    #interpTH=interpolate.Akima1DInterpolator(t,th)
    #Calculing new r and thêta from spline function
    rop=interpR(t)
    thop=interpTh(t)
    #Calculing polar distance derivative from the new r and thêta and the
ir derivatives
    ds=np.sqrt(interpR(t,1)**2+(interpTh(t,1)*rop)**2)
    #integration of ds for getting polar distance s
    s=integrate.cumtrapz(ds,t,initial=0)
    #Making a linear knots based spline for s
    [s_prime,interpS,Np,knots] = Spline_function(t,s,deg_S,acceptable_per
centage_S,deltaS)
    #interpS=interpolate.Akima1DInterpolator(t,s)

    #uniform base time raise error on recreation of x and y
    #base_temps=np.arange(t.min()+0.1,t.max(),0.1)
    base_temps=t
    #Getting Distance, Velocity, Acceleration and Jerk from the s spline
function and its derivative
    Distance=interpS(base_temps)
    Velocity=interpS(base_temps,1)
    #numeric calculation can lead to near 0 negative velocity that are co
rrected here
    Velocity[Velocity<0]=0
    Acceleration=interpS(base_temps,2)
    Jerk=interpS(base_temps,3)

    #Complex solving method
    #Declaring list for saving data
    X=[]
    Y=[]
    zr=[]
    #initialisating first value
    zr.append(cmath.rect(r.iloc[0],th[0]))
    X.append(zr[0].real)
    Y.append(zr[0].imag)
    #We parse through the s vector
    for i in range(1,len(s)):
        #setting last r as new depart point
        r0=cmath.polar(zr[i-1])[0]
        #setting the distance from s
        rayon=s[i]-s[i-1]
        #calculating the two possibles solutions
        zr1=cmath.rect((r0+rayon),thop[i])
        zr2=cmath.rect((r0-rayon),thop[i])
        #calulating distance from the original data
```

```
        dist1=np.abs(z.iloc[i]-zr1)
        dist2=np.abs(z.iloc[i]-zr2)
        #we keep the smallest distance
        if dist1<=dist2:
            zr.append(zr1)
        else :
            zr.append(zr2)
        #Getting back from complex to cartesian
        X.append(zr[i].real)
        Y.append(zr[i].imag)

    #Saving data !
    Id=np.ones(len(base_temps))*id
    #print(len(Id),len(x))
    Trajectory_Fit=pd.DataFrame(np.array([Id,base_temps,X,Y,Distance,Velo
city,Acceleration,Jerk,rop,thop]).T,
                                columns=['id','time','X','Y','Distance','
Velocity','Acceleration','Jerk','r','th'])
    #each fiited trajectory dataframe is add to the trajectories datafram
e
    Trajectories_Fit=pd.concat([Trajectories_Fit,Trajectory_Fit],sort=Fal
se)
Wall time: 2min 52s
```

Macro Analyse

In [10]:

```
#Number of trajectories with time gap
j
```

In [12]:

```
#Creating histogram of acceleration distribution
hist, edges = np.histogram(Trajectories_Fit.Acceleration,bins=100,range=[
-4,4])
title='Acceleration distribution'
x_label='Acceleration (m/s²)'
y_label='Number of observations'
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label)
p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="
red",legend_label='fitted data')
show(p)
```

In [13]:

```
#Determining jerk changing sign
Jerk_changing_sign=Trajectories_Fit.loc[:,['id','time','Jerk']]
Jerk_changing_sign['Jerk1']=Jerk_changing_sign.Jerk.shift(1)
Jerk_changing_sign.Jerk1[0]=-1
Jerk_changing_sign['Change']=Jerk_changing_sign.Jerk*Jerk_changing_sign.J
erk1
Jerk_changing_sign=Jerk_changing_sign[Jerk_changing_sign.Change<0]
Jerk_changing_sign.time=np.append(1e9,np.diff(Jerk_changing_sign.time))
```

```
Jerk_changing_sign.time[Jerk_changing_sign.time<0]=1e9
#percentage of jerk changing sign < 1 s
(len(Jerk_changing_sign[Jerk_changing_sign.time<1])/len(Jerk_changing_sig
n.time))*100
```

In [14]:

```
#Plotting jerk changing sign histogram
hist, edges = np.histogram(Jerk_changing_sign.time,bins=100,range=[0,10])
title='Time between each change of sign of jerk values distribution over
all the fitted trajectories'
x_label='Time between each change of sign of jerk values (s)'
y_label='Number of observations'
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label)
p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="
red")
show(p)
```

In [15]:

```
#Determining jerk changing value
Jerk_changing=Trajectories_Fit.loc[:,['id','time','Jerk']]
Jerk_changing['Jerk1']=Jerk_changing.Jerk.shift(1)
Jerk_changing['Change']=Jerk_changing.Jerk-Jerk_changing.Jerk1
Jerk_changing=Jerk_changing[Jerk_changing.Change!=0]
Jerk_changing.time=np.append(1e9,np.diff(Jerk_changing.time))
Jerk_changing.time[Jerk_changing.time<0]=1e9
#percentage of jerk changing value < 1 s
(len(Jerk_changing[Jerk_changing.time<1])/len(Jerk_changing.time))*100
```

In [16]:

```
#Plotting jerk changing value histogram
hist, edges = np.histogram(Jerk_changing.time,bins=100,range=[0,8])
title='Time between each change of jerk values distribution over all the
fitted trajectories'
x_label='Time between each change of jerk values (s)'
y_label='Number of observations'
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label)
p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="
red")
show(p)
```

In [17]:

```
#plotting histogram of jerk changing sign by value
hist, edges = np.histogram(Jerk_changing_sign.Jerk,bins=100,range=[-1,1])
title='Jerk value when jerk changing sign distribution over all the fitte
d trajectories'
x_label='Jerk value when jerk changing sign (s)'
y_label='Number of observations'
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label)
```

```
p.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:], line_color="
red")
show(p)
```

In [18]:

```
title='Time in each jerk changing sign by jerk value'
x_label='time in each jerk changing sign (s)'
y_label='Jerk value'
x_range=[0,20]
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label,x_range=x_range)
p.circle(Jerk_changing_sign.time,Jerk_changing_sign.Jerk)
show(p)
```

In [19]:

```
title='Time between each jerk changing value by jerk value'
x_label='time between each jerk changing value (s)'
y_label='Jerk value'
x_range=[0,20]
p = figure(plot_width=900, plot_height=500,title = title,x_axis_label = x
_label,y_axis_label =y_label,x_range=x_range)
p.circle(Jerk_changing.time,Jerk_changing.Jerk)
show(p)
```

Individual analyse

In [20]:

```
#Creating list of id used in Trajectories
ListId=Trajectories_Fit.id.unique()
#Initialising i at 0 for updating viewer window
i=0
#initialisition value with B the raw data and A the fitted one
B=Trajectories[Trajectories.id==ListId[i]]
A=Trajectories_Fit[Trajectories_Fit.id==ListId[i]]
x=B.x
y=B.y
x=x-x.iloc[0]
y=y-y.iloc[0]
z=x+1j*y
r=np.abs(z)
th=np.angle(z)
```

In [21]:

```
def update(i=0):
    #function for updtating the draw in the next cell
    A=Trajectories_Fit[Trajectories_Fit.id==ListId[i]]
    B=Trajectories[Trajectories.id==ListId[i]]
    x=B.x
    y=B.y
    x=x-x.iloc[0]
    y=y-y.iloc[0]
    z=x+1j*y
```

```
    r=np.abs(z)
    th=np.angle(z)
    r0.data_source.data={'x':x,'y':y}
    r1.data_source.data={'x':A.X,'y':A.Y}
    r2.data_source.data={'x':A.time,'y':A.Distance}
    r3.data_source.data={'x':A.time,'y':A.Velocity}
    r4.data_source.data={'x':A.time,'y':A.Acceleration}
    r5.data_source.data={'x':A.time,'y':A.Jerk}
    r6.data_source.data={'x':A.time,'y':A.r}
    r7.data_source.data={'x':B.time,'y':r}
    r8.data_source.data={'x':A.time,'y':A.th}
    r9.data_source.data={'x':B.time,'y':th}
    p.title.text='trajectoire n° '+str(ListId[i])
    push_notebook()
```

In [23]:

```
#run it twice for removing error message.
title='trajectoire n° '+str(ListId[i])
p = figure(plot_width=900, plot_height=500,title=title,x_axis_label ='x (
m)',y_axis_label ='y (m)')
r0 = p.line(x,y,legend_label='raw data')
r1 = p.line(A.X,A.Y,color='red',legend_label='fitted data')
p.legend.location = "bottom_right"
p1 = figure(plot_width=450, plot_height=250, title='Distance',x_axis_labe
l ='temps (s)',y_axis_label ='Distance (m)')
r2 = p1.line(A.time,A.Distance)
p2 = figure(plot_width=450, plot_height=250, title='Speed',x_axis_label =
'temps (s)',y_axis_label ='Vitesse (m/s)')
r3 = p2.line(A.time,A.Velocity)
p3 = figure(plot_width=450, plot_height=250, title='Acceleration',x_axis_
label ='temps (s)',y_axis_label ='Acceleration (m/s²)')
r4 = p3.line(A.time,A.Acceleration)
p4 = figure(plot_width=450, plot_height=250, title='Jerk',x_axis_label ='
temps (s)',y_axis_label ='Jerk (m/s\u00B3)')
r5 = p4.line(A.time,A.Jerk)
p5 = figure(plot_width=900, plot_height=300, title='Rayon polaire',x_axis
_label ='temps (s)',y_axis_label ='r (m)')
r7 = p5.circle(B.time,r,legend_label='données brutes',size=6)
r6 = p5.line(A.time,A.r,color='red',line_width=3,legend_label='données li
ssées')
p5.legend.location = "bottom_right"
p6 = figure(plot_width=900, plot_height=300, title='Thêta',x_axis_label =
'temps (s)',y_axis_label ='thêta (rad)')
r9 = p6.circle(B.time,th,legend_label='données brutes',size=6)
r8 = p6.line(A.time,A.th,color='red',line_width=3,legend_label='données l
issées')
interact(update, i=(0,len(ListId)-1))
grid = column(p,row(p1,p2),row(p3,p4),p5,p6)
show(grid, notebook_handle=True)
```