Les technologies spatiales au service de l'écoconduite

Rapport de Projet Tutoré

Julien Baladier
Yorrick Barnay
Loïc Boyeldieu
Aurélien Tamas-Leloup

Les technologies spatiales au service de l'écoconduite

Rapport de Projet Tutoré

Julien Baladier
Yorrick Barnay
Loïc Boyeldieu
Aurélien Tamas-Leloup

Remerciements

Tout d’abord nous souhaitons remercier Romain Desplats qui nous a accompagné tout au long de ce projet. Nous avons pu bénéficier de ses conseils et de son suivi pour mener à bien la réalisation de ce travail. Nous remercions aussi Léo Lebrat pour son aide de génie sur les problèmes mathématiques rencontrés. Enfin, nous tenions absolument à mentionner Javier qui nous a fait part de son expertise sur Matlab.

Sommaire

Introduction

L'économie de l'énergie est l'un des enjeux majeurs du XXIème siècle. De nos jours, il est indispensable pour les entreprises, les citoyens et les collectivités territoriales de pouvoir se déplacer à tout moment et de transporter des marchandises. Les émissions de gaz à effet de serre, les embouteillages, la pollution de l'air sont alors des problématiques majeures pour notre société. En effet les transports sont les premiers responsables de l'émission des gaz à effet de serre en France et deuxièmes en Europe avec 26% des émissions de dioxyde de carbone.

Il est donc évident qu'un secteur aussi largement propagé que l'automobile est concernée. L'invention des véhicules à moteurs a su se rendre complexe et indispensable dans nos quotidiens au fil des années. Aussi essentielle qu'est une voiture, elle nécessite donc des coûts constants en carburant, ressource polluante dont la durée de vie n'est pas infinie.

L'écoconduite, procédé visant à conduire en optimisant sa consommation de carburant, est donc aujourd'hui un domaine de recherche très en vogue.

Notre projet sera de créer un coach virtuel sous la forme d'une application qui offrira la possibilité aux automobilistes de réduire leur consommation de carburant en temps réel. Pour ce faire nous adapterons des technologies utilisées dans l'aéronautique et le spatial à l'automobile. En effet de nombreuses missions spatiales dépendent de l'optimisation de l'énergie utilisée comme par exemple pour les séquences de réorientation de satellites. Nous parlerons donc tout au long de ce rapport de véhicules à moteur de manière générale.

I. La réalité de l'écoconduite

1. L'écoconduite aujourd'hui

À ce jour, on compte plusieurs applications d'écoconduite basées sur la simulation grâce au référencement de données sur les catégories de véhicules par exemple. Cependant il n'existe aucune application qui conseille en temps réel le conducteur sur son comportement au volant en s'appuyant sur la consommation instantanée. Par exemple, GECO est une application qui calcule en temps réel la conduite à adopter.

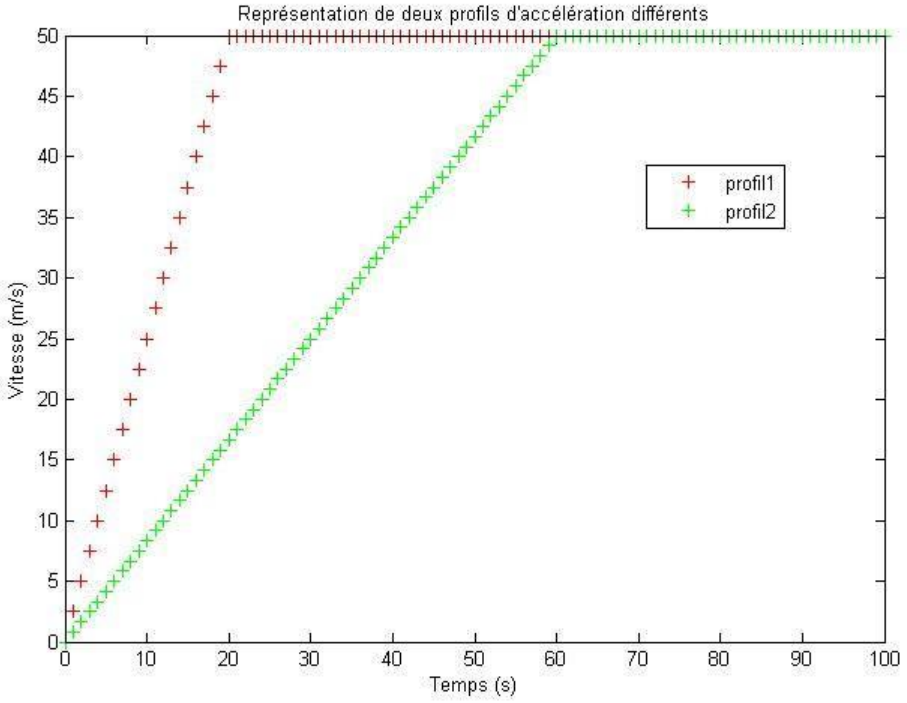
Voici la liste des conseils généralement donnés pour faire de l'écoconduite.
Accélérer doucement : Il est possible d'économiser 15% de la consommation de carburant en accélérant doucement car plus on accélère brusquement et plus on consomme de carburant.
Prévoir la densité de la circulation : La consommation de carburant est liée au style de conduite. Conduire brusquement en accélérant et freinant augmente la consommation. Il faut donc prévoir les changements de circulation.
Éviter la conduite à haute vitesse : Plus une automobile roule vite et plus elle consomme du carburant. Ralentir permet donc une conduite plus économique et moins polluante.
Réduire la charge : Le porte-vélo à l'arrière de la voiture et les portes bagages sur le toit des véhicules diminuent l'aérodynamisme et augmentent la consommation de carburant.

2. Le domaine spatial au service de l'écoconduite

Un des enjeux majeurs dans le domaine du spatial est l'optimisation des consommations de carburant. Il en va de la réussite des missions. Pour la réorientation des satellites par exemple il faut prévoir les trajectoires optimales pour économiser le carburant. La problématique d'optimisation de l'énergie dans le domaine spatial s'étend largement dans le domaine automobile.

En s'appuyant sur des modèles de la physique, nous pouvons montrer qu'accélérer rapidement ou lentement n'a pas de réel impact sur la consommation de carburant contrairement aux conseils qui nous sont donnés.

Nous aurions tendance à penser que pour faire des économies de carburants, le plus important, c'est de ne pas accélérer trop fort pour atteindre une vitesse cible. Nous allons vous montrer dans cette partie que la différence de consommation entre une personne qui accélère fortement et une autre doucement est négligeable. . Il ne s'agit pas là d'une démonstration rigoureuse et précise mais elle permet de visualiser le principe. Pour cela nous avons établi deux profils.



Le profil 1 correspond à un conducteur qui accélère fortement. Le profil 2 correspond à un conducteur qui accélère doucement. Nous prendrons une durée totale de cent secondes.

Le conducteur 1 atteint la vitesse cible en 20 secondes et le conducteur 2 en 60 secondes. Dans notre cas, on a : $Vcible = 50\ m/s$. Nous pouvons établir les équations des courbes profil1 et profil2 que nous appellerons respectivement y_1 et y_2.

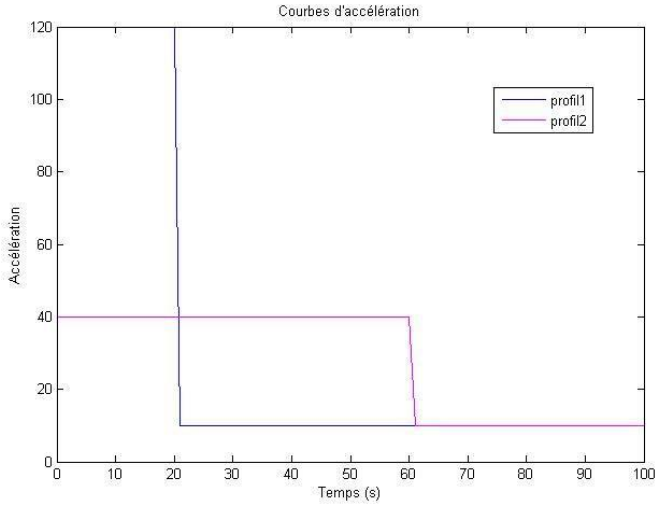
$$y_1 = \frac{5}{2}t \ si\ t \leq 20$$

$$y_1 = 50 \ si\ t > 20$$

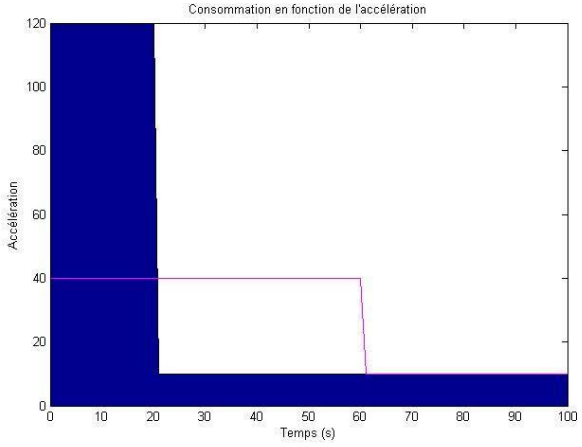
$$y_2 = \frac{5}{6}t \ si\ t \leq 60$$

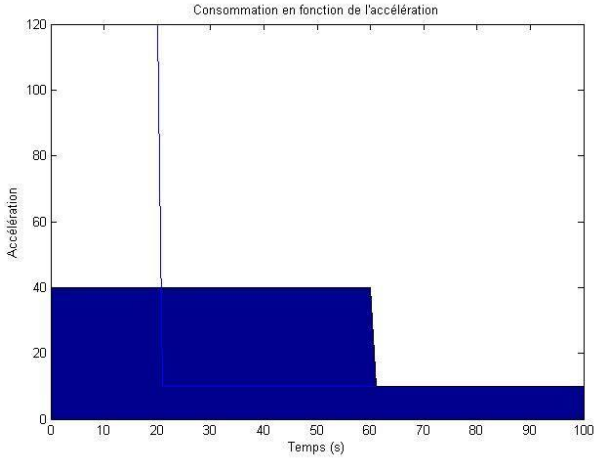
$$y_2 = 50 \ si\ t > 60$$

L’accélération étant la dérivée de la vitesse, nous pouvons déduire les courbes d’accélération :



Pour le profil 1, l’accélération est très élevée mais pour une durée plus courte. À partir de l’accélération, nous pouvons déterminer la consommation. Pour simplifier nous considérons que la consommation est l’intégrale de l’accélération. La consommation correspond donc à l’aire sous la courbe.





Calculons les aires en utilisant nos deux profils.

$$Conso1 = \int_0^{20} \frac{5}{2} dt + \int_{20}^{100} c\, dt = 50 + 80c$$

$$Conso2 = \int_0^{60} \frac{5}{6} dt + \int_{60}^{100} c\, dt = 50 + 40c$$

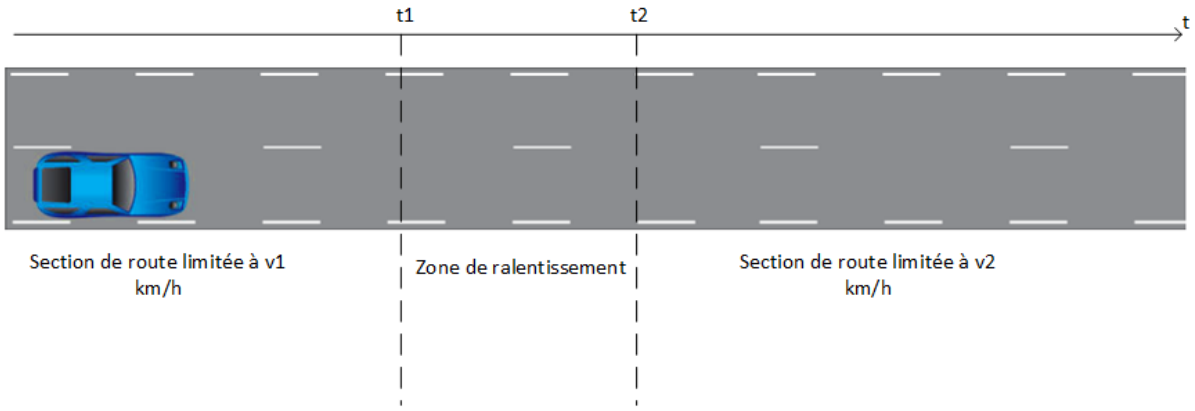
La différence entre les deux profils est $40 \times c$; c représente une constante correspondant à l'accélération lorsque que l'on a atteint la vitesse cible. Cette accélération étant très faible, on peut ainsi négliger la différence entre les deux profils. De plus, on voit bien que plus le trajet sera long, plus cette valeur va diminuer.

Nous montrons ainsi que l'écoconduite n'est pas une question d'accélération, c'est une question de freinage. C'est sur ce constat que nous avons développé notre application.

II. La génération des profils

1. Introduction

Nous avons donc vu et démontré que le freinage est l'élément primordial lorsqu'on parle d'écoconduite. L'important étant d'atteindre une vitesse dite cible au bon moment afin de ne pas gaspiller d'énergie. L'objectif de notre application sera par conséquent de calculer cette vitesse cible puis d'avertir l'automobiliste au bon moment pour qu'il l'atteigne sans avoir à trop freiner. La question est maintenant de savoir comment faire pour connaître avec certitude le moment où il faut commencer à décélérer en supposant que toutes les vitesses cibles sont connues à l'avance. Expliquons ce principe plus en détail à l'aide du schéma suivant :



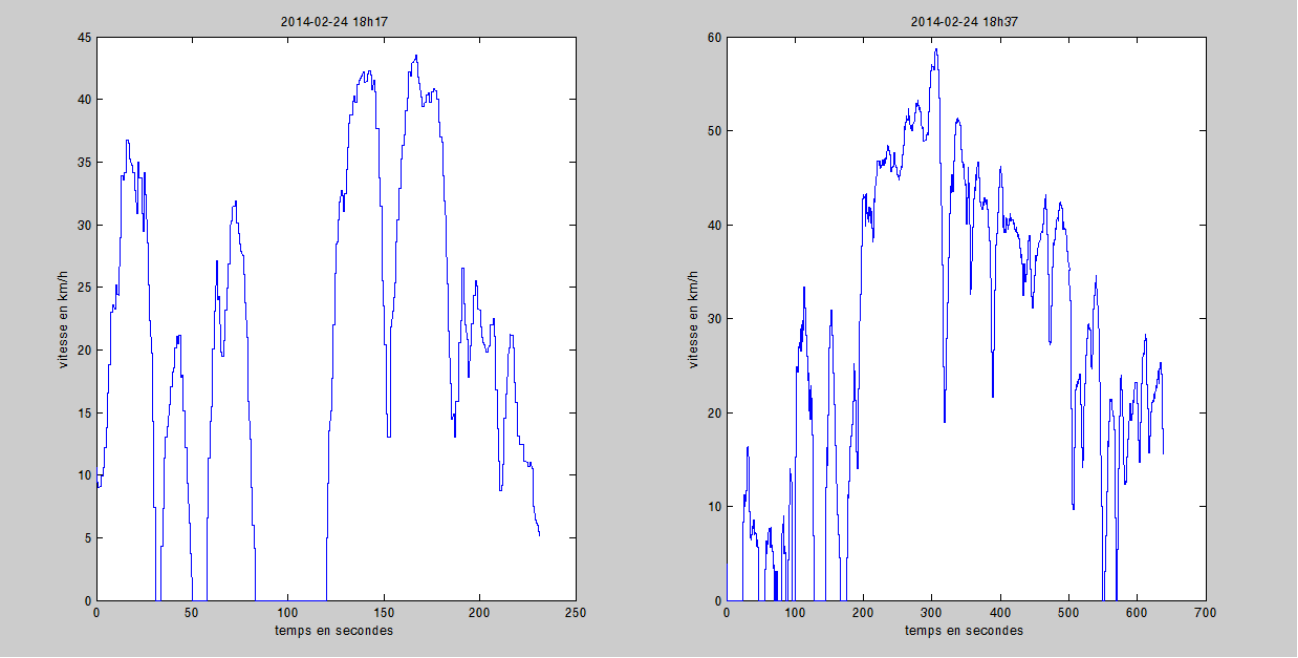
L'automobiliste est sur une route limitée à une vitesse quelconque v_1 mais il est sur le point d'arriver sur une section de route dont la vitesse, v_2 est inférieure. Soit t_1 l'instant ou l'automobiliste doit commencer à ralentir et t_2 l'instant où il peut conserver sa vitesse. Il est évident que ces deux valeurs sont inconnues à l'avance puisqu'ils vont dépendre de la manière dont l'automobiliste ralentit. Sur le schéma ils sont positionnés de manière idéale, t_2 coïncide avec le moment où la vitesse limite change. Si notre objectif est d'obtenir un t_2 idéal à chaque décélération alors le problème revient à calculer la différence entre t_2 et t_1 à partir des vitesses v_2 et v_1. Pour ce faire, nous avons donc besoin de savoir comment les vitesses évoluent par rapport au temps pour une décélération respectant les consignes d'éconduite. L'idéal serait donc d'obtenir une fonction

liant la vitesse au temps dans ce cas précis. Nous avons répondu à cette problématique en nous basant sur des données expérimentales que nous avons ensuite exploitées en utilisant le logiciel Matlab.

Dans un premier temps, on présentera nos données expérimentales, on expliquera ensuite comment on a pu exploiter ces données. Enfin nous présenteront et commenteront les résultats obtenus.

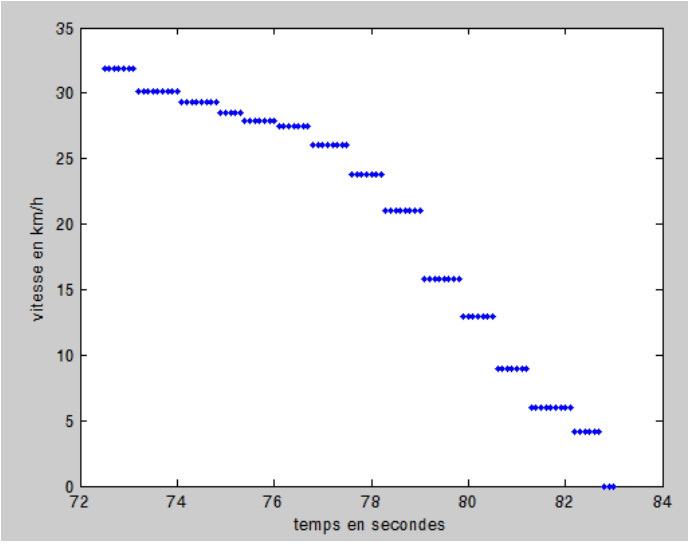
2. Les données d'écoconduite

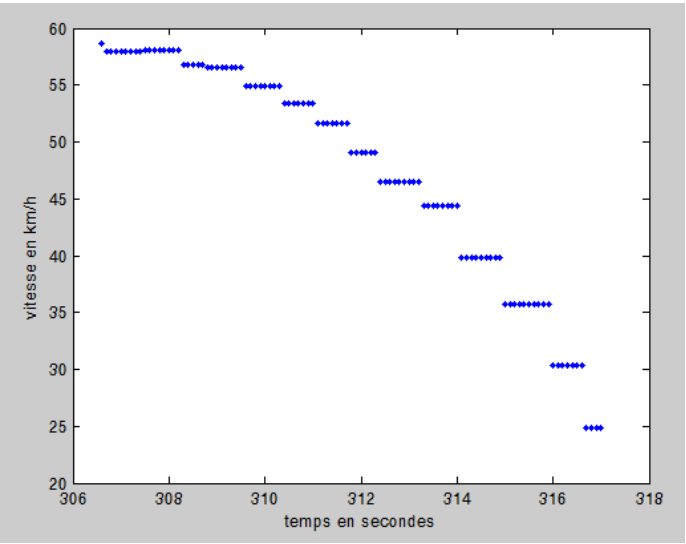
Nous avons reçu de la part de notre tuteur des logs enregistrés lors de deux trajets aux alentours de Montpellier. Ces parcours ont été réalisés par un chauffeur professionnel et expérimenté, celui-ci applique à la lettre les principes d'écoconduite abordés précédemment, surtout en ce qui concerne la décélération. Un capteur a été placé dans le véhicule lors de ses trajets. Ce capteur a permis de mesurer de nombreuses grandeurs physiques (le format complet des logs est donné en annexe) dont la vitesse du véhicule. Voici les résultats obtenus:



Ces courbes représentent l'évolution de la vitesse du véhicule en fonction du temps, bien entendu, nous nous intéressons principalement aux décélérations, deux zones retiennent particulièrement notre attention. Sur la première courbe entre 50 et 100 secondes on observe que le véhicule passe de 30 km/h à 0, on se basera sur cette décélération pour établir un profil d'arrêt. Sur la seconde courbe on atteint la vitesse de 60 km/h, ce qui est le maximum sur les deux trajets, puis le conducteur ralenti jusqu'à 20 km/h. On établira ainsi notre profil de ralentissement.

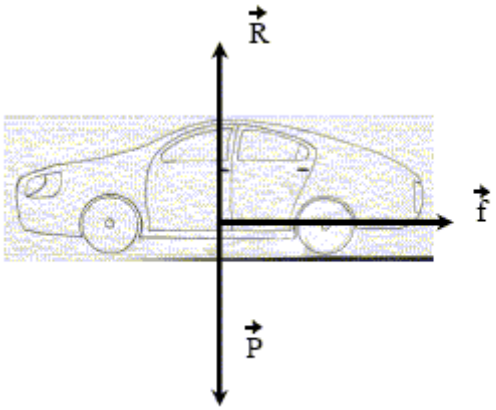
Voici les points retenus pour établir les profils :





3. Approximation du profil

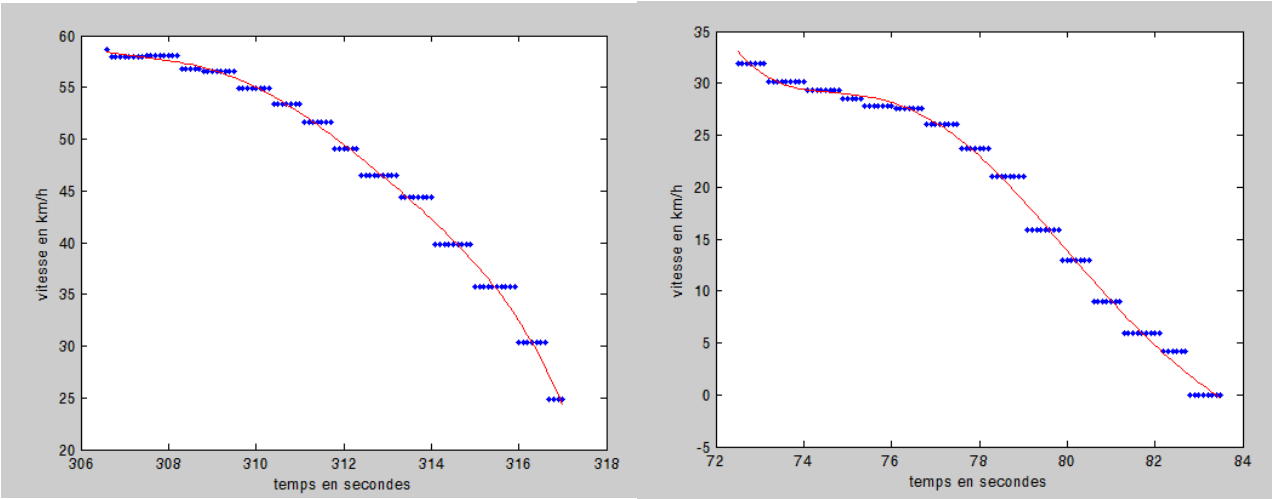
Comment faire maintenant, pour extraire de ces données expérimentales une fonction mathématique liant le temps à la vitesse ? Le premier élément de réponse à cette question serait de revenir aux lois de Newton et plus particulièrement à son principe fondamental de la dynamique. Les forces appliquées à un véhicule en décélération sont : son poids, la réaction de la route par rapport au véhicule ainsi que les frottements de l'air qu'on négligera. La réaction de la route a deux composantes : la composante verticale $\vec{R}$ qui compense le poids du véhicule et la composante horizontale $\vec{f}$ dont le sens est opposé au mouvement.



D'après le principe fondamental de la dynamique : $\vec{f} = m\vec{a}$ On sait que $\vec{a}$ est la dérivée temporelle de la vitesse, on sait aussi que $\vec{f}$ est fonction de la vitesse. Pour établir v en fonction de t de manière exacte, nous avons besoin de connaître l'expression de f par rapport à v afin de résoudre l'équation différentielle. Or,

celui-ci dépend de nombreux paramètres[1] et pour établir une formule correcte de f nous devrions mener de nombreux test. La théorie nous mène finalement dans une impasse, nous allons donc nous concentrer sur les résultats expérimentaux.

L'ajustement de courbe est une méthode mathématique permettant à partir d'un nuage de points expérimentaux d'obtenir une fonction mathématique s'en rapprochant le plus possible. On utilise pour ce faire des méthodes de régression, il en existe plusieurs types, nous avons ici utilisé des méthodes de régression polynomiale, plus particulièrement, la méthode des moindres carrés élaborée par Gauss. Fort heureusement, Matlab fournit deux fonctions très utiles pour utiliser simplement cette méthode. `polyfit(x,y,n)` renvoie un polynôme, $p(x)$ de degré n qui s'ajuste le mieux possible, d'après la méthode des moindres carrés aux données `y`. Rappelons que dans Matlab tout est vecteur, même les polynômes. En effet si on a par exemple $p(x) = 2x^2 + 3x + 1$ son équivalent dans Matlab sera `p = [2 3 1]`. C'est ici que la fonction `polyval(p,x)` intervient, celle-ci renvoie l'image y de x par le polynôme p.
Voyons maintenant la forme des polynômes obtenus grâce à ces fonctions.



4. Conclusion

Nous avons donc réussi à estimer la vitesse par rapport au temps dans le cas d'une décélération respectant les principes d'écoconduite. Nous allons conclure cette partie en commentant l'allure des deux courbes. Le profil d'arrêt comporte un point d'inflexion qui divise la courbe en deux parties, l'une avec une pente bien plus forte que l'autre. La deuxième partie de ce profil est à rapprocher du second profil, on peut dire que les allures des deux courbes sont assez similaires. On peut interpréter cela en disant que l'utilisation du frein moteur dans

[1] Voir annexe A

ces deux cas est maximale ce qui ralentit beaucoup la voiture. Cette hypothèse est renforcée par la première partie du profil d'arrêt qui montre une vitesse qui diminue lentement et donc une utilisation minimale du frein moteur suivie par une vitesse qui diminue bien plus rapidement. On peut donc conclure assez logiquement que le profil d’arrêt inclut un passage de vitesse, qu’il faudra prendre en compte pour la suite.

III. Le format et la génération des routes

Le but initial de notre application Android étant de conseiller un conducteur sur un circuit qu'il aura défini à l'avance, il nous semblait donc logique de lui proposer de définir son trajet afin de pouvoir récupérer de manière automatique toutes les informations nécessaires au traitement algorithmique qui nous permet de définir les vitesses cibles à proposer au conducteur tout au long de son parcours.

Dans cette partie, nous allons donc à nous intéresser à comment récupérer un trajet et en tirer des informations qui nous seraient utiles telles que la vitesse d'une portion de route, les différents points considérés comme des "obstacles" (ronds-points, carrefours, etc) en plus d'en tirer des informations clés concernant le dessin du parcours afin de pouvoir adapter l'allure du véhicule dans les virages et selon l'inclinaison de la pente. Récapitulatifs des informations clés d'un trajet :

- courbures (virages)
- longueur d'une voie
- ronds-points
- carrefours
- feux tricolores
- type de voie (autoroute, rocade, etc) ⇔ limitation de vitesse
- état d'encombrement de la route (embouteillages)
- inclinaison de pente ⇔ altitude

1. Simplification du modèle

Nous avons décidé de simplifier l'algorithme permettant de calculer la vitesse cible afin d'être en mesure de délivrer une première version simple mais fonctionnelle de notre projet plutôt qu'une version plus complexe mais non fonctionnelle.

Les informations concernant les courbures d'une route, l'altitude, les ronds-points, les embouteillages et les obstacles (feux tricolores, carrefours,...) sont pour

l'instant mises de côté dans l'algorithme ; cependant la récupération de certaines de ces données sera tout de même étudiée dans cette partie, cela pourrait être utile pour une version ultérieure de l'algorithme.

Nous noterons également qu'une difficulté supplémentaire concernant les feux tricolores se présentera si jamais ils sont implémentés dans une version suivante, les feux étant des obstacles aléatoires (ils peuvent être rouge, orange, vert, orange clignotant voire non fonctionnels). Chaque état amenant des décisions différentes au niveau de l'algorithme, il faudrait récupérer l'état des feux en temps réel.

2. Les services de google

Afin de permettre à l'utilisateur d'avoir une expérience simple et interactive, il nous a paru logique d'utiliser les services de google afin de créer son trajet, leur service de cartographie étant très bien pensé. Il est en effet très facile de dessiner son trajet et de le télécharger sous format kml à partir d'un service spécial de Google maps appelé "My Maps" et disponible à cette URL : https://www.google.com/maps/d/

Le centre d'aide de Google précise ceci à propos du format kml : "KML (Keyhole Markup Language) est un format de fichier et de grammaire XML pour la modélisation et le stockage de caractéristiques géographiques comme les points, les lignes, les images, les polygones et les modèles pour l'affichage dans Google Earth, dans Google Maps et dans d'autres applications. "

Un tracé au format kml est principalement constitué d'une suite de coordonnées sous forme "longitude, latitude" représentant un ensemble de points entre le départ et l'arrivée du trajet.

Ce format parait donc dans un premier temps répondre à deux de nos besoins : récupérer la courbure d'une route et la longueur d'un tracé après de savants calculs effectués entre chaque point du chemin.

Google étant capable de nous fournir le reste des informations nécessaires à travers son service classique de Google Maps, miser sur cette solution paraissait une bonne idée.

3. L'API Google Places

Il ne restait donc qu'à se plonger dans l'API Google afin de voir comment récupérer automatiquement les informations concernant chaque point du tracé.

Au sein du service Google Developers, on s'est rendu compte qu'il existe 4 API : iOS, Android, Javascript, WebService. Nous avons donc cherché à determiner quelle API nous pourrions utiliser. Dans un premier temps, il était logique d'éliminer l'API dédiée à iOS, notre application étant sous environnement Android.Dans un second temps, nous avons consulté les documents d'aide à

propos des autres API. Chaque API est résumée en une phrase par Google afin d'aider les développeurs dans le choix de l'API optimale par rapport à leurs besoins. La librairie Google Places pour JavaScript est résumée ainsi : " Add place search to your map, from within the Google Maps JavaScript API.". De même, la librairie Google Places pour WebService est résumée ainsi : "Build location-based applications backed by Google's database of places." Il est donc évident que ces API ne répondent pas à nos attentes également, la première permettant simplement d'ajouter une barre de recherche à une carte Google Map présente sur un site exactement comme sur la photo ci-dessous.

La seconde n'est pas destinée à une client-side application mais plutôt à une application tournant sur u serveur, ce qui aurait pu être intéressant dans le cadre de notre projet. Cependant elle permet uniquement de faire des recherches sur des lieux prédéfinis (sites historiques, magasins, etc) ou d'en créer de nouveaux.Nous avons donc placé notre espoir dans la librairie Google Places Android et l'avons étudié de plus près.

GUIDES

- Developer's Guide
  - Signup and API Keys
  - **Getting Started**
  - Place Picker
  - Current Place
  - Place Autocomplete
  - Place Add
  - Place Report
  - Place IDs and Details

L'API se décompose en plusieurs parties comme le montre le plan du guide sur l'image ci-dessus. Pour des raisons de pertinences, nous nous sommes intéressés plus particulièrement à la partie Place IDs and details. Nous nous sommes alors rendu compte que tout comme la librairie WebService, on ne peut travailler que sur des lieux prédéfinis (sites historiques, magasins, etc) ou en créer de nouveaux. Il est donc maintenant évident que la bibliothèque Google Places ne correspond pas du tout à nos attentes.

4. Les autres API Google

Fortement déçu par le fait que l'API Google Places ne répondait pas à nos besoins, nous avons continué nos recherches. Nous nous sommes alors rendu compte qu'il existait d'autres bibliothèques qui pourraient répondre à nos attentes. C'est ainsi que nous avons découverts Google Maps Roads API qui permet de récupérer la limitation de vitesse d'une route ; cette API répond donc parfaitement à un de nos besoins. Malheureusement il s'agit d'une API payante car dédiée aux personnes possédant un contrat avec Google. Dans le cadre de l'implémentation de notre projet pour un client, nous aurions pu donc nous reposer sur cette API afin de réaliser notre système d'écoconduite final. Google Maps JavaScript API a également attiré notre attention car cette API permet de récupérer l'état de circulation d'une route en temps réel, ce qui répond à un autre de nos besoins. Cependant ce besoin n'a pas d'utilité dans notre système actuel à cause des simplifications apportées et expliquées dans la partie A.

5. L'API Overpass

Présentation de l'API : Comme indiqué dans le wiki d'OpenStreetMap, l'API Overpass est une API en lecture seule qui permet de travailler sur des parties de cartes OpenStreetMap bien définies. Elle agit en fait comme une base de données sur le Web : le client envoie une requête à l'API et récupère l'ensemble de données qui correspond à la requête.

Il est très facile de récupérer les noms de rue, les vitesses limites, le type de voie ou d'autres informations concernant les voies et lieux d'une ville. Cette API est donc parfaite si on souhaite faire quelques requêtes afin de récupérer des informations concernant les routes empruntées pour ensuite les renseigner à l'organe du système en charge d'exécuter l'algorithme de détermination des vitesses cibles.

Ces requêtes peuvent être effectuées sous la forme de requêtes XML ou de requêtes Overpass QL, format de requête spécifique à l'API Overpass.

Nous avons choisi de les effectuer en Overpass QL car c'est un langage de requête très puissant.Il est également très facile de lancer une requête en Overpass QL car cela correspond, du point de vue du code, à lancer une simple

requête http. Pour questionner l'API Overpass, il y a malheureusement quelques contraintes : on est obligé de questionner la base de données sur un sous-ensemble de carte ; on ne peut la questionner sur un point précis.

En effet, chaque requête étudie un carré dont le coin inférieur gauche a pour coordonnées (lower latitude, lower longitude) et le coin supérieur droit (upper latitude, upper longitude) ; ces couples de coordonnées étant passés en paramètre dans la requête. Ce carré est dénommé "bounding box".

Idée de départ : Notre idée de départ était donc de récupérer le fichier kml de Google Map contenant la liste des coordonnées représentant le trajet à effectuer puis de questionner l'API avec les coordonnées de deux points successifs du trajet pour définir la "bounding box".

En comparant ensuite les résultats obtenus couple de points par couple de points, nous en aurions déduit des portions de trajet caractérisées par un nom de voie, une longueur et une vitesse maximum. Bien que l'idée semble facile à mettre en place, un problème majeur s'est présenté : le nombre de requêtes nécessaires était trop élevé, entraînant un temps de récupération des données et un temps de traitement de ces données trop important, empêchant cette solution d'être viable.

Solution Théorique : Heureusement Overpass QL étant puissant, une solution est à portée de main.Il est en effet possible de récupérer toutes les routes contenues dans une bounding box puis tous les points appartenant à cette route.On partirait donc d'un couple de points (les deux premiers points du trajet dans le fichier kml) pour définir une première bounding box, on récupérerait ensuite la route associée ainsi que tout les points lui appartenant. On travaillerait sur le résultat de la requête afin de déterminer une première portion de trajet puis on recommencerait la même opération jusqu'à la fin à partir du premier couple de points du trajet dans le fichier kml n'appartenant pas à la route étudiée précédemment. On pourrait donc déterminer les différentes portions de trajet relativement facilement.

Problème lors de la mise en pratique : Malheureusement OpenStreetMap et Google Maps étant des services distincts, il existe entre les deux des différences entre les coordonnées d'un même point. Bien qu'infimes, ces différences empêchent le fonctionnement de la solution théorique développée plus haut.

Standardisation des résultats : Afin de permettre aux autres membres de pouvoir travailler sur la partie algorithmique parallèlement au travail de génération de route, un format a été défini. Cela permet de standardiser les échanges entre les différentes parties prenantes. Ce standard est utilisé dans un fichier json[2] et se présente sous la forme suivante : un label désignant le type

[2] Fournit en annexe

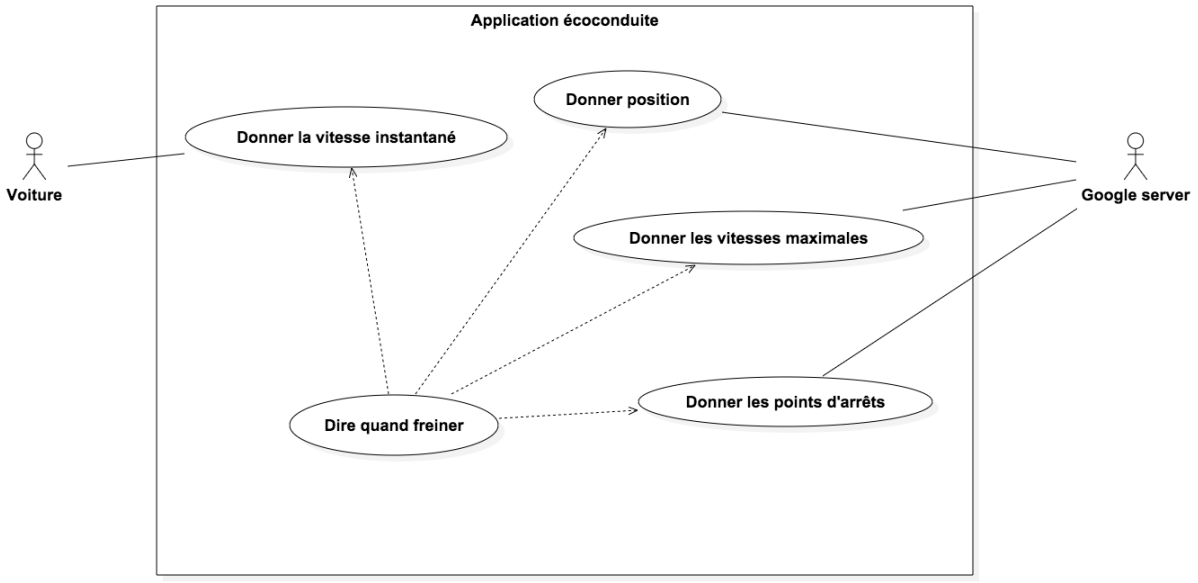
d’objet suivi de “:” et de l’objet en question. La liste des labels est la suivante : roads, name, length, max_speed_unit, segments, max_speed, length. Le standard choisi est le suivant :

- un segment de route est caractérisé par une vitesse maximum “max_speed” et une longueur “length”
- une route est caractérisée par un nom “name”, une unité de longueur “length_unit”, une unité de vitesse “max_speed_unit” ainsi qu’une liste de segments “segments”.
- le fichier json est composé d’une liste de routes “roads”

IV. L'application de test : EcoDriver

1. Introduction

Grâce aux études que nous avons présentées dans les parties précédentes, nous pouvons affirmer dès à présent que la démarche la plus judicieuse serait de créer une application qui se base principalement sur le freinage du conducteur. En effet, nous sommes maintenant en mesure de jouer sur ce point grâce aux courbes du profil idéal que nous avons mises en place. Le but de l'application embarquée sera donc d'aider l'automobiliste à éviter tout freinage : il ne se servirait plus que du frein moteur et des forces de frottements afin de décélérer. Voici un diagramme de cas d'utilisation de l'application que nous avons réalisée :



Comme on peut le voir sur ce diagramme, l'application dit simplement au conducteur quand freiner afin de que celui-ci consomme le moins de carburant possible. Pour faire de telles prédictions deux approches différentes se présentent à nous :

La première consiste à utiliser uniquement le trajet que va effectuer l'utilisateur. Dans ce cas, nous considèrerons que l'automobiliste roule à la vitesse maximale autorisée sauf dans les zones de décélération.

La seconde vise à mettre à jour à tout instant cette estimation en fonction de la vitesse instantanée, la position du conducteur et le trajet qu'il veut effectuer.

Avant de décrire en détail le développement de cette application Android, nous allons dans un premier temps présenter une application de test que nous avons conçu en parallèle. Elle se nomme EcoDriver. Dans cette partie, nous nous attacherons donc à prouver l'utilité d'une telle application de test, puis nous détailleront ses fonctionnalités et son implémentation en JAVA.

2. Pourquoi une application de test ?

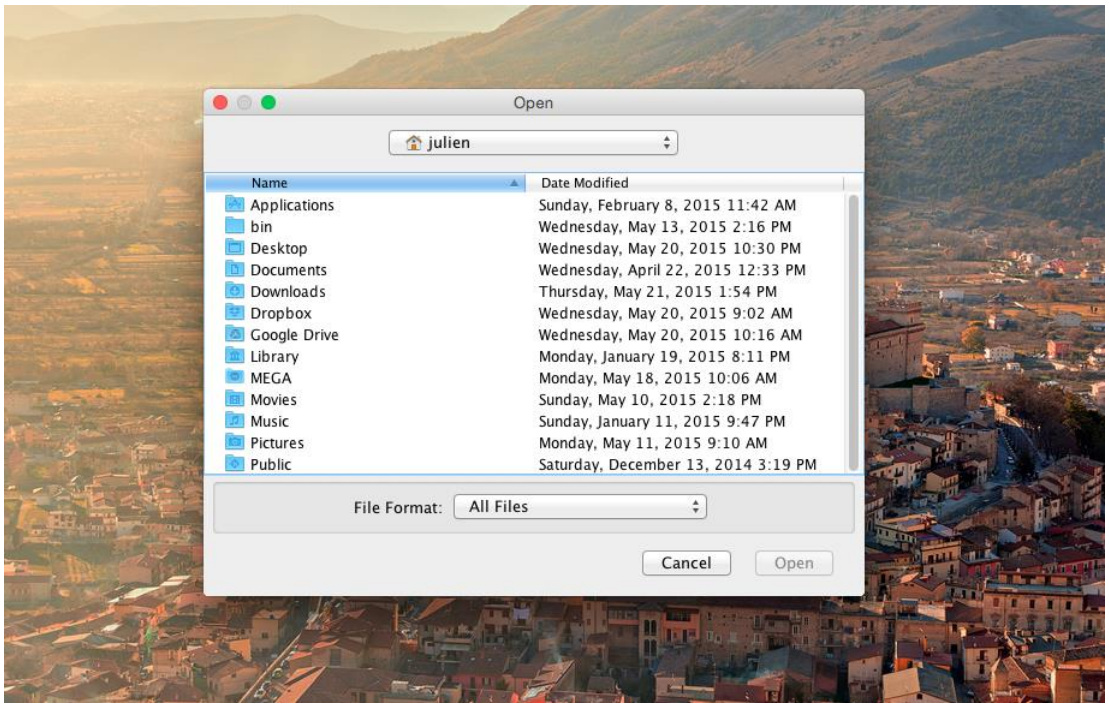
Suite aux problèmes rencontrés lors de la recherche d'un service nous permettant de récupérer les vitesses maximales ainsi que les points d'arrêts de manière automatisée, nous avons dû faire face à d'autres complications. En effet, aucun membre de notre groupe ne possède de véhicule. Il est donc compliqué de pouvoir effectuer des tests sur notre application le long des routes de notre région. Par conséquent, nous n'étions pas en mesure de représenter les deux acteurs qui figurent sur notre use case diagram et qui donc agissent sur notre système. C'est pourquoi nous avons décidé d'introduire une application de test qui représentera ces acteurs. Elle fournira une interface sur laquelle notre application Android pourra se baser pour faire ses calculs et ainsi assister le conducteur. Enfin, il nous sera facile de transposer notre application dans le monde réel en changeant ses interfaces.

Nous pouvons également ajouter qu'EcoDriver nous a grandement facilité la simplification du monde réel qui était nécessaire dans un premier temps. Nous pourrons par la suite ajouter de la complexité aisément, comme par exemple l'inclinaison des routes, les virages... en envoyant d'autres informations au téléphone exécutant notre code. L'application Android peut donc être développée dans de bonne condition, avec des prédictions à effectuer de plus en plus précises.

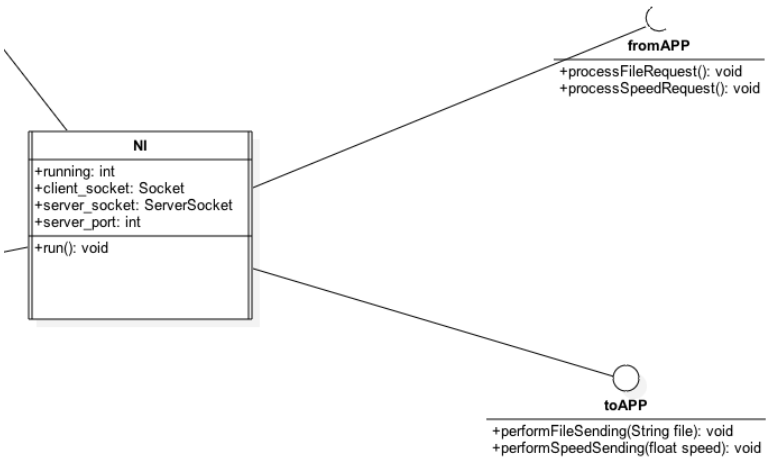
3. Présentation de son implémentation

Présentons à présent le fonctionnement interne d'EcoDriver. Comme il a été stipulé précédemment, ce logiciel représente non seulement le véhicule mais également les serveurs de Google. Il doit donc répondre aux requêtes de l'application Android qui lui demandera dans un premier temps le trajet que le conducteur veut effectuer puis périodiquement la vitesse instantanée de la voiture.

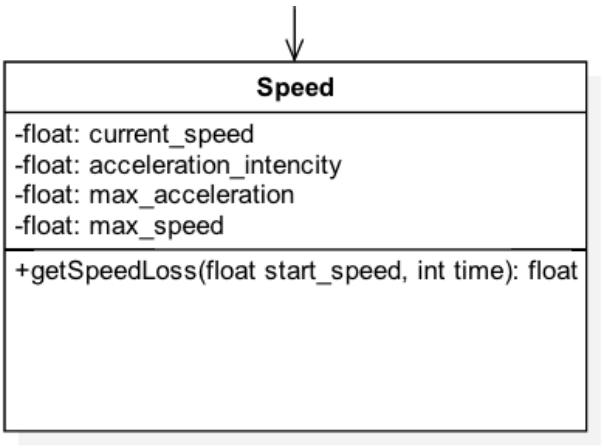
Pour effectuer cette première tâche, elle propose à son démarrage de choisir un fichier dans le format que nous avons défini dans la partie précédente. Cette demande se matérialise par une fenêtre qui apparait devant l'utilisateur comme nous pouvons le voir sur cette copie d'écran :



Ensuite le fichier doit être envoyé au téléphone via TCP. EcoDriver représente le server dans cette communication, il doit donc attendre une demande de connexion de la part de l'application Android pour pouvoir être en mesure de lui envoyer le fichier. Nous avons fait le choix d'utiliser le protocole TCP pour sa fiabilité. En effet, nous souhaitons que le fichier soit reçu en intégralité par le destinataire pour éviter toute confusion. En interne cette partie est traitée par le NI dont voici la représentation en UML :



Comme nous pouvons remarquer sur les interfaces qu'il implémente, il permet également de répondre aux requêtes de demande de vitesse instantanée. Pour pouvoir satisfaire le client, notre application a donc besoin de gérer une vitesse en interne. Cela est matérialisé par une instance de la classe Speed qui représente notre seule donnée :

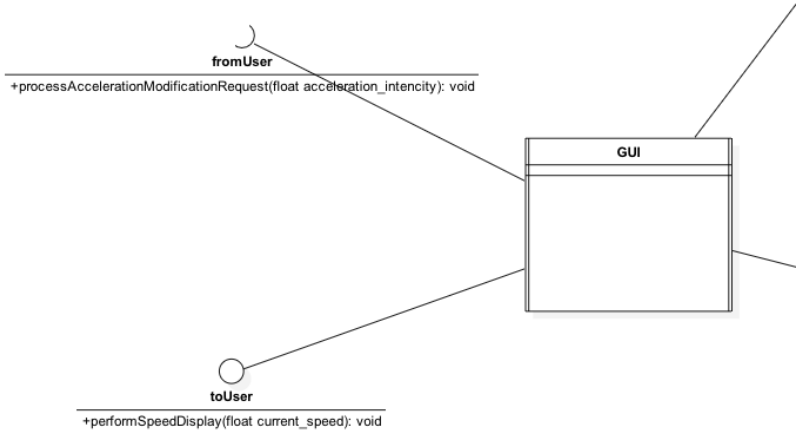


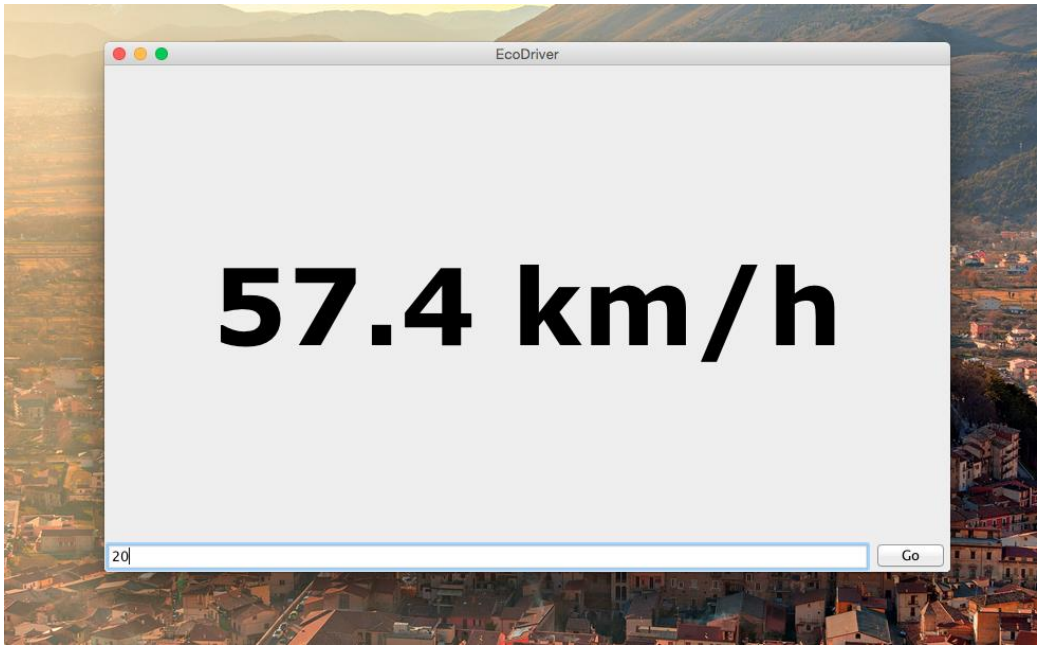
Un Controller pilote cette vitesse en jouant sur le vecteur accélération lorsque le conducteur appuie sur une pédale. Nous avons simplement utilisé ici le fait que l'accélération est la dérivé de la vitesse et donc la formule :

$$a = \frac{v_2 - v_1}{t_2 - t_1}$$

Cette accélération est mise à jour toutes les 500 ms et ne tient pas compte pour l'instant des forces de frottements. Cependant lorsqu'il n'y a ni freinage, ni accélération, on soustrait à cette vitesse ce qu'aurait perdu le conducteur idéal en décélérant à cette vitesse grâce aux valeurs du profil type générées par Matlab. Elles sont récupérées via la méthode `getSpeedLoss` et tiennent compte des forces de frottements et des ralentissements dus au frein moteur. Nous utiliserons les mêmes dans notre application Android pour rester cohérant.

Enfin pour permettre à l'utilisateur de contrôler cette vitesse une interface graphique est mise à sa disposition. Voici la représentation interne et externe de ce GUI :





Le champ texte que nous pouvons apercevoir sur la copie d'écran permet de spécifier la valeur avec laquelle le vecteur accélération va être multiplié. Cette valeur est un pourcentage, on peut donc donner des valeurs entre -100 et 0 pour un freinage et entre 0 et 100 pour une accélération. Si on donne au logiciel la valeur 0, la voiture décélère suivant le profil idéal. Pour conclure cette partie on peut ajouter que toutes ces classes sont structurées suivant une architecture MVC comme on peut le voir le diagramme de classe fourni en annexe.

Nous avons faire ce choix pour pouvoir faire évoluer le logiciel plus facilement et pour qu'il soit plus compréhensible par quelqu'un qui voudrait poursuivre notre travail.

V. L'application Android

1. Introduction

Dans cette partie, nous nous attacherons à décrire précisément le fonctionnement de l'application Android que nous avons développée. Nous commencerons par donner les détails de la communication avec l'Ecodriver, nous expliquerons comment nous l'avons implémentée en justifiant nos choix. Ensuite, on présentera les différentes fonctionnalités techniques de l'application.

2. Interactions avec l'Ecodriver

Communication: L'application Android et l'Ecodriver communiquent par socket TCP, l'application Android étant le client et l'Ecodriver le serveur. La classe `TCPClient` est détaillé dans le diagramme suivant:

Cette classe contient un constructeur `TCPClient` qui prend en argument une interface `OnMessageReceived` qui décrit l'action à effectuer. On lance le client TCP avec la méthode `run`. Le numéro de port ainsi que l'adresse IP du serveur sont définis en statique dans la classe. Il faut les écrire en dur, ce qui n'est

pas vraiment un problème puisqu'on travaille systématiquement sur réseau local (aucun problème de NAT ou autre).

Dans l'application, le client fonctionne en tâche de fond grâce au service Android `AsyncTask`.

Ce service permet de construire facilement des tâches de fond lancées directement depuis l'interface graphique.

Pour faciliter les choses, on lance la tâche asynchrone depuis l'activité principale de notre application. A l'appel de `onDestroy`, on arrête le client via la méthode `stopClient`.

Récupération et parsing du json : Comme vous l'avez vu précédemment nous utilisons notre propre format de routes. L'application Android récupère ce fichier et le convertit en classes java. Pour ce faire, nous utilisons la bibliothèque GSON, ce qui nous oblige à définir des classes java associées à chacun des objets json. Voici les différentes classes définies.

Ces classes sont des entités et ne contiennent donc que quelques méthodes simples (getters, setters, toString,...).

Nous avons considéré que le premier message reçu correspondait à la route au format json envoyée par l'Ecodriver. Voici le code associé à la conversion de json vers java :

```
premiere=false;
Gson gson = new Gson();
roadWrapper = gson.fromJson(json, RoadWrapper.class);
Log.v("json", roadWrapper.toString());
for (int i = 0; i < roadWrapper.getRoads().length; i++) {
    messageList.add(roadWrapper.getRoads()[i]);
}

Log.v("COPIE", "Copie terminee");

for (Road r : messageList) {
    Log.v("route", r.toString());
}
adapter.notifyDataSetChanged();
```

Une fois les différentes routes du fichier converties et stockées dans le `roadWrapper`, objet qui contient toutes les routes, on ajoute une à une les routes dans la liste `messageList`. Cette liste permet d'afficher les routes à l'utilisateur dans une `ListView`, afin que celui-ci puisse sélectionner sa route.

Bien entendu il s'agit ici simplement de tester le parsing du json, dans une implémentation réaliste, il est hors de question de recevoir l'intégralité des routes à chaque connexion. On peut plutôt imaginer que l'utilisateur tape sa destination et qu'on lui envoie en réponse uniquement les routes menant à une destination ressemblant à sa demande.

Demande périodique de la vitesse : Venons-en maintenant au dernier point de communication avec l'Ecodriver, la demande périodique de la vitesse. Pour simuler le fonctionnement réel de l'application, c'est à dire le coach embarqué. On a besoin de connaître constamment la vitesse instantanée du véhicule. On a simulé cela en intégrant à l'application une tâche demandant périodiquement à l'Ecodriver la valeur de la vitesse. Pour ce faire nous avons utilisé la classe `BestCountDownTimer` qui est fortement inspirée de la classe `CountDownTimer` fournie par android. Cette classe permet d'une part d'utiliser un compte à rebours en tâche de fond mais aussi de définir des actions à effectuer via la méthode `onTick`. Dans notre cas, on définit donc un compte à rebours avec une très grande valeur de décompte ainsi qu'une activation toutes les secondes pour la fonction `onTick`. Ceci est réalisé grâce au code ci-dessous :

```
countDownTimer=new BestCountDownTimer(Integer.MAX_VALUE,1000) {

    @Override
    public void onTick(long millisUntilFinished) {
        Log.v("Tache","t :"+millisUntilFinished);
        // Demande de mise a jour de la vitesse
        client.sendMessage("Donne moi la vitesse");
        // La mise a jour se fait automatiquement

        updateData();
        updateDisplay();


    }

    @Override
    public void onFinish() {
        Log.v("Tache", "Fini");
    }
}.start();
```

Pour conclure, notre application Android dispose donc :

- d'une structure de données représentant une route
- d'une connexion avec l'Ecodriver
- de la vitesse instantanée du véhicule

Nous allons maintenant présenter ses différentes fonctionnalités en détaillant leur implémentation.

3. Les fonctionnalités de l'application

Choix de routes : Le choix de la route est très important puisque tant que l'utilisateur n'a pas choisi, il n'a pas accès aux autres fonctions de l'application.

Au démarrage de l'application, l'utilisateur peut choisir de se connecter ce qui provoque deux choses. D'une part le client TCP établit la connexion avec l'Ecodriver, d'autre part la réponse json est parsée et convertie en une liste de routes java. C'est ensuite cette liste qui lui est affichée. Il sélectionne ensuite la route qu'il souhaite emprunter.

Evolution de la position : A chaque fois que la méthode : `onTick` est appelée on met à jour la position du véhicule avec la vitesse qu'on a reçu. Ceci permet de simuler le fait que le véhicule avance sur la route.

Intégration du profil : Les profils dont nous disposons nous permettent donc de connaître les temps de décélération pour toute vitesse comprise entre 50 et 0 km/h. C'est peu, mais suffisant pour effectuer nos tests de bon fonctionnement. On charge donc dans l'application le tableau suivant :

Vitesse (km/h)	50	30	0
Temps (s)	0	5.01	14.91
Distance (m)	0	56	103

Celui-ci indique le temps de décélération pour passer de 50 à une vitesse inférieure choisie. Ces temps sont tirés des profils, on notera bien que les profils étant des polynômes de degré 5, ils sont difficilement inversibles, par conséquent on utilise une recherche dichotomique pour trouver ces résultats. Ceci étant fait, nous pouvons remonter à la distance parcourue pendant la décélération, pour ce faire il suffit d'intégrer le polynôme entre les bornes que nous avons précédemment calculées ce qui nous donne la troisième ligne du tableau.

Les conseils : Notre algorithme a un fonctionnement prédictif, c'est à dire que le calcul est effectué une seule fois, avant le départ du véhicule. Voici son principe. Tout d'abord il faut parcourir la liste de segments à l'envers (en partant de la fin). Ensuite pour chaque segment, on compare sa vitesse avec du segment précédant (qui est en fait le segment suivant puisqu'on est à l'envers), si on identifie une décélération, on ajoute la distance de décélération à la liste finale reconstruite.

Lors du parcours, il suffira de regarder cette liste pour voir si le segment contient une telle distance.

L'envoi des conseils se fait par émission d'un son à ce moment précis, puis d'un deuxième son au moment où la vitesse doit être maintenue.

4. Conclusion

Dans cette partie nous avons démontré le fonctionnement de notre application dans des conditions que nous maîtrisons parfaitement puisque ce sont des conditions volontairement simplifiées. Nous ne pouvions donc pas terminer sans parler de ce qui pourrait éventuellement être fait pour que l'application soit utilisable dans un cas concret d'utilisation. Le premier point à aborder est l'utilisation d'un serveur réel pour envoyer des json adaptés. La difficulté ici comme nous l'avons vu précédemment viens du fait qu'il n'existe pas d'API gratuite permettant d'obtenir un itinéraire complet avec toutes les informations nécessaires sur les trajets à parcourir. De plus pour se débarrasser complétement de l'EcoDriver et passer dans un contexte réel. Nous devons récupérer la vitesse instantanée du véhicule grâce à l'appareil Android ce qui peut être fait assez simplement puisque la plupart des appareils disposent d'un gps. Il suffit donc de mesurer la position du véhicule à intervalle de temps proches les uns des autres et ainsi de déduire chaque distance et donc chaque vitesse instantanée.

Conclusion

L'éco-conduite est un sujet très complexe, passionnant et d'avenir. Retraçons les éléments majeurs de ce rapport qui nous ont permis l'élaboration de notre application d'écoconduite. Nous avons constaté dans un premier temps que le freinage est déterminant pour économiser du carburant contrairement à l'accélération qui a un impact plus faible. Nous avons donc concentré nos efforts sur l'optimisation des freinages pour un conducteur. À partir des données provenant d'un expert de l'écoconduite nous avons pu extraire un profil de conduite idéal : ce profil constitue une base solide de notre algorithme. Nous avons ensuite évoqué les difficultés que nous avons rencontrées et les simplifications que nous avons faites. Enfin, nous vous avons présenté EcoDriver, notre application dont le but est de simuler l'environnement extérieur à l'application Android.

Bien entendu, le projet tel que nous l'avons construit est loin de constituer une solution grand public permettant de servir le domaine de l'écoconduite. Néanmoins, il aura permis plusieurs choses. D'une part, celui-ci constitue un bon point de départ pour la réalisation d'une application grand publique. En effet, les algorithmes les plus compliqués ont été implémentés et testés. De plus, l'utilisation de deux applications séparées rend le code facilement réutilisable. D'autre part nous avons effectué un vrai travail de recherche en ce qui concerne les API de cartes géographique.

Pour conclure, nous pouvons ajouter que projet tutoré nous aura permis de nous familiariser avec l'écoconduite, une problématique importante pour notre époque autant en tant qu'ingénieur qu'en tant que citoyen.

Bibliographie

- Regression Polynomiale *Wikipedia* **[En ligne]** Consulté le 21/05/15
  http://fr.wikipedia.org/wiki/Régression_polynomiale
- Distance d'arrêt *Wikipedia* **[En ligne]** Consulté le 21/05/15
  http://fr.wikipedia.org/wiki/Distance_d%27arrêt
- Transfert de charge *Wikipedia* **[En ligne]** Consulté le 21/05/15
  http://fr.wikipedia.org/wiki/Transfert_de_charge_%28automobile%29
- Ajustement de courbe *Wikipedia* **[En ligne]** Consulté le 21/05/15
  http://fr.wikipedia.org/wiki/Ajustement_de_courbe
- Programmatic fitting *Mathworks* **[En ligne]** Consulté le 22/05/15
  http://fr.mathworks.com/help/matlab/data_analysis/programmatic-fitting.html
- Polyval *Mathworks* **[En ligne]** Consulté le 22/05/15
  http://fr.mathworks.com/help/matlab/ref/polyval.html?searchHighlight=polyval
- Facteurs influençant le coefficient de frottement transversal *Belgian Road Research Centre* **[En ligne]** Consulté le 15/05/15
  http://www.brrc.be/publications/f/f3058.pdf
- Guide de Formation à l'éco conduite *ADEME* **[En ligne]** Consulté le 22/05/15
http://www.ademe.fr/sites/default/files/assets/documents/66885_guide_ecocond uite.pdf Consulté le 15/05/15
- Centre d'aide google Earth **[En ligne]** Consulté le 25/05/15
  https://support.google.com
- Google Developers **[En ligne]** Consulté le 25/05/15
  https://developers.google.com/
- Wiki OpenStreetMap **[En ligne]** Consulté le 25/05/15
  http://wiki.openstreetmap.org/

Annexes

A. Paramètres influant sur la force de frottement

B. Script Matlab et profils

C. Format des logs

D. Diagramme de classe de l'EcoDriver

E. Format du fichier json

Liste des paramètres influant sur la force de frottements.

- Angle imposé à la roue
- Pression de gonflage du pneu de mesure
- Poids sur la roue de mesure
- Revêtement sec, mouillé ou souillé
- Vitesse de mesure
- Nature et composition du revêtement béton ou revêtement à base de liants hydrocarbonés
- Température
- Polissage et écrasement des matériaux

Script Matlab et Profils

```
% Nettoyage du workspace
clc
clear all
clf

% Importation des données
M1 = csvread('data.csv',1,1);
M2 = csvread('data2.csv',1,1);

% Récupération des données
intéressantes temps et vitesse
t1 = M1(:,2);t2 = M2(:,2); % en
secondes
v1= M1(:,6);v2= M2(:,6); % en m/s

v1=v1*3,6;v2=v2*3,6; % on convertit
les m/s en km/h

figure(1);

subplot(1,2,1), plot(t1,v1);
title('2014-02-24 18h17')
xlabel('temps en secondes') % x-
axis label
ylabel('vitesse en km/h') % y-axis
label
subplot(1,2,2), plot(t2,v2);
title('2014-02-24 18h37')
xlabel('temps en secondes') % x-
axis label
ylabel('vitesse en km/h') % y-axis
label

figure(2);

 tzoom = t1(725:835);
 vzoom = v1(725:835);

 plot(tzoom,vzoom,'.');
 xlabel('temps en secondes') % x-
axis label
ylabel('vitesse en km/h') % y-axis
label

figure(3);

tzoom3 = t2(3066:3170);
vzoom3 = v2(3066:3170);
plot(tzoom3,vzoom3,'.');
xlabel('temps en secondes') % x-
axis label
ylabel('vitesse en km/h') % y-axis
label
figure(4);
    plot(tzoom,vzoom,'.')
    hold on
    pp=polyfit(tzoom,vzoom,5);
    approx=polyval(pp,tzoom);
    plot(tzoom,approx,'r')
    xlabel('temps en secondes') %
x-axis label
    ylabel('vitesse en km/h') % y-
axis label
    title('Profil d''arret')
figure(5)
    plot(tzoom3,vzoom3,'.')
    hold on
    pp3=polyfit(tzoom3,vzoom3,5);
    approx3=polyval(pp3,tzoom3);
    plot(tzoom3,approx3,'r')
     xlabel('temps en secondes') %
x-axis label
ylabel('vitesse en km/h') % y-axis
label
title('Profil de ralentissement')
syms t x
 profil1 = poly2sym(pp,t)
 profil2 = poly2sym(pp3,t)
```

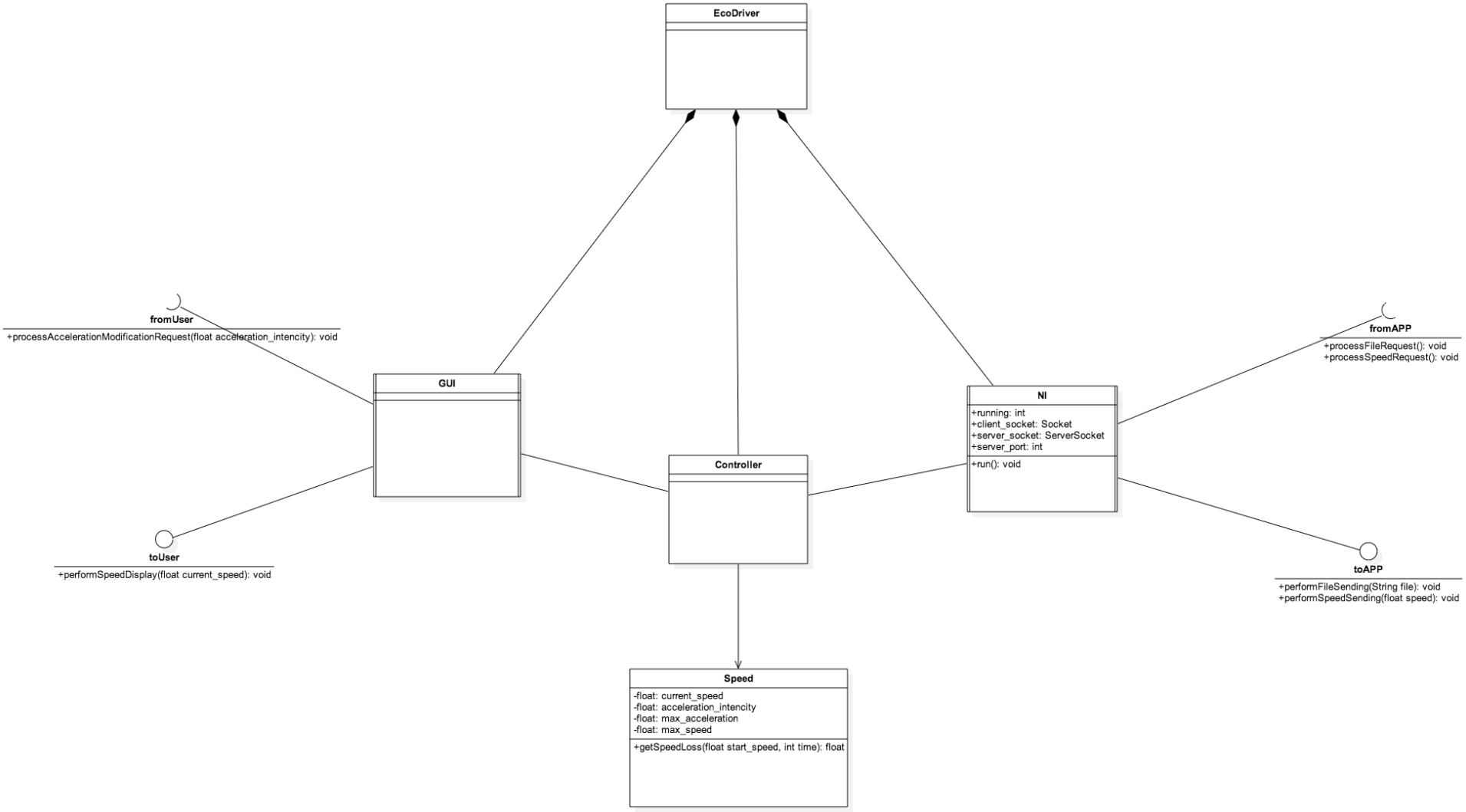
```
Profil1 =
-0,00142049788062601     0,564414022224674   -89,6054535552631
     7104,77311732288    -281345,854765376    4451415,62877784

Profil2 =
-0,00140592861694294     2,18742320301397    -1361,28602643003
     423567,582575140    -65895010,3251713    4100422211,95420
```

Format des logs

Premières lignes du fichier csv.

time;timestamp;recordtime;lat;long;alt;speed;course;verticalAccuracy;horizontalAccuracy;locTimeStamp;accelerationX;accelerationY;accelerationZ;HeadingX;HeadingY;HeadingZ;TrueHeading;MagneticHeading;HeadingAccuracy;RotationX;RotationY;RotationZ;motionYaw;motionRoll;motionPitch;motionRotationRateX;motionRotationRateY;motionRotationRateZ;motionUserAccelerationX;motionUserAccelerationY;motionUserAccelerationZ;DeviceOrientation
2014-02-24 18:17:01.814;1;0.1;43.61254;3.912179;24.9822;3.527539;171.0768;9;5;1393262220;0.1624451;-0.393219;-0.9277954;-75.68192;-121.0272;7.307053;134.9848;134.2765;25;-0.03307406;0.01997424;-0.02162886;0.04975956;0.1873391;0.4956875;-0.04867782;0.09143406;-0.07113435;0.03090357;0.03079704;-0.06474977;1
2014-02-24 18:17:01.919;2;0.2;43.61251;3.912186;24.62286;3.137119;171.3232;9;5;1393262221;0.1526489;-0.3698273;-0.8959045;-75.02324;-121.8938;8.547379;135.6398;134.9316;25;-0.001269529;-0.01582051;0.007180415;0.04920786;0.1883696;0.4946157;-0.04773393;0.01021542;-0.02340836;0.02669811;0.04280869;-0.07722091;1
2014-02-24 18:17:02.041;3;0.3;43.61251;3.912186;24.62286;3.137119;171.3232;9;5;1393262221;0.1516724;-0.3737183;-0.8851624;-75.68192;-122.4355;10.12598;136.7836;136.0753;25;0.104222;-0.005625395;0.04680666;0.04961919;0.1875775;0.4913193;-0.01601472;-0.02559091;0.005389;-0.001975933;0.07857053;-0.06155042;1

Diagramme de classe de l'EcoDriver



D

Format du fichier json


```
{
   "roads":[
      {
         "name":"Route de test 1",
         "length_unit":"m",
         "max_speed_unit":"km/h",
         "segments":[
            {
               "max_speed":"30",
               "length":"2000"
            },
            {
               "max_speed":"0",
               "length":"0"
            }
         ]
      },
      {
         "name":"Route de test 2",
         "length_unit":"m",
         "max_speed_unit":"km/h",
         "segments":[
            {
               "max_speed":"50",
               "length":"200"
            },
            {
               "max_speed":"30",
               "length":"100"
            },
            {
               "max_speed":"0",
               "length":"0"
            }
         ]
      }

   ]
}
```