Evan BLONDEAU
Technologie de l'information option internet des objets, sécurité et ville intelligente
ESIR2

Johann BOURCIER
Responsable de la spécialité Technologies de l'Information

Plateforme MobBi
Campus de Beaulieu, Batiment 6, 263 av. général leclerc, cs74205
35042 Rennes, France
02 23 23 68 22

Ghislain NOUVEL
Ingénieur animateur de la plateforme

Développement d'un démonstrateur OBDII vers différents supports tels qu'une application sous Android ainsi qu'un casque Hololens

Année universitaire 2018-2019

Remerciements

Tout d’abord, je voudrais remercier mon tuteur de stage Mr Ghislain Nouvel pour m’avoir accueilli à la plateforme MobBi. Ainsi que pour m’avoir guidé, informé durant ce stage et fait découvrir des choses entre autre à travers le casque Hololens ainsi que pour m’avoir donné la possibilité de le tester et de développer dessus.

Ensuite, je voudrais aussi remercier l’école supérieur des ingénieurs de Rennes ainsi que mon tuteur, pour m’avoir donné la possibilité de réalisé un stage dans ma spécialité, ce qui m’a permis d'acquérir énormément de connaissances sur le plan technique mais aussi relationnel.

De plus, je voudrais aussi remercier David Levalois et Régis qui m’ont accueilli dans le bâtiment 6. Ainsi que d’avoir toujours été présents quand nous avions des questions ou des problèmes au niveau du projet ou de l'électronique.

Enfin, je voudrais aussi remercier les autres stagiaires présents durant mon stage pour leur soutien et les échanges constructifs que nous avons pu avoir sur le sujet, nos sujets étant étroitement corrélés.

1 Introduction

La voiture électrique prend de nos jours de plus en plus de place dans le parc automobile mondiale. Selon l'agence internationale de l'énergie, près de 3,1 millions étaient en circulation dans le monde à fin 2017. Et ces chiffres vous ne faire que grandir. Mais cela à un coups financier important et on sait tous qu'une voiture traditionnelle n'est pas très ouverte à nous dire comment elle est faite à l'intérieur.

Dans le cadre de ma quatrième année à l'école supérieur des ingénieurs de Rennes en spécialité technologie de l'information option IoT, sécurité et ville intelligente, j'ai souhaité réaliser mon stage de 4ème année durant 2 mois à la plateforme MobBi qui innove et accompagne la recherche et le développement dans divers domaines. La mission qui m'a été proposée à la plateforme m'a tout de suite plu et questionné car elle s'ancre dans un projet innovant, ambitieux et open source au niveau de l'automobile. C'est un domaine qui m'intéresse énormément car il manie plusieurs disciplines tels que l'informatique embarquée, l'électronique et le réseau entre autres qui me correspondent.

De plus cette mission a aussi une partie avec un casque de réalité mixte, le casque hololens, ce qui ouvre encore plus ma curiosité à son sujet. J'ai pour volonté d'acquérir beaucoup de connaissances dans divers langages de programmation car je sais qu'il faudrait en manier plusieurs pour différentes utilisations, de renouer avec l'électronique car c'est un sujet qui s'y prête, ainsi que d'améliorer mes connaissances et ma pratique dans ce domaine.

Enfin, dans un premier temps nous décrirons la plateforme MobBi et son secteur d'activité. Puis nous étudierons ma mission ainsi que son implantation dans le projet de l'OSV, puis ce que j'ai réalisé lors de ce stage avant de dresser un bilan de celui-ci avec mon retour d'expérience.

2 Présentation de la société et de la mission du stage

a La plateforme MobBI: Systèmes Embarqués pour la Mobilité et les Bâtiments Intelligents

MobBI est une plateforme technologique qui dispose de moyens matériels et humains au service des entreprises pour favoriser et accompagner leurs projets en matière de R&D et d'Innovation, et ainsi permettre leur développement économique. Cette structure d'appui peut aussi bien intervenir dans le cadre de projets "individuels" que "collaboratifs".

Ils ont comme objectif d'activer le développement économique des entreprises par l'innovation, via un transfert de connaissances universitaires en matière technologique en intégrant des problématiques environnementales.

Depuis 2017, la plateforme est labellisée PFT (plateforme technologique). "Les plateforme technologique sont des dispositifs proches des entreprises, bien ancrés dans le tissu régional, constituent un relai entre les laboratoires de recherche et le monde des entreprises."

Image 1: Label PFT

Cette plateforme a été crée en fin d'année 2012:

- Issue du transfert de la plateforme OPASS (Ouest Plateforme Automobile Systèmes Embarqués), anciennement hébergée par la MEITO, à laquelle s'est greffée la plateforme HID (Habitat Intelligent et Développement Durable) de l'Université de Rennes 1.

Voilà comment on peut schématiser les actions de la plateforme MobBi :

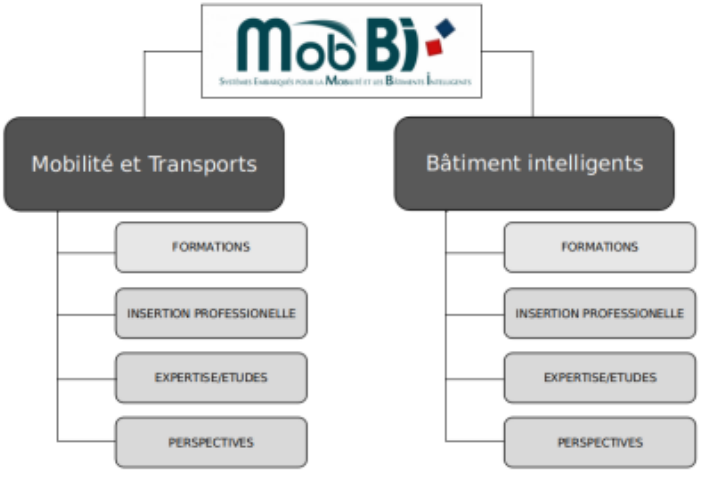


Image 2: Schéma de la plateforme MobBI

Concrètement, la plateforme MobBi c'est :

- Un animateur : Ghislain Nouvel
- Une localisation : Université de Rennes 1 - Campus de Beaulieu Bâtiment 6
- Deux laboratoires d'adossement: l'IETR et l'IRISA
- De multiples financeurs tel que le conseil régional de Bretagne, le conseil général d'Ille et Vilaine, Rennes Métropole etc.

b. La mission du stage

Le but principal de ma mission était de faire communiquer l'elm327 ou OBD(on-board diagnostic) qui est un appareil de diagnostic automobile disposant d'un modèle bluetooth avec un casque hololens. Donc de récupérer et d'afficher les données du véhicule en minimisant la latence tel que la vitesse, les tour par minutes etc... Pour ce faire, il faut programmer en C# sur unity. Cela a pour objet de donner une autre approche au niveau du conducteur et de l'affichage tête haute, car nous aurons directement les informations dans le champ de vision du conducteur.

Image 3: Communication entre l'OSV et Hololens

Par la suite, nous avons décidé qu'il était aussi intéressant de réaliser une application mobile en JAVA pour android qui avait les mêmes fonctionnalités que l'application sur Hololens (communication bluetooth, affichage etc...). Cependant, celui ci va avoir un serveur qui permet de partager les informations capturées avec d'autres supports tels qu'un site web ou un casque hololens. Il fallait donc être le plus générique possible afin d'améliorer plus facilement l'application plus tard.

Image 4 : Communication entre l'application et l'OSV ainsi que les autres supports

Enfin, j'ai aussi été chargé de réaliser la partie qui devait répondre à l'appareil de diagnostic dans la partie voiture. Et cette partie devait être réalisée en C++ ainsi qu'à l'aide d'un oscilloscope numérique

3 Analyse du problème ainsi que de son contexte

a Mise en contexte

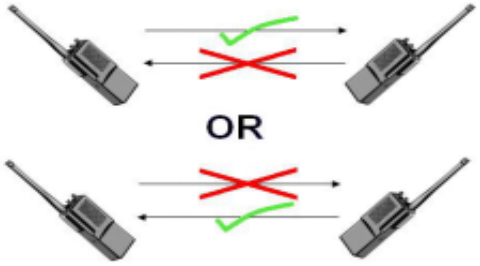
Pour mieux vous expliquer ma mission durant ce stage, il faut d'abord remettre tout cela dans son contexte car il y a beaucoup de choses à prendre en compte.

Ma mission s'écrit à la suite de plusieurs mois/années de recherche pour que l'université de rennes 1 réalise un véhicule open source (que l'on nommera OSV par la suite), c'est à dire libre de droit et modifiable par qui le veut. Et surtout d'avoir à disposition les plans, les différents programmes, les matériaux utilisés etc.. Ce qui n'est absolument pas le cas aujourd'hui. Toute voiture, plan, programme et électronique utilisé dedans est protégé par ceux qui la produise. Vous pouvez voir une image de l'OSV ci-dessous.

Image 5: Le véhicule open source(OSV)

Ensuite, à mon arrivée, il y avait un groupe d'étudiants en Licence automobile sur Rennes qui avait comme projet de réaliser la partie électronique et informatique du véhicule. Donc ils devaient s'occuper de la conception du tableau de bord ainsi que les feux, les différents radars de recul et bien sûr que toutes ces parties communiquent comme dans un vrai véhicule.

Le véhicule communique avec des réseaux CAN (Controller Area Network) qui sont des bus de données série bidirectionnel half-duplex (dans notre cas) qui permet une communication rapide et robuste face aux perturbations.



Imag 6: Explication hal-duplex bidirectionnel

Le Schéma ci dessous représente comment ces élèves ont conçu les réseau CAN (3 ici) et les différentes parties de la voiture. Cette partie a été reprise, modifiée et “sécurisée” par un autre stagiaire, Antoine Barthélémy, qui a repris tous les travaux des élèves et les a modifiés, corrigés, câblés et soudés pour que la plus grosse partie fonctionne correctement. Pour que par la suite d’autres élèves dans les années futures puissent reprendre et poursuivre le projet avec une base solide, fonctionnelle et testée.

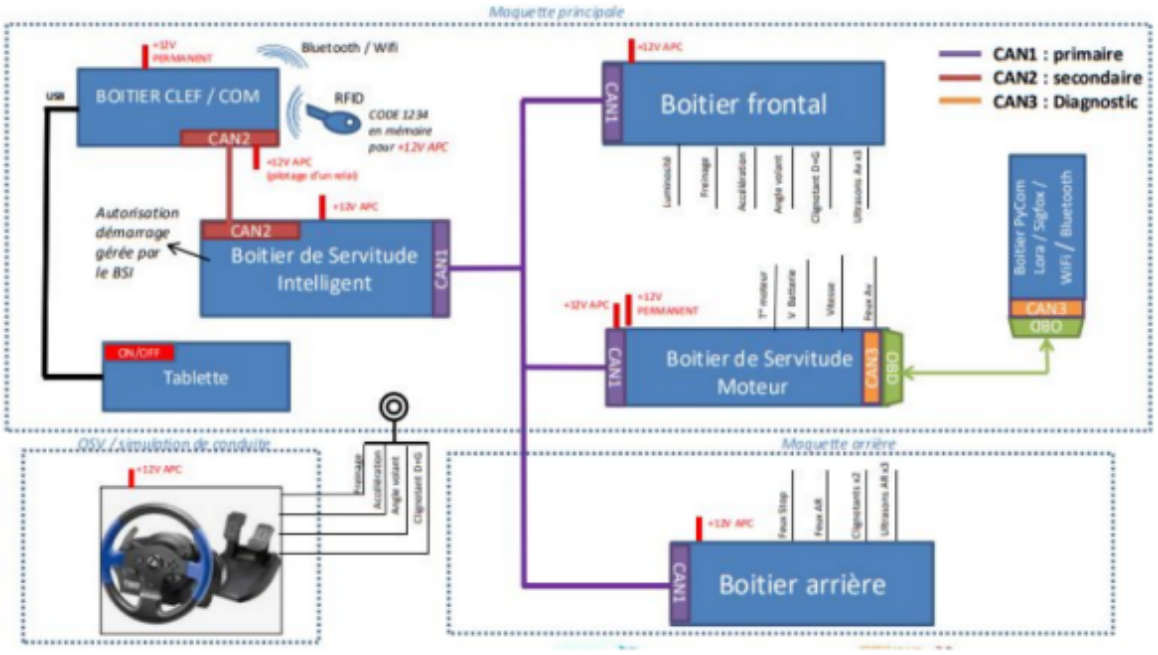


Image 7: Schéma de la maquette

Ma partie quant à elle se situe au niveau du carré rouge, au niveau de la prise OBD (On-Board Diagnostics) qui est la prise de diagnostic utilisée par les garagistes entre autre pour connaître les potentiels problèmes de votre voiture. Cette partie fait le lien entre l’intérieur de la voiture et l’extérieur. Pour ce faire, j’avais comme contrainte un module; un Elm327 qui est une prise de diagnostic open source que l’on peut trouver sur leur site etc. Donc au début, j’avais un de ces petits modules qui avait aussi un élément bluetooth donc qui va nous permettre de nous y appairer et avoir les données etc.

b Etude de l'existant

Au niveau du casque Hololens, il y avait déjà une seule application qui existait pour communiquer avec l'ELM327 (par une connection bluetooth) avec une version gratuite et une version payante. Elle ne permettait pas de faire tous les paramètres que l'on voulait dessus mais cela fonctionnait bien.

Au niveau des applications sur smartphone, il y en a beaucoup mais la plupart ont une version payante et aucune n'est open source. Ensuite, je n'ai trouvé aucune application qui proposait notre solution, c'est à dire de passer par une application pour ensuite donner la possibilité à d'autres systèmes de récupérer les informations de la voiture par le wifi de l'appareil.

4 Travail réalisé / résultats développés

La première partie de mon travail consistait à communiquer en Bluetooth avec un ELM327 qui est un microcontrôleur qui permet de traduire et requêter les commandes OBD sur un véhicule. Pour ce faire, j'ai réalisé une application Android développée en JAVA avec Android Studio.

a Connection Bluetooth sur Android

Donc la premiere partie de cette application consistait à se connecter en bluetooth à des appareils puis par la suite à l'ELM327 car je ne l'avais pas les premiers jours. La première tâche consistait à analyser le réseau, regarder les appareils disponibles autour de soi ainsi que les appareils déjà appairés avec l'appareil. Cette partie était assez simple dans l'ensemble à réaliser. Cela donne l'image ci-dessous.

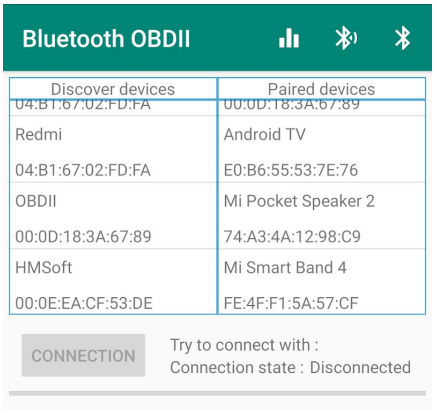


Image 8: Découverte bluetooth

Ce qui m'a rendu la tâche plus compliquée, comme je l'ai dit auparavant, c'est que je ne disposais pas de l'elm327 au début et les appareils avec lesquels je testais mes programmes étaient ma manette de PS4 ou encore une enceinte bluetooth et je n'arrivais jamais ou presque à me connecter à eux en ouvrant la socket et en gardant une connexion persistante. Dans le contexte d'un logiciel, une socket peut être assimilée à un connecteur réseau ou encore à un interface de connexion. Ici on utilise une socket bluetooth pour communiquer entre deux appareils en bluetooth.

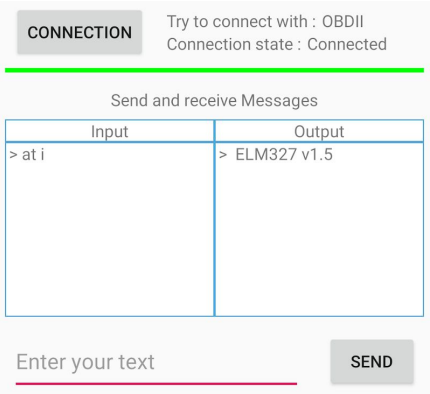


Image 9: Connection à l'Elm327

Dans ce cas, j'étais vraiment bloqué, rien de fonctionnait, j'ai même réalisé des programmes en c# pour essayer de voir si cela venait de là mais rien de fonctionnait non plus. Puis quand j'ai eu le l'ELM327, tout à fonctionné directement aussi bien sur la partie android que c#. De plus, par la suite j'ai réalisé une classe pour que la connection soit persistante à travers les activités, ce qui facilite un développement futur de l'application.

b Communication avec l'ELM327

i Présentation général de l'elm327:

Comme je le disais auparavant, l'elm327 est un microcontrôleur programmé qui est produit par ELM Electronics qui traduit l'interface OBD. Comme nous pouvons le voir ci-dessous, ce microcontrôleur peut répondre à plusieurs types de protocoles de communication automobile. Par exemple, SAE-J1939 est utilisé par les constructeurs européens ou encore PWM, principalement utilisé par Ford.

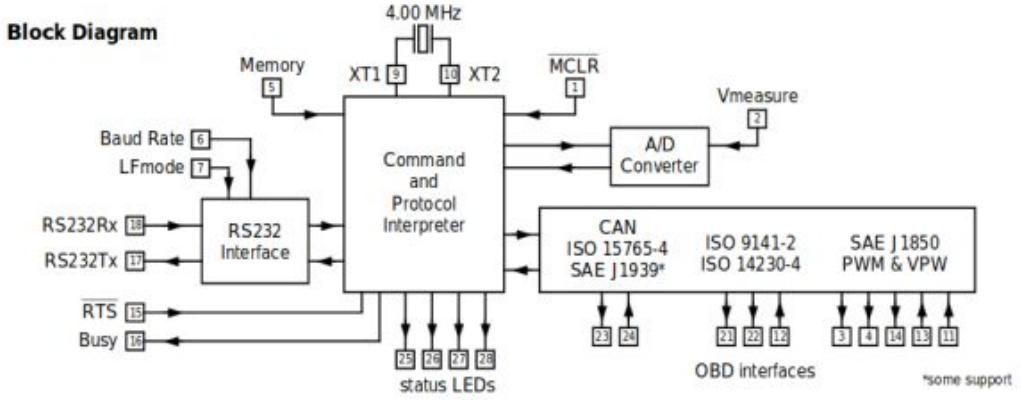


Image 10: Schéma bloc de l'Elm327

A terme, le CAN (ISO 15765) est celui qui sera utilisé par tous les constructeurs et c'est celui que nous utiliserons dans l'OSV. Mais il faut pouvoir communiquer avec la voiture, même sur des modèles un peu ancien et c'est là que ce microcontrôleur rentre en jeu car il détecte quel protocole est utilisé et fait le nécessaire pour pouvoir communiquer.

ii La communication entre l'application Android et l'ELM327

1 Commande AT:

Comme on peut le voir sur la figure n°10 il y a une interface RS232 qui est une norme de standardisation de communication série et avant cela nous avons notre

module bluetooth. Donc quand notre connection bluetooth est faite, nous envoyons des messages et le module l'envoie sur le port série pour communiquer avec le microcontrôleur.

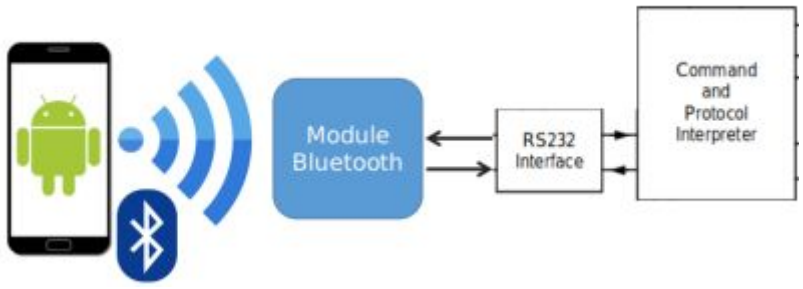


Image 11: représentation du parcours des commandes AT

Et donc nous passons par des commandes AT. Chaque commande est envoyée sous la forme d'une ligne de texte encodée en ASCII, terminée par le caractère "\r". Toutes les commandes sont disponibles dans la datasheet du microcontrôleur. Ces commandes servent dans la plupart des cas à configurer celui-ci et à vérifier qu'il fonctionne. Par exemple "AT I" sert à connaitre sa version d'ID. Ou encore "AT DP" sert à connaître quel protocole est actuellement utilisé par l'OBD.

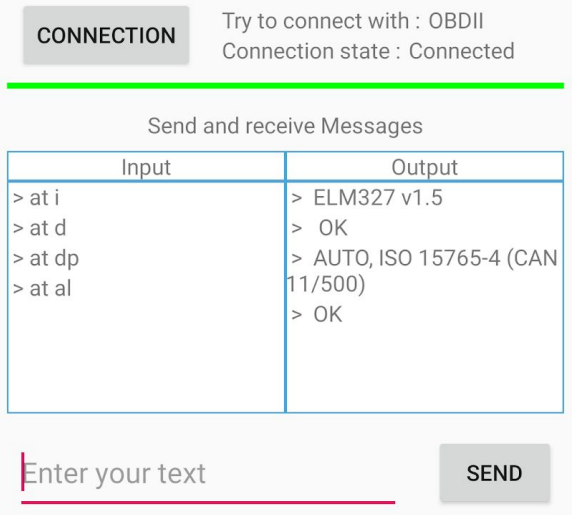


Image 12: Commande AT sur l'ELM327

2 Commande OBD:

Si le messages que nous avons envoyé depuis le téléphone ne commence pas par les caractères "A" et "T", l'elm327 les interprète comme devant être des commandes OBD qui seront destinées au véhicule.

01 - show current data
02 - show freeze frame data
03 - show diagnostic trouble codes
04 - clear trouble codes and stored values
05 - test results, oxygen sensors
06 - test results, non-continuously monitored
07 - show 'pending' trouble codes
08 - special control mode
09 - request vehicle information
0A - request permanent trouble codes

Image 13: Modes de diagnostic

Il fera donc le nécessaire pour les envoyer au véhicule sous le format adapté etc.. (nous expliquerons cette partie par la suite dans la compréhension du réseau CAN). Donc en regardant toujours dans la documentation, en commençant par les suites de nombre ci-contre, nous pouvons requêter la voiture sur différents points.

Donc logiquement si l'on veut par exemple la vitesse à laquelle roule l'utilisateur, nous commençons par "01" suivi du code PID de la vitesse que nous pouvons trouver dans la norme OBD qui est "OC". Donc si l'on envoie "O1 OC" on recevra la vitesse du véhicule, et peu importe son modèle, le constructeur etc.. ils doivent respecter la norme OBD-II.

Pour l'exemple, je l'ai testé sur un Dacia Duster et nous pouvons voir que nous recevons "41 0C ..", le 41 veut dire qu'il répond à une requête 01, et le 0C, c'est les tours par minutes du moteur. Puis l'autres Hexa correspond à la vitesse, certaines valeurs obtenues peuvent ensuite être rentrées dans une formule pour avoir le resultat final. Ici nous l'avons fait avec la vitesse mais c'est exactement la même chose si l'on veut avoir les codes d'erreur qu'il y a sur un véhicule et c'est ce qui est le plus utilisé quand nous utilisons ce genre d'application. De plus, on peut envoyer plusieurs requêtes en même temps comme la vitesse, les rpm etc..

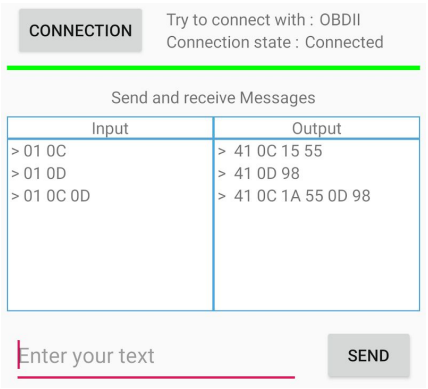


Image 14: Test des commandes OBD sur un Dacia Duster

c Traitement des données dans l'application

Comme nous pouvons le voir avec les parties précédentes, un utilisateur lambda ne va pas s'amuser à chercher chaque code PID et la formule etc. Donc c'est pourquoi nous avons voulu créer une interface pour que ça soit plus explicite avec une sorte de tableau de bord à l'aide de jauges ou encore des zones de texte.

i Fichier JSON

Pour que l'on n'ai pas à écrire tous les codes PID dans l'application JAVA, nous avons décidé de faire un fichier JSON qui regroupe tous les PIDs et toute autre information sur ceux-ci qui est nécessaire au bon fonctionnement de l'application. Un fichier JSON ou JavaScript Object Notation est un format de données. Pour le dire d'une autre manière, c'est une façon de stocker des informations, un peu comme une base de données. Voilà à quoi ressemble un code PID dans un fichier JSON

Image 15: Exemple d'un PID dans le fichier JSON

Comme ça par la suite, si la norme évolue avec de nouveaux codes, la personne qui doit l'implémenter n'a pas besoin de savoir coder en JAVA pour compléter la liste des codes PIDs.

ii Affichage à l'aide de jauge et de texte.

Une partie intéressante de l'application c'est que nous pouvons afficher les résultats sous forme de jauge ou de texte au désir de l'utilisateur. Nous avons mis en place des jauges avec une désignation fixe comme les RPM ou la vitesse, comme on peut le voir sur l'image ci-dessous. Cela a été fait avec la libraire de Pygmalion69, disponible sur Github. Le soucis de cette librairie c'est qu'on ne peut pas changer la vitesse de l'aiguille et comme sur les rpm, cela change très rapidement, il y a un petit décalage.

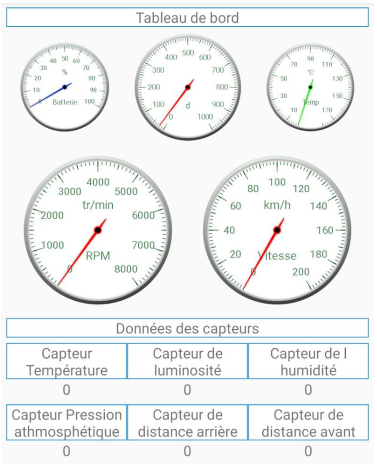


Image 16 : Le tableau de bord

Mais aussi avec des tableaux de données avec des champs de texte où l'utilisateur peut décider d'afficher les données qui l'intéressent. Et cela se passe avec l'ajout d'un champ de texte avec le button "+".

Ensuite, il peut choisir parmis tous les PIDs disponibles et cela lui crée dynamiquement le champ sur la page précédente et ajoute la requête à la liste des requêtes à envoyer au véhicule.

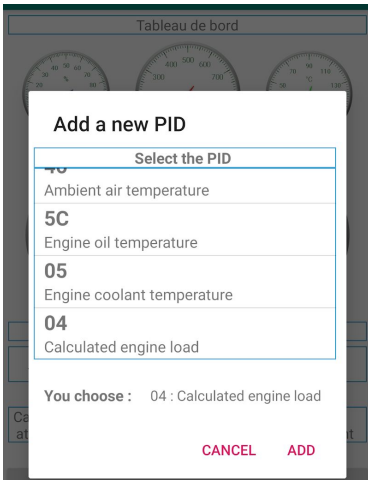


Image 17: Ajouter un PID

iii Partie algorithmique

La plus grosse partie de l'application était de réaliser tous les algorithmes pour que tout ce que j'ai présenté fonctionne. Car la plupart des choses à faire était de faire du traitement de chaînes de caractères, avec des paramètre à prendre en compte comme par exemple s'ils ont besoin de formule pour avoir le résultat final, savoir si c'était une requête multiple car si c'est le cas il y avait un autre traitement etc.. Puis ensuite, il faut aussi un algorithme pour savoir où envoyer cette valeur et l'afficher, comme tout peut être modulé à volonté ou presque. Donc tout est stocké dans des listes, ce qui veut dire qu'il faut les parcourir, trouver la cible et envoyer la valeur.

De plus, la gestion des erreurs était importante, car il ne faut pas que l'application se ferme si il n'y a pas eu le bon message ou que la connection bluetooth soit perdue. Donc il y a des paramètres à prendre en compte et des exceptions à mettre en place.

d Réalisation de la partie véhicule qui doit répondre à l'ELM327

Par la suite, on m'a demandé de réaliser la partie en amont de la prise OBD, celle qui doit répondre à l'elm327. Ce qu'il faut savoir, c'est que la communication entre le microcontrôleur et le module qui lui répond dans l'OSV s'effectue par le réseau CAN.

i Compréhension du réseau CAN

Comme je vous l'avais présenté avant, le CAN est un bus de données série bidirectionnel half-duplex qui a un faible coût avec un média robuste et une détection d'erreurs performante (CSMA-CR) avec un transfert temps réel des informations entre les noeuds. Dans sa typologie, il n'y a pas de relation spécifique entre les noeuds, quand quelqu'un envoie un message, tous les noeuds le reçoivent.

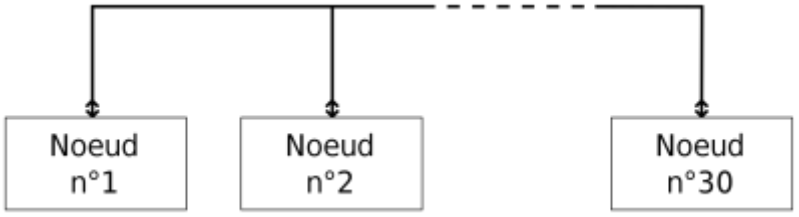


image 18: exemple de typologie d'un réseau CAN

Ci-dessous, nous pouvons sommairement vous montrez comment est constitué une trame CAN et surtout les parties qui nous intéressent par la suite.

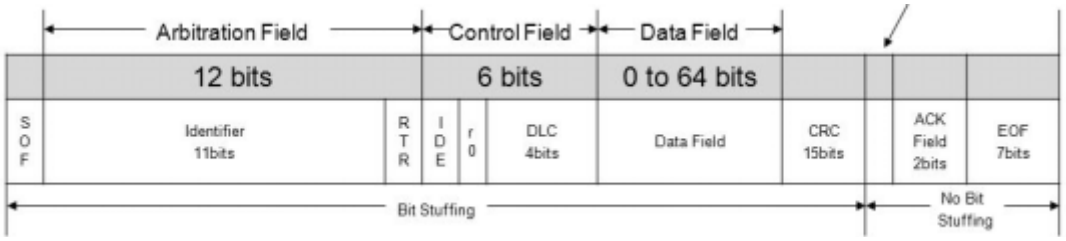


Image 19: Schéma d'une trame CAN

Il y a le champ d'identification qui est sur 11 bits, c'est la première partie que l'on traite quand on reçoit un message CAN car on sait directement s'il nous est destiné ou pas. Comme on parle avec tout le monde en CAN, ce champ permet de cibler à qui parler car seule la cible pourra traiter l'identificateur qu'on lui envoie. Ensuite, il faut savoir que le champ de données, donc ce qui est vraiment intéressant pour nous, ne peut pas dépasser 8 octets en CAN.

ii Réalisation

Pour réaliser cette partie, j'ai utilisé la carte arduino mega ainsi qu'un shield CAN, cette carte était aussi utilisée pour l'éclairage des feux de la voiture etc.. Elle a été réalisée par un autre stagiaire donc je ne suis pas partie de rien sur cette partie.

Ensuite, j'ai remarqué qu'il n'y avait presque aucune information sur la manière dont les constructeurs répondaient aux trames OBD, savoir si il y avait un ID spécifique, ce qu'il fallait prévoir dans la trame, comment répondre à une requête multiple car on peut demander jusqu'à 6 informations en même temps, avec le champ de la commande OBD(ex: "01") donc on est déjà au maximum d'une trame CAN. Donc comment faire avec les réponses en plus? Donc nous avons mis en place un oscilloscope numérique qui décode les trames CAN, pour comprendre son fonctionnement.

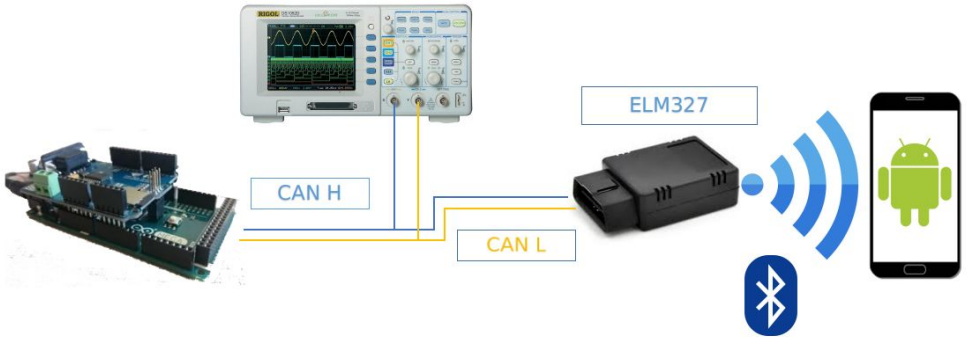


Image 20:Montage réalisé pour décoder le CAN

Donc à l'envoi d'une trame à partir de l'application, on voit que l'ELM327 envoie à l'OSV une trame CAN avec comme ID 7DF, comme il était indiqué dans la documentation par défaut (on peut choisir de 7E8 à 7DF). Ensuite, on remarque que le premier hexadécimal est le nombre d'octets à traiter (ici 2) pour le calculateur et ensuite, c'est exactement ce que l'on a envoyé de l'application "01 0C" pour le rpm.

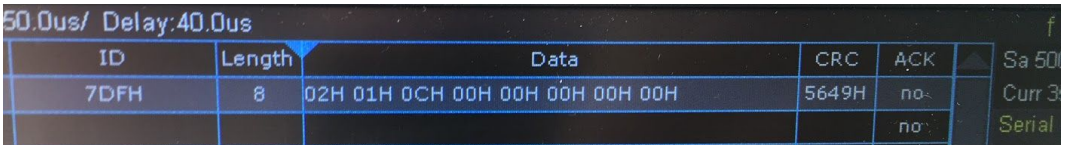


Image 21: Trame CAN d'envoie capturé à l'oscilloscope

Au niveau de l'arduino, tout se fait en C++. Ce qui était le plus délicat c'est qu'il faut faire de la manipulation de chaine de caractère en C++. Donc il faut remplir une chaîne avec à peu près les mêmes choses qu'à l'envoi, c'est à dire la taille des données à traiter, le service ici "41" car on répond à une requête "01" et les PID avec leur réponse. Donc la réponse est du type "04 41 0C XX XX" avec comme ID de trame "7E8" sinon c'est identique à l'envoi. (pour afficher l'ID etc, c'est une commande AT : "AT H0" ou "AT H1")

> 01 0C > at h1 > 01 0C	> 41 0C 15 55 > OK > 7E8 04 41 0C 1A 55

Image 22: réponse "O1 OC" avec l'Arduino Mega

Au niveau des requêtes multiples quand la réponse fait plus de 8 octets, on doit modifier la trame pour dire que l'on doit envoyer d'autres trames par la suite et donc il ne faut pas que l'Elm327 en envoie lui aussi sinon il y aura des conflits..

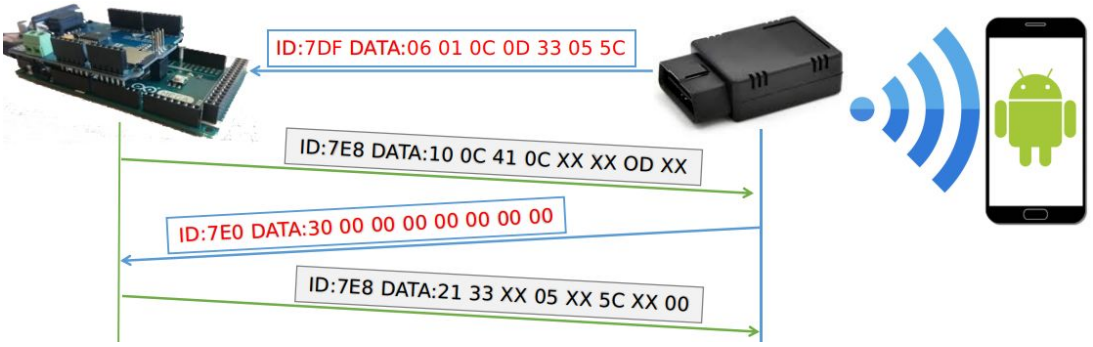


Image 23 : Schéma d'une requête multiple

Quand on renvoie la réponse, le premier octet "0x10", signifie que ça sera une réponse sur plusieurs trames. Ensuite le deuxième donne le nombre d'octets à traiter par la suite et après c'est identique. Quand l'elm327 reçoit cette trame pour accepter d'envoyer plusieurs trames d'affilée, il renvoie sur ID 7E0 avec comme données juste le premier octet à "30". A partir de là, on peut envoyer autant de données que l'on veut par trame de 8 octets avec au début "2x" et ça s'incrémente à chaque fois donc si il y avait eu une trame ça aurait été "22"

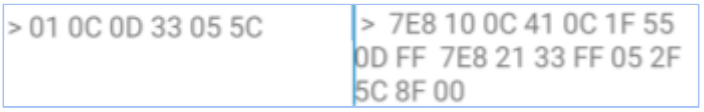


Image 24 : Réponse requête multiple avec l'Arduino Mega

e Création d'un serveur sur android

Nous voulons créer un serveur web pour pouvoir partager les données stockées par l'application à différents clients (sur le même réseau wifi) et cela fonctionnera avec des supports différents comme une page Web ou encore un hololens. Avec comme facilité que tout le travail du traitement des données soit déjà fait.

Pour créer ce serveur web, je me suis aidé de la librairie "AndroidAsync by koush" qui permet de créer un serveur web avec une API REST. C'est à dire que l'on réalise une requête GET et l'on obtient la réponse voulue. Si dans mon navigateur je rentre l'adresse du serveur (ex: 192.168.1.1:5000), voici ce que j'obtiens. C'est un JSON ce qui est facile à traiter par la suite pour le client.

Image 25: JSON obtenu avec la requête HTTP

Mais ce qui est embêtant avec ce serveur, c'est qu'il faut faire une requête pour avoir une réponse et si entre deux requêtes, il y a des valeurs qui changent, on n'en a pas idée et de la même façon si nos valeurs ne changent pas, on fait toujours plein de requêtes pour rien. C'est pourquoi j'ai mis en plus en place un serveur WebSocket qui quand on se connecte à lui, le serveur garde votre IP et il peut nous

parler quand il le désire. Je l'ai essayé avec un client écrit en NodeJS déjà opérationnel.

f Création du client sur unity pour le casque Hololens

i Partie C#

Pour cette partie, j'ai utilisé visual code pour créer des applications windows forms ou console avant d'avoir à disposition l'hololens, pour commencer à écrire mes script en C#.

1 Bluetooth

Pour la partie bluetooth en C#, j'ai utilisé la librairie 32Feet et comme je l'avais fait déjà auparavant en Java pour Android, le code, la compréhension du mécanisme d'appairage et de connection à la socket c'est fait plus rapidement. Et ça a fonctionné assez rapidement du premier coup. Après, c'est identique à la partie JAVA ou presque niveau algorithmique.

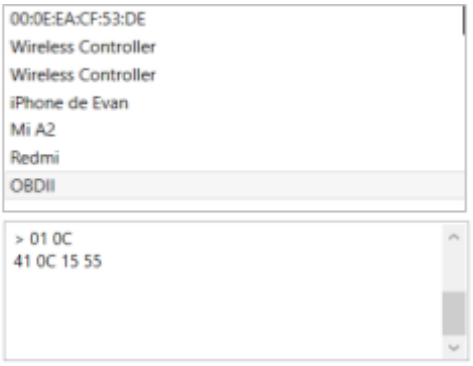


Image 26: Windows Forms Bluetooth

2 Client Socket

Pour le client, j'ai utilisé la librairie Websocket-sharp, elle nous permet de créer un client http basic. Donc qui envoie une requête http Get à un serveur et d'attendre une réponse. Mais aussi de faire une Websocket avec un client websocket ce qui est plus intéressant pour les raisons données plus haut..

```
connected
Message received: {"name":"Jerome","age":"18","classe":"5emeB"}second
Message received: {"name":"Jerome","age":"18","classe":"5emeB"}
```

Image 27 : Réponse à la requête Http et ainsi qu'à la webSocket

Donc nous obtenons deux scripts fonctionnant en C#, donc maintenant nous devons essayer de l'implémenter sur Unity pour essayer de le faire fonctionner dans le casque Hololens.

ii Partie Unity

1 Présentation du casque Hololens

Microsoft HoloLens est un casque de réalité mixte permettant de simuler des hologrammes qui s'intègrent dans le champ de vision de l'utilisateur. C'est à dire que nous continuons à voir normalement à contrario d'un casque de réalité virtuel ou l'on

est dans un environnement fictif. Si l'on voit une vraie table et que l'on veut poser un vase fictif dessus (en simulation d'hologramme), il y aura des contraintes, on ne pourra pas faire passer le vase à travers la table, c'est ça la réalité mixte. C'est celle qui interagit avec son environnement.

2 Création d'application pour l'Hololens

La partie sur Unity et sur le développement d'application pour le casque Hololens en réalité mixte se basait sur des tutoriels de mise en place de tout l'environnement et de la mécanique de déploiement de l'application sur le casque qui est très longue en temps et en ressource pour l'ordinateur.

Au niveau de l'interface sur l'Hololens, nous avions pensé qu'il était plus intéressant d'afficher les informations dans le champ de vision du conducteur car le but final sera un peu dans le même aspect que l'afficheur tête haute. Sauf avec l'alerte d'angle mort et toute autre alerte pouvant passer par l'OBDII.

Image 28: Concept d'interface sur Hololens

Actuellement, je suis arrivé au point de faire correspondre les scripts du bluetooth et de l'envoi de requêtes à des événements comme "cliquer sur un bouton" etc. Mais comme la compilation et le déploiement mettent énormément de temps à se faire à chaque fois, je perds beaucoup de temps sur ce point.

Image 29: Requête Http Get qui fonctionne sur le casque

La partie requête http fonctionne correctement ainsi que la partie WebSocket. Comme on peut le voir ci dessus, Nous recevons bien les informations envoyées par le serveur dans Unity ainsi que dans l'Hololens, ce qu'il est vital de préciser car des fois, cela marche en simulation dans unity mais pas quand on le déploie sur l'Hololens. Maintenant il y a à faire une grosse partie design pour rendre ça plus attractif et intuitif. Au niveau du bluetooth, c'est toujours en cours de développement.

5 Bilan du stage

Durant ce stage de deux mois réalisé à la plateforme MobBi, j'ai eu la chance de découvrir des technologies fascinantes et très enrichissantes pour la suite, notamment le casque Hololens et la multitude de possibilités qu'il offre, ainsi que l'ELM327, qui est un appareil conçu très minutieusement où chaque chose à un intérêt pour requêter ou debugger et qui est open source, ce qui rejoint le projet OSV. J'ai aussi pu rencontrer des professionnels de l'immobilier pour leur présenter le casque et leur faire une démonstration de ses capacités.

De plus, j'ai travaillé essentiellement en autonomie bien qu'il aurait été appréciable d'avoir un expert dans certains domaines quelques fois, je pense que j'ai assez bien appréhendé le sujet et que j'arrivais à produire souvent des choses convenables qui avaient un vrai but pour le projet. J'ai apprécié comprendre petit à petit comment fonctionnait l'ELM327 avec la datasheet ainsi que les processus mis en place dans une voiture pour répondre à cet appareil avec la compréhension du CAN et à la norme OBDII. Cela m'a donné plus confiance en moi car je n'avais pas vraiment le choix, il y avait peu de personnes auprès de qui me renseigner et peu d'informations disponibles sur internet (surtout au niveau de la réponse de la voiture).

Ensuite, je pense que ça n'aurait pas été de refus d'avoir un peu plus de temps sur ce stage car il y a des parties extrêmement chronophages, notamment sur Android avec le design de l'application ainsi que pour Unity avec le déploiement sur le casque et l'environnement etc. Cela peut nous prendre une après midi pour un aspect graphique. De plus, pour rendre cela exploitable par la suite, il faut implémenter dans le JSON toute la norme OBDII avec des règles précises.

Enfin, le fait de retoucher à l'électronique ou d'aider mes collègues dans ce domaine me conforte dans ma volonté de carrière dans l'internet des objets, les systèmes embarqués ou la communication entre systèmes, car cela ne m'intéresse pas de faire que de la programmation. Ce qui est intéressant pour moi, c'est de manier plusieurs choses et d'avoir une vue global du problème. Faire principalement de la communication entre systèmes m'a vraiment plu et faire la connection entre appareils qui à première vue ne sont pas fait pour se parler directement a été super intéressant.

6 Conclusion

Pour conclure, nous pouvons dire que ma mission de départ, qui était de faire communiquer l'ELM327 avec un Hololens ainsi que d'autres supports a été respectée à 90%. J'arrive à faire communiquer le module en bluetooth à un téléphone android qui traite les informations, les afficher et les stocker en tant que serveur. Ensuite, on peut s'y connecter soit avec un Hololens et les afficher pour un conducteur notamment, soit avec un autre appareil pour faire ce que l'on veut des données, comme un site Web par exemple. On savait très bien que ce n'est pas optimisé pour l'Hololens, c'est pour ça que l'on travaille encore sur la connexion bluetooth.

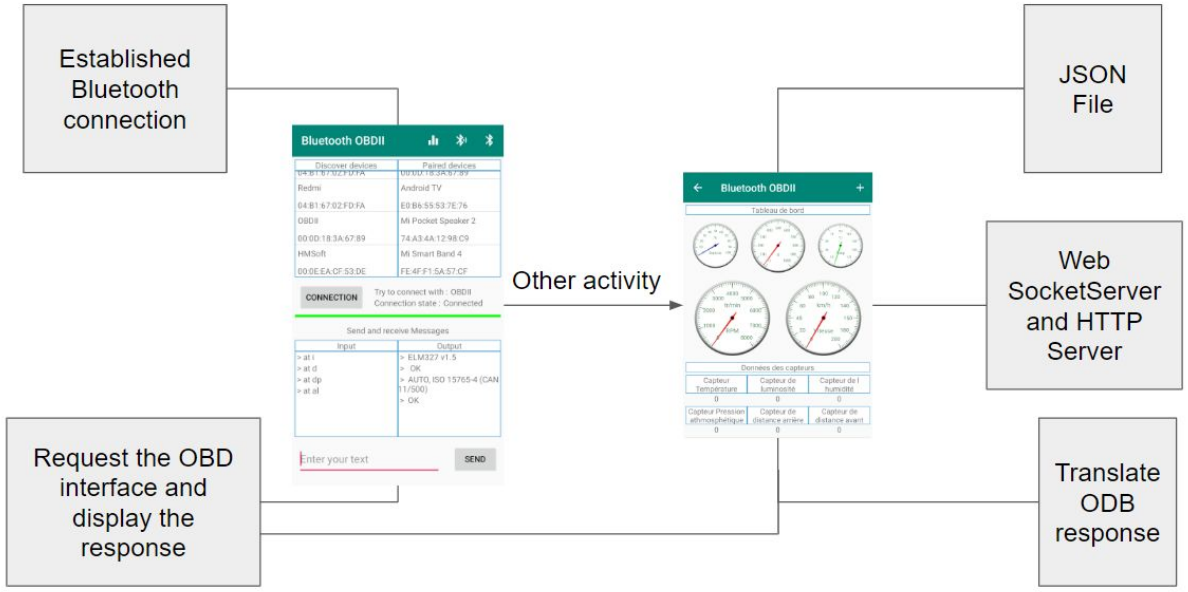
Cela permettra à la plateforme MobBi de montrer une autre facette des projets dans laquelle elle s'investit, pour tous ses clients futurs. De plus, cela à un réel intérêt pour l'université de Rennes pour son projet du véhicule open source, l'OSV, qui est électrique. Cela répond à la norme OBDII, ce qui en fait un peu plus une voiture réelle, d'abord parce que c'est une obligation pour les constructeurs et ensuite parce que ça leur permet aussi de montrer leur système, d'en retirer les valeurs et de les afficher où elle veut, pour des démonstrations par exemple.

Sur l'avenir du projet, je pense qu'il y a un réel potentiel pour avoir une alternative tête haute mais il n'y a pas assez de données et de type de PID qui transitent pour avoir des choses très performantes. Ici, on marche sur le principe de requête, ce qui n'est pas le plus performant selon moi. De plus, l'hololens 1 présente un champ de vision trop étroit pour l'utilisateur. Il faut voir comment cela évolue avec le deuxième, qui va bientôt être commercialisable.

7 Annexes, glossaire et bibliographie

a Annexes

Diagramme de l'application android:



b Glossaire

OSV : Open source véhicule
CAN : Controller Area Network
OBD : On-board diagnostics
CSMA-CR/CA: Carrier Sense Multiple Access - Collision resolution / Collision Avoidance
Noeud(dans le réseau CAN): un calculateur, ici une carte arduino plus shield CAN

c Bibliographie

Polycopié : Les réseaux de terrain dans l'automobile de Jérome Blanchard de 2017
Datasheet du ELM327 : https://www.elmelectronics.com
Documentation Android develloper: https://developer.android.com
Documentation Unity/Hololens: https://docs.microsoft.com/
Article du Ministère de l'Enseignement supérieur de la Recherche et de l'Innovation: 22.05.2017. *Les structures de diffusion de technologies*. Consulté sur http://www.enseignementsup-recherche.gouv.fr