Université de Rennes 1

Projet OSV 2020

Partie instrumentation et temps réel

Brault Lilian – Coquio Ewen – Vorburger Lucas
2020/2021

Table des matières

I- REMERCIEMENT

Nous tenons tout d'abord remercier nos « tuteurs de projet » David Levalois ainsi que Damien Hardy sans qui ce projet n'aurait très certainement pas pu aboutir. En effet leur aide apportée nous aura permis de nous débloquer de nombreuses fois, notamment sur la partie BSI et compréhension de la structure de l'OSV.

Nous remercions également nos collègues avec qui nous avons collaboré tout au long de ce projet.

II- INTRODUCTION

L'Open Source Vehicle (OSV) est un projet collaboratif qui met en relation plusieurs départements de l'Université de Rennes 1 : le FabLab, l'INSA, l'IETR, l'IRISA, etc.… Tout ces acteurs collaborent sur ce projet lancé en 2015 sur différentes parties telles que le moteur, l'électronique embarquée, les communications, etc.…

Tous les plans du véhicule, les codes, les circuits etc.… sont disponible au public et en fait donc un projet Open Source.

Ce projet s'inspire du mode de fonctionnement que l'on retrouve chez les constructeurs automobiles avec différents calculateurs, communication en bus CAN.

III- PRESENTATION DE L'OSV

Le projet de réalisation de l'OSV à débuté fin 2018 ou le projet est de créer notre propre architecture électronique sur un véhicule. L'architecture est composée de plusieurs calculateurs fait à partir d'Arduino et de de deux bus CAN.

Voici l'architecture 2019 de l'OSV :

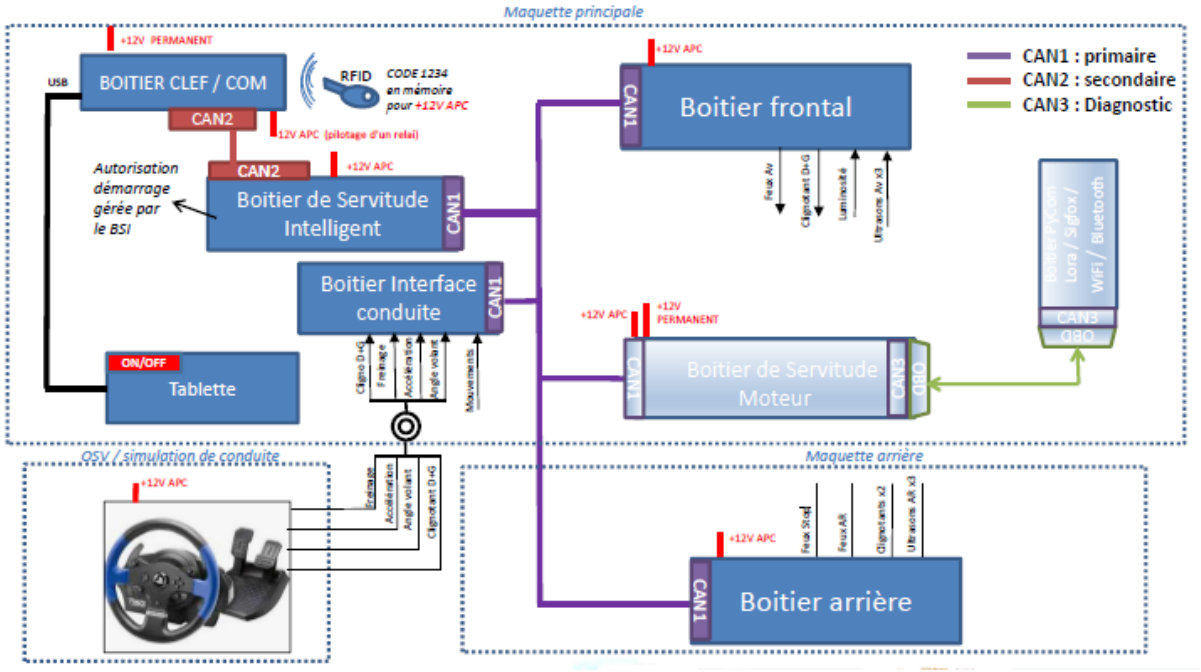


Figure 1 : Architecture 2019

Sur cette paquette nous retrouvons :

- Le Boitier de servitude intelligent (BSI) qui fait office de passerelle entre le CAN 1 et 2
- Le boitier Interface conduite (BC) qui récupère les informations des clignotants, des feux et l'angle volant
- Le boitier de servitude moteur (BSM) qui récupèrera les informations du moteur
- Le boitier frontal (BF) et arrière (BA) qui gèrent les phares, les radars de stationnement et les clignotants.
- Le boitier clé com qui gère le verrouillage et le déverrouillage de l'OSV

IV- OBJECTIF

L'objectif de notre projet est d'améliorer ce qui est existant et ajouter des nouvelles fonctionnalités pour obtenir l'architecture 2020 :

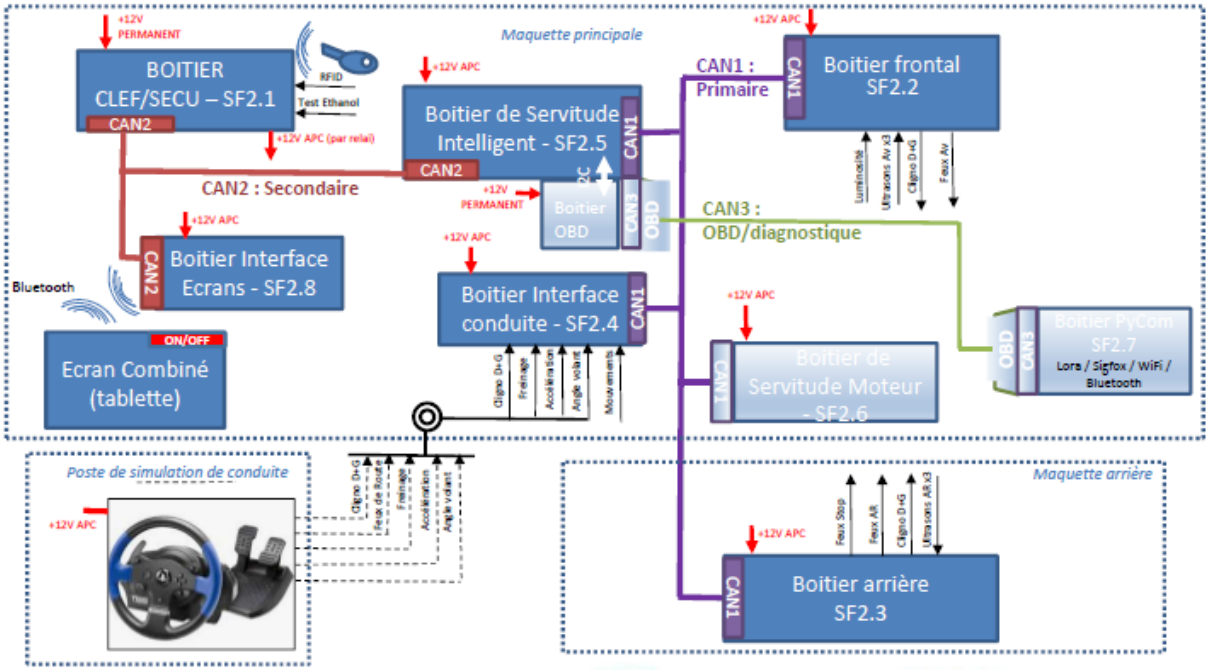


Figure 2 : prévision architecture 2020

L'objectif de cette année est de créer le boitier de servitude moteur, ajouter un boitier interface écran pour communiquer les informations au conducteur, ajouter un lidar pour détecter des obstacles, rendre la maquette plug and Play, ajouter un boitier diagnostique et améliorer le boitier de servitude intelligent.

Notre partie du projet se concentre principalement sur l'optimisation de la BSI et du boitier de servitude moteur.

a) Partie temps réel

Dès le début du projet nous savions que la BSI existante était fonctionnelle mais avec un problème majeur dans son fonctionnement. En effet un ajout d'un système de cryptage des trames a permis de mettre en évidence le fait que certaines trames qui circulaient sur le réseau CAN étaient tout simplement ignorées. Or pour le moment nous ne savions pas si le problème venait de code de la BSI ou d'un problème matériel (le code est implémenté sur un Arduino Mēga). Notre objectif va être alors de comprendre comment fonctionne le code existant, l'adapter aux nouvelles fonctionnalités et contraintes, identifier et corriger le problème de pertes de trames.

b) *Partie instrumentation*

Afin de permettre une meilleure intégration pour l'affichage et la gestion du véhicule, il nous faut rajouter différents capteurs pour pouvoir installer le nouveau boîtier de servitude moteur. Le cahier des charges est assez simple : un ajout de capteur d'angle volant, de vitesse, de pédale de frein et d'accélérateur, d'un capteur de température et d'un « capteur » de tension batterie. L'ajout de tous ces capteurs permettrons d'utiliser l'OSV dans un environnement virtuel mais aussi de faire du diagnostic.

V- INTEGRATION DES CAPTEURS

a) *Capteur de température*

Dans un premier temps, notre objectif va être de mesurer la température du moteur de l'OSV afin d'alerter l'utilisateur en cas de surchauffe.

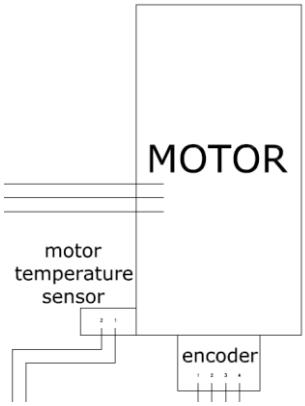


Figure 3 : extrait du schéma de câblage de l'OSV

Nous avons observé que dans le moteur se trouvait déjà un capteur de température de type thermistance. Pour tracer la caractéristique de ce dernier, nous avons mesuré la température du moteur à l'aide d'un thermomètre ainsi que la tension aux bornes de la thermistance.

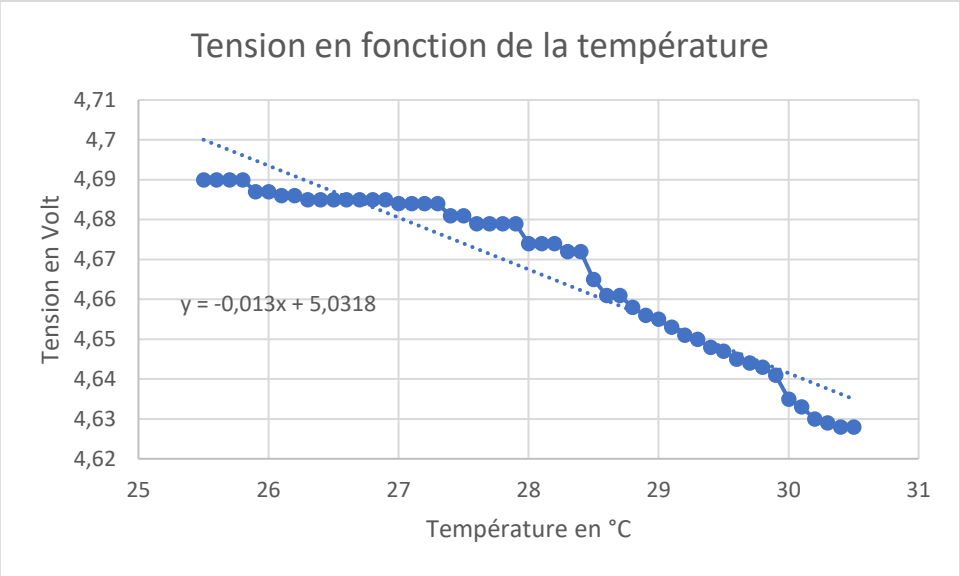


Figure 4 : caractéristique de la tension en fonction de la température

Nous remarquons que cette variation correspond à une thermistance CTN (coefficient de température négatif). Sur une large plage de température, la caractéristique de la CTN n'est pas linéaire. En revanche, sur notre plage d'utilisation, nous allons la considérer comme telle car la mesure de la température ne nécessite pas une grande précision. En effet cette mesure ne sert qu'à prévenir l'utilisateur en cas de surchauffe.

Maintenant, notre problématique est que la variation de tension est trop faible pour être mesurée par le convertisseur analogique numérique de l'Arduino.

Pour pallier à ce problème, nous devons adapter la tension pour obtenir une tension pleine échelle de 0-5V en sortie du capteur.

Pour ce faire nous allons utiliser le montage suivant qui est un soustracteur avec un gain de 2,1. Nous savons que la tension de sortie du convertisseur analogique numérique est de 0V – 5V. Or l'équation de la caractéristique du capteur de température donné par Excel est :

$$V = -0{,}013\ T° + 5{,}0318$$

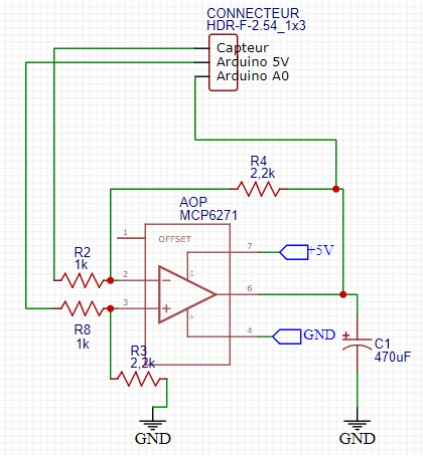


Figure 5 : montage soustracteur avec un gain de 2.1

Ainsi, nous relevons la tension mesurée aux bornes de la thermistance, nous la soustrayons au 5V de référence de l'Arduino, puis nous amplifions ce résultat par 2,1. Après adaptation de la tension, nous avons donc l'équation suivante :

$$V_s = (5 - V) \times 2{,}1$$

$$V_s = (5 + 0{,}013\,T° - 5{,}0318) \times 2{,}1$$

Ainsi, l'équation de la température en fonction de la tension de sortie de l'adaptateur.

$$T° = \frac{V_s + 0{,}06678}{0{,}0273}$$

Désormais, nous pouvons donc en déduire la température minimale et maximale mesurée.

- Si $V_s = 5V$ on a $T° = \frac{5+0{,}06678}{0{,}0273} = \mathbf{185{,}59°C}$
- Si $V_s = 0V$ on a $T° = \frac{0{,}06678}{0{,}0273} = \mathbf{2{,}44°C}$

Donc $\Delta T° = 185{,}59 - 2{,}44 = 183{,}15°C$

Ainsi la précision de la mesure avec un convertisseur 10 bits est de $\frac{183{,}15}{1024} = \mathbf{0{,}18°C}$

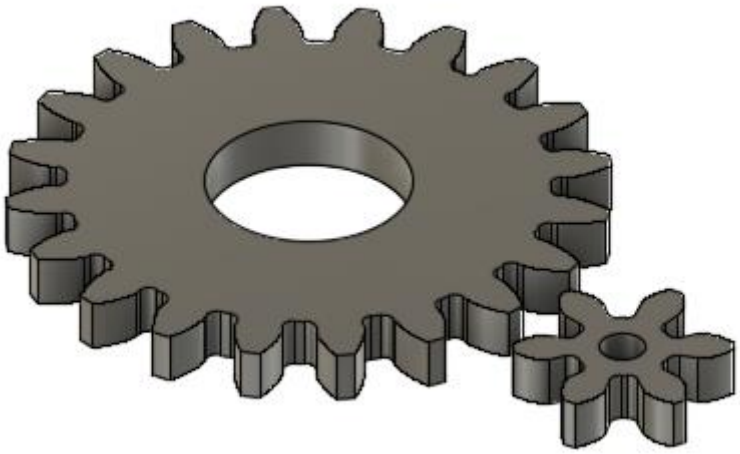
b) Mesure de l'angle volant

Pour mettre en œuvre la mesure de l'angle volant, nous avons testés différentes solutions. Dans un premier temps nous nous sommes inspirés des systèmes utilisés par les constructeurs automobiles avec un capteur capacitif. Nous avons alors tenté de le mettre en œuvre avec un système de 3 engrenages et de 2 capteurs à effet hall. Il s'est avéré que cette solution n'était pas viable de part l'absence de référentiel, en effet, impossible de savoir si le volant a tourné lors du démarrage du véhicule.

Nous nous sommes alors rabattus sur un système plus simple et tout aussi, voire plus, précis. Nous avons fait le choix d'utiliser un potentiomètre 10 tours de 10kΩ (référence RS 1070785).

Figure 6 : potentiomètre 10 tours de 10kΩ

Le volant de l'OSV fait 1,5 tour dans un sens et 1,5 tour dans l'autre. Nous avons donc besoin de réaliser une adaptation mécanique. Pour cela nous allons utiliser un système d'engrenages car très simple à mettre en œuvre. Comme le volant fait 3 tours et le potentiomètre 10, nous avons un rapport de réduction de 0,3.



Pour obtenir le rapport de réduction nécessaire nous avons une grande roue avec 20 dents et une petite roue avec 6 dents. Soit un rapport $r = 6\backslash 20 = 0.3$

Ces pièces ont été imprimées en PLA avec une épaisseur de couche de 0,28mm. La grande roue sera fixée sur l'axe du volant et la petite sur le potentiomètre. Ce système mécanique sera intégré dans le haut de colonne de l'OSV.

Figure 8 : petite roue sur le potentiomètre

Figure 7 : grande roue sur le volant

Les 3 tours du volant font 1080° et le convertisseur analogique numérique de l'Arduino est de 10 bits (soit 1024 valeurs) nous avons donc une précision de **1,05**°.

c) *Mesure de la course de la pédale d'accélération*

Dans l'objectif à l'avenir de faire de l'autodiagnostique et d'utiliser l'OSV en réalité virtuelle, connaitre la course de la pédale d'accélérateur permettrait de détecter une anomalie entre la vitesse du moteur et l'enfoncement de la pédale.

Nous avons remarqué sur le schéma de câblage de l'OSV que la pédale d'accélération est un simple potentiomètre.

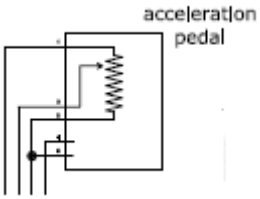


Figure 9 : extrait du schéma de câblage de l'OSV

Lorsque nous mesurons la variation de tension en fonction de l'enfoncement de la pédale, on obtient une tension entre 0V et 4,4V dont le convertisseur 10 bits de l'Arduino peut mesurer cette tension. Cette valeur sera exprimée entre 0 et 255 pour un maximum de précision.

d) *Détection d'appui sur la pédale de frein*

Toujours dans l'objectif d'utiliser l'OSV dans un environnement virtuel et de faire de l'autodiagnostique, nous avons besoin de savoir l'état de la pédale de frein (enfoncée ou non). Sur le schéma de câblage de l'OSV, nous observons que la pédale de frein fonctionne comme un interrupteur.

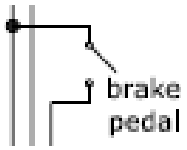


Figure 10 : extrait du schéma de câblage de l'OSV

Lorsque nous mesurons les tensions, nous avons :

- 11,5V quand la pédale et au repos
- 0V quand la pédale est enfoncée

Or la logique Arduino est entre 0 et 5V. Nous devons donc adapter cette tension.

Pour cela, nous réalisons le pont diviseur suivant avec un rapport de 1,42 :

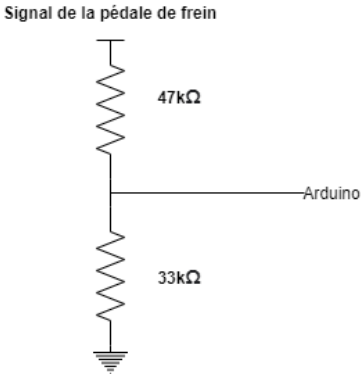


Figure 11 : pont diviseur de tension

Nous avons donc en sortie une tension de 4,95V, ce qui est acceptable pour l'Arduino.

e) Mesure de la vitesse

Afin de réaliser une mesure de la vitesse nous avons commencé par tenter de mettre un capteur inductif au niveau de la roue avec une lamelle en métal sur le pneu. Le problème principal allait être l'installation de ce capteur. Après avoir étudié la documentation du

moteur, nous avons vu que le moteur embarquait un encodeur qui s'occupe de rendre la fréquence du moteur.

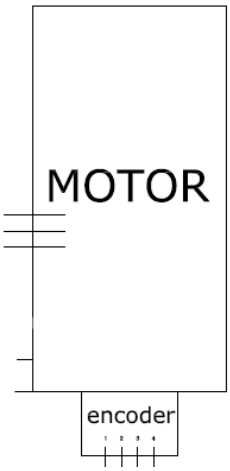
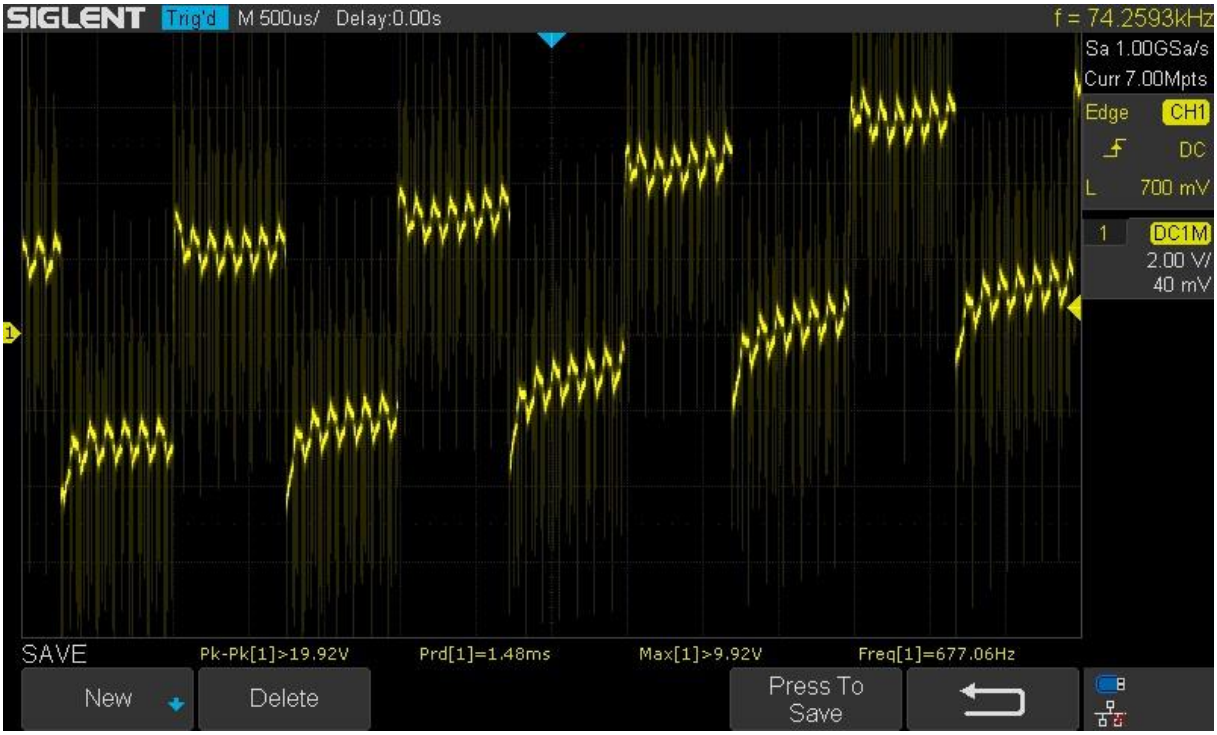


Figure 12 : extrait du schéma de câblage de l'OSV

Figure 13 : encoder sur l'OSV

Lorsque nous relevons le signal en sortie de l'encoder, nous avons un signal carré dont la fréquence varie en fonction de la vitesse du moteur.



Nous remarquons que le signal est très bruité. Donc pour le lisser, nous ajoutons un condensateur de 100nF. Nous avons ensuite tracé la caractéristique de la vitesse du moteur en fonction de la fréquence de l'encoder.

Pour cela, la documentation de l'encoder nous informe que pour 1 tour du moteur, nous avons 64 impulsions. Donc nous divisons la fréquence par 64 puis nous multiplions le tout par 60 pour avoir une vitesse de rotation en tour/min.

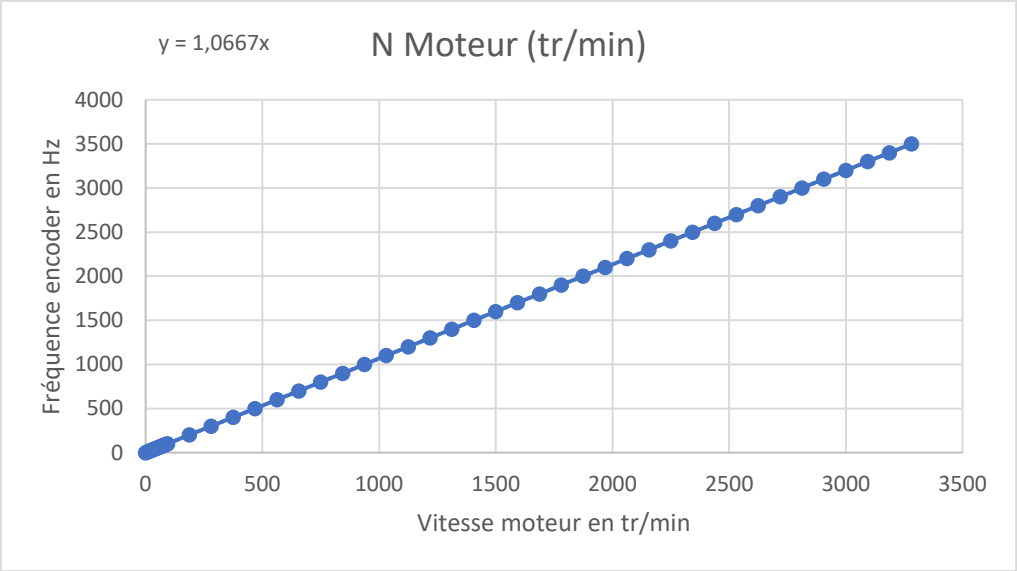


Figure 14 : caractéristique de la vitesse en fonction de la fréquence

Nous avons donc une caractéristique linéaire avec un facteur d'environ 1. Ce qui signifie qu'a 300Hz, nous avons 300tr/min.

Pour mesurer la fréquence de ce signal, nous utilisons un Arduino Micro à côté du BSM car le code utilise la fonction pulseIn d'Arduino. Cette fonction mesure la durée du signal à l'état haut. Donc si cette fonction se trouve dans le BSM, la fonction risque de « rater » le moment ou le signal passe à l'état bas car elle peut être occupé par une autre tâche. Donc l'Arduino Mini va communiquer la vitesse du moteur au BSM par une liaison I2C.

```
mesure_periode();
vitesse_moteur = 1.0667 * frequence; // correspond à la caractéristique Fréquence Vitesse
```

Figure 15 : extrait du code de mesure de la vitesse

Ainsi, nous obtenons la vitesse du moteur en tr/min qui sera envoyé sur le bus CAN.

VI- CREATION D'UN CONNECTEUR PLUG AND PLAY POUR LA MAQUETTE

L'OSV appartenant à l'IMT atlantique, nous devons rendre la maquette plug and play. Pour cela, nous réalisons à partir d'un connecteur SUB D 25 broches.

Figure 16 : connecteur SUB D 25 broches

Le connecteur femelle sera fixé sur l'OSV et on y retrouvera les données suivantes :

- Angle volant
- Clignotant droit
- Clignotant gauche
- Klaxon
- Feux
- Feux de recul
- Pédale d'accélérateur
- Pédale de frein
- Température moteur
- Encodeur OSV (pour la vitesse du moteur)

Pour protéger le tout, nous avons mis un fusible

Pour un rendu durable et solide, nous avons fait le circuit imprimé suivant :

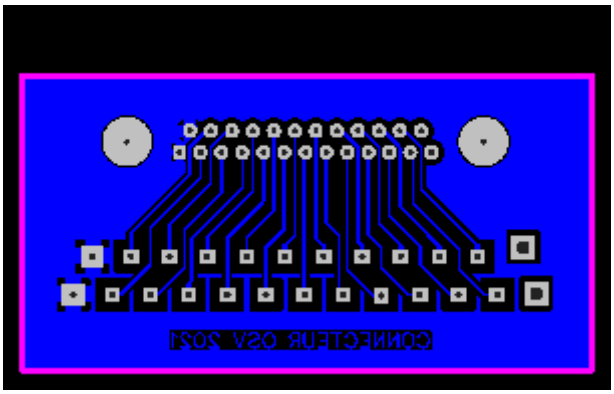


Figure 18 : côté Bottom de la carte

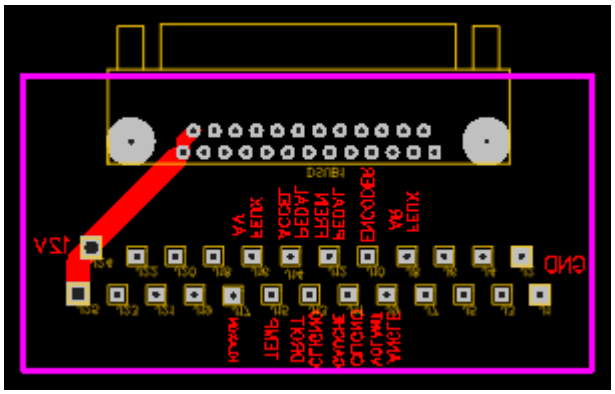


Figure 17 : côté Top de la carte

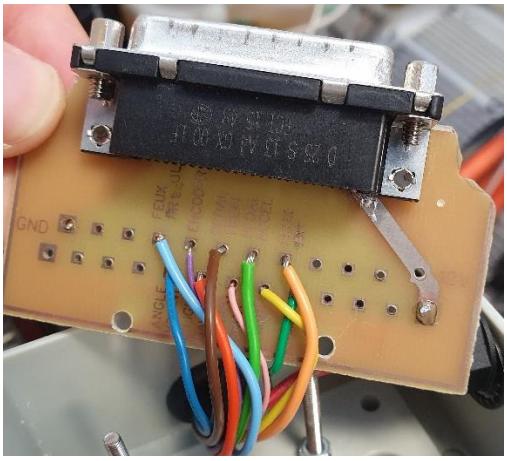


Figure 20 : connecteur Plug and Play

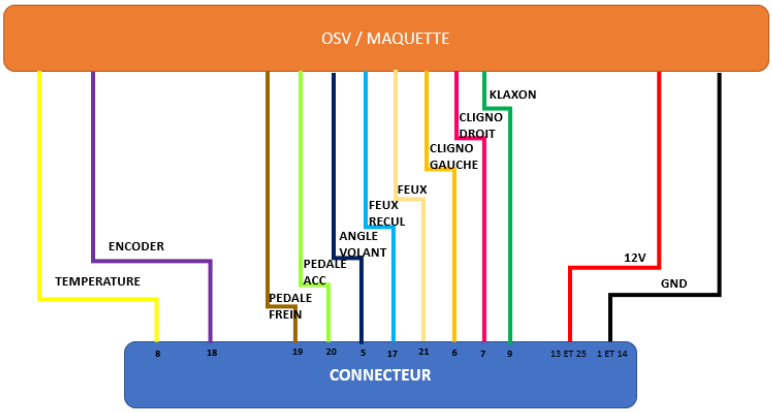


Figure 19 : schéma de connexion du connecteur

Pour finir, nous fixons le connecteur sur la carcasse de l'OSV côté passager. Il faudra également rebrancher l'alimentation de la maquette sur le connecteur. En effet jusqu'à maintenant la maquette était alimentée par une alimentation 12V externe. Pour ce faire nous allons donc quadrupler les pins du connecteur pour plus de sécurité, soit les pins (insérer nombres) pour le +12V et les pins (insérer nombres) pour la masse. En revanche il nous reste encore à ramener ce 12V au connecteur. Pour ce faire nous allons utiliser le convertisseur DC-DC de la batterie (80V – 12V) que nous allons tirer jusqu'au connecteur.

Pour le test nous ferons un raccord sur une nouvelle alimentation externe pour éviter d'endommager la batterie du véhicule. De plus nous rajouterons un fusible (insérer la valeur) sur la branche +12 qui arrive sur le boîtier avant (boîtier qui s'occupera de dispatcher les alimentations entre toutes les autres cartes).

VII- CREATION DU BOITIER SERVITUDE MOTEUR

a) Partie logicielle

Etant donné que pour ce boitier nous partions de zéro, il nous a d'abord fallut bien comprendre comment fonctionnait les différents boîtiers déjà présents. Au bout de quelque séance le principal des codes était compris. Afin de ne pas partir dans tous les sens nous avons commencé par modifier la messagerie.

```
#define CAN1_ID_DATA_BSM                    CAN1_ID_CMD_BC_PERIODIC+1     //0x200 in messagerie xls
```

```
#define CAN1_SIZE_DATA_BSM                              7

//value: 0..255 in degree celsius (bit 0.0 .. 0.7)
#define CAN1_GET_DATA_BSM_TEMPERATURE_ENGINE(t)             ((u8)t[0])
#define CAN1_SET_DATA_BSM_TEMPERATURE_ENGINE(t,val)         t[0]=val;

//value 0..100 in % (bit 1.0 .. 1.7)
//Rq: can be compressed in 7 bits if needed
#define CAN1_GET_DATA_BSM_BATTERY_PERCENTAGE(t)             ((u8)t[1])
#define CAN1_SET_DATA_BSM_BATTERY_PERCENTAGE(t,val)         t[1]=val;

//value: 0..255 in km/h (bit 2.0 .. 2.7)
#define CAN1_GET_DATA_BSM_OSV_SPEED(t)                  ((u8)t[2])
#define CAN1_SET_DATA_BSM_OSV_SPEED(t,val)              t[2]=val;

#define CAN1_GET_DATA_BSM_ENGINE_SPEED(t)           ((s16)(t[4]<<8 | t[3] ))
#define CAN1_SET_DATA_BSM_ENGINE_SPEED(t,val)           t[3]=(val & 0x00ff);t[4]= ((val & 0xff00) >>8);

#define CAN1_GET_DATA_BSM_BRAKE_PEDAL(t)            ((u8)t[5])
#define CAN1_SET_DATA_BSM_BRAKE_PEDAL(t,val)            t[5]=val;

#define CAN1_GET_DATA_BSM_ACCEL_PEDAL(t)            ((u8)t[6])
#define CAN1_SET_DATA_BSM_ACCEL_PEDAL(t,val)            t[6]=val;


#define CAN1_SIZE_STATE_BSM                                 2

//value 0/1 (bit 0.0)
#define CAN1_GET_STATE_BSM_ENCRYPTION_MODE(t)                   ((u8)bitRead(t[0], 0))
#define CAN1_SET_STATE_BSM_ENCRYPTION_MODE(t,val)               bitWrite(t[0], 0, val);

//value: 0/1 (bit 0.1)
#define CAN1_GET_STATE_CMB_BC_BRAKE(t)                  ((u8)bitRead(t[1], 0))
#define CAN1_SET_STATE_CMB_BC_BRAKE(t,val)              bitWrite(t[1], 0, val);
```

Figure 21 : partie BSM de la messagerie

Pour faire cela nous avons dû comprendre comment la messagerie .h fonctionnait. Un fois ceci fait nous avons pu créer et adapter notre nouvelle trame et les signaux qui la composent.

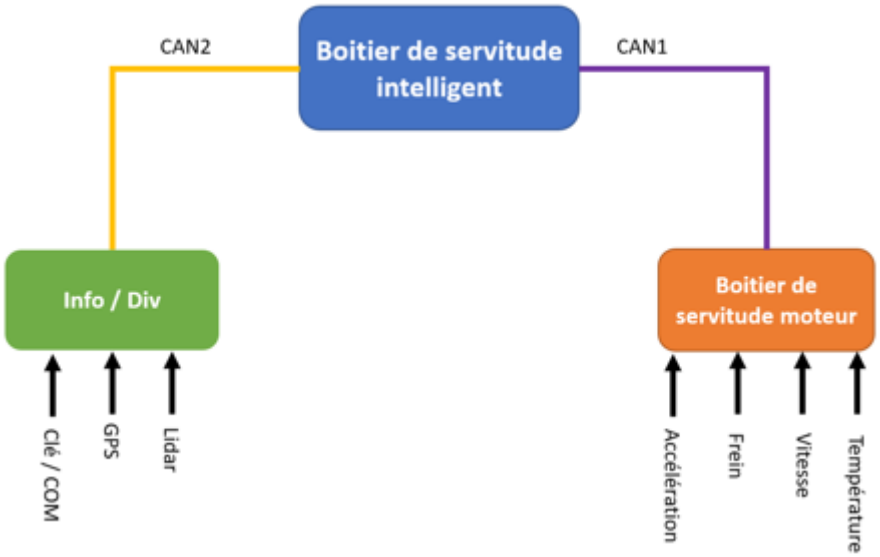


Figure 22 : Schéma BSI

La messagerie étant faite nous pouvons passer maintenant au code BSM. Avant de commencer il nous faut déterminer quels capteurs seront reliés sur notre BSM et où iront les capteurs restants

Donc sur le boîtier, nous retrouverons les capteurs de température, de vitesse et de pédales (frein et accélérateur). Comme expliqué précédemment nous allons utiliser les formules des courbes des capteurs pour notre code. Nous stockerons ces différentes valeurs dans des bytes afin de les organiser sur la trame. Puis en utilisant la librairie « mcpCAN » nous pourrons les envoyer à la BSI via le réseau CAN 1.

c) *Partie matérielle*

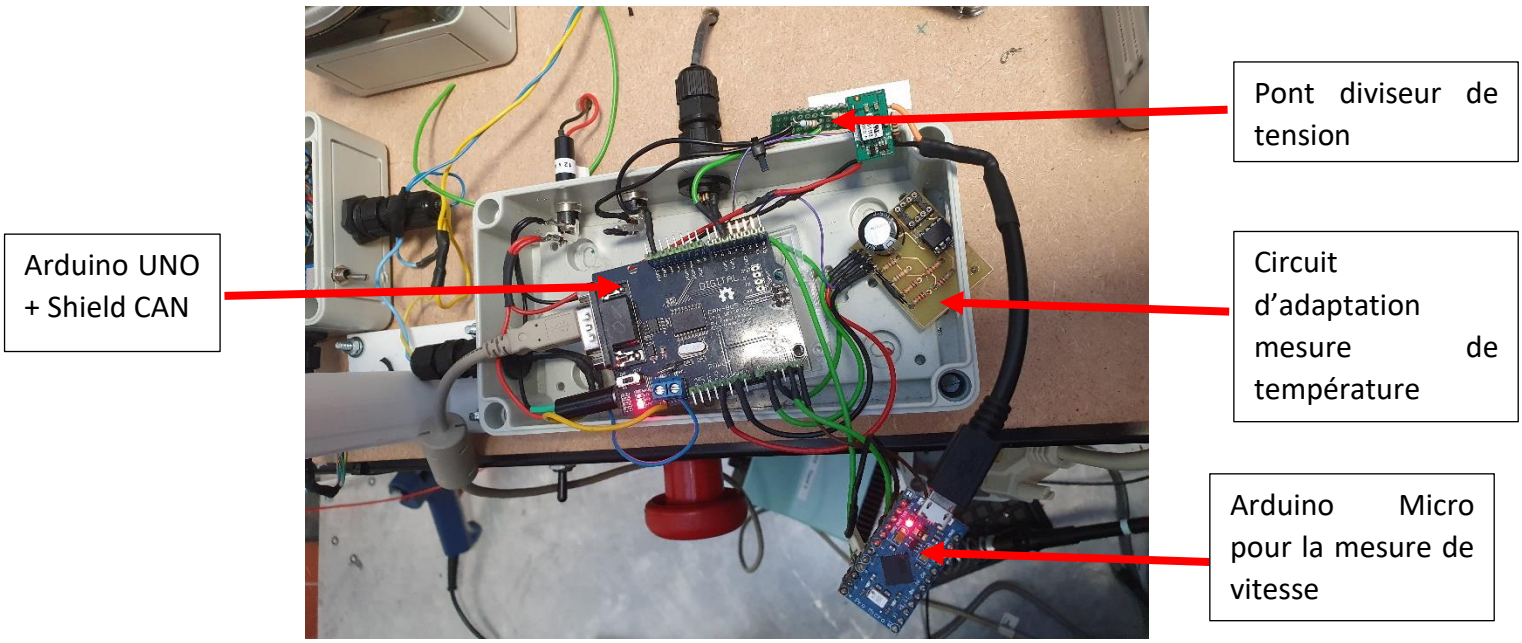


Figure 23 : Boitier Servitude Moteur terminé

Une fois la partie code terminée, il ne nous reste plus qu'à câbler le boîtier. Pour cela nous utiliserons un Arduino UNO munis d'un Shield can pour la transmission. Un des quatre signaux ne nécessite aucun conditionnement, il nous suffira donc de souder directement l'accélérateur sur la carte. En revanche pour la température, la vitesse et le frein il nous faudra mettre un ajout physique pour permettre le bon fonctionnement du capteur.

Pour le frein nous utiliserons, comme énoncé précédemment, un pont diviseur de tension afin de diviser la tension initiale de 11,5V en 5V.

Pour la température il faudra utiliser un montage un peu plus complexe en revanche. Nous sommes partis au début sur un montage avec un AOP en montage différentiel avec une alimentation symétrique +6/-6V. Il nous fallait alors rajouter un montage suiveur avec un pont

diviseur de tension pour obtenir deux tensions opposées avec du +12V. Après réflexion nous en avons déduit que nous pouvions utiliser un AOP particulier avec une alimentation non symétrique en 0/+5V, ce qui permet alors de ne pas mettre de suiveur et donc de simplifier notre schéma.

Quant à la vitesse, étant donné que nous utilisons la fonction pulse In d'Arduino, il nous faut isoler cette fonction dans une carte à part. En effet comme la fonction utilise les données temporelles de l'Arduino, si nous rajoutons de nouveaux traitements cela va entrainer un retard de la fonction. En isolant cette fonction sur un Arduino mini et en renvoyant la donnée sur un port série nous évitons ce désagrément.

VIII- OPTIMISATION DU BOITIER DE SERVITUDE INTELLIGENT

Le Boîtier de Servitude Intelligent, ou BSI, peut-être comparé au cerveau du véhicule ou à un chef d'orchestre. C'est lui qui est chargé de transmettre les données qui passent d'un réseau CAN vers un autre. La partie optimisation du projet se portait sur la modification de ce boîtier clef qui fonctionne en temps réel.

Le terme temps réel signifie que pour une tâche donnée son échéance est connue ou au moins limitée dans le temps. Ce concept ajoute certaines contraintes telle que la nécessité d'une validation particulière avec un outil qui a été développé l'année précédente par un étudiant en master.

a) Présentation du fonctionnement de départ

Après avoir reçu les codes, nous avons tous les trois tentés de nous approprier les différents codes des boîtiers afin d'en comprendre le fonctionnement global. Le boîtier le plus volumineux était bien évidemment le BSI qui a pris un temps considérable à être compris. Au départ le boîtier récupérait toutes les 80 ms une trame reçue sur le réseau CAN, la traitait en enregistrant les valeurs dans une variable globale et était accompagnée de deux tâches d'écriture sur le réseau CAN2.

La variable globale était protégée par une ressource partagée qui permet de sécuriser l'accès aux données. Il est nécessaire d'utiliser une ressource partagée lorsque plusieurs tâches ont accès à la même variable globale pour un soucis de sécurité.

Sans ressource partagée le risque est que si on modifie la variable globale dans une tâche et qu'elle se fait arrêter par une autre qui modifie également la variable globale (en ajoutant 1 à la variable globale par exemple), la valeur finale soit faussée. Avec une ressource partagée si une tâche essaie d'accéder à une variable qui est déjà utilisée elle est arrêtée. Dans

le cas présent la variable globale est accessible par trois tâches, d'où l'utilisation d'une ressource partagée.

```
22  ////////////////////////////////////////////////
23  //
24  // Trampoline declarations
25  //
26  ////////////////////////////////////////////////
27
28  DeclareTask(Start);
29
30  //Testing the easy way
31  DeclareTask(read_CAN);
32  DeclareAlarm(alarm_read_CAN);
33
34  DeclareTask(write_CAN2_medium);
35  DeclareAlarm(alarm_write_CAN2_medium);
36
37  DeclareTask(write_CAN2_fast);
38  DeclareAlarm(alarm_write_CAN2_fast);
39
40  DeclareTask(write_I2C);
41  DeclareAlarm(alarm_write_I2C);
42
43
44  DeclareResource(R_SPI);
45  DeclareResource(R_OSV_STATE);
46  DeclareResource(R_OSV_DATA);
47  DeclareResource(R_OSV_CMD);
48  DeclareResource(R_AUTOMATE_CURRENT_STATE);
49  DeclareResource(R_CAN1_FRAMES_ARRAY);
50  DeclareResource(R_CAN2_FRAMES_ARRAY);
```

Figure 24 : ressources partagées code BSI

La tâche de lecture du BSI était séparée en deux grandes parties elle-même séparée en deux sous parties. La lecture et le traitement du CAN1 et du CAN2. Chacune des lectures se faisait avant le traitement du réseau correspondant, traitement qui ne portait que sur la trame qui vient d'être lue.

Les deux tâches d'écriture étaient gérées en fonction de la vitesse d'envoi, soit rapide soit moyenne. Les deux tâches envoyaient vers le CAN2 les données enregistrées par la tâche de lecture.

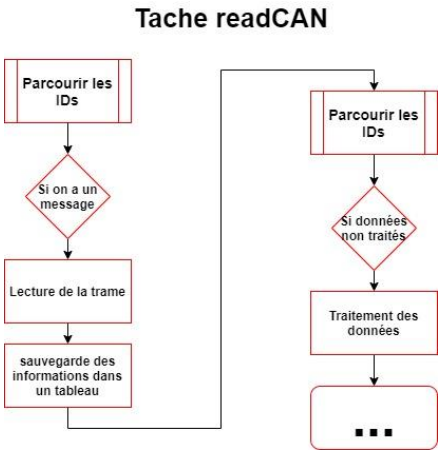


Figure 25 : Schéma ReadCAN du code BSI

b) Problématique

Après plusieurs séances, une présentation a été faite à Damien Hardy afin d'expliquer ce que nous avions compris et mettre au clair ce qui n'était pas compris. C'est après ça que nous avons pu bien visualiser le travail à effectuer sur le boîtier et que nous avons bien compris la problématique du BSI.

Le mode d'encodage avait permis, à l'aide d'un compteur, de mettre en évidence une perte de trame. En effet il arrivait parfois que certaines trames ne soient pas traitées à temps et soient simplement perdues, effacées par la trame de même identifiant suivante. Ce problème est une réelle faille dans la sécurité du véhicule et devait être corrigé.

c) Solutions trouvées

Après en avoir discuté entre nous et avec Damien Hardy nous avons conclu que les corrections possibles sont de revoir le code de la tâche de lecture afin de récupérer les trames d'une autre manière et de les traiter autrement, de vérifier les périodes des tâches, revoir les priorités des trames si besoin et si ceci ne suffit pas, penser à une architecture possible pour corriger le problème l'année prochaine avec éventuellement plusieurs Arduino MEGA si besoin.

d) Réalisation

La première étape de correction a donc été de modifier la tâche de lecture afin de mettre chacun des traitements dans une fonction dédiée. Le fonctionnement n'a pas été modifié de suite afin de s'approprier le code de la tâche mais pour ce dernier nous avons pris notre temps afin de ne rien oublier lors de la modification. La création de fonction était assez simple, une attention particulière devait être prêtée au niveau de l'utilisation de la ressource partagée afin de la relâcher au bon endroit.

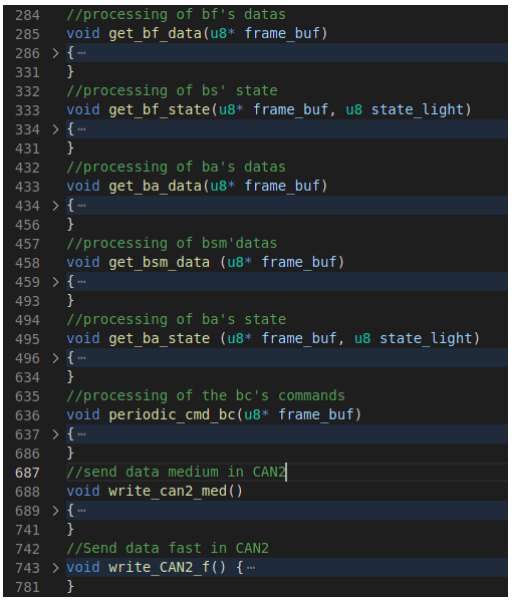


Figure 26 : prototype des fonctions de traitement du code BSI

Une fois la création de fonctions terminée, la présentation à Damien Hardy a eu lieu et nous avons pu passer à la modification de l'acquisition des trames et du traitement. Après la création d'une tâche de traitement afin de séparer la lecture et l'envoi de trame qui devaient travailler de pair le code a été relu par Damien Hardy qui a objecté le dédoublement de tâche. En effet créer une deuxième tâche demande d'être sûr de la synchronisation afin que le traitement soit réalisé immédiatement après la lecture et que jamais les trames ne soient ratées.

Finalement réintégrer le traitement dans la tâche de lecture n'était pas complexe et fut donc rapide. La tâche de lecture à ce moment comportait donc deux grandes parties, une par réseau CAN, chacune séparée également en deux, une partie pour la lecture de toutes les trames reçues, puis une seconde pour le traitement.

Après ces modifications et une relecture par Damien Hardy il a fallu supprimer toutes utilisations du mode de chiffrement, supprimer les restes d'évènement qui rendaient la validation en temps réel compliquée. La tâche d'écriture sur le réseau CAN2 n'avait pas besoin de modifications puisque l'envoi se passait toujours bien, donc au final seule la tâche de lecture et de traitement a eu besoin d'être changée.

Après un débogage du code BSI et plusieurs tests sur la maquette afin de vérifier l'envoi et la réception de trames nous avons pu passer aux tests sur le réseau du véhicule « open-source ». Il aura également fallu faire une validation de l'exécution en temps réel à l'aide d'un outil créé l'année dernière qui permet de vérifier les temps des pires cas d'exécution et s'assurer qu'ils restent dans les bornes.

e) Bilan

Le fonctionnement initial du BSI suffisait pour avoir un véhicule fonctionnel en temps normal mais le risque venait si des erreurs se produisaient. Le problème a été mis en évidence l'année dernière et la perte de trame n'est pas acceptable pour un souci de sécurité. De ce fait sa modification était capitale. Cette correction nous aura permis de mieux comprendre l'utilité des scripts de test et de mieux cerner le fonctionnement d'un BSI dans un véhicule en ayant une idée de ce qu'il se passe à l'intérieur.

Nous avons pu approfondir notre maîtrise du langage C++ et donc d'Arduino qui en est tiré, de trampoline ainsi que voir l'utilité de python en tant que générateur de code et de Linux, même si l'affection pour ce dernier est très hétérogène au sein du groupe.

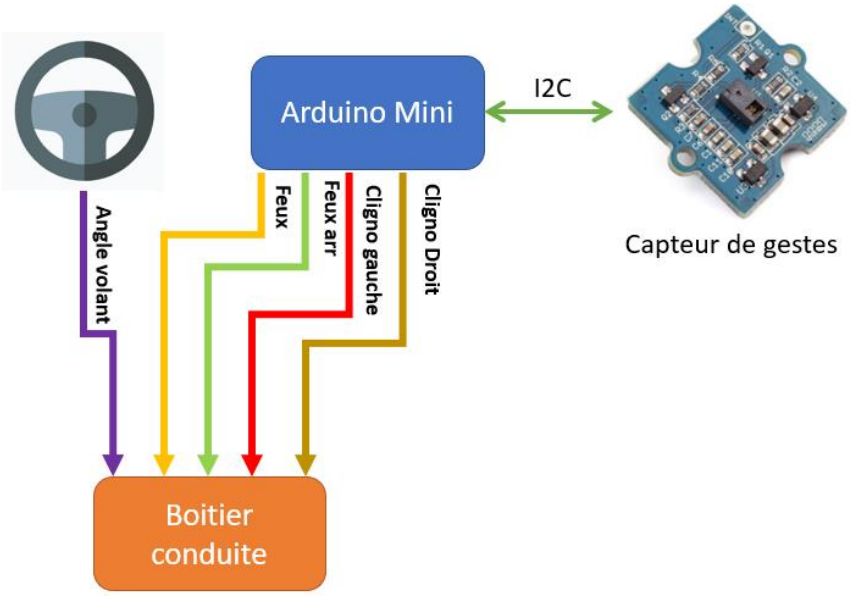
IX- ADAPTATION DES CODES EXISTANTS

a) Fonctionnement des clignotants

L'an passé, les étudiants ont ajoutés une commande des clignotants grâce à un capteur de gestes dans le boitier conduite. Cette année, nous l'avons intégré dans le haut de colonne de l'OSV. Pour cela, nous avons dû adapter le code du boitier conduite car le changement d'état des clignotants est géré par l'Arduino mini dans le haut de colonne.

Dans le code du boitier conduite de 2019, quand il y avait un appui sur un bouton du poste de simulation ou un geste détecté, l'état du clignotant changeait. Maintenant, pour alléger le boitier conduite, nous avons mis cette fonction dans l'Arduino Mini du haut de colonne. En sortie de cet Arduino, nous avons 1 fil par clignotant et feux. Si l'état est à haut, alors le clignotant ou le feu doit être allumé.

Nous avons donc le schéma suivant :



b) Ajout du Bluetooth au boitier conduite

Le dialogue entre l'OSV et le casque de réalité virtuelle se fait via Bluetooth. Auparavant, ce module était directement sur le poste de simulation. Mais pour utiliser les commandes de l'OSV, il a fallu transférer ce module sur le boitier conduite.

Ainsi, le boitier conduite récupère l'information d'accélération et de frein sur le bus CAN et les informations des clignotants du haut de colonne puis les envois au casque de réalité virtuel.

```
if (millis() > (tempsMillis + 100)) {  // ttes les 100ms defini ds messagerie
    CAN.sendMsgBuf(CAN1_ID_CMD_BC_PERIODIC, 0, CAN1_SIZE_FRAME, Trame_CAN_Conduite);
    // envoie BT donneees Analogiques
    BT.print("AngleV="); BT.print(Angle_Volant_BT); BT.print(";");
    BT.print("vitM="); BT.print(PedaleDAc); BT.print(";");
    // envoie BT donneees Binaires
    BT.print("feu="); BT.print(EtatPhares,BIN); BT.print(";");
    BT.print("Recule="); BT.print(EtatFeuxRecul,BIN); BT.print(";");
    BT.print("cliG="); BT.print(EtatClignoG,BIN); BT.print(";");
    BT.print("cliD="); BT.print(EtatClignoD,BIN); BT.println(";");
    tempsMillis = millis();
```

Figure 27 : extrait du code d'envoi Bluetooth

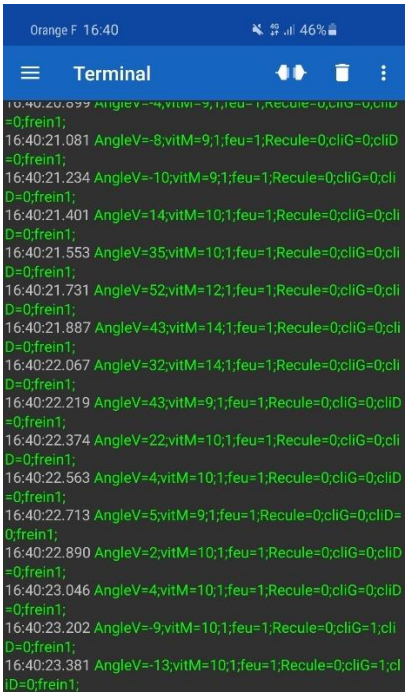


Figure 28 : données BC et BSM envoyés en Bluetooth

Ainsi le système de réalité virtuel peut traiter les données et les reproduire un environnement réaliste.

X- REALISATION DU HAUT DE COLONNE

Nous allons maintenant nous intéresser au haut de colonne de l'OSV. Ce boîtier intègre le potentiomètre qui permet la mesure de l'angle volant, les engrenages d'adaptation et le capteur de gestes pour la commande des clignotants.

L'objectif est de faire un emplacement pour placer le potentiomètre. Pour fixer le haut de colonne, nous réalisons un système à visser.

Toutes ces pièces ont été imprimés en PLA avec une épaisseur de couche de 0,28mm.

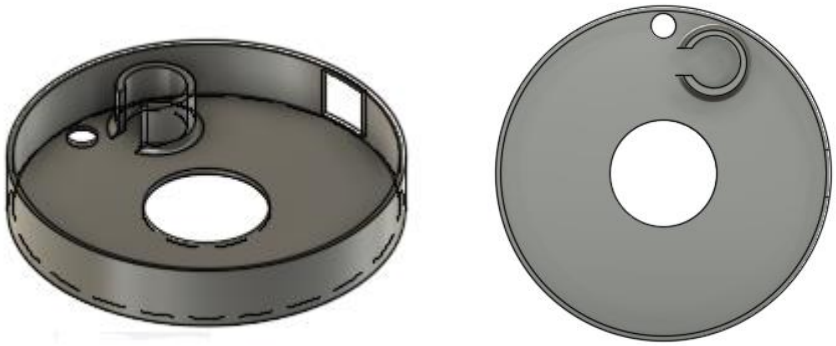


Figure 29 : Haut de colonne

Pour fixer le haut de colonne, nous réalisons un système à visser.

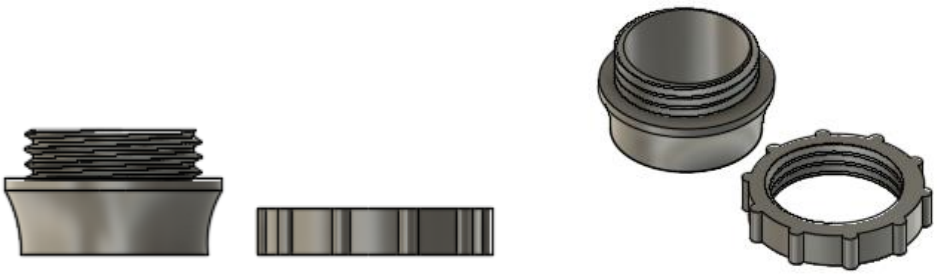


Figure 30 : système de fixation du haut de colonne

Et voici le système installé :

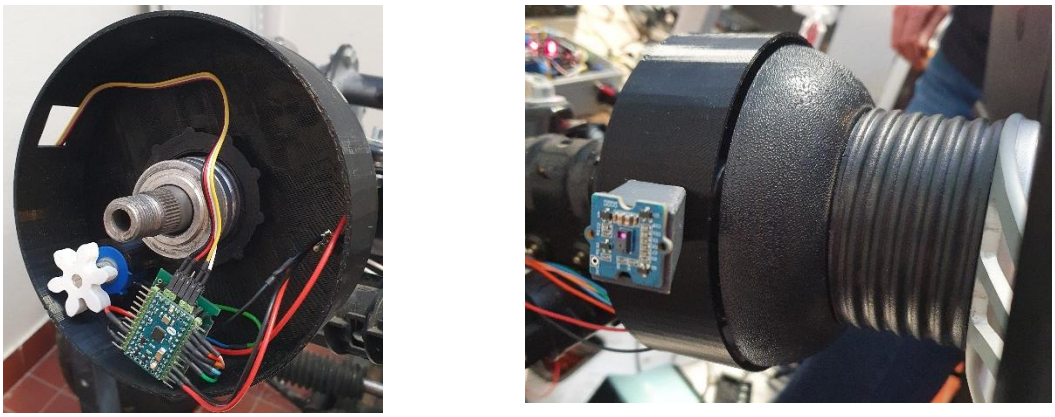


Figure 31 : Haut de colonne sur l'OSV

XI- CONCLUSION

L'aboutissement de ce projet nous aura permis de mettre en œuvre les différents aspects des systèmes embarqués vus en cours cette année. Tout le long du projet nous avons énormément communiqué entre les groupes afin de garder une certaine cohérence, notamment au niveau de la messagerie. Cela n'a pas été trop compliqué mais nous avons vite vu que sans cette communication et cette entraide au sein des groupe nous n'aurions jamais pu terminer le projet.

Le corps principal du projet aura été, comme attendu, la réadaptation du code BSI. En effet, ne pas savoir si l'Arduino est assez puissant pour traiter les nouvelles fonctionnalités rends le challenge plus éprouvant.

D'un autre coté l'intégration des capteurs aura été plus compliquée que prévu. En effet si sur banc de test cela se déroulait comme prévu, en condition réelle (sur l'OSV) nous avons pu voir que les résultats n'étaient pas du tout cohérents. Nous avons donc dû passer plusieurs heures à prendre en compte le bruit des différents boîtiers, des problèmes de masses ou encore des erreurs de calculs.