

h e p i a

**Haute école du paysage, d'ingénierie
et d'architecture de Genève**

Projet Assetto Corsa

Microcontrôleurs et Périphériques

BACHELOR I.S.C

Auteur(s) :

Joachim Bach (22650667)

David Da Silva Marques (226449792)

Institution :

HEPIA Genève

Date :

22 juin 2024

Version :

1.0.0

Professeur-re :

M. Abegg Christian

Table des matières

| | |
|------------------------------------------------------------------------|----------|
| Table des figures | 2 |
| Table des extraits de code | 2 |
| 1 Introduction | 3 |
| 1.1 Présentation du projet | 3 |
| 2 Architecture | 4 |
| 2.1 Schéma de l'architecture | 4 |
| 2.2 Implémentation côté PC | 4 |
| 2.2.1 Détails du protocole entre le PC et le tableau de bord | 5 |
| 2.3 MyLab2 Volant | 6 |
| 2.3.1 Modifications du protocole CAN | 7 |
| 2.4 MyLab2 Tableau de bord | 8 |
| 2.5 Matériel | 8 |
| 3 Problèmes rencontrés | 9 |
| 4 Conclusion | 9 |
| Références | 9 |

Table des figures

| | | |
|---|-------------------------------------------------------|---|
| 1 | Image du volant et du tableau de bord | 3 |
| 2 | Image du jeu avec volant et tableau de bord | 4 |
| 3 | Schéma de l'architecture | 4 |
| 4 | Image du volant | 7 |
| 5 | Image du tableau de bord | 8 |

Table des extraits de code

| | | |
|---|-------------------------------------------------------------------------------|---|
| 1 | Données télémétriques envoyées par le python | 5 |
| 2 | Contenu d'une "trame" de données télémétriques côté tableau de bord | 5 |
| 3 | Variable de ralentissement de l'envoi des données par le python | 6 |
| 4 | Exemple de réception CAN en fonction du champ de commande | 7 |
| 5 | Liste des commandes CAN disponibles | 8 |

1 Introduction

Dans le cadre du cours Microcontrôleur et Périphériques (MIPS), nous avons dû réaliser un projet. Celui-ci a pour but, et exigences, de réunir l'ensemble des interfaces et fonctionnalités étudiées durant ce semestre.

La contrainte principale est que nous devions utiliser deux cartes MyLab2. Nous avions aussi pour obligation d'utiliser les interfaces suivantes :

1. CAN
2. Écran avec texte
3. Communication UART
4. Dalle tactile
5. SysTick / Timer
6. Boutons

Le code doit être le même sur les deux cartes, la différence se faisant au lancement en fonction de la position du dipswitch 0.

1.1 Présentation du projet

Le projet que nous avons choisi consiste à développer une sorte de manette de jeu pour jouer au simulateur de course automobile "Assetto Corsa"¹.

Voici une description de l'utilisation des deux MyLab2 : Une des deux cartes sert de volant, grâce à l'accéléromètre, avec des boutons pour l'accélérateur et le frein. Les LEDS du dipswitchs sont utilisées comme indicateur de compte-tours. L'autre carte sert de tableau de bord affichant la télémétrie du jeu et ayant un ensemble de boutons pour permettre quelques inputs supplémentaires.

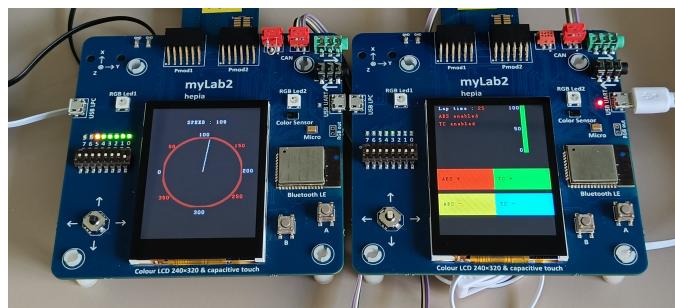


FIGURE 1 : Image du volant et du tableau de bord

¹ Assetto Corsa. URL : https://store.steampowered.com/app/244210/Assetto_Corsa/.

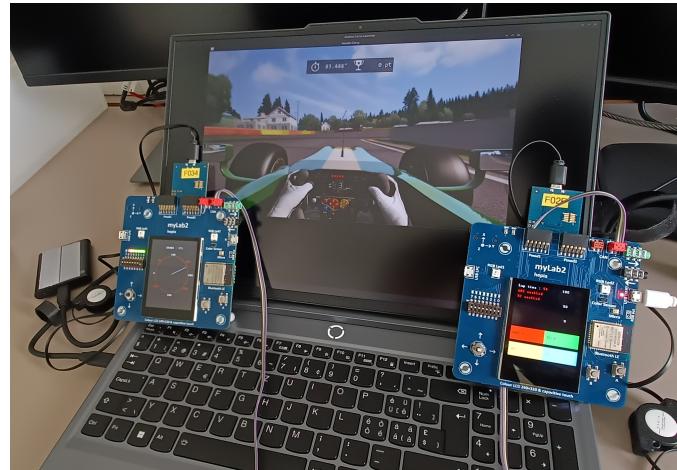


FIGURE 2 : Image du jeu avec volant et tableau de bord

2 Architecture

Dans cette section de notre rapport, nous allons présenter les différentes parties composant notre périphérique. Il s'agira de discuter de la partie PC, de la carte servant de volant ainsi que de celle servant de tableau de bord.

2.1 Schéma de l'architecture

Voici un schéma conceptuel décrivant l'architecture mise en place dans le cours de ce travail pratique :

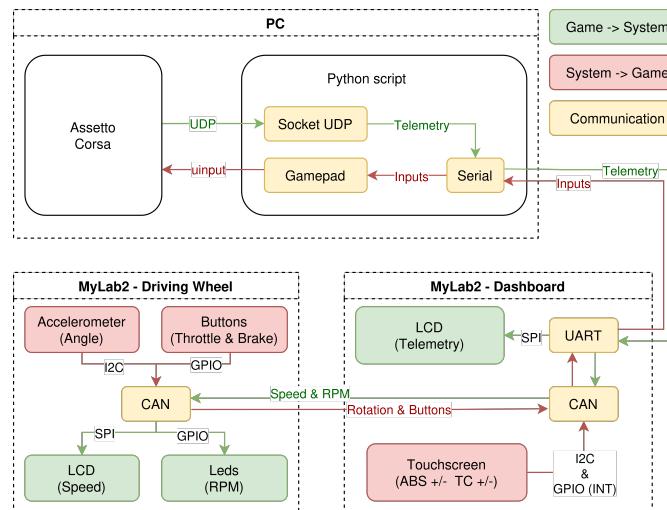


FIGURE 3 : Schéma de l'architecture

2.2 Implémentation côté PC

Du côté du PC se trouve Assetto Corsa, un jeu de voiture ainsi qu'un script python tournant en arrière-plan. Ce script python se charge de récupérer la télémétrie d'Assetto Corsa à travers un

socket UDP mis à disposition par le jeu²³ (il s'agit d'une méthode d'accès à la télémétrie courante dans le monde des simulateurs). Le python récupère alors les informations qui l'intéressent et les envoie sur l'UART vers le "Tableau de bord", autrement dit l'une des deux cartes.

Depuis cette même connexion UART, le script récupère les inputs (inclinaison du volant, pression des boutons, etc.) sous forme de tableau d'octets envoyé par la carte tableau de bord. Ces informations sont ensuite transmises à une classe "Gamepad" qui simule un périphérique qui est ensuite utilisée par le jeu comme un n'importe quel autre périphérique/manette/etc.

2.2.1 Détails du protocole entre le PC et le tableau de bord

Le code python s'exécutant sur le PC est chargé à la fois d'envoyer les données de télémétrie, mais aussi de recevoir les inputs provenant du tableau de bord.

Pour l'envoi de la télémétrie, il envoie en UART un ensemble de données récupérées via le port UDP du jeu. Voici l'ensemble des données composant une trame :

Extrait de code 1 : Données télémétriques envoyées par le python

```
uart.serial_send(struct.pack("?", False)) #reset screen bool
uart.serial_send(struct.pack("f", live_data['speed_Kmh']))
uart.serial_send(struct.pack("I", live_data['lapTime']))
uart.serial_send(struct.pack("f", live_data['gas']))
uart.serial_send(struct.pack("f", live_data['brake']))
uart.serial_send(struct.pack("I", int(live_data["engineRPM"])))
uart.serial_send(struct.pack("?", live_data['isAbsEnabled']))
uart.serial_send(struct.pack("?", live_data['isTcEnabled']))
uart.serial_send(struct.pack("I", message_counter))
```

Du côté du tableau de bord, la réception de l'UART se charge de remplir une structure contenant toutes les informations nécessaires à un traitement de données (une sorte de trame complète).

Voici le contenu de cette "trame" :

Extrait de code 2 : Contenu d'une "trame" de données télémétriques côté tableau de bord

```
typedef struct __attribute__((__packed__)) _uart_telemetry
{
    bool stop_display;
    float speed_kmh;
    uint32_t lap_time;
    float gas;
    float brake;
    uint32_t engine_RPM;
    bool is_abs_enabled;
    bool is_tc_enabled;
    uint32_t message_counter;
}uart_telemetry;
```

Une donnée supplémentaire envoyée par le python est le "message_counter". Cela permet au tableau de bord de détecter une erreur, dans le cas où le message counter n'est pas le même que le sien, et ainsi provoquer un reset de la communication. Cela peut arriver si le script python

² AC Remote Telemetry Documentation. URL : <https://docs.google.com/document/d/1KfkZiIluXZ6mMhLWfDX1qAGbvhGRC3ZUzjVIt5FQpp4/pub>.

³ rickwest/ac-remote-telemetry-client. URL : <https://github.com/rickwest/ac-remote-telemetry-client/tree/master?tab=readme-ov-file>.

n'est pas correctement synchronisé avec le tableau de bord, entraînant un remplissage erroné de la structure d'une trame.

Le code python peut aussi demander au tableau de bord de se réinitialiser, dans le cas où, par exemple, l'utilisateur quitte une course. En effet, si le jeu ne s'exécute plus, les données de télémétrie ne sont plus envoyées et il faut alors "nettoyer" les écrans du tableau de bord et du volant afin qu'ils n'affichent pas de données erronées.

Il est très important de noter qu'il y a une limite à la vitesse d'envoi du python, qu'il est très important de ne pas dépasser. Cette limite est assurée par la variable globale suivante :

Extrait de code 3 : Variable de ralentissement de l'envoi des données par le python

```
MESSAGE_SENT_WAIT_MICROSECONDS = 100000
```

Si l'envoi devient trop rapide, il est possible que le tableau de bord n'arrive pas à suivre et doive alors se réinitialiser en permanence.

Le script gère les coupures de connexion avec le jeu et/ou la carte, tout comme les cartes gèrent les pertes de connexion entre elles et avec le PC, affichant la situation actuelle à l'utilisateur de la manière suivante :

```
AC TELEMETRY & VIRTUAL GAMEPAD:
```

```
-----  
Assetto Corsa link : NOT CONNECTED  
Dashboard link : NOT CONNECTED
```

Les états possibles pour "Assetto Corsa link" sont :

1. "NOT CONNECTED", la connexion n'a pas encore été établie.
2. "TRYING HANDSHAKE", le script tente de faire un handshake avec Assetto Corsa dans le but de recevoir les données de télémétrie.
3. "TIMEOUT", le script n'a pas reçu de réponse de la part d'Assetto Corsa depuis un moment, indiquant que le jeu s'est arrêté ou qu'il y a un problème.
4. "CONNECTED", le script reçoit les données correctement.

Les états possibles pour "Dashboard link" sont :

1. "NOT CONNECTED", le câble UART n'est pas branché.
2. "RESETING", le script se reset pour corriger un problème de mismatch avec le dashboard.
3. "TIMEOUT", le script n'a pas reçu de donnée du dashboard depuis un moment
4. "CONNECTED", le script reçoit les inputs du dashboard correctement.

2.3 MyLab2 Volant

La carte MyLab2 servant de volant est celle tenue entre les mains du joueur. Son rôle est de simuler un volant de course et doit ainsi permettre de guider la voiture, mais aussi de contrôler l'accélérateur, les freins et d'afficher quelques informations sur la voiture.

Le volant s'occupe alors de récupérer les valeurs de son accéléromètre et de les convertir en un angle. Il envoie en continu le résultat de ce calcul par CAN à la carte servant de tableau de bord. Lors de l'appui sur le bouton A ou B, un message CAN est transmis immédiatement au tableau de bord.

Le volant reçoit sur le CAN les RPM du moteur et la vitesse de la voiture. Ceux-ci seront affichés respectivement sur les LEDS des dipswitchs et sur le LCD sous forme de cadran avec une aiguille.

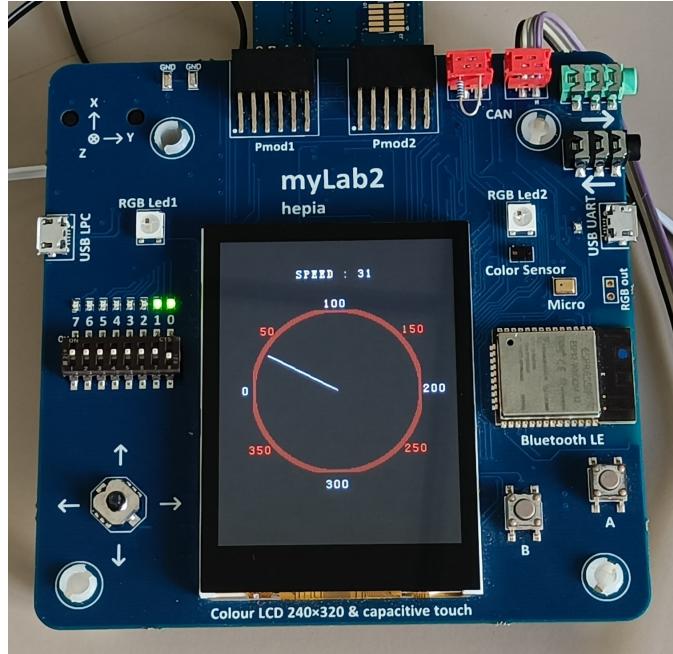


FIGURE 4 : Image du volant

2.3.1 Modifications du protocole CAN

Le protocole CAN a été légèrement modifié afin d'ajouter une zone de commande ainsi qu'une zone de données dans chaque message. Les 8 premiers bits sont utilisés pour signaler la commande utilisée. Cela permet de faire un traitement en fonction du type de message reçu. Voici l'exemple de la réception CAN du volant :

Extrait de code 4 : Exemple de réception CAN en fonction du champ de commande

```
uint32_t received_id;
uint8_t* received_data;

can_get_message(&received_id, &received_data);

if(received_data[0] == CAN_SPEED_DATA_NUMBER)
{
    uint32_t speed = (received_data[1] + (received_data[2] << 8) + (received_data[3] << 16) + (received_data[4] << 24));
    gui_draw_speed(80, 30, speed);

    gui_draw_speedometer(120, 160, 80, speed);
}
else if(received_data[0] == CAN_RPM_DATA_NUMBER)
{
    uint32_t engine_RPM = (received_data[1] + (received_data[2] << 8) + (received_data[3] << 16) + (received_data[4] << 24));

    gui_display_shift_indicator_leds(engine_RPM, CAR_MAX_RPM);
}
```

Voici l'ensemble des commandes différentes possibles actuellement :

Extrait de code 5 : Liste des commandes CAN disponibles

```
#define CAN_RESET_CMD_NUMBER 0
#define CAN_SPEED_DATA_NUMBER 1
#define CAN_BTN_A_DATA_NUMBER 2
#define CAN_BTN_B_DATA_NUMBER 3
#define CAN_RPM_DATA_NUMBER 4
#define CAN_WHEEL_ROTATION 5
```

2.4 MyLab2 Tableau de bord

Le tableau de bord est la pièce centrale du projet : cette carte sert d'intermédiaire entre le volant et le PC. Le tableau de bord reçoit à travers l'UART (Ref section 2.2.1) la télémétrie venant du jeu, dont il affiche l'accélérateur et le frein, mais aussi le temps du tour (lap time). Il retransmet ensuite la vitesse et les RPM au volant via CAN (Ref section 2.3.1). De celui-ci il récupère la rotation et les boutons, qu'il stocke dans un tableau local. Toutes les 20ms, il envoie alors ce tableau par la même connexion UART pour que le PC puisse interpréter les inputs.

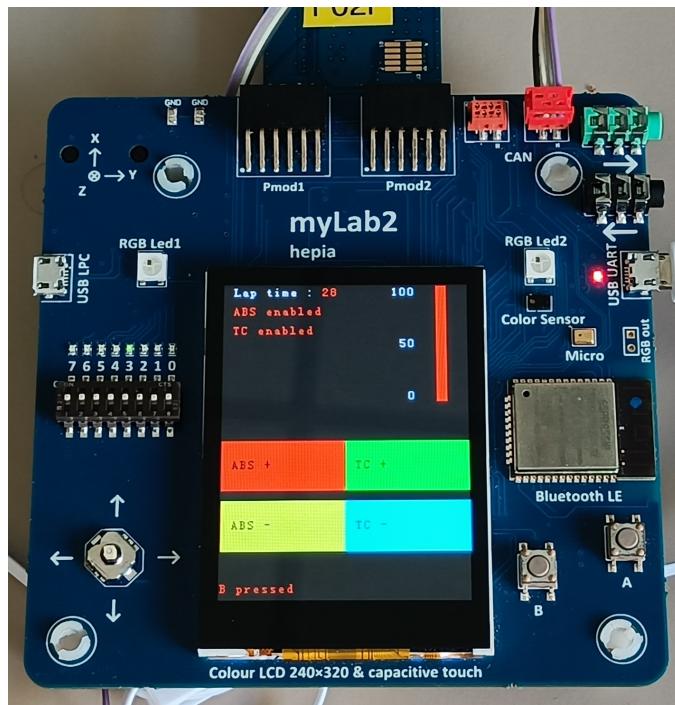


FIGURE 5 : Image du tableau de bord

2.5 Matériel

Pour ce projet nous avons donc besoin du matériel suivant :

1. 1 PC avec Assetto Corsa et 1 script python permettant d'interfacer avec les cartes
2. 2 MyLab2 + LPC1769
3. 1 câble USB A → micro-USB (UART)
4. 1 câble CAN
5. 2 Câbles d'alimentation

3 Problèmes rencontrés

Initialement il avait été prévu que les cartes soient utilisé en tant périphériques USB. Cependant, il s'avère qu'il s'agit d'un protocole relativement sensible et précis, rendant compliqué son implémentation dans un délai aussi court. Heureusement un plan de secours avait été prévu : utiliser un script python couplé à de l'UART pour la transmission des données pour pouvoir simuler un périphérique logiciellement. Cependant, cette solution n'est pas miraculeuse et ne fonctionne pas de manière 100% stable.

L'installation du jeu Asseto Corsa sous Linux n'est malheureusement pas aussi simple que ça. Il faut en effet ajouter une couche de compatibilité car il s'agit d'un jeu créé pour Windows. Bien que l'ajout de Proton GE⁴ aide et permette de lancer le jeu, il est assez fréquent qu'il crash suivant le PC qui le lance. Il arrive par exemple que le jeu freeze complètement lorsqu'il est en plein écran sur l'un de nos PC. De plus, cela ne garanti pas que le jeu fonctionnera toujours ni qu'une mise à jour ne viendra pas le rendre inopérant.

L'utilisation de la librairie python Uinput fut une très grande perte de temps. En effet, le peu de documentation disponible et l'impossibilité de valider le fonctionnement en période de test nous a fait perdre un temps considérable. De plus, nous avons constaté que les modifications (p.ex ajout d'une touche) ne sont pas nécessairement prise en compte immédiatement et qu'il faut parfois redémarrer son ordinateur ou faire redémarrer Uinput (ce qui n'est pas indiqué), ce qui nous a causé d'immenses soucis lors de l'ajout de contrôles.

4 Conclusion

Le résultat de ce projet nous est très satisfaisant. Il est possible de contrôler la voiture avec le volant et le tableau de bord, l'affichage des données en temps réel fonctionne à merveille.

Cependant, il ne respecte pas entièrement le cahier des charges que nous avions proposés, car il n'utilise malheureusement pas le protocole USB. Cela n'impacte pas la fonctionnalité de l'ensemble, mais il est dommage de ne pas avoir réussi à implémenter ce nouveau périphérique. Rétrospectivement, il nous aurait fallu plus de temps et de connaissances afin d'y parvenir.

Il serait intéressant de fiabiliser encore plus l'ensemble du projet. Bien que nous ayons implémenté diverses fonctions de reconnexion, redémarrage automatique, contrôle de numéro de trames, il est toujours possible d'améliorer cela afin de le rendre utilisable dans 100% des cas sans erreurs ou comportement imprévisible.

Pour conclure, nous souhaitons remercier M. Abegg Christian pour son aide apportée au long de ce travail pratique.

⁴ GloriousEggroll/proton-ge-custom. URL : <https://github.com/GloriousEggroll/proton-ge-custom>.

Références

- [1] *AC Remote Telemetry Documentation*. URL : <https://docs.google.com/document/d/1KfkZiIluXZ6mMhLWfDX1qAGbvhGRC3ZUzjVIt5FQpp4/pub>.
- [2] *Assetto Corsa*. URL : https://store.steampowered.com/app/244210/Assetto_Corsa/.
- [3] *GloriousEggroll/proton-ge-custom*. URL : <https://github.com/GloriousEggroll/proton-ge-custom>.
- [4] *rickwest/ac-remote-telemetry-client*. URL : <https://github.com/rickwest/ac-remote-telemetry-client/tree/master?tab=readme-ov-file>.