Ecole Publique d'Ingénieurs en 3 ans

Rapport

PROJET RACE

Date de soutenance

27/08/20

Brice Poignant
Année Universitaire 2019/2020
Electronique et physique appliquée / SATE

Tuteur ENSICAEN : **Hugo DESCOUBES**

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

www.ensicaen.fr

TABLE DES MATIERES

TABLE DES FIGURES

TABLE DES TABLEAUX

1. Introduction

Le projet RACE (Remote Automotive Challenge of ENSICAEN) consiste au développement du firmware d'une voiture télécommandée depuis un ordinateur.

Figure 1 : Représentation schématique du projet

Pour contrôler la voiture, on utilise une carte PIC32 de Microchip avec une carte wifi WINC1500. La liaison, entre ces deux cartes, se fait avec une carte mère créé par Filipe Lima de Melo lors de son stage en 2018. Elle incorpore aussi les éléments de puissance nécessaire pour contrôler le moteur de la voiture.

Figure 2 : Microcontrôleur PIC32

Figure 3 : Carte mère RACE v1

Figure 4 : Carte wifi WINC 1500

Le firmware se code en utilisant Harmony sur l'interface de travail MPLAB. Ce logiciel propose des exemples pré-codés que l'on utilisera afin de faciliter la programmation.

Figure 5 : Logos de Harmony et de X IDE de MPLAB

L'interface QT utilisée est créée en réutilisant le travail de Martin CUVELIER, un ancien élève de SATE. Cependant certains ajustements seront faits au programme afin d'améliorer la sécurité de la connexion par exemple.

Figure 6 : Logo de QT

2. Prototypage wifi

2.1. Routage

Le routage, avec la carte prototype, est assez difficile car il y a 2 intermédiaires avant que les pins du connecteur J1 du PIC32 atteignent le module wifi.

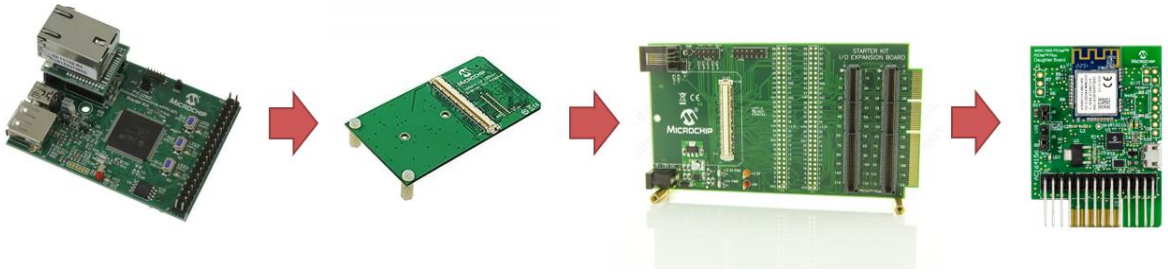

Figure 7 : Représentation de la connectique du prototype

Il faut donc retracer le passage du module wifi vers le pic32 pour chaque pin afin de savoir quel port utiliser pour les différentes fonctions. On utilise les ressources données par le site Microchip pour connaître les caractéristiques des connecteurs du PIC32 [1], celles de la carte adaptateur [2], puis celles de la carte expansion [3] et enfin celles du module wifi [4].

	RESET	WAKE	IRQ	ENABLE	SS	SDO	SDI	SCK
Carte Wifi Pictail	28	14	18	30	1	7	5	3
Expansion *J1*	18 RF0	64 AN4	85 INT1	16 RF1	97 SS1	95 SDO1	93 SDI1	91 SCK1
Adaptateur *J1*		103	93		95	114	94	118
PIC32		RB4	RE8		RE9	RD10	RD14	RD1

Tableau 1 : Table de routage du prototype wifi

A noter qu'il est nécessaire de mettre deux cavaliers sur la carte d'adaptation 168 -> 122 pins afin que les pins enable et reset puissent correctement fonctionner.

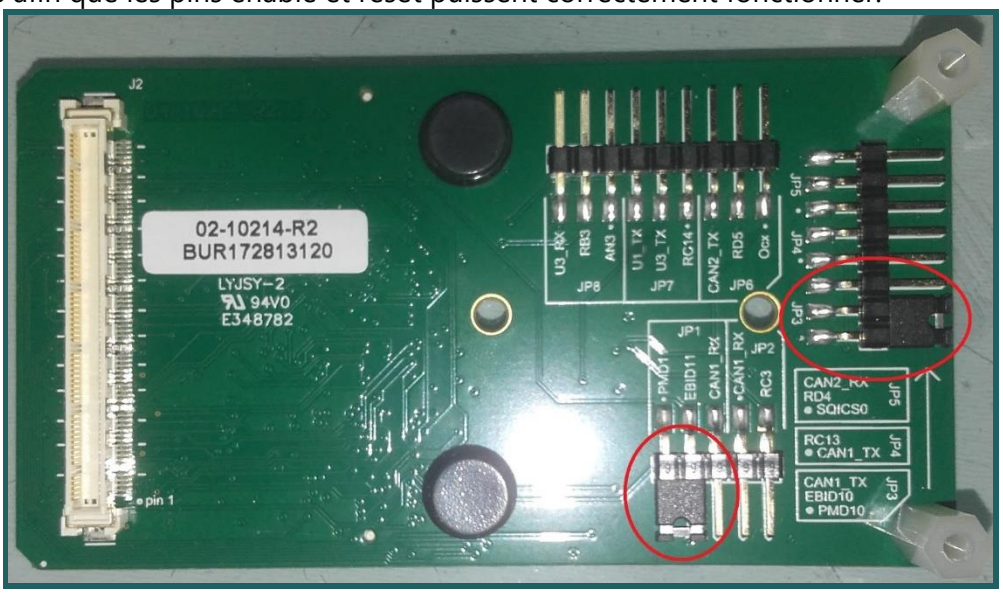

Figure 8 : Verso de la carte d'adaptation 168 -> 122 pins

Aussi le pin WAKE n'étant pas utilisé par Harmony il n'est pas nécessaire de l'initialiser.

2.2. Configuration Harmony

La configuration d'Harmony est assez complexe car le module wifi requiert de nombreuses dépendances. Un guide fournit par Microchip permet de définir tous les modules nécessaires [5].

Les seules différences notables prises avec le guide sont les pins utilisés ainsi que l'utilisation du SoftAP pour le driver wifi. Contrairement au mode infrastructure qui oblige de passer par l'intermédiaire d'un routeur, on peut se connecter directement en wifi à la carte.

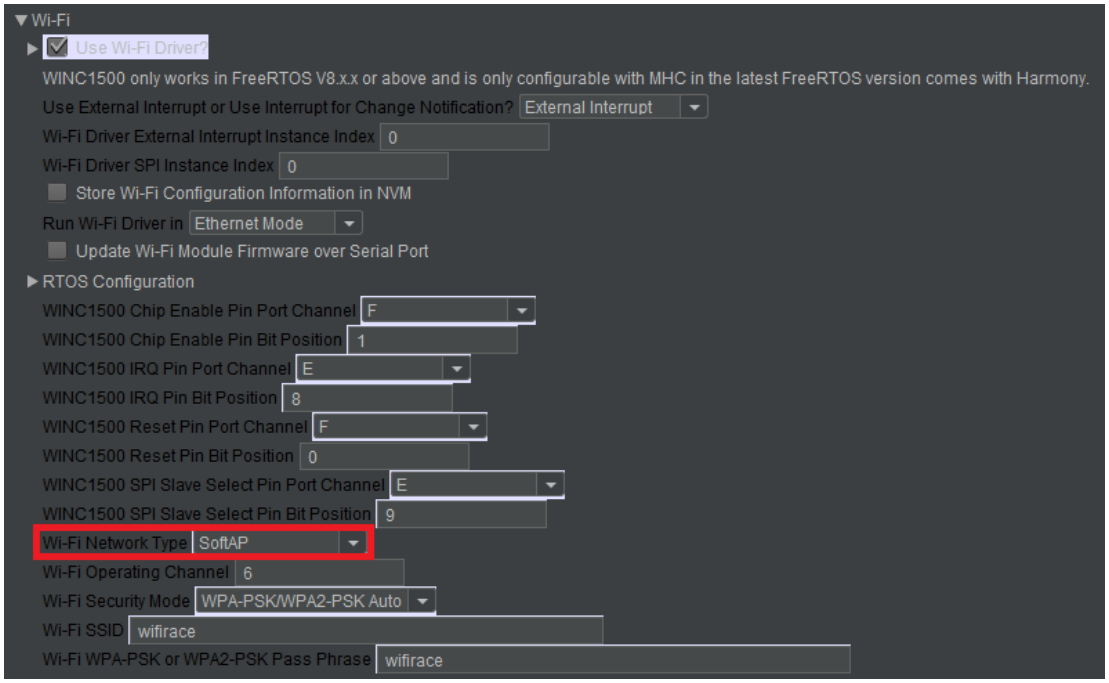

Figure 9 : Configuration du module wifi sur Harmony

Cependant, il est nécessaire de mettre en place un serveur DHCP dans la stack TCP/IP afin que le module wifi puisse donner une adresse IP à l'appareil qui se connecte.

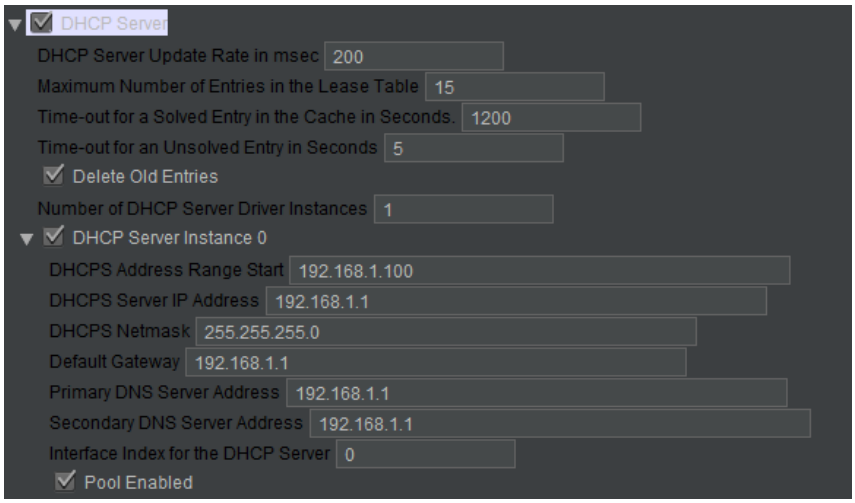

Figure 10 : Configuration du serveur DHCP sur Harmony

Aussi, il faut configurer un flag pour le DHCP que l'on trouve sous la configuration du network comme indiqué dans l'image suivante.

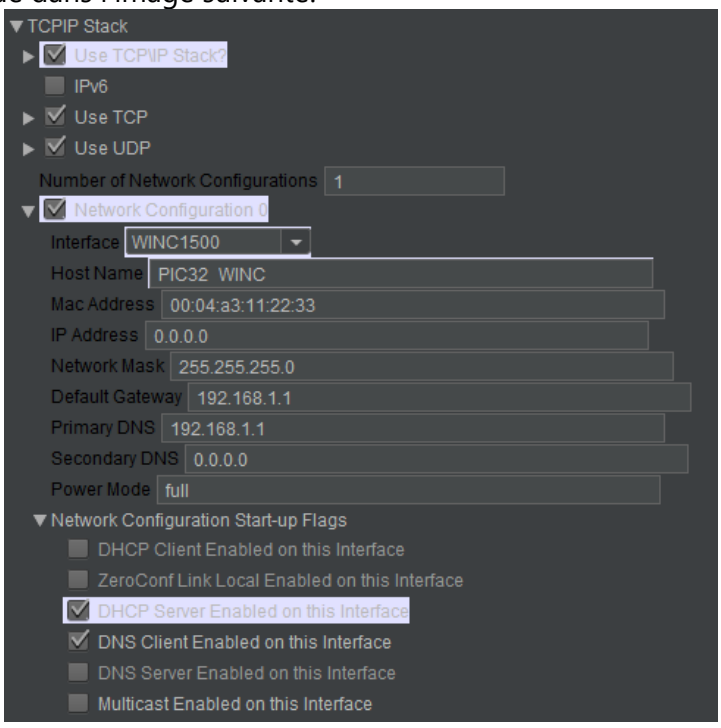

Figure 11 : Configuration de la stack TCP/IP sur Harmony Code

Le code de la stack TCP/IP est automatiquement généré par Harmony. Cependant ce code seul ne permet pas d'être performant, il faut ajouter quelques lignes de code comme décrit dans le chapitre IV du guide mentionné précédemment [5].

Pour nos besoins nous devons nous assurer de la présence de ces deux éléments :

- La réception et la mise à jour de la commande moteur et de la direction.
- L'envoi de la consommation de puissance.

```
static char wnetcomMsgToClient[] = "Hello Client!\n\r";
static char wnetcomOkToClient[] = "Msg reçu!\n\r";
```

```
if (TCPIP_TCP_GetIsReady(wnetcomData.socket))
{
    TCPIP_TCP_ArrayGet(wnetcomData.socket, wnetcomMsgFromClient, sizeof(wnetcomMsgFromClient) - 1);
    SYS_CONSOLE_PRINT("TCP_TXRX[%d]: Client sent: %s\r\n",
            wnetcomData.txrxTaskState, wnetcomMsgFromClient);
    TCPIP_TCP_ArrayPut(wnetcomData.socket, wnetcomOkToClient, sizeof(wnetcomOkToClient));
    memset(&wnetcomMsgFromClient[0], 0, sizeof(wnetcomMsgFromClient));
}
```

Figure 12 : Code de l'état "Réception de trame sur le socket"

2.3. Interface QT

2.3.1. Connexion

Pour l'application QT, j'ai repris le travail de Martin en le modifiant un peu.

Auparavant, le bouton de connexion redirigeait directement à l'écran de contrôle de la voiture sans vérifier si la connexion était bien établie ou non. J'ai décidé de changer le comportement en modifiant la fonction *connexion()* liée à l'appui du bouton « *connection* ».

```
void
Fenetre::connexion()
{
    connexion_boutton->setEnabled(false);
    soc->abort();   //désactive les anciennes connexions
    connexionok=false;
    soc->connectToHost(IP,port);

    connect(soc, SIGNAL(readyRead()), this, SLOT(receive()));

    QTimer *timers = new QTimer(this);
    timers->setSingleShot(true);
    connect(timers, SIGNAL(timeout()), this, SLOT(checkconnexion()));
    timers -> start(100);

}
```

Figure 13 : Fonction liée au bouton « connection »

```
void
Fenetre::receive()
{
    QByteArray datas = soc->readAll();
    QString msg=QString(datas);
    if (msg == "Hello Client!\n\r") {
        connexionok=true;
    }
    else if (msg != NULL) {
        emit newPwr(msg);
    }

}
```

Figure 14 : Fonction liée à la réception du trame sur le socket

On connecte alors le signal de réception du socket à la fonction *receive()*. Si dans les 100ms, la trame « Hello Client!\n\r » est reçue, alors la connexion est approuvée dans la fonction *checkconnexion()*.

```
void
Fenetre::checkconnexion()
{
    if (connexionok) {
        MainWindow(true);
        connect(soc, SIGNAL(disconnected()), this, SLOT(errconnexion()));
    }
    else {
        disconnect(soc, SIGNAL(readyRead()), this, SLOT(receive()));
        soc->abort();
        warningcoWindow();
        connexion_boutton->setEnabled(true);
    }

}
```

Figure 15 : Fonction qui vérifie si la trame d'accueil a bien été reçue

```
void
Fenetre::errconnexion()
{
    soc->abort();
    connexionok=false;
    MainWindow(false);
    warningcoWindow();
}
```

Figure 16 : Fonction liée à une erreur due à la connexion

Si la connexion est approuvée, alors l'écran de connexion passe à l'écran de contrôle de la voiture. Sinon, il fait apparaître un message d'erreur et reste à l'écran de connexion. Dans le cas où elle est approuvée elle connecte le signal de déconnexion à une fonction *errconnexion()* qui fait un retour à l'écran de connexion avec un message d'erreur. Cela permet donc d'être toujours informé de l'état de connexion de la voiture.

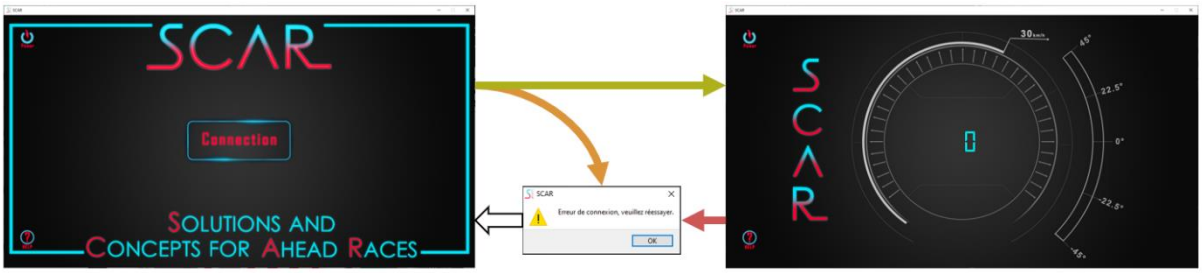

Figure 17 : Schéma du fonctionnement de l'interface

2.3.2. Ajout du graphique

Dans la fonction *receive()* vue précédemment, on envoie un signal *newPwr(QString)* qui contient l'information sur la consommation de puissance de la voiture. Afin d'afficher cette valeur, on utilise un exemple de graphique fournit par QT que l'on modifie pour nos besoins.

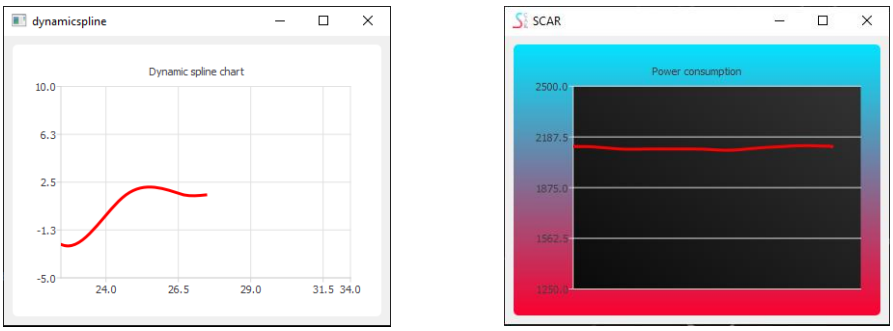

Figure 18 : A gauche le graphique par défaut, à droite le graphique personnalisé pour l'application

Il suffit alors de connecter notre signal à une nouvelle fonction nommée *newPwr()* dans le code du graphique. Elle prend la place d'une fonction mettant des valeurs aléatoires sur l'axe y.

```
QObject::connect(&ma_fenetre, SIGNAL(newPwr(QString)), chart, SLOT(newPwr(QString)));
```

Figure 19 : Connexion entre la réception de la consommation et le graphique

On a alors une fenêtre, externe à l'application, qui permet de connaître la consommation de la voiture télécommandée.

3. Prototypage hardware

3.1. Routage

Pour la carte RACE, le routage est beaucoup plus simple qu'avec les cartes de prototypage car il n'y a pas d'intermédiaire entre les connecteurs.

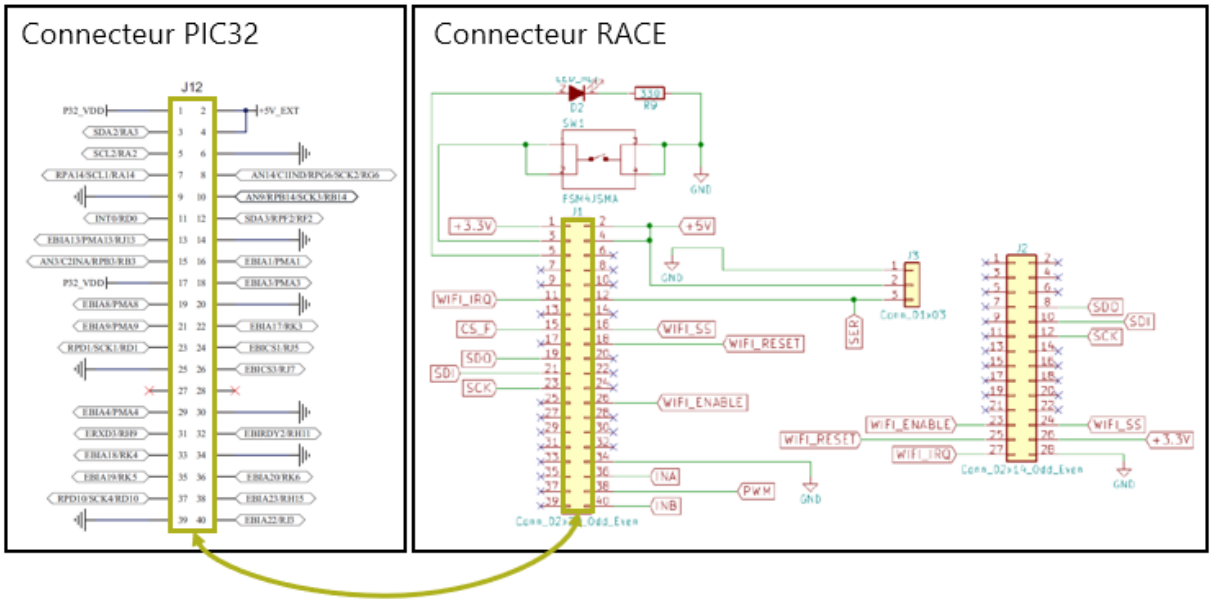

Figure 20 : A gauche le connecteur J12 du pic32 et à droite le schéma de la carte mère RACE

Il suffit alors de relier les fonctions dont on a besoin pour le contrôle du moteur et de la direction. CS est la sortie de l'information sur la consommation en puissance du contrôleur. SER et PWM sont des modulateurs de largeur d'impulsion qui servent respectivement à contrôler l'angle de la direction des roues et l'accélération du moteur. INA et INB sont des pins qui servent à déterminer si les moteurs doivent accélérer vers l'avant ou vers l'arrière, INA allant vers l'avant et INB vers l'arrière.

Fonction	CS	SER	PWM	INA	INB
RACE mb	15	12	28	36	40
PIC 32	RB3 AN3	RF2 OC2	RD10 OC3	RK6 GPIO_OUT	RJ3 GPIO_OUT

Tableau 2 : Table de routage du prototype hardware

Il est nécessaire d'initialiser des Timers avec Harmony afin de faire fonctionner le module ADC ainsi que le SER et le PWM.

Fonction	ADC	SER	PWM
Module utilisé	AN3	OC2	OC3
Timer	3	4	5
PRESCALE	64	64	8
PERIOD	31250	31250	625
Fréquence	50 Hz	50 Hz	20 kHz

Tableau 3 : Tableau de configuration HARMONY des modules

Le Timer 2 doit être laissé car il est déjà utilisé par le module wifi. Il sera nécessaire pour la suite du projet. Pour trouver les valeurs du PRESCALE et du PERIOD, il suffit d'utiliser l'équation suivante en fonction de la fréquence voulue sachant que $fclk$ =100 MHz.

$$f = \frac{fclk}{PRESCALE * PERIOD}$$

Équation 1 : Formule pour la configuration des timer

3.2. Initialisation ADC

Dans *system_interrupt.c,* il faut ajouter une fonction afin que le programme puisse compiler. On ajoute d'abord une référence externe vers l'application de test.

```
extern APP_DATA appData;
```

On modifier la variable *current* à chaque itération de la fonction qui est liée au timer3. Il suffit ensuite d'utiliser les fonctions *DRV_ADC0_Open()* et *DRV_TMR_Open()* dans la partie initialisation de *app.c*.

```
void IntHandlerDrvAdc(void)
{
    /* Clear ADC Interrupt Flag of INT_SOURCE_ADC_1*/
    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_ADC_1);

    if (DRV_ADC_SamplesAvailable(3)){
        appData.current=DRV_ADC_SamplesRead(3);
    }
}
```

Figure 21 : Fonction nécessaire pour la compilation du projet

3.3. Test avec une application

3.3.1. Code de l'application

Afin de tester les modules, on met en place une application afin de tester toutes les valeurs possibles des OC ainsi que le bon fonctionnement de l'ADC.

```
uint32_t min0 = 1900 ;
uint32_t max0 = 3100 ;
```

```
uint32_t min1 = 60 ;
uint32_t max1 = 400 ;
```

Figure 22 : A gauche, les valeurs minimales et maximales de OC2 sur 31250, à droite les valeurs minimales et maximales de OC3 sur 625.

Ces valeurs sont déterminées en testant préalablement le PWM et le SER sur la voiture télécommandée. Le SER a un angle libre faible, donc on passe de 6% à 10% du OC pour la direction. Concernant le contrôle des moteurs on commence directement à 10% du PWM car, avant cela, le moteur ne démarre pas. Le maximum est arbitrairement choisi à 65% du PWM.

```
case APP_OC_DONE:

    sprintf(textToWrite,"%lu", appData.current);
    SYS_CONSOLE_PRINT(textToWrite);
    SYS_CONSOLE_PRINT("\r\n");

    int i;

    for (i = 0 ; i < 100 ; i++)
    {
        DRV_OC_PulseWidthSet(appData.handleOC0, min0+i*(max0-min0)/100);
        DRV_OC_PulseWidthSet(appData.handleOC1, min1+i*(max1-min1)/100);
        BSP_DelayUs(10);
    }
    for (i = 99 ; i > 0 ; i--)
    {
        DRV_OC_PulseWidthSet(appData.handleOC0, min0+i*(max0-min0)/100);
        DRV_OC_PulseWidthSet(appData.handleOC1, min1+i*(max1-min1)/100);
        BSP_DelayUs(10);
    }
```

Figure 23 : Image du code utilisé pour tester les modules.

Convertie la variable current d'entier à une chaîne de caractère et l'envoi sur la console UART.	Modifie la valeur de l'OC2 correspondant au contrôle de la direction du moteur (SER)	Modifie la valeur de l'OC3 correspondant au contrôle du moteur (PWM)

On teste ce code d'abord en observant les signaux émis sur un oscilloscope en utilisant les points de test prévus à cet effet.

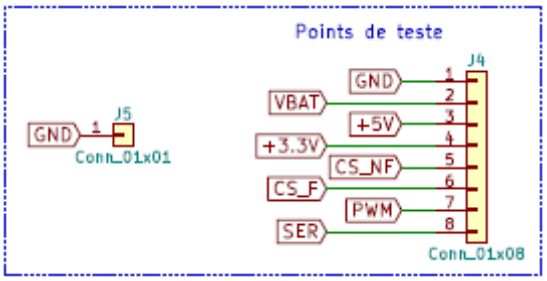

Figure 24 : Représentation schématique des points de test de la carte RACEv1

3.3.2. Résultat SER

On observe bien que l'on atteint 50 Hz et que la direction réagit parfaitement aux commandes souhaitées.

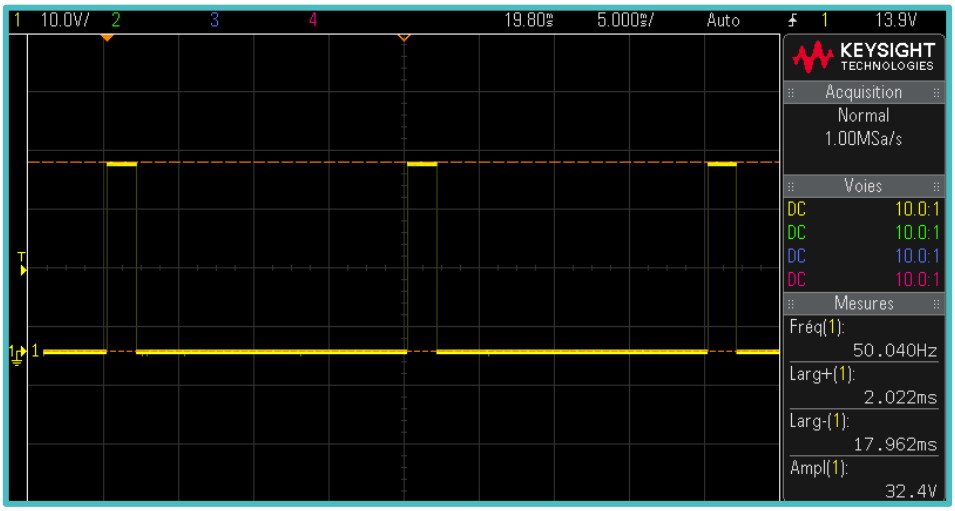

Figure 25 : Résultat du contrôle de la direction de la voiture sur un oscilloscope.

3.3.3. Résultat PWM

On observe bien que l'on atteint 20 kHz et que le moteur de la voiture réagit parfaitement aux commandes souhaitées.

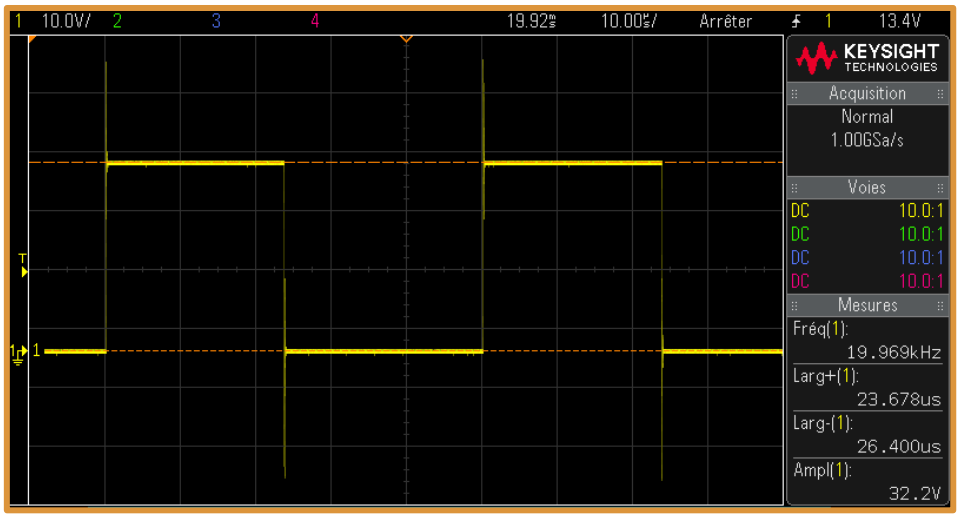

Figure 26 : Résultat du contrôle du moteur de la voiture sur un oscilloscope.

3.3.4. Résultat ADC

En utilisant la console UART, on remarque que la valeur donnée correspond à la consommation du moteur de la voiture comme prévue, car elle change lors de l'accélération et la décélération.

Figure 27 : Résultat de la réception UART

4. Implémentation des deux projets ensemble

4.1. Routage

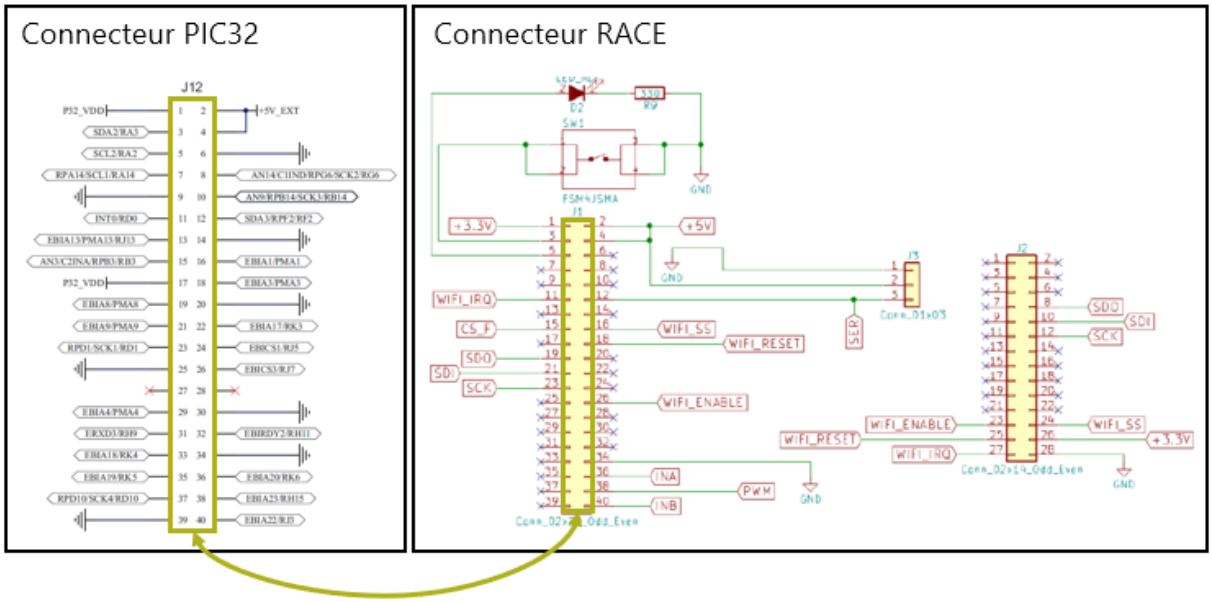

Figure 28 : A gauche le connecteur J12 du pic32 et à droite le schéma de la carte mère RACE

Carte wifi	SDO	SDI	SCK	SS	RESET	ENABLE	IRQ
RACE mb	19	21	23	16	18	26	16
PIC 32	EBIA8 PMA8 RF5	EBIA9 PMA9 RF4	RPD1 SCK1 RD1	EBIA1 PMA1 RK1	EBIA3 PMA3 RK2	EBICS3 RJ7	INT0 RD0

Tableau 4 : Table de routage du projet RACE

La mise en commun des deux projets séparés a posé un problème car la librairie wifi fournie par Harmony requiert des dépendances demandant l'utilisation d'un Timer dynamique. Or nous avons besoin de Timer statiques, deux choix s'offrent alors à nous :

- Mélanger les deux bibliothèques bien que ce soit extrêmement complexe car il y a beaucoup de fonctions ayant le même nom qui doivent être renommés.
- Laisser la bibliothèque dynamique pour le wifi et initialiser les autres Timers de manière "classique" en modifiant les registres TXCON et PRX de chaque Timer nécessaire.

J'ai donc choisi la seconde méthode pour sa facilité. De plus, au lieu de faire les calculs à la main, on peut exécuter le projet « Test contrôleur » en utilisant la bibliothèque statique et ensuite récupérer les valeurs des registres qui nous intéressent grâce au debugger.

BF80_141C	T3CKR	0x00000000	0
BF84_0400	T3CON	0x0000C060	49248
BF80_1420	T4CKR	0x00000000	0
BF84_0600	T4CON	0x0000C060	49248
BF80_1424	T5CKR	0x00000000	0
BF84_0800	T5CON	0x0000C030	49200

BF84_0420	PR3	0x00007A12	31250
BF84_0620	PR4	0x00007A12	31250
BF84_0820	PR5	0x00000271	625

Figure 29 : Tableau des registres

Il suffit donc de définir les registres suivants lors de l'initialisation du programme :

```
PR3=10000;
PR4=8000;
PR5=625;

T3CON=49264;
T4CON=49264;
T5CON=49200;
```

Figure 30 : Registres nécessaires pour initialiser et démarrer les Timers.

4.2. Sécurisation de la connexion wifi et traitement des trames

4.2.1. Introduction des problèmes concernant la connexion

Pour le code wifi, on reprend le code utilisé lors du prototypage. Cependant, après quelques tests, on se rend compte que certaines incertitudes de la connexion posent problèmes.

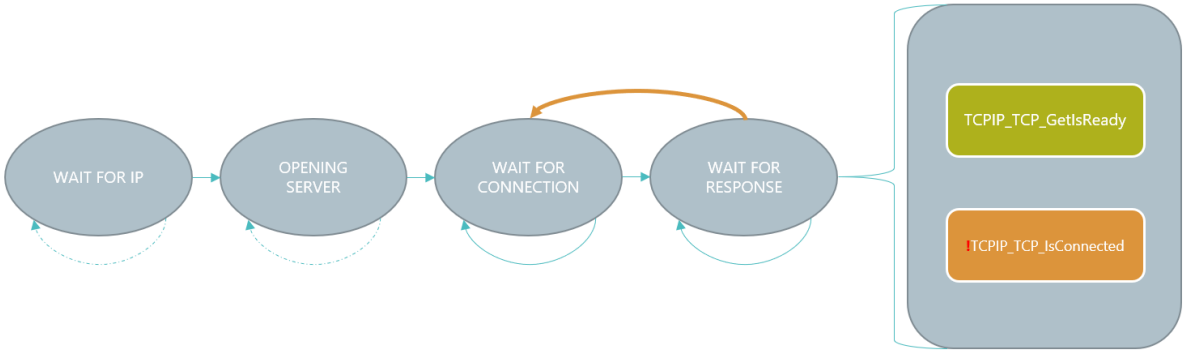

Figure 31 : Description schématique du fonctionnement de l'application wifi

En effet comme on l'a vu précédemment, le module est en Soft AP. Ainsi, les deux premières étapes sont presque facultatives. Car dans tous les cas, ce dernier s'attribue lui-même son IP. Il n'a pas besoin d'attendre un IP ou de s'assurer que le serveur puisse s'ouvrir car il est son propre routeur. Les problèmes surgissent lorsque le client se connecte et est en attente de réponse. En effet, les outils de déconnexion fournis par QT ne sont pas toujours fonctionnels comme expliqué sur un thread de forum [6]. La seule manière d'être sûr à 100%

que la déconnexion soit bien transmise est de faire une transmission de trame manuelle de type « GoodBye Client! » qui annonce la déconnexion. Aussi si on se déconnecte directement du wifi, l'information de déconnexion n'est pas transmise à l'application car gérée par le driver du WINC1500. Ces deux cas sont très problématiques car si la connexion est perdue lors d'un envoi de commande au moteur alors la voiture continuera d'avancer sans pouvoir s'arrêter. Il est donc utile d'appliquer des correctifs.

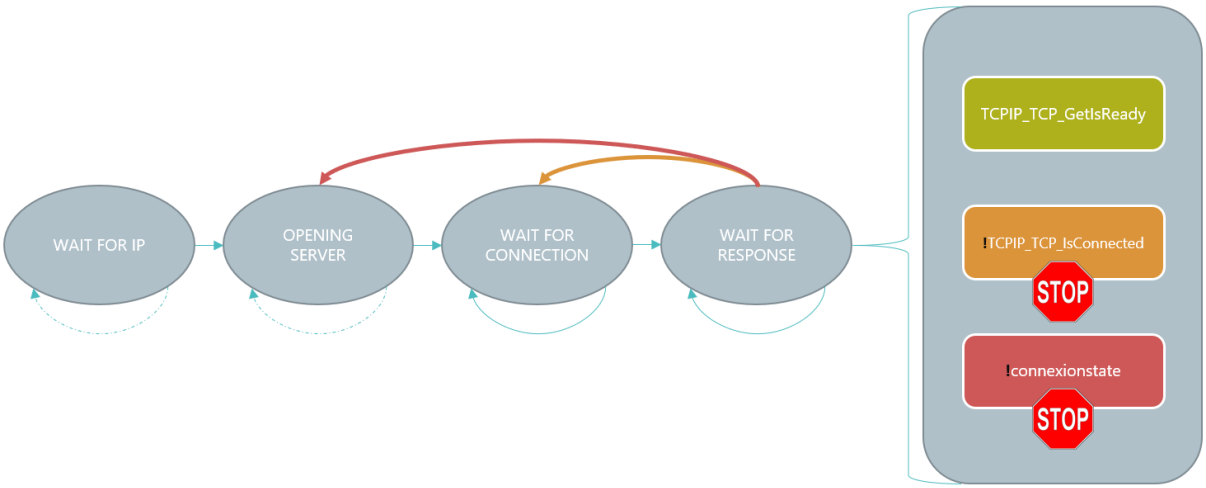

Figure 32 : Description schématique du fonctionnement de l'application wifi avec les corrections

On ajoute alors une commande d'arrêt moteur lorsqu'une déconnexion est détectée. Également, on ajoute un troisième état (En rouge) qui détecte lorsque le client se déconnecte du réseau wifi sans déconnexion au préalable et en appliquant les correctifs cités précédemment.

4.2.2. Application des correctifs

TCPIP_TCP_GetIsReady

L'application de détection de trame prend la forme suivante :

```
if (TCPIP_TCP_GetIsReady(wnetcomData.socket))
{
    TCPIP_TCP_ArrayGet(wnetcomData.socket, wnetcomMsgFromClient, sizeof(wnetcomMsgFromClient) - 1);
    if (strcmp(wnetcomMsgFromClient,wnetcomMsgFromClientGb)==0){
        SYS_CONSOLE_PRINT("TCP_TXRX[%d]: Received disconnection\r\n", wnetcomData.txrxTaskState);
        TCPIP_TCP_Disconnect(wnetcomData.socket);
        break;
    }
    decodestr(wnetcomMsgFromClient);

    sprintf(textToWrite,"%lu", wnetcomData.current);
    TCPIP_TCP_ArrayPut(wnetcomData.socket, textToWrite, sizeof(textToWrite));

    memset(&wnetcomMsgFromClient[0], 0, sizeof(wnetcomMsgFromClient));
}
```

Figure 33 : Code de l'état "Réception de trame sur le socket"

Après la réception du message on vérifie si ce n'est pas le message de déconnexion. Si c'est le cas on déconnecte le client, sinon on décode la trame grâce à la fonction *decodestr() (voir description en 3.3)*.	Conversion de la donnée de consommation de puissance en char* et envoi sur le socket vers QT.	Efface le message reçu ce qui permet de s'assurer que les informations ne se répètent à la prochaine itération.

Le correctif prend donc la forme d'une comparaison de chaînes de caractère entre le message reçu et le message suivant :

```
static char wnetcomMsgFromClientGb[] = "Goodbye Client!\n\r";
```

Figure 34 : Message de déconnexion de la part du client

Si cette condition est vérifiée, alors le PIC32 se déconnecte manuellement du socket renvoyant alors à la condition *!TCPIP_TCPIsConnected*.

!TCPIP_TCPIsConnected

Lorsque la stack TCP/IP perçoit une déconnexion, on peut utiliser la fonction préétablie par le code généré par Harmony afin de faire la réinitialisation.

```
if (!TCPIP_TCP_IsConnected(wnetcomData.socket))
{
    SYS_CONSOLE_PRINT("TCP_TXRX[%d]: Connection closed/lost\r\n", wnetcomData.txrxTaskState);
    SYS_CONSOLE_PRINT("TCP_TXRX[%d]: Waiting for Client Connection on port: %d\r\n",
            wnetcomData.txrxTaskState, wnetcomData.port);
    wnetcomData.txrxTaskState = WNETCOM_TCPIP_WAIT_FOR_CONNECTION;

    //DRV_OC_PulseWidthSet(wnetcomData.handleOC0, min0+(max0-min0)/2);
    DRV_OC_PulseWidthSet(wnetcomData.handleOC1, min1);

    break;
}
```

Figure 35 : Code de l'état "Déconnexion détectée"

Il suffit alors de réinitialiser la commande moteur PWM. Il n'est pas nécessaire de faire de même pour la commande de direction car un changement brusque de cette dernière pourrait être dangereux pour la voiture.

!connexionstate

Comme expliqué précédemment, une déconnexion brutale du wifi est plus compliquée à gérer car ce n'est pas la stack TCP/IP qui s'en occupe mais le driver wifi. Il faut donc modifier le fichier wdrv_winc1500_connmgr.c qui est le fichier c qui s'occupe de la gestion de clients.

D'abord il faut y inclure le fichier h de notre application et y ajouter une variable *connexionstate* que nous utiliserons pour la vérification de connexion.

```
#include "wnetcom.h"
extern WNETCOM_DATA wnetcomData;
```

Ensuite on modifie la fonction *ConnectEventCB()* comme suit :

```
void ConnectEventCB(bool connected, bool isServer, const uint8_t *const client)
{
    static uint8_t mac[6];

    if (gp_wdrv_cfg->networkType == WDRV_NETWORK_TYPE_INFRASTRUCTURE) {
        ConnectionStateUpdate(connected);
    } else if (gp_wdrv_cfg->networkType == WDRV_NETWORK_TYPE_SOFT_AP) {
        if (isServer) {
            ConnectionStateUpdate(connected);
        } else {
            if (connected) {
                ClientCacheUpdate(connected, client);
                WDRV_DBG_INFORM_PRINT("Client %02x:%02x:%02x:%02x:%02x:%02x is c
                memcpy(mac, client, sizeof(mac));
                wnetcomData.connexionstate=true;
            } else {
                ClientCacheUpdate(connected, mac);
                WDRV_DBG_INFORM_PRINT("Client %02x:%02x:%02x:%02x:%02x:%02x has
                wnetcomData.connexionstate=false;
            }
        }
    }
}
```

Figure 36 : Fonction ConnextEventCB(), les ajouts sont entourés en rouge.

Enfin dans le fichier c de notre application on ajoute une vérification de cette variable dans l'état d'attente de réponse. On coupe alors le moteur et on fait un renvoi vers l'état d'ouverture du serveur en cas de déconnexion brutale du wifi.

```
if (!wnetcomData.connexionstate)
{
    SYS_CONSOLE_PRINT("TCP_TXRX[%d]: Emergency disconnection\r\n", wnetcomData.txrxTaskState);
    TCPIP_TCP_Close(wnetcomData.socket);
    DRV_OC_PulseWidthSet(wnetcomData.handleOC1, 0);
    wnetcomData.txrxTaskState = WNETCOM_TCPIP_OPENING_SERVER;
    break;
}
```

Figure 37 : Code de l'état "Déconnexion urgente détectée"

4.3. Décodage des trames

```
void decodestr(char* str)
{
    const char s[2] = ":";
    char *token;
    token = strtok(str, s);
    SYS_CONSOLE_PRINT("moteur : ");
    SYS_CONSOLE_PRINT(token);
    wnetcomData.moteur = atoi(token);
    token = strtok(NULL, s);
    SYS_CONSOLE_PRINT(" direction : ");
    SYS_CONSOLE_PRINT(token);
    wnetcomData.direction = atoi(token);
    SYS_CONSOLE_PRINT("\r\n");
    ctrlrace();
}
```

Figure 38 : Code d'envoi d'informations sur l'UART.

La fonction decodestr() ne sert qu'à extraire les données utiles des trames envoyées par l'application. De plus, elle affiche le résultat sur la console UART. Une deuxième fonction sert ensuite à contrôler les moteurs.

```
void ctrlrace(void) {
    DRV_OC_PulseWidthSet(wnetcomData.handleOC0, min0+(wnetcomData.direction+45)*(max0-min0)/90);
    uint32_t mtr = abs(wnetcomData.moteur);
    if (wnetcomData.moteur < 0){
        PLIB_PORTS_PinClear (PORTS_ID_0, PORT_CHANNEL_K, PORTS_BIT_POS_6);
        PLIB_PORTS_PinSet (PORTS_ID_0, PORT_CHANNEL_J, PORTS_BIT_POS_3);
        DRV_OC_PulseWidthSet(wnetcomData.handleOC1, min1+mtr*(max1-min1)/20);

    }
    else {
        PLIB_PORTS_PinSet (PORTS_ID_0, PORT_CHANNEL_K, PORTS_BIT_POS_6);
        PLIB_PORTS_PinClear (PORTS_ID_0, PORT_CHANNEL_J, PORTS_BIT_POS_3);
        DRV_OC_PulseWidthSet(wnetcomData.handleOC1, min1+mtr*(max1-min1)/50);
    }

}
```

Figure 39 : Code de contrôle des moteurs et de la direction

Change le PWM de la direction, un simple calcul permet de convertir la donnée émise par QT en donnée utile.	Pour le contrôle des moteurs, on a deux cas de figure. La marche avant, qui requiert que la pin INA soit mis en état haut, et la marche arrière, qui requiert que la pin INA soit mis en état bas.

Il faut faire attention lors de la conversion signée vers non signée de bien utiliser la fonction abs() et de ne surtout pas se contenter d'un cast. Car sinon lorsque le nombre atteint les négatifs, -1 se convertie en 4294967296.

```
uint32_t mtr = (uint32_t)(wnetcomData.moteur);
```

Figure 40 : Exemple de mauvaise utilisation d'un cast pour conversion en valeur absolue.

4.4. Bouton de déconnexion et bouton d'arrêt du programme

Lors de la fermeture du programme, il faut s'assurer que la trame « GoodBye Client ! » soit envoyée. Ce n'est pas évident à cause de certains problèmes liés à la façon dont QT gère les envois de trames.

Figure 41 : Représentation de la croix de fermeture sous Windows

Afin de lancer une application après l'appui de la croix de fermeture, il faut faire une surcharge de la fonction *closeEvent()*.

```
void Fenetre::closeEvent(QCloseEvent *event)
{
    fenetrequit1();
}
```

Figure 42 : Surcharge de la fonction closeEvent()

La fonction qui suit va donc dans un premier temps déconnecter le Timer lié à l'envoi de trames. Ceci dans le but d'éviter d'envoyer deux trames en même temps. Elle attend 50 ms avant de passer à l'envoi du message de déconnexion et attend une nouvelle fois 50 ms le temps que l'envoi se fasse avant de quitter le programme.

```
void
Fenetre::fenetrequit1()
{
    disconnect(sendTimer,SIGNAL(timeout()),this,SLOT(send()));
    disconnect(soc, SIGNAL(disconnected()), this, SLOT(errconnexion()));
    QTimer *timers = new QTimer(this);
    timers->setSingleShot(true);
    connect(timers, SIGNAL(timeout()), this, SLOT(fenetrequit2()));
    timers -> start(50);
}

void
Fenetre::fenetrequit2()
{
    char message[] = "Goodbye Client!\n\r";
    soc -> write(message);
    QTimer *timers = new QTimer(this);
    timers->setSingleShot(true);
    connect(timers, SIGNAL(timeout()), qApp, SLOT(quit()));
    timers -> start(50);
}
```

Figure 43 : Fonctions nécessaires pour l'arrêt de l'interface.

J'ai aussi décidé de lier le bouton de clôture inclus sur l'interface à la déconnexion manuelle du module wifi. Ainsi il permet la déconnexion et la reconnexion à tout moment.

Figure 44 : Représentation du bouton de déconnexion sur l'interface.

```
QObject::connect(close_bouton,SIGNAL(clicked()),this,SLOT(deconnexion1()));
```

Figure 45 : Connexion entre le click du bouton et le lancement de la fonction deconnexion1()

```
void
Fenetre::deconnexion1()
{
    disconnect(sendTimer,SIGNAL(timeout()),this,SLOT(send()));
    disconnect(soc, SIGNAL(disconnected()), this, SLOT(errconnexion()));
    QTimer *timers = new QTimer(this);
    timers->setSingleShot(true);
    connect(timers, SIGNAL(timeout()), this, SLOT(deconnexion2()));
    timers -> start(50);
}

void
Fenetre::deconnexion2()
{
    char message[] = "Goodbye Client!\n\r";
    soc -> write(message);
    QTimer *timers = new QTimer(this);
    timers->setSingleShot(true);
    connect(timers, SIGNAL(timeout()), this, SLOT(errconnexion()));
    timers -> start(50);
}
```

Figure 46 : Fonctions nécessaires pour la déconnexion.

Le principe est le même que ci-dessus. Cependant ici, la série de fonctions renvoie au menu d'accueil au lieu d'arrêter le programme.

4.5. Mise en place de la manette

Dans Qt il existe une bibliothèque que l'on peut inclure directement au code c++. Cependant cette bibliothèque ne fonctionne pas très bien car elle ne reconnaît pas tous les types de manettes. Il existe un exemple donné par QtCreator, nommé quickGamepad, qui marche très bien mais qui est codé sous QML et non en C++.

Afin de charger le programme QML dans le main.cpp, on doit procéder de la manière suivante :

```
/* Application Quick Gamepad QML */
QQmlApplicationEngine engine;
engine.load(QUrl(QStringLiteral("qrc:///qml/main.qml")));
```

Figure 47 : Appel de la fonction QML

Dans le main.qml on doit ensuite déclarer les signaux que l'on utilisera pour communiquer vers le main.cpp. Il faut aussi rendre la fenêtre invisible en mettant l'argument visible en false.

```
Window {
    id: applicationWindow1
    visible: false
    width: 800
    height: 600
    title: qsTr("QtGamepad Example")
    color: "#363330"
    signal aclicked
    signal bclicked
    signal leftclicked
    signal rightclicked
```

Figure 48 : Déclaration des signaux dans main.qml

Afin que ces signaux soient envoyés au bon moment, il faut ajouter des arguments aux objets qui nous intéressent. Ici ce sont les boutons A et B et les croix directionnelles gauche et droite.

```
ButtonImage {
    id: buttonA
    anchors.bottom: parent.bottom
    anchors.horizontalCenter: parent.horizontalCenter
    source: "xboxControllerButtonA.png";
    active: gamepad.buttonA
    onActiveChanged: {
        //console.log("button a clicked")
        applicationWindow1.aclicked()
    }
}
ButtonImage {
    id: buttonB
    anchors.right: parent.right
    anchors.verticalCenter: parent.verticalCenter
    source: "xboxControllerButtonB.png";
    active: gamepad.buttonB
    onActiveChanged: {
        //console.log("button b clicked")
        applicationWindow1.bclicked()
    }
}
```

Figure 49 : Envoi des signaux aclicked et bclicked lors de leurs activation

```
Rectangle {
    id: leftArea
    visible: gamepad.buttonLeft
    color: "#3814abff"
    radius: 5
    width: parent.width * 0.3
    height: parent.height * 0.3
    anchors.left: parent.left
    anchors.leftMargin: parent.width * 0.05
    anchors.verticalCenter: parent.verticalCenter
    onVisibleChanged: {
        //console.log("left clicked")
        applicationWindow1.leftclicked()
    }
}

Rectangle {
    id: rightArea
    visible: gamepad.buttonRight
    color: "#3814abff"
    radius: 5
    width: parent.width * 0.3
    height: parent.height * 0.3
    anchors.right: parent.right
    anchors.rightMargin: parent.width * 0.05
    anchors.verticalCenter: parent.verticalCenter
    onVisibleChanged: {
        //console.log("right clicked")
        applicationWindow1.rightclicked()
    }
}
```

Figure 50 : Envoi des signaux leftclicked et rightclicked lorsque leurs rectangles respectifs sont visibles

Et dans le main.cpp il faut connecter ces signaux à des fonctions qui changent les bonnes variables pour faire en sorte que la manette contrôle le programme. Cependant afin de répertorier les signaux QML il faut utiliser une fonction spéciale dénommée rootObject().

```
QObject* item = (QObject*)engine.rootObjects()[0]; //Nécessaire pour l'envoi des signaux QML vers c++
QObject::connect(item, SIGNAL(aclicked()), &ma_fenetre, SLOT(aclicked()));
QObject::connect(item, SIGNAL(bclicked()), &ma_fenetre, SLOT(bclicked()));
QObject::connect(item, SIGNAL(leftclicked()), &ma_fenetre, SLOT(leftclicked()));
QObject::connect(item, SIGNAL(rightclicked()), &ma_fenetre, SLOT(rightclicked()));
```

Figure 51 : Dans main.cpp, connexion des différents signaux venant de main.qml vers fenetre.cpp

5. Conclusion

La voiture télécommandée fonctionne correctement en utilisant l'application.

Figure 52 : Photo de la voiture télécommandée

La connexion wifi est assez sécurisée pour une utilisation normale. Cependant un cas n'est pas pris en compte, c'est l'arrêt brutal de l'application QT. Dans ce cas, la trame n'est pas envoyée et le wifi n'est pas déconnecté. On perd donc le contrôle de la voiture. La solution serait de mettre une boucle d'interruption toutes les 500 ms vérifiant que l'application ait bien reçu des trames de la part de QT. Si aucune trame n'est reçue dans ce laps de temps, alors elle se déconnecte et arrête les moteurs.

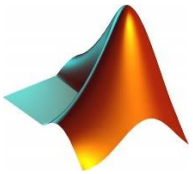

Figure 53 : Logo de Matlab

D'autre part, le manque de régulation des moteurs peut poser un problème. En effet, pour avoir un contrôle des moteurs optimal, il est nécessaire d'implémenter un régulateur en boucle fermée grâce au logiciel Matlab. Car ici lorsque l'opérateur souhaite freiner, la voiture ne fait que décélérer lentement, ceci afin d'éviter d'atteindre la limite de 30 Ampères du pont en H utilisé pour contrôler les moteurs [7]. Un véritable frein demande une bonne régulation dans le but d'éviter d'atteindre cette limite tout en permettant un freinage performant.

De plus il serait utile de compiler l'application pour un système d'exploitation mobile de type Android. Car emporter un ordinateur portable avec soi, pour contrôler la voiture, est encombrant. Quelques ajustements seront nécessaires dans le code afin de pouvoir contrôler le moteur et la direction avec un écran tactile au lieu d'un clavier ou d'une manette.

REFERENCES

[1] http://ww1.microchip.com/downloads/en/DeviceDoc/70005230B.pdf (PIC32)

[2] http://ww1.microchip.com/downloads/en/DeviceDoc/50002199A.pdf (Adaptateur)

[3] http://ww1.microchip.com/downloads/en/DeviceDoc/51950B.pdf (Carte d'expansion)

[4] http://ww1.microchip.com/downloads/en/DeviceDoc/70005309A.pdf (WINC1500)

[5] https://microchipdeveloper.com/wifi:wh

[6] https://forum.qt.io/topic/81000/qtcpsocket-state-always-connected

[7] https://www.st.com/en/automotive-analog-and-power/vnh5019a-e.html

Résumé

RACE est un projet qui consiste en l'élaboration du cerveau d'une voiture télécommandée. En utilisant des outils fournit par MPLAB, j'ai développé une application qui permet de commander la voiture en utilisant une application sur ordinateur connecté en wifi. Une attention particulière est apportée à la sécurisation de la connexion afin d'éviter que l'on perde le contrôle de la voiture. Cependant, la solution n'est pas parfaite car il manque une régulation de la commande moteur par rapport à l'intensité du courant. Ce qui empêche la voiture de pouvoir correctement freiner.

Mots Clés : Outils, wifi, sécurisation, régulation

Summary

RACE is a project that consists in developing the brain of a remote-controlled car. Using tools provided by MPLAB and QT, I developed an application that allows to control the car using a WIFI connected computer application. Special attention is given to securing the connection in order to avoid losing control of the car. However, the solution is not perfect because the engine control lacks a regulation of the motor control in relation to the intensity of the current which prevents the car from being able to brake properly.

Keywords : Tools, wifi, securing, regulation

ENSI CAEN

ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Ecole Publique d'Ingénieurs en 3 ans

6 boulevard Maréchal Juin, CS 45053
14050 CAEN cedex 04

ministère éducation nationale enseignement supérieur recherche

Normandie Université

Cti Commission des titres d'ingénieur

cdefi Conférence des Directeurs des Écoles Françaises d'Ingénieurs

European Accreditation of Engineering Programmes EUR-ACE®

CONFÉRENCE DES GRANDES ÉCOLES