université PARIS-SACLAY
IUT DE CACHAN
Département GEii 2

Groupe D1

Rapport de projet d'IT2R

Véhicule Intelligent et Communicant

2022-2023

Remerciements

Avant d'entamer la genèse de ce rapport, nous tenons à remercier particulièrement M. Mininger et M. Martincic, les professeurs qui ont encadré notre groupe, ont su nous laisser la liberté de chercher des solutions aux problèmes rencontrés, tout en nous aidant lorsqu'un binôme se retrouvait dans une impasse.

Les professeurs du groupe 2 ont aussi joué un rôle important dans la réalisation de projet, en apportant de bons conseils lorsque nous les sollicitions, ainsi, nous saluons l'apport de M. Clamens et de Mme. B

Il est aussi important de souligner le rôle de M. Ardiller, technicien à l'IUT de Cachan, dans ce projet, qui nous a aidé à de nombreuses reprises pour résoudre des problèmes mécaniques sur la voiture ou de soudure sur les cartes électroniques.

Enfin, nous nous devons de remercier l'IUT, qui nous a mis à disposition le matériel professionnel dont il dispose, qui nous permet de travailler dans les meilleures conditions possibles.

Table des matières

Introduction

Ce travail a été réalisé dans le cadre du projet de quatrième semestre en spécialité Informatique Embarquée par des équipes composées d'une dizaine de membres. L'objectif était d'ajouter des fonctionnalités embarquées à un modèle miniature de châssis de voiture, pour la rendre "intelligente et communicante". Pour illustrer notre propos, on peut prendre l'exemple de l'allumage automatique des feux, ou encore la détection des obstacles, qui faisaient partie de notre cahier des charges. Au total, nous avions pour objectif d'intégrer neuf fonctionnalités à notre voiture, pour se faire, nous avons employé la méthode agile "Scrum". Celle-ci consiste à diviser le travail en 3 laps de temps (ici cinq séances de quatre heures), appelés des "sprint", en organisant une réunion entre chaque sprint pour débattre de l'avancement du projet et des rapports de force à établir. De plus, chaque fonction appartenant au cahier des charges est réalisée par des petits groupes de deux ou trois personnes (quatre lorsque la difficulté technique ou le retard l'imposait).

Ce projet a fait appel à toutes les compétences que nous avons assimilées au cours de ces deux premières années de BUT, avec de la gestion de capteurs, de la transmission d'informations (liaisons séries, Bluetooth…) et la mise en œuvre d'actionneurs (tel que le moteur de la voiture).

Le versioning a aussi représenté une composante importante du projet. En outre, cela nous a permis de rendre notre pratique plus professionnelle puisque l'avancée de chaque groupe était consultable par les autres membres de l'équipe ainsi que par le professeur encadrant. Ainsi, toute personne impliquée pouvait avoir une vision globale de l'avancée du projet. Cependant, la plus grande force du versioning est de pouvoir travailler simultanément sur un même objectif de programmation tout en évitant d'éventuels conflits sur les versions. Pour se faire, nous avons utilisé le site GitHub.

I. Gestion du son

Par Kamil et Sergio

1. Présentation du module

Pour cette partie de gestion sonore de notre véhicule intelligent, nous nous baserons sur un module en particulier « le DFPlayer mini ».

Le DFPlayer est un module audio compact conçu pour être facilement intégré dans des systèmes électroniques. Ce dernier est souvent utilisé pour la lecture de fichiers audio tels que des sons d'alarme et dans notre cas pour la lecture de l'ensemble des sons relatifs au véhicule (klaxon, clignotants, moteur...). Le module prend en charge les formats de fichiers MP3 et WAV pour notre projet nous utiliserons des fichiers mp3 et dispose d'une interface série pour la communication avec un microcontrôleur (UART). Il est également doté de plusieurs fonctionnalités telles que la lecture en boucle, la sélection de fichiers et le contrôle du volume.

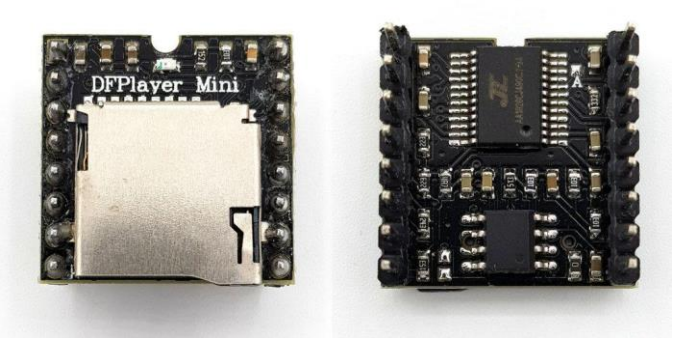


Figure 1 : Module DFPlayer

L'utilisation d'un DFPlayer Mini dans un projet électronique tel que notre voiture intelligente présente plusieurs avantages. Dans un premier temps, le DFPlayer Mini est très compact et léger, ce qui le rend idéal pour les projets portables ou nécessitant une intégration dans un espace restreint. Sa faible consommation d'énergie le rend également adapté aux projets alimentés par batterie.

Ce module est conçu pour être facile à utiliser et à intégrer dans un projet électronique. En effet, au niveau de sa programmation, cette dernière peut être réalisée sans trop de difficultés par l’envoi de trames bien précises afin de réaliser l'ensemble des commandes souhaitées. Ce qui nous permet de l’intégrer facilement au projet final et ainsi rendre son fonctionnement auto-exécutable en fonction des conditions dans lesquelles se trouvent le véhicule.

2. Programmation

Sur ce projet nous souhaitons fonctionner en temps réel ce qui va nous permettre de garantir que les tâches critiques soient exécutées dans des délais prédéterminés et avec une précision élevée. La première étape est donc de créer notre tâche avec l'ensembles des initialisations relatives au projet :

```
1   /*----------------------------------------------------------------------------
2    * CMSIS-RTOS 'main' function template
3    *---------------------------------------------------------------------------*/
4
5   #define osObjectsPublic                     // define objects in main module
6   #include "osObjects.h"                      // RTOS object definitions
7   #include "Driver_USART.h"               // ::CMSIS Driver:USART
8
9   extern ARM_DRIVER_USART Driver_USART1;
10
11  void Tache1 (void const*argument);
12  osThreadId  ID_Tache1;//Définition idebtifiant tâche
13  osThreadDef(Tache1,osPriorityNormal,1,0);//Définition caractéristique tâche
24  int main (void) {
25    osKernelInitialize ();                    // initialize CMSIS-RTOS
27
28          ID_Tache1=osThreadCreate(osThread(Tache1),NULL);
        osKernelStart ();                         // start thread execution
              osDelay(osWaitForever);
              return 0;
      }
```

Comme dit précédemment ce module fonctionne en UART, et l'envoi d'instruction ce fait à travers l'envoi de trames de dix octets du µc vers le module grâce à la liaisons série. Il faut alors réaliser une fonction permettant l'utilisation de l'UART et son initialisation au sein de notre code :

```
81  void Init_UART(void){
82          Driver_USART1.Initialize(NULL);
83          Driver_USART1.PowerControl(ARM_POWER_FULL);
84          Driver_USART1.Control(  ARM_USART_MODE_ASYNCHRONOUS |
85                                                  ARM_USART_DATA_BITS_8          |
86                                                  ARM_USART_STOP_BITS_1          |
87                                                  ARM_USART_PARITY_NONE          |
88                                                  ARM_USART_FLOW_CONTROL_NONE,
89                                                  9600);
90          Driver_USART1.Control(ARM_USART_CONTROL_TX,1);
91          Driver_USART1.Control(ARM_USART_CONTROL_RX,1);
92  }
```

Figure 2 : Fonction UART

```
5   #define osObjectsPublic                     // define objects in main module
6   #include "osObjects.h"                      // RTOS object definitions
7   #include "Driver_USART.h"                 // ::CMSIS Driver:USART
8
9   extern ARM_DRIVER_USART Driver_USART1;
10
11  void Tache1 (void const*argument);
12  osThreadId  ID_Tache1;//Définition idebtifiant tâche
13  osThreadDef(Tache1,osPriorityNormal,1,0);//Définition caractéristique tâche
17
18  void Send_DFP(char commande, char P1, char P2);
19  void Init_UART(void);
20
21  /*
22   * main: initialize and start the system
23   */
24  int main (void) {
25    osKernelInitialize ();                    // initialize CMSIS-RTOS
26          Init_UART();
27
28          ID_Tache1=osThreadCreate(osThread(Tache1),NULL);
```

Chaque octet envoyer via la liaison série a un rôle bien précis, en effet dans la datasheet, nous sont présenté l'utilité de chaque octet et les valeurs à leur attribuer en fonction de nos besoins tel que :

Format: $S VER Len CMD Feedback para1 para2 checksum $O		
$S	Start byte 0x7E	Each command feedback begin with $, that is 0x7E
VER	Version	Version Information
Len	the number of bytes after "Len"	Checksums are not counted
CMD	Commands	Indicate the specific operations, such as play / pause, etc.
Feedback	Command feedback	If need for feedback, 1: feedback, 0: no feedback
para1	Parameter 1	Query high data byte
para2	Parameter 2	Query low data byte
checksum	Checksum	Accumulation and verification [not include start bit $]
$O	End bit	End bit 0xEF

On comprend alors que 5 octets seront fixés à une seule valeur peu importe l'envoi effectuer ($S (0x7E), VER (0xFF), Len (0x06), Feedback (0x00), $O (0xEF))

Format: $S VER Len CMD Feedback para1 para2 checksum $O

Les 5 autres paramètres eux peuvent changer en fonction des différentes commandes que l'on souhaite réaliser.

Format: $S VER Len CMD Feedback para1 para2 checksum $O

« CMD » permet en la sélection de la commande souhaitée :

CMD	Function Description	Parameters(16 bit)
0x01	Next	
0x02	Previous	
0x03	Specify tracking(NUM)	0-2999
0x04	Increase volume	
0x05	Decrease volume	
0x06	Specify volume	0-30
0x07	Specify EQ(0/1/2/3/4/5)	Normal/Pop/Rock/Jazz/Classic/Base
0x08	Specify playback mode (0/1/2/3)	Repeat/folder repeat/single repeat/ random
0x09	Specify playback source(0/1/2/3/4)	U/TF/AUX/SLEEP/FLASH
0x0A	Enter into standby – low power loss	
0x0B	Normal working	
0x0C	Reset module	
0x0D	Playback	
0x0E	Pause	
0x0F	Specify folder to playback	1~10(need to set by user)
0x10	Volume adjust set	{DH＝1:Open volume adjust }{DL: set volume gain 0~31}
0x11	Repeat play	{1:start repeat play}{0:stop play}
0x41	Reply	

Dans notre cas seuls 3 commandes nous intéressent :

- **0x09** nous permettant d'indiquer sur quel type de périphérique se trouvent nos audio, dans notre cas il s'agit d'une TF card il faudra alors fixer la valeur de param1 a 0x00 et param2 a 0x01.
- **0x03** nous permettant sélectionner la track que nous souhaitons jouer, il faut alors spécifier la piste en fonction de son numéro grâce aux octet param1 et param2.
- **0x41** permettent de replay la piste (param1 et param2 à 0).
- **0x01** permettant de passer à la piste suivante.

Afin d'avoir un code plus épuré et facile à manipuler nous avons mis en place une fonction que nous avons nommée « send_DFP » » qui nous permet d'envoyer la trame souhaitée en ne renseignant que les octets relatifs à la commande et aux paramètres. Cette fonction va nous permettre de calculer automatiquement le checksum en fonction des divers octets présents dans la trame, pour ensuite envoyer l'ensemble de la trame au module.

```
60  void Send_DFP(char commande, char P1, char P2)
61  {
62          short CS;
63          short CS1, CS2;
64          char tab[10] = {0x7E, 0xFF, 0x06, 0, 0x00, 0, 0, 0, 0, 0xEF}; ;
65
66          CS = -0xFF-0x06-commande-P1-P2;
67          CS1 = (CS & 0xFF00) >> 8;
68          CS2 = CS & 0x00FF;
69
70          tab[3] = commande;
71          tab[5] = P1;
72          tab[6] = P2;
73          tab[7] = CS1;
74          tab[8] = CS2;
75
76          while(Driver_USART1.GetStatus().tx_busy == 1); // attente buffer TX vide
77          Driver_USART1.Send( tab,10); //0x7E ,0xFF, 0x06, 0x09, 0x00, 0x00, 0x02, 0x01, 0x10, 0xEF
78          //osDelay(50);
79  }
```

Figure 3 : fonction send_DFP

Dans la main nous intégrons la ligne permettant d'envoyer la première trame permettant le choix du périphérique utilisé (TF card) ainsi qu'un délai de 10 ms :

```
24      int main (void) {
25        osKernelInitialize ();                    // initialize CMSIS-RTOS
26              Init_UART();
27
28              ID_Tache1=osThreadCreate(osThread(Tache1),NULL);
35                      Send_DFP(0x09, 0x00, 0x01);
36                      osDelay(10);
41        osKernelStart ();                         // start thread execution
42              osDelay(osWaitForever);
43              return 0;
44      }
```

Figure 4 : Main

Dans la boucle while de la tâche 1, on intègre les trames permettant respectivement à lancer la piste 1, replay la piste et enfin passer à la piste suivante :

```
46  void Tache1(void const * argument)
47  {
48
49          while(1)
50          {
51                  Send_DFP(0x03, 0x00, 0x01); //Selection de la piste n°1
52                  Send_DFP(0x41, 0x00, 0x00); //replay de la piste
53                  osDelay(5000);
54
55                  Send_DFP(0x01, 0x00, 0x00); // passage a la piste suivante
56                  osDelay(5000);
57          }
58  }
```

Figure 5 : Tache 1

3. Branchement du DFPlayer

Le module est alimenté en 5v et relié au microcontrôleur que par la liaison série et partage une masse commune à ce dernier.

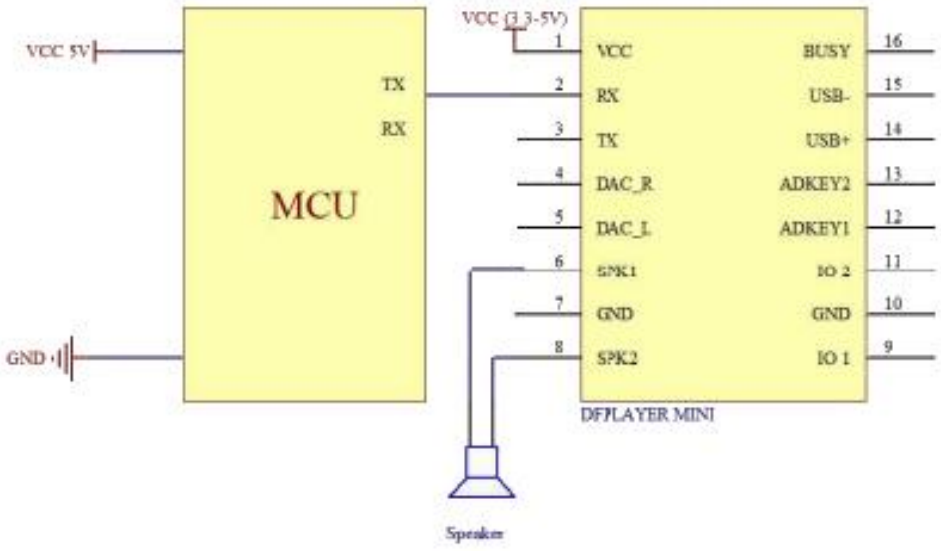


Figure 6 : Branchement DFPlayer

II. Les feux automatiques

Par Ziqian et Kelly

Pour le service des feux automatiques de la voiture intelligente et communicante, l'objectif était d'allumer les feux de la voiture en fonction de la luminosité.

1. Câblage

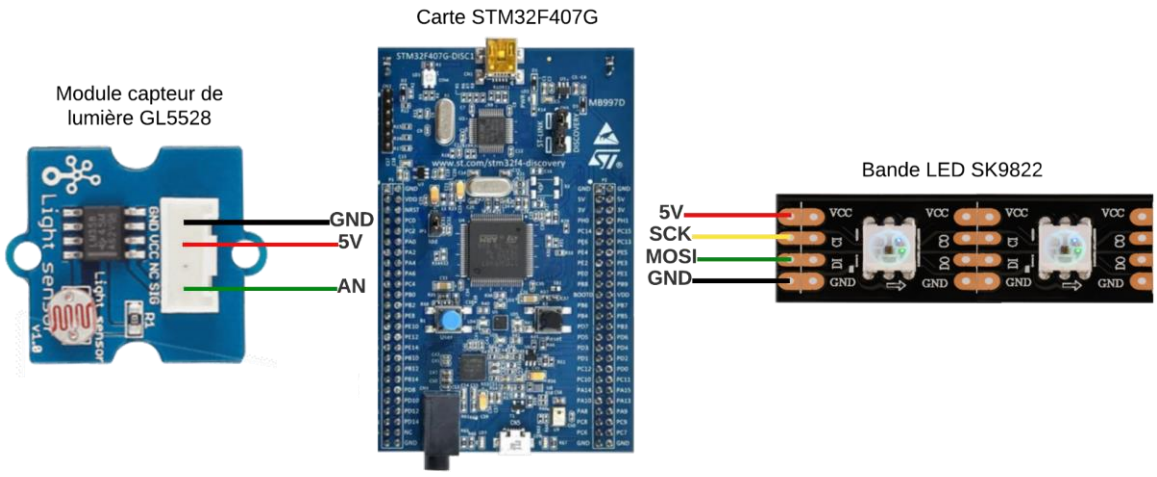


Figure 7 : Câblage des feux automatiques

Pour réaliser ce service, nous disposions d'un module capteur de lumière GL5528 qui renvoie une tension de 0 à 3.3V en fonction de la lumière reçue. Nous l'avions donc câblé à une entrée analogique de la carte STM32F4 qui possède un convertisseur analogique numérique.

Nous disposons également d'une bande de LED SK9822 pour les feux de la voiture. Celle-ci communique en liaison SPI (Serial Peripheral Interface), soit en liaison synchrone.
Nous avons donc câblé la broche SCK de la carte STM32F47 à la broche CI (Clock In) de la bande LED pour lui transmettre l'horloge.
Et nous avons également câblé la broche MOSI (Master Out Slave In) à la broche DI (Data In) pour lui transmettre les données.

2. Programmation

Quant à la programmation de ce service, vous trouverez ci-dessous un logigramme simplifié du programme que nous avons réalisé.

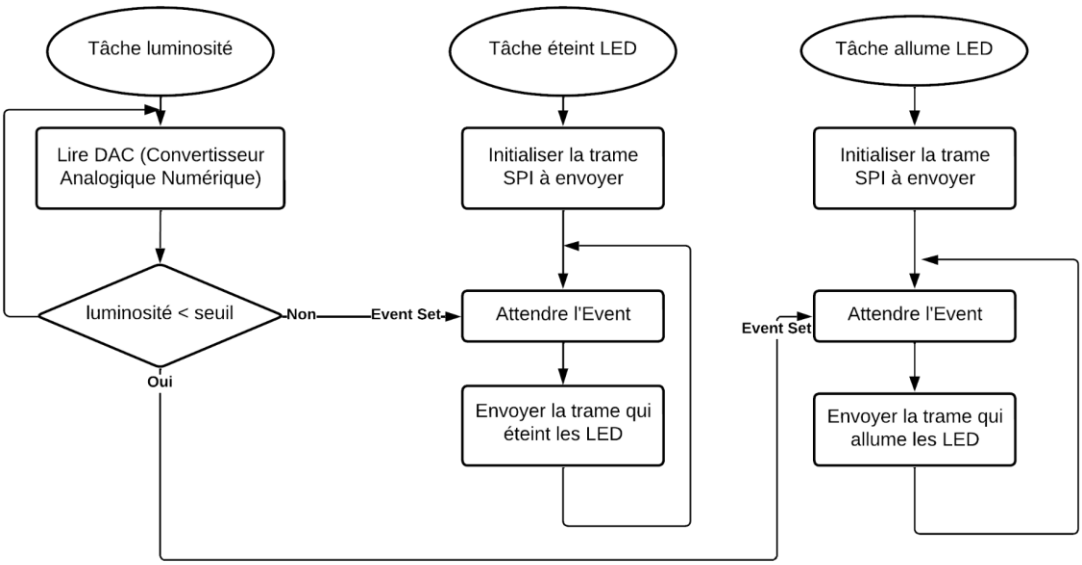


Figure 8 : Logigramme du programme des feux automatiques

Dans un premier temps, nous avions programmé une première tâche qui va dans une boucle, convertir l'entrée analogique sur laquelle est reliée le capteur de lumière en valeur numérique.
Cette valeur numérique représentant le niveau de luminosité va être comparée à un seuil que nous avions déterminé en faisant des essais.

En fonction de cela, la tâche va activer avec un Event une autre tâche : la tâche pour allumer les LED ou la tâche pour les éteindre.
Ces deux tâches attendent respectivement l'activation d'un Event qui bloque la tâche. Et une fois l'Event activé, ces tâches envoient une trame par liaison SPI pour contrôler les LED.

Ces trames SPI ayant la forme suivante :

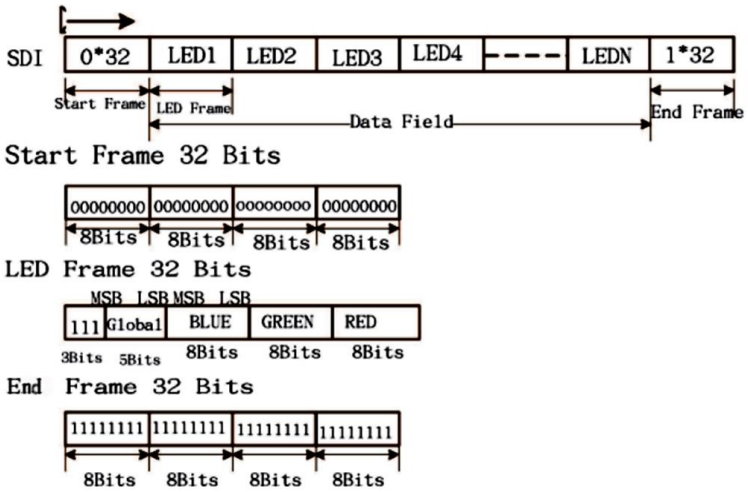


Figure 9 : Trame SPI pour contrôle la bande LED

Les trames SPI pour contrôler les LED commencent par quatre octets avec des bits à 0 pour signaler le début de la trame.
Elle est ensuite composée de quatre octets de données par LED :

- La première est composée de trois bits à 1 et de 5 bits qui déterminent la luminosité. Ainsi, pour allumer la LED, ce premier octet correspond à 0xFF, et l'éteindre, 0xE0.
- Les trois autres octets correspondent à la luminosité des LED bleu, vert et rouge.

La trame se termine avec quatre octets de bits à 1 pour signaler la fin.

III. La position GPS

Par Ziqian et Kelly

Pour le service de la position GPS de la voiture l'objectif était d'afficher sur un écran LCD la position GPS.

1. Câblage

Pour réaliser le service de la position, nous avons cette fois-ci programmer la carte MCB1768 auquel nous avons câblé le module GPS PmodGPS.
Ce module fonctionne en liaison série, c'est pourquoi nous l'avons câblé à l'interface UART de la carte en reliant le Tx du module au Rx de la carte MCB 1768 comme ci-dessous.

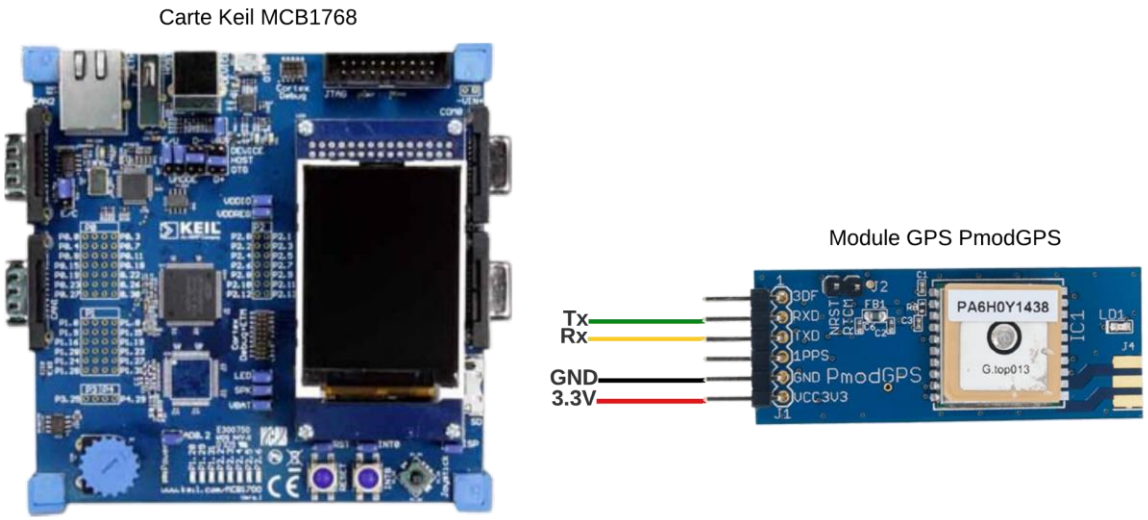


Figure 10 : Câblage du module GPS

2. Programmation

Le module GPS envoie des données en utilisant le protocole NMEA (National Marine Electronics Association). Pour récupérer les informations concernant la position GPS, soit la latitude et la longitude, nous devons extraire la trame commençant par $GPGGA dont les informations sont organisées comme ci-dessous :

Example	Description
$GPGGA	Message ID
064951.000	UTC Time (hhmmss.sss)
2307.1256	Latitude (ddmm.mmmm)
N	N/S indicator
12016.4438	Longitude (dddmm.mmmm)
E	E/W indicator
1	Position Fix Indicator
8	Satellites used
0.95	HDOP
39.9	MSL Altitude
M	Units
17.8	Geoidal Separation
M	Units
	Age of Diff. Corr.
*65	Checksum
<CR><LF>	End of message indicator

Table 2. GGA.

```
$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65<CR><LF>
```

Figure 11 : Trame $GPGGA à récupérer

Nous avons donc récupéré la trame reçu seulement si elle commençait par $GPGGA en suivant le logigramme suivant :

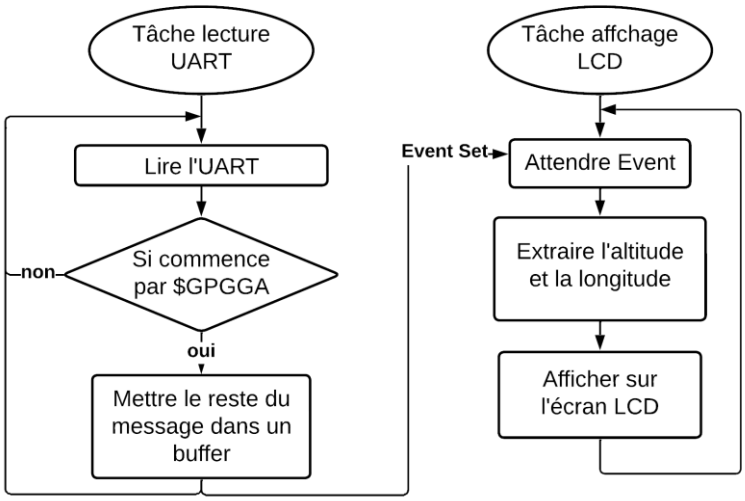


Figure 12 : Logigramme du programme de la position GPS

Après avoir récupéré la trame, nous avons extrait la latitude et la longitude et nous l'avons affiché sous forme un point sur une carte sur l'écran LCD.
La carte est constituée d'un rectangle représentant le bâtiment G comme sur l'image suivante avec en vert les mesures et les coordonnées à l'échelle de l'écran LD donc en pixels.

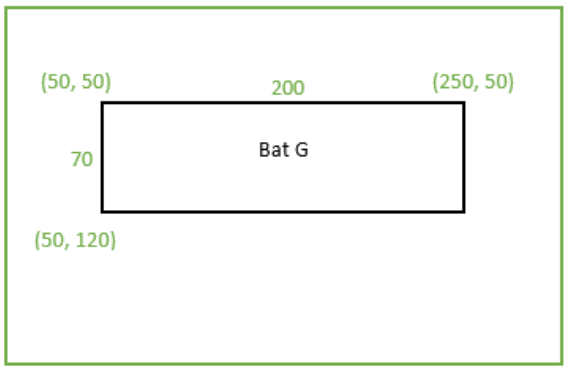
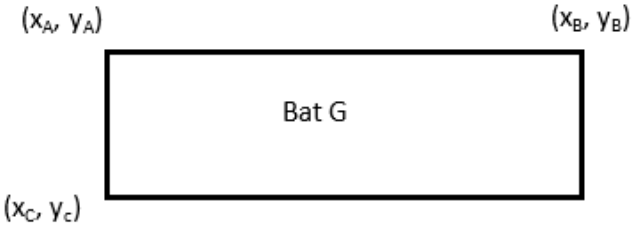


Figure 13 : Schéma de la carte réalisé sur l'écran LCD

Ainsi, pour mettre afficher la position GPS mesurée à l'échelle de la carte sur l'écran LCD, nous avions tout d'abord mesuré la position GPS des coins du bâtiment G comme ci-dessous.



Soit x_M et y_M, respectivement la latitude et la longitude d'une position détectée par le module GPS. Nous avons calculé cette position avec l'échelle de l'écran LCD avec les calculs suivants avec x et y respectivement l'abscisse et l'ordonnée de l'écran LCD :

$$x = 50 + (x_M - x_A) \times \frac{200}{x_B - x_A}$$
$$y = 50 + (y_M - y_A) \times \frac{70}{y_C - y_A}$$

IV. L'IHM de Supervision

Par Ziqian et Kelly

Pour le service de l'IHM de supervision de la voiture, l'objectif était d'afficher et contrôler des éléments de la voiture sur une interface graphique.

Pour réaliser ce service, nous avons utilisé la carte STM32F7 qui dispose d'un écran tactile permettant à l'utilisateur d'interagir avec les éléments de la voiture.

Figure 14 : Image de la carte STM32F7

Nous avons également récupéré l'interface graphique générée par la bibliothèque graphique emWin qui était à notre disposition :

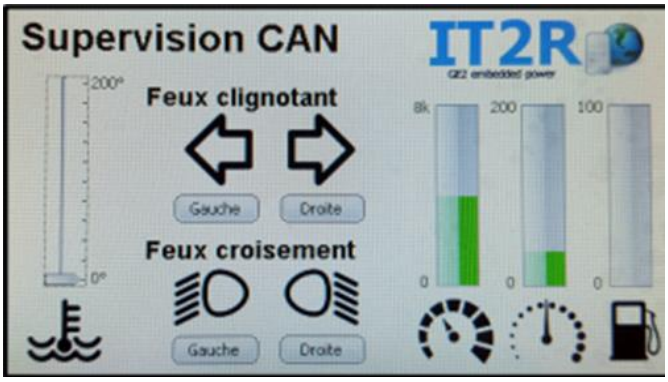


Figure 15 : Image de l'interface graphique

A partir de cette interface, nous avions donc la possibilité de contrôler la température du liquide de refroidissement avec un curseur, ainsi que les feux clignotants et de croisement avec des boutons. Nous pouvions également afficher le compteur de vitesse, le compte tour et la jauge d'essence.

Pour l'échange des données, contrairement à ce qui est affiché sur l'interface graphique, nous avions utilisé les interfaces UART de la carte en respectant un protocole que nous avons déterminé.

	Eléments	Trames
Réception : 1 octet	Compte-tour	01xx xxxx
	Vitesse	10xx xxxx
	Réservoir	11xx xxxx
Envoi : 2 octets 1er octet : température liquide 2e octet : clignotants et feux	Clignotant gauche	0xF1
	Clignotant droit	0xF2
	Feu de croisement gauche	0xF4
	Feu de croisement droit	0xF8

Figure 16 : Tableau du protocole de supervision

Pour programmer cela, nous disposions d'une fonction de callback généré par la bibliothèque et qui met à jour l'écran. Cette fonction est appelée lorsque qu'un contact est fait avec l'écran tactile. Mais elle peut également être appelée par le programme.

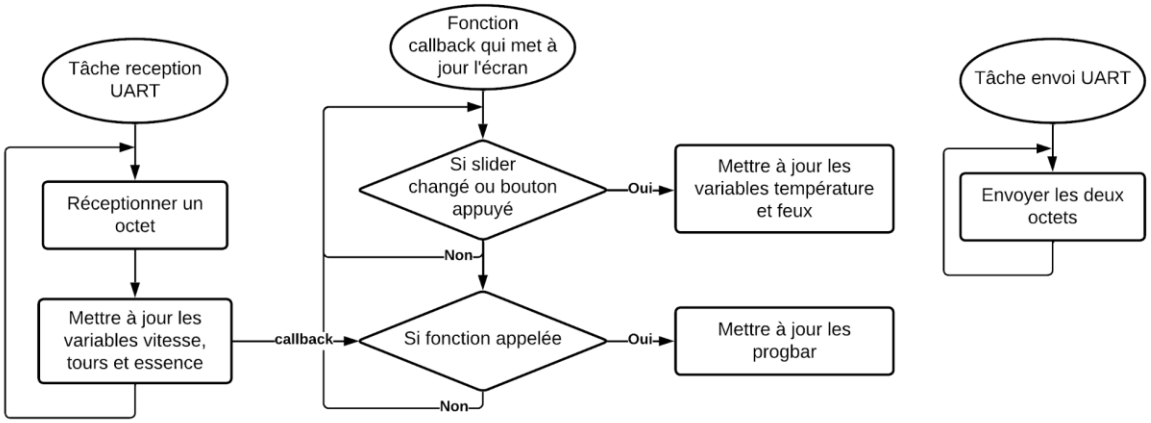


Figure 17 : Logigramme simplifié du programme de supervision

Nous allons donc avec une tâche de transmission par UART qui en permanence envoyer la valeur du liquide de refroidissement qui est mis à jour lorsque la valeur du curseur change, et la variable feux qui indique quel bouton des feux est appuyé.

Nous avons également programmé une tâche de réception par UART qui, avec un switch case, va mettre à jour les variables du compteur de vitesse, du compte tour et de la jauge d'essence selon les deux bits de poids forts en respectant le protocole établi.
Cette tâche appelle ensuite la fonction de callback qui va mettre à jour l'interface.

V. Tag RFID

Par Julie et Louca

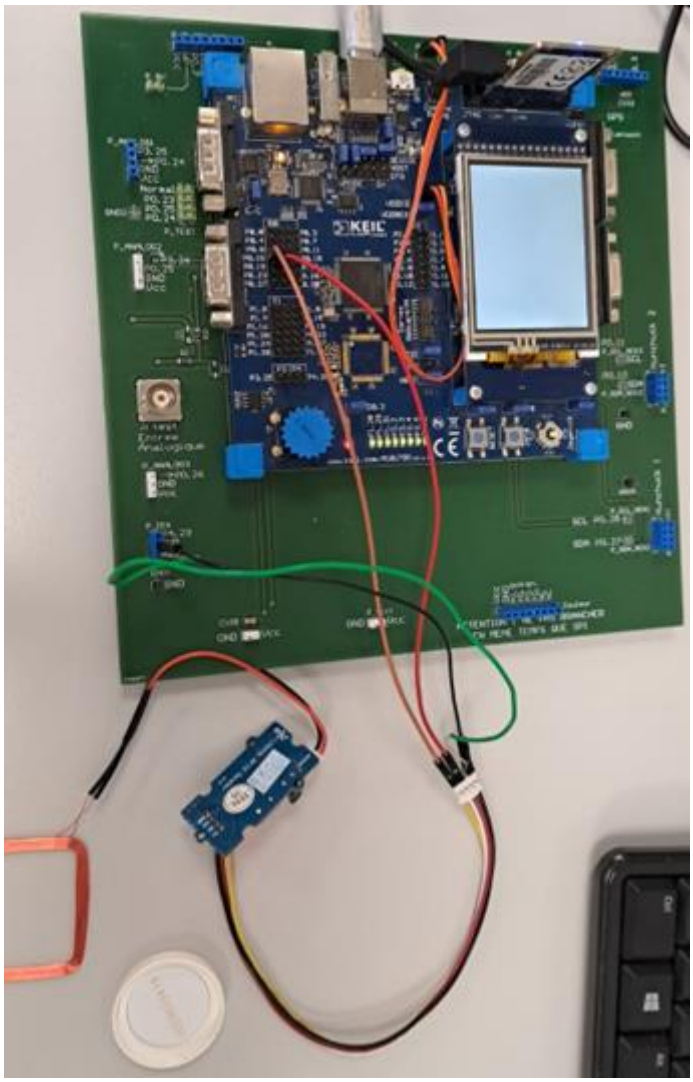
L'objectif de cette fonctionnalité était de pouvoir ouvrir les portes de notre voiture à l'aide d'un tag RFID unique. Nous avons couplé cette fonction avec la gestion du son, puisque lorsque le tag était le bon, un bruit d'ouverture était déclenché, alors qu'au moment de présenter un tag autre, une alarme s'activait.

1. Récupération de l'identifiant

La première étape fut de récupérer l'identifiant de notre tag RFID, pour ensuite pouvoir programmer la carte afin de faire en sorte de n'accepter que cette valeur. Pour cela, nous avons utilisé une carte Seeeduino fournie par l'enseignant encadrant. Celle-ci, branchée au PC (en USB) ainsi qu'au lecteur RFID, nous a permis de récupérer l'identifiant de notre tag dans l'hyperterminal. Un identifiant fait 12 caractères, il possède 1 bit de START, 1 bit de STOP, 1 checksum et le reste en données.

Figure 18: Carte Seeeduino utilisée pour récupérer l'identifiant

2. Câblage



125 kHz RFID Reader vers Keil LPC 1768 :

TX : P015
RX : P016
VCC : VCC
GND: GND

3. Programmation

Pour cette fonctionnalité, la programmation était très basique, nous avons cependant dû la réaliser en RTOS pour pouvoir la coupler avec d'autres actions. Le noyau le plus important du code réside dans la condition "if", qui teste toutes les valeurs d'un tableau de valeurs récupérées en liaison UART correspondant à l'identifiant du tag testé.

```
void tache1(void const *argument) {
        uint8_t tab_W[64];
        uint8_t tab_R[15];
        int i;
        while(1)
        {
//      while(Driver_USART1.GetStatus().tx_busy == 1); // attente buffer TX vide
//      Driver_USART1.Send(tab_W,16);

          Driver_USART1.Receive(tab_R,14);
//              while (Driver_USART1.GetRxCount() <1); // on attend que 1 case soit pleine
                if((tab_R[0] == 0x02) && (tab_R[13] == 0x03))
                {
                        if((tab_R[1]==0x30) && (tab_R[2]==0x44) && (tab_R[3]==0x30) && (tab_R[4]==0x30) && (tab_R[5]==0x39) && (tab_R[6]==0x33) && (tab_R[7]==0x36) &&
                        {
                                GLCD_DrawString(50,100,"                ");
                                GLCD_DrawString(50,100,"Acces autorise");
//                              osDelay(2000);
                                for(i=0; i<15; i++)
                                {
                                        tab_R[i] = 0;
                                }
                        }

                        else
                        {
                                GLCD_DrawString(50,100,"                ");
                                GLCD_DrawString(50,100,"Acces refuse");
//                              osDelay(2000);
                                for(i=0; i<15; i++)
                                {
                                        tab_R[i] = 0;
                                }

                        }
                }

        }
}
```

Figure 19 : Code du tag RFID

VI. Gestion des déplacements

Par Kryspian et Charles

Pour pouvoir se déplacer, la voiture est équipée d'un moteur à courant continu ainsi que d'un servomoteur. Le moteur à courant continu permet de faire avancer et reculer la voiture quant à lui le servomoteur permet de tourner à gauche et à droite.

Dans un premier temps, nous avons découvert dans la documentation technique que la fréquence des deux moteurs est différente. En effet, le moteur à courant continu fonctionne en 20 kHz alors que le servomoteur fonctionne en 50 Hz.
Cette différence de fréquence nous pose un problème puisque la carte dont nous disposons peut fournir différents signaux PWM ayant la même fréquence. Pour résoudre ce problème, nous avons décidé de fournir le signal PWM à 20 kHz et à l'aide d'une fonction d'interruption, le signal à 50 Hz est créé.

```
220     void Init_MOTEUR1(void){
221
222             LPC_SC->PCONP = LPC_SC->PCONP | 0x00000040;                              // Ena
223
224             LPC_PWM1->MCR = LPC_PWM1->MCR | 0x00000002;
225
226             LPC_PWM1->PR = 0;
227             LPC_PWM1->MR0 = 1249;
228
229             LPC_PINCON->PINSEL7 = LPC_PINCON->PINSEL7| 3<<18;       // Validation des sort
230
231             LPC_PWM1->LER = LPC_PWM1->LER | 0x0000000F;
232
233             LPC_PWM1->PCR = LPC_PWM1->PCR | 1<<10;
234
235             LPC_PWM1->MR2 =1200;
236
237             LPC_PWM1->TCR = 1;
238
239             LPC_GPIO0->FIODIR = LPC_GPIO0->FIODIR | (1<<16)| (1<<17)| (1<<18)| (1<<19);
240
241             LPC_GPIO0->FIOPIN = LPC_GPIO0->FIOPIN | (0<<16)| (0<<17)| (1<<18)| (1<<19);
242
```

Figure 20 : Initialisation du moteur permettant la propulsion (Carte embarquée)

En premier, on active la PWM1. Puis on relance le compteur MR0. Ensuite, on valide la PWM 1.2.

```
246     void Init_MOTEUR2(void){
247
248             LPC_GPIO3->FIODIR = LPC_GPIO3->FIODIR | (1<<26);
249
250             LPC_TIM0->PR=0;
251             LPC_TIM0->MR0=3750-1;
252
253             LPC_TIM0->MCR = LPC_TIM0->MCR | (3<<0);
254
255             LPC_TIM0->TCR = 1;
256
257     }
```

Figure 21 : Code de la fonction d'initialisation du servomoteur pour la direction (Carte embarquée)

VII. Pilotage Wireless

Par Kryspian et Charles

Une fois capable de faire avancer / reculer et tourner à gauche / droite grâce au PWM, nous nous sommes donc intéressés au pilotage à distance de la voiture. Pour cela, nous disposons d'un nunchuck.

Une nouvelle fois grâce à la documentation technique nous avons découvert que le nunchuck renvoyait les données via le protocole de communication I2C.
Cependant, une fois les données récupérées sur la carte, nous avons besoin de les transmettre à la carte embarquée sur la voiture.
Cette transmission se fait quant à elle en Bluetooth avec le protocole UART grâce à deux modules (un maître et un esclave).

```
void Init_I2C(void){
        Driver_I2C0.Initialize(i2c_nun);
        Driver_I2C0.PowerControl(ARM_POWER_FULL);
        Driver_I2C0.Control(    ARM_I2C_BUS_SPEED,                          // 2nd argument = d□bit
                                                ARM_I2C_BUS_SPEED_STANDARD  );  // 100 kHz
        Driver_I2C0.Control(    ARM_I2C_BUS_CLEAR,0 );
}
```

Figure 22 : Fonction d'initialisation du protocole I2C

```
  void Init_UART(void){
          Driver_USART1.Initialize(uart_nun);
          Driver_USART1.PowerControl(ARM_POWER_FULL);
          Driver_USART1.Control(  ARM_USART_MODE_ASYNCHRONOUS |
                                                  ARM_USART_DATA_BITS_8           |
                                                  ARM_USART_STOP_BITS_1           |
                                                  ARM_USART_PARITY_NONE           |
                                                  ARM_USART_FLOW_CONTROL_NONE,
                                                  115200);
          Driver_USART1.Control(ARM_USART_CONTROL_TX,1);
          Driver_USART1.Control(ARM_USART_CONTROL_RX,1);
  }
```

Figure 23 : Fonction d'initialisation du protocole UART

```
30      void i2c_nunchuck(void const* argument)
31      {
32              uint8_t programming[2];
33              uint8_t conversion_command[1];
34              uint8_t data_read_command[6];
35
36              char texte1[30];
37              char direction[1];
38
39              uint8_t PosX,PosY;
40
41              programming[0]= Command1;
42              programming[1]= Command2;
43
44              conversion_command[0]=ConversionCommand;
45
46              Driver_I2C0.MasterTransmit (SLAVE_I2C_ADDR , programming, 2, false);            // false = avec stop
47              osSignalWait(0x0001,osWaitForever);
48              osDelay(50);
```

Figure 24 : Début de la fonction du nunchuck

Sur le code ci-dessus, on initialise le transfert des données du nunchuck.

```
50              while(1)
51              {
52
53                      Driver_I2C0.MasterTransmit (SLAVE_I2C_ADDR, conversion_command, 1, false);              // false = avec stop
54                      osSignalWait(0x0001,osWaitForever);
55                      osDelay(5);
56
57                      Driver_I2C0.MasterReceive (SLAVE_I2C_ADDR, data_read_command,6, false);         // false = avec stop
58                      osSignalWait(0x0001,osWaitForever);
59
60                      PosX=data_read_command[0];
61                      PosY=data_read_command[1];
62
63
64                      if((PosX>=122)&&(PosX<=142)&&(PosY>=213)&&(PosY<=233)){//Avance
65                              sprintf(texte1,"mouvementA:  %4d %4d",PosX,PosY);
66                              direction[0]='A';
67                      }
68
69                      if((PosX>=196)&&(PosX<=216)&&(PosY>=190)&&(PosY<=210)){//AvanceDroite
70                              sprintf(texte1,"mouvementAD:  %4d %4d",PosX,PosY);
71                              direction[0]='P';
72                      }
73
74                      if((PosX>=47)&&(PosX<=67)&&(PosY>=188)&&(PosY<=208)){//AvanceGauche
75                              sprintf(texte1,"mouvementAG: %4d %4d",PosX,PosY);
76                              direction[0]='O';
77                      }
```

Figure 25 : Suite de la fonction du nunchuck

Après avoir initialiser la transmission des données, on effectue la réception des donnés convertie puis on fait des tests selon la valeur reçue, pour pouvoir envoyer un caractère permettant d'effectuer l'action souhaitée.

VIII. LIDAR

Par Dorian et Massyl

1. Présentation du Lidar

Le Lidar des technologies SLAMTEC propose le fonctionnement suivant : un laser est envoyé par une diode et la réflexion sur un objet est renvoyé à capteur. Le capteur récupère le signal et le traite. De là, on peut tirer une distance avec des calculs de Doppler.

Le Lidar RPLIDAR A2 sur lequel nous avons travaillé a la particularité d'être circulaire et peut prendre les informations autour de lui sur un plan. A chaque distance mesurée est associée une valeur d'angle.

Le Lidar est câblé de la manière suivante :

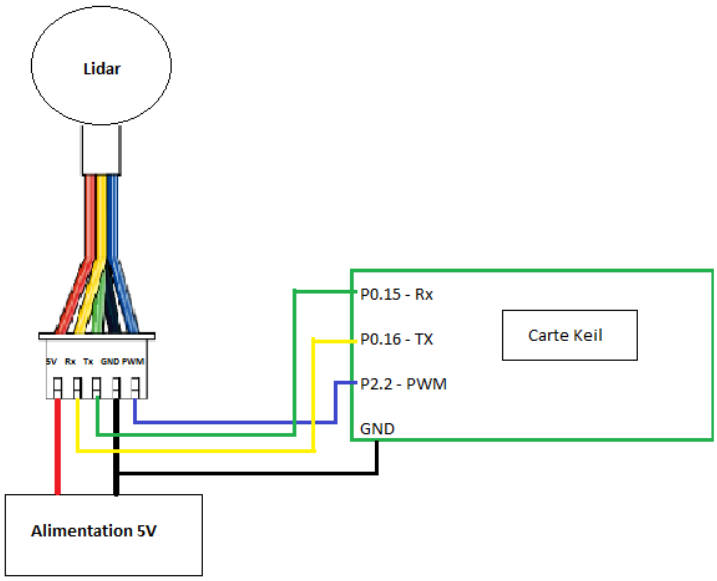


Figure 26 : Câblage du Lidar sur la carte Keil

2. Fonctionnement

Après avoir fait une version sans RTOS (car nous n'avions encore jamais discuté de ce mode de fonctionnement à ce moment-là), une version avec RTOS a été faite.

L'objectif de fonctionnement est le suivant :

- Une première trame doit être envoyée au Lidar pour lui demander de commencer l'acquisition des données.

- Avant de recevoir le flux de données, le Lidar confirme avoir reçu la trame en renvoyant un signal de témoin, le microcontrôleur lis la trame et si elle est celle qui est attendu, alors le code peut continuer de s'exécuter. Le cas contraire, nous avons scripté l'arrêt de la rotation du Lidar et de la diode laser. Une LED (la P2.7 ici) s'allume, témoignant d'un dysfonctionnement. Une fois l'arrêt effectué, après une légère attente (6 secondes environ pour être certain que la rotation soit bien arrêtée) le Lidar est relancé.

- Le signal de témoin renvoyé, le flux de données est émis. Celui-ci est découpé en trois parties :

1) La qualité : codée sur un octet et variant entre 0x3E et 0x01, elle est traitée de manière à ce que si la valeur est trop basse (et donc l'information serait potentiellement erronée), les quatre octets suivants soient oubliés

2) L'angle : codée en virgule fixe sur 2 octets (avec seulement 15 bits utiles) avec 9 bits pour la partie entière et 6 pour la partie décimale, est acquis et traité pour être compté comme un entier tant on ne prend pas en compte la partie décimale (et nous sommes certes moins précis mais nous n'avons pas besoin d'être précis à 10^-3 dans cette application).

3) La distance : codée sur 2 octets en virgule fixe avec 16 octets utiles : 14 pour la partie entière et 2 pour la partie décimale, est traitée pour que nous ne gardions que la partie entière. Cette partie entière désigne le nombre de mm entre l'objet et le lidar, une fois une conversion faite en entier, une multiplication par 0.001 est nécessaire pour l'avoir en mètre.

- L'acquisition et les traitements faits, ces valeurs de distance et d'angle sont renvoyées dans une autre tâche d'affichage via une mailbox. La mailbox faisant le lien entre les deux tâches est constituée de 2 éléments : un short contenant l'angle et un contenant la distance (les deux paramètres ont des noms équivoques dans le code).

64 exemplaires de la mailbox existent ; le but est de délayé le moment où l'utilisation de la CPU par l'écran fait planter le code (l'écran est très demandeur en CPU)

- La tâche d'affichage affiche sur l'écran avec une petite police (pour éviter que la ressource ne prenne trop de CPU et que le code plante parce que la CPU est tirée de tous les côtés) la distance mesurée (affichée en mm) et l'angle (en degrés). Un mutex permet d'assurer l'affichage complet des valeurs mesurées.

L'écran ici n'est pas vraiment adapté à l'application affichage des données en brut ; le Lidar peut envoyer 10000 octets (2000 mesures avec 5 octets à chaque fois) à la seconde là où l'affichage d'un caractère prend environ 1ms. Le plantage rapide du code est inévitable.

L'objectif pour éviter de faire planter le code est de procéder de la manière suivante : on doit récupérer toutes les données dans des tableaux, moyenner les distances entre les tranches de 10° (on moyenne les distances entre 0 et 10°, entre 10° et 20°…) afin de ne garder que 36 distances pour 36 angles. On pourrait du coup ne faire un affichage qu'à chaque tour de Lidar (quand l'angle revient à 0°).

L'objectif est d'avoir un affichage de la sorte avec les données traitées comme dit précédemment :

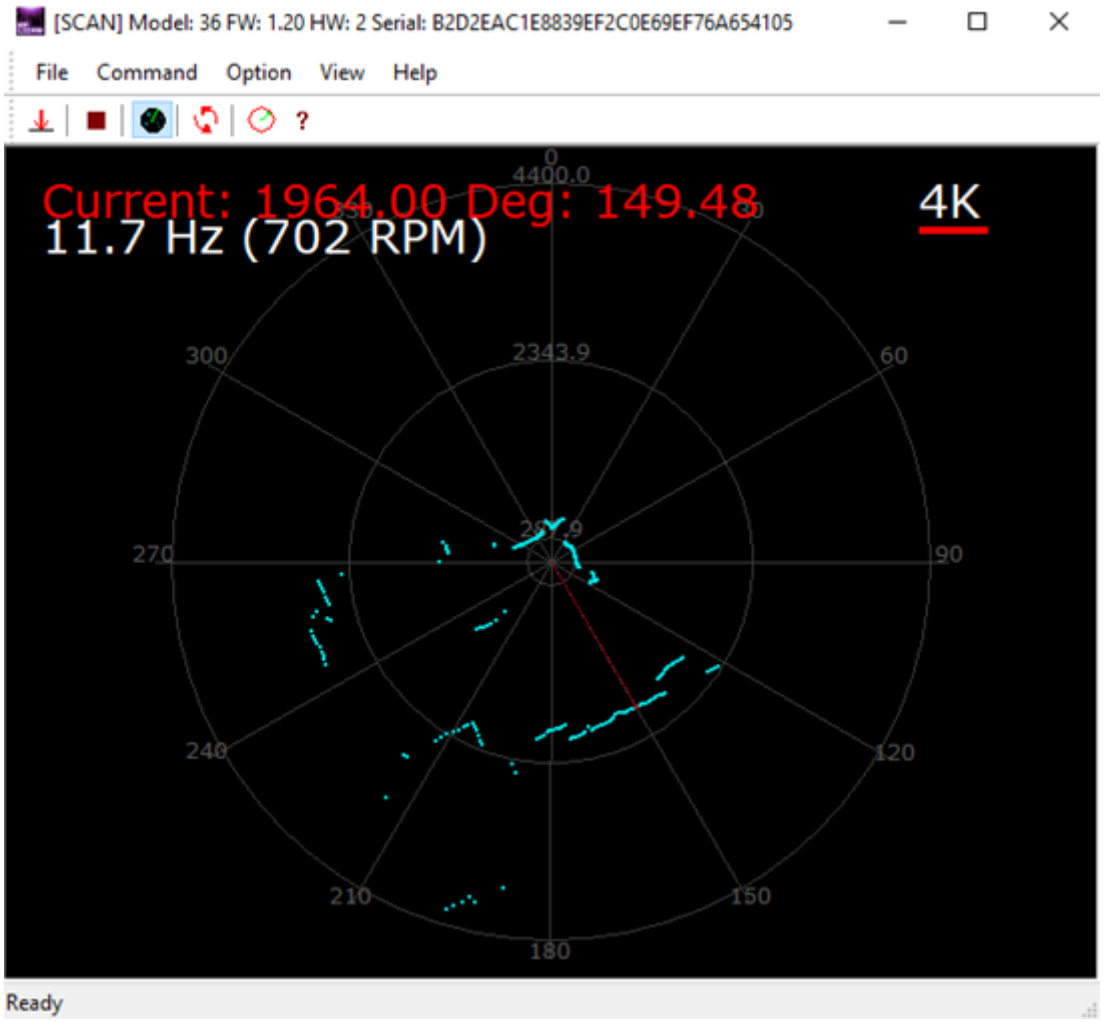


Figure 27 : Capture d'écran de frame Grabber, logiciel cartographiant l'entourage du LIDAR.

Conclusion

À la fin du semestre, nous avions accompli toutes les tâches présentes dans le cahier des charges, mis à part l'aide au stationnement et l'interconnexion. Le projet a été très enrichissant pour l'ensemble du groupe, qui a pu travailler sur un grand nombre d'aspects différents.

Du point théorique, l'ensemble des chapitres étudiés en Informatique Embarquée ont été traités, ainsi, les incompréhensions ont été effacées, de plus, le fait de combiner toutes ces compétences permet aussi de mieux comprendre leur intérêt global.

Au-delà de cela, le fait de travailler en méthode agile et d'utiliser le versioning a permis de travailler dans un contexte plus proche du milieu professionnel. Le travail d'équipe et l'organisation de groupe ont aussi été des compétences primordiales, puisque ce fut le premier projet réalisé en BUT avec des effectifs aussi riches, nous sommes passés de groupes de deux étudiants pour les précédents projets, à un groupe de dix.

Table des figures