Culture Sciences de l'Ingénieur

La Revue 3E.I

Édité le
Xx/xx/2024

école normale supérieure paris-saclay

CoVAPSY : Premiers programmes en langage C sur la voiture réelle

Antoine AZAN[1] – Anthony JUTON[2]

[1] *Professeur agrégé de génie électrique, Lycée Bergson, PARIS*
[2] *Professeur agrégé de physique appliquée au département Nikola Tesla de l'ENS Paris-Saclay*

Cette ressource fait partie du N°111 de La Revue 3EI de janvier 2024.

L'objectif de cette ressource est de présenter la configuration et programmation de la voiture autonome CoVAPSy en langage C à base d'un microcontrôleur de la famille des STM32 de STMicroelectronics.

La voiture type est présentée en détail dans la ressource « *Course Voitures Autonomes Paris Saclay (CoVAPSy) : Travaux pratiques autour des voitures autonomes* » [1]. Une version nommée CoVAPSy_STM32, sans nano-ordinateur, et une version encore plus simplifiée dite *CoVAPSy_STM32only, avec juste un lidar et un micro-contrôleur, sur une carte de prototypage, sont* présentée ici.

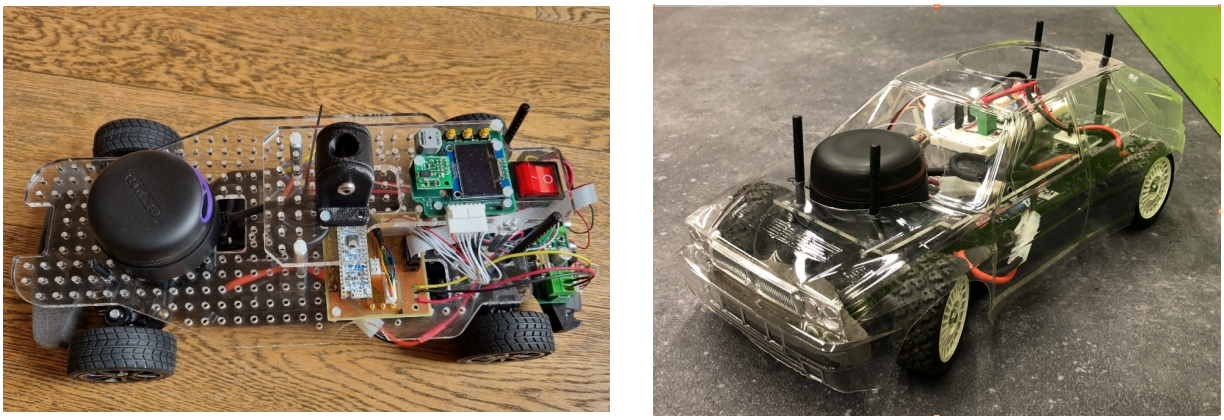

Figure 1: Voitures CoVAPSy_STM32 et CoVAPSy_STM32only

Cette ressource fait écho à la ressource sur la programmation de la voiture en langage Python « *CoVAPSy : Premiers programmes python sur la voiture réelle* » (2]. Les fonctions utilisées ici sur la voiture réelle sont les mêmes que les fonctions en langage C utilisées sur le simulateur webots présenté dans la ressource « *CoVAPSy : Mise en œuvre du Simulateur Webots* » [3].

Les bibliothèques et programmes de test en langage C sont disponibles sur le dépôt github :https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay/tree/main/Bibliotheques_logicielles/programmes_langageC_base_lidar_propulsion_direction_conduite.

1 - Architecture matérielle

Le diagramme de définition de bloc SysML ci-dessous représente la composition matérielle de la voiture autonome *CoVAPSy_STM32only*.

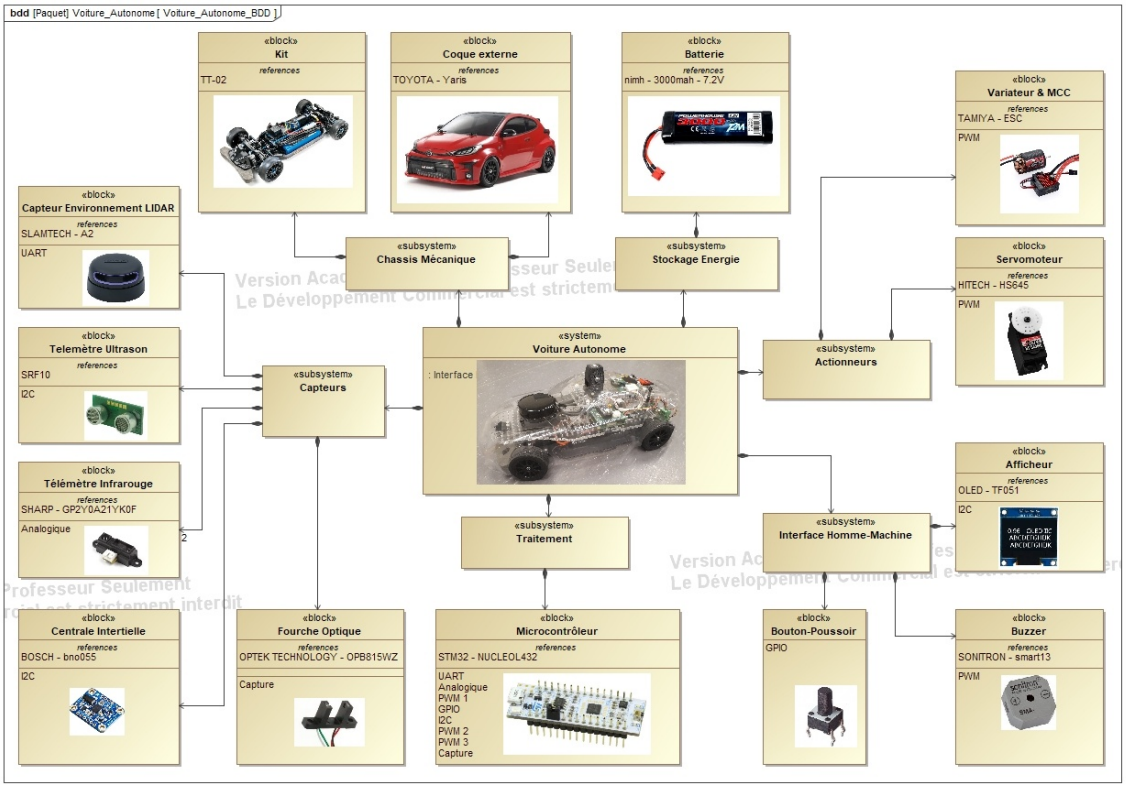

Figure 2 : Diagramme de définition de bloc SysML de la voiture autonome CoVAPSy_STM32only

Le diagramme de définition de bloc interne ci-dessous présente les flux de données entre les différents périphériques et le microcontrôleur.

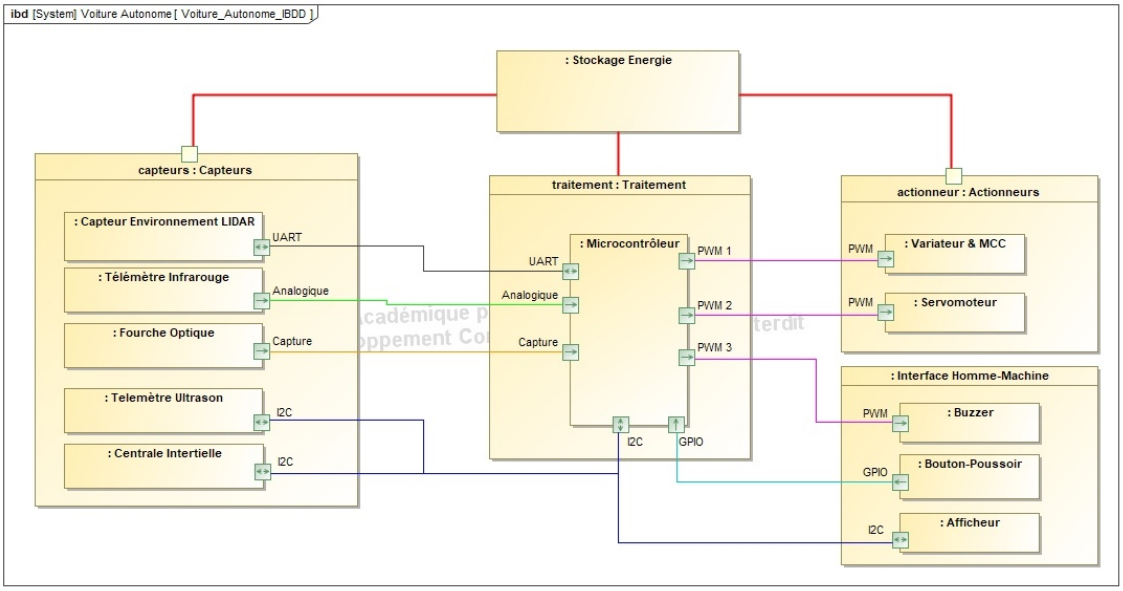

Figure 3 : Diagramme de définition de blocs internes de la voiture autonome CoVAPSy_STM32_only

Le microcontrôleur utilisé a donc besoin de gérer :

- Une liaison UART pour recevoir les données du Lidar

- Un bus I2C pour le télémètre à ultrason, la centrale inertielle et l'afficheur,
- Quatre signaux à modulation de largeur d'impulsion (Pulse Width Modulation PWM) pour la propulsion via l'ensemble variateur et moteur à courant continu, le servomoteur de direction, le Lidar et le buzzer
- Deux entrées analogiques pour les télémètres infrarouges
- Deux entrées Tout ou Rien (GPIO) pour les boutons-poussoirs
- Une entrée capture pour la fourche optique

Le microcontrôleur utilisé pour gérer l'ensemble des périphériques est un microcontrôleur STM32G431KB. Afin de faciliter l'intégration dans la voiture autonome CoVAPSy_STM32, la carte de développement NUCLEO-G431KB est utilisée. Cette carte de développement comporte un microcontrôleur STM32G431KB soudée sur une carte électronique au format Arduino nano. Le microcontrôleur STM32G431KB est basé sur un ARM Cortex-M4-32 bits, pouvant être cadencé jusqu'à une fréquence d'horloge de 170MHz avec une mémoire Flash de 128ko et un mémoire RAM de 32ko.

1.1 - Configuration matérielle minimale

Une configuration matérielle minimale comportant uniquement la partie châssis mécanique, l'ensemble variateur et moteur à courant continu, le servomoteur, le Lidar, la batterie et le microcontrôleur est envisageable. Cette configuration minimale permet, à un coût réduit de 472€ TTC, de mettre en œuvre la voiture autonome *CoVAPSy_STM32only*.

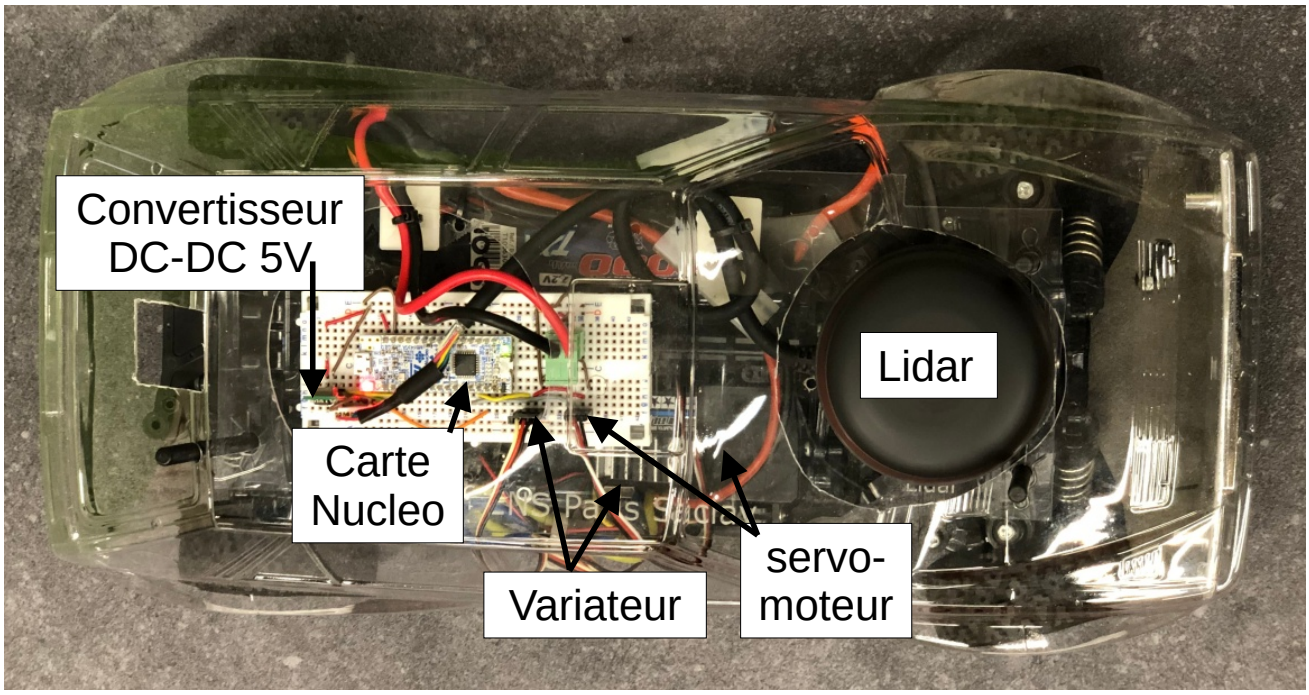

Figure 4: Voiture CoVAPSy_STM32only, avec une carte de prototypage

Le tableau ci-dessous liste les éléments de la configuration minimale *CoVAPSy_STM32only*, un exemple de distributeur ainsi que le prix de chaque élément.

Matériel	Description	Exemple de distributeur - référence	Prix unitaire TTC (€)
Chassis mécanique	Tamiya TT-02 Toyota GR 86	RCTeam - 58694	134,90
Batterie	T2M Accu 7.2v Nimh 3000mah	RCTeam - T1006300	27,30
Servomoteur de direction	Konect Servo 9kg 0.13s Digital KN-0913LVMG	RCTeam - KN-0913LVMG	19,90
Microcontrôleur STM32	Carte de développement With Stm32g431kb Mcu	RS 196-2534 NUCLEO-G431KB	13,78
Convertisseur DC-DC 5V	Régulateur de commutation Murata, sortie 5V, 1.5A, 7.5W	RS 796-2132 OKI-78SR	6,89
Lidar	RPLIDAR A2M12 360 Slamtec A2-M12	Robotshop - RB-Rpk-22	269,03
TOTAL			472

La figure ci-dessous présente le schéma de câblage des différents composants de cette configuration minimale. La connexion entre les différents périphériques peut se faire à l'aide d'une platine d'essai (breadboard) et de fils. Un circuit imprimé supportera mieux les vibrations.

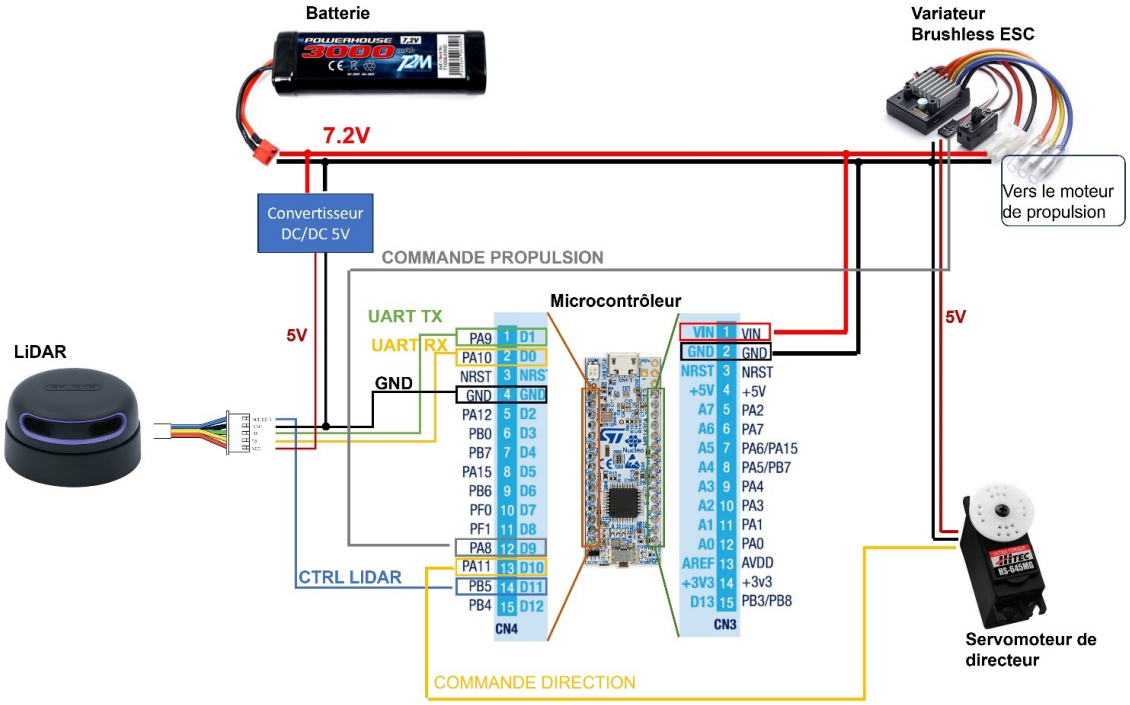

Figure 5: Schéma de câblage pour la configuration minimale CoVAPSy_STM32only

1.2 - Configuration matérielle complète

Une configuration matérielle complète avec l'ensemble des périphériques listés dans le diagramme de définition de bloc SysML permet de mettre en œuvre l'ensemble des fonctionnalités de la voiture autonome *CoVAPSy_STM32*.

Afin d'intégrer l'ensemble de ces périphériques sur la voiture autonome *CoVAPSy_STM32*, une carte électronique a été conçue spécifiquement afin de permettre l'intégration de la carte NUCLEO-G321KB dans la voiture dotée des cartes interface (convertisseur DC/DC et connectiques vers les différents capteurs et actionneurs et la batterie) et mezzanine (écran, boutons, buzzer). Les schémas et PCB des cartes sont sur le dépôt git de la course : https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay/tree/main/Materiel/Cartes_electroniques

Le schéma électronique ainsi que son circuit imprimé sont représentés ci-dessous.

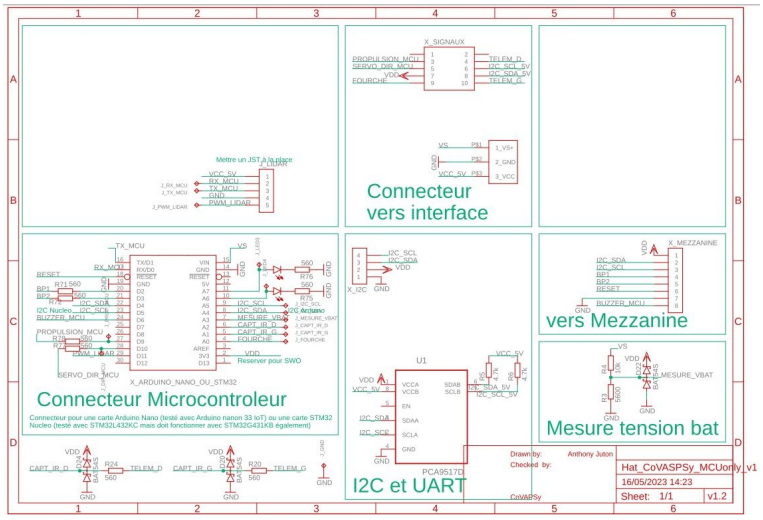

Figure 6 : Schéma électronique de la carte d'intégration du microcontrôleur STM32G431KB

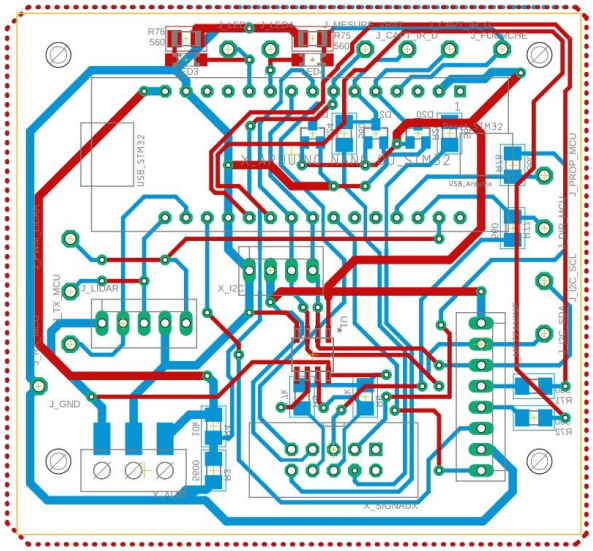

Figure 7 : Circuit Imprimé de la carte d'intégration du microcontrôleur STM32G431KB

Le microcontrôleur est programmé à l'aide du logiciel STM32CubeIDE version 1.13.1 développé par ST Microelectronics. Ce logiciel permet la configuration des périphériques, des broches, de l'horloge, la génération du code d'initialisation, la compilation et le téléversement du programme ainsi que le débogage.

2 - Mise en œuvre des entrées/sortie et bibliothèques associées

Dans cette partie, il est fait référence à de nombreuses bibliothèques et fonctions spécifiques à chaque périphérique. Toutes les bibliothèques et le programme de démonstration complet sont disponibles sur le dépôt github :

https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay/tree/main/Bibliotheques_logicielles/programmes_C_base_lidar_propulsion_direction_conduite.

2.1 - Configuration des entrées/sorties du microcontrôleur STM32

Précédemment, l'ensemble des broches nécessaires pour commander les actionneurs et recevoir les données des capteurs a été listé.

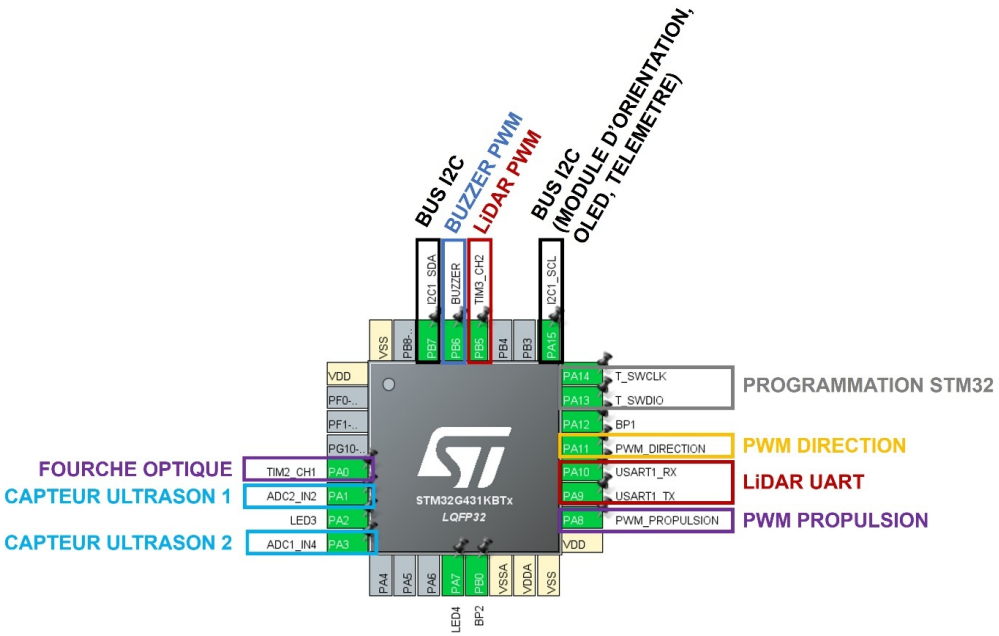

Figure 8 : Configuration complète des broches du STM32

Le tableau ci-dessous résume les broches du microcontrôleur STM32 utilisé pour chaque périphérique.

PERIPHERIQUES	BROCHES	TYPE
Lidar	PA9	Liaison série Tx
	PA10	Liaison série Rx
	PB5	PWM sur Timer 3 Channel 2
PROPULSION	PA8	PWM sur Timer 1 Channel 1
DIRECTION	PA11	PWM sur Timer 1 Channel 4
CENTRALE INERTIELLE	PA15	I2C SCL
	PB7	I2C SDA
AFFICHEUR OLED	PA15	I2C SCL
	PB7	I2C SDA
TELEMETRE ULTRASON	PA15	I2C SCL
	PB7	I2C SDA
TELEMETRE INFRAROUGE 1	PA1	Entrée Analogique
TELEMETRE INFRAROUGE 2	PA3	Entrée Analogique
BUZZER	PB6	PWM sur Timer 4 Channel 1
FOURCHE OPTIQUE	PA0	Timer 2 - Channel 1 en mode capture

Une présentation détaillée de la mise en œuvre des périphériques principaux : Lidar, Moteur à propulsion et servomoteur est proposée ci-dessous. La mise-en-œuvre des autres périphériques (Centrale inertielle, fourche optique, capteur ultrason, télémètre, afficheur OLED) sera présentée via des fiches individuelles qui seront publiées sur le dépôt github.

2.2 - Lidar

Le Lidar permet de sonder l'environnement de la voiture autonome *CoVAPSy_STM32* à 360°. Les mesures du Lidar vers l'arrière de la voiture sont perturbées par l'habitacle de la voiture.

Le Lidar utilisé est un Lidar de la marque SLAMTECH de référence A2M12. Le Lidar envoie les données via une liaison série avec un débit binaire de 256 000 bps. La version A2M8 encore

disponible chez certains revendeurs fonctionne à 115200 bps, débit plus facile à gérer par la carte microcontrôleur.

La liaison série du microcontrôleur est configurée de la manière suivante :

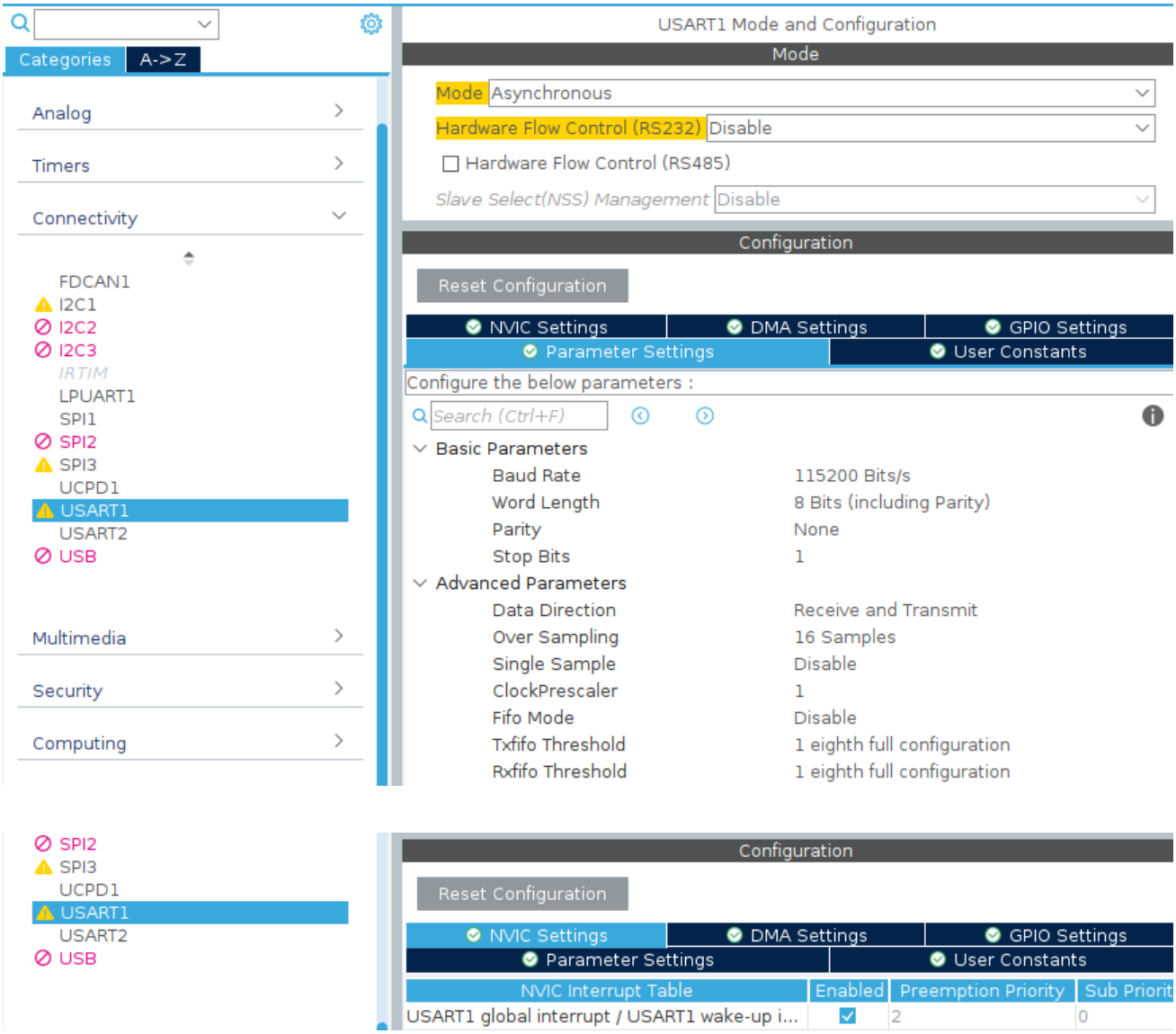

Figure 9: Configuration de la liaison série associée à un Lidar A2M8 (115200 bit/s)

La librairie *CoVAPSy_Lidar* comporte une fonction permettant d'initialiser le Lidar et de fixer sa vitesse de rotation. La fonction *Lidar_init()* démarre et réinitialise le Lidar. Un signal PWM généré par le microcontrôleur permet de fixer la vitesse de rotation du Lidar. La fréquence du timer associé à cette PWM est fixée à 10 MHz. La période de la PWM est fixée à 40µs. La durée à l'état haut de la PWM est fixée à 30µs. La fonction *Lidar_init()* envoi la requête *START_SCAN* pour démarrer les acquisitions Lidar, puis cette fonction active l'interruption de la liaison UART associée au Lidar.

La réception des données du Lidar s'effectue via une liaison série UART sur interruption. A chaque réception d'un octet, la donnée reçue est stockée dans une variable *Data_RX_LIDAR* de type entier non-signé de 16 bits.

Les sept premiers octets émis par le Lidar suite à la requête *START SCAN* sont des octets dit *descriptor* ne contenant pas d'information de mesure d'angle ou de distance. Ces sept premiers octets reçus ne sont pas donc stockés dans le tableau *Data_Lidar_mm*.

Une fois le *descriptor* passé, les données sont envoyées par groupe de 5 octets contenant l'information de qualité de la mesure (codée sur 1 octet), l'information d'angle de la mesure (codée sur 2 octets) et l'information de distance mesurée (codée sur 2 octets).

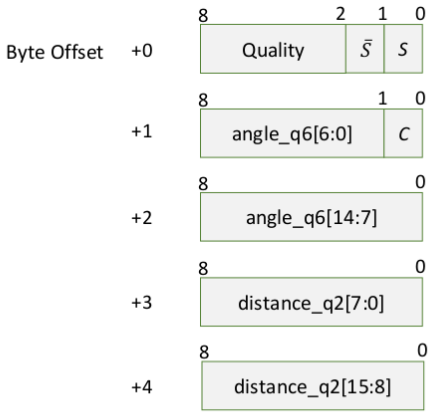

Figure 4-4 Format of a RPLIDAR Measurement Result Data Response Packet

Figure 10: Format des paquets envoyés par le Lidar, extrait de la documentation slamtec_rplidar_protocol

À chaque nouveau groupe de 5 octets reçus, le programme extrait les informations d'angle et de distance puis stocke la distance dans un tableau de 360 lignes nommé *Data_Lidar_mm*. Chaque index du tableau *Data_Lidar_mm* correspond à un angle de mesure compris entre 0° et 359°. La donnée reçue est stockée dans le tableau *Data_Lidar_mm* si la qualité de la mesure associée est suffisante. Les angles dans le programme sont comptés dans le sens trigonométriques, comme sur le simulateur, alors que le lidar les mesure dans le sens horaire. C'est la raison de la courte conversion avant le stockage dans le tableau.

Le code ci-dessous présente le programme pour réceptionner les trames du Lidar à chaque interruption et compléter le tableau *Data_Lidar_mm*. Le tableau *Data_Lidar_mm* peut être utilisé dans le programme principal pour prendre la décision de vitesse et de direction de la voiture. La drapeau *drapeau_fin_tour*, mis à un par la fonction d'interruption indique que le lidar a fini son tour (passage à -100°). C'est le bon moment pour le programme principal pour récupérer des données fraîches et remettre le drapeau à zéro.

Le programme principal :

```
/* USER CODE BEGIN PV */
//////////////////////     Variables pour le Lidar          //////////////
uint8_t Data_RX_LIDAR;
uint16_t Data_Lidar_mm[360];
uint8_t drapeau_fin_tour = 0;
uint16_t data_lidar_mm_main[360];
…

int main(void)
{
       uint8_t drapeau_fin_tour_old = 0;    // Variable pour le Lidar
       uint32_t i = 0;
 (...)

  /// Initialisation du Lidar ///
  HAL_UART_Receive_IT(&huart1, &Data_RX_LIDAR, 1);

  while (1)
  {
  //Recopie du tableau lidar en fin de tour                                    ///////
        if((drapeau_fin_tour == 1) && (drapeau_fin_tour_old == 0))
             {
               for (i=0;i<360;i++)
```

```
                        data_lidar_mm_main[i]=Data_Lidar_mm[i];
                }
        drapeau_fin_tour_old = drapeau_fin_tour;
  }
}
```

La fonction d’interruption :

```c
/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
	static uint32_t index = 0;
	static uint8_t drapeau_demarrage = 0; // Variable indiquant pour réception des 7 1er octets
	static uint8_t tableau_trame[7];      // Variable de stockage des 7 premiers octets
	uint16_t angle;
	uint16_t distance;
	  ////////////////////////////////////////////////////////////////////////////
	  /////// Réception des sept octets correspondant au descriptor     ///////////
	  ////////////////////////////////////////////////////////////////////////////
	if(drapeau_demarrage == 0){
		tableau_trame[index]=Data_RX_LIDAR;
		index++;
	}
	if ((index == 7) && (drapeau_demarrage == 0)) {
		index = 0;
		drapeau_demarrage = 1;                      // Fin des sept octets du descriptor
	}

	  ////////////////////////////////////////////////////////////////////////////
	  /////// Réception des données par groupe de 5 octets               /////////
	  /////// Qualité (1 octet), Angle (2 octets) et Distance (2 octets)   /////////
	  ////////////////////////////////////////////////////////////////////////////
	if (drapeau_demarrage == 1) {
		if (index <= 4) {
			if ((Data_RX_LIDAR == 0x3e)||(Data_RX_LIDAR == 0x3d)) //trame de démarrage
				index = 0;
			tableau_trame[index] = Data_RX_LIDAR;
			if(tableau_trame[0] > 0x02)
				index++;
		}

		  ////////////////////////////////////////////////////////
		  /////// Traitement de donnée pour convertir       /////////
		  /////// les octets reçus en données réelles       /////////
		  ////////////////////////////////////////////////////////
		if (index >= 5) {
			index = 0;
			angle = ((uint16_t) tableau_trame[2] << 1)
					+ ((uint16_t) tableau_trame[1] >> 7);
			distance = ((uint16_t) tableau_trame[4] << 6)
					+ ((uint16_t) tableau_trame[3] >> 2);

			// Stockage de la dist. mesurée dans le tableau à l'indice associé à l'angle
			if (angle == 0)
				Data_Lidar_mm[0] = distance;
			if ((angle < 360) && (angle>0))
				Data_Lidar_mm[360-angle] = distance;

			  ////////////////////////////////////////////////////////
			  /////// Détection de la fin d'un tour        /////////
			  ////////////////////////////////////////////////////////
			if ((angle > 100) && (angle < 180) && (drapeau_fin_tour == 0)) {
				drapeau_fin_tour = 1;
			}
			if ((angle > 270) && (angle < 360) && (drapeau_fin_tour == 1)) {
				drapeau_fin_tour = 0;
			}
		}
	}
	///// Ré-activation de l'interruption UART RX
	HAL_UART_Receive_IT(&huart1, &Data_RX_LIDAR, 1);
}
/* USER CODE END 4 */
```

La figure ci-dessous montre le programme de mise en œuvre du Lidar en cours d'exécution en mode debugage avec la visualisation en direct de la variable *Data_Lidar_mm*.

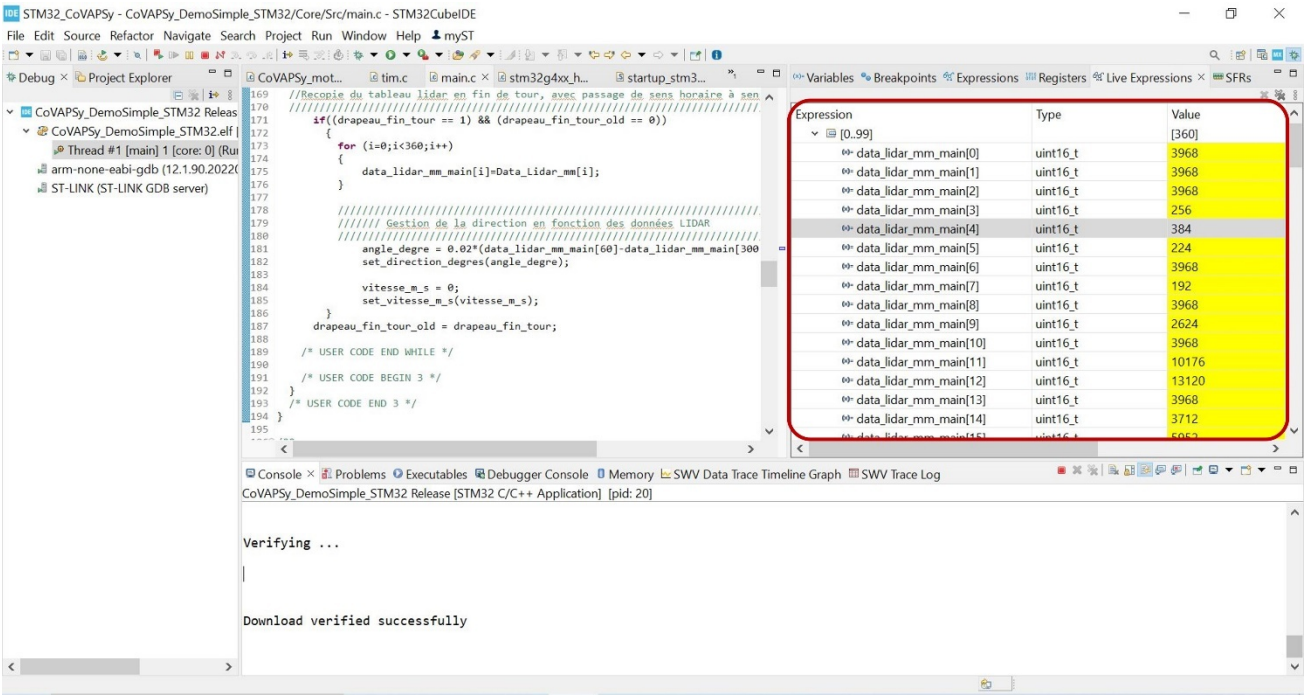

Figure 11: Programme de mise en œuvre du Lidar avec visualisation de la variable Data_Lidar_mm

Il peut être nécessaire d'effectuer un reset hardware du microcontrôleur ou un ON/OFF général de la voiture autonome pour un bon fonctionnement du Lidar, celui-ci hésitant à redémarrer une connexion suite à une interruption de la connexion précédente.

2.3 - Moteur de propulsion

Le moteur de propulsion est géré par un signe de type PWM associé au timer 1 et channel 4. La fréquence de l'horloge du timer est fixé à 1MHz et le compteur est limité à 20 001. La configuration du timer est précisé dans la figure ci-dessous.

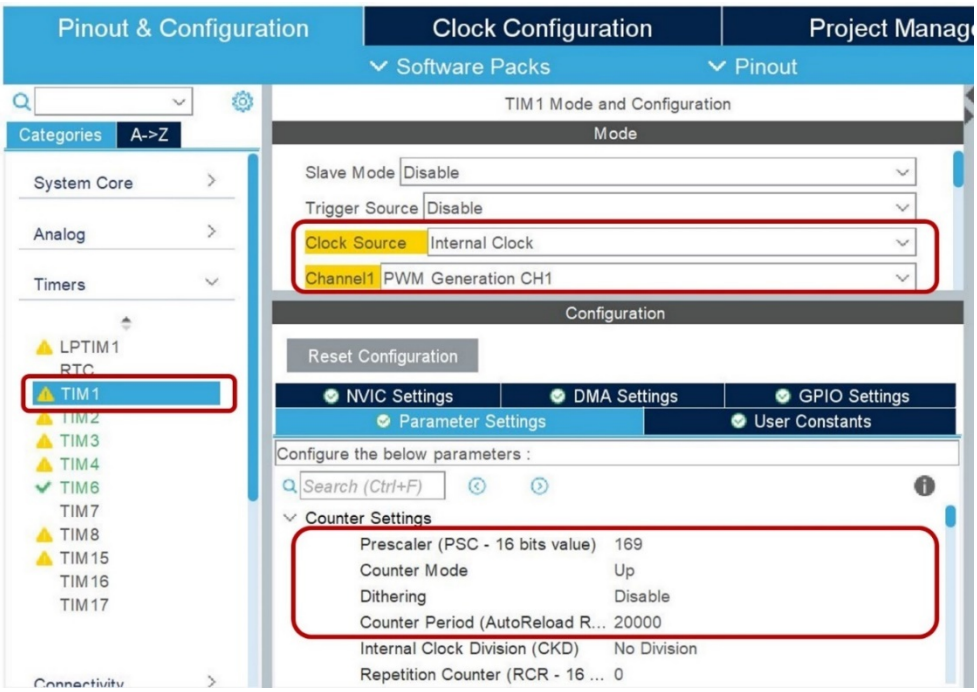

Figure 12: Configuration du timer 1 associé à la propulsion

La librairie *CoVAPSy_moteurs* comporte des fonctions permettant de commander le variateur de vitesse et donc la propulsion de la voiture. Parmi les fonctions de la librairie, la fonction

Propulsion_init() permet d'initialiser la commande du moteur de propulsion. La fonction *set_vitesse_m_s(vitesse_m_s)* permet de commander le moteur de propulsion à une vitesse fixée en m/s. Le code ci-dessous est en exemple pour initialiser la commande du moteur puis le commander avec une vitesse approximative de +2.5m/s.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "CoVAPSy_moteurs.h"
/* USER CODE END Includes */

( …)

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
 (…)
int main(void)
{
  (...)

  /* USER CODE BEGIN 2 */
  Propulsion_init();          // Initialisation de la propulsion
  set_vitesse_m_s(2.5);               // Propulsion fixée à 2.5m/s
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Le code ci-dessous détaille les fonctions *Propulsion_init*() et *set_vitesse_m_s()*.

```
void Propulsion_init(void){
	HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
}



void set_vitesse_m_s(float vitesse_m_s){
	uint32_t largeur_impulsion_us;
	if (vitesse_m_s == 0)
	{
		largeur_impulsion_us = PROP_REPOS ;
	}
	else if (vitesse_m_s < 0){
		if(vitesse_m_s < -V_MAX_HARD)
			vitesse_m_s  = -V_MAX_HARD;
		largeur_impulsion_us = PROP_POINT_MORT_NEG + (PROP_MAX - PROP_POINT_MORT) *
vitesse_m_s / V_MAX_HARD;
//version variateur bizarre
//largeur_impulsion_us = PROP_POINT_MORT - (PROP_MAX - PROP_POINT_MORT) * vitesse_m_s /
V_MAX_HARD;
	}
	else if (vitesse_m_s > 0){
		if (vitesse_m_s > V_MAX_SOFT)
			vitesse_m_s = V_MAX_SOFT;
		largeur_impulsion_us = PROP_POINT_MORT + (PROP_MAX - PROP_POINT_MORT) *
vitesse_m_s / V_MAX_HARD;
//version variateur bizarre
//largeur_impulsion_us = PROP_POINT_MORT_NEG - (PROP_MAX - PROP_POINT_MORT) * vitesse_m_s /
```

```
V_MAX_HARD;
        }
        __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, largeur_impulsion_us);
}
```

Il est important de noter que la vitesse maximale de la voiture est fixée à 8 m/s dû aux caractéristiques techniques du variateur. A noter que même Tamiya livrant ses voitures avec des variateurs de référence différente, il a été constaté que certains variateurs devaient se commander différemment des autres (l'impulsion est entre 1 et 1,5 ms pour la marche avant et entre 1,5 et 2 ms pour la marche arrière). Ainsi, un rapport cyclique de 1,8 ms peut faire tourner la voiture en marche avant ou en marche arrière ! C'est pour ces variateurs que le code de la bibliothèque *CoVAPSy_moteurs* comporte des lignes de code en commentaire. Il est donc important de prendre le temps de bien régler les paramètres de sa bibliothèque *CoVAPSy_moteurs*.

Dans la bibliothèque *CoVAPSy_moteurs*, La fonction *recule()* permet de faire reculer la voiture en cas de rencontre d'un obstacle.

2.4 - Servomoteur de direction

Le servomoteur de direction est géré par un signal de type PWM dont les caractéristiques sont les mêmes que celle de la direction. En effet, le même timer est utilisé pour générer la PWM de la direction et de la propulsion. La configuration de ce timer est précisé dans la partie associé à la propulsion.

La librairie *CoVAPSy_moteurs* comporte des fonctions permettant de commander le servomoteur de direction. Parmi les fonctions de la librairie, la fonction *init_direction*() permet d'initialiser la commande du moteur de propulsion. La fonction *set_direction_degres(angle_degre)* permet de commander le servomoteur de direction à un angle de direction défini en argument de cette fonction. Le code ci-dessous est en exemple pour commander le servomoteur de direction avec un angle de +10°.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "CoVAPSy_moteurs.h"
/* USER CODE END Includes */

(...)

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
 (…)
int main(void)
{
  (...)

  /* USER CODE BEGIN 2 */
  Direction_init();          // Initialisation de la direction
  set_direction_degres(10);  // Direction fixée à 10°
```

```
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Le code ci-dessous détaille les fonctions *Direction_init*() et *set_direction_degres()*.

```
void Direction_init(void){
	HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_4);
}

void set_direction_degres(float angle_degre)
{
	uint32_t largeur_impulsion_us;
	if (angle_degre < -DIR_ANGLE_MAX)
		angle_degre = -DIR_ANGLE_MAX;
	else if (angle_degre > DIR_ANGLE_MAX)
		angle_degre = +DIR_ANGLE_MAX;
	largeur_impulsion_us = DIR_MILIEU + (DIR_BUTEE_GAUCHE - DIR_BUTEE_DROITE)*angle_degre/
(2*DIR_ANGLE_MAX);
	__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, largeur_impulsion_us);

}
```

Attention, suivant les servo-moteurs, un rapport cyclique de 1,8 ms peut faire tourner la direction vers la gauche ou vers la droite ! Il est donc important de prendre le temps de bien régler les paramètres de sa bibliothèque *CoVAPSy_moteurs*. De plus, il est important de signaler que l'architecture mécanique de la voiture limite l'angle de braquage entre -18° et +18°.

3 - Exemple de code complet

Le programme de démonstration permet de mettre en œuvre l'ensemble des périphériques présentés précédemment. Il permet de :

- Faire tourner les roues à deux angles de directions différents (+10° puis -10°),
- Faire avancer la voiture à différentes vitesses (1m/s, 2m/s puis 3m/s),
- Faire reculer la voiture,
- Réceptionner les données du Lidar,
- Gérer la direction en fonction des données du Lidar.

Dans ce programme de démonstration, la propulsion est fixée à une vitesse lente de 0.5 m/s. La gestion de la direction dépend de la mesure des obstacles à +60° et -60°. En fonction de ces deux mesures de distance, le programme modifie l'angle de direction de la voiture.

Le programme complet de démonstration est disponible sur le dépôt github et présenté ci-dessous.

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
//// Ajout des bibliothèques CoVAPSy_moteur et CoVAPSy_Lidar
#include "CoVAPSy_moteurs.h"
#include "CoVAPSy_Lidar.h"
```

```c
/* USER CODE END Includes */

(...)
/* USER CODE BEGIN PV */
////////////////////////////////////////////////////////////////////////
//////////////////////     Variables pour le Lidar          //////////////
////////////////////////////////////////////////////////////////////////
uint8_t Data_RX_LIDAR;
uint16_t Data_Lidar_mm[360];
uint8_t drapeau_fin_tour = 0;
uint16_t data_lidar_mm_main[360];

////////////////////////////////////////////////////////////////////////
/////////     Variables pour la propulsion et direction          ////////
////////////////////////////////////////////////////////////////////////
float angle_degre,vitesse_m_s;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/

/* USER CODE BEGIN PFP */
void SystemClock_Config(void);
/* USER CODE END PFP */
(...)
/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  (...)
  /* USER CODE BEGIN 2 */
    ////////////////////////////////////////////////////////////////////
  ///////// Initialisation de la propulsion et de la direction ///////
  ////////////////////////////////////////////////////////////////////
  Propulsion_init();
  Direction_init();
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */

  ////////////////////////////////////////////////////////////////////
  ////////// Commande Direction à différents angles //////////////////
  ////////////////////////////////////////////////////////////////////
  set_direction_degres(0);
  HAL_Delay(500);
  set_direction_degres(-10);
  HAL_Delay(500);
  set_direction_degres(10);
  HAL_Delay(500);
  set_direction_degres(0);
  HAL_Delay(500);



  ////////////////////////////////////////////////////////////////////
  ////////// Commande Propulsion à différentes vitesses //////////////
  ////////////////////////////////////////////////////////////////////
  set_vitesse_m_s(1.0);
  HAL_Delay(1000);
  set_vitesse_m_s(2.0);
  HAL_Delay(1000);
  set_vitesse_m_s(3.0);
  HAL_Delay(1000);
  set_vitesse_m_s(0.0);
  HAL_Delay(1000);
  recule();
  HAL_Delay(1000);
  set_vitesse_m_s(0.0);
  HAL_Delay(1000);

  ////////////////////////////////////////////////////////////////////
  ////////////////////// Initialisation du Lidar /////////////////////
  ////////////////////////////////////////////////////////////////////
  Lidar_init();
```

```
  while (1)
  {
  ////////////////////////////////////////////////////////////////////////////////
  //Recopie du tableau lidar en fin de tour, avec passage de sens horaire à sens trigo///////
  ////////////////////////////////////////////////////////////////////////////////
        if((drapeau_fin_tour == 1) && (drapeau_fin_tour_old == 0))
              {
                for (i=0;i<360;i++)
                {
                      data_lidar_mm_main[i]=Data_Lidar_mm[i];
                }
                ////////////////////////////////////////////////////////////////////////////
                /////// Gestion de la direction en fonction des données LIDAR
                ////////////////////////////////////////////////////////////////////////////
                 //distance à 60° - distance à -60°
                angle_degre = 0.02*(data_lidar_mm_main[60]-data_lidar_mm_main[300]);
                set_direction_degres(angle_degre);
                vitesse_m_s = 0.5;
                set_vitesse_m_s(vitesse_m_s);
              }
         drapeau_fin_tour_old = drapeau_fin_tour;
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

4 - Axes d'amélioration

Outre le travail sur les algorithmes de conduite autonome, il existe plusieurs axes d'améliorations des performances de la voiture commandée par un microcontrôleur STM32. Tout d'abord la mise en place d'un asservissement numérique de vitesse permettrait de mieux contrôler la vitesse de la voiture. En remplaçant le servomoteur analogique par un servomoteur numérique de type Dynamixel XL430-W250-T, on gagne en dynamique et on peut lire la position réelle de la direction.

Actuellement, la voiture autonome *CoVAPSy_STM32*, avec un Lidar A2M12 à 256 kbit/s) peut présenter un défaut de réactivité à l'approche d'un obstacle. En effet, le Lidar envoie beaucoup de donnée au microcontrôleur ce qui engendre une forte activité pour cette tâche. Il pourrait être intéressant de mieux gérer ce flux de donnée. Un microcontrôleur plus rapide STM32H7 résoudrait le problème également.

5 - Conclusion

Cet article présente une voiture autonome *CoVAPSy_STM32only* gérée entièrement par un seul microcontrôleur programmé en C.

Les bibliothèques et programmes proposés permettent de mettre en œuvre rapidement les périphériques principaux afin de laisser place à la créativité pour la partie stratégie de course.

Cette voiture autonome destinée principalement aux étudiants d'école d'ingénieur ou d'IUT peut également être un support d'étude de projet pour les étudiants de BTS CIEL ou les élèves de STI2D. Les possibilités pédagogiques sont nombreuses. Un futur article présentera un projet de projet pour le BTS CIEL option ER basé sur cette voiture.

Références :

[1]: Course Voitures Autonomes Paris Saclay (CoVAPSy) : Travaux pratiques autour des voitures autonomes, T. Boulanger, E. Délègue, K. Hoarau, A. Juton, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/covapsy-tp-autour-des-voitures-autonomes

[2]: CoVaPSy : Premiers programmes python sur la voiture réelle, T. Boulanger, E. Délègue, K. Hoarau, A. Juton, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/covapsy-premiers-programmes-python-sur-voiture-reelle

[3]: CoVaPSy : Mise en œuvre du Simulateur Webots, T. Boulanger, E. Délègue, K. Hoarau, A. Juton, https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/covapsy-mise-en-oeuvre-du-simulateur-webots

[4]: Ressource publiée sur Culture Sciences de l'Ingénieur : https://eduscol.education.fr/sti/si-ens-paris-saclay