```
// Projet VOIE-TURFU IéTI 2021 Groupe 5 : Meguebel Mohamed, Moutakki Ali, Moutaouakil Zakariae, Noël Vincent et Loiselet Etienne

#include "mbed.h"

//////////////////////////////      PORTS NUCLEO      ////////////////////////////////
PwmOut mot(PC_9);                                   // Port moteur
PwmOut servo(PB_13);                                // Port servomoteur
Serial pc(USBTX, USBRX);                            // Port communication PC
Serial BT(PC_10,PC_11);                             // Port communication module BlueTooth
//AnalogIn capteur(PC_2,PC_3,A5,A4,A3,A2);          // Entrées des capteurs (check carte)
Serial __lidar(A0,A1,115200);                       // Port communication Lidar
PwmOut __rotation(PB_9);                            // Commande en vitesse du Lidar
DigitalOut tour_ok(D13);                            // Indispensable au fonctionnement du Lidar
AnalogIn capD(PC_2);                                // Capteur IR droite
AnalogIn capG(A2);                                  // Capteur IR gauche
//////////////////////////////      PORTS NUCLEO      ////////////////////////////////



////////////////////////        DECLARATION FONCTIONS        ////////////////////////////
void Avant(void);               // Fonction fait avancer la voiture à la vitesse v
void Arriere(void);             // Fonction fait reculer la voiture à la vitesse v
void BT_stop(void);             // Fonction qui lit la telecommande BT
int seuil_frontal();            // Ancien !! fonction qui renvoie le seuil de distance (capteur IR)
void Gauche(void);              // Fonction qui augmente la rotation des roue vers la gauche
void Droite(void);              // Fonction qui augmente la rotation des roues vers la droite
void Milieu(void);              // Fonction qui réinitialise la direction des roues
void IRQ_Rx_Lidar(void);            // Fonction pour Lidar
void IRQ_dataScan(void);            // Fonction pour Lidar
void findMaxDist(void);         // Fonction Lidar pour obtenir maxDist
void findMinDist(void);         // Fonction Lidar pour obtenir minDist
void get_lidar(void);           // Ticker du lidar

void GoD(int angle);            // Fonction qui décide de tourner à droite ou à gauche
bool DetectionD(void);          // Fonction qui gère le capteur IR droit
bool DetectionG(void);          // Fonction qui gère le capteur IR gauche
////////////////////////        DECLARATION FONCTIONS        ////////////////////////////



////////////////////////        DECLARATION VARIABLES   ////////////////////////////
double v=0;                     // "booléen" pour choisir entre vitesse nulle ou non
int t=1500;                     // pulsation du moteur (1500 = arrêt)
char valeur='0';                // Commande BT
int l=1270;                     // Commande en direction du servomoteur (1285 = milieu)
int a=600;                      // Durée des phases d'accélération (ms)
```

```
int __acq_lidar_1[360];          // Distance in function of angle collected by Lidar
int __acq_lidar_1_old[360];              // Storage of distance in function of angle collected by
Lidar
int __somme;
int __maxDist;                   // distance maximale
int __minDist;                   // distance minimale
int __angleDistMax;              // angle associé à maxDist
int __angleDistMin;              // angle associé à minDist
unsigned char __data_tab[7];             // Data received by Lidar
unsigned char tab[7]= {};        // necessaire pour le Lidar
uint16_t acq_lidar_1[360]= {};           // necessaire pour le lidar
uint16_t acq_lidar_2[360]= {};           // necessaire pour le lidar
int num_tab_acq=1 ;              // necessaire pour le lidar
int flag_chgt_tableau = 0;       // necessaire pour le lidar
float commande_moteur;                   // necessaire pour le lidar
float distance_max=0;            // necessaire pour le lidar

Ticker T_get_lidar;              // Ticker recevant les informations du Lidar
int seuil_capteur_IR = 1000;             // Seuil de détection pour les capteurs IR
int seuil = 1000;                // Premier seuil de détection pour le Lidar
int seuil0 = 250;                // Second seuil de détection pour le Lidar
int mes;
//////////////////////          DECLARATION VARIABLES   ////////////////////////



///////// Pour le moteur ! il est indispensable de l'initialiser à 1500 au début
// le temps que le transfo fasse deux bips, ensuite les vitesse en dessous de 1600
// ne fonctionne pas, pour des manoeuvres précises il vaut mieux utiliser 1615
// dans les zones sûres utiliser 1620 et plus

int main() {
        // Lidar Rotation //
        __rotation.period_us(40);
        __rotation.pulsewidth_us(20); //vitesse rotation

        //###########  Initialisation Moteur   ###########//
        mot.period_ms(20);               // Initialisation période
        mot.pulsewidth_us(1500);         // Initialisation en position 0
        wait_us(3000000);                // Temps de calibration
        //###########  Initialisation Moteur   ###########//

        //###########  Initialisation Servomoteur   ###########//
        servo.period_ms(20);             // Initialisation en période
        servo.pulsewidth_us(l);          // Initialisation tout droit
        wait_us(3000000);                // Temps de calibration
```

```
	//########## Initialisation Servomoteur   ##########//

	pc.baud(115200);		// Liaison PC (Terraterm)

	//########## Initialisation Bluetooth   ##########//
	BT.baud(9600);			// Débit de communication
	BT.printf("Bluetooth on\n");		// Affichage de réussite de connection sur le
téléphone
	BT.attach(&BT_stop, Serial::RxIrq);  // Initialisation des procédure d'iinterruption par
bluetooth
	//########## Initialisation Bluetooth   ##########//


	//################   Initialisation Lidar   ################//
	int i=0;
	__lidar.putc(0XA5);
	__lidar.putc(0X20); //Request
	while(i<7) {
	if(__lidar.readable()) {
	tab[i]=__lidar.getc(); //Response
	i++;
	}
	}
	__lidar.attach(IRQ_Rx_Lidar,Serial::RxIrq);
	//################   Initialisation Lidar   ################//

	// Coeur du code

	while(1){
	servo.pulsewidth_us(l);
	switch (valeur){
	case '0':		// Arrêt de la voiture
		v=0;		// Vitesse nulle
		Avant();	// On avance à la vitesse nulle (donc arrêt)
		break;

	case '1':		// Marche avant
		v=1;		// Vitesse max (variable pouvant prendre la valeur 0 ou 1)
		Avant();	// On avance à cette vitesse
		wait_ms(1000);	// Mais seulement pendant une seconde (sert aux
phases de tests de pilotage)
		valeur='0';	// Puis on envoie la commande "arrêt"
		break;

	case '2':		// Marche arrière
		v=1;		// Vitesse max
		Arriere();	// On part en marche arrière
		wait_ms(500);	// On attend un demie-seconde
```

```
                valeur='0';      // On s'arrête (et ce sera pareil jusqu'en 6)
                break;

        case '3':                // Roues directionnelles à gauche
                Gauche();        // Gestion du servomoteur
                valeur='0';
                break;

        case '4':                // Roues directionnelles à droite
                Droite();        // Gestion du servomoteur
                valeur='0';
                break;

        case '5':                // Roues directionnelles tout droit
                Milieu();        // Gestion du servomoteur
                valeur='0';
                break;

        case '6':                // Faire avancer la voiture avec une nouvelle vitesse (pour des
tests)
                mot.pulsewidth_us(1600); // on change le cycle du PWM (ici on accélère)
                wait_ms(a);      // Durée de l'accélération
                valeur='0';
                break;

        case '7':                // Mode éviter les obstacles
                v=1;
                Avant();         // On commence par avancer en permanence
                findMinDist();   // On cherche l'obstacle le plus proche dans le champ de vision
                if((__minDist<400)&&((__angleDistMin<45)||(__angleDistMin>315))){   // S'il
est dans cette zone alors il y a un mur
                Milieu();        // On met les roue droite
                Arriere();       // Et on fait machine arrière
                wait_ms(1000);   // Pendant une seconde
                }
                else{            // Sinon c'est qu'il n'y a pas d'obstacles devant, donc continue
tout en évitant les obstacles plus lointain sur les côtés
                Avant();         // En bref la voiture cherche à se placer le plus loin possible de
tout obstacle
                GoD(__angleDistMin);          // Elle ne va donc pas toujours tout droit comme
on peut le voir sur la vidéo
                }
                break;

        case '8':                // Manoeuvre de recul
                v = 1;           // Ici on enchaîne des instructions moteurs pour tourner alors
que la voiture est face à un mur
                servo.pulsewidth_us(1500);
```

```
                Arriere();
                wait_ms(400);
                v=0;
                Avant();
                servo.pulsewidth_us(1000);
                v=1;
                Avant();
                wait_ms(1000);
                valeur='0';
                Milieu();
                break;

        default:
                v=0;
                Avant();
        }
        }
}

void get_lidar(){
        /*
        Procédure permettant de récupérer la distance minimale détéctée par le lidar ainsi
que l'angle de détection
        Pas d'entrée ni de sortie
        */
        //findMaxDist();          // On n'a pu se servir que de Min Dist
        findMinDist();
}

void Avant(void){
        /*
        Procédure permettant de faire avancer la voiture, utilise la variable v
        Pas d'entrée ni de sortie
        */
        t=v*1620+(1-v)*1500; // Si v=0 la voiture s'arrete (PWM à 1500 signifie arrêt) si elle
est à 1 la voiture avance (PWM à 1620)
        mot.pulsewidth_us(t);
}

void Arriere(void){
        /*
        Procédure permettant de faire reculer la voiture, utilise la variable v
        Pas d'entrée ni de sortie
        */
        mot.pulsewidth_us(1500);      //arret
        wait_us(100000);              //attente de 3 période pour que le moteur comprenne la
nouvelle position
        mot.pulsewidth_us(1400);      //manip pour enclencher la marche arriere
```

```
        wait_us(100000);
        mot.pulsewidth_us(1500);
        wait_us(100000);
        t=v*1385+(1-v)*1500;            // On traite v de façon similaire à la fonction marche
avant
        mot.pulsewidth_us(t);
}

void BT_stop(void) {
        /*
        Fonction d'interruption pour le module bluetooth qui envoie les instructions données
par l'utilisateur au programme
        */
        valeur = BT.getc();             // Choix du mode
        BT.printf("valeur=%c\n\r",valeur);      // Retour écrit sur le téléphone pour confirmer la
commande
        //pc.printf("BT valeur=%c\r\n",valeur);
        return;
}


void Gauche(void){
        /*
        Procédure permettant d'orienter les roues vers la gauche
        Pas d'entrée ni de sortie
        */
        l=1000;
}

void Droite(void){
        /*
        Procédure permettant d'orienter les roues vers la droite
        Pas d'entrée ni de sortie
        */
        l=1500;
}

void Milieu(void){
        /*
        Procédure permettant d'orienter les roues au milieu
        Pas d'entrée ni de sortie
        */
        l=1285;
}


void findMinDist(void){
        /*
```

```
	Procédure du lidar permettant de trouver la distance minimum et son angle de
détection
	Pas d'entrée ni de sortie
	*/
	__minDist = 16000;
	__angleDistMin = 0;
	for(int ki = 0; ki < 360; ki++){
	if ((ki<140)||(ki>220)){
	if(__acq_lidar_1_old[ki] < __minDist){
		if(__acq_lidar_1_old[ki] > 0){
		__angleDistMin = ki;
		__minDist = __acq_lidar_1_old[ki];
		}
	}
	}
	}
}

void IRQ_dataScan(void){
	/*
	Procédure de dialogue avec le Lidar
	Pas d'entrée ni de sortie
	*/
	char i=0;
	int k = 0;
	unsigned char valeur_recue;
	unsigned int angle;
	unsigned int distance;
	static int trame_ok;

	valeur_recue=__lidar.getc(); //Response

	if (i==0) {
	if ((((valeur_recue&0X03)==0X01) || ((valeur_recue&0X03)==0X02))) {
	__data_tab[i] = valeur_recue;
	trame_ok=1;
	i++;
	} else {
	trame_ok=0;
	i++;
	}
	} else {
	__data_tab[i]=valeur_recue;
	i++;
	if ( (i==1) && ((valeur_recue&0x01) ==  0) ) {
	trame_ok = 0;
	}
	}
```

```
        if(i==5) {
        i=0;
        if (trame_ok==1) {
        angle = ((__data_tab[1] >> 7) + ((unsigned int)__data_tab[2] << 1));
        if(__acq_lidar_1[angle] == 0){
                __somme++;
                __acq_lidar_1[angle] = (__data_tab[3]>>2)+((unsigned int)__data_tab[4]<<6);
        }
        if(__somme == 360){
                __somme = 0;
                for(k = 0; k < 360; k++){
                __acq_lidar_1_old[k] = __acq_lidar_1[k];
                __acq_lidar_1[k] = 0;
                }
        }
        trame_ok=0;
        }
        }
}

void IRQ_Rx_Lidar(void)
        /*
        Procédure de dialogue avec le lidar
        Pas d'entrée ni de sortie
        */
{
        static char i=0;
        static int angle_old=0;
        unsigned char valeur_recue;
        unsigned int angle;
        unsigned int distance;
        static int trame_ok;

        valeur_recue = __lidar.getc(); //Response



        if (i==0) {
        if ((((valeur_recue&0X03)==0X01) || ((valeur_recue&0X03)==0X02)) ) {
        tab[i]=valeur_recue;
        trame_ok=1;
        i++;
        } else {
        trame_ok=0;
        }
        } else {
        tab[i]=valeur_recue;
        i++;
```

```
        if ( (i==1) && (valeur_recue&0x01 ==  0) ) {
        trame_ok = 0;
        }
        }
        if(i==5) {
        i=0;
        if (trame_ok==1) {
        angle=((tab[1] >> 7) + ((unsigned int)tab[2] << 1));
        distance= (tab[3]>>2)+((unsigned int)tab[4]<<6);
        if((angle>=0)&&(angle<360)) {
                if (angle < (angle_old -180)) {
                num_tab_acq = ((num_tab_acq)%2) +1;
                flag_chgt_tableau = 1;
                tour_ok = flag_chgt_tableau;
                }
                angle_old = angle;

                if (num_tab_acq == 1) {
                __acq_lidar_1[angle]=distance;

                } else __acq_lidar_1_old[angle] = distance;

        }
        trame_ok=0;
        }
        }


}

void GoD(int angle){
        /*
        Fonction permettant d'éloigner la voiture au maximum de tout obstacle
        L'angle d'entrée correspond à l'angle de l'ostacle le plus proche renvoyé par le Lidar
        */
        if (angle<70){  // L'obstacle le plus proche est à droite
        l=1050;         // Donc on tourne à gauche

        }
        if(angle>290){          // L'obstacle le plus proche est à gauche
        l=1550;         // Donc on tourne à droite
        }
        if((angle<290)&&(angle>70)){// L'obstacle le plus proche est suffisamenet derrière
        l=1300;                 // On continue tout droit

        }
}
```

```
bool DetectionD(void){ // inutilisée
	/*
	Procédure permettant de détecter un obstacle trop proche du capteur IR droit
	Pas d'entrée ni de sortie
	*/
	mes=capD.read_u16();
	if ( mes > seuil_capteur_IR ){
	return true;
	}
	else{
	return false;
	}
}

bool DetectionG(void){  // Inutilisée
	/*
	Procédure permettant de détecter un obstacle trop proche du capteur IR droit
	Pas d'entrée ni de sortie
	*/
	if (capG.read_u16()>seuil_capteur_IR){
	return true;
	}
	else{
	return false;
	}
}
```