Alexandra Carrez Ambre Visive Tristan Stevens Sarah Nyeki

LUMIDIOTE©

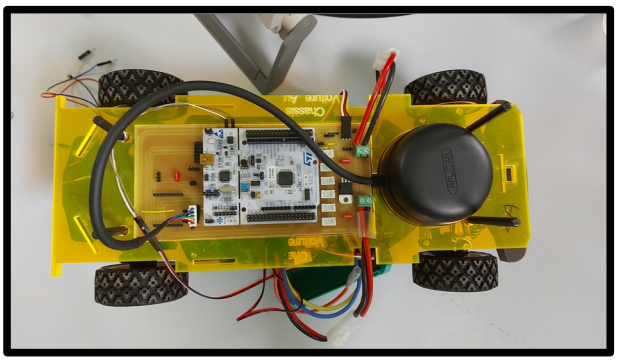
produit par LUMIGANG®

RAPPORT TECHNIQUE ProTIS

Mars-Avril 2021

Présentation

Lumidiote est une voiture à taille réduite autonome. Son but est d'avancer tout en évitant les obstacles se trouvant sur son chemin sans contrôle extérieur.



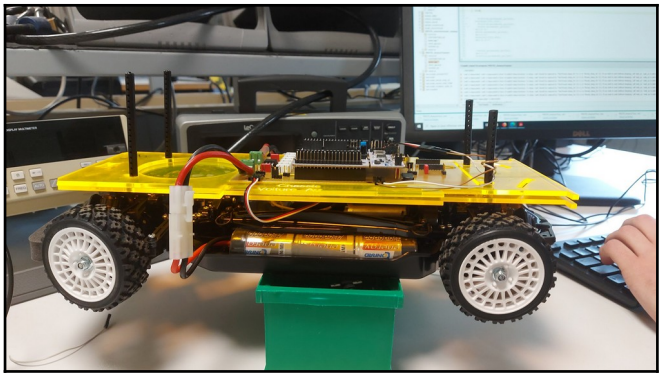
Sommaire

Introduction

Pendant ce projet, nous nous étions donné comme but de rendre autonome une voiture. Nous souhaitions qu'à terme elle puisse traverser un couloir et le foyer de l'école. Pour cela, il nous fallait comprendre d'un côté la mécanique de la voiture : comment la faire avancer / reculer, comment la faire tourner, puis d'un autre côté comment contrôler son déplacement en fonction de l'environnement qui l'entoure, en soi comment lui faire éviter des obstacles. Pour cette deuxième partie, nous avions donc le choix entre utiliser un radar infrarouge ou un Lidar tournant. Il nous a été conseillé d'utiliser le second, car plus pratique.

Partie I : Présentation et fonctionnement global

I.A - Pièces composant la voiture



La voiture se compose d'une partie motorisation électrique et direction, puis d'une seconde partie qui gère l'autonomie du déplacement. Le chassis **(Tamiya Lancia Delta)** basé sur un châssis de voiture radio-télécommandé, comporte les éléments suivants :

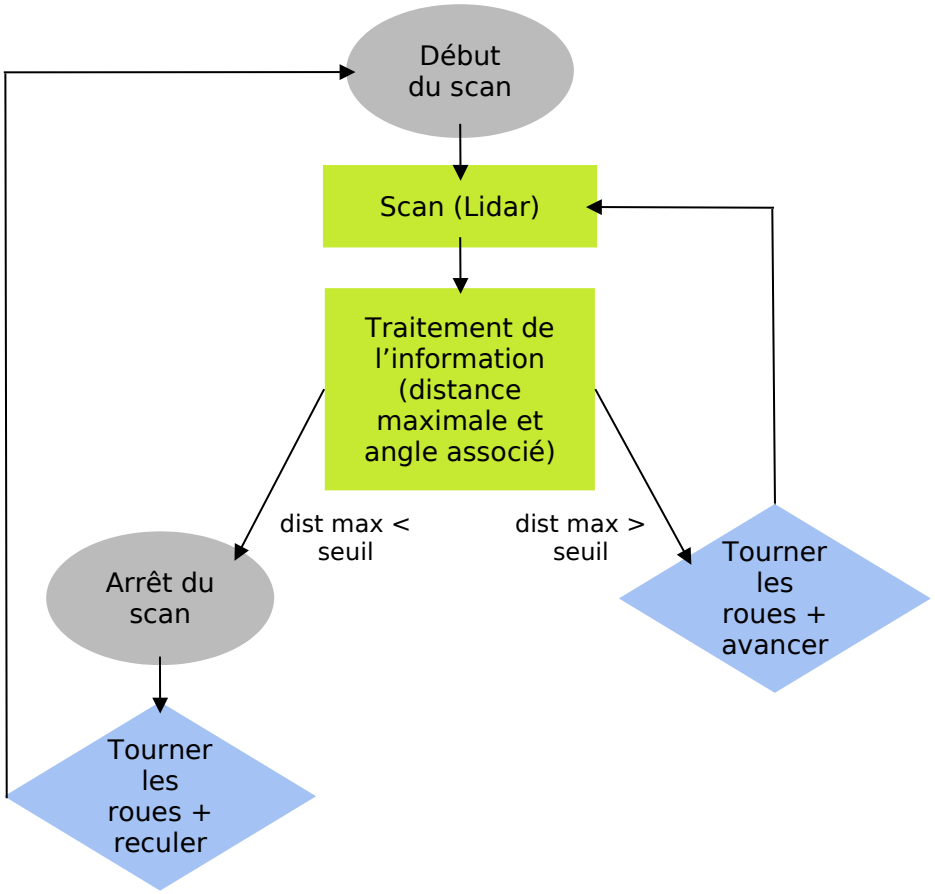
- Un moteur à courant continu et d'un système de contrôle spécifique au modélisme, permettant le déplacement du véhicule.
- Un servomoteur permettant la direction.
- Un batterie NiMH – 7.2V / 3000 ou 4000 mAh.
- Un ESC (Electronic Speed Controler) assurant la répartition de l'énergie provenant de la batterie vers les différents composants.

Pour la partie autonome, nous avons :

- Une carte Nucleo afin de communiquer depuis un ordinateur avec la voiture et lui envoyer des instructions et commandes.
- Un Lidar **RPLidar** A2M8, qui permet de faire un scan jusqu'à 360°. A sa vitesse de rotation la plus haute, il peut renseigner jusqu'à 8000 échantillons de distance (jusqu'à 12 m) et d'angle par seconde. Sa fréquence de rotation typique est de 10 Hz, soit 600 rpm, sa résolution est alors de 0.45°. Sa plage de fonctionnement va de 5 Hz à 15 Hz.

(*Source : Lense* - http://lense.institutoptique.fr/projet-voiture-autonome/)

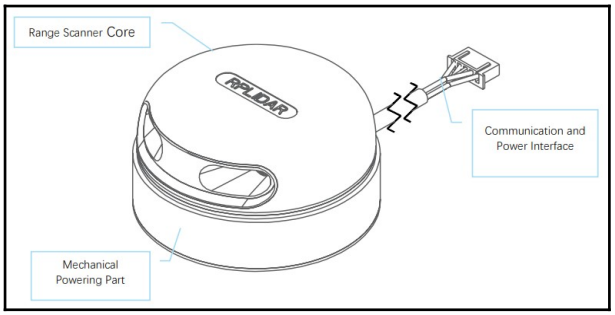
I.B - Fonctionnement global



Le but de la voiture autonome est d'avancer, sans point d'arrivée déterminé, en évitant les obstacles. Nous avons décidé, plutôt que de contourner les obstacles, c'est-à-dire d'aller dans une direction jusqu'à ce qu'un obstacle soit détecté et tourner à ce moment, que notre voiture ira dans la direction où il n'y a pas d'obstacles. Pour cela nous renseignons en temps continu la distance maximale sans obstacle détectée par le Lidar et tournons pour suivre la direction associée. Si la distance maximale est trop faible (la voiture est entourée d'obstacles), la voiture recule en tournant et va dans une nouvelle direction.

II - Partie information : Détection d'obstacles - LIDAR

II.A - Fonctionnement et initialisation du LIDAR



Le Lidar **RPLidar** A2M8 permet de scanner 360° (pas de 1°) et pour chaque angle de donner la distance à un éventuel obstacle (jusqu'à 12m). Pour communiquer avec le Lidar, on doit vérifier d'abord son bon fonctionnement via la vérification de plusieurs trames. D'abord, on envoie une requête au Lidar afin qu'il réponde une trame et en vérifiant cette trame on peut vérifier l'état de la connexion. De même, on vérifie l'état et les informations du Lidar lui-même. Ensuite, pour chaque valeur reçue, on a de nouveau une trame à vérifier afin de valider la mesure. Une mesure comprend un angle (entre 0° et 360°) et une distance en mm (facteur /4 avec la valeur réelle). Nous stockons les mesures dans un tableau, mis à jour en continu. Nous devons attendre que ce tableau soit complet (mesures pour tous les angles) lors de la mise en marche du système.

II.B - Détection d'obstacles

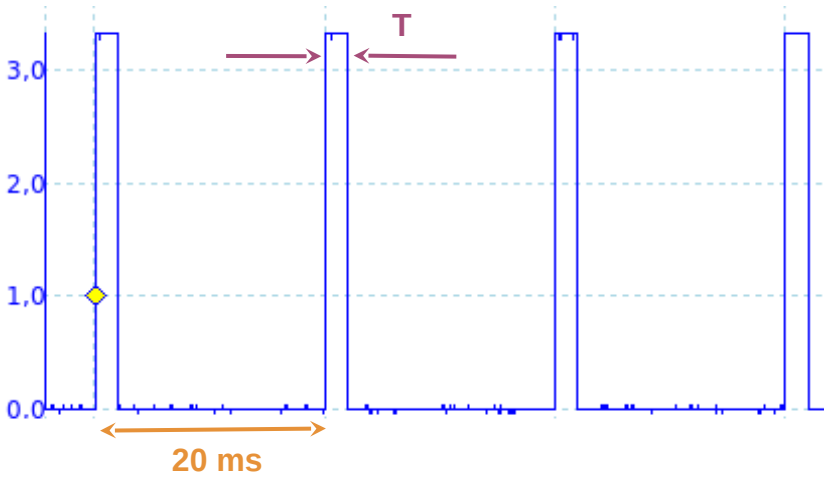
On a donc accès à un tableau de valeur de 360 cases (correspondant aux 360°). A partir de ce tableau, on réduit à l'angle de vision que l'on veut, soit [-20°,20°] dans notre cas donc on sélectionne les 40 première valeurs (nous avions vérifié que le 0° est en -20° sur notre voiture) et nous ajoutons un offset de -20°. On détermine la distance maximale et son angle associé et ce sera la direction envoyée pour le déplacement après comparaison à une distance seuil. La distance seuil, imposée à 10 cm, permet de ne pas avoir la voiture trop proche d'un obstacle si elle est entourée et d'ensuite reculer pour repartir dans une autre direction.

III - Partie mécanique : contrôle de la direction et de la vitesse - moteur et servomoteur

III.A - Contrôle du moteur et du servomoteur

Le servomoteur et le moteur sont contrôlés via une carte Nucléo et l'interface Mbed, en appliquant une tension créneau dont le rapport cyclique indique dans le premier cas le

sens de rotation et l'angle des roues, et pour le second la vitesse et le sens de marche (avant ou arrière). Le signal créneau en question devra être de période de 20 ms (soit 50 Hz) et de temps haut entre 1 et 2 ms. On le prendra d'amplitude 1 volt.



➔ **Commandes du moteur :**

- ★ T = 1,5 ms : la voiture est à l'arrêt.
- ★ T > 1,5 ms : la voiture avance et accélère de manière quasi-linéaire plus T se rapproche de 2 ms.
- ★ T < 1,5 ms : la voiture recule, comme précédemment en accélérant lorsque T se rapproche de 1 ms.

Nous avons remarqué en pratique que le T minimal pour commencer à avancer était de 1563 µs, et pour reculer de 1443 µs.

Dans le programme final, l'avance est toujours imposée pour T=1570 µs et le recul pour T = 1440 µs.

➔ **Commandes du servomoteur :**

- ★ T = 1,3 ms : les roues sont orientées parallèles à la voiture, celle-ci avancera alors tout droit.
- ★ T > 1,3 ms : rotation des roues entre 0° et ≃40°, soit rotation à droite.
- ★ T < 1.3 ms : rotation des roues entre 0° et ≃-40°, soit rotation à gauche.

Ici aussi en pratique nous avons remarqué un fonctionnement différent : les roues semblent atteindre leur angle maximal à droite pour T=1,5 ms, à gauche pour T=1,1 ms.

Dans la structure du code on commence par envoyer deux signaux rectangulaires de 20

ms sur les broches de la carte Nucléo correspondant au moteur et au servomoteur, initialisés avec les temps hauts suivant 1,5 ms et 1,3 ms (respectivement). Ensuite, à l'aide de boucles infinies et de wait on choisit la vitesse, la direction, et le durée appliquée…

```
 */

#include "mbed.h"

// Définition des entrées/sorties

PwmOut direction(PB_13);    // Servomoteur (direction des roues)
PwmOut moteur(PC_9);        // Motorisation / ESC

void Accelerer(int v_init, int v_fin);

int main() {

    direction.period_ms(20);         // Initialisation de la période de la tension rectangulaire envoyée au servomoteur
    direction.pulsewidth_us(1300);   // Initialisation en position à 0°. Durée d'un pulse pour aller tout droit = 1300µs
    moteur.period_ms(20);            // Initialisation de période la tension rectangulaire envoyée au moteur
    moteur.pulsewidth_us(1500);      // Initialisation en position où la voiture est à l'arrêt. Durée d'un pulse = 1500µs

    while(1){
        direction.pulsewidth_us(1500); // La voiture tourne ses roues vers la droite
        wait(2.0);
        moteur.pulsewidth_us(1570);    // La voiture avance tout droit
        wait(5.0);                     // pendant 5 secondes
        direction.pulsewidth_us(1100); // La voiture tourne à gauche
        wait(0.5);
        moteur.pulsewidth_us(1500);    // La voiture s'arrête
        wait(0.5);


       // Pour reculer : (petite particularité du moteur)
       moteur.pulsewidth_us(1443);    // La voiture devrai reculer, mais ne le fait pas
       wait(0.5);
       moteur.pulsewidth_us(1500);    // La voiture s'arrête
       wait(0.5).                     // pendant une demi-seconde
       moteur.pulsewidth_us(1300);    // La voiture recule cette fois
       wait(5.0)                      // pendant 5 secondes
```

Nous avons utilisé des fonctions wait de durées différentes pour vérifier que la voiture respectait bien nos instructions. En testant, notre code et différentes commandes, nous nous sommes rendus compte que nos instructions pour reculer ne fonctionnaient pas toujours. Si notre toute première instruction était de reculer, ça fonctionnait. Par contre, si nous demandions plus tard dans le code de reculer après avoir avancé, ça ne fonctionnait plus. Nous avons trouvé une solution un peu fortuitement, en remarquant que ce problème se réglait en donnant une première instruction pour reculer (T=1,443 ms) qui ne fonctionnait pas, puis en retournant au point d'arrêt de la voiture (T= 1,5 ms). Après cela la nouvelle consigne pour reculer s'opérait cette fois (T=1,3 ms par exemple). Un professeur ayant déjà réalisé une voiture autonome avec le même ESC (Electronic Speed Controler), nous a rassuré en nous disant qu'il avait eu le même problème et qu'il était inhérent à l'ESC. Néanmoins, il en reste que reculer avec la voiture est un peu fastidieux.

Nous avons également testé une fonction qui nous permettait d'accélérer et décélérer, même si nous ne l'avons pas utilisé dans la suite avec le Lidar. En voici, le code du *main* :

```
#include "mbed.h"

// Définition des entrées/sorties

PwmOut direction(PB_13);     // Servomoteur (direction des roues)
PwmOut moteur(PC_9);         // Motorisation / ESC

void Accelerer(int v_init, int v_fin); // Appel de la fonction

int main() {
    direction.period_ms(20);          // Initialisation de la période de la tension rectangulaire envoyée au servomoteur
    direction.pulsewidth_us(1300);    // Initialisation en position à 0°. Durée d'un pulse pour aller tout droit = 1300µs
    moteur.period_ms(20);             // Initialisation de période la tension rectangulaire envoyée au moteur
    moteur.pulsewidth_us(1500);       // Initialisation en position où la voiture est à l'arrêt. Durée d'un pulse = 1500µs

    while(1){

        Accelerer(1500,1750);         // La voiture va acccélérer jusqu'à la vitesse correspondant à T=1750 µs
        wait(0.5);                    // Elle reste à la vitsse finale pendant 0.5 seconde
        Accelerer(1750,1500);         // La voiture décélère jusqu'à son arrêt (T=1500µs)
        wait(5.0);
```

Le code de la fonction *void* :

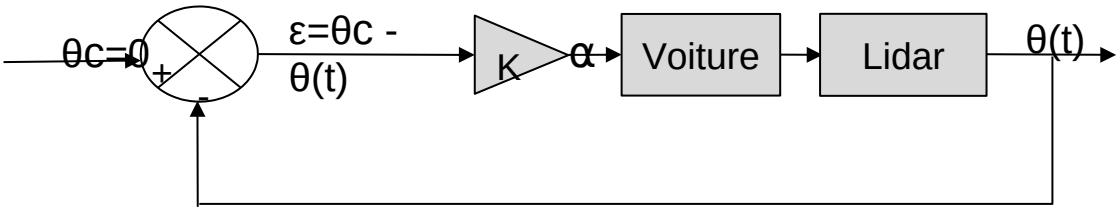
```
void Accelerer(int v_init, int v_fin){
    /*
    Auteurs: Sarah et Tristan
    Date: 18/03/2021
    Fonction: Permet d'atteindre une vitesse finale à partir d'une vitesse initiale progressivement via une accélération ou une décélération
    Entrées:
    v_init: durée d'impulsion correspondant à la vitesse initiale; valeur entre 1000 et 2000 (format int)
    v_fin: durée d'impulsion correspondant à la vitesse finale; valeur entre 1000 et 2000 (format int)
    */
    int i; // vitesse intermédiaire entre v_init et v_fin
    if(v_init < v_fin)                    // accélération si la vitesse initiale est plus petite que la finale demandée
        {
        for(i=v_init; i<(v_fin +1); i+=50)  // où 50µs est le pas entre chaque vitesse intermédiaire
            {
            moteur.pulsewidth_us(i);
            wait(0.5);
            }
        }
    else                                  // décélération si la vitesse initiale est plus grande que celle finale demandée
        {
        for(i=v_init; i>(v_fin -1); i-=50)  // par pas de 50 µs on diminue la durée du pulse
            {
            moteur.pulsewidth_us(i);
            wait(0.5);
            }
        }
    }
```

III.B - Asservissement de la direction

Passée la partie où nous avons compris comment on pouvait contrôler la direction et la vitesse de la voiture, nous voulions maintenant être capables d'orienter la voiture dans un direction précise, choisie grâce aux informations du Lidar. Celui-ci devait nous donner l'angle θ_{dmax}correspondant à la distance maximale entre un obstacle et la voiture. Notre travail était de traduire cet angle en instruction et de l'envoyer à la carte Nucléo qui commande le moteur et le servomoteur. Mais nous nous sommes vite rendus compte que nous ne pouvions pas trouver une relation mathématique fiable qui donnerait l'angle de rotation des roues qu'il faudrait appliquer en fonction de θ_{dmax}(et donc le temps haut du signal rectangulaire), ou la calculer serait trop complexe. Ainsi, nous nous sommes orientés vers un asservissement de la direction de la voiture, et donc de la rotation de ses roues.

Le schéma-bloc suivant illustre la boucle asservie. Les angles donnent la direction vers la distance maximale dans le référentiel de la voiture. L'angle de consigne $\theta_c = 0$ exprime de la volonté d'orienter la voiture dans la direction de l'obstacle le plus éloigné. Cette valeur de

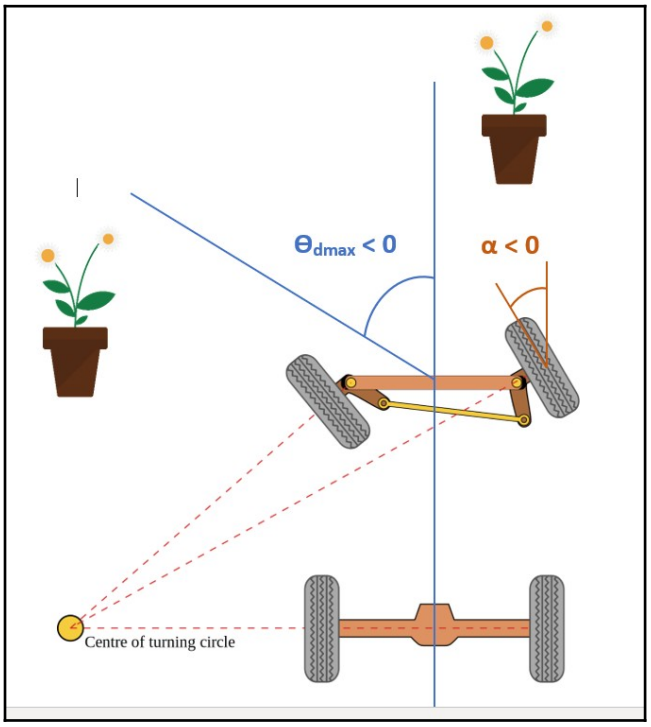
consigne est comparée avec la valeur de l'angle réel, puis l'erreur ainsi obtenue est multipliée par un gain K (constituant le bloc correcteur) pour déterminer l'angle α à prendre par les roues. La variable $\alpha(t)$, à son tour, influence l'angle $\theta(t)$ à la sortie du système (la voiture). Enfin, le Lidar effectue une mesure de $\theta(t)$.



$$\alpha = -K\,[\theta c - \theta(t)]$$

La valeur de K est déterminée expérimentalement. Elle est dans le code fixée à 20000. Une valeur trop petite entraînerait trop peu de correction dans la direction à prendre à la voiture, donc un temps de virage trop grand. A l'inverse, une trop grande valeur pourrait entraîner trop de correction, donc impliquer une nouvelle erreur. Cependant, cette configuration est moins gênante en pratique, et un premier modèle dans lequel les roues seraient constamment braquées à fond pourrait fonctionner.

Ci-dessous un schéma de la voiture et des différents angles qui interviennent dans l'asservissement : θ_{dmax} = $\theta(t)$ et évolue dans le temps à mesure que la voiture avance et tourne.



En suivant, voici le code de l'asservissement que nous avions prévu avant de fusionner la partie maitrise de la direction et la partie Lidar :

```
/*
PROGRAMME D'ASSERVISSEMENT DE LA DIRECTION DU VEHICULE
Date: 22/03/2021
Auteurs: Sarah et Tristan
*/




#include "mbed.h"

PwmOut direction(PB_13);     // Servomoteur de direction
PwmOut moteur(PC_9);         // Motorisation / ESC

double AngleServo(double theta_voulu, double d0);

int main (){
    double theta_voulu;      // direction de la distance max, vers laquelle on veut tendre
    double d0;               // distance avec l'obstacle en face
    double alpha;            // angle du servomoteur

    while(1){
        alpha=AngleServo(theta_voulu, d0);
        direction.pulsewidth_us(alpha);
    }

}
```

```
double AngleServo(double theta_voulu, double d0){
/*
ENTREES:
        -   theta_voulu : nombre entier, angle en degré, donnant la direction voulue du véhicule.
                Il faut que l'angle de référence (dirigé vers l'avant de la voiture) vaille 0°.
        -   d0 : double réel, distance entre le véhicule et l'obstacle en face [mm].
                En première approximation, on peut ne pas utiliser d0.
SORTIE:     alpha : entier réel, largeur du pulse en µm contrôlant l'orientation des roues.
               alpha=1300 correspond à 0°
               alpha<1300 correspond à une routation vers la gauche
               alpha>1300 correspond à une routation vers la droite
               il faut 1050<alpha<1500
*/
    double K=2000; // theta_voulu<50°, d0>500mm. Valeur prise abitrairement qui pourra être affiné.
    double alpha = 1300 + K*theta_voulu/d0;

}
```

IV - Résultats / Critique et amélioration / Retour sur le projet

IV.A - Résultats et critiques

Comme voulu, la voiture est capable d'avancer tout en évitant la majorité des obstacles. Elle est également capable, pour des obstacles trop proches, de reculer pour prendre une autre direction, puis de continuer sa course. Le véhicule avance à une vitesse de l'ordre de 1 à 2 m/s. Lorsque nécessaire, il recule à 1 m/s sur 0,5 m à 1 m.

Cependant, pour les obstacles trop proches, elle ne s'arrête pas à temps et percute donc l'obstacle avant de reculer. Ce défaut, en plus de ne pas respecter le cahier des charges, risque de causer des dommages au véhicule.

Le bon fonctionnement du véhicule est également limité par la complexité de son environnement. Lorsqu'il évolue dans un couloir, dont la direction à privilégier est aisément déterminable, son comportement est correct. A l'inverse, dans une milieu complexe, dont plusieurs directions peuvent être privilégiées, le logiciel peut fréquemment changer l'angle $\theta(t)$, ce qui résulte en une conduite en "zig zag".

Enfin, un obstacle fin est mal interprété par la voiture. En effet, le code lui impose de s'arrêter et reculer lorsque la distance maximale avec un obstacle est inférieure à 10 cm. Cependant, si la voiture est bloquée par un obstacle prenant la moitié de son champ de vue mais que l'autre moitié lui permet de détecter des obstacles plus éloignés que 10 cm, elle continue d'essayer d'avancer. Dans ce cas de figure, les obstacles sont mal détectés et non évités.

IV.B - Améliorations

Pour remédier à ces défauts, plusieurs axes d'amélioration sont possibles:

- Une définition de la distance d_{min} qui, de même que d_{max} donne la distance avec l'obstacle le plus éloigné donnerait ici la distance avec l'obstacle le plus proche. Le critère de recul serait alors basé sur cette valeur, afin d'éviter le dernier défaut cité.
- Une révision de la valeur seuil de cette distance (plus élevée) afin d'éviter d'impacter un obstacle

Enfin d'autres améliorations pourraient voir le jour pour améliorer les performances du système, telles que :

- Une augmentation de la vitesse lorsque d_{min} est suffisamment élevée.
- Une prise en compte de cette vitesse, mais aussi de d_{max} dans le calcul de l'angle α de la direction des roues.

IV.C - Retour sur le projet

Ce projet, nous a permis d'un côté de nous familiariser avec la commande à distance via ordinateur du servomoteur et du moteur, puis lorsque nous avons voulu maîtriser la direction de la voiture à la mise en pratique de la notion d'asservissement vu en cours d'Automatique. D'un autre côté, la récupération des données d'angles et de distances depuis le Lidar, ainsi que sa maîtrise, nous ont amené à revoir des éléments de programmation en langage C. Finalement, nous nous sommes aussi confrontés aux difficultés de la phase de tests

et corrections suivant la phase de conception : comprendre les erreurs et les bugs, et trouver des solutions. Même après plusieurs essais et améliorations apportées au code global dirigeant la voiture et le lidar, des bugs résident et de nouvelles améliorations peuvent être ajoutées. Nous aurions aimé avoir plus de temps pour arriver à stade où on aurait réduit une majorité des erreurs décrites dans les paragraphes précédents. Cependant, pour certaines erreurs, nous sommes limités par le matériel, par exemple le Lidar a très peu de chances de détecter une grille devant lui, or celle-ci ne permet à la voiture ni d'avancer, ni de reculer. Ainsi, la voiture se bloque toute seule.

En dépit des bugs et des améliorations possibles, nous avons tout de même réussi à réaliser une voiture relativement autonome qui s'adapte (selon ses bons désirs) à l'environnement qui l'entoure. Avec quelques semaines d'éducation en plus, elle aurait pu devenir plus obéissante (et époustouflante) !

ANNEXE : Code complet de la voiture (Lidar + asservissement de la direction)

```
 /*
Récupérer les données du Lidar, trouve l'angle correspondant à la distance max
Déplace les roues pour aller dans cette direction
Pas de recherche du min, de ralentissement ni de la distance à 0

NB: Lorsqu'il ne trouve pas de dist max, renvoie un angle de -20°

PROJET PROTIS : Voiture autonome
Alexandra CARREZ, Ambre VISIVE, Tristan STEVENS, Sarah NYEKI

ETAT: LIDAR OK
TEST : OK

MAJ : 20/05/2021
*/

// inclusion des bibliothèques
#include "mbed.h"
#include "rplidar.h"

// défintion des constantes globales
#define BLINKING_RATE_MS        500
#define NB_DATA_MAX             20
#define AFF_DATA                0

// définition des variables
char            pc_debug_data[128];
char            received_data[64];
int             data_nb = 0;
int             data_scan_nb = 0;
char            mode = LIDAR_MODE_STOP;
char            scan_ok = 0;
int             distance_scan[360] = {0};
int             distance_scan_old[360] = {0};
int             distance_scan_old40[40] = {0};
char            tour_ok = 0;
char            trame_ok = 0;
double          K=20000;        // theta_voulu<50°, d0>500mm. Cette valeur est à tester et à affiner

// définition des ports
UnbufferedSerial    pc(USBTX, USBRX, 115200);
DigitalOut          led(LED1);
DigitalOut          debug_data(D10);
DigitalOut          debug_tour(D9);
DigitalOut          debug_out(D7);
DigitalOut          data_ok(D5);
DigitalOut          data_ok_q(D4);

UnbufferedSerial    lidar(A0, A1, 115200);
PwmOut              rotation(PB_9);

PwmOut direction(PB_13);    // Servomoteur de direction
PwmOut moteur(PC_9);        // Motorisation / ESC

struct lidar_data   ld_current;
```

```
/** MAIN FUNCTION **/
int main()
{
    led=1; // indicateur visuel de l'initialisation
    moteur.pulsewidth_us(1500);  //définition de T pour le moteur en µs
    direction.pulsewidth_us(1300); //définition de T pour le servomoteur en µs
    int nb_tour = 0;
    double theta_voulu;     // direction de la distance max, vers laquelle on veut tendre
    double d0=1000;         // distance avec l'obstacle en face (mm)    [default value]
    double alpha;           // angle du servomoteur
//    bool b=true;
    wait_s(3.0);
    rotation.period(1/25000.0);    // default : 1/25000.0
    rotation.write(0.4);    // default : 0.4
    wait_s(2.0);
    pc.write("\r\nLIDAR Testing\r\n", sizeof("\r\nLIDAR Testing\r\n"));
    lidar.attach(&IT_lidar);
    wait_s(1.0);
    pc.write("\r\nLIDAR OK\r\n", sizeof("\r\nLIDAR OK\r\n"));
    getHealthLidar();
    wait_s(1.0);
    pc.write("\r\nLIDAR Info\r\n", sizeof("\r\nLIDAR Info\r\n"));
    getInfoLidar();
    wait_s(1.0);
    pc.write("\r\nLIDAR Testing Info\r\n", sizeof("\r\nLIDAR Testing Info\r\n"));
    getSampleRate();
    // Start a new scan
    startScan();
    // led=0;
    // moteur.pulsewidth_us(1570);
    // Infinite Loop
    while (true) {

        if(trame_ok){
            debug_tour = !debug_tour;
        }

        if(tour_ok == 4){
            led=0;
            int maxDistance, maxAngle;
            tour_ok = 0;
            for (int compteur=0; compteur <40; compteur ++){
                distance_scan_old40[compteur]=distance_scan_old[compteur];}
    findMax(distance_scan_old40, 40, &maxDistance, &maxAngle);
    theta_voulu=maxAngle-20;
    print_int("A", theta_voulu);
    if (maxDistance>100/4)              // Cas où il n'y a pas d'obstacle proche, la voiture se dirige vers la distance max
    {
        moteur.pulsewidth_us(1570);           // instruction pour que la voiture avance
        alpha = 1300 + K*theta_voulu/d0;      // angle des roues = temps haut (en microsecondes)du signal créneau
        direction.pulsewidth_us(alpha);       // rotation des roues en fonction de l'angle correspondant à la distance max
    }
    else                                // Sinon, ça signifie qu'elle a un obstacle devant elle et elle doit reculer
    {
        moteur.pulsewidth_us(1500);           // Arrêt de la voiture
        direction.pulsewidth_us(1300);        // On replace les roues à 0° (parallèles à la voiture)
        moteur.pulsewidth_us(1440);           // Initialisation du recul (la voiture ne recule pas)
        stopScan();
        wait_s(0.5);
        moteur.pulsewidth_us(1500);           // Retour à l'arret
        wait_s(0.5);
        direction.pulsewidth_us(1500);        // Les roues se braquent
        moteur.pulsewidth_us(1440);           // La voiture recule lentement
        wait_s(1.0);                          // Temps du recul
        startScan();
        moteur.pulsewidth_us(1500);           // Nouvel arrêt
        direction.pulsewidth_us(1300);        // On replace les roues à 0°
                                              //  -> à partir de là, la voiture peut repartir (on recommence la boucle While)
    }
}
/*
stopScan();
tour_ok = 0;
print_int("NB ", data_scan_nb);
// affichage données
if(AFF_DATA){
    for(int k = 0; k < 360; k++){
        if(distance_scan_old[k] != 0){
            sprintf(pc_debug_data, "\t%d = %d", k, distance_scan_old[k]);
            pc.write(pc_debug_data, strlen(pc_debug_data));
        }
    }
}
```

```
        ,
        getHealthLidar();
        wait_s(0.001);
        startScan();
        */
    }
}
```