

RAPPORT TECHNIQUE

– VOITURE AUTONOME SAM –

BIVAS Elsa | DE BEAUREGARD Alban | TAN Stella | UTHAYAKUMAR Jashaani



« Nous attestons que ce travail est original, que nous citons en référence toutes les sources utilisées et qu'il ne comporte pas de plagiat. »

TABLES DES MATIÈRES

	Page
I. Comprendre le projet.....	3
1.1.Introduction.....	3
1.2. Contraintes et performances	4
1.3. Découpage fonctionnel	4
II. Réaliser le prototype.....	5
2.1. Détection des obstacles avec les capteurs de distance SHARP (niveau 1)..	5
2.1.1. Partie électronique.....	5
2.1.2. Partie algorithmique.....	7
2.2. Détection des obstacles avec le RPLIDAR (niveau 2).....	11
2.2.1. Partie électronique.....	11
2.2.2. Partie algorithmique.....	12
III. Valider et caractériser le système final.....	14
3.1. Test et validation.....	14
3.2. Résultat final.....	17
IV. Comprendre les étapes de réalisation.....	18
4.1. Planning des séances.....	18
4.2. Difficultés rencontrées et analyse du travail d'équipe.....	19
V. Conclusion et développements possibles.....	20
ANNEXES.....	21
ANNEXE 1 : Code pour les capteurs de distance Sharp.....	21
ANNEXE 2 : Code pour le Lidar.....	25
ANNEXE 3 : Vidéo de démonstration.....	30

I. Comprendre le projet

1.1. Introduction

Les voitures autonomes sont un secteur en plein essor – en témoigne la filiale Waymo d'Alphabet dédiée entièrement à développer cette technologie. En effet, ces voitures pourraient révolutionner le monde des transports en proposant aux utilisateurs de pouvoir effectuer d'autres tâches au cours d'un trajet en voiture (lire, discuter, jouer) ou encore et surtout proposer une solution potentielle quant à la sécurité routière.

Notre projet consiste à faire rouler de manière autonome une voiture miniature, de sorte à ce qu'elle se déplace en ligne droite et qu'elle évite tout obstacle qui se trouve dans sa trajectoire en changeant de direction à l'aide de capteurs situés à l'avant du véhicule.

Nous disposons du matériel suivant¹:

- **Châssis Tamiya Lancia Delta** composé de :
 - 1 moteur à courant continu (MCC) et d'un système de contrôle spécifique au modélisme, permettant le déplacement du véhicule
 - 1 servomoteur permettant la direction
 - 1 batterie NiMH – 7.2V / 3000 ou 4000 mAh
- **Carte électronique Voiture Autonome**
 - 3 Capteurs de distance SHARP (10-80 cm et 20-150 cm) (Fig. 1)
 - 1 RPLIDAR A2M8 (Fig. 2)

Mots-clés : Détection d'obstacles, Pilotage autonome, Capteurs de distance, Lidar

Nous avons défini plusieurs objectifs à atteindre par la voiture :

- **Se déplacer** en ligne droite
- **Détecter** la présence d'un obstacle
- **Éviter** les obstacles en tournant ou en effectuant une marche arrière
- **Avertir** l'utilisateur lors de la détection d'un obstacle
- **Avertir** l'utilisateur lors d'un changement de direction

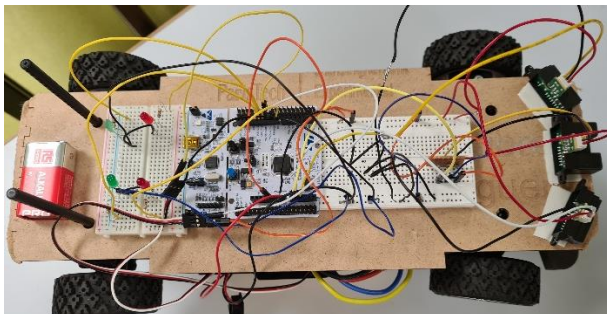


Figure 1 – Voiture munie de 3 capteurs SHARP

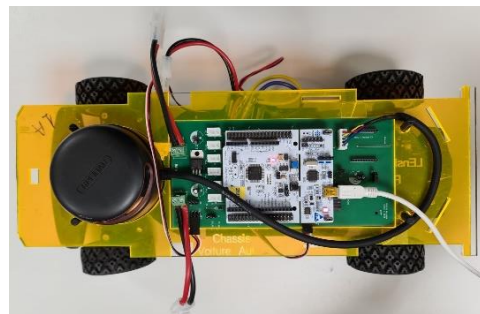


Figure 2 – Voiture munie d'un RPLIDAR

¹ <http://lense.institutoptique.fr/projet-voiture-autonome/>

1.2. Contraintes et performances

Nous avons également défini un cahier des charges :

- Se déplacer en ligne droite à vitesse constante
- Détecter un obstacle à l'aide de 3 capteurs de distance SHARP en avant de la voiture : 1 central (obstacle devant la voiture) et 2 latéraux (obstacles à gauche et à droite de la voiture) orienté à environ 45° par rapport à celui au centre
- Éviter les obstacles en ralentissant de manière quasi-linéaire
- Éviter un obstacle en tournant vers le côté où le prochain obstacle est le plus loin
- Avertir l'utilisateur de la détection d'un obstacle et de la direction de changement de direction à l'aide de LEDs
- Piloter la voiture avec une carte Nucléo sous Mbed

1.3. Découpage fonctionnel

Notre travail s'est découpé en 2 grandes parties :

1. Acquérir et traiter les données reçues par les différents capteurs pour détecter les obstacles (Fig. 3 et 4)
2. Communiquer les instructions nécessaires à la voiture en fonction des données transmises par les capteurs pour éviter les obstacles (Fig. 3 et 4)

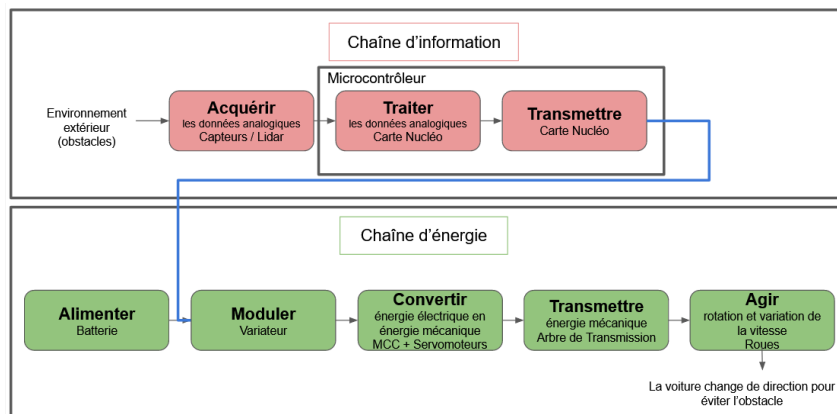


Figure 3 – Schéma fonctionnel 1

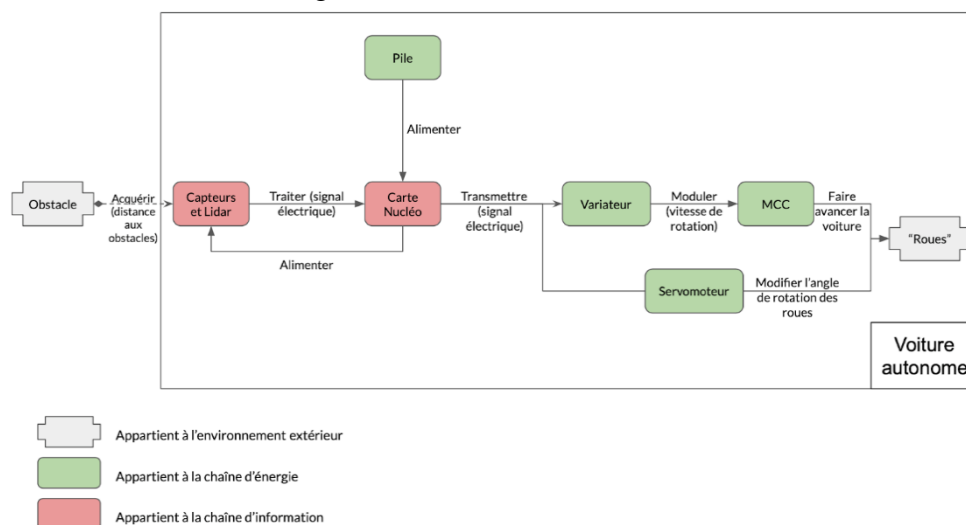


Figure 4 – Schéma fonctionnel 2

II. Réaliser le prototype

Nous avons dans premier temps travaillé avec les capteurs de distance SHARP. Puis, lorsqu'il ne restait plus qu'à faire de légers ajustements et améliorations, nous avons réalisé une voiture autonome avec un Lidar.

2.1. Détection des obstacles avec les capteurs de distance SHARP (niveau 1)

2.1.1. Partie électronique

Alimentation

Afin d'avoir une voiture autonome, il faut l'alimenter. Nous utilisons pour cela deux piles, l'une de 12V qui sert à alimenter le moteur, et une pile de 9V qui sert à alimenter la carte Nucléo. Pour les autres composants actifs, c'est-à-dire les capteurs, le servomoteur, les LEDs, nous nous sommes servis de la carte Nucléo pour les alimenter en 5V.

Détection

Nous avons utilisé deux capteurs de distance SHARP. Le capteur central, qui permet de mesurer la distance à un obstacle qui se trouve en face, est capable de détecter à une distance de 20 cm à 50 cm. De même, deux capteurs se trouvent sur les côtés gauche et droit à l'avant de la voiture, orientés à 45° par rapport au capteur central et ils peuvent détecter la présence d'obstacles à une distance de 10 cm à 80cm. Ces capteurs nécessitent d'être alimentés aux alentours de 5V.

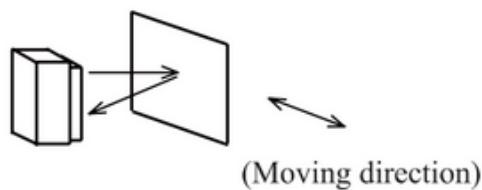


Figure 5 – Fonctionnement du capteur d'après la datasheet du capteur

Les capteurs évaluent la distance à un obstacle en déterminant le temps de retard du signal infrarouge ($\lambda = 850 \pm 50 \text{ nm}$) émis par une diode (InfraRed Emitting Diode) et le signal de retour reçu par un détecteur (Position Sensitive Detector) (Fig. 5).

Afin d'utiliser les capteurs et vérifier les données du constructeur, nous avons décidé de les caractériser afin de déterminer quelle tension correspondait à quelle distance (Fig. 6). Pour cela, nous nous sommes servis d'un générateur de tension pour alimenter les capteurs à 5V, et d'un oscilloscope pour lire la tension en sortie du pin commande du capteur.

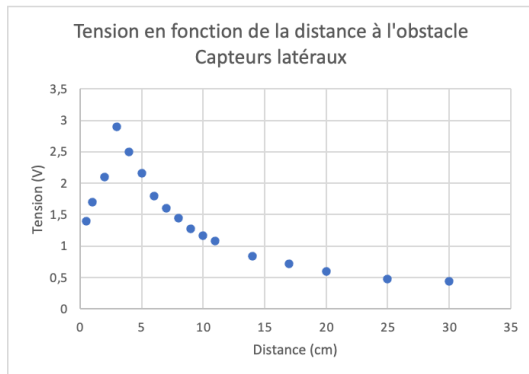


Figure 6.1 – Caractérisation expérimentale du capteur (10-80cm)

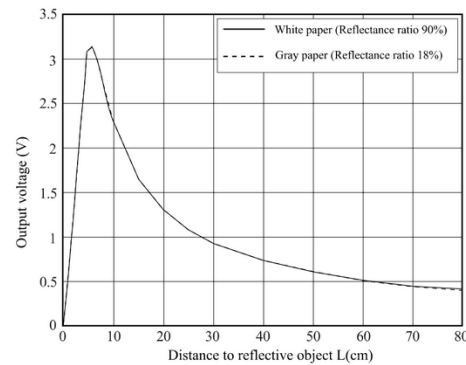


Figure 6.2 – Données fournies par la DataSheet pour un capteur (10-80cm)

Nous avons fait de même avec le capteur (20-150 cm). Nous avons obtenu une courbe similaire à celle du fabricant, validant ainsi les données du constructeur. Grâce à ces courbes, nous avons pu choisir des valeurs seuils de tension pour définir la distance à laquelle nous voulions que la voiture commence à ralentir, à tourner voire même à faire marche arrière.

Changement de trajectoire

Afin de faire tourner les roues et changer la trajectoire de la voiture, nous avons utilisé un servomoteur. Le servomoteur est un petit composant, constitué de roues d'engrenage, qui font tourner son "arbre de transmission" d'un angle précis en fonction de la tension moyenne qui lui est appliquée. Afin de contrôler le mouvement angulaire de celui-ci, il faut lui transmettre un signal modulé en largeur d'impulsions, appelé MLI (Modulation de la Largeur d'Impulsion) ou PWM (Pulse Width Modulation). Le principe du MLI consiste à faire varier la durée du temps haut du signal rectangulaire de commande, c'est-à-dire le rapport cyclique, ce qui va faire varier la valeur moyenne de ce signal. En fonction de cette valeur moyenne, l'angle de sortie du servomoteur va varier.

L'angle de sortie du servomoteur varie entre -90° et $+90^\circ$. Nous avons choisi un signal de période de 20 ms. En faisant quelques petites manipulations, nous avons déterminé les correspondances suivantes :

Temps haut (ms)	Position angulaire de la roue ($^\circ$)
1.2	0
0.1	90
2.5	-90

Cela nous a permis de choisir l'angle de braquage des roues.

Déplacement de la voiture

La partie la plus importante a été de faire rouler la voiture. Pour cela, nous avons un moteur à courant continu fourni avec le châssis Tamiya Lancia Delta, ainsi qu'un variateur qui gère les puissances, en fournissant au moteur un courant bien plus élevé que ce qui circule dans le reste du système. Comme le variateur était préprogrammé, nous n'avons pas besoin d'y toucher. Tout comme pour le servomoteur, le pilotage du moteur à courant continu se fait avec un signal modulé en largeur d'impulsions.

Pour déterminer le fonctionnement du moteur, nous nous sommes servis d'un générateur de tension. En faisant varier la durée du temps haut, nous avons déterminé que pour une période de 20 ms, un temps haut de 1,5 ms correspondait à la vitesse nulle. En augmentant ce temps haut, la vitesse de rotation en sortie du moteur augmente. De même, en diminuant la valeur de ce temps haut, on peut inverser le sens de rotation. Il a cependant fallu faire attention au fait que l'inversion de sens de rotation n'est pas continue. Pour procéder à la marche arrière, il faut descendre en dessous de 1,5 ms une fois, puis revenir à la vitesse nulle (i.e. 1,5 ms) puis descendre à nouveau. Cette manipulation est nécessaire pour l'inversion du sens de rotation. À la suite de toutes ces manipulations, nous savons à présent comment faire avancer la voiture et comment lui faire changer de direction. Nous sommes donc ensuite passés au câblage de l'ensemble des composants (Fig. 7).

Nous avons également utilisé 2 LEDs vertes pour indiquer lorsqu'une ou deux des valeurs seuils sont atteintes (voir explication dans la partie suivante) et 2 LEDs rouges qui indiquent dans quelle direction la voiture doit tourner pour éviter l'obstacle. Chacune des LEDs a été mis en série avec une résistance de protection $R_{LED} = 100 \Omega$ pour limiter le courant la traversant et éviter ainsi de l'endommager.

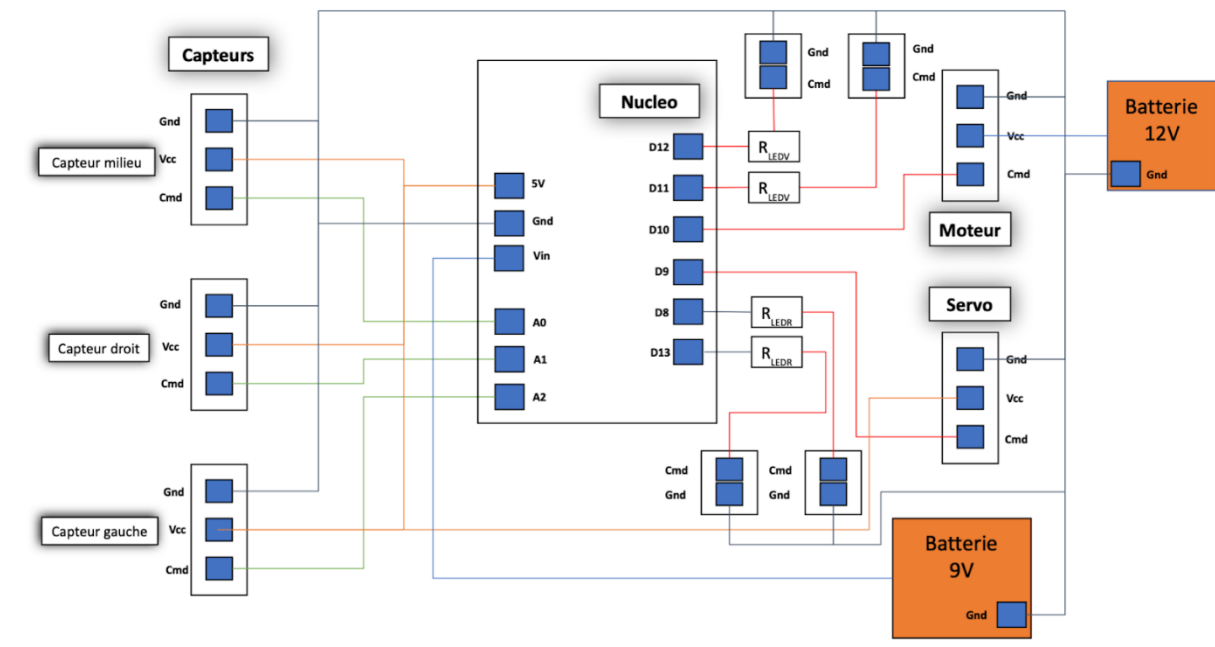


Figure 7 – Schéma du câblage avec les capteurs de distance Sharp

2.1.2. Partie algorithmique

Maintenant que nous savons théoriquement comment faire avancer la voiture en évitant des obstacles, il faut désormais procéder à la programmation sous MBED.

Les grandes lignes du code

Le but est que la voiture aille tout droit et en ligne droite, mais de la faire ralentir de manière linéaire à partir d'un premier seuil de tension pour le capteur de devant, donc d'un premier seuil de distance à un obstacle, puis de l'arrêter si un deuxième seuil est atteint. On souhaite également faire tourner la voiture de manière plus ou moins forte selon les valeurs données par les capteurs des côtés, donc selon la distance à l'obstacle et le côté où il se trouve.

De plus, selon que le seuil 1 ou 2 soit dépassé, on veut qu'une ou deux LEDs vertes s'allument, et que lorsque la voiture tourne une LEDs rouge s'allume pour indiquer la direction dans laquelle elle tourne.

```
PwmOut servo(D9);
PwmOut moteur(D10);
AnalogIn analog_in_cpt_d(A1);
AnalogIn analog_in_cpt_g(A2);
AnalogIn analog_in_cpt_mil(A0);

DigitalOut led_seuil_1(D11);
DigitalOut led_seuil_2(D12);

DigitalOut phare_arriere_d(D8);
DigitalOut phare_arriere_g(D13);
```

Figure 8 – Définition des variables

Explication du code

On commence avant tout par définir les abréviations utilisées (Fig. 8), puis les entrées et sorties de la carte. Les entrées analogiques sont celles des capteurs, A0 pour celui de devant, A1 et A2 pour ceux des côtés, et les sorties PWM sont celles du moteur à courant continu (D10) et du servomoteur (D9). On définit aussi des sorties pour les deux LEDs vertes (D11 et D12) et rouges (D8 et D13).

On définit ensuite les constantes utilisées (Fig. 9) telles que la vitesse de base, à laquelle la voiture roule s'il n'y a pas d'obstacles, ou la vitesse minimale que peut atteindre la voiture. Ces constantes sont exprimées en microsecondes car on va agir en réalité sur le rapport cyclique du signal délivré par la sortie attribuée au moteur. Par exemple, pour une période de 20 ms, la voiture est immobile pour un temps haut de 1500 µs et est à sa vitesse de base pour 1620 µs.

```
int vit_de_base = 1620; // µs
int vit_max = 1700; // µs
int vit_min = 1550; // µs
int vit_nulle = 1500; // µs
int vit_m_a = 1350; // µs
```

Figure 9 – Définition des constantes

De même, on définit les constantes utiles pour les capteurs et pour le servomoteur, ce dernier ayant un fonctionnement similaire au moteur, l'angle donné aux roues dépendant du rapport cyclique de la sortie PWM attribuée.

On doit ensuite calculer les coefficients pour les expressions de la vitesse (Fig. 10) et de l'angle imposé aux roues à l'aide des constantes définies précédemment. Puis dans la boucle *while*, on lit et on convertit en volts les tensions données par les capteurs.

```
double a_v = (vit_min - vit_de_base) / (vs_mil_2 - vs_mil_1);
double b_v = vit_de_base - a_v*vs_mil_1;
```

Figure 10 – Calcul des coefficients pour la vitesse

Puis, si le premier seuil de tension défini précédemment est dépassé (mais pas le deuxième), on commence par faire ralentir la voiture en modifiant le rapport cyclique du signal envoyé au moteur, et on allume l'une des deux LEDs vertes. On regarde ensuite si l'obstacle est plus proche du capteur de gauche ou de droite : on modifie en conséquence le rapport cyclique du signal envoyé au servomoteur pour faire tourner la voiture dans la direction adaptée et on allume la LED rouge correspondante (Fig. 11).


```

if ( tens_cpt_mil >= vs_mil_1 and tens_cpt_mil <= vs_mil_2 ) {
    led_seuil_1 = 1;
    led_seuil_2 = 0;
    moteur.pulsewidth_us(a_v * tens_cpt_mil + b_v);

    if ( tens_cpt_g >= tens_cpt_d ) {
        servo.pulsewidth_us(a_servo_d * tens_cpt_d + b_servo_d);
        phare_arriere_d = 1;
        phare_arriere_g = 0;
    }

    else {
        servo.pulsewidth_us(a_servo_g * tens_cpt_g + b_servo_g);
        phare_arriere_d = 0;
        phare_arriere_g = 1;
    }
}

```

Figure 11 – Codage lorsque la voiture atteint le seuil 1

Lorsque le deuxième seuil est dépassé, on allume les deux LEDs vertes, on enclenche la marche arrière. On recule en tournant dans la direction adaptée et on allume la LED rouge correspondante. On attend une seconde pour que la voiture puisse finir sa manœuvre (Fig. 12).

```

else {
    if ( tens_cpt_mil > vs_mil_2 ){           // Si on dépasse le deuxième seuil
        moteur.pulsewidth_us(vit_m_a);
        led_seuil_1 = 1;
        led_seuil_2 = 1;
        if ( tens_cpt_g >= tens_cpt_d ) {
            servo.pulsewidth_us(a_servo_d * tens_cpt_d + b_servo_d);
            phare_arriere_d = 1;
            phare_arriere_g = 0;
        }

        else {
            servo.pulsewidth_us(a_servo_g * tens_cpt_g + b_servo_g);
            phare_arriere_d = 0;
            phare_arriere_g = 1;
            wait(1.0);
        }
    }
}

```

Figure 12 – Codage lorsque la voiture atteint le seuil 2

Enfin, si aucun des deux seuils n'est dépassé : on éteint les LEDs, on réinitialise la position des roues et on avance à la vitesse de base (Fig. 13)

```

else {           // Sinon, on avance tout droit à la vitesse de base et on éteint le
    moteur.pulsewidth_us(vit_de_base);
    servo.pulsewidth_us(angle_zero);
    led_seuil_1 = 0;
    led_seuil_2 = 0;
    phare_arriere_d = 0;
    phare_arriere_g = 0;
}

```

Figure 13 – Codage dans le cas où aucun seuil n'est atteint

En résumé, nous avons la démarche suivante (Fig. 14).

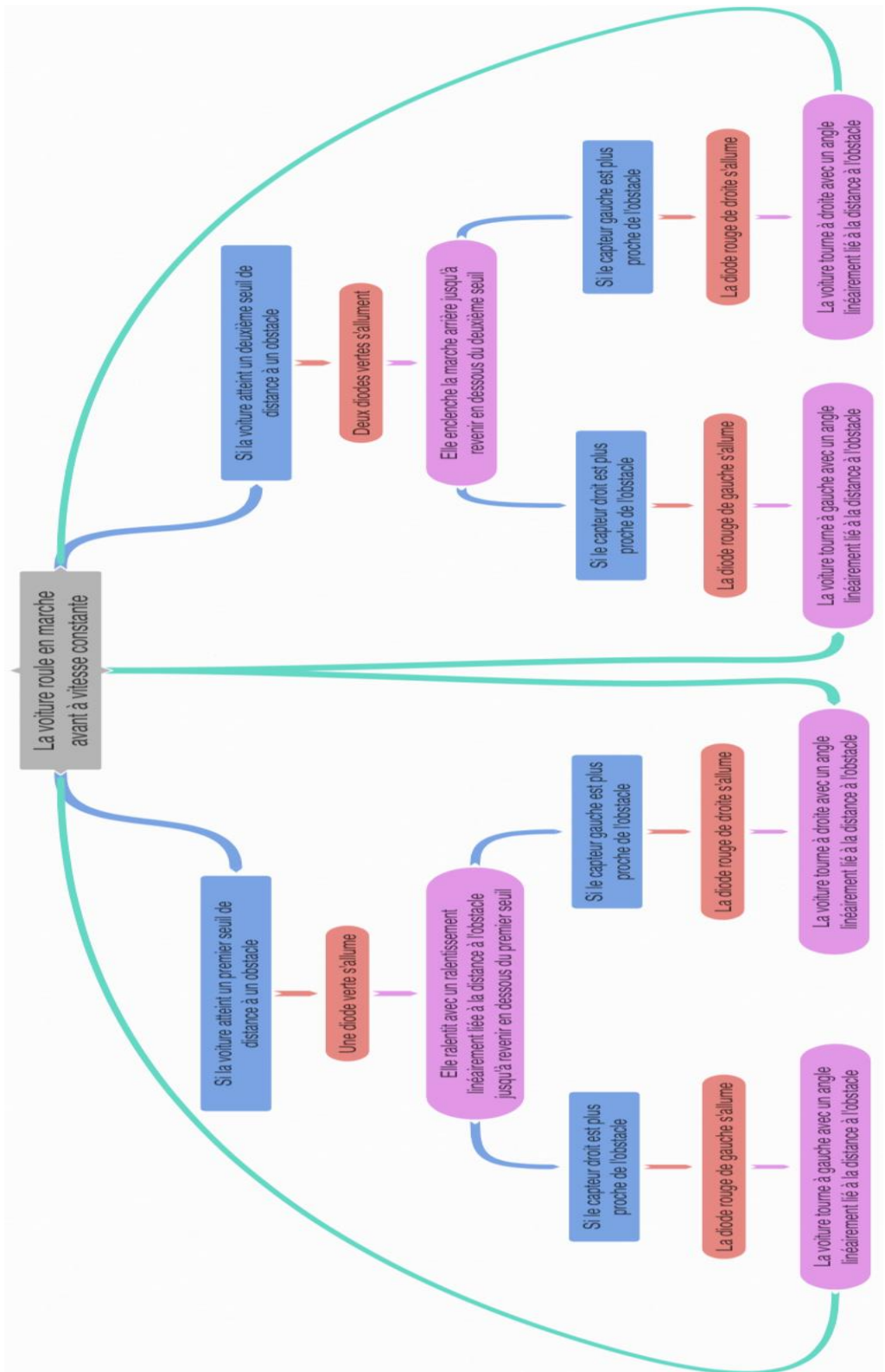


Figure 14 – Algorithme pour les capteurs de distance Sharp

2.2. Détection des obstacles avec le RPLidar (niveau 2)

2.2.1. Partie électronique

Alimentation

L'alimentation du moteur de la voiture se fait grâce à une pile de 12 V, qui alimente également la carte Nucléo. Le Lidar nécessite d'être alimenté en 5V, ce que nous pouvons faire grâce à la sortie 5V de la carte Nucléo (Fig. 15).

Détection

Afin d'améliorer la détection des obstacles et éviter d'avoir des angles morts, nous avons utilisé un capteur Lidar. Cet appareil permet de visualiser son environnement via des mesures de distance et d'angle. En effet, le Lidar est constitué d'un émetteur qui génère un signal et d'un récepteur qui vient récupérer ce signal et ainsi déterminer la distance de l'objet réfléchi. Comme le Lidar est mis en rotation, nous recevons des informations à 360° autour de l'appareil. Ainsi, l'ensemble des données qu'il acquiert seront stockées dans un tableau à chaque tour.

Nous avons utilisé le logiciel du fabricant de Lidar pour visualiser ce que capte le Lidar autour de lui en temps réel via l'ordinateur.

La vitesse de rotation du Lidar est réglable, et celui-ci capte 16000 données par secondes. Ainsi nous pouvons jouer sur la vitesse de rotation afin d'obtenir des mesures plus ou moins précises sur des objets se trouvant jusqu'à 25 mètres.

Fonctionnement

Le fonctionnement de la voiture avec le Lidar est très similaire à celle des capteurs Sharp. En effet, nous utilisons le même moteur et servomoteur. Cependant, trop d'informations sont envoyées par le code au servomoteur, ce qui occasionnait parfois des erreurs et un mauvais pilotage de la voiture.

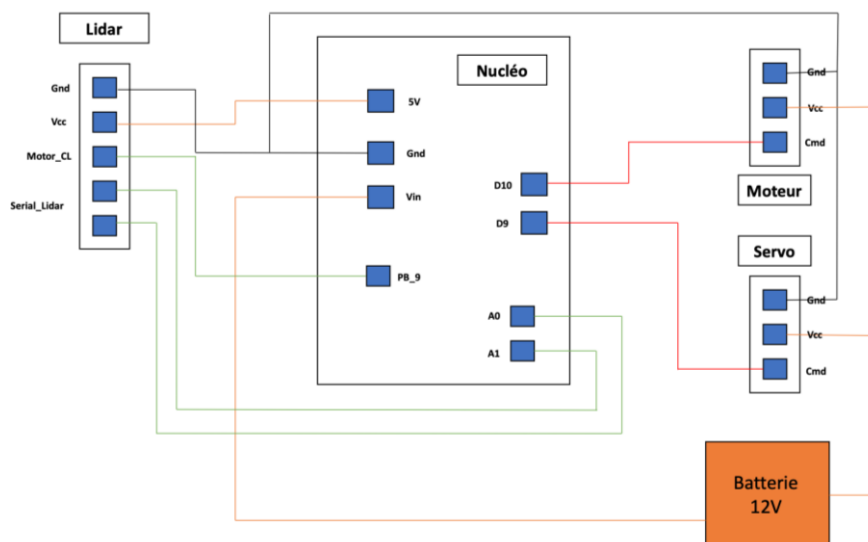


Figure 15 – Schéma du câblage avec le Lidar

2.2.2. Partie algorithmique

Voici comment nous avons procédé à la programmation de la voiture avec le capteur Lidar.

Le but de ce programme est que la voiture évite des obstacles lorsque le Lidar en détecte. Le Lidar est un capteur qui détecte en continu et à 360° les objets qui l'entourent. En effet, une fois en rotation, celui-ci détecte les objets et via le programme qui suit, nous pouvons récupérer la distance et l'orientation angulaire de l'obstacle le plus proche et de celui le plus loin par rapport au Lidar.

L'idée que nous avons eue au départ était d'écrire un code tel que la voiture évite l'obstacle le plus proche quand celui-ci dépasse une certaine valeur seuil. Ensuite, nous voulions que la voiture s'oriente dans la direction de l'objet le plus loin afin de ne pas rencontrer d'obstacle. Cependant, nous nous sommes rendu compte que notre programme ne fonctionnait pas très bien. Nous avons écrit un deuxième code, dans lequel la voiture s'intéresse seulement à l'objet qui se trouve le plus proche. Nous avons remarqué que sans se préoccuper de l'objet le plus distant, la voiture évitait mieux les obstacles. Ainsi, nous allons détailler le code final.

On commence par définir la fonction *FindMinDist* (Fig. 16). Cette fonction renvoie la distance *_minDist* et l'orientation angulaire *_angleDistMin* de l'obstacle le plus proche. Nous allons utiliser cette fonction dans la suite du code pour diriger la voiture. Nous avons montré sur le schéma électrique du Lidar les entrées et les sorties des composants (Fig. 15).

```
void findMinDist(void){
    __minDist = 16000;
    __angleDistMin = 0;
    for(int ki = 0; ki < 360; ki++){
        if(__acq_lidar_1_old[ki] < __minDist){
            if(__acq_lidar_1_old[ki] > 0){
                __angleDistMin = ki;
                __minDist = __acq_lidar_1_old[ki];
            }
        }
    }
}
```

Figure 16 – Fonction *FindMinDist*

Nous initialisons ensuite le Lidar (Fig. 17), puis le moteur et le servomoteur (Fig. 18) de la même manière que dans la voiture avec les capteurs Sharp. Vous retrouverez les mêmes abréviations que pour le code avec les capteurs Sharp.

```

// initialisation lidar
int i=0;

__rotation.period_us(40);
__rotation.pulsewidth_us(20); //vitesse rotation
pc.printf("LIDAR TEST \n");

__lidar.putc(0XA5);

__lidar.putc(0X20); //Request

while(i<7) {

    if(__lidar.readable()) {
        tab[i]=__lidar.getc(); //Response
    }
    pc.printf("%d \n",i);
    i++;
}

pc.printf("Lidar repond ");
for (i=0; i<7; i++) pc.printf("%XX ", tab[i]);
pc.printf("\n\n");

__lidar.attach(IRQ_Rx_Lidar,Serial::RxIrq);
    
```

Figure 17 – Asservissement du Lidar

```

// initialisation moteur
moteur.period_us(periode); // Initialisation de la periode du moteur
moteur.pulsewidth_us(vit_nulle); // Initialisation en position 0 de l'angle des roues
wait_us(200000);
servo.period_us(20000); // Initialisation de la période du servomoteur
servo.pulsewidth_us(angle_zero);
    
```

Figure 18 – Asservissement du moteur et des servomoteurs

Une fois ces éléments initialisés, nous avons décidé des contraintes suivantes : si le Lidar détecte l'objet le plus proche entre les angles 200° et 345° (Fig. 19), on impose une condition sur le servomoteur pour que la voiture tourne à droite. Symétriquement, nous avons le code suivant pour que la voiture tourne à gauche (Fig. 20). Nous avons décidé d'afficher la fonction transmise au servomoteur, pour qu'on puisse voir en temps réel via TeraTerm si le Lidar récupère les informations souhaitées.

```

if (200< __angleDistMin && __angleDistMin < 345) {
    pc.printf("tourne a droite");
    servo.pulsewidth_us(1800);
    moteur.pulsewidth_us(1605);
    wait_us(300000); }
    
```

Figure 19 – Code pour tourner à droite

```

if (160> __angleDistMin && __angleDistMin > 15)
    pc.printf("tourne a gauche");
    servo.pulsewidth_us(300);
    moteur.pulsewidth_us(1605);
    wait_us(300000);
}
    
```

Figure 20 – Code pour tourner à gauche

De plus, si la distance est inférieure à un certain seuil, définie ici par la valeur seuil $_minDist=100$, et que l'angle est inclus dans l'intervalle $[0^\circ ;15^\circ] \cup [345^\circ ; 360^\circ]$, la voiture reculera. (Fig. 21).

```

if((\_minDist<100) || (15>__angleDistMin && __angleDistMin>0) || (360>__angleDistMin && __angleDistMin>345)){
    pc.printf("marche arriere");
    moteur.pulsewidth_us(1380);
    wait_us(300000); } //marche arriere
    
```

Figure 21 – Code pour la marche arrière

Enfin, dans les autres cas, la voiture avancera, et remettra l'orientation des servomoteurs à la position nulle.

En résumé, nous avons la démarche suivante (Fig. 22).

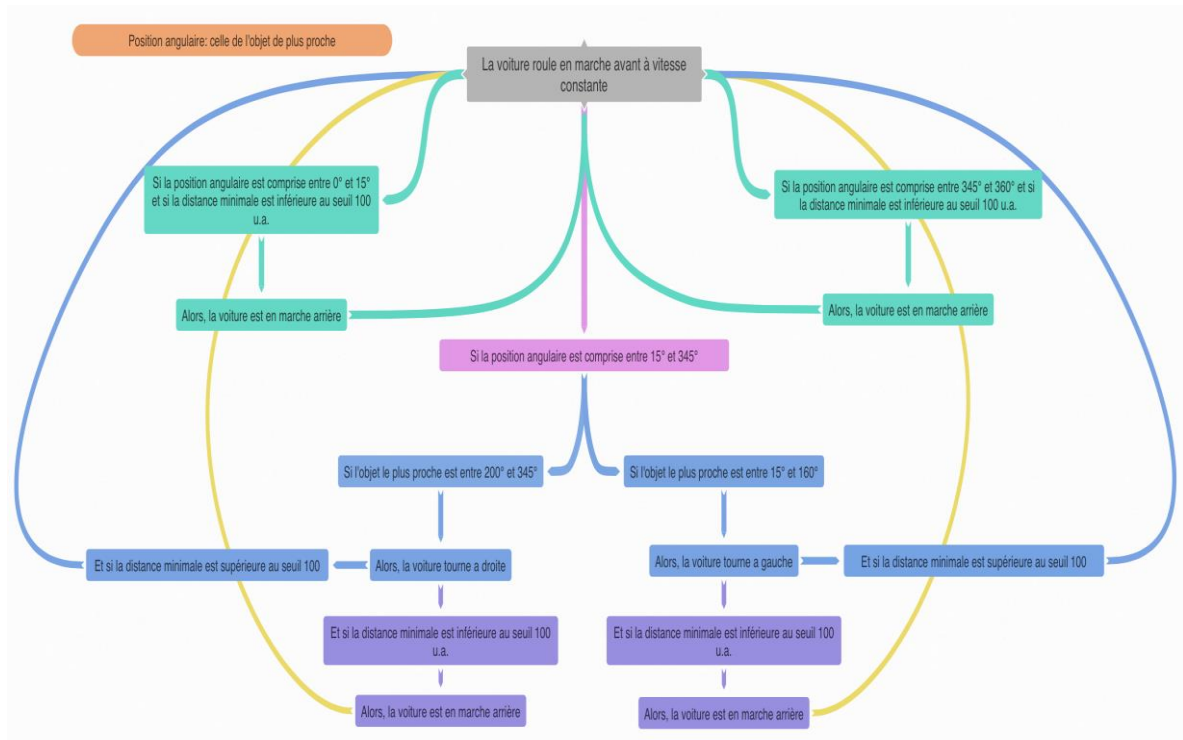


Figure 22 - Algorithme pour le Lidar

III. Valider et caractériser le système final

3.1. Test et validation

À chaque début de séance, nous effectuons le câblage de notre voiture et nous lui transmettons le code Mbed écrit à la dernière séance. Si tout fonctionne, on continue le projet et on teste chaque nouvelle fonctionnalité avant d'en écrire une nouvelle dans le code. Nous avons testé et caractérisé chaque composant (voir les parties ci-dessus) afin de comprendre comment ils fonctionnent séparément.

Capteurs

Pour les capteurs Sharp, nous avons établi la correspondance entre la distance de l'obstacle et la tension fournie par ces composants. Nous avons constaté par ailleurs que les capteurs étaient très directifs, ce qui faisait que si un mur n'était détecté que par les capteurs sur les côtés, la voiture ne l'évitait pas car notre code s'appuie en premier sur la détection d'un obstacle par le capteur central.

Concernant le Lidar, nous avons constaté qu'il fallait le surélever un peu du châssis car il détectait systématiquement la carte Nucléo à l'arrière, ce qui faussait l'acquisition des données, puisque la carte correspondait à l'obstacle le plus proche.

Démarrage de la voiture

Durant certaines séances, la voiture démarrait que si l'on effectuait un certain protocole. Lors de la séance 7, il fallait :

- Maintenir enfoncé le bouton « reset » de la carte Nucléo
- Allumer la voiture et entendre un « bip » court
- Relâcher le bouton « reset »

Vérification via des LEDs

Comme nous rencontrions souvent des problèmes liés aux valeurs seuils de la voiture (celle-ci semblait détecter l'obstacle très tôt, i.e. avant la distance que nous lui avons définie ou n'effectuait que des marche arrière sans jamais choisir de tourner à gauche ou à droite). Nous avons mis en place des LEDs vertes pour nous indiquer quand la voiture atteignait les différents seuils. Nous avons alors constaté que l'intervalle entre les 2 valeurs seuils était très faible, ce qui faisait que la voiture choisissait presque systématiquement la marche arrière.

Vitesse de la voiture

Nous avons fait rouler la voiture dans les couloirs de l'école afin de trouver une vitesse qui nous satisfaisait en réalisant un étalonnage. En effet, lorsque nous testions sa vitesse sur un support dans la salle d'électronique, la voiture roulait très vite mais une fois mis à terre, nous la trouvions trop lente. Cela est dû à l'adhérence de la roue au sol, et à un couple moteur pas assez élevé.

Élaboration du code pour les capteurs de distance Sharp

L'élaboration du code ne s'est pas faite automatiquement. En effet, nous avons dû construire le code étape par étape. Nous avons commencé par des codes simples visant à faire fonctionner les parties individuelles du montage final.

Nous avons ensuite appris à utiliser le servomoteur, en initialisant la période du signal PWM envoyé et en agissant sur le rapport cyclique, pour uniquement le faire tourner. Puis nous avons fait de même avec le moteur à courant continu.

Par exemple pour le servomoteur, nous l'avons d'abord fait alterner entre deux positions, avec des angles positif et négatif (Fig. 23) Pour le moteur, nous avons également appris à nous servir d'un pont en H afin de comprendre son fonctionnement une fois relié à la voiture.

```
#include "mbed.h"

PwmOut servo_mot(D10);
int main() {
    servo_mot.period_ms(20); // Initialisation période
    servo_mot.pulsewidth_us(1500); // Initialisation en position 0
    while(1) {
        servo_mot.pulsewidth_us(2500); // Angle négatif
        wait_us(800000);
        servo_mot.pulsewidth_us(900); // Angle positif ; 2500us = position à 90°
        wait_us(800000);
    }
}
```

Figure 23 – Asservissement du servomoteur 1

Nous avons ensuite appris à nous servir des capteurs, par exemple en changeant l'angle du servomoteur si un obstacle est proche d'un capteur ou d'un autre (Fig. 24). Nous avons ensuite écrit un code permettant la linéarité de la rotation des roues en fonction de la distance à un obstacle.


```

AnalogIn analog_ing(A0);
AnalogIn analog_ind(A1);
PwmOut roue_avant(D10);

int main()
{
    int i;
    roue_avant.period_ms(20); // Initialisation période
    roue_avant.pulsewidth_us(1300); // Initialisation en position 0
    int meas_gauche;
    int meas_droite;
    double tension_gauche;
    double tension_droite;

    while(1) {

        meas_gauche = analog_ing.read_u16(); // Convertit et lit la tension d'entrée analogique (valeur entre 0 et 65535

```

Figure 24 – Asservissement du servomoteur 2

Pour ce qui est du moteur à courant continu, nous avons commencé par écrire un code simple permettant de faire avancer la voiture en ligne droite à vitesse constante (Fig. 25) Puis nous avons progressivement complexifié ce code, par exemple pour faire avancer la voiture tant qu'elle ne rencontre pas d'obstacle puis l'arrêter lorsqu'elle en détecte un, puis en ajoutant les capteurs des côtés et en rendant linéaire son ralentissement.

```

PwmOut moteur(D10);

int main() {
    int vitesse_de_base = 1500; //µs

    int periode = 20000; //µs
    moteur.period_us(periode); // Initialisation de la periode

    while(1)
    {
        moteur.pulsewidth_us(vitesse_de_base);
    }
}

```

Figure 25 – Asservissement du moteur

Nous avons ensuite essayé de la faire reculer une fois qu'un deuxième seuil de distance à un obstacle est atteint, dans la situation où la voiture est trop proche pour pouvoir faire autrement qu'en reculant (comme un cul-de-sac). Nous nous sommes heurtés ici à quelques difficultés, car en théorie il aurait suffi de définir un temps haut pour le signal PWM qui soit inférieur à 1500µs (pour une période de 20000µs), mais cela ne fonctionnait pas. Nous avons ensuite essayé de repasser par la vitesse nulle, mais la voiture ne faisait que s'arrêter la plupart du temps (Fig. 26).

```

else if ( tens_cpt_mil > vs_mil_2 ){
    moteur.pulsewidth_us(vit_m_a);
    wait_us(50000);
    moteur.pulsewidth_us(vit_nulle);
    wait_us(50000);
    moteur.pulsewidth_us(vit_m_a);
    wait_us(500000);
}

```

Figure 26 – Passage par la vitesse nulle

En changeant l'alternateur pour un autre modèle plus récent, cette alternance de rapport cycliques n'était plus nécessaire, et il suffisait d'une seule ligne `moteur.pulsewidth_us(vit_m_a);`.

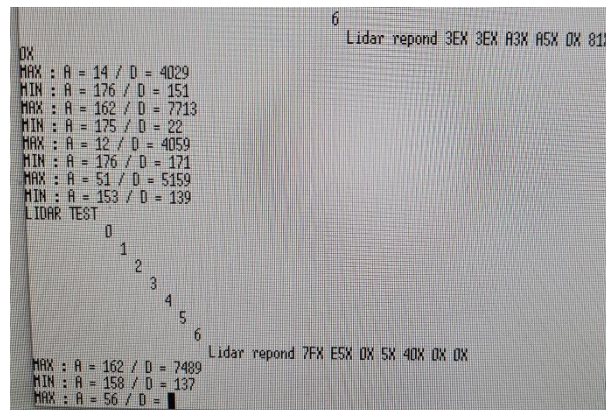
Mais il a fallu ajouter dans le dossier contenant le code sur mbed un nouveau fichier intitulé « mbed_app.json » contenant la ligne « requires » : [“bare-metal”].

Avec ces changements et en ajoutant un *wait* d'une seconde pour que la voiture ait le temps de finir ses manœuvres, nous avons bien réussi à faire en sorte que la voiture effectue ce que nous souhaitions.

Élaboration du code avec le Lidar

La mise en place de ce code était plus facile à faire, car nous avons déjà vu la plupart des composants nécessaires pour faire avancer la voiture avec le code pour la voiture avec les capteurs Sharp.

Cependant nous avons eu certaines difficultés pour trouver les seuils, trouver les angles seuils, l'angle zéro et vérifier le bon fonctionnement du code. Pour remédier à ce problème, nous avons utilisé TeraTerm. Cet outil nous a permis de voir en temps réel la réponse de la voiture en fonction du code implémenté. Ainsi nous avons pu définir les seuils, trouver les angles nécessaires et les temps d'attente appropriés. TeraTerm était un outil primordial qui a permis le bon fonctionnement de ce projet (Fig. 27).



```
6 Lidar repond 3EX 3EX A3X A5X 0X 01X
0X
MAX : A = 14 / D = 4029
MIN : A = 176 / D = 151
MAX : A = 162 / D = 7713
MIN : A = 175 / D = 22
MAX : A = 12 / D = 4059
MIN : A = 176 / D = 171
MAX : A = 51 / D = 5159
MIN : A = 153 / D = 139
LIDAR TEST
0
1
2
3
4
5
6
Lidar repond 7FX E5X 0X 5X 40X 0X 0X
MAX : A = 162 / D = 7489
MIN : A = 158 / D = 137
MAX : A = 56 / D =
```

Figure 27 – Test sur le logiciel TeraTerm

Nous avons eu des problèmes avec Mbed, car le Lidar ne voulait pas tourner. Mais il a fallu ajouter dans le dossier contenant le code sur Mbed un nouveau fichier intitulé « *mbed_app.json* » contenant la ligne *requires* : [*bare-metal*]. afin que le Lidar devienne compatible avec Mbed, tout comme avec les capteurs de distance Sharp.

De plus nous avons remarqué que le Lidar était trop bas, et celui-ci captait en permanence la carte Nucléo. Ainsi, nous avons rajouter une ligne de code afin qu'un message s'affiche dans la console pour vérifier si le Lidar capte la carte ou non.

3.2. Résultat final

À la fin du projet, la voiture avec les 3 capteurs de distance SHARP remplissait pleinement le cahier des charges et effectuait toutes les manœuvres demandées. La voiture fonctionnant avec le Lidar réussissait à tourner pour éviter les obstacles mais avait parfois du mal à effectuer une marche arrière.

IV. Comprendre les étapes de réalisation

4.1. Planning des séances

Nous avons réussi à respecter notre planning déterminé à la séance 2 (Fig. 28). Il nous a fallu toutefois plus de temps pour comprendre le fonctionnement du Lidar. Les tutoriels sur le site du LEnsE (Fig. 29) nous ont grandement aidés dans l'avancement de notre projet.



Figure 28 – Déroulé des séances

Trouver des infos dans la documentation
Piloter une LED
Contrôler un mouvement angulaire (servomoteur)
Faire varier la vitesse d'un moteur à courant continu
Piloter un moteur pas à pas
Générer une tension analogique
RPLidarA2 MBED
Faire une action après un événement
Faire une action instantanément après un événement externe
Faire une action à intervalle régulier
Faire plusieurs actions à intervalle régulier
Caractériser un traitement numérique
RPLidarA2 : Interface Protocol and Application Notes
Brancher un pont en H

Figure 29 – Nouvelles compétences acquises à l’aide des tutoriels du LEnsE (liste non exhaustive)

4.2. Difficultés rencontrées et analyse du travail de l’équipe

Le principal problème était de refaire systématiquement le câblage en début de chaque séance. En effet, nous avons soudé certaines parties de notre montage et quand nous arrivions en séance, il fallait recommencer la procédure. Les voitures étaient parfois différentes d’une séance à l’autre. D’autre part, nous avons souvent rencontré des composants défectueux (carte Nucléo par exemple) ou usés (piles), ce qui a affecté l’avancement de notre projet. En ce qui concerne les piles, nous n’avions en effet pas eu le réflexe au départ de vérifier si les piles étaient bien chargées avant de commencer à démarrer la voiture. En outre, il a fallu tester chaque dispositif du montage à l’aide l’oscilloscope afin de déterminer quel composant ne fonctionnait plus. Toutes ces manipulations étaient fortement chronophages. Les roues des voitures n’étaient pas bien alignées avec le châssis, ce qui entraînait une déviation naturelle de la voiture lorsqu’elle roulait.

Pour la répartition des tâches, nous avons estimé que c’était complexe au départ, car pour coder le pilotage de la voiture, il faut connaître le fonctionnement des divers composants. Mais pour savoir si les composants étaient défectueux, il faut d’abord prendre connaissance du code afin de déterminer si ce n’est pas ce dernier qui est en cause. Puis, comme la maîtrise du Lidar était plus compliqué que les capteurs Sharp, nous avons pris un peu plus de temps à comprendre le code fourni sur le site du LEnsE. Vers les dernières séances, 2 groupes se sont formés : un pour réaliser les derniers ajustements pour la voiture avec les capteurs Sharp, et un autre pour prototyper la voiture avec le Lidar. Ainsi, la communication entre les différents groupes était cruciale pour permettre l’avancement du projet. La tenue régulière d’un carnet de bord s’est

révélée très utile pour lister les problèmes rencontrés et les solutions que nous avons apportées. Cela permettait de gagner en efficacité.

V. Conclusion et développement possibles

Conclusion

Au-delà de l'aspect technique, ce projet nous a permis de gagner en autonomie et d'acquérir ou d'approfondir des compétences transverses : gestion de projet, gestion du temps, planification, travail d'équipe, se documenter, effectuer un état de l'art, etc. Ces séances de quatre heures, bien qu'éprouvantes parfois, ont été enrichissantes et nous ont challengé à réaliser une voiture autonome dans un temps imparti.

Bien que nous ayons mené à terme l'ensemble des fonctionnalités facultatives que nous nous sommes fixés en début de projet, nous avons pensé à d'autres améliorations possibles :

- Prendre une moyenne de plusieurs valeurs données par le Lidar pour avoir une information plus « stable » et permettre de mieux piloter la voiture
- Garder en mémoire les obstacles rencontrés et cartographier son environnement
- Piloter la voiture par Bluetooth et lui demander d'aller d'un point A à un point B en évitant de manière autonome les obstacles à partir de la cartographie précédente

Remerciements

Nous tenons à remercier M. Villemejeane et l'ensemble des encadrants des projets IETI pour leur investissement et leur soutien sans quoi la réalisation de ce projet n'aurait pas été possible.

ANNEXES

ANNEXE 1 : Code pour les capteurs de distance SHARP

```
/* PROJET IETI : VOITURE AUTONOME AVEC CAPTEURS SHARP
 * BIVAS Elsa - DE BEAUREGARD Alban - TAN Stella - UTHAYAKUMAR Jashaani
 * Version 26/05/2021
 * Code pour faire ralentir la voiture de manière linéaire à partir d'un premier seuil de
 tension puis initier une marche arrière si un deuxième est atteint selon les données
 du capteur de devant,
 * Et pour faire tourner la voiture de manière plus ou moins importante selon les
 valeurs données par les capteurs des côtés
 * Des LEDs s'allument lorsque la voiture ralentit ou recule, et lorsqu'elle tourne
 */
```

```
#include "mbed.h"
```

```
/*
Definition des abréviations :
vit = vitesse
cpt = capteur
mil = milieu
d = droit
g = gauche
cot = côtés
vs = valeur seuil
m_a = marche arrière
tens = tension
servo = servomoteur
*/
```

```
PwmOut servo(D9);
PwmOut moteur(D10);
AnalogIn analog_in_cpt_d(A1);
AnalogIn analog_in_cpt_g(A2);
AnalogIn analog_in_cpt_mil(A0);
```

```
DigitalOut led_seuil_1(D11);
DigitalOut led_seuil_2(D12);
```

```
DigitalOut phare_arriere_d(D8);
DigitalOut phare_arriere_g(D13);
```

```
// Definition des constantes utiles pour contrôler la vitesse :
int vit_de_base = 1620;    // µs
int vit_max = 1700;       // µs
int vit_min = 1550;       // µs
```

```

int vit_nulle = 1500;      // µs
int vit_m_a = 1350;      // µs

// Definition des constantes utiles pour les capteurs :
double vs_mil_1 = 1.0;    // V
double vs_mil_2 = 2.0;    // V
double vs_cot_1 = 0.8;    // V
double vs_cot_2 = 1.5;    // V

// Definition des constantes utiles pour le servomoteur :
int angle_zero = 1300;    // µs
int angle_max = 300;      // µs
int angle_min = 2300;     // µs
int periode = 20000;     // µs

// Definition des variables :
int meas_cpt_mil;
double tens_cpt_mil;

int meas_cpt_d;
double tens_cpt_d;

int meas_cpt_g;
double tens_cpt_g;

// Calcul des coefficients pour l'expression linéaire de la vitesse :
double a_v = (vit_min - vit_de_base) / (vs_mil_2 - vs_mil_1);
double b_v = vit_de_base - a_v*vs_mil_1;

// Calcul des coefficients pour l'expression linéaire de la tension du servomoteur pour
tourner à droite (et donc de la rotation des roues) :
double a_servo_d = (angle_zero - angle_max) / (vs_cot_1 - vs_cot_2);
double b_servo_d = angle_max - a_servo_d*vs_cot_2;

// Calcul des coefficients pour l'expression linéaire de la tension du servomoteur pour
tourner à gauche (et donc de la rotation des roues) :
double a_servo_g = (angle_zero - angle_min) / (vs_cot_1 - vs_cot_2);
double b_servo_g = angle_min - a_servo_g*vs_cot_2;

int main() {
moteur.period_us(periode);    // Initialisation de la periode du moteur
moteur.pulsewidth_us(vit_nulle);
wait(2.0);                    // Initialisation et étalonnage du moteur
servo.period_us(20000);       // Initialisation de la période du servomoteur
servo.pulsewidth_us(angle_zero); // Initialisation en position 0 de l'angle des roues
led_seuil_1 = 0;              // Initialisation des leds en position éteinte
led_seuil_2 = 0;
phare_arriere_d = 0;
phare_arriere_g = 0;

```



```

while(1)
{
    // Convertit et lit la tension d'entrée analogique (valeur entre 0 et 65535)
    meas_cpt_mil = analog_in_cpt_mil.read_u16();
    meas_cpt_d = analog_in_cpt_d.read_u16();
    meas_cpt_g = analog_in_cpt_g.read_u16();

    // Conversions en tensions
    tens_cpt_mil = (meas_cpt_mil / 65535.0) * 3.3; // V
    tens_cpt_d = (meas_cpt_d / 65535.0) * 3.3; // V
    tens_cpt_g = (meas_cpt_g / 65535.0) * 3.3; // V

    if ( tens_cpt_mil >= vs_mil_1 and tens_cpt_mil <= vs_mil_2 ) {
        led_seuil_1 = 1;
        led_seuil_2 = 0;
        moteur.pulsewidth_us(a_v * tens_cpt_mil + b_v); // Modification
        du rapport cyclique et donc de la vitesse en fonction de la tension lue par le capteur

        if ( tens_cpt_g >= tens_cpt_d ) {
            servo.pulsewidth_us(a_servo_d * tens_cpt_d + b_servo_d); // Si la
            tension mesurée à gauche est plus élevée qu'à droite, on tourne à droite
            phare_arriere_d = 1; // On allume le phare droit
            et on éteint le gauche
            phare_arriere_g = 0;
        }

        else {
            servo.pulsewidth_us(a_servo_g * tens_cpt_g + b_servo_g); // Si
            c'est l'inverse, on tourne à gauche
            phare_arriere_d = 0; // On allume le phare
            gauche et on éteint le droit
            phare_arriere_g = 1;
        }
    }

    else {
        if ( tens_cpt_mil > vs_mil_2 ){ // Si on dépasse le deuxième seuil
            moteur.pulsewidth_us(vit_m_a);
            led_seuil_1 = 1;
            led_seuil_2 = 1;
            if ( tens_cpt_g >= tens_cpt_d ) {
                servo.pulsewidth_us(a_servo_d * tens_cpt_d + b_servo_d); //
                Si la tension mesurée à gauche est plus élevée qu'à droite, on tourne à droite en
                marche arrière
                phare_arriere_d = 1; // On allume le phare
                droit et on éteint le gauche
                phare_arriere_g = 0;
            }
        }
    }
}

```

```

        else {
            servo.pulsewidth_us(a_servo_g * tens_cpt_g +
b_servo_g); // Si c'est l'inverse, on tourne à gauche
            phare_arriere_d = 0; // On allume le
phare gauche et on éteint le droit
            phare_arriere_g = 1;
            wait(1.0); // On attend une
seconde pour que la voiture puisse finir sa manoeuvre
        }
    }
    else { // Sinon, on avance tout droit à la vitesse de base et on
éteint les leds
        moteur.pulsewidth_us(vit_de_base);
        servo.pulsewidth_us(angle_zero);
        led_seuil_1 = 0;
        led_seuil_2 = 0;
        phare_arriere_d = 0;
        phare_arriere_g = 0;
    }
}
}
}
}

```


ANNEXE 2 : Code pour le Lidar

```
/* PROJET IETI : VOITURE AUTONOME AVEC LIDAR
 * BIVAS Elsa - DE BEAUREGARD Alban - TAN Stella - UTHAYAKUMAR Jashaani
 * Version 26/05/2021 - Écriture du code à partir de celui de M.Villemejeane
 */

#include "mbed.h"
#include "platform/mbed_thread.h"

PwmOut servo(PB_13);
PwmOut moteur(PC_9);

// Definition des constantes utiles pour contrôler la vitesse :
int vit_de_base = 1600;    // µs
int vit_max = 1610;       // µs
int vit_min = 1560;       // µs
int vit_nulle = 1500;     // µs
int vit_m_a = 1350;       // µs

// Définition des constantes du Lidar :
int Amin; // position angulaire de l'objet le plus proche
int Amax; // position angulaire de l'objet le plus loin détecté
int Dmin; // distance de l'objet le plus proche
int Dmax; // distance de l'objet le plus loin détecté

// Definition des constantes utiles pour le servomoteur :
int angle_zero = 1300;    // µs
int angle_max = 300;     // µs
int angle_min = 2300;    // µs
int periode = 20000;     // µs

DigitalOut tour_ok(D13);
Serial pc(USBTX, USBRX, 115200);
Serial __lidar(A0, A1, 115200);
PwmOut __rotation(PB_9); // Pour le MOTORCL

// Distance en fonction de l'angle récupéré par le Lidar
int __acq_lidar_1[360];
int __acq_lidar_1_old[360];

// Variables pour stocker différents paramètres : distance maximale, distance
minimale, angles associés
int __somme, __maxDist, __minDist, __angleDistMax, __angleDistMin;

// Data received by Lidar
unsigned char __data_tab[7];
unsigned char tab[7]= {};
uint16_t acq_lidar_1[360]= {};
```

```

uint16_t acq_lidar_2[360]= {};
int num_tab_acq=1 ;
int flag_chgt_tableau = 0;
void IRQ_Rx_Lidar(void);
void IRQ_dataScan(void);
float commande_moteur;
float distance_max=0;

// Fonction déterminant la distance et l'angle de l'objet le plus loin détecté
void findMaxDist(void){
    __maxDist = 0;
    __angleDistMax = 0;
    for(int ki = 0; ki < 360; ki++){
        if(__acq_lidar_1_old[ki] > __maxDist){
            __angleDistMax = ki;
            __maxDist = __acq_lidar_1_old[ki];
        }
    }
    printf("MAX : A = %d / D = %d \r\n",__angleDistMax, __maxDist);
}

// Fonction déterminant la distance et l'angle de l'objet le plus proche détecté
void findMinDist(void){
    __minDist = 16000;
    __angleDistMin = 0;
    for(int ki = 0; ki < 360; ki++){
        if(__acq_lidar_1_old[ki] < __minDist){
            if(__acq_lidar_1_old[ki] > 0){
                __angleDistMin = ki;
                __minDist = __acq_lidar_1_old[ki];
            }
        }
    }
    printf("MIN : A = %d / D = %d \r\n",__angleDistMin, __minDist);
}

int main()
{
// initialisation lidar
    int i=0;
    __rotation.period_us(40);
    __rotation.pulsewidth_us(20); //vitesse rotation
    pc.printf("LIDAR TEST \n");

    __lidar.putc(0XA5);
    __lidar.putc(0X20); //Request
    while(i<7) {
        if(__lidar.readable()) {
            tab[i]=__lidar.getc(); //Response

```

```

    }
    pc.printf("%d \n",i);
    i++;
}
pc.printf("Lidar repond ");
for (i=0; i<7; i++) pc.printf("%XX ", tab[i]);
pc.printf("\n\r");
__lidar.attach(IRQ_Rx_Lidar,Serial::RxIrq);

// initialisation moteur
moteur.period_us(periode); // Initialisation de la periode du moteur
moteur.pulsewidth_us(vit_nulle); // Initialisation en position 0 de l'angle des roues
wait_us(2000000);
servo.period_us(20000); // Initialisation de la période du servomoteur
servo.pulsewidth_us(angle_zero);

while (1) {
    wait_us(800000);
    findMaxDist();
    findMinDist();
    if (200< __angleDistMin && __angleDistMin < 345) {
        pc.printf("tourne a droite");
        servo.pulsewidth_us(1800);
        moteur.pulsewidth_us(1605) ;
        wait_us(300000); }
    if (160> __angleDistMin && __angleDistMin > 15) {
        pc.printf("tourne a gauche");
        servo.pulsewidth_us(300);
        moteur.pulsewidth_us(1605);
        wait_us(300000);
    }
    if (160<__angleDistMin && __angleDistMin<200) {
        moteur.pulsewidth_us(1612);
        pc.printf("trop bas");
        wait_us(300000);}
    if((__minDist<100) || (15>__angleDistMin && __angleDistMin>0) ||
(360>__angleDistMin && __angleDistMin>345)){
        pc.printf("marche arriere");
        moteur.pulsewidth_us(1380);
        wait_us(300000); }//marche arriere
    else {
        moteur.pulsewidth_us(1612);
        pc.printf("marche avant");
        wait_us(400000);
        servo.pulsewidth_us(angle_zero);
    }
    //servo.pulsewidth_us(angle_zero);
}
}

```

```

void IRQ_dataScan(void){
    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)
{

```

```

    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;
    }
}
}
}

```

ANNEXE 3 : Vidéo de démonstration

