INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | université PARIS-SACLAY

Projet léti 1A :

La Voiture Autonome

Cannelle CLAVIER
Samia Ouggahi
Maxime NURWUBUSA
Maxime TOMASIAN

Introduction

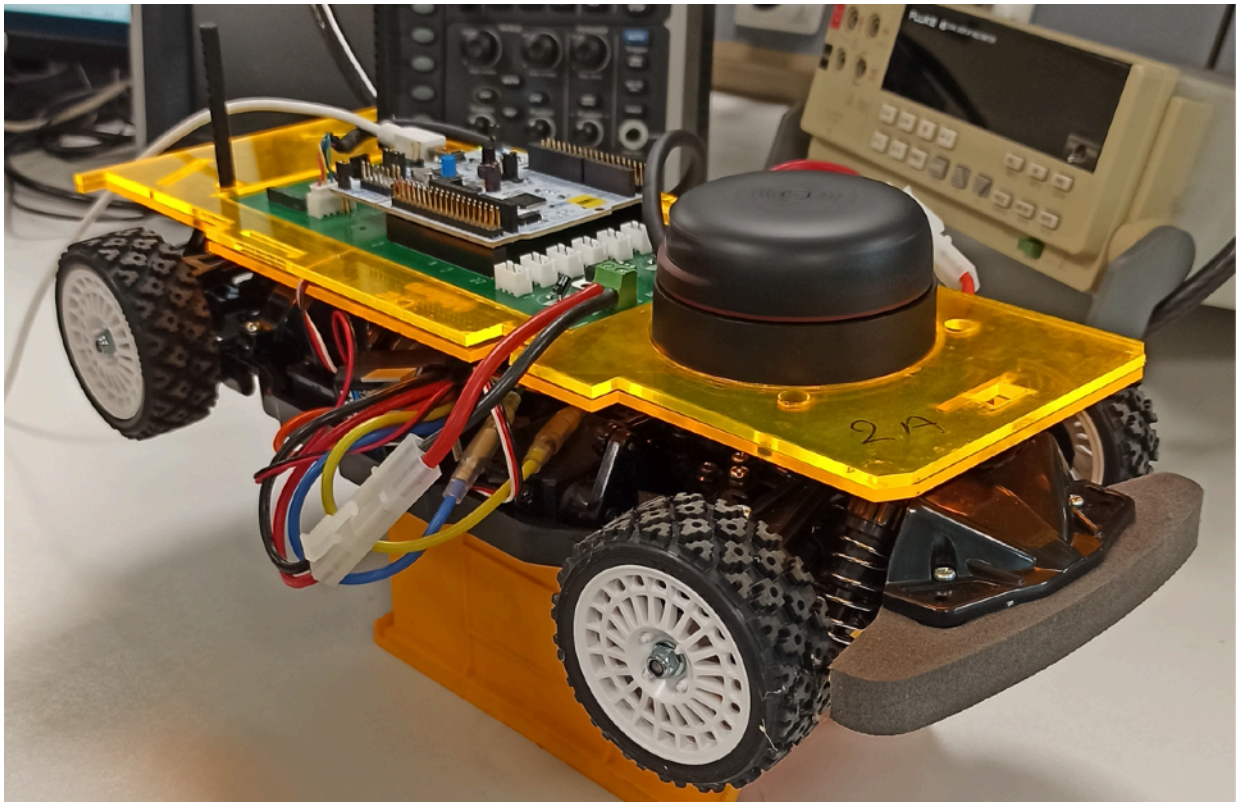


Photo de la voiture autonome

La thématique que nous avons choisi est la voiture autonome. Thématique récente, le principe d'une voiture autonome est d'avoir une voiture qui puisse se conduire sans pilote. Une voiture autonome est constituée de capteurs qui envoient l'information à un microcontrôleur, qui après traitement des données définit la vitesse et la direction des roues.

Notre but au cours de ce projet a été de réaliser une voiture qui puissent avancer dans une zone plate non-glissante, sans heurter d'obstacle et s'arrêter avant de le percuter dans le cas échéant.

Pour nos objectifs, il y a la détermination de la relation entre le rapport cyclique du signal de sortie et la vitesse et direction des roues. Par la suite, il faut travailler sur la transmission des ordres aux roues par la carte nucléo. Il faut aussi créer un programme de démarrage qui initialise le départ de la voiture après allumage et d'un programme d'arrêt qui arrête la voiture si un obstacle est trop proche. Enfin, la voiture doit contourner les obstacles et ajuster sa trajectoire à partir des informations fournies par les détecteurs.

Pour, cela nous avons dans un premier temps étudié le pilotage des roues par la carte nucléo. Ensuite, nous avons en deux temps travailler sur la détection et le traitement des données. Nous avons d'abord utilisé des capteurs classiques puis un Lidar.

Découpage fonctionnel

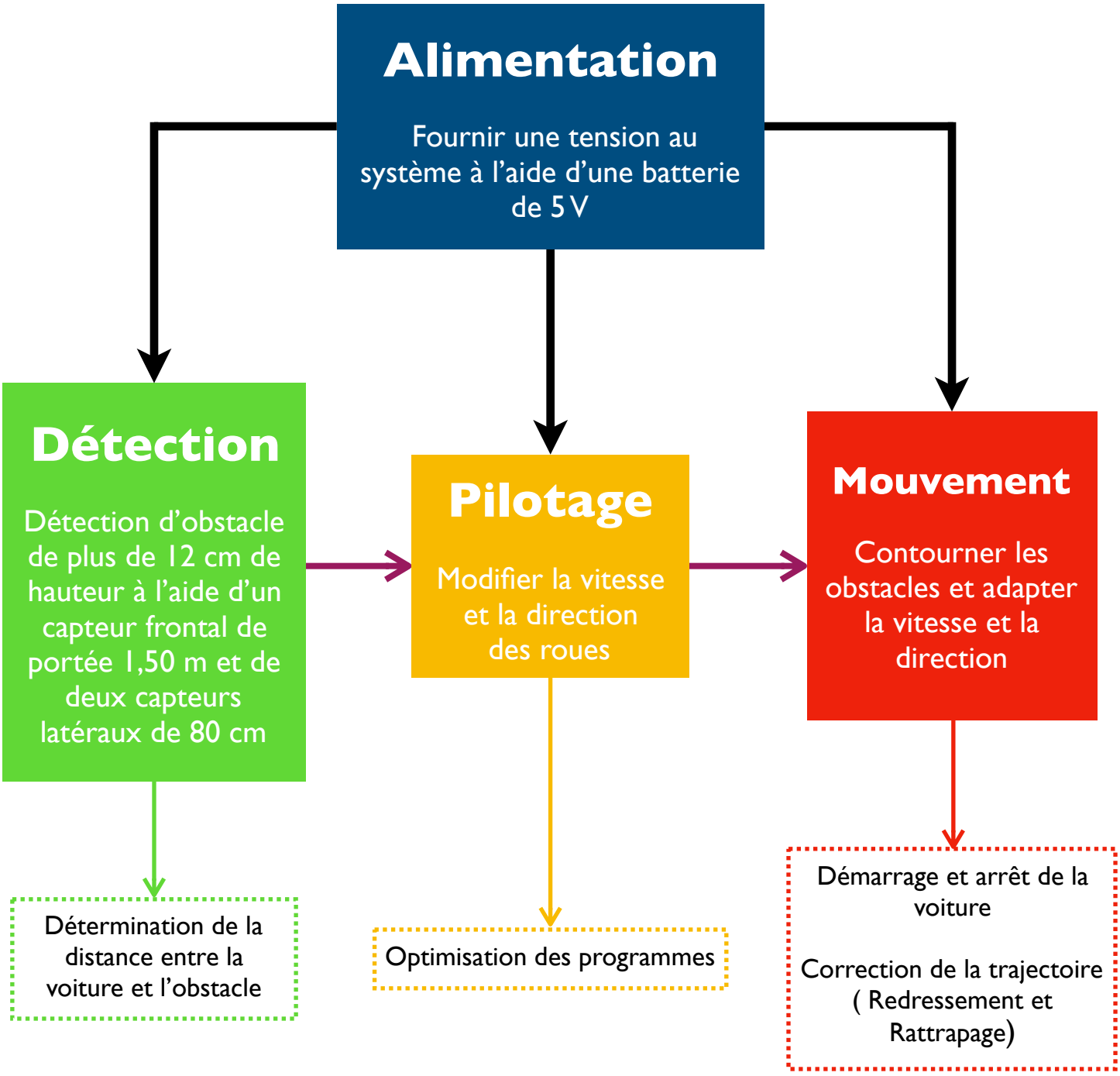


Schéma bloc de la voiture autonome

Schémas électroniques

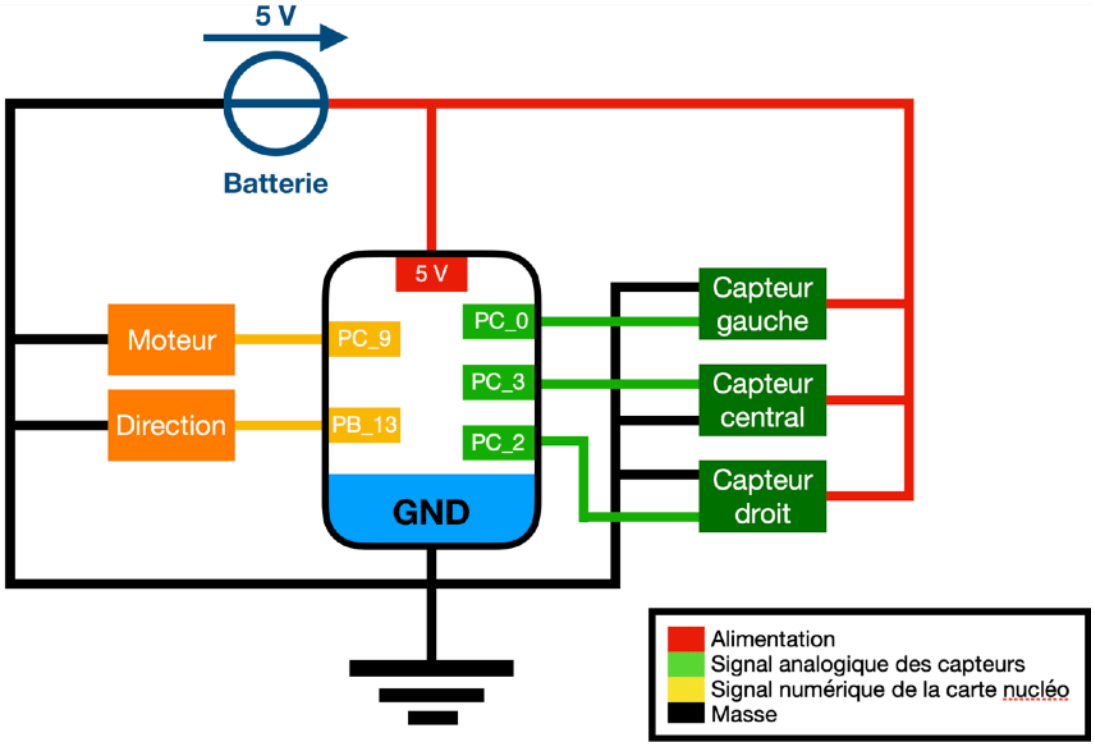


Schéma du circuit avec les 3 détecteurs

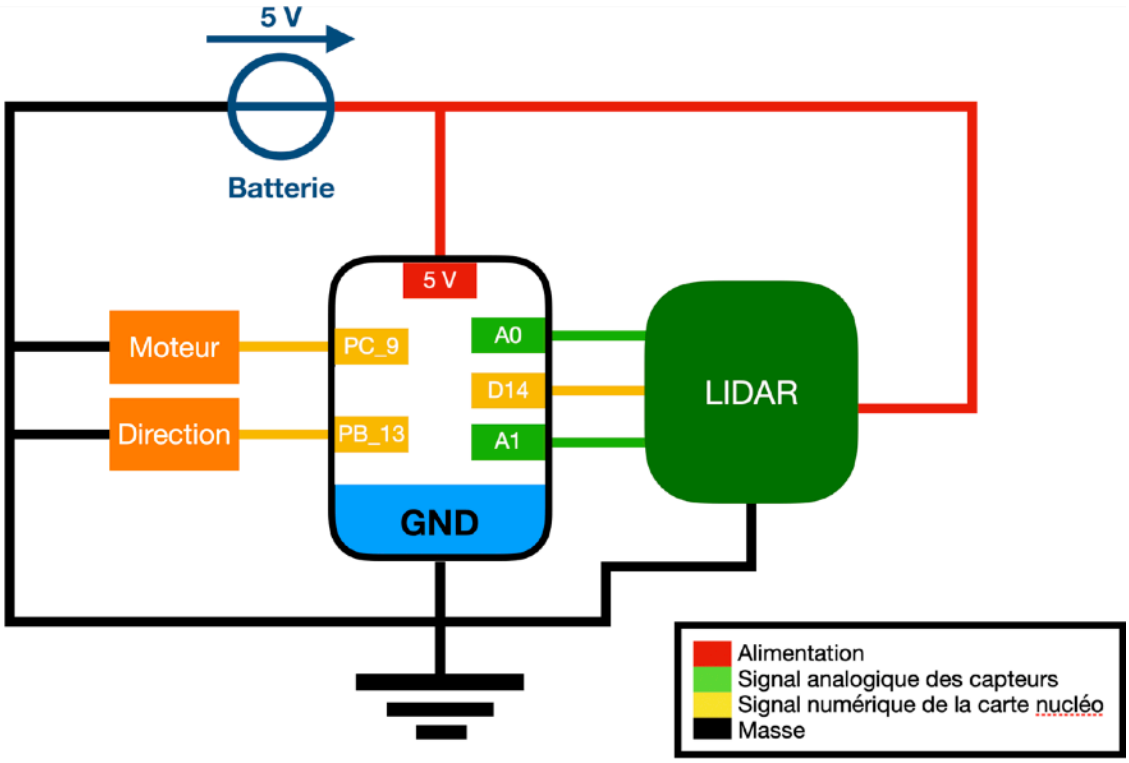
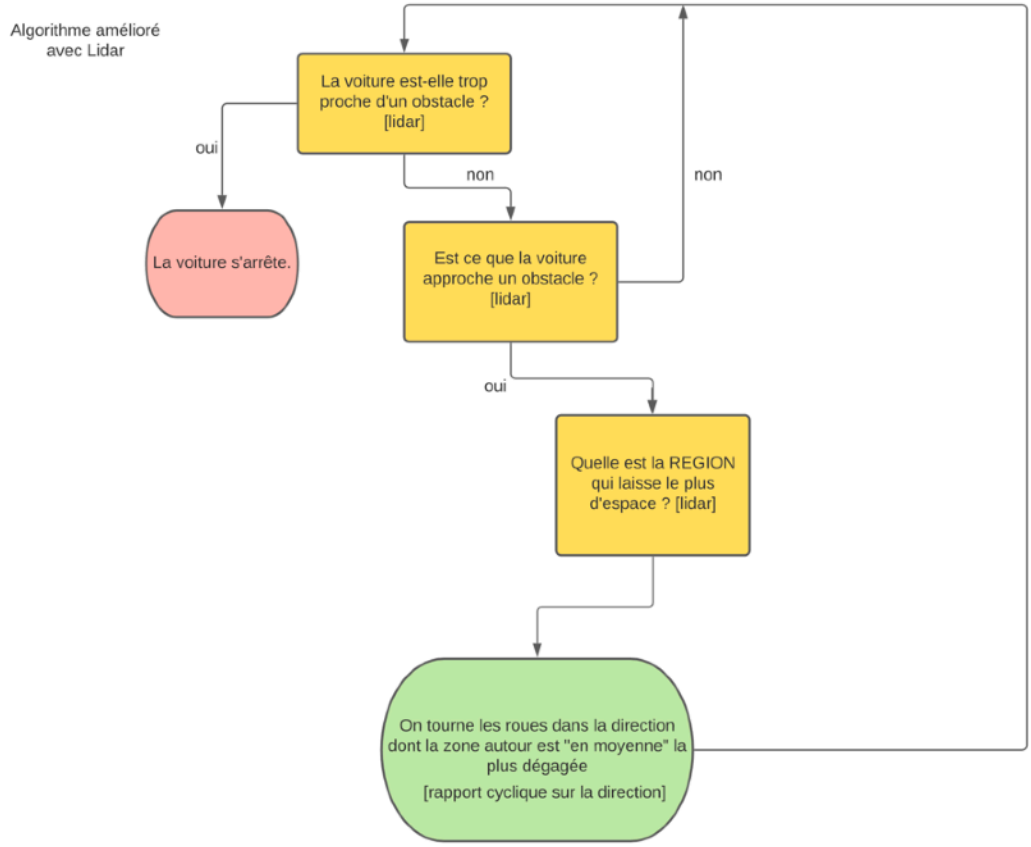


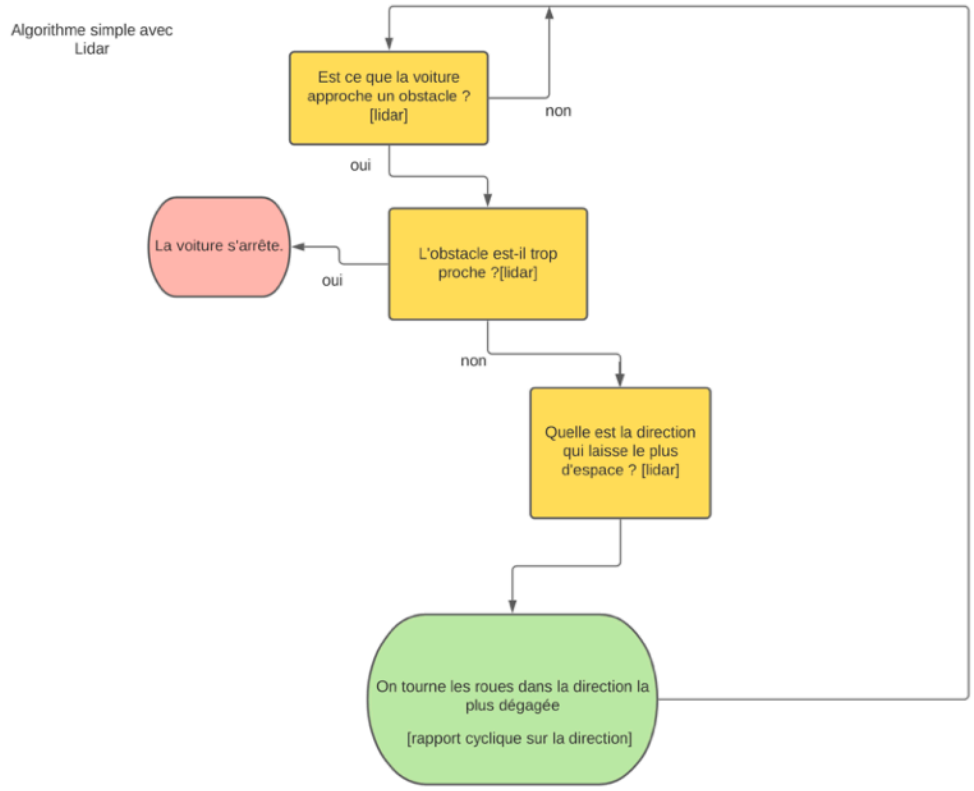
Schéma du circuit avec Lidar

Algorithmes

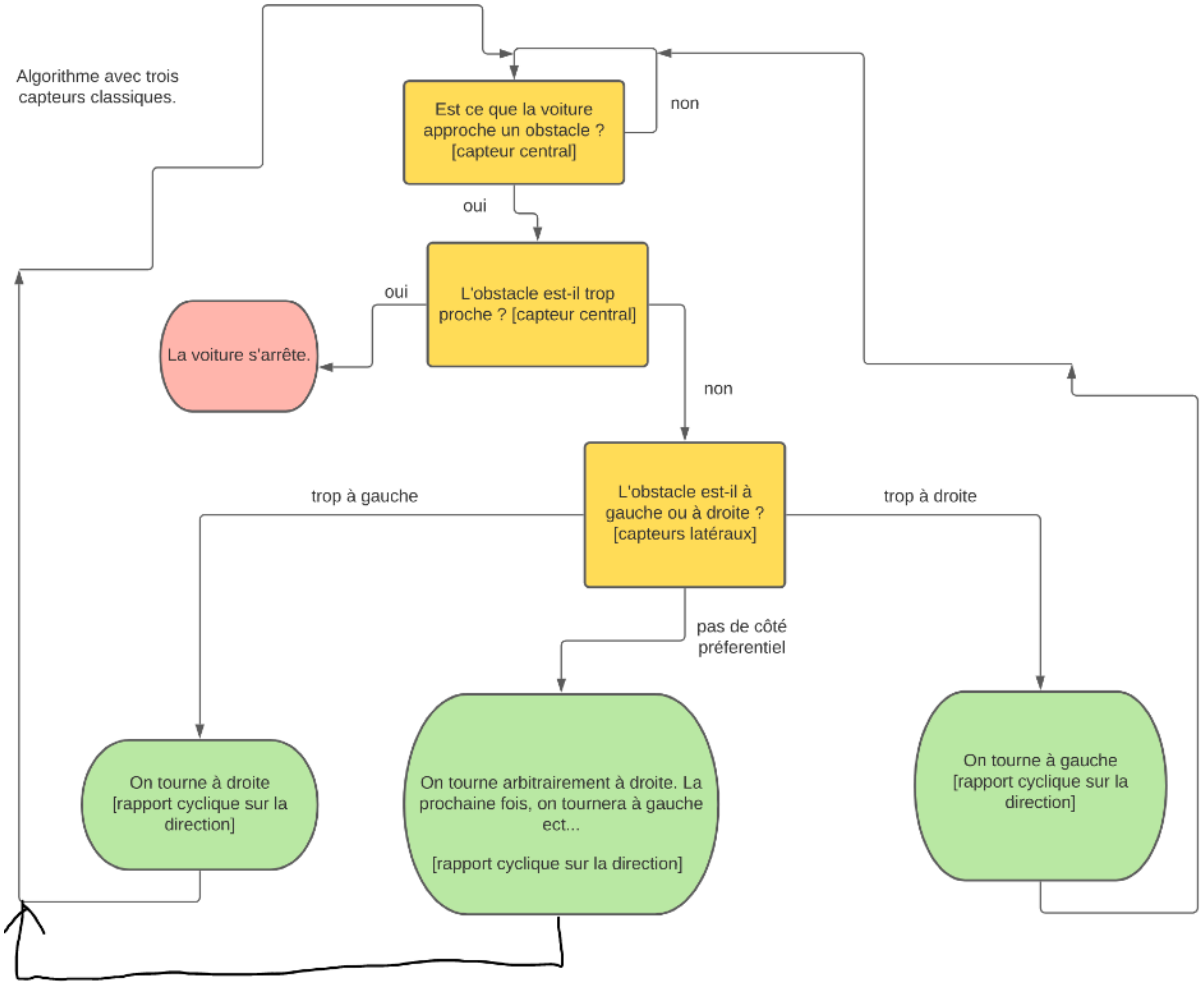
Pilotage avec trois détecteurs



Pilotages avec lidar simple



Pilotage avec lidar amélioré



SOLUTION RETENUE :

Algorithme amélioré avec Lidar. La partie essentielle de cet algorithme est le moyennage. Contrairement à l'algorithme naïf qui choisissait une direction préférentielle. Ici, nous choisissons une région préférentielle. Plus concrètement, on affecte à chaque direction(i.e.chaque case du tableau des valeurs du Lidar) la distance moyenne dans une zone d'étendue 10° autour de la direction en question (i.e.la valeur moyenne des 10cases autour de cette case).
Avantages notables:
-Permet de donner des valeurs pertinentes pour des angles non mesurés par détection.
-Réduit le poids des valeurs erronées
-Evite la voiture de trop osciller (car la direction préférentielle varie chaotiquement contrairement à la région préférentielle)
-Donne une cartographie plus pratique des lieux (typiquement, la voiture n'essaiera pas de passer entre les deux pieds d'une chaise s'ils sont trop proche, malgré le trou entre les deux pieds)

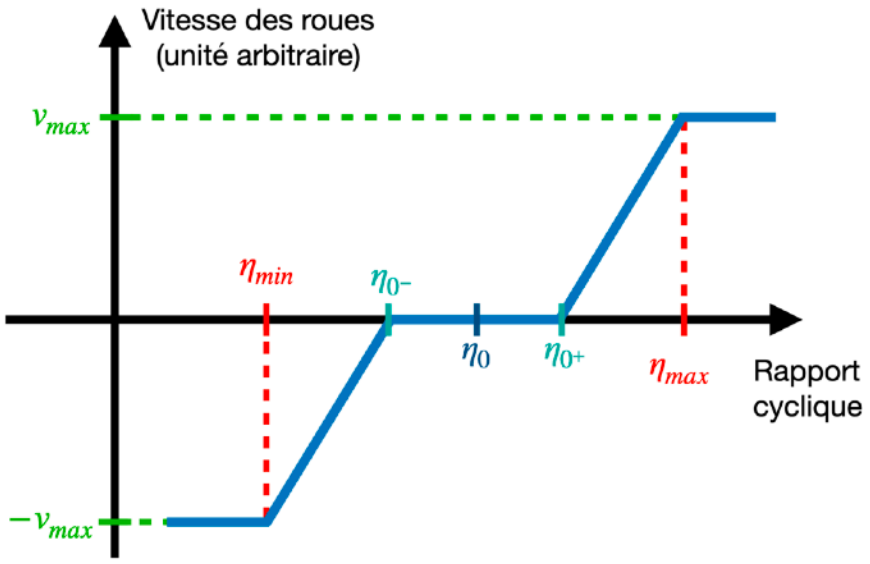
Finalement, nous avons remarqué que cet algorithme donne effectivement une conduite qui évite une plus grande variété d'obstacles et surtout la conduite est grandement fluidifié.La trajectoire est claire.
(Pour les codes voir annexe A)

Tests et validation

Notre travail repose sur plusieurs instruments et algorithmes.
Il a donc été essentiel de déterminer les modèles théoriques des instruments mais surtout d'étudier la pertinence de ceux-ci. Nous avons également dû valider expérimentalement nos algorithmes.
Voici les tests que nous avons effectuées dans notre travail :

1) RAPPORT CYCLIQUE DE VITESSE

Quel est le lien entre le rapport cyclique de l'alimentation du moteur et la vitesse effective des roues ?
Le constructeur suggère une relation linéaire. Pour vérifié cela, nous avons placé la voiture sur un support de sorte à ce que les roues ne touche pas le sol, on modifie le rapport cyclique du signal de période 20 ms.
Expérimentalement, on remarque qu'autour de η_0 , il y a un plateau sur lequel la vitesse reste nulle. Ce plateau doit-être pris en compte dans nos calculs. Au-delà de ce plateau, la relation est effectivement linéaire (figure ci-dessous)

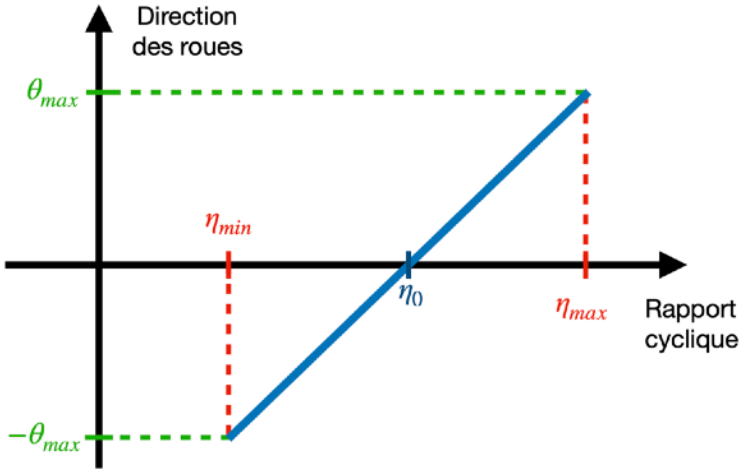


Allure de la vitesse des roues en fonction du rapport cyclique

On a déterminé : $\eta_0 = 0{,}075$, $\eta_{0+} = 0{,}078$, $\eta_{0-} = 0{,}072$ et $\eta_{max} = 0{,}85$
La vitesse dépend de l'état de la batterie et de la voiture donc nous l'avons pas déterminé
Nous avons ainsi pu choisir vitesse convenable de notre voiture autonome pour les tests.

2) RAPPORT CYCLIQUE DE DIRECTION

Quel est le lien entre le rapport cyclique de l'alimentation du servomoteur et la direction des roues ?
Le constructeur suggère une relation linéaire. Expérimentalement, sur la zone mécaniquement acceptable (pas de frottement des roues sur le châssis), la relation peut effectivement s'approximer par une relation linéaire (figure suivante)
On a trouvé : $\eta_0 = 0{,}065$ et $\eta_{max} = 0{,}070$ avec $\theta_{max} = 15°$



Allure de la direction des roues en fonction du rapport cyclique

Pour un châssis différent, il faut revérifier les relations entre rapport cyclique et vitesse et direction.

3) TENSION DES CAPTEURS CLASSIQUES

La tension des capteurs est inversement proportionnelle à la distance à l'obstacle. Nous vérifions ceci sur un oscilloscope en faisant bouger un obstacle vers le capteur (voir annexe C). On constate que la tension est toutefois légèrement inférieur à celle de la fiche technique.
On remarque qu'il existe une distance maximale de détection mais également une distance minimale en dessous de laquelle, le résultat n'est plus exploitable. Sur cette plage de distance, tension et distance sont effectivement inversement proportionnelles.
Durant cette phase de test, nous avons juste vérifier qualitativement le comportement du capteur. Pas de modèle théorique plus précis. (car pas utile, la distance de freinage/ tension seuil dépend de trop de facteurs tel que la vitesse et le poids du véhicule, il faudra de toute façon la déterminer empiriquement)

4) *DISTANCE D'ARRÊT (PEU IMPORTE LE CAPTEUR)*

Nous souhaitions un arrêt d'urgence pour un objet situé à 30 cm.
Pour la distance de freinage, le protocole est de choisir une valeur de distance d'arrêt, lancer l'algorithme, rapprocher lentement un écran jusqu'à ce que la voiture s'arrête (comme sur la photo). Il faut ensuite noter la distance et vérifier que ce soit la distance souhaité.

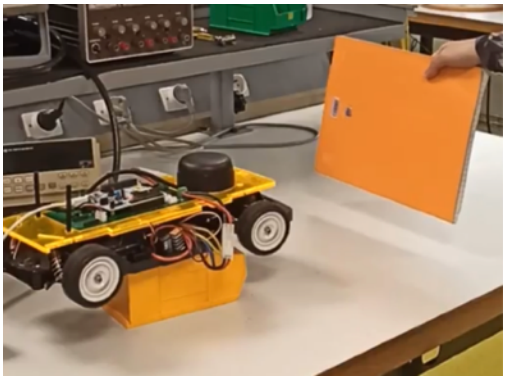


Photo du test d'arrêt

Ensuite lors des tests sur le terrain, d'affiner la valeur de l'angle de détection et de distance pour permettre à la voiture de d'avancer dans des endroits suffisamment étroit sans qu'elle heurte d'obstacle.

5) CARTOGRAPHIE DU LIDAR

C'est une question essentielle qui mérite d'être posée. La cartographie du Lidar est-elle fiable ?
Pour la cartographie, à l'aide de la fonction printf et du logiciel teraterm, on peut afficher la valeur du tableur renvoyé par le lidar ainsi que l'angle associé et regarder si les valeurs sont cohérente par rapport à l'environnement du lidar. Le tableau possède des valeurs nulles pour certains angles, ce qui correspond aux angles pour lesquelles la détection ne s'est pas effectué. Sinon les valeurs sembles cohérentes et proportionnelles à la distance d'observation. Pour une distance de 30cm, la valeur vaut 600.
On a donc la relation : $D_{lidar} = k \times D$ avec $k = 20\,cm^{-1}$
Néanmoins, malgré les défauts du Lidar, les tests sur le terrain ont montré que la détection permettait d'éviter les obstacles plus facilement qu'avec les trois capteurs.

6) SENSIBILITÉ DU LIDAR À LA LUMIÈRE ET AUX MATÉRIAUX RÉFLÉCHISSANTS

Le constructeur avertit dans la notice que le Lidar réagit moins bien en présence de surfaces réfléchissantes ou zones trop lumineuses. Comme les données brutes était difficiles à interpréter, nous avons testé directement en plaçant la voiture face à une porte réfléchissante ou dans une zone pleine de lumière. Nous remarquons effectivement que parfois la voiture percute un obstacle.
C'est donc un défaut inhérent au capteur.

7) ALGORITHMES.

Au-delà des erreurs de syntaxes ou de logique, le seul moyen de vérifier nos algorithmes était l'expérience. C'est ainsi que nous avons pu corriger certains algorithmes grâce aux sur support décrit au début et sur terrain.
Par exemple, dans la configuration des 3 capteurs, en plaçant des obstacles de manière astucieuse, on a pu cibler précisément les lacunes de l'algorithme (la voiture tourne trop à gauche, elle détecte trop loin à droite, etc....)
Finalement, la grande partie de notre travail a été expérimentale. C'est la validation expérimentale des 7 points cités précédemment qui permet le résultat final.

Planning

SÉANCE 1 :
-Découverte du sujet : La Voiture Autonome

SÉANCE 2 :
-Ecriture du cahier des charges, du planning et du schéma bloc
-Première pose au sol réussi mais la voiture ne roule pas droit

SÉANCE 3 :
-Programme d'arrêt et d'accélération (programme de démarrage)
-trouver le rapport cyclique pour lequel on roule droit
Début éviter un obstacle.

SÉANCE 4 :
-Eviter obstacles avec capteurs, méthode binaire

SÉANCE 5 :
-Eviter obstacles avec un Lidar, méthode simple

SÉANCE 6 :
-Audit
-Éviter les obstacles avec le Lidar de manière plus intelligente

SÉANCE 7 :
-Rattraper au mieux la trajectoire de la voiture dans un circuit aléatoire.

SÉANCE 8 :
-Présentation du projet.

Difficultés et travail d'équipe

En général, pour le travail d'équipe, nous avons divisé le groupe en deux duos. Un premier duo travaillait sur la partie algorithmique et les codes ainsi que de recherches techniques. Le second duo était chargé de réaliser le montage et les tests. Le travail d'équipe s'est généralement bien passé avec une répartition adéquat du travail entre chaque membre de l'équipe.
Les difficultés sont de l'ordre électronique et algorithmique. Au niveau électronique, nous avons eu des problèmes de batterie déchargée, ou de modes. Parfois, les fils ne fonctionnaient pas. De même, l'ensemble des coefficients déterminés expérimentalement dépendent du châssis utilisé, lorsque le châssis change, il faut recalculer ces coefficients.
Une autre difficulté est la transition entre capteur classique et LiDar. Il nous a fallu revoir nos codes étant donné que les détections et les informations reçues diffèrent. Le LiDar à l'avantage de détecter sur une plus grande distance, et de fournir des données sur 360°. Néanmoins, la mise à jour des données est plus longue, la distance minimale de détection est plus grande et sur un tour, les valeurs pour certains angles ne sont pas calculées. Nous avons alors utilisé des codes du Lense. De ce fait, la compréhension du rôle de certaines fonctions nous a posé problème.
Le plus difficile est de savoir si le non-fonctionnement du système est dû au code, au montage ou au deux.
Néanmoins, le planning a été respecté excepté que nous avions prévu de travailler sur le LiDar plus tard au début du projet et qu'à la fin nous n'avons pas testé notre algorithme d'adaptation des vitesses (voir annexe B).

Annexes

A) Codes

1) Programmes de changement de vitesse et direction

```
#include "mbed.h"

PwmOut direction(PB_13);
PwmOut moteur(PC_9);


int main()
{
    direction.period_ms(20);
    direction.write(0.065);
    moteur.period_ms(20);
    moteur.pulsewidth_us(1500);
    wait(4);

    double rapport_cyclique = 0.075;


    direction.write(0.065);
    moteur.write(0.078);
    while (true) {

    }
}
```

2) Code avec détecteurs classiques

```
#include "mbed.h"

PwmOut direction(PB_13);
PwmOut moteur(PC_9);
AnalogIn capteur1(PC_3); //capteur central
AnalogIn capteur2(PC_0); // capteur de gauche
AnalogIn capteur3(PC_2); // capteur de droite
Serial pc(USBTX, USBRX);

int main()
{
    direction.period_ms(20);
    direction.write(0.0637);
    moteur.period_ms(20);
    moteur.pulsewidth_us(1500);
    pc.baud(115200);
    wait(2);

    moteur.write(0.079);
    double tension1;
    double tension2;
    double tension3;
    tension1=capteur1.read();
    tension2=capteur2.read();
    tension3=capteur3.read();
    direction.write(0.0637);
    direction.write(0.0637);


    while ((tension1<0.55)&&(tension2<0.55)&&(tension3<0.55)) { // condition d'arrêt d'urgence
    tension1=capteur1.read();
    tension2=capteur2.read();
    tension3=capteur3.read();
    if ((tension2>tension3)&&(tension2>0.08)){
        direction.write(0.071); // tourner à droite
        wait_ms(20);}

    else if ((tension3>tension2)&&(tension3>0.08)){
        direction.write(0.0564); // tourner à gauche
        wait_ms(20);}

    else {
        if(tension1>0.08){
            direction.write(0.071); // tourner à droite par défaut
            wait_ms(20);}
        else {
            direction.write(0.0637); // rester droit lorsqu'il n'y a pas de raison de tourner
            wait_ms(20);}

    }
    }

    moteur.pulsewidth_us(1500); // la voiture s'arrête

}
```

3) Fonction principale de pilotage avec lidar

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */

#include "mbed.h"
#include "rplidar.h"

#define BLINKING_RATE_MS        500
#define NB_DATA_MAX             20
#define AFF_DATA                0

PwmOut direction(PB_13);
PwmOut moteur(PC_9);



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             moy[360]={0};
char            tour_ok = 0;
char            trame_ok = 0;

Serial              pc(USBTX, USBRX, 115200);
DigitalOut          led(LED1);
/* DEBUG OUTPUTS*/
DigitalOut          debug_data(D10);
DigitalOut          debug_tour(D9);
DigitalOut          debug_out(D7);
DigitalOut          data_ok(D5);
DigitalOut          data_ok_q(D4);
/* DEBUG OUTPUTS*/

Serial              lidar(A0, A1, 115200);
PwmOut              rotation(D14);

struct lidar_data   ld_current;


/** MAIN FUNCTION
 */
int main()
{
    direction.period_ms(20);
    direction.write(0.0637);
    moteur.period_ms(20);
    moteur.pulsewidth_us(1500);
    wait(2);


    int nb_tour = 0;
```

```
wait_s(3.0);
rotation.period(1/25000.0);
rotation.write(0.4);
wait_s(2.0);
pc.printf("\r\nLIDAR Testing\r\n");
lidar.attach(&IT_lidar);
wait_s(1.0);
pc.printf("\r\nLIDAR OK\r\n");


getHealthLidar();
getInfoLidar();
getSampleRate();
// Start a new scan
startScan();
// Infinite Loop
int maxDistance, maxAngle, minDistance, minAngle;
double angle_calc;
int flag;
flag=1;

 moteur.write(0.07813);
//int i;

while (flag==1) {
    if(trame_ok){
        debug_tour = !debug_tour;
    }

    //int i;
    //for (i=0;i<360;i++){
        //pc.printf("%d; ", distance_scan_old[i]);}
    //pc.printf("Tour1 : %d\t\n",tour_ok);
    //wait(0.1);
    //int s;


    if(tour_ok >= 6){
        //pc.printf("Tour2 : %d\t\n",tour_ok);
        tour_ok = 0;
        moyennage(distance_scan_old,360,moy);              //transforme le tableau distance_scan_old en
                                                             un tableau de ses moyennes sur 10 valeurs
        findMax(moy, 360, &maxDistance, &maxAngle);
        //print_int("A", maxAngle);
    for (int s=0;s<360;s++){
    pc.printf("%d; ", moy[s]);}



        if (maxAngle<50) {
            pc.printf("%d\t\n", maxAngle);
            angle_calc = maxAngle*0.0003+0.0637;
            pc.printf("%lf\t\n", angle_calc);
            direction.write(angle_calc);
        }

        else {
            angle_calc = ((maxAngle-360)*0.0002)+0.0637;          //changement de direction
            direction.write(angle_calc);
        }
    }
```

```
    flag=stop(distance_scan_old,360);        //condition d'arrêt

    /*
    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();
    */
}
direction.write(0.065);        // vitesse et direction d'arrêt si on sort de la boucle while
moteur.write(0.075);
}
```

Remarque : Une partie du code provient du Lense.

4) Bibliothèque utilisée pour la fonction principale

```
#include "mbed.h"
#include "rplidar.h"
#include <cstdio>

void print_int(const char *name, int ki){
    pc.printf("\t %s = %d\r\n", name, ki);
}

void print_data(const char *name, char *datai, int sizedata){
    pc.printf("\t %s = ", name);
    for(int i = 0; i < sizedata; i++){
        pc.printf("%x ", datai[i]);
    }
    pc.printf("\r\n");
}

void wait_s(float sec){
    wait_us(sec*1000000);
}

void findMax(int *int_data, int size_data, int *value, int *indice){      //cherche le maximum parmi size_data
    *value = 0;
    *indice = 0;
    for(int k = 0; k < size_data; k++){
        if ((int_data[k] > *value)&&((k<25)||(k>335))){
            *value = int_data[k];
            *indice = k;
        }
    }
}


void findMin(int *int_data, int size_data, int *value, int *indice){      //cherche le minimum parmi size_data
    *value = 0;
    *indice = 0;
    for(int k = 0; k < size_data; k++){
        if ((int_data[k] < *value)&&((k<25)||(k>335))){
            *value = int_data[k];
            *indice = k;
        }
    }
}

void IT_lidar(void){
    char data, startt, nostartt;
    debug_data = 1;
    data = lidar.getc();

    if(scan_ok){
        switch(data_scan_nb % 5){
            case 0 :
                data_ok = 0;
                data_ok_q = 0;
                if (((data&0X03)==0X01) || ((data&0X03)==0X02)) {
                    trame_ok=1;
                } else {
                    trame_ok=0;
                }
```

```
            ld_current.quality = data >> 2;
            startt = data & 0x01;
            nostartt = (data & 0x02) >> 1;
            if((data & 0x01) == 0x01){
               debug_out = 1;
               for(int k = 0; k < 360; k++){
                  distance_scan_old[k] = distance_scan[k];
                  distance_scan[k] = 0;
               }
               debug_out = 0;
               tour_ok++;
            }
            if(startt == nostartt)      data_scan_nb = 0;
            break;
         case 1 :
            if((data&0x01) == 0){
               trame_ok = 0;
               data_scan_nb = 0;
            }
            // angle_q6[6:0] / 64 and check (degre)
            ld_current.angle = data >> (1 + 6);
            // check ?
            break;
         case 2 :
            // angle_q6[14:7] / 64 (degre)
            ld_current.angle += data << 1;
            break;
         case 3 :
            // distance_q2[7:0] / 4 (mm)
            ld_current.distance = data >> 2;
            break;
         default :
            // distance_q2[15:8] / 4 (mm)
            ld_current.distance += data << 6;
            if(trame_ok){
               distance_scan[ld_current.angle%360] = ld_current.distance;
               data_ok = 1;
               if(ld_current.quality > 0) data_ok_q = 1;
            }
      }
      data_scan_nb++;
   }
   else{
      data_ok = 0;
      data_ok_q = 0;
      received_data[data_nb] = data;
      data_nb++;
   }
   debug_data = 0;
}


void sendResetReq(void){
   mode = LIDAR_MODE_RESET;
   char data[2] = {0xA5, 0x40};
   lidar.putc(data[0]);
   lidar.putc(data[1]);
   wait_us(10000);
}

void getHealthLidar(void){
```

```
    stopScan();
    mode = LIDAR_MODE_HEALTH;
    char data[2] = {0xA5, LIDAR_MODE_HEALTH};
    lidar.putc(data[0]);
    lidar.putc(data[1]);
    data_nb = 0;
    while(data_nb != (NB_BYTE_HEALTH_REQ + NB_BYTE_HEALTH_RESP)){__nop();}
    //print_data("Health", received_data, (NB_BYTE_HEALTH_REQ + NB_BYTE_HEALTH_RESP));
    if(received_data[7] == 0)   pc.printf("\r\nGOOD\r\n");
    else   pc.printf("\r\nBAD\r\n");

}

void getInfoLidar(void){
    stopScan();
    mode = LIDAR_MODE_INFO;
    char data[2] = {0xA5, LIDAR_MODE_INFO};
    lidar.putc(data[0]);
    lidar.putc(data[1]);
    data_nb = 0;
    while(data_nb != (NB_BYTE_INFO_REQ + NB_BYTE_INFO_RESP)){__nop();}
    print_data("Info", received_data, (NB_BYTE_INFO_REQ + NB_BYTE_INFO_RESP));
}

void getSampleRate(void){
    stopScan();
    mode = LIDAR_MODE_RATE;
    char data[2] = {0xA5, LIDAR_MODE_RATE};
    lidar.putc(data[0]);
    lidar.putc(data[1]);
    data_nb = 0;
    while(data_nb != (NB_BYTE_RATE_REQ + NB_BYTE_RATE_RESP)){__nop();}
    print_data("Rate", received_data, (NB_BYTE_RATE_REQ + NB_BYTE_RATE_RESP));
    int usRate = (received_data[8] << 8) + received_data[7];
    print_int("Standard (uS) ", usRate);
    usRate = (received_data[10] << 8) + received_data[9];
    print_int("Express (uS) ", usRate);
}

void startScan(void){
    stopScan();
    mode = LIDAR_MODE_SCAN;
    char data[2] = {0xA5, LIDAR_MODE_SCAN};
    lidar.putc(data[0]);
    lidar.putc(data[1]);
    data_nb = 0;
    data_scan_nb = 0;
    while(data_nb != (NB_BYTE_SCAN_REQ)){__nop();}
    scan_ok = 1;
}

void stopScan(void){
    mode = LIDAR_MODE_STOP;
    scan_ok = 0;
    char data[2] = {0xA5, LIDAR_MODE_STOP};
    lidar.putc(data[0]);
    lidar.putc(data[1]);
}

int stop(int *int_data, int size_data){        // flagb dit si la condition d'arrêt est vérifié
int flagb;
```

```
flagb = 1;
for(int k = 0; k < size_data; k++){
        if ((int_data[k] <600)&&(int_data[k] !=0)&&((k<25)||(k>335))){
            flagb=0;
            break;}        //si la condition d'arret est vérifié pour un point, on arrête la boucle
    }
    return(flagb);
}

void moyennage (int *int_data, int size_data, int *moy){

    int compteur;
    int h;
    for(int i = 0; i < size_data; i++){
    moy[i]=0;
    compteur=0;
    for (int j =0; j< 10; j++) {                // calcule la moyenne sur 10 valeurs
        if (i+j<5){
            h=360+i+j-5; }
        if (i+j>364){
            h=i+j-5-360; }
        else {
            h=i+j-5;}
        if (int_data[h] != 0){
        moy[i]= moy[i] + int_data[h];
        compteur=compteur+1;}
        }
    if (compteur!=0){
        moy[i]=int(moy[i]/compteur);}
    }
    }
```

Remarque : Une partie du code provient du Lense.

B) Adaptation de vitesse

1) Principe

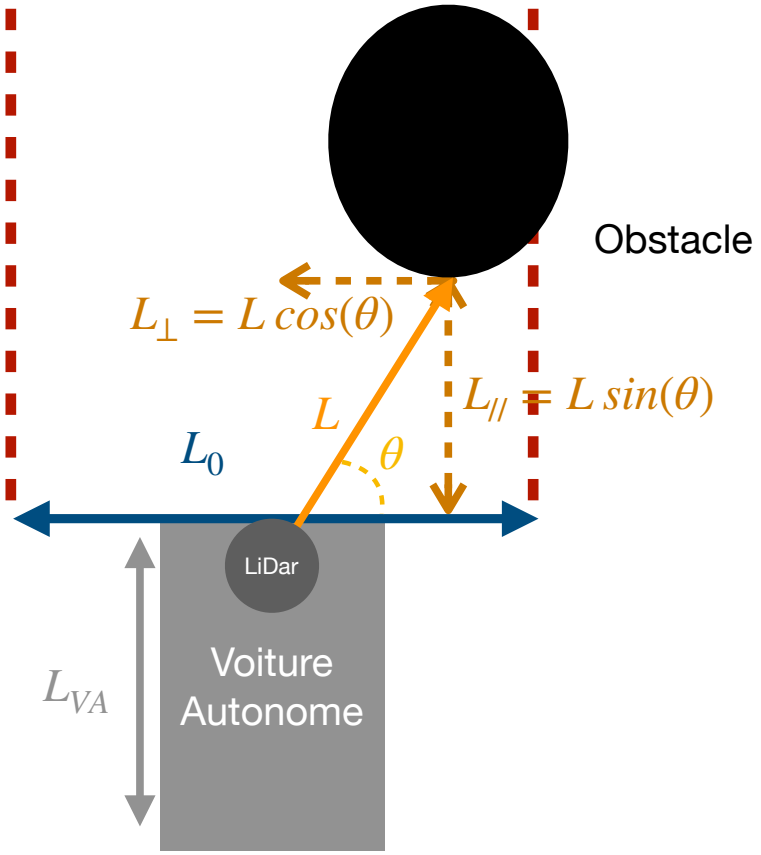


Schéma de la détection pour l'adaptation de vitesse

Le programme d'adaptation de vitesse calcule la plus petite distance $L_{//}$ de l'obstacle se trouvant à une distance $L_\perp$ inférieur $\frac{L_0}{2}$.
Il adapte ensuite la vitesse de manière linéaire et croissante en fonction de $L_{//,min}$.
Se restreindre sur une zone L_0 autour de le direction de la voiture permet de s'assurer que la voiture aura toujours une distance $\frac{L_0}{2}$ pour tourner à vitesse maximale et évite de regarder les obstacles suffisamment éloigné de sa trajectoire pour qu'elle puisse tourner.
Le choix de $L_{//}$ pour adapter la vitesse s'explique par le fait que tout objet dans son champ, qu'il soit au centre ou sur un des bords, ne doit pas gêner le changement de direction de la voiture.

Les fonctions sinus et cosinus sont implémentés à l'aide de leur développement limité.

2) fonctions de code

```
double cos(int y) {                 //développement limité du cosinus à l'ordre 40
int cosinus;
int u;
int x;
x=(2*PI*y)/360;
cosinus=1;
u=1;
for (int i=1; i<=20;i++){
u=-(x*x*u)/((2*i-1)*(2*i));
cosinus=cosinus+u; }
return(cosinus);
}

double sin(int y) {                 //développement limité du cosinus à l'ordre 41
int sinus;
int u;
int x;
x=(2*PI*y)/360;
sinus=x;
u=x;
for (int i=1; i<=20;i++){
u=-(x*x*u)/((2*i+1)*(2*i));
sinus=sinus+u; }
return(sinus);
}


double speed(int *int_data, int size_data){
    double speed;
    int value;
    int indice;
value = 20 000;     //valeur maximale que peut prendre L
    indice = 1;
    for(int k = 0; k < size_data; k++){
            if (int_data[k]*sin(k)<600&&((int_data[k]*cos(k)) < (value*indice))&&(int_data[k] !=0) &&((k<80)||
(k>280))){
                value = int_data[k];
                 indice = cos(k);
            }
    }
speed=value*indice*0.00005+0.075;
}
```

C) Relation entre le distance et la tension d'un capteur

(d'après la fiche technique du capteur sharp gp2y0a21yk)