université
PARIS-SACLAY

IUT DE CACHAN

Rapport d'Etude et Réalisation

Année Universitaire 2021-2022, Semestre 3

SAPH TEAM RACING

Course de voitures autonomes

Présenté par : Antoine Beauvarlet,

Axel Manadi, Guillaume Dumesnil

Janvier 2022

Sous la direction de M. Martincic et M. Bellanger

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de notre projet et qui nous ont aidés lors de la rédaction de ce rapport.

Nous voudrions dans un premier temps remercier M. Martincic, professeur d'Etude et Réalisation à l'IUT de Cachan, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à la réussite de ce projet.

Nous remercions notre professeur de Culture et Communication M. Bellanger qui nous a guidé dans la rédaction du rapport.

Également, nous tenons à remercier M. Ardillier ainsi que tout le corps enseignant de l'IUT de Cachan pour avoir assuré le bon déroulement de nos différentes séances et permis, à terme, de mener à bien ce projet semestriel.

Enfin, merci à nos camarades de promotion avec lesquels nous avons échangé sur nos projets respectifs.

Résumé

Notre projet de troisième semestre consiste à créer une **voiture électrique autonome** capable de se déplacer sans intervention humaine.

Pour ce projet, nous avions besoin de plusieurs **cartes électroniques** que nous avons élaborées avec pour objectif de tenir dans le châssis du véhicule. Un **moteur brushless** relié aux quatre roues motrices ainsi qu'un **servomoteur** assurent le déplacement de la voiture. Tous les composants ont été assemblés sur un châssis de **Lancia Delta de modélisme**.

Pour rendre le véhicule autonome, nous l'avons équipé de **capteurs infrarouges et à ultrasons** ainsi que d'un **LiDAR**. Par ailleurs un **module Bluetooth** est connecté au microcontrôleur afin d'assurer une transmission sans fil et de collecter les données perçues en temps réel.

Pour la conception et la réalisation de ce projet, nous avons eu besoin de passer par plusieurs logiciels : **Altium** pour la conception des cartes électroniques, **Mbed Studio et MBed.org** pour la programmation des microcontrôleurs **DISCO F746-NG** ainsi que le logiciel **Octave** pour la visualisation des données en temps réel.

Sommaire

Introduction

À l'IUT de Cachan dans le département GEII2 les étudiants de S3 réalisent un projet semestriel qu'ils choisissent. Ce projet semestriel a pour objectif de mettre en application les connaissances acquises en génie électrique (TEC*[1], SA*, CE*), physique, mathématiques et programmation (IENA*).

Notre groupe, constitué de trois élèves, s'est donné comme objectif de créer une voiture autonome pour participer à une course organisée en avril 2022 où concourent diverses écoles d'ingénieurs et universités. Dans ce contexte notre problématique est d'obtenir un véhicule capable d'interagir avec son environnement sans opérateur humain, et sans collision avec les autres véhicules ou obstacles.

Ce projet de voiture autonome nous a permis de travailler en petit groupe avec des sous-tâches attribuées à chacun des membres. Cela permet d'appréhender concrètement l'électronique, la programmation, la mécanique et surtout le travail en équipe.

Travailler en groupe permet de se répartir les tâches selon nos spécialités, afin de respecter les délais et d'être plus efficace. De plus nous sommes obligés de rendre compte au groupe et respecter nos engagements.

Bien entendu durant tout ce projet notre professeur peut être sollicité, afin de nous accompagner et de nous faire progresser. Au début du semestre, chaque groupe d'étudiant du semestre 3 a préparé une soutenance afin de présenter les objectifs de leur projet et ont livré un diagramme de Gantt* prévisionnel.

Contrairement aux semestres précédents, ce projet nous plonge dans une situation presque professionnelle : définir et répondre à un cahier des charges tout en prévoyant et respectant les délais de livraison.

Dans un premier temps, nous présenterons la mécanique de la voiture, puis nous étudierons les différentes cartes électroniques réalisées. Ensuite, nous examinerons comment la réception des données constituant l'environnement est réalisée puis comment l'interprétation de celles-ci s'effectue. Enfin nous conclurons.

[1] Les mots accompagnés d'une * sont des mots définis dans le lexique à la fin de ce document.

Présentation du projet

Lors du semestre 3 nous avons pu choisir notre projet : une course de voitures autonomes. Ce projet, organisé depuis longtemps, consiste à récupérer le travail que le groupe de l'année dernière a produit et le continuer. Ainsi notre groupe a pu, en s'appuyant sur le rapport du groupe précédent, améliorer, repenser les concepts de programmation et recommencer certaines parties.

Nous sommes partis d'un châssis de Lancia Delta de modélisme, imposé par le règlement de la course, que nous avons adapté pour recevoir nos capteurs. Le travail du groupe précédent ayant concerné l'assemblage du véhicule, cela nous a permis de nous focaliser sur l'instrumentation de celui-ci.

Nous avons dû concevoir des cartes imprimées : carte de puissance et carte d'acquisition. Puis mettre en œuvre des capteurs afin de modéliser l'environnement et générer une réflexion sur les comportements de la voiture.

Ces 3 grandes tâches réparties sur le groupe ont été effectuées plus ou moins de façon asynchrone. Les deux cartes imprimées sur lesquelles nous reviendrons plus tard, répondent aux besoins suivants :

- Carte d'acquisition : contrôler les capteurs et modéliser l'environnement.
- Carte de puissance : générer une réponse au modèle et piloter les moteurs.

Toutes ces étapes sont détaillées dans la suite de ce rapport et en annexes[2].

• Les règles de la course

1. Le cahier des charges

Lors de la course nous serons mis en compétition contre 5 autres véhicules autonomes. La course s'effectue en 3 tours.

[2] Afin de rendre synthétique ce document nous avons préféré placer certaines informations en annexe. De plus un repository GitHub a été mis en place pour transmettre la totalité de notre travail.

Pour y participer, nous devons nous plier au cahier des charges rédigé par les organisateurs :

- Véhicule issu d'un kit de modélisme de Tamiya TT-02.
- Découpe de la carrosserie à condition qu'elle recouvre 80% du véhicule.
- Batterie de type NiMH* de 7,2V.
- Dimensions imposées afin d'être détectable (voir la figure suivante).
- Peinture n'absorbant pas les longueurs d'onde infrarouges.
- Contact bluetooth au moins 1 fois par seconde avec un appareil immobile (arrêt du véhicule sinon).

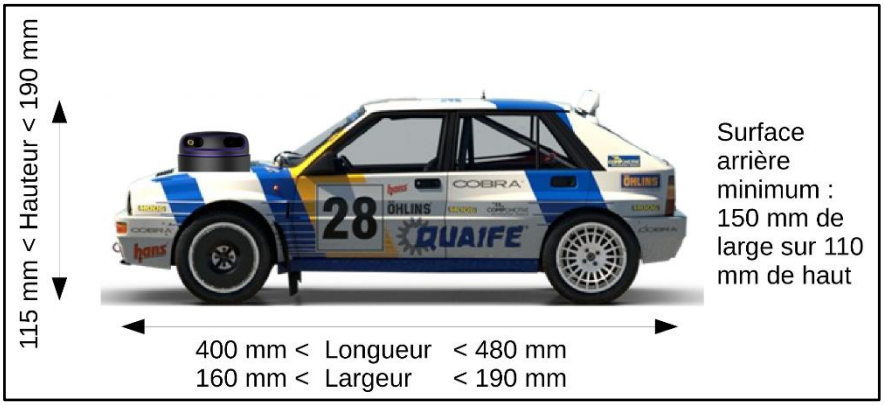


FIGURE 1: DIMENSIONS REGLEMENTAIRES DU VEHICULE

Concernant les capteurs, nous sommes libres dans le choix des capteurs tant qu'un capteur LiDAR* est l'élément principal de détection de notre véhicule.

Pour s'assurer que les véhicule répondent au cahier des charges, les organisateurs procéderont, lors de la course, à une homologation de ceux-ci.

2. Le circuit

Nous découvrirons le tracé du circuit le jour de la course. Actuellement nous sommes uniquement informés des mesures de la piste. Celle-ci aura des bordures de 20 cm de hauteur avec une largeur minimum en tout point de 50 cm. Par ailleurs elle contiendra des obstacles conformément à l'exemple donné ci-dessous.

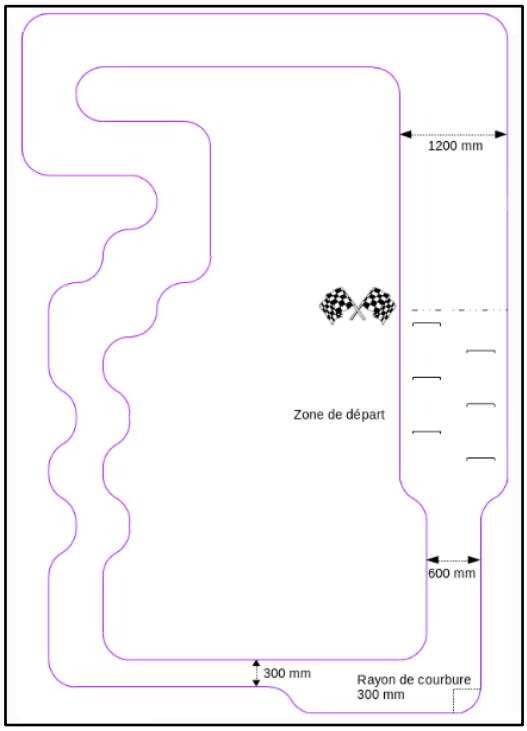


FIGURE 2: EXEMPLE DE PARCOURS FOURNI PAR LES ORGANISATEURS

Pour remporter cette compétition, il faut passer des qualifications sur une première piste, chaque véhicule la parcourant seul. Deux passages sont exigés : un premier sans obstacle puis un second avec. Chacune des qualifications nécessite deux passages successifs réussis, c'est-à-dire sans collision avec les obstacles.

La somme des temps des deux passages détermine ensuite l'ordre de passage. Chaque équipe dispose uniquement de 3 minutes pour préparer le véhicule sur la piste. Après ce délai, aucune intervention humaine n'est tolérée.

- Schéma synoptique du véhicule

Nous avons réalisé un schéma synoptique* afin de synthétiser et d'appréhender l'assemblage des composants électriques qui constituent la voiture.

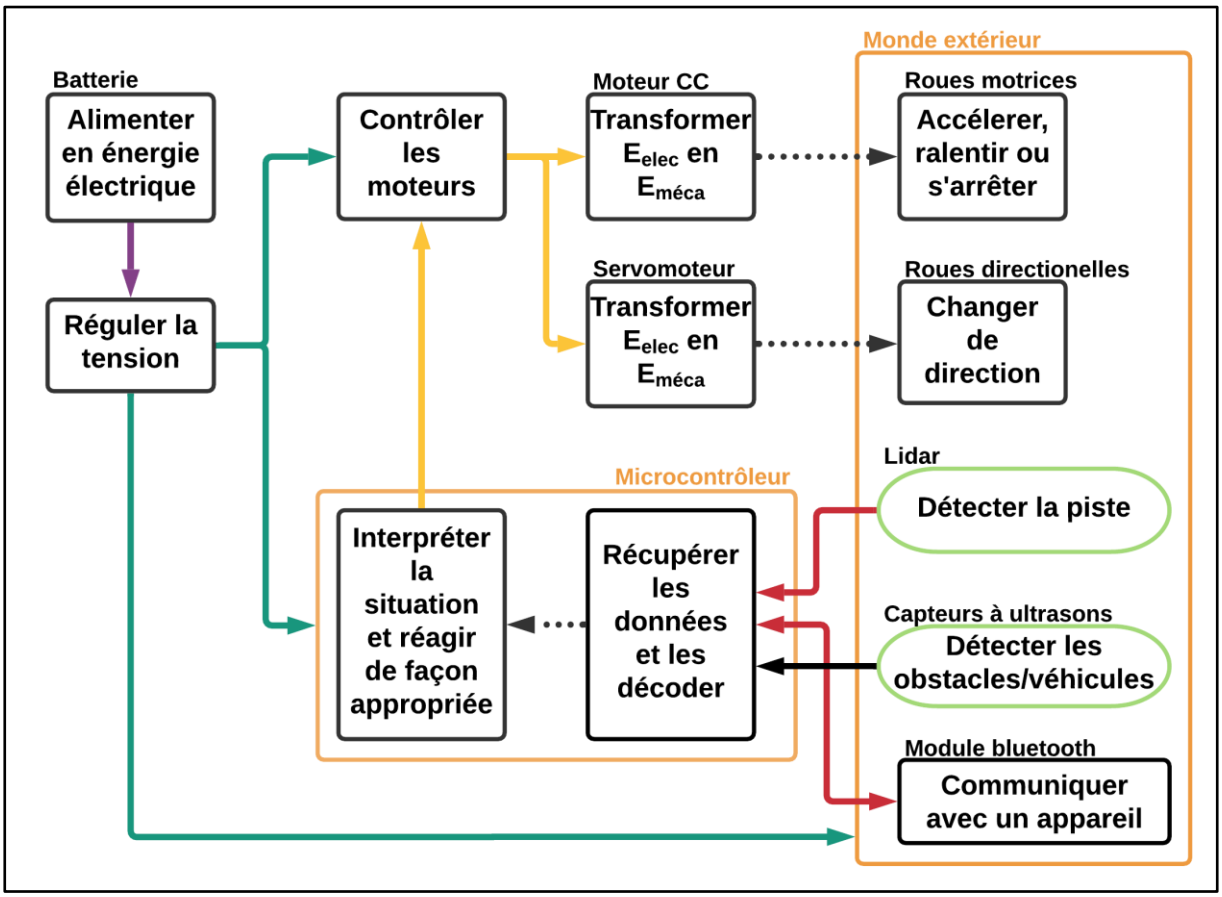


FIGURE 3: SCHEMA SYNOPTIQUE DU PROJET

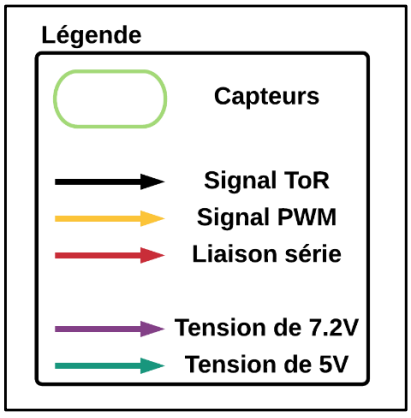


FIGURE 4: LEGENDE DU SCHEMA SYNOPTIQUE

- ## Diagramme de Gantt prévisionnel

Pour la soutenance de début de projet et répondre de manière professionnelle au cahier des charges, nous avons réalisé un diagramme de Gantt prévisionnel. Celui-ci, bien qu'optimiste, nous a permis de saisir une chose importante. En effet il faut prendre en compte qu'un projet est souvent constitué de difficultés imprévisibles tant en quantité qu'en temps nécessaire pour les résoudre. Le diagramme prévisionnel ci-dessous n'a effectivement pas pu être respecté en totalité.

Cependant nous avons effectué plus de 80 % du prévisionnel, ce que l'on peut constater sur le « diagramme de Gantt effectif » ci-dessous, qui reprend le calendrier que nous avons effectivement suivi. Nous pouvons considérer notre estimation comme une réussite.

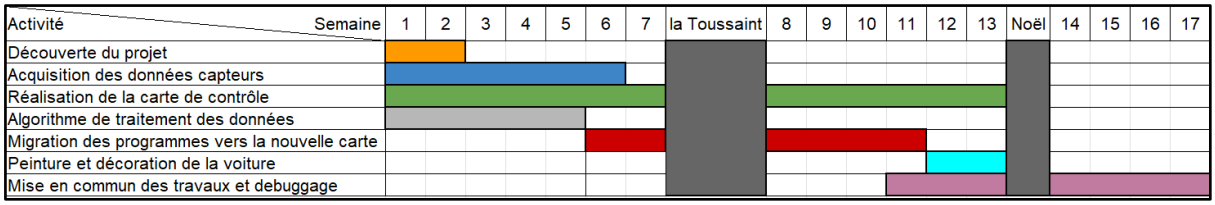


FIGURE 5: DIAGRAMME DE GANTT PREVISIONNEL

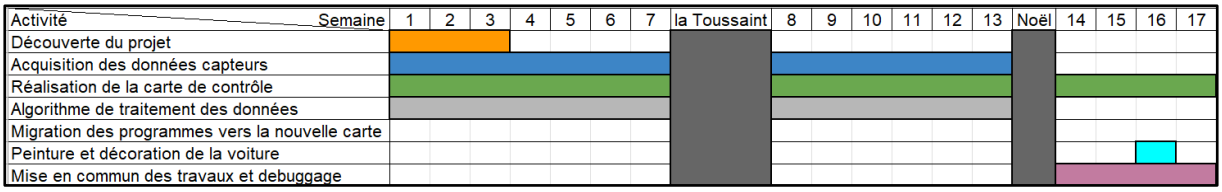


FIGURE 6: DIAGRAMME DE GANTT EFFECTIF

Conception des PCB

• Carte de puissance

Pour que le véhicule puisse se diriger durant la compétition, il faut créer une carte qui puisse communiquer des instructions aux actionneurs présents sur le véhicule :

- Un servomoteur, pour la direction
- Un moteur brushless, pour avancer ou reculer.

La carte de puissance va communiquer avec les actionneurs grâce à un signal nommé PWM (Pulse Width Modulation ou Modulation de Largeur d'Impulsion), qui est un signal alternatif TOR (tout ou rien).

Ce signal alternatif, avec des vitesses de commutation très rapides, permet de d'approximer la tension sur la liaison grâce notamment à l'inertie des moteurs : la tension ne devient plus une série de commutations « 0/1 » mais une tension moyenne ajustable par un rapport cyclique α.

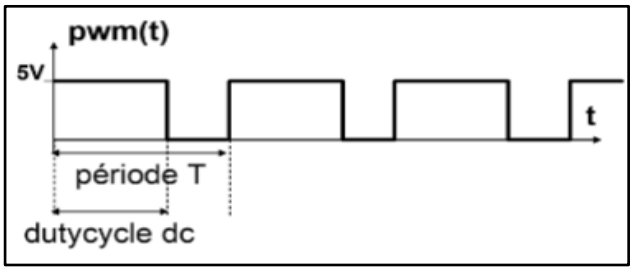


FIGURE 7: EXEMPLE DE SIGNAL PWM.

Par ailleurs cette carte devra obéir aux instructions d'une unité de traitement comme une autre carte par exemple. Il faut donc ajouter un moyen de communication.

• Carte de communication

En plus de conduire le véhicule, nous avons besoin de traiter les données fournies par les différents capteurs embarqués.

La carte communication recevra donc des informations de la part de :

- 1 capteur LiDAR (capteur principal).
- 3 capteurs à ultrasons (capteurs auxiliaires).
- 3 capteurs infrarouges (capteurs auxiliaires).

Dans cette carte nous allons téléverser un programme qui s'occupe de l'acquisition des données, c'est-à-dire décoder ce que chaque capteur transmet. Celle-ci comprendra également un algorithme décisionnel afin d'orienter le véhicule en fonction de l'environnement : prendre un virage, éviter un obstacle ou une autre voiture par exemple.

On en déduit donc un rôle de carte maître vis-à-vis de la carte de puissance, car elle lui donnera les instructions à exécuter. Enfin cette carte générera le signal PWM de commande du lidar.

FIGURE 8: CARTE KEIL UTILISEE EN TP DE IENA

Nous avons commencé à réaliser cette carte, sur le modèle de la « carte Keil » de TP. Malheureusement nous n'utiliserons pas cette dernière car elle ne sera achevée dans les temps.

• Conception des cartes

Pour concevoir ces deux cartes, nous avons utilisé le logiciel de CAO* : « Altium Designer ». Plus précisément la version « 20.2.4 » du logiciel.

FIGURE 9: LOGO DU LOGICIEL ALTIUM DESIGNER

Une licence ainsi que des manuels d'utilisation nous ont été fournis, pour faciliter la prise en main de cet outil professionnel.

Afin de trouver les composants nécessaires à la création de notre carte nous avons eu accès aux bibliothèques de composants de l'université et des projets des années précédentes.

L'utilisation de ce logiciel est enrichissante : elle permet de découvrir le fonctionnement d'un outil professionnel largement utilisé dans le monde de l'industrie.

Nous allons détailler, ci-dessous, les étapes à suivre pour réaliser un PCB.

1. Création d'un schéma

En premier lieu il faut créer un schéma : le but de cette opération est de placer les composants et de les relier par des liaisons électriques théoriques. Cela permet de savoir quels composants communiquent électriquement entre eux.

Notre professeur d'étude et réalisation, M. Martincic, nous a fourni le schéma des « cartes Keil » utilisées à l'université durant les TPs d'IENA. Après une étude du schéma nous remarquons que cette carte possède de nombreux composants superflus pour l'utilisation que nous souhaiterions en faire.

Notre première mission consistait donc à supprimer les composants inutiles (et leurs connexions) afin de simplifier le schéma et d'optimiser les performances de notre microcontrôleur.

Une fois cette étape passée nous avons dû intégrer de nouveaux composants.

Un fichier contenant les fonctions de chaque broche de notre contrôleur nous a été fourni, afin de relier chacun des composants à sa fonction associée.

Port	Utilisation Keil	Utilisation IUT	Nappe 40	Remarque
P0.0	CAN1	CAN		
P0.1	CAN1	CAN		
P0.2	COM0	Série		
P0.3	COM0	Série		
P0.4				D1
P0.5	LCD			
P0.6	LCD			
P0.7	LCD			
P0.8	LCD			
P0.9	LCD			
P0.10	USB micro AB	Train2-PB4		d2/Sd2_4
P0.11	USB micro AB	Train2-PB5		Sg1_4
P0.12	Pas dispo.	Pas dispo.		Pas dispo.
P0.13	Pas dispo.	Pas dispo.		Pas dispo.
P0.14	Pas dispo.	Pas dispo.		Pas dispo.
P0.15	Carte SD	Train1-PA5		Sg2_4
P0.16	Carte SD	Interrupteurs		C1
P0.17	Carte SD	Interrupteurs		B1 / Sd1 4
P0.18	Carte SD	Interrupteurs		A1
P0.19		Interrupteurs		M1
P0.20		Interrupteurs		C4
P0.21		Interrupteurs		B2 / B4
P0.22	USB micro AB	Interrupteurs		A4
P0.23		Interrupteurs		M4
P0.24		Cnv. AN+Train		S4_3_2
P0.25	Potentiomètre			S4 1
P0.26	Speaker	CNA		S4 2 1
P0.27	USB micro AB	I2C		I2C0 SDA
P0.28	USB micro AB	I2C		I2C0 SCL
P0.29	USB-B (Device)			
P0.30	USB-B (Device)			
P0.31	Pas dispo.	Pas dispo.		Pas dispo.

Port	Utilisation Keil	Utilisation IUT	Nappe 40	Remarque
P1.0	Ethernet			
P1.1	Ethernet			
P1.2	Pas dispo.	Pas dispo.		Pas dispo.
P1.3	Pas dispo.	Pas dispo.		Pas dispo.
P1.4	Ethernet			
P1.5	Pas dispo.	Pas dispo.		Pas dispo.
P1.6	Pas dispo.	Pas dispo.		Pas dispo.
P1.7	Pas dispo.	Pas dispo.		Pas dispo.
P1.8	Ethernet			
P1.9	Ethernet			
P1.10	Ethernet			
P1.11	Pas dispo.			Pas dispo.
P1.12	Pas dispo.	Pas dispo.		Pas dispo.
P1.13	Pas dispo.	Pas dispo.		Pas dispo.
P1.14	Ethernet			
P1.15	Ethernet 50MHz			
P1.16	Ethernet			
P1.17	Ethernet			
P1.18	USB-B (Device)			
P1.19	USB-A (Host)			
P1.20	Joystick Keil			
P1.21				Sd3 4
P1.22	USB-A (Host)			
P1.23	Joystick Keil	Train3		C3
P1.24	Joystick Keil	Train3		B3 / Sg 3 4
P1.25	Joystick Keil	Train3		A3
P1.26	Joystick Keil	Train3		M3
P1.27	USB-A (Host)			
P1.28	IO LED+Cortex dbg	E/S + Train		ON train 2
P1.29	IO LED+Cortex dbg	E/S + Train		AV/AR train 2
P1.30	USB-B (Device)			
P1.31	IO LED+Cortex dbg	E/S + Train		ON train 4

Port	Utilisation Keil	Utilisation IUT	Nappe 40	Remarque
P2.0	COM1	Série		
P2.1	COM1	Série		
P2.2	IO LED+Cortex dbg	E/S + Train		AV/AR Train 4
P2.3	IO LED+Cortex dbg	E/S + Train		ON Train 3
P2.4	IO LED+Cortex dbg	E/S + Train		AV/AR Train 3
P2.5	IO LED+Cortex dbg	E/S + Train		ON train 1
P2.6	IO LED+Cortex dbg	E/S + Train		AV/AR train 1
P2.7	CAN2	CAN		
P2.8	CAN2	CAN		
P2.9	USB-B (Device)			
P2.10	Jumpers ISP et INT0	Train2		C2
P2.11		Train4		S4 3
P2.12		Train2		A2
P2.13		Train2		M2
P2.14	Pas dispo.	Pas dispo.		Pas dispo.
P2.15	Pas dispo.	Pas dispo.		Pas dispo.
P3.25		Timer/PWM		MAT0.0/PWM1.2
P3.26		Timer/PWM		MAT0.0/PWM1.3
P4.28	LCD			
P4.29	Carte SD			Plus rien

S4 2 1 déplacé à la main vers P0.26 sur la v3 (1er jet)

Pas d'interruptions sur le GPIO 1

Version 2.0 – 8 février 2016

Version 3.0 – 9 février 2017 inversion trains 2 et 4 pour éliminer le pb de nappe qui bloque le RESET (nappe 2 branchée au RESET)

FIGURE 10: TABLEAU EXCEL DES FONCTIONS DU MICROCONTROLEUR

Nous avons laissé des ports CAN* pour que la carte puisse communiquer et recevoir des informations ou instructions de la part d'un ordinateur (ou un autre microcontrôleur).

Après avoir relié chaque composant, nous pouvons passer à la deuxième étape : le câblage du PCB*.

2. Câblage du PCB

Nous avons donc une nouvelle interface qui est une modélisation de la carte (réelle) que nous allons imprimer. Pour que notre carte fonctionne correctement nous devons effectuer le routage de celle-ci.

Cette étape, indispensable, consiste à tracer les pistes qui permettent de transmettre l'information et d'alimenter les composants du circuit. Dans notre cas nous avons réussi à nous limiter à une carte avec des pistes sur les deux couches externes (ce qui réduit considérablement les coûts de production).

Positionner les composants sur les deux côtés de la carte permet une meilleure organisation pour le routage. Le routage des cartes respecte une règle implicite et universelle : les pistes ne doivent pas se croiser sur une même couche et ne doivent pas former d'angles droit.

Pour éviter que les pistes se rencontrent, on utilise des « via(s) » qui sont des outils qui permettent de passer une connexion électrique d'une couche à l'autre.

Enfin quand le PCB est réalisé, nous devons faire un plan de masse. Cela consiste à connecter chaque composant à une masse, pour que ceux-ci aient une tension de référence commune et puissent fonctionner en régime nominal.

Remarque : Idéalement dans un véhicule il est recommandé de relier la masse au châssis du véhicule. Dans notre cas le châssis est en plastique et cela ne fonctionnerait pas. Relier la masse au châssis permet d'augmenter la surface de la tension de référence et d'être moins sujet à des perturbations, par exemple les perturbations électromagnétiques que génère le moteur.

Quand le plan de masse est finalisé, nous pouvons passer à la dernière étape : imprimer la carte et souder les composants sur la carte.

3. Soudure des composants

Avant de procéder à la soudure, nous devons effectuer une commande qui va regrouper les éléments de notre PCB. Cette commande demande d'inclure un fichier Gerber (format standard, norme « d'impression ») qui pourra être reconnu et utilisé par les machines de découpe des circuits.

Pour souder les composants nous utilisons la méthode de soudure à l'étain chaud : nous fixons les différents composants du circuit à l'emplacement qui leur est dédié et nous venons appliquer de l'étain sur les pattes des composants. Cela permet la connexion électrique entre la piste et le composant.

Pour s'assurer du bon fonctionnement de la carte, nous effectuons des tests de connexion des pistes. De plus nous devrons mettre en place des protocoles, simples, pour vérifier le bon fonctionnement de chacun des composants assemblés.

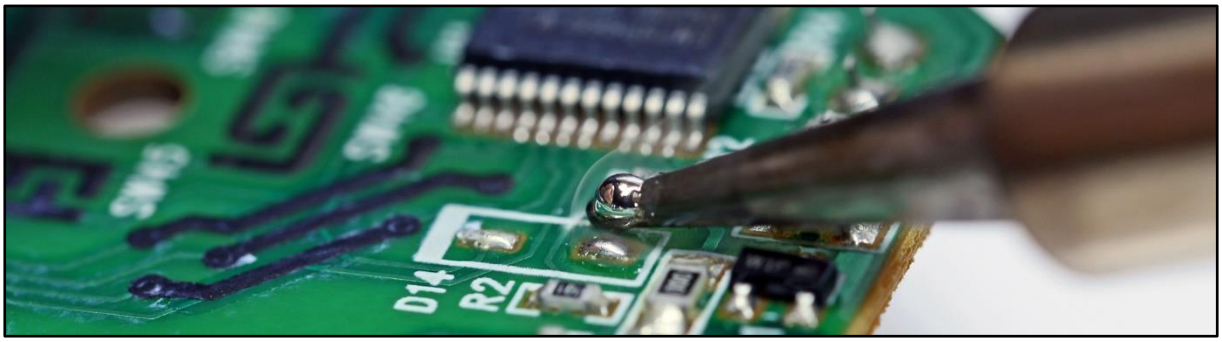


FIGURE 11: IMAGE DE MICROSOUDURE

- Problèmes rencontrés

Le problème principal des PCB est que les éléments constituants ces derniers fonctionnent soit en 5, soit en 3,3 volts. Or notre alimentation fournit une tension de 7,2 volts, tension trop importante qui pourrait abimer les composants.

Pour résoudre ce problème nous pensions tout d'abord ajouter un régulateur de tension qui aurait eu pour rôle d'abaisser la tension d'alimentation. Cependant nous en avons discuté avec notre professeur qui nous a déconseillé de le faire. Cela induirait des courants importants qui provoquerait des perturbations sur les composants de notre carte.

Nous suivons donc le conseil de notre enseignant, et nous allons ajouter une carte supplémentaire qu'il a réalisé pour pallier ce problème.

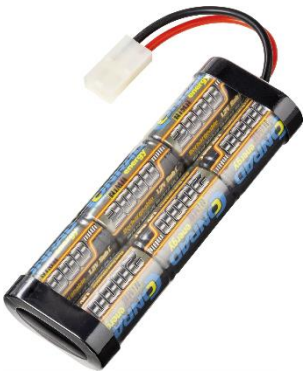


FIGURE 12: BATTERIE UTILISEE DANS NOTRE PROJET

Après avoir été capable de relier entre eux les composants nous devons les piloter. Ce pilotage passe par la compréhension des composants que nous allons traiter. La partie suivante traitera donc ce sujet.

Modélisation de l'environnement

La voiture, pour interpréter et interagir avec son environnement, doit savoir comment celui-ci est constitué. Pour ce faire le véhicule fait appel à divers capteurs qui permettent de cartographier l'espace dans lequel il se déplace.

Cependant utiliser des capteurs implique divers délais qui compliquent la tâche :

- Un premier délai imposé par le principe de fonctionnement physique du capteur.
- Un second délai imposé par le type de transmission des informations.

Ainsi pour rendre efficace la tâche de reconnaissance de l'environnement, nous utiliserons 3 capteurs différents avec des positionnements sur le châssis bien spécifiques : 3 capteurs à ultrasons et 3 capteurs infrarouges que nous appellerons des capteurs de proximité et un LiDAR.

- Le LiDAR

1. Principe de fonctionnement d'un LiDAR

Le capteur LiDAR pour « Light Detection And Ranging » est un système opto-électronique* qui permet de mesurer des distances. Le fonctionnement repose sur un capteur infrarouge monté sur un moteur : en actionnant le moteur, le capteur infrarouge tourne sur lui-même.

Le capteur infrarouge envoie ensuite des impulsions laser régulières qui sont perçues après avoir été réfléchies sur un obstacle. Ainsi, plus la lumière émise met de temps à revenir, plus l'obstacle est loin. Cela permet de faire des mesures de distance qui sont associées à un angle et de créer une carte de l'environnement très précise[3].

[3] Le LiDAR utilisé dans notre cas est de la marque SLAMTEC, modèle A2, dont la documentation n'est pas simple à comprendre en début de projet. Nous essaierons d'être le plus précis possible pour simplifier la prise en main du projet.

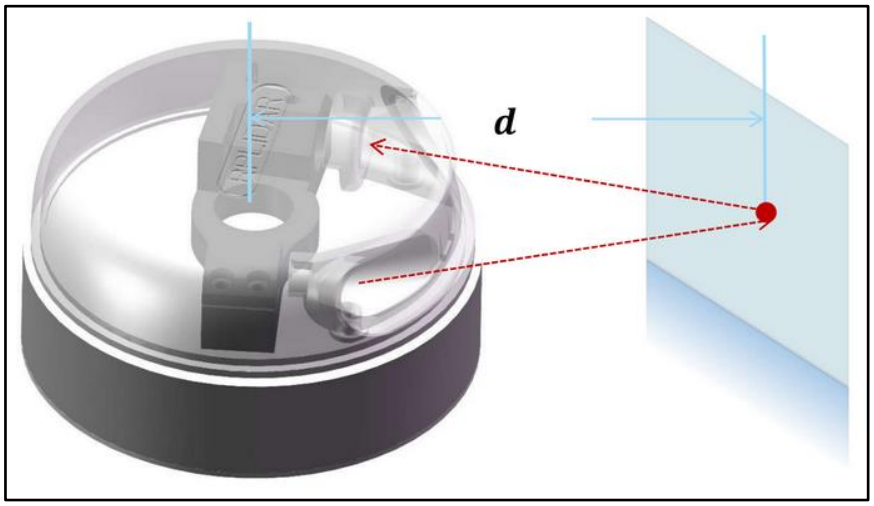


FIGURE 13: MODELE DE FONCTIONNEMENT DU LIDAR

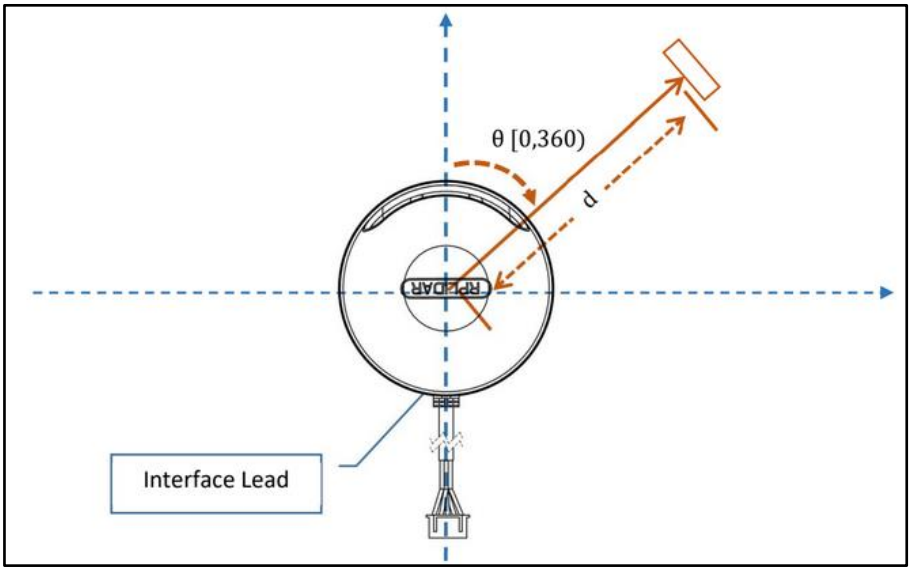


FIGURE 14: REPRESENTATION PRECISE DU FONCTIONNEMENT DU LIDAR

Le LiDAR, en fonctionnement normal, effectue 2000 mesures par secondes. Le réglage du signal PWM généré pour contrôler le moteur de celui-ci permet d'ajuster la précision du LiDAR.

- Si le moteur tourne très vite nous aurons peu de mesures par tour mais beaucoup de tours par seconde à analyser.
- S'il tourne très lentement nous aurons beaucoup de mesures pour un tour mais peu de tours par seconde à analyser.

2. La transmission des informations

➢ L'interprétation des trames

La transmission des informations captées par le LiDAR se fait au moyen d'une liaison série (RS232) dont nous étudierons le fonctionnement plus tard. La bonne compréhension des données transmise nécessite de s'approprier le concept de trames (ou encodage d'informations sur plusieurs octets) que nous allons détailler ici.

Une trame informatique est, par définition, « une structure de base d'un ensemble de données encadrée par des bits de début et des bits de fin ». Cela veut simplement dire qu'une trame est un assemblage de bits qui sont liés à diverses informations.

Dans notre cas, UNE trame transmise par le LiDAR contient 3 informations utiles associées à une mesure (ou un point[4]).

- La qualité de la mesure.
- L'angle de la mesure.
- La distance associée à cet angle.

Cette trame est un assemblage de 5 octets avec la répartition de l'information détaillée sur la figure suivante, selon la documentation du constructeur. De cette lecture on peut retenir la chose suivante. Sur une trame constituée de $5 * 8 = 40$ bits nous avons :

- La qualité sur les bits 2 à 7 (6 bits).
- L'angle sur les bits 9 à 23 (15 bits).
- La distance sur les bits 24 à 39 (16 bits).

[4] Par la suite nous supposerons qu'une mesure (une trame) correspond à un point (en coordonnées polaires).

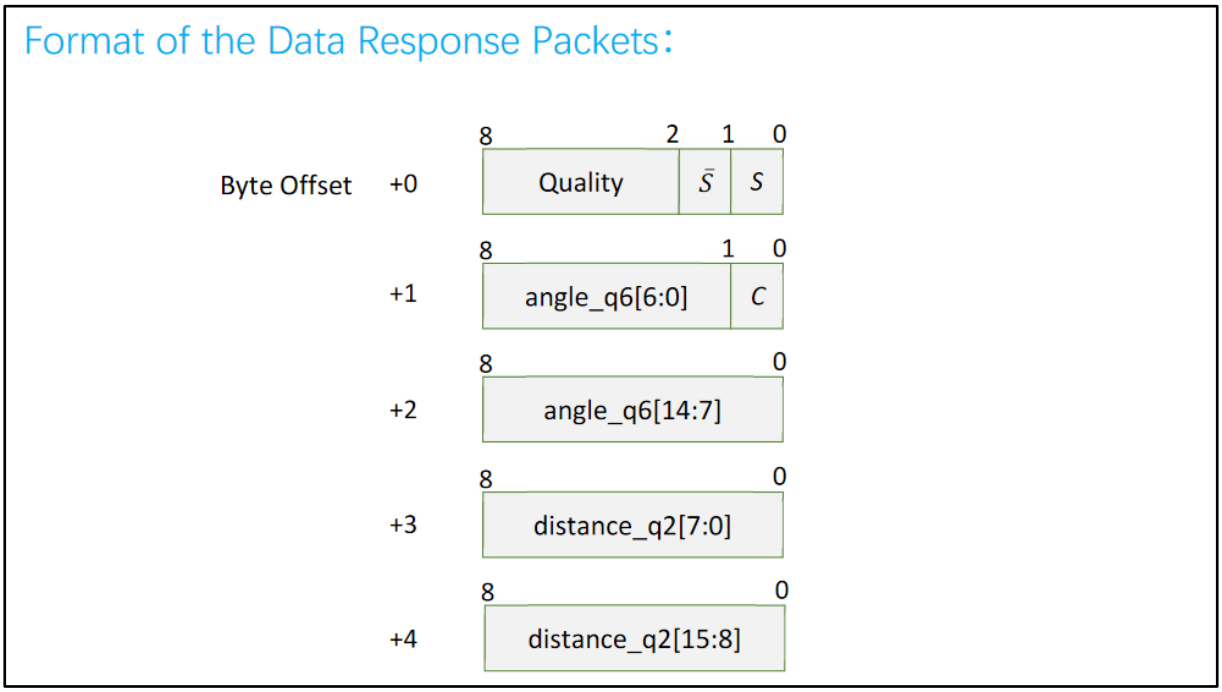


FIGURE 15: CONSTITUTION D'UNE TRAME DE LIDAR

Field Name	Description	Examples / Notes
S	Start flag bit of a new scan	When S is set to 1, the current and incoming packets belong to a new 360° scan.
$\bar{S}$	Inversed start flag bit, always has $\bar{S} = !S$	Can be used as a data check bit.
C	Check bit, constantly set to 1	Can be used as a data check bit.
quality	Quality of the current measurement sample	Related the reflected laser pulse strength.
angle_q6	The measurement heading angle related to RPLIDAR's heading. In degree unit, [0-360) Stored using fix point number.	Refer to the below figure for details. Actual heading = angle_q6/64.0 Degree
distance_q2	Measured object distance related to RPLIDAR's rotation center. In millimeter (mm) unit. Represents using fix point. Set to 0 when the measurement is invalid.	Refer to the below figure for details. Actual Distance = distance_q2/4.0 mm

FIGURE 16: DESCRIPTION PRECISE DE L'ENCODAGE SUR UNE TRAME

Ainsi en réceptionnant les trames dans un tableau de 5 caractères (un caractère correspondant à un octet) il est très simple de reconstituer la distance, l'angle et la mesure d'un point. Voici trois exemples de fonctions qui reconstituent l'information en prenant les octets correspondants en argument.

Remarque : Il existe cependant deux nuances essentielles pour décoder ces trames. La valeur binaire de la distance doit être convertie en flottant puis divisée par 4. La valeur sera alors en millimètres. De même pour l'angle avec une conversion en flottant divisée par 64 pour obtenir des degrés[5].

```
float arrayToAngle(char *data) {
  // Récupère la valeur entière de l'angle
  return (float)((int)data[2] << 7 | (int)data[1] >> 1) / 64;
}

float arrayToRange(char *data) {
  // Bride la distance
  float range = (float)((int)data[4] << 8 | (int)data[3]) / 4;
  if (range > MAX_RANGE) {
    range = MAX_RANGE;
  }
  return range;
}

bool isQualityEnough(char *data) {
  // Récupère la qualité
  return (((data[0] >> 2) & 0x3F) >= POINT_QUALITY);
}
```

FIGURE 17: FONCTIONS DE DECODAGE DES TRAMES

On notera la présence d'une saturation numérique sur la valeur de la distance. Dans notre cas voir à 16 m n'est pas utile et permettra une interprétation plus simple des données.

Nous verrons par la suite que selon le modèle d'exploitation des données, ces trames contiennent deux autres informations utiles : S et /S ainsi que C. Cela permettra de mettre en place un algorithme de synchronisation ou de vérification de synchronisation[6].

[5] Ces informations se trouvent sur la Figure 4.
[6] Se référer à la sous-partie « Désynchronisation des trames »

Maintenant que nous avons compris de quoi est constituée une trame il nous reste à définir les instructions à envoyer au LiDAR pour le piloter. Nous verrons la mise en œuvre de la communication LiDAR<->Microcontrôleur dans la partie suivante.

➢ Les commandes pour piloter le LiDAR

Le LiDAR, pour fonctionner, doit recevoir des instructions de la part du microcontrôleur. Ces instructions qui sont proches du concept de trames permettent de changer l'état du LiDAR. Les instructions ont nous aurons besoin sont présentés ci-dessous avec leur instruction machine associée.

	Trame complète	
Instruction	Octet 1	Octet 2
Stop	0xA5	0x25
Reset	0xA5	0x40
Scan	0xA5	0x20
ForceScan	0xA5	0x21

FIGURE 18: TABLEAU DES INSTRUCTIONS MACHINE

On observe que l'octet n°1 est toujours le même : 0xA5. En effet cet octet quand il est envoyé averti le LiDAR qu'il va recevoir une instruction dans la milliseconde suivante. Si aucune information n'est envoyée (octet 2), le LiDAR reste dans son état actuel.

Sous cet aspect, piloter le LiDAR semble très simple. Malgré tout il existe des informations importantes à savoir pour chacune des instructions.

- Utilité de la fonction *Reset*

La fonction *Reset* permet de redémarrer le système du LiDAR. Il peut être nécessaire de le faire si l'on détecte un défaut de fonctionnement de la part du LiDAR. Par exemple si l'on ne reçoit pas de réponse de celui-ci.

Conclusion : Faire un reset lors de la mise en route du LiDAR permet de s'affranchir de certains problèmes.

- Différence entre les fonctions *Scan* et *ForceScan*

La fonction *Scan* nécessite que la vitesse de rotation du LiDAR soit stable avant que celui-ci n'envoie des points en réponse à l'instruction. La fonction *ForceScan* permet de s'affranchir de cette nécessité et permet d'avoir des mesures plus rapidement.

Ainsi il semble préférable de toujours utiliser *ForceScan* et non *Scan* lors du déploiement du logiciel pilote. Cependant la fonction *ForceScan* ne permet pas d'assurer la continuité des points[7], c'est-à-dire la stricte croissance de l'angle.

Conclusion : Il est préférable d'utiliser la fonction *Scan* lors du déploiement de la version finale du software et *ForceScan* uniquement pour débugger.

Lorsqu'on lui envoie une demande de scan le LiDAR répond par une trame type. Celle-ci est constituée de sept octets : deux octets de synchronisation et cinq octets (une trame complète) pour indiquer l'instruction interprétée par le LiDAR.
Après cela le LiDAR envoie en continu des trames sans s'arrêter, sauf si on le lui demande.

Remarque : Il est important de comprendre qu'une trame équivaut à un paquet de 5 octets. Ainsi un problème de synchronisation ou une perte de trame (non envoyée, mal réceptionnée) entraine des interprétations erronées qui rendent l'utilisation du LiDAR impossible[8].

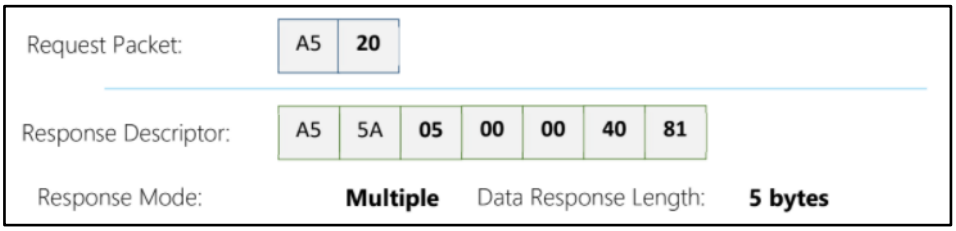


FIGURE 19: REPONSE DU LIDAR A UNE REQUETE "SCAN"

Maintenant que nous avons compris du point de vue théorique comment piloter le LiDAR et interpréter ses données nous allons voir comment cela est réalisé en utilisant un microcontrôleur.

- ## Acquisition des données LiDAR

Afin de réaliser l'acquisition des points collectés par le LiDAR de façon efficace il est nécessaire de définir plusieurs aspects de cette acquisition.

1) Le fonctionnement physique de la communication « microcontrôleur<->LiDAR ».
2) Les outils informatiques mis à disposition en C/C++.
3) Les couches de communication et les objectifs de notre programme.

---

[7] Se référer à l'annexe « *1 :* Continuité des points » pour plus d'informations à ce sujet.
[8] Se référer à la sous-partie « Désynchronisation des trames »

De ces 3 points nous pouvons constituer des algorithmes, plus ou moins efficaces, qui acquièrent les trames envoyées par le LiDAR. L'algorithmie de l'acquisition des points constitue une partie en elle-même. Celle-ci reposant sur des bases nous allons voir, nous y reviendrons donc plus tard.

1. Le protocole de communication : la liaison RS232

La liaison série RS232 asynchrone[9] est un type de protocole d'échange d'informations. Ce protocole permet la communication entre deux modules avec un nombre de fils inférieur à d'autres protocoles (I2C*, CAN).

Ce protocole permet une liaison bipolaire avec deux fils et unipolaire avec un fil, en plus de la masse commune. La connexion entre les deux modules est relativement simple. Contrairement aux protocoles I2C et CAN, la liaison série permet de relier uniquement deux appareils entres eux.

La liaison série peut être vue comme un canal. En supposant un transmetteur TA et un récepteur RB : on relie le canal d'émission de TA sur le canal de réception de RB. Ainsi B peut écouter A au fur et à mesure que A envoie les informations. L'inverse est bien entendu possible si B est capable de transmettre et A d'écouter. Cependant A et B doivent décider d'une cadence commune de lecture et d'écriture.

La liaison série ne repose pas sur un fil qui constitue une horloge commune mais sur le fait que l'émetteur et le récepteur se sont auparavant mis d'accord sur la cadence à suivre. On peut comparer ce fonctionnement de cadence commune au morse : un opérateur doit savoir différencier ce qu'est un trait d'un point. Si le récepteur (l'opérateur humain) ne sait pas différencier les traits des points envoyés par l'émetteur (l'autre opérateur) cela pose un problème.

On note usuellement TX et RX les pins des liaisons séries. De plus lorsque la liaison série (du côté du récepteur) reçoit des bits celle-ci les stocke jusqu'à reconstituer un octet.

Remarque : La liaison avec le LiDAR est asynchrone. Cela sous-entend qu'il peut y avoir un décalage des bits au bout d'un certain temps d'utilisation de la liaison[10].

[9] Nous appellerons par la suite cette liaison simplement « liaison série ». Celle-ci est peut aussi être appelée interface « UART ».
[10] Se référer à la sous-partie « Désynchronisation de la liaison série »

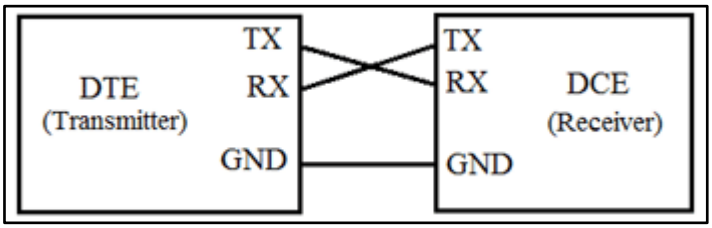


FIGURE 20: CABLAGE D'UNE LIAISON SERIE

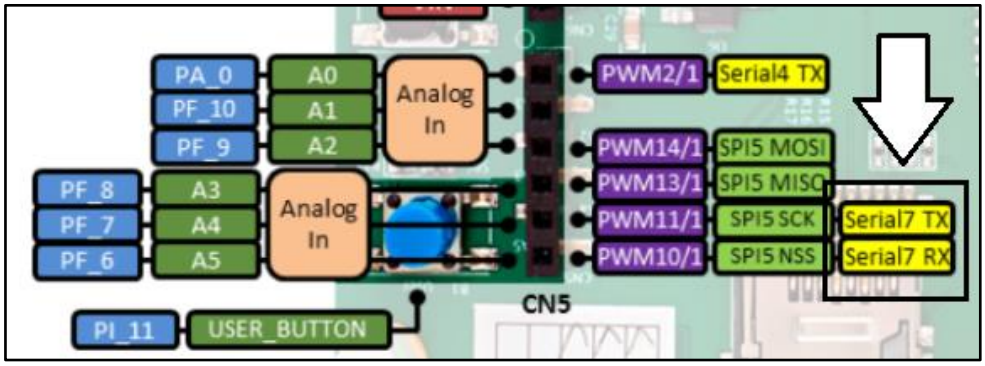


FIGURE 21: NOMS USUELS DONNES AU PINS QUI SUPPORTENT LA LIAISON SERIE

Ainsi après avoir établi une liaison physique entre le LiDAR et le microcontrôleur il est nécessaire de pouvoir écrire dessus ou lire ce qu'on a reçu.

2. Les outils informatiques nécessaires au projet

Le microcontrôleur choisi pour toute la réalisation de ce projet (coté acquisition des données) est un MBED DISCO-F746NG. L'idée à l'origine étant de migrer le code C++ en C pour un processeur CORTEX M3 LPC1768 (cf. carte Keil). Cependant par manque de temps la migration n'a pu être faite. Pour autant il est possible d'utiliser le microcontrôleur MBED NXP LPC1768 pour réaliser ces tâches sans migration de code[11].

➢ Les interruptions et la liaison série

Lors du semestre 3, nous apprenons l'existence de fonctions d'interruptions. Sans rentrer dans les détails ces fonctions sont, comme leur nom l'indique, des fonctions qui suspendent temporairement la tâche qu'exécute le microcontrôleur. Cela permet au microcontrôleur d'exécuter une autre tâche, plus importante, avant de reprendre là où il en était.

[11] Documentation disponible sur le site : https://os.mbed.com/docs/mbed-os/v5.15/apis/index.html
➔ **Attention :** Version 5.15 de MBED OS.

Ainsi nous allons utiliser une fonction d'interruption dans ce programme : une interruption du microcontrôleur déclenchée lorsqu'un caractère est arrivé sur la liaison série. Une fois cette interruption déclenchée, on peut la traiter. « Traiter » entend ici « stocker le caractère » et « s'acquitter de l'interruption »[12].

```
// Instances du programme
RawSerial SLidar(TXLIDAR, RXLIDAR, 115200);
```

FIGURE 22: CREATION DE LA LIAISON SERIE "SLIDAR"

```
// Défini le formats de la liaison série
SLidar.format(8, SerialBase::None, 1);
// Défini l'interruption sur reception de caractères
SLidar.attach(&getChar, Serial::RxIrq);
```

FIGURE 23: CONFIGURATION DE LA LIAISON SERIE ET DE L'INTERRUPTION A EXECUTER

Ainsi on réalise trois tâches successives :

- Déclarer la liaison série avec les pins TXLIDAR et RXLIDAR : ces variables sont définies dans un fichier d'entête (ou « header/.h »). On déclare en même temps que l'horloge de fonctionnement mutuelle est de 115200 bauds*.
- Donner le format de la liaison série selon la documentation de MBED OS v5.15.
- Associer une fonction à l'interruption « caractère reçu » en passant comme paramètre l'adresse de cette fonction (ici « getChar »).

Maintenant que l'on a créé la liaison série et donné un nom à la fonction d'interruption on peut affirmer deux choses : le microcontrôleur est capable de communiquer avec le LiDAR (liaison bipolaire) et celui-ci appellera systématiquement la fonction « void getChar(void) » lors de cette interruption. Il faut maintenant stocker les caractères qui sont arrivés pour pouvoir les traiter.

➢ Le buffer circulaire

Un « buffer » (ou tableau) est un espace mémoire alloué pour stocker des informations. Ainsi un tableau de caractère pourrait être suffisant pour stocker les octets qui arrivent sur la liaison série. Cependant si l'on en vient à saturer ce tableau il peut y avoir un

[12] Cette notion sera précisée au cours du semestre 3 à l'IUT de Cachan, en cours de IENA.

phénomène de dépassement de capacité : on demande de stocker plus d'informations que possible.

Cela présente un énorme problème : le microcontrôleur ne sait pas comment réagir et prendra une décision qui nous est inconnue. Pour pallier ce souci on peut choisir d'utiliser une autre structure de données : le « buffer circulaire ».

Le principe du buffer circulaire repose sur le principe que celui se rempli de façon cyclique. Quand on arrive à la fin du tableau on recommence au début. Si celui-ci arrive à saturation on peut choisir quoi exécuter :

- Ecrire l'informations reçu sur des informations non traitées (écraser des données anciennes)
- Ignorer l'informations reçu tant que le buffer est saturé (rejeter des données récentes)

Le buffer circulaire respecte le principe de FIFO* qui permet de garder une continuité dans l'arrivée des données.

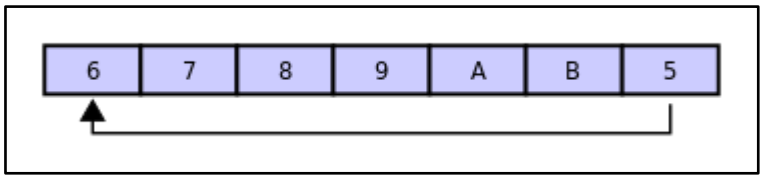


FIGURE 24: SCHEMA FONCTIONNEL D'UN BUFFER CIRCULAIRE

Ainsi en liant ce type de structure à notre fonction d'interruption on peut stocker les octets qui arrivent un à un en évitant tout comportement imprévisible. On profite aussi de l'aspect FIFO qui évite de devoir manipuler un itérateur avec le risque de mal le manipuler. Il faut tout de même noter que le buffer circulaire, plus complexe, introduit des délais plus importants qu'un tableau simple.

```
CircularBuffer<char, RAW_BUFFER_SIZE> rawBuffer;
```

FIGURE 25: DECLARATION D'UN BUFFER CIRCULAIRE (STOCKANT DES CARACTERES)

```
void getChar(void) {
  // Récupère un caractère et le stock dans le buffer circulaire
  // prévu à cet effet

  rawBuffer.push((char)SLidar.getc());
}
```

FIGURE 26: FONCTION D'INTERRUPTION QUI TRAITE LE CARACTERE DISPONIBLE SUR LA LIAISON

Après avoir réceptionné les octets, les avoir rassemblés et être capable de décoder les trames il reste la question de savoir quoi en faire et comment les transmettre à la couche d'interprétation.

3. Les couches de communications et les objectifs du programme

D'une part le microcontrôleur reçoit des octets un à un que le microcontrôleur doit décoder (par paquet de 5 octets, c-à-d des trames ou des points). D'autre part, ces données doivent être transmises pour être interprétées. Plusieurs possibilités ont été émises quant à l'envoi des points collectés à la couche d'interprétation.

Il a été décidé plusieurs choses pour la conception de ces « paquets de points » :

- La nécessité d'un format simple qui soit abordable par n'importe quel type de programmeur.
- Une quantité d'informations limitée pour faciliter et accélérer le processus de traitement.
- Une conversion des angles dans un format adapté au traitement.
- Un protocole qui permette un transfert simplifié vers d'autres modules.

Pour répondre à ces critères nous avons décidé de stocker 181 points dans un tableau de type flottants où l'indice du tableau correspond au degré scruté.

Effectivement seul l'avant de la voiture nous intéresse : il n'est donc pas nécessaire de mémoriser ce qu'il se passe derrière le véhicule. De plus selon la figure suivante il convient de changer le format de l'angle : le format fourni par le LiDAR ne correspondant pas à nos besoins.

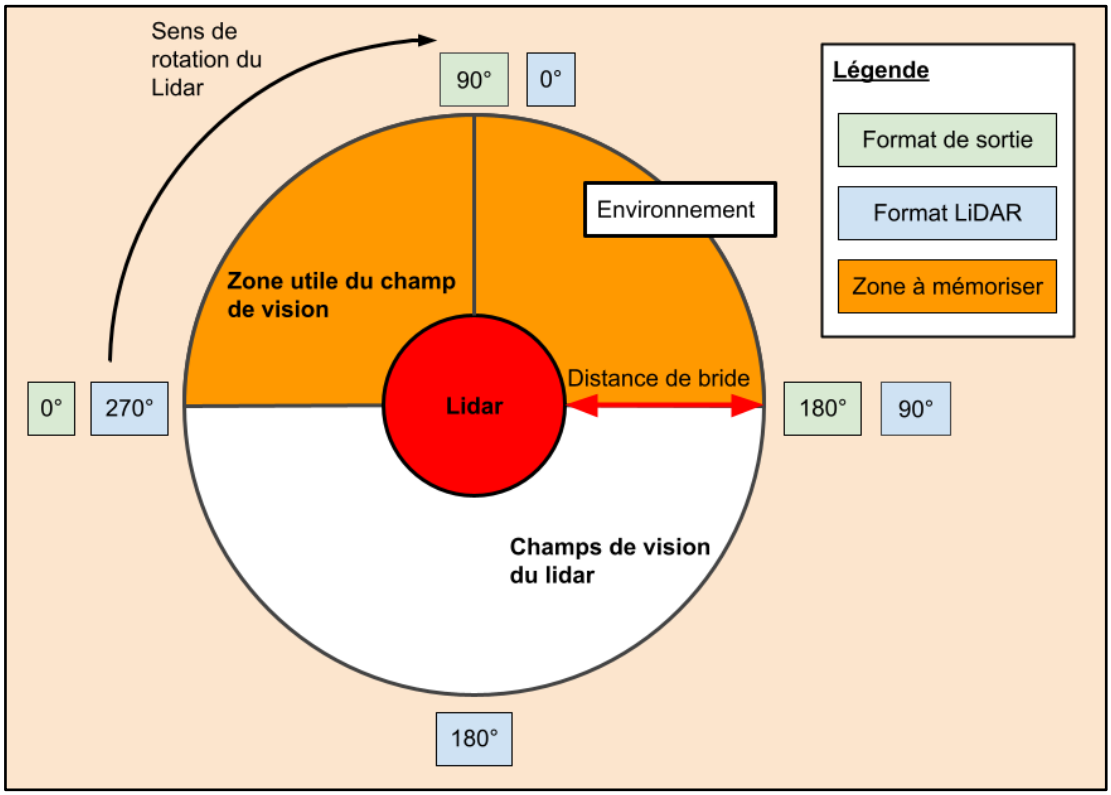


FIGURE 27: REPRESENTATION DE LA VISION DU LIDAR ET DES FORMATS D'E/S

On peut représenter le format de sortie des données sous une forme plus explicite.

	Tableau de flottant de taille 181						
Indice	0	1	…	90	91	…	180
Flottant (distance) associée	Distance à 0°	Distance à 1°	…	Distance à 90°	Distance à 91°	…	Distance à 180°
Position par rapport à la voiture	Gauche			Devant			Droite

FIGURE 28: REPRESENTATION EXPLICITE DU FORMAT DE SORTIE

Pour passer de la couche d'entrée à la couche de sortie il faut donc composer un algorithme qui fasse l'interface entre les couches « d'entrée » et de « sortie ». Nous nommerons en plus des critères précédents d'autres critères afin de mesurer l'efficacité de notre algorithme[13] :

- La rapidité d'exécution.
- Le temps de réponse du programme (temps mis pour reconstituer un tour complet).
- La fiabilité du programme face à des évènements aléatoires : perte de trames, décalage des bits de la liaison asynchrone.
- La capacité mémoire occupée par le programme.

[13] Se référer à l'annexe B2, « Synchronisation en temps réel » pour plus d'informations à ce sujet.

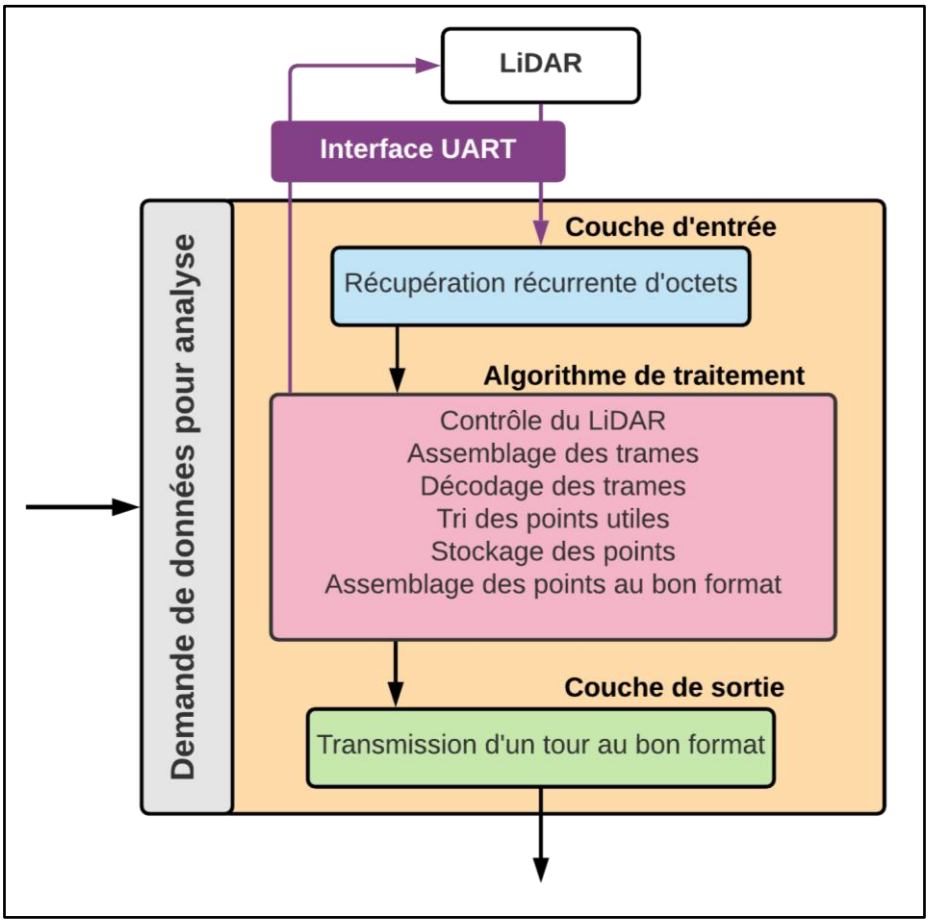


FIGURE 29: SCHEMA RESUMANT L'ACQUISITION DES DONNEES

- ## Algorithmes de traitement des données

Après avoir établi les principes de fonctionnement de l'algorithme de traitement nous pouvons nous atteler à le concevoir. Nous allons dans un premier temps étudier deux modèles « primitifs » et un modèle plus complet.

Les deux premiers modèles bien que très simples en comparaison avec le dernier ont été nécessaires. Ce sont eux qui ont servi de base d'apprentissage pour appréhender le LiDAR, décrypter sa documentation complexe et permettre l'existence d'algorithmes* plus légers ou rapides.

Cette partie, moins rédigée et plus schématisée tentera d'aborder un problème paradoxal soulevé par le LiDAR. Ce capteur envoie une quantité d'information très importante (2 000 points par seconde). Cependant la gestion plus ou moins bonne de ces données vient ralentir l'acquisition des points et donc la prise de décision.

1. Acquisition à durée fixe

Le premier modèle consiste à se déconnecter puis se reconnecter de façon régulière pour obtenir les points qui constituent l'environnement. Ainsi on alterne entre les commandes « start » et « stop » pour obtenir une quantité finie de points N_p sur une durée θ_T. Après avoir acquis ces points, on les traite tous : on isole ceux qui nous intéressent, si possible les plus récents. Puis on recommence indéfiniment.

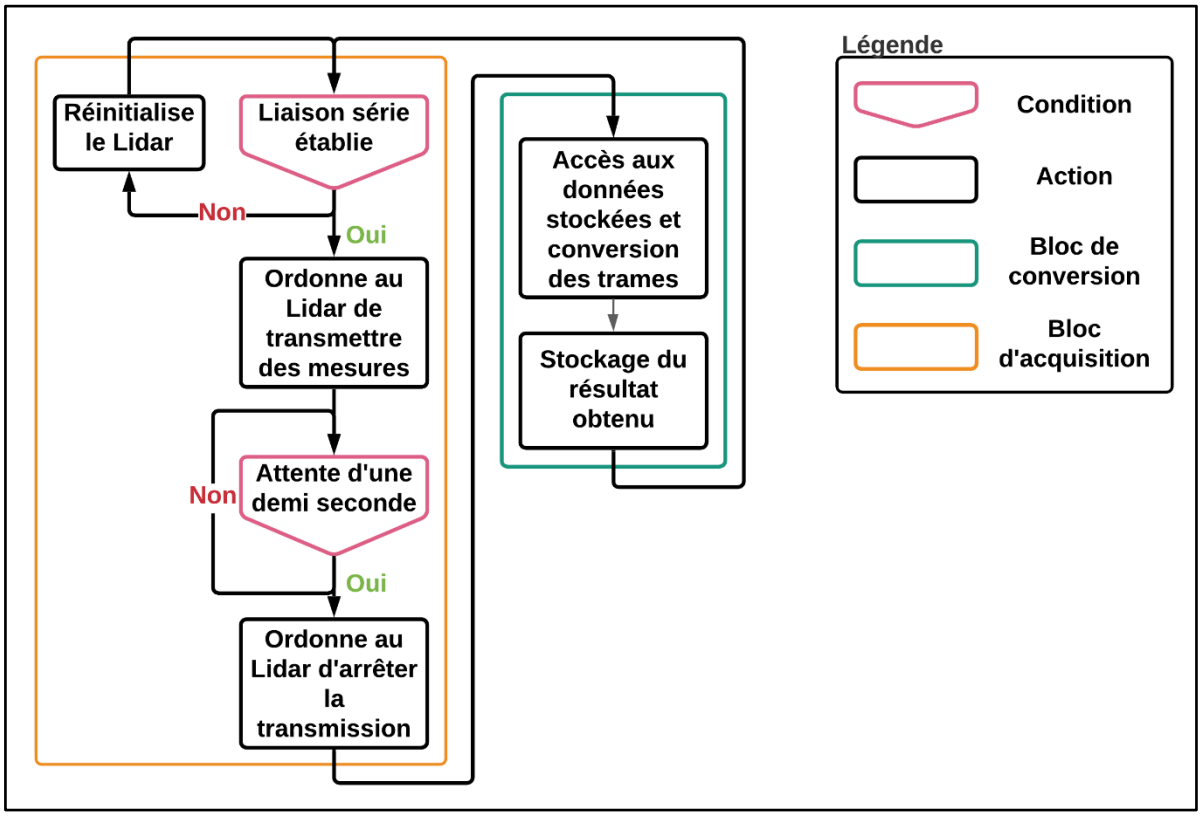


FIGURE 30: ALGORIGRAMME DU MODELE N°1

Avantages

- Simple à mettre en œuvre.
- Permet de comprendre comment fonctionne le LiDAR et confirmer son mode de fonctionnement.

Défauts

- Lent à l'exécution.
- Enregistre plusieurs tours à la chaine quand seulement le plus récent nous intéresse.
- Le temps arbitraire d'une demi-seconde n'est pas suffisant. En réalité on constate que : $\boxed{T_{acquisiton} > 0{,}8s}$

2. Acquisition avec boucle de contrôle « temps réel »

Le second modèle, un peu plus complexe en comparaison au premier, repose sur le même principe d'acquisition avec l'alternance entre les commandes « start » et « stop ». La différence étant que la durée d'acquisition θ_T n'est plus fixe mais variable. Ainsi on décode en « temps réel » les points qui arrivent. Cela est possible car $T_{acquisition\,trame} \gg T_{conversion\,trame}$. Ainsi on peut récupérer les points, les décoder et les stocker au fur et à mesure qu'ils arrivent. On obtient donc un modèle plus efficace que le premier.

Cependant le principe de connexion/déconnection n'est pas très efficace : en effet ordonner au LiDAR de se mettre en mode « acquisition » a un coût en temps (valeurs mesurées expérimentalement) :

- De l'ordre de 0,3s pour l'instruction ForceScan.
- De l'ordre de 0,7s pour l'instruction Scan.

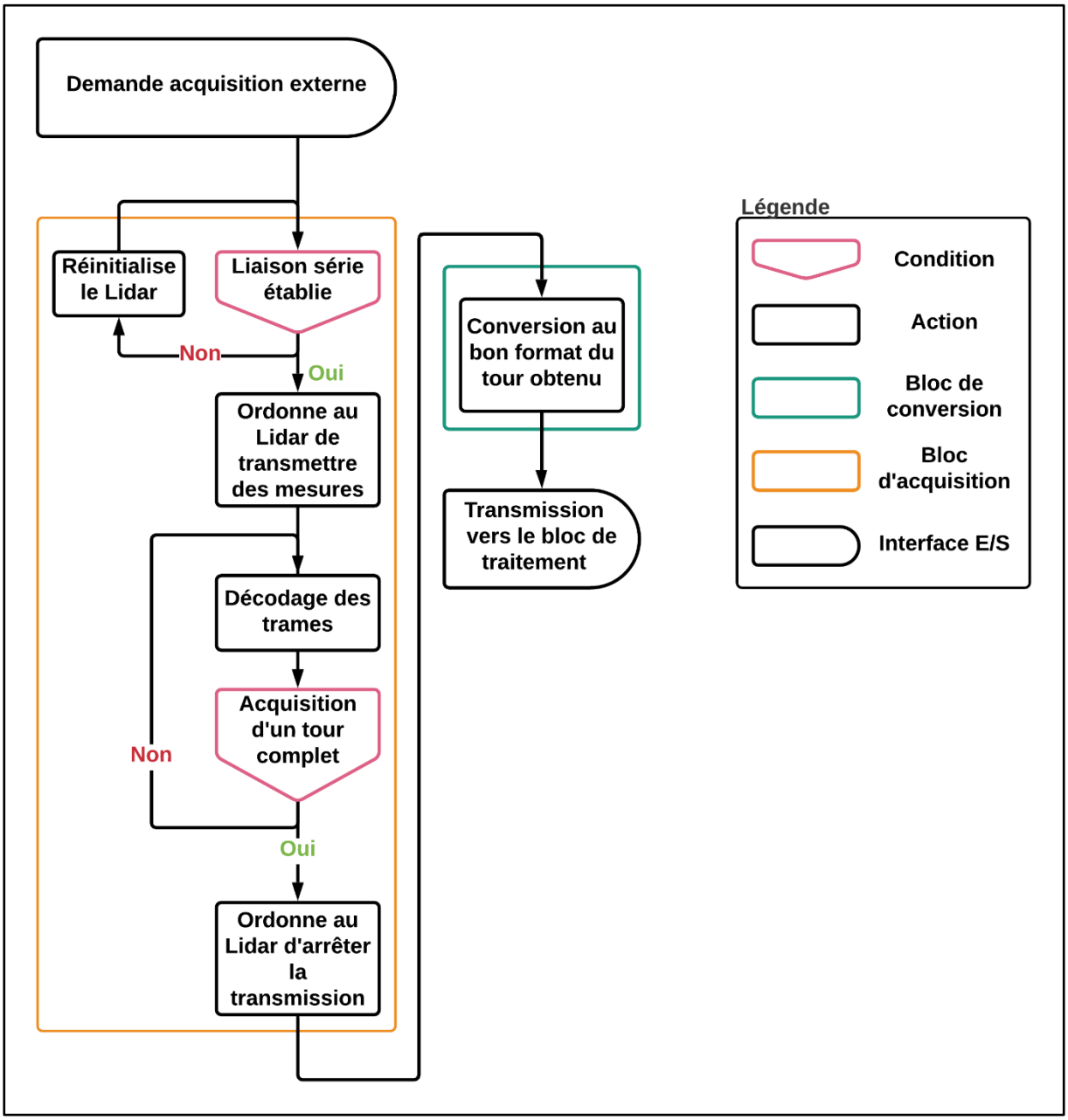


FIGURE 31: ALGORIGRAMME DU MODELE N°2

Avantages

- Plus rapide que le précédent.
- N'enregistre qu'un seul tour et un tour « complet ».
- Transmet toujours le tour le plus récent.

Défauts

- Bien que plus rapide que le modèle précédent celui-ci reste lent à l'exécution.
- Compliqué à mettre en œuvre.

3. Acquisition avec rejet de trames et boucle de contrôle « temps réel »

Ce modèle qui diffère très peu du précédent permet d'éviter les temps longs d'attente générés par le modèle de connexion / déconnexion. Celui-ci consiste à rejeter les trames une à une tant qu'il n'y a pas de demande d'acquisition de tour : cela est géré par la fonction d'interruption. Cependant si une demande d'acquisition est adressée les trames sont traitées et le dernier tour disponible est renvoyé.

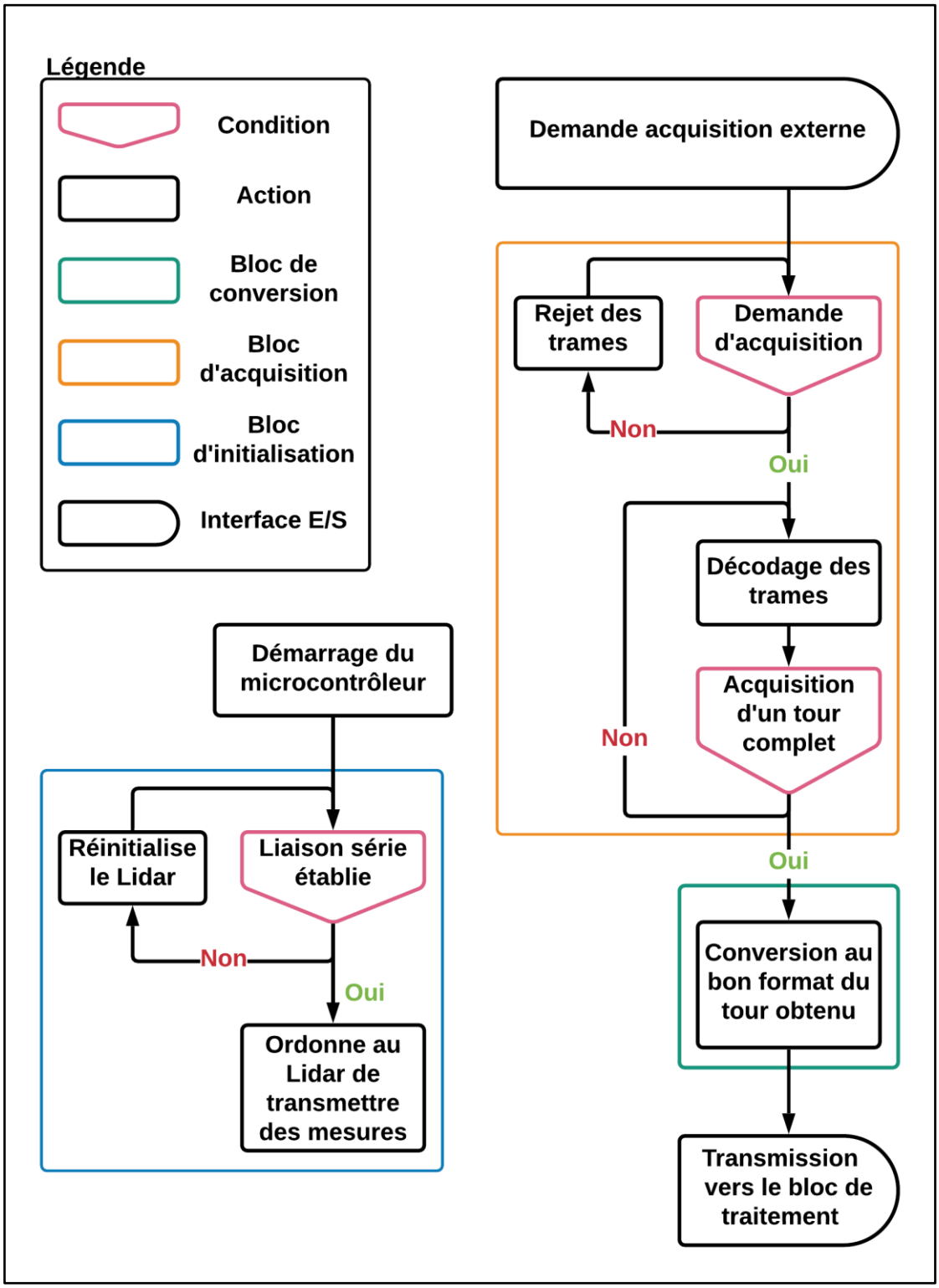


FIGURE 32: ALGORIGRAMME DU MODELE N°3

Avantages

- Beaucoup plus rapide que les algorithmes précédents.
- N'enregistre qu'un seul tour et un tour complet.

- Transmet toujours le tour le plus récent.
- Variante de l'algorithme numéro 2.

Défauts

- Risque de désynchronisation très élevé**.**
- Peu robuste. En cas de désynchronisation, aucun contrôle ne permet d'en être informé.
- Nécessité de reconnexion avec le LiDAR toutes les 10 secondes pour palier au décalage des bits de la liaison série asynchrone.

Nous avons donc proposé des modèles d'algorithmes de plus en plus efficace. L'acquisition d'une quantité importante de données n'étant pas simple, l'optimisation est primordiale. Cependant certains problèmes persistent et ces algorithmes pourraient être encore améliorés.

D'autres outils informatiques pourraient être utilisés comme le multitâche, la DMA. Par ailleurs un autre protocole (plus complexe) de transmission existe : le LiDAR met à disposition un mode « express scan » qui permet de passer de 2000 à 4000 points par secondes. Par manque de connaissances nous ne détaillerons aucune de ces notions citées.

- Les difficultés rencontrées

1. Cout en temps de la méthode « start/stop »

Dans la partie précédente nous avons abordé le problème de la méthode « connexion déconnection » qui vient rajouter un temps considérable du au changement d'état du LiDAR. Ce temps peut être mis en évidence à l'aide de « timer » qui mesurent le temps de réception des trames. On constate que le premier point est réceptionné après un délai important d'environ 0,25 secondes.

Cependant la réponse du LiDAR à la requête est immédiate : « 0xA5 0x5A » pour établir la transmission. Puis « 0x05 0x00 0x00 0x40 0x81 » pour confirmer que la requête reçue est la bonne. Il y a donc un délai entre la réception de la requête et la mise en œuvre de celle-ci.

```
char1 = A5 char2 = 5A
//Connection établie

Octet numero 1 en 6 us
Val_octet : 5
Octet numero 2 en 6 us
Val_octet : 0
Octet numero 3 en 6 us
Val_octet : 0
Octet numero 4 en 5 us
Val_octet : 40
Octet numero 5 en 6 us
Val_octet : 81
Octet numero 6 en 245119 us
Val_octet : 3E
Octet numero 7 en 6 us
Val_octet : 7D
Octet numero 8 en 6 us
Val_octet : 41
TOctet numero 9 en 5 us
Val_octet : 0
Octet numero 10 en 5 us
Val_octet : 0
```

FIGURE 33: TEMPS D'ACQUISITION DES 10 PREMIERS OCTETS (INSTRUCTION FORCESCAN)

En ForceScan on constate que le temps est de $245119\mu s \approx 0{,}25s$. Pour une requête de Scan ce temps est plus long (environ $0{,}6s$).

2. Désynchronisation de la liaison série

Passé un certain délai, d'environ 15 secondes, la liaison série se désynchronise. De ce fait on récupère des points convertis qui sont aberrants : par exemple de angles de 520 degrés.

Remarque : Avec notre professeur nous avons soulevé l'hypothèse suivante, cependant nous n'avons aucune certitude de ce qu'il se passe réellement. Les drivers de la liaison série devraient d'eux-mêmes gérer ce type de problème. Une étude plus poussée serait nécessaire.

Cet effet est dû au type de liaison. Dans une liaison série il n'y a pas d'horloge commune mais deux horloges respectivement égales (ou pas) : l'une chez l'envoyeur et l'autre chez l'émetteur. Cependant ces horloges ne sont pas exactement les même car leur calcul repose sur la fréquence de fonctionnement des quartz qui constituent l'horloge de référence des microcontrôleurs[14].

[14] Cette notion sera précisée au cours du semestre 3 à l'IUT de Cachan, en cours de IENA.

La formule permettant de calculer le nombre de bauds est la suivante.

$$Baud_{rate} = \frac{P_{clk}}{16 * (256 * DLM + DLL) * (1 + \frac{DIV_{add\ val}}{MUL_{val}})}$$

Avec :

- P_{clk} l'horloge du microcontrôleur en Hz
- $Baud_{rate}$ en bits/secondes.

On peut sans calcul en déduire que plusieurs valeurs des paramètres permettent d'atteindre 115 200 bauds, de façon plus ou moins précise. Ainsi une tolérance existe entre $\mathrm{Baud}_{\mathrm{théorique}}$ et $\mathrm{Baud}_{\mathrm{effectif}}$. Cette tolérance est de l'ordre d'un pourcent au maximum. C'est ce décalage entre les deux horloges qui au bout d'un certain temps entraine un décalage des bits. Les octets entre l'émission et la réception sont alors différents et la liaison n'est plus exploitable.
Pour pallier ce problème nous relançons la connexion entre le LiDAR et le microcontrôleur régulièrement, environ toutes les 10 secondes.

3. Désynchronisation des trames

Les algorithmes présentés ne sont pas suffisamment robustes à l'état actuel : ils ne permettent pas de vérifier qu'un octet constituant une trame n'a pas été perdu entre l'émetteur et le récepteur. Cela aurait pour effet de décaler les octets qui constituent les trames : on croit avoir l'octet 0 de la trame mais celui-ci a été « perdu ». En réalité on a l'octet 1 qu'on croit être le 0 et ainsi de suite : la conversion des trames en point devient faussée.

Des repères de vérifications des octets qui constituent les trames sont malgré tout disponibles et un algorithme de vérification de la synchronisation serait souhaitable. Celui-ci pourrait s'appuyer sur les 3 informations suivante :

- Le bit C toujours égal à 1.
- Les bits S et /S avec la règle qui s'applique selon la documentation.
- La valeur entière de l'angle avec : $0 \leq Angle < 360$.

Hypothétiquement on pourrait aussi vérifier l'équation de récurrence suivante :

$$Angle_{précédent} < Angle_{actuel} \text{ pour } S = 0 \; et \; \bar{S} = 1$$

Cependant il été constaté qu'il y a parfois des sauts de valeur en *ForceScan :* la stricte croissance de l'angle n'est pas vraie. On suppose qu'en *Scan* cela ne devrait pas arriver mais aucune vérification n'a été faite.

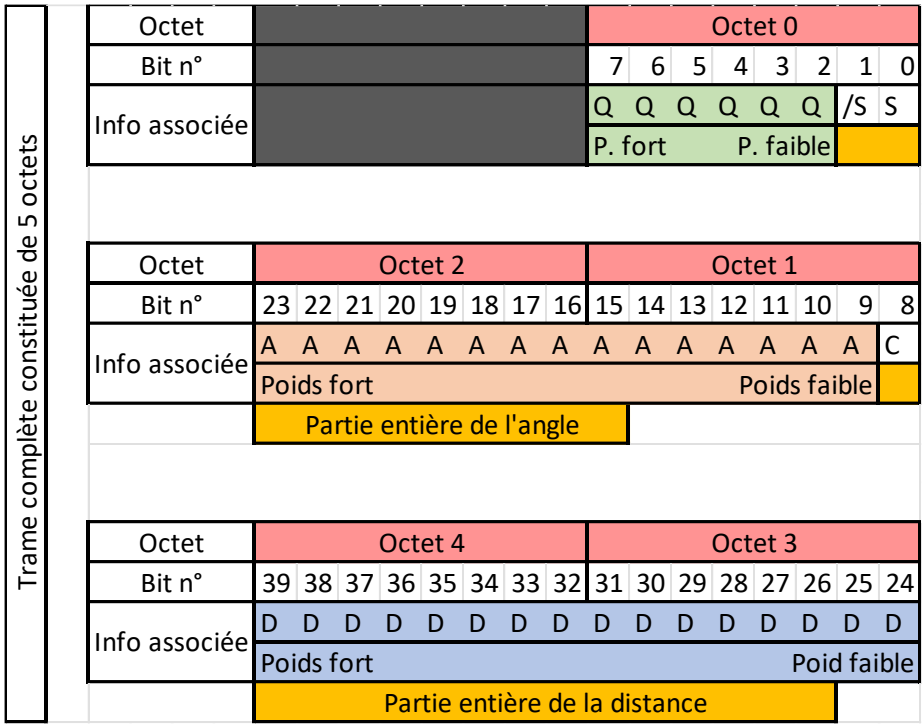


FIGURE 34: DESCRIPTION COMPLETE D'UNE TRAME POUR OPTIMISATION

4. Optimisation des conversions

Dans un souci d'optimisation avec, à terme, un microcontrôleur sans FPU* il est préférable de transformer les opérations de division par des décalages de bits (shifts).

En effet le degré de précision de la distance a été considéré comme suffisant en millimètre. De même selon le protocole de transmission des données seul la partie entière de l'angle nous intéresse. On rappelle qu'en programmation diviser par une puissance de 2 revient à faire un décalage binaire vers la droite de cette même puissance.

Ainsi on omet la précision décimale d'une valeur pour gagner en nombre d'instructions élémentaires. Le gain est considérable et ne devrait pas être négligé !

Les capteurs PING et infrarouges

- Présentation des deux capteurs

Les capteurs complémentaires utilisés pour compléter l'acquisition de l'environnement sont des capteurs à ultrasons et des capteurs infrarouges. Ceux-ci permettent d'épauler le LiDAR, de confirmer son acquisition ou de la contredire selon le cas de figure. Ces capteurs dits de « proximités » présentent l'avantage d'être très fiables et d'avoir des temps d'acquisition de l'environnement constants.

On peut supposer un guidage « primitif » (sous-entendu court terme) issu des données de ces capteurs de proximité et un guidage « poussé » (sous-entendu long terme) qui s'appuie sur les données du LiDAR.

L'étude de ces capteurs sera moins complète en comparaison avec celle du LiDAR : ces capteurs sont bien moins complexes.

1. Le capteur PING

Le capteur à ultrason est un capteur qui permet de connaitre la distance entre lui et un objet qui se trouve dans son champ de vision (ou de rayonnement). Le capteur envoi un ultrason qui est réfléchi sur l'objet : le temps entre l'émission et la réception permet de mesurer la distance avec l'objet. Plus la distance avec l'obstacle augmente, plus la largeur de l'impulsion reçue augmente : ainsi un obstacle proche sera détecté plus rapidement qu'un obstacle éloigné.

Remarque : La propagation du son est très lente en comparaison avec la lumière. Ainsi les capteurs à ultrasons que nous utilisons permettent de faire, au maximum, 20 mesures par secondes.

Les capteurs à ultrason utilisés permettent en revanche d'avoir un secteur angulaire assez large et d'avoir une idée globale de l'environnement en une mesure. Ceux-ci seraient très utiles pour savoir si une mesure d'urgence est à prendre : s'arrêter, braquer les roues, décélérer fortement.

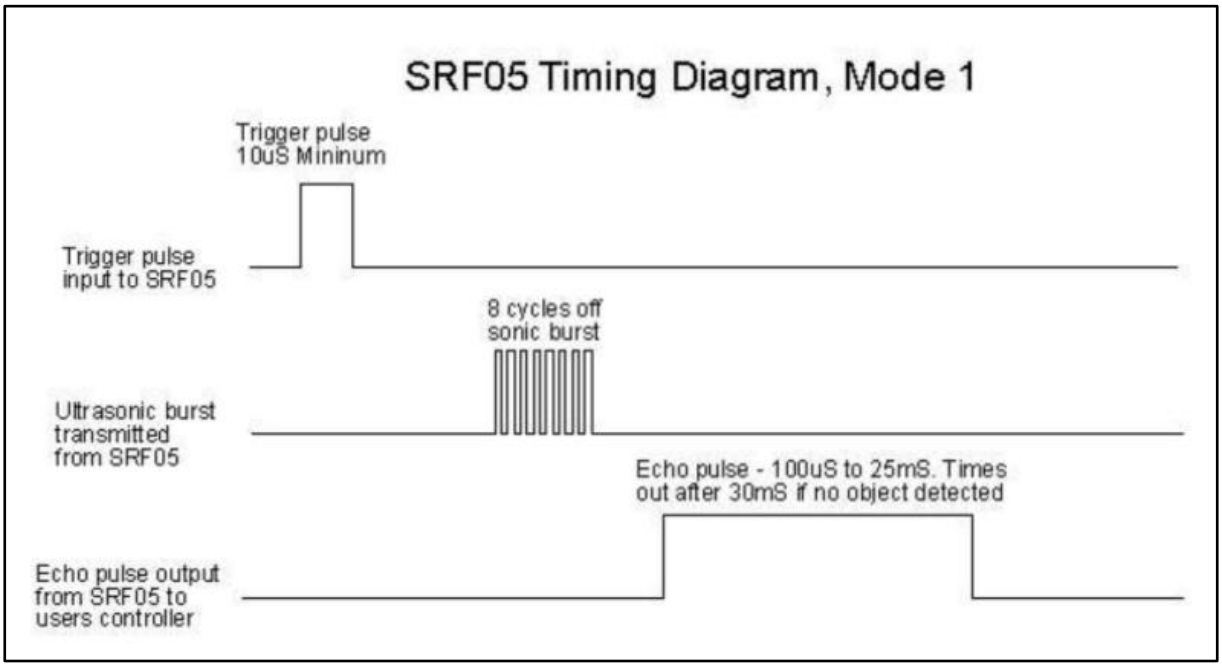


FIGURE 35: COMMANDE ET REPONSES D'UN CAPTEUR A ULTRASONS

On observe que la commande des ultrasons se fait en deux temps : envoyer une impulsion de start de 10 µs puis recevoir une impulsion. Cela demande d'utiliser au moins 2 pins GPIO pour chacun des ultrasons.

Cependant on pourrait limiter le nombre de PINs de commande. En reliant la commande des ultrasons à une ligne commune, on peut envoyer une impulsion de 10 µs qui sera perçue par plusieurs capteurs ultrason.

2. Le capteur infrarouge

Les capteurs infrarouges sont extrêmement rapides et fournissent une mesure analogique qui correspond à la distance entre le capteur et l'obstacle le plus proche. Ceux-ci permettent un guidage très simple et très rapide. D'une part ils permettent de prendre des mesures d'urgences, d'autre part de guider le véhicule quand aucune autre information n'est disponible.

Le principal désavantage de ces capteurs repose sur le secteur angulaire qu'ils couvrent : ceux-ci ne voient que devant eux.

- ## Utilité des capteurs de proximité

Ces capteurs peuvent sembler dérisoires après toute l'étude sur le LiDAR. Cependant deux problèmes viennent être résolus par ces capteurs : la durée longue d'acquisition du LiDAR et les problèmes de soulèvement du châssis.

Comme vu précédemment, les données LiDAR selon l'algorithme utilisé, peuvent mettre plus ou moins de temps à arriver à la partie décisionnelle. Ces capteurs de proximité sont très rapides en comparaison et permettent de réagir vite.

De plus, si la voiture subit un choc ou accélère d'un coup, le châssis peut se soulever et mener à un LiDAR qui pointe vers le ciel ou le sol : celui n'est alors pas du tout exploitable.

En venant épauler le LIDAR, ils peuvent confirmer ou contredire ce que le LiDAR croit voir. Si plusieurs processus différents arrivent à la même conclusion alors l'information transmise est fiable. Sinon il est préférable de croire les capteurs de proximité plus fiables et robustes à la casse.

- ## Positionnement des capteurs

Nous avons dû faire des choix afin de positionner les capteurs de proximité. Le but étant de limiter leur nombre en se concentrant sur les secteurs angulaires ou l'information est la plus utile.

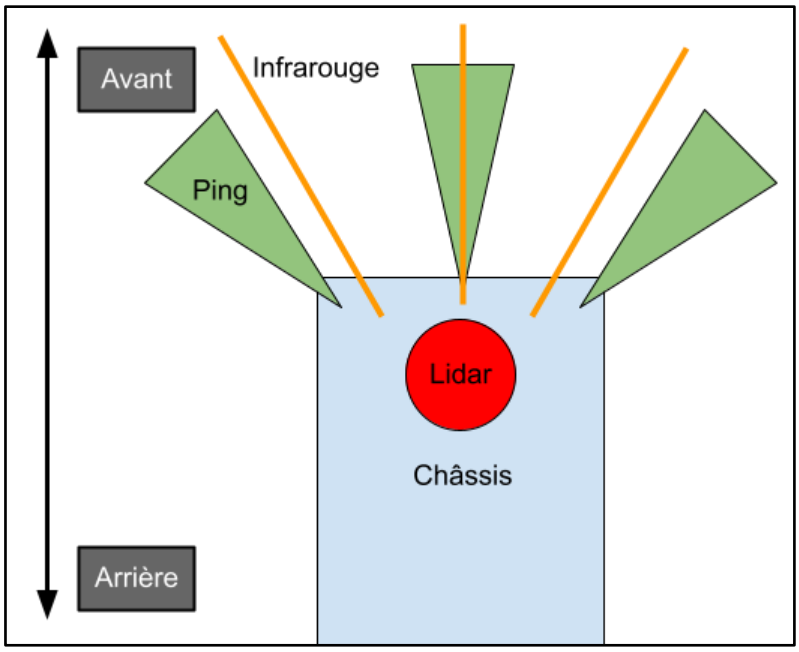


FIGURE 36: PLACEMENT DES CAPTEURS SUR LE CHASSIS

Remarques :

- Dans cette partie nous ne traitons aucunement les algorithmes de récupération des informations des capteurs de proximité. Cependant il existe un répertoire GitHub qui comprend plusieurs fonctions à ce sujet, qui sont détaillées dans les fichiers README correspondant (voir annexe A1).

- Nous n'avons pas abordé le contrôle des moteurs. Cependant des fonctions ont été réalisées et testées et sont disponible sur ce répertoire. L'annexe C explique comment celles-ci ont été réalisées.

- Nous ne parlons pas non plus de possibles capteurs arrière pour nous sortir de situations où il n'y a aucun autre choix que reculer. Cela reste à déterminer : quel capteur, quelle quantité.

- Nous n'abordons pas un processus complémentaire à l'acquisition des données LiDAR. Celui de la complétion des points : si certains angles n'ont pas reçu de distance associée alors cet angle aura comme valeur celle de l'angle précédent.

Le pilotage

- Le traitement des données du LiDAR et des ultrasons

1. Les données du Lidar

Comme vu précédemment, le LiDAR récupère uniquement un secteur angulaire de 180 degrés (figure suivante) : cela correspond à l'avant de la voiture. Pour le format de transmission, les points acquis par le LiDAR sont stockés dans un tableau de 181 cases. L'index du tableau correspond à l'angle de la mesure et, la valeur stockée (pour cet indice) correspond à la distance mesurée.

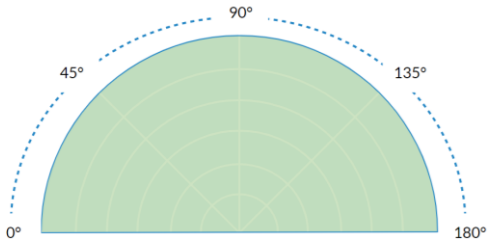


FIGURE 37: REPRESENTATION DE LA PLAGE DE MESURE DU LIDAR

Angle α	0°	1°	2°	3°	4°	5°	. . .	175°	176°	177°	178°	179°	180°
Distance associée	60	65	68	70	84	90	.	113	110	104	85	74	67

FIGURE 38: TABLEAU REPRESENTANT LA DISTANCE EN FONCTION DE L'ANGLE DU LIDAR

2. Les données des capteurs à ultrasons

Les capteurs à ultrasons couvrent une plage de mesure restreinte ce qui impose de les positionner de manière efficace tout en limitant leur nombre. On optimise ainsi la zone utile de ces capteurs de proximité.

Les capteurs à ultrasons agissent comme des détecteurs de présence et renvoient la distance associée à l'obstacle le plus proche. Par conséquent, cette distance renvoyée par les capteurs couvre un secteur angulaire plus large que le LiDAR : cette valeur doit donc être imposée à chaque angle de la plage de mesure.

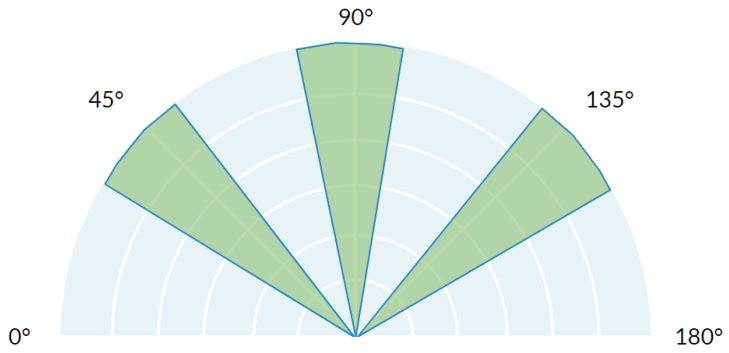


FIGURE 39: PLACEMENT DES CAPTEURS A ULTRASONS EN FONCTION DES PLAGES DE MESURE

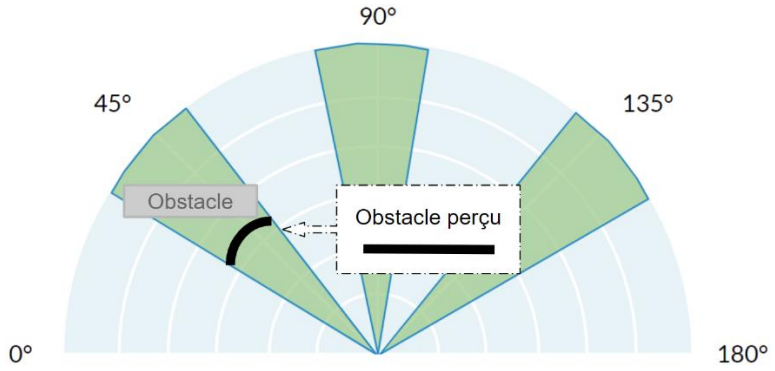


FIGURE 40: COMPORTEMENT D'UN CAPTEUR A ULTRASONS FACE A UN OBSTACLE

Les capteurs à ultrasons couvrent une plage de valeurs (un secteur angulaire). Pour être consistant il est préférable de garder le même format de stockage que celui du LiDAR. La distance mesurée sera donc appliquée à la plage de valeurs du tableau.

Angle α	0° ~ 29°	Plage 1	61°~ 75°	Plage 2	105°~ 120°	Plage 3	150°~ 180°
Distance associée		Distance 1		Distance 2		Distance 3	

FIGURE 41: TABLEAU REPRESENTANT LE STOCKAGE DES MESURES DES ULTRASONS

3. L'association des capteurs

Sur notre véhicule, plusieurs capteurs viennent remplir la même tâche : détecter les obstacles. Il y a donc des conflits de mesures lors de l'association des valeurs du LiDAR et des ultrasons. Il est nécessaire de mettre en place un moyen de trier ces informations en répondant aux questions suivantes :

- Quelle est la tolérance si les deux valeurs fournies divergent peu.
- Faut-il réagir de manière spécifique quand les valeurs divergent beaucoup, par exemple : s'arrêter.
- Quel processus est le plus susceptible d'avoir raison.

De nos choix et réponses à ces questions, les mesures des capteurs à ultrasons viennent recouvrir les mesures du LIDAR quand l'écart de valeurs est trop important.

En effet le LIDAR sera forcément soumis à des oscillations dues aux accélérations positives et négatives.

Angle α	0° ~ 29°	Plage 1	61°~ 75°	Plage 2	105°~ 120°	Plage 3	150°~ 180°

	Distance LiDAR		Distance ultrasons

FIGURE 42: EXEMPLE DE SUPERPOSITION DES MESURES

- ## Le lissage des données reçues

Lors de son acquisition, le LIDAR peut parfois renvoyer des valeurs non cohérentes : c'est à dire que sur un ensemble de point de même ordre de distance, un point est renvoyé à l'écart. Pour corriger ce problème et pour tous les points de mesure, on programme un vérificateur qui récupère les points P_N, P_{N+1} et P_{N+2}.

Si le rapport $\frac{P_{N+1}}{P_N}$ est dans une gamme de ± 5% et que le point P_{N+1} est dissident alors on vient récupérer la distance du point P_N et celle du point P_{N+2}. On prend alors la moyenne de ces deux points et on l'assigne à la valeur de la distance du point P_{N+1}. Ainsi le point P_{N+1} est « équidistant » aux points P_N et P_{N+2}.

- ## L'angle des roues, fonction du traitement des ensembles

1. Changement du format des coordonnées

Pour pouvoir être manipulées, il est nécessaire de récupérer la position cartésienne des points : les données des points fournies par le LiDAR sont en coordonnées polaires. En utilisant les formules de trigonométrie on peut récupérer les coordonnées du point $P(x\,;y)$ en coordonnées cartésiennes.

$$y_{point} = distance * \sin(\alpha)$$

$$x_{point} = distance * \cos(\alpha)$$

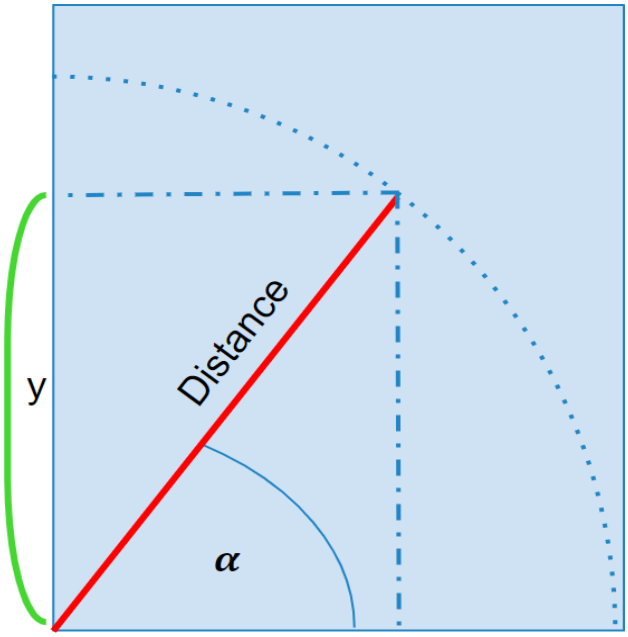


FIGURE 43: REPRESENTATION DE L'ORDONNEE D'UN POINT

Avec leurs coordonnées cartésiennes, il est possible de connaître l'écart entre deux points et également l'écart relatif : c'est à dire la valeur du rapport $\frac{P_{N+1}}{P_N}$ Ce rapport permet de connaître le pourcentage d'écart entre deux points. Ainsi on établit un pourcentage de tolérance qui définit si le point P_{N+1} est suffisamment proche du point P_N. Si c'est le cas les deux points font partie d'un ensemble commun, sinon ils font partie de deux ensembles distincts.

Nous allons illustrer ce concept avec un exemple dont on donne les paramètres suivants :

$$P(\%)_{tolérance} = \pm 5\%,\ Y(P_N) = 100 \text{ cm},\ Y(P_{N+1}) = 103 \text{ cm}$$

Le rapport $\frac{P_{N+1}}{P_N}$ est égale à 103 %.

Soit $Y(P_{n+1}) = Y(P_N) + 3\%$

3% étant inférieur à 5% on considère que ces deux points font partie d'un même ensemble (un mur de la piste par exemple).

2. La notion d'ensemble vide

Nous n'exploitons pas toutes les valeurs que peut fournir le LiDAR, nous y appliquons une saturation numérique. Pour toutes valeurs de distance au-dessus de la saturation, on vient remplacer la distance par -1. Ce changement permet de définir des zones de « vide » c'est-à-dire des espaces sans obstacles proches.

Il est donc possible de différencier les ensembles « vides » des ensembles « obstacles ».

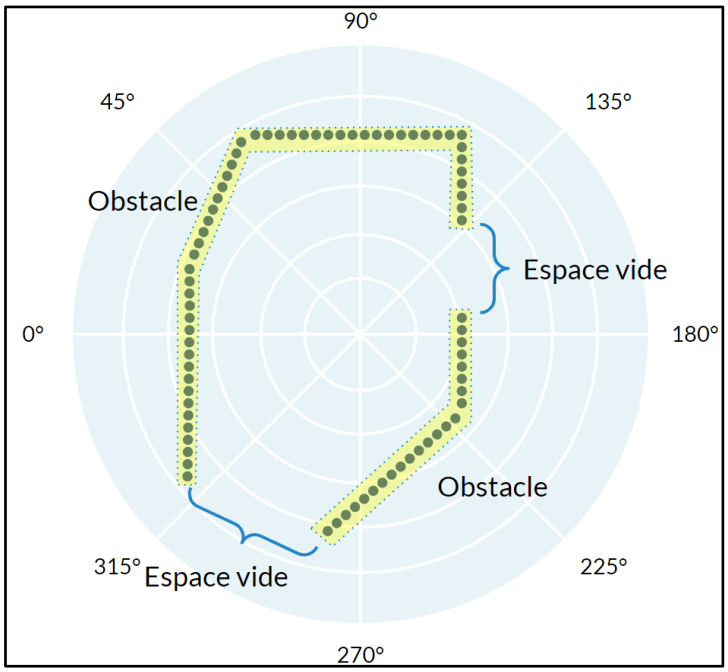


FIGURE 44: : REPRESENTATION DES ENSEMBLES « VIDE » ET ENSEMBLES « OBSTACLE » (COORDONNEES CARTESIENNES)

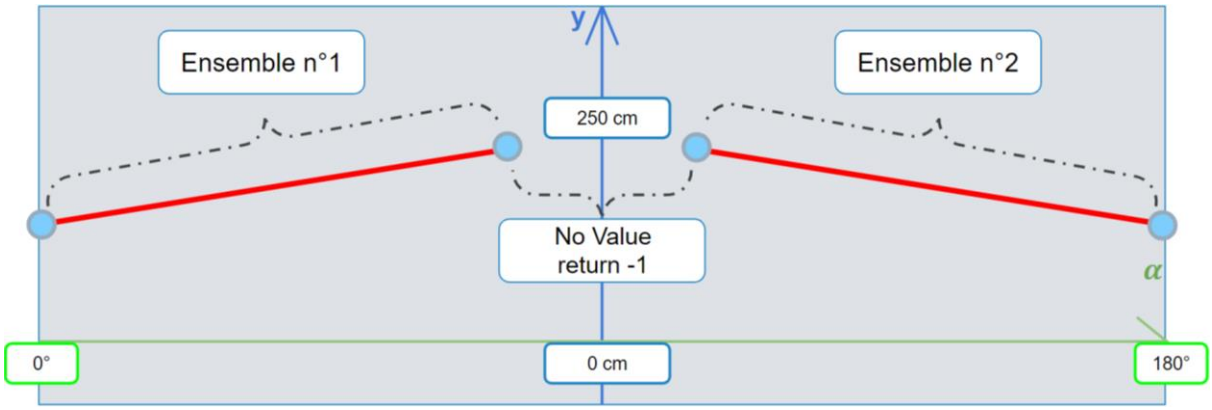
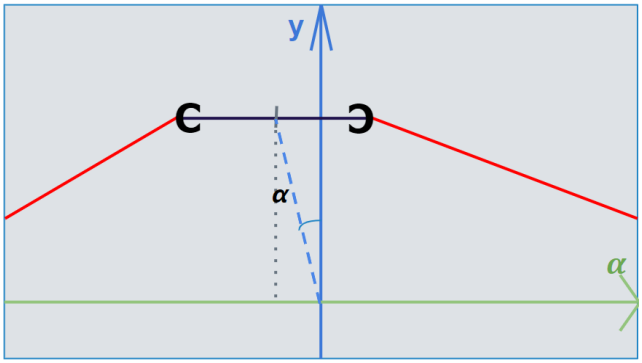
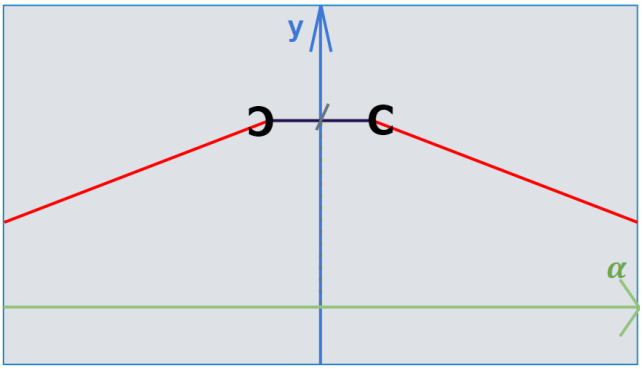


FIGURE 45: : REPRESENTATION DES ENSEMBLES « VIDE » ET ENSEMBLES « OBSTACLE » (CORDONNEES POLAIRES)

Sur l'ensemble de la plage de mesure on vient ainsi rassembler les points en dans un même ensemble. Avec la capacité de différencier les espaces « vides » des ensembles « obstacles », on peut exploiter ces espaces vides. En effet on peut déterminer le milieu de cet ensemble vide. Ainsi comme le montre la figure suivante, selon les coordonnées du milieu un angle se crée par rapport à la normale de la voiture.



L'espace se trouve légèrement à gauche



L'espace se trouve droit devant

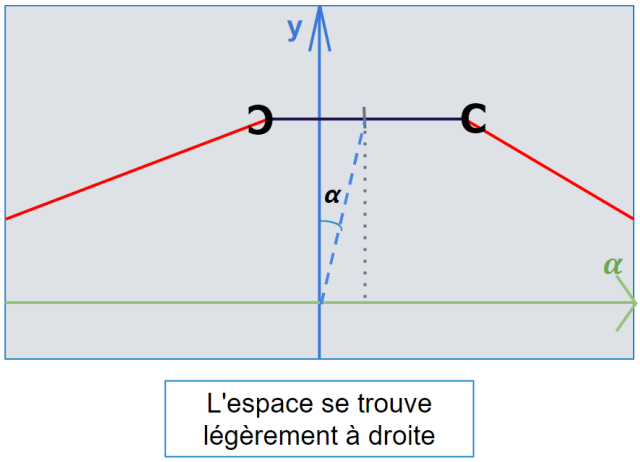
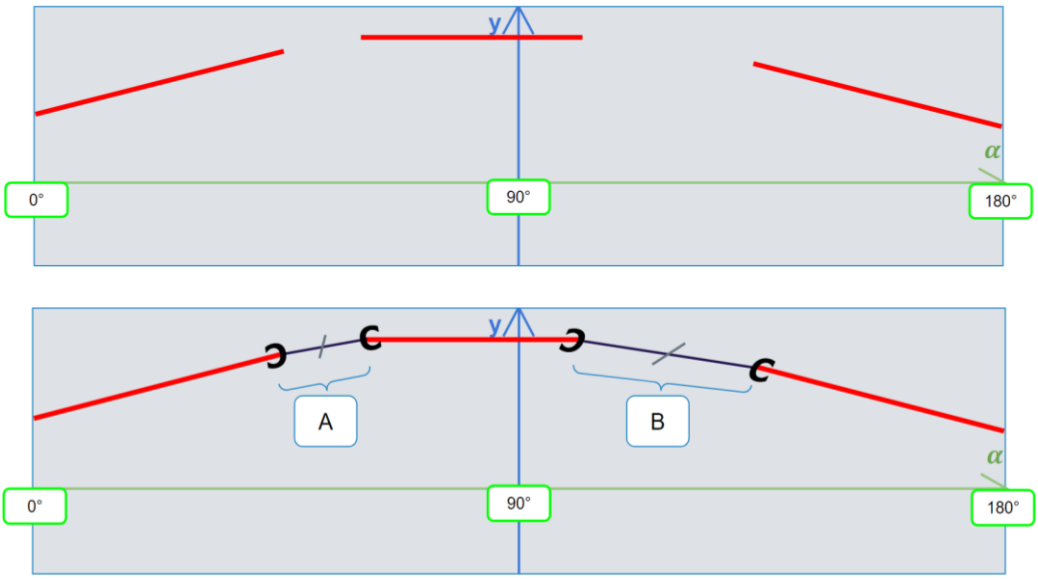


FIGURE 46: : ILLUSTRATIONS DE L'ALGORITHME DE DECISION DE L'ANGLE DES ROUES

Si plusieurs espaces vides se présentent, il est nécessaire de choisir le plus adapté. Pour le choisir, on vient récupérer le nombre de points composant chaque ensemble vide pour connaître leur taille. On se focalise donc sur l'ensemble vide le plus grand. Il est également nécessaire de ne pas fournir un angle supérieur à l'angle de braquage des roues, de ce fait on privilégiera un espace dont $|\alpha_{milieu\_ensemble}| \geq |\alpha_{\max\ \_roues}| = 22°$.



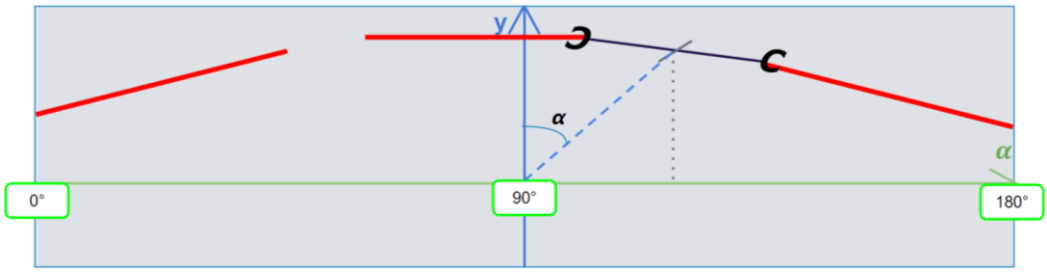


FIGURE 47: ILLUSTRATIONS DE L'ALGORITHME DE DECISION DE L'ANGLE DES ROUES AVEC PLUSIEURS ENSEMBLES VIDES

Enfin pour permettre une rotation sans à-coups, c'est-à-dire sans variations brutales de l'angle des roues, nous avons établi qu'il était nécessaire de mémoriser les valeurs d'angle donnés précédemment afin de réaliser un filtrage. Nous pensions par exemple à un filtre à moyenne glissante ou bien à une saturation numérique de la variation de l'angle.

- La gestion de la vitesse

A ce stade nous n'avons pas atteint cette partie en pratique. Nous allons cependant développer ici une approche théorique.

1. Accélération

Le premier des problèmes que soulève la vitesse est la notion d'accélération. Lorsque la voiture accélère, une rotation s'effectue sur l'axe de tangage et vient perturber le fonctionnement du LIDAR.

Conclusion : De ce fait, il faut numériquement saturer l'accélération et, probablement, la décélération du véhicule.

2. Les virages

Lorsque le véhicule se déplace, celui-ci est amené à prendre des virages. Si sa vitesse est trop élevée, une rotation s'effectue sur l'axe de lacet qui va venir effectuer un survirage (en cas d'accélération) et un sous-virage (en cas de perte d'adhérences au niveau des roues avant).

Conclusion : Il est donc nécessaire que la vitesse du véhicule soit fonction de l'angle des roues, voire de la variation de l'angle des roues.

3. La vitesse maximale.

Lorsque la voiture est en ligne droite, on souhaite qu'elle accélère d'elle-même. Cependant si la vitesse est trop élevée, nous ne pourrons anticiper les obstacles ou les virages.

Conclusion : Il est nécessaire de venir limiter la vitesse maximale de la voiture.

Conclusion

Nous avons été capable de répondre au cahier des charges imposé par l'IUT : à savoir créer un premier prototype de véhicule autonome. Bien que nous ayons eu des problèmes nous avons été à même de comprendre leur origine et de trouver une solution adaptée.

A ce jour, les cartes de communication et de puissance ne sont pas terminées : elles sont en cours d'impressions et de soudures. Les programmes de base de la voiture sont terminés :

- Avancer et tourner.
- Récupérer les données LiDAR.
- Envoyer des informations en temps réel en Bluetooth.
- Prendre des décisions en fonction des données LiDAR, infrarouge et ultrason.

Nous attendons d'assembler le tout pour vérifier le bon fonctionnement global. Nous pourrons alors faire des tests pour mesurer le degré d'autonomie du véhicule dans un terrain semblable à celui de la course.

Ce projet, bien qu'il ait pour objectif principal de mettre en pratique nos connaissances dans le domaine de l'électronique, nous a aussi permis de développer nos qualités humaines et compétences organisationnelles.

De plus il nous a permis d'utiliser le savoir acquis au semestre 3 et de développer diverses compétences : nous avons pu travailler sur divers domaines comme l'informatique et l'électronique embarquée, les mathématiques, la programmation en langage C++.

L'aspect humain du projet est tout aussi important et bien que certaines séances aient été plus difficiles nous avons su nous soutenir et avancer collectivement. Les erreurs que nous avons pu commettre lors de ce projet nous ont aussi été bénéfiques pour apprendre et comprendre : c'est un apprentissage utile à notre avenir professionnel.

Enfin nous avons appris à travailler en autonomie avec un professeur qui a su mettre en avant l'apprentissage autodidacte.

Lexique

TEC : cours de traitement du signal à l'IUT de Cachan.

SA : cours de Systèmes Asservis (automatique) à l'IUT de Cachan.

CE : cours de Conversion d'Energie à l'IUT de Cachan.

IENA : cours d'informatique embarquée à l'IUT de Cachan.

Diagramme de Gantt : Un diagramme de Gantt désigne une présentation, en général un tableau, qui permet de visualiser dans le temps les différentes tâches composants un projet.

Batterie NiMH (Nickel-hydrure métallique) **:** Technologie de batterie. D'autres types de batterie existent comme les batteries au plomb. Les propriétés des batteries varient en fonction des technologies (poids, capacité, puissance nominale).

Capteur LiDAR : Le LiDAR est une méthode de télédétection et de télémétrie semblable au radar. Celle-ci émet des impulsions de lumière infrarouge, au lieu d'ondes radio, puis en mesure le temps de retour après avoir été réfléchies sur des objets à proximité.

Un schéma synoptique : Présentation, en général graphique, permettant d'appréhender l'assemblage électrique d'un système complexe.

CAO : Conception Assistée par Ordinateur. Désigne un logiciel.

CAN : Protocole de communication, qui permet de relier plusieurs systèmes entre eux afin qu'ils communiquent sur un bus de données commun.

Carte électronique / PCB (Printed circuit board) **:** Support, en général une plaque, permettant de maintenir et de relier électriquement un ensemble de composants électroniques entre eux dans le but de réaliser un circuit électronique complexe.

I2C : Protocole de communication informatique.

Bauds : Unité de mesure de la rapidité de communication entre deux systèmes. On associe un Baud à un « bit/secondes ». Les liaisons séries ont des standards de communications en bauds. La valeur par défaut est de 9600 bauds. Cependant il existe d'autres vitesses de transmission comme 115200 Bauds (requis pour communiquer avec le LiDAR).

FIFO (First In First Out) **:** Pour garder une continuité logique entre la transmission et la réception des caractères il est nécessaire de conserver l'ordre. La notion de FIFO pour First

In First Out permet d'acter cette logique de traitement et d'assurer ce mode de fonctionnement : le premier caractère envoyé, donc le premier arrivé devrait être le premier à être traité.

Algorithme : Un algorithme est une suite finie et non ambiguë d'instructions et d'opérations permettant de résoudre une classe de problèmes. Le domaine qui étudie les algorithmes est appelé l'algorithmique.

FPU (Floating Point Unit) **:** Une unité de calcul en virgule flottante est une partie d'un processeur, spécialement conçue pour effectuer des opérations sur des nombres à virgule flottante. Le calcul en virgule flottante est possible avec une unité de traitement arithmétique mais demande une centaine d'instruction machines pour réaliser un calcul.

Annexes

A1 : Guide pour exploiter un répertoire GitHub

Adresse du répertoire : https://github.com/Widelx/saphteamracing.git

Afin d'accéder au travail effectué sur le LiDAR et diverses fonctions d'acquisition, nous avons choisi d'utiliser un répertoire sur GitHub. Le lien précédent permet d'y accéder.

Une fois que vous avez accédé à celui-ci, cliquer sur **Code**.

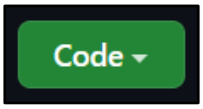


FIGURE 48: UTILISER GITHUB (1)

Un menu déroulant s'affiche. Vous pouvez alors télécharger ce répertoire sur votre ordinateur en cliquant sur **Download ZIP**.

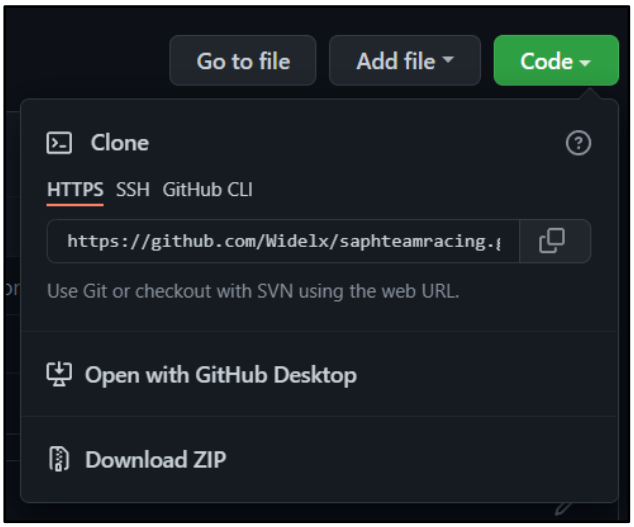


FIGURE 49: UTILISER GITHUB (2)

Vous aurez alors accès à toutes les fonctions qui ont été faites. Vous pouvez bien entendu réutiliser ce code et je vous invite vivement à utiliser Git (ou/et apprendre) pour garder votre code propre. Vous préserverez la version la plus récente qui soit opérationnelle.

A2 : Utilisation d'Octave pour afficher les données LiDAR

Sur ce répertoire se trouve un dossier Octave. Celui-ci contient des scripts afin de visualiser en liaison série ou Bluetooth ce que voit le LiDAR.

Note : Il est possible que d'ici la rédaction de ce rapport ce répertoire ai évolué et que vous puissiez aussi voir l'angle des roues que le programme de prise de décision ordonne. Le fichier README contient les directives à suivre pour utiliser ce script. Si une version plus à jour que celle-ci existe je vous invite à aussi lire ce dernier en complément d'informations.

Téléchargez Octave : https://www.gnu.org/software/octave/download

- Lancez Octave et rendez-vous dans le dossier où se trouve le dossier Octave (téléchargé depuis le répertoire) (1).
- Double-cliquez sur « App_Lidar_Viewer.m » (2).

Une fois exécuté une fenêtre s'ouvre à droite dans l'éditeur.

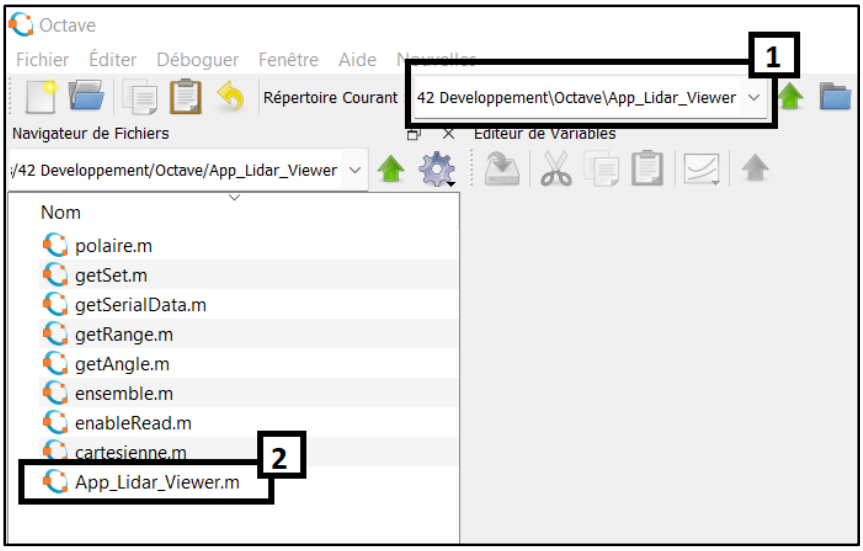


FIGURE 50: UTILISER OCTAVE (1)

- Connectez-vous au module Bluetooth avec votre ordinateur.
- Changez le nom « COM10 » par le numéro du port COM qui correspond (3).
  Cf:
  https://helpdesk.microsurvey.com/index.php?/Knowledgebase/Article/View/1345/6/
- Exécutez le script (4).

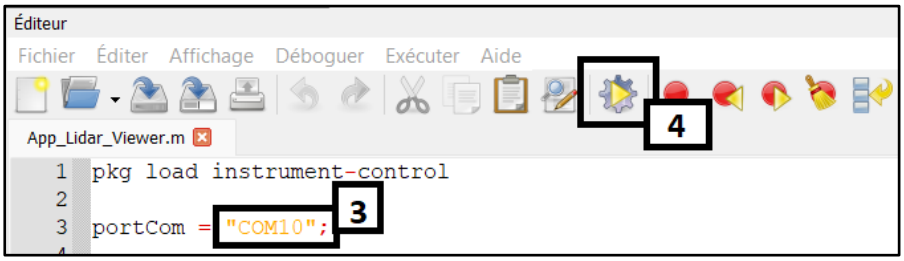


FIGURE 51: UTILISER OCTAVE (2)

Important : Pour arrêter le script il faut se rendre dans la console sur octave et faire « CTRL + C ». Cela arrêtera le script. Fermer la fenêtre qui affiche les points n'arrête pas le script.
Après avoir arrêté le programme il est nécessaire d'exécuter la commande « clear » dans la fenêtre de commande.

Amélioration du programme :
Si l'envie vous prend d'éditer le code voici un tutoriel très complet d'Octave qui est aussi disponible au format vidéo : https://www.jdbonjour.ch/cours/matlab-octave/
L'idée d'ajouter une interface de contrôle pourrait être utile : un bouton d'arrêt de la voiture, un de démarrage, un slider de contrôle de la vitesse/accélération et quelques informations utiles comme le nombre de points par tour ou autre.

Il est bon de savoir que sur Octave le « ; » en fin d'une ligne de commande évite que lors de l'exécution du script, le résultat (de cette ligne) soit affiché dans la fenêtre de commande. De plus la commande « whos » est très utile pour comprendre l'allocation mémoire dynamique.

- ## B1 : Continuité des points

En mode ForceScan il arrive qu'il y ait des sauts d'angle. Cela veut dire que la stricte croissance de l'angle n'est pas assurée. Cela n'a pas été étudié pour la fonction Scan. Ainsi la fonction de synchronisation ne peut pas s'appuyer sur la continuité des points.

Mathématiquement la continuité de l'angle aurait sous-entendu la relation de récurrence suivante.

$Angle_{précédent} < Angle_{suivant}$ hormis pour le passage entre 360 et 0.

On constate que par moment, cette relation est fausse et est remplacée par la suivante :

$Angle_{actuel} < Angle_{précedent} < Angle_{suivant}$

Lors de ces sauts il a été constaté que les sauts n'ont pas lieu sur plusieurs valeurs à la suite.

• B2 : Algorithme plus complexe

Pour pousser l'étude des algorithmes, nous proposons ici un algorithme plus compliqué qui intègre la notion de synchronisation des trames. Celui permettrait deux améliorations considérables :

- Modèle proche d'un fonctionnement dit « multitâche ».
- Réponse quasi immédiate de la fonction « récupérer un tour ».

Par ailleurs, sous la condition que le programme est bien écrit, celui-ci ne devrait pas surcharger le microcontrôleur dans l'exécution de l'interruption sur RX (réception d'un octet). Une étude complète de cette surcharge devrait tout de même être menée pour confirmer cette hypothèse et mesurer le bénéfice réel de ce modèle. Bien entendu des connaissances en multitâche permettrait une mise en place plus robuste d'un fonctionnement similaire.

Note : L'algorigramme (en page suivante), bien que complexe, n'est pas forcément très explicite. Malgré l'effort pour le rendre lisible et simple, nous détaillerons ici les points de fonctionnement qui ne seront probablement pas clair. Par ailleurs cet algorithme n'a pu être perfectionné par manque de temps. Les fonctions les plus utiles seront détaillés mais un réagencement de ces blocs pourrait, probablement, améliorer le modèle proposé.

➢ Synchronisation en temps réel

Ce modèle intègre un algorithme de synchronisation en temps réel qui permet de vérifier la cohérence des trames. Cela assure la robustesse de l'acquisition dans le cas où des trames auraient été perdues, ou, si l'on souhaite décoder les informations du LiDAR et s'y synchroniser sans passer par les trames « 0xA5 0x5A » de début de transmission.

Cet algorithme pourrait déjà être utilisé pour améliorer les modèles présentés précédemment et s'acquitter du modèle « connexion/déconnection ».

Pour analyser les trames on se base sur les éléments cités dans la sous partie « *Désynchronisation des trames* ». On vérifie la cohérence sur N points. Une étude serait nécessaire pour déterminer le nombre de trames N nécessaires pour assurer la synchronisation.

Par ailleurs si l'analyse échoue, une fonction de reconnexion avec le LiDAR devrait être exécutée en employant la méthode « stop/start ».

Bénéfices :

- Assure la robustesse du modèle.

- Réduit considérablement le temps de synchronisation.
- Rend la synchronisation possible à n'importe quel moment.

➢ Alterner le lieu de stockage

Afin de pouvoir répondre facilement à une demande du dernier tour disponible on peut utiliser plusieurs lieux de stockage. Ainsi pendant que l'un se rempli, l'autre peut être lu. En alternant les emplacements de stockage on peut garder le tour le plus récent et continuer l'acquisition du tour en cours. En optimisant le stockage des points on accélère ce processus de stockage. Cependant la partie complétion des points, que nous n'avons pas abordé, est nécessaire uniquement si une lecture est demandée.

Le mécanisme de complétion employé ici est tel que : si un angle n'a pas été associé à une distance lors du tour, nous recopions la distance associée à l'angle précédent. Ainsi il n'est pas nécessaire d'effectuer ce processus si aucune lecture n'est effectuée.

Bénéfices :

- Assure d'avoir toujours le dernier tour enregistré et stocké.
- N'utilise que deux emplacements de stockage de taille minime.

➢ Précautions

Plusieurs précautions sont à prendre avant de se jeter sur ce type d'acquisition.

- Ce modèle repose sur un précepte essentiel : le traitement d'un même tour de façon répétitive n'a aucun sens. Il faudrait donc s'assurer que le même tour n'est pas donné en entrée du traitement plusieurs fois de suite.
- La synchronisation sur trame peut marcher sur le papier. Il faut cependant vérifier si cela est faisable sans analyser un trop grand nombre de points.
- L'alternance des stockages peut poser un problème si on remet à zéro un stockage qui est en cours de lecture. Il faut s'assurer que cela est appréhendé. On peut supposer qu'en utilisant 3 stockages différents et que $t_{lecture} \ll t_{acquisition\ nouveau\ tour}$ cela ne posera pas de problème.

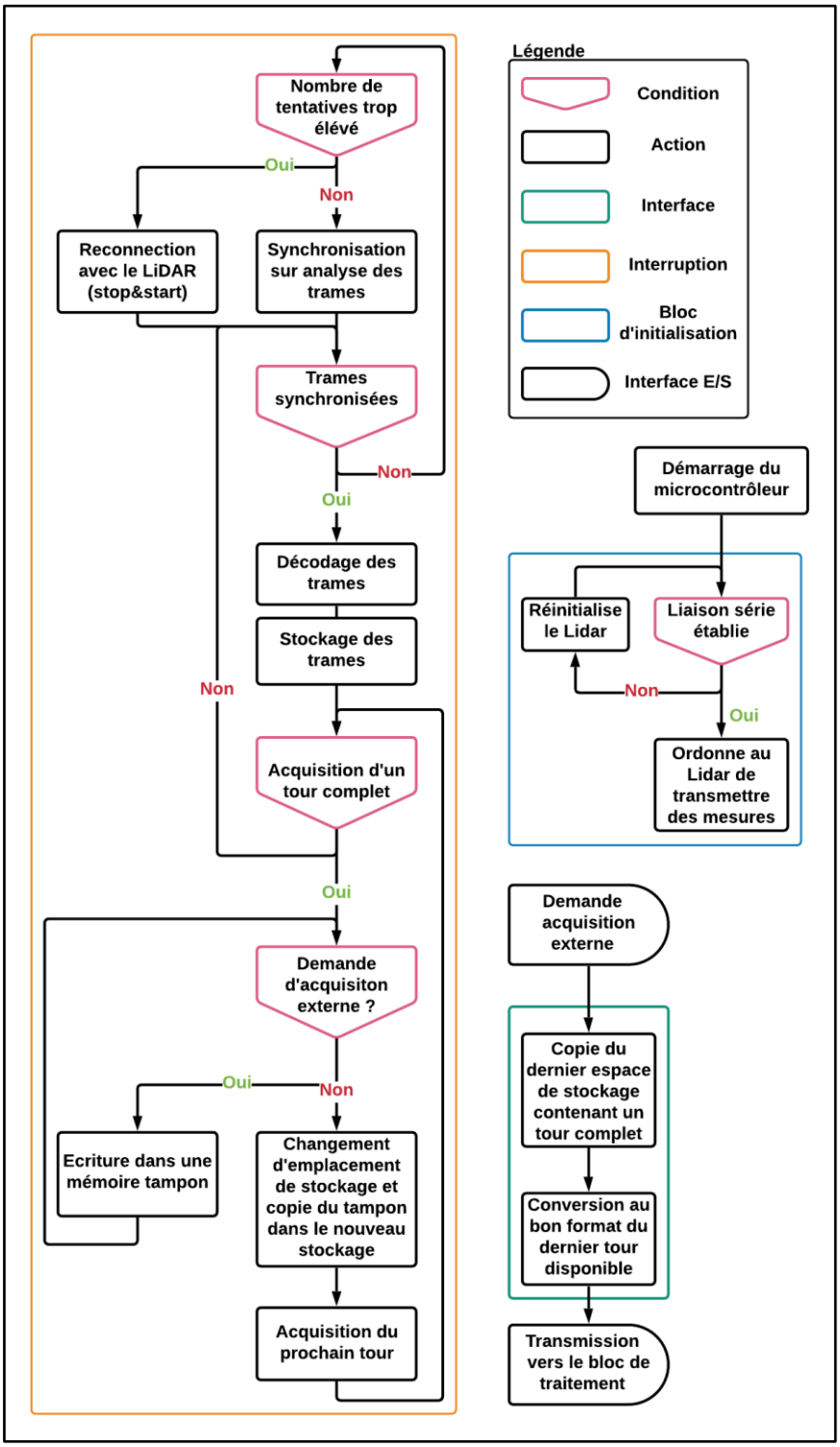


FIGURE 52: ALGORIGRAMME DU MODELE ANNEXE

- C : Gestion de la direction

Pour contrôler les moteurs (servomoteur, moteur brushless) nous devons générer un signal MLI. En effectuant des tests nous relevons les valeurs suivantes de rapports cycliques.

Angle (degré)	-22	22
PWM (rapport cyclique)	9%	6,30%
Vitesse (absolue)	Nulle	Maximale
PWM (rapport cyclique)	8%	5,60%

FIGURE 53: EVOLUTION DES RAPPORTS CYCLIQUES

Nous allons uniquement détailler le procédé utilisé pour générer la commande du servomoteur. Le principe étant similaire pour le moteur brushless.

Nous souhaiterions donc faire une fonction de contrôle de la direction pour laquelle nous fournissons un angle entre -22° et 22° : un angle négatif correspondant à la gauche. Celle-ci prend donc en argument une variable comprise entre -22 et 22° et donnant en sortie une valeur comprise entre 0.09 et 0,063 (rapport cyclique compris entre 0 et 1 sur MBed) qui sera appliquée sur la sortie PWM.

Sur le logiciel GéoGébra nous devons donc tracer deux points :

$$f(-22) = 0{,}09 \qquad f(22) = 0{,}063$$

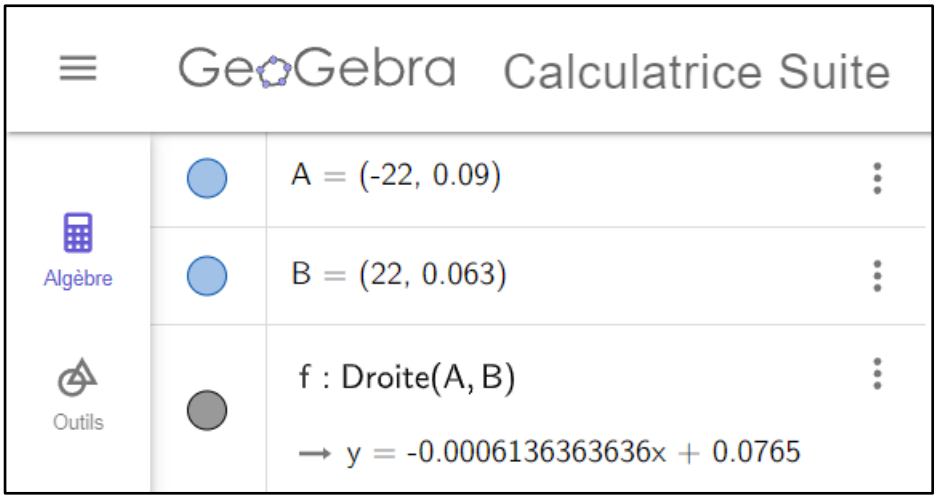


FIGURE 54: DROITE AFFINE GENERANT LE RAPPORT CYCLIQUE DU SERVOMOTEUR

Après avoir entré cette équation nous devons saturer numériquement les valeurs de l'angle pour ne pas envoyer des signaux PWM qui seraient hors de la plage de valeurs acceptée. Nous obtenons alors la fonction suivante en C.

```
void turn(float angle) {
  float anglePWM;
  if (angle > ANGLE_ROUE_MAX) {
    angle = ANGLE_ROUE_MAX;
  } else if (angle < ANGLE_ROUE_MIN) {
    angle = ANGLE_ROUE_MIN;
  }
  anglePWM = -0.0006428571429 * angle + 0.0765;
  turnWheel.write(anglePWM);
}
```

FIGURE 55: FONCTION EN C POUR CONTROLER LA DIRECTION

- ## D : Travaux des différents étudiants

Pour transmettre notre travail, nous l'avons stocké sur un répertoire GitHub. Cela permet de ne pas surcharger l'annexe de ce rapport avec des centaines de ligne de code.

Ainsi notre travail commun se trouve sur le répertoire GitHub suivant, dans le dossier « Documents ». Chaque partie des étudiants se trouve dans le répertoire associé à son nom. Pour l'élève Guillaume Dumesnil, son travail se trouve dans le reste du repository.

Par ailleurs le repository contient d'autres fichiers, comme les datasheets de tous les composants, des modèles STL de pièces 3D, le rapport de l'équipe précédente, afin de rendre la prise en main du projet plus simple.

Adresse du GitHub : https://github.com/Widelx/saphteamracing.git

- ## E : Images annexes

FIGURE 56: PHOTO DE LA VOITURE

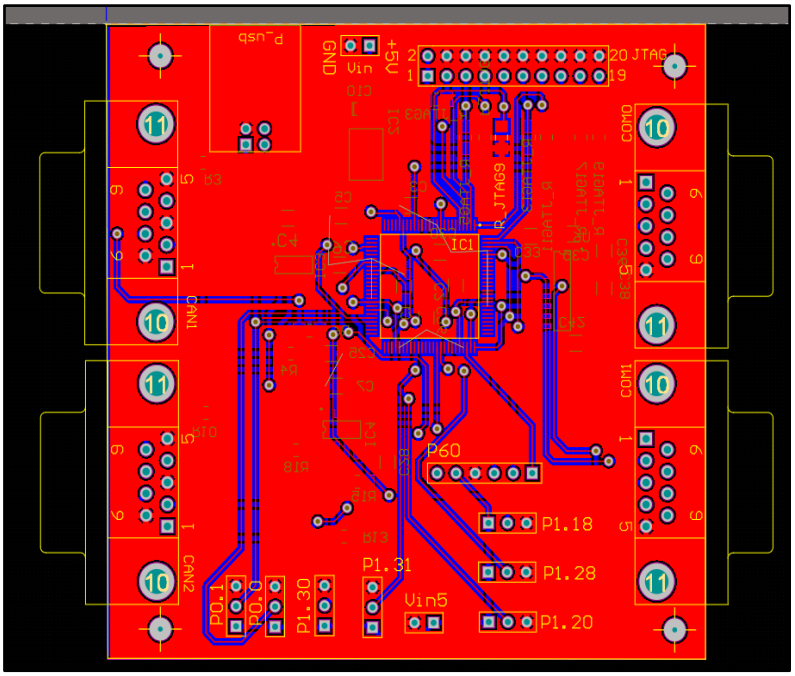


FIGURE 57: COUCHE SUPERIEURE DE LA CARTE PUISSANCE

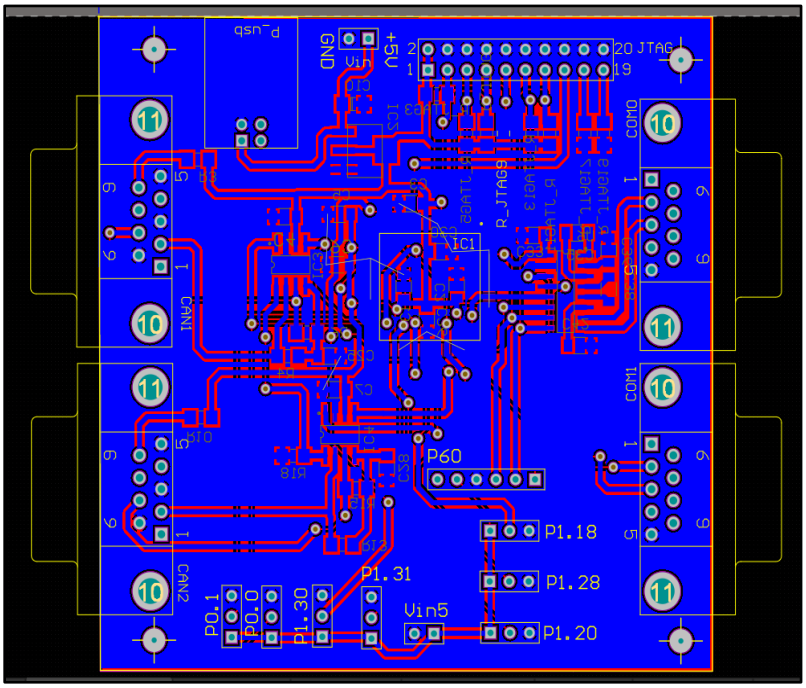


FIGURE 58: COUCHE INFERIEURE DE LA CARTE PUISSANCE

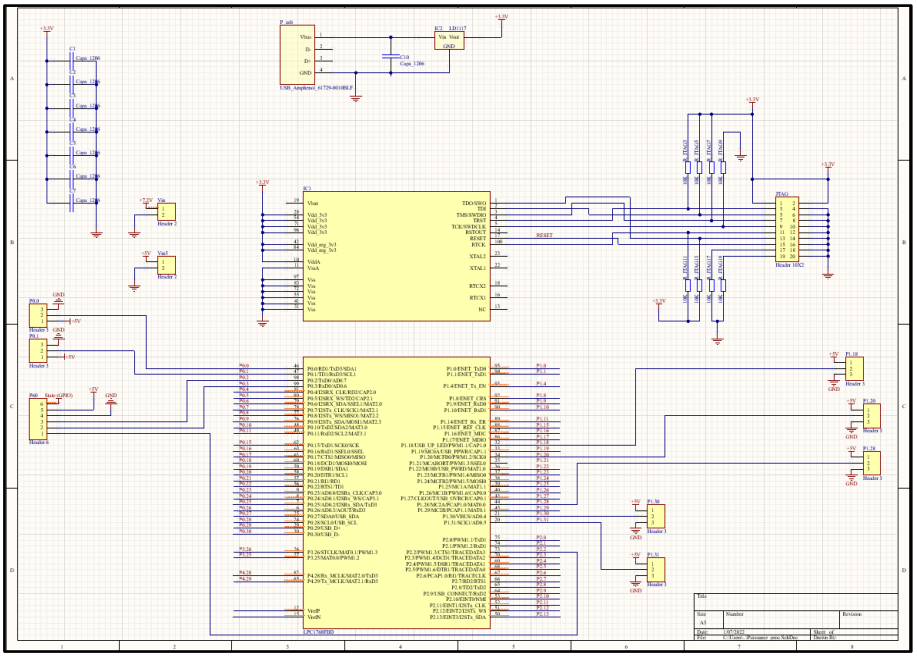


FIGURE 59: SCHEMA PROCESSEUR DE LA CARTE PUISSANCE

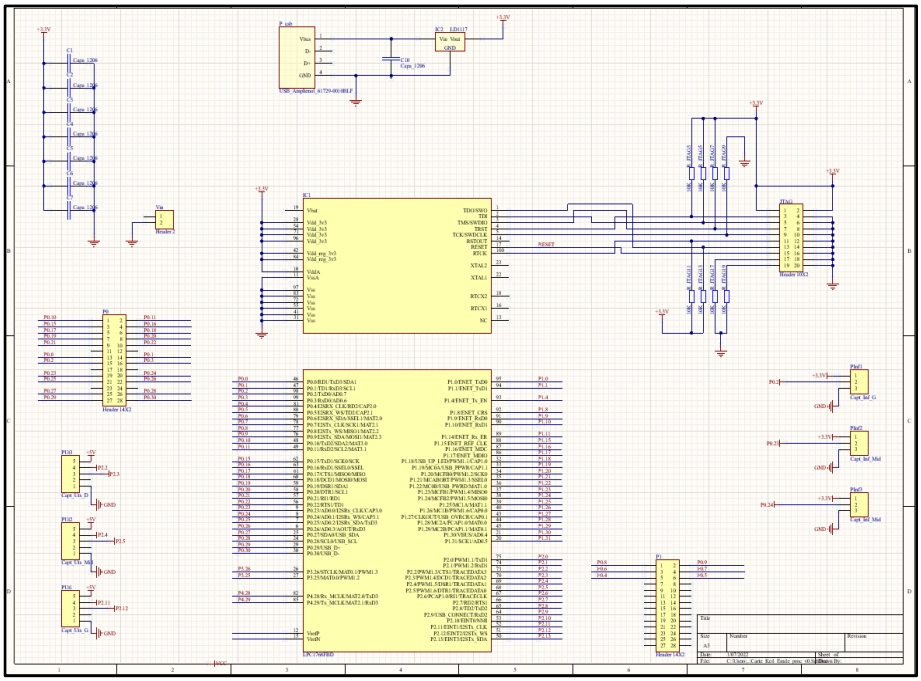


FIGURE 60: SCHEMA PROCESSEUR DE LA CARTE TRAITEMENT

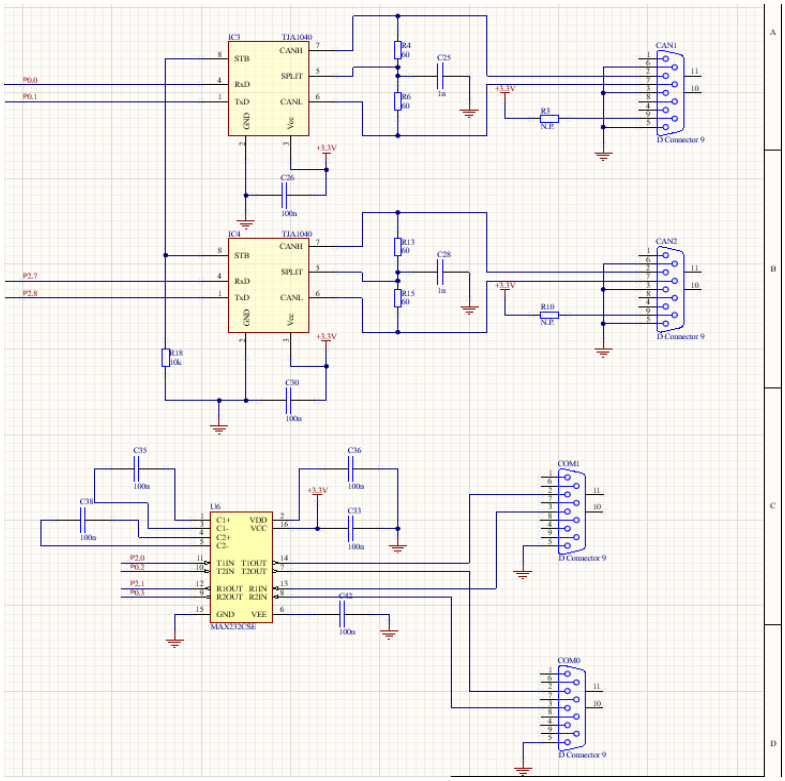


FIGURE 61: SCHEMA DE LA PARTIE CAN

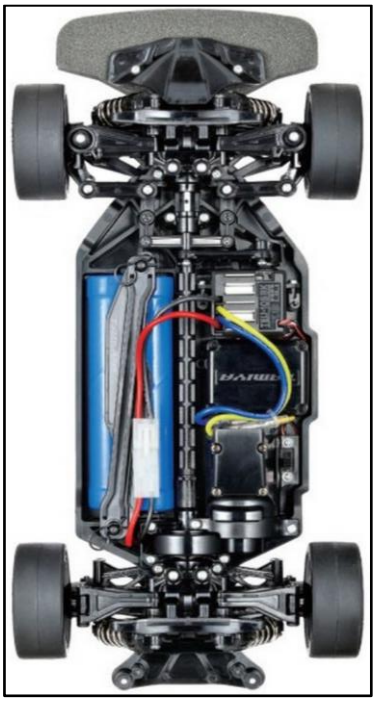


FIGURE 62: MAQUETTE DE LA VOITURE VUE DE DESSOUS

université PARIS-SACLAY

IUT DE CACHAN

Table des figures

Webographie

Diverses définitions

https://fr.wikipedia.org/wiki/Trame_(informatique)#:~:text=Dans%20les%20r%C3%A9seaux%20informatiques%2C%20une,donn%C3%A9es)%20dans%20le%20mod%C3%A8le%20OSI.

https://fr.wikipedia.org/wiki/Baud_(mesure)

DOC LIDAR

https://www.slamtec.com/en/Support#rplidar-a-series

MBED.ORG

https://os.mbed.com/docs/mbed-os/v5.15/apis/index.html

Buffered serial
https://os.mbed.com/docs/mbed-os/v6.14/mbed-os-api-doxy/classmbed_1_1_file_handle.html#a33b0a9b2e4f378134f6c603f28abc926

Serial UART
https://os.mbed.com/docs/mbed-os/v6.6/apis/serial-uart-apis.html

SITE COURSE

https://ajuton-ens.github.io/CourseVoituresAutonomesSaclay/