



Suicide Quad

Projet S4

Dernière Soutenance

Vianney Marticou, Florian Ruiz,
Baptiste Cormorant, David Tchekachev

Professeurs référents : Khelifa Saber et Nouha Chaoued

2023

Table des matières

1	Introduction	3
2	Expériences Personnelles	4
2.1	Vianney	4
2.2	Baptiste	4
2.3	Florian	4
2.4	David	4
3	Origine et nature du projet	5
4	Objet d'études	6
4.1	Synopsis	6
4.2	Pré-requis	6
5	Répartition des tâches	8
6	Achat	9
6.1	Bilan des Matériaux	9
6.2	Fournisseurs	10
7	Électronique	11
7.1	Alimentation	11
7.2	Alimentation des Moteurs	12
7.3	Moteur	12
7.4	Capteurs	13
7.5	Débug	15
8	Mécanique	16
8.1	Conception	16
8.2	Réalisation 3D	17
8.3	Usinage	18
8.4	Impression 3D	18
8.5	Parallélisme des roues	19
9	Software	20
9.1	PWM	20
9.2	Régulateur PID	22
9.3	Horloge et Interruption	22
9.4	Odométrie	23
9.5	Programmation du micro-contrôleur	24
9.6	Asservissement	26
	Asservissement en vitesse	26

Asservissement en position	28
Démonstration des formules	29
9.7 Détection de notre position	32
9.8 Prise Photo	32
9.9 Détection de l'ArUco	33
9.10 Stockage du labyrinthe	35
9.11 Interface Utilisateur	37
10 Communication	39
10.1 Protocole	39
10.2 Direct Memory Access	40
10.3 ESP32	40
10.4 STM32	41
10.5 Serveur	41
10.6 Mise en place	42
11 Site Web	43
12 Remerciements	44
13 Conclusion	45
Sources	47

1 Introduction

Qui sommes-nous ?

Nous sommes des étudiants en informatique passionnés, en quête de connaissances. Notre but avec ce projet est d'apprendre à travailler en groupe, d'apprendre à travailler avec des concepts physiques et mathématiques, dans notre cas un robot. Nous nous sommes posé un défi de taille. Celui de créer un robot de A à Z, de la mécanique au software, qui aurait pour mission de sortir d'un labyrinthe. Il faut préciser que nous n'avons aucune formation en mécanique et électronique. Toutefois, nous avons Internet et la volonté

Pourquoi ce projet ?

Ce projet de robot est un défi technique, on s'est réellement rendu compte de la difficulté à partir du moment où on est rentré dans le concret. Le problème du projet physique est que de nombreux paramètres que nous ne contrôlons pas entrent en jeu et complique notre avancée. Nos connaissances sur les robots, étaient pour la plupart très proche du néant. Ce projet nous a obligé à en savoir plus et à apprendre, sur le tas, un grand nombre de concepts qu'il soit électronique, mécanique ou informatique. De plus, nous savions que ce projet était possible car tout n'est qu'une question de temps

Comment ?

L'atelier Lyon, nous a généreusement permis de stocker notre robot et laisser utiliser leurs machines. De plus, ils nous ont apporté une aide toute particulière sur la conception du circuit imprimé, pièce maîtresse du projet. Nous avons bien sûr parcouru de nombreux sites pour comprendre comment faire ce projet. Nous avons eu l'assistance d'étudiant en 2e année d'ingénieur, pour nous conseiller sur ce projet. Ils nous ont été d'une grande utilité dans le choix des technologies utilisées et de l'implémentassions.

2 Expériences Personnelles

2.1 Vianney

Pendant mon temps libre, j'ai eu l'occasion de programmer des ATmega328p en bare-metal avec et sans Arduino IDE. Je suis en train de découvrir le monde des STM32, les micro-contrôleurs que nous allons utiliser dans ce projet. Je me passionne aussi pour la modélisation 3D, cela m'a permis de créer de nombreux assemblages. J'ai aussi pour avantage d'être une personne extrêmement curieuse et motivée.

2.2 Baptiste

J'ai travaillé sur des projets de création de sites web et d'applications. J'ai ensuite découvert l'univers de l'électronique avec l'utilisation d'Arduino et de Raspberry Pi, que j'ai utilisée dans le cadre de mini-projets. Grâce à ces expériences, j'ai développé une certaine compréhension de la programmation, de l'électronique et de la robotique. J'ai pu mettre ces compétences en pratique dans divers projets personnels.

2.3 Florian

J'ai pu dans le passé utiliser Arduino IDE, et participer à la création d'un circuit imprimé, dans l'association Evolutek dans le but de créer une vitrine pour la coupe de robotique de l'année dernière. J'ai aussi suivi les cours d'arduino de l'année dernière en NTS. Et pour finir, je suis en ce moment sur un projet qui vise à mettre un écran LCD et un haut-parleur sur une carte arduino UNO, pour un cadeau. Je pense donc beaucoup m'amuser durant ce projet et découvrir en profondeur le domaine des robots.

2.4 David

En classe de Terminale, pour les olympiades de SI, j'avais déjà eu à travailler sur un projet plutôt similaire. Avec mon groupe, nous avions assemblé une voiture contrôlée par un micro-contrôleur (Raspberry Pi) et équipée de multiples capteurs, dont des capteurs de distance à base d'ultra-sons, une caméra Pixy pour la reconnaissance du chemin à suivre, d'un module Bluetooth/WiFi pour la communication avec un téléphone ou ordinateur. J'ai donc une petite expérience sur ce genre de projet et j'espère pouvoir l'apporter à ce projet pour le mener à bien.

3 Origine et nature du projet

Notre projet vise à combiner les aspects de la robotique matérielle et de l'informatique en mettant en œuvre des techniques de robotique mobile et de SLAM pour créer un robot capable de se positionner dans un environnement connu, ici un labyrinthe. Cela se fait à l'aide d'ArUco positionné sur tout le labyrinthe, un ArUco est un QR code simplifier permettant d'avoir un identifiant unique pour chacun des ArUco. Ils permettent ainsi de savoir où le robot se situe pour qu'il puisse y naviguer. Il s'agit d'un projet à la fois technique et pratique qui vise à développer les compétences en matière de robotique matérielle, d'électronique et de programmation en C.

En plus de la réalisation d'un robot capable de se déplacer dans un labyrinthe, nous avons également mis en place un système de communication pour envoyer les données récoltées par le robot à un serveur distant. Ce serveur sera utilisé pour faire des calculs complexes que ne permet pas notre robot et d'obtenir les résultats de ces calculs. Cet aspect du projet est particulièrement intéressant car il nous permettra de mettre en pratique les concepts de réseau en C, en utilisant des techniques de communication client-serveur et de lecture de paquets.

Il paraît intéressant de dire que ce projet est un projet d'étudiant qui ne possède pas de formation en mécanique et électronique ainsi il est tout à fait probable que nous soyons passés à côté d'une technologie qui aurait pu nous aider. Nous avons cherché à faire le robot le plus qualitatif possible mais ce n'est pas pour autant que nous y sommes parvenus. Le fait que la robotique soit un métier n'est pas anodin, il faut posséder un grand nombre de compétences dans trois sciences différentes : la mécanique, l'informatique et l'électronique.

L'origine de ce projet est née d'un défi, celui de savoir que ce projet était faisable. "Je peux donc je veux" voici notre raisonnement lors de notre première réunion. Nous savions parfaitement que ce projet avait de grandes chances de finir dans un mur d'où le nom plutôt ironique que nous lui avons donné. De plus nous avons choisi, pour ce projet, de ne faire que des choses de qualité et dont nous pourrions être fiers. On citera pour l'exemple l'architecture de notre code, qui a forcément ralenti notre groupe mais rendu notre code plus facilement lisible et maintenable.

4 Objet d'études

4.1 Synopsis

Le but de notre projet va être de créer de toutes pièces un robot qui a pour mission de sortir d'un labyrinthe. En parallèle, il devra envoyer l'intégralité des données récoltée à un serveur distant. Ce serveur aura pour tâche de faire des calculs complexes, nécessitant une allocation de mémoire dynamique.

4.2 Pré-requis

— Le robot :

Le robot doit se déplacer sur un terrain plat. Le labyrinthe étant représenté par une feuille, on peut considérer qu'il n'y a pas de pente ou d'obstacle sur son chemin. Ce robot doit aussi pouvoir "voir" la feuille et son environnement. Afin de reconnaître, les ArUco (QR code servant de balises). Une autre de ses compétences sera de pouvoir échanger avec le monde extérieur, pour envoyer les données récoltées sur le serveur.

— Le serveur :

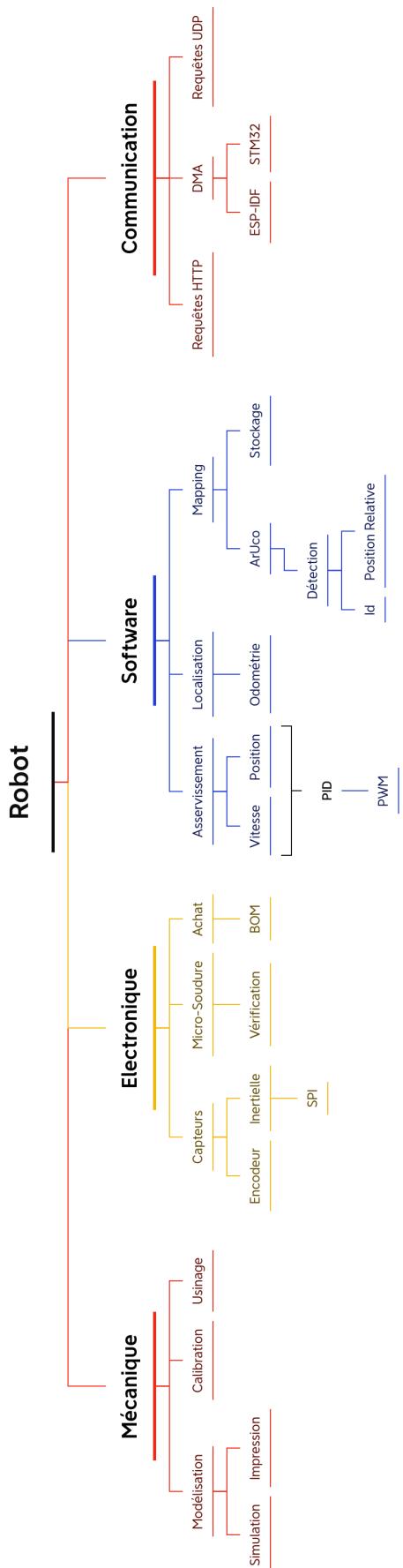
Il doit être capable de gérer la récolte de données via le robot. Ces données seront des images pour connaître sa position dans le labyrinthe et les informations des différents capteurs, afin de vérifier leurs fiabilités.

— Le labyrinthe :

Il est symbolisé par une feuille blanche sur laquelle, on a préalablement dessiné des murs (traits noirs).

— Les ArUco :

Les ArUco nous servent donc de balises. En effet chacun des ArUco a un identifiant unique, ainsi connaissant leurs placements dans le labyrinthe une fois l'identification du ArUco faite le robot peut par la suite aller au centre de l'ArUco et rejoindre la sortie du labyrinthe.



Presented with xmind

FIGURE 1 – Organisation du Projet

5 Répartition des tâches

Pour rappel, voici la répartition de tâches de notre projet tel que nous l'avions mis sur notre Cahier des Charges.

Tâches	David	Vianney	Florian	Baptiste	Soutenance
— Mécanique —					
Cinématique/Modélisation		X	X		1-2
Impression/Usinage		X			1-2
Calibration Moteur		X			1-2
— Electronique —					
Encodeur Rotatif		X			1-2
Caméra		X	X		3
Montage		X			1-2
Communication		X			2-3
— Logiciel —					
Filtre Kalman	X				obsolète
Localisation		X			1-2
Asservissement		X	X		1-2
Mapping	X			X	2-3
Résolution du labyrinthe	X			X	3
— Serveur —					
Communication		X	X	X	2-3
GUI		X		X	3
Recherche du plus court chemin	X			X	obsolète

6 Achat

Un des principaux facteurs à prendre en compte est l'achat des composants. Sans les composants, le projet ne peut pas réellement avancer. Nous avons pu commander la plupart des composants dans un temps raisonnable. Cependant, la réception a pu se faire longue notamment sur l'esp32 qui est une pièce essentielle de la communication.

6.1 Bilan des Matériaux

Dans cette partie, nous allons faire un rapide point sur le coût de ce projet.

Composants	Prix	Quantité	Utilisation
Rotary Encoder	23,46 €	2	Lit la distance parcourut
Nucleo-F446ZE	16,47 €	1	Le cerveau de notre projet
Centrale inertielle	7,11 €	1	Localisation du robot
Resistance (10Ω, 10k Ω, 12k Ω)	0,84 €	24	Élément résistif
Pont en H	12,23 €	4	Gestion des moteurs
Moteur Planetar DC	19,25 €	2	Convertisseur
ESP32	14,90 €	1	Communique et Déetecte
Programmateur FTDI	15,90 €	1	Programme l'ESP32
Roue Folle	6,50 €	1	Support le robot
Traco 24V - 5V	5,63 €	1	Convertisseur de Tension
Traco 24V - 1.8V	4,69 €	1	Convertisseur de Tension
Traco 24V - 3.3V	4,69 €	1	Convertisseur de Tension
Traco 24V - 12V	4,69 €	1	Convertisseur de Tension
Condensateurs 100 μ F	0,16 €	9	Lissage des variations
JST (toutes tailles confondues)	3,57 €	35	Connecte les différents câbles
Circuit Imprimé	8,60 €	1	Circuit Imprimé
Bouton (toutes tailles confondues)	1,25 €	4	Aide au Debug
XT-60 (toutes tailles confondues)	2,54 €	7	Connecte les différents câbles
Support Fusible	1,65 €	2	Sécurise
Dupont F - M & M - M	4,58 €	2	Supporte la Nucleo sur PCB
Condensateur 0.1 μ F	0,14 €	7	Lissage des variations
Total	232.10 €		

6.2 Fournisseurs

Afin de mener à bien ce projet, nous avons dû commander un grand nombre de composants. Voici la liste des fournisseurs utilisés pour ce projet :

- **Mouser** : C'est un fournisseur de composants électroniques, nous avons commandé la plupart de nos composants chez lui. Un autre intérêt de ce fournisseur est qu'il fournit la datasheet des composants, ce qui est fort utile lorsque l'on est en recherche de capteur. On peut étudier et comparer les spécifications de chaque élément.
- **Robotshop** : C'est un fournisseur d'éléments mécaniques, il nous fournit notamment les roues et les moteurs. Nous avons passé de nombreuses heures à comparer les différents éléments que nous allions mettre sur notre robot. Ce site, nous a été d'une grande utilité.
- **JLCPCB** : Son nom l'indique c'est un fabricant de PCB. C'est chez ce fournisseur que nous avons acheté le circuit imprimé du robot. Nous avons dû faire attention pour la commande de ce PCB car outre le fait que ce soit un circuit coûteux. Le circuit venait d'Asie, le temps de livraison est plutôt long.
- **Farnell** : Tout comme Mouser, c'est un fournisseur électronique. Nous avons dû utiliser ce fournisseur plutôt que Mouser. Il y avait certains composants qui demandaient plus d'un an de délai avant la livraison chez d'autres grossistes.
- **Grossiste 3D** : Notre fournisseur de matières premières pour l'impression 3d. Il y a eu plusieurs test de matières premiers. On a testé l'impression en PLA et ABS. Notre choix s'est posé sur le PLA.

7 Électronique

7.1 Alimentation

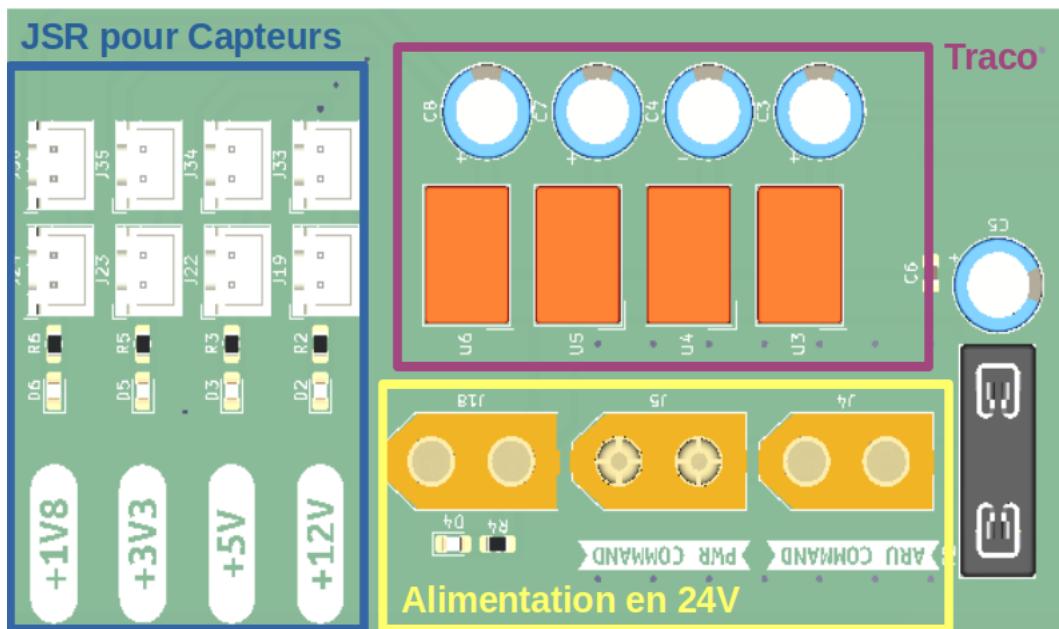


FIGURE 2 – Alimentation générale

Cette partie de la carte permet de faire toute la gestion de l'alimentation du robot. Les blocs représentés en oranges sur la modélisation sont des Traco. Ils permettent de convertir une source de tension, en élevant ou abaissant sa tension. Dans notre cas, nous avons équipé notre robot de :

- Un traco 24V vers 12V
- Un traco 24V vers 5V
- Un traco 24V vers 3.3V
- Nous avons laissé libre l'emplacement du convertisseur 12V vers 1.8V. Pour le moment, aucun de nos capteurs n'est alimenté en 1.8V. Néanmoins nous ne nous fermons aucune porte surtout s'agissant des capteurs.

De plus, nous avons soudé les connecteurs blancs, des JST, ils nous seront utiles lorsque nous devrons alimenter nos capteurs. Nous avons aussi calibré notre fusible en fonction du courant de notre circuit. On retrouve bien entendu des condensateurs de découplage à la sortie de nos tracos afin d'assurer une alimentation lisse et stable.

7.2 Alimentation des Moteurs

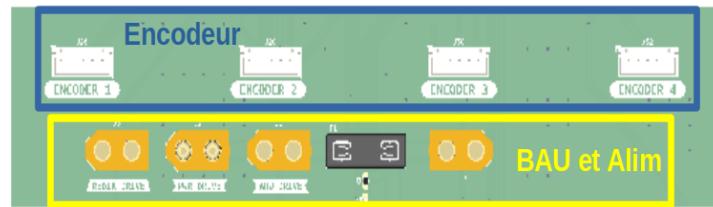


FIGURE 3 – Alimentation des moteurs et des encodeurs

Dans cette partie, il y a 4 connecteurs JST qui permettent de récupérer les données des encodeurs rotatifs. Ils nous seront très utiles pour faire le Régulateur PID. De plus, il y a les connecteurs XT-60 qui ont pour mission de gérer l'alimentation des moteurs. Nous avons aussi un connecteur sur lequel nous connectons un arrêt d'urgence. En effet, nous n'avons pas encore parlé de la partie sécurité de notre projet, mais nous devions absolument pouvoir couper l'alimentation faite aux moteurs.

7.3 Moteur

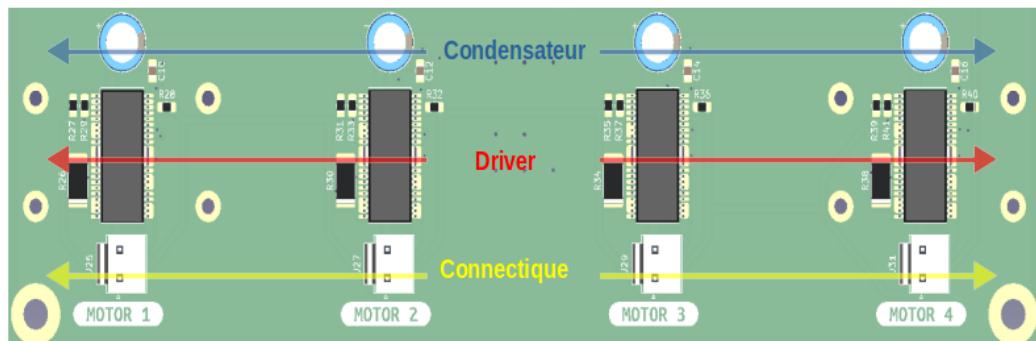


FIGURE 4 – Connectiques des moteurs

Cette zone contient la gestion des différents moteurs, à chaque moteur est associé un composant que l'on appelle un driver. Il permet de contrôler le moteur, une de ces principales tâches est d'inverser la borne - et la borne +. En effet, le moteur est un dipôle dit "réversible". Les guillemets sont nécessaires, car un moteur n'est pas à proprement dit un dipôle réversible dans le sens, ou si l'on inverse les bornes le moteur marchera, mais dans le sens inverse.

Pour le bon fonctionnement du circuit, nous avons mis un condensateur, appelé condensateur de découplage, qui permet d'être certain que le signal soit propre. Pour les résistances, ce sont des prérequis posés par le composant en lui-même et l'on peut retrouver cette information dans la datasheet du composant.

7.4 Capteurs

Circuit imprimé

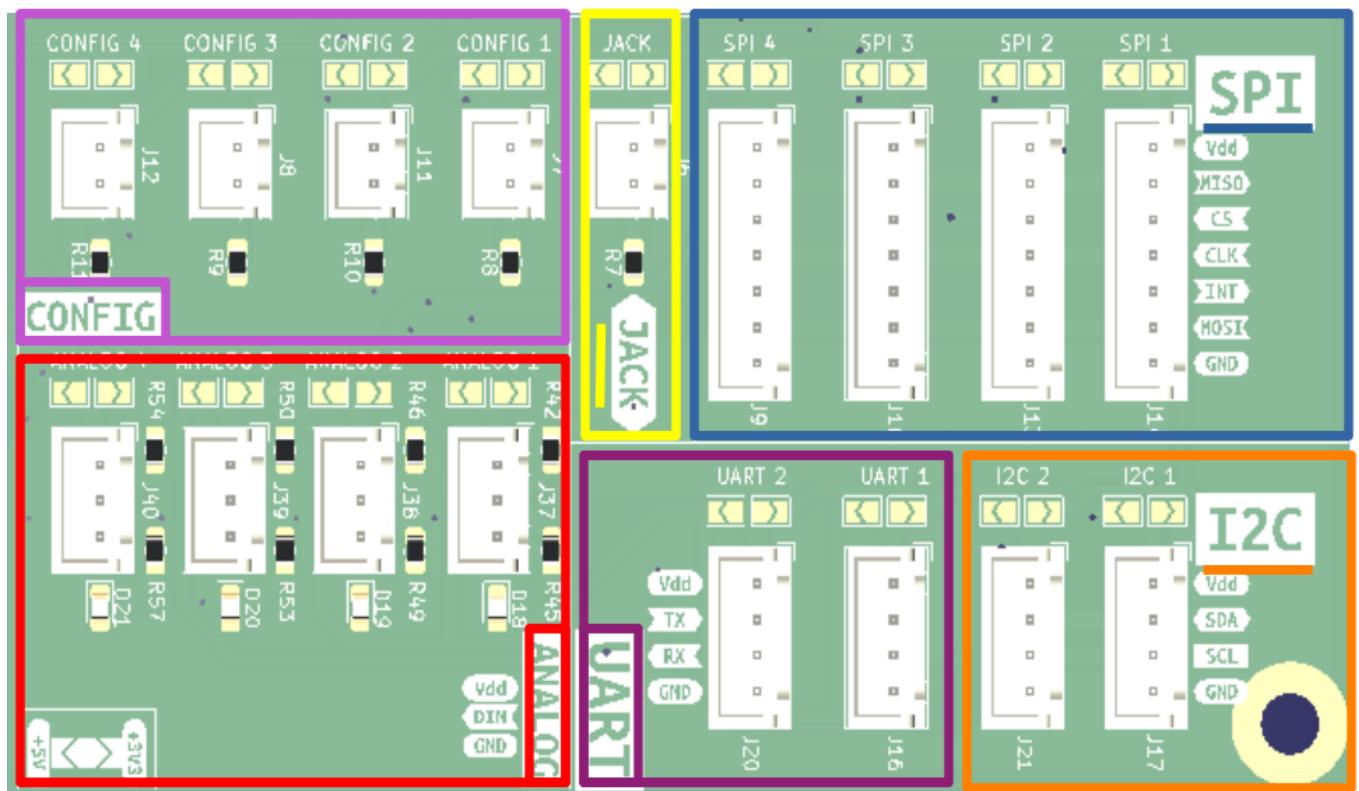


FIGURE 5 – Connectiques et alimentation des capteurs

Cette partie de la carte est la zone, la plus "future proof". En informatique, il existe un grand nombre de composants qui permettent de faire une infinité de choses. Cependant, le revers de la médaille n'est que pour une multitude de possibilités, une multitude de moyens de communication. Heureusement pour nous, il n'existe qu'une poignée de moyens de communication (du moins répandu). Nous appelons donc les passés en vue et voir une des utilisations possibles de ces protocoles.

- **SPI** (Serial Peripheral Interface) : C'est un mode de communication qui repose sur le principe de maître - esclave. Le maître contrôle la communication. Ce protocole a l'avantage de synchronisé les deux composants sur l'horloge et d'avoir un meilleur débit de l'UART et que le I2C. Cependant, il monopolise plus de Pin que ces derniers.
- **UART** (Universal Asynchronous Receiver Transmitter) : Comme son nom l'indique, c'est un protocole qui repose sur une communication asynchrone entre un émetteur - et un receveur. L'UART repose sur des vitesses de transmission particulière, par ex : 9600 bauds.
- **I2C** (Inter-Integrated Circuit) : Ce protocole reprend le concept de maître - esclave du bus SPI, mais il rajoute le fait qu'un esclave peut prendre la position de maître, en inversant les rôles.

- **JACK** : C'est un canal qui achemine des signaux logiques.
- **ANALOGIQUE** : De même, c'est un canal qui achemine des signaux analogiques.

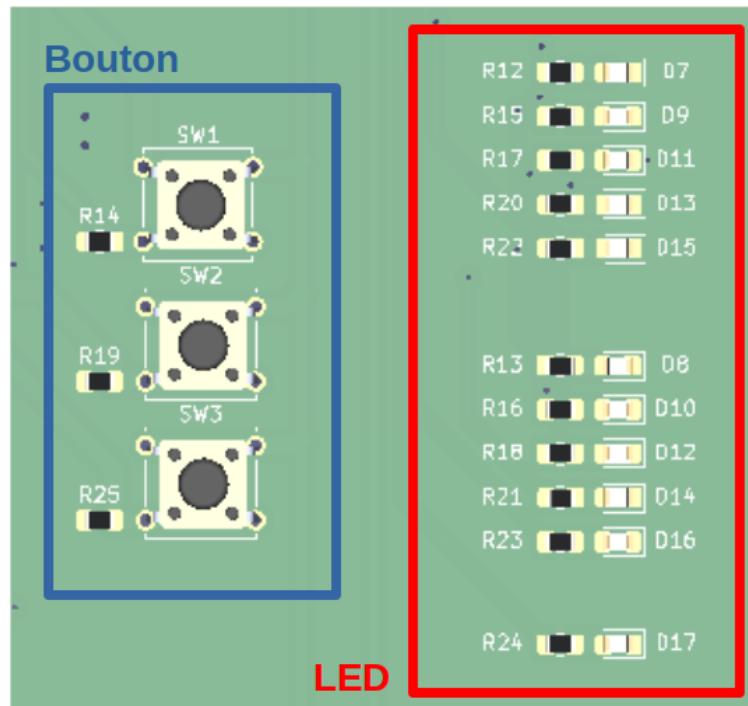
Types de capteurs

Pour détecter nos mouvements nous avons décidé de nous diriger vers un ensemble de 2 capteurs, des encodeurs rotatifs et une centrale inertuelle. Les encodeurs rotatifs nous indiquent le nombre de tours quelle à effectuer. Ainsi on peut calculer les déplacements de notre robot. Pour la centrale inertuelle, elle nous renvoie une accélération et une vitesse angulaire. Or l'accélération étant une dérivée de la vitesse, on peut aisément calculer la vitesse et la position de notre robot grâce à cette dernière.

Pour la prise de photo nous avons choisi une OVE2640 fourni avec un ESP32 qui lui sert de module et qui permet la communication de notre robot.

7.5 Débug

FIGURE 6 – Boutons et leds de Débug



Ce morceau a pour seul et unique but de faire du débug sur la carte. Nous avons trois boutons que l'on lie avec des actions particulières. Ainsi que 11 Leds de couleurs différentes qui nous permet d'afficher des informations. Par exemple, un niveau de criticité, si les receveurs des protocoles sont connectés ou bien juste montré que le robot est allumé.

Résistance

Nous avons mis, sur cette partie du robot, un certain nombre de résistances. Pour les résistances des Leds, elles permettent de ne pas faire brûler la Led.

Pour les résistances des boutons, ce sont des résistances de pull down. Elles servent à maintenir la tension à 0V lorsque le bouton n'est pas pressé.

8 Mécanique

8.1 Conception

Pour ce projet, nous avons dû concevoir un robot. Pour cela, nous nous sommes fixé un certain nombre de prés requis, pour pouvoir créer notre cahier des charges. Pour la partie mécanique de notre robot, voici les contraintes majeures :

- Supporter un PCB
- Poser une batterie
- Être le plus léger possible
- Soutenir la caméra
- Être le moins d'encombrant possible
- Avoir une bonne répartition de la masse

Une fois notre cahier des charges fixées, nous nous sommes réunis afin de voir les idées de chacun puis de choisir notre meilleure alternative. Nous avons donc choisi un robot deux roues accompagné d'une roue folle. Cela facilite la direction du robot ainsi que sa création.

Nous avions d'abord pensé à faire un support entièrement imprimé en 3D mais après réflexion, nous sommes rendus compte que nous devions pouvoir poser une batterie ainsi que la caméra sur notre robot. De plus, l'impression 3D n'est pas une technologie rapide. Ce genre de châssis aurait mis plus d'une 30h à être imprimé.

Notre choix s'est finalement porté sur un simple châssis en bois. Sa facilité de conception, nous permet aisément de pouvoir rajouter des modules tels qu'une caméra ou bien d'autres capteurs (ex : un lidar). Cependant bien que le châssis soit fait en bois, nous avons fait une conception quasi globale de notre robot en 3D et certaines parties ont été imprimé car impossible à faire en bois.

8.2 Réalisation 3D

Afin de voir la viabilité de notre robot, nous avons modélisé une grande partie de ses pièces en 3D. Tout cela dans l'optique de pouvoir réaliser un assemblage virtuel de notre projet. On peut dire que nous avons créé un digital twin de notre robot.

Ce domaine nous a donné un peu de fil à retordre, un des principaux problèmes était que l'on devait créer des pièces qui devaient pouvoir résister aux contraintes suivantes :

- Le poids du robot, c'est une information qu'il ne faut négliger notre robot va contenir de nombreux composants dont des batteries.
- La vitesse angulaire de notre robot, nous avons pris des moteurs qui sont légèrement sur dimensionné pour notre projet. Maintenant nous allons devoir assumer cela.
- La faisabilité de ces pièces, nous sommes nos propres fabricants. Et malheureusement pour nous, nous n'avons pas toutes les machines et connaissances du monde. Pour faire nos pièces, nous avons opté pour l'impression 3D, bien que cette méthode contienne de nombreux désavantages

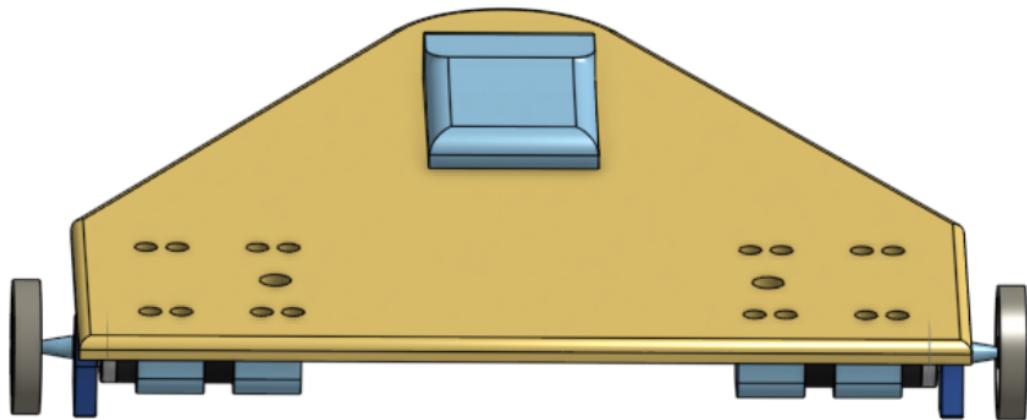


FIGURE 7 – Modélisation 3D du robot

8.3 Usinage

L’usinage est le fait fabriquer une pièce. Nous avons dû fabriquer un certain nombre de pièces pour ce projet. La plupart de nos réalisations ont été faites en 3D puis imprimé sur place. Toutefois, nous avons aussi dû modifier nos roues. En effet, nos roues initiales avaient un diamètre interne de moyeu de seulement 5mm. Or nos moyeux de moteur avaient un diamètre externe de 6.2mm.

Pour régler ce problème, nous avons mis nos roues sous une perceuse à colonne et pris une mèche métal de 6mm. Une fois le trou fait, il nous a suffi de passer un petit coup de Dremel pour gagner le 0.2m restant.

Malheureusement dans notre réparation, nous avons détruit le filet de la vis qui permettait de serrer le moyeu. Il a fallu que l’on rachète une nouvelle roue. Cela nous a fait perdre un précieux temps pour la calibration de PID.

8.4 Impression 3D

Nous avons créé un châssis en bois pour notre robot mais certaines pièces ont été imprimées en 3D. On peut citer nos premiers adaptateurs de moyeu (avant le perçage de nos roues), les supports moteurs. L'impression 3D nous a permis de créer des pièces qui nous étaient impossibles de créer en bois ou tout autre matériau. Je pense notamment aux supports des encodeurs rotatifs.

Pour le choix du filament, nous avons dû faire des tests entre le PLA et l'ABS. Il a suffi d'imprimé deux fois la même pièce dans nos deux matériaux puis de comparer les deux résultats. Nous avons étudié la capacité mécanique, la précision du trait, leur coût ainsi que leur temps d'impression. Finalement avec cette liste de critère, nous avons choisi le PLA pour sa rapidité d'impression et son coût.

8.5 Parallélisme des roues

Le parallélisme des roues est un sujet extrêmement important sur un robot terrestre. En effet, un robot qui ne va pas droit n'est pas un robot fonctionnel. De plus, dans notre cas, nous avons une problématique de taille, l'usinage de nos roues. En effet, comme dit plutôt, nous usinons nos roues nous-mêmes afin de pouvoir les utiliser avec nos moteurs. On a beau essayé de faire attention à garder un trou perpendiculaire avec notre perceuse à colonne, nous avons percé un trou qui n'est pas parfaitement droit.

Cela pose un réel problème surtout sur les grandes distances et les hautes vitesses. Notre robot dévie énormément de sa trajectoire initiale. Pour régler cela, nous avons implémenté un régulateur PID, qui permet de réguler la vitesse d'une roue par rapport à une autre. En théorie, cela permet de régler l'intégralité du souci. Cependant c'est en théorie, en pratique si nos deux roues essayent de dévier de leur trajectoire le régulateur suivra la déviation.

Une idée de résolution

Une solution qui aurait permis de résoudre ce problème en pratique aurait été d'avoir plusieurs capteurs de position. Seulement nous n'avons pas réussi à utiliser notre centrale inertie. Nous nous sommes rendu compte après investigation que ce capteur fonctionnait avec une tension moyenne de 1.8V et non 3.3V. Avec les coordonnées de cette centrale inertie et un filtre de Kalman, nous aurions pu avoir un positionnement de bonne qualité et ainsi avoir une auto-correction de notre trajectoire.

9 Software

9.1 PWM

Certains composants nécessitent une tension spécifique. Par exemple, un moteur avance en fonction de sa tension. Il faut donc l'alimenter en conséquence pour gérer sa vitesse. Nous devons donc trouver un moyen de piloter une tension de manière fiable et rapide. Cette solution s'appelle le PWM.

Explication

Le PWM est un acronyme anglais qui signifie Pulse Width Modulation, ou Modulation de Largeur d'Impulsion (MLI) en français. Le but de ce procédé est de produire un signal d'une valeur donnée avec uniquement deux états possibles, 0 ou 5V. Nous voyons déjà que l'on ne va pas pouvoir produire ce signal à un instant T, nos signaux de bases étant 0 et 5V. Mais si nous ne pouvons pas le produire à un instant T, nous le pouvons sur une période.

Et voilà, toute la subtilité de ce procédé. Nous allons produire un signal qui en moyenne à une valeur donnée. Pour faire cela, nous devons connaître le cycle de notre PWM, la valeur max ainsi que la valeur voulue. Une fois cela connu, un simple produit en croix suffit pour savoir le nombre de Tick à activer dans un cycle, pour avoir notre tension.

Fonctionnement

Lorsque l'on parle de PWM, on utilise souvent le terme de rapports cycliques. C'est un pourcentage qui représente la proportion de Tick à 5V dans un cycle.

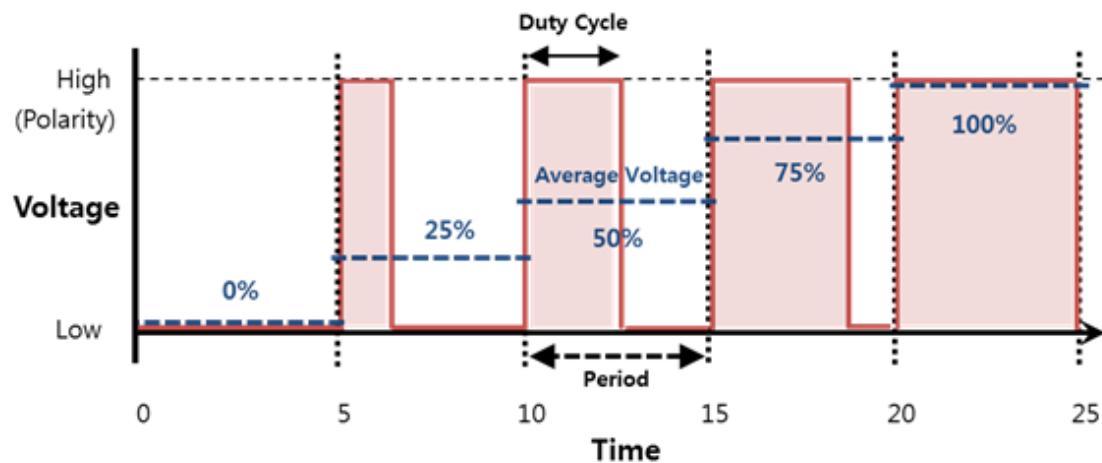


FIGURE 8 – Voltage Moyen de la période

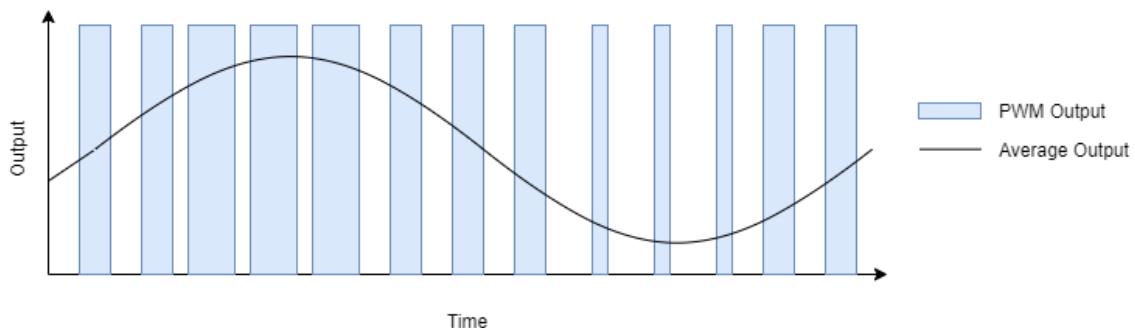


FIGURE 9 – Courbe moyenne d'un PWM

9.2 Régulateur PID

Un régulateur PID est un outil qui permet l'asservissement de notre robot, c'est un élément indispensable à tout robot, en effet il est à la base de ses mouvements et de sa robustesse. Son principe est de réguler la consigne qui est envoyée aux moteurs, enfin que le robot atteit un objectif progressivement et qu'ils s'y stabilisent. Il permet aussi que notre robot reste à une position fixe même s'il est poussé et d'avancer à une distance donnée même s'il est retenu pendant sa trajectoire. Ceci permet en cas de ralentissement inopiné de notre robot qu'il conserve sa trajectoire et que son objectif soit atteint.

Le régulateur PID est donc un outil qui permet cette régulation. Il se repose sur des calculs basés sur la différence entre l'objectif voulu et la valeur réelle mesurée (l'erreur de la consigne) et la valeur réelle mesurée. Le calcul consiste à prendre une valeur Proportionnelle de l'erreur multipliée par un coefficient, d'y ajouter une valeur Intégrale des erreurs (leurs sommes) multiplié par un autre coefficient et d'y ajouter une valeur Dérivée de la valeur mesurée (la différence entre les deux dernières valeurs mesurées) multipliée par un dernier coefficient. D'où le nom de régulateur PID (Proportionnelle Intégral Dérivé).

9.3 Horloge et Interruption

Le STM32 possède un grand nombre de Timer, ce sont des horloges que l'on peut configurer via un prédiviseur et une période. Un prédiviseur est un facteur de division de l'horloge interne. De base, l'horloge interne de notre carte est de 168MHz. Ainsi le prédiviseur permet de ramener l'horloge interne à une fréquence plus raisonnable et plus utilisable. Une fois, le prédiviseur fixé, nous pouvons configurer la période de l'horloge, en Tick.

Ce timer va nous permettre de pouvoir appeler une fonction à un intervalle précis. Pour cela, nous devons juste réécrire une fonction particulière qui se nomme HAL_TIM_PeriodElapsedCallback. Cette fonction est appelée par toutes les horloges définies à la fin de chaque période. Une fois la fonction appelée, il suffit de vérifier quelle horloge appelle la fonction et de lancer notre fonction.

Listing 1 – Fonction de Callback

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef* htim)
{
    if (htim == &htim9)
    {
        // Any instruction
    }
}
```

9.4 Odométrie

Calcul de la position et de l'orientation du robot

Grâce au régulateur PID, notre robot a un moyen supplémentaire de calculer sa position et son orientation dans le labyrinthe. Pour effectuer ces calculs, on distingue 2 types de déplacement, les déplacements horizontaux et angulaires.

Quand notre robot avance ou recul on connaît à chaque fin d'appel de notre fonction PID la distance parcourue depuis le dernier appel. Ainsi avec un simple calcul on retrouve la nouvelle position de notre robot dans le labyrinthe. Quand notre robot tourne on connaît aussi la distance parcourue par les roues, on peut ainsi y calculer l'angle de rotation qu'effectue notre robot et y ajouter à l'angle de rotation actuelle pour y trouver notre nouvel angle. De plus quand il tourne, la rotation étant calculée pour que le centre du robot ne bouge pas, aucun changement de position n'est à calculer à ce moment-là.

Ce système empêche à notre robot d'effectuer des arcs de cercle, mais augmente grandement notre précision et simplifie les calculs. Effectivement, si notre robot effectuait des arcs de cercle, il y aurait fallu calculer un changement de position et de rotation simultanément, en augmentant les calculs utilisant PI et les fonctions trigonométriques (\cos , \sin , ...) et qui accumule les uns sur les autres baisse énormément la précision de nos résultats.

Finalité

Notre algorithme est capable d'indiquer à notre robot la vitesse à laquelle il doit aller pour avancer ou tourner. Avec les valeurs que nous avons l'ordre de grandeur d'erreur de parcours est entre le millimètre et le micromètre.

Toutefois ceci n'est pas un problème majeur car il connaît son erreur par rapport à la distance ou l'angle voulu, ainsi quand on calculera sa position et son orientation dans le labyrinthe, il n'y a pas de problème car il rajoute l'erreur à la distance désirée.

9.5 Programmation du micro-contrôleur

Compiler pour notre architecture

La compilation de notre code se fait via le Makefile produit par STM32CubeMx (le logiciel qui nous permet de configurer les différents pins). Toutefois, nous avons quand même dû modifier ce Makefile pour y ajouter nos fichiers C. En effet, le logiciel génère un Makefile en "hard codant" les différents fichiers C nécessaires. Il a suffi de rajouter manuellement le nom de nos fichiers dans la liste cible.

Pour la compilation en tant que telle, nous devons utiliser un autre compilateur que GCC. En effet, le STM32 n'est pas une architecture x86 ou x64 mais une architecture ARM. Ainsi nous utilisons le compilateur arm-none-eabi-gcc ceci explique le fait que nous ne puissions pas compiler notre code sur un ordinateur de la salle machine.

Un des problèmes que pose ce Makefile est l'intégration de librairie. La nécessité de renseigner à la main l'intégralité des fichiers C n'encourage pas l'utilisation de librairie. Heureusement pour nous, la philosophie de notre projet S4 est l'utilisation du moins de librairies possibles. C'est pour cela que nous n'avons pas utilisé de librairie sur le STM32, bien qu'une librairie mathématique ait été fortement utile.

Flasher le STM32

Une fois, le code compilé, nous devons flasher notre code sur le micro-contrôleur. Il faut savoir que le STM32 que nous utilisons possède déjà un programmateur intégré. Cela signifie que nous n'avons pas besoin d'une carte externe (un programmateur) pour pouvoir flasher le microprocesseur. Le programmateur est la partie supérieure de la carte, il est délimité par une ligne de découpe dans la carte. Cela permet de pouvoir "casser" la carte en deux, une fois le travail fini. Ainsi on bloque l'accès au micro-contrôleur à une future personne voulait reprogrammer le STM32. Dans notre cas, cette fonctionnalité n'est pas vraiment nécessaire.

Un des nombreux intérêts d'avoir un programmateur intégré à notre carte est la simplicité du flash. Il nous suffit de brancher la carte à notre PC via un simple câble USB. Puis de lancer la commande suivante :

st-flash write notrebinaire 0x800000

Cette commande est assez simple à comprendre, il va chercher à flash le binaire généré par notre Makefile. Il y a cependant une subtilité. Pourquoi flasher notre code à partir de l'adresse 0x800000 ?

Cela est dû au bootloader de la carte. Comme tout bootloader, il est situé dans les premiers blocs mémoires. Ainsi le premier bloc mémoires disponibles se trouvent

à l'adresse 0x800000. Cette adresse est bien entendu fournie par le fabricant par le biais de la datasheet.

Flasher l'ESP32

Contrairement à ce que l'on peut penser le flash de l'ESP n'est pas aussi simple que celui du STM32. Cela est dû au fait que notre ESP ne possède pas de programmeur intégré. Nous devons donc utiliser une carte externe que l'on appelle un programmeur FTDI. Globalement cela consiste à relier une clef USB à notre PC puis de relier l'ESP à la clef en utilisant un schéma de câblage bien précis. Une

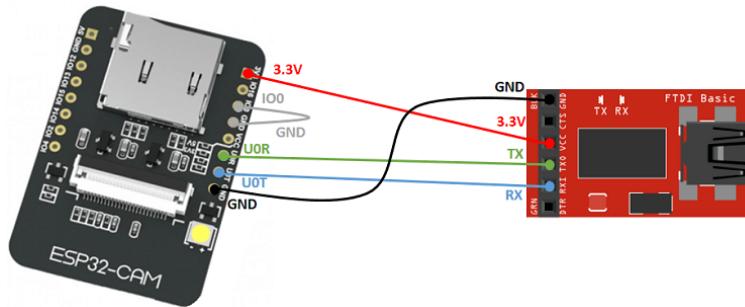


FIGURE 10 – Câblage du programmeur FTDI

de nos principales contraintes était que nous devions défaire tous le câblage pour pouvoir tester notre code. Heureusement, cela n'a pas duré, nous avons fini par souder quatre câbles pour la liaison avec le robot en laissant les pins accessibles pour le programmeur.

De manière purement technique, il est intéressant de savoir qu'un programmeur ne fait qu'envoyer du code via le protocole UART, par le biais des pins RX et TX, au microcontrôleur. Le jumper permet à la carte de savoir quelle est bien en mode re-programmation. Bien entendu, il sert aussi de source d'alimentation pour la carte via les pins 5V et GND.

Une fois, le programmeur branché à notre PC, ils nous suffisaient d'utiliser l'IDE d'Arduino pour qu'il puisse reconnaître le type de carte et flasher le code à la bonne adresse mémoire.

Debugger le robot

Il faut retenir que, par défaut le port utilisé pour le debug d'information, celui que l'on pourrait comparer à STDOOUT sur Linux, est le même que celui utilisé pour la programmation. Ainsi il suffit d'ouvrir un logiciel capable de lire le protocole UART passant par la clef ou le câble et de configurer les bauds (la fréquence de communication) pour pouvoir debugger notre robot. Il existe plusieurs possibilités comme GTKTerm ou screen pour lire un protocole UART.

9.6 Asservissement

Asservissement en vitesse

Dans le concept on peut comparer le principe d'asservissement en vitesse à un humain qui conduirait une voiture qui est à l'arrêt et qui veut aller à 100 km/h. Tout d'abord si on ne prend que le terme proportionnel, cela reviendrait à appuyer fort quand on est bas de l'objectif, ainsi une fois arriver à 100 km/h, nous allons relâcher l'accélérateur car nous avons dépassé les 100 km/h. Une fois redescendue en dessous des 100 km/h, on rappuie sur l'accélérateur. On va ainsi tendre vers une vitesse inférieure à notre objectif, ici 90 km/h. Pour régler ce problème on rajoute le terme intégral, cela revient maintenant à accélérer plus fort quand on reste longtemps éloigné de l'objectif, ainsi il y aura une oscillation pour atteindre la vitesse voulue. Finalement on rajoute le dernier terme celui de la dérivée, il revient lui à moins accélérer lorsqu'on se rapproche de l'objectif, ainsi on dépasse moins l'objectif et on redescend moins en dessous de l'objectif.

Finalement on a donc un régulateur qui va rapidement à notre objectif avec peu d'oscillation.

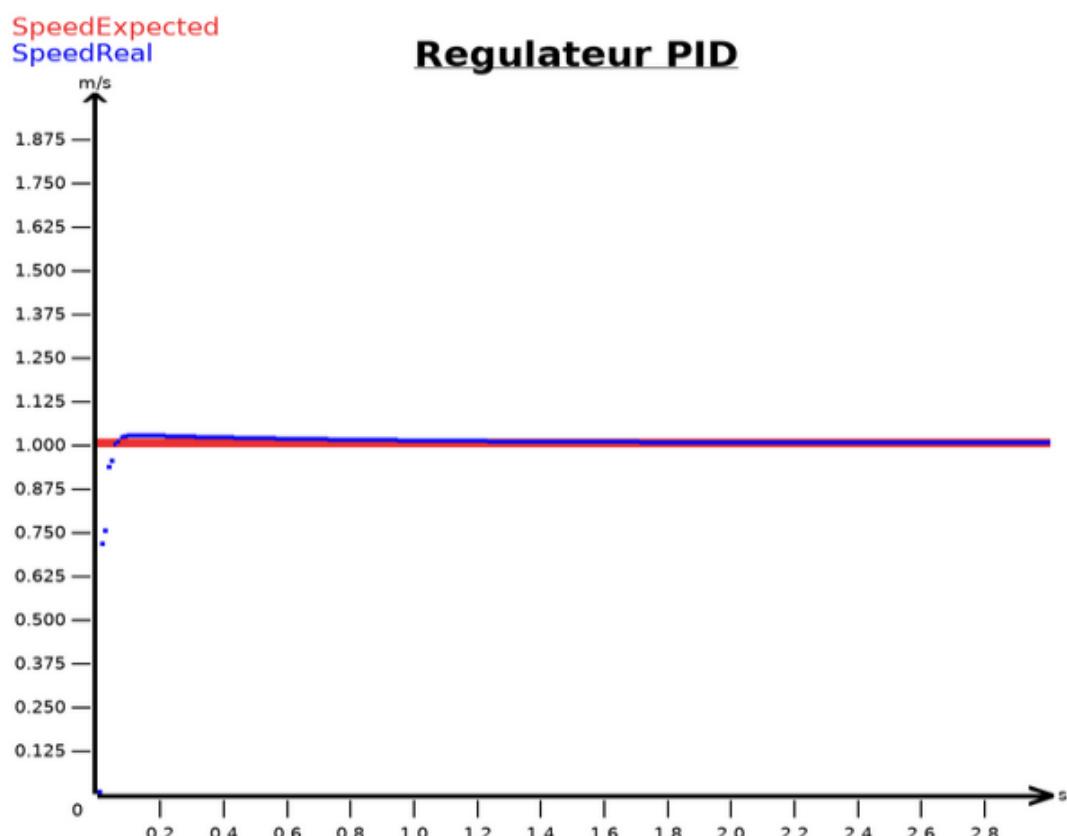


FIGURE 11 – Exemple Régulateur PID

Pour cela, nous devons utiliser une formule mathématique qui nous permet de connaître notre commande à un instant t .

$$u(t) = K_p \epsilon(t) + K_i \int_0^t \epsilon(t) dt + K_d \frac{d\epsilon(t)}{dt}$$

ou :

- K_p est le gain proportionnel
- K_i est le gain intégral
- K_d est le gain dérivé
- $\epsilon(t)$ la fonction erreur (*consigne – mesure*)

L'étape de la calibration, nous permet de trouver les K_p, K_i, K_d adaptés à nos moteurs et notre robot. Afin de dé-dramatiser cette fonction, voici l'implémentation en C de notre PID.

Listing 2 – Implémentation du PID

```

double computePid(double measure,\n
                  double order,\n
                  pidController* controller)\n{
    double error = order - measure;\n
    controller->errorTotal += error;\n
    double derivative = error - controller->errorLast;\n\n
    double command = (controller->Kp * error) + \n
                    (controller->Ki * controller->errorTotal) + \n
                    (controller->Kd * derivative);\n\n
    controller->errorLast = error;\n
    return command;
}

```

Calibration du PID

La calibration du PID est une étape longue mais nécessaire pour le bon fonctionnement de ce dernier. La formule repose sur trois variables que l'utilisateur doit calibrer, K_p, K_i, K_d .

Pour les calibrer, nous fixons K_i, K_d à 0 puis cherchons une valeur d'oscillation pour la variable K_p . Une fois, cette valeur trouvée, il suffit de diviser par deux et nous avons la valeur de K_p . Nous passons ensuite aux valeurs suivantes avec le même procédé. Le problème de cette technique est sa longueur. A chaque valeur, nous devons recompiler notre programme, l'envoyer sur notre robot puis tester le robot et recommencer ainsi en boucle.

Asservissement en position

On va donc maintenant vouloir que notre robot avance d'une certaine distance et tourne d'un certain sens. Pour ce faire il ne suffit pas simplement d'exécuter le calcul du PID, en effet il produit une secousse brusque au démarrage et une fois la distance voulue atteinte, on ne peut pas arrêter d'un coup notre robot à pleine vitesse. On doit lui faire suivre un trapèze de vitesse. C'est-à-dire qu'il doit, au début, accélérer uniformément puis rester à sa vitesse désirée et par la suite décélérer. Il suffit donc pour cela de diviser le mouvement en trois phases.

Au début la vitesse voulue de notre robot commence à la vitesse minimum pour augmenter à intervalles réguliers. À partir de là, il y a deux possibilités, soit la vitesse limite est atteinte alors la vitesse stagne, on prend en compte la distance qu'il a fallu pour accélérer pour pouvoir décélérer de la même façon, soit on atteint la moitié de la distance voulue avant la fin de l'accélération alors on décélère. Pour la décélération la diminution de la vitesse doit se faire 1.05 fois plus rapidement que l'accélération sinon la distance parcourue est atteinte avant la fin de la décélération. Cela est dû au fait que la vitesse augmente plus rapidement quand on demande une vitesse plus haute que la vitesse actuelle du robot. Ensuite après la décélération le robot se stabilise à une vitesse minimum qui lui permet de parcourir les derniers centimètres.

Ainsi si on dessine la courbe de la distance parcourue dans le temps, on voit que le départ se fait progressivement, ensuite l'évolution est stable et enfin sur l'arrivée la distance atteint la distance désirée progressivement. Pour tourner, on effectue la même démarche. Dans la pratique on fait tourner les roues dans le sens opposé et on calcule l'angle parcouru avec la distance parcouru par une roue.

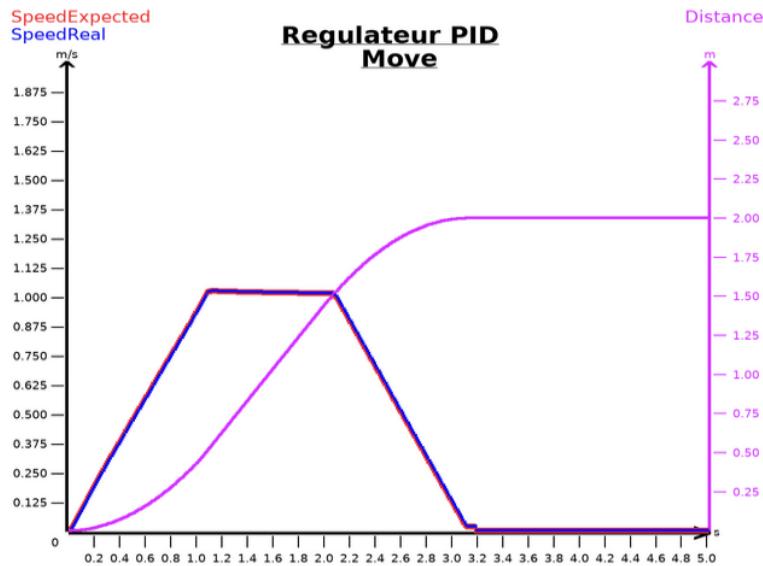


FIGURE 12 – Exemple Régulateur PID Move

Démonstration des formules

Alors tout d'abord, il faut fixer quelques valeurs arbitraires V_{max} ainsi que A . Elles correspondent à la vitesse maximale de mon robot et l'accélération, décélération de mon robot.

On note :

D *La distance voulue*

T *Le temps total*

On pose $T_a = T_r - T_c$

$$D = \int_0^T f(t) dt \quad (1)$$

$$= \int_0^{T_a} f(t) dt + \int_{T_a}^{T_c} g(t) dt + \int_{T_c}^{T_r} h(t) dt \quad (2)$$

$$= 2 \times \int_0^{T_a} f(t) dt + \int_{T_a}^{T_c} g(t) dt \quad (3)$$

(4)

Selon la formule du coefficient directeur $\frac{Y_b - Y_a}{X_b - X_a} = "Coef\ Directeur"$, dans notre cas, on prendra toujours $X_a = Y_a = 0$ ainsi on en conclut que $\frac{Y}{X} = A$
Pour cette partie, nous devons faire une disjonction de cas.

$$K = \int_0^{T_a} f(t) dt + \int_{T_c}^{T_r} h(t) dt$$

Si $K = D \Leftrightarrow (T_c - T_a) = 0$:

$$\frac{V_{max}}{T_a} = A \Leftrightarrow \frac{V_{max}}{A} = T_a$$

$$\boxed{T_a = T_r = \frac{V_{max}}{A}}$$

Si $K \leq D$:

$$\frac{V_{max}}{T_a} = A \Leftrightarrow \frac{V_{max}}{A} = T_a$$

$$D = 2 \times \frac{V_{max}}{2 \times A} \times V_{max} + \int_{T_a}^{T_c} g(t) \, dt \quad (5)$$

$$= \frac{(V_{max})^2}{A} + V_{max} \times (T_c - T_a) \quad (6)$$

$$= \frac{(V_{max})^2}{A} + V_{max} \times T_c - \frac{(V_{max})^2}{A} \quad (7)$$

$$= V_{max} \times T_c \quad (8)$$

$$\boxed{T_c = \frac{D}{V_{max}} \quad T_a = T_r = \frac{V_{max}}{A}}$$

Si $K \geq D$:

Dans cette condition, notre courbe ne ressemble pas à un trapèze mais à un triangle.

$$\frac{B \times h}{2} = D$$

h étant la vitesse maximale à atteindre

$$\frac{h}{B} = A \Leftrightarrow B \times A = h$$

$$B^2 = \frac{2 \times D}{A}$$

$$\boxed{T_a = T_r = \frac{\sqrt{\frac{2 \times D}{A}}}{2}}$$

Suite de test

Pour pouvoir tester notre algorithme, il a d'abord fallu simuler l'utilisation de nos roues et le dire à notre algorithme, qui déplace notre robot de combien nos roues avaient avancé. Pour faire simple nous avons dit que nos roues suivaient parfaitement la consigne qu'envoyait le régulateur PID et qu'aucune perturbation déréglaît notre robot. Il a fallu avoir un retour visuel également.

Tout d'abord il s'agissait d'un fichier texte et de la console où à chaque étape on notait les valeurs : enregistré/calculé de l'encodeur, l'erreur par rapport à la vitesse demandée, la distance parcourue par la roue depuis le dernier calcul, le temps qu'il a fallu et la distance totale parcourue. Toutefois ceci est utile pour comprendre l'origine des erreurs de calcul mais pas pour suivre le robot dans le temps. Il a donc fallu modéliser un graphique, à l'aide de SDL2, pour pouvoir suivre la vitesse demandée au robot, la vitesse mesurée sur nos roues et la distance parcourue. On avait donc un moyen de comprendre les erreurs et de visualiser nos valeurs et voir si notre trajectoire était satisfaisante. Pour ce faire, on dessine un point à chaque intervalle de temps et une fois tous collés, on obtient une courbe et des droites.

Maintenant qu'on observe notre mouvement on peut le régler. En effet pour calculer le PID, on utilise des coefficients devant chaque terme, on peut donc maintenant les choisir. Nous avons décidé de les choisir par tâtonnement, en réglant d'abord celui du P, ensuite celui du I et celui du D par la suite. Mais cela n'est pas simple car notre robot ne doit avoir aucune secousse, ces coefficients sont donc très bas et à chaque coefficient rajouté on doit re-régler les anciens.

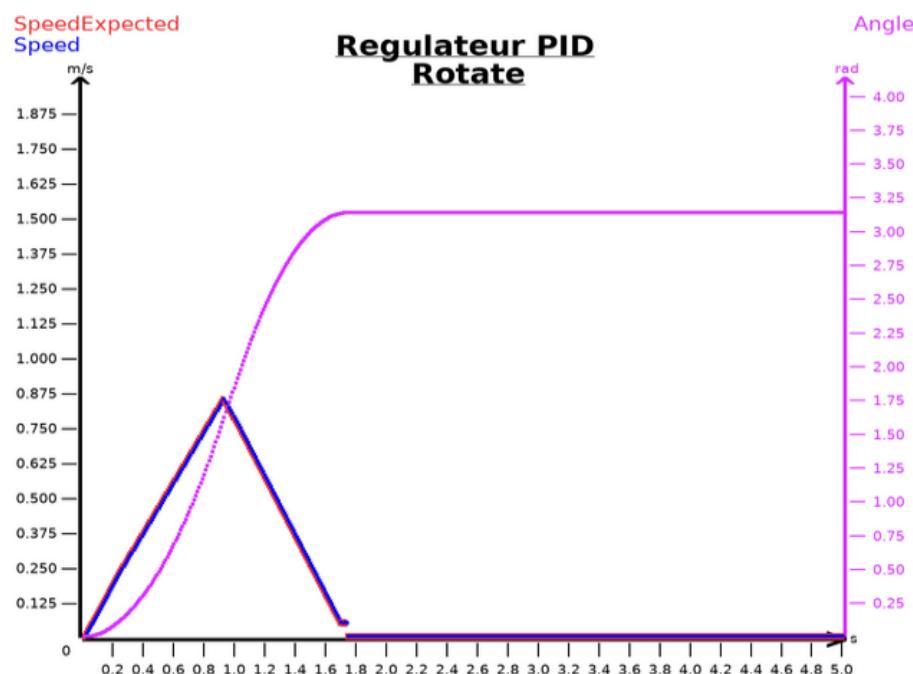


FIGURE 13 – Exemple Régulateur PID Rotate

9.7 Détection de notre position

Nous avons besoin de notre position pour pouvoir nous diriger dans le monde terrestre. Pour cela, il existe un grand nombre de capteurs capable de nous envoyer des coordonnées plus ou moins précises. Dans notre projet, nous "utilisons" des encodeurs et une centrale inertuelle. La nuance est importante car nous n'avons jamais réussi à utiliser la centrale inertuelle. En effet, nous nous sommes rendu compte après coup que la centrale ne fonctionnait pas avec du 3.3V mais avec du 1.8V. Le problème est que notre PCB est désigné pour lui envoyer la mauvaise tension.

Ainsi nous nous retrouvons avec uniquement nos encodeurs rotatifs pour nous guider. Il est donc devenu inutile de faire un filtre de Kalman pour avoir une estimation combinée de nos capteurs. Un des problèmes soulevés par ces encodeurs est le fait qu'il ne gère pas la déviation du robot. Cela fait qu'il ne peut pas nous indiquer comment notre robot dévie de l'angle par rapport à notre trajectoire initiale.

9.8 Prise Photo

Avant de détecter quoi que ce soit autour du robot, nous devons prendre une photo de notre environnement. Pour prendre nos photos nous utilisons un ESP32 équipé d'une caméra et d'une carte SD. Pour pouvoir manipuler ces composants il existe des librairies et des fonctions déjà existantes. Ainsi au départ l'idée pour prendre des photos et pouvoir la transférer à notre serveur était de : prendre la photo, la stocker sur une carte SD et d'ensuite la lire pour envoyer la photo au serveur. Or l'ESP ne permet pas d'allouer de la mémoire à l'infini ainsi nous ne pouvions pas lire le fichier enregistré car il demandait d'allouer nous-mêmes de la mémoire. Pour contrer ça nous avons simplement décidé d'enregistrer notre image sur la carte SD pour pouvoir la regarder et la comparer à celle envoyé au serveur et en même temps de directement l'envoyer au serveur sans passer par une étape de lecture de cartes SD. Un autre problème est survenu pendant la prise de la photo, ce problème est que très souvent nos images étaient en nuances de vert. Le problème venait d'une prise trop rapide de la photo avant qu'elle soit bien initialisée. Pour contrer cela, il a fallu prendre plusieurs photos et de garder la dernière.

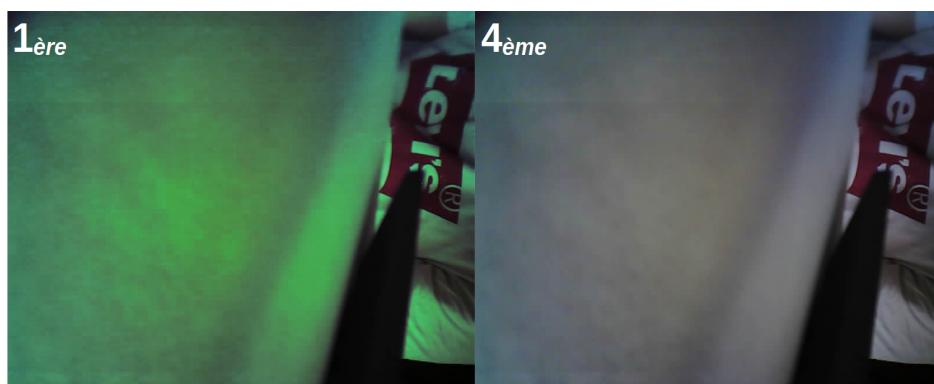


FIGURE 14 – Image de la Caméra

9.9 Détection de l'ArUco

ArUco

Avant de chercher à les détecter, nous devons savoir ce qu'est un ArUco. Pour le faire sommairement, un ArUco est comme un QRCode mais simplifié. Ils sont composés de 4, 6 ou 8 pixels de côté et possèdent pour seule information un ID. Il faut savoir qu'un ArUco est accompagné d'un dictionnaire qui permet de faire

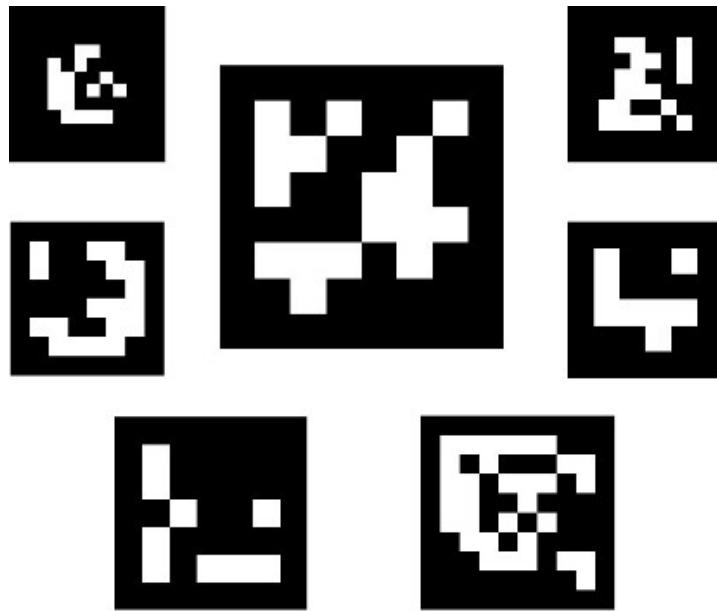


FIGURE 15 – Exemple de ArUco

le lien entre son ID et sa forme. Dans notre projet, nous utilisons le dictionnaire original du standard ArUco.

Nous utilisons les ArUcos afin de savoir où notre robot se trouve sur le terrain. L'intérêt des ArUco est qu'ils sont assez simples à détecter dans une image.

Détection

La partie détection se fait de manière assez simple sur une image. Pour cela, il suffit de faire une binarisation de notre image. Une fois cela fait, nous pouvons rechercher l'intégralité des carrés de notre image. Il se peut que nos carrés soit déformé, pour les redresser nous utilisons une simple matrice de déformation.

Il suffit d'extraire les bits de notre ArUco en le divisant en carrés puis de stocker cela dans une matrice. Cette matrice sera notre clef pour pouvoir accéder à notre dictionnaire. Ce dictionnaire, nous retournera notre ID.

Calcul de notre position relative à l'ArUco

Pour connaître notre position relative par rapport à notre ArUco, il nous faut un paramètre que l'utilisateur passe en paramètre. C'est la longueur de la diagonale

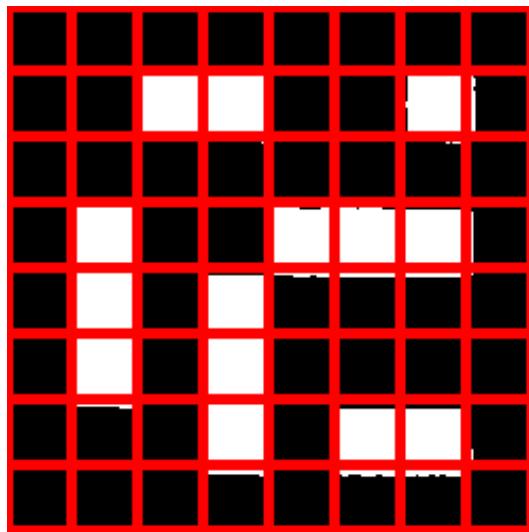


FIGURE 16 – Extraction de bits sur un ArUco

de notre carré. Il suffit juste que tous nos ArUco aient la même diagonale. Dans notre cas, nous avons choisi de prendre une diagonale de 10cm \Leftrightarrow 0.1m.

Grâce à la longueur de notre diagonale et la matrice de déformation (qui rend notre ArUco carré), nous pouvons récupérer notre position relative par rapport au ArUco. Cette même distance que nous utiliserons pour déplacer notre robot.

Implémentation

Pour des problématiques de rapidité et d'efficience, nous avons choisi de ne pas implémenter nous-mêmes la détection d'ArUco mais d'utiliser une librairie. Ainsi nous utilisons OpenCV en C++ pour détecter nos marqueurs. Cette librairie a l'avantage d'avoir en mémoire la plupart des dictionnaires connus de ArUco et de renvoyer la position relative du ArUco.

Listing 3 – Detection des marqueurs

```

Mat image;
inputVideo.retrieve(image);

vector< int > ids;
vector< vector< Point2f > > corners, rejected;
vector< Vec3d > rvecs, tvecs;

// detect markers and estimate pose
aruco::detectMarkers(image, dictionary, corners, ids, detectorParams,
    rejected);

```

9.10 Stockage du labyrinthe

Pour le stockage, nous avons rendu la chose la plus simple possible. Nous avions un cahier des charges assez précis :

- Pouvoir stocker un dictionnaire de ArUco
- Relier un ArUco à une distance Linéaire et un angle Théta

Ainsi nous avons implémenté une structure Node qui contient l'index du ArUco père ainsi que la distance Linéaire et Angulaire. Voici l'implémentation en C du labyrinthe.

Listing 4 – Stockage du labyrinthe

```
struct node {
    uint8_t idfather;
    uint8_t distanceLinear;
    uint8_t distanceAngular;
}

struct node Map[NUMBER_OF_ARUCO] = {
{3, 2, EAST},
// ... //
{36, 1, NORTH},
}
```

Représentation de la MAP

Nous avons une première représentation du labyrinthe utilisé, nous pouvons dire que c'est un labyrinthe dit parfait. Il ne possède pas d'îlot.

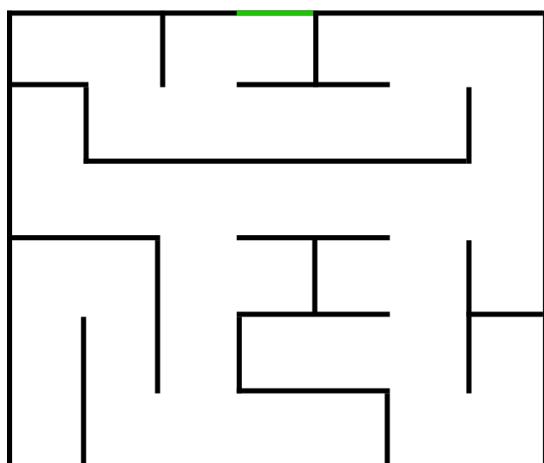


FIGURE 17 – Map Labyrinthe

Et ici une représentation du labyrinthe avec les différents ArUco

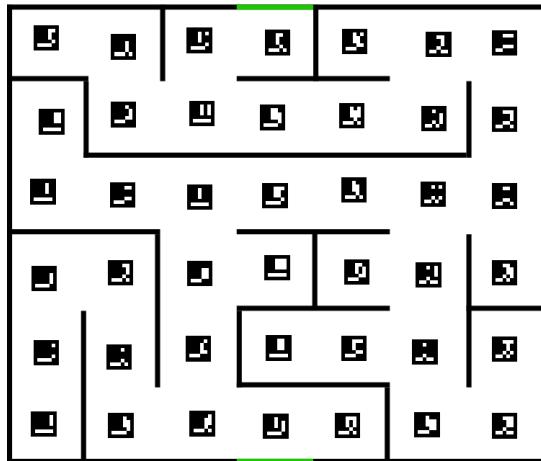


FIGURE 18 – Map Labyrinthe avec ArUco

Parcours de la Map

Pour parcourir le labyrinthe, nous avons plusieurs fonctions

Listing 5 – Parcour du labyrinthe

```

const uint8_t father = 0;

void init_map(uint8_t id)
{
    father = id;
}

struct intResult get_father()
{
    uint8_t old_father = father;
    father = map[old_father].idfather;
    struct intResult result = {map[old_father].nvbrmvt,
        map[old_father].orientation};
    return result;
}

```

La première fonction va nous permettre d'initialiser le labyrinthe avec l'id de l'arUco reconnue par le robot. On va donc le faire une fois, juste après la reconnaissance.

La seconde fonction va permettre d'avoir le nombre de mouvement à faire et l'orientation vers le prochain ArUco et l'id est mise à jour pour rappeler à la prochaine itération getfather qui utilisera l'id mis à jour. Pour chaque itération on va attendre que le mouvement se fasse après l'avoir initialisé.

9.11 Interface Utilisateur

Afin de pouvoir débugger notre programme, nous avons dû utiliser un software que l'on fait tourner dans un docker qui écoute sur un port définit nos requêtes. Il nous suffit de faire des requêtes UDP sur ce port avec une syntaxe particulière pour voir apparaître nos valeurs. Cette interface, nous a été primordiale pour calibrer notre PID.

Notre utilisation

Nous avons décidé d'utiliser une librairie externe pour pouvoir visualiser nos données temporelles. Le premier intérêt de cette librairie et sa simplicité, le fait que son créateur fournisse un docker prêt à l'usage, nous a grandement facilité la vie. Cette librairie, nous permettait de pouvoir visualiser via des graphiques des données envoyés par notre robot. Ce software a été nécessaire dans la calibration de notre PID. En effet, il paraît compliqué de détecter l'oscillation d'un PID par le biais d'un simple print sur la sortie standard.

Ce software, nous a permis de faire ce que l'on appelle communément dans le jargon de la robotique de la télémétrie. Nous pouvions voir les actions effectuées et essayer de comprendre ses choix en fonction des valeurs qu'ils possédaient. Le stockage des données d'entraînement est notamment utilisé dans le milieu professionnel pour la simulation numérique et ainsi avoir une meilleure compréhension des algorithmes mis en place.



FIGURE 19 – Teleplot

La syntaxe

Nous avons utilisé ce software afin d'afficher des graphiques mais il faut savoir que nous n'avons utilisé que 1% de ce software. Il offre, par exemple, la possibilité d'afficher des formes géométriques simples dans un espace en 3D (utile pour le debug d'un lidar).

Afin de pouvoir envoyer nos valeurs au software, nous devions respecter une certaine syntaxe. Voici quelques exemples de syntaxe autorisée par Teleplot :

- myValue :1234
- myValue :1234|
- myValue :12.34e+2
- myValue :1627551892437 :1234
- myValue :hello|t
- myValue :1234§km²

Chacune de ces lignes permet de déclarer un point sur une courbe s'appelant myValue. On peut voir que l'on peut préciser des puissances ou bien des unités en plus d'une valeur.

Nous pouvons aussi utiliser Teleplot pour afficher un point sur une carte pour cela, il existe aussi un format particulier.

- trajectory :12.3 :45.67|xy
- trajectoryTimestamped :1 :1 :1627551892437
;2 :2 :1627551892448 ;3 :3 :1627551892459|xy
- trajectory :1 :1 ;2 :2 ;3 :3 ;4 :4|xy
- myValue :1627551892444 :1 ;1627551892555 :2 ;1627551892666 :3
- myValue :1627551892444 :1 ;1627551892555 :2 ;1627551892666 :3rad
- state :1627551892444 :state_a ;1627551892555 :state_b|t

Dans ce cas-ci, nous envoyons un ou plusieurs points à la fois. Un timestamp peut aussi être envoyé de pouvoir débug plus facilement notre robot.

Open-Source

Bien entendu, il paraît évident que ce software est open-source et libre de droits. L'utilisation de software open-source prend une place particulièrement importante dans notre cheminement de pensée. En effet, il nous paraît important de promouvoir la création de librairie pareil ainsi que son utilisation.

<https://github.com/nesnes/teleplot>

10 Communication

La détection du ArUco se faisant sur la partie serveur, il est nécessaire que notre robot puisse communiquer avec l'extérieur. Pour cela, nous avons relié à notre robot, une autre carte, un esp32. Si l'on devait simplifier cette carte, ce serait une antenne WI-FI. Elle nous permet de communiquer sur un réseau WI-FI via des requêtes HTTP, TCP ou autre.

10.1 Protocole

Criteres

Afin de pouvoir communiquer, nous avons dû créer un protocole de communication. Elle devait répondre à certains critères :

- Savoir quand ma trame commence
- Connaître le type de ma donnée : Une Variable ou une commande à exécuter
- Connaître la longueur de la trame
- Être intègre

Solution

Pour respecter ces critères, nous avons décidé de créer un protocole pour nos requêtes. Il se compose d'un octet qui délimite le début de notre requête afin de ne pas prendre en compte le flux non désiré. On inscrit ensuite le type de notre requête, qui permet de connaître la taille de la donnée qui arrive ensuite. Le type est donc suivi d'un espace dédié aux données, qui dépendent du type de données envoyé, 4 octets pour un int, 8 pour un float, 16 pour une position... Et pour finir on effectue un checkSum et on l'insère à la fin de notre requête. Un checkSum est un calcul qui se fait sur notre donnée et qui donne un résultat différent pour chaque donnée différente. Ceci est semblable au principe de hachage qui va renvoyer une valeur unique pour une donnée unique. Ce calcul permet ainsi d'identifier qu'aucun bit n'a été ajouté ou modifié, donc on est sûr de l'intégrité de notre donnée.

Listing 6 – enumeration de nos type de données

```
enum TypeFrame
{
    NONE = -1,
    ACK = 1,
    RESPONSE_POSITION = 2,
    DEBUG_POSITION = 3,
    DEBUG_INT = 4,
    DEBUG_FLOAT = 5,
    ASK_POSITION = 6,
};
```

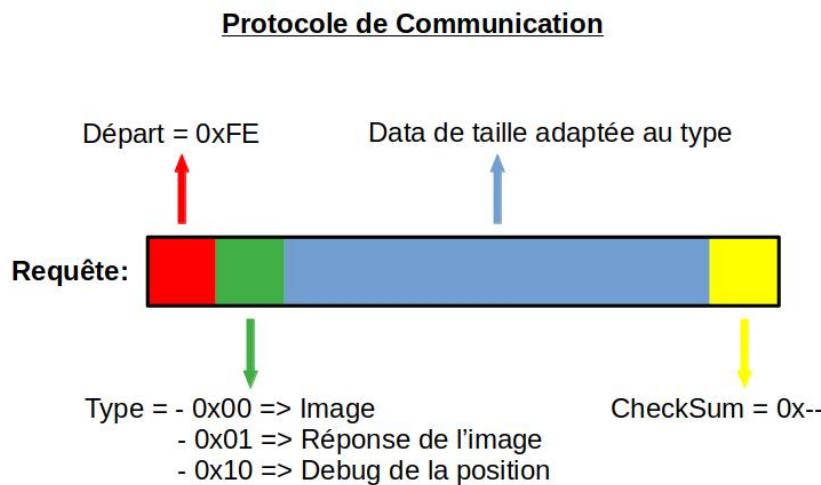


FIGURE 20 – Protocole de Communication

10.2 Direct Memory Access

Le direct Memory Access (DMA) permet de régler un de nos principaux problèmes. Voici le souci, nous avons un grand nombre de requêtes qui arrive sur un port de notre carte et nous ne voulons rater aucune information. Cependant nous ne pouvons pas rester à écouter à chaque instant notre port, nous devons aussi faire avancer notre robot. Pour cela, il existe cette technologie, le direct Memory Access.

Nous ne chercherons pas à expliquer le fonctionnement de cette technologie, qui est légèrement trop complexe pour notre niveau en informatique embarquée. La seule chose qu'il faut comprendre, c'est que la carte stop de manière régulière son processus afin de lire le port et stocke le tout dans une pile.

10.3 ESP32

L'esp32 est une petite carte que l'on utilise majoritairement pour de l'IOT. En effet, elle possède quelques pins et intègre une antenne Wi-Fi ce qui la rend très utile pour la création d'objets connectés. Dans notre cas, nous l'utilisons comme antenne WI-FI pour la communication entre le robot et notre serveur. Une autre utilisation de notre ESP est sa caméra embarquée. Elle nous permet de capter une vidéo en 1080p et 15FPS, ce qui est largement suffisant pour notre utilisation. Afin de pouvoir envoyer les requêtes du STM et n'en rater aucune, nous utilisons le DMA cela nous permet de pallier le manque de fiabilité du bus SPI.

Pour implémenter la DMA, nous avons dû redoubler d'efforts. En effet, Arduino le framework que nous pensions utiliser ne supporte pas la technologie DMA. Nous avons donc dû utiliser un autre compilateur que celui d'Arduino, ESP-IDF. Pour pouvoir l'utiliser, nous avons du rebuild le compilateur car il n'existe pas de version pré-build pour notre système d'exploitation (btw I use arch).

10.4 STM32

Cette carte est le cerveau de robot, elle n'a pas d'antenne WI-FI mais elle possède un grand nombre de pins pour pouvoir communiquer avec d'autres cartes. On peut l'utiliser pour communiquer avec un bus SPI, UART, I2C, JACK et bien d'autres. Nous pouvons envoyer des requêtes via le bus SPI grâce aux pins RX et TX.

Nous avons dû configurer le DMA afin de pouvoir récupérer toutes les requêtes envoyées par notre ESP. Ce fut assez simple, il existe une bonne documentation sur Internet et le sujet.

10.5 Serveur

Le serveur a de nombreuses missions. Il permet d'effectuer tous nos calculs sur les images, notamment pour identifier les ArUco et notre position relative à ce dernier (entre le robot et l'ArUco). Il sert également à rediriger nos requêtes vers l'interface utilisateur.

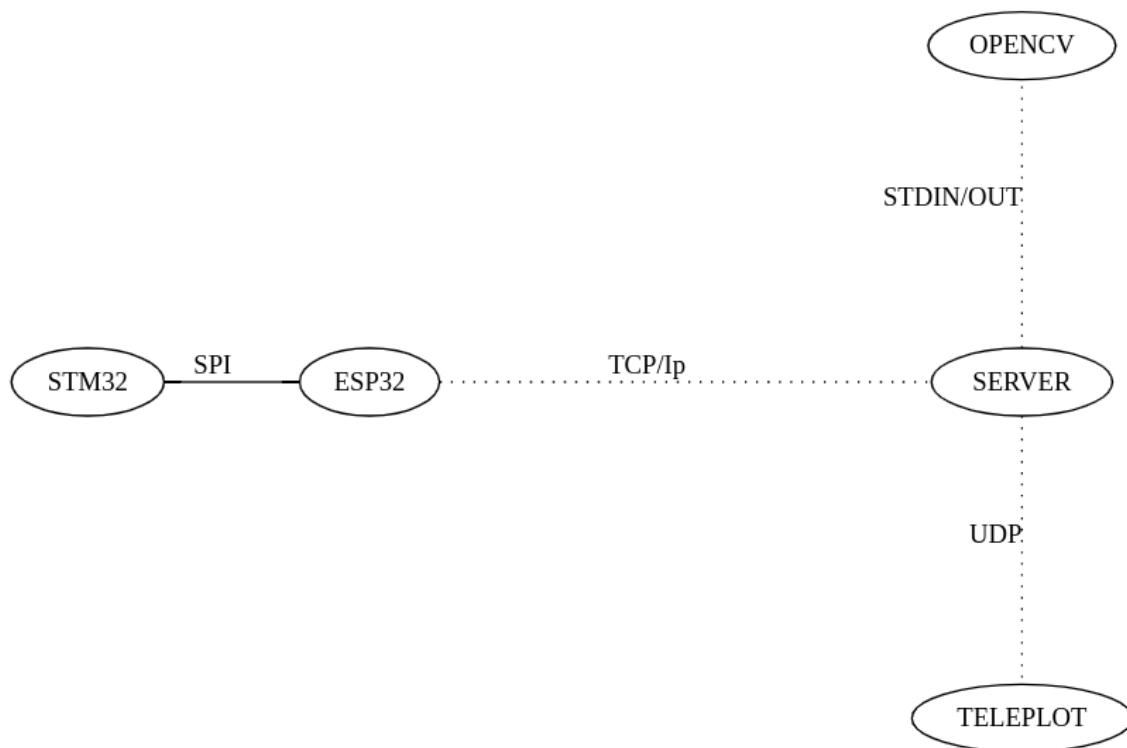


FIGURE 21 – Communication Inter-Module

10.6 Mise en place

Pour ce faire nous avons d'abord créé la liaison serveur/ESP32. En effet côté serveur il s'agit d'une mise en place classique d'un serveur sur un port TCP. Le plus dur pour lui c'est qu'il doit être capable de savoir traiter les données reçues afin de soi les envoyées à la partie opencv, soit à teleplot. Par exemple quand nous recevons des entiers, ils sont sur 32 bits or nous sommes en tableau de 8 bits, il a ainsi fallu faire des décalages à droite et gauche de chacun des côtés de la chaîne et de procéder à une opération de masquage pour ne pas avoir des données non désirées pendant le décalage.

Pour envoyer des données de l'ESP32 le principe est vraiment simple, il consiste à faire comme si on écrivait dans un fichier à l'aide de fonctions déjà existant qui utilise un file descriptor. Le problème qui est arrivé à être que ces fonctions pouvaient dans certains cas s'auto appeler plusieurs fois si toutes les données ne pouvait pas s'envoyer en une fois. La raison à cela c'est que même si l'ESP a beau être puissant, il ne peut pas procéder à des envois volumineux. Ceci a particulièrement posé problèmes pour les photos, en effet nous sommes dans un cas où les fichiers à envoyer sont vraiment grands. Nous avons donc dû découper notre photo en plusieurs requêtes, en indiquant leur nombre lors de la première requête.

Pour la liaison STM32/ESP32 cela a posé beaucoup plus de problèmes. Pour effectuer cette liaison nous communiquons via un protocole UART. Il est très bien intégré aux deux systèmes avec des fonctions dédiées qui existent pour communiquer entre eux. Mais en utilisant ces fonctions aucune donnée n'arrivait jusqu'à l'ESP32. Nous avons testé l'ESP32 grâce à l'outil GTKterm et découvert que l'ESP32 marchait très bien quand nous écrivions nous-mêmes sur le port UART. Pour tester le STM32 au lieu d'écrire seulement sur le port UART2 (notre ESP32) nous avons décidé d'écrire également sur le port UART3 qui est l'emplacement micro usb où nous pouvons lire les données grâce à GTKTerm toujours.

Par conséquent nous avons vu que les données partaient bien du port UART3, et les deux ports étant configuré pareil, il n'y a pas de raison que l'UART2 agisse différemment. Nous avons donc un problème les données partent bien de l'STM32, si elle était envoyée à l'ESP elle serait bien lue mais elles ne le sont pas. Même après avoir découvert des erreurs de branchement et les avoirs corrigés le problème était toujours présent. Ainsi pour le moment les problèmes ont été résolus mais un défaut de l'électronique et des composants n'est pas à mettre de côté.

11 Site Web

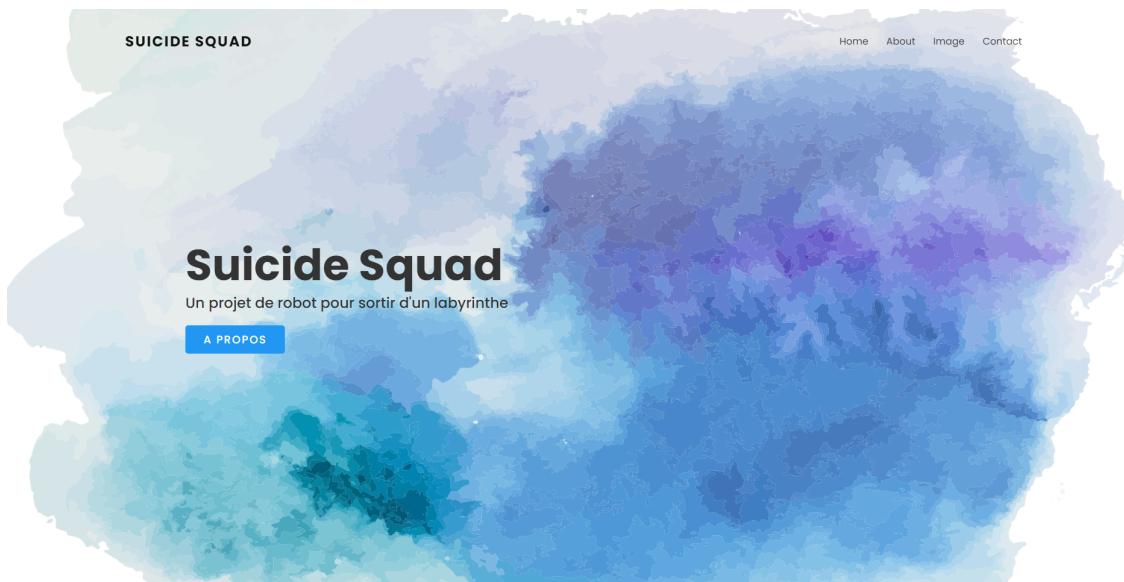


FIGURE 22 – Accueil du Site Web

Sur le site, on peut retrouver une description du projet pour comprendre qu'elle a été notre but et le chemin que nous avons parcouru pour en arriver à aujourd'hui. Nous pouvons retrouver plusieurs images du robot en 3d et dans la réalité. ainsi que des plans. Pour finir on peut télécharger les fichiers pdf de chaque rapport de soutenance et il y a le lien du github et du discord.

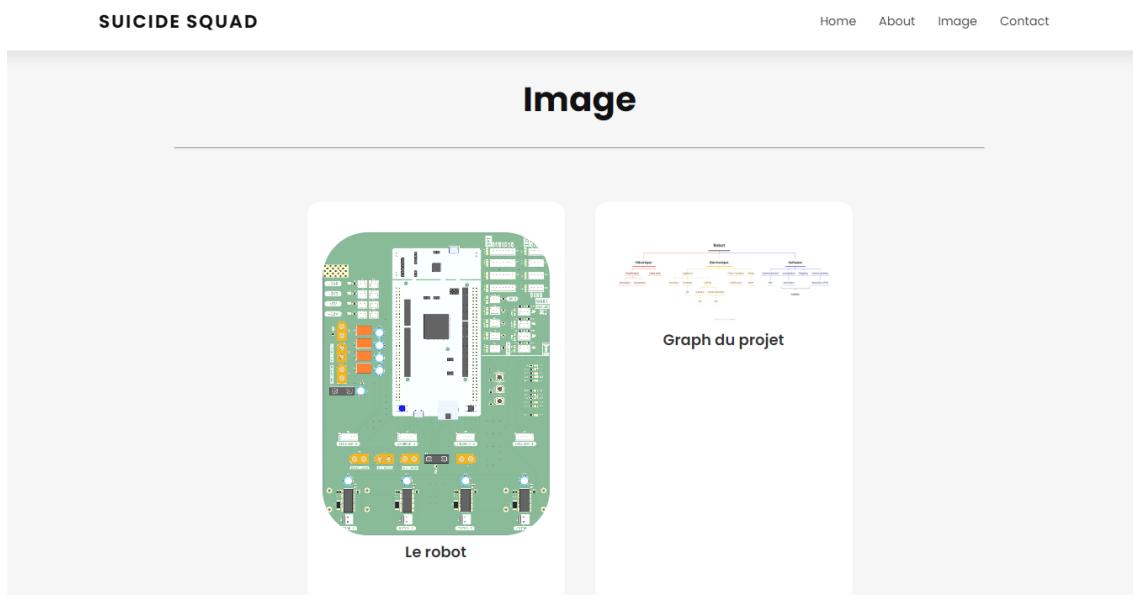


FIGURE 23 – Image du site

12 Remerciements

Nous remercions l'association Atelier Lyon qui nous a prêté son matériel et ses locaux. Cela nous a permis d'avoir un lieu pour le stockage des composants, du robot mais aussi pour pouvoir tester en condition réelle notre projet.

L'Atelier Lyon est aussi notre mécène, elle a en effet financer la plus grande partie du robot. Sans ce financement, il y a peu de chances que ce projet ait vues le jour. Il faut reconnaître que son prix (env. 300 €) a de quoi décourager les plus téméraires.

Nous remercions aussi Florian Reimat qui nous a fourni le circuit imprimé du robot, sans quoi il aurait été beaucoup plus compliqué de faire ce projet. De plus, il nous a été d'un grand conseil dont tous les aspects pratiquent de ce projet, mécanique et électronique.



En tant que chef de projet, je souhaite personnellement remercier tous les membres de mon groupe qui ont contribué activement à l'avancement de ce projet. Je les remercie de m'avoir soutenu dans mes choix, aussi bien techniques que pratiques.

13 Conclusion

La réussite de ce projet

Ce projet a été une grande nouvelle pour chacun de nous. Il faut rappeler que personne, dans notre groupe, n'avait de réelles expériences dans la programmation embarquée. Nous avons dû nous adapter à la situation et apprendre sur le tas. La majeure partie de nos connaissances a été apprise en autodidacte. Néanmoins malgré tout les obstacles dans la création de ce projet, nous avons en grande partie réussi notre défi. Celui d'apprendre les bases de la robotique, c'est-à-dire la mécanique, l'électronique puis de l'informatique associée.

Des problèmes non résolues

Bien que la majorité de notre projet soit réussie, cela ne veut pas dire que le monde est parfait. Il reste un certain nombre de points à régler. Je parle au présent car à l'opposé de la plupart des projets S4, nous allons continuer à développer ce robot. Nous pouvons parler du parallélisme de nos roues, la simulation des données récupérer ou bien une meilleure intégration de l'esp. De plus, nous avons de nombreuses améliorations que nous pouvons apporter à ce projet, l'ajout de roues holonomes pour un déplacement multi directionnel.

Des connaissances acquises

Nous allons tout d'abord faire un résumé des domaines que nous avons touchés pour ce projet. Tout d'abord le côté électronique avec la création du robot, l'utilisation des moteurs, des capteurs. Ensuite le côté Mécanique, où a conçu différentes pièces pour notre robot à l'aide d'imprimante 3d. Au niveau du software, On a plusieurs choses nécessaires pour le robot comme le PWM, le régulateur PID ou encore l'asservissement. Ensuite au niveau de la résolution du problème de base c'est-à-dire sortir d'un labyrinthe, on a dû détecter les ArUco au sol pour savoir où on est et ainsi sortir du labyrinthe donc on a dû apprendre à utiliser open cv. Pour finir, il faut bien connecter tous les appareils donc on a dû apprendre à utiliser des requêtes UDP, http ou autre. Donc nous avons acquis des compétences en électronique, mécanique et dans la communication client serveur.

Table des figures

1	Organisation du Projet	7
2	Alimentation générale	11
3	Alimentation des moteurs et des encodeurs	12
4	Connectiques des moteurs	12
5	Connectiques et alimentation des capteurs	13
6	Boutons et leds de Débug	15
7	Modélisation 3D du robot	17
8	Voltage Moyen de la période	21
9	Courbe moyenne d'un PWM	21
10	Câblage du programmeur FTDI	25
11	Exemple Régulateur PID	26
12	Exemple Régulateur PID Move	29
13	Exemple Régulateur PID Rotate	31
14	Image de la Caméra	32
15	Exemple de ArUco	33
16	Extraction de bits sur un ArUco	34
17	Map Labyrinthe	35
18	Map Labyrinthe avec ArUco	36
19	Teleplot	37
20	Protocole de Communication	40
21	Communication Inter-Module	41
22	Accueil du Site Web	43
23	Image du site	43

Listings

1	Fonction de Callback	22
2	Implémentation du PID	27
3	Detection des marqueurs	34
4	Stockage du labyrinthe	35
5	Parcours du labyrinthe	36
6	enumeration de nos type de données	39

Source

- > (en) en.wikipedia.org/wiki/PID_controller#Mathematical_form
- > (fr) vgies.com/projet-robot
- > (en) github.com/Floride1
- > (en) st.com/resource/en/datasheet/stm32f446re.pdf
- > (fr) pm-robotix.eu/2022/02/02/asservissement-et-pilotage-de-robot-autonome
- > (en) youtube.com/@MitchDavis2
- > (fr) youtube.com/@EricPeronnin
- > (en) github.com/atelierlyon/Robot_Main_Board/blob/master/Export/schematics.pdf
- > (en) github.com/nesnes/teleplot
- > (fr) randomnerdtutorials.com
- > (en) components101.com/modules/esp32-cam-camera-module
- > (fr) www.debug-pro.com/epita/prog/s4/lecture/network_programming.pdf
- > (fr) poivron-robotique.fr/+-Robot-holonomic-24-+.html
- > (fr) github.com/VRAC-team/la-maxi-liste-ressources-eurobot
- > (fr) youtube.com/c/barbatroniclive
- > (en) media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf
- > (en) st.com/resource/en/datasheet/lsm6dsm.pdf
- > (en) mouser.se/datasheet/2/670/amt10_v1775837.pdf