Projet S8

EIRBOT – Fabrication d'un Petit Actionneur Mobile Indépendant (PAMI)

Groupe :

Estebane GOUHEY

Guillaume PROUTEAU

Table des matières

I- Contexte du projet

Ce projet s'inscrit dans une démarche de participation de l'association EIRBOT à la coupe de France de robotique. Chaque année depuis 1998, l'association EIRBOT participe à la coupe de France de robotique avec une équipe par année, souvent pour les 1ères années et les 2èmes années. Cette année est apparu une nouveauté dans le règlement qui impose la création des Petits Actionneurs Mobiles Indépendants (PAMI), en plus du robot principal. C'est donc dans ce contexte que ce projet a été inspiré et réalisé.

Du point de vue personnel, nous sommes tous deux membres d'EIRBOT : Guillaume Prouteau était anciennement secrétaire, et Estebane Gouhey est actuellement Trésorier. Il s'agit de la 3ème participation à la coupe de France pour Guillaume et de la 2ème pour Estebane.

De plus, nous sommes tous deux passionnés par le domaine de la robotique et de la mécatronique en plus de nous intéresser à l'informatique basique qui se cache derrière ces domaines.

II- Taches relatives au projet et répartition du travail

a) Cahier des charges

Ce projet repose sur un cahier des charges qui a été imposé par le règlement et inspiré par notre expérience. Dans ce contexte, nous avons respectés les points suivants :

- Mise en place d'un asservissement précis de deux moteurs DC
    - Asservissement en position et/ou en vitesse calculé par rapport à l'erreur de commande.
    - Les caractéristiques de l'asservissement devront permettre au PAMI de rejoindre les différentes zones du plateau de la compétition en 10 secondes maximum.
    - Le Matériel/Les Composants doivent loger dans une limite de volume
    - Choix libre pour le contrôleur et les composants annexes
- Mise en place d'un algorithme de recherche de chemin
    - Algorithme qui devra gérer la stratégie de déplacement du PAMI en fonction des obstacles
- Mise en place d'un protocole de communication pour envoyer et recevoir des informations entre les PAMI.
- Mise en place d'une mécanique qui respecte les dimensions imposées
    - Taille de zone de départ donnée à 150x450 mm
    - Limite de hauteur à 150 mm
    - Le PAMI doit être plus grand qu'un cube de 60 mm de côté
    - Limite de déploiement d'une augmentation de 100 mm par rapport à son périmètre initial
    - Limite d'évolution en altitude de 350 mm
    - Doit comporter une zone de 30x30 mm pour accueillir l'autocollant avec le numéro de stand

    - Masse maximale fixée à 1,5 kg
- Mise en place de l'électronique de commande et d'alimentation
    - L'électronique de commande et d'alimentation devra s'intégrer dans la mécanique du PAMI dans le respect des dimensions maximales imposées
    - L'électronique de commande et d'alimentation doit respecter les conditions imposées par le règlement de la coupe de France :
        - Les tensions embarquées hors des dispositifs commerciaux ne doivent pas dépasser 48 V en courant continue et 48 V crête à crête en alternatif.
    - Les batteries à base de Lithium comprenant un BMS (Battery Management System) intégré par le fabricant avec une enveloppe solide n'ont pas d'obligation d'être protégée par un sac anti-feu.
    - Le robot devra être équipé d'un bouton d'arrêt d'urgence et d'une tirette de démarrage.
- Date fixe de fin de projet
    - Déroulement de la coupe de France du 8 au 11 Mai
    - Homologation des robots le 8 mai

b) Répartition du travail

Au cours de ce projet, la répartition du travail a évolué en suivant les différents besoins que le projet a imposé. Dans un premier temps, les tâches ont été réparties dès Décembre comme suit :

- Estebane :
    - Recherches pour la mise en place d'un algorithme de recherche de chemin
    - Modélisation mécanique avec un outils de CAO 3d du PAMI (Onshape)
- Guillaume :
    - Recherche des composants pour la commande et le contrôle des moteurs
    - Recherche des composants pour les capteurs du PAMI
    - Recherches pour la mise en place d'un code d'asservissement pour les moteurs DC

Aux vues de l'avancée du projet au mois de février, nous avons redéfinit la répartition des tâches pour s'adapter au mieux à l'emploi du temps imposé par la date butoir de la Coupe de France de Robotique. La répartition du travail a été redéfinie comme ceci :

- Estebane :
    - Mise en place et test d'un algorithme de recherche de chemin
    - Recherches et mise en place du code d'asservissement des moteurs DC sur le microcontrôleur choisi avec les composants installés
    - Finalisation de la modélisation mécanique du PAMI
- Guillaume :
    - Mise en place des composants pour la commande, l'asservissement et les capteurs du PAMI
    - Aide sur la partie réalisation de la partie mécanique
    - Mise en place de l'électronique d'interconnexion et d'alimentation avec l'implémentation d'un PCB dédié

III - Travail effectué

a) Choix des composants de contrôle moteur

Limites de coût et de dimensions :

Au cours de notre projet, il a été impératif de définir rapidement l'électronique que nous allions utiliser, dans l'objectif de démarrer rapidement sur un prototype valide et fonctionnel. Dans le cadre du projet, le coût de ces éléments a été limité par le budget de l'association Eirbot. De plus, le règlement du concours qui est la Coupe de France de Robotique impose un volume et une hauteur maximale du PAMI qui inclue aussi son électronique. Du point de vue de notre stratégie, nous nous étions fixés de pouvoir loger au minimum 3 PAMI dans la zone de départ (150x450 mm).

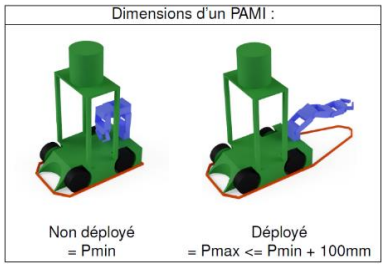


Figure 1 : Dimensions du PAMI imposées par le règlement

Alimentation :

Dans notre recherche, nous avons décidé de baser notre alimentation sur des batteries de lunettes de drone FPV, délivrant une tension de 7.4V, rechargeables en USB et comprenant un système de sécurité pour la recharge et la décharge (BMS). Ce choix a été motivé par un don de ces mêmes batteries en une quantité suffisante pour subvenir à nos besoins. De plus, la taille de ces batteries a encouragé notre choix vis-à-vis des contraintes de coût et de dimensions.

Figure 2 : Batterie de lunettes de drone fpv 7.4V

A la suite de la création du 1er prototype de PAMI et des tests qui ont été effectués, il s'est imposé que la sous-alimentation du pont en H L298N avec la tension batterie de 7.4V au lieu des 12V minimum imposé, pose beaucoup de problèmes au niveau de la stabilité de la commande. Il est donc apparu qu'il était nécessaire de générer du 12V pour l'alimentation de ce pont en H. Pour cela, nous avons utilisé un convertisseur buck-boost step-up XL6019 acheté dans le commerce.

Figure 3 : Régulateur DC-DC step-up XL6019

Moteurs :

Toujours dans la recherche d'économie de place, nous nous sommes tournés vers des mini-moteur DC 6V 500 RPM 0,18 kg/cm avec étage de réduction. Ce choix a été motivé par leur prix faible par rapport à la quantité (80€ pour 10 moteurs et frais de livraison => 6,5€ le moteur), ainsi que par rapport à leur vitesse. De plus, ces moteurs sont nativement équipés avec deux encodeurs effet hall déphasés de 90°. Cet ensemble représente une solution motrice de faible volume et intégrant tout le matériel nécessaire pour un asservissement.

Figure 4 : Moteur DC 6V 500 RPM 0,18 kg/cm

Contrôle moteur :

La solution la plus utilisée pour le contrôle d'un moteur DC réside dans l'utilisation d'un pont en H. Dans un premier temps nous avons réutilisé une carte déjà existante et réalisé par des anciens membres de l'association. Cette carte est composée de 2 ponts en H LMD18201 et possède une interface à 2 pins pour chaque moteur : la pin DIR qui permet de choisir le sens de rotation du moteur en pilotant les transistors par la commande de leur grille ; et la pin PWM qui permet d'injecter un signal PWM en tension logique pour moduler la vitesse de rotation du moteur.

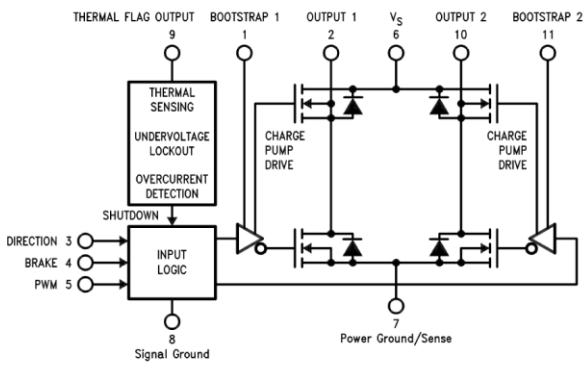


Figure 1. Functional Block Diagram of LMD18201

Figure 5 : Schéma du pont en H LMD18201

Malheureusement, dans le cadre de plusieurs tests, et à la suite d'un mauvais branchement, un des ponts en H de la carte à brulé, ce qui nous a obligé à changer de matériel.

Dans un second temps et toujours dans le cadre de nos restrictions, nous avons choisi un modèle généraliste et peu couteux : le L298N. Ce modèle s'achète sur une carte déjà assemblée et qui met à disposition un double pont en H avec une connectique pour chaque moteur, un bornier d'alimentation en 12V, une interface pour connecter le microcontrôleur ainsi qu'une sortie 5V. La commande moteur se fait par 3 pins : la 1ère (ENA) permet d'envoyer le signal logique en PWM qui régulera la vitesse du moteur ; la 2ème (IN1) et la 3ème (IN2) contrôlent les grilles de 2 transistors opposés, ce qui permet de définir le sens de rotation du moteur.

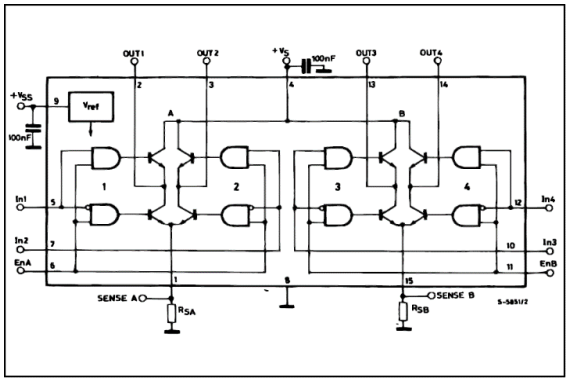


Figure 6 : Schéma du pont en H L298N

Microcontrôleur :

Pour gérer la commande moteur ainsi que tous les autres actionneurs et capteurs que le PAMI embraquera, il est impératif de choisir un microcontrôleur comme centre névralgique. Du point de vue de notre expérience commune relative à nos participations aux éditions précédentes de la coupe de France de Robotique, nous avons choisis d'utiliser un microcontrôleur ESP32. L'argument qui a favorisé ce choix par rapport à celui d'un Arduino repose sur ses caractéristiques techniques qui l'avantage dans sa vitesse et son efficacité de calcul.

ESP-Wroom32 :

- Processeur : Xtensa double cœur 32-bit LX6
- ROM : 440 KBytes
- SRAM (on-chip): 520 KBytes
- Bande passante : 72 MHz
- Vitesse de lecture de données : 150 Mbits/s
- Fréquence d'horloge max : 2,484 GHz
- Nombre de pin GPIO : 32
- Nombre de channels CAN : 16
- Modules SPI : 4
- Modules UART : 3
- Wifi : 802.11 b/g/n/d/e/i/k/r (150 Mb/s)
- Bluetooth : 4.2 BR/EDR + BLE 5.0 et 5.1

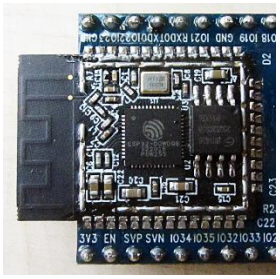


Figure 7 : Espressif ESP WROOM 32

D'autre part, dans l'optique de pouvoir communiquer avec les autres PAMI en communication sans-fil, il était obligatoire que le microcontrôleur intègre au minimum le Bluetooth. Finalement, son prix extrêmement faible (~5€ par unité) nous a convaincu que son utilisation était la seule et unique solution viable dans notre cas de figure.

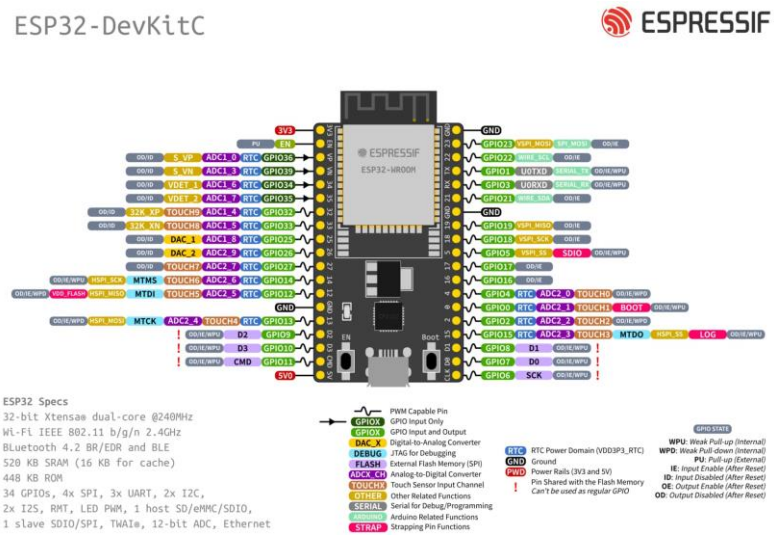


Figure 8 : Pinout d'une carte ESP32 Devkit

Assemblage du 1er prototype :

A la suite des différentes recherches pour les composants du PAMI, la construction d'un prototype permettant d'en tester l'ensemble est apparu comme obligatoire pour commencer les tests sur le système complet. C'est donc en janvier que le 1er prototype a été assemblé.

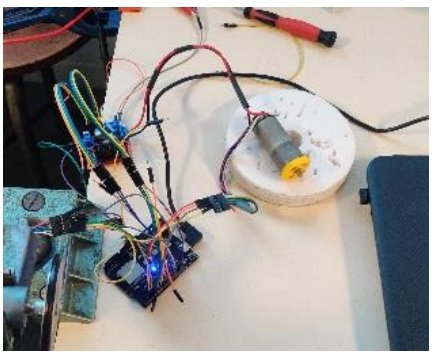


Figure 9 : Pré-version fonctionnelle du prototype

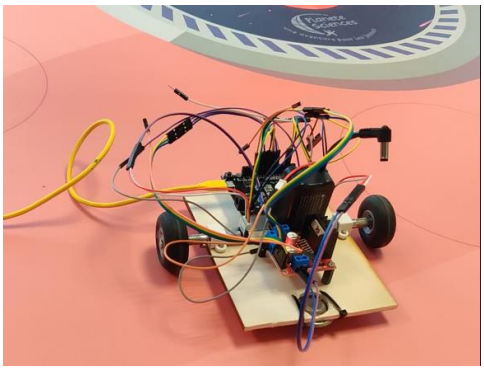


Figure 10 : 1er prototype roulant de PAMI

b) Asservissement des moteurs

Pour contrôler et asservir les moteurs dc choisis, on utilise un pont en H : un L298N. Il y a 6 pins à utiliser pour contrôler les deux moteurs, 3 par moteur. Parmi ces pins, deux d'entre eux permettent de choisir le sens de rotation du moteur par leur mise en état haut ou bas logique en suivant la logique du tableau suivant :

Etat pin IN1	Etat pin IN2	Etat du moteur
HAUT	BAS	Rotation sens 1
BAS	HAUT	Rotation sens 2
BAS	BAS	Arrêt

Le dernier pin (EN) reçoit un signal logique PWM. La tension envoyée au moteur sera alors proportionnelle au rapport cyclique de ce signal.

i. 1ère version du code de l'asservissement moteur (Guillaume)

PID avec méthode classique Arduino :

Dans un premier temps, il a été déterminé par quelques tests sur alimentation continue, que les moteurs nécessitaient un asservissement pour pouvoir être implémentés sur les PAMI. Les prémisses de cet asservissement se sont basées sur un PID numérique inspiré d'une méthode trouvée en ligne (https://curiores.com/positioncontrol). Cette première méthodologie se base sur la lecture des tics envoyés par les encodeurs à rythme régulier, pour le calcul en direct de l'erreur de position moteur. On utilise ensuite l'équation numérique d'un asservissement proportionnel intégral dérivé, en plus de l'erreur calculée, pour commander le moteur en boucle fermée.

Les étapes pour cette implémentation sont les suivantes :

1) Récupération de l'erreur par une fonction en interruption qui se déclenche sur une détection de front montant sur un encodeur précisé par son numéro de pin :

On utilise ici le préfixe *IRAM_ATTR* qui permet de passer la tâche en RAM pour optimiser son exécution sachant qu'elle se répète sur toute la durée du code pour récupérer la position moteur.

2) Calcul de la base de temps pour le calcul de l'erreur :

Utilisation de la fonction *micros()* qui permet de récupérer le temps depuis lequel l'ESP32 à commencer à exécuter le code, que l'on compare avec une valeur précédente que l'on a sauvegardé lors du passage précédent dans la boucle.

3) Calcul des coefficients proportionnel, intégral et dérivé :

Ici on traduit les équations suivantes dans le domaine numérique :

Erreur de position :

$$error = consigne - positon$$

Coefficient de la composante dérivé :

$$\frac{de}{dt} \Rightarrow \frac{e - e_{prev}}{\Delta t}$$

Coefficient de la composante intégrale :

$$\int e(t)dt \Rightarrow e_{int} + e.\Delta t$$

4) Calcul du PID :

$$u(t) = k_p.e(t) + k_i.(e_{int} + e.\Delta t) + k_d.\frac{e - e_{prev}}{\Delta t}$$

On obtient donc une approximation numérique d'un PID, qui peut être appliqué sur la commande des moteurs. Pour ce faire, on vérifie que celle-ci, qui sera appliqué en tant que PWM, ne dépasse pas 255, puis on l'applique sur le moteur.

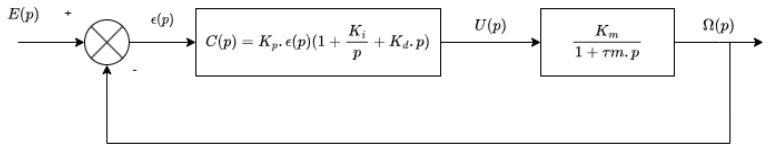


Figure 11 : Schéma bloc de l'asservissement en position avec PID

A la suite de cette implémentation, les tests sur un 1er moteur furent peu concluants car les coefficients du correcteur étant réglés de façon incorrecte.

Réglage du PID avec Zieger-Nichols :

Dans un second temps il a été nécessaire de trouver un moyen de régler ces coefficients de façon empirique due aux non-linéarités des moteurs. Pour cela nous avons appliqué la méthode de Ziegers-Nichols. Cette méthode repose sur l'implémentation d'un correcteur proportionnel dont le coefficient K_p est modifié jusqu'à obtenir un système oscillant et qui le reste. Une fois cela déterminé, la période d'oscillation doit être aussi relevée. A partir de ces deux résultats, la méthode nous donne les coefficients que l'on doit appliquer pour tous les types de régulateurs (P, PI, PID ...).

Après avoir utilisé cette méthode dans de nombreux test, il s'est avéré que l'implémentation de l'asservissement était incorrecte, et que pour deux moteurs, les boucles ne s'exécutaient pas en même temps, causant des problèmes de synchronisation.

Découverte et utilisation de RTOS :

Après documentation, il s'est présenté à nous que la gestion de l'exécution des tâches sur l'ESP32 pouvait être gérée manuellement. L'utilisation de RTOS permet d'attribuer à une fonction une priorité et un cœur d'exécution. Dans l'objectif de pouvoir asservir en position deux moteurs en parallèle, il était donc logique d'attribuer aux fonctions des correcteurs une priorité élevée sur des cœurs différents.

Malgré plusieurs tests avec différents types de correcteurs, cette méthode n'a jamais abouti au bon fonctionnement de l'asservissement en position.

Problématique de la vitesse des moteurs :

Il s'est avéré que les moteurs DC choisis possédaient une différence de vitesse de sortie notable pour une même commande en entrée. C'est donc pour cela qu'un asservissement en vitesse en plus de celui en position s'est présenté comme une solution.

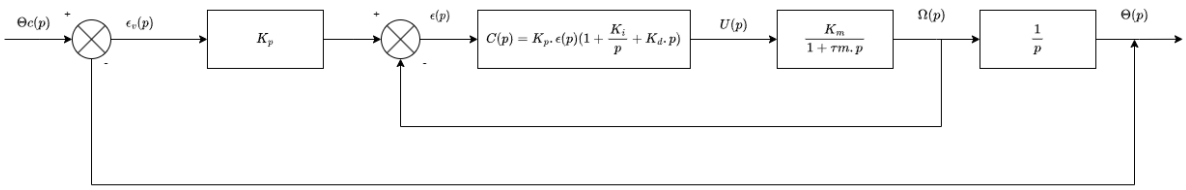


Figure 12 : Schéma bloc de l'asservissement en vitesse couplé à celui en position

C'est à l'issue des tests peu concluants de ce type d'asservissement qu'il a été pris la décision d'échanger de rôles pour que Estebane s'occupe du code et y apporte un regard nouveau.

ii. 2ème et dernière version du code de l'asservissement moteur (Estebane)

Tout d'abord, Nous avons dû réaliser un asservissement en vitesse. En effet, nous nous sommes rendu compte que les moteurs que nous avons choisis ne tournaient pas à la même vitesse pour un même signal de commande. Il faut donc dans un premier temps les asservir en vitesse avant de les asservir en position.

On commence par soustraire la vitesse mesurée à la consigne pour obtenir l'erreur, puis on corrige celle-ci avec un correcteur Proportionnel Intégral (PI). On calcule finalement la valeur du rapport cyclique de la pwm qui servira de signal de commande envoyé à notre L298N.

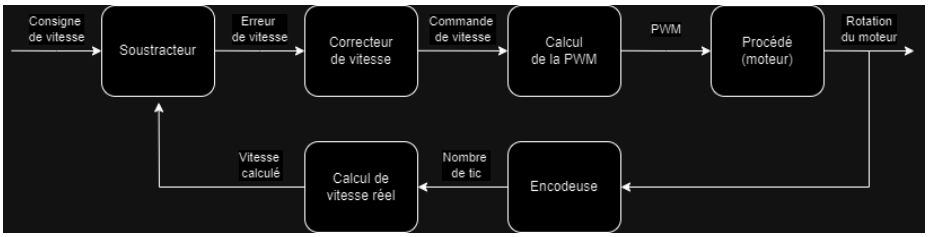


Figure 13 : Schéma bloc de l'asservissement en vitesse

Maintenant que les moteurs sont asservis en vitesse, il faut les asservir en position. L'avantage est que les encodeuses donnent la position directe des moteurs. On peut donc calculer l'erreur par rapport à la consigne de position, qui sera donnée en tic. On calcule ensuite la commande de vitesse et on utilise l'asservissement en vitesse.

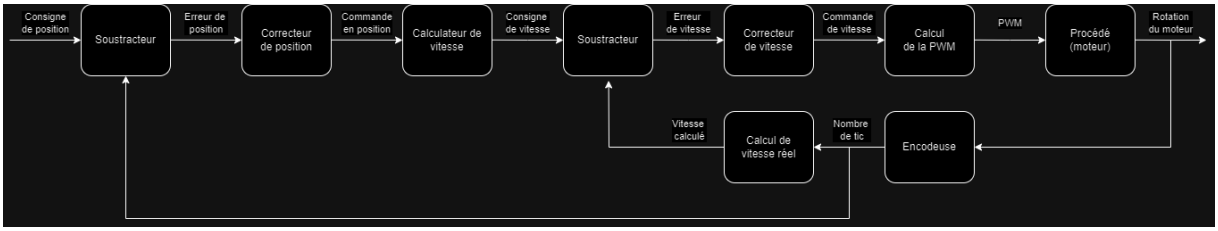


Figure 14 : Schéma bloc de l'asservissement en vitesse et en position

Nous savons à présent comment asservir les moteurs, mais pas comment implémenter ceci sur notre esp32. Les moteurs sont équipés d'encodeuses qui permettent de récupérer leurs sens et leur distance parcourue, c'est cela qui nous permettra de réaliser le retour (feedback). Lorsque le moteur tourne, deux signaux numériques sont envoyés au microcontrôleur. Une interruption scrute alors le signal 1 et se déclenche à chaque front montant : si le signal 2 est à un niveau haut, alors cela signifie que le moteur a avancé dans un sens, sinon dans l'autre. On incrémente alors un compteur de plus ou moins un « tic » à chaque déclenchement de cette interruption : ici nos moteurs réalisent un tour en 200 tics. Cette étape fut la première réalisée car nous devions être sûr du bon fonctionnement des encodeuses, car les informations reçues permettront de debugger notre code et seront absolument nécessaires pour le bon fonctionnement de notre asservissement.

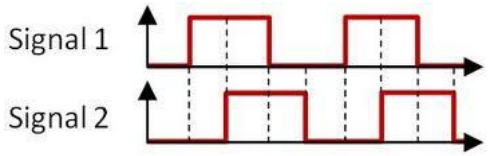


Figure 15 : Signaux issus des deux encodeuses

Nous avons eu beaucoup de problèmes pour réaliser le forward de l'asservissement. Nous avons donc tenté de réaliser une interruption, qui mettrait à jour la pwm à chacun de ces déclenchements. Cette méthode permet également de maîtriser la fréquence à laquelle les celles-ci se déclenchent. Cet asservissement fonctionne relativement bien, on calcule le temps qui s'est écoulé depuis la dernière interruption, ce qui permet de résoudre les problèmes de calcul de base de temps, et enfin on met à jour la pwm avec la bibliothèque *ledc*. Cette bibliothèque permet de gérer la création, la modification et l'envoi d'un signal pwm sur un pin choisit, à partir de la spécification du channel et de sa valeur de rapport cyclique compris entre 0 et 255. Cependant, l'esp32 avait tendance à redémarrer au bout d'un laps de temps variable mais trop court pour être utilisable lors d'un match. En effet, la mémoire allouée aux interruptions était trop petite pour l'ensemble des calculs de correcteur, et cela générait des overflows de registres qui provoquait le redémarrage du microcontrôleur.

Finalement, la méthode retenue et qui apporte le plus de stabilité repose dans la conservation de la structure classique Arduino : une fonction *setup()* qui s'exécute une seule fois

au démarrage, une fonction *loop()* qui s'exécute en boucle ainsi que des interruptions qui scrutent le retour des encodeuses. On réalise toute la mise à jour des PWM dans la partie *loop()*.

On utilise des correcteurs PID pour les asservissements de nos deux moteurs. Il reste maintenant à régler les coefficients des correcteurs. Le problème pour régler ceux-ci avec la méthode apprise en cours repose sur la détermination de la fonction de transfert du procédé, qui nous est habituellement donnée. Il nous faut donc utiliser d'autres méthodes. Grâce aux encodeuses et à l'IDE, on peut tracer les courbes de vitesse et de position des moteurs, et donc leurs réponses à un échelon. On peut à partir de là utiliser la méthode empirique basée sur la méthode de Ziegler-Nichols, pour obtenir les caractéristiques voulues : on regardera notamment le dépassement, le temps de réponse à 5% et le temps de monté. A noter que l'on ne garde pas les mêmes coefficients en fonction du correcteur.

Finalement, l'asservissement obtenu est utilisable, car précis au tic près, mais améliorable car le pami continue de dévier sur de longues distances.

c) Détection des obstacles et algorithme de recherche de chemins

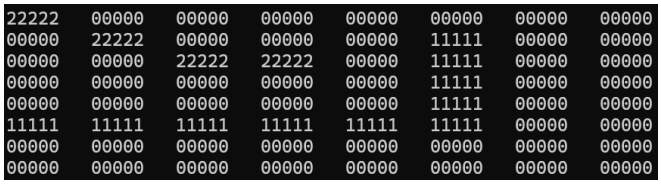
Dans le cadre de la compétition, le pami doit être capable de reconnaitre son environnement et de réagir en conséquence. Il est alors équipé par des capteurs à ultrason qui sont capable de détecter un obstacle et d'en mesurer la distance à laquelle ils se trouvent. Dans un premier temps, on implémentera une stratégie qui consistera à s'arrêter lors d'une détection d'un obstacle trop proche, mais celle-ci peut être améliorée avec une stratégie d'évitement.

On veut donc que le pami soit capable de calculer un itinéraire qui lui permettrai de naviguer d'un point à un autre de la table en évitant tous les obstacles. Cet algorithme devra pouvoir être exécutable sur un microcontrôleur, donc avoir une complexité limitée, et être codé en C ou C++. Il est à noter que les cours de C++ n'avaient pas encore eu lieu lors du codage de cet algorithme, il a donc été implémenté en C et n'a pas été modifié par la suite. Des problèmes ont été rencontré avec l'utilisation du C, qui aurait été solvable simplement si nous avions su coder en C++ lors de sa création, notamment des problèmes liés aux tableaux et à leur taille non modifiable, qui doit obligatoirement être connue à l'avance, ainsi qu'à l'absence des méthodes *pop()* et *push_back()*.

Pour mettre en place un algorithme permettant de trouver un chemin sur le plateau de la compétition, on travaille sur une grille qui représente la table en version discrétisée. On considère que la distance entre chaque point voisin latéralement et horizontalement est égale.

Le premier algorithme envisagé a été l'algorithme A*, un algorithme qui, s'il possède une bonne heuristique, peut atteindre une complexité quasiment linéaire. L'heuristique en question est la capacité de l'algorithme à fournir une estimation précise et efficace de la distance restante pour atteindre la destination voulue, à partir de chaque nœud du graphe. La difficulté de la mise en place de cet algorithme réside dans les 2 listes à tailles variables nécessaires à son bon fonctionnement : une liste des nœuds visités et une liste des nœuds à visiter. Nous voulions l'algorithme le plus polyvalent possible, ainsi fixer les tailles de ces tableaux reviendraient soit à

fixer une limite de distance entre le départ et l'arrivé, soit à implémenter un tableau trop grand pour la mémoire de l'ESP32. Nous avons donc tenté de reproduire le fonctionnement de l'algorithme A* avec deux structures qui créent respectivement les nœuds et la grille sur laquelle nous évoluons. La structure nœud possède des attributs pour noter si le nœud a été visité, s'il est un obstacle etc... L'algorithme ainsi obtenu réussi à trouver un chemin lorsqu'il n'y a aucun obstacle, et lorsque les obstacles sont simples à éviter. Cependant, la fonction heuristique que nous avons réalisée ne permet pas de trouver tous les chemins. Par exemple sur la grille suivante, où le point de départ est en haut à gauche et l'arrivé en bas à droite, et où les obstacles sont notés 11111 et le chemin suivit est noté 22222, on observe que l'algorithme est bloqué.



```
22222   00000   00000   00000   00000   00000   00000   00000
00000   22222   00000   00000   00000   11111   00000   00000
00000   00000   22222   22222   00000   11111   00000   00000
00000   00000   00000   00000   00000   11111   00000   00000
00000   00000   00000   00000   00000   11111   00000   00000
11111   11111   11111   11111   11111   11111   00000   00000
00000   00000   00000   00000   00000   00000   00000   00000
00000   00000   00000   00000   00000   00000   00000   00000
```

Figure 16 : Grille de l'algorithme A avec blocage*

L'algorithme que nous avons ensuite testé et validé est l'algorithme de Dijkstra. Cet algorithme a une complexité polynomiale, soit une complexité plus grande que le A*. Cependant, il a l'avantage d'être plus simple à coder en C, puisqu'il est récursif, et qu'il est assuré de trouver un chemin optimal s'il y en a un. L'algorithme fonctionne comme ceci : il marque le nœud actuel comme visité dans la grille, il explore ensuite les voisins non visités du nœud actuel et met à jour leur distance par rapport au nœud de départ si nécessaire. Il recherche le prochain nœud à explorer en sélectionnant celui avec la plus petite distance parmi les nœuds non visités. Par la suite, il s'appelle récursivement avec le prochain nœud à explorer jusqu'à ce que le nœud objectif soit atteint. Une fois le nœud objectif atteint, l'algorithme se termine, et le chemin le plus court est alors retracé en remontant les parents des cellules du nœud objectif jusqu'au nœud de départ.

Pour finir, on rajoute une phase de de traitement supplémentaire à l'initialisation. Lors du choix de chemin avec notre algorithme, on considère notre robot comme un point. C'est une erreur car le pami est large et prend donc plusieurs points dans la réalité. Pour remédier à ce problème, on applique un prétraitement à la grille qui consiste à marquer comme des obstacles un certain nombre de point autour des points marqués comme des obstacles. Le pami se déplacera alors en prenant de la marge par rapport aux obstacles et les évitera convenablement.

On peut voir sur cette dernière image une application de l'algorithme de Dijkstra : chaque numéro correspond à un point, et le numéro correspond à l'état de ce point. A la base chaque point est à 0, pour montrer que le point est libre. Puis les points considérés comme des obstacles sont notés avec des 1, et les points où le pami n'a pas le droit de passer seront notés avec des 2. Finalement, le pami part d'en haut à gauche et doit arriver en bas à droite, et chaque point parcouru entre son point de départ et sa destination est noté avec un 3. On observe que l'algorithme trouve bien le chemin optimal.

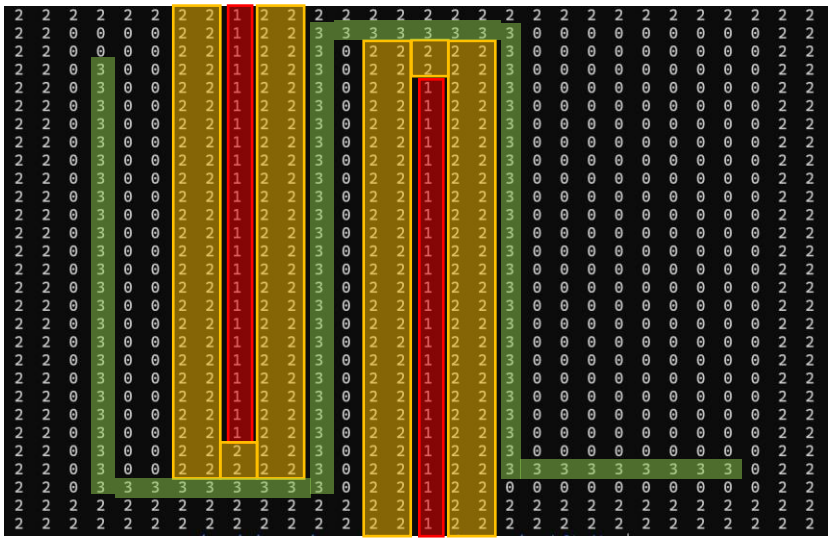


Figure 17 : Test de l'algorithme de Djikstra

Au moment de l'écriture de ce rapport, l'algorithme de pathfinding est prêt à être implémenté : à partir de la grille ci-dessus, nous obtenons les mouvements à réaliser par le PAMI. Cependant, cet algorithme n'est utilisable que si l'asservissement est assez précis. Celui-ci devant encore être amélioré, l'algorithme n'est pas encore utilisé. Cependant, si nous devions l'implémenter, la stratégie serait la suivante : On calcule le chemin idéal en considérant que la table est vide, et lorsqu'on trouve un obstacle, on met à jour les paramètres de la grille et on recalcule un nouveau chemin. On répète l'opération jusqu'à atteindre notre objectif. Une liaison wifi ayant été établie entre les PAMI, et le robot principal étant équipé d'un routeur wifi et relié à une caméra qui filme la table, il serait envisageable de repérer les obstacles avant le départ des PAMI, et de calculer un chemin optimal avant même d'avoir rencontré les obstacles.

d) Modélisation et Fabrication du PAMI

Le point de départ de cette modélisation repose sur un pavé ou la base était constituée du PCB sur lequel on venait fixer les moteurs, l'ESP32, le L298N ainsi que tous les fils pour la connexion des capteurs. De plus, la batterie était intégrée dans un petit emplacement situé à l'arrière pour des questions d'ergonomie et de répartition des charges. Il avait été aussi décidé que 3 capteurs ultrasons seraient équipé sur le devant et les deux côtés du PAMI.

Il est très rapidement apparu qu'avec la limite de taille de celui-ci, ce type de mécanique ne pourrait pas être viable avec des composant en discret, et un PCB en composants traversants. En effet, si tout ce que nous avions prévu avait été intégré dans un PCB, la surface de la base d'un de nos PAMI aurait très fortement limité le nombre de ses confrères dans la zone de départ.

Il a donc été décidé dans un second temps de considérer le PCB comme étant le haut du pavé. Avec cette conception, le problème de place lié aux éléments tels que les moteurs et le L298N disparait car étant fixés sur sa base.

Figure 18 : 2ème version du PAMI

Malgré cette nouvelle conception, avec le rajout des différents fils ayant pour but de connecter entre eux les différents éléments, est apparu un nouveau problème : le PAMI n'est pas assez haut et certains connecteurs appuient sur le L298N et son dissipateur. Il a donc été nécessaire d'implémenter une dernière version plus haute, et prenant en compte l'ajout d'un bouton d'arrêt d'urgence obligatoire pour son homologation, ainsi qu'une tirette pour son déclenchement en début de match.

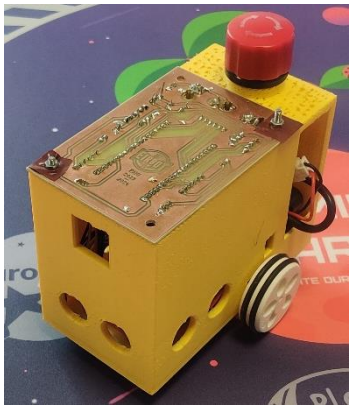


Figure 19 : 3ème version du PAMI

Cette version a été amenée à évoluer légèrement par l'ajout d'un PCB fabriqué industriellement et d'interrupteurs permettant de choisir la stratégie liée à l'équipe désignée en début de match.

En plus de cette modélisation, il a été impératif de modéliser des roues « parfaites » pour qu'elles s'adaptent au mieux aux moteurs, dans le but de ne pas fausser l'asservissement par une mécanique défaillante.

Figure 20 : Roue du PAMI

Il était important d'avoir une adhérence maximale pour garantir l'asservissement, il a donc été ajouté un joint en caoutchouc dans la rigole de la roue présente sur la figure précédente.

e) Communication sans fils

Les ESP32 sont équipés d'une puce wifi et d'une puce Bluetooth. Nous souhaitons faire communiquer les faire communiquer entre elles pour pouvoir coordonner leurs départs. L'établissement d'une connexion sans fils pourrait aussi à terme permettre de récupérer la position des obstacles présents sur la table, et de calculer des chemins optimaux. Il faut noter que lors du concours, les bandes passantes wifi et Bluetooth sont saturées par les autres robots présents sur site.

Nous avons commencé par tenter de réaliser une connexion Bluetooth entre les ESP32, en utilisant la bibliothèque *Bluetooth.h*. En effet, le Bluetooth est censé être moins sensible aux interférences. Le code de base pour les connecter est très court mais nous n'avons jamais réussi à le faire. Par manque de temps, nous avons vite abandonné pour essayer une autre méthode.

Nous avons donc établi un serveur web asynchrone sur une des ESP32. Les autres pourront alors se connecter en wifi à ce serveur et lui envoyer des requêtes. Nous pouvons configurer ces réponses sur le serveur et avons donc réalisé un système de communication entre les ESP32. Les PAMI étant côte à côte lors du démarrage, il ne devrait pas y avoir de problème d'interférences, et dès qu'ils démarrent, la connexion n'est plus nécessaire.

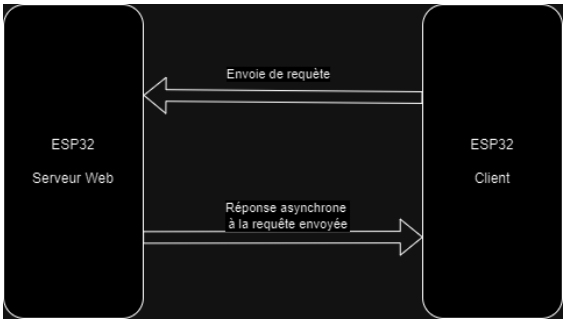


Figure 21 : Principe de communication en Wifi entre ESP32 avec un serveur web

La stratégie de démarrage est la suivante : Les PAMI envoient des requêtes à intervalle régulier en attendent de recevoir le signal de départ. Lorsque la tirette est déclenchée (élément physique de déclenchement), le timer du pami servant de serveur web est déclenché, et lorsqu'il dépasse les 90 secondes, il envoi le signal de départ aux autres PAMI. Après leur départ, les PAMI se déconnectent entre eux.

f) Electronique d'interconnexions et d'alimentation

Dans un premier temps, l'interconnexion des différents éléments s'est faite avec des câbles Arduino sur notre 1er prototype, ce qui n'était pas optimal pour la fiabilité de branchement, et ce qui occasionnait régulièrement des faux contacts. (voir figure 10)

Dans l'optique du projet et de notre participation avec l'association EIRBOT, le design du PCB a été réalisé par nous-même avec l'utilisation de logiciels open source (Kicad). Ce PCB devait intégrer les éléments suivants :

- L'ESP32
- Le L298N

- Une connexion pour ou un régulateur step-up 7V -> 12V
- Une connexion pour la batterie
- Une connexion pour les moteurs
- Un interrupteur pour changer l'alimentation de l'ESP32 entre l'USB et la batterie
- Une alimentation 5V à partir de la tension batterie pour les capteurs ultrasons ainsi que l'ESP32

Dans sa 1ère version, cette carte était en correspondance avec l'ancienne version mécanique du PAMI. De plus, dans sa première itération, celle-ci présentait des erreurs de conception :

- Plan de masses non-reliés
- Pads de composants trop petits

En plus de ces erreurs, des problèmes d'ordre électroniques sont apparus :

- Alimentation de l'ESP32 par un régulateur 5V (LM7805) insuffisante
- Alimentation des encodeuses des 2 moteurs par le 3.3V de l'ESP32 insuffisante

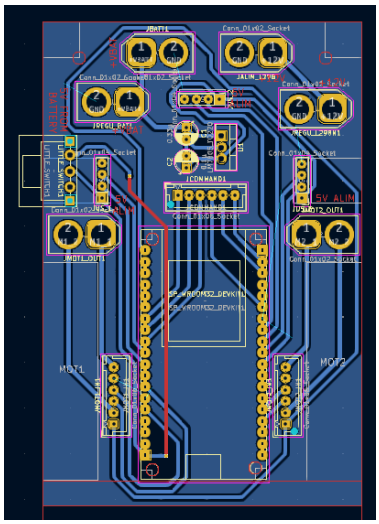


Figure 22 : 1ère version du PCB du PAMI

La 2ème itération de cette carte a permis de résoudre beaucoup de nos précédents problèmes :

- Alimentation de l'ESP32 par le régulateur 5V du L298N
- Alimentation des encodeuses des moteurs par le régulateur 5V

En plus de la résolution de ces problèmes, cette 2ème version nous a permis d'ajouter les fonctionnalités suivantes qui se sont dessinées au fur et à mesure des tests sur la 1ère version :

- Ajout d'un interrupteur pour le changement de stratégie
- Ajout d'un connecteur pour la tirette

Les deux versions précédentes avaient pour but d'être fabriquées au sein du laboratoire électronique de l'école, mais la dernière implémentation de ce PCB a eu pour but d'être fabriqué industriellement et en moyenne quantité pour une question de fiabilité et de reproductibilité.

Cette itération finale a subi quelques modifications d'empreintes par rapport à la disponibilité de certains composants, ainsi que des modifications visuelles.

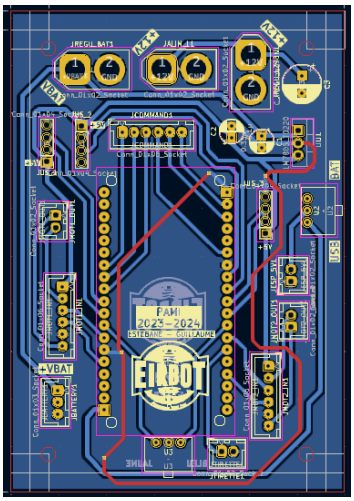


Figure 23 : Version finale du PCB du PAMI

IV - Résultats obtenus

a) Asservissement des moteurs

Dans la forme finale de notre projet, nous avons réalisé un asservissement qui permet au PAMI de se déplacer convenablement sur la table de la coupe de France : les déplacements sur de courtes distances ne sont que rarement un problème, mais les déplacements sur de longues lignes droites posent encore problème, car le pami dévie. L'asservissement doit donc encore être amélioré, et des tests sont en cours de réalisation dans l'objectif que cela fonctionne pour le mercredi 8 mai, date de l'homologation.

b) Détection des obstacles et algorithme de recherche de chemins

A l'issue des tests sur les différents algorithmes, nous en avons obtenu un capable de trouver le chemin optimal, s'il y en a un, à travers une grille en 2D avec des obstacles. Le pami est également capable de repérer son environnement grâce à des capteurs à ultrason. Cependant, dans l'optique de l'amélioration de l'asservissement, celui-ci n'a pas encore été implémenté sur les PAMI.

c) Modélisation et fabrication du PAMI

Cette étape touche à sa fin car nous avons réussi à créer et assembler un pami qui respecte l'ensemble des critères que nous nous sommes imposés, et que le règlement de la coupe de France de robotique indique.

d) Communication sans fils

Cette étape a été implémentée et testée que récemment, en lien avec la priorité que représentait un asservissement fonctionnel. Nous avons donc réussi à mettre en place une communication wifi entre nos ESP32 : l'une prend le rôle de serveur web, et les autres représentent les clients qui envoient des requêtes au serveur.

e) Electronique (PCB)

Après de nombreuses itérations de test en vue d'améliorations, nous avons finalement conçu et produit un PCB d'interconnexion fonctionnel, qui permet de relier l'ensemble de nos composants et capteurs. La version finale comprend comme source d'alimentation principale une batterie délivrant une tension de 7.4V, et un convertisseur de tension nous permettant d'élever celle-ci à 12V pour notre pont en H. D'autre part, l'alimentation de l'ESP32 est gérée par le régulateur 5V de la carte qui comprend ce même pont en H. En plus de tout cela, cette carte assure toutes les fonctionnalités liées au différents capteurs et actionneurs que nous nous étions fixés, et que le règlement nous impose.

Figure 24 : Version finale du PAMI

V - Conclusion

Tout au long ce projet, notre objectif était de concevoir et développer un Petit Actionneur Mobile Indépendant (PAMI) conforme aux exigences strictes de la Coupe de France de Robotique, et à notre stratégie en vue de notre participation. Avec une répartition des tâches efficace et une collaboration étroite, nous sommes parvenus à nos objectifs dans la réalisation de notre mission.

Malgré les efforts déployés et les avancées réalisées, il est essentiel de reconnaître que notre système d'asservissement n'a pas encore atteint son potentiel optimal. Bien que nous ayons réussi à mettre en place des algorithmes de recherche de chemin ainsi que des codes d'asservissement pour les moteurs DC, des ajustements et améliorations restent nécessaires pour obtenir des performances optimales en vue de notre participation.

À l'approche de la date butoir de la Coupe de France de Robotique, nous avons dû concentrer nos efforts sur la finalisation de la conception et de la fabrication des composants mécaniques et électroniques, ce qui a laissé peu de marge pour des itérations supplémentaires visant à optimiser l'asservissement. Cependant, nous sommes convaincus que les fondations

que nous avons posées au niveau de la brique d'asservissement nous permettront de l'améliorer pour en obtenir une version fiable et efficace pour le jour de la compétition.

En conclusion, ce projet nous a permis de mettre en pratique nos connaissances en robotique, mécatronique, informatique et électronique dans un contexte réel et stimulant par les contraintes imposées. Bien que nous n'ayons pas encore atteint l'objectif final d'un asservissement optimal, nous sommes fiers des progrès réalisés et des compétences acquises tout au long de ce processus. Nous sommes impatients de poursuivre notre travail pour affiner et améliorer le système de contrôle de notre PAMI.

VI - Perspectives du projet

La 1ère perspective de ce projet repose sur sa capacité à répondre aux critères imposés par la Coupe de France de robotique dans l'horizon d'être homologué. Dans la finalité de celui-ci, les différents types d'utilisation de ce type de mécanique s'ouvre à des domaines pouvant aller du robot de livraison sur roues au fauteuil roulants autonomes du futur.