

ECOLE NATIONALE POLYTECHNIQUE D'ALGER
DÉPARTEMENT D'ÉLECTRONIQUE.
LABORATOIRE DES DISPOSITIFS DE COMMUNICATION ET DE
CONVERSION PHOTOVOLTAÏQUE



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

RAPPORT DE STAGE

BOUCHAKOUR MOHAMED
KHELAFI MASSYLIA

19 JUIN 2021

Sommaire

I	Introduction	3
II	Construction Mécanique	5
1	Moteurs	6
1.1	Choix	6
1.1.1	Les dimensions	6
1.1.2	La disponibilité :	7
1.2	Modifications	7
2	Roues	10
3	Châssis	12
III	Alimentation Électrique	16
4	Betterie	17
5	Régulation et stabilisation	19
5.1	Élévation 7V	19
5.1.1	Principe de fonctionnement	19
5.1.2	Circuit utilisé	21
5.2	Régulation 5V	22
5.3	Alimentation et contrôle des moteurs	23
5.4	Problèmes rencontrés	24
IV	Capteur TCRT5000	25
5.5	Présentation du capteur	26
5.6	Fonctionnement	26
5.6.1	Solution	27
5.7	Circuit interne	28

5.8	Disposition sur le robot	29
5.8.1	Capteurs Avant	29
5.8.2	Capteurs d'intersection	30
5.8.3	Capteur Arrière	31
5.9	Code	31
V	Contrôle	33
6	Microcontrôleur	34
7	Algorithme de contrôle suiveur de ligne	36
8	Fonctions de mouvement	38
8.1	Turn Right	38
8.2	Turn Left	40
8.3	Rotation(Demi-Tour)	40
8.4	Stop	42
VI	Algorithme	43
9	Labyrinthe	44
10	Structure du code	45
10.1	Les Fonctions	45
10.1.1	Règle de la main gauche [left hand LH]	45
10.1.2	Règle de la main main droite [right hand RH]	48
10.1.3	Short-path	49
10.1.4	Translate	52
10.1.5	Increment	54
10.2	Partie setup	56
10.3	Partie loop	57
VII	Conclusion	59
VIII	Bibliographie	61

Chapitre I

Introduction

Du 4 avril au 22 mai, Nous avons effectué un stage interne a l'ECOLE NATIONAL POLYTECHNIQUE.

Nous avons choisi le thème que l'on voulait abordé durant ce stage et nous l'avons présenter a monsieur ADNANE qui a gentillement accepté de nous encadré.

Ce stage est un projet conçu suivant le cahier de charge de la compétition POLYMAZE, qui s'est déroulé le 22 mai a l'école nationale polytechnique et à laquelle nous avons eu l'honneur de participer.

Ce document résume des heures de travaille ,d'essais, d'échecs et d'apprentissage.

La version que l'on présentera dans ce dernier, n'est pas la première version ,cette dernière qu'on a appeler M2-KB était loin de notre but, qui était la conception d'un robot de type "MICROMOUSE", nous avons d'ailleurs rencontré beaucoup de problèmes durant la conception de cette première version, et malgré tous les efforts déployés durant la conception de cette version ; nous avons décider de tout reprendre a zéro .

La version qu'on présentera dans ce document est la version 2.0 du robot M2-KB de type "MICROMOUSE" : un suiveur de ligne capable de résoudre un labyrinthe.

Ce document couvrira en détails tout ce que l'on a apprit durant la conception de ce robot, microcontrôleur, les capteurs, les moteurs, la batterie.... que l'on a utilisé. L'algorithme sera détaillé en partant de l'introduction des différentes variables jusqu'à la résolution du labyrinthe,

Chapitre II

Construction Mécanique

Section 1

Moteurs

1.1 Choix

Le choix des moteurs a été basé sur plusieurs critères, mais principalement deux raisons nous ont poussé à les choisir :

1.1.1 Les dimensions

Nous voulions réaliser un robot de petites dimensions, et pour cela, on a eu besoin des moteurs les plus petits qu'on pourrait avoir, c'est pour cela qu'on a opté pour les moteurs DC «N20».



FIGURE 1.1 – Moteur N20

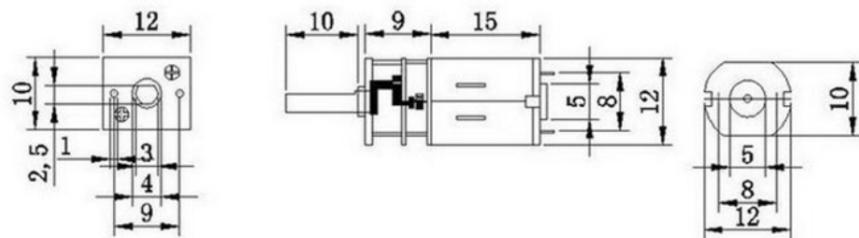


FIGURE 1.2 – Moteur N20

Ce moteur est disponible dans plusieurs configurations : 3v, 6v et 12v. Et pour chaque tension, plusieurs coefficients de réduction sont disponibles, ce qui détermine la vitesse et le couple du moteur.

1.1.2 La disponibilité :

Idéalement on aurait opté pour un moteur d'une vitesse de 300 jusqu'à 500rpm, avec une tension de 6V ou 12V.

Malheureusement, on a seulement pu trouver ce moteur en une seule configuration, 6v, 30rmp, et un rapport de réduction 1000 :1.

1.2 Modifications

Ce moteur est beaucoup trop lent pour notre application, 30rpm sur des roues de 42mm, va nous donner une vitesse de 4m/min, qui est beaucoup trop lente. C'est pour cela qu'on a dû modifier la boîte de vitesses du moteur, pour diminuer le rapport de réduction.

Cette modification consiste à supprimer un étage de réduction dans la boîte, comme on peut le voir dans les images suivantes :

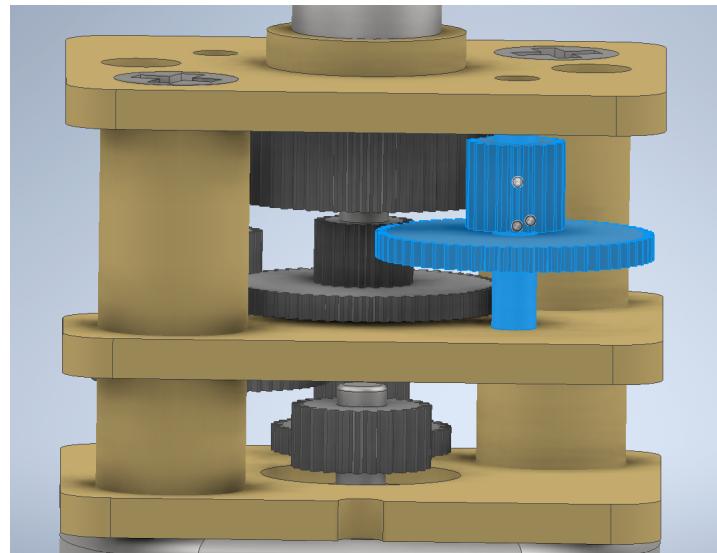


FIGURE 1.3

On enlève l'axe sélectionné en bleu dans ce modèle 3D, avec les engrenages qui vont avec.

Et on connecte les engrenages sélectionnés dans l'illustration suivante :

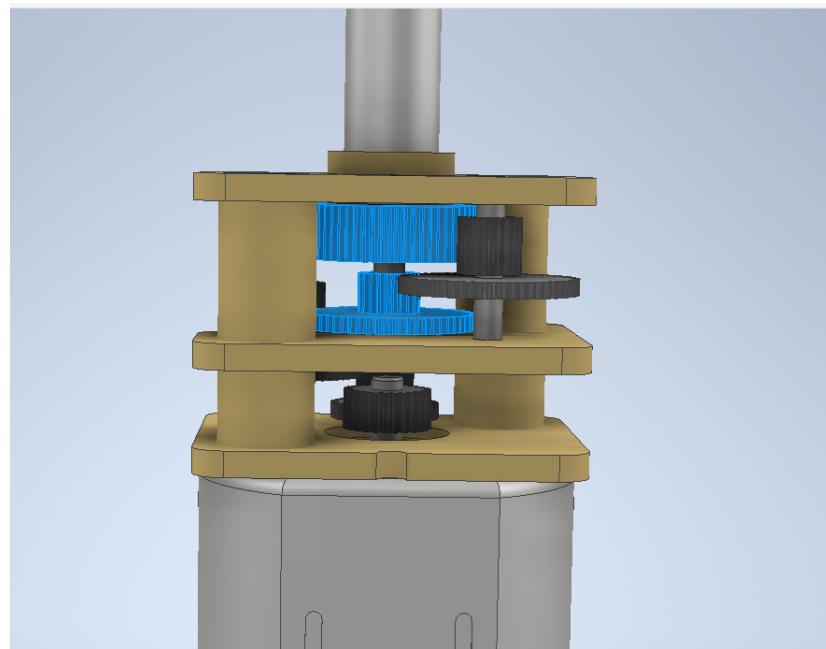


FIGURE 1.4

On obtient le résultat suivant :

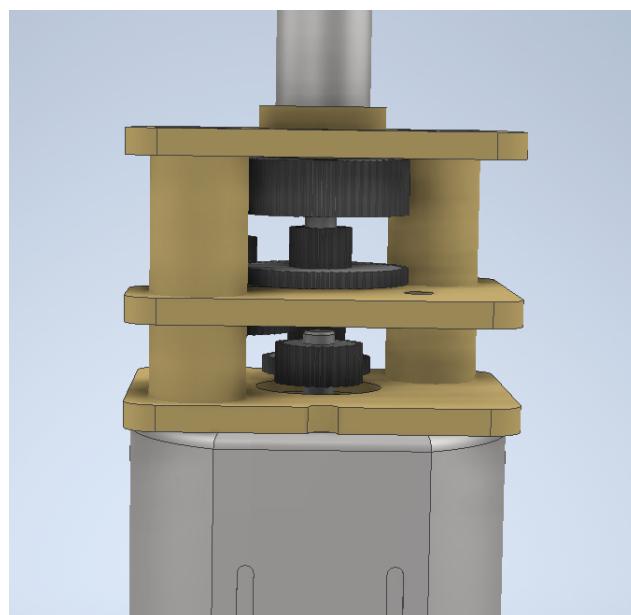
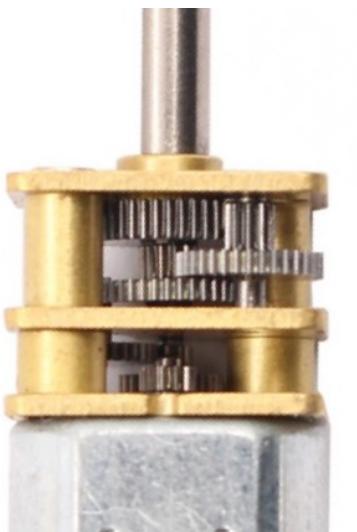


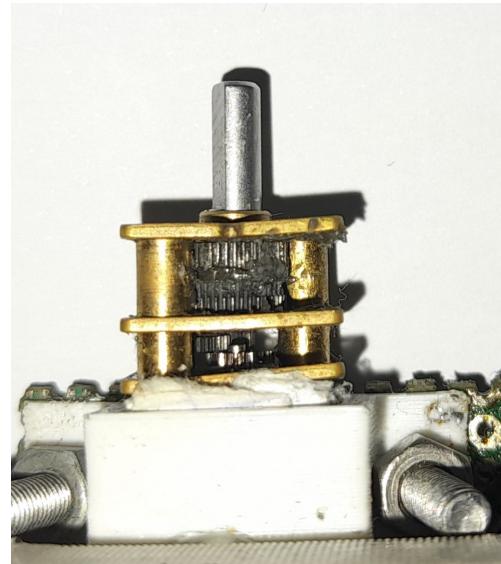
FIGURE 1.5

On a utilisé de la colle à base d'Epoxy, pour attacher les deux engrenages.

On peut voir le résultat final dans la figure ci-contre :



(a) Avant



(b) Après

FIGURE 1.6

Après cette modification, la vitesse du moteur a augmenté, et à vide, elle est approximativement de 500 rpm.

Avec cette configuration, le couple du moteur a considérablement diminué, ce qui nous pose des problèmes pour les faibles vitesses, mais la vitesse est bien plus que ce dont on a besoin.

Section 2

Roues

Dû à l'indisponibilité dans le marché, les roues ont été conçus et fabriqués manuellement.



FIGURE 2.1 – Modèle 3D des roues

Le modèle a ensuite été imprimé en 3D en utilisant du PLA, on voit le résultat ci-dessous :



(a)



(b)

FIGURE 2.2 – Résultat d'impression

La surface de l'objet imprimé est trop glissante, et ne peut être utiliser directement, c'est pour cela qu'on a ajouté une couche pour augmenter la friction avec le terrain.



FIGURE 2.3 – Surface

On a utilisé des joints de tubes, qui sont fabriqués en caoutchouc, comme revêtement de surface.

Cette petite modification augmente la tenue de route. Mais dès que la vitesse est augmenté, le glissement des roues augmente, et donc la stabilité diminue. Et donc, on doit se contenter des vitesses relativement basses.

Section 3

Châssis

Pour tenir la construction mécanique, on a utiliser directement la plaque perforé, sur laquelle on a fixe tout les autres composants. (Moteurs, composants, capteurs, roue...). On peut le voir ci-dessous :

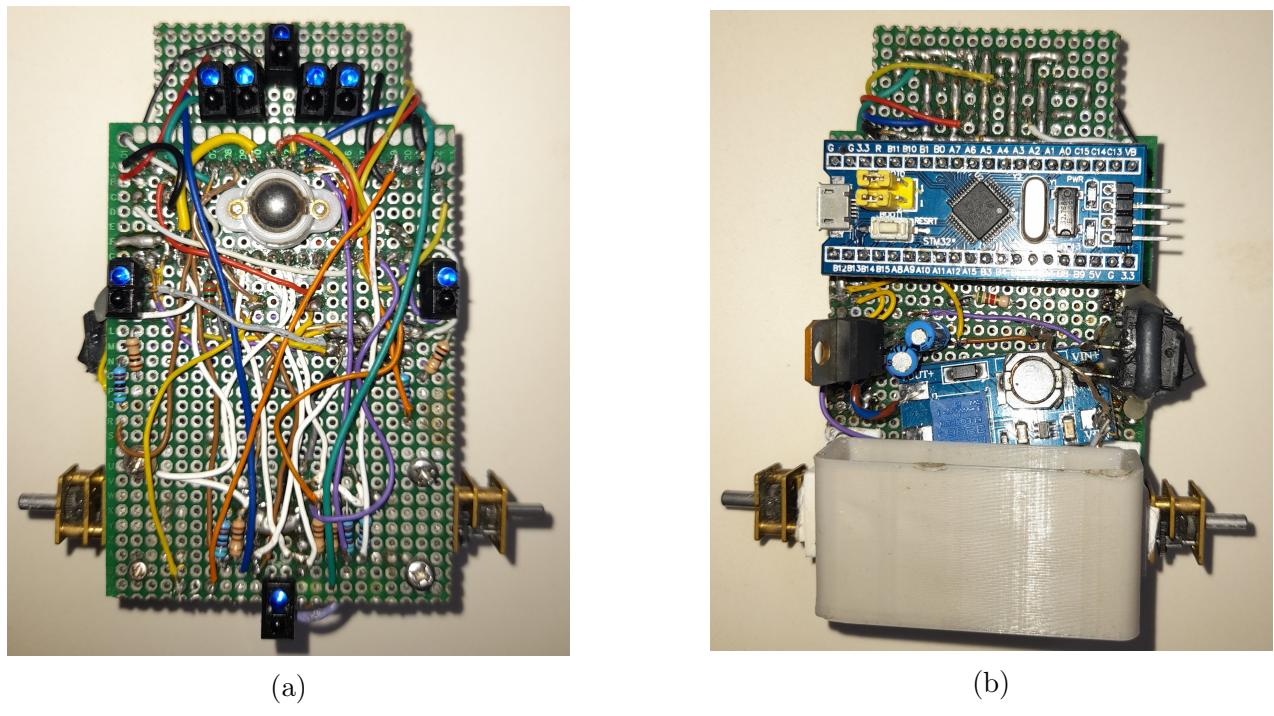


FIGURE 3.1

Cependant, la plaque perforé est seulement utilisé comme premier prototype. Pour la version finale, on utilise un circuit imprimé. Il va connecter (Mécaniquement et électriquement) les différentes parties du robot.

On a utilisé Autodesk Eagle pour la conception du circuit imprimé.

Comme première partie, on introduit le schéma de la circuiterie électrique dans le logiciel, et ça donne le résultat suivant :

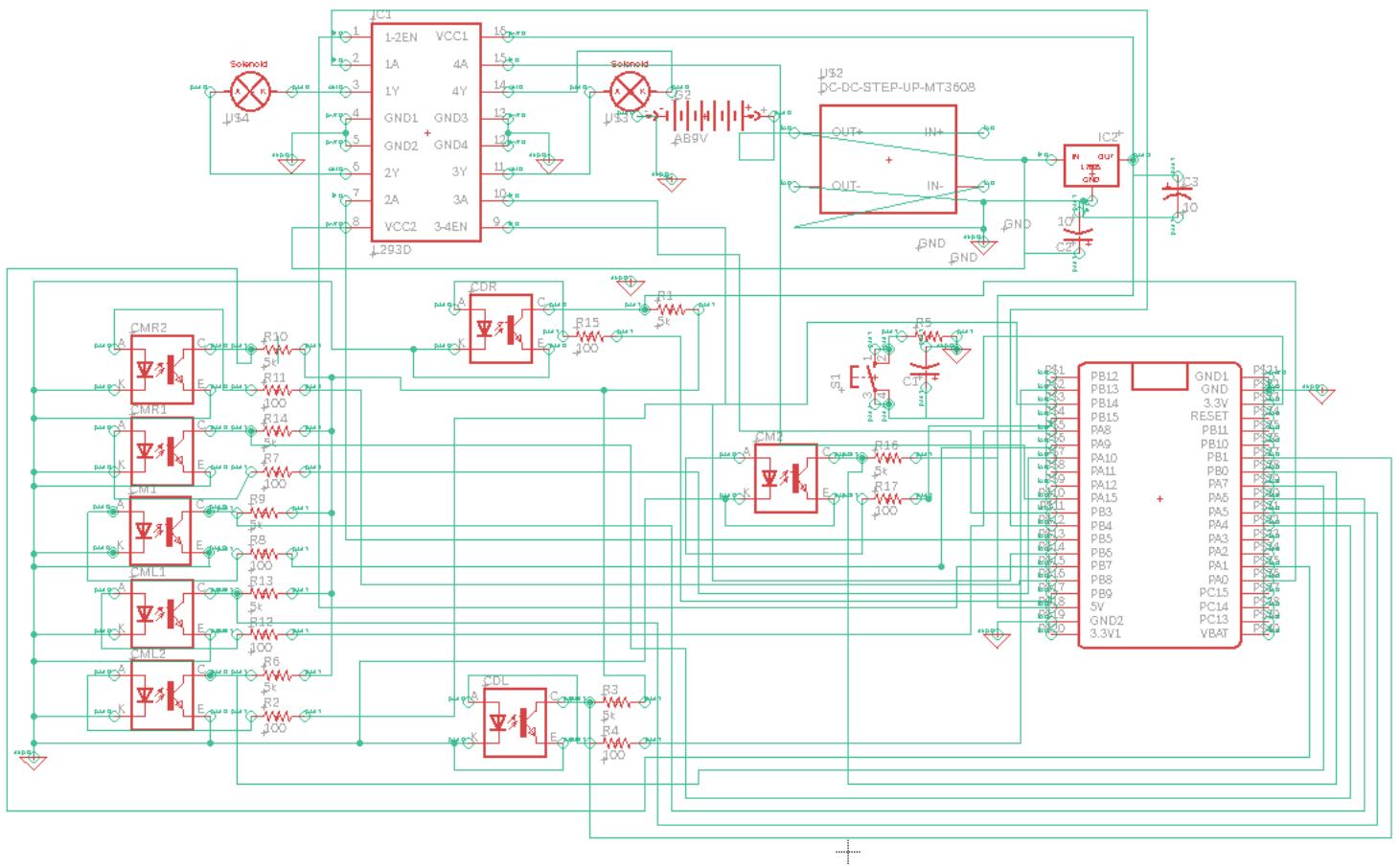


FIGURE 3.2 – Schéma du la circuiterie du robot

Après que cela a été fait, on bascule vers le «Board View» et on procède au placement de nos composants sur la plaque, en choisissant les dimensions du PCB.

On utilise des traces de 0.5mm pour les signaux de données et d'alimentation des petits composants. Ces traces peuvent supporter jusqu'à 0.5A. Qui est largement suffisant. Pour les connections des moteurs et des rails d'alimentation (5V, 7V, 3,7V...). On utilise des traces de 1mm. Qui peut peuvent supporter jusqu'à 0.8A de façon continue. Et peut supporter jusqu'à 2A pour de brèves délais.

Le circuit au totale utilise 0.5A max. Avec des pics de 1A.

De plus, un plan de masse a été mis en place. C'est a dire, tous les espaces non occupés sur la plaque seront mis en terre.

On ajoute aussi les trous pour placer les moteurs et la roue. Qui seront percés avec précision lors de la production du PCB.

Plan finale du circuit imprimé :

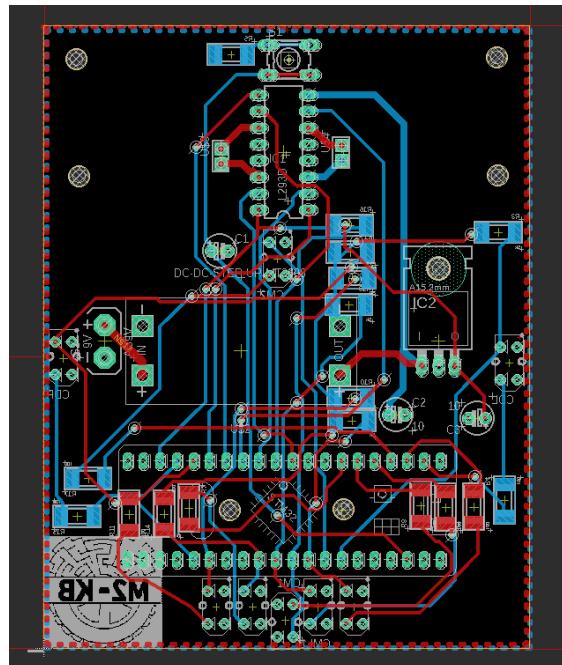


FIGURE 3.3 – Traces plan de masse

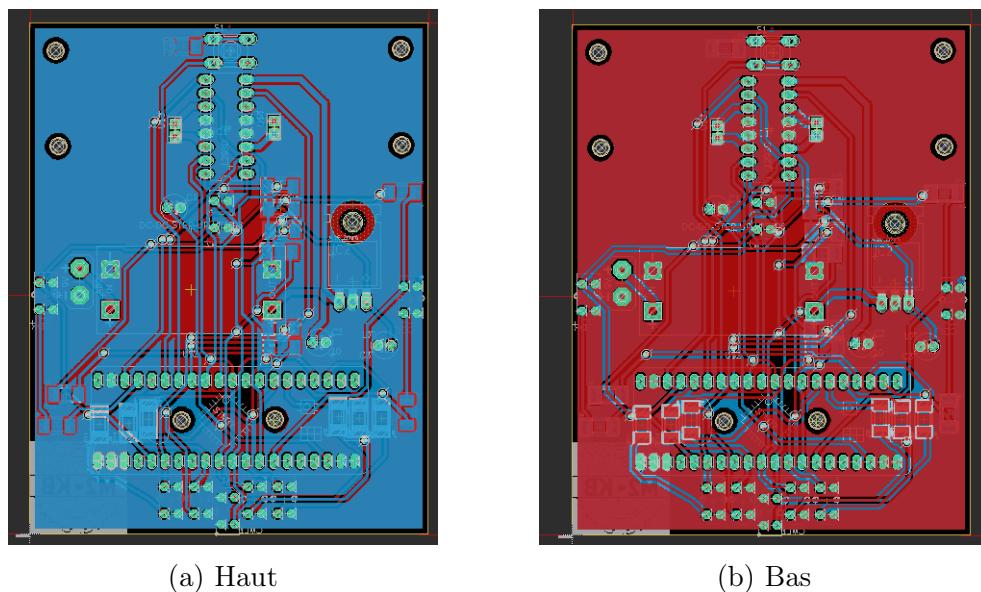
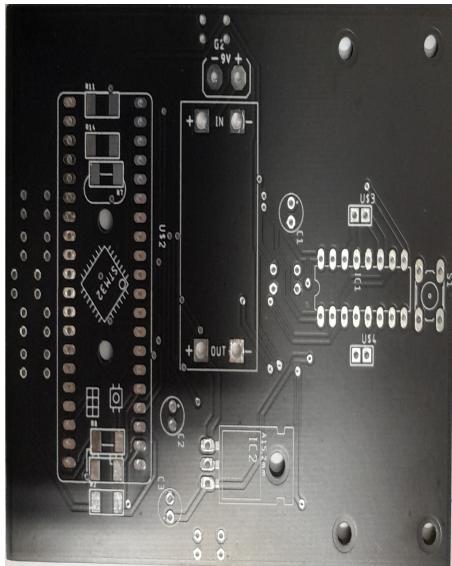
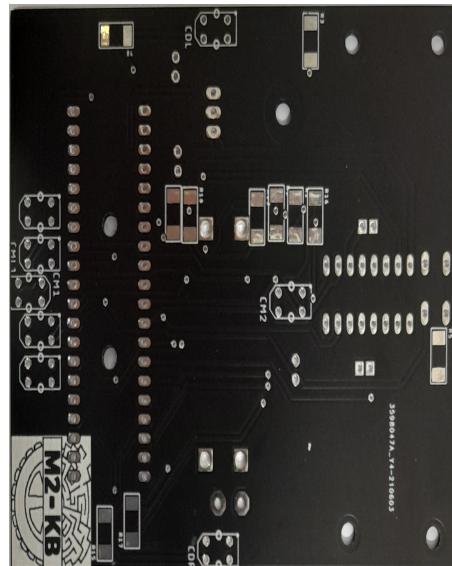


FIGURE 3.4 – Traces du PCB avec Plan de Masse



(a) Haut



(b) Bas

FIGURE 3.5 – Résultat final

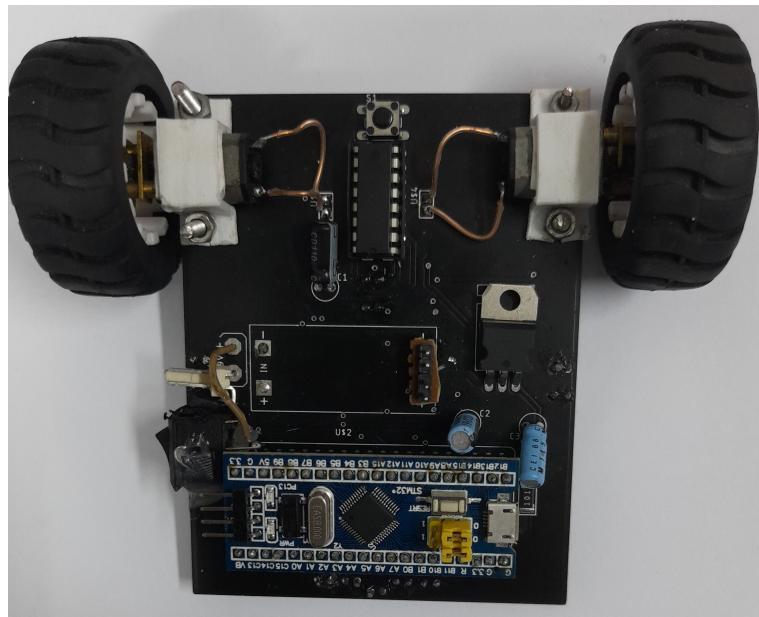


FIGURE 3.6 – Version finale du robot

Chapitre III

Alimentation Électrique

Section 4

Batterie

Pour le choix de batterie, plusieurs critères doivent être pris en compte, notamment la tension, la capacité et la courant maximal de sortie.

Les moteurs fonctionnent sur 6V, en prenant en compte la chute de tension au niveau du pont en H, pour obtenir 6V en sortie, il nous faut aux alentours de 7V en entrée. De plus, le régulateur 5V (7805) qu'on doit utiliser pour alimenter les capteurs et micro-contrôleur, a besoin d'une tension minimale de 6.5V en entrée pour pouvoir fonctionner. Et vu la tension 3.7V des batteries au lithium, qui sont les plus utilisées, il nous faudrait idéalement deux cellules en série.

Sachant que la consommation totale de notre circuit est aux alentours de 200mA sous 7V. Une capacité de 1500mWh nous donnerait assez d'autonomie pour effectuer nos test. Et largement assez de capacité pour compléter le labyrinthe.

De même que pour les moteurs, deux facteurs principaux ont contribuer à notre choix de batterie :

- Taille : On voulait une taille minimale pour le robot finale, et donc il nous fallait une batterie qui serait la plus petite possible. Tout en ayant une capacité acceptable.

- Disponibilité : Vu les prix très élevés des batteries, ont a du travailler avec ce qui était disponible au niveau du laboratoire.



FIGURE 4.1 – Batterie Li-Po utilisé

Au final, on a utiliser une batterie Li-Po, avec une capacité de 750mAh(2.7Wh). Cette batterie est parfaite en termes de dimensions et de capacité. Idéalement on aurait utilisé 2 batteries en série, mais du à l'indisponibilité, on a du se contenter d'une seule batterie, et utiliser un circuit élévateur de tension pour obtenir la tension dont on a besoin (détailé dans la section suivante).

Section 5

Régulation et stabilisation

5.1 Élévation 7V

Pour pouvoir obtenir la tension 7v dont on a besoin pour le fonctionnement de notre circuit, a partir des 3.7V obtenus de la batterie, on utilise un circuit convertisseur DC-DC élévateur (Boost converter). Et c'est un type d'alimentation de découplage qui convertie une tension continue d'entrée en une tension plus élevée en sortie.

5.1.1 Principe de fonctionnement

Le circuit de base de ce type de circuit contient au moins deux semis conducteurs, et au moins un composant qui peut stocker de l'énergie (un condensateur ou une bobine). Pour expliquer le principe de fonctionnement, on prend le circuit suivant comme exemple :

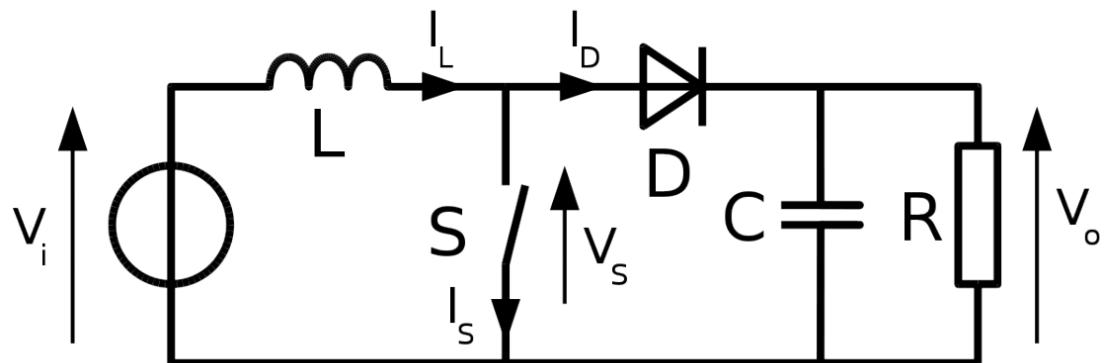


FIGURE 5.1 – Circuit Boost simple

Ce circuit fonctionne en deux phases :

1-Quand l'interrupteur S est fermé : la bobine alors se charge en se dotant d'un champ magnétique, qui lui permet de stocker de l'énergie. La diode reste bloquée pendant cette phase.

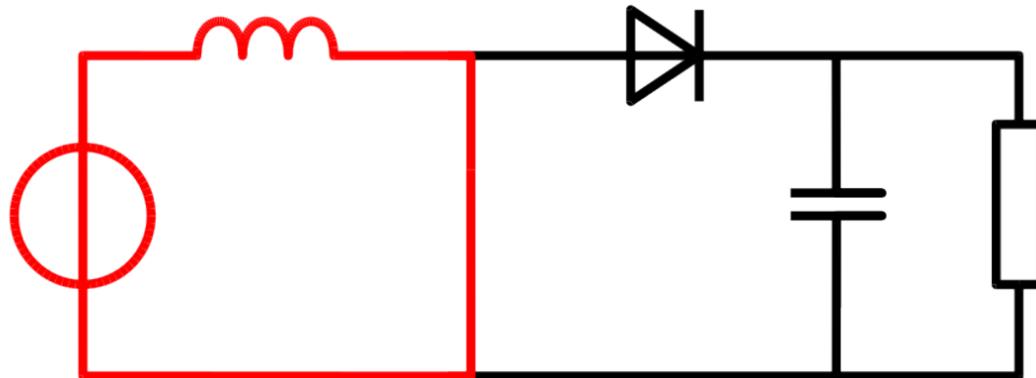


FIGURE 5.2 – Première Phase

2-Quand l'interrupteur S est ouvert : Les bobines ont tendance à vouloir garder un courant constant. Donc, quand l'interrupteur s'ouvre, le champ magnétique dans la bobine se transforme en une différence de potentiel. Cette tension, ajouté à la tension d'origine du générateur, va produire une tension supérieur à la tension d'entrée du générateur aux bornes de la capacité, qui va stocker et stabiliser cette tension, et qui sera ensuite utilisé par la charge en sortie.

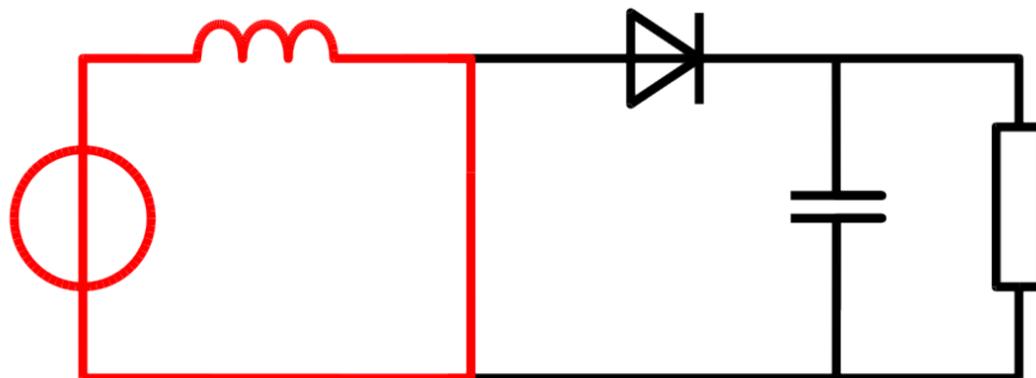


FIGURE 5.3 – Deuxième Phase

Ceci est le principe de fonctionnement de base de ce type de circuits.
Mais pour pouvoir obtenir un résultat utilisable dans des applications réels, Il nous fau-

dra un circuit bien plus compliqué que celui-ci. Dans un circuit réel, l'interrupteur est remplacé par un transistor, qui va jouer le même rôle.

Pour pouvoir contrôler la tension de sortie, on doit varier la fréquence de basculement de l'interrupteur (transistor), ainsi que le rapport ON/OFF (Duty Cycle).

Un autre problème se pose, pour la même fréquence de basculement et le même rapport ON/OFF, en changeant la tension d'entrée ou en changeant la charge en sortie, la tension va varier.

C'est pour cela qu'il faut un monitoring en continue de la sortie, et le basculement doit être en fonction de la tension de sortie pour essayer de la garder constante, peu importe la variation de l'entrée et de la charge en sortie.

La réalisation d'un tel circuit de façon manuelle sera trop compliqué à faire. Heureusement que des circuits intégrés prêts à cette application existent. Dans notre cas, le MT3608.

5.1.2 Circuit utilisé

Le MT3608 est un convertisseur élévateur à fréquence constante. Il fonctionne avec une fréquence de 12MHz. Ce circuit intégré contient aussi des protections de sous-tension et de courant. Son fonctionnement peut être résumé dans le schéma suivant :

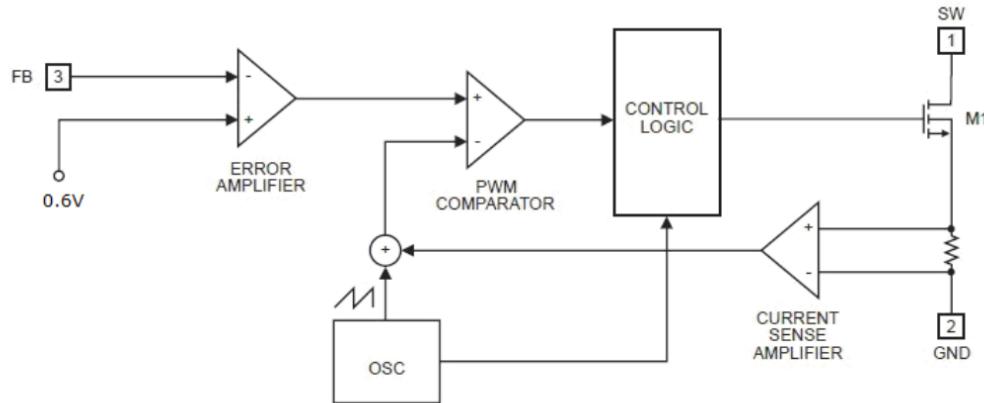


FIGURE 5.4 – Schéma MT3680

Ce composant est disponible seul comme composant, et la circuiterie qui va avec (Bobine, capacités, diode et potentiomètre pour pouvoir régler la sortie), n'est pas difficile à trouver ou à mettre en place.

Mais pour éviter une complexité pas nécessaire, on a préféré travailler avec un module contenant tout les composants qui permettent au circuit intégré de fonctionner, et nous permet aussi d'ajuster la sortie.



FIGURE 5.5 – Module MT3680

5.2 Régulation 5V

Pour pouvoir alimenter le microcontrôleur et les capteurs, on utilise un régulateur 5V, dans ce cas, le «7805». Il prend en entrée une tension entre 7V et 24V, avec un courant maximum 1.5A.

Pour que ce régulateur puisse marcher, il lui faut des condensateurs polarisés en entrée et en sortie. On utilise des capacités électrolytiques d'une valeur 10 μF .

On le branche suivant le circuit ci-dessous :

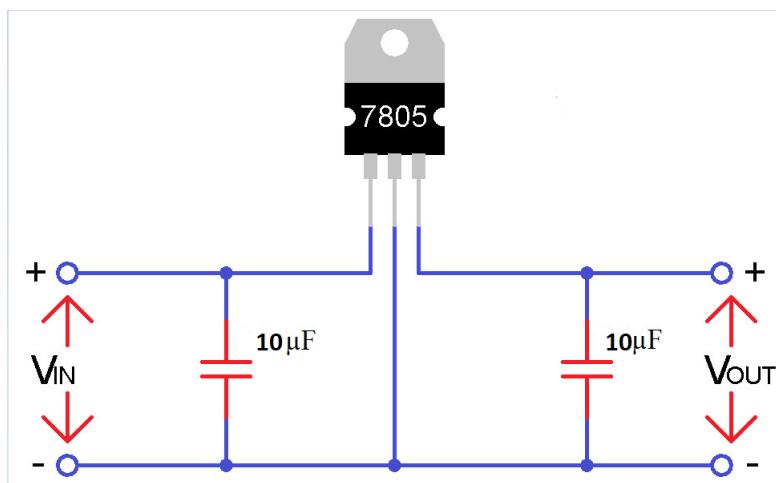


FIGURE 5.6 – Module MT3680

5.3 Alimentation et contrôle des moteurs

Pour contrôler un moteur DC, notamment pour contrôler la vitesse et la direction de mouvement, le circuit le plus utilisé est le «Pont en H». Ce circuit est principalement constitué de transistors (notamment des MOSFETs), qui agissent comme étant des interrupteurs. Le circuit de base de ce types de circuit comme suit :

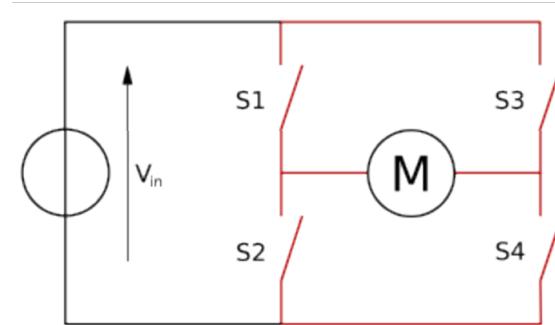


FIGURE 5.7 – Circuit de base pont en H

Plusieurs circuits intégrés et modules comportant ce circuit sont disponibles. Mais dû à sa taille très petite, courant nominale élevé, et protections disponibles, on utilise le circuit intégré «L293D».

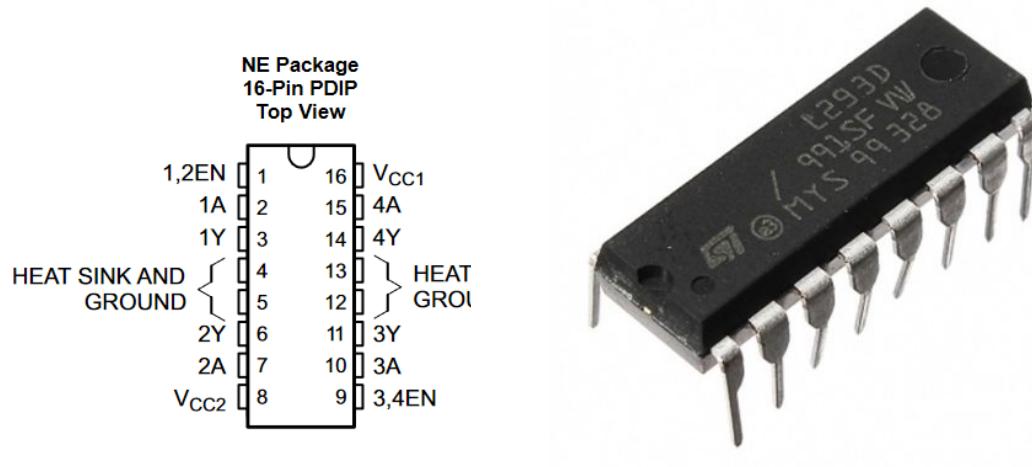


FIGURE 5.8 – L293D

Ce circuit est constitué de quatre demi pont en H, deux pour contrôler chaque moteur. Ça nous permet de contrôler deux moteurs au même temps, en direction et en vitesse.

Pour pouvoir contrôler les moteurs, on a 3 entrée pour chacun : 2 entrée pour contrôler la direction, et une seule pour la vitesse.

On plus, on a deux autres entrées pour l'alimentation : une entrée de 5V pour alimenter la circuiterie logique. Et une entrée qui prend une tension entre 5V et 24V.

Le reste des broches sont des broches «Ground».

Ce composant peut supporter un courant nominale de 600mA pour chaque moteur. Ce qui est largement suffisant pour nos moteurs.

5.4 Problèmes rencontrés

Après avoir implémenter nos circuits, et après les avoir tester, on a constaté qu'à chaque fois les moteurs essayent de démarrer ou de changer de direction, le microcontrôleur se redémarre, et donc le programme se relance depuis le début. Et donc c'était impossible de le faire marcher dans cette état.

Après un peu plus de testes, et à l'aide d'un oscilloscope, on a pu constater qu'à chaque fois les moteurs démarrent, on a une brève chute de tension, ce qui cause le redémarrage du microcontrôleur.

Pour résoudre ce problème, on a ajouté un condensateur d'une valeur de $100\mu F$ a la rail de 3.3V (Le microcontrôleur fonctionne avec 3.3V).

Chapitre IV

Capteur TCRT5000

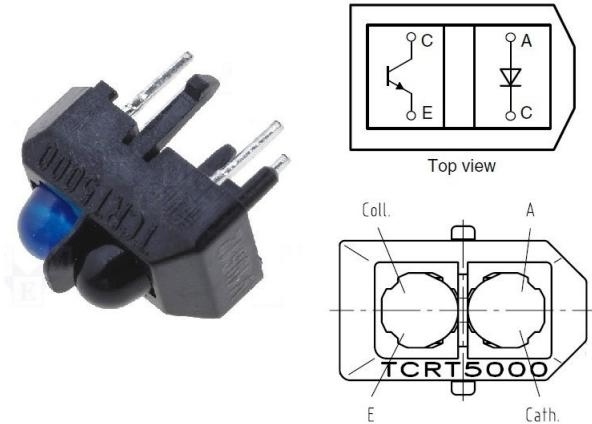


FIGURE 5.9 – Capteur TCRT5000

5.5 Présentation du capteur

Ce capteur est un capteur à réflexion qui comprend un émetteur infrarouge et une photodiode dans un emballage plombé qui bloque la lumière visible.

5.6 Fonctionnement

Un faisceau de rayonnement IR (Infra Rouge) est transmis à la cible et le faisceau réfléchi est capturé par la photodiode.

La diode IR ne mesure pas uniquement l'intensité de la lumière infrarouge, mais elle est également sensible à la lumière visible. L'intensité de la lumière infrarouge enregistrée par la photodiode représente la distance et la nature de la cible et le capteur. Cette méthode n'est pas très précise. Le rayonnement infrarouge du soleil, l'éclairage intérieur ou une quelconque source de chaleur constituent du bruit pour les capteurs , nous devons donc calibrer les capteurs à chaque fois que les conditions ambiantes changent.

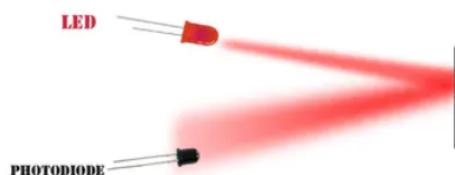


FIGURE 5.10 – Fonctionnement TCRT5000

5.6.1 Solution

Pour remédier à ce problème nous devons soustraire le signal du bruit du signal que l'on veut capter. Si nous allumons la LED, la photodiode mesure le bruit + le signal et si la LED s'éteint, la photodiode ne reçoit que du bruit. La différence entre ces deux valeurs donnera des données débruitées. Si nous prenons deux lectures très rapidement, le débruitage s'améliorera.

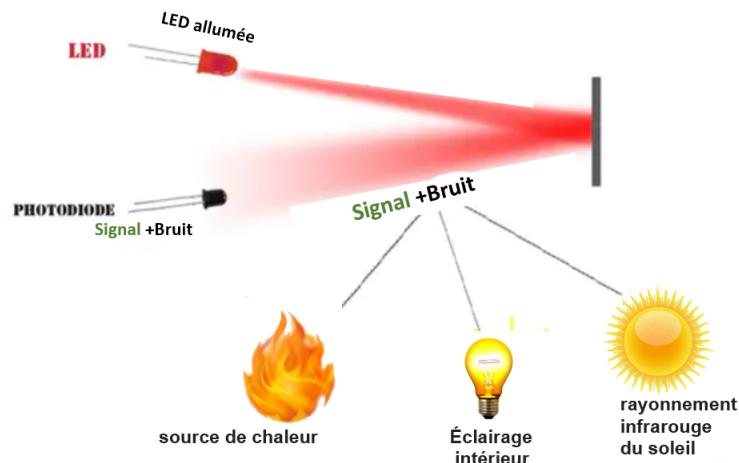


FIGURE 5.11 – Lecture du signal+Bruit

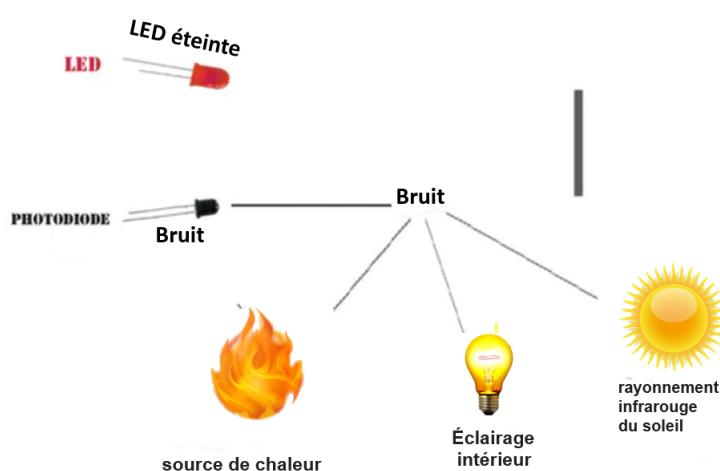


FIGURE 5.12 – Lecture du Bruit

Remarque : Cependant, cette soustraction des données de bruit ne donnera pas une sortie parfaitement précise car la relation entre la distance et la sortie du capteur n'est pas linéaire. Cependant, pour les applications de faible précision, cela devrait fonctionner de manière satisfaisante.

5.7 Circuit interne

L'émetteur et la cathode de la LED du capteur TCRT500 sont connectés au GND , l'anode de la LED est connecté à la résistance d'une valeur de $100\ \Omega$ et le collecteur est connecté à une résistance de $5K\Omega$.

Les deux résistances sont ensuite connectées au microcontrôleur. La résistance connecté à la diode est connecté à un pin digital et la résistance connecté au collecteur est connecté à un pin analogique.

le schéma suivant résume le circuit de chaque capteur.

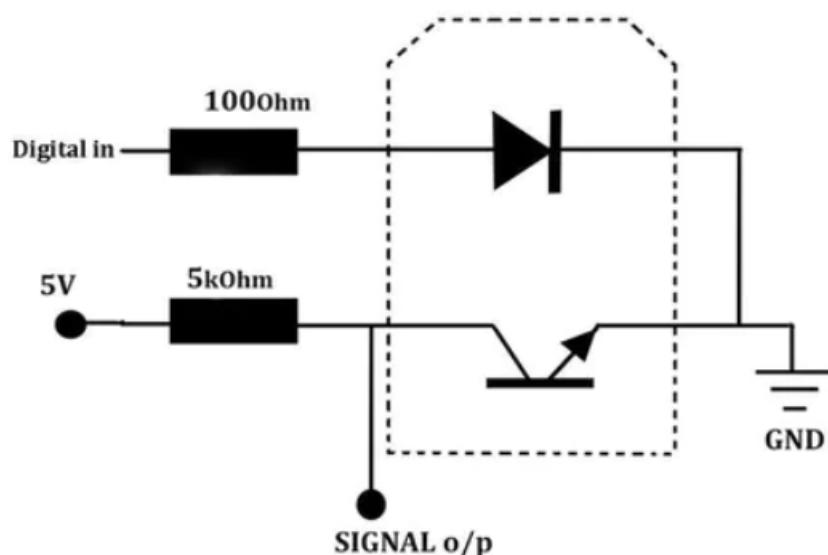


FIGURE 5.13 – Circuit interne du TCRT5000

Chaque capteur est connecté au microcontrôleur STM32 de la façon suivante :

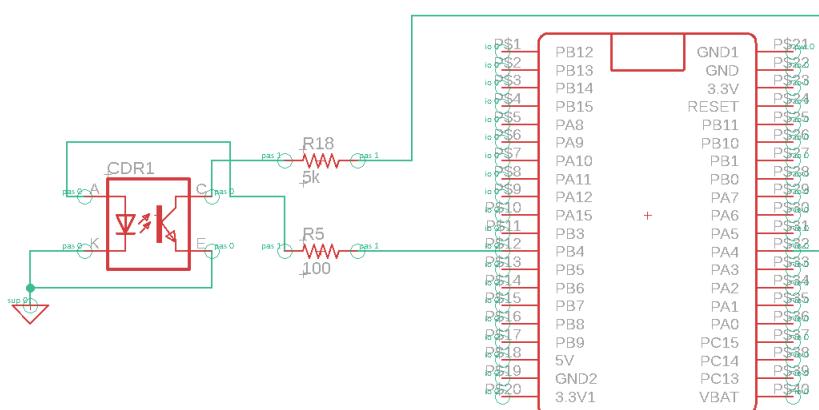


FIGURE 5.14 – Circuit du TCRT5000 avec le micro-contrôleur

5.8 Disposition sur le robot

Les capteurs ont été placés d'une manière à faciliter le contrôle du robot et la détection des différentes intersections.

L'espacement entre chaque capteur a été trouvé suite à de nombreux tests et suivant le cahier de charge fournit. L'emplacement des capteurs que l'on présente dans ce rapport est celui qui a donné les meilleurs résultats, un changement de la dimension de la ligne noire induira au changement de cet emplacement, ceci dans le cas où la ligne noire dépasse la limite de 3cm de largeur mais si la largeur reste inférieure à cette limite ; l'emplacement restera inchangé, mais il faudra comme même régler les constantes du PID (Plus de détails dans le chapitre de contrôle).

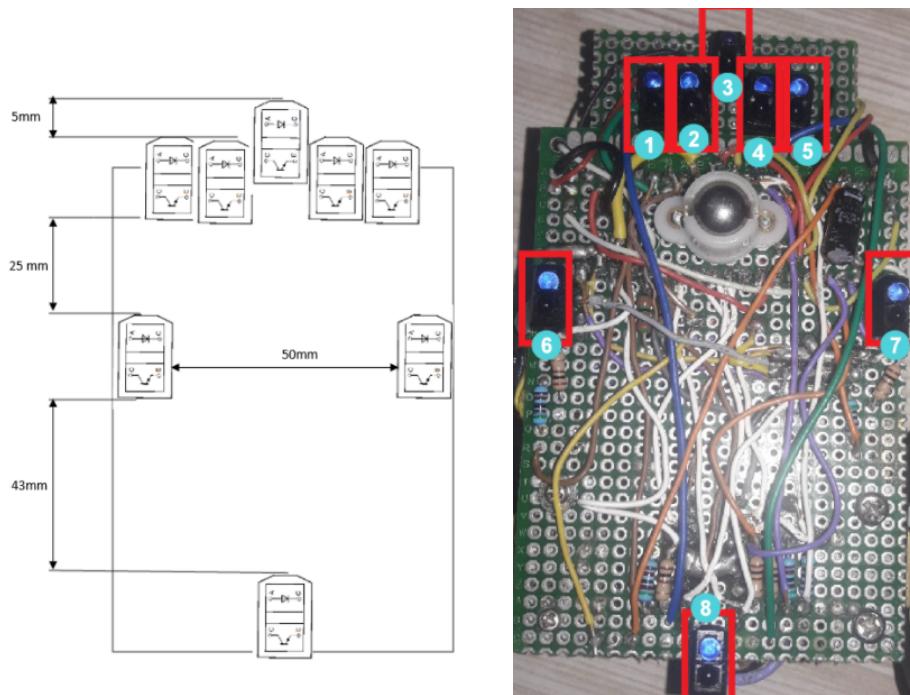


FIGURE 5.15 – Disposition des capteurs sur le robot

5.8.1 Capteurs Avant

Les capteurs 1,2,3,4,5 sont des capteurs utilisés pour détecter la ligne droite et ils sont aussi utilisés pour l'asservissement.

les capteurs 2,3,4 afficheront toujours une valeur de zéro ce qui équivaut à être sur la ligne noir.

Le capteur 3 est avancé de 5 mm des 4 autres capteurs est cela pour pouvoir détecter une discontinuité de la ligne droite avant que les autres capteurs ne l'atteignent, et ça nous permettra d'arrêter l'algorithme d'asservissement, qui autrement, serait perturbé par la discontinuité de la ligne.

Les capteurs 4 et 5 restent sur la frontière entre la ligne noir et la surface blanche. Et donc leurs valeurs changeront avec le changement de la position du robot. Ils seront utiliser pour le contrôle.

5.8.2 Capteurs d'intersection

Les deux capteurs 6 et 7 servent à détecter les intersection pour pouvoir ensuite appliquer l'algorithme de résolution, que l'on introduira plus tard dans le chapitre de résolution.

Remarque : Il faut s'assurer que ces capteurs soit colinéaires car si l'un d'entre eux est plus avancé que l'autre cela faussera la lecture, car le robot prendra les intersection de type T comme des intersection de type L ou L inverse.

Nous avons quatre type d'intersection :

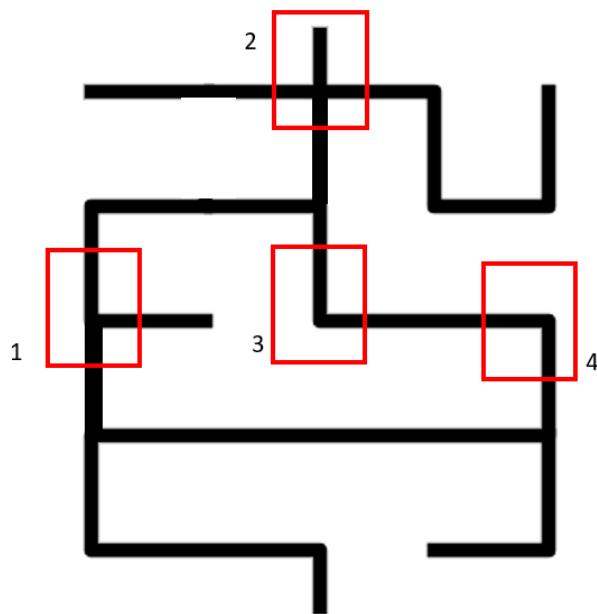


FIGURE 5.16 – Les types d'intersections

1)Intersection type "T"

Dans ce type d'intersection, les capteurs avant détectent la zone blanche, ce qui dit au robot que il n'y a pas de ligne droite, et les capteurs 7 et 6 détectent la ligne noir, et donc ça lui indique l'existence d'un tournant gauche et droit.

2)Intersection type "+"

Dans ce type d'intersection, les capteurs avant détectent une ligne noire, ce qui dit au robot que il n'y a une ligne droite, et les capteurs 7 et 6 détectent la ligne noir, et donc ça lui indique l'existence d'un tournant gauche et droit.

Remarque : Ce type d'intersection sera traiter différemment du type T si on utilise un algorithme qui considère le labyrinthe comme une matrice .

3) Intersection type "L" (Où L inverse)

Dans ce type d'intersection l'un des capteurs capte une valeur égale à '0' et l'autre capte une valeur de '1' , et les capteur avant capte tous une valeur égale à '1' dans ce cas le robot n'as pas d'autre choix que de tourner à droite ou à gauche et ce quelque soit l'algorithme utilisé.

5.8.3 Capteur Arrière

Nous avons utilisé qu'un seul capteur arrière est cela est largement suffisant, ce capteur servira dans la détection d'impasse.

Dans ce cas de figure tout les capteurs affichent une valeur différente de '0' , ce capteur sert alors à différencier le cas d'une intersection de type T avec une impasse , si l'on est dans le cas d'une intersection de type T ; les 6 autres capteurs capte '1' mais le capteur arrière capte une valeur qui est égale à zéro.

Mais dans le cas d'une impasse tous les capteurs captent une valeur égale à '1' .

5.9 Code

Tout d'abord nous devons définir les pins utilisés pour chaque capteur, pour ce projet nous avons décidé d'utiliser un STM32

la syntaxe est la suivante :

```
#define nom_variable P_nom_pin
```

On définit ainsi chaque pin pour les 8 capteurs

```
#define anaIRSensorm1 PA6
#define anaIRSensormleft1 PA5
#define anaIRSensormright1 PA4
#define digIRSensorm1 PA9
#define digIRSensormleft1 PA8
#define digIRSensormright1 PA10
#define anaIRSensorm2 PB0
#define anaIRSensormleft2 PA7
#define anaIRSensormright2 PA1
#define anaIRSensordirleft PB1
#define anaIRSensordirright PA0
#define digIRSensorm2 PB15
#define digIRSensormleft2 PB14
#define digIRSensormright2 PB8
#define digIRSensordirleft PB13
#define digIRSensordirright PB9
```

Nous définissons 8 fonctions ; une fonction pour chaque capteur que l'on utilisera tout au long de notre code, chaque fonction retournera le signal débruité, noté "c" dans le code qui suit :

```
int sensor()
{
    int a, c, b;
    digitalWrite(digIRSensorM1, HIGH); // on allume la LED
    delayMicroseconds(500);           //delay 500µs
    a = analogRead(anaIRSensorM1);   //Bruit+signal
    digitalWrite(digIRSensorM1, LOW); //on éteint la LED
    delayMicroseconds(500);          //delay de 500µs
    b = analogRead(anaIRSensorM1);   //Bruit
    c = b - a;                     //la différence:[ (Bruit+signal)-(bruit)] nous donne le signal débruité
    return c;
}
```

Remarque : Les fonctions diffèrent seulement dans le nom du signal qu'elle retourne et le nom de la fonction, mais elle repose sur le même principe introduit plus tôt dans le point du fonctionnement au début de ce chapitre.

```
> int sensorm1() ...
> int sensorml1() ...
> int sensormr1() ...
> int sensorm2() ...
> int sensorml2() ...
> int sensormr2() ...
> int sensordr() ...
> int sensordl() ...
```

Chapitre V

Contrôle

Section 6

Microcontrôleur

Pour le contrôle de notre robot de façon général, et pour exécuter les différents algorithmes et prendre les différentes dévisions, on a besoin d'une unité de contrôle principale, un microcontrôleur par exemple.

On a choisi d'utiliser le «STM32F103», qui est un microcontrôleur de la familles des «STM32» qui fonctionnent avec l'architecture «ARM».

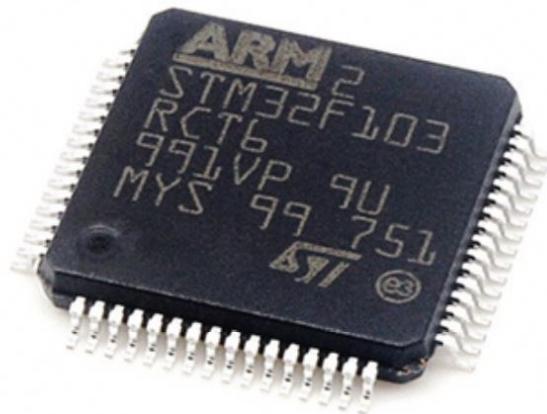
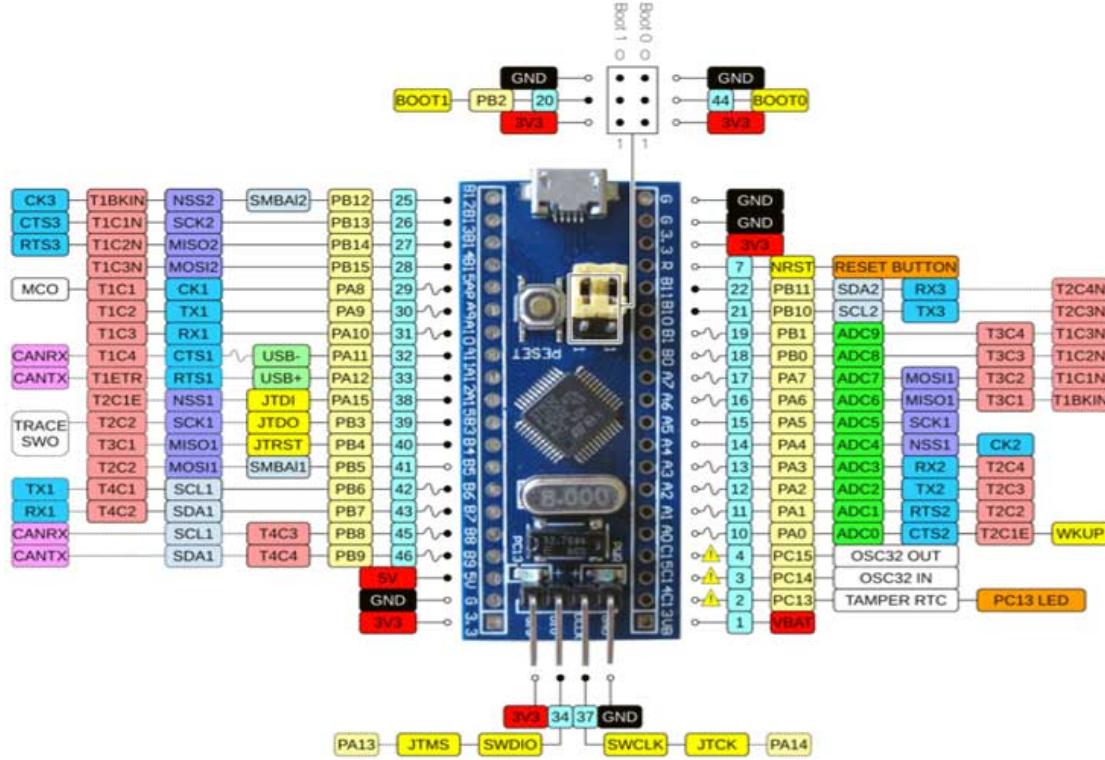


FIGURE 6.1 – Circuit Intégré STM32

Ce microcontrôleur utilise le processeur ARM Cortex M3, avec une fréquence de 72MHz, une RAM de 20Ko, et une mémoire de stockage de 128Ko.

Ce microcontrôleur est disponible sous plusieurs modules, on utilise le module populaire connue sous le nom «bluepill».

Il offre une grande variété d'entrées et sorties. Avec 37 broches d'entrée/sorties numériques qui supportent tous les interruptions. Parmi ces broches, 13 supportent les signaux PWM, et 10 qui supportent les signaux analogiques.



Section 7

Algorithme de contrôle suiveur de ligne

Nous avons vu dans le chapitre précédent, les capteurs utilisés, leurs emplacements et l'algorithme utilisé pour lire leur valeurs.

Maintenant on va voir la façon avec laquelle ces capteurs sont utilisé pour contrôler les mouvements du robot.

Pour que le robot puisse suivre une ligne droite (voir même un peu courbé), on doit trouver une façon pour le contrôler, et corriger son mouvement et sa trajectoire.

L'une des méthodes les plus efficaces et utilisées pour le contrôle, est la méthode connue sous le nom «PID» (Proportional, Integral, Derivative). Cette méthode vise à minimiser une erreur, en trouvant une équation linéaire qui calcule la correction à appliquer pour la trajectoire, qui est sous la forme :

$$D = K_p.P + K_d.D + K_i.I$$

Où :

P : C'est la valeur de l'erreur.

D : C'est la dérivée de l'erreur par rapport au temps.

I : C'est l'intégrale de l'erreur par rapport au temps.

Et K_p , K_d et K_i sont des coefficients à déterminer selon chaque système.

Pour pouvoir utiliser cette méthode, la première étape est de trouver une formule de calcul d'erreur, qui peut décrire la position du robot par rapport à la ligne.

La ligne recouvre les 3 capteurs centraux, et les capteurs aux bords sont en dessus du bord de la ligne, et donc n'importe quel mouvement va changer la valeur lu, vu qu'on utilise une lecture analogique. En prenant la différence entre le capteur gauche et le capteur droit, on obtient une formule d'erreur assez précise pour notre application.

On ajoute la différence entre les deux autres capteurs centraux, pour plus de précision. La formule finale de l'erreur p est donné par :

$$P = 3.cm{r1} + cm{r2} - 3.cm{l1} \sim cm{l2}$$

Cette valeur est également utilisée pour calculer la dérivée et l'intégrale, qui seront ensuite utilisés pour le calcul d'un taux de correction, qui est donnée avec la formule :

$$D = kp.P + kd.D + ki$$

Les coefficients Kp, Kd et Ki sont déterminés de façon empirique.

Ce taux de correction sera directement traduit en différence de vitesse entre les deux roues.

Le code de contrôle des moteurs sera donnée par :

```
255 | I = I + p;
256 | if (millis - old_millis > 1)
257 | {
258 |
259 |     d = (old_d - d);
260 |     o = (kp * p + kd * d + ki * I);
261 |
262 |     old_d = d;
263 |     old_millis = millis;
264 |     I = 0;
265 |
266 |     if (speeed - o < 0)
267 |     {
268 |         analogWrite(enA, 0);
269 |     }
270 |     else if (speeed - o > 255)
271 |     {
272 |         analogWrite(enA, 255);
273 |     }
274 |     else
275 |     {
276 |         analogWrite(enA, speeed - o);
277 |     }
278 |     if (speeed + o < 0)
279 |     {
280 |         analogWrite(enAA, 0);
281 |     }
282 |     else if (speeed + o > 255)
283 |     {
284 |         analogWrite(enAA, 255);
285 |     }
286 |     else
287 |     {
288 |         analogWrite(enAA, speeed + o);
289 |     }}
```

FIGURE 7.1 – Algorithme PID

Section 8

Fonctions de mouvement

A l'addition de la fonction de mouvement de base, ce qui permet de suivre une ligne droite. On a besoin de fonctions qui vont nous permettre d'effectuer des rotations et de suivre des virages de 90°.

8.1 Turn Right

On définit une fonction pour effectuer un tour à droite. Pour tourner à droite, la roue gauche doit tourner plus vite que la roue droite. Dans notre cas, pour pouvoir tourner sans avancer, la roue droite reste à l'arrêt et la roue gauche tourne à une vitesse fixe. De cette manière, on peut tourner mais on a besoin d'une façon pour le contrôler et pour savoir quand s'arrêter, c'est pour cela qu'on utilise nos capteurs. Quand la commande de tourner à droite est reçue, on peut être dans l'une de ces intersections :

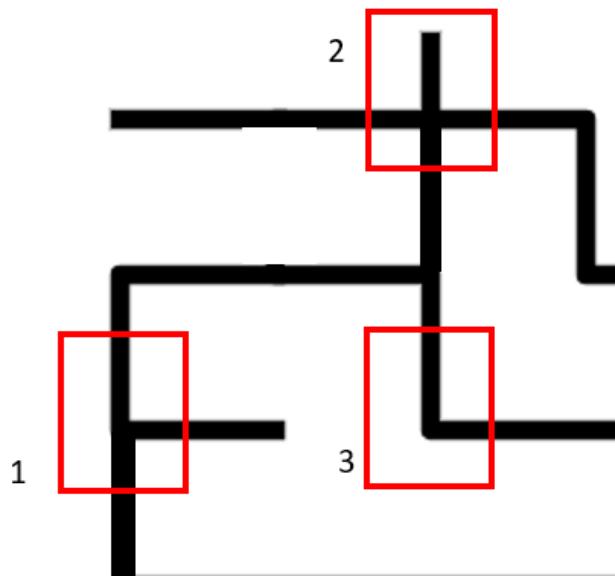


FIGURE 8.1 – Types Intersections

L'algorithme pour tourner se compose de 4 parties principales :

1. On initie le mouvement du moteur gauche à une vitesse prédéterminée (On détermine la vitesse max qui reste stable).
2. On attend que tous les capteurs avant soient dans le blanc.
3. Une fois que tous les capteurs sont en dessus de la zone blanche, on attend que le capteur centrale soit de nouveau en dessus de la ligne noir. On initie ensuite une séquence pour arrêter le mouvement de rotation. (On fait marcher le moteur gauche dans le sens opposé pour un instant).
4. On lance une boucle de mouvement en avant, pour redresser la position du robot, avant de sortir de la fonction. Ceci pour éviter de fausses lectures sur les autres capteurs qui risquent de lancer d'autres séquences.

Le code détaillé est donné ci-dessous(expliqué avec des commentaires) :

```
81 void right()
82 {
83     //Lecture des capteurs:
84     cmr1 = sensormr1();
85     cmr2 = sensormr2();
86     cml2 = sensorml2();
87     cm1 = sensorm1();
88     //Initialisation des mouvements des moteurs
89     analogWrite(enA, 55);
90     analogWrite(enAA, 0);
91     digitalWrite(int1, HIGH);
92     digitalWrite(in2, LOW);
93     analogWrite(enAA, 200);
94     delay(80);
95     digitalWrite(int1, LOW);
96     digitalWrite(in2, HIGH);
97     analogWrite(enAA, 0);
```

FIGURE 8.2 – Première partie

```
98 //Boucle pour attendre que tous les capteurs avant soient en zone blanche
99 while (cm1 == 0 || cmr2 == 0 || cml2 == 0 || cmr1 == 0 || cml1 == 0)
100 {
101     delay(5);
102     cmr1 = sensormr1();
103     cml1 = sensorml1();
104
105     cmr2 = sensormr2();
106     cml2 = sensorml2();
107     cm1 = sensorm1();
108 }
```

FIGURE 8.3 – Deuxième partie

```

109 //On attend que le capteur centrale soit de nouveau sur la ligne noir
110 while (cmr1 != 0)
111 {
112     delay(1);
113     cmr1 = sensormr1();
114 }
115 //On lance le moteur gauche en inverse pour un breve instant pour arreter la rotation
116 digitalWrite(in1, HIGH);
117 digitalWrite(in2, LOW);
118 analogWrite(enA, 240);
119 delay(30);
120 analogWrite(enA, 0);
121 digitalWrite(in1, LOW);
122 digitalWrite(in2, HIGH);
123 digitalWrite(in11, LOW);
124 digitalWrite(in22, HIGH);

```

FIGURE 8.4 – Troisième partie

```

128 //On lance un mouvement en avant jusqu'a ce que le robot soit en bonne orientation:
129 while (cdl == 0 || cdr == 0 || cmr1 != 0 || cml1 != 0 || cm1 != 0 || cm2 != 0)
130 {
131     cdr = sensordr();
132     cdl = sensordl();
133     cml1 = sensorml1();
134     cml2 = sensorml2();
135     cm1 = sensorm1();
136     cm2 = sensorm2();
137     cmr1 = sensormr1();
138     cmr2 = sensormr2();
139     forward(speeed);
140     delayMicroseconds(10);
141 }
142 }

```

FIGURE 8.5 – Quatrième partie

8.2 Turn Left

On refait le même algorithme que tourner à droite, on inverse seulement les capteurs et les mouvements des roues .

8.3 Rotation(Demi-Tour)

On a besoin d'avoir une fonction qui effectue un demi-tour.
Cette fonction est divisé en 4 parties :

1. On initialise le mouvement des moteurs, les moteurs tournent avec une vitesse fixe dans des sens opposés.
2. Au début de la séquence de rotation, tous les capteurs sont en zone blanche, on continue donc le mouvement de rotation jusqu'à ce que le capteur centrale soit de nouveau sur la ligne noir.

3. On initie une petite séquence pour arrêter la rotation (en tournant les moteurs dans le sens opposé pour un bref moment).

4. On lance une séquence de mouvement droit, jusqu'à ce que le robot soit de nouveau en bonne orientation.

```

30 void rotation()
31 {
32   cm1 = sensorm1();
33   cmr1 = sensormr1();
34   cml1 = sensorml1();
35   digitalWrite(in1, HIGH);
36   digitalWrite(in2, LOW);
37   analogWrite(enAA, 45);
38   analogWrite(enA, 45);
39   while (cml2 != 0)
40   {
41     delay(1);
42
43     cml2 = sensorml2();
44   }
45   digitalWrite(in1, LOW);
46   digitalWrite(in2, HIGH);
47   digitalWrite(in11, HIGH);
48   digitalWrite(in22, LOW);
49   analogWrite(enAA, 200);
50   analogWrite(enA, 200);
51   delay(15);
52   analogWrite(enA, 0);
53   analogWrite(enAA, 0);
54   digitalWrite(in1, LOW);
55   digitalWrite[in2, HIGH];
56   digitalWrite(in11, LOW);
57   digitalWrite(in22, HIGH);
58 }
```

FIGURE 8.6 – Rotation Première Partie

```

59   cdr = sensordr();
60   cdl = sensordl();
61   cml1 = sensorml1();
62   cml2 = sensorml2();
63   cm1 = sensorm1();
64   cm2 = sensorm2();
65   cmr1 = sensormr1();
66   cmr2 = sensormr2();
67   while (cdl == 0 || cdr == 0 || cmr1 != 0 || cml1 != 0 || cm1 != 0 || cm2 != 0)
68   {
69     cdr = sensordr();
70     cdl = sensordl();
71     cml1 = sensorml1();
72     cml2 = sensorml2();
73     cm1 = sensorm1();
74     cm2 = sensorm2();
75     cmr1 = sensormr1();
76     cmr2 = sensormr2();
77     forward(speeed);
78     delayMicroseconds(10);
79   }
80 }
```

FIGURE 8.7 – Rotation Deuxième Partie

8.4 Stop

Pour pouvoir arrêter le robot rapidement, on utilise la fonction stop, qui fait tourner les moteurs dans le sens opposé pour une durée très courte. Cette durée est déterminée de façon empirique.

```
202 void stoop()
203 {
204     spd = speeed;
205     digitalWrite(in1, HIGH);
206     digitalWrite(in2, LOW);
207
208     digitalWrite(in1, HIGH);
209     digitalWrite(in2, LOW);
210     analogWrite(enAA, 255);
211     analogWrite(enA, 255);
212     delay(speeед / 2);
213     analogWrite(enAA, 0);
214     analogWrite(enA, 0);
215     digitalWrite(in1, LOW);
216     digitalWrite(in2, HIGH);
217
218     digitalWrite(in1, LOW);
219     digitalWrite(in2, HIGH);
220 }
```

FIGURE 8.8 – Algorithme D'arrêt

Chapitre VI

Algorithme

Section 9

Labyrinthe

Il faut savoir que dans le cadre de la compétition POLYMAZE, le labyrinthe avait les caractéristiques suivantes :

1. La hauteur des murs est de 15cm.
2. La distance entre un mur et son opposé est de 30cm.
3. Une ligne noire de 2cm d'épaisseur est tracée au long des pistes au milieu de deux murs opposés.
4. Le labyrinthe est en bois (parterre et murs), le parterre est blanc.
5. La fin du labyrinthe est marquée par un carré noir de 20cm*20cm.

Mais pour le cas d'un robot suiveur de ligne les murs sont inexistant donc les seules caractéristique qu'on prendra en compte sont la 2^{eme}, 4^{eme} et la 5^{eme}.

La figure de droite représente un exemple de labyrinthe de [taille 5x5] et celle de gauche est le même labyrinthe vu dans le cas d'un suiveurs de ligne :

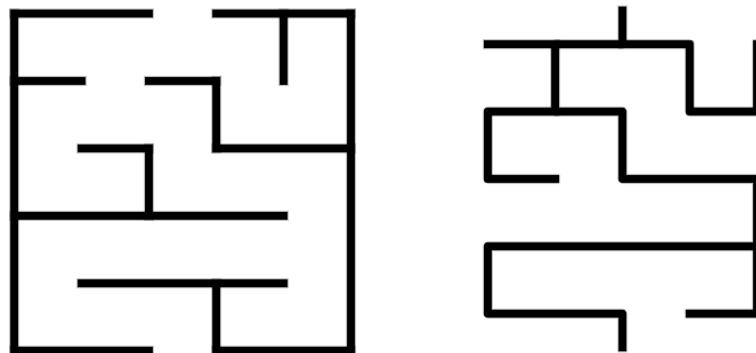


FIGURE 9.1 – Labyrinthe [taille5x5]

Section 10

Structure du code

Le code peut être diviser en quatre grandes parties ,une première partie dédié aux variables , une deuxième pour les fonctions (qui nous facilitera ensuite la tache dans la partie main("loop") du code), une partie "Setup" et une autre partie "loop". Chaque partie sera expliqué en détail dans ce chapitre.

La figure suivante donne un aperçu du contenu de chaque partie du code :



FIGURE 10.1 – Structure du code

10.1 Les Fonctions

10.1.1 Règle de la main gauche [left hand|LH]

La règle la plus connue pour traverser les labyrinthes est le suiveur de mur, également connu sous le nom de règle de gauche ou de règle de droite. Si le labyrinthe est simplement

connecté, c'est-à-dire que tous ses murs sont connectés entre eux ou à la limite extérieure du labyrinthe, alors en gardant une main en contact avec un mur du labyrinthe, le solveur est garanti de ne pas se perdre et atteindra une sortie différente. Si il y en a une ; sinon, l'algorithme retournera à l'entrée après avoir traversé chaque couloir à côté de cette section de murs connectée au moins une fois. L'algorithme est un parcours d'arborescence dans l'ordre en profondeur.

Définition

Cette fonction est définie ainsi :

```
void LH()
```

Principe

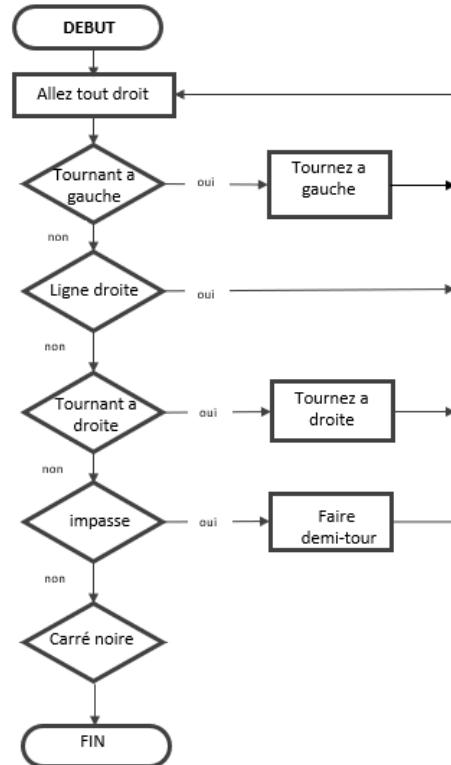


FIGURE 10.2 – Algorigramme de la fonction LH

Pseudo-code

Tant que les capteurs ne capte pas tous la même valeur qui est de zéro
si cdl==0 and cmr2 ou cml2!=0
 relire la valeur de cdl grâce a la fonction sensrdl apres un delay(2) si cdl==0
 Tourner a gauche[fonction left()] et rajouter l'élément 3 au tableau des mouvement
effectuer=path
 Sinon
 si cm1 ou cml1 ou cmr1 ==0 Si cdr==0
 delay2 puis relire les valeur de cdr et cdl[pour confirmer]
 si cdr ==0 et cdl!=0 et (cmr2 ou cml2!=0)
 Allez tout droit[fonction Forward()] et rajouter l'élément 1 au tableau des mouvement
effectuer=path
 Tant que cdr==0 allez tout droit [fonction forward()]
 Sinon
 si cdr ==0 et cmr2ou cml2!=0
 Delay de 10 relire les valeurs de cdr et cdl [confirmation]
 si cdl!=0 et cdr==0
 allez a droite[fonction forward()] et rajouter l'élément 2 au tableau des mouvement
effectuer=path
 si cm2!=0 et cm1==0
 Effectuer une rotation [fonction rotation()] et rajouter l'élément 0 au tableau des mouvement
effectuer=path
 Sinon le Robot s'arrête

Cet algorithme de la main gauche sur le mur peut être simplifié comme suit :
Si vous pouvez tourner à gauche, tournez à gauche
Sinon, si vous pouvez continuer tout droit, continuez tout droit
Sinon, si vous pouvez tourner à droite, tournez à droite
Si vous êtes dans une impasse, faites demi-tour
Pour nous faciliter l'existence et tout au long de ce document les mouvements seront
remplacé par des chiffres comme suit
Rotation (impasse)=0
Continuer tout droit=1
Tourner a droite= 2
Tourner a gauche= 3

mouvement	code
Rotation (impasse)	0
Continuer tout droit	1
Tourner à droite	2
Tourner à gauche	3

TABLE 10.1 – Tableau récapitulatif des mouvements et de leurs code

10.1.2 Règle de la main main droite [right hand|RH]

Cette fonction suit le même principe que la fonction left hand

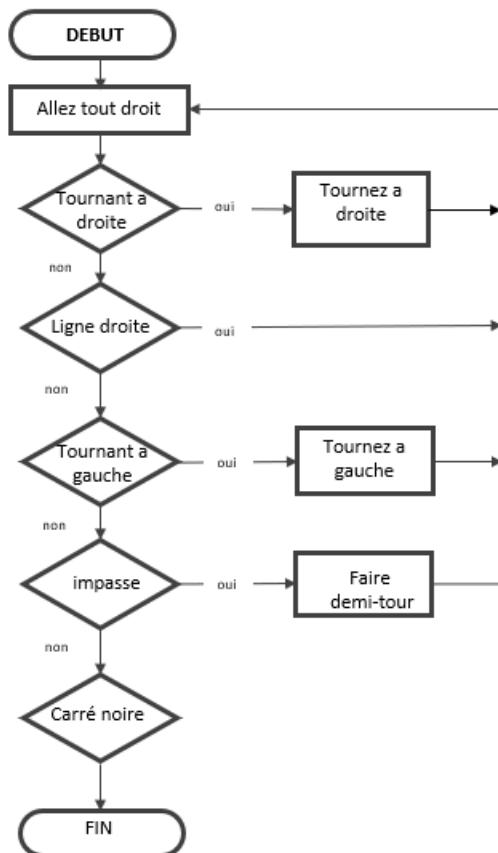


FIGURE 10.3 – Algorigramme de la fonction RH

Définition

Cette fonction est définie ainsi :

```
void RH()
```

10.1.3 Short-path

Cette fonction a pour but de donner les différents mouvements pour atteindre la sortie du labyrinthe, tout en supprimant tout les cas d'impasse, elle ne donne pas nécessairement la solution optimale mais nous donne une solution qui prendra beaucoup moins de temps qu'un passage simple en suivant la règle de la main droite ou la règle de la main gauche, mais dans le cas de labyrinthes simples (Pas d'îles de murs), ça donnera le chemin le plus court. Pour des tailles de labyrinthe inférieur à 20x20 la solution donnée par cette fonction est assez satisfaisante et est presque la même solution si l'on avait appliqué un algorithme de recherche comme Djikstra ou bien A*

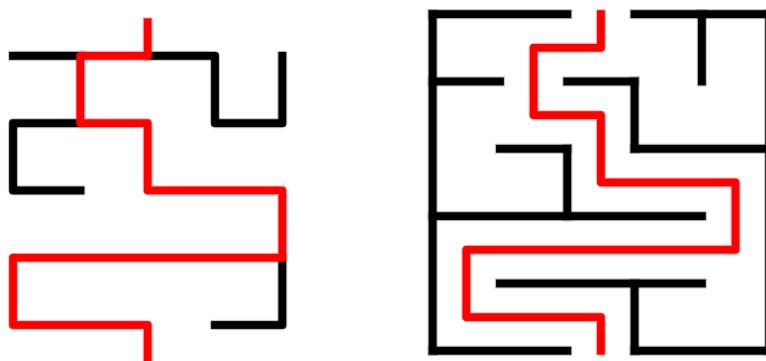


FIGURE 10.4 – Chemin le plus court d'un labyrinthe [taille5x5]

Définition

Cette fonction est définie comme suit :

```
void short_path(int v[], int tiime[], int taille);
```

v[] : est un tableau qui contient les différents mouvements

tiime[] : est un tableau qui contient le temps entre chaque “node”¹

taille : est la taille du tableau v[]

1. node=noeud dans notre cas cela correspond à une intersection

Principe

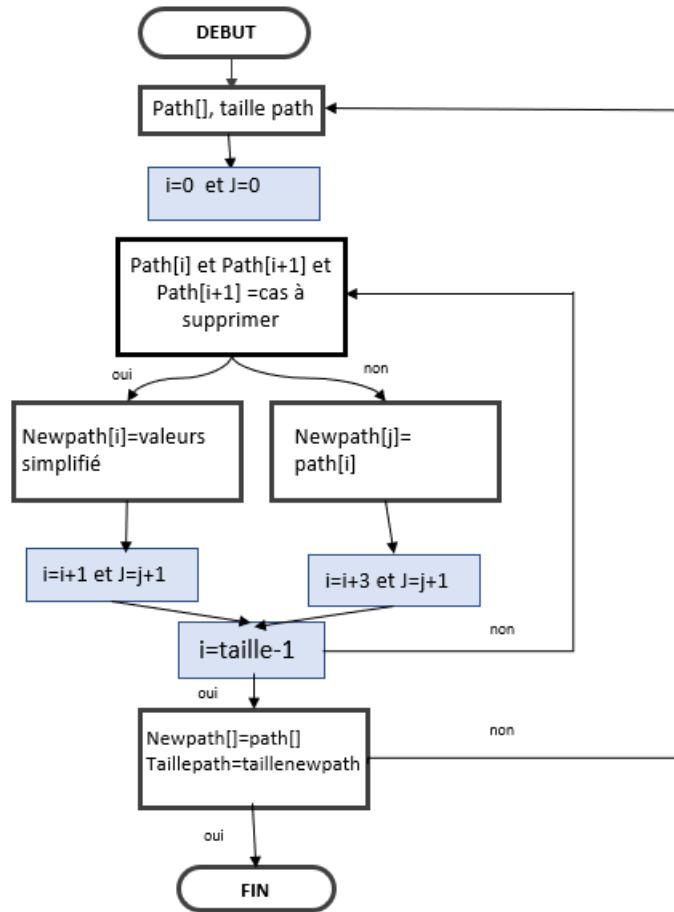


FIGURE 10.5 – Algorigramme de la fonction $\text{short}_{\text{path}}$

Pseudo-code

définir un tableau path[] de taille (taille)
 définir un tableau time[] de taille (taille)
 définir deux indice j et m initialisés à zero

pour un indice i allant de 0 à taille-1

Si v[i] et v[i+1] et v[i+2] est l'un des cas à supprimé ajouté l'élément de substitution au tableau path

time[i] reçoit timee[i]

on incremente i=i+2 et j=j+1

Sinon path[j] reçoit l'élément v[i] et time[j] reçoit timee[i]

pour i allant de 0 à taille-1

Si un élément de path[i] est différent de v[i]

Alors m=4 ;

Si m=0

on obtient path[] et time[] simplifié

Sinon

short path(path,time,taille)

[récursivité avec comme condition d'arrêt : l'ancien path sera égale au nouveau path générée par les fonction , il n' y aura donc plus de simplification a effectué]

Toute les simplification possibles pour le cas de l'algorithme de la main gauche et celui de la main droite sont présenté dans ce tableau

mouvement	simplification
101	0
102	3
103	2
201	3
202	1
203	0
301	2
302	0
303	1

TABLE 10.2 – Tableau récapitulatif des mouvements et de leurs simplifications

10.1.4 Translate

Définition

Cette fonction est définie comme suit :

```
void translate()
```

On définit deux fonctions similaires, une pour traduire le chemin résultant de la fonction de la main droite et une autre résultant de la fonction de la main gauche :

```
> void translatel() ...
> void translater() ...
```

Principe

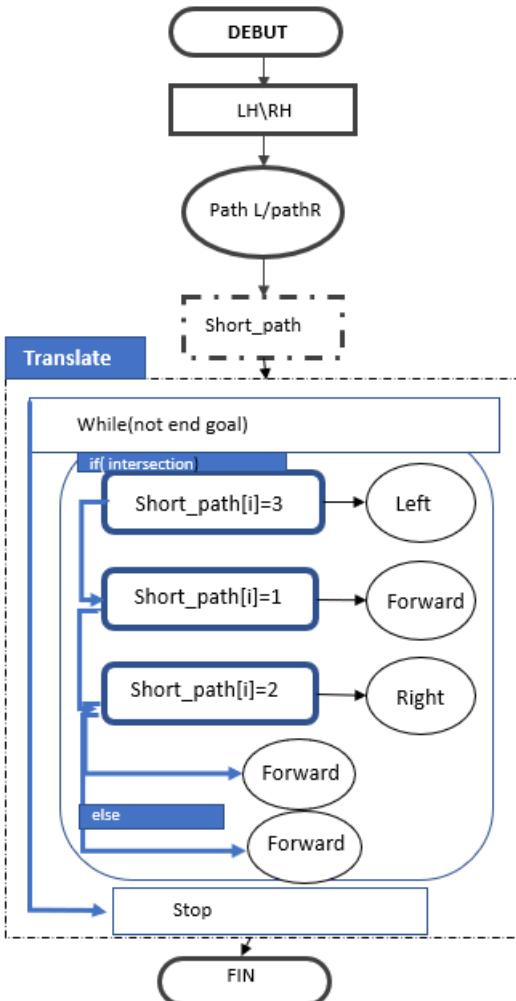


FIGURE 10.6 – Algorigramme de la fonction translate

Pseudo-code

Crée une boucle while avec comme condition tout les signaux C capté par les capteur(retourner par la fonction sensor) différent de zéro.

Tant que cette condition est satisfaire faire ce qui suit : Si cdr ou cdl ==0 et cmr2 ou cml2 sont !=0

-Delay 2 puis relire la valeur de cdr et cdl [cette condition est ajoutée pour confirmer la valeur de cdr et cdl]

Si cdr ou cdl ==0 [on a eu une deuxième confirmation]

Si l'élément i dans le tableau du chemin raccourci[produit par short-path]== 3
gauche())[fonction left]

sinon si il est égale a 1

While cdr et cdl ==0

Tout droit())[fonction forward]

Sinon si il est égale a 2

droite())[fonction right]

Sinon Allez tout droit

Si on arrive a la fin du labyrinthe donc tous les capteurs captent zéro alors :

Le robot s'arrête [fonction stoop]

10.1.5 Increment

Attachinterrupt()

Les interruptions sont utiles pour faire en sorte qu'une certaine partie du code se produise automatiquement dans les programmes de microcontrôleur et peuvent aider à résoudre les problèmes de synchronisation. Les bonnes tâches pour l'utilisation d'une interruption peuvent inclure la lecture d'un encodeur rotatif ou la surveillance d'une entrée utilisateur.

À propos des routines de service d'interruption

Les RSI sont des types spéciaux de fonctions qui ont des limitations uniques que la plupart des autres fonctions n'ont pas. Une RSI ne peut pas avoir de paramètres, et elles ne devraient rien retourner.

En règle générale, une RSI doit être aussi courte et rapide que possible. Si votre esquisse utilise plusieurs RSI , une seule peut s'exécuter à la fois, les autres interruptions seront exécutées après la fin de l'interruption en cours dans un ordre qui dépend de la priorité dont elles disposent. Généralement, les variables globales sont utilisées pour transmettre des données entre une RSI et le programme principal. Pour vous assurer que les variables partagées entre une RSI et le programme principal sont correctement mises à jour, déclarez-les comme volatiles.

syntaxe

```
attachInterrupt(digitalPinToInterrupt(pin), RSI, mode)
```

interrupt : le numéro de l'interruption. Types de données autorisés : int.

pin : le numéro de broche du microcontrôleur.

RSI : l'RSI à appeler lorsque l'interruption se produit ; cette fonction ne doit prendre aucun paramètre et ne rien renvoyer. Cette fonction est parfois appelée routine de service d'interruption.

mode : définit quand l'interruption doit être déclenchée.

Quatre constantes sont prédéfinies comme valeurs valides :

mode	explication
LOW	pour déclencher l'interruption chaque fois que le pin est au niveau bas b.
CHANGER	pour déclencher l'interruption chaque fois que le pin change de valeur
RISING	pour déclencher l'interruption lorsque le pin passe d'un niveau bas a un niveau haut.
FALLING	pour déclencher l'interruption quand le pin passe d'un niveau haut a un niveau bas.
HIGH	pour déclencher l'interruption a chaque fois que le pin est au niveau haut.

TABLE 10.3 – Tableau récapitulatif des modes et de leurs explications

Definition

```
void increment()
```

Principe

On définit la variable mood, la fonction toute seule ne fait rien d'autre que d'incrémenter la valeur de mood mais cette dernière trouve tout son sens lors de l'utilisation de attachinterrupt() et durant la fonction LOOP . Car c'est grâce à cette dernière que l'on peut utiliser les différents "mode" en d'autre terme choisir qu'elle partie de notre code nous voulons exécuter.

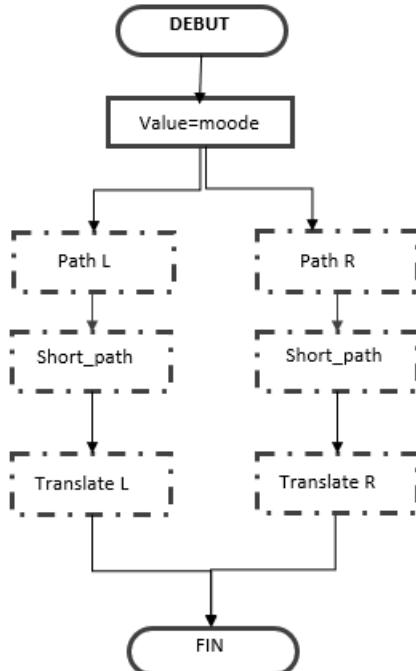


FIGURE 10.7 – Algorigramme de l'utilisation de increment dans le code final

10.2 Partie setup

Définition

La fonction `setup()` est appelée au démarrage du programme. Cette fonction est utilisée pour initialiser les variables, le sens des broches, les librairies utilisées. La fonction `setup` n'est exécutée qu'une seule fois, après chaque mise sous tension ou reset (réinitialisation) de la carte Arduino.

Code

```
void setup()
{
    pinMode(push_button, INPUT_PULLDOWN);
    attachInterrupt(digitalPinToInterrupt(PA2), increment, FALLING);
    pinMode(anaIRSensorM1, INPUT);
    pinMode(anaIRSensorMleft1, INPUT);
    pinMode(anaIRSensorMright1, INPUT);
    pinMode(anaIRSensorDirLeft, INPUT);
    pinMode(anaIRSensorDirRight, INPUT);
    pinMode(digIRSensorM1, OUTPUT);
    pinMode(digIRSensorMleft1, OUTPUT);
    pinMode(digIRSensorMright1, OUTPUT);
    pinMode(digIRSensorDirLeft, OUTPUT);
    pinMode(digIRSensorDirRight, OUTPUT);
    pinMode(anaIRSensorM2, INPUT);
    pinMode(anaIRSensorMleft2, INPUT);
    pinMode(anaIRSensorMright2, INPUT);
    pinMode(digIRSensorM2, OUTPUT);
    pinMode(digIRSensorMleft2, OUTPUT);
    pinMode(digIRSensorMright2, OUTPUT);
    pinMode(in1, OUTPUT);
    pinMode(in2, OUTPUT);
    pinMode(enA, OUTPUT);
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    pinMode(in11, OUTPUT);
    pinMode(in22, OUTPUT);
    pinMode(enAA, OUTPUT);
    digitalWrite(enA, LOW);
    digitalWrite(enAA, LOW);
    digitalWrite(in11, LOW);
    digitalWrite(in22, LOW);
    delay(2000);
    digitalWrite(in1, LOW);
    digitalWrite(in2, HIGH);
    digitalWrite(in11, LOW);
    digitalWrite(in22, HIGH);
```

10.3 Partie loop

Définition

Après avoir créé une fonction `setup()`, qui initialise et fixe les valeurs de démarrage du programme, la fonction `loop ()` (boucle en anglais) fait exactement ce que son nom suggère et s'exécute en boucle sans fin, permettant à votre programme de s'exécuter et de répondre.

Principe

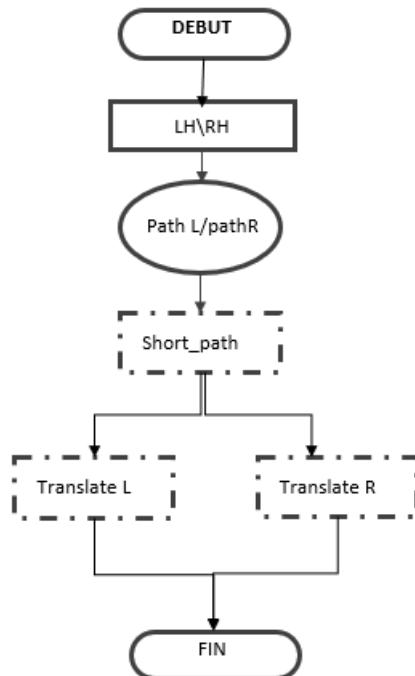


FIGURE 10.8 – Algorigramme de la fonction LOOP

Pseudo-code

Pour simplifier ce qui se passe dans la fonction `loop` nous allons la résumer dans les points suivants :

D'après la valeur de `mode` [variable de la fonction `increment`] :

-Le robot effectue une routine suivant la fonction `LH` ou la fonction `RH`

Remarque : Dans ce qui suit nous allons parler de la fonction `LH` seulement , autrement dit un passage du robot dans le labyrinthe suivant la règle de la main gauche , mais la fonction `RH` suit le même principe .

- A chaque fois qu'il effectue un mouvement se mouvement est enregistré dans un tableau nommer path[chemin en anglais] , dans ce cas arrivé a la fin du labyrinthe on aura un tableau qui contient tout les mouvement effectuer lors de ce premier passage.

-Ce chemin est ensuite simplifié grâce a la fonction short-path

-Puis grâce a la fonction increment qui change la valeur de moode lorsqu'on clique sur le push-botton [boutton poussoir en anglais], le robot suivra un chemin dicté par la fonction translate cette dernier traduira le chemin raccourci en des mouvements .

Suivant le cahier de charge de la compétition POLYMZE nous avons le droit durant la première phase de WARM-UP d'effectuer autant de passage que l'on souhaite ,c'est pour cela qu'on a décidé de suivre les étapes suivante

1. Un premier passage grâce a la fonction LH
2. Un deuxième passage grâce a la fonction RH
3. La fonction translate compare le temps prit non pas par les deux tableau de chemin résultant mais des deux tableau de chemin raccourcis celui qui aura prit le moins de temps sera choisit pour effectuer le dernier passage durant la phase du SPRINT

Chapitre VII

Conclusion

Pour conclure , ce stage a était très enrichissant pour nous. Nous avons apprit de nos erreurs , proposer de nouvelles alternatives a chaque fois que nous rencontrions un problème, maîtriser un peu plus la programmation en c++ , la modélisation sous SOLIDWORK, l'impression 3D, la conceptions de PCB sous EAGLE et bien d'autre compétences.

Nous somme très fière d'être arriver a ce résultat , il y'as certe des modifications a apporter a cette dernière version [on peut toujours mieux faire] mais avec le temps impartie les nombreux problemes rencontraient ce résultat ne peut que nous convenir
Ce rapport résume un travails de plus d'un mois et demi et beaucoup d'heures de travaille , nous avons essayé de vulgariser et de détailler au mieux chaque partie de notre travaille .

Nous souhaitons que ce rapport de stage soit a la porter de toute personne voulant ce documenter sur ce sujet la , ou bien une personne qui voudrait participé a une des prochaines éditions de POLYMAZE.

Chapitre VIII

Bibliographie

Bibliographie

- TCRT5000 datasheet :
<https://www.vishay.com/docs/83760/tcrt5000.pdf>
- <https://create.arduino.cc/projecthub/abhilashpatel121/using-ir-sensor-tcrt-5000-with-arduino-7cf482>
- STM32 datasheet :
<https://www.st.com/resource/en/datasheet/stm32f103c8.pdf>
- Regulateur 7805 datasheet :
<https://www.sparkfun.com/datasheets/Components/LM7805.pdf>
- L293 Driver :
<https://www.ti.com/lit/ds/symlink/l293.pdf>
- MT3608 Datasheet :
<https://www.olimex.com/Products/Breadboarding/BB-PWR-3608/resources/MT3608.pdf>