Rapport de projet IMA4 2016/2017

ROBOT MOBILE POLYTECH'LILLE

Said Ahmed Cheikh Soilihi IMA4 SA

Encadrants : Alexandre **Boé** – Xavier **Redon**

Sommaire:

I/INTRODUCTION

Ce projet rentre dans le cadre de la 4e année de formation ingénieur en Informatique, Microélectronique et Automatique. Il m'a offert l'occasion d'approfondir et d'appliquer les connaissances acquises au cours de notre formation, aussi bien dans le domaine technique que dans le domaine du management de projet.

J'ai choisi de travailler sur la réalisation d'un robot mobile contrôlé par un Atmega328. Ce projet consiste à reprendre un précédent stage et apporter des modifications au niveau :

- **De la carte principale**: Lier les leds à une tension VCC au lieu de la masse, changer la capacité des condensateurs du quartz à 22 pF, changer l'emplacement du quartz, qui doit être éloigné du convertisseur afin d'éviter les perturbations causées par ce dernier, tout en étant proche du µC. De même que pour les capacités de découplage qui doivent être proche du µC. Changer le quartz de la carte à base d'Atmega CMS

- **Du châssis**:concevoir un châssis qui arrive à porter les deux types de motorisation : moto-réducteur d'entrée de gamme et moto-réducteur de qualité correcte .

- **De la carte moteur** : relier les deux plans de masse et implanter un capteur de vitesse sur toutes les motorisations

- **De la carte suiveur de ligne** : vérifier sa performance, prévoir son positionnement sur le châssis.

- **Des piles** : à mettre au dessous du châssis.

- **La carte ultrason** : concevoir une carte électronique avec des capteurs ultrason plus performants que sur les capteurs d'entrée de gamme du commerce. Réaliser le schématique et le PCB avec le logiciel Fritzing.

A terme, ce robot doit pouvoir aller droit et suivre une ligne tout en mesurant la vitesse de ses deux moteurs.

II/Électronique embarquée

Dans cette partie, je vais décrire la technologie hardware utilisée ainsi que les différentes modifications apportées au système initial.

1)Carte principale

Il s'agit de la carte centrale du robot. Elle est constitué de différents composants remplissant chacun un rôle dans le fonctionnement du robot :

- **LM1117 :**

Il s'agit d'un régulateur 5V . Il convertit la tension 9V fourni par les piles en tension de 5V utilisée par tous les composants du robot. Conformément à la documentation fourni par datasheet, j'ai modifié la valeur des condensateurs à 10µF.

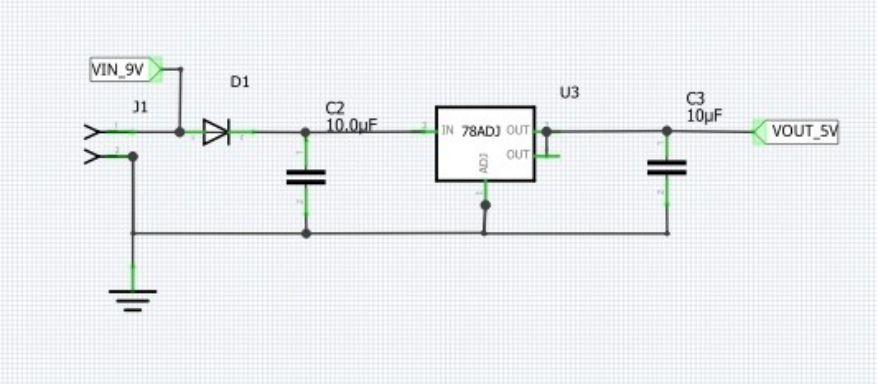


Figure 1

- **FT232R**:

C'est le composant choisi pour réaliser l'interface USB/UART. Contrairement au circuit du LM1117, j'ai procédé à plusieurs modifications. En effet, après lecture de la datasheet et discussion avec mes encadrants j'ai modifié le sens des leds aux bornes des broches TXLED et RXLED et raccordé celles-ci à l'alimentation Vcc. J'ai aussi enlever la capacité à la broche DTR.

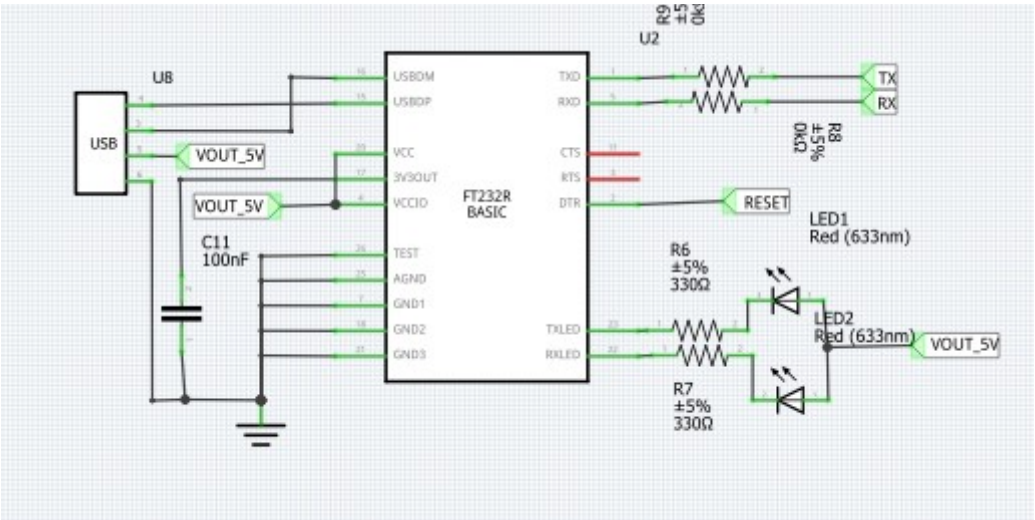


Figure 2

- **TB6612FNG :**

Ce composant nous offre la possibilité de contrôler deux moteurs simultanément dans les deux sens de rotation. La commande de chaque moteur est réalisé grâce à une combinaison de quatre signaux suivant la table fournit dans l'annexe (annexe 1).

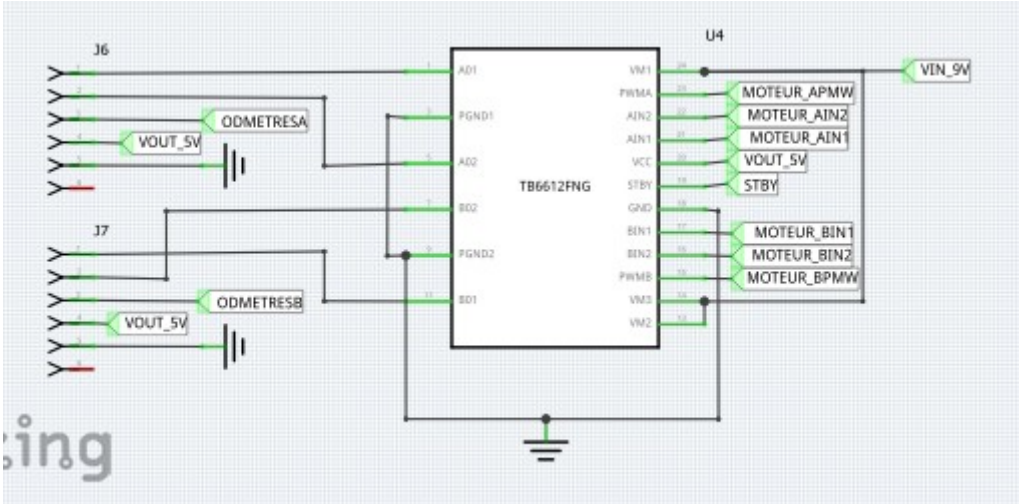


Figure 3

- **Atmega328**

Sur la version traversante, j'ai modifié la disposition des broches reliées au **TB6612FNG** afin d'avoir une disposition du PCB plus intelligente. J'ai rapproché le quartz et les capacités de découplge du microcontrôleur et éloigné le quartz du **TB6612FNG**

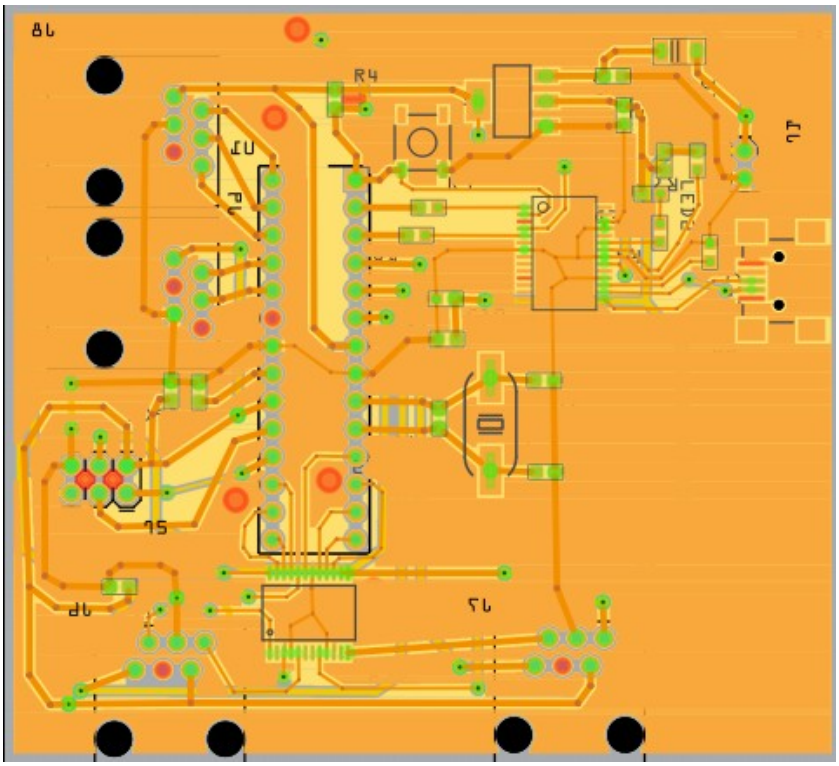


Figure 5. PCB Atmega traversant

Quant à la version CMS, j'ai remplacé le quartz par celui utilisé dans la version traversant

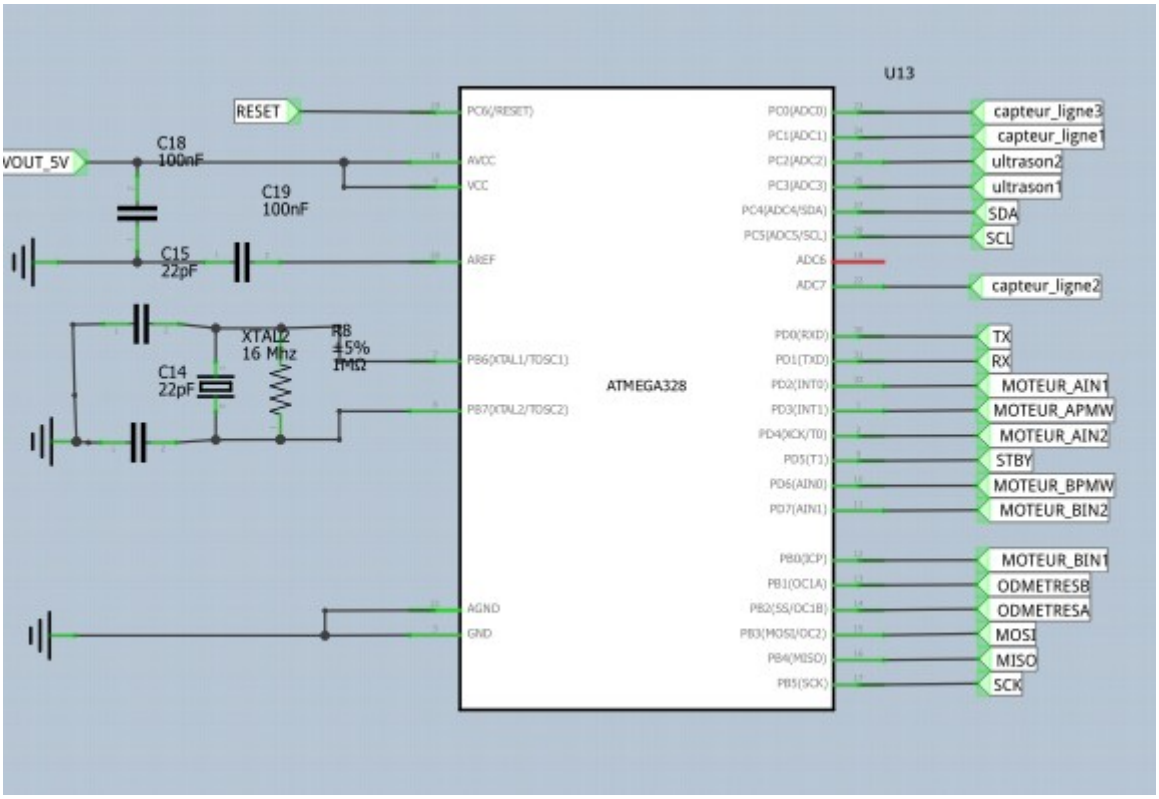


Figure 6. Atmega CMS

2)Carte moteur

C'est sur cette carte qu'on aura le système permettant de réaliser la fonction « capteur de vitesse de moteurs ». Ce système est formé du composant **KTIR0221DS** et d'une roue à encoches.

Le **KTIR0221DS** est une fourche optique composée d'une diode infrarouge et de deux phototransistors montés en Darlington, ce qui permet une réponse très rapide. Ainsi si nous placions une roue à encoches entre la diode et le phototransistor, la fourche renverrait une série d'impulsions: état bas lorsque une encoche se présente devant la diode et laisse ainsi passer le faisceau, état haut sinon (annexe 2). Ce qui nous permettra par la suite de déterminer la vitesse de rotation ou mesurer la position du robot.

Le **KTIR0221DS** n'existant pas dans la bibliothèque Fritzing, j'ai dû la crée moi même en s'aidant de la datasheet, puis réalisé le schéma de câblage en se référant à cette dernière.

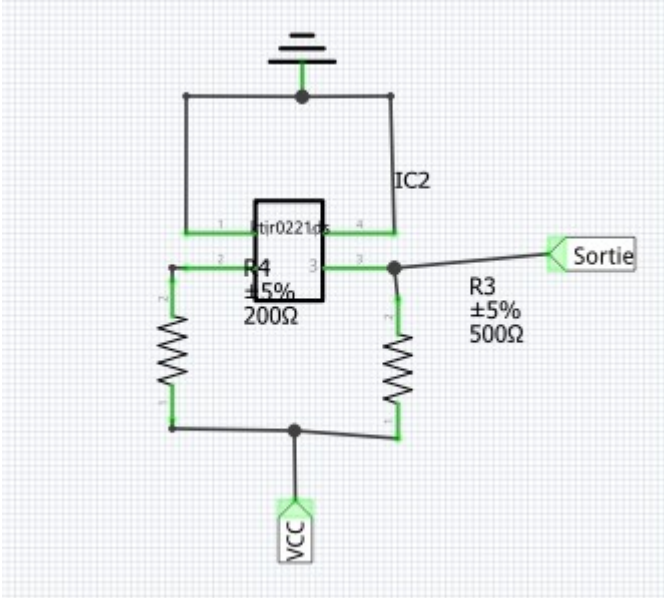


Figure 7

3)Carte suiveur de ligne

Le capteur utilisé est le **QRE1113**, il est composé d'une led infrarouge et d'un phototranstior. Le principe de fonctionnement est le suivant: le capteur étant orienté vers le sol, la LED émettrice envoie une lumière infrarouge que le sol réfléchit en direction du phototransistor qui capte ainsi la quantité de lumière en retour. Sachant que les couleurs foncées réfléchissent moins facilement la lumière que les couleurs claires. On pourra ainsi différencier une ligne noire d'une zone blanche. (annexe 3)

L'image de l'ensemble des PCB produits est fourni en annexe(annexe 6)

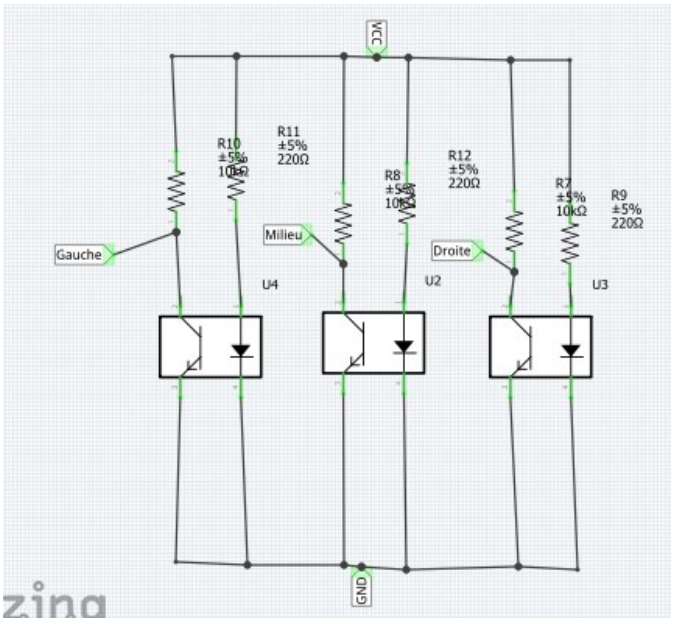


Figure 8

III/Programmation

Connaissant le fonctionnement de chaque composant, et muni d'Atmel Studio comme compilateur, j'ai pu entamer l'implémentation du code qui permettra de réaliser les différentes fonctions du robot.

1)Commande moteurs

Comme explicité dans la partie II.1, la commande des moteurs se fait par l'intermédiaire du **TB6612FNG.** En se référant au schéma de câblage entre l'atmega et le **TB6612FNG** (annexe 4), je dois configurer les pins PB3, PB1, PD7 et PD8 en sortie et générer un PWM sur les pins de sortie OC0A du timer0 et OC1B du timer1. Cette configuration est réalisée par la fonction **init_mot**

```
void init_mot()
{
    //PB3, PB1, PD7 et PD8 en sortie
    DDRD=0xE0;
    DDRB=0x0F;
    // PWM 8 bit sur la broche OC1B
    TCCR1A |= (1 << COM1B1) | (1 << WGM10);
    TCCR1B|=( 1 << CS10) | (1 <<WGM12);
    // PWM 8 bit sur la broche OC0A
    TCCR0A|= (1 << WGM01) |(1 << WGM00) | (1<< COM0A1);
    TCCR0B|=(1 << CS00);
}
```

Ensuite j'ai crée des fonctions pour chaque combinaison de commande du moteur. Pour la commande avancé tout droit, par exemple, je doit mettre la broche 1 du port B à l'état 0 et la broche 3 du port B à l'état 1 pour le moteur A, la broche 7 du port D à l'état haut et la broche 0 du port B à l'état bas conformément au schéma de câblage(annexe 4) ainsi qu'à la table de fonctionnement du TB6612FNG(annexe 1). La vitesse des moteurs est quant à elle fonction du rapport cyclique **α** du PWM. Ce dernier peut être modifié grâce aux registres OCR0A et OCR1B, par la relation

$$\alpha = \frac{OCRxx}{255}$$

Pour avancer droit, on doit logiquement donner la même vitesse au deux moteurs.(fig 9)

```
void Avance(uint8_t vitesse)
{  //La vitesse à entrer doit être un pourcentage de la vitesse maximale
    uint8_t PWM=(vitesse*255)/100;
    //Pour le moteur A
    PORTB|= (1 << PORTB3); //IN1
    PORTB&=~(1 << PORTB1); //IN2
    OCR1B=PWM;
    //Pour le moteur B
    PORTB&= ~(1 << PORTB0); //IN2
    PORTD|=(1 << PORTD7); //IN1
    PORTD|=(1 << PORTD5);//STDBY
    OCR0A=PWM;
}
```

Figure 9. Fonction d'initialisation

En suivant la même méthodologie,j'ai implémenté et testé les fonctions permettant de reculer, s'arrêter, tourner à droite et tourner à gauche:

```
void Stop()
{
    //Pour le moteur A
    PORTB&= ~(1 << PORTB1);
    PORTB&= ~(1 << PORTB3);
    OCR1B=255; //Equivaut à un etat haut permanent sur la broche OC1B
    //Pour le moteur B
    PORTB&= ~(1 << PORTB0);
    PORTD|=(1 << PORTD5);
    PORTD&=~(1 << PORTD7);
    OCR0A=255;//Equivaut à un etat haut permanent sur la broche OC1B
}
```

Figure 10. Fonction pour s'arrêter

```
void short_brake()
{
    //Pour le moteur A
    PORTB|= (1 << PORTB1)|( 1 << PORTB3);
    //Pour le moteur B
    PORTB|= (1 << PORTB0);
    PORTD|=(1 << PORTD7)|(1 << PORTD5);
}
```

Figure 11. Short brake

```
void Av_droite()
{
    //Pour le moteur A
    PORTB|= (1 << PORTB3); //IN1
    PORTB&=~(1 << PORTB1); //IN2
    OCR1B=10;
    //Pour le moteur B
    PORTB&= ~(1 << PORTB0); //IN2
    PORTD|=(1 << PORTD7); //IN1
    PORTD|=(1 << PORTD5);//STDBY
    OCR0A=80;
}
```

Figure 12. Tourner à droite

```
void Av_gauche()
{
    //Pour le moteur A
    PORTB|= (1 << PORTB3); //IN1
    PORTB&=~(1 << PORTB1); //IN2
    OCR1B=80;
    //Pour le moteur B
    PORTB&= ~(1 << PORTB0); //IN2
    PORTD|=(1 << PORTD7); //IN1
    PORTD|=(1 << PORTD5);//STDBY
    OCR0A=10;
}
```

Figure 13. Tourner à gauche

2)Lecture de la vitesse

Pour cette partie, il m'a paru judicieux d'utiliser les interruptions externes INT0 et INT1. Un front descendant sur ces broches générera une interruption dont la routine servira à incrémenter un compteur. Cette configuration est réalisée par la fonction **init_encod.** La définition de cette fonction ainsi que les routines est explicitée dans la figure 14

```
void init_encod()
{
    SREG|=1<<SREG_I;//Activation des interruptions
    EICRA|=(1<<ISC11)|(1<<ISC01);//Interruption sur front déscendant sur INT0 et INT1
    EIMSK|=(1 << INT1)|(1<<INT0);//Activation des demandes d'interruptions sur INT0 et INT1
}

ISR(INT0_vect)
{
    compteur_motB++;
}

ISR(INT1_vect)
{
    compteur_motA++;
}
```

Figure 14

Ensuite pour lire la vitesse, j'utilise le timer 2 pour générer des interruptions. Étant un timer 8 bits, il n'est pas possible de générer une interruption toutes les secondes. Ainsi pour remédier à cela, je décide d'en générer toutes les millisecondes et dans la routine incrémenter un compteur qui, lorsque elle vaut la valeur 1000 entre dans une boucle qui nous donnera la vitesse

```
void init_vitesse()
{
    OCR2A = 15; //remise à 0 toutes les millisecondes
    TCCR2A |= (1 << WGM21);// CTC mode
    TIMSK2 |= (1 << OCIE2A);//Faire une interruption à chaque égalité
    TCCR2B |= (1 << CS20)|(1 << CS21)|(1 << CS22); //prescaler 1024
}
```

Figure 15. Configuration Timer 2

Le disque que j'ai modélisé sur Onshape (figure 17)contient 8 encoches. Ainsi pour avoir la vitesse en tour par secondes, on doit diviser la valeur du compteur par 8

```
ISR (TIMER2_COMPA_vect)
{
    cpt_vit++;
    if(cpt_vit==1000) // pour avoir 1s
    {
        vitA=(compteur_motA/8)-vitA; //Vitesse en tour par secondes( le disque a 8 encoches)
        vitB=(compteur_motB/8)-vitB;
        cpt_vit=0;
    }
}
```

Figure 16. Routine d'interruption

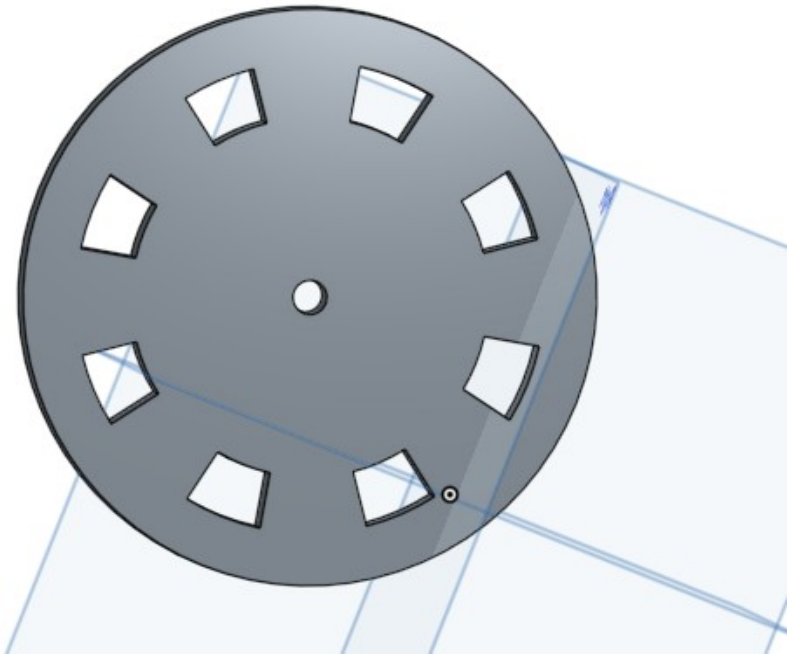


Figure 17

3)Suiveur de ligne

Les capteurs utilisés pour réaliser cette fonction étant analogique, j'ai utilisé le CAN du µC. Les voies de lectures sont les voies 3,4 et 5 (annexe 5). je définis alors une fonction **lecture_CAN** qui prend en argument la voie de lecture et retourne la valeur du CAN correspondant à la voie de lecture

```
uint16_t lecture_CAN(int voie)
{
    switch(voie)
    {
        case 5:
            ADMUX= (ADMUX & 0xF8)|0x05; // On efface les 3 derniers bits du registre ADMUX puis on choisit ADC5 comme pin de lecture avec 0x05
            break;
        case 4:
            ADMUX= (ADMUX & 0xF8)|0x04; // On efface les 3 derniers bits du registre ADMUX puis on choisit ADC4 comme pin de lecture avec 0x04
            break;
        case 3:
            ADMUX= (ADMUX & 0xF8)|0x03; // On efface les 3 derniers bits du registre ADMUX puis on choisit ADC3 comme pin de lecture avec 0x03
            break;
    }
    ADCSRA|=(1 << ADSC);// On commence une conversion
    while (ADCSRA & (1<<ADSC))//On attends la fin de la conversion
    {
    }
    return(ADC);
}
```

Figure 18

Dès lors, j'ai pu définir l'algorithme général de fonctionnement du robot :

```
int main(void)
{
    init_encod();
    init_mot();
    init_CAN();
    init_vitesse();
    int choix=5;//On commence d'abord par lire la voie 5
    uint16_t capteur;
    while (1)
    {
        Avance(40);
        capteur=lecture_CAN(choix);
        if(capteur>Seuil_suiv) //Si un capteur en lecture détecte une sortie de ligne sombre
        {
            short_brake(); //On arrête le robot
            switch(choix) //Puis on prend l'information sur le capteur
            {
                case 5: //si c'est celui de gauche
                while(capteur>Seuil_suiv)
                r
```

Figure 19.1. Main

```
{
    Av_droite(); // on tourne à droite tant que le capteur de gauche est en dehors de la ligne sombre
    capteur=lecture_CAN(5);
}
break;
case 3://si c'est celui de droite
while(capteur>Seuil_suiv)
{
    Av_gauche(); // on tourne à gauche tant que le capteur de droite est en dehors de la ligne sombre
    capteur=lecture_CAN(3);
}
break;
case 4:// Si c'est le capteur du milieu
while(capteur>Seuil_suiv)
{
    Recule(80);//On recule
    capteur=lecture_CAN(4);
}
break;
```

Figure 19.2. Main

```
        }

        switch(choix) //Cette boucle nous assure un changement de voie de lecture du CAN
        {
            case 5:
            choix=4;
            break;

            case 4:
            choix=3;
            break;

            case 3:
            choix=5;
            break;
        }

}
```

Figure 19.3. Main

IV/Simulation

Afin de tester mes différentes fonctions et de m'assurer du bon fonctionnement, j'ai mis en place différentes simulations sous Proteus (figure 20), et sur 123D circuit

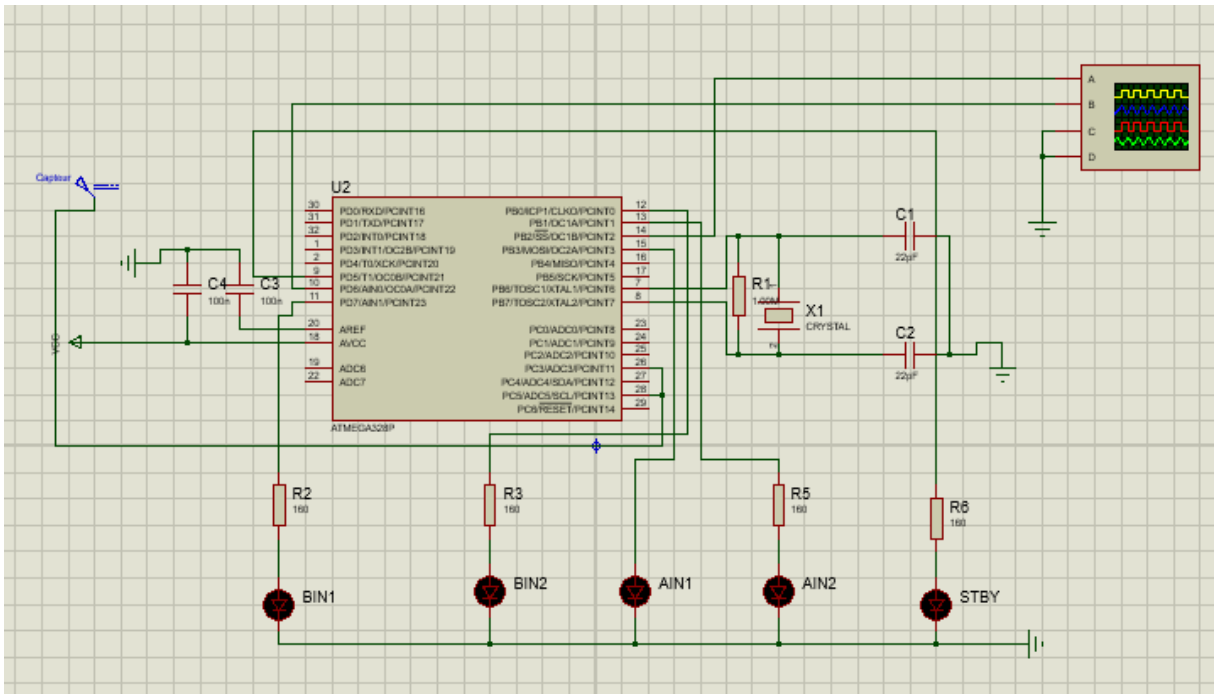


Figure 20

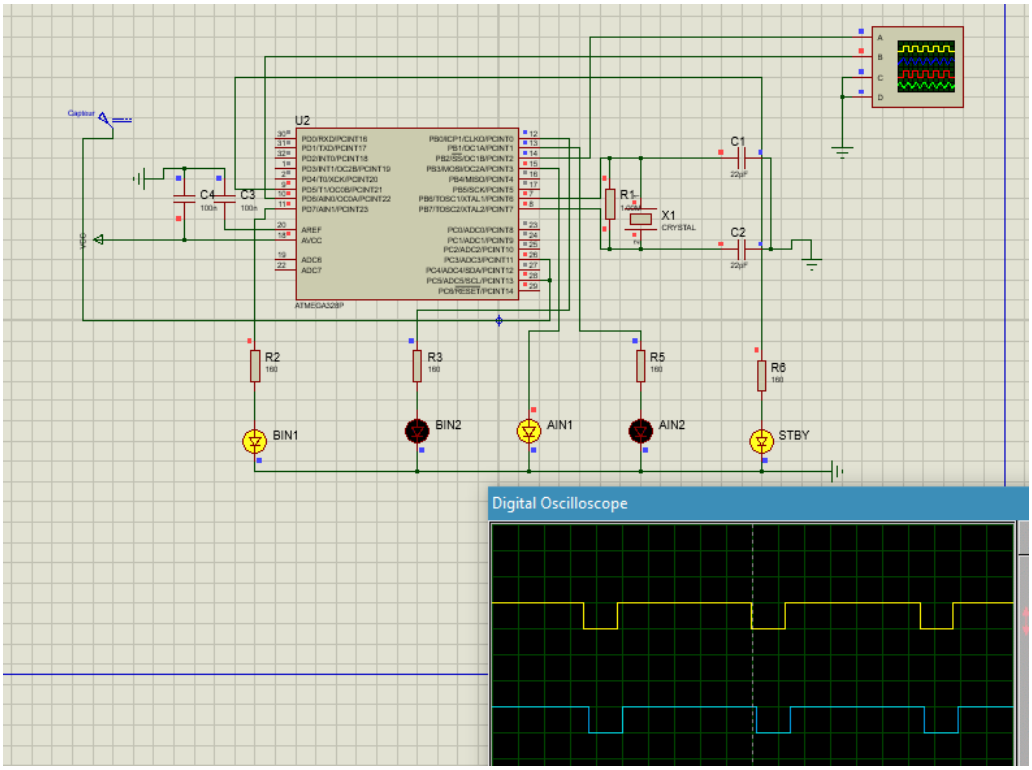


Figure 21. État de la simulation lors de l'exécution de la fonction Avance

V/Conception

Outre le disque à encoche, j'ai aussi travaillé sur la conception d'un châssis qui permettra d'accueillir les deux types motorisations ainsi que la carte de capteurs **QRE1113** qui se sera suspendu à l'avant du châssis

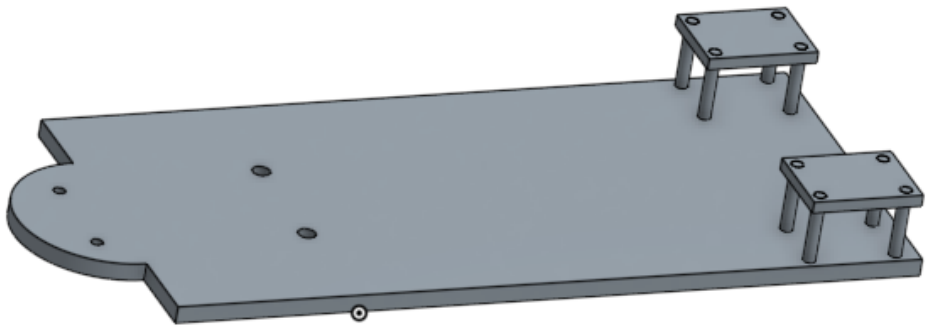


Figure 22

III/Conclusion

Ce projet a été effectué dans le cadre de ma quatrième année dans la formation Informatique Microélectronique Automatique de Polytech Lille.

Ce projet, nous avons mis en pratique les connaissances acquises au cours de ma formation mais aussi développé de nouvelles dans les domaines de l'électronique, de l'informatique et de la CAO. D'une part de manière autodidacte, et d'autre part grâce à l'aide apportée par mes encadrants et mes enseignants.

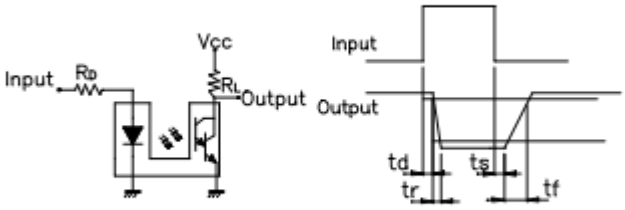
Néanmoins, j'aurais aimé avoir eu plus temps afin de réaliser le soudage des cartes. Je tenais beaucoup à voir le système en plein fonctionnement.

ANNEXE 1:

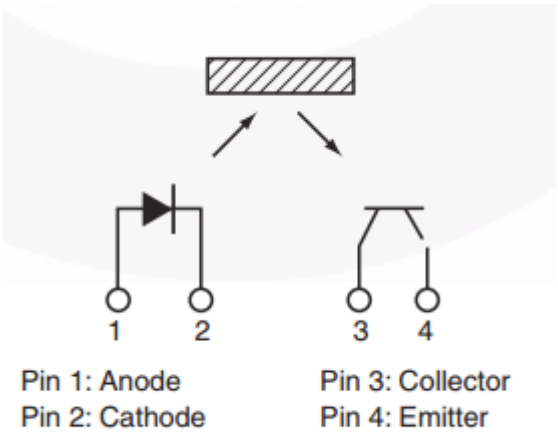
H-SW Control Function

Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

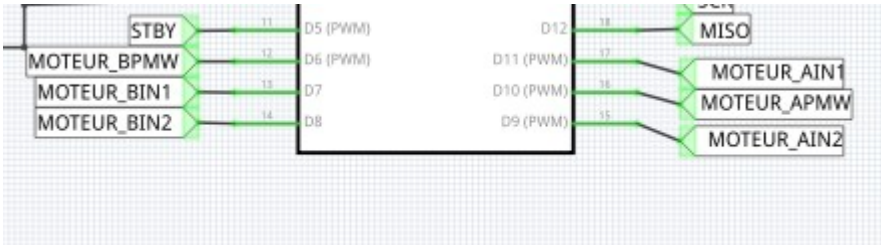
ANNEXE 2:



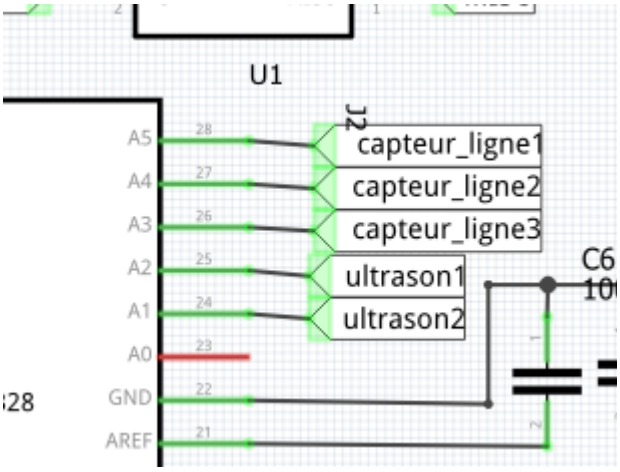
ANNEXE 3:



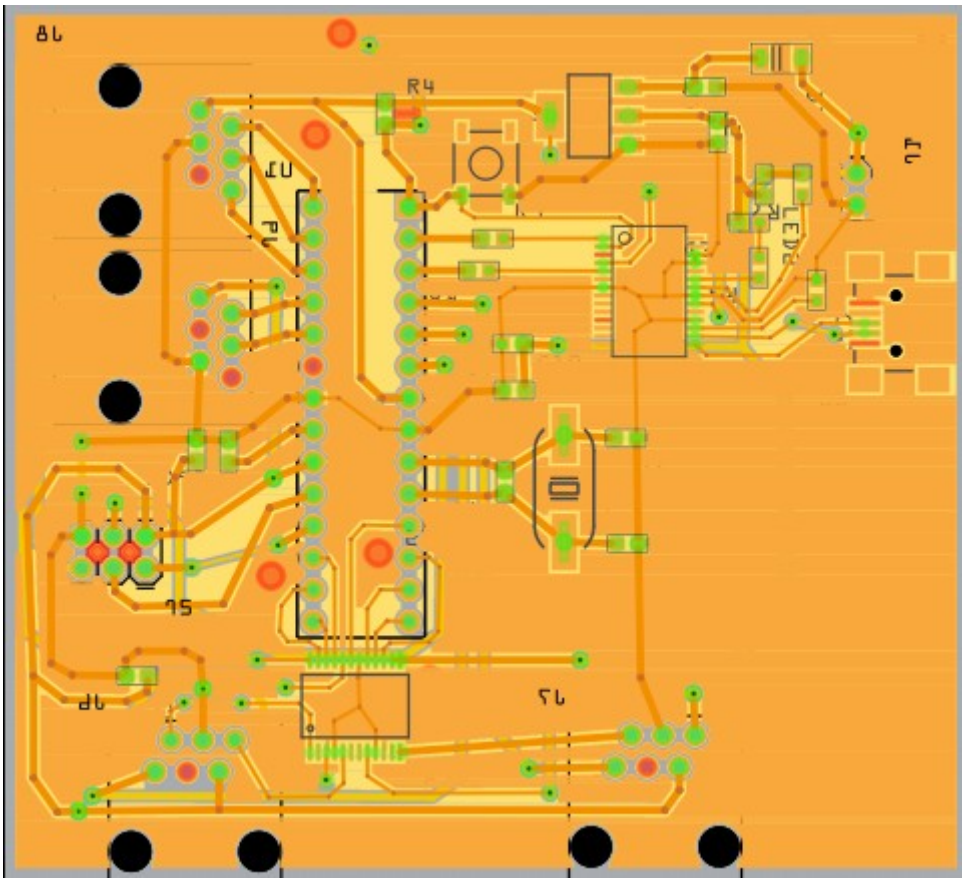
ANNEXE 4:



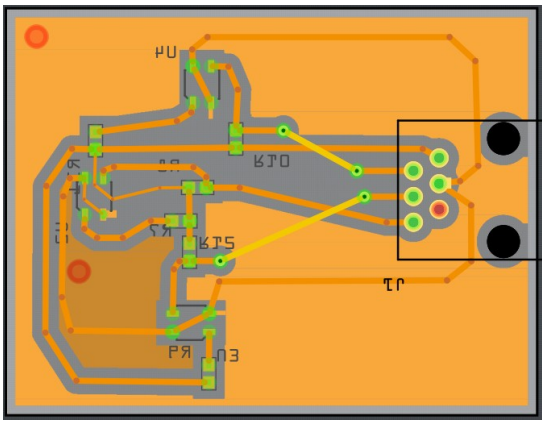
ANNEXE 5:



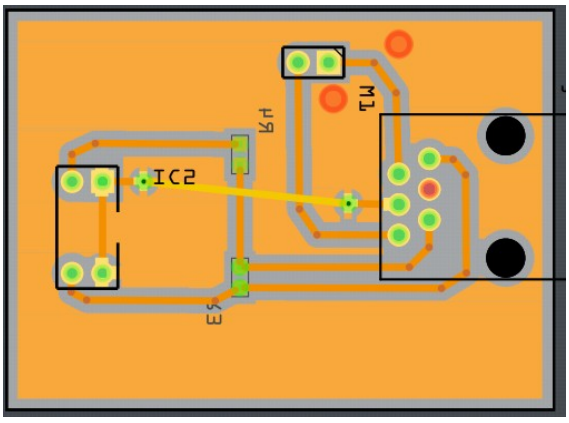
ANNEXE 6:



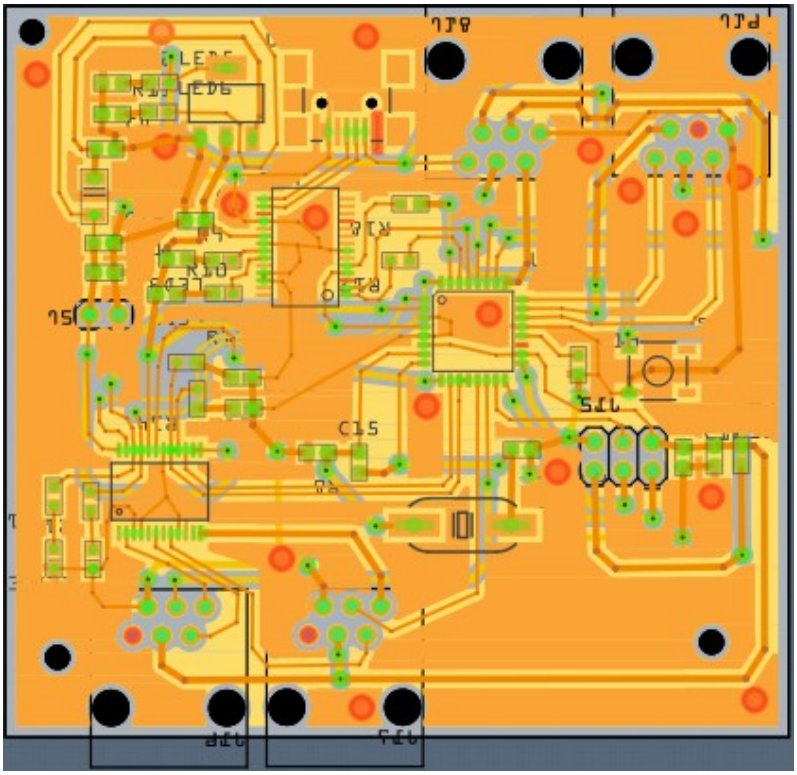
Carte principle Atmega traversant



Capteur QRE1113



Capteur KTIR022DS



Carte principle Atmega CMS