Département de Génie Electrique

Filière d'Ingénieurs

Contrôle commande des systèmes électriques

(C.C.S.E)

Livrable N°3 du Projet Robot MARK

Pilotage du robot et étude énergétique

Réalisé par :

- **VEYRIOL Luc**
- **RAOULT Lucas**

Sous l'encadrement de :

- **SAHLI Issam**
- **BARAKAT Abdallah**
- **DELFIEU David**

Année universitaire : 2019/2020

I. Table des matières

A. Introduction

1. Etat d'avancement du projet

Durant la troisième phase du projet nous avions pour objectifs de terminer la gestion de tous les calculs et de l'affichage des informations demandées par le cahier des charges, ainsi que la finition de la partie concernant le pilotage du robot. Avec plus de recul, nous avons pu améliorer la qualité de notre cahier des charges afin de définir des réponses plus adaptées aux demandes de l'utilisateur avec plus de précisions et d'explications.

Nous avons aussi changé la structure principale de notre programme. Afin qu'il soit lisible, plus facilement, nous avons créé des fonctions pour chaque tâche.

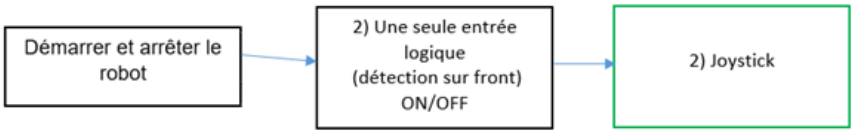
Nous nous sommes aperçus depuis peu que les moteurs n'étaient plus calibrés de manière équivalente. Nous avons donc définit une correction moteur afin de régler ce problème. Le moteur gauche, à sa valeur maximale, étant à 72 au lieu de 100 écrits dans le programme (à vitesse maximum). En pratique, il nous a fallu mettre la vitesse du moteur droit à 72 pour le faire avancer le plus vite possible de manière rectiligne. De ce fait, la vitesse maximale de notre robot est nettement diminuée.s

2. Modification de l'analyse fonctionnelle

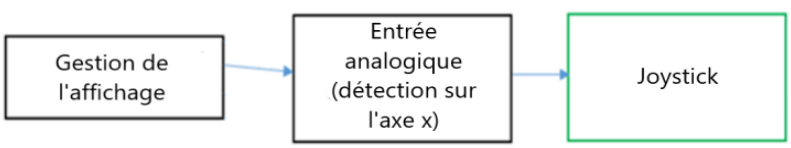
a) Le joystick

Le joystick a plusieurs fonctions. Nous remarquions dans les précédant livrables qu'il permettait à la fois de changer l'état de fonctionnement du robot et de gérer la gestion de l'affichage LCD. Cette fois, nous lui rajoutons une fonction qui permet de réinitialiser les données du robot. Cela permet de faire plusieurs tests à la suite et nous évitons ainsi de le mettre hors tension.

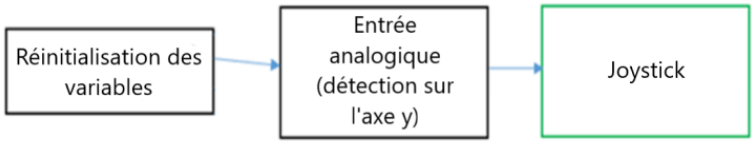
- Concernant l'état de fonctionnement du robot, nous le démarrons ou arrêtons en provoquant une impulsion sur je joystick.



- Pour la gestion de l'affichage nous avons gardé le même principe de fonctionnement en agissant sur les côtés du joystick (axe X)



- Pour réinitialiser les valeurs des variables du robot, nous avons choisis d'agir sur joystick vers le haut (axe Y)



b) *L'écran*

Nous nous servons de l'écran pour afficher les informations utiles à l'utilisateur. Comme dit précédemment, on se sert du joystick pour naviguer dans le menu et choisir l'écran qui intéresse l'utilisateur. Voici les différentes informations affichées sur les écrans pour un cycle de marche (sans redémarrage) :

- L'état de marche du robot ;
- La vitesse maximale de chaque roue (en m/s);
- Le nombre de tour de chaque roue ;
- La distance parcourue (en cm) ;
- Le nombre de fois où le robot est passé à une distance inférieure à 20 cm d'un mur ;
- Le chronomètre du trajet ;
- L'énergie consommée.

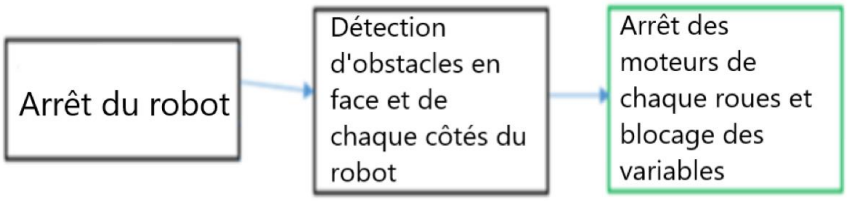
c) *Les capteurs ultrasons*

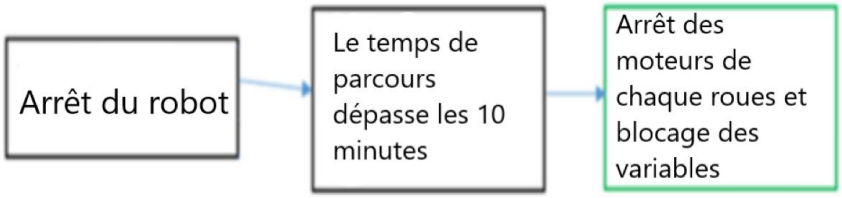
Les capteurs ultrasons nous permettent de définir la distance séparant le robot d'un obstacle. Au nombre de 3, ils sont placés à l'avant du robot, l'un détecte devant le robot, les deux autres sur les côtés.

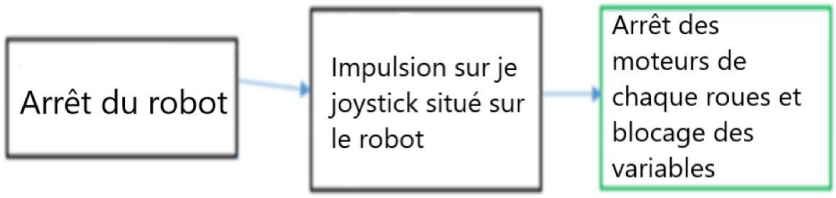
A savoir que la détection des capteurs se fera toutes les 30 interruptions (soit à peu près 18ms) afin d'éviter de consacrer un temps trop important à la détection.

Nous utilisons principalement le capteur avant et droit pour l'acquisition des données, nécessaire aux déplacements de notre robot. Il se déplacera à une distance du mur droit, prédéfinie dans notre programme. Le capteur avant permet de détecter un obstacle et ainsi d'agir en conséquence lors des virages.

Nous utilisons le capteur gauche lorsque le robot arrive à la fin du parcours et qu'il doit s'arrêter. En effet nous avons abandonné l'idée d'utiliser le capteur infrarouge pour la détection d'une ligne « d'arrivée » car la détection ne fonctionnait pas correctement. Ainsi, l'arrêt du robot pourra être provoquer de 3 façons :

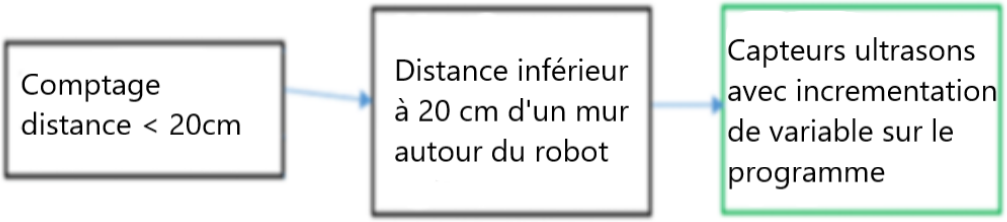






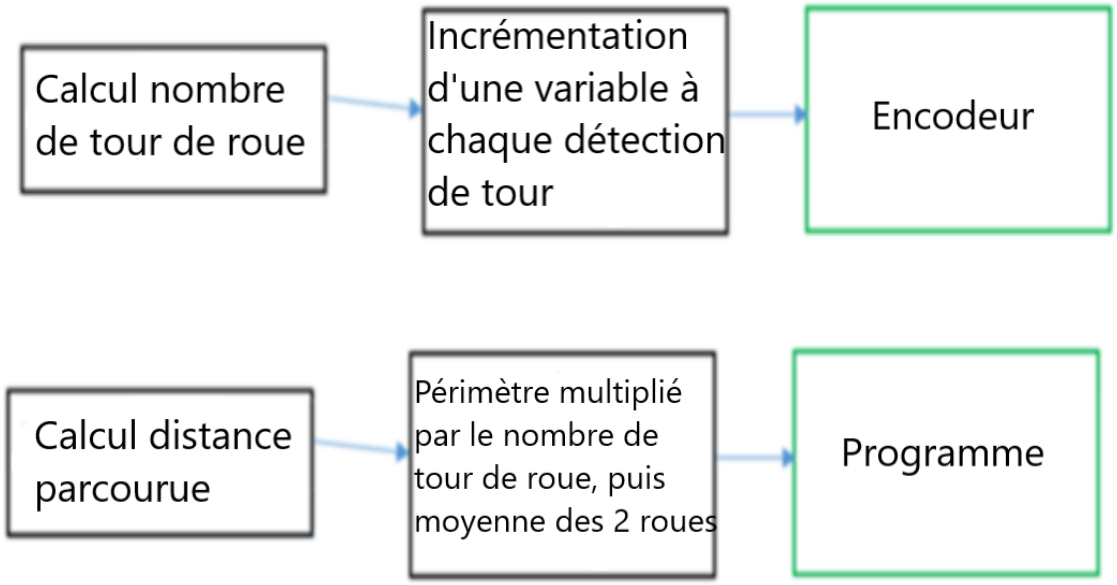
Le blocage des variables permet de visualiser sur l'écran les dernières données enregistrées, telles que le chronomètre, la distance parcourue…

Les capteurs vont aussi détecter si le robot est à moins de 20 cm d'un mur, nous incrémentons alors une variable dans notre programme permettant alors de compter ce nombre de fois.



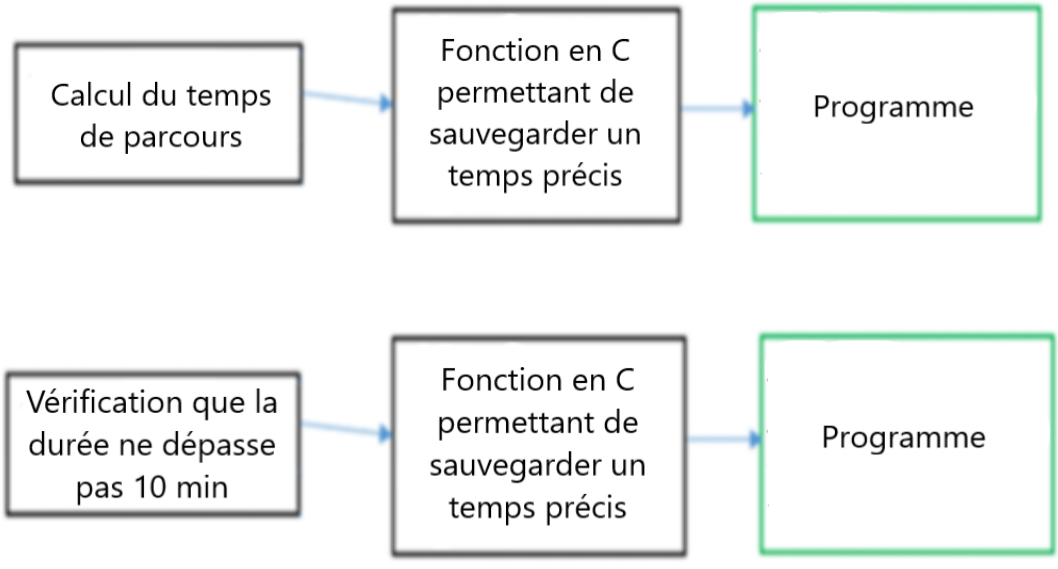
d) Encodeur

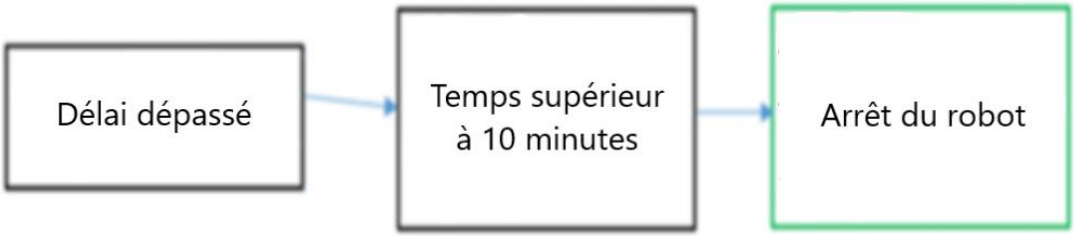
L'encodeur permet de calculer le nombre de tours de roue, ce qui nous donnera aussi après un bref calcul, la distance parcourue par le robot (en multipliant par la circonférence de la roue et en faisant une moyenne des valeurs de chaque roues).



e) Gestion du temps :

Pour calculer du temps de parcours, nous avons décidé de ne pas utiliser le timer, mais plutôt une fonction déjà existante en C, « *millis()* ». Cette fonction permet de ressortir un temps précis lors de son déclenchement. En appelant cette fonction lors de l 'activation du mode marche et du mode arrêt du robot, on obtient 2 temps fixes, *t0* et *t1.* Il faut alors soustraire ses deux temps pour obtenir la durée du trajet. Aussi, il faut que cette même durée ne dépasse pas les 10 minutes, On appel à cette fonction à chaque boucle du programme, permettant ainsi de voir le chrono en temps réel et donc arrêter le robot si la durée dépasse les 10 minutes. Si cette durée est dépassée le robot s'arrête et stop les calculs des variables (temps, énergies…).





f) Calcul de l'énergie :

La théorie dit que l'énergie électrique est le produit de la puissance électrique consommée par le temps.

$$P * t = \varepsilon$$

On peut déduire la puissance P en connaissant la tension fournit par la batterie et par le courant consommé du système. Après des mesures du courant sous différentes contraintes des couples (engendrées par la rotations des roues), on constate une variation du courant de 350 à 450 mA. Dès lors, on admet que le courant est constant. On prendra $I = 400mA$.

La tension est mesurée par le diviseur de tension. Le temps est chronométré comme expliqué dans le paragraphe *e) Gestion du temps.*

B. Code

Comme dit précédemment nous avons complétement changer la structure de notre programme afin de le rendre plus lisible. Nous avons par exemple fait des fonctions pour l'affichage sur l'écran ou encore le mode marche/arrêt du robot.

1. Timer

```
//------------FONCTIONS D'INTERRUPTION PAR TIMER2----------------
ISR(TIMER2_OVF_vect){ //Timer2 = 8bits --> 256: valeurs 0 --> 255   freq=16MHz  T=625µs
    tempo++;  //A CHAQUE TOP ATTEINT --> INCRÉMENTATION DU COMPTEUR = TEMPORISATION ENTRE 2 MESURES
    //tempo2++;
    if(tempo >= 30 ){  // QUAND LE TOP A ÉTÉ ATTEINT 30 fois --> ON AFFECTE LES MESURES DES ULTRASONS À LEUR VARIABLE RESPECTIVE! Limitation des mesures
        //*****CAPTEURS ULTRASONS*****
        dist_us_front = us_front.MeasureInCentimeters(); //variable globale "dist_us_front" prend la valeur de la mesure en cm de l'US de front
        dist_us_left = us_left.MeasureInCentimeters();  //..................................................................... de gauche
        dist_us_right = us_right.MeasureInCentimeters(); //..................................................................... de droite
        joystick_X = analogRead(A3);  //Lecture analogique du joystick --> axe x + appuie
        joystick_Y = analogRead(A2);  //Lecture analogique du joystick --> axe y
        voltage = analogRead(A0) * TENSION_ECH ; //voltage reçoit le signal du diviseur de tension mis à l'échelle part rapport au 7V délivré par la batterie
        /*if(tempo2 > 208){ //
          seconde ++; //incrémentation d'un compteur par interruption de débordement
          tempo2 = 0;
        }*/
        tempo = 0;  //Remise à 0 du compteur
    }
}
```

Figure 1 : fonction d'interruption

Un compteur principal représenté par *tempo,* actualise les mesures d'Ultrasons, du Joystick et du diviseur de tension toutes les x microseconde . Cela permet de ne pas obstruer le microcontrôleur en lisant en permanence ces mesures.

La période de hachage correspond au temps entre deux cycles d'interruption.

$Fhach = \frac{Fclk}{N}$ avec N : prédiviseur affecté à 256

$Fclk$: Fréquence d'horloge de l'arduino de 16 MHz

En effectuant les mesures que lorsque $tempo >= 30$, on actualise les mesures toutes les

$$Tmesure = \frac{Thach}{30} = 480\mu sec$$

Hors, un second compteur représenté par *tempo2* (en commentaire), actualise la variable *seconde* à chaque seconde. C'est en fait une deuxième manière de concevoir le chronomètre du robot. Par ajustage manuel, la valeur de *tempo2* pour laquelle seconde est incrémenter chaque seconde est de 208.

Hors par calcul :

$$Tseconde = \frac{Thach}{30 * 208} = 10{,}01\ secondes$$

Soit un facteur 10 de trop. Nous travaillons actuellement sur ce problème malgré le bon fonctionnement de l'interruption et de ses conditions.

2. Déclaration des variables

```
//*****Déclaration des variables non matériel
volatile short int tempo = 0, tempo2 = 0; //tempos de l'interruption de débordement
volatile short int start = 0;
volatile short int mode_motor;

//*****Définition des équipements et déclaration de leurs variables associées*****
//Ultrasons
Ultrasonic us_front(8);
Ultrasonic us_right(10);
Ultrasonic us_left(2);
volatile int dist_us_right, dist_us_left, dist_us_front;
volatile short int test_20cm = 0;
```

Figure 2 : Exemple de déclarations de travail

Certaines variables difficilement associable au matériel sont regroupées tel que les compteurs *tempo*, *tempo2* ou bien le variable associée au mode de fonctionnement des moteur *mode_motor.*

Les variables liées aux capteurs sont toutes déclarées sous la déclaration de leur matériel associé. On a l'exemple ci-dessus des ultrasons.

3. Déclaration des « constantes »

```
#define COEF_TPS 1.290
#define COEF_DIST 1.
#define CIRC_ROUE 2*3.1459*3  //3cm = rayon d'une roue
#define CORRECTION_MOTEUR 28
#define CURRENT 0.400 //On suppose le courant conssommée par les moteurs constant = 150mA //En réalité lecourant varie entre 100 et 200 mA sur un sol
#define TENSION_ECH 7.00/143.00              //Tension mis à l'échelle GAIN inclu 143 = signal retourné par A0 non modifié
```

Figure 3 : déclaration des étiquettes

Nous considérons ces étiquettes comme constantes (informatiquement, elles n'en sont pas). Cela permet de poser un nom sur un calcul logique ou tout simplement un nombre.

4. Programme principal

```
void loop() {
  energie = voltage * CURRENT * chrono;  //gain de 10 / CURRENT = 150mA / "/3600" --> Wh
  affichage_LCD();
  marche_arret();
  test_US();
  calcul_tour_roue();
  nb_dist20();

  if( start == 1 ){
      gest_motors_start();
  }else{
      gest_motors_stop();
  }
}
```

Figure 4 : programme principal (effectué à l'infini)

Le programme principal effectue en boucle le cycle suivant :

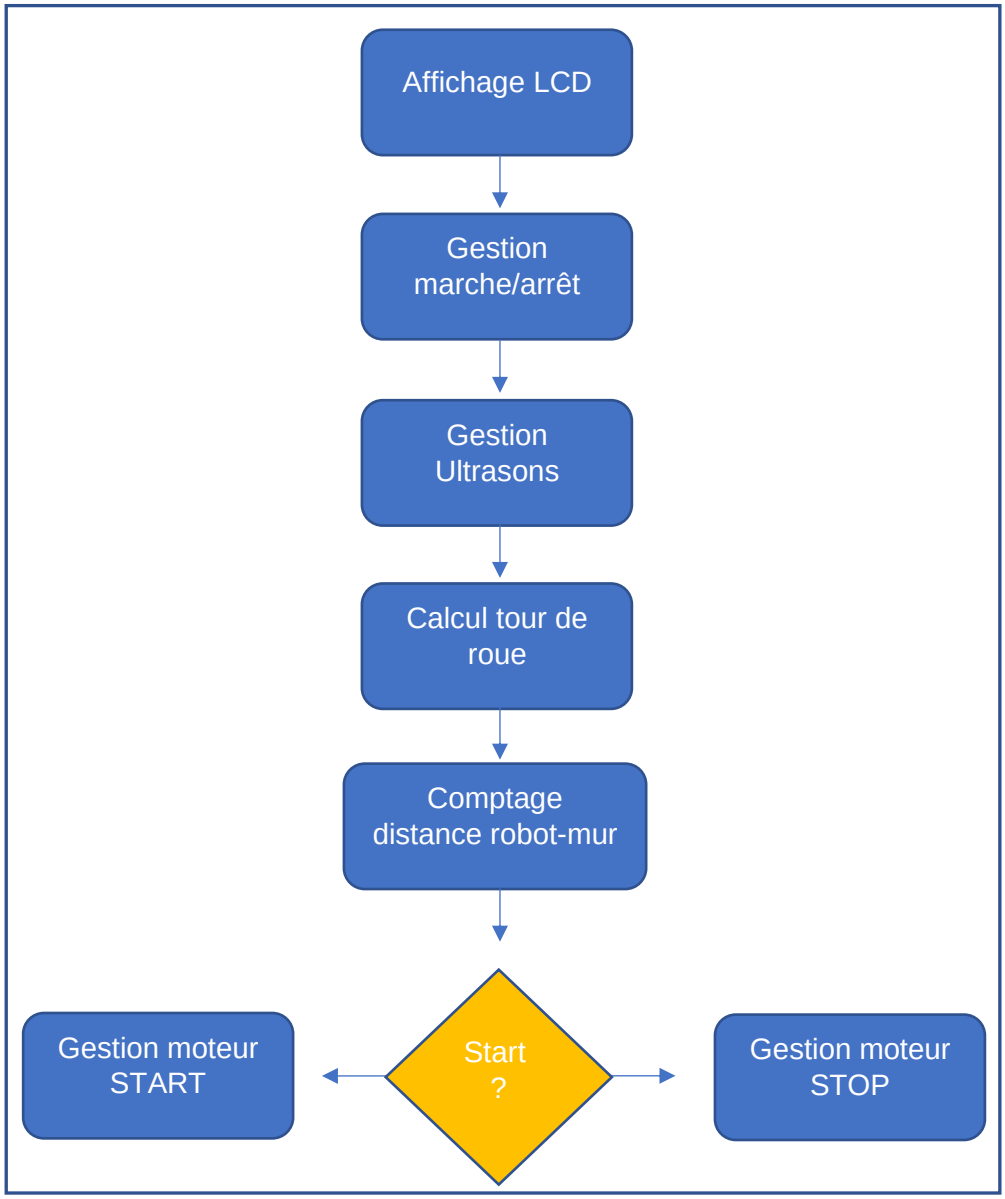


Figure 5 : organigramme du cycle principal du programme

5. Fonctions

a) *Marche / Arrêt*

```
void marche_arret(){
    if (analogRead(A2) == 1023 && start == 0){    //ORDRE DE MARCHE
        start = 1;
        t0 = millis();                  //enregistrement du compteur "millis" à l'appui sur START
        _delay_ms(200);
    }
    if (analogRead(A2) == 1023 && start == 1){    //ORDRE D'ARRÊT
        start = 0;
        t1 = millis();                  //enregistrement du compteur "millis" à l'appui sur STOP
        _delay_ms(200);
    }
    if (start == 1){
       t1 = millis();                   //enregistrement du compteur "millis" en temps réel pour voir en direct le chrono
    }
    chrono = (t1 - t0) * COEF_TPS / 1000 ;     //COEF_TPS = facteur d'echelle - chrono en seconde
    if (chrono == 600){           //10 minutes de parcours --> STOP
        start = 0 ;
    }
}
```

Figure 6 : fonction Marche / Arrêt

Pour cette fonction, nous utilisons la détection de l'impulsion sur le joystick pour donner l'ordre au robot de démarrer ou de s'arrêter. Nous utilisons une variable booléenne permettant de mémoriser en quel état est le robot. Dans les conditions « Si », nous utilisons un délai qui évitera ainsi au programme de faire plusieurs fois de suite les conditions de marche et d'arrêt du robot.

De plus on sert de la fonction millis() pour mémoriser des instant précis dans la lecture du programme, ainsi on peut calculer le temps du trajet du robot en enregistrement les moments de démarrage et d'arrêt.

Afin de respecter les 10 minutes de trajet maximum imposé, nous rajoutons une condition qui vérifie où en est le chrono, et ainsi qui peut détecter un éventuel dépassement de temps et qui permet de l'arrêter automatiquement.

b) *Détection ultrasons*

```
//------------------- GESTION DES DETECTION ULTRASON "LEFT"/"RIGHT"/"FRONT" -------------------------------
void test_US(){
    if( (dist_us_right <= 60) && (dist_us_right >= 50) ){ //entre 55 et 60 cm
        mode_motor = 0;
    }
    if(dist_us_right > 60){
        mode_motor = 1;
    }
    if(dist_us_right < 50){
        mode_motor = 2;
    }
    if( (dist_us_front < 70 && dist_us_left > 100) ){ //condition rajoutée
        mode_motor = 3;
    }
    if( (dist_us_right > 250) && (dist_us_front > 170) ){
        mode_motor = 4;
    }
    if( (dist_us_right < 90) && (dist_us_left < 90) && (dist_us_front < 75) ){
        mode_motor = 5;
    }
```

Figure 7 : fonction gestion des ultrasons

Cette fonction permet de sélectionner un choix de mode de fonctionnement en fonction des valeurs que les capteurs ultrasons retournent.

A l'aide des 3 capteurs, nous avons étudié quels étaient les meilleures solutions pour arriver le plus rapidement possible à la fin du parcours tout en se basant par rapport à un mur à droite.

Son rôle est de recevoir et de donner un ordre de mode de fonctionnement à la fonction de mode de marche des moteurs

```
//------------------- GESTION MOTEUR START --------------------------------
void gest_motors_start(){
    switch(mode_motor){
    case 0:
        Motor.speed(MOTOR1,100);  //vitesse (-100 ; +100)
        Motor.speed(MOTOR2, 100);   // Ajustement du moteur droit par une correction = -28
        break;
    case 1: //VIRE A DROITE
        Motor.speed(MOTOR1,100);  //MOTOR1 coté gauche
        Motor.speed(MOTOR2,50);  //MOTOR2 coté droite //60 trop lent 50 ok
        break;
    case 2: // VIRE A GAUCHE
        Motor.speed(MOTOR1,50);
        Motor.speed(MOTOR2,100);
        break;
    case 3: //VIRAGE SERRE A GAUCHE
        Motor.speed(MOTOR1,-50);  //MOTOR1 coté gauche
        Motor.speed(MOTOR2,100);  //MOTOR2 coté droite
        break;
    case 4: // VIRAGE SERRE A DROITE
        Motor.speed(MOTOR1,100);
        Motor.speed(MOTOR2,0);
        break;
    case 5:
        start = 0;
        break;
    }
}
```

Figure 8 : fonction moteur en marche

c) Mode marche des moteurs

Cette fonction nous permet d'affecter des vitesses de rotations aux deux moteurs en fonction des valeurs retournées par la fonction de détection ultrason.

Chacun des cas va en effet correspondre à l'environnement dans lequel se trouve le robot, plus particulièrement à la distance le séparant des murs.

Son rôle est de communiquer et d'envoyer les ordres d'instructions vers les moteurs.

d) Mode arrêt des moteurs

```
//------------------- GESTION MOTEUR STOP --------------------------------
void gest_motors_stop(){
    Motor.stop(MOTOR1);
    Motor.stop(MOTOR2);
    if(joystick_Y > 650){   //Remise à 0 des données
        t0 = 0; t1 = 0; //chrono = t0 - t1
        //vit_max_droite = 0;
        //vit_max_gauche = 0;
        knobLeft.write(0);
        knobRight.write(0);
        compt_dist = 0;
        energie = 0;
    }
}
```

Figure 9 : fonction moteur en arrêt

Cette fonction permet de donner l'ordre aux moteurs de s'arrêter, nous avons rajouter une fonction qui permet de réinitialiser toutes les variables affichées sur les différents menus de l'écran afin de pouvoir utiliser le robot une seconde fois sans devoir passer par l'interrupteur ON/OFF.

e) Comptage de distance inférieures à 20cm

```
//------------------Detection capteurs US à moins de 20 cm ---------------
void nb_dist20(){
    if ( (((dist_us_front < 20) || (dist_us_right < 20) || (dist_us_left < 20)) and test_20cm == 0) and start == 1){  // or dist_us_left < 20 or dist_us_right < 20)
       compt_dist = compt_dist + 1;                                                                                   // or dist_us_front < 20
       test_20cm = 1;
    }
    else if (dist_us_front > 20 and dist_us_right > 20 and dist_us_left > 20 and test_20cm == 1 and start == 1){   //dist_us_left < 20 and dist_us_right < 20 and
        test_20cm = 0;
    }
}
```

Figure 10 : fonction gestion distance inférieur à 20 centimètres

Nous effectuons le comptage simplement à l'aide d'une variable qui va s'incrémenter à chaque fois que l'un des capteurs ultrasons trouvera un mur à une distance inférieur à 20cm.

Nous avons fait en sorte de bloquer l'incrémentation tant que le robot n'est pas sorti des 20cm de distance du mur gênant.

f) Calcul du nombre de tour de roue et de la distance parcourue

```
//------------------- Nombre de tour de chaque Roue -------------------
void calcul_tour_roue(){
    newLeft = knobLeft.read();  //variable globale "newLeft" prend la valeur retournée par l'encodeur gauche
    newRight = knobRight.read();  //variable globale "newRight" prend la valeur retournée par l'encodeur droit
    if( newLeft != positionLeft || newRight != positionRight ) {
        positionLeft = newLeft / 2500;  //2500 est la valeur retournée par l'encodeur de newLeft/newRight pour 1 tour de roue
        positionRight = newRight / 2500;
    }
    distance_parcours = (positionLeft + positionRight)* CIRC_ROUE / 2;
}
```

Figure 11 : fonction calcul nombre tour de roue + distance parcourue

Dans un premier temps, l'idée est de compter le nombre de tour de chacune des roues. Celles-ci effectuant un nombre de tour différent selon la vitesse de chacune d'elle. Pour parvenir à cela, deux variables *newLeft* et *newRight* réceptionne les valeurs des encodeurs, placés dans la continuité des moteurs afin de détecter leurs rotations. Finalement convertit par un rapport ces détecter de position de la roue reviennent à calculer le nombre de tour. C'est ce qu'on effectue dans la condition *if{…}* les valeurs de *newLeft et de newRight* qui ne sont pas remises à zéro (sauf commande forcée) sont divisées par 2500. En effet, pour un tour de roue, l'encodeur compte 2500 intervalles. Ce calcul est affecté à affecté à *positionLeft* ou *postionRight* renvoyant le nombre de tour pour chacune des deux roues.

AN : Un degré donne 6.94 unité encodeur.

Dans un second temps, nous calculons la distance en additionnant les nombres de tour de chaque roue, nous multiplions ce résultat par la circonférence des deux roues roue. On obtient donc une valeur très proche de la distance réelle parcourue, que l'on renvoie à la variable *distance parcourue* affiché ultérieurement par la fonction d'affichage.

g) Affichage écran

```
//------------------- AFFICHAGE ECRAN L'ECRAN -------------------------------
void affichage_LCD(){
//Gestion des changements d'écran
  if (joystick_X > 700){ //GESTION CHANGEMENT D'ÉCRAN QUAND JOYSTICK VIRE A GAUCHE
      ecran = ecran + 1;
      lcd.clear();
      _delay_ms(200);
  }
   if (joystick_X < 400){  //GESTION CHANGEMENT D'ÉCRAN QUAND JOYSTICK VIRE A DROITE
      if(ecran == 0){
          lcd.clear();
          ecran = 6;
          _delay_ms(200);
      }
      else{
          ecran = ecran - 1;
          lcd.clear();
          _delay_ms(200);
      }
  }
```

Figure 12 : fonction gestion de l'affichage

Cette première partie de la fonction permet de gérer le changement de l'écran avec joystick. Un décalage vers la droite ou la gauche va permettre le changement en incrémentant ou décrémentant la variable « ecran » et ainsi l'écran affiché.

Chaque valeur d' « ecran » permet d'afficher une information différente comme par exemple l'état de marche, le chronomètre…

```
//Séléction de l'écran selon la valeur de la variable 'ecran'
  switch (ecran){
      case 0:
          lcd.setCursor(0,0);
          lcd.print("ETAT DU ROBOT :");
          lcd.setCursor(0,1);
          if(start == 1){
              lcd.print("Marche");
          }
          else{
              lcd.print("Arret ");
          }
          break;
      case 1:
          lcd.setCursor(0,0);
          lcd.print("VITESSE MAX :");
          lcd.setCursor(0,1);
          lcd.print("R :");
          lcd.print(vit_max_droite);
          lcd.setCursor(9,1);
          lcd.print("L :");
          lcd.print(vit_max_gauche);
          break;
```

Figure 13 : écriture des deux premiers ecrans par switch case

Ci-dessus une partie du switch qui gère la gestion de l'écran.

```
default :
    ecran = 0;
    break;
```

Figure 14 : valeur par défaut d' "ecran"

Si un nombre affecter à la variable est différent de 0 à 6, celle-ci se réinitialisera directement à 0.

```
if(ecran == 0){
    lcd.clear();
    ecran = 6;
    _delay_ms(200);
}
```

Figure 15 : gestion d' "ecran"

En revanche on peut aussi aller du premier au dernier écran.

C. Conclusion

Notre robot parvient à aller jusqu'à la fin du parcours sans rencontrer de problème. Au niveau du planning nous suivons bien ce qui était prévu à l'original, dans la prochaine période il nous est demandé d'effectuer des tests pour vérifier que le cahier des charges est bien respecté, essayer avec des positions de départs différentes...

Actuellement avec un temps de 1 minute et 16 secondes pour faire le trajet, le fait d'avoir une vitesse maximale diminué augmente nettement notre temps.

D. Bibliographie

« Afficheur LCD RGB Grove 104030001 ». https://www.generationrobots.com/fr/402442-afficheur-lcd-rgb-grove.html (17 octobre 2019).

« Batterie Li-Ion 7.4v 2200mAh (2S2P) avec PCM ». https://www.generationrobots.com/fr/403175-batterie-li-ion-74v-2200mah-2s2p-avec-pcm.html (17 octobre 2019).