Programme

#include <DRV8835MotorShield.h>

#include <hcsr04.h>

Le code du Robot inclut deux bibliothèques :

- L’une pour utiliser le driver des deux moteurs.

- L’autre pour utiliser le capteur de distance ultrason.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define MODE\_PIN 11

#define TRIG\_PIN 2

#define ECHO\_PIN 3

#define LED\_C 4 //JAUNE

#define LED\_P 5 //BLEU

#define BOUTON 6

#define SOL 12

#define SENS\_DPIN 0 //Droite

#define SENS\_GPIN 1 //Gauche

#define SENS\_CPIN 2 //Croisement

#define SENS\_PDPIN 3 //Palet Droite

#define SENS\_PGPIN 4 //Palet Gauche

Tous les pins du Robot sont nommés ici.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#define SEUIL 200

#define SEUILBLANC 200

#define SEUILNOIR 600

Les valeurs de seuil pour les capteurs infrarouges définissent comment est reconnu une ligne noire.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

void LedCheck() {

 if (SEUIL > analogRead(SENS\_DPIN)) digitalWrite(LED\_C, HIGH);

 else digitalWrite(LED\_C, LOW);

 while (**Serial**.available()) {

   delay(3);

   char c = **Serial**.read();

   readString += c;

 }

 if (readString.length() > 0) {

**Serial**.println(readString);

   if (readString == "s") {

     motors.setM2Speed(0);

     motors.setM1Speed(0);

     digitalWrite(LED\_C, LOW);

     digitalWrite(LED\_P, LOW);

     digitalWrite(BOUTON, HIGH);

     digitalWrite(SOL, HIGH);

     digitalWrite(MODE\_PIN, HIGH);

     asm volatile ("jmp 0");

   }

   readString = "";

 }

 int a = hcsr04.distanceInMillimeters();

 delay(1);

 countB++;

 if (39 > a) {

   digitalWrite(LED\_P, HIGH);

 }

 else {

   digitalWrite(LED\_P, LOW);

 }

 if (countB >= 400) {

   if (39 >= a && tir == false) **Serial**.println("d");

   if (39 < a && tir == false) **Serial**.println("n");

   if (tir == true) {

**Serial**.println("t");

     countT++;

   }

   if (countT >= 5) {

     countT = 0;

     tir == false;

   }

   countB = 0;

 }

}

La fonction LedCheck() est une sous fonction qui se trouve dans la totalité des boucles du programme. Elle dure approximativement 1 milliseconde + le reste des exécutions. Elle servait dans un premier temps à faire des vérifications en dirigeant des LED. Finalement elle permet de communiquer avec l’application Bluetooth, ainsi que de stopper le programme à la fin du temps.

Il existe plusieurs petites boucles avec paramètre de temps pour faire avancer les roues de toutes les manières possibles ;

GoUp(time) : Deux roues vers l’avant

GoDown(time) : Deux roues vers l’arrière

GoLeft(time) : Roue gauche vers l’arrière, roue droite vers l’avant

GoRight(time) : …… etc

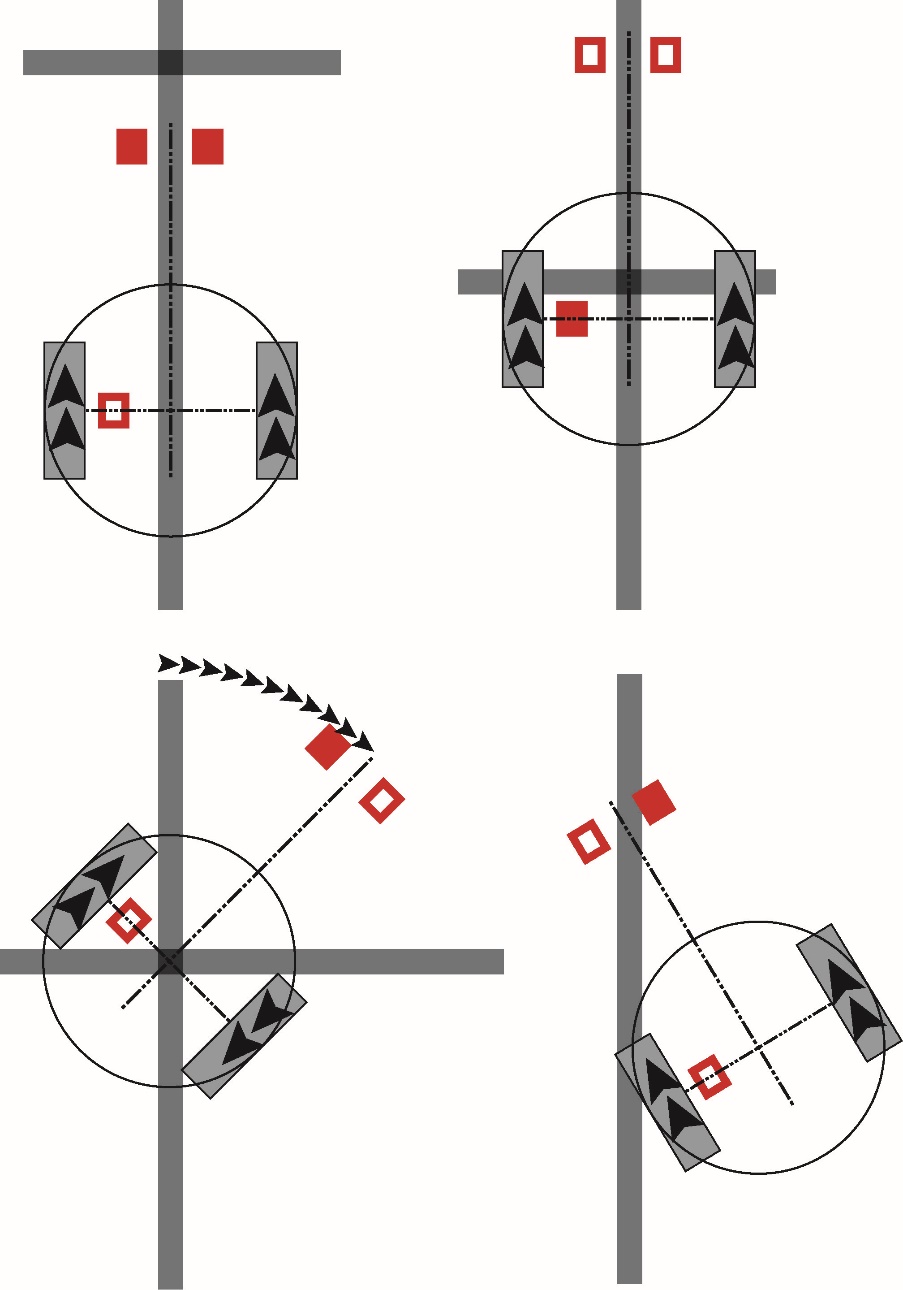
Il existe des fonctions plus complexes servant à diriger le robot par rapport aux lignes noires et croisement.

FIG.3

FIG.4

FIG.2

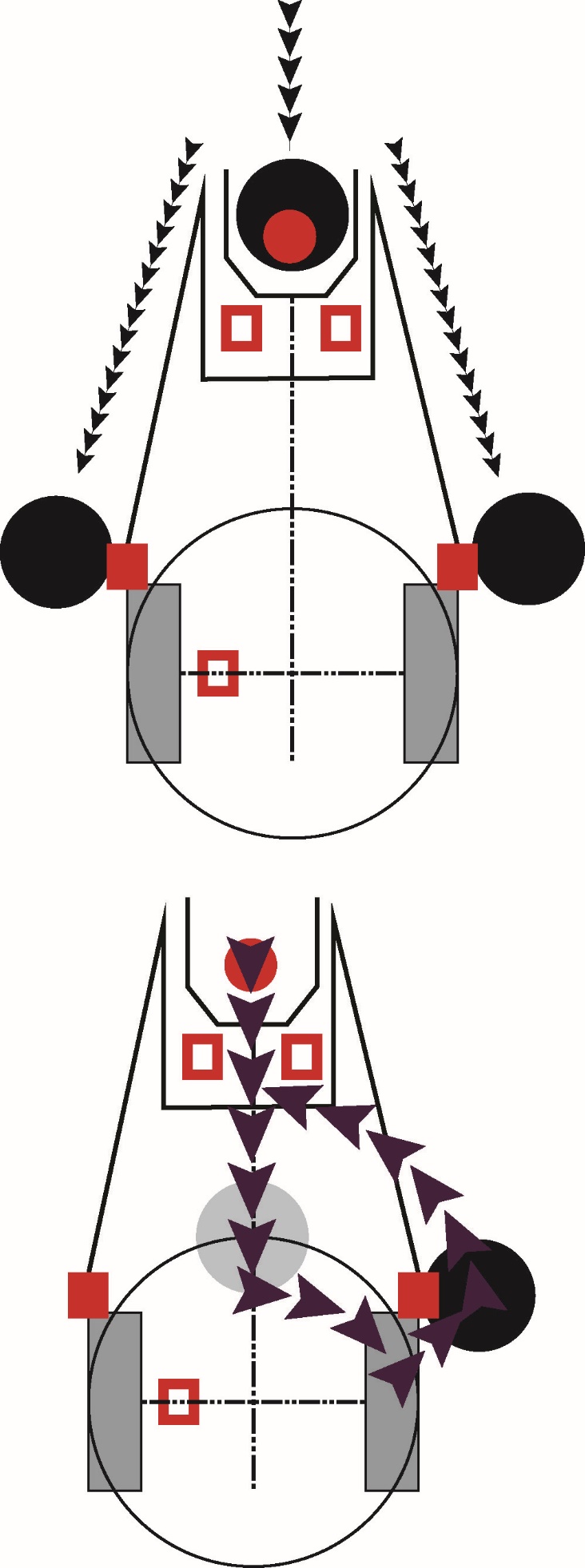
FIG.1

La figure 1 montre la fonction GoToCross(). Ces capteurs infrarouges servent à suivre la ligne avec une légère régulation. Dès que les deux sont sur une ligne noire, la boucle se stoppe.

Pour la figure 2, c’est le capteur près du centre de rotation du robot qui termine la boucle, la fonction est Center().

En figure 3, on tourne jusqu’à la prochaine ligne pour effectuer un virage, ce sont deux fonctions appelés CrossLeft() et CrossRight().

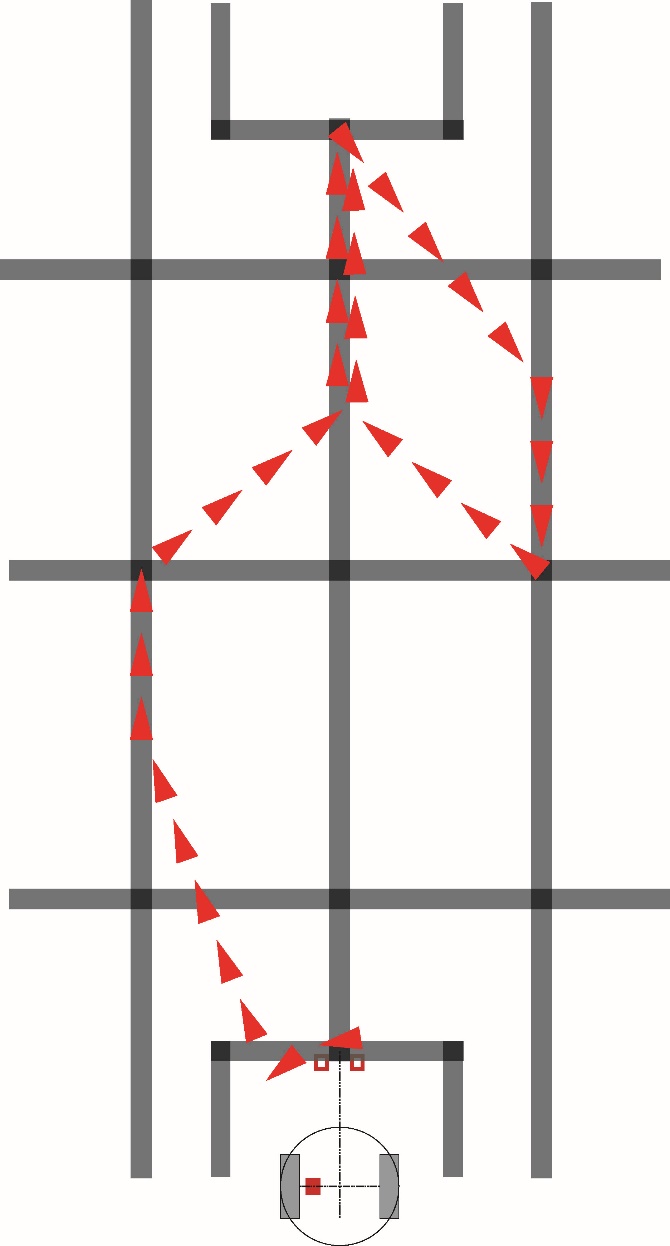
La figure 4 illustre d’autres fonctions permettant de rejoindre une ligne, le fait d’avoir deux capteurs suiveurs de ligne est dans ce cas très avantageux.

Pour le repérage des palets, on utilise le capteur de distance pour avoir l’information de la présence du palet dans la réserve de tir.

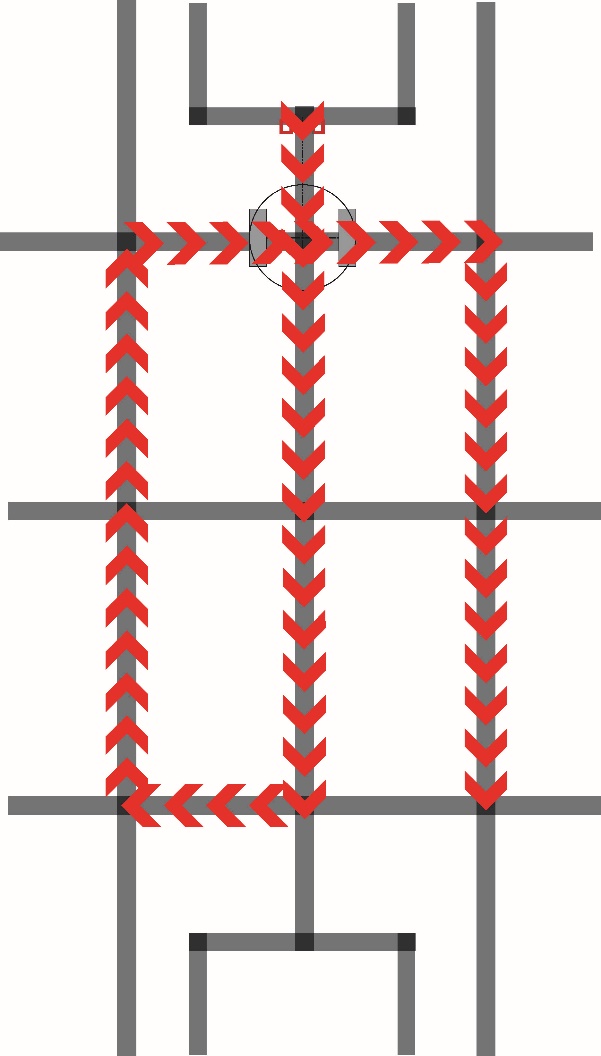
L’avancée du robot permet d’écarter des palets. Sur les côtés du robot, des languettes rouges sont rabattues par la présence d’un palet. La présence de ce dernier est remarquée par un capteur infrarouge .

Après la détection, une manœuvre purement séquentielle peut se déclencher (TakePRight(),TakePLeft()).

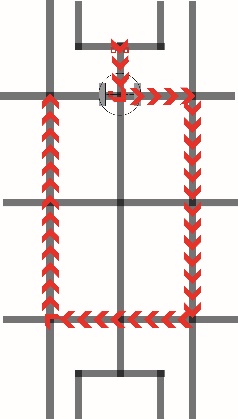
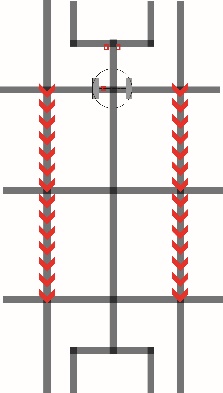
Un recul est préféré car il s’agit d’avoir une position a peu près identique au moment de la détection et à la fin de la manœuvre afin que la position du robot ne soit pas perdue dans le programme.

Le début de parcours du Robot est séquentiel, enchainant des virages, rejoindre une ligne, poursuite de ligne, centrage, etc…

Les deux premiers palets dont la position est connue sont cherchés et tirés dans le but.

Une fois en position de tir après avoir tiré le deuxième palet, une boucle de recherche se déclenche. On suit les longues lignes noires verticales. Si une grande ne signale aucun palet ont change de parcours de recherche.

Par exemple, la figure à gauche ci-dessous est le parcours suivis dans le cas où la ligne centrale est vide. Celle de droite montre deux parcours restants en fonctions de celle qui est vide.



Les parcours ont pour point commun de démarrer par la position de tir. A chaque fois qu’un palet est trouvé, le robot prend un chemin en fonction de la figure ci-contre.

La position du robot est actualisé dans le programme par séquence et sert à faciliter l’écriture du code.

Dans cette cartographie, le robot vat vers le croisement 9.

L’information de l’orientation du robot est donnée par le croisement d’où il vient. Le robot est ici en position 69. Il ira en 98 puis en 85, puis en 52 avant d’aller tirer.

