PROJET IMA4 SA-SC 2015-2016

Département Informatique – Microélectronique - Automatique

Robot Autonome pour la cartographie

Réalisation : Alexandre DESCAMD - Pierre MICHEL

Encadrants : Alexandre BOE - Xavier REDON - Thomas VANTROYS

Avec la collaboration de Thierry FLAMEN

Sommaire

Introduction 3

I. Présentation du projet 4

1. Description du projet 4
2. Choix techniques 4

II. Autonomie du robot 6

1. Gestion des déplacements et des obstacles 6
2. Pilotage des moteurs 8
   2.1. Le pont en H 8
   2.2. Le PCB 8

III. Mesures électromagnétiques 10

1. Wireless Tools 10
2. Traitement des données RSSI 10

IV. Asservissement du robot 12

1. Asservissement en vitesse 12
2. Calibrage en position 13

V. Bilan 15

1. Difficultés rencontrées 15
2. Limites du projet et évolution 15

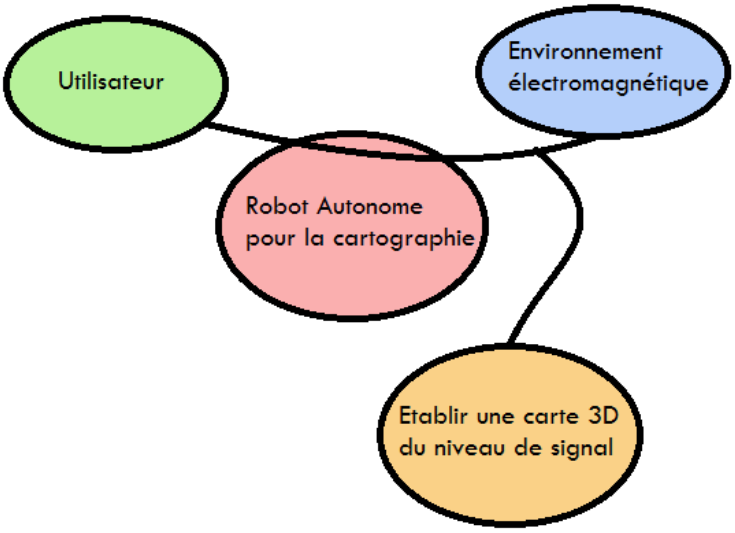
Conclusion 16

Introduction

Notre projet se situe au cœur d'une problématique de localisation indoor. En effet, c'est un enjeu de notre époque, et il est plutôt complexe de répondre à cet objectif par l'utilisation du GPS. C'est pourquoi nous avons tenté de répondre à ce problème par l'utilisation d'un robot autonome capable de cartographier un environnement électromagnétique.

Les applications sont nombreuses et relèvent des domaines médical, militaire ou encore commercial. En effet, il peut être intéressant de mesurer le taux d'exposition d'un patient à certaines ondes, ou encore de localiser un potentiel client dans un centre commercial afin de lui indiquer les magasins qui peuvent l'intéresser.

Nous nous sommes donc orientés, à partir du développement mécanique réalisé par des élèves de CM5, vers un robot autonome dans ses mouvements (et aussi dans sa gestion de l'énergie). Celui-ci est capable de mesurer le niveau de qualité d'un signal de type Wi-Fi et de sauvegarder et stocker l'ensemble des mesures qu'il réalise. A partir de ces mesures, il sera alors possible à l'aide d'un logiciel, d'établir une cartographie électromagnétique 3D de ce signal. Cette carte permettra alors, par simple comparaison, d'obtenir sa position en relevant le niveau du signal à la position de l'utilisateur.



I. Présentation du projet

1. Description du projet

La qualité d'un signal Wi-Fi n'est pas la même en tout point de l'espace. En effet, elle dépend des modifications apportées (ajout, retrait, ou déplacement) aux bornes Wi-Fi ou encore du mobilier présent dans l'espace, et de tout système pouvant interférer avec le signal. Il est donc indispensable pour le robot d'être autonome dans la gestion de son énergie (réduction du temps nécessaire à la cartographie d'un bâtiment), dans ses déplacements (éviter les obstacles permanents ou temporaires), et dans la prise des mesures de qualité (réalisation et stockage de nombreuses mesures).

Afin de ne pas effectuer plusieurs fois les mêmes mesures, il est important de connaître « en temps réel » la position du robot. Celui-ci effectue donc un calibrage régulier de sa position par lecture de tags optiques placés dans son environnement. Un asservissement des moteurs du robot permet de corriger une potentielle dérive du robot pendant ses déplacements.

2. Choix techniques

Pour ce qui est du matériel, l'architecture principale du robot repose sur le travail coopératif d'une Raspberry Pi et d'un Arduino Uno. Ils communiquent via le protocole I2C. L'Arduino est responsable de la commande des moteurs et des capteurs. La Raspberry s'occupe de la partie lecture de tags via une caméra et des mesures RSSI via plusieurs clés Wi-Fi.

Un capteur à ultrason, placé sur un servomoteur capable de le faire pivoter dans toutes les directions, permet au robot de détecter les obstacles sur son chemin.

Deux moteurs à courant continu (MFA 540/1) et des pilotes de moteurs (MC34931EK) permettent, sous les ordres de l'Arduino, la mise en mouvement du robot.

L'asservissement en vitesse repose sur l'utilisation de roues crantées sur lesquelles sont fixées des fourches optiques. Une consigne de vitesse est alors imposée au robot et corrigée.

La caméra, sous le contrôle de la Raspberry, réalise un traitement d'image qui permet d'isoler une couleur afin de reconnaître un motif/une couleur pré-choisi(e). Le robot utilise cette méthode de détection pour se repositionner.

Plusieurs clés Wi-Fi sont placées à des hauteurs variables afin de relever le niveau du signal Wifi.

Enfin, le logiciel Autocad Map 3D permet d'établir une carte électromagnétique du signal Wifi.

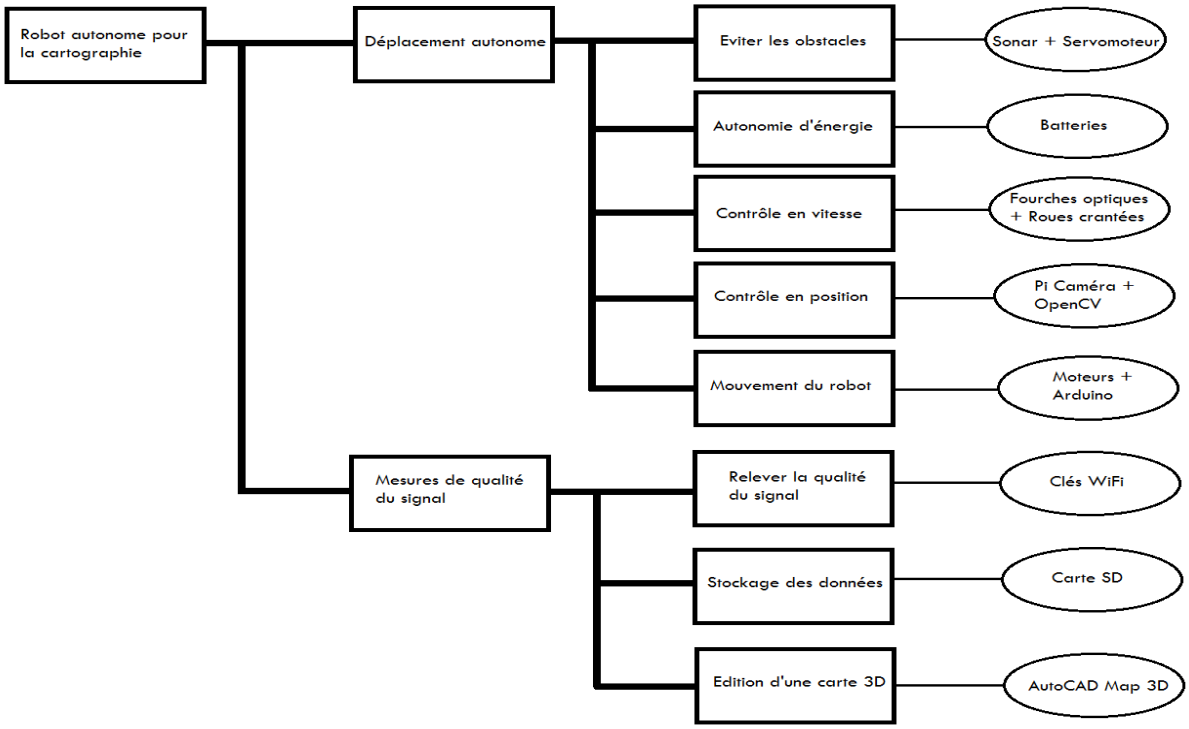
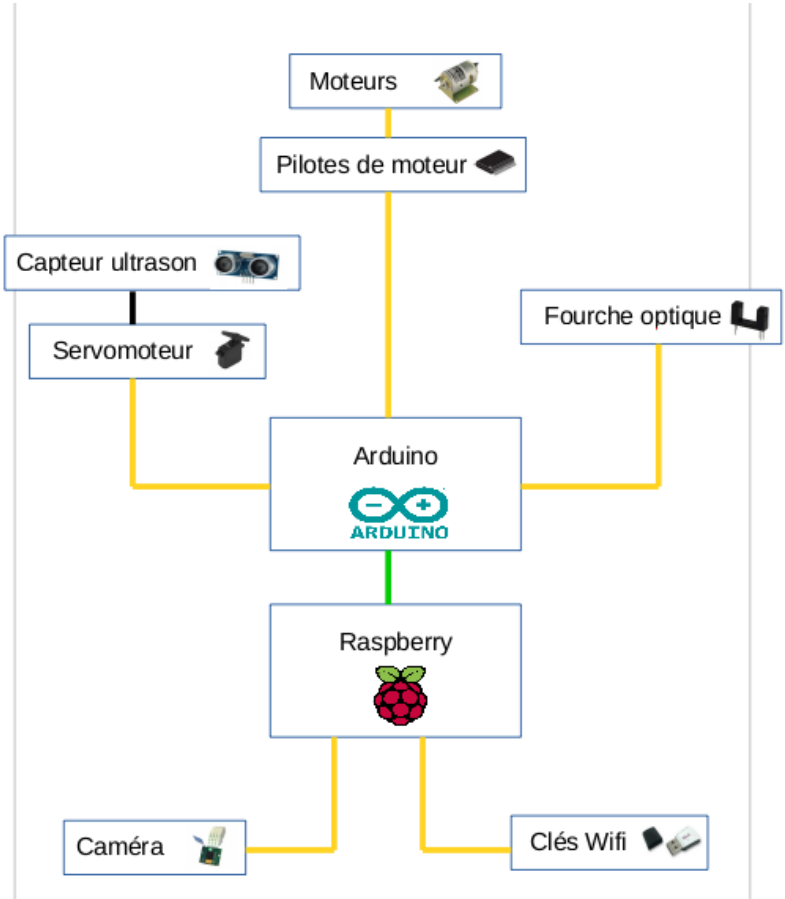


Diagramme SADT



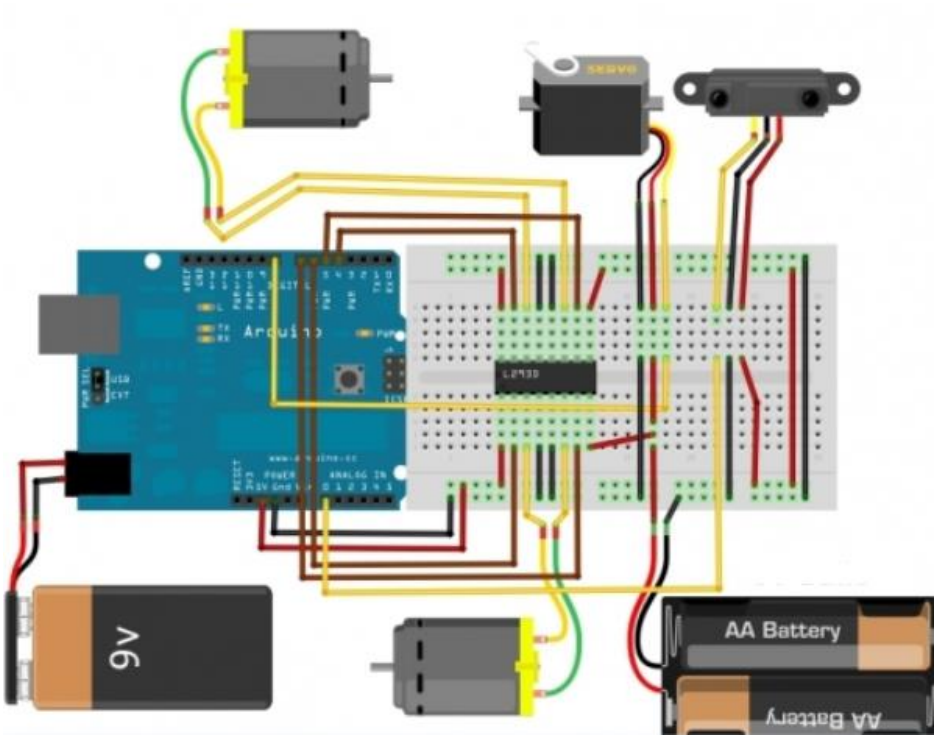
Architecture matérielle

II. Autonomie du robot

1. Gestion des déplacements et des obstacles

Pour des questions de facilités de programmation et d'implémentation, le choix s'est porté sur l'utilisation de la carte Arduino.

Les différents composants nécessaires à la mise en mouvement du robot sont contrôlés par l'Arduino. Ceux-ci sont reliés aux différentes broches de l'Arduino de la manière suivante :



Gestions des entrées/sorties des différents composants

Le robot doit être autonome en termes de déplacement, c'est-à-dire qu'il doit être capable de circuler librement dans son environnement, en tenant compte des possibles obstacles qui pourraient lui barrer la route.

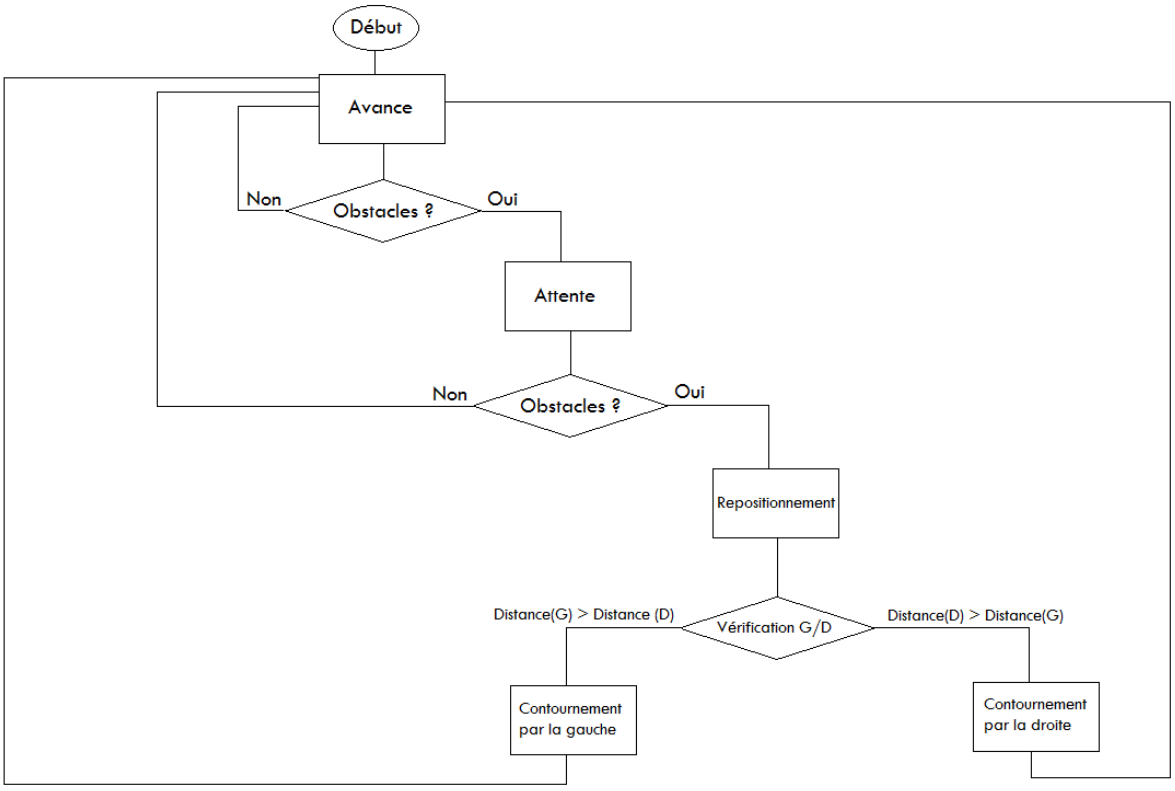
Concernant la détection d'obstacle, notre choix s'est tourné vers l'utilisation d'un capteur ultrason au détriment des capteurs lasers ou infrarouges pour des questions de coût et de facilité d'utilisation. Le capteur ultrason est monté sur un servomoteur placé à l'avant du robot. L'utilisation du servomoteur permet alors au capteur ultrason de relever différents valeurs de distance selon toute les directions et ainsi de limiter l'utilisation de capteur ultrason au nombre de un.

La gestion des obstacles temporaires a été prise en compte, ainsi le passage bref d'une personne devant le robot n'aura pas la même incidence qu'un réel obstacle. Ce cas est géré par l'utilisation d'une temporisation lors d'une détection d'obstacle trop proche.

L'un des problèmes rencontrés concernant le capteur ultrason est le manque de précision durant certaine mesure. Pour remédier à ce problème, nous choisissons d'opérer sur des valeurs moyennes de mesures réalisées dans un cours intervalle de temps.

Concernant le servomoteur, ses défauts d'asservissement en position le faisant tanguer autour de position demandée par l'utilisateur ont été solutionnés en le faisant pivoter pas à pas jusqu'à atteindre la position demandée.

Un algorithme de déplacement du robot a été établit et programmé sur l'Arduino via le logiciel Arduino IDE selon la figure suivante :



Algorithme de déplacement du robot

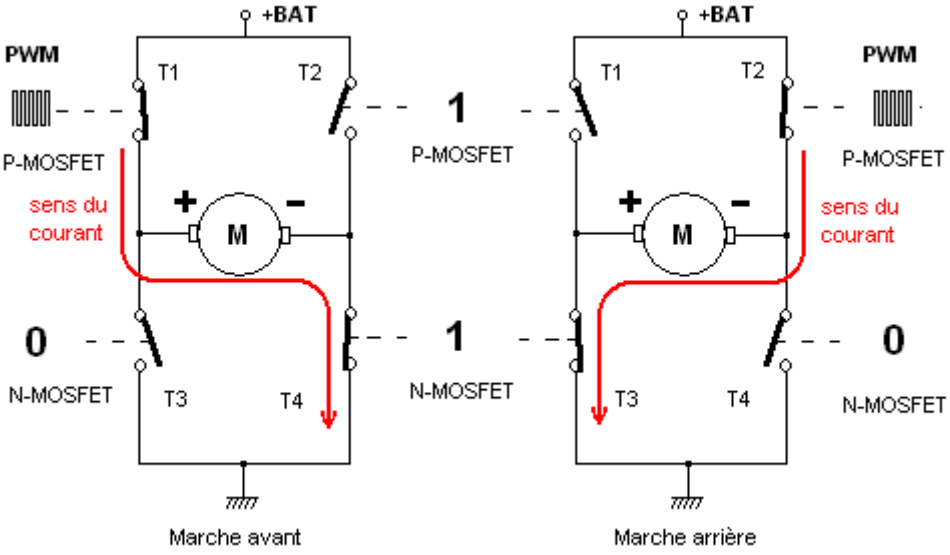
L'étape de repositionnement correspond à un recul du robot pouvant permettre sa future rotation sans heurter l'obstacle se trouvant devant lui.

L'étape de vérification consiste à relever différents valeurs de distance aussi bien à droite qu'à gauche du robot. Selon les valeurs relevées, le robot effectuera une rotation à droite ou à gauche et contournera l'objet. Lorsque le capteur ne détectera plus la présence de l'objet, le robot se remettra dans son sens initial.

2. Pilotage des moteurs

2.1. Le pont en H

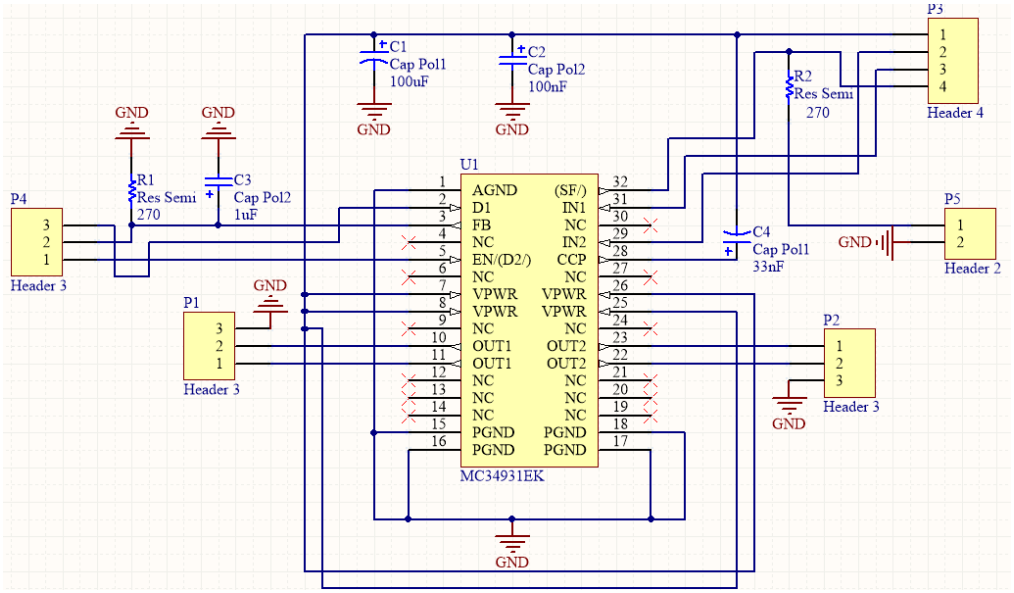
Afin de pouvoir piloter les deux moteurs du robot selon les deux sens de rotation, l'utilisation d'un pont en H a donc été nécessaire. Le pont en H est une structure électronique composée de commutateurs. Selon leur état, les deux sens de rotation des moteurs peuvent ainsi être contrôlés. Vu les besoins en intensité que nécessitaient les deux MCC, le choix s'est porté sur les puces pilote de moteurs MC34931EK.



2.2. Le PCB

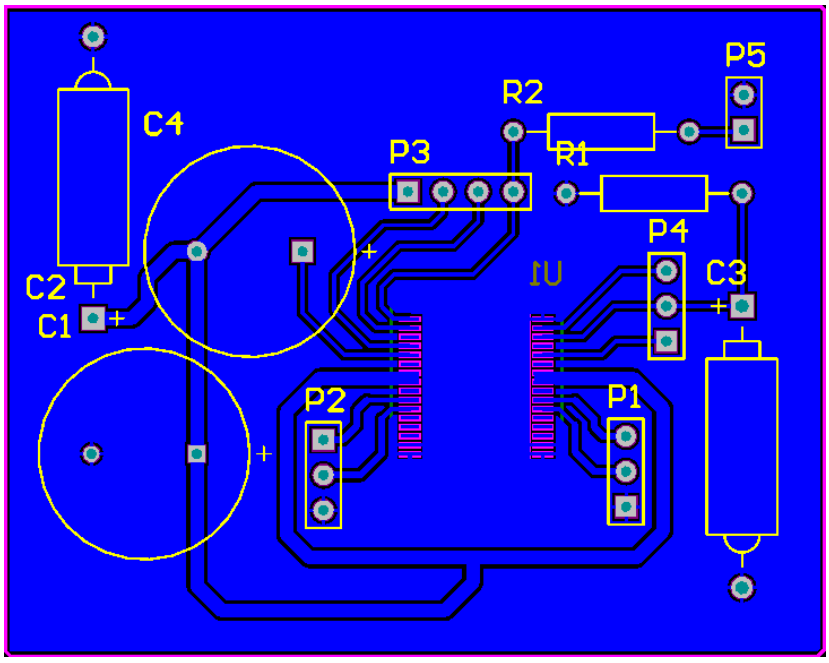
Pour la réalisation des PCB, nous avons utilisé le logiciel Altium Designer qui est assez complexe et peu intuitif à la prise en main. Plusieurs étapes ont été nécessaires à cette réalisation, notamment une première étape primordiale qui consiste en la création d'emprunte pour les composants utilisés. C'est pourquoi nous avons dû établir plusieurs librairies contenant les informations sur chaque composant (taille, pins, empruntes, …).

Une fois ces librairies créées et fonctionnelles, nous avons pu réaliser un schéma électronique des pilotes de moteurs et de leurs interactions avec d'autres composants de type capacité ou résistance. Ce schéma reprend le fonctionnement global présenté dans la datasheet.



Schematic des pilotes de moteurs

A partir de ce schéma, nous avons pu établir le PCB en lui-même, qui correspond à notre future carte électronique. Il ne restait alors plus qu'à effectuer le routage, l'exportation du fichier en Gerber (pour la gravure), la gravure et les soudures.



PCB après routage

III. Mesures électromagnétiques

1. Wireless Tools

Le niveau d'un signal Wifi se mesure grâce à des valeurs de puissance en réception appelée RSSI (Received Signal Strenght Indication), cette valeur varie usuellement entre -30 décibel et -120 décibel.

Afin de pouvoir récupérer ces données, nous disposons de trois clés wifi connectées à la Raspberry. C'est elle qui va permettre le traitement des données. Les commandes nécessaires ont ensuite été réalisées afin de pouvoir connecter les clés au réseau wifi, en l'occurrence celui de Polytech Lille.

Nous avons donc utilisé le logiciel Wireless Tools, un outil en libre accès, via la Raspberry. Cet outil permet entre autres d'obtenir différentes informations en provenant de signaux wifi, dont la valeur RSSI du signal. Voici un aperçu des informations obtenues avec la commande iwconfig sur un terminal Linux.

```
root@raspberrypi:/home/pi/rssi# iwconfig
wlan0     IEEE 802.11bgn  ESSID:"PolytechLille"
          Mode:Managed  Frequency:2.412 GHz  Access Point: 00:19:07:C5:
          Bit Rate=12 Mb/s   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=55/70  Signal level=-55 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:3   Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.
```

2. Traitement des données RSSI

Le traitement des valeurs en décibel, rattachées à la mention ''Signal level'', s'est fait dans un premier temps à l'aide d'un script Shell. Celui-ci copie les informations issues de la commande iwconfig dans un fichier texte. Les opérations grep et cut permettent ensuite d'extraire la valeur et de l'afficher sur le terminal.

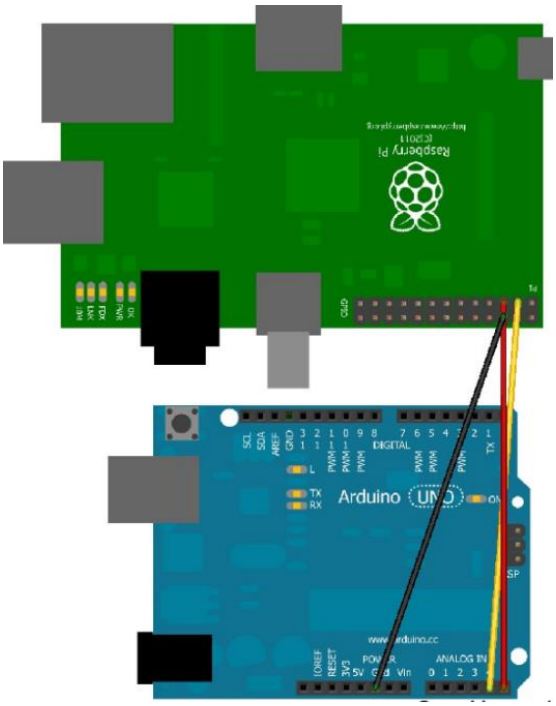
Voici le script Shell utilisé pour réaliser cette opération.

```
#!/bin/bash
# extract.sh
# Releve rssi

iwconfig > rssi.txt
fichier=rssi.txt
while read Ligne
do
echo $Ligne | grep "Quality" | cut -c 33-35
done < $fichier

exit 0
```

L'idée était ensuite d'utiliser un code c faisant appel à ce script afin de pouvoir manipuler les valeurs RSSI et les enregistrer dans une base de données. Ce point n'a pas été abouti par manque de temps et des erreurs de programmation non résolues.

L'hypothèse d'utiliser directement les codes sources de Wireless Tools a ensuite été envisagée. Cependant, l'utilisation de lourdes bibliothèques et de drivers non accessibles a freiné notre progression. Dans un souci de priorités de tâches à réaliser, nous avons choisi de laisser de côté cette partie et d'y revenir par la suite, chose malheureusement non aboutie par manque de temps et dû aux complications matérielles.

Pourtant nos idées étaient claires. Une liason I2C a donc été réalisée. La liaison I2C (Inter Integrated Circuit) permet la communication entre la Raspberry et l'Arduino en mode série. Trois fils sont ainsi nécessaires pour établir une relation de maître-esclave entre nos deux composants électroniques. La Raspberry en tant que maître, ordonnerait à l'Arduino d'effectuer son programme implanté permettant la mise en mouvement du robot. Ainsi, en fonction de la position du robot, le traitement des données RSSI serait alors effectué.



Afin de pouvoir établir une cartographie électromagnétique, l'utilisation du logiciel Autocad Map 3D entre en jeux.

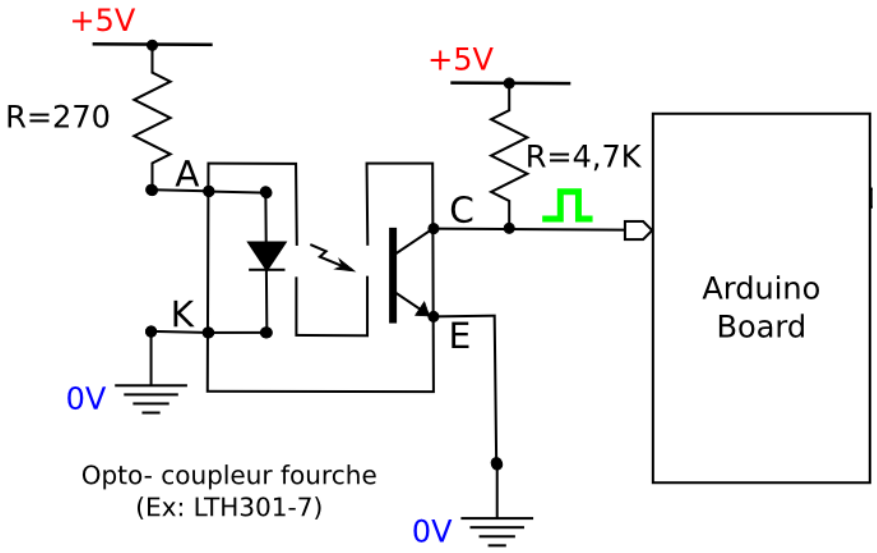
IV. Asservissement du robot

1. Asservissement en vitesse

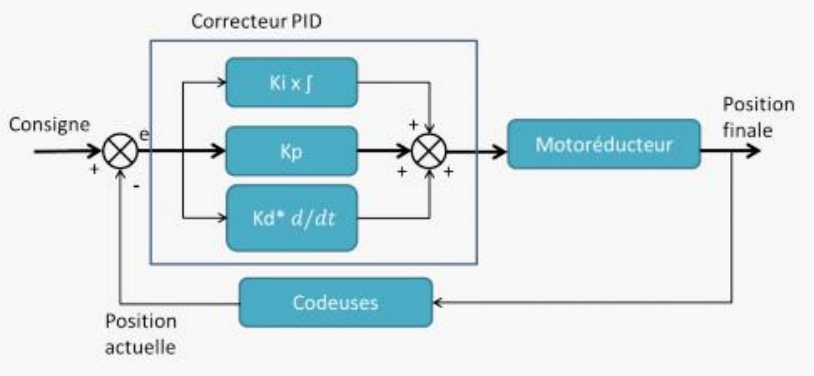
Dans l'optique de réel contrôle de déplacement du robot, la question d'asservir en vitesse les moteurs s'est imposée à nous. Les deux moteurs utilisés pour assurer le déplacement du robot se doivent d'être contrôlés. Si un moteur tourne plus vite que son homologue, le robot n'effectuera pas les déplacements ordonnés de manière précise. Ainsi nous avons besoin de savoir à quelle vitesse tournent réellement les deux moteurs. Pour cela, l'utilisation de codeurs incrémentaux s'impose. Pour notre projet, nous avons choisis de nous tourner vers l'utilisation de fourche optique.

Les fourches optiques sont des codeurs incrémentaux de type optique

. Elles possèdent un émetteur et un récepteur. La présence d'obstacle entre ces derniers génère un changement d'état, une impulsion. L'utilisation de roues crantés reliées aux arbres des moteurs, associées à une fourche optique permet de connaitre après traitement la véritable vitesse du moteur. A chaque passage d'obstacle entre la fourche optique, une impulsion est envoyée à l'Arduino.



L'asservissement en vitesse repose sur l'utilisation de correcteur. Avec l'utilisation d'un correcteur de type PID, c'est-à-dire à effet proportionnel, intégral et dérivé, on peut modéliser notre système de la manière suivante :



Dû au fait que nous n'avons pas réussi à mettre en mouvement le robot, cette partie n'a pas pu être testé. Le tout reposait donc sur le comptage du nombre d'impulsions que relevaient les deux fourches optiques et ainsi en déduire la vitesse réelle des deux moteurs. Grâce à des tests par tâtonnement, les valeurs des différents paramètres du PID (Ki, Kp et Kd) auraient été déterminées, et la vitesse finale suivrait la consigne d'entrée.

2. Calibrage en position

Le robot étant amené à dériver naturellement lors de ses déplacements, malgré l'asservissement en vitesse de ses moteurs, nous avons dû établir une stratégie pour repositionner celui-ci pendant ses déplacements.

Le choix « imposé » mais aussi le plus judicieux était de procéder par lecture optique de tags visuels par exemple. De ce fait, le robot est muni d'une caméra (webcam USB ou la Pi Caméra du module Raspberry) grâce à laquelle il va reconnaître des tags de différentes formes et de différentes couleurs afin de pouvoir se replacer correctement près de ces derniers.

Notre premier choix s'est porté vers ARToolKit qui semblait plutôt efficace et simple d'utilisation. Cependant, nous avons très exclu cette possibilité, étant donné l'incompatibilité entre ARToolKit et la Raspberry. C'est pourquoi nous nous sommes alors tournés vers OpenCV (bibliothèque de traitement d'images en open source) dont les fonctions de traitement d'images sont à la fois plus complète, mais plus complexe à maitriser.

Après avoir installé la bibliothèque OpenCV, nous avons essayé d'accéder au flux vidéo d'une webcam USB quelconque. Une fois l'image obtenue en temps réel, nous avons effectué une segmentation de cette image, ou plus précisément une binarisation de cette image. La binarisation va permettre de séparer les pixels de l'image en deux classes distinctes (travaux réalisés en TP de traitement d'images à Polytech) : une première classe, les pixels blancs, correspondant à la couleur que l'on veut isoler, et une deuxième classe, les pixels noirs, correspondant à toutes les autres couleurs de l'image.

Exemple de segmentation (utilisation d'un seuil)

Pour la segmentation, il est important de définir un seuil, cela va permettre de différencier les pixels qui vont devenir blanc, de ceux qui vont devenir noir. Pour procéder à cette segmentation, nous avons dû convertir l'image de base de la caméra qui est en RGB (Red Green Blue) en HSV (Hue Saturation Value). Des opérations morphologiques ont aussi été effectuées afin d'éliminer les pixels résiduels qui pourraient altérer la qualité de l'image final.

Une fois l'image finale obtenue, il est possible d'ordonner au robot soit de « chasser » cette couleur (color tracking) ou de se réorienter vers cette couleur en utilisant sa position. Cependant, cette dernière partie n'a pas été réalisée par manque de temps.

V. Bilan

1. Difficultés rencontrées

Ce projet s'est établi sur une durée d'un semestre et cela nous a fait prendre conscience que de nombreuses difficultés, notamment matérielles ou logicielles, peuvent survenir et qu'il faut être capable de proposer une solution adéquate afin de ne pas prendre trop de retard.

Premièrement, au niveau de la gestion des obstacles, nous avons rencontré quelques légères difficultés que nous avons très vite pu résoudre. Le manque de précision des mesures des capteurs a été compensé par des mesures moyennes, alors que le manque de stabilité dans la rotation du servomoteur a été compensé par une évolution pas-à-pas de celui-ci.

Pour ce qui est du contrôle des moteurs, il a été assez difficile de trouver un composant adéquat à nos besoins. De plus, la maitrise d'Altium et la réalisation d'un PCB nous ont causés de nombreuses difficultés et nous avons alors perdu énormément de temps à obtenir une solution convenable. C'est d'ailleurs ces problèmes qui nous ont empêchés de finaliser les tests nécessaires à la mise en mouvement du robot. Le fait d'avoir finaliser les PCB très tardivement relève également un défaut de stratégie de notre part.

La bibliothèque OpenCV est assez complexe à maîtriser, d'autant qu'il nous a fallu un peu de temps pour nous mettre à niveau en Python et C++ (la plupart des codes utilisés sur internet sont dans ces langages).

2. Limites du projet et évolution

Malgré la volonté d'autonomie totale du robot, celui-ci reste dépendant d'un scénario préétabli. Cette obligation de rester cantonné à ce scénario constitue une limite de ce projet. C'est pourquoi l'élaboration d'un scénario plus complet, prenant en compte un parcours de toutes les positions possibles d'un bâtiment, serait une évolution intéressante.

La conception mécanique actuelle du robot constitue dans un second temps une limite plus légère. En effet, le robot est assez imposant et il y a beaucoup d'espaces non utilisés. Une structure plus fine, permettant un choix de composants plus fins, plus légers et moins coûteux pourrait être plus judicieuse.

Il existe aussi quelques limites moins contraignantes actuellement telles que l'obligation de passer par un logiciel pour établir une carte électromagnétique 3D utilisable ou encore la capacité de stockage de la carte SD limitant ainsi le nombre de mesures.

La tâche de traitement d'images et celle de relevés de qualité du signal sont assez dépendantes de la qualité de la caméra et de la stabilité du signal Wi-Fi à une position donnée.

Conclusion

Ce projet constitue une expérience assez forte pour nous, autant sur le plan personnel que pédagogique. Le travail en autonomie, avec des contraintes matérielles, des contraintes de temps, et dans le respect d'un cahier des charges et d'un planning de travail est une vraie simulation de ce qui nous attend dans le cadre de l'entreprise. Nous avons également apprécié le fait de devoir se répartir les tâches afin de mener un travail coopératif efficace.

Nous sommes cependant déçus de ne pas avoir pu arriver à la fin du cahier des charges. Nous avions des objectifs ambitieux auxquels nous n'avons pas pu proposer de solutions convenables.

Nous avons par contre perfectionné nos connaissances personnelles sur des points théoriques vus en cours, et cette mise en pratique nous a été très bénéfique.

Les nombreuses difficultés et les nombreux ralentissements rencontrés nous ont permis de comprendre nos erreurs dans notre approche et dans notre stratégie de travail. Les réflexions apportées afin de proposer des solutions convenables sont une partie du projet que nous avons particulièrement apprécié.

Les possibilités d'optimisation du robot sont nombreuses et nous laissent alors optimistes quant à une réalisation aboutie du projet.

Annexes

Datasheet de la puce pilote de moteur

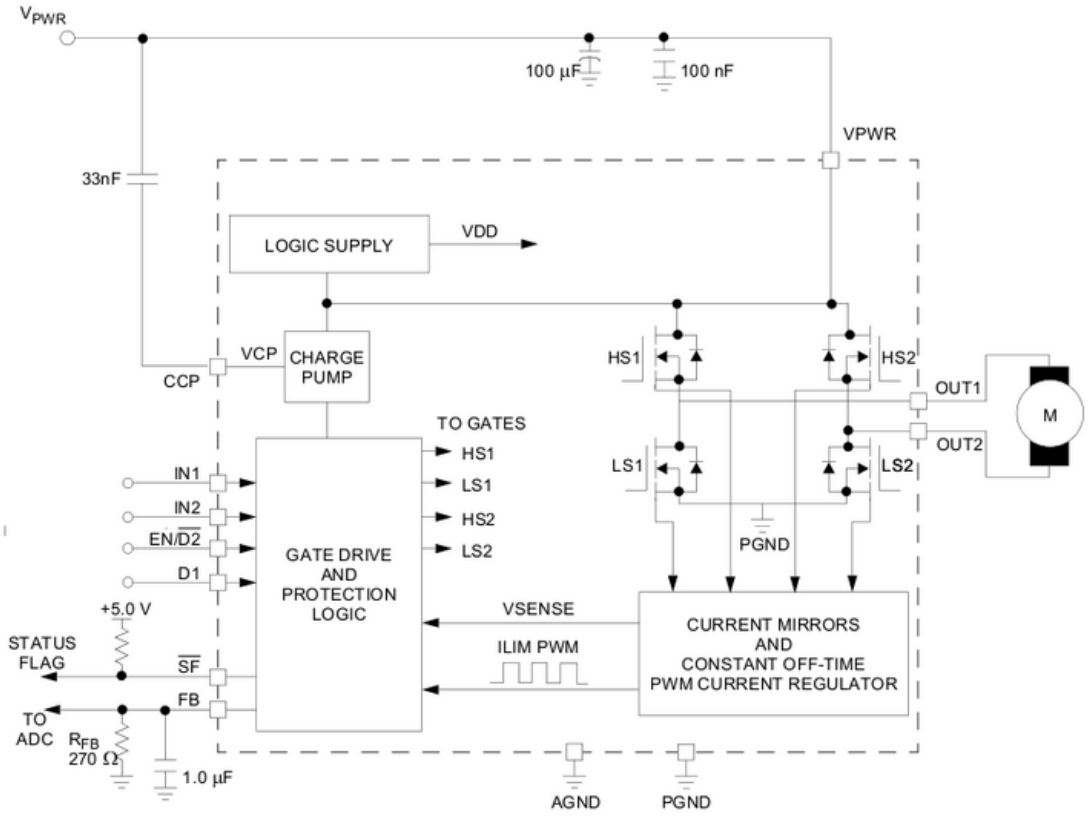


Figure 14. 34931 Typical Application Schematic

Code Arduino

```
#include <Servo.h>
#include <SimpleTimer.h>

Servo servo;
int trig = 12; //broche trig du sonar
int echo = 11;

int periode = 20000;

long lecture_echo;
long val;
long val_moy;
long val_left;
long val_right;
int etat;
int dir;
int pos;
bool avant = true;
long valright = 0;
long valleft = 0;
int motor11 = 2;
int motor12 = 4;
int motorpower1 = 5;
int motor21 = 7;
int motor22 = 8;
int motorpower2 = 10;
int motorspeed = 0;

int fourche1_in = ?;
int fourche2_in = ?;
int fourche1_out = ?;
int fourche2_out = ?;

const int frequence_echantillonage = 50;
const int rapport_reducteur = ?;
const int tick_par_tour_codeuse = ?;

float erreur_precedente = consigne_moteur; // (en tour/s)
float somme_erreur = 0;

```

```
40
41  Definition des constantes du correcteur PID
42  float kp = ?;            // Coefficient proportionnel
43  float ki = ?;           // Coefficient intégrateur
44  float kd = ?;           // Coefficient dérivateur
45  * /
46
47  void setup()
48  {
49    Serial.begin(9600);
50    servo.attach(9, 700, 2300); // servomoteur sur la broche 9 de l'arduino
51    pinMode(motor11, OUTPUT);
52    pinMode(motor21, OUTPUT);
53    pinMode(motor12, OUTPUT);
54    pinMode(motor21, OUTPUT);
55    pinMode(motorpower1, OUTPUT);
56    pinMode(motorpower2, OUTPUT);
57    servo.write(90); // servo en position centrale
58    pinMode(9, OUTPUT);
59    pinMode(fourche1_in, INPUT);
60    pinMode(fourche2_in, INPUT);
61    pinMode(fourche1_out, OUTPUT);
62    pinMode(fourche1_out, OUTPUT);
63    analogWrite(motorpower1, 0);
64    analogWrite(motorpower2, 0);
65    delay(1000); // delai pour attendre la fin de rotation du servo et moteur à l'arret
66    digitalWrite(9, LOW);
67    pinMode(trig, OUTPUT);
68    digitalWrite(trig, LOW);
69    pinMode(echo, INPUT);
70    attachInterrupt(0, compteur, CHANGE);
71    timer.setInterval(1000 / frequence_echantillonnage, asservissement);
72
73  }
74
```

```
76  void loop()
77  {
78    delay(1000);
79    if (avant == true) {
80      servo.write(90);
81      val_moy = distance_moyenne();
82
83      if (val_moy < 10)
84      {
85        delay(2000); // en cas d'obstacle temporaire
86        val_moy = distance_moyenne();
87        if (val_moy < 10) // si obstacle
88        {
89          Serial.print("obstacle\n");
90          etat = findroute();
91          if (etat == 1)
92          {
93            goleft();
94          }
95          if (etat == 2)
96          {
97            goright();
98          }
99          avant == true;
100       }
101       else
102         forward();
103       avant = false;
104     }
105     else
106       forward();
107     avant = false;
108   }
```

```
int distance_moyenne()
{
  int i;
  int val_i;
  int somme = 0;
  for (i = 0; i < 10; i += 1)
  {
    val_i = getDistance();
    delay(100);
    somme = somme + val_i;
  }
  val_moy = somme / 10;
  Serial.print("valeur moyenne:");
  Serial.println(val_moy);
  return val_moy;
}

int getDistance()
{
  digitalWrite(trig, HIGH);
  delayMicroseconds(10);
  digitalWrite(trig, LOW);
  lecture_echo = pulseIn(echo, HIGH);
  val = lecture_echo / 58;
  Serial.print("Distance : ");
  Serial.println(val);
  return val;
}


int lookleft()
{
  Serial.print("lookleft\n");
  leftposition();
  val_left = distance_moyenne();
  dir = 1;
  return val_left;
}
```

```
int lookright()
{
  Serial.print("lookright\n");
  rightposition();
  val_right = distance_moyenne();
  dir = 2;
  return val_right;
}

void centralposition()
{
  Serial.print("centralposition\n");
  //setAngle(90);
  if (dir == 2)
  {
    Serial.print(dir);
    for (pos = 0; pos <= 90; pos += 1) // retour en position centrale depuis la droite
    {
      servo.write(pos);
      delay(20);
    }
  }

  if (dir == 1)
  {
    Serial.print(dir);
    for (pos = 180; pos >= 90; pos -= 1) // retour en position centrale depuis la gauche
    {
      servo.write(pos);
      delay(20);
    }
  }
}
```

```
185  void rightposition()
186  {
187    Serial.print("rightposition\n");
188    for (pos = 90; pos >= 0; pos -= 1)
189    {
190      servo.write(pos);
191      delay(20);
192    }
193  }
194
195  void leftposition()
196  {
197    Serial.print("leftposition\n");
198    for (pos = 90; pos <= 179; pos += 1)
199    {
200      servo.write(pos);
201      delay(20);
202    }
203  }
204
205    void compteur()
206  {
207    tick_codeur++;
208  }
209
210
211  void asservissement()
212  {
213
214    int tick = tick_codeuse;
215    tick_codeuse = 0;
216
217    int frequence_codeuse = frequence_echantillonnage * tick;
218    float nb_tour_par_sec = (float)frequence_codeuse / (float)tick_par_tour_codeuse / (float)rapport_reducteur;
219    float erreur = consigne_moteur_nombre_tours_par_seconde - nb_tour_par_sec;
220    somme_erreur += erreur;
221    float delta_erreur = erreur - erreur_precedente;
222    erreur_precedente = erreur;
223
224    motorspeed = kp * erreur + ki * somme_erreur + kd * delta_erreur;
```

```
225
226     if (motorspeed < 0) motorspeed = 0;
227     else if (motorspeed > 255) motorspeed = 255;
228     forward(motorspeed);
229   }
230
231   void forward(int powerRate)
232   {
233     Serial.print("forward\n");
234     digitalWrite(motor11, HIGH);
235     digitalWrite(motor12, LOW);
236     digitalWrite(motor21, HIGH);
237     digitalWrite(motor22, LOW);
238     analogWrite(motorpower1, powerRate);
239     analogWrite(motorpower2, powerRate);
240   }
241
242   void backward()
243   {
244     Serial.print("backward\n");
245     digitalWrite(motor11, LOW);
246     digitalWrite(motor12, HIGH);
247     digitalWrite(motor21, LOW);
248     digitalWrite(motor22, HIGH);
249   }
250
251   void turnleft()
252   {
253     Serial.print("turnleft\n");
254     digitalWrite(motor11, HIGH);
255     digitalWrite(motor12, LOW);
256     digitalWrite(motor21, LOW);
257     digitalWrite(motor22, HIGH);
258     delay(1000);
259     //halt();
260   }
261
262   void turnright()
263   {
264     Serial.print("turnright\n");
265     digitalWrite(motor11, LOW);
266     digitalWrite(motor12, HIGH);
267     digitalWrite(motor21, HIGH);
268     digitalWrite(motor22, LOW);
269     delay(1000);
270     halt();
271   }
272
273   void halt()
274   {
275     Serial.print("halt\n");
276     digitalWrite(motor_pin11, LOW);
277     digitalWrite(motor_pin12, LOW);
278     digitalWrite(motor_pin21, LOW);
279     digitalWrite(motor_pin22, LOW);
280     delay(500); * /
281   }
282
283
284   int findroute()
285   {
286     halt();
287     Serial.print("findroute\n");
288     backward();
289     val_left = lookleft();
290     centralposition();
291     val_right = lookright();
292     centralposition();
293
294     if ( val_left > val_right )
295     {
296       turnleft();
297       etat = 1;
298       Serial.print(etat);
299     }
```

```
  else
  {
    turnright();
    etat = 2;
    Serial.print(etat);
  }
  return etat;
}

void goleft()
{
  Serial.print("goleft\n");
  rightposition();
  do
  {
    val_moy = distance_moyenne();
    forward();
  }
  while (val_moy < 10);
  dir == 2;
  if (val_moy > 10)
  {
    turnright();
    centralposition();
  }
}

```

```
void goright()
{
  Serial.print("goright");
  leftposition();
  do
  {
    val_moy = distance_moyenne();
    forward();
  }
  while (val_moy < 10);
  dir == 1;
  if (val_moy > 10)
  {
    turnleft();
    centralposition();
  }
}
```

Code Raspberry

Code pour la détection et l'isolement d'une couleur.

```
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;
using namespace std;

 int main( int argc, char** argv )
 {
   VideoCapture cap(0); //capture the video from web cam

   if ( !cap.isOpened() )  // if not success, exit program
   {
        cout << "Cannot open the web cam" << endl;
        return -1;
   }

   namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called "Control"

 int iLowH = 0;
 int iHighH = 179;

 int iLowS = 0;
 int iHighS = 255;

 int iLowV = 0;
 int iHighV = 255;

 //Create trackbars in "Control" window
 cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
 cvCreateTrackbar("HighH", "Control", &iHighH, 179);

 cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)
 cvCreateTrackbar("HighS", "Control", &iHighS, 255);
```

```
cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
cvCreateTrackbar("HighV", "Control", &iHighV, 255);

  while (true)
  {
     Mat imgOriginal;

     bool bSuccess = cap.read(imgOriginal); // read a new frame from video

      if (!bSuccess) //if not success, break loop
     {
          cout << "Cannot read a frame from video stream" << endl;
          break;
     }

  Mat imgHSV;

  cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the captured frame from BGR to HSV

  Mat imgThresholded;

  inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

  //morphological opening (remove small objects from the foreground)
  erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
  dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

  //morphological closing (fill small holes in the foreground)
  dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
  erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

  imshow("Thresholded Image", imgThresholded); //show the thresholded image
  imshow("Original", imgOriginal); //show the original image


       if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
      {
          cout << "esc key is pressed by user" << endl;
          break;
      }
   }

   return 0;

}
```

Code pour le tracking.

```
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;
using namespace std;

 int main( int argc, char** argv )
 {
   VideoCapture cap(0); //capture the video from webcam

   if ( !cap.isOpened() )  // if not success, exit program
   {
        cout << "Cannot open the web cam" << endl;
        return -1;
   }

   namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called "Control"

 int iLowH = 170;
 int iHighH = 179;

 int iLowS = 150;
 int iHighS = 255;

 int iLowV = 60;
 int iHighV = 255;

 //Create trackbars in "Control" window
 createTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
 createTrackbar("HighH", "Control", &iHighH, 179);

 createTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)
 createTrackbar("HighS", "Control", &iHighS, 255);
```

```
createTrackbar("LowV", "Control", &iLowV, 255);//Value (0 - 255)
createTrackbar("HighV", "Control", &iHighV, 255);

int iLastX = -1;
int iLastY = -1;

//Capture a temporary image from the camera
Mat imgTmp;
cap.read(imgTmp);

//Create a black image with the size as the camera output
Mat imgLines = Mat::zeros( imgTmp.size(), CV_8UC3 );;


  while (true)
  {
     Mat imgOriginal;

     bool bSuccess = cap.read(imgOriginal); // read a new frame from video



      if (!bSuccess) //if not success, break loop
     {
          cout << "Cannot read a frame from video stream" << endl;
          break;
     }

 Mat imgHSV;

cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV); //Convert the captured frame from BGR to HSV

Mat imgThresholded;
```

```
inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

//morphological opening (removes small objects from the foreground)
erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

//morphological closing (removes small holes from the foreground)
dilate( imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );
erode(imgThresholded, imgThresholded, getStructuringElement(MORPH_ELLIPSE, Size(5, 5)) );

//Calculate the moments of the thresholded image
Moments oMoments = moments(imgThresholded);

double dM01 = oMoments.m01;
double dM10 = oMoments.m10;
double dArea = oMoments.m00;

// if the area <= 10000, I consider that the there are no object in the image and it's because of the noise, the area is not zero
if (dArea > 10000)
{
 //calculate the position of the ball
 int posX = dM10 / dArea;
 int posY = dM01 / dArea;

 if (iLastX >= 0 && iLastY >= 0 && posX >= 0 && posY >= 0)
 {
  //Draw a red line from the previous point to the current point
  line(imgLines, Point(posX, posY), Point(iLastX, iLastY), Scalar(0,0,255), 2);
 }

 iLastX = posX;
 iLastY = posY;
}


 imshow("Thresholded Image", imgThresholded); //show the thresholded image

 imgOriginal = imgOriginal + imgLines;
 imshow("Original", imgOriginal); //show the original image

      if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
     {
         cout << "esc key is pressed by user" << endl;
         break;
     }
   }

  return 0;
}
```