Rodrigues Hugo

Neyret Anthony

Rapport de projet de fin d’étude

Robot détecteur de Gaz



IUT GEII Le Creusot Année scolaire 2020 – 2021

Table des matières

[2 Cahier des charges 3](#_Toc67938444)

[2.1 Objectifs 3](#_Toc67938445)

[2.2 Présentation du robot 3](#_Toc67938446)

[2.3 Matériel mis à disposition 4](#_Toc67938447)

[2.4 Matériel acheté pour la réalisation 4](#_Toc67938448)

[3 Planning 5](#_Toc67938449)

[4 La conception 6](#_Toc67938450)

[4.1 Les fonctions mises en œuvre 6](#_Toc67938451)

[4.2 Camera Open MV H7 11](#_Toc67938452)

[4.3 Communication 12](#_Toc67938453)

[4.4 Les capteurs 12](#_Toc67938454)

[5 Conception des cartes électroniques 15](#_Toc67938455)

[5.1 La carte d’alimentation 16](#_Toc67938456)

[5.2 La carte d’extension pour le démultiplexeur 16](#_Toc67938457)

[6 Conception et réalisation logicielle 16](#_Toc67938458)

[7 Difficultés rencontrées : 18](#_Toc67938459)

[7.1 Le déplacement 18](#_Toc67938460)

[8 Au niveau hardware, les points d’amélioration pourraient être : 18](#_Toc67938461)

[9 Conclusion 19](#_Toc67938462)

[10 Annexes 20](#_Toc67938463)

[10.1 Arduino Uno 20](#_Toc67938464)

[10.2 Shield Moteur L298N 21](#_Toc67938465)

[10.3 Code ESP-01 21](#_Toc67938466)

[10.4 Code Principal (Deplacmenet.ino) 23](#_Toc67938467)

[10.5 Librairies C++ 25](#_Toc67938468)

# Cahier des charges

## Objectifs

L’objectif de ce projet a été de transformer un robot construit précédemment à l’IUT en un robot capable de se déplacer dans un environnement connu et détecter quelque chose.

Nous avons compléter le cahier des charges comme suit :

* Créer un robot mobile ayant pour but de trouver l’origine d’un gaz sur un terrain connu et de le signaler à l’utilisateur par un mail.
* Le robot doit être dérivé d’une base de robot préexistant et fourni par l’IUT.
* Le robot sera alimenté par batterie (2 x 12V)
* Il devra disposer d’un mécanisme de détection de gaz et/ou fumée
* Il devra utiliser un système Wi-Fi pour transmettre l’information d’alerte via un mail.
* Il devra suivre un chemin prédéfini par une bande noire au sol.
* Il devra transmettre une photo/vidéo via une caméra de la zone de détection de gaz/fumées.
* Il devra stopper son parcours dès qu’il arrive dans sa zone de stockage (fin de parcours)
* Il devra stopper son parcours s’il rencontre un obstacle.

## Présentation du robot

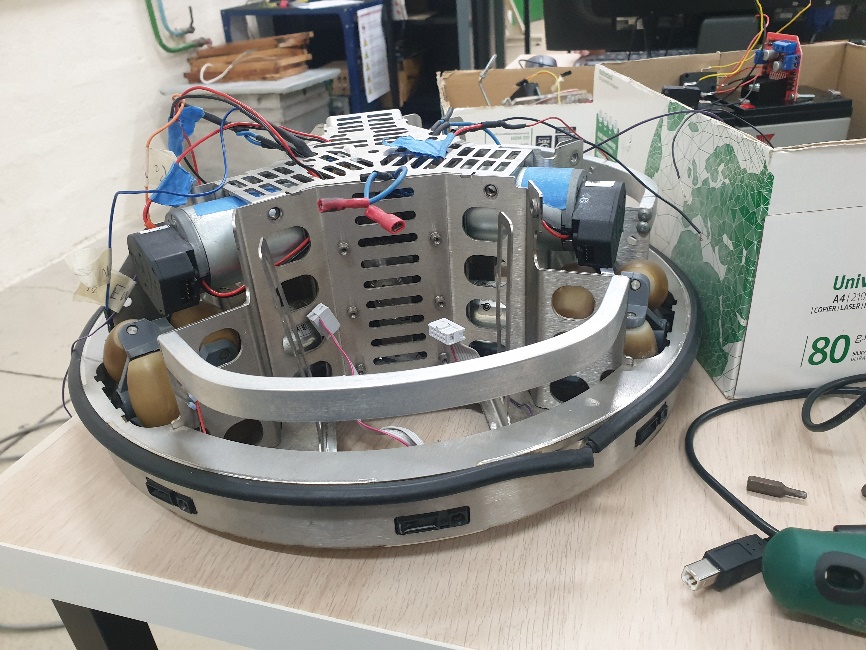
Il s’agit d’un robot construit à l’IUT les années précédentes avec :

* une armature en acier,
* trois roues avec moteurs disposées en étoile,
* deux supports de batterie,
* un capteur de contact digital présent sur tout le tour du robot,
* des capteurs de proximités analogiques positionnés autour du robot (9 capteurs).

Les trois moteurs fonctionnent et les roues tournent correctement en sens horaire et antihoraire suivant la tension d’alimentation (positive/négative).

Les deux batteries sont opérationnelles et disposent d’une tension de 12V chacune.

**Note : Les batteries ne sont pas rechargeables.**



## Matériel mis à disposition

L’IUT a fourni le matériel suivant pour la réalisation du robot :

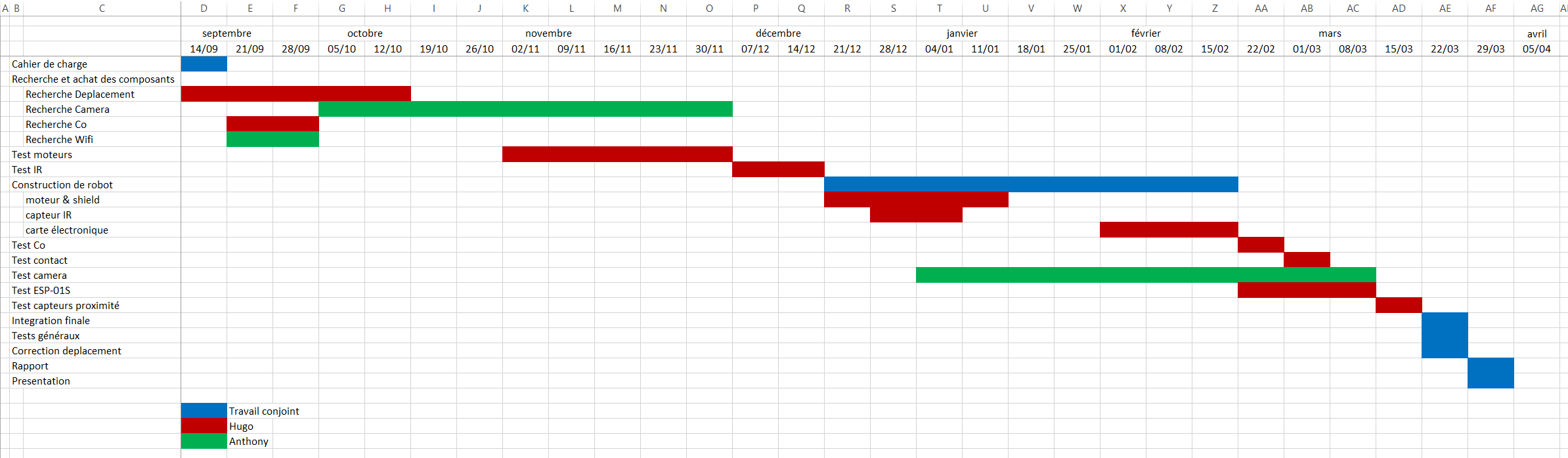
* Le robot de combat
* Les deux batteries de 12V
* Un microcontrôleur de type Arduino UNO
* Une caméra de type OpenMV H7
* Des câbles

## Matériel acheté pour la réalisation

Afin de permettre la réalisation du projet, le matériel suivant a été acheté en complément, son utilisation sera détaillée par la suite :

* Un capteur de détection de gaz/fumée
* Trois capteurs digitaux de proximité à LED
* Un multiplexeur de type 74HC595
* Des borniers (Dupont et à vis)
* Des câbles

# Planning



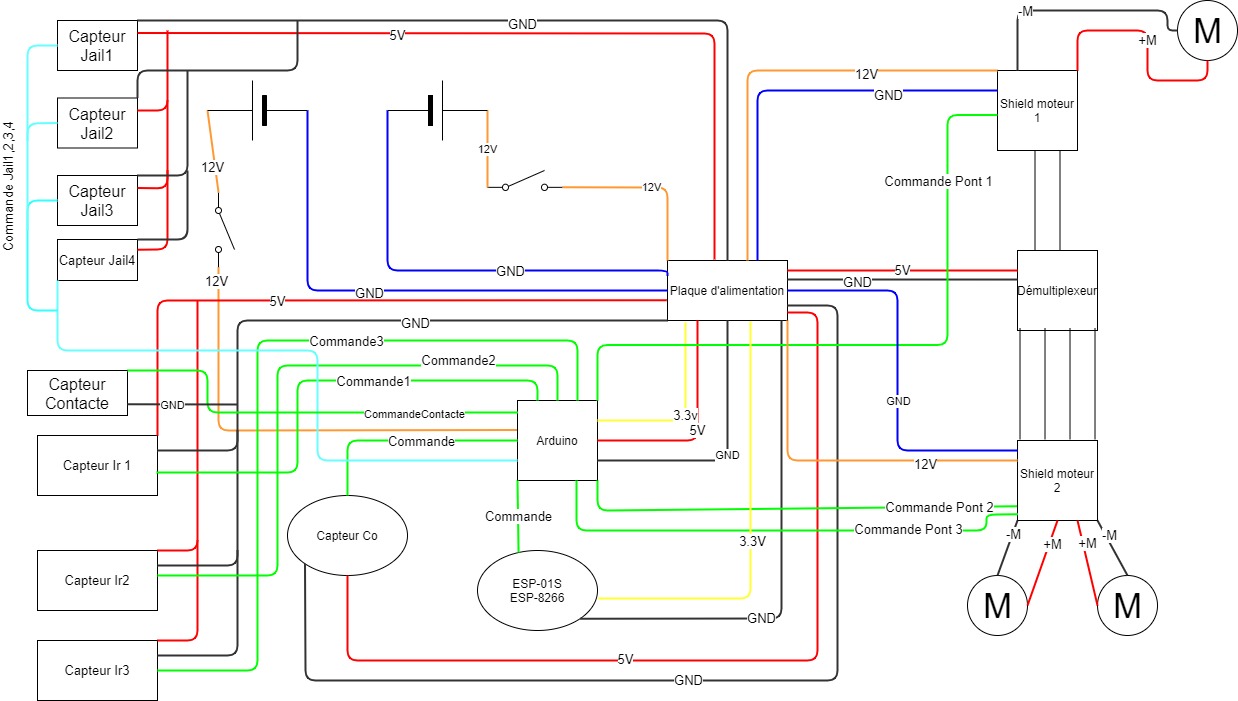
# La conception

## Les fonctions mises en œuvre

Les différentes fonctions mises en œuvre pour le robot sont :

* Le déplacement
* La détection de gaz/fumées
* La détection d’obstacles
* L’émission d’un mail d’avertissement en cas de détection de gaz/fumées.
* La gestion d’une zone de stockage

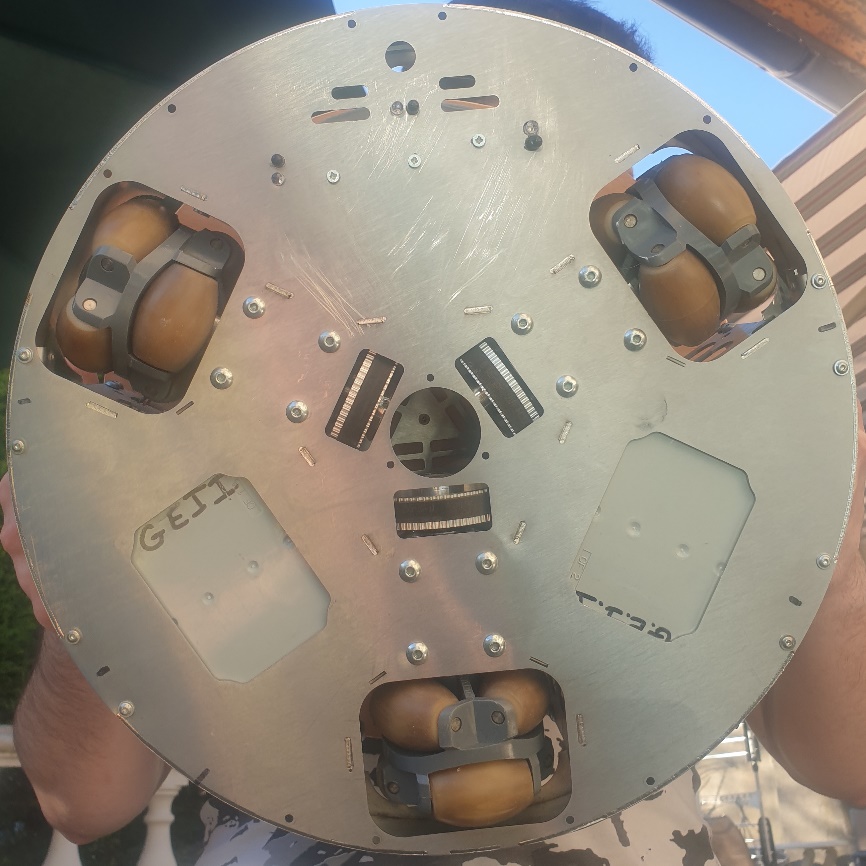
Voici une vue globale du montage réalisé :



En orange, on distingue les composants en charge de gérer le déplacement, en jaune la détection de gaz, en vert la détection de contact, en violet l’envoi de mail et en bleu la zone de stockage.

### La gestion du déplacement

Le robot est constitué de trois roues montées en étoile dont on pilote le sens la vitesse de rotation pour guider le robot.



Le tableau suivant récapitule les actions à effectuer sur le robot pour gérer son déplacement :

|  | **Roue 1 (avant droit)** | **Roue 2 (avant gauche)** | **Roue 3 (arrière)** |
| --- | --- | --- | --- |
| Avancer | Sens horaire | Sens antihoraire | Stop |
| Ajuster à droite | Sens horaire | Sens antihoraire | Sens antihoraire |
| Ajuster à gauche | Sens horaire | Sens antihoraire | Sens horaire |
| Tourne à droite | Sens antihoraire | Sens antihoraire | Sens antihoraire |
| Tourne à gauche | Sens horaire | Sens horaire | Sens horaire |
| Stop | Stop | Stop | Stop |

Afin de piloter les moteurs et gérer la trajectoire du robot, on a choisi d’utiliser des shields moteur et des capteurs infra-rouge placés dans le nez du robot pour détecter la ligne noire.

Pour choisir les bons cas pour suivre la ligne on a dressé un tableau logique des entrées et sorties inversés :

| **Entrées** | | | **Sorties** | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Capteur**  **Gauche** | **Capteur Centre** | **Capteur**  **Droit** | **D** | **G** | **Ad** | **Ag** | **Stop** | **Av** |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | x | x | x | x | x | x |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | x | x | x | x | x | x |

Les « 0 » représentent quand les capteurs IR détectent la ligne noire.

* **D/G** représente une rotation droite/gauche (le robot tourne sur place)
* **Ad/Ag** représente un ajustement droite / gauche (le robot continue d’avancer mais tourne en même temps grâce à la roue arrière)
* **Stop** : arrêt des moteurs
* **Av** : représente une avance du robot

### Les Moteurs

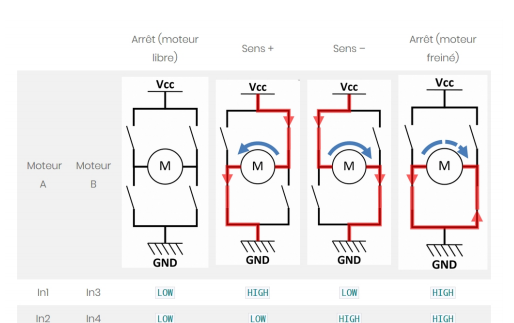
Les moteurs intégrés dans le robot sont des GR42X25 :

* Tension nominale : 24V,
* Un courant nominal de 0.86A.
* Puissance nominale de 14.3W.

Nous avant fait le choix d’alimenter les moteurs en 12V car les Shields L298N ne peuvent pas délivrer plus de 12V.

### Les Shields L298N

Afin de piloter les moteurs et gérer le sens de rotation, on a choisi de mettre en œuvre des shields moteur de type L298N qui intègrent un double pont en H.



L’activation d’un pont en H (broches Enable ENA et ENB) ainsi que le sens de rotation des moteurs (broches In1 à In4) sont pilotés par l’Arduino. Ces broches d’activation sont des entrées PWM permettant ainsi de contrôler la vitesse de rotation du moteur.

On justifie le choix d’avoir pris des Shields par le fait que l’Arduino ne peut pas alimenter seul les moteurs (sortie Arduino 5V et max 40 mA).

Les broches permettant de gérer le sens de rotation sont câblées à l’Arduino par l’intermédiaire d’un démultiplexeur.

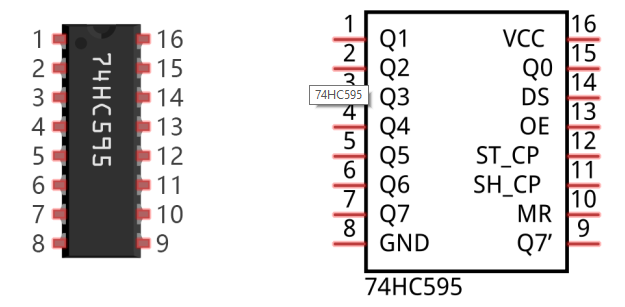
Les Shields sont alimentés par les batteries via une plaque d’alimentation en 12 V.

### Démultiplexeur 74HC595

Lors de la conception, nous nous sommes rendu compte que nous allions manquer de pins digitales sur l’Arduino pour piloter les 2 Shields Après réflexion et recherche sur Internet on a décidé d’opter pour l’ajout d’un démultiplexeur 74HC595N pour étendre le nombre de pins digitales disponibles plutôt que de rajouter un autre Arduino.

Le 74HC595 est en fait un registre à décalage qui reçoit en entrée une information sous la forme d’un champ de bits (codé sur 8 bits) et qui le transforme en 8 signaux parallèles (état haut ou bas) sur ses sorties Qn.

L’alimentation du démultiplexeur se fait par la plaque d’alimentation en 5V.



L’entrée des données issues de l’Arduino permettant de commander les sorties se trouve sur **Serial Data Input** (DS).

La modification de l’état de sortie du registre nécessite d’activer le **Latch** (ST\_CP) que l’on passe à l’état bas. Une fois le registre reconfiguré, il est remis à l’état haut pour modifier les sorties du registre.

Afin de savoir qu’une nouvelle donnée est disponible en lecture (sur la broche DS), le 74HC595 a besoin d’un front montant d’horloge **Clock** (SH\_CP) qui est fournie par l’Arduino.

Une fois le **Latch** remis à l’état haut, les pins de sorties sont à l’image du champ de bits fourni en entrée sur la pin **DS**.

Pour chaque moteur, le sens de rotation horaire est codé sous la forme 01, alors que le sens antihoraire est codé sous la forme 10.

La pin Q0 n’est pas utilisée ainsi que la pin Q7 et sont mises à 0. Les valeurs pour la pin DS obtenues après un OU logique sont :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | M1 | M2 | M3 | DS |
| **Avancer** | 0b0**01**00000 | 0b000**10**000 | 0b00000000 | 0b0**011000**0 |
| **AjusteGauche** | 0b0**01**00000 | 0b000**10**000 | 0b00000**01**0 | 0b0**011001**0 |
| **AjusteDroite** | 0b0**01**00000 | 0b000**10**000 | 0b00000**10**0 | 0b0**011010**0 |
| **Gauche** | 0b0**01**00000 | 0b000**01**000 | 0b00000**01**0 | 0b0**010101**0 |
| **Droite** | 0b0**10**00000 | 0b000**10**000 | 0b00000**10**0 | 0b0**101010**0 |
| **Stop** | N/A | N/A | N/A | N/A |

## Camera Open MV H7

La caméra utilisée est une caméra de chez OpenMV model H7, fournie avec le robot.

On avait pour but de faire communiquer la caméra et l’Arduino. Ce dernier devait détecter du gaz et demander à la caméra de prendre une photo, puis l’Arduin aurait récupéré cette photo pour l’envoyer par mail en Wifi.

Puis nous nous sommes orientés vers un envoi de la photo directement depuis la caméra grâce à un module Wifi.

La caméra OpenMV H7 ne supporte pas d’autres modules Wifi que l’extension Shield Wifi fourni par OpenMV.

Il nous a été impossible de pouvoir transmettre les photos prises par la caméra avec le module Wifi ESP8266.01S que nous avions acheté.

Nous nous sommons penchés trop tard sur la fonction d’envoi des photos depuis la caméra et le tarif du Shield (30€) ainsi que les délais de livraison à quelques semaines de la fin du projet nous ont obligé à supprimer cette fonctionnalité.

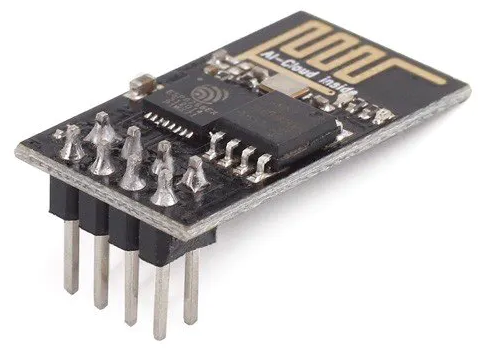
Toutefois, pour régler ce problème, on a testé deux solutions pour le projet :

* La première solution consistait à utiliser notre module ESP-01S branché sur l’OpenMV et faire communiquer les deux. OpenMV fournit des librairies qui permettent de faire du transfert de données via des bus UART, I2C ou SPI qui auraient pu nous permettre de transférer les images octets par octets à l’Arduino ou à l’ESP-01S. Solution assez complexe et probablement impossible du fait du peu de mémoire disponible sur l’ESP.
* La deuxième solution a été d’acheter une caméra Arduino OV7670, mais nous avons trouvé très peu d’exemples de mise en œuvre ou alors avec d’autres version d’Arduino (la librairie Adafruit OV7670 n'est pas compatible avec un Arduino UNO mais avec un µcontroleur de gamme SADM et pas ATM32 comme l'UNO).

Donc les deux solutions n’étaient pas viables et nous avons opté d’utiliser notre module ESP-01S pour envoyer uniquement un mail en cas de détection de gaz par le capteur MQ2.

## Communication

Le module communication a été réalisé à l’aide d’un **ESP8266-01S**, qui a été programmé via les liaisons RX/TX de l’Arduino. Le code utilisé sur l’ESP-01 est disponible dans l’annexe 10.3.



L**’ESP8266-01S** est activé par l’Arduino via la pin **IO2**. Lorsque l’**ESP8266-01S** est activé, il envoie un mail pour signaler que le robot a détecté du gaz.

Il est alimenté en 3.3V et nécessite.

## Les capteurs

A l’origine, le robot était équipé de deux types des capteurs différents :

* Un capteur digital de contact.
* 9 capteurs analogiques infra-rouge de proximité.

En complément, nous avons ajouté au robot un capteur de gaz MQ-2 pour gérer la détection des gaz/fumées.

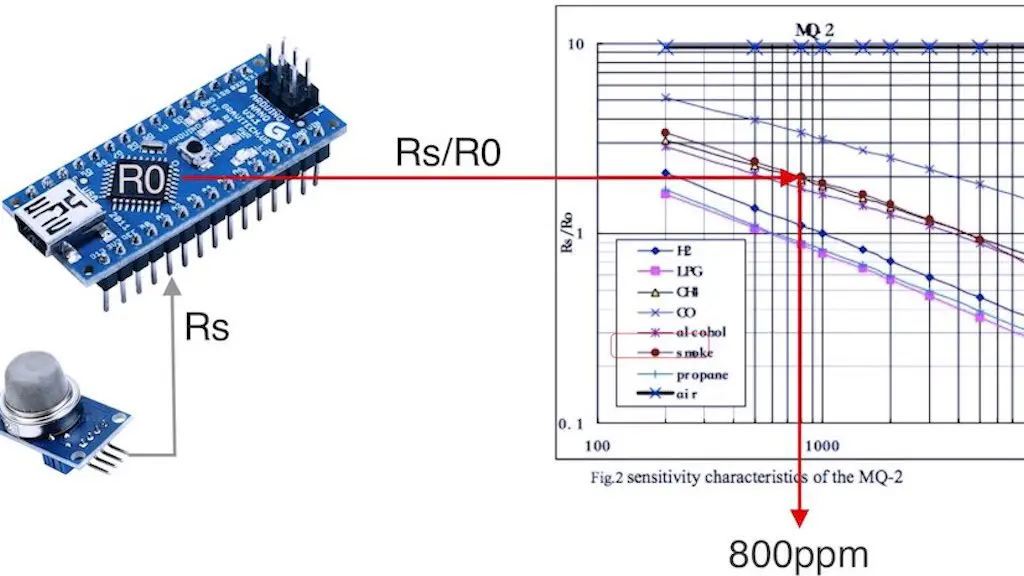
Les capteurs infra-rouges sont utilisés pour détecter la zone de stockage du robot et le capteur de contact quant à lui est utilisé pour détecter un possible obstacle sur le chemin tracé.

### Capteur de gaz

Afin de pouvoir détecter les gaz/fumées, nous avons acheté un capteur de gaz de type MQ-2 :



On a décidé de prendre ce capteur car sa gamme de détection nous offre des choix plutôt variés :



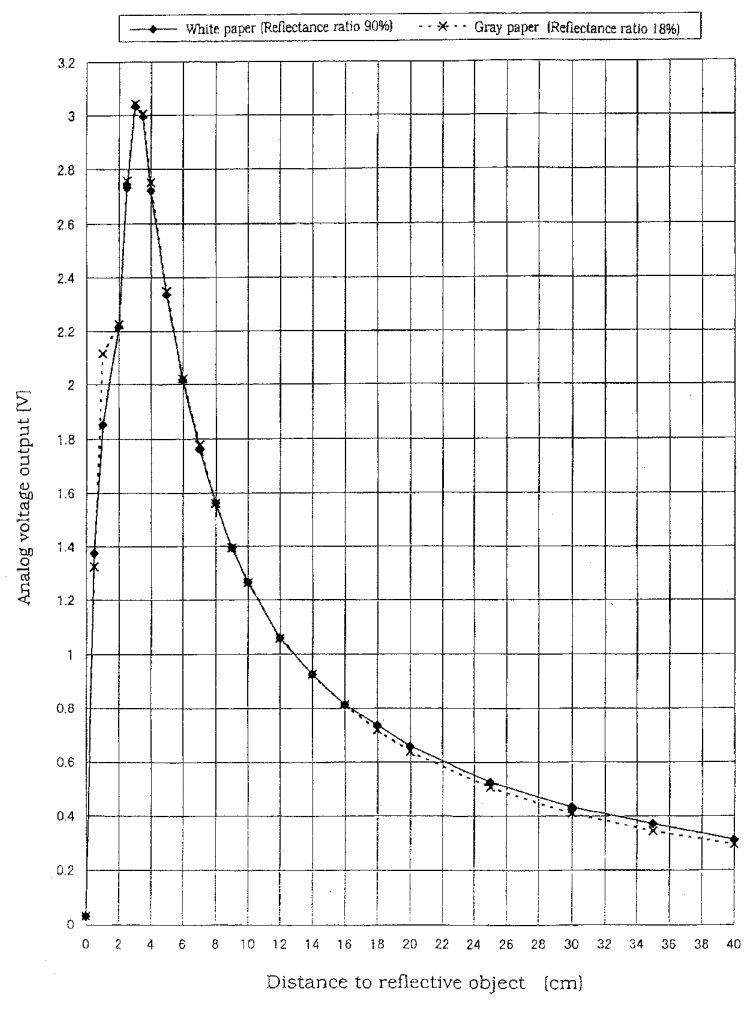
Comme on le voit sur le graphique, plusieurs types de gaz différents peuvent être détectés. A des fins de simplifications pour le projet, nous nous sommes contenté de lire la valeur renvoyée par le capteur sans mettre en œuvre la partie calibration et analyse du module (un exemple est disponible sur ce site : <https://projetsdiy.fr/utiliser-detecteur-gaz-fumees-mq2-code-arduino/>).

Le capteur dispose de deux sorties : une digitale qui renvoie un état haut en cas de détection basée sur un potentiomètre de réglage du seuil et une analogique qui renvoie simplement la valeur mesurée.

Nous avons fait le choix de prendre la sortie analogique car nous souhaitions contrôler un niveau de seuil de 300 ppm pour le CO, afin de préparer le terrain en vue de la mise en œuvre de la partie calibration/analyse. Nous avons fixé une valeur moyenne de détection en cas de présence de gaz de type briquet (butane) pour le projet.

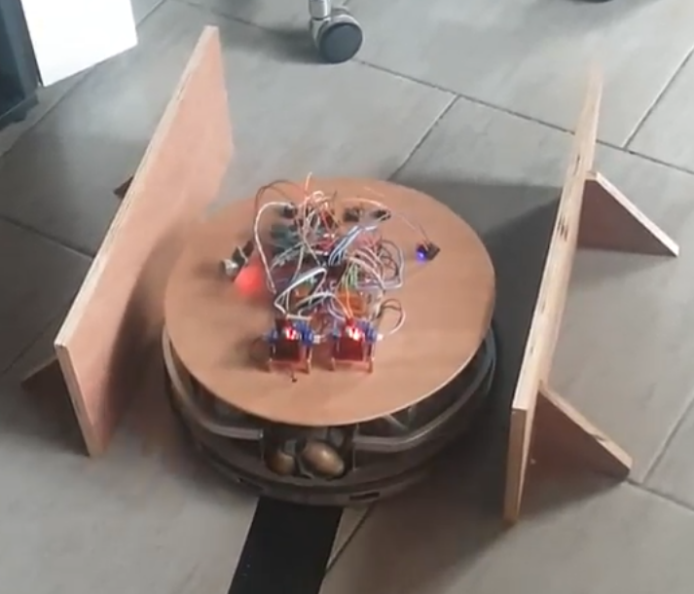
### Capteur de la zone de stockage :

Les capteurs Infrarouge montés d’origine sur le robot sont des capteurs analogiques de type GP2Y0A41SK0F. Ils répondent à la courbe de détection suivante :



On distingue dans la courbe une zone d’incertitude liée à une trop grande proximité d’un obstacle avec le capteur. On a donc choisi de mettre un seuil de déclenchement à 18 cm, ce qui correspond à environ 0.7 V en sortie du capteur.

Afin de réutiliser les capteurs infra-rouges fournis, on a décidé de gérer une zone de stockage pour que le robot vienne se stationner lorsqu’il a fini son tour de ronde :



Le capteur est alimenté en 5V et relié à la masse sur la plaque d’alimentation.

Les pins de détection sont reliées à des pins analogiques de l’Arduino.

Lorsque 3 capteurs sur 4 détectent la zone de stockage à moins de 18 centimètres (zone seuil de détection), le robot s’arrête.

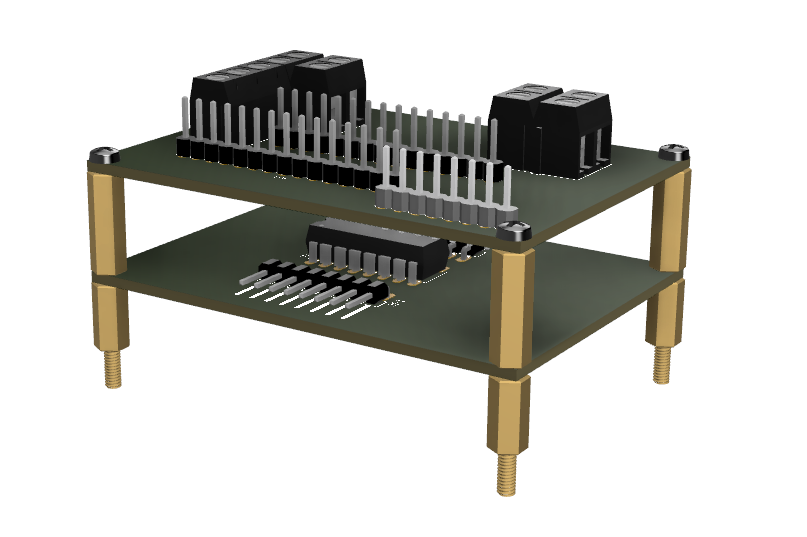
### Capteur de contact

Le capteur de proximité d’obstacles est un capteur digital de contact qui se matérialise par une bande contact autour du robot qui est un interrupteur normalement ouvert.

Il est branché sur une pin digitale de l’Arduino et sur la masse de la plaque d’alimentation.

Il fonctionne de telle sorte que quand il y a un objet sur le chemin du robot, le capteur rentre en contact avec l’obstacle et le robot s’arrête.

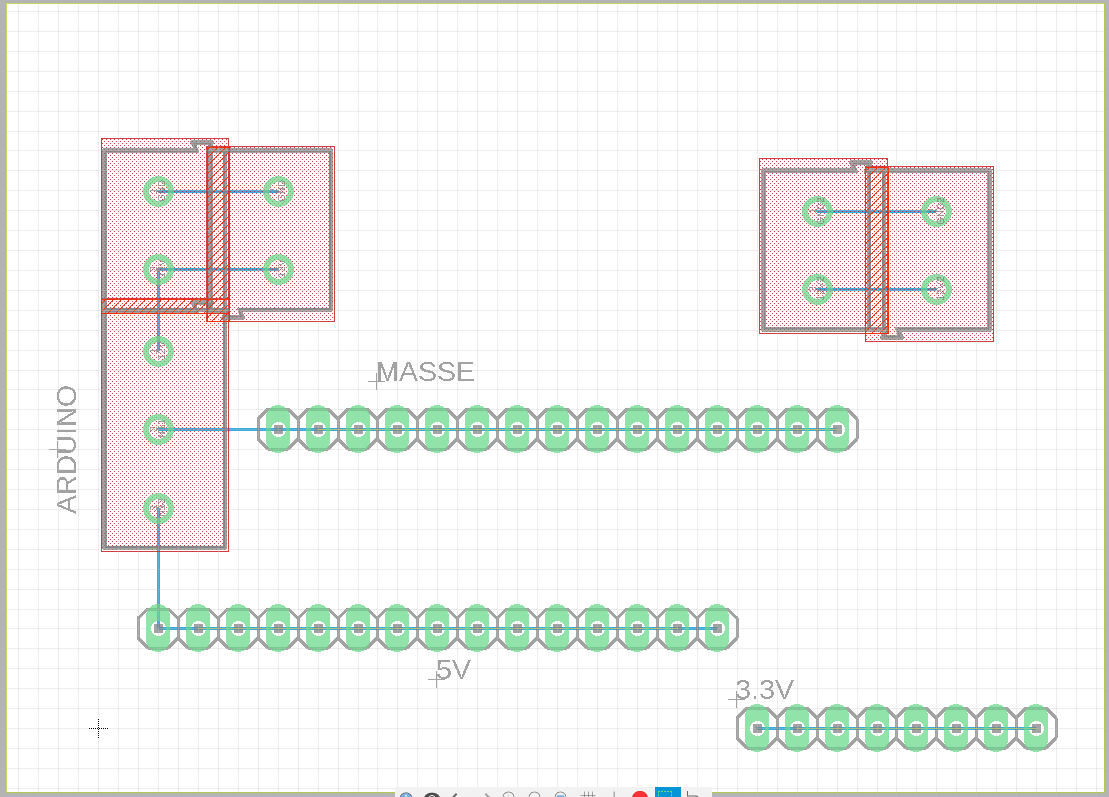
# Conception des cartes électroniques



## La carte d’alimentation

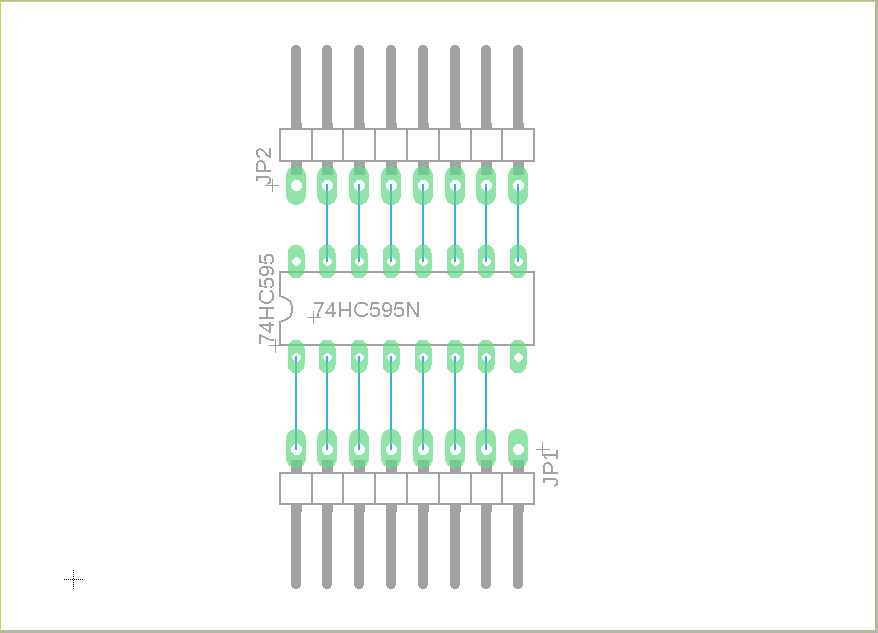
On a décidé de déporter les alimentations (12V, 3.3V, 5V et masse) de l’Arduino sur une carte d’alimentation complémentaire, afin de pallier au manque de pins disponibles sur l’Arduino.

La carte permet aussi d’alimenter les Shields moteur en 12V et l’Arduino en 12 V par la pin Vin.



## La carte d’extension pour le démultiplexeur

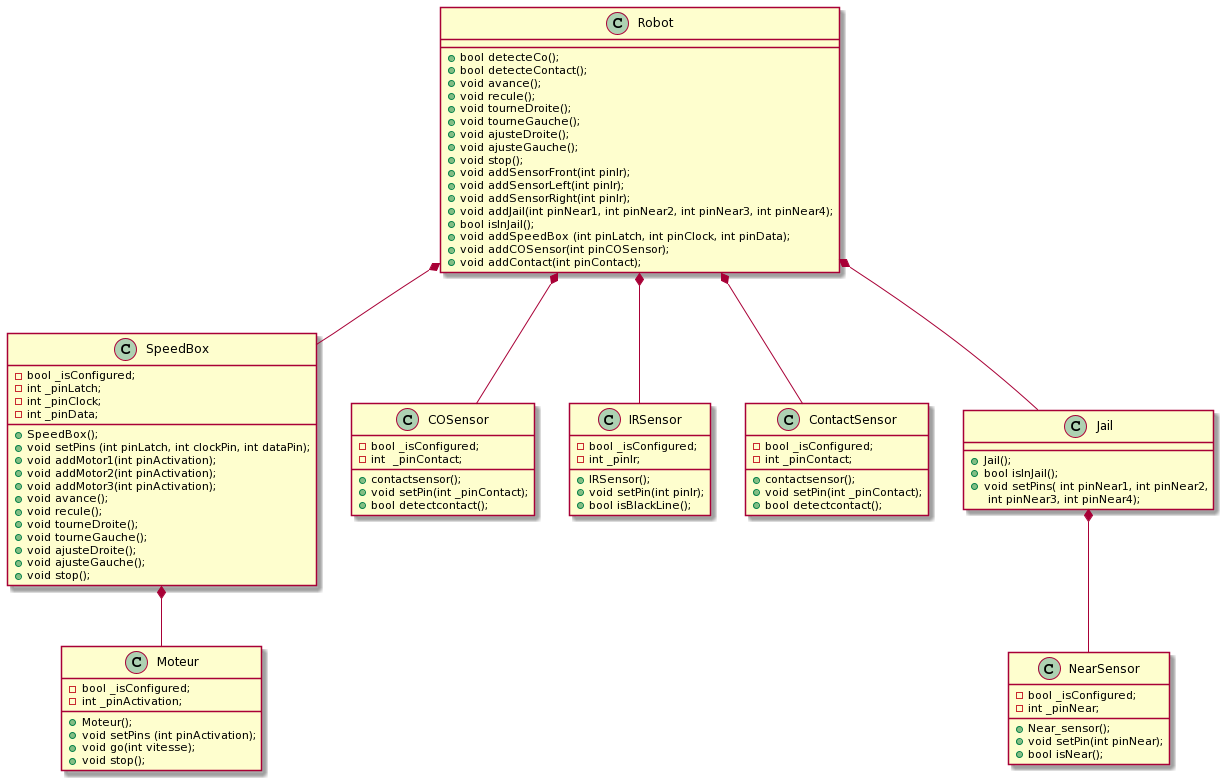
Cette carte permet simplement de câbler les différents ports du 74HC595.



# Conception et réalisation logicielle

Le microcontrôleur mis en œuvre étant un Arduino, nous avons donc codé le logiciel en C++.

Comme nous avions plusieurs modules à gérer, nous avons fait le choix de séparer le code entre le module de base, le « main » de l’Arduino nommé ***Deplacement\_robot.ino*** et des librairies écrites en C++, par exemple ***Robot.h*** et ***Robot.cpp***, ceci afin de nous simplifier la tâche et découper le code en plus petits modules plus faciles à réaliser et à tester.

Nous avons donc modélisé les objets de la manière suivante dans un diagramme UML : 

La classe ***Robot*** est la classe de base et elle modélise le robot avec ses différents capteurs et/ou fonctions.

A noter la présence d’une classe ***SpeedBox*** qui a été introduite à la suite de la mise en œuvre du démultiplexeur. En effet, chaque moteur ne peut pas être piloté séparément, typiquement on ne peut pas changer le sens de rotation d’un moteur car cela joue directement sur les autres moteurs (cf. chapitre 4.1.4). Il nous fallait donc disposer d’une classe intermédiaire permettant cette gestion centralisée sans pour autant que cela soit supporté par la classe ***Robot*** elle-même.

Il en va de même pour la classe ***Jail*** qui est une centralisation de l’état des différents capteurs de proximité et qui gère notamment la détection par 3 capteurs sur 4.

La partie envoi de mail n’est pas présente dans cette modélisation car il s’agit en fait d’un code différent, directement écrit en C++ dans un fichier de type **.ino**. Il est tiré d’une librairie : **EMailSender library for Arduino**.

Le code a été modifié pour permettre l’ajout du message et des paramètres d’envoi de mail :

* Nom du serveur SMTP  (gmail)
* Identifiants d’envoi
* Sujet
* Message

# Difficultés rencontrées :

## Le déplacement

### Vitesse des moteurs

Lors du déplacement, dans les virages, les capteurs IR n’avaient pas le temps de capter la bande noire.

Nous avons dû ralentir les moteurs en abaissant la tension d’alimentation des moteurs en jouant sur la pin Enable du shield et la tension délivrée par le pin PWM de l’Arduino. De plus, chaque moteur dispose d’une résistance interne différente et nous avons dû adapter les tensions délivrées par l’Arduino moteur par moteur afin d’obtenir des tensions d’alimentation identiques et assurer ainsi des vitesses de rotation identiques.

### Dérapage des roues

La grande inertie du robot et le sol glissant provoquaient des dérapages et empêchaient le robot de tourner correctement.

Nous avons dû ajouter les fonctions AjusterGauche et AjusterDroite ainsi qu’adapter l’algorithme de suivi de trajectoire afin d’aider le robot mieux rester aligner avec les bandes noires.

### Le module Wifi incompatible avec la caméra

Malgré les différentes solutions envisagées et testées, le matériel à disposition et le planning nous ont contraint à uniquement envoyer un mail.

# Au niveau hardware, les points d’amélioration pourraient être :

* Acheter un Shield Wifi OpenMV pour faire fonctionner la caméra, l’intégration avec l’Arduino pour déclencher la prise de vue devrait être simple.
* Changer les batteries pour des batteries rechargeables et créer une station de recharge dans la zone de stockage.

Au niveau software :

* Ajouter dans le code de la zone de stockage un horodatage pour décider de la mise en route du robot et la fin de son cycle
* Modifier le code du capteur d’obstacle pour que le robot contourne l’objet et continu sa mission
* Permettre de paramétrer les mails envoyés par l’ESP-01S, par exemple pour signaler qu’il y a un obstacle à enlever ou que le robot a fini sa ronde et se trouve dans sa zone de stockage et qu’il charge.

# Conclusion

Pour conclure sur ce projet de fin d’année, il nous a permis de pouvoir comprendre les démarches d’un projet sur plusieurs mois, la gestion des contraintes de livraison et de la compatibilité des fournisseurs que l’on peut rencontrer dans une entreprise.

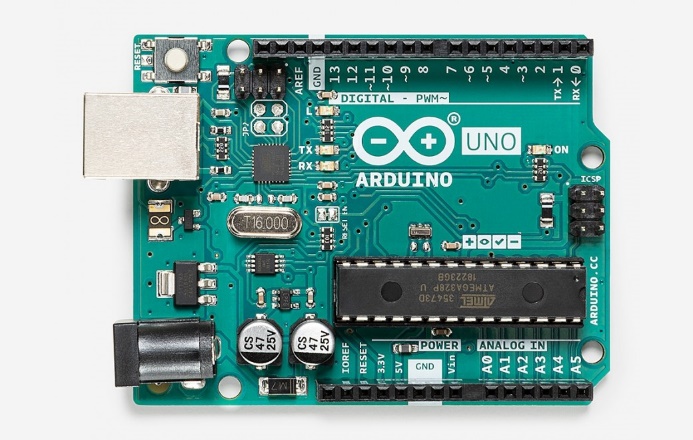
Cela nous a également permis de nous amener à rechercher des solutions alternatives pour répondre à un cahier des charges quand la solution initiale ne fonctionne pas.

Nous avons également découvert la programmation sur des microcontrôleurs que nous n’utilisons pas régulièrement comme l’Arduino UNO et l’ESP8266-01S ainsi que la mise en œuvre de librairies en C++.

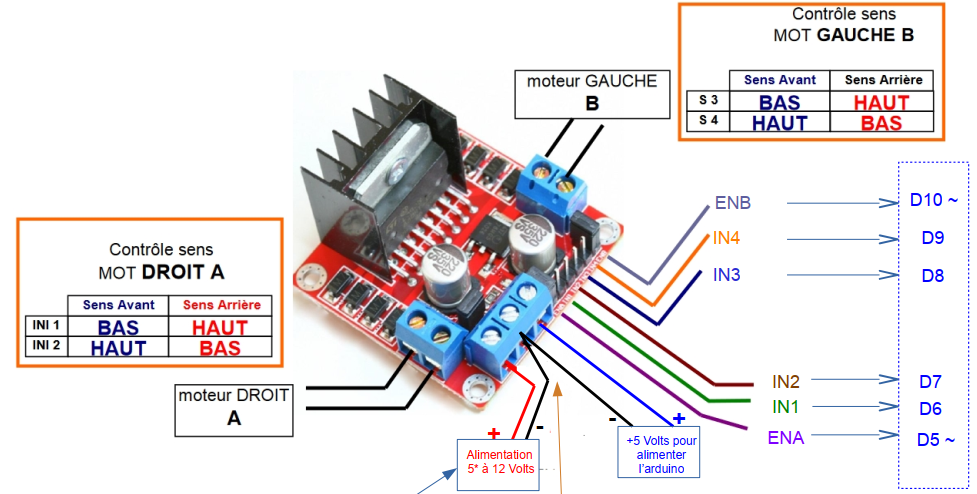
Nous avons également abordé de la conception 3D et électronique sous Fusion 360 pour la construction des cartes d’extension, développer des connaissances sur les protocoles de messagerie (SMTP), sur la mise d’une carte Wifi, sur l’utilisation de GitHub, la modélisation UML…

# Annexes

## Arduino Uno



## Shield Moteur L298N



## Code ESP-01

/\*

\* EMailSender library for Arduino, esp8266 and esp32

\* Simple esp32 Gmail send to a distribution list example

\*

\* https://www.mischianti.org/category/my-libraries/emailsender-send-email-with-attachments/

\*

\*/

#include "Arduino.h"

#include <EMailSender.h>

uint8\_t connection\_state **=** 0**;**

uint16\_t reconnect\_interval **=** 10000**;**

EMailSender emailSend**(**"hugo38.test1@gmail.com"**,** "toToeap1!"**);**

uint8\_t WiFiConnect**(**const char**\*** nSSID **=** **nullptr,** const char**\*** nPassword **=** **nullptr)**

**{**

static uint16\_t attempt **=** 0**;**

Serial**.**print**(**"Connecting to "**);**

**if(**nSSID**)** **{**

WiFi**.**begin**(**nSSID**,** nPassword**);**

Serial**.**println**(**nSSID**);**

**}**

uint8\_t i **=** 0**;**

**while(**WiFi**.**status**()!=** WL\_CONNECTED **&&** i**++** **<** 50**)**

**{**

delay**(**200**);**

Serial**.**print**(**"."**);**

**}**

**++**attempt**;**

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

**if(**i **==** 51**)** **{**

Serial**.**print**(**"Connection: TIMEOUT on attempt: "**);**

Serial**.**println**(**attempt**);**

**if(**attempt **%** 2 **==** 0**)**

Serial**.**println**(**"Check if access point available or SSID and Password\r\n"**);**

**return** **false;**

**}**

Serial**.**println**(**"Connection: ESTABLISHED"**);**

Serial**.**print**(**"Got IP address: "**);**

Serial**.**println**(**WiFi**.**localIP**());**

**return** **true;**

**}**

void Awaits**()**

**{**

uint32\_t ts **=** millis**();**

**while(!**connection\_state**)**

**{**

delay**(**50**);**

**if(**millis**()** **>** **(**ts **+** reconnect\_interval**)** **&&** **!**connection\_state**){**

connection\_state **=** WiFiConnect**();**

ts **=** millis**();**

**}**

**}**

**}**

void SendMail**(){**

EMailSender**::**EMailMessage message**;**

message**.**subject **=** "Robot detection"**;**

message**.**message **=** "Detection CO"**;**

// Send email

const char**\*** arrayOfEmail**[]** **=** **{**"hugo38.test1@gmail.com"**};**

EMailSender**::**Response resp **=** emailSend**.**send**(**arrayOfEmail**,** 1**,** message**);**

Serial**.**println**(**"Sending status: "**);**

Serial**.**println**(**resp**.**status**);**

Serial**.**println**(**resp**.**code**);**

Serial**.**println**(**resp**.**desc**);**

**}**

void setup**()**

**{**

Serial**.**begin**(**115200**);**

const char**\*** ssid **=** "Portablehugo"**;**

const char**\*** password **=** "0987654321"**;**

connection\_state **=** WiFiConnect**(**ssid**,** password**);**

**if(!**connection\_state**)** // if not connected to WIFI

Awaits**();** // constantly trying to connect

pinMode**(**2**,**INPUT\_PULLUP**);**

**}**

void loop**()**

**{**

**if(**digitalRead**(**2**)** **==**HIGH**){**

Serial**.**println**(**"debut de sendmail"**);**

SendMail**();**

**}**

delay**(**100**);**

**}**

## Code Principal (Deplacement.ino)

#include <Robot.h>

// Configuration des PIN Driver 1

const int pin\_shie1d1\_ENA **=** 10**;** // PWM Orange

const int pin\_shie1d1\_ENB **=** 0**;** // N/A

// Configuration des PIN Driver 2

const int pin\_shie1d2\_ENA **=** 3**;** // PWM Gris

const int pin\_shie1d2\_ENB **=** 5**;** // PWM Vert

// Configuration des PIN\_IR

const int pin\_IrFront **=** 7**;** // Orange

const int pin\_IrLeft **=** 9**;** // Vert

const int pin\_IrRight **=** 8**;** // Marron

// Configuration des PIN\_NEAR

const int pin\_NearRightFront **=** A1**;** //

const int pin\_NearRightBack **=** A2**;** //

const int pin\_NearLeftFront **=** A3**;** //

const int pin\_NearLeftBack **=** A4**;** //

// Configuration des Pins du multiplexeur

const int pinLatch **=** 12**;** // Pin connected to ST\_CP of 74HC595（Pin12）Orange

const int pinClock **=** 13**;** // Pin connected to SH\_CP of 74HC595（Pin11）Blanc

const int pinData **=** 11**;** // Pin connected to DS of 74HC595（Pin14）Blanc

// Configuration des Pins du capteur Co et fumée

const int pinCOSensor **=** A0**;**

//Configuration des Pins du capteur de contacte

const int pinContactSensor **=**4**;**

//Configuration de la Pin du wifi

const int pinWifi **=** 2**;**

Robot robot **=** Robot**();**

bool turnRight**=false;**

bool turnLeft**=false;**

void setup**()**

**{**

Serial**.**begin**(**9600**);**

Serial**.**println**(**"Setup configuration"**);**

// Creation du capteur

robot**.**addCOSensor**(**pinCOSensor**);**

// Creation du wifi

Serial**.**println**(**"Creation du module wifi"**);**

pinMode**(**pinWifi**,**OUTPUT**);**

digitalWrite**(**pinWifi**,**LOW**);**

// Creation de la boite de vitesse

Serial**.**println**(**"Creation de la boite de vitesse"**);**

robot**.**addSpeedBox**(**pinLatch**,** pinClock**,** pinData**);**

robot**.**speedBox**.**addMotor1**(**pin\_shie1d1\_ENA**);** // Moteur gauche

robot**.**speedBox**.**addMotor2**(**pin\_shie1d2\_ENA**);** // Moteur droit

robot**.**speedBox**.**addMotor3**(**pin\_shie1d2\_ENB**);** // Moteur arrière

Serial**.**println**(**"Ajout des capteurs IR"**);**

robot**.**addSensorFront**(**pin\_IrFront**);**

robot**.**addSensorLeft**(**pin\_IrLeft**);**

robot**.**addSensorRight**(**pin\_IrRight**);**

Serial**.**println**(**"Ajout des capteurs de la prison"**);**

robot**.**addJail**(**pin\_NearRightFront**,** pin\_NearRightBack**,** pin\_NearLeftFront**,** pin\_NearLeftBack**);**

Serial**.**println**(**"ajout du capteur contacte"**);**

robot**.**addContact**(**pinContactSensor**);**

Serial**.**println**(**"Robot prêt... Attente 2\""**);**

robot**.**stop**();**

delay**(**2000**);**

**}**

void loop**()** **{**

Serial**.**println **(**"#### Start new loop"**);**

digitalWrite**(**pinWifi**,**LOW**);**

**if** **((**robot**.**isInJail**()** **||** robot**.**detecteContact**()))**

**{**

Serial**.**println**(**"Robot en prison... Stop ou capteur touché"**);**

robot**.**stop**();**

**}**

**else** **if** **(**robot**.**detecteCo**())**

**{**

Serial**.**println**(**"Détection de CO ou de fumée"**);**

robot**.**stop**();**

Serial**.**println**(**"send mail"**);**

digitalWrite**(**pinWifi**,**HIGH**);**

delay**(**150**);**

digitalWrite**(**pinWifi**,**LOW**);**

**while(true)** **{**

Serial**.**println**(**"Le robot est bloqué, faire intervenir un technicien"**);**

delay**(**10000**);**

**}**

**}**

**else** **{**

**if(**robot**.**IRSensorRight**.**isBlackLine**()){**

Serial**.**println**(**"Détection d'une ligne à droite"**);**

turnRight**=true;**

turnLeft**=false;**

robot**.**ajusteDroite**();**

**}**

**if(**robot**.**IRSensorLeft**.**isBlackLine**()){**

Serial**.**println**(**"Détection d'une ligne à gauche"**);**

turnRight**=false;**

turnLeft**=true;**

robot**.**ajusteGauche**();**

**}**

**if(**robot**.**IRSensorFront**.**isBlackLine**())** **{**

Serial**.**println**(**"Détection d'une ligne devant"**);**

robot**.**avance**();**

**}**

**else** **{**

Serial**.**println**(**"Aucune ligne détectée devant, arrêt du robot"**);**

**if** **(**turnLeft**){**

Serial**.**println**(**"Rotation vers la gauche"**);**

robot**.**tourneGauche**();**

**}**

**else** **if** **(**turnRight**){**

robot**.**tourneDroite**();**

Serial**.**println**(**"Rotation vers la gauche"**);**

**}**

**}**

**}**

delay**(**20**);**

**}**

## Classes C++

### contactsensor.h

/\*

\* Contactsensor.h - Librairie de gestion d'un capteur de contacte

\* Créé par Hugo Rodrigues/Anthony Neyret - 03/03/2020

\*/

#ifndef CONTACTSENSOR\_H

#define CONTACTSENSOR\_H

#include "Arduino.h"

class contactsensor

**{**

public**:**

contactsensor**();**

void setPin**(**int \_pinContact**);**

bool detectcontact**();**

private**:**

bool \_isConfigured**;**

int \_pinContact**;**

**};**

#endif // CONTACTSENSOR\_H

### contactsensor.cpp

/\*

\* Contactsensor.h - Librairie de gestion d'un capteur de contacte

\* Créé par Hugo Rodrigues/Anthony Neyret - 03/03/2020

\*/

#include "contactsensor.h"

#include "Arduino.h"

contactsensor**::**contactsensor**()**

**{**

\_isConfigured **=** **false;**

**}**

void contactsensor**::**setPin**(**int pinContact**){**

pinMode **(**pinContact**,** INPUT\_PULLUP**);**

\_pinContact**=**pinContact**;**

\_isConfigured **=** **true;**

**}**

bool contactsensor**::**detectcontact**(){**

**if** **(**\_isConfigured **&&** **!**digitalRead**(**\_pinContact**))**

**return** **true;**

**else**

**return** **false;**

**}**

### cosensor.h

/\*

\* Co.h - Librairie de gestion d'un capteur CO

\* Créé par Hugo Rodrigues/Anthony Neyret - 12/02/2020

\*/

#ifndef COSENSOR\_H

#define COSENSOR\_H

#include "Arduino.h"

class COSensor**{**

public**:**

COSensor**();**

void setPin**(**int pinSensor**);**

bool detecteCo**();**

void calibCo**();**

private**:**

bool \_isConfigured**;**

int \_pinSensor**;**

**};**

#endif // COSENSOR\_H

### cosensor.cpp

/\*

\* Co.cpp - Librairie de gestion d'un capteur CO

\* Créé par Hugo Rodrigues/Anthony Neyret - 12/02/2020

\*/

#include "cosensor.h"

#include "Arduino.h"

COSensor**::**COSensor**()**

**{**

\_isConfigured **=** **false;**

**}**

void COSensor**::**setPin**(**int pinSensor**){**

pinMode **(**pinSensor**,** INPUT**);**

\_pinSensor**=**pinSensor**;**

\_isConfigured **=** **true;**

**}**

bool COSensor**::**detecteCo**(){**

int val **=** analogRead **(**\_pinSensor**);**

Serial**.**println**(**"valeur CO:" **+** String**(**val**));**

**if** **(**\_isConfigured **&&** val **>** 350**)**

**return** **true;**

**else**

**return** **false;**

**}**

### IRSensor.h

/\*

\* IRSensor.h - Librairie de gestion d'un capteur IR

\* Créé par Hugo Rodrigues/Anthony Neyret - 14/02/2021

\*/

#ifndef Sensor\_h

#define Sensor\_h

#include "Arduino.h"

class IRSensor **{**

public **:**

IRSensor**();**

void setPin**(**int pinIr**);**

bool isBlackLine**();**

private **:**

bool \_isConfigured**;**

int \_pinIr**;**

**};**

#endif

### IRSensor.cpp

/\*

\* IRSensor.cpp - Librairie de gestion d'un capteur IR

\* Créé par Hugo Rodrigues/Anthony Neyret - 14/02/2021

\*/

#include "Arduino.h"

#include "IRSensor.h"

IRSensor**::**IRSensor**(){**

\_isConfigured **=** **false;**

**}**

void IRSensor**::**setPin**(**int pinIr**){**

pinMode **(**pinIr**,** INPUT**);**

\_pinIr**=**pinIr**;**

\_isConfigured **=** **true;**

**}**

bool IRSensor**::**isBlackLine**(){**

**if** **(**\_isConfigured **&&** digitalRead**(**\_pinIr**)==**HIGH**)**

**return** **true;**

**else**

**return** **false;**

**}**

### jail.h

/\*

\* Jail.h - Librairie de gestion de la prison

\* Créé par Hugo Rodrigues/Anthony Neyret - 14/02/2021

\*/

#ifndef JAIL\_H

#define JAIL\_H

#include "Near\_sensor.h"

class Jail**{**

public**:**

Jail**();**

bool isInJail**();**

void setPins**(**int pinNear1**,** int pinNear2**,** int pinNear3**,** int pinNear4**);**

private**:**

Near\_sensor \_Near\_sensor1**;**

Near\_sensor \_Near\_sensor2**;**

Near\_sensor \_Near\_sensor3**;**

Near\_sensor \_Near\_sensor4**;**

bool \_isConfigured**;**

**};**

#endif // JAIL\_H

### jail.cpp

/\*

\* Jail.cpp - Librairie de gestion de la prison

\* Créé par Hugo Rodrigues/Anthony Neyret - 14/02/2021

\*/

#include "jail.h"

#include "Arduino.h"

Jail**::**Jail**(){**

\_isConfigured **=** **false;**

\_Near\_sensor1 **=** Near\_sensor**();**

**}**

void Jail**::**setPins**(**int pinNear1**,** int pinNear2**,** int pinNear3**,** int pinNear4**){**

\_Near\_sensor1**.**setPin**(**pinNear1**);**

\_Near\_sensor2**.**setPin**(**pinNear2**);**

\_Near\_sensor3**.**setPin**(**pinNear3**);**

\_Near\_sensor4**.**setPin**(**pinNear4**);**

\_isConfigured **=** **true;**

**}**

bool Jail**::**isInJail**(){**

int compteur**=**0**;**

**if** **(**\_Near\_sensor1**.**isNear**())**

compteur**++;**

**if** **(**\_Near\_sensor2**.**isNear**())**

compteur**++;**

**if** **(**\_Near\_sensor3**.**isNear**())**

compteur**++;**

**if** **(**\_Near\_sensor4**.**isNear**())**

compteur**++;**

**if(**\_isConfigured **&&** compteur**>=**3**)** **{**

Serial**.**println**(**"3 capteurs sur4 détectent la prison"**);**

**return** **true;**

**}**

**else**

**return** **false;**

**}**

### Moteur.h

/\*

\* Moteur.h - Librairie de gestion d'un moteur DC

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#ifndef Moteur\_h

#define Moteur\_h

#include "Arduino.h"

class Moteur **{**

public **:**

Moteur**();**

void setPins **(**int pinActivation**);**

void go**(**int vitesse**);**

void stop**();**

private **:**

bool \_isConfigured**;**

int \_pinActivation**;**

**};**

#endif

### Moteur.cpp

/\*

\* Moteur.cpp - Librairie de gestion d'un moteur DC

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#include "Arduino.h"

#include "Moteur.h"

Moteur**::**Moteur**(){**

\_isConfigured **=** **false;**

**}**

void Moteur**::**setPins**(**int pinActivation**){**

// Configuration des pins moteur

// Utilisation de pin PWM pour contrôler la vitesse des moteurs

pinMode **(**pinActivation**,** OUTPUT**);**

\_pinActivation **=** pinActivation**;**

\_isConfigured **=** **true;**

**}**

void Moteur**::**go**(**int vitesse**)** **{**

**if** **(**\_isConfigured**)** **{**

Serial**.**println **(**"Go motor, speed"**+**String**(**vitesse**)+**"%"**);**

// Vitesse de 80%

analogWrite**(**\_pinActivation**,** map **(**vitesse**,** 0**,** 100**,** 0**,** 255**));**

**}**

**}**

void Moteur**::**stop**()** **{**

**if** **(**\_isConfigured**)** **{**

Serial**.**println**(**"Stop motor"**);**

// Stop moteur

analogWrite**(**\_pinActivation**,** 0**);**

**}**

**}**

### Near\_sensor.h

#ifndef NEAR\_SENSOR\_H

#define NEAR\_SENSOR\_H

#include "Arduino.h"

class Near\_sensor**{**

public **:**

Near\_sensor**();**

void setPin**(**int pinNear**);**

bool isNear**();**

private **:**

bool \_isConfigured**;**

int \_pinNear**;**

**};**

#endif // NEAR\_SENSOR\_H

### Near\_sensor.cpp

/\*

\*Near\_Sensor.cpp - Librairie de gestion d'un capteur Ir de proximité

\* Créé par Hugo Rodrigues/Anthony Neyret - 14/02/2021

\*/

#include "Arduino.h"

#include "Near\_sensor.h"

Near\_sensor**::**Near\_sensor**(){**

\_isConfigured **=** **false;**

**}**

void Near\_sensor**::**setPin**(**int pinNear**){**

pinMode **(**pinNear**,** INPUT**);**

\_pinNear**=**pinNear**;**

\_isConfigured **=** **true;**

**}**

bool Near\_sensor**::**isNear**(){**

//d'apres l abbaque 0.7V correspond à 18 cm

**if** **(**\_isConfigured **&&** analogRead**(**\_pinNear**)** **>** 1023**\***0.7**/**5**)**

**return** **true;**

**else**

**return** **false;**

**}**

### SpeedBox.h

/\*

\* SpeedBox.h - Librairie de gestion de la boite de vitesses contrôlant le sens des moteurs

\* La boite de vitesse s'appuie sur un multiplexeur de type 74HC595

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#ifndef SpeedBox\_h

#define SpeedBox\_h

#include "Arduino.h"

#include "Moteur.h"

class SpeedBox **{**

public **:**

SpeedBox**();**

void setPins **(**int pinLatch**,** int clockPin**,** int dataPin**);**

void addMotor1**(**int pinActivation**);**

void addMotor2**(**int pinActivation**);**

void addMotor3**(**int pinActivation**);**

void avance**();**

void recule**();**

void tourneDroite**();**

void tourneGauche**();**

void ajusteDroite**();**

void ajusteGauche**();**

void stop**();**

private **:**

bool \_isConfigured**;**

int \_pinLatch**;**

int \_pinClock**;**

int \_pinData**;**

Moteur \_moteur1**;**

Moteur \_moteur2**;**

Moteur \_moteur3**;**

// Bits 5 et 6 gère le sens de rotation du moteur 1

// 5 LOW, 6 HIGH : Sens rotation horaire

// 5 HIGH, 6 LOW : Sens rotation anti-horaire

int \_moteur1Horaire **=** 0b00100000 **;** // Q6 sur le 74HC595

int \_moteur1AntiHoraire **=**0b01000000**;** // Q5 sur le 74HC595

// Bits 3 et 4 gère le sens de rotation du moteur 2

// 4 LOW, 3 HIGH : Sens rotation horaire

// 4 HIGH, 3 LOW : Sens rotation anti-horaire

int \_moteur2Horaire **=** 0b00001000**;** // Q4 sur le 74HC595

int \_moteur2AntiHoraire **=** 0b00010000**;** // Q3 sur le 74HC595

// Bits 1 et 2 gère le sens de rotation du moteur 3

// 2 LOW, 1 HIGH : Sens rotation horaire

// 2 HIGH, 1 LOW : Sens rotation anti-horaire

int \_moteur3Horaire **=** 0b00000010**;** // Q2 sur le 74HC595

int \_moteur3AntiHoraire **=** 0b00000100**;** // Q1 sur le 74HC595

void UpdateMutex**(**int data**);**

**};**

#endif

### SpeedBox.cpp

/\*

\* SpeedBox.cpp - Librairie de gestion de la boite de vitesses contrôlant le sens des moteurs

\* La boite de vitesse s'appuie sur un multiplexeur de type 74HC595

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#include "Arduino.h"

#include "Moteur.h"

#include "SpeedBox.h"

SpeedBox**::**SpeedBox**()** **{**

\_isConfigured **=** **false;**

\_moteur1 **=** Moteur**();**

\_moteur2 **=** Moteur**();**

\_moteur3 **=** Moteur**();**

\_pinLatch **=** 0**;**

\_pinClock **=** 0**;**

\_pinData **=** 0**;**

**}**

void SpeedBox**::**setPins **(**int pinLatch**,** int pinClock**,** int pinData**)** **{**

// Configuration des pins moteur

// Utilisation de pin PWM pour contrôler la vitesse des moteurs

pinMode **(**pinLatch**,** OUTPUT**);**

pinMode **(**pinClock**,** OUTPUT**);**

pinMode **(**pinData**,** OUTPUT**);**

\_pinLatch **=** pinLatch**;**

\_pinClock **=** pinClock**;**

\_pinData **=** pinData**;**

\_isConfigured **=** **true;**

**}**

void SpeedBox**::**addMotor1**(**int pinActivation**)** **{**

\_moteur1**.**setPins **(**pinActivation**);**

**}**

void SpeedBox**::**addMotor2**(**int pinActivation**)** **{**

\_moteur2**.**setPins **(**pinActivation**);**

**}**

void SpeedBox**::**addMotor3**(**int pinActivation**)** **{**

\_moteur3**.**setPins **(**pinActivation**);**

**}**

void SpeedBox**::**avance**()** **{**

// Set moteur1horaire + moteur2anti\_horaire;

UpdateMutex**(**\_moteur1Horaire **|** \_moteur2AntiHoraire **|** \_moteur3Horaire**);**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**go**(**92**);**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**go**(**65**);**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**stop**();**

**}**

void SpeedBox**::**recule**()** **{**

// Set moteur1antihoraire + moteur2horaire;

UpdateMutex**(**\_moteur1AntiHoraire **|** \_moteur2Horaire **|** \_moteur3Horaire**);**

\_moteur1**.**go**(**60**);**

\_moteur2**.**go**(**60**);**

\_moteur3**.**stop**();**

**}**

void SpeedBox**::**ajusteDroite**(){**

// Set moteur123antihoraire +

UpdateMutex**(**\_moteur1Horaire **|** \_moteur2AntiHoraire**|** \_moteur3AntiHoraire**);**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**go**(**92**);**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**go**(**65**);**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**go**(**80**);**

**}**

void SpeedBox**::**tourneDroite**(){**

// Set moteur123antihoraire +

UpdateMutex**(**\_moteur1AntiHoraire **|** \_moteur2AntiHoraire**|** \_moteur3AntiHoraire**);**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**go**(**92**);**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**go**(**65**);**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**go**(**80**);**

**}**

void SpeedBox**::**tourneGauche**(){**

// Set moteur123horaire +

UpdateMutex**(**\_moteur1Horaire **|** \_moteur2Horaire **|** \_moteur3Horaire**);**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**go**(**92**);**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**go**(**65**);**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**go**(**80**);**

**}**

void SpeedBox**::**ajusteGauche**(){**

// Set moteur123horaire +

UpdateMutex**(**\_moteur1Horaire **|** \_moteur2AntiHoraire **|** \_moteur3Horaire**);**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**go**(**92**);**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**go**(**65**);**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**go**(**80**);**

**}**

void SpeedBox**::**stop**(){**

Serial**.**println**(**"moteur 1"**);**

\_moteur1**.**stop**();**

Serial**.**println**(**"moteur 2"**);**

\_moteur2**.**stop**();**

Serial**.**println**(**"moteur 3"**);**

\_moteur3**.**stop**();**

**}**

void SpeedBox**::**UpdateMutex**(**int data**)** **{**

Serial**.**print**(**"data for mutex: 0b"**);**

Serial**.**print**(**data**,** BIN**);**

Serial**.**print**(**" (0x"**);**

Serial**.**print**(**data**,** HEX**);**

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

// On désactive le 74HC595

digitalWrite**(**\_pinLatch**,** LOW**);**

// Mise à jour du registre à décalage

shiftOut**(**\_pinData**,** \_pinClock**,** MSBFIRST**,** data**);**

// On active le 74HC595

digitalWrite**(**\_pinLatch**,** HIGH**);**

**}**

### Robot.h

/\*

\* Robot.h - Librairie de gestion du robot

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#ifndef Robot\_h

#define Robot\_h

#include "Arduino.h"

#include "IRSensor.h"

#include "Near\_sensor.h"

#include "jail.h"

#include "SpeedBox.h"

#include "cosensor.h"

#include "contactsensor.h"

class Robot **{**

public **:**

Robot **();**

bool detecteCo**();**

bool detecteContact**();**

void avance**();**

void recule**();**

void tourneDroite**();**

void tourneGauche**();**

void Robot**::**ajusteDroite**();**

void Robot**::**ajusteGauche**();**

void stop**();**

void addSensorFront**(**int pinIr**);**

void addSensorLeft**(**int pinIr**);**

void addSensorRight**(**int pinIr**);**

void addJail**(**int pinNear1**,** int pinNear2**,** int pinNear3**,** int pinNear4**);**

bool isInJail**();**

void addSpeedBox **(**int pinLatch**,** int pinClock**,** int pinData**);**

void addCOSensor**(**int pinCOSensor**);**

void addContact**(**int pinContact**);**

IRSensor IRSensorFront**;**

IRSensor IRSensorLeft**;**

IRSensor IRSensorRight**;**

SpeedBox speedBox**;**

COSensor \_COSensor**;**

contactsensor \_ContactSensor**;**

private **:**

Jail \_jail**;**

**};**

#endif

### 

### Robot.cpp

/\*

\* Robot.cpp- Librairie de gestion du Robot

\* Créé par Hugo Rodrigues/Anthony Neyret - 26/12/2020

\*/

#include "Arduino.h"

#include "Robot.h"

#include "Moteur.h"

#include "IRSensor.h"

#include "Near\_sensor.h"

#include "jail.h"

#include "SpeedBox.h"

#include "cosensor.h"

#include "contactsensor.h"

Robot**::**Robot**()** **{**

IRSensorFront **=** IRSensor**();**

IRSensorLeft **=** IRSensor**();**

IRSensorRight **=** IRSensor**();**

\_jail **=** Jail**();**

speedBox **=** SpeedBox**();**

\_COSensor **=** COSensor**();**

\_ContactSensor **=** contactsensor**();**

**}**

void Robot**::**addContact**(**int pinContact**){**

\_ContactSensor**.**setPin**(**pinContact**);**

**}**

void Robot**::**addCOSensor**(**int pinCOSensor**){**

\_COSensor**.**setPin**(**pinCOSensor**);**

**}**

void Robot**::**addSpeedBox **(**int pinLatch**,** int pinClock**,** int pinData**)** **{**

speedBox**.**setPins**(**pinLatch**,** pinClock**,** pinData**);**

**}**

void Robot**::**addJail**(**int pinNear1**,** int pinNear2**,** int pinNear3**,** int pinNear4**){**

\_jail**.**setPins**(**pinNear1**,** pinNear2**,** pinNear3**,** pinNear4**);**

**}**

void Robot**::**addSensorFront**(**int pinIr**){**

IRSensorFront**.**setPin**(**pinIr**);**

**}**

void Robot**::**addSensorLeft**(**int pinIr**){**

IRSensorLeft**.**setPin**(**pinIr**);**

**}**

void Robot**::**addSensorRight**(**int pinIr**){**

IRSensorRight**.**setPin**(**pinIr**);**

**}**

bool Robot**::**detecteContact**(){**

**return** \_ContactSensor**.**detectcontact**();**

**}**

bool Robot**::**detecteCo**(){**

**return** \_COSensor**.**detecteCo**();**

**}**

void Robot**::**avance**()** **{**

speedBox**.**avance**();**

**}**

void Robot**::**recule**()** **{**

speedBox**.**recule**();**

**}**

void Robot**::**tourneDroite**()** **{**

speedBox**.**tourneDroite**();**

**}**

void Robot**::**tourneGauche**()** **{**

speedBox**.**tourneGauche**();**

**}**

void Robot**::**ajusteDroite**()** **{**

speedBox**.**ajusteDroite**();**

**}**

void Robot**::**ajusteGauche**()** **{**

speedBox**.**ajusteGauche**();**

**}**

void Robot**::**stop**()** **{**

speedBox**.**stop**();**

**}**

bool Robot**::**isInJail**()** **{**

**return** \_jail**.**isInJail**();**

**}**