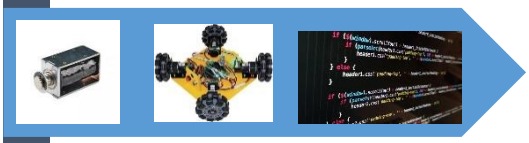
ynov
AÉRONAUTIQUE &
SYSTÈMES EMBARQUÉS

2020
RoboCup
BORDEAUX - FRANCE



# Robot footballeur Holonome

Cadre : RoboCup 2020


Groupe 1 : Samuel Huet, Thomas Coutant, Jean Gabriel Massicot, Antoine Venier.

Antoine Venier

2017/2018

# Table des matières

# 1. Introduction

Notre projet de fin de troisième année consistait à réaliser par groupe de quatre un robot répondant aux dernières règles en vigueur définies pour la Small Size League. L'objectif de ce projet est de concevoir un robot de A à Z pouvant jouer au football. Le robot doit être capable de se déplacer dans toutes les directions, de dribler et de taper la balle. Pour cela une réparation des tâches a été effectuée au sein de l'équipe afin de déterminer le rôle de chaque membre. Le robot doit également être piloté à distance. Toutes ses exigences, nous ont permis de faire appel à beaucoup de connaissances vues en cours et d'appliquer ce que l'on a pu voir de manière concrète. Ce projet a fait appel à nos compétences techniques, mais aussi d'organisation, car chacune des parties du groupe devant se regrouper, il était primordial de communiquer entre nous durant tout le long du projet. En résumé, le robot devait être alimenté, donc autonome en énergie, il devait être capable de se déplacer en répondant à une consigne communiquée par le pilote du robot. Il y a donc une partie hardware, pour l'alimentation du robot, une partie software pour la gestion intelligente de ces déplacements et une partie mécanique pour le châssis du robot ainsi que les éléments qui vont lui permettre de dribler et frapper la balle.

# 2. Répartitions des tâches

Au début du projet un ensemble de feuilles nous a été remise pour nous informer des différentes fonctions que devait réaliser le robot, une répartition des tâches était également proposée :

| Fonction | Détails |
|---|---|
| FP5 | Brushless+roues+Encodeur+ESC |
| FP3 | Gestion moteur haut niveau+calcul du déplacement + capteurs optiques |
| FP6 | Dribleur+kicker |
| FP1+FP2+FP4 | Com RF+ Carte-mère |

Nous avons décidé de suivre cet exemple et de nous répartir les différentes tâches entre nous.

| Fonction | Détails | |
|---|---|---|
| FP5 | Brushless+roues+Encodeur+ESC | Jean-Gabriel |
| FP3 | Gestion moteur haut niveau+calcul du déplacement + capteurs optiques | Moi |
| FP6 | Dribleur+kicker | Thomas |
| FP1+FP2+FP4 | Com RF+ Carte-mère | Samuel |

Je suis donc chargé de la partie programmation des déplacements du robot.

En quoi cela consiste ?

Ma partie consiste à faire en sorte que le robot puisse se déplacer le mieux possible. Pour cela différents capteurs ont été utiles pour connaître la vitesse du robot et pouvoir par la suite ajuster celle-ci pour répondre le mieux possible à une consigne envoyée par l'ordinateur qui contrôle le robot.

Matériels utilisés :

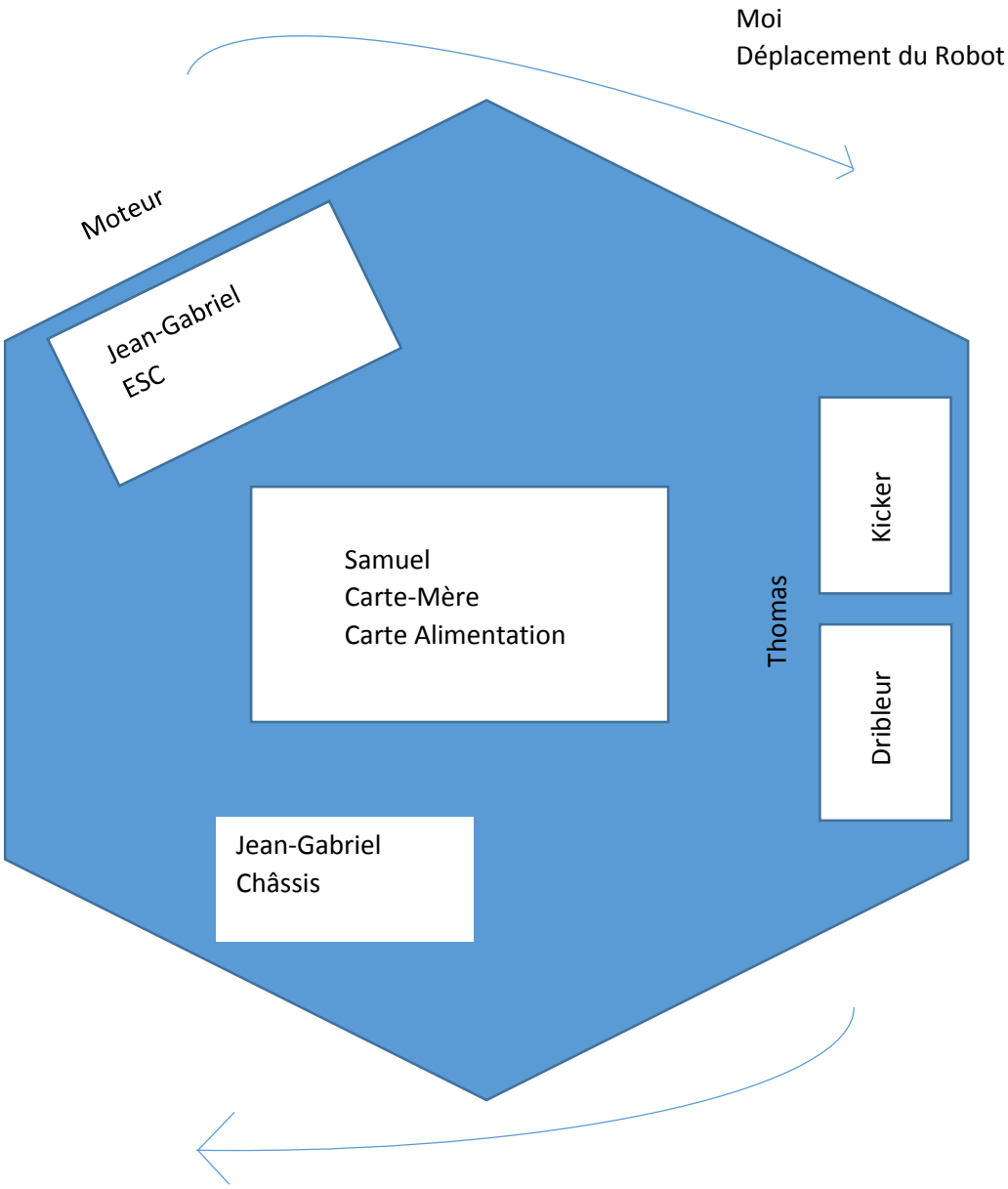
Pour réaliser à bien ma partie j'ai eu besoin de commander :

- Une centrale inertielle
- Deux capteurs optiques
- Des Encodeurs magnétiques

Pour tester ces capteurs j'ai utilisé :

- Une carte d'évaluation de l'école qui intègre un microcontrôleur MCS09QE8
- Une carte 'évaluation pour le protocole I2C
- Un moteur brushless de modélisme pour tester la pwm.
- Putty pour afficher les données des capteurs sur un terminal
- CodeWarriors pour programmer le microcontrolleur

Schéma répartition des tâches :

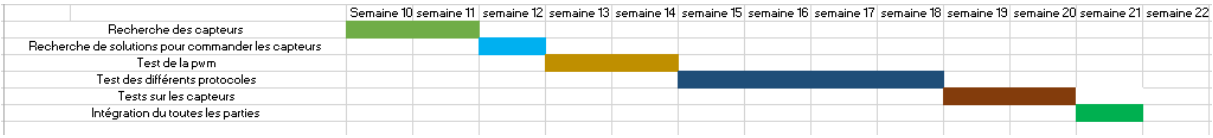



Samuel
Communication Radio

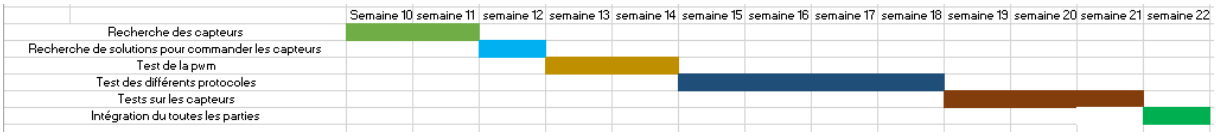
## 3. Gestion du projet :

Les plannings commencent le 6 mars 2018 et finissent le 15 Juin 2018

Planning Prévisionnel




Planning Réel




Sur ce planning la dernière semaine n'est plu libre elle servait de sécurité en cas d'imprévu.

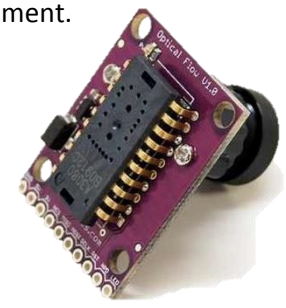
## 4. Recherche des capteurs :

Les capteurs occupant une grande place dans ma partie, je devais bien les choisir, pour que je puisse les utiliser. Le premier critère de sélection d'un capteur a été son protocole de communication, est ce que c'était un protocole que je connaissais ou bien un nouveau protocole. Bien sûr si je ne trouvais aucun capteur répondant à ce critère j'utilisais un capteur communicant avec un protocole qui m'était inconnu, mais cela aurait demandé plus de temps. Ensuite le deuxième critère était l'encombrement, en effet le robot étant restreint à une taille minimum je devais faire en sorte de trouver des capteurs qui répondaient à cette exigence.

### 4.1 Les capteurs Optiques :

Pour connaitre la vitesse du robot l'utilisation de capteurs optiques a été suggérée. Ce sont les mêmes capteurs optiques que l'on trouve dans les souris optiques ou dans certains appareils photo. Leur but étant de prendre des images à un intervalle de temps régulier et ainsi détecter un déplacement.

Les recherches pour ce capteur ont été compliquées, car bien qu'il y ait de nombreuses souris d'ordinateur qui en utilise, peu de capteurs sont vendus séparément. Une solution pour régler ce problème était de démonter une souris optique et de récupérer le capteur à l'intérieur. Le protocole de communication n'était pas un problème tous les capteurs trouvés communiquaient en SPI un protocole vu en cours. Après quelques recherches, j'ai finalement réussi à trouver un capteur disponible en stock l'ADNS3080, il était vendu sur une carte avec une lentille et des headers pour faciliter le branchement.



**Features**

- High speed motion detection – up to 40 ips and 15g
- New architecture for greatly improved optical navigation technology
- Programmable frame rate over 6400 frames per second
- SmartSpeed self-adjusting frame rate for optimum performance
- Serial port burst mode for fast data transfer
- 400 or 1600 cpi selectable resolution
- Single 3.3 volt power supply
- Four-wire serial port along with Chip Select, Power Down, and Reset pins

## 4.2 La centrale inertielle :

Ce capteur a pour but de faire redondance aux capteurs optiques et ainsi donnée une information plus précise de la vitesse du robot en détectant si le robot fait du sur place par exemple. Ce capteur a été trouvé assez facilement car beaucoup de centrale inertielle fonctionne en I2C, un protocole vu en cours. J'ai une des versions la plus récente des MP60x0 soit la MPU-6050.


**5.1 Gyroscope Features**
The triple-axis MEMS gyroscope in the MPU-60X0 includes a wide range of features:

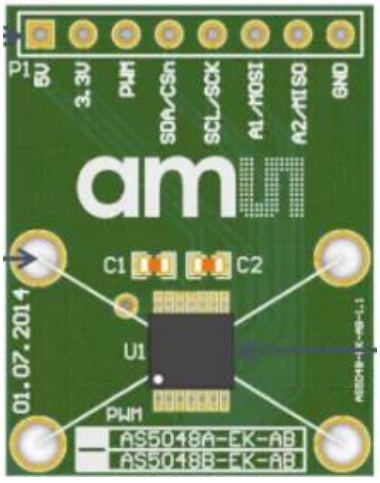
- Digital-output X-, Y-, and Z-Axis angular rate sensors (gyroscopes) with a user-programmable full-scale range of ±250, ±500, ±1000, and ±2000°/sec
- External sync signal connected to the FSYNC pin supports image, video and GPS synchronization
- Integrated 16-bit ADCs enable simultaneous sampling of gyros
- Enhanced bias and sensitivity temperature stability reduces the need for user calibration
- Improved low-frequency noise performance
- Digitally-programmable low-pass filter
- Gyroscope operating current: 3.6mA
- Standby current: 5μA
- Factory calibrated sensitivity scale factor
- User self-test

**5.2 Accelerometer Features**
The triple-axis MEMS accelerometer in MPU-60X0 includes a wide range of features:

- Digital-output triple-axis accelerometer with a programmable full scale range of ±2*g*, ±4*g*, ±8*g* and ±16*g*
- Integrated 16-bit ADCs enable simultaneous sampling of accelerometers while requiring no external multiplexer
- Accelerometer normal operating current: 500μA
- Low power accelerometer mode current: 10μA at 1.25Hz, 20μA at 5Hz, 60μA at 20Hz, 110μA at 40Hz
- Orientation detection and signaling
- Tap detection
- User-programmable interrupts
- High-G interrupt
- User self-test

## 4.3 Encodeur magnétique :

Les encodeurs qui seront au nombre de quatre seront placés proches de chaque moteur, ils sont associés à un aimant qui est collé sur l'axe du moteur pour nous donner une indication sur la vitesse du moteur et ainsi pour ajuster sa vitesse s'il elle ne correspond pas à la consigne. C'est Jean-Gabriel qui c'est charger de choisir les encodeurs et moi je les ai testés avec la carte d'évaluation de l'école. Il a opté pour l'AS5048A.



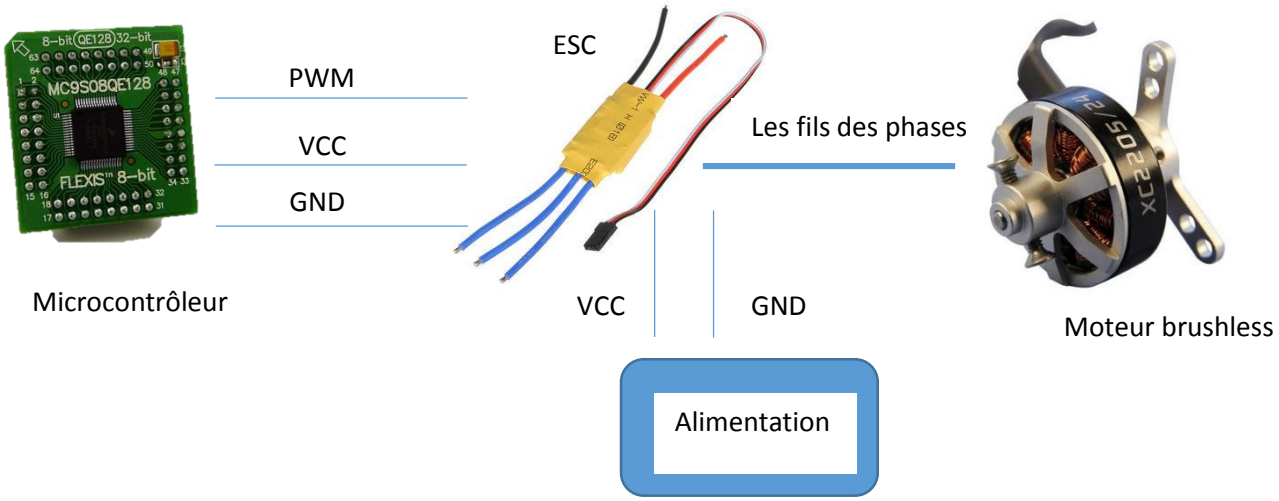

## Product parameters

| | |
|---|---|
| **Resolution [bit]** | 14 |
| **Interfaces** | SPI |
| **Output** | Digital angle (interface), PWM |
| **Max. Speed [rpm]** | |
| **Overvoltage Protection** | |
| **Redundant** | |
| **Supply Voltage [V]** | 3.3 or 5.0 |
| **Temperature Range [°C]** | -40 to +150 |
| **Package** | TSSOP-14 |
| **Automotive Qualified** | |

# 5. Gestion des déplacements

PWM moteur Brushless :

Le moteur brushless va nous permettre dans notre projet de faire avancer les roues de notre robot, ces moteurs ont besoin d'un ESC (Electronic Stability Controller). Le microcontrôleur envoie une pwm à l'ESC et lui se charge de convertir cette pwm en une alimentation des différentes phases du moteur à une certaine vitesse pour répondre à la pwm envoyé par le microcontrôleur.




## 5.1 Moteur Brushless :

Définition : Le moteur de type Brushless se comporte comme un moteur à courant continu traditionnel. Il présente des caractéristiques semblables à celles des moteurs courant continu et alternatif sans les inconvénients : une forte dynamique de vitesse et d'accélération sans l'usure mécanique des moteurs courant continu ; la commutation électronique se substituant à la commutation mécanique. La durée de vie du moteur brushless se voit alors limitée uniquement par la durée de vie de ses roulements. Les caractéristiques principales du moteur brushless sont une durée de vie exceptionnelle (jusqu'à plusieurs dizaines de milliers d'heures), une forte constante de couple, une puissance importante et une haute vitesse de fonctionnement.

Définition tiré du site : http://www.mdp.fr/documentation/lexique/brushless/definition.html

### 5.1.1 La PWM

Etude de registre pwm

Registre PWM pour le microcontrôleur MC9S08Q8RM de la carte d'évaluation de l'école :

- TPMxSC

-TPMxCNTH :TPMxCNTL

-TPMxMODH :TPMxMODL

-TPMxCnSC

-TPMxCnVH :TPMxCnVL

Le registre TPMxSC : Ce registre configure le l'horloge du timer utilisé.

TOF : Ce bit nous indique si le compteur TPM a atteint ou dépassé la valeur préprogrammé

TOIE : Ce bit autorise ou pas l'utilisation du timer en mode interruption.

CPWMS : Sélection du mode edge-aligned(0) ou center-aligned(1)

CLKSB : CLKSA Ces 2 bits sélectionnent l'horloge du timer (bus clock, Fixed frequency clock, External clock)

PS [2 :0] : Ces 3 bits sélectionnent la valeur de la division de l'horloge.

TPMxCNTH : TPMxCNTL : Ces deux registres de 8 bits contiennent les valeurs hautes et basses du compteur TPM.

TPMxMODH : TPMxMODL : Ces deux registres vont nous permettre de gérer la période de notre PWM.

TPMxCnSC : Ce registre contient les bits de statuts des canaux d'interruptions.

-TPMxCnVH :TPMxCnVL : Ce registre nous permet d'entrer la valeur souhaitée pour le rapport cyclique de notre PWM.

### 5.1.2 Test Moteur Brushless

Test moteur brushless de modélisme avec la carte d'évaluation de l'école

Code d'initialisation PWM

```
/*Fonction d'initialisation de la PWM*/
void initPWM(void){

        TPM1MOD=0x02CA;// periode 20ms
        TPM1SC=0x0F;//horloge du bus divisée par 128
        TPM1C2SC=0x28;//edge algned

}
```

Le moteur de modélisme utilisé pour le test a besoin d'une pwm de 50Hz pour fonctionner, c'est pour cela que j'ai configuré la période à 20ms.

Code d'initialisation moteur

```
void init_moteur(void){
        RapportCyclique(1);//rapport cyclique de 0
        delay(5000);
        RapportCyclique(29);//Ton  pendant 4% de la periode soit 800µs
        delay(50000);//Attendre les 4 beep du moteur
}
```

Pour initialiser le moteur de modélisme, il faut lui envoyer dans un premier temps une pwm avec un rapport cyclique de 0, ensuite on lui envoie une pwm avec un faible rapport cyclique et on attend que le moteur émette 4 beeps sonores consécutifs. Une fois cela terminé le moteur est initialisé et on peut lui envoyer par la suite le rapport cyclique que l'on veut.

Code rapport cyclique

```
/*fonction qui prend en paramètre le rapport cyclique voulue*/
void RapportCyclique(unsigned short val){

        /*Rapport cyclique de 43.9%
        205*0.439=90(10)=(16)
        unsigned char valeur;
        valeur à rentrer dans le registre TPM1C1 pour obtenir le rapport cyclique souhaité
        valeur=205*Ton;*/

        TPM1C2V=val;
}
```

Fonction principale :

```
void main(void)
{
  /* Write your local variable definition here */
   uint16_t value;
   PTCDD=0xFF;
   PTCD=0xFF;
  //PTADD_PTADD3=1;
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */

  /*initPWM();
  init_moteur();*/




  for(;;){
        RapportCyclique(value);//Ton  pendant 6%
  }

 }
```

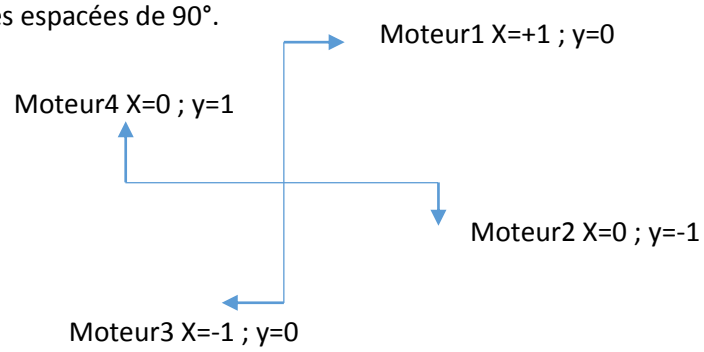
Résultat

J'ai donc branché les entrées pwm de l'ESC à la carte d'évaluation de l'école  et je l'ai alimenté avec une alimentation de l'école en 9V. Le moteur tourne bien.

## 5.2 Calcul de trajectoire

Explication :

Le robot comporte quatre roues holonomes, et pour qu'il aille dans une direction précise à une vitesse précise, il faut que chacun de ses moteurs aille à une vitesse précise par rapport aux autres. Pour comprendre un peu mieux ce principe j'ai d'abord pris l'exemple d'un robot dont les roues étaient toutes espacées de 90°.




Chaque moteur possède un vecteur unitaire. Ces vecteurs vont nous permettre de déterminer la vitesse de chaque moteur pour aller dans une direction donnée.

Pour simuler la vitesse des différents moteurs j'ai utilisé le logiciel Excel. Voici ce que ça donne pour le premier cas, c'est-à-dire tous les moteurs sont espacés de 90°.

| | dx | dy | | | | dx | dy | Norme |
|---|---|---|---|---|---|---|---|---|
| Moteur1 | 0 | 1 | | | Vitesse | 0 | 1 | 1 |
| Moteur2 | 1 | 0 | | | | | | |
| Moteur3 | 0 | -1 | | | | | | |
| Moteur4 | -1 | 0 | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | Résultat | | | |
| | | | | Moteur1 | 0,5 | | | |
| | | | | Moteur2 | 0 | | | |
| | | | | Moteur3 | -0,5 | | | |
| | | | | Moteur 4 | 0 | | | |

La ligne vitesse en dx et dy nous permet de donner une direction au robot ensuite, de ces deux valeurs est calculé une norme (ou vitesse) grâce à la formule :
Racine $((dx)^2+(dy)^2)$
Les valeurs de dx et dy des moteurs 1, 2,3 et 4 correspondent aux valeurs de leur vecteur unitaire. La colonne résultat correspond à la vitesse en m/s de chaque moteur pour que le robot puisse aller dans la bonne direction à la bonne vitesse.

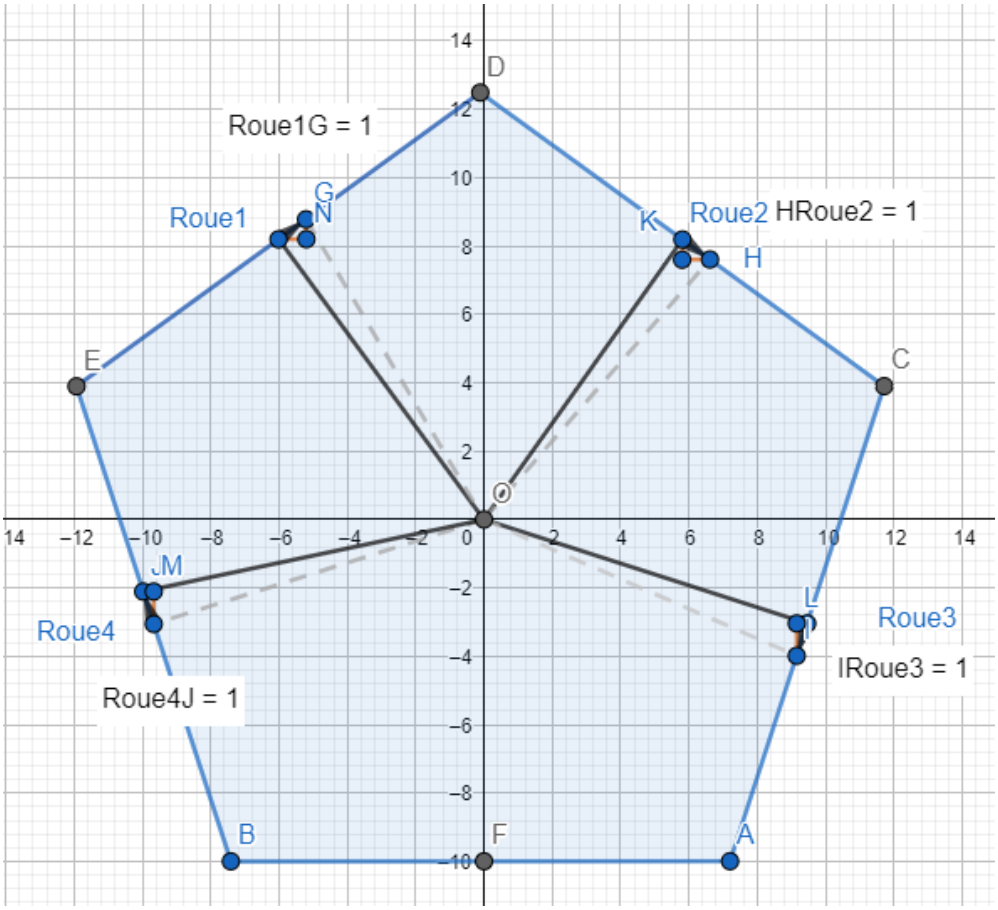
Pour calculer la vitesse en m/s de chaque moteur, le calcul est le suivant (pour le moteur 1 par exemple) :
Vitesse (m/s) = (norme)*((dx (de la norme) * dx (du moteur 1))+(dy (de la norme) * dy (du moteur 1)))

On multiple la norme par le produit de  la valeur du dx correspondant au dx du vecteur unitaire du moteur 1 et le dx de la norme du robot et on l'additionne à la valeur du dy correspondant au dy du vecteur unitaire du moteur 1 au dy de la norme du robot.

Condition réelle :

Une fois la simulation pour un cas simple effectué. J'ai demandé à Jean-Gabriel qui était en charge de la partie mécanique du robot de bien vouloir me donner les différentes dimensions du châssis du robot pour que mes calculs soient le plus précis possible. Une fois que je connaissais toutes les dimensions du châssis du robot, j'ai pu faire un petit schéma sur géogebra pour que ce soit plus parlant.

Schéma sur géogebra



Fichier Excel associé

| | | Configuration réelle du robot | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | dx | dy | | | | dx | dy | Norme |
| Moteur1 | 0,81 | 0,58 | | | vitesse | 0 | 1 | 1 |
| Moteur2 | 0,81 | -0,59 | | | | | | |
| Moteur3 | -0,31 | -0,95 | | | | | | |
| Moteur4 | -0,31 | 0,95 | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | Résultat(m/s) | | Vitesse (tr/min) | |
| | | | | Moteur1 | 0,29 | | 0,9222 | |
| | | | | Moteur2 | -0,295 | | -0,9381 | |
| | | | | Moteur3 | -0,475 | | -1,5105 | |
| | | | | Moteur4 | 0,475 | | 1,5105 | |

Cette fois-ci la valeur des vecteurs unitaires de chaque moteur est différente. Cette valeur dépend de l'angle des roues et de la distance du moteur au centre du robot. Les équations pour la norme du robot et la vitesse en m/s de chaque moteur sont les mêmes, mais j'ai rajouté une colonne qui correspond à la vitesse de chaque moteur en tour/s car c'est cette vitesse-là que je vais utiliser dans mes programmes.
Pour calculer la vitesse des moteurs en tours par seconde, j'ai eu besoin de connaître leurs diamètres pour pouvoir en déduire un périmètre et ainsi connaitre la distance (en mètres) parcourue par un moteur pendant un tour. J'ai ensuite converti les m/s en t/s grâce à cette formule :

Vitesse en tour/seconde = vitesse en (m/s) / la distance (en mètre) parcourue par un moteur pendant un tour.

## 5.3 Le PID

Proportionnelle Intégrateur Dérivateur, Le PID va nous servir pour réguler la vitesse de notre robot, car les calculs théoriques faits précédemment ne nous permettrons peut-être pas d'être exactement à la vitesse souhaitée en pratique.

Comment fonctionne le PID :

J'ai lu sur le site : *http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/* que le PID ne nécessitait pas forcement de gros calculs. Pour cela il fallait bien comprendre comment il fonctionnait. Ce site prend l'exemple d'une voiture pour nous expliquer le PID.

Dans un premier temps le proportionnelle, imaginons que l'on veuille se maintenir à 130 km/h sur l'autoroute, nous allons accélérer jusqu'à arriver à cette vitesse et l'accélération va donc être proportionnelle à l'erreur entre la vitesse voulue et la vitesse réelle.

L'intégrateur, le problème en utilisant seulement le proportionnelle, c'est que la vitesse de la voiture risque de stagner en dessous de la vitesse souhaité, en effet une fois atteinte la vitesse souhaitée on lâche l'accélérateur pour ne pas la dépasser. Il va donc falloir qu'en plus d'accélérer proportionnellement à l'erreur commise, vous allez aussi mémoriser cette erreur au cours du temps.

Le dérivateur, cette partie du PID va nous permettre d'être plus performant, c'est-à-dire que l'on va anticiper notre vitesse, plus elle se rapproche de la consigne moins on accélère et moins elle se rapproche de la consigne moins on accélère. Cela va nous permettre d'atteindre la vitesse souhaité plus rapidement.

### 5.3.1 Implémentation du PID

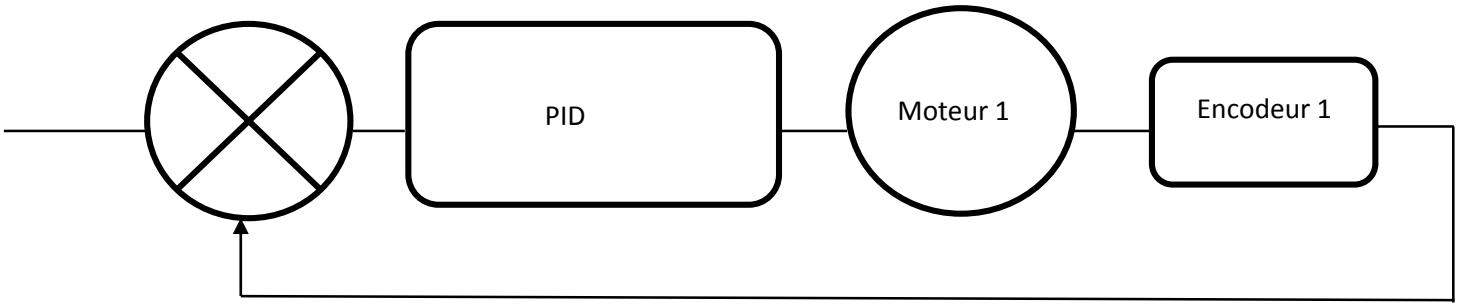
```
void main(){

    unsigned char erreur,consigne, mesure, somme_erreurs;
    unsigned char variation_erreur, erreur_précédente;
    unsigned char commande, Kp, Ki, Kd;
    //Tous les x millisecondes, faire :
    erreur = consigne - mesure;
    somme_erreurs += erreur;
    variation_erreur = erreur - erreur_précédente;
    commande = Kp * erreur + Ki * somme_erreurs + Kd * variation_erreur;
    erreur_précédente = erreur

}
```

Chacun des coefficients Kd, Kp et Ki seront déterminés par des tests sur le robot.

Schéma pour notre robot

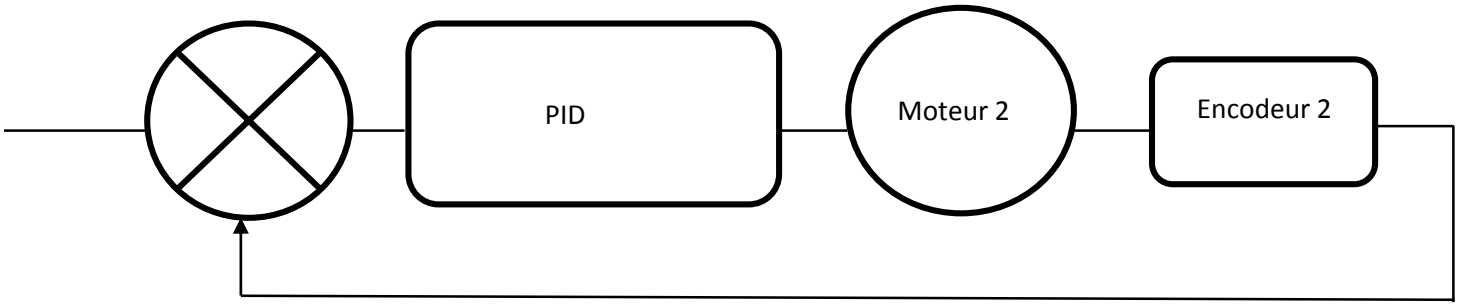
Le PID va servir à réguler la vitesse du robot et cette vitesse dépend de la vitesse des moteurs donc les capteurs qui vont nous être utiles sont les capteurs optiques pour la vitesse générale du robot et les encodeurs magnétiques pour la vitesse de chaque moteur.

Consigne de vitesse pour le moteur1



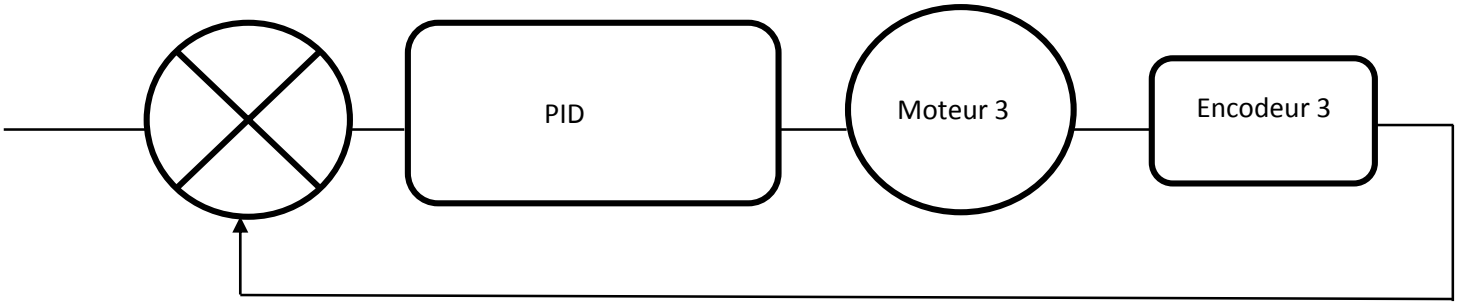

Erreur sur la consigne

Consigne de vitesse pour le moteur2



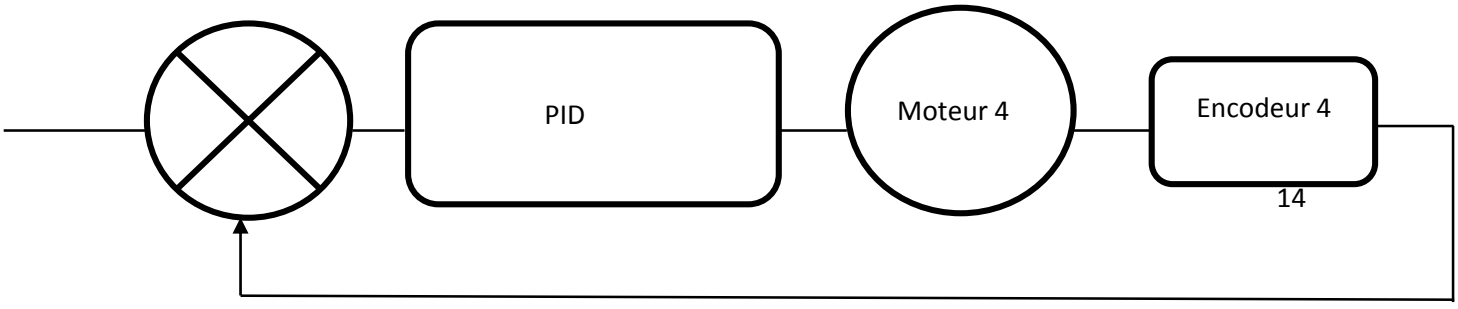

Erreur sur la consigne

Consigne de vitesse pour le moteur3




Erreur sur la consigne

Consigne de vitesse pour le moteur4



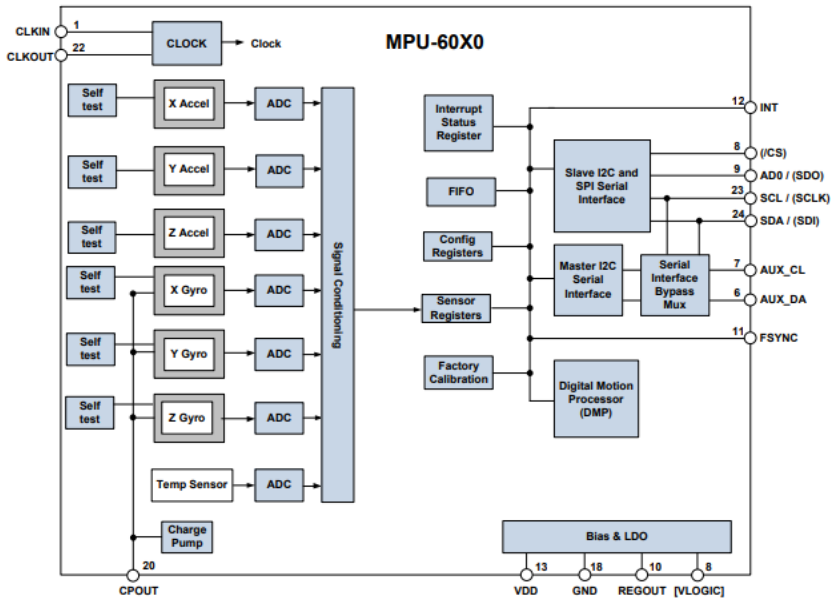
Résultat PID : Le PID se faisant une fois que le robot et tous les capteurs fonctionnent je n'ai pas eu le temps de l'implémenter avec les bonnes valeurs de coefficient, j'ai préféré passer du temps pour que les capteurs fonctionnent correctement.

# 6. Fonctionnement des capteurs retenus.

## 6.1 Central inertielle :

La centrale inertielle MPU-6050 nous permet d'avoir des informations concernant l'accélération du robot. Elle est composée d'un accéléromètre, d'un gyroscope et d'un capteur de température. La communication entre le microcontrôleur et la centrale inertielle se fait en I2C.

**7.5 Block Diagram**




### 6.1.1 Mode communication i2C :

Définition : Un bus **I²C** (pour *Inter Integrated Circuit*) est un bus série et synchrone composé de trois fils conduisant :

- le signal de données (SDA) ;
- le signal d'horloge (SCL) ;
- la référence (masse).

Le bus I²C fut développé par Philips pour les applications de domotique et d'électronique domestique au début des années 1980, notamment pour permettre de relier facilement à un microprocesseur les différents circuits d'une télévision moderne.

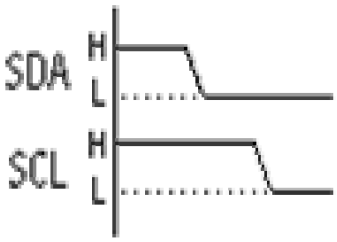
*Définition tiré du site : [http://dictionnaire.sensagent.leparisien.fr/I2C/fr-fr/](http://dictionnaire.sensagent.leparisien.fr/I2C/fr-fr/)*

Les différentes étapes du protocole I2C :

L'I2C fonctionne sur le principe maître esclave, c'est-à-dire que le maître (celui qui génère l'horloge) interroge l'esclave, et l'esclave lui répond. A chaque fois il y a un bit de confirmation de l'esclave ou du maître pour dire « J'ai bien reçu les données que tu m'as envoyées. »

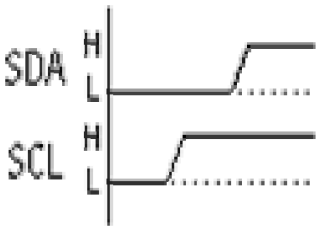
START

Pour commencer la communication en I2C il faut faire ce que l'on appelle le bit de start. C'est-à-dire que l'on impose un niveau logique 0 sur la broche SDA, on attend un peu et on impose un niveau logique 0 sur la broche SCL.




STOP

Pour terminer la communication en I2C il faut faire ce que l'on appelle le bit de stop. C'est-à-dire que l'on impose un niveau logique 1 sur la broche SCL, on attend un peu et on impose un niveau logique 1 sur la broche SDA.




TRANSMISSION

La transmission des données se fait sur 8 bits, le maître transmet dans un premier temps le bit de poids fort, donc le bit 7 (parce que les bits sont numéroté de 0 à 7) sur la broche SDA. Il valide la donnée en appliquant un niveau logique 1 sur la broche SCL. Lorsque l'horloge retombe à 0 le maitre envoie le bit 6 sur SDA et ainsi de suite jusqu'à ce que les 8 bits soient transmis. Ensuite le maitre envoie une demande à l'esclave pour savoir si la transmission c'est bien passée, c'est le bit ACKnowledge(ACK) qui est mis à un niveau logique 1 sur la broche SDA. L'esclave doit ensuite imposer un niveau logique 0 pour signaler que la transmission s'est déroulée correctement. Si le maître voit le niveau 0 imposé par l'esclave, il peut passer à la suite.

Si c'est une adresse que l'on transmet, par exemple l'adresse d'un registre I2C la valeur du bit de poids nous permettra de choisir si l'on veut écrire ou lire les données de l'esclave.

0 : mode écriture
1 : mode lecture

Ecriture

1. Envoie de l'adresse du registre souhaité
2. Sélectionne le mode écriture (bit 0 à 0)
3. Transmission de la donnée

Lecture

1. Envoie de l'adresse du registre souhaité en mode lecture (bit 0 à 1) et on attend l'ACK
2. L'ACK est envoyé par l'esclave ainsi que les données demandé par le maître.
3. Le maître positionne l'ACK à 1 pour stopper la transmission ou à 0 pour la continuer.

Test de l'I2C

Dans un premier temps je me suis inspiré d'un code trouvé sur le site internet : https://www.nxp.com/docs/en/application-note/AN4481.pdf. Pour programmer le MC9S08QE8RM de la carte d'évaluation de l'école.

Code C

Fonction START et STOP de l'I2C (code c)

### 4.1 IIC_Start

```
/*************************************************************\
* Initiate IIC Start Condition
\*************************************************************/
void IIC_Start(void)
{
  IICC_MST = 1;
  timeout = 0;
  while ((!IICS_BUSY) && (timeout<1000))
    timeout++;
  if (timeout >= 1000)
    error |= 0x01;
} //*** Wait until BUSY=1
```

### 4.2 IIC_Stop

```
/*************************************************************\
* Initiate IIC Stop Condition
\*************************************************************/
void IIC_Stop(void)
{
  IICC_MST = 0;
  timeout = 0;
  while ( (IICS_BUSY) && (timeout<1000))
    timeout++;
  if (timeout >= 1000)
    error |= 0x02;
} //*** Wait until BUSY=0
```

Fonction d'écriture I2C (code c)

### 4.4 IIC_CycleWrite

```
/*************************************************************\
* IIC Cycle Write
\*************************************************************/
void IIC_CycleWrite(byte bout)
{
  timeout = 0;
  while ((!IICS_TCF) && (timeout<1000))
    timeout++;
  if (timeout >= 1000)
    error |= 0x08;
  IICD = bout;
  timeout = 0;
  while ((!IICS_IICIF) && (timeout<1000))
    timeout++;
  if (timeout >= 1000)
    error |= 0x10;
  IICS_IICIF = 1;
  if (IICS_RXAK)
    error |= 0x20;
}
```

Fonction de lecture (code c)

### 4.5 IIC_CycleRead

```
/*************************************************************\
* IIC Cycle Read
\*************************************************************/
byte IIC_CycleRead(byte byteLeft)
{
  byte bread;
  timeout = 0;
  while ((!IICS_TCF) && (timeout<1000))
    timeout++;
  if (timeout >= 1000)
    error|=0x08;
  IICC_TX = 0;
  IICC_TXAK = byteLeft <= 1 ? 1 : 0;  //Set NACK when reading the last byte

 bread = IICD;
 timeout = 0;
 while ((!IICS_IICIF) && (timeout<1000))

 while ((!IICS_TCF) && (timeout<1000))
   timeout++;
 if (timeout >= 1000)
   error|=0x08;
 IICC_TX = 0;
 IICC_TXAK = byteLeft <= 1 ? 1 : 0;  //Set NACK when reading the last byte
 bread = IICD;
 timeout = 0;
 while ((!IICS_IICIF) && (timeout<1000))
    timeout++;
 if (timeout) >= 1000)
    error |= 0x10;
 IICS_IICIF=1;
 return bread; }
```

Conclusion : N'ayant pas réussi à lire des données en I2C j'ai décidé de d'utiliser la fonction hardware de CodeWarrior pour faire fonctionner le bus I2C.

Configuration de l' hardware :

| | | | |
|---|---|---|---|
| I2C channel | IIC | | IIC |
| Mode selection | MASTER | | |
| ▷ **Interrupt service/event** | Disabled | | |
| ◢ **MASTER mode** | Enabled | | |
| Polling trials | 2000 | D | Warning: The value should be ad |
| ◢ **Initialization** | | | |
| Address mode | 7-bit addressing | | |
| Target slave address init | 104 | D | |
| ▷ **SLAVE mode** | Disabled | | |
| ◢ **Data and Clock** | | | |
| SDA pin | PTA2_KBIP2_SDA_ADP2 | | PTA2_KBIP2_SDA_ADP2 |
| SCL pin | PTA3_KBIP3_SCL_ADP3 | | PTA3_KBIP3_SCL_ADP3 |
| Internal frequency (multiplier fact | 4.194304 MHz | | 4.194 MHz |
| Bits 0-2 of Frequency divider regi | 011 | | |
| Bits 3-5 of Frequency divider regi | 000 | | |
| SCL frequency | 161.319 kHz | | high: 161.319 kHz |
| SDA Hold | 1.907 us | | high: 1.907 us |
| ◢ **Initialization** | | | |
| Enabled in init code | yes | | |

Code C

L'initialisation étant gérée de manière matérielle je n'ai eu à implémenter que la fonction écriture et lecture de l'I2C en me servant des fonctions existantes générée par CodeWarrior.

Fonction générée par CodeWarrior

FONTION QUI SELCTIONNE L'ESCLAVE

```
byte CI2C1_SelectSlave(byte Slv)
{
  if (IICC1_MST == 1U) {               /* Is the device in the active state? */
    return ERR_BUSY;                   /* If yes then error */
  }
  CI2C1_SlaveAddr = (byte)(Slv << 1);  /* Set slave address */
  return ERR_OK;                       /* OK */
}
```

FONCTION QUI ENVOIE UN CARACTERE

```
byte CI2C1_SendChar(byte Chr)
{
  if((IICS_BUSY) || (InpLenM) || (CI2C1_SerFlag & (CHAR_IN_TX|WAIT_RX_CHAR|IN_PROGRES))) { /* Is the bus busy *
    return ERR_BUSOFF;                 /* If yes then error */
  }
  ChrTemp = Chr;                       /* Save character */
  return (CI2C1_SendBlock(&ChrTemp, 1U, &CI2C1_SndRcvTemp)); /* Send character and return */
}
```

FONCTION QUI RECOIT LES DONNEES

```
byte CI2C1_RecvBlock(void* Ptr,word Siz,word *Rcv)
{
  if (!Siz) {                          /* Test variable Size on zero */
    *Rcv = 0U;
    return ERR_OK;                     /* If zero then OK */
  }
  if((IICS_BUSY) || (InpLenM) || (CI2C1_SerFlag & (CHAR_IN_TX|WAIT_RX_CHAR|IN_PROGRES))) { /* Is the bus busy */
    return ERR_BUSOFF;                 /* If yes then error */
  }
  PtrSndRcv = Rcv;                     /* Safe Rcv pointer */
  InpLenM = Siz;                       /* Set lenght of data */
  InpPtrM = (byte *)Ptr;               /* Save pointer to data for reception */
  IICC1_TX = 1U;                       /* Set TX mode */
  if(IICC1_MST) {                      /* Is device in master mode? */
    IICC1_RSTA = 1U;                   /* If yes then repeat start cycle generated */
  }
  else {
    IICC1_MST = 1U;                    /* If no then start signal generated */
  }
  IICD = (byte)(CI2C1_SlaveAddr+1U);   /* Send slave address */
  return (MainComm());                 /* Call main communication method and return */
}
```

FONCTION QUI ENVOIE LES DONNEES

```
byte CI2C1_SendBlock(const void * Ptr,word Siz,word *Snt)
{
  if (!Siz) {                          /* Test variable Size on zero */
    *Snt = 0U;
    return ERR_OK;                     /* If zero then OK */
  }
  if((IICS_BUSY) || (InpLenM) || (CI2C1_SerFlag & (CHAR_IN_TX|WAIT_RX_CHAR|IN_PROGRES))) { /* Is the bus busy */
    return ERR_BUSOFF;                 /* If yes then error */
  }
  PtrSndRcv = Snt;                     /* Safe Snd pointer */
  OutLenM = Siz;                       /* Set lenght of data */
  OutPtrM = (byte *)Ptr;               /* Save pointer to data for transmitting */
  IICC1_TX = 1U;                       /* Set TX mode */
  if(IICC1_MST) {                      /* Is device in master mode? */
    IICC1_RSTA = 1U;                   /* If yes then repeat start cycle generated */
  }
  else {
    IICC1_MST = 1U;                    /* If no then start signal generated */
  }
  IICD = CI2C1_SlaveAddr;              /* Send slave address */
  return (MainComm());                 /* Call main communication method and return */
}
```

Fonction d'écriture

```
/*Fonction d'écriture en I2C*/
void write(unsigned char registre, unsigned char data){
    unsigned char error;
    unsigned short var;
    var=(registre<<8)|data;
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_SendBlock(&var, 2, &CI2C1_SndRcvTemp);
}
```

Fonction lecture

```
/*Fonction de lecture en I2C*/
unsigned char read(unsigned char registre){
    unsigned char resultat,error;
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_SendBlock(&registre, 1, &CI2C1_SndRcvTemp);
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_RecvBlock(&resultat, 1, &CI2C1_SndRcvTemp);
    return resultat;
}
```

### 6.1.2 Test de la centrale inertielle

Code pour la Centrale inertielle
(voir en annexe)

Le premier registre que l'on va utiliser pour la centrale MPU-6050 est le registre de configuration. Son adresse est 0x1A.

```
/*Initialisation de l'accelerometre*/
void initACC(void){
    unsigned short PWR_MGMT,GYRO_CONFIG;/*INT_PIN_CRG,ACCEL_CONFIG,CONFIG;*/
    unsigned char error;
    PWR_MGMT=0x6B;
    //INT_PIN_CRG=0x37;
    //ACCEL_CONFIG=0x1C;
    //CONFIG=0x1A;
    GYRO_CONFIG=0x1B;
    write(PWR_MGMT,0x01);
    write(GYRO_CONFIG,0x08);
}
```

Les six autres registres utilisés sont les registres qui contiennent les informations x et y et z de l'accéléromètre.

| Registre | Adresse | Fonction |
|---|---|---|
| ACCEL_XOUT_H | 0x3B | Ce registre contienne les bits de poids fort de l'axe des x de l'accéléromètre. |
| ACCEL_XOUT_L | 0x3C | Ce registre contienne les bits de poids faible de l'axe des x de l'accéléromètre. |
| ACCEL_YOUT_H | 0x3D | Ce registre contienne les bits de poids fort de l'axe des y de l'accéléromètre. |
| ACCEL_YOUT_L | 0x3E | Ce registre contienne les bits de poids faible de l'axe des y de l'accéléromètre. |
| ACCEL_ZOUT_H | 0x3F | Ce registre contienne les bits de poids fort de l'axe des z de l'accéléromètre. |
| ACCEL_ZOUT_L | 0x40 | Ce registre contienne les bits de poids faible de l'axe des z de l'accéléromètre. |

J'ai commencé par tester le capteur de température qu'il y a sur la centrale, on ne va pas sans servir sur le robot mais ça me permettait de tester l'I2C en sachant quelle valeur je devais recevoir, sachant que la température de la pièce était approximativement de 25°C. Le registre du capteur de température s'appelle TEMP_OUT et il est divisé en deux registres de 8 bits TEMP_OUT_H à l'adresse 0x41 qui correspondent aux bits de poids fort de la température et TEMP_OUT_L à l'adresse 0x42 qui correspond aux bits de poids faible de la température.

Code pour récupérer la valeur des deux registres.

```
/*Obtention de la température*/
signed short getTemp(void){
        unsigned char registre1, registre2,val1,val2;
        signed short Temp;
        registre1=0x41;
        registre2=0x42;
        val1=read(registre1);
        val2=read(registre2);
        Temp=(((signed short)(val1<<8))|((signed short) (val2)));
        Temp=(Temp/340)+36;
        return Temp;
}
```

Une fois la valeur des deux registres assemblé en une seule variable de 16 bits, j'utilise la formule donnée dans la datasheet de la centrale pour transformer la valeur de la variable en degrés Celsius.

Formule : Température en degré C= (variable de 16 bits)/340+36.53

Une fois le test de l'I2C avec le capteur de température de la centrale finit, je suis passé à la récupération des valeurs de l'accéléromètre sur les axes x, y.

Résultat :

| | | |
|---|---|---|
| (x)= Adresse | 0x68 | 0x00c8 |
| (x)= Temperature | 25 | 0x00c9 |
| (x)= axeX | 17160 | 0x00cb |
| (x)= axeY | -13572 | 0x00cd |

Les valeurs des axes x, y et de la température sont bien récupéré. On peut remarquer que comme indiqué dans la datasheet de la centrale les valeurs de la position sur les axes sont signées.

Afin de me rendre compte des valeurs que peuvent prendre les accélérations des différents axes, j'ai de faire une petite animation en utilisant le port série du microcontrôleur et putty sur l'ordinateur. L'ordinateur et le microcontrôleur fonctionne en UART et sur le logiciel putty j'affiche une grille avec l'axe des x et l'axe des y et un petit rond qui correspond au robot, je simule ainsi les déplacements du robot et j'ai une idée un peu plus précise des valeurs qui seront retourné par la centrale quand elle sera sur le robot.

### 6.1.3 L'UART

Définition : Un **UART**, pour ***Universal Asynchronous Receiver Transmitter***, est un émetteur-récepteur asynchrone universel.
En langage courant, c'est le composant utilisé pour faire la liaison entre l'ordinateur et le port série. L'ordinateur envoie les données en parallèle (autant de fils que de bits de données). Il faut donc transformer ces données pour les faire passer à travers une liaison série qui utilise un même fil pour faire passer tous les bits de données.

Définition trouvé sur le site internet : *http://dictionnaire.sensagent.leparisien.fr/UART/fr-fr/*

Pour configurer l'UART j'ai utilisé les cours de Vincent Poulailleau. Pendant la réalisation du jeu Pong nous avions eu recoure à l'UART pour afficher les raquettes et la balle sur l'écran en VT100.

VT100 : Le **VT100** était un terminal informatique à écran cathodique produit par Digital Equipment Corporation (DEC), qui est devenu un standard de fait dans le domaine des terminaux.

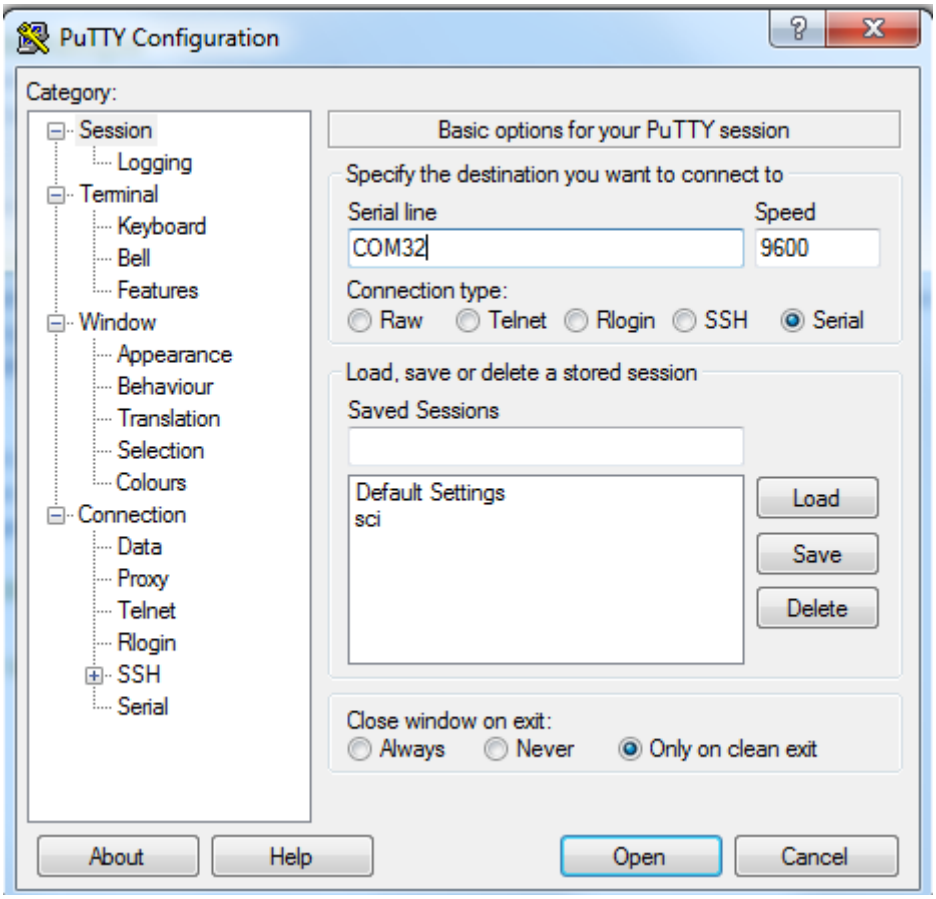
Configuration hardware de l'UART
J'ai configuré dans la partie hardware de CodeWarrior une liaison UART à 9600 bauds.

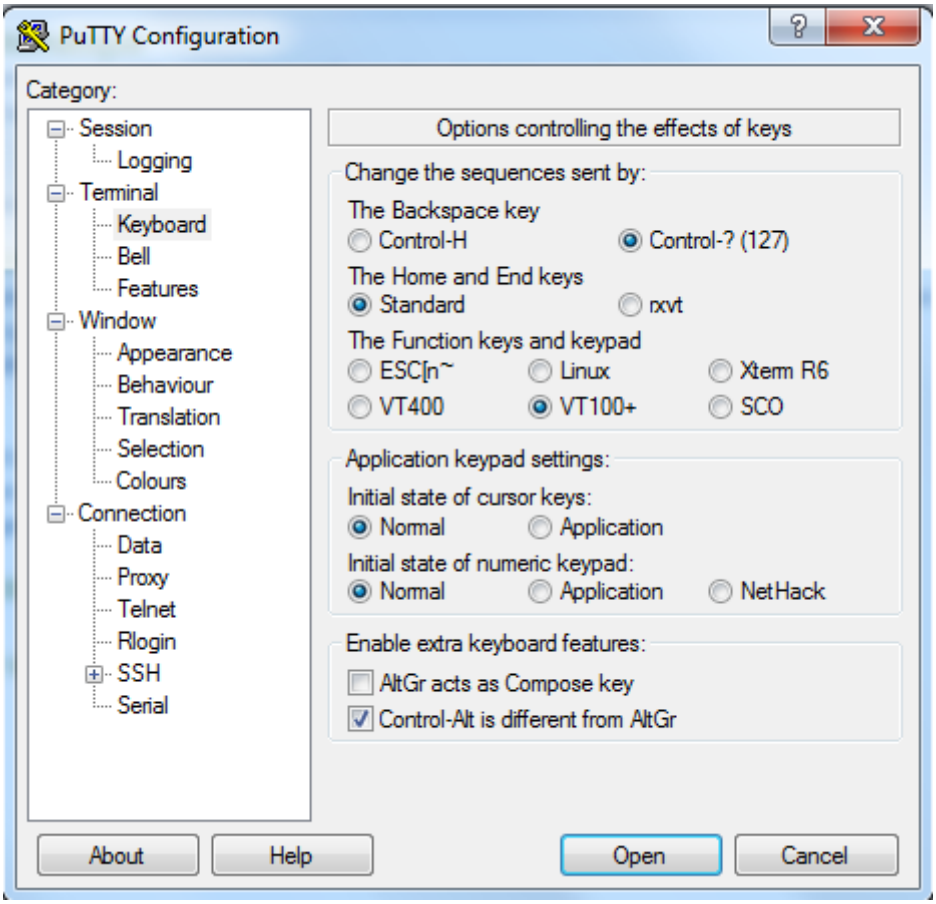
- Clock settings
  - SCI clock gate : Enabled
  - Baud rate divisor : 27
  - Baud rate : environ 9 600 bauds (à 5% près), c'est important car cela rentre dans la configuration du terminal que l'on fera par la suite.
- Loop mode disabled
- Data format : 8 bits (c'est aussi une configuration du terminal)
- Parity : none (c'est aussi une configuration du terminal)
- Pins
  - RxD : disabled
  - Txd : enabled (PTB1)
- Initialization
  - Transimitter : enabled

Code d'affichage de la grille
(Voir annexe)

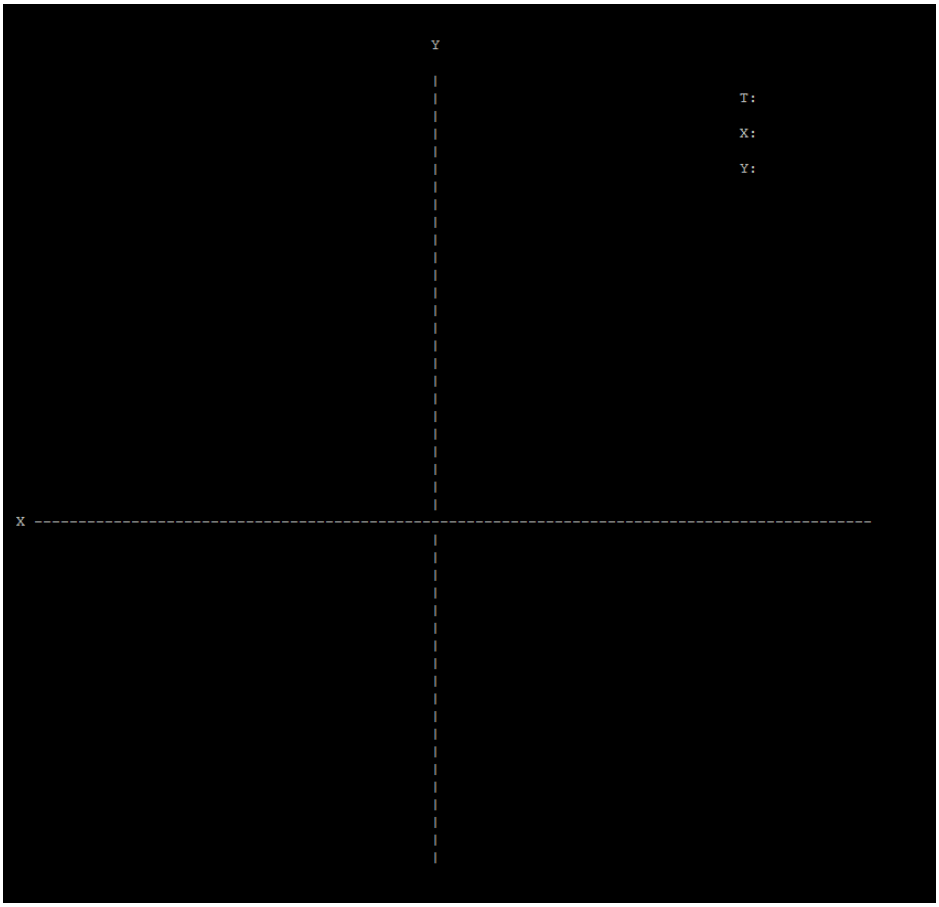
Code déplacement du robot (représenté par un rond)
(Voir dans l'annexe la fonction affichageBalleAcc)

Configuration de Putty



Résultat




Au moment de la capture d'écran je n'ai pas réussi à refaire fonctionner l'affichage des informations, c'est-à-dire qu'en haut à droite du terminal devrait être affiché la température et la valeur en x et y.

Une fois tous ces tests effectué sur la carte d'évaluation de l'école, j'ai fait des tests sur la carte microcontrôleur, conçue par Samuel Huet avec le MCS08QE128RM.

Configuration de l'I2C

| I2C channel | IIC1 | IIC1 |
|---|---|---|
| Mode selection | MASTER | |
| **Interrupt service/event** | Enabled | |
| **Buffers for SLAVE mode** | Disabled | |
| **MASTER mode** | Enabled | |
| Polling trials | 2000 D | |
| **Initialization** | | |
| Address mode | 7-bit addressing | |
| Target slave address init | 8 D | |
| **SLAVE mode** | Disabled | |
| **Data and Clock** | | |
| SDA pin | PTA2_KBI1P2_SDA1_ADP2 | PTA2_KBI1P2_SDA1_ADP2 |
| SCL pin | PTA3_KBI1P3_SCL1_ADP3 | PTA3_KBI1P3_SCL1_ADP3 |
| Internal frequency (multiplier fact | 6.291456 MHz | 6.291 MHz |
| Bits 0-2 of Frequency divider regis | 000 | |
| Bits 3-5 of Frequency divider regis | 000 | |
| SCL frequency | 314.573 kHz | high: 314.573 kHz |
| SDA Hold | 1.113 us | high: 1.113 us |
| **Initialization** | | |
| Enabled in init code | yes | |

Envoie

```
/*Fonction d'écriture en I2C*/
void write(unsigned char registre, unsigned char data){
    unsigned char error;
    unsigned short var;
    var=(registre<<8)|data;
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_SendBlock(&var, 2, &CI2C1_SndRcvTemp);
}
```

Lecture

```
/*Fonction de lecture en I2C*/
unsigned char read(unsigned char registre){
    unsigned char resultat,error;
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_SendBlock(&registre, 1, &CI2C1_SndRcvTemp);
    error = CI2C1_SelectSlave(0x68);
    error = CI2C1_RecvBlock(&resultat, 1, &CI2C1_SndRcvTemp);
    return resultat;
}
```

Résultat

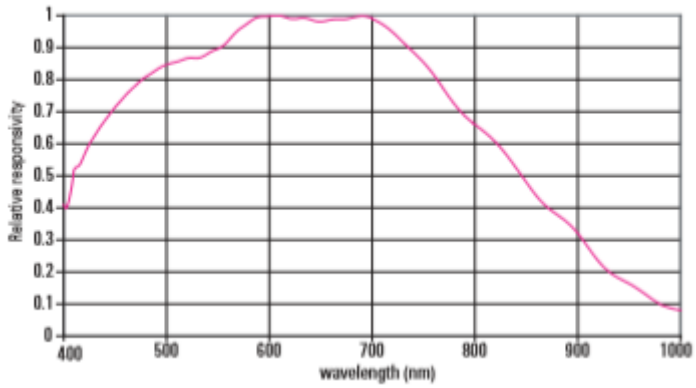
Le résultat était le même qu'avec la carte d'évaluation de l'école je n'ai par contre pas pu afficher l'information sur un terminal car la carte mère n'avait pas de sortie série.

## 6.2 Capteur Optique ADNS-3080:

ADNS-3080 est un capteur optique utilisé dans les souris d'ordinateur pour détecter les impuretés présentes sur la surface ou est posées la souris et ainsi mesurer un déplacement. Dans notre cas ce capteur va nous servir à mesurer la vitesse du robot. Il fonction en SPI (MOSI, MISO, CS, SCLK). Ce capteur contient un port série (SPI), un oscillateur, un processeur pour traiter l'image qu'il reçoit et une sortie pour gérer une led.

Caractéristique:

L'ADNS-3080 peut être alimenté en 3.3V ou 5V et la fréquence de la communication SPI ne doit pas dépasser 2MHz. La résolution de ce capteur est exprimé en images par inches ont à le choix entre 400 et 1600. Le choix d'une led rouge pour éclairer la surface en dessous du capteur a été retenue car c'est la longueur d'onde la plus visible par ce dernier.

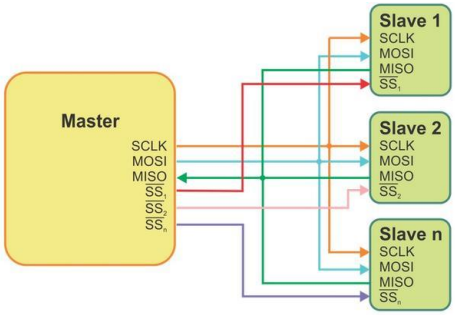



Fonctionnement:

Pour communiquer avec le capteur il faut interroger ses registres en lecture ou écriture. Chaque registre correspond à une fonction dans le capteur (par exemple l'axe des x, l'axe des y ou l'adresse du capteur). Une fois les informations du capteur reçu par le microcontrôleur c'est à nous d'exploiter les valeurs brutes reçues pour en déduire une vitesse.

### 6.2.1 La SPI:

Sérial Peripherical interface est mode communication série entre deux appareils électroniques fonctionnant sur le principe maître esclave. Dans notre cas le maître sera le microcontrôleur et l'esclave le capteur optique ADNS-3080.Ce mode de communication nécessite 4 fils, le Chip Select (CS) qui permet de sélectionner l'appareil avec lequel le microcontrôleur doit communiquer, l'horloge (SCLK) qui cadence la communication, MasterOutSlaveIn (MOSI) c'est le fil qui transport les données sortant du microcontrôleur et entrant dans le capteur optique et MasterInSlaveOut (MISO) c'est le fil qui transport les données sortant de l'esclave et entrant dans le maître. Deux autres fils sont aussi nécessaires pour alimenter le capteur, VCC (3.3V) et la masse (GND).




Pour pouvoir communiquer avec le capteur en SPI, il a fallu faire une étude de registre. C'est à dire étudier tous les registres qui sont nécessaire pour activer la SPI, envoyer et recevoir des données.

Etude de registre:

Chaque microcontrôleur à des registre qui lui sont propre, même si des microcontrôleurs d'un même fabriquant auront des registres très similaires.

L'étude de registre qui va suivre concerne le microcontrôleur MC9S08QE128RM utilisé dans la carte mère du projet.

Registre SPI du MC9S08QE128RM:

-SPxC1
-SPxC2
-SPIxBR
-SPIxS
-SPxD

SPxC1: Ce registre d'écriture et de lecture autorise/ou pas le SPI, les interruptions et d'autre option de configuration.

SPIE: Ce bit à 0 active les interruptions lorsque le buffer de réception est complet.

SPE: Ce bit active la SPI lorsqu'il est à 1 et la désactive lorsqu'il est à 0.

SPTIE: Ce bit à 0 active les interruptions lorsque le buffer de transmission est vide.

MSTR: Ce bit sélectionne le mode du microcontrôleur
1: le microcontrôleur fonctionne en tant qu'esclave
0: le microcontrôleur fonctionne en tant que maître

CPOL: Ce bit gère la polarité de la communication SPI
0: La communication commence avec un front montant d'horloge
1: La communication commence avec un front descendant d'horloge

CPHA: Ce bit gère la phase de la communication SPI
0:Le premier front d'horloge a lieu au milieu du premier bit du premier cycle de 8bits
1:Le premier front d'horloge à lieu au début du premier bit du premier cycle de 8bits

SSOE: Ce bit associé au bit MODFEN du registre SPIxC2 détermine la fonction du Chip Select.

LSBFE: 0: La SPI commence avec le MSB (Most Significant Bit)
1:La SPI comment avec le LSB (Least Significant Bit)

---

SPxC2: Ce registre d'écriture et de lecture est utilisé pour contrôler les caractéristiques optionnelles de la SPI.

MODFEN: Ce bit est associé au bit SSOE du registre SPxC1
0: Le Chip Select est une broche entrée sortie général
1: Le Chip Select est une sortie automatique ou une entrée

BIDIROE: Ce bit gère la SPI lorsque le MISO et le MOSI sont sur la même broche.

SPISWAI: Ce bit gère l'horloge lorsqu'il ne se passe rien sur le bus SPI
0:l'horloge continue de fonctionner
1:l'horloge s'arrête

SPC0: Ce bit décide si le MISO et le MOSI sont sur la même broche du microcontrôleur, ou si se sont deux broches séparées.
0:La SPI utilise deux broches séparées pour les données entrant et sortant du microcontrôleur.
1:La SPI est configuré sur un seul fil.

---

SPIxBR: Ce registre est utilisé pour choisir la vitesse de l'horloge du bus SPI.

SPPR2[2:0]: Ces trois bits décident du prescaler de la division de l'horloge du bus SPI les choix sont (1,2,3,4,5,6,7,8).

SPR[2:0]: Ces trois bits décident de la valeur de la division du baud rate(vitesse de transmission.

---

SPIxS: Ce registre à trois bits de lecture.

SPRF: Ce bit nous indique l'état du buffer de réception.
0: Aucune donnée ne sont pas disponible dans le buffer de réception.
1: Les données sont disponibles dans le buffer de réception

SPTEF: Ce bit nous indique l'état du buffer de transmission
0: L e buffer de transmission SPI est plein

1: Le buffer de transmission SPI est vide

MODF: Ce bit nous indique si une erreur est détectée:
0: pas d'erreur
1: Une erreur est détectée

---

SPIxD: Lorsque ce registre est lu la valeur du buffer de réception est récupérée et lorsque l'on écrit dans ce registre les données sont écrites dans le buffer de transmission.

---

code C

Initialisation de la SPI

```
#define CS PTBD_PTBD5

/*fonction d'initialisation de la SPI
  On autorise l'interruption
  Le Microcontrolleur est le maitre
  On envoie les données sur un front d'horloge decendant
  Le fronts d'horolge on lieu au début de la donnée
  On transmet le bit de poid faible en premier
  SS est une entrée sortie générale*/
void initSPI(void){

    //SPIC2 On fait la SPI sur 2fils et l'horloge continue pendant le wait mode
    //SPIC1 On autorise la communication SPI et on met le µC en maitre, la communication commence sur un front descendant de l'horloge
    SPIC1_SPE=1;
    SPIC1_MSTR=1;
    SPIC1_CPHA=1;
    SPIC1_CPOL=1;

    SPIBR_SPPR1=1;//division de l'horloge par 2

    PTBDD_PTBDD5=1;
    PTBPE_PTBPE5=1;
    PTBPE_PTBPE4=1;//pull up;
}
```

Envoie des données

```
/*Fonction d'écriture en SPI*/
void writeSPI(unsigned char registre, unsigned char data){

    //CS=0;
    /*Si le buffer de transmission est vide*/
    if(SPIS_SPTEF){
        SPID=registre;//envoie des données sur le bus SPI
    }
    delay(1);
    if(SPIS_SPTEF){
        SPID=data;//envoie des données sur le bus SPI
    }

    //CS=1;

}
```

Réception des données

```
unsigned char Transfert(unsigned char registre){
    unsigned char data;

    CS=0;

    if (SPIS_SPTEF) {
        SPID = registre; //envoie des données sur le bus SPI
        delay(200);
    }

    if (SPIS_SPRF) {
        data = SPID; //on récupere la valeur du registre de SPID (en mode lecture)
    } //on attend que le registre de lecture soit plein

    if (SPIS_SPTEF) {
        SPID = 0x00; //envoie des données sur le bus SPI
        delay(200);
    }


    if (SPIS_SPRF) {
        data = SPID; //on récupere la valeur du registre de SPID (en mode lecture)
    } //on attend que le registre de lecture soit plein

    CS=1;
    return data;

}
```

### 6.2.2 Test du capteur optique

```
signed char CapteurOptique(void){
    volatile unsigned char result;
    volatile signed char valeur;

    result=Transfert(0x02);
    if(result & 0x80){
        valeur=(signed char)Transfert(0x03);
    }

    return valeur;
}
```

Résultat :

| | | |
|---|---|---|
| (x)= resultat | 0b10000001 | 0x00c3 |
| (x)= valeur | 58 | 0x00c4 |

L'image détectée par le capteur optique change bien. Je n'ai malheureusement pas eu le temps d'écrire un programme qui calculait à partir de cette valeur une vitesse.

## 6.3 Encodeur Magnétique AS5048A:

Présentation: Ce capteur est associé à un aimant polarisé radialement qui est positionné sur l'axe du moteur et qui doit être le plus proche possible de l'encodeur (entre 0.5mm et 2.5mm) selon les recommandations donnée par le fabricant.
Dans notre projet l'encodeur a pour fonction de nous permettre de mesurer la vitesse de chaque moteur et ainsi pouvoir faire un PID c'est-à-dire une régulation de la vitesse.

Principe de fonctionnement:
Ce capteur de position à 360° à une résolution de 14 bits et une précision maximum de 0,05°. L'AS5048A mesure la position absolue de l'angle de rotation du champ magnétique, la valeur étant analogique, l'AS5048A la convertie en numérique. Le capteur communique avec le microcontrôleur en SPI il faudra donc que l'on transforme l'angle reçue en une vitesse de rotation.

Caractéristique:
Le capteur s'alimente en 3,3V et communique avec le microcontrôleur en SPI. La particularité de l'encodeur par rapport aux autres capteurs du projet, c'est que ses registres sont sur 16 bits. Le microcontrôleur utilisé pendant les tests (MC9S08QE8RM) et celui utilisé sur la carte mère de notre projet (MC9S08QE128RM) ont des registres qui sont sur 8 bits, il va donc falloir trouver une solution pour envoyer et recevoir des données qui sont sur 16 bits avec un microcontrôleur 8bits.

Registre de l'encodeur :
L'encodeur AS5048A comporte 7 registres permettant au microcontrôleur de lui indiquer quelles informations ils souhaitent recevoir. Chaque registre comporte une adresse qui permet au microcontrôleur de l'identifier en envoyant l'adresse du registre et en lui indiquant s'il souhait lire les données de ce registre ou écrire dans ce registre.

| Registre | Adresse | Fonction |
|---|---|---|
| SPI NOP | 0x0000 | Ce registre permet de savoir si l'encodeur fonction correctement, si c'est le cas il répond 0x0000 lorsque l'on l'interroge. |
| Clear Error Flag | 0x0001 | Ce registre nous permet de savoir s'il y a un problème de communication en envoyant un 1. |
| Programming Control | 0x0003 | Ce registre permet de programmer l'AS5048A. Par exemple la position de l'angle 0 |
| OTP Register Zero Position High | 0x0016 | Ce registre contient les bits de poids fort de l'angle 0. |

| | | |
|---|---|---|
| OTP Register Zero Position low | 0x0017 | Ce registre contient les bits de poids faible de l'angle 0 |
| Diagnostics+Automatic Gain Control | 0x3FFD | Ce registre configure le gain du champ magnétique. |
| Magnitude | 0x3FFE | Ce registre contient la valeur du champ magnétique crée par l'aimant. |
| Angle | 0x3FFF | Ce registre nous permet de connaitre l'angle de l'aimant par rapport à l'encodeur. |

### 6.3.1 Test de l'encodeur magnétique :

Pour tester l'encodeur j'ai utilisé les supports moteurs de Jean Gabriel qui s'occupait du châssis du robot. On a fixé l'aimant sur l'axe d'un des moteurs brushless, puis on a inséré l'encodeur dans son support et relié l'encodeur au microcontrôleur de la carte d'évaluation de l'école (MC9S08QE8RM).

Code C :

Initialisation de la SPI

```
#define CS PTBD_PTBD5

/*fonction d'initialisation de la SPI
  On autorise l'interruption
  Le Microcontrolleur est le maitre
  On envoie les données sur un front d'horloge decendant
  Le fronts d'horolge on lieu au début de la donnée
  On transmet le bit de poid faible en premier
  SS est une entrée sortie générale*/
void initSPI(void){

    //SPIC2 On fait la SPI sur 2fils et l'horloge continue pendant le wait mode
    //SPIC1 On autorise la communication SPI et on met le µC en maitre, la communication commence sur un front descendant de l'horloge
    SPIC1_SPE=1;
    SPIC1_MSTR=1;
    SPIC1_CPHA=1;
    SPIC1_CPOL=1;

    SPIBR_SPPR1=1;//division de l'horloge par 2

    PTBDD_PTBDD5=1;
    PTBPE_PTBPE5=1;
    PTBPE_PTBPE4=1;//pull up;
}
```

Fonction d'écriture

```
/*Fonction d'écriture en SPI*/
void writeSPI(unsigned char registre, unsigned char data){

    //CS=0;
    /*Si le buffer de transmission est vide*/
    if(SPIS_SPTEF){
        SPID=registre;//envoie des données sur le bus SPI
    }
    delay(1);
    if(SPIS_SPTEF){
        SPID=data;//envoie des données sur le bus SPI
    }

    //CS=1;

}
```

(voir dans l'annexe la fonction readTransfert)

L'encodeur AS5048A renvoie au microcontrôleur une valeur d'angle sur 14 bits. Il faut donc caster et décaler la valeur reçue afin d'avoir deux variable de 8 bits, une qui contient les bits de poids fort et une qui contient les bits de poids faible. Une fois les deux opérations effectuées on rassemble les deux variables en une variable de 16 bits en utilisant un ou logique (|).

Une fois l'angle récupéré, il a fallu trouver un moyen d'obtenir une vitesse. Pour cela je me suis servi de la formule V=d/t ou V la vitesse, d la distance et t le temps. Pour le temps j'ai utilisé un timer, c'est-à-dire que tous les x ms je récupère une valeur d'angle et la différence entre deux mesures d'angle espacé de x (unités de temps) me donne une distance. La vitesse sera donc une vitesse angulaire.
V= (angle =1-angle2)/ (valeur du timer).

Fonction Timer

### 6.3.2 Calcul de la vitesse

```
for (;;) {

		angle1 = ReadTransfert(0x3FFF);
		vitesse = (((angle2 - angle1) * 25) / 7400); //vitesse en Tr/s virgule fix il faut diviser par
10
		angle2 = angle1;
	}
```

La vitesse est exprimé en tour par seconde pour quelle puisse être interprété par le PID.

## 7. Conclusion : La récupération des données de chaque capteur fonctionne mais je n'ai pas réussi à les exploiter pleinement pour ma partie il me reste à faire l'implémentation du PID et réussir à obtenir une vitesse avec les valeurs que me renvoient les capteurs optiques. Il me reste également à interroger chacun leur tour les capteurs et modifier mon code pour l'encodeur pour qu'il soit compatible pour le microcontrôleur qui se trouve sur la carte-mère.

## 8. Remerciement :

J'aimerais remercier pour l'encadrement de ce projet Monsieur Steve N'Guyen qui a été d'une grande aide technique toute au long du projet et qui a passé du temps pour nous partager ses compétences et sa passion pour la robotique. J'aimerais également remercier Aloïs Miclo et Monsieur Aubry pour leur disponibilité durant ces mois de projet. Enfin j'aimerais remercier Clément Cailleau, Clément Lavergne et Joris Offouga pour l'entraide dont ils ont fait preuve et mon équipe composée de Samuel Huet Thomas Coutant et Jean Gabriel Massicot pour la bonne ambiance et la bonne communication qu'il y a eu pendant le projet.

# 9. ANNEXES

## 9.1 CODE

Code du fichier Centrale.c

```
/*Obtention de l'adresse*/
unsigned char getAdress(void) {
    unsigned char registre,valeur;
    registre = 0x75;
    valeur=read(registre);
    return valeur;
}

/*Obtention du déplacement en X de l'accelerometre*/
unsigned short getX(void){
    unsigned char registre1, registre2, val1, val2;
    unsigned short X;
    registre1 = 0x3B;
    registre2 = 0x3C;
    val1=read(registre1);
    val2=read(registre2);
    X = ((unsigned short) (val1 << 8)|(unsigned short) val2);
    return X;
}

/*Obtention du déplacement en Y de l'accelerometre*/
unsigned short getY(void){
    unsigned char registre1, registre2, val1, val2;
    unsigned short Y;
    registre1 = 0x3D;
    registre2 = 0x3E;
    val1=read(registre1);
    val2=read(registre2);
    Y = ((unsigned short) (val1 << 8)|(unsigned short) val2);
    return Y;
}

/*Obtention du déplacement en Z du gyroscope*/
unsigned short GYRO_Z(void){
    unsigned char registre1, registre2, val1, val2;
    unsigned short Z;
    registre1 = 0x71;
    registre2 = 0x72;
    val1=read(registre1);
    val2=read(registre2);
    Z = ((unsigned short) (val1 << 8)|(unsigned short) val2);
    return Z;
}
```

```
/*Affichage des axes x et y sur putty*/
void affichageGrill(void){
    unsigned short ordonnee,abscisse;
    for(ordonnee=5;ordonnee<50;ordonnee++){
        move(50,ordonnee);
        putc('|');
    }
    for(abscisse=5;abscisse<100;abscisse++){
        move(abscisse,30);
        putc('-');
    }
    move(50,3);
    putc('Y');
    move(3,30);
    putc('X');
    move(85,6);
    putc('T');
    move(86,6);
    putc(':');
    move(85,8);
    putc('X');
    move(86,8);
    putc(':');
    move(85,10);
    putc('Y');
    move(86,10);
    putc(':');
}
```

```
/*Affichage de la balle (sur Putty) correspondant à la position de la central*/
void affichageBalleAcc(void){
    volatile signed short axeX,axeY,dx,dy;
    axeX = getX();
    axeY = getY();
    dx = (axeX / 1000) + 30;
    dy = (axeY / 1000) + 40;
    move(dx,dy);
    putc('O');
    //delay(1000);
    move(dx,dy);
    if(dx==50)
        putc('|');
    if(dy==30)
        putc('-');
    if(dx!=50&&dy!=30)
        putc(' ');
}
```

```
/*affichage numérique des différentes info (temp,x,y)*/
void information(unsigned short val, unsigned short nbre){
    unsigned short ch,i,j;
    j=1;
    if(val>1000)
        val=val/1000;
    for(i=nbre;i>1;i--){
        j=10*j;
    }
    for(i=j;i>=10;(i=i/10)){
        ch = ((val/i)%10) + 48;
        while (SCIS1_TDRE == 0);
        SCID = (ch);
        delay(1000);
    }
    ch = (val%10) + 48;
    while (SCIS1_TDRE == 0);
    SCID = (ch);
}
```

Code du fichier Centrale.h

```
* central.h

#ifndef CENTRAL_H_
#define CENTRAL_H_

#include "Cpu.h"
#include "Events.h"
#include "CI2C1.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

unsigned char getAdress(void);
signed short getTemp(void);
unsigned short getX(void);
unsigned short getY(void);
unsigned short GYRO_Z(void);
void affichageBalleAcc(void);
void affichageGrill(void);
void information(unsigned short val, unsigned short nbre);
void initACC(void);
void write(unsigned char registre, unsigned char data);
unsigned char read(unsigned char registre);


#endif /* CENTRAL_H_ */
```

SPI

```
/*Fonction de lecture des donnée en SPI*/
unsigned short ReadTransfert(unsigned short registre) {
        unsigned short data;
        unsigned char var;


        CS = 0;
        //envoie du registre
        if (SPIS_SPTEF) {
                var=((unsigned char)(registre>>8) & 0xFF);
                SPID = var;
                }
        if (SPIS_SPTEF) {
                var=((unsigned char)(registre) & 0xFF);
```

```
            SPID = var;
            delay(1);
        }
        CS=1;

        /*//lecture
        if (SPIS_SPRF) {
            data = SPID; //on récupere la valeur du registre de SPID (en mode lecture)
        } //on attend que le registre de lecture soit plein
        if (SPIS_SPRF) {
            data = SPID; //on récupere la valeur du registre de SPID (en mode lecture)
        } //on attend que le registre de lecture soit plein

        //envoie de 0
        if (SPIS_SPTEF) {
            SPID = 0x00; //envoie des données sur le bus SPI
        }
        if (SPIS_SPTEF) {
            SPID = 0x00; //envoie des données sur le bus SPI
            delay(1);
        }*/

        CS=0;
        if (SPIS_SPTEF) {
                SPID = 0xFF;
        }
        //lecture
        if (SPIS_SPRF) {
            var = SPID;
            data = (((unsigned short) (var) << 8) & 0x3F00); //on récupere la valeur du registre de
SPID (en mode lecture)
```

```
	}
	if (SPIS_SPTEF) {
			SPID = 0xFF;
	}
	if (SPIS_SPRF) {
		var = SPID;
		data = data | ((unsigned short) (SPID) & 0x00FF);
	} //on attend que le registre de lecture soit plein

	CS = 1;
	return data;

}
```

## 9.2 Bibliographie


Site internet :

http://www.mdp.fr/documentation/lexique/brushless/definition.html

*http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/*

https://www.nxp.com/docs/en/application-note/AN4481.pdf

## 9.3 Datasheet

Datasheet du microcontrôleur de la carte d'évaluation de l'école :
https://www.nxp.com/docs/en/reference-manual/MC9S08QE8RM.pdf

Datasheet du microntolleur utilisé sur la carte mère :
https://www.nxp.com/docs/en/data-sheet/MC9S08QE128.pdf

Datasheet du capteur optique :
https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2009/ncr6_wjw27/ncr6_wjw27/docs/adns_3080.pdf

Datasheet de l'encodeur :
https://media.digikey.com/pdf/Data%20Sheets/Austriamicrosystems%20PDFs/AS5048A,B.pdf

Datasheet de la centrale inertielle :
Caractéristiques
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
Registres
https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf