

Rapport de projet

Robot gyropode



William BRUNET **Jérémie ROBLES** **Sofiane AYAT**

Session 2018 Lycée Le Corbusier

mainbot
building emotions

Remerciements

Tout d'abord, nous tenons à remercier tout particulièrement et à témoigner toute notre reconnaissance aux personnes suivantes, pour leur dévouement et leur soutien dans la concrétisation de ce projet :

- M. Belkacem JARRAY, Professeur d'informatique et de réseau, pour toute l'aide et les conseils apportés durant ce projet
- M. Antoine MULIN, responsable technique chez Mainbot, pour ce projet passionnant et ses conseils en électronique.
- M. Neou, Professeur d'informatique et de réseau, pour toute l'aide et les conseils apportés durant ce projet
- M. DAKAR, Professeur d'informatique et de réseau, pour ses conseils et son investissement dans nos recherches pour notre projet
- M. Christian GOMEZ, ingénieur mécanique chez Mainbot, pour ses conseils avisés en mécanique.



TABLE DES MATIÈRES

I. INTRODUCTION	5
II. RÉPARTITION DES TÂCHES	6
III. PLANNING PRÉVISIONNEL	7
IV. CAHIER DES CHARGES	8
V. SEMAINIER	9
VI. ANALYSE UML	12
A. Diagramme de cas d'utilisation	12
B. Diagramme de déploiement	13
C. Contexte	14
GESTION DE L'ÉQUILIBRE DU GYROPODE (ROBLES Jérémie étudiant 1)	15
I. PRÉSENTATION DU TRAVAIL DEMANDÉ	16
II. SCHÉMA DE MON SYSTÈME	17
III. TRAVAIL DEMANDÉ	18
IV. LES COMPOSANTS	19
A. Les micro-contrôleurs (atmega328p)	19
B. Le centrale inertielles (MPU6050)	21
C. Le capteur de mouvement (APDS-9960)	24
D. Le pont en H (LN298N)	26
E. Les capteurs à effet Hall (SS411P)	28
V. LA PROGRAMMATION	30
A. L'IDE Arduino	30
B. Liaison micro-contrôleur au PC	32
VI. LA RÉALISATION DU GYROPODE	33
VII. CONCLUSION	41
II. GESTION DE L'INTERFACE BLUETOOTH (BRUNET William étudiant 2)	42
I. PRÉSENTATION DU TRAVAIL DEMANDÉ	43
II. TRAVAIL DEMANDÉ	44



III. SCHÉMA DE CÂBLAGE	45
IV. LE MATÉRIEL UTILISÉ	46
A. Les Hardwares	46
B. Les Softwares	52
Android Studio	54
CONCLUSION	57
III. COMMANDE PC (AYAT Sofiane étudiant 3)	57
I. PRÉSENTATION DU PC DE CONTRÔLE	58
II. TRAVAIL DEMANDÉ	59
III. MODULE DE COMMUNICATION WIFI	60
IV. PROGRAMMATION	63
A. Logicielle : l'IDE Arduino	63
B. Électronique	67
V. ENVOI ET RÉCEPTION DES COMMANDES	69
A. Programmation côté Wifi	70
B. Programmation côté processeur	71
BOUTON DE L'IHM	72
A. Comment installer ?	72
B. Logicielle Qt Creator	73
C. Les Boutons	75
L'APPLICATION STREAMING	77
A. Matériel	77
B. Mise en place de la caméra	77
C. Création d'une connexion	78
D. Les bonnes librairies	79
L'IHM PC	80
DÉROULEMENT DU PROJET	81
CONCLUSION	82
ANNEXE :	83
PROGRAMME GESTION EQUILIBRE	86
PROGRAMME TRANSFERT DES INFORMATIONS	141



PROGRAMME APPLICATION MOBILE	166
PROGRAMME IHM	179
Programme module wifi	186



I. INTRODUCTION

La société Mainbot est spécialisée dans le développement de robots éducatifs de type « robot scratch », pour les petits enfants. Ce projet consiste à réaliser, pour un robot de la société Mainbot, un châssis mobile et auto-équilibré de type gyropode. Ce dispositif doit être autonome en termes d'alimentation et de traitement de l'information.

Néanmoins, la plateforme pourra être pilotée en local par Bluetooth, via une application Smartphone/Android ou à distance par Wifi, via une application sur PC/Windows/TCP/IP. La masse du système ne devra pas excéder 800 grammes.

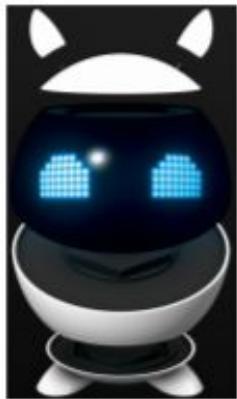


figure 1. robots de la société Mainbot

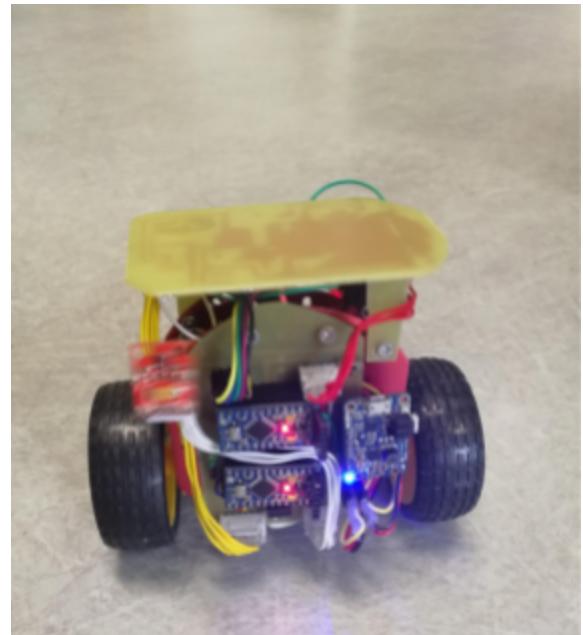


figure 2 : gyropode

II. RÉPARTITION DES TÂCHES

Étudiant 1 Jérémie ROBLES

- Prise en main du projet
- Installation des logiciels nécessaires
- Mise en œuvre de l'accéléromètre
- Mise en œuvre du gyropode
- Mise en œuvre du détecteur de mouvement
- Mise en œuvre du capteur à effet Hall
- Réalisation de la gestion de l'équilibre et de mouvement de l'équilibre

Étudiant 2 William BRUNET

- Prise en main du projet
- Installation des logiciels nécessaires
- Mise en œuvre d'une application mobile
- Mise en œuvre du Bluetooth
- Réalisation des commandes et de la communication Bluetooth

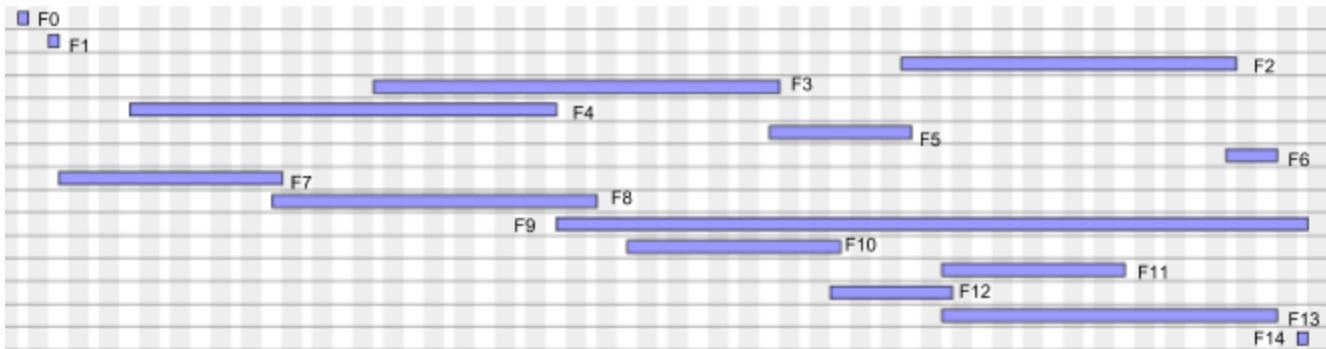
Étudiant 3 Sofiane AYAT

- Prise en main du projet
- Installation des logiciels nécessaires
- Mise en œuvre d'une IHM PC (applications PC)
- Mise en œuvre du module de communication Wifi
- Mise en œuvre de l'application streaming
- Réalisation des commandes et de la communication Wifi



III. PLANNING PRÉVISIONNEL

• F0 : Prendre en main le projet	29/01/18	29/01/18
• F1 : Définir un protocole de dialogue entre les deux processeurs	01/02/18	01/02/18
• F2 : Développer la communication RS232 entre les processeurs	26/04/18	28/05/18
• F3 : Gérer l'équilibre du robot (gyroscope + accéléromètre)	05/03/18	13/04/18
• F4 : Gérer la commande des moteurs (PWM vers pont en H)	09/02/18	22/03/18
• F5 : Gérer la vitesse et la direction des moteurs (Capteur à effet Hall)	13/04/18	26/04/18
• F6 : Gérer le capteur de mouvement (APDS-9960)	28/05/18	01/06/18
• F7 : définir un protocole de dialogue entre le processeur et l'opérateur(protocol unique pour Bluetooth et wi-fi)	02/02/18	23/02/18
• F8 : Développer le module de communication Bluetooth côté processeur	23/02/18	26/03/18
• F9 : Développer l'application de commande sous Android (smartphone)	23/03/18	04/06/18
• F10 : Développer le module de communication wi-fi coté processeur	30/03/18	19/04/18
• F11 .Développer l'application streaming (script de connexion et transfert d'image)	30/04/18	17/05/18
• F12 : Développer l'IHM PC	19/04/18	30/04/18
• F13 : Développer la communication avec le module wifi	30/04/18	01/06/18
• F14 : Documenter l'application	04/06/18	04/06/18



IV. CAHIER DES CHARGES

L'équilibre du robot doit être assuré par des algorithmes internes, à développer en fonction des capteurs disponibles sur le robot.

Le mouvement du robot doit être assuré par des commandes locales ou distantes :

- En mode local, l'opérateur a le robot sous les yeux et il lui envoie les commandes par Bluetooth, grâce à un smartphone Android.
- En mode distant, l'opérateur est devant son PC et dispose d'une image en temps réel du robot (streaming) qui lui permettra d'envoyer des commandes par Wifi, via un point d'accès et du réseau internet.

À un moment donné, un seul mode est utilisé (seul le module adéquat est connecté matériellement pour chaque mode). L'interface côté robot doit donc être uniforme afin de pouvoir répondre sans modification aussi bien dans le mode local que dans le mode distant.



V. SEMAINIER

Semaine 1 : du 29 Janvier au 2 Février	
ROBLES Jérémie	<ul style="list-style-type: none">● Prise en main du projet● Répartition des tâches
BRUNET William	<ul style="list-style-type: none">● Prise en main du projet● Répartition des tâches
AYAT Sofiane	<ul style="list-style-type: none">● Prise en main du projet● Répartition des tâches● Installation des différents logiciels

Semaine 2 : du 5 Février au 9 Février	
ROBLES Jérémie	<ul style="list-style-type: none">● Définir un protocole unique entre l'opérateur et le gyropode● Commencer à développer en logicielle les ponts en H
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none">● Définir les différents protocoles entre les deux processeurs● Faire le premier pas sur Arduino

Semaine 3 : du 12 Février au 16 Février	
ROBLES Jérémie	<ul style="list-style-type: none">● Terminer les contrôles moteur en logicielle dans mes fichiers sources
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none">● Recherche sur la caméra● Découverte du fonctionnement





Semaine 4 : du 5 Mars au 9 Mars	
ROBLES Jérémie	<ul style="list-style-type: none">• Implémenter la centrale inertielles• Recherche du filtre de kalman et application de celui ci
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none">• Trouver l'adresse IP de la caméra• Prendre un photo et afficher une image à l'IHM

Semaine 5 : du 12 Mars au 16 Mars	
ROBLES Jérémie	<ul style="list-style-type: none">• Recherche coef PID et programmation de fonction PID• Recherche de bon Kd, Kp, Ki
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none">• Recherche des différentes données sur l'IHM

Semaine 6 : du 19 Mars au 23 Mars	
ROBLES Jérémie	<ul style="list-style-type: none">• Recherche de bon Kd, Kp, Ki
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none">• Développer l'IHM PC

Semaine 7 : du 26 Mars au 30 Mars	
ROBLES Jérémie	<ul style="list-style-type: none">• Recherche de bon Kd, Kp, Ki



BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none"> ● Faire les différents boutons ● Définir les signaux et les slots

Semaine 8 : du 2 Avril au 6 Avril	
ROBLES Jérémie	<ul style="list-style-type: none"> ● Recherche de bon Kd, Kp, Ki ● Mise à jour du shield du robot
BRUNET William	
AYAT Sofiane	<ul style="list-style-type: none"> ● Créer un réseau afin de se connecter ● Faire la communication avec le module Wifi

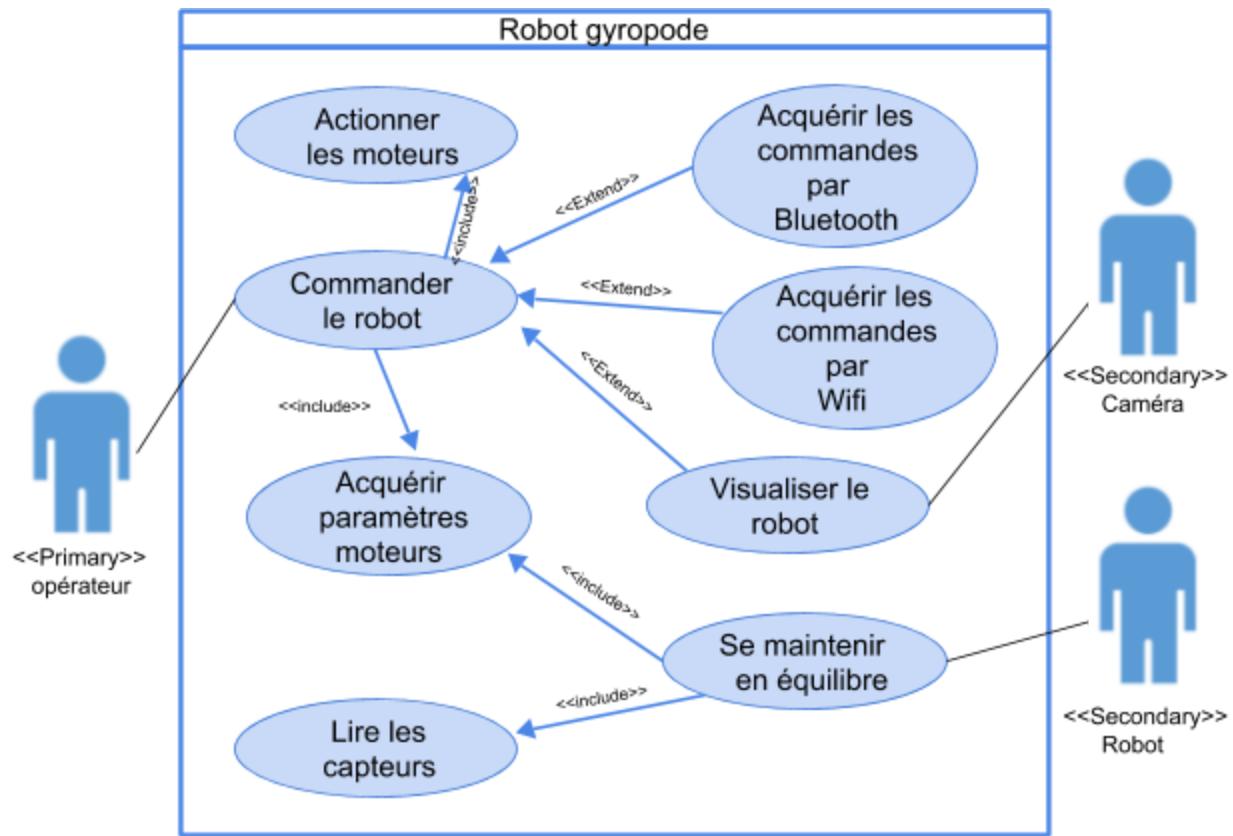
Semaine 9 : du 7 Mai au 11 Mai	
ROBLES Jérémie	<ul style="list-style-type: none"> ● Examen et CCF
BRUNET William	<ul style="list-style-type: none"> ● Examen et CCF
AYAT Sofiane	<ul style="list-style-type: none"> ● Examen et CCF

Semaine 10 : du 28 Mai au 1 Juin	
ROBLES Jérémie	<ul style="list-style-type: none"> ● Rapport de projet ● Maintenance du robot et de ses moteurs, changement des roues (le robot tient enfin en équilibre).
BRUNET William	<ul style="list-style-type: none"> ● Rapport de projet
AYAT Sofiane	<ul style="list-style-type: none"> ● Rapport de projet



VI. ANALYSE UML

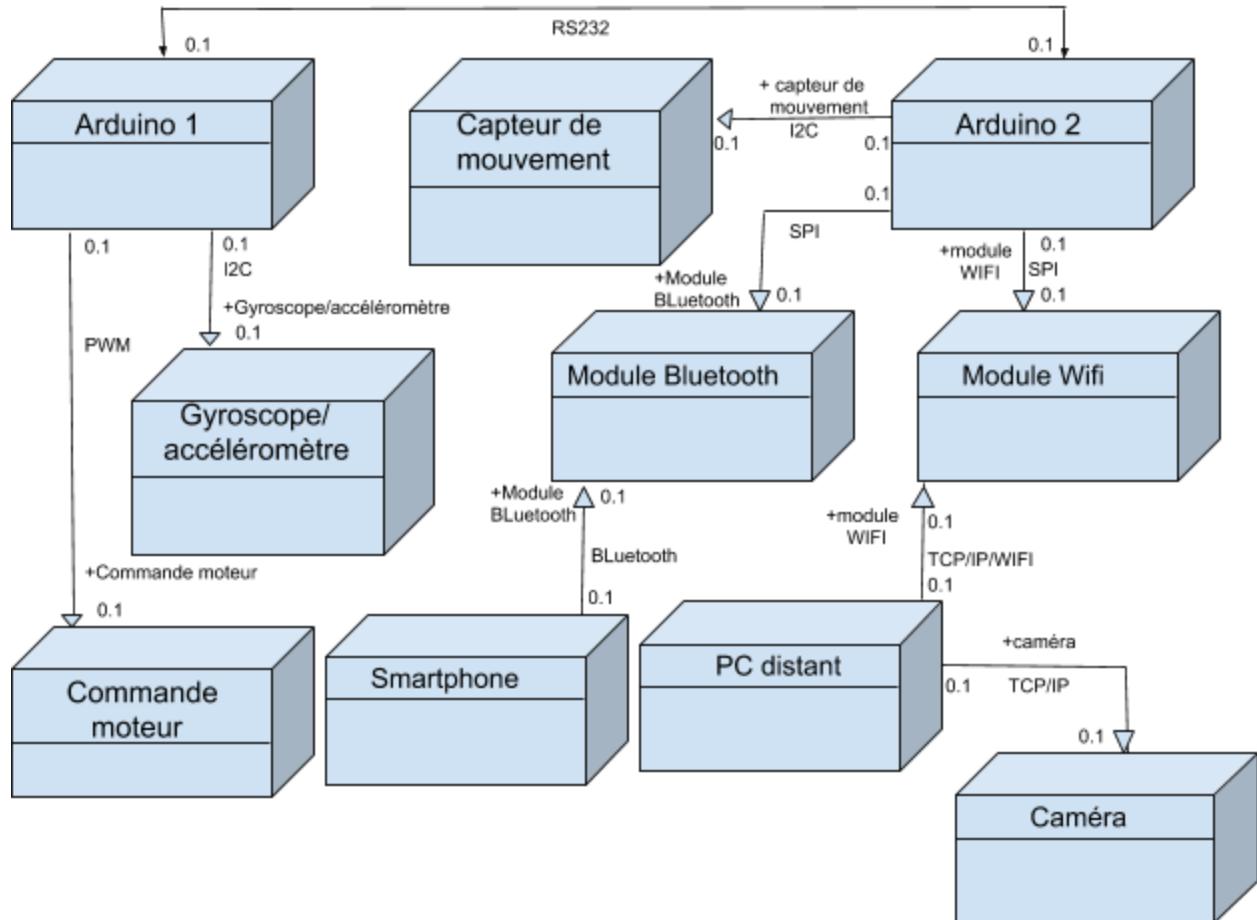
A. Diagramme de cas d'utilisation



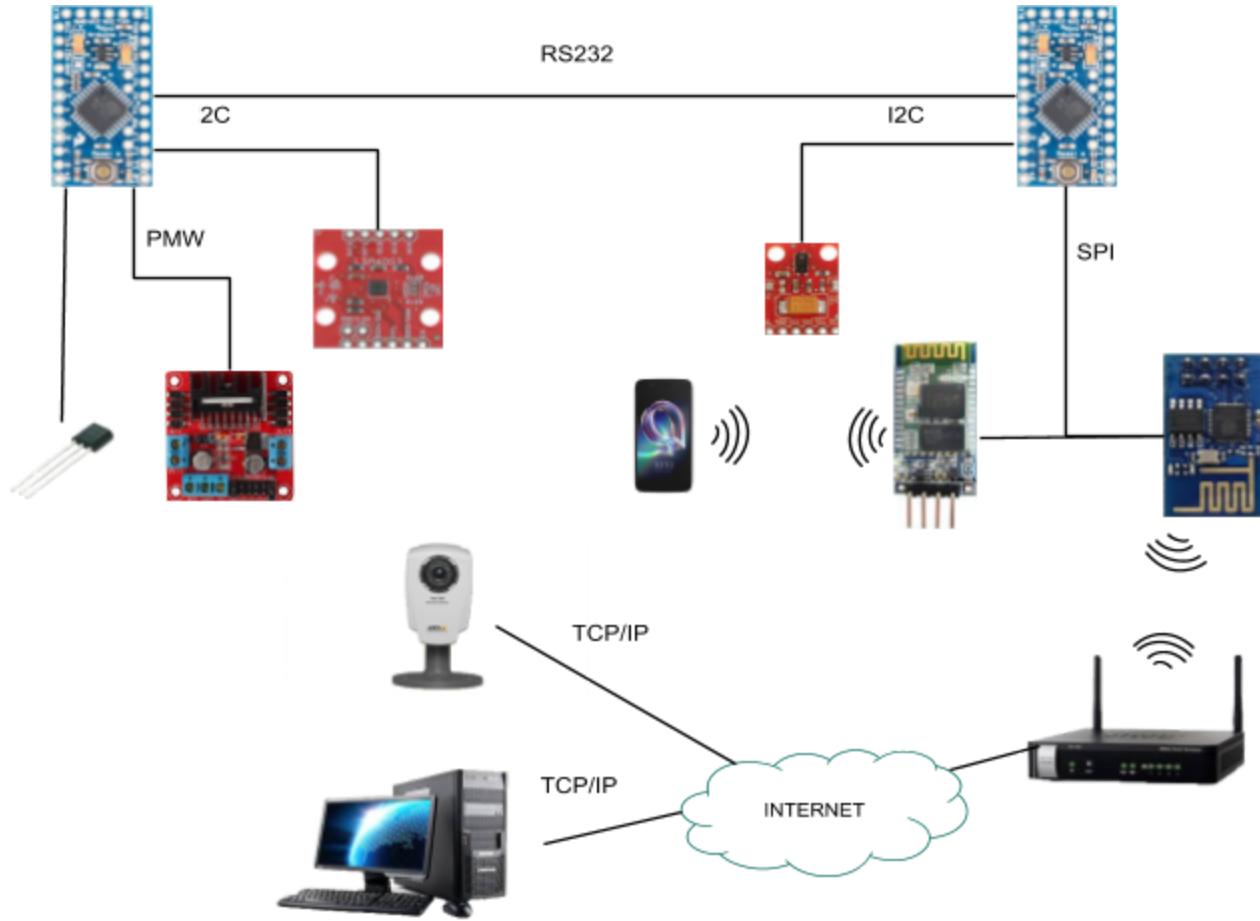
L'opérateur a la possibilité de commander la plateforme, pour acquérir les différentes commandes ; il a la possibilité d'utiliser le Bluetooth via une application sur son smartphone ou alors sur un PC ; de visualiser le robot et de le commander.

Pour commander le winky il faut actionner les moteurs afin de faire avancer la plateforme. Le châssis doit acquérir les paramètres moteur, comme : se maintenir en équilibre à l'aide des données récupérées par les capteurs.

B. Diagramme de déploiement



C. Contexte



Le premier micro-contrôleur est relié principalement au module qui permet la gestion de l'équilibre ; quant au second micro-contrôleur, il s'occupe de la communication entre l'utilisateur et le système (les deux IHM).

La raison pour laquelle nous utilisons deux micro-contrôleurs est d'éviter la surcharge d'un micro-contrôleur avec toutes les tâches nécessaires pour le fonctionnement du gyropode.



Jérémie ROBLES

Étudiant 1

I. GESTION DE L'ÉQUILIBRE DU GYROPODE

I. PRÉSENTATION DU TRAVAIL DEMANDÉ

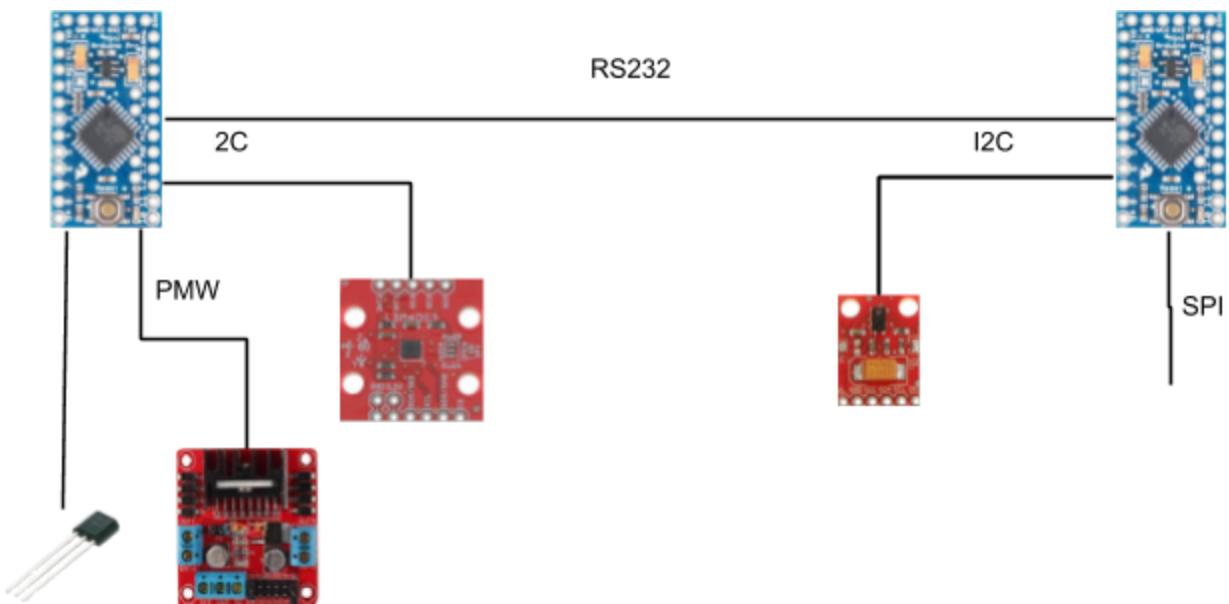
L'objectif de cette partie est de maintenir debout le gyropode sur ses deux roues, de le déplacer et d'éviter les obstacles venant de front à une distance inférieure à 6 cm. Pour cela, j'ai développé les programmes en C / C++ sur l'IDE Arduino.

Afin de le maintenir en équilibre, il m'a fallu récupérer l'angle du robot grâce à la centrale inertieille.

Ensuite, après avoir récupéré l'angle du gyropode, il faut asservir les moteurs afin de relever la plateforme à son point d'équilibre en pilotant les moteurs grâce à un pont en H permettant de contrôler la vitesse et le sens de rotation des moteurs.



II. SCHÉMA DE MON SYSTÈME



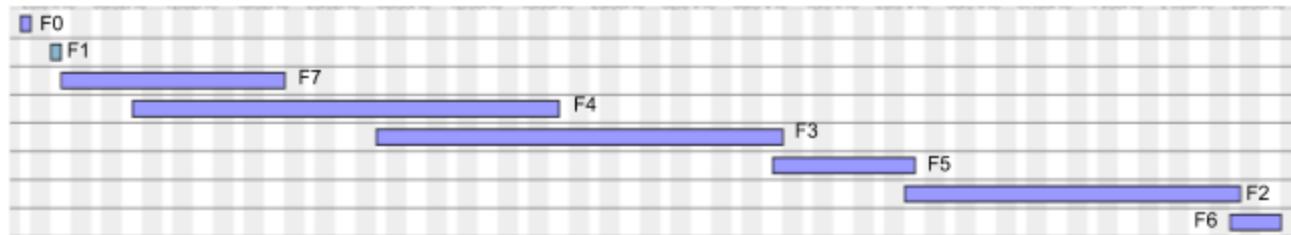
III. TRAVAIL DEMANDÉ

Les missions exigées concernant ma partie sont :

- Prise en main du projet
- Installation des logiciels nécessaires
- Mise en œuvre de l'accéléromètre
- Mise en œuvre du gyropode
- Mise en œuvre du détecteur de mouvement
- Mise en œuvre du capteur à effet Hall
- Réalisation de la gestion de l'équilibre et de mouvement de l'équilibre

Afin de réaliser mes tâches, j'ai dû suivre ce diagramme de GANTT pour ma partie :

• F0 : Prendre en main le projet	29/01/18	29/01/18
• F1 : Définir un protocole de dialogue entre les deux processeurs	01/02/18	01/02/18
• F7 : définir un protocole de dialogue entre le processeur et l'opérateur(protocole unique pour Bluetooth et wi-fi)	02/02/18	23/02/18
• F4 : Gérer la commande des moteurs (PWM vers pont en H)	09/02/18	22/03/18
• F3 : Gérer l'équilibre du robot (gyroscope + accéléromètre)	05/03/18	13/04/18
• F5 : Gérer la vitesse et la direction des moteurs (Capteur à effet Hall)	13/04/18	26/04/18
• F2 : Développer la communication RS232 entre les processeurs	26/04/18	28/05/18
• F6 : Gérer le capteur de mouvement (APDS-9960)	28/05/18	01/06/18



IV. LES COMPOSANTS

A. Les micro-contrôleurs (atmega328p)

Dans notre projet nous avons utilisé deux Arduinos pro-mini basés sur un processeur atmega328p, car ils disposent de :

CARACTÉRISTIQUES	UTILISATION
32 KBytes de mémoire flash	30% utilisés sur l'Arduino qui maintient l'équilibre du robot et 28% sur l'Arduino qui transfère les informations.
2 KBytes de mémoire ram	28% utilisés sur l'Arduino qui maintient l'équilibre du robot et 37% sur l'Arduino qui transfère les informations.
Pins I2c, A4-A5	Pour la communication avec le mpu6050 ou l'apds-9960.
Pins SPI, 10-11-12-13	Pour la communication avec le module Wifi ou Bluetooth.
Pins interrupts, 2-3	Pour développer les encodeurs.
Pins UART (RS232), 0-1	Pour la communication entre les deux Arduino.
Pins PWM, 5-6-9-10	Pour le contrôle des moteurs, le PWM est réalisé en hardware, ce qui permet d'optimiser le logiciel.



Name	ADC
Power	PWM
GND	Serial
Control	Ext Interrupt
Arduino	PC Interrupt
Port	Misc

Arduino Pro Mini (DEV-11114)

Programmed as Arduino Pro Mini w/ ATmega328
8MHz/ 3.3V

	PCINT17	TxD	PD1	D1	TX0
	PCINT16	RxD	PD0	D0	RXI
	PCINT14	PC6	Reset	RST	
			GND	GND	
	PCINT18	INT0	PD2	D2	2
OC2B	PCINT19	INT1	8-bit	PD3	D3
	XCK	T0	PCINT20	PD4	D4
TI	OCOB	PCINT21	8-bit	PD5	D5
AINO	OCCOA	PCINT22	8-bit	PD6	D6
		INT1	PCINT23	PD7	D7
	CLKO	ICP1	PCINT0	PB0	D8
	OCIA	PCINT1	8-bit	PB1	D9
					9



RAW	RAW			
GND	GND			
RST	Reset	PC6	PCINT14	
VCC	VCC			
A3	A3/D17	PC3	ADC3	PCINT11
A2	A2/D16	PC2	ADC2	PCINT10
A1	A1/D15	PC1	ADC1	PCINT9
A0	A0/D14	PC0	ADC0	PCINT8
13	D13	PB5	SCK	PCINT5 LED
12	D12	PB4	MISO	PCINT4
11	D11	PB3	8-bit	MOSI PCINT3 OC2A
10	D10	PCB2	8-bit	SS PCINT2 OC1B
A5	A5/D19	PC5	ADC5	SCL PCINT13
A4	A4/D18	PC4	ADC4	SDA PCINT12
A7	A7	ADC7		
A6	A6	ADC6		

Power
Raw: 3.3V-16V (4V-12V recommended)
VCC: 3.3V
Maximum current: 150mA @ 3.3V

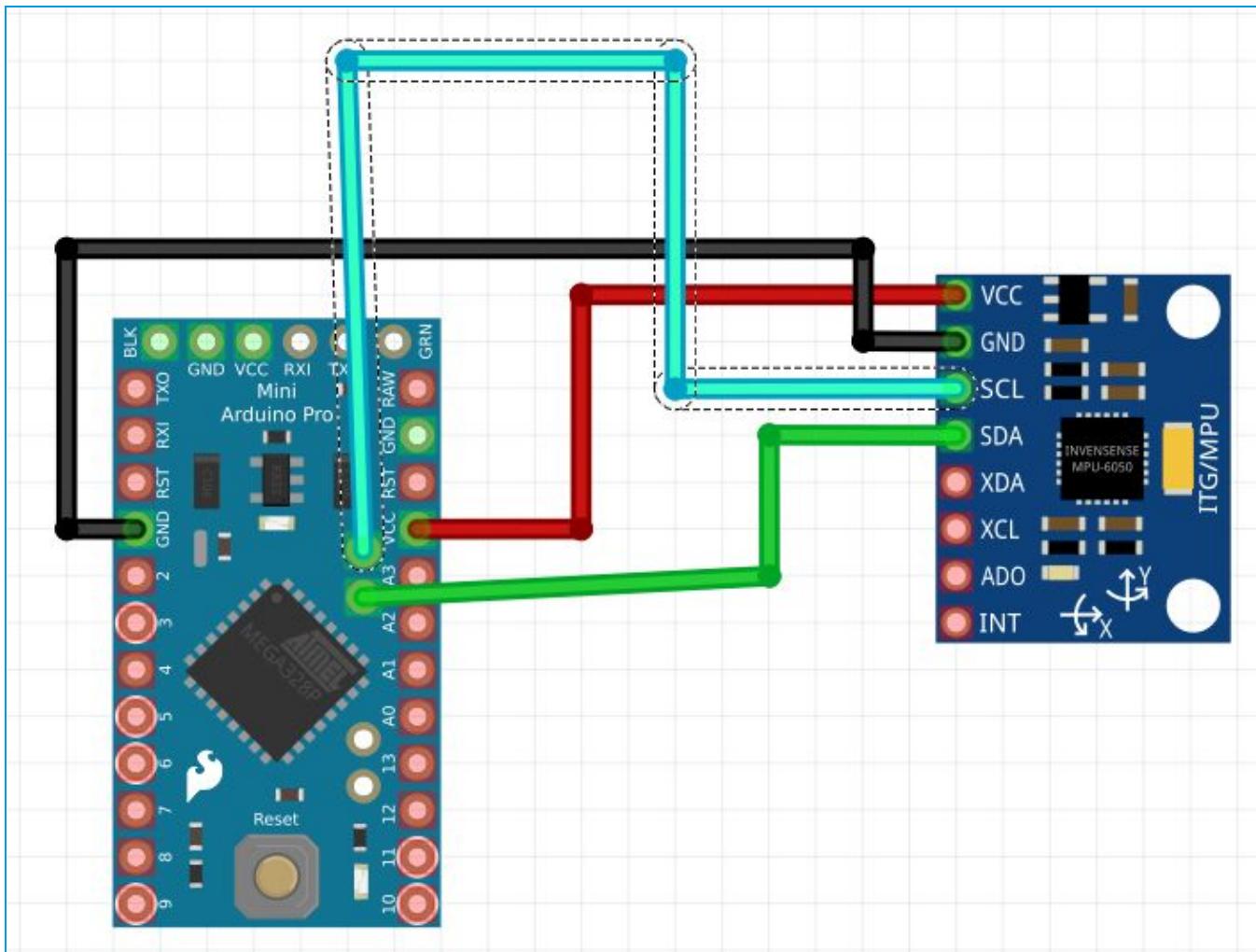
ATmega328P
Absolute maximum VCC: 6V
Maximum current for chip: 200mA
Maximum current per pin: 40mA
Recommended current per pin: 20mA
8-bit Atmel AVR
Flash Program Memory: 32kB
EEPROM: 1kB
Internal SRAM: 2kB
ADC: 10-bit
PWM: 8-bit

LEDs
Power: Red
User (D13): Green



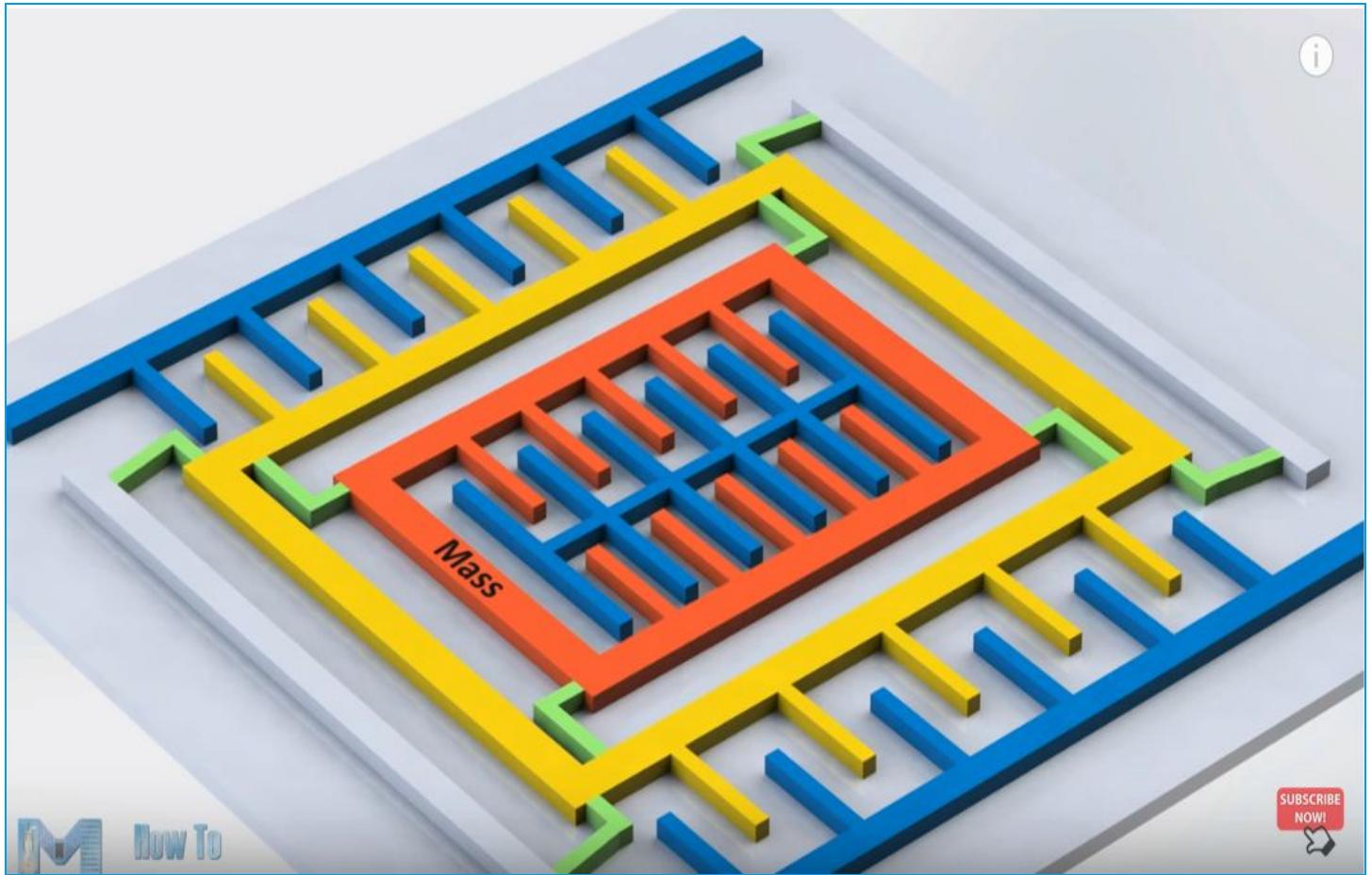
B. Le centrale inertielle (MPU6050)

La centrale inertielle est le capteur qui nous permet de récupérer l'inclinaison du robot. Pour cela, nous avons une liaison I2C entre le composant et le micro-contrôleur dédié à la gestion de l'équilibre et aux déplacements de la plateforme, le composant est à l'adresse 0x68.



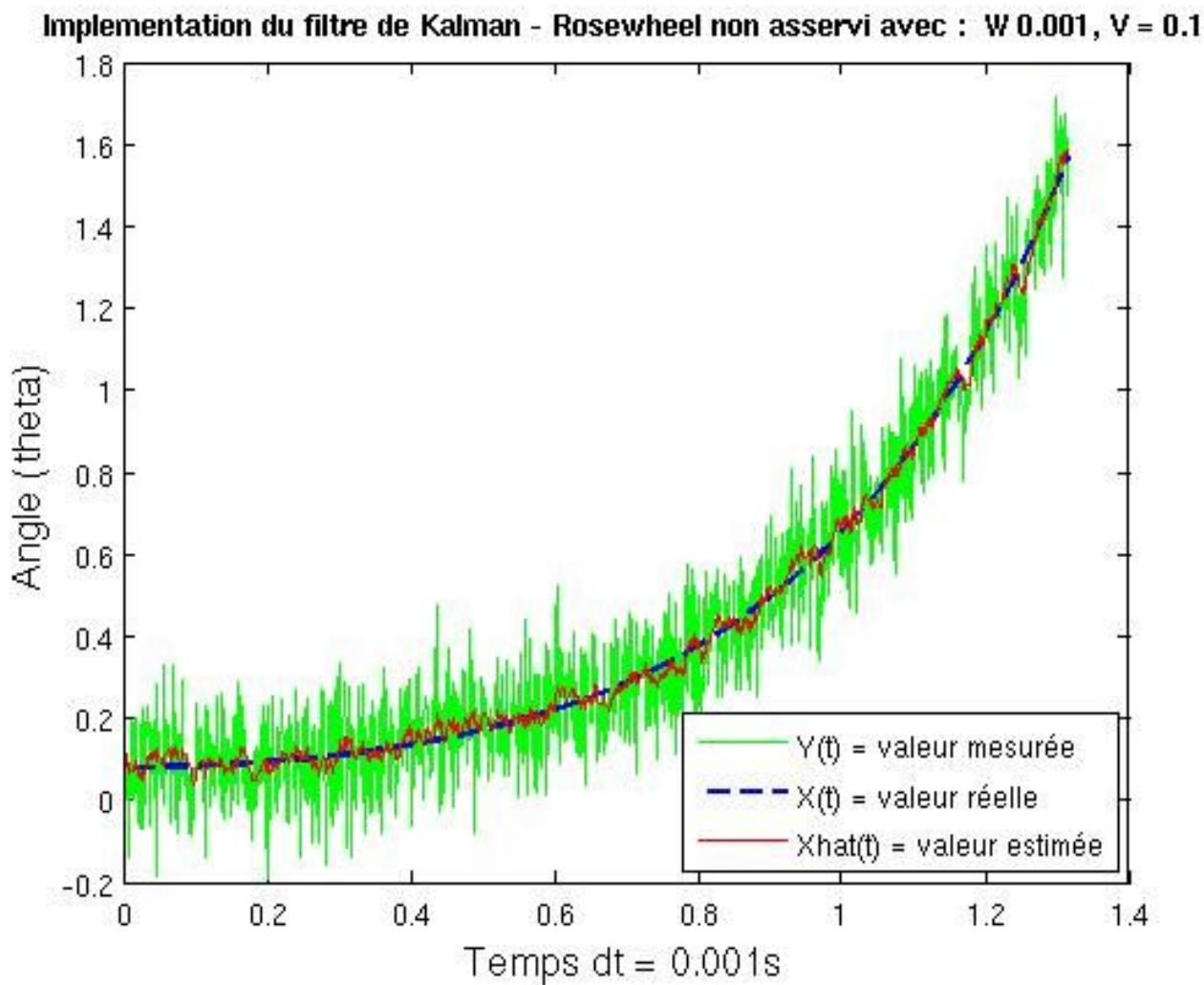
Le fonctionnement d'une centrale inertielle repose un système de glissement et de ressorts de plaques métalliques afin de récupérer l'angle d'inclinaison et l'accélération (la vitesse d'inclinaison) du composant.

Voici une vue intérieure d'une centrale inertielles :



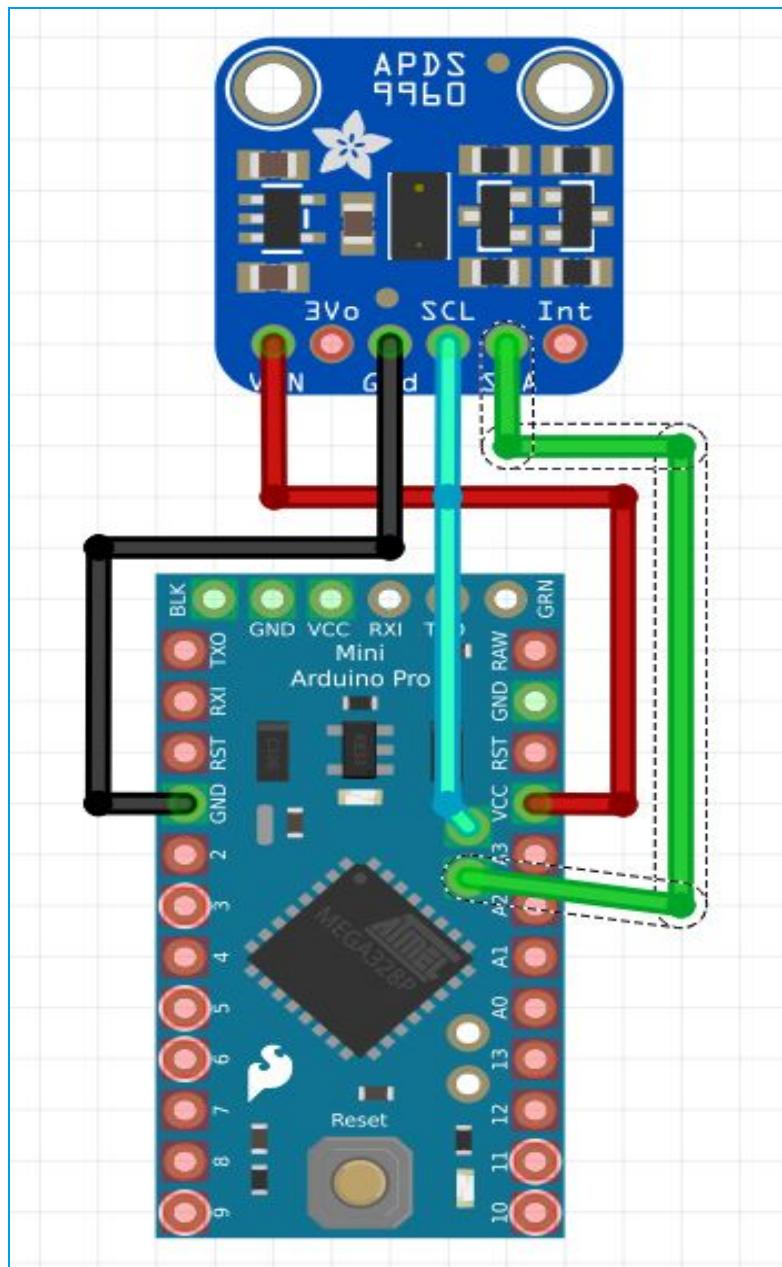
Nous avons rencontré un problème concernant l'acquisition des valeurs du MPU6050 car l'acquisition étant rapide, les valeurs reçues sont parasitées. Pour pallier cela, nous avons dû utiliser le filtre de kalman qui est un algorithme d'estimation basée sur des séries de mesures, permettant de récupérer des valeurs proches de la réalité. voir annexe p85, p121

Exemple d'application de filtre de kalman :



C. Le capteur de mouvement (APDS-9960)

L'APDS-9960 est un capteur infrarouge, capable de reconnaître différents mouvements devant lui, tel un mouvement de main de la gauche vers la droite et cela avec une portée de 10 cm. Dans notre cas nous l'utilisons comme capteur de distance. Ce composant est connecté au micro-contrôleur qui s'occupe d'acquérir les différentes commandes de déplacements, la liaison est en I2C et l'adresse du composant est 0x39.



Ce capteur permet de récupérer les valeurs des quatre points HAUT, BAS, GAUCHE, DROIT variant tous entre 0-255. Ainsi, l'analyse de leurs variations permet de reconnaître les différents mouvements devant lui, mais dans notre cas, juste la distance est une variable qui nous intéresse. Pour cela il a fallu faire la moyenne des quatre points et convertir l'intervalle 0-255 en 0-10 cm pour mesurer la distance. voir annexe p85, 146.

Extrait du code pour mesurer la distance :

```
extern String continued_rgb()
{
    String posi = "nul";
    work = I2C_READ(0xAE); //READ GESTUR FIFO LEVEL REGISTER
    if (work != 0) //IF FIFO HAS SOME DATA
    {
        DATA_U = I2C_READ(0xFC);
        DATA_D = I2C_READ(0xFD);
        DATA_L = I2C_READ(0xFE);
        DATA_R = I2C_READ(0xFF);

        distance = mapping((DATA_D+DATA_L+DATA_R+DATA_U)/4, 0, 255, 10, 0);

        if ((DATA_U > NOISE_LEVEL) && (DATA_D > NOISE_LEVEL) && (DATA_L > NOISE_LEVEL))
        {
            DATA_SYORI();
            PHASE_COUNTER++;
            DISP_FLAG = 1;
        }
        else
        {
            if (DISP_FLAG)
            {
                DISP_FLAG = 0;
                posi = DISP_DIR();
            }
            RESET_VARIABLE();
            return posi;
        }
    }
}
```

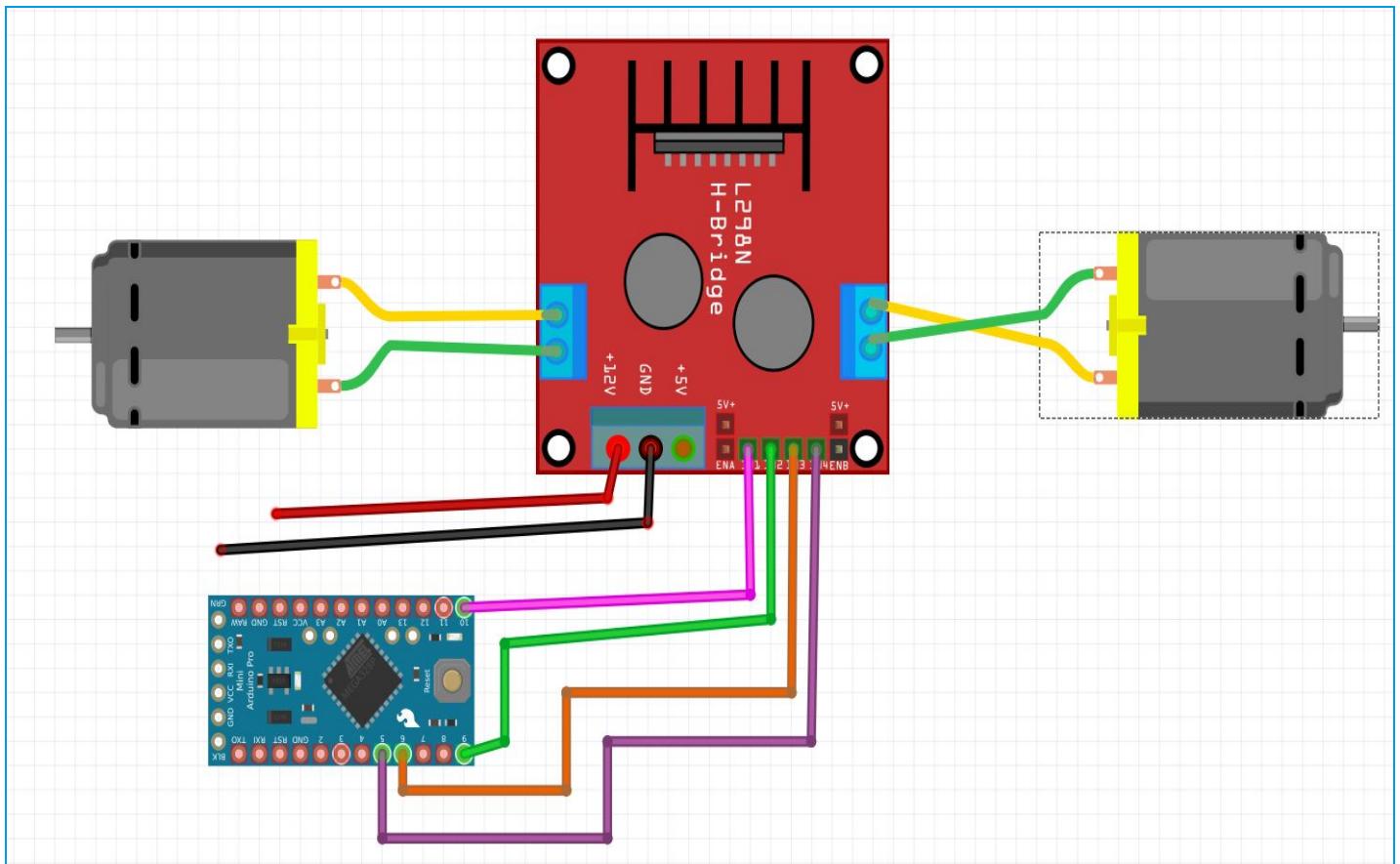
Variables entre 0-255 (0-10cm) pour les différents points (HAUT, BAS, GAUCHE, DROIT)



D. Le pont en H (LN298N)

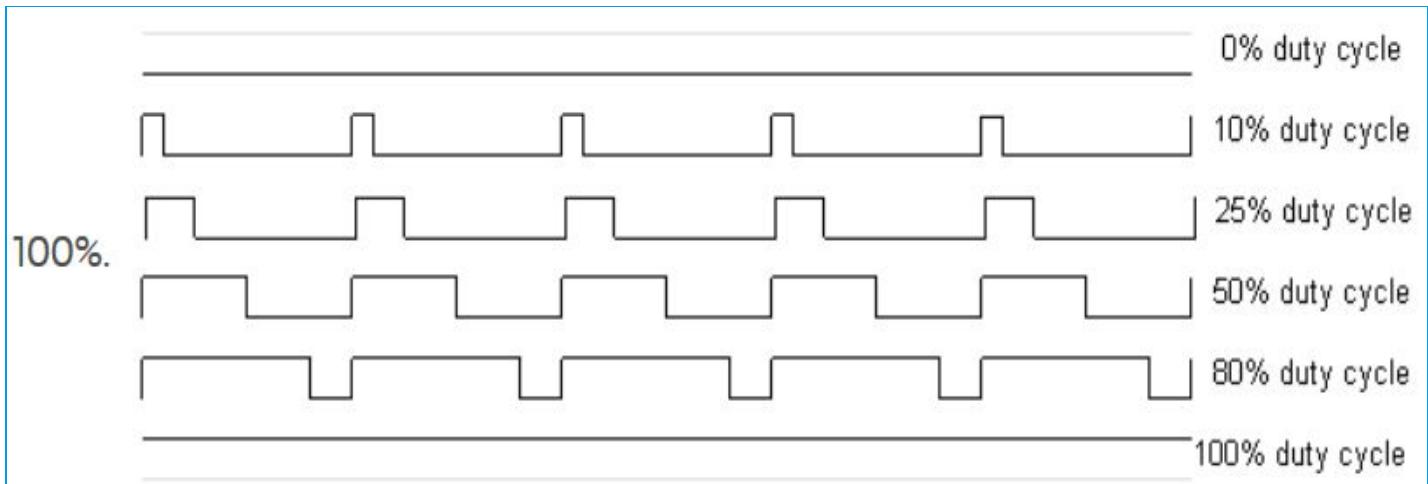
Le pont en H est un composant qui permet de faire varier la tension, ainsi donc le sens de rotation des moteurs et leur vitesse grâce à un signal modulé en PWM. Le signal PWM est un signal formé à partir de variation d'état HAUT et d'état BAS sur une période donnée. Plus l'état HAUT sera présent sur la période, plus la valeur du signal sera élevée.

Schéma de câblage du pont en H:



Notre pont en H a son alimentation branchée sur +15v car l'autre entre ne peut excéder les 5V et notre batterie dédiée aux moteurs est en 7.4V.

Exemple de signaux PWM :



Plus le pourcentage du cycle sera élevé, plus la vitesse des moteurs le sera. Afin de se maintenir debout, le gyropode actionne les moteurs en avant et en arrière mais à une vitesse contrôlée par un régulateur de PID (proportionnel, intégral, dérivée) qui est un algorithme qui régule des variables afin d'obtenir au plus près la valeur de sortie souhaitée. Dans notre cas, la valeur à rejoindre est le point d'équilibre. voir annexe p110.

L'algorithme utilisé :

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

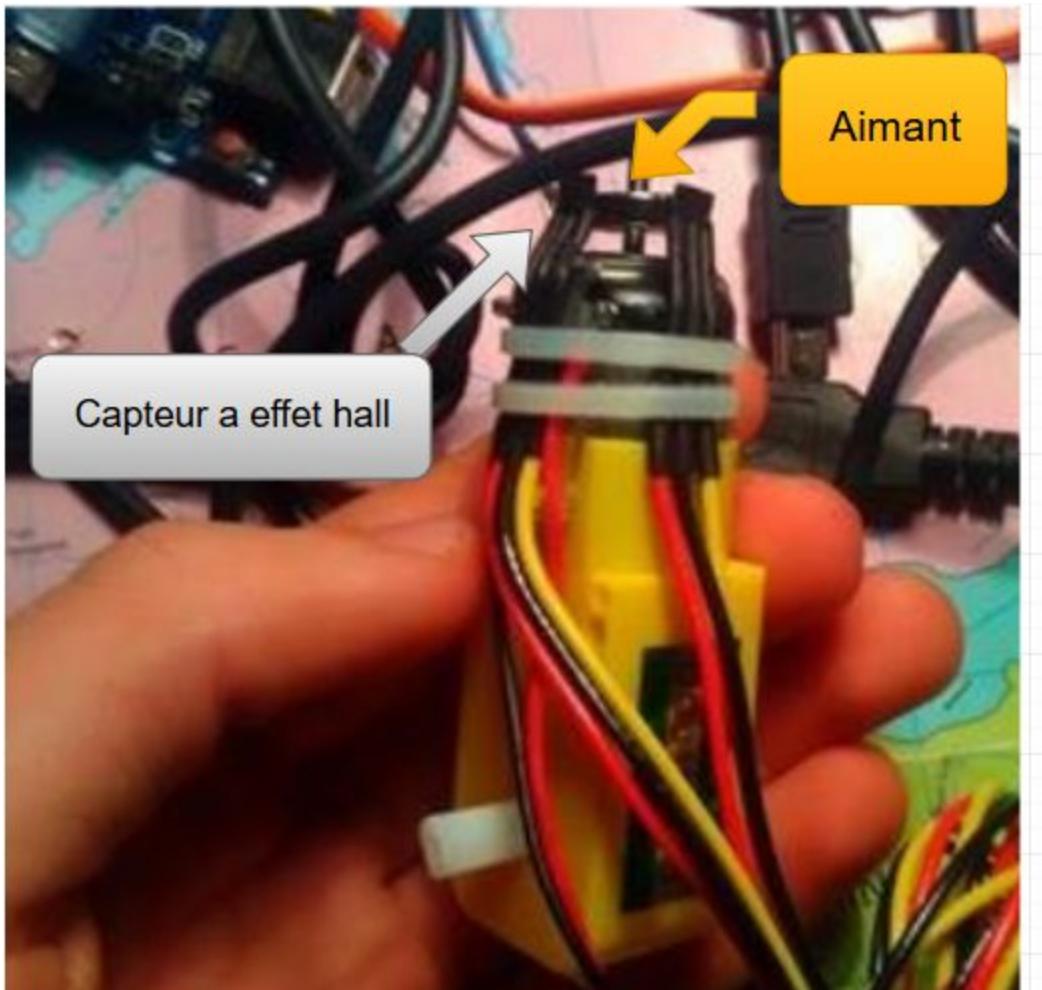
Tous les x millisecondes , faire :

```
erreur = consigne - mesure;  
somme_erreurs += erreur;  
variation_erreur = erreur - erreur_précédente ;  
commande = Kp * erreur + Ki * somme_erreurs + Kd * variation_erreur ;  
erreur_précédente = erreur
```

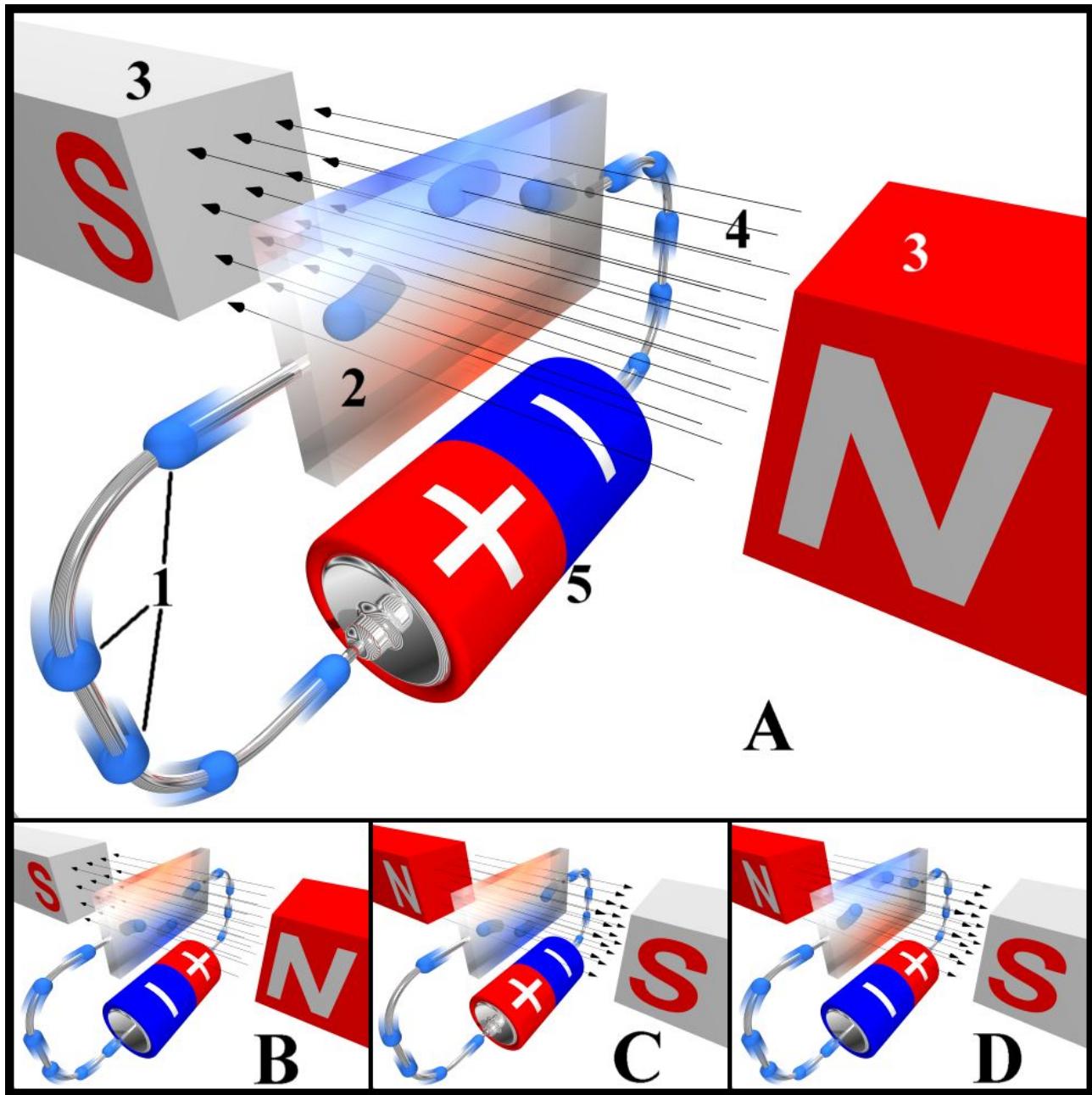
E. Les capteurs à effet Hall (SS411P)

Le capteur à effets Hall permet de détecter la variation des pôles Nord et Sud grâce à la déviation d'électrons dans un conducteur (le capteur) du au champ magnétique de l'aimant. Pour cela nous avons fixé un aimant à l'arrière du moteur, plus précisément sur son arbre. Grâce à cela, à chaque rotation du moteur, l'aimant tourne sur lui-même et le capteur fixé à proximité lance une fonction en interrupt qui va incrémenter l'interruption sur un temps d'échantillonnage. voir annexe p 103

Montage du moteur du robot gyropode :



Représentation du fonctionnement du capteur :



V. LA PROGRAMMATION

A. L'IDE Arduino

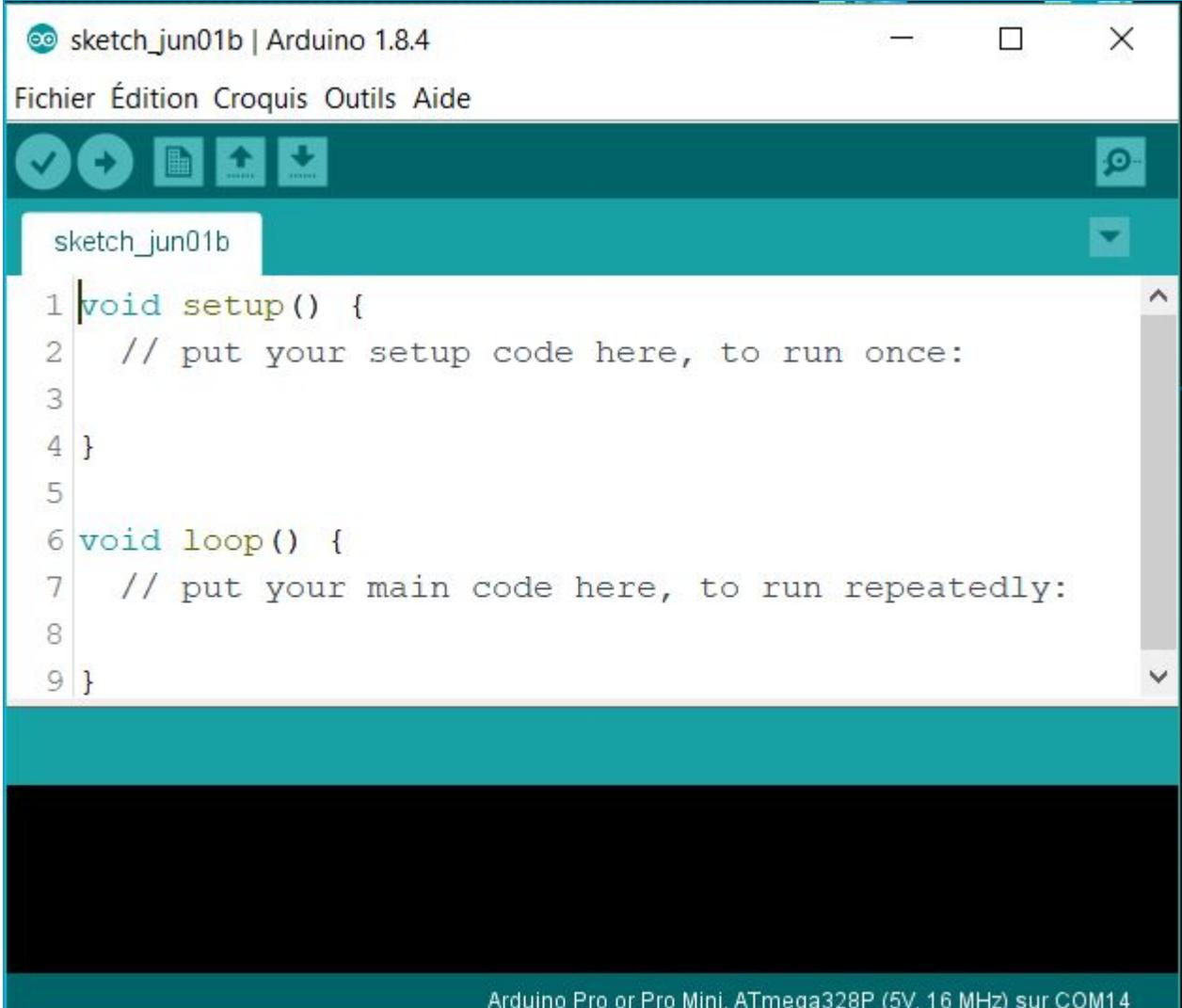
Arduino est une marque de carte électronique principalement composée de processeur ATMEL AVR dont les schémas sont sous licence libre.

Arduino c'est aussi un IDE (Integrated Development Environment) JAVA, comprenant son langage orienté objet fondé sur les langages C / C++, le Arduino permet de faciliter la programmation de leurs cartes et bien d'autres, tel que les ESP ou des micro-contrôleurs avec un processeur ARM.

L'environnement Arduino est composé d'une grande communauté permettant de faciliter la compréhension et la résolution de divers problèmes électroniques ou logiciels.



Aperçus IDE Arduino:



The screenshot shows the Arduino IDE interface. The title bar reads "sketch_jun01b | Arduino 1.8.4". The menu bar includes "Fichier", "Édition", "Croquis", "Outils", and "Aide". Below the menu is a toolbar with icons for save, run, upload, and download. The main area displays the code for "sketch_jun01b":

```
1 void setup() {  
2     // put your setup code here, to run once:  
3  
4 }  
5  
6 void loop() {  
7     // put your main code here, to run repeatedly:  
8  
9 }
```

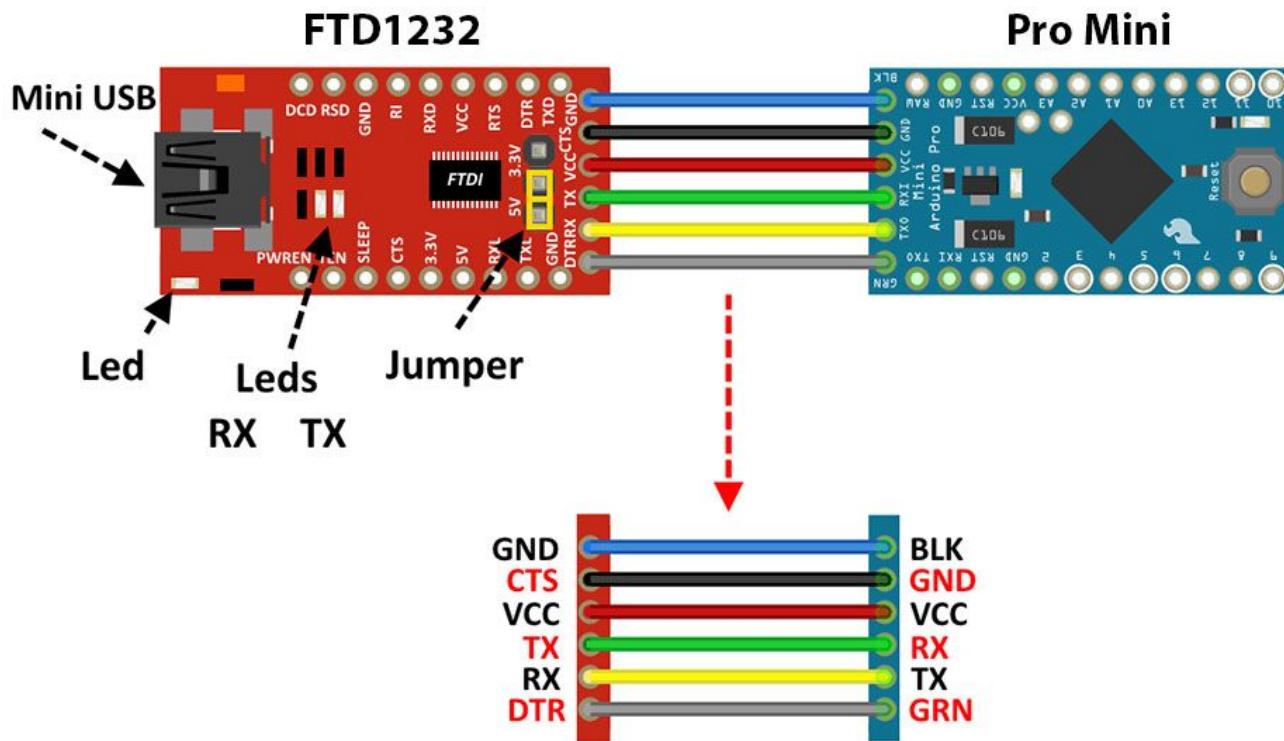
At the bottom of the code editor, there is a black bar. The status bar at the bottom right of the IDE window displays the text "Arduino Pro or Pro Mini, ATmega328P (5V, 16 MHz) sur COM14".



B. Liaison micro-contrôleur au PC

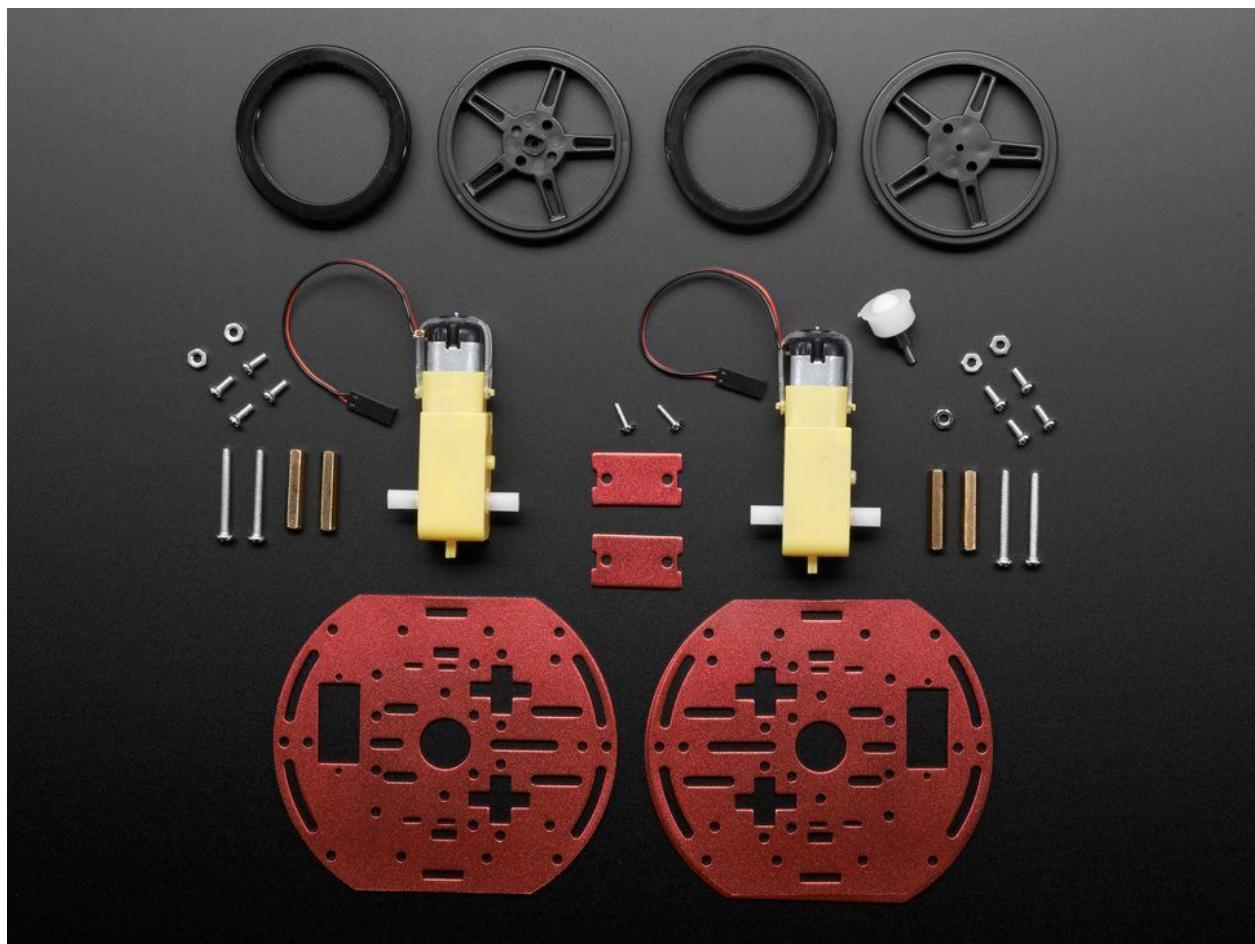
Notre Arduino étant dépourvu d'interface USB, pour la liaison avec le PC nous avons dû utiliser un FTDI (Future Technology Devices International). Un FTDI est une carte électronique du nom de l'entreprise qui l'a conçue, qui permet d'interfacer notre micro-contrôleur avec le PC afin de pouvoir téléverser nos programmes.

Schéma de câblage du FTDI avec l'Arduino pro mini:



VI. LA RÉALISATION DU GYROPODE

Pour réaliser notre gyropode, la startup Mainbot nous a fourni ce kit de robot adafruit sur 3 roues (2 sont motorisés et une roue folle) du nom de “Mini Round Robot Châssis Kit - 2WD with DC Motors” sur le site du vendeur.

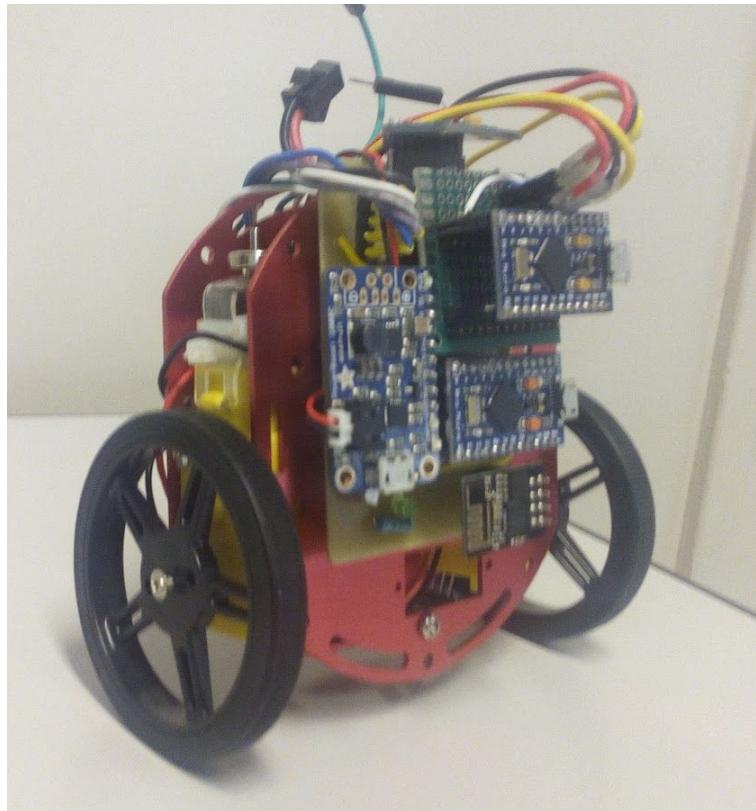




Ils nous ont aussi fourni l'ensemble des composants électroniques de notre plateforme robotique, tels que les micro-contrôleurs Arduino, un power boost 1000C (un rehausseur de tension 3v en 5v), une batterie 3v3 à 700 mA et une batterie 7v4 a 500 mA.

J'ai réalisé différents prototypes de gyropode (en utilisant toujours le même chassis), j'avais commencé notre premier gyropode avec notre première version de shield (un adaptateur reliant l'ensemble de nos composants électroniques entre eux) réalisé en impression chimique.

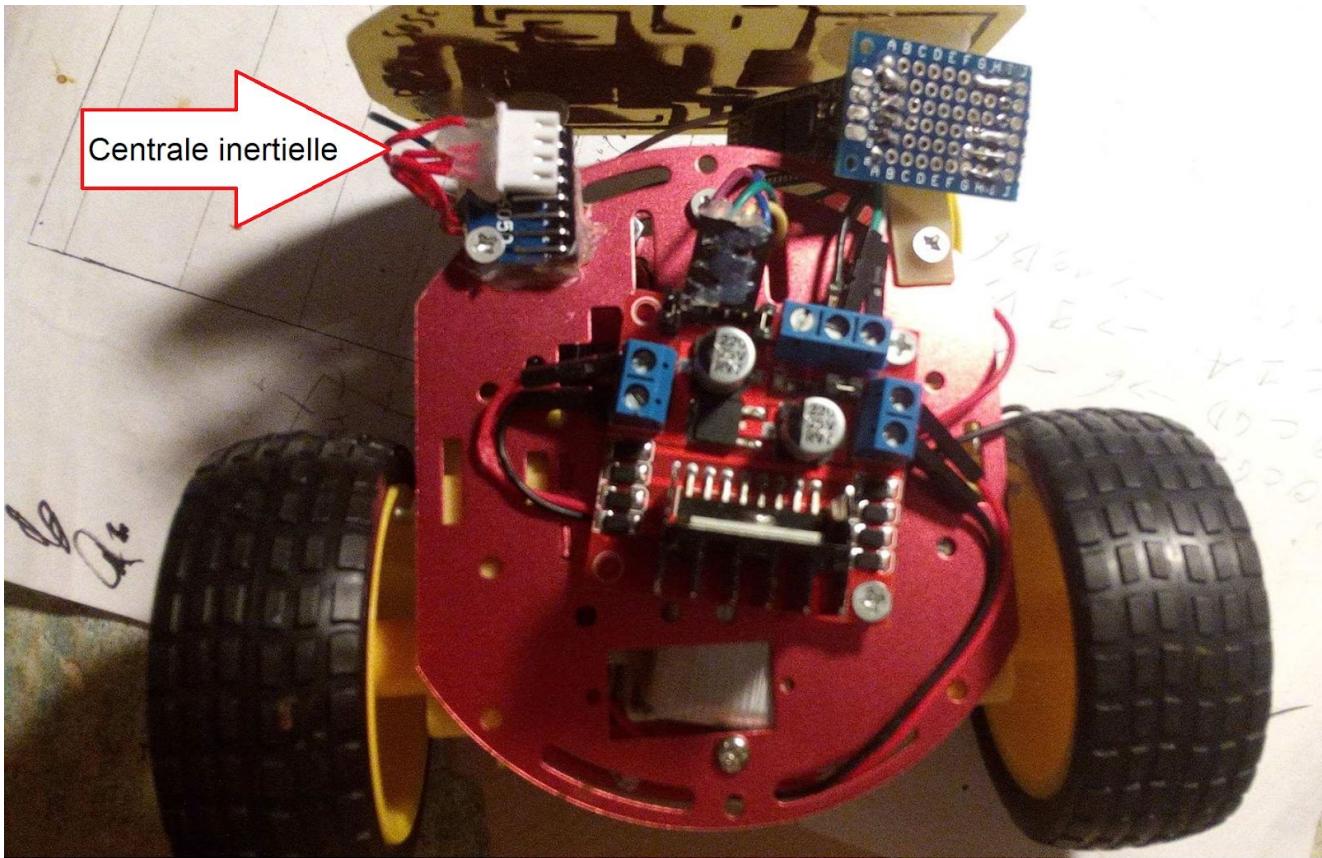
Première version du gyropode :



J'ai dû revoir notre conception en raison de divers problèmes rencontrés, comme la mauvaise conception du shield, car j'ai réalisé des liaisons RS232 au lieu de faire une liaison SPI à cause d'une mauvaise compréhension du cahier des charges. J'ai aussi eu des retours de la centrale inertuelle assez variables, car le composant était fixé sur la carte électronique qui, lors des mouvements du gyropode, tremblait, n'était pas assez stable et ne pouvait donc pas récupérer correctement la position du châssis.

J'ai donc fixé notre centrale inertuelle au châssis et sur un point en hauteur afin d'avoir une plus grande sensibilité par rapport à un point proche des roues.

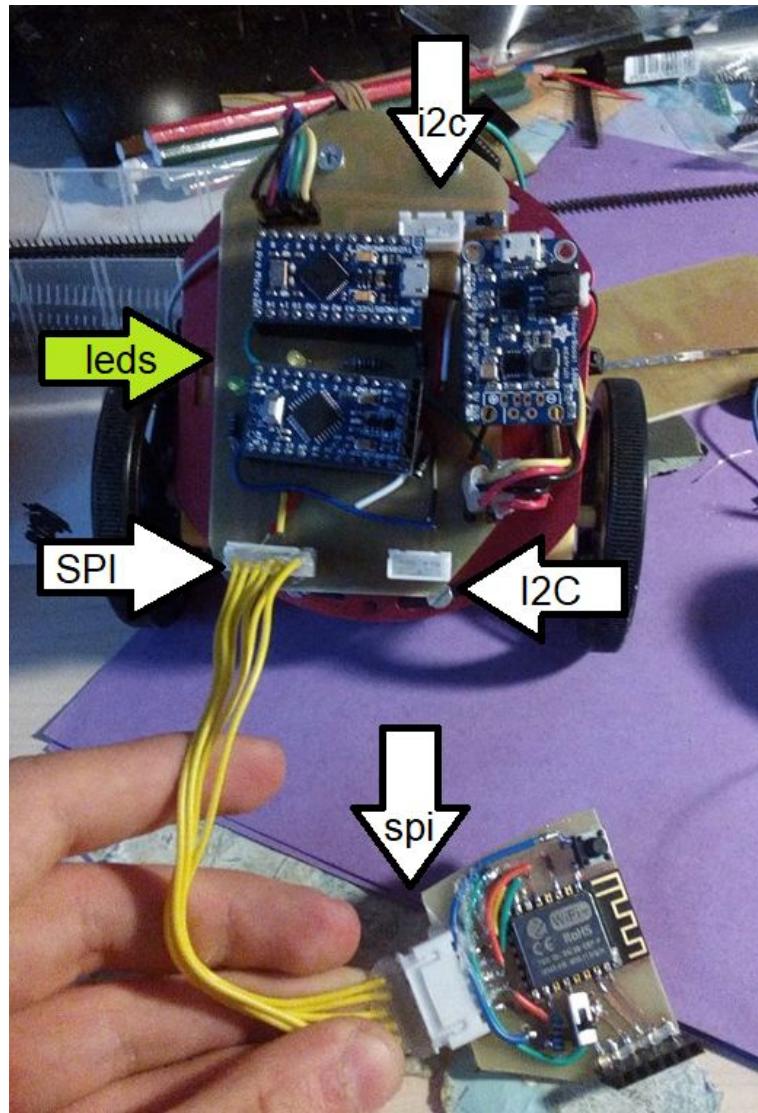
Nouvelle position de la centrale inertuelle :



Concernant le développement du shield, j'ai tenu compte de la liaison SPI, j'ai aussi ajouté des leds pour du debug de programme (afin de vérifier la réception des commandes de l'utilisateur par exemple). J'ai mis un régulateur de tension en 3v (un LM1117T) afin de pouvoir alimenter correctement notre module Wifi (notre module Bluetooth n'est pas impacté car il peut être lui aussi alimenté en 3v).

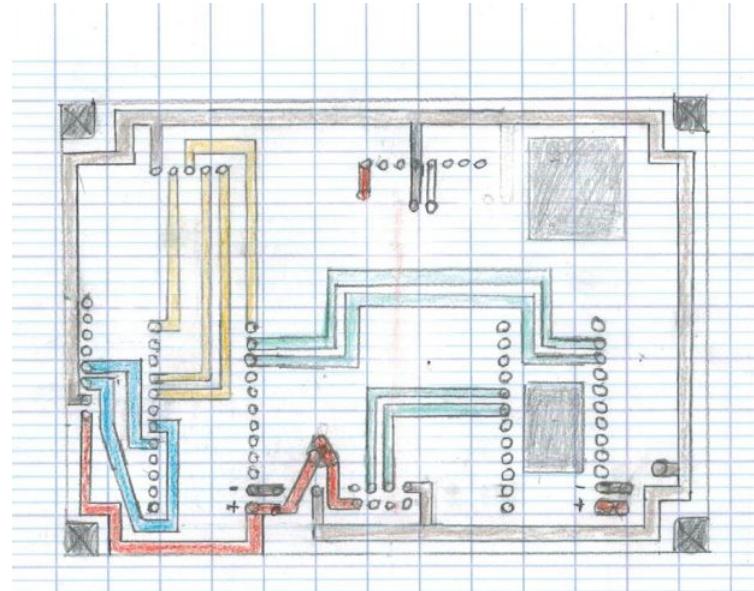
Pour faciliter la fixation et le remplacement de composants, j'ai utilisé un système de connecteur entre mon shield et mes composants.

Image du nouveau shield :

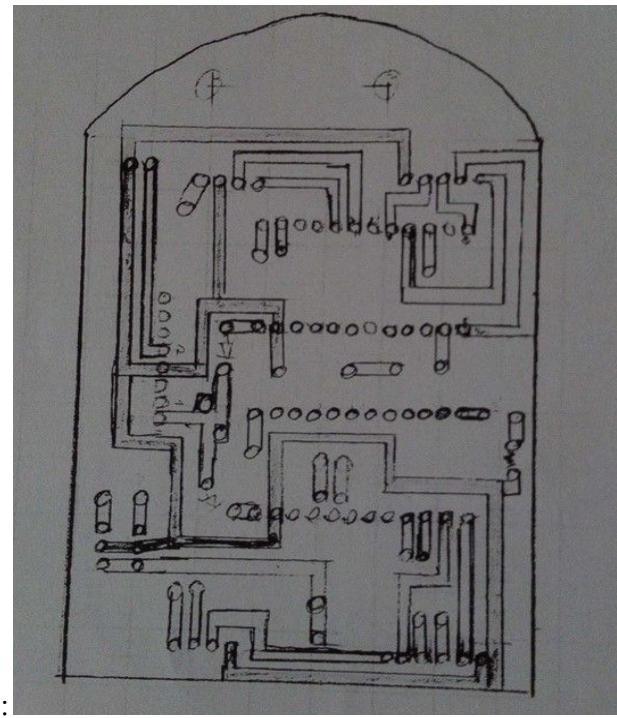


Pour réaliser mes shields en impression chimique, j'ai commencé par faire divers patrons à échelle 1 sur papier des liaisons électroniques entre les composants.

Premier patron de la première version du shield:

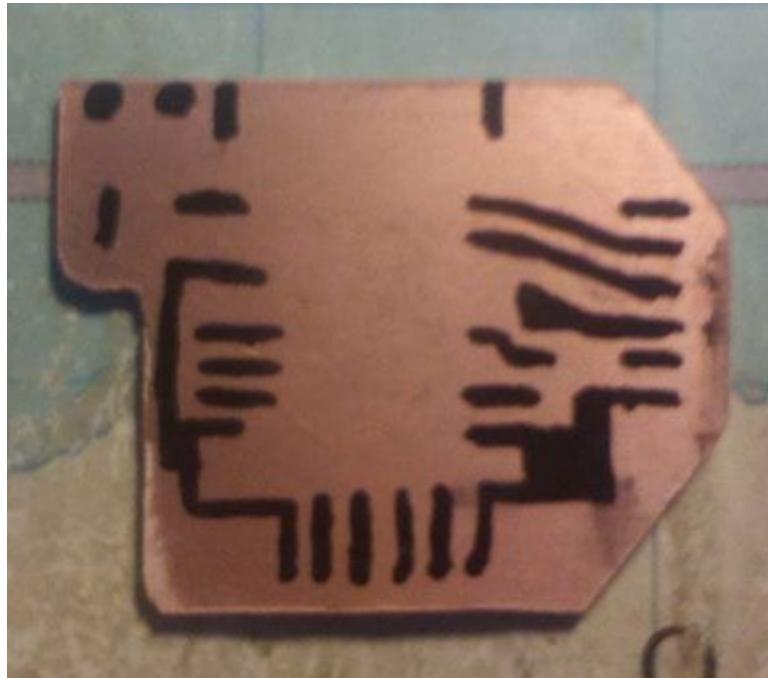


Second patron de la dernière version du shield :



Après avoir découpé les pistes sur le patron en papier, on le pose sur une plaque de cuivre (poncée par avance afin d'enlever la couche protectrice) et on trace nos pistes au feutre indélébile.

Exemple pour notre shield Wifi :



Attention :

La gravure chimique a été effectuée en dehors du lycée avec du perchlorure de fer, moins dangereux que l'acide chloridrique combiné à de l'eau oxygénée, mais il reste dangereux et corrosif sur la peau si les précautions de sécurité ne sont pas respectées : **toujours porter des gants lors de la manipulation du produit, le faire dans des récipients en plastique et dans une pièce aérée**. De plus, le perchlorure de fer est un produit polluant, donc peu recommandable, il a été utilisé à titre exceptionnel car il est peu coûteux et permet de produire des cartes électroniques rapidement.

L'utilisation et l'apprentissage de logicielle comme Kicad (logicielle de conception électronique) aurait été préférable mais l'acquisition des cartes aurait été trop longue (de par l'apprentissage, la conception, l'importation et les tests).

À gauche, notre récipient de perchlorure de fer et à droite, notre perchlorure de fer :



Après avoir rincé notre carte abondamment avec de l'eau et une fois effacées les pistes tracées au feutre indélébile avec de l'alcool à 90° on a pu récupérer notre carte, il ne reste plus qu'à la percer à la dremel et à souder les composants.



VII. CONCLUSION

Je remercie la startup Mainbot de nous avoir permis de contribuer au développement de l'un de leurs futurs produits et je remercie leur équipe de nous avoir reçus et conseillés pendant ce projet.

J'ai personnellement apprécié ce challenge assez compliqué de réaliser un gyropode et de le maintenir en équilibre, j'ai pu ainsi approfondir mes connaissances en informatique surtout en programmation registre de processeur AVR.

J'ai beaucoup aimé réaliser mes propres extension pour nos microcontrôleurs et l'ensemble de nos composants.

Je remercie mes deux camarades de m'avoir accompagné et soutenu durant ce projet.





I. PRÉSENTATION DU TRAVAIL DEMANDÉ

Le but de cette partie est de développer une application mobile grâce au logiciel Android Studio, écrit en java et en xml, qui permettra de piloter librement une plateforme (gyropode).

Cette application enverra ensuite des données utiles au pilotage du gyropode à un module Bluetooth, le HC-06, qui à son tour transférera ces données à la carte Arduino 2.

Le code nécessaire au transfert de ces données sera développé sur le logiciel Arduino écrit en C++, utilisé pour programmer les fameuses cartes Arduino.

Enfin, cette dernière aura pour rôle de communiquer ces données aux deux moteurs qui s'actionnent afin de faire avancer, reculer ou tourner la plateforme selon l'envie de l'utilisateur.



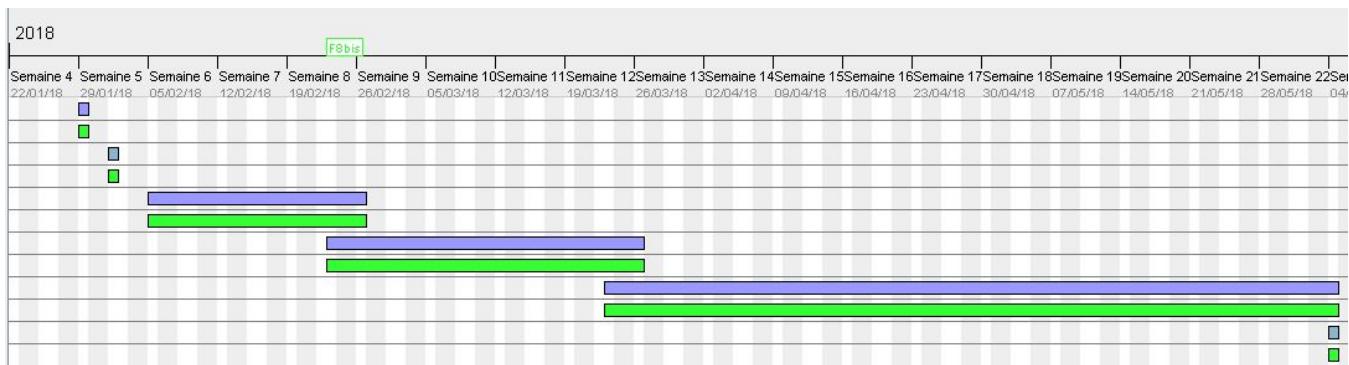
II. TRAVAIL DEMANDÉ

Les missions exigées concernant ma partie sont :

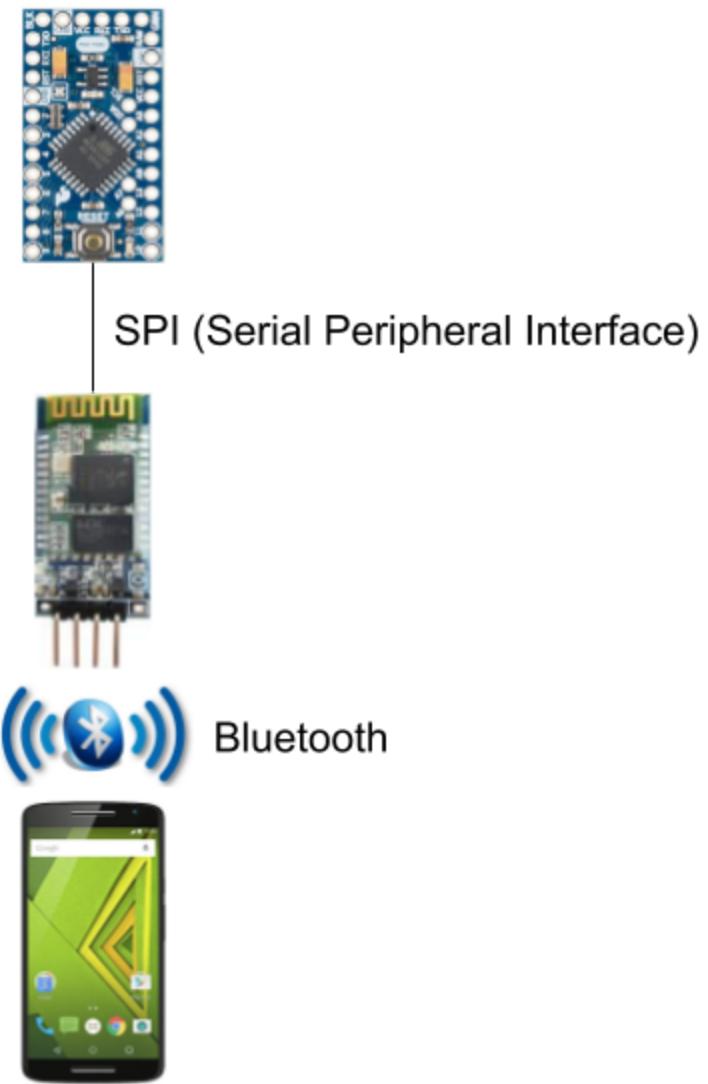
- Définir un protocole de dialogue entre les deux processeurs.
- Définir un protocole de dialogue entre le processeur et l'opérateur.
- Développer le module de communication Bluetooth côté processeur.
- Développer l'application de commande sous Android (smartphone).
- Documenter l'application.

Voici le planning sur lequel je me suis appuyé pour accomplir ces tâches :

Nom	Date de début	Date de fin
• F0 : Prendre en main le projet	29/01/18	29/01/18
• F0bis	29/01/18	29/01/18
• F1 : Définir un protocole de dialogue entre les deux processeurs	01/02/18	01/02/18
• F1bis	01/02/18	01/02/18
• F7 : définir un protocole de dialogue entre le processeur et l'opérateur(...	05/02/18	26/02/18
• F7bis	05/02/18	26/02/18
• F8 : Développer le module de communication Bluetooth côté processeur	23/02/18	26/03/18
• F8bis	23/02/18	26/03/18
• F9 : Développer l'application de commande sous Android (smartphone)	23/03/18	04/06/18
• F9bis	23/03/18	04/06/18
• F14 : Documenter l'application	04/06/18	04/06/18
• F14bis	04/06/18	04/06/18



III. SCHÉMA DE CÂBLAGE



IV. LE MATÉRIEL UTILISÉ

A. Les Hardwares

Un appareil Android :

Tablette

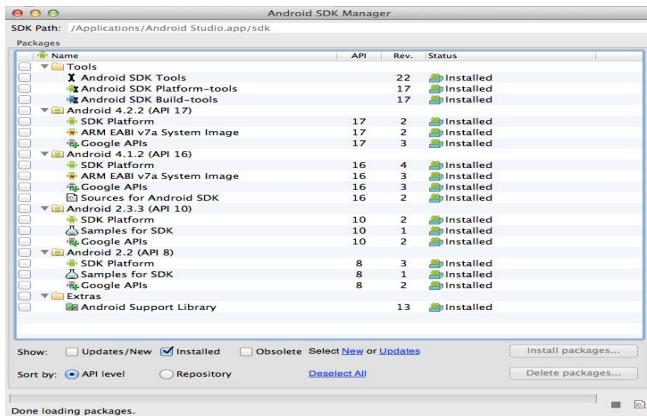
ou

Smartphone

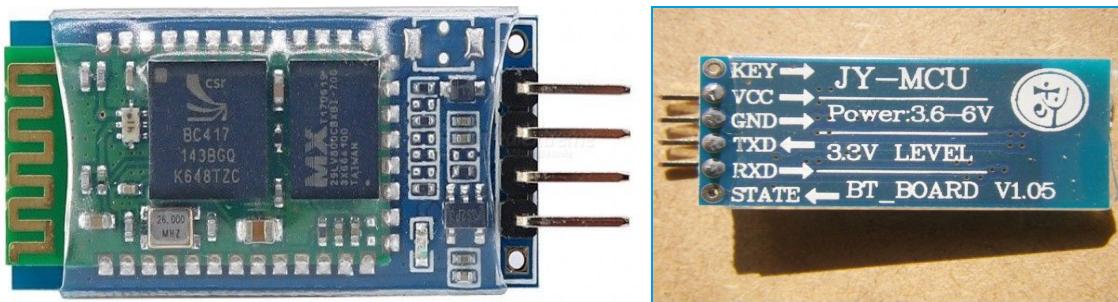


Il suffit de posséder un smartphone ou une tablette fonctionnant sous la version 4.4 ou plus récente d'Android pour que l'application puisse fonctionner correctement.

J'ai notamment installé tous les SDK (fichiers ressources d'Android) permettant de compiler des programmes.



Le module Bluetooth (HC-06)



Par soucis d'informations sur les pâtes MOSI, MISO et CLK (ou SLK) je n'ai pas pu faire communiquer mon module Bluetooth avec la carte Arduino Pro Mini en SPI.

En effet, les caractéristiques des pâtes ne sont pas spécifiées dans la datasheet du module.

Cependant, j'ai pu faire communiquer la carte et le module en Uart grâce aux pâtes RX et TX.

Comme il est indiqué sur le module, l'intervalle de tension à respecter afin que le module puisse fonctionner correctement est de 3,6V - 6V.

Cependant les pattes RX (envoyer) et TX (recevoir) servent quant à elles à l'envoi et à la réception de données, et doivent être alimentées par 3,3V.

Voici une partie de la datasheet du module informant sur les pâtes que j'ai du utiliser pendant ce projet :

PIN Name	PIN #	Pad type	Description	Note
----------	-------	----------	-------------	------

GND	13 21 22	VSS	Ground pot
-----	----------	-----	------------

VCC	12	3.3V	
-----	----	------	--

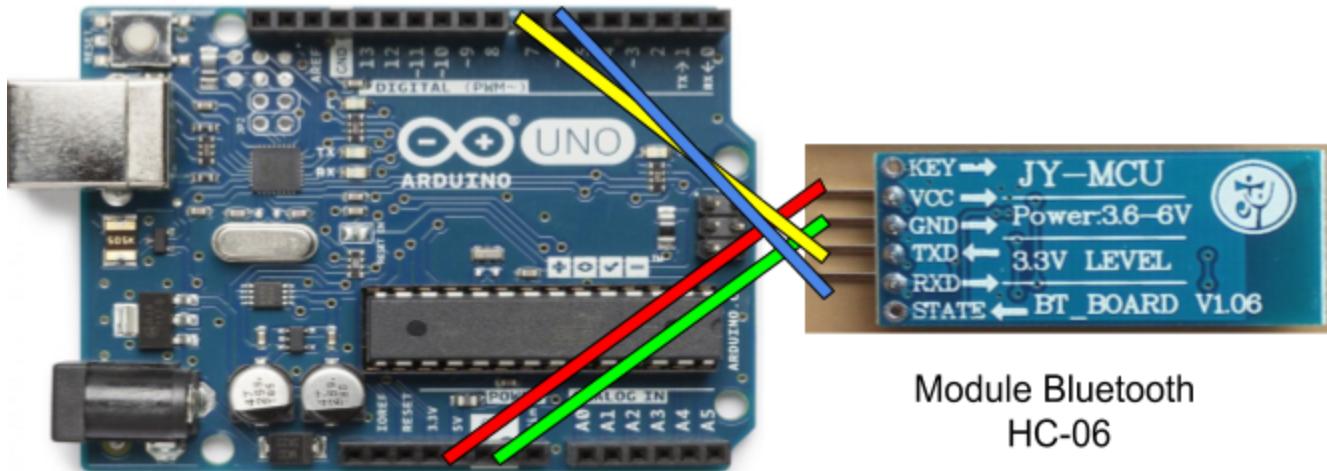
UART_RX	2	CMOS input with weak internal pull-down	UART Data input
UART_TX	1	CMOS output, Tri-stable with weak internal pull-up	UART Data output



La carte Arduino UNO

C'est cette carte-ci qui m'a permis de faire les premiers tests pour vérifier si l'application mobile fonctionnait bien.

Voici le montage réalisé avec le module Bluetooth connecté à la carte Arduino UNO :

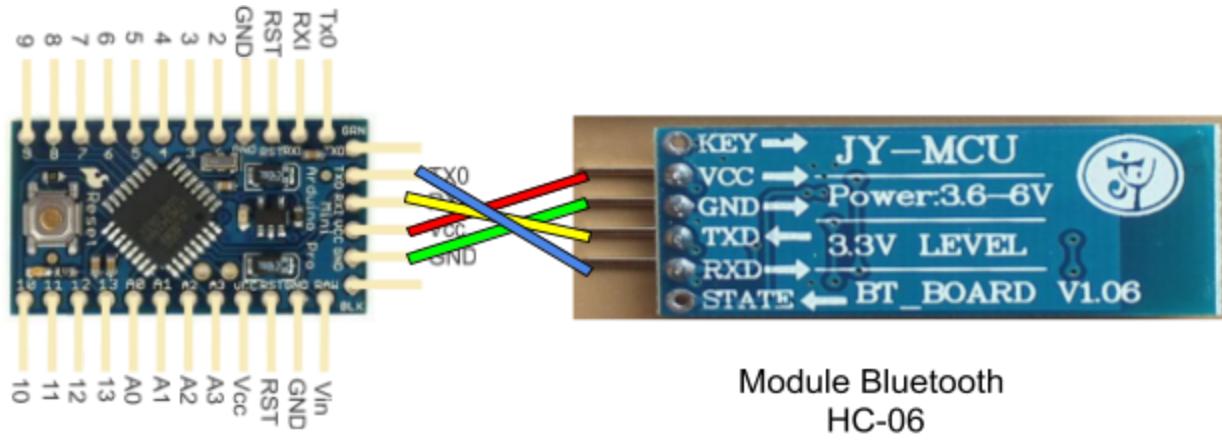


Carte Arduino UNO

Module Bluetooth
HC-06

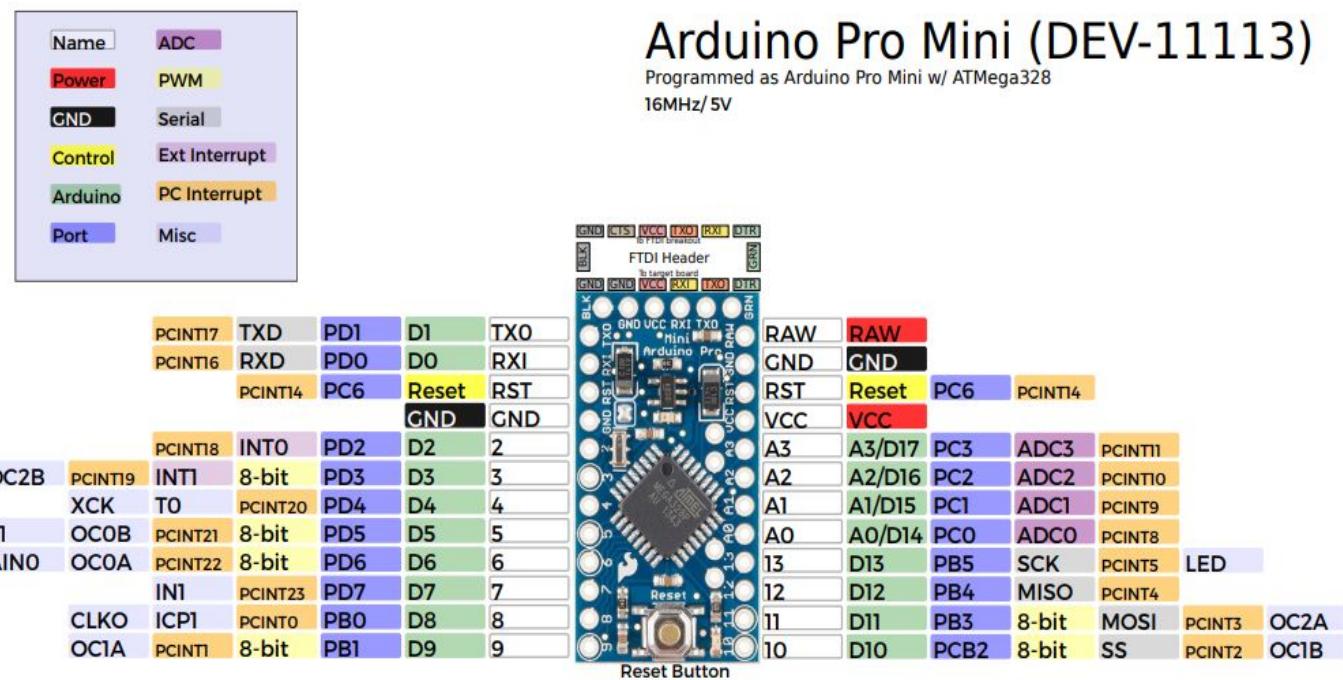
La carte Arduino Pro Mini

Le montage réalisé entre la Arduino Pro Mini et le module Bluetooth est le même que celui utilisé entre la Arduino UNO et le module Bluetooth pour les tests :



La Arduino Pro Mini que j'ai utilisé pour relier le module Bluetooth fonctionne en 5V, sur une fréquence de 16MHz et possède 32Kb de Ram.

Voici une partie de la datasheet de la Arduino Pro Mini détaillant les pâtes dont j'ai eu besoin :



B. Les Softwares

Arduino

Créé en 2006, ce logiciel codé en C++ permet de programmer les cartes Arduino.

Les microcontrôleurs de ces cartes peuvent être programmés pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique (contrôle des appareils domestiques, éclairage, chauffage...), le pilotage d'un robot, de l'informatique embarquée...



Le code arduino

Voir en annexe (pages 145 et 146).

Dans un premier fichier h j'ai déclaré que j'allais communiquer en Uart en utilisant les pâtes 11 pour RX et 12 pour TX.

J'ai ensuite, dans un autre fichier cpp, initialisé le Bluetooth pour qu'il fonctionne à 9600 bauds et j'ai fait une boucle qui indique par défaut que lorsque la valeur 5 stockée dans la variable nommée "c" est envoyée, alors la plateforme pilotable se maintient en équilibre.

Sinon grâce à une fonction if, si l'on reçoit d'autres valeurs telles que 1 (pour avancer), 2 (pour aller à gauche), 3 (pour reculer) et 4 (pour aller à droite) alors la valeur est lue et retournée.



Android Studio

De 2009 à 2014, avant que Android Studio n'existe, Google proposait déjà un environnement de développement nommé Eclipse qui contenait notamment les SDK d'Android Studio.

C'est le 15 mai 2013 que Android Studio est annoncé et une version d'accès anticipée fut disponible.

Le 8 décembre 2014 Android Studio passe en version bêta 1.0 et est plus stable qu'auparavant.

Ce logiciel très populaire permet alors de créer des applications codées en java et en xml.

De nombreuses applications sont aujourd'hui disponibles sur le net étant donné l'accessibilité et la facilité à prendre en main ce logiciel.



Le code android studio



Voir les annexes.

Dans un premier fichier XML j'ai éditer la partie graphique de mon application qui compte alors :

- 5 boutons avec leur dimensions (avancer, gauche, reculer, droite et bluetooth ON/OFF),
- 3 images avec leur dimensions (pour l'esthétisme)

page 178

Ensuite, dans un deuxième fichier java, j'ai codé l'application qui permet de contrôler la plateforme :

page 169

Tout d'abord j'ai dû inclure les bibliothèques qui m'ont permis d'utiliser certaines fonctions et boutons.

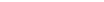
Dans une classe, j'ai par la suite indiqué l'adresse MAC ainsi que le port UUID du module Bluetooth auquel il faut se connecter lorsque l'application est lancée.

J'ai initialisé :

- Le bouton avancer,
- le bouton gauche,
- le bouton reculer,
- le bouton droite,
- le bouton bluetooth,
- et une chaîne de caractère qui stocke la valeur envoyée.

J'ai alors déclaré les boutons précédemment cités grâce à leur ID initialisés dans la partie XML en leur donnant une couleur pour l'esthétisme.





J'ai fait en sorte que lorsque l'on reste appuyé sur un bouton, une valeur est envoyée :

- 1 pour avancer,
- 2 pour aller à gauche,
- 3 pour reculer,
- 4 pour aller à droite,
- et 5 lorsque l'on relâche un bouton.

J'ai ensuite initialisé le Bluetooth.

Je demande alors si le Bluetooth est activé lorsque l'application est lancée et si l'utilisateur est d'accord pour l'activer.

Il détecte alors les objets aux alentours et récupère l'adresse de notre module.

Une fois le mot de passe "1234" (défini par défaut) entré le téléphone ou la tablette se connecte au module Bluetooth.

Ainsi, il peut communiquer à travers le port UUID défini au début du fichier java.





CONCLUSION

Pour conclure, ce projet m'a beaucoup plu.

J'ai appris énormément de par l'échange avec mes camarades mais aussi grâce à mes recherches personnelles.

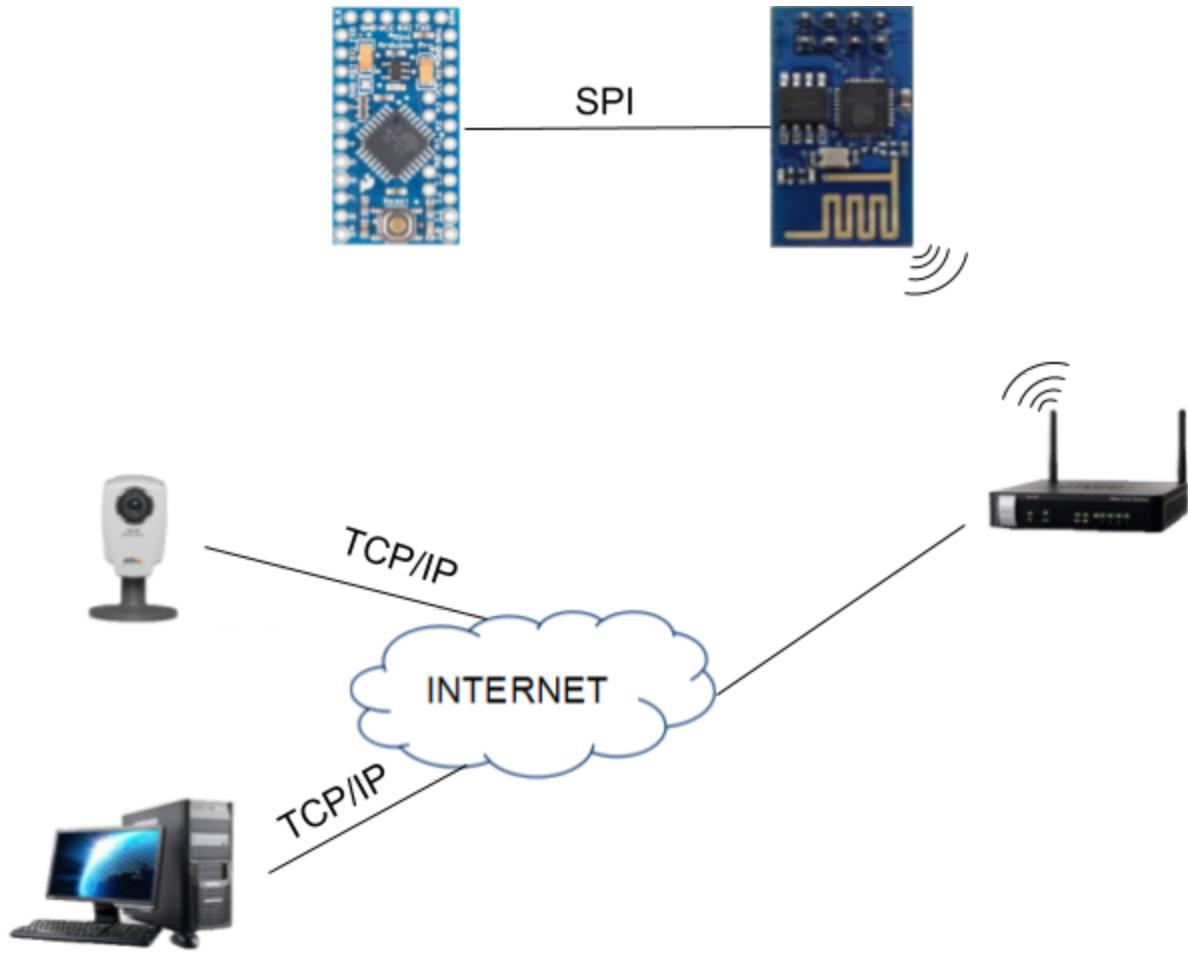
J'ai particulièrement bien aimé la partie Android Studio, le logiciel a été simple à prendre en main.

Et je me suis alors rendu compte de l'étendue des projets qui étaient possibles de créer.









II. TRAVAIL DEMANDÉ

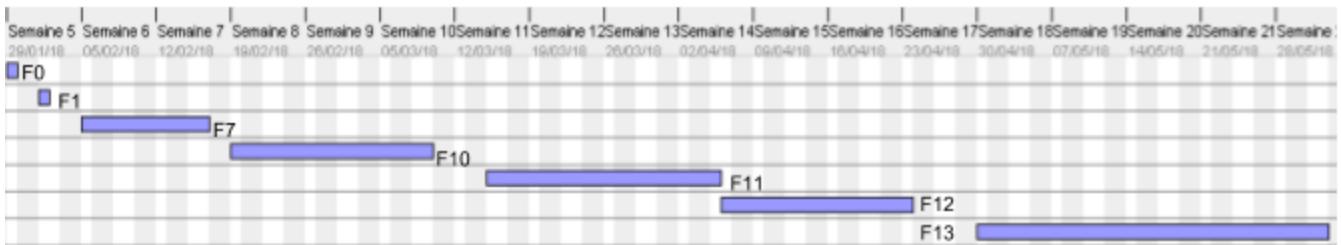
Les missions exigées concernant ma partie commande PC sont :

- Mettre en œuvre une IHM PC (applications PC)
- Mise en œuvre du module de communication Wifi
- Mise en œuvre de l'application streaming
- Réalisation des commandes et de la communication Wifi

Cette IHM permet d'établir un contrôle à distance : l'opérateur est devant son PC et peut observer une image du robot en temps réel (en streaming), tout cela en envoyant des commandes par Wifi, via un point d'accès.

Tout en essayant de suivre le planning prévisionnel suivant :

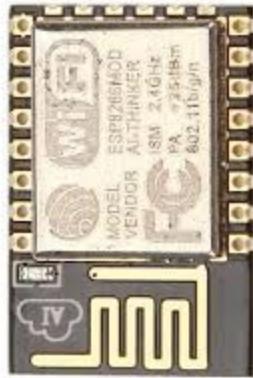
- | | | |
|---|----------|----------|
| • F0 : Prendre en main le projet | 15/03/18 | 15/03/18 |
| • F1 : Définir un protocole de dialogue entre les deux processeurs | 16/03/18 | 16/03/18 |
| • F7 : Définir un protocole de dialogue entre le processeur et l'opérateur(protocol unique pour Bluetooth et wi-fi) | 19/03/18 | 30/03/18 |
| • F10 : Développer le module de communication wi-fi coté processeur | 30/03/18 | 19/04/18 |
| • F11 : Développer l'application streaming (script de connexion et transfert d'image) | 30/04/18 | 17/05/18 |
| • F12 : Développer l'IHM PC (commande de mouvements par flèches et par souris) | 19/04/18 | 30/04/18 |
| • F13 : Développer la communication avec le module wifi | 30/04/18 | 01/06/18 |



III. MODULE DE COMMUNICATION WIFI

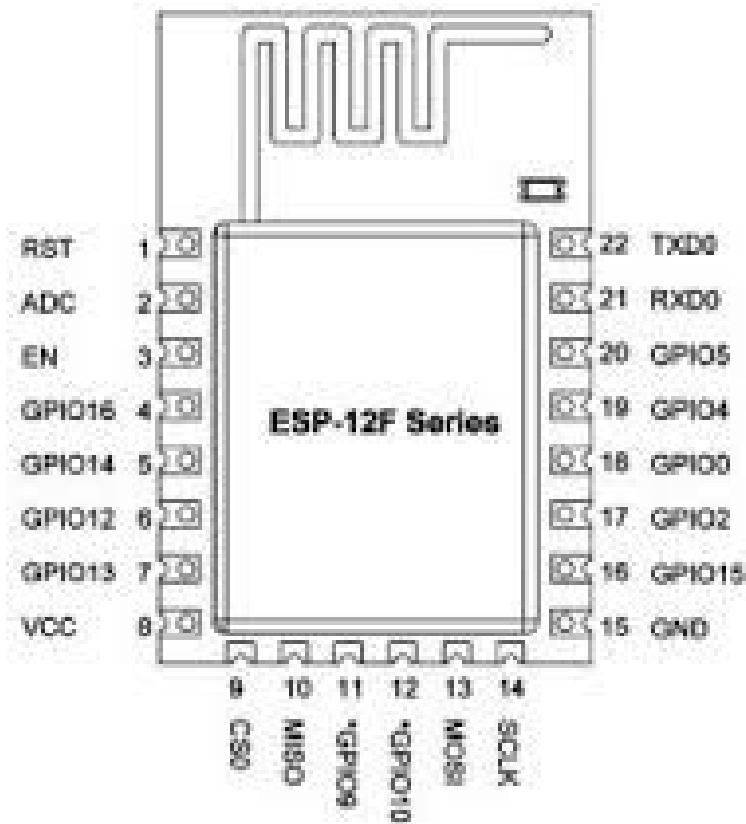
Les cartes basées sur le micro-contrôleur ESP12F sont programmables comme les cartes Arduino et peuvent communiquer en Wifi avec d'autres appareils (ordinateurs, smartphones, etc.). Il existe plusieurs modèles, l'ESP-01, l'ESP-03, l'ESP-12 ... et l'ESP-12F que j'ai utilisé :





L'ESP-8266 fonctionne à 3.3V et ne possède pas de régulateur de tension, il fallait donc bien veiller à toujours l'alimenter en 3.3V et non en 5V. Le processeur est cadencé à 80 MHz et possède 32 KB de RAM. Cette carte intègre une mémoire flash d'environ 4 MB.

Cette carte possède plusieurs entrées et sorties comme le montre le schéma ci-dessous :



Mais attention, on ne peut pas toutes les utiliser sans impacter une fonctionnalité nécessaire au bon fonctionnement de l'ensemble. Voici les entrées-sorties que j'ai employées, avec l'utilisation que j'ai pu en faire :

PATTE	UTILISATION
IO 0	Flash select : pin flottante en fonctionnement et au GND lors de la programmation
IO 15	Flash select : toujours à la masse
IO 2	Flash select : toujours au 3.3 V
3.3 V	Alimentation déjà connectée aux autres pattes 3.3V sur la carte
RST	Normalement relié au 3.3V. relié à GND pour réinitialiser la carte
GND	Masse déjà connectée aux autres pattes GND sur la carte
TX	Communication série également utilisée lors de la programmation
RX	Communication série également utilisée lors de la programmation

IV. PROGRAMMATION

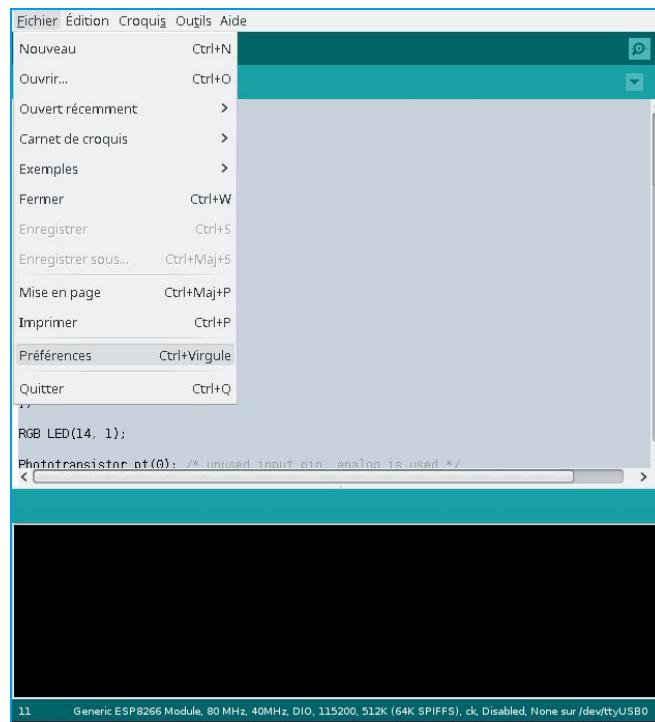


Pour mettre notre propre programme communiquant en Wifi dans l'ESP-8266, j'ai utiliser le logiciel Arduino et de quoi relier la carte à l'ordinateur.

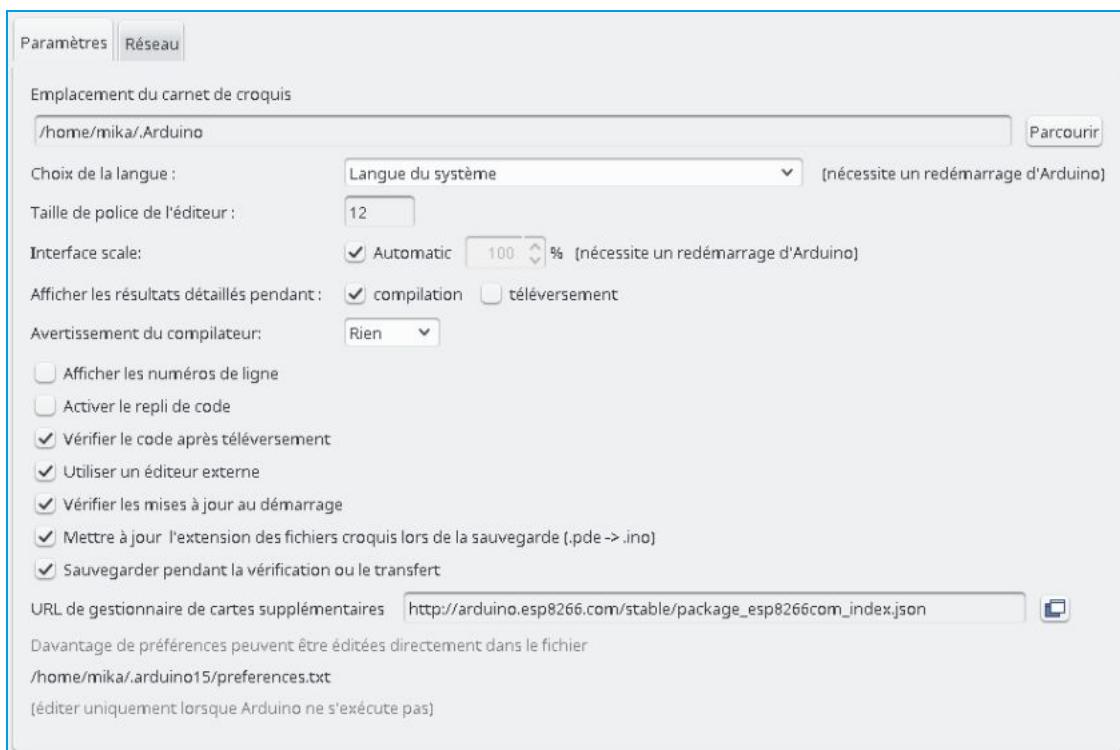
A. Logicielle : l'IDE Arduino

Il faut avoir l'IDE Arduino avec une version supérieure ou égale à la 1.6.4 que j'ai pu télécharger et installer depuis le site officiel d'Arduino (voir page 29).

Pour que cet IDE puisse gérer l'ESP-8266, il faut ajouter cette carte dans la configuration du logiciel.
Pour cela, il fallait lancer l'IDE Arduino puis dans Fichier→Préférences



Puis valider en cliquant sur "OK", puis aller dans Outils→Type de carte→Gestionnaire de cartes, dans le paragraphe concernant l'ESP-8266 il fallait cliquer sur "Installer".



L'ESP8266 en bref :



-
1. Support du protocole 802.11 ;
 2. Se connecte au PC et fonctionne comme client avec sa propre adresse IP ;
 3. Le module ESP-8266 (qui sera le serveur) comprend deux broches numériques qui peuvent être configurées en entrée ou en sortie (pour piloter des LED, des relais, etc.). D'autres versions du module comportent davantage de broches d'entrées-sorties (l'ESP-12 par exemple), mais ils se programmme tous de façon similaire ;
 4. Peut fonctionner en communiquant avec une carte Arduino ou être programmé pour fonctionner en toute autonomie ;
 5. Il existe de nombreux outils et environnements de développement (EDI) pour programmer le module.



Type Tout ▾ Filtrez votre recherche

Cartes incluses dans ce paquet:
EMoRo 2560.
[Online help](#)
[More info](#)

AMEL-Tech Boards by replaced by Arrow Boards
Cartes incluses dans ce paquet:
SmartEverything Fox.
[Online help](#)
[More info](#)

esp8266 by ESP8266 Community
Cartes incluses dans ce paquet:
Generic ESP8266 Module, Olimex MOD-WIFI-ESP8266(-DEV), NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Adafruit HUZZAH ESP8266 (ESP-12), ESPRESSO Lite 1.0, ESPRESSO Lite 2.0, SparkFun Thing, SweetPea ESP-210, WeMos D1, WeMos D1 mini, ESPino (ESP-12 Module), ESPino (WROOM-02 Module), WifInfo, ESPDuino.
[Online help](#)
[More info](#)

2.2.0 ▾ [Installer](#)

Nous allons maintenant regarder la partie électronique avant de revenir sur ce logiciel pour envoyer un programme sur l'ESP-8266.



B. Électronique

Il fallait faire deux choses pour que l'ESP-8266 soit prêt à recevoir un programme :

- Connecter l'ESP-8266 à l'ordinateur
- Connecter certaines pattes de l'ESP-8266 à une tension particulière (GND ou 3,3V)

Dans le but de connecter l'ESP-8266 à l'ordinateur, j'ai utilisé un adaptateur FTDI qui se branche en USB sur l'ordinateur et qui fournit les connexions suivantes à l'ESP-8266 :

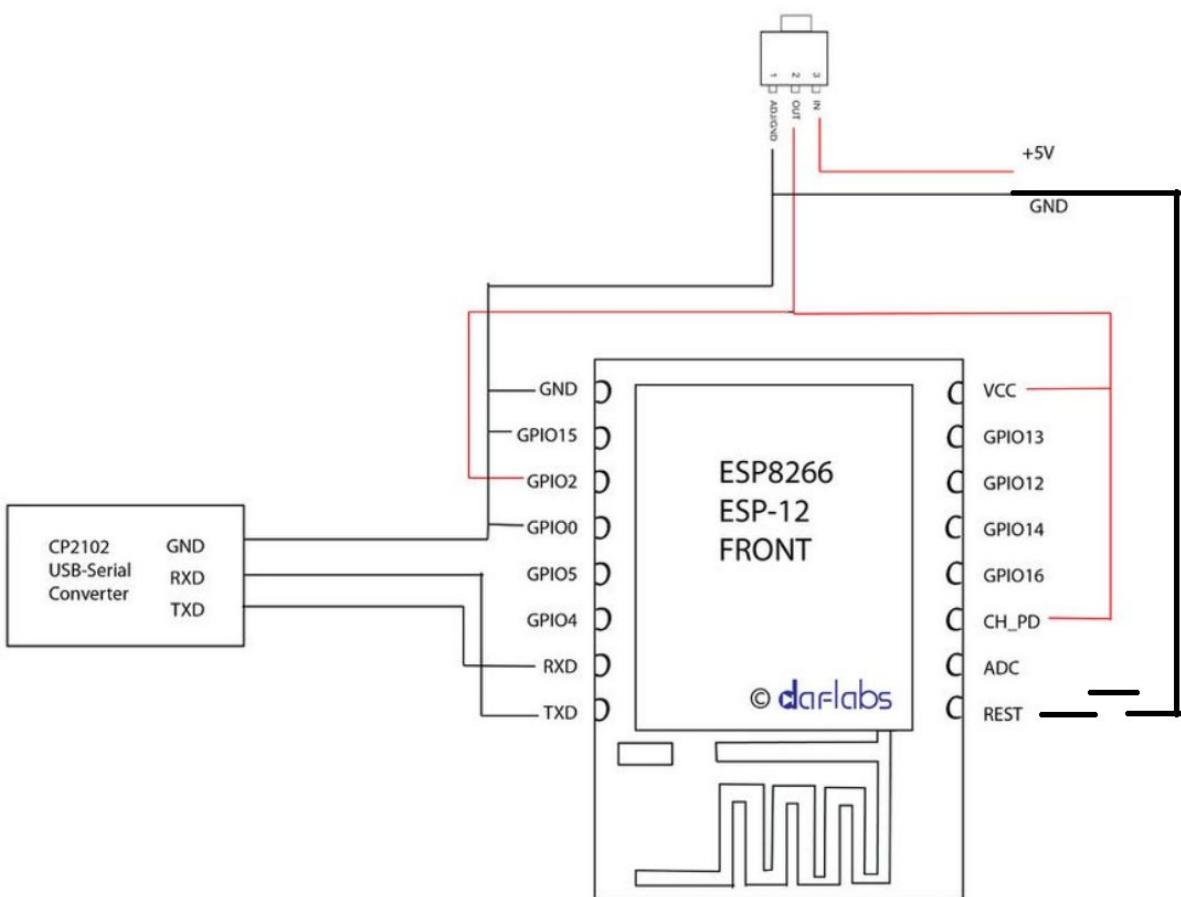
- GND : la masse
- +Vcc : la tension positive pour alimenter le module
- Rx : les données transitent de l'ESP-201 vers l'ordinateur sur cette connexion
- Tx : les données transitent de l'ordinateur vers l'ESP-201 sur cette connexion

Attention : **Pour un câble FTDI, même s'il est marqué 3,3V, +Vcc vaut 5V**, il fallait convertir cette tension pour alimenter l'ESP-201 en 3,3V.



Pour mettre l'ESP12F en mode “prêt à être programmé”, reprenant le tableau des entrées-sorties donné plus haut, j'ai connecté :

- IO2 au 3,3V
- IO15 au GND
- IO0 doit être relié au GND pour être en mode programmation. Si au contraire, nous voulions juste utiliser le module que nous avons déjà programmé, nous devons connecter IO15 au 3,3V.



Pour le programmer, j'ai donc mis un interrupteur, afin de pouvoir mettre le programme dans la carte ESP12F lorsque le pin GPIO15 était connecté au gnd et le mettre en tant que patte flottante (donc non connectée) sur la carte.

V. ENVOI ET RÉCEPTION DES COMMANDES

UDP et TCP sont deux protocoles qui permettent de communiquer à travers un réseau. Ils ont cependant différents avantages et inconvénients.

Utiliser UDP si :

Les délais qui découlent des accusés de réception sont inacceptables :

Si les options de contrôle des flux de TCP ne sont pas utiles ou gênent pour l'application.

Si on a besoin d'un bon rapport délai et nombre d'envois (pour les conférences audio ou vidéo par exemple).

Utiliser TCP si :

On veut s'assurer de recevoir les données de façon complète et sûre. Les paquets seront réémis en cas de pertes ou de données erronées.

Cependant l'envoi de nos données ne nécessite pas un accusé de réception ; de plus, si l'utilisateur insiste sur le bouton haut par exemple, les trames seront récupérées quoi qu'il arrive donc peuvent causer un risque de perte de contrôle de la plateforme. Afin d'éviter cela, mon choix s'est porté sur l'UDP.

voir annexe page 188



A. Programmation côté Wifi

Tout d'abord, j'ai créé un point d'accès Wifi afin de pouvoir connecter le PC directement au gyropode comme indiqué ci-dessous :

```
#include "wifi.h"

extern String wifi_connection()
{
    IPAddress apIP(192, 168, 0, 177);

    WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
    WiFi.softAP(ssid, password);

    UDPTestServer.begin(UDPPort); // run UDP port

    return "connecté";
}
```

Premièrement on définit une adresse IP statique afin de pouvoir se connecter à elle à l'aide de l'objet apIP de type IPAddress. Pour configurer ce nouveau réseau, j'ai utilisé l'objet softAPConfig de type Wifi, j'utilise softAP afin de configurer l'identifiant et le mot de passe que je déclaré de cette façon :

```
#include <ESP8266WebServer.h>

static const char* ssid      = "SosoMobile";
static const char* password = "mainbot123";
```

La library #include <ESP8266WebServer.h> contient les objet Wifi et IPAddress.

Pour définir l'identifiant et le mot de passe, on utilise un static const, *static* qui permet que la variable soit propre aux fichiers sources, et *const* pour que les variables soient constantes et non modifiables.

voir annexe page 184



B. Programmation côté processeur

```
void loop()
{
    //Serial.println(get_wifi_message());
    digitalWrite(CS, LOW);
    | SPI.transfer(get_wifi_messageChar());
    delayMicroseconds(10);
    digitalWrite(CS, HIGH);
}
```

Comme on peut le voir, on a plusieurs objets dont `digitalWrite(CS, LOW)` qui, lorsqu'il est à l'état bas, transfère le message à l'autre processeur puis attend 10 microsecondes, afin de laisser le temps à la trame spi d'arriver, et remet `digitalWrite` à l'état haut.



BOUTON DE L'IHM

A. Comment installer ?

Pour utiliser Qt, nous allons utiliser la version LGPL qui est gratuite.

Il faut ensuite choisir entre :

- Qt SDK : la bibliothèque Qt + un ensemble d'outils pour développer avec Qt, incluant un IDE spécial appelé Qt Creator.
- Qt Framework : contient uniquement la bibliothèque Qt.

Qt SDK possédant un plus grand nombre d'outils, nous allons donc l'utiliser.

On a choisi soit "Qt pour Windows: C++", "Qt pour Linux/X11: C++" ou "Qt pour Mac: C++" en fonction du système d'exploitation utilisé.

Sous Linux, et notamment sous Debian ou Ubuntu, il est plus facile d'installer directement le paquet qtcreator avec la commande aptget install qtcreator. La version est légèrement plus ancienne mais l'installation est centralisée et plus facile à gérer.



B. Logicielle Qt Creator

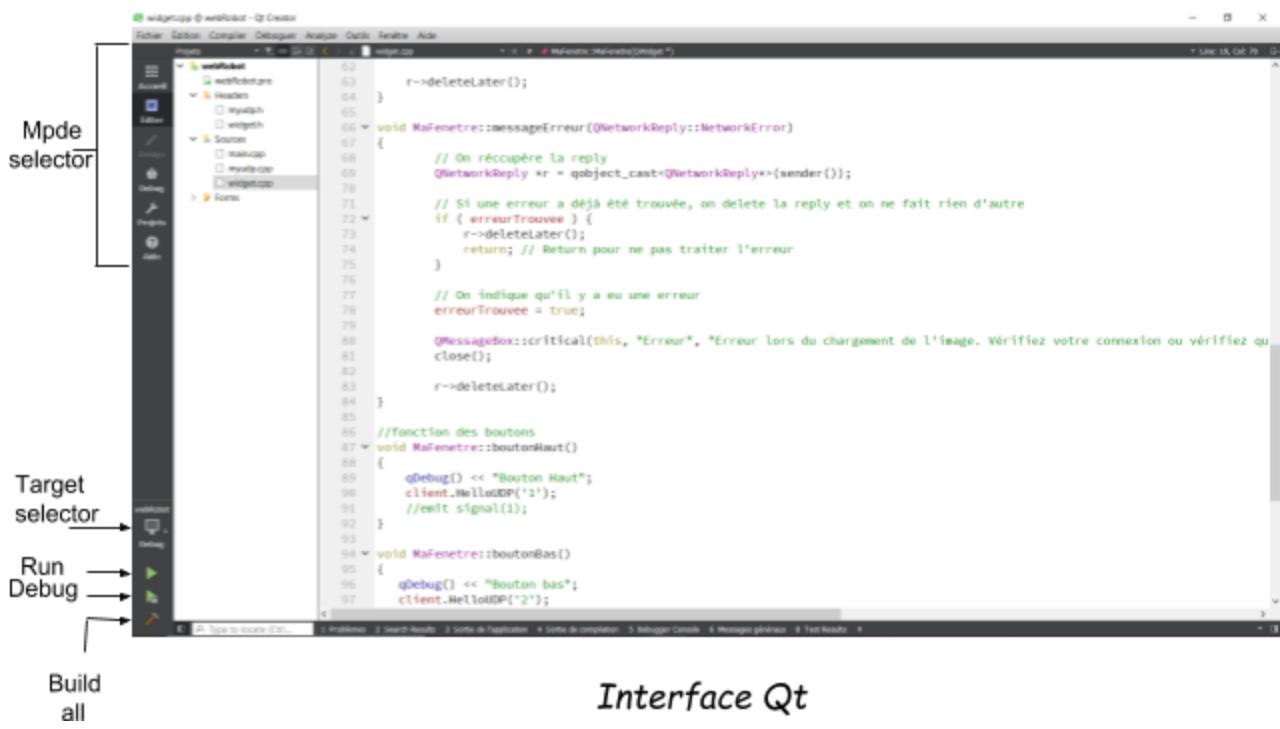
Qt Creator est un environnement multi-plateforme de développement intégré (IDE) adapté aux besoins des développeurs de Qt. Qt Creator fonctionne sur Windows, Linux/X11 et systèmes Mac OS X d'exploitation de bureau, et permet aux développeurs de créer des applications pour le bureau et plateformes mobiles.

Bien qu'il soit possible de développer en C++ avec Qt en utilisant notre IDE (comme Code::Blocks) il est plus intéressant d'utiliser l'IDE Qt Creator que l'on vient d'installer. Il est particulièrement optimisé pour développer avec Qt. En effet, c'est un programme tout-en-un qui comprend entre autres :

- Un IDE pour développer en C++, optimisé pour compiler des projets utilisant Qt (pas de configuration fastidieuse)
- Un éditeur de fenêtres, qui permet de dessiner facilement le contenu de ses interfaces à la souris.
- Une documentation pour tout savoir sur Qt.

Voici ce à quoi ressemble Qt Creator lorsqu'on le lance pour la première fois :





Qt est constituée d'un ensemble de bibliothèques, appelées «modules». On peut y trouver entre autres ces fonctionnalités :

- **Module GUI** : c'est toute la partie création de fenêtres. Nous nous concentrerons surtout sur le module GUI.
- **Module OpenGL** : Qt peut ouvrir une fenêtre contenant de la 3D gérée par OpenGL.
- **Module de dessin** : pour dessiner dans une fenêtre (en 2D), le module de dessin est très complet !
- **Module réseau** : Qt fournit une batterie d'outils pour accéder au réseau, que ce soit pour créer un logiciel de Chat, un client FTP, un client BitTorrent, un lecteur de flux RSS...
- **Module de script** : Qt prend en charge le Javascript (ou ECMAScript), que l'on peut réutiliser dans ses applications pour ajouter des fonctionnalités, par exemple sous forme de plugins.
- **Module XML** : si l'on connaît le XML, c'est un moyen très pratique d'échanger des données à partir de fichiers structurés à l'aide de balises, comme le XHTML.
- **Module SQL** : permet d'accéder aux bases de données (MySQL, Oracle, PostgreSQL...).

C. Les Boutons

Pour créer un bouton, nous nous servons du code suivant :

```
BoutonHaut = new QPushButton("bouton haut", this);
```

le *this* sert à voir apparaître le bouton dans la fenêtre, dans le cas où le widget concerné par une telle déclaration ne serait pas englobé par un autre widget qui aurait reçu cela (comme un layout par exemple). Afin de pouvoir placer le bouton, on se sert de

```
move(abscisse, ordonnee);
```

et pour le placer et le dimensionner, on se sert de

```
setGeometry(abscisse, ordonnee, largeur, hauteur);
```

et ce de la manière suivante :

```
BoutonHaut->setGeometry(999, 580, 80, 70);
```

comme on peut le voir on remplace “abscisse” “ordonnée” “largeur” et “hauteur” par les valeurs adéquates. De plus on a la possibilité d’ajouter différents signaux pour différentes utilisations.

Les signaux associés aux boutons

- `clicked()` : c'est le plus courant et le plus utilisé des signaux des QPushButtons ;
- `pressed()` : signal déclenché quand le bouton est enfoncé ;
- `released()` : signal déclenché quand le bouton est relâché.



Pour réaliser l'IHM j'ai dû développer mon constructeur où j'ai mis mes différents boutons comme montré ci-dessous :

```
MaFenetre::MaFenetre(QWidget *parent)
: QWidget(parent), m(new QNetworkAccessManager())
{
    m->setParent(this);
    erreurTrouvee = false;

    label=new QLabel(this);
    label->move(40, 50);

    timer = new QTimer(this);
    timer->start(40);
    connect(timer, SIGNAL(timeout()), this, SLOT(changement()));

    BoutonHaut = new QPushButton("bouton haut", this);
    BoutonHaut->setGeometry(999, 580, 80, 70);
    connect(BoutonHaut, SIGNAL(clicked()), this, SLOT(boutonHaut()));

    BoutonBas = new QPushButton("bouton bas", this);
    BoutonBas->setGeometry(999, 720, 80, 70);
    connect(BoutonBas, SIGNAL(clicked()), this, SLOT(boutonBas()));

    BoutonDroit = new QPushButton("bouton droit", this);
    BoutonDroit->setGeometry(1880, 650, 80, 70);
    connect(BoutonDroit, SIGNAL(clicked()), this, SLOT(boutonDroit()));

    BoutonGauche = new QPushButton("bouton gauche", this);
    BoutonGauche->setGeometry(920, 650, 80, 70);
    connect(BoutonGauche, SIGNAL(clicked()), this, SLOT(boutonGauche()));

    BoutonArret = new QPushButton("Arret", this);
    BoutonArret->setGeometry(999, 650, 80, 70);
    connect(BoutonArret, SIGNAL(clicked()), this, SLOT(boutonArret()));
}
```

voir annexe page 185



L'APPLICATION STREAMING

A. Matériel

Afin de réaliser le streaming de la plateforme, le lycée met à disposition une caméra ip de la marque Axis, dont le modèle est le suivant :



AXIS 205

B. Mise en place de la caméra

La caméra AXIS 205 est conçue pour une installation dans un réseau Ethernet. Une adresse IP doit donc être attribuée à la caméra réseau, soit automatiquement (DHCP par ex.), soit manuellement mais nous verrons cela ultérieurement.

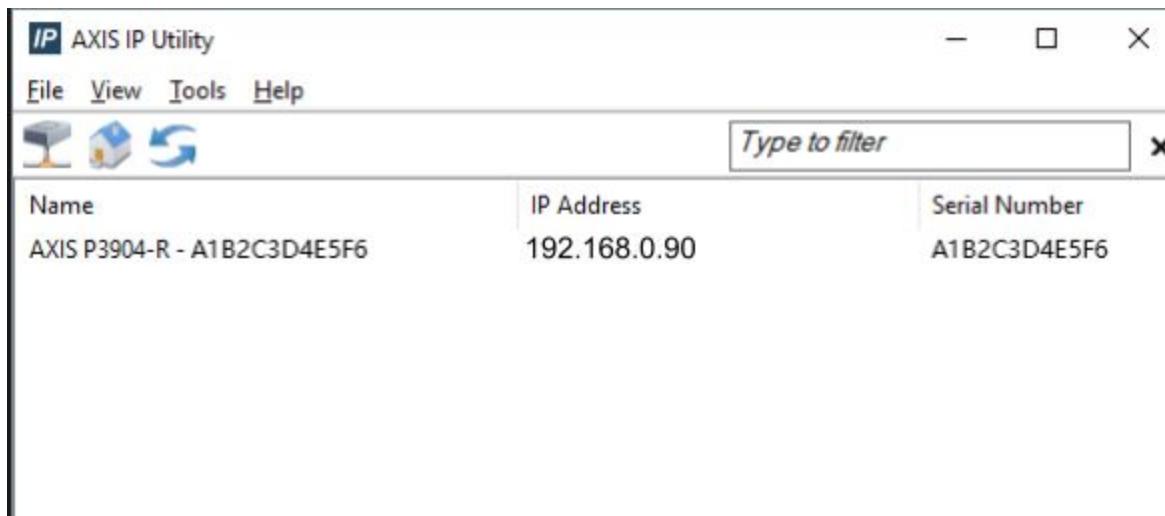
Deux étapes suffisent afin d'installer le matériel :

1. Connecter la caméra AXIS 205 au réseau avec un câble de réseau RJ-45 standard.
2. Connecter l'alimentation à la caméra.

C. Création d'une connexion

Pour ce faire, il faut suivre les étapes suivantes :

- Trouver une adresse IP inutilisée pour la caméra AXIS 205 à l'aide du logiciel AXIS IP Utility.
- Choisir une adresse IP Caméra fixe ; pour notre application, on a comme IP Caméra 172.168.0.90 mais avant tout, saisir le numéro de série de la caméra indiqué derrière l'appareil (A1B2C3D4E5F6).



- La caméra doit se connecter automatiquement au réseau local.
- Envoyer un Ping à l'adresse IP de la caméra via le terminal pour tester la connectivité.

D. Les bonnes librairies

On distingue trois sortes de librairies à inclure pour le fonctionnement du programme :

Les includes pour le langage C++

```
#include <iostream>  
  
#include <QString>
```

Les includes pour Qt

```
#include<QApplication>  
  
#include<QNetwork>  
  
#include<QNetworkReply>  
  
#include<QNetworkRequest>  
  
#include <QWidget>  
  
#include <QLabel>  
  
#include <MessageBox>  
  
#include <QTimer>
```

Les includes pour L'IHM

```
#include <QUrl>  
  
#include <Qlabel>  
  
#include <QPushButton>  
  
#include<QByteArray>
```



L'IHM PC

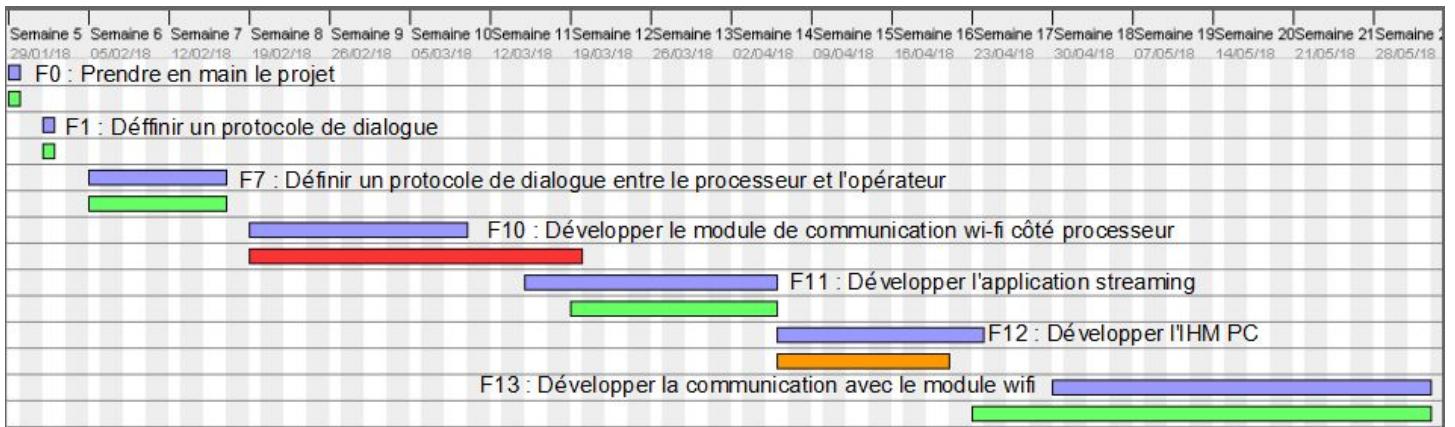
Pour clore ma partie, voici le résultat final de l'IHM, après plusieurs mois de travail :



voir annexe page 187

DÉROULEMENT DU PROJET

Comme nous l'avons vu précédemment, j'ai essayé de suivre un planning prévisionnel, cependant le déroulement du projet ne m'a pas permis de le suivre à la lettre. Par manque de connaissances dans certains domaines ou par surestimation de temps dans d'autres, certaines tâches ont pris plus ou moins de temps :



Comme vous pouvez le constater, développer le module Wifi faisait partie des tâches qui n'ont pas pu être réalisées dans le temps imparti. Les raisons sont :

- Le manque de connaissances sur le langage Arduino
- Un temps d'apprentissage trop long

De plus, le module Wifi (ESP 12 F) était très complexe à câbler.

CONCLUSION

Les missions qui m'ont été confiées, c'est-à-dire le développement de l'IHM, ont été une très grande expérience pour moi. Ce projet m'a donné une nouvelle vision du monde du travail et de ses attentes.

En effet, j'ai appris à réfléchir sur un projet à moyen terme qui m'a permis de développer mon esprit de décision en me confrontant aux problèmes rencontrés tout au long du développement. Bien que ces problèmes soient souvent restés mineurs, il fallait trouver rapidement des solutions.

Par ce projet, j'ai pu mettre à contribution les connaissances acquises au cours de ma formation, mais surtout il m'a permis d'acquérir de nouvelles connaissances dans le domaine de la gestion de projet et de code.

Je remercie par ailleurs tous les acteurs de ce projet qui se sont investis pour le rendre fonctionnel.



ANNEXE :

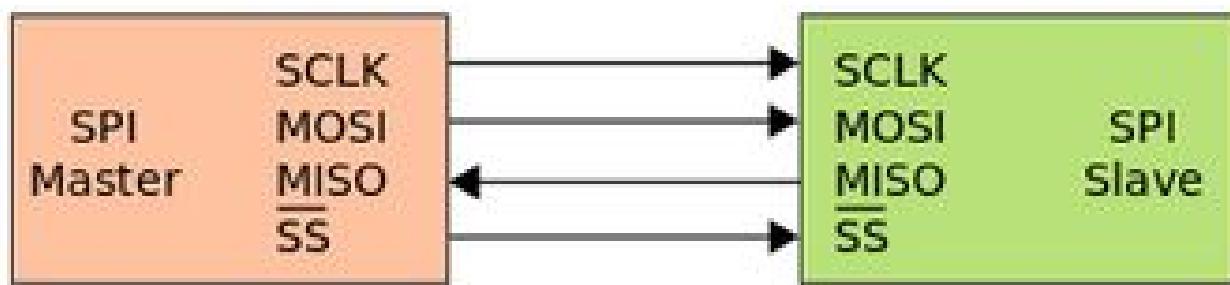
Les protocoles de communications

Le SPI

La liaison SPI ou Serial Peripheral Interface, est une liaison synchrone en full-duplex (où les composants peuvent communiquer en simultané) avec un débit plus important que le i2c (jusqu'à 20 MBit/s), cette liaison fut inventée par Motorola en 1980.

Cette liaison fonctionne sur un modèle de maître-esclave : le maître contrôle la communication en sélectionnant l'esclave (il ne peut y avoir qu'un maître mais un ou plusieurs esclaves) à qui il s'adresse en mettant le câble CS ou SS au niveau BAS (0) puis il le remet au niveau HAUT (1) lorsqu'il n'a plus besoin de communiquer avec le composant.

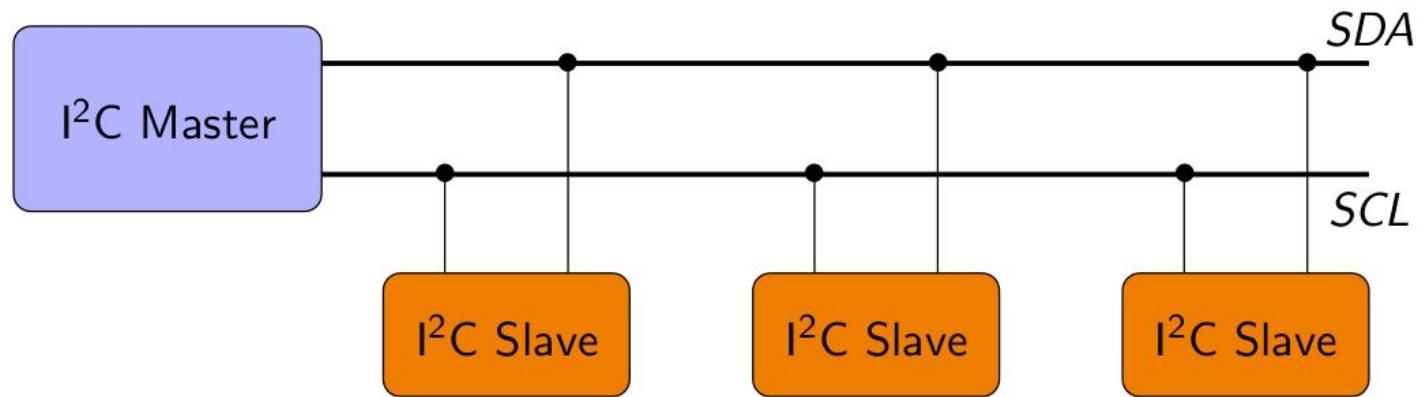
- SCLK : fils d'horloge (généré par le maître) pour la synchronisation.
- MOSI : fils de communication maître vers l'esclave (master output, slave input)
- MISO : fils de communication de l'esclave vers le maître (master input, slave output)
- SS : fils de sélection d'esclave pour le maître (slave select)



Le I2C

La liaison I2C est un bus synchrone de communication half-duplex (permet de dialoguer dans les deux sens mais un composant après l'autre) basé sur le modèle maître-esclave (il ne peut y avoir qu'un maître mais un ou plusieurs esclaves). Cette liaison est basée sur deux fils SDA (Serial Data Line) et SCL (Serial Clock Line), cette liaison simple de réalisation en électronique ne peut excéder une vitesse de 5 MBit/s.

Exemple de liaison I2C :

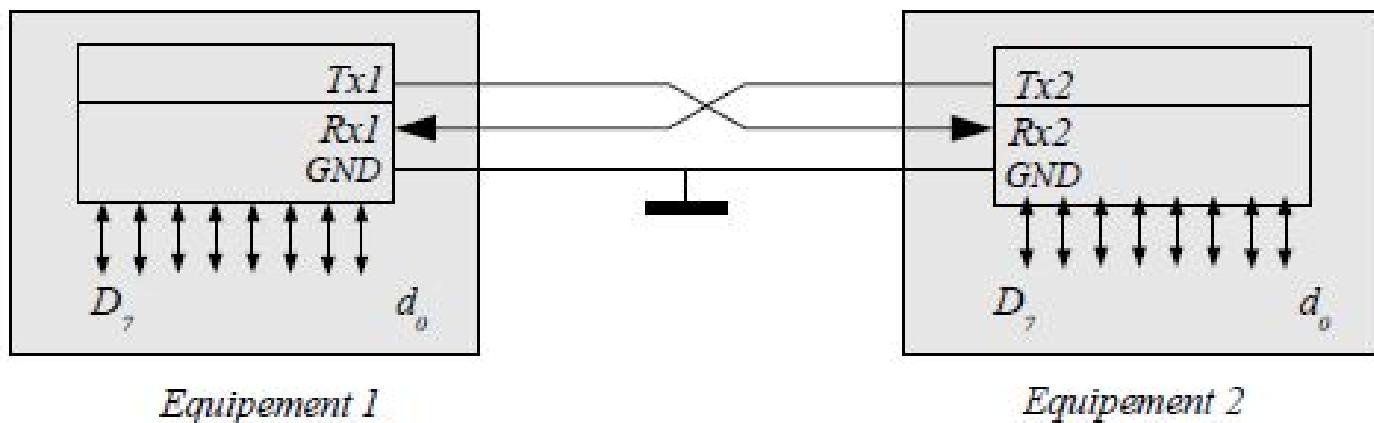


Le RS232

Dans notre cahier des charges, il est mentionné RS232 et nous avons donc gardé la même syntaxe bien que le RS232 comme l'UART soit une liaison série mais de norme RS232, c'est à dire qu'il a des niveaux logiques imposés de -3v à 15v ou -15v à 3v. Dans notre projet nous sommes en UART car les niveaux logiques sont de 0v à 5v ce qui est plus approprié à notre microcontrôleurs.

Les liaisons séries reposent sur deux fils de communication, le RX (Received Data) et le TX (Transmitted Data). Dans notre cas en UART nous avons une communication asynchrone en full-duplex (dialogue dans les deux sens et en simultané).

Exemple de liaison UART :



Les codes sources

PROGRAMME GESTION EQUILIBRE :

```
*****  
*  
*      BBBB B      TTTTTTTT      SSSSS      SSSSS      NN      N      II      RRRR R      *  
*      BB   B      TT      SS      SS      N N N      RR   RR      *  
*      BBBB B      TT      SSS      SSS      N N N      II      RRRR R      *  
*      BB   B      TT      SS      SS      N N N      II      RR   RR      *  
*      BBBB B      TT      SSSS      SSSS      n   NN      II      RR   RR      *  
*  
*****
```

***** PROGRAMME main.ino *****

```
#include "mpu6050.h"  
  
#include "moteur.h"  
  
#include "utilitaire.h"  
  
#include "monPid.h"  
  
#include "encoder.h"  
  
#include "serialRX.h"  
  
  
#define margeSetPoint 5 // ecart echantillon filtre médián  
  
float majSetpoint(float pointActuelle,float y,char tillon, float origin);  
  
void runMotors(double PIDOutput);
```



```

void setup()
{
    double PIDOutput=0;
    led_init();
    init_moteur();
    init_MPU();
    UART_Init(115200);
    init_encodeur();

    float origin, setPoint = 116.35;          //getY();      // set point dans la position de demarage du robot
    origin = setPoint;

    while(1)
    {
        loop_MPU(); // run loop sensor MPU /!\ OBLIGATOIRE a chaque tour de boucle
        float New_Y = getX();           // recup de la position réel (thx filtre de Kalman) en Y
        New_Y = mapping( New_Y, 37.7, 113.7, 0, 180);

        switch(get_buffer())
        {
            case '1':
                setPoint = f_avant(origin);
                led_power(true);
        }
    }
}

```



```
break;

case '2':

setPoint = f_gauche(origin);

led_power(true);

break;

case '3':

setPoint = f_arriere(origin);

led_power(true);

break;

case '4':

setPoint = f_droit(origin);

led_power(true);

break;

case '5':

setPoint = f_arret(origin);

led_power(false);

break;

default:

setPoint = f_arret(origin);

led_power(false);

break;

}
```



```

getPid( setPoint, New_Y, PIDOutput, 0); // mpu, point objectif, point actuelle, sortie, periode d
// echantillonage ms

runMotors(PIDOutput);

}

}

void runMotors(double PIDOutput)

{

if( PIDOutput < -88 ) // intervalle stop

{

change_vitesse(false, PIDOutput * -1); // 1 = D, 0 = G

change_vitesse(true, PIDOutput * -1 );

arrière();

}

else if( PIDOutput > 88 )

{

change_vitesse(true, PIDOutput ); // 1 = D, 0 = G

change_vitesse(false, PIDOutput );

avant();

}

else

{

power_off();

}

}

```



```

float majSetpoint(float pointActuelle,float y,char tillon, float origin)

{
    static double timer=millis();

    static float echantillons[12]={0};

    static int i=0;

        if( millis() - timer > tillon ) // 15ms           10      50      120

    {

        timer = millis();

        echantillons[i] = y; // /\! WARNING ! is dangerous

        i++;

        if(i>12)

        {

            croissant( echantillons ); // on trie nos echantillon

            if( ( echantillons[6] < origin + margeSetPoint ) && ( echantillons[6] > origin - margeSetPoint ) )

            {

                pointActuelle = echantillons[6]; // on recup la val du milieu

            }

            echantillons[12]={0}; // on nettoie nos echantillons

            i=0;

            return pointActuelle; // on return le nv Setpoint (point d'équilibre)

        }

    }

    return pointActuelle; // on a pas finis l'échantillonage on return le setpoin de base
}

```



```
}
```

```
***** I2C.h *****
```

```
/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
```

```
This software may be distributed and modified under the terms of the GNU  
General Public License version 2 (GPL2) as published by the Free Software  
Foundation and appearing in the file GPL2.TXT included in the packaging of  
this file. Please note that GPL2 Section 2[b] requires that all works based  
on this software must also be made publicly available under the terms of  
the GPL2 ("Copyleft").
```

Contact information

Kristian Lauszus, TKJ Electronics

Web : <http://www.tkjelectronics.com>

e-mail : kristianl@tkjelectronics.com

```
*/
```

```
#ifndef _I2C_H
```

```
#define _I2C_H
```



```
#include <arduino.h>
#include <Wire.h>

static const uint8_t IMUAddress = 0x68; // ADO is logic low on the PCB
static const uint16_t I2C_TIMEOUT = 1000; // Used to check for errors in I2C communication

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop);
uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop);
uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes);

#endif
```

***** I2C.cpp *****

```
#include "I2C.h"

uint8_t i2cWrite(uint8_t registerAddress, uint8_t data, bool sendStop)
{
    return i2cWrite(registerAddress, &data, 1, sendStop); // Returns 0 on success
}

uint8_t i2cWrite(uint8_t registerAddress, uint8_t *data, uint8_t length, bool sendStop) {
```



```

Wire.beginTransmission(IMUAddress);

Wire.write(registerAddress);

Wire.write(data, length);

uint8_t rcode = Wire.endTransmission(sendStop); // Returns 0 on success

if (rcode) {

// Serial.print(F("i2cWrite failed: "));

// Serial.println(rcode);

}

return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission

}

uint8_t i2cRead(uint8_t registerAddress, uint8_t *data, uint8_t nbytes) {

uint32_t timeOutTimer;

Wire.beginTransmission(IMUAddress);

Wire.write(registerAddress);

uint8_t rcode = Wire.endTransmission(false); // Don't release the bus

if (rcode) {

// Serial.print(F("i2cRead failed: "));

// Serial.println(rcode);

return rcode; // See: http://arduino.cc/en/Reference/WireEndTransmission

}

Wire.requestFrom(IMUAddress, nbytes, (uint8_t)true); // Send a repeated start and then release the bus after
reading

for (uint8_t i = 0; i < nbytes; i++) {

```





```
if (Wire.available())  
  
    data[i] = Wire.read();  
  
else {  
  
    timeOutTimer = micros();  
  
    while (((micros() - timeOutTimer) < I2C_TIMEOUT) && !Wire.available());  
  
    if (Wire.available())  
  
        data[i] = Wire.read();  
  
    else {  
  
        // Serial.println(F("i2cRead timeout"));  
  
        return 5; // This error value is not already taken by endTransmission  
  
    }  
  
}  
  
}  
  
return 0; // Success  
}
```

***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****

Kalman.h

***** ***** ***** ***** ***** ***** ***** ***** ***** ***** ***** *****

/* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.



This software may be distributed and modified under the terms of the GNU General Public License version 2 (GPL2) as published by the Free Software Foundation and appearing in the file GPL2.TXT included in the packaging of this file. Please note that GPL2 Section 2[b] requires that all works based on this software must also be made publicly available under the terms of the GPL2 ("Copyleft").

Contact information

Kristian Lauszus, TKJ Electronics

Web : <http://www.tkjelectronics.com>

e-mail : kristianl@tkjelectronics.com

*/

```
#ifndef _Kalman_h
```

```
#define _Kalman_h
```



```

class Kalman

{
public:

    Kalman();

    // The angle should be in degrees and the rate should be in degrees per second and the delta time in
    seconds

    double getAngle(double newAngle, double newRate, double dt);

    void setAngle(double newAngle); // Used to set angle, this should be set as the starting angle

    double getRate(); // Return the unbiased rate

    /* These are used to tune the Kalman filter */

    void setQangle(double newQ_angle);

    void setQbias(double newQ_bias);

    void setRmeasure(double newR_measure);

    double getQangle();

    double getQbias();

    double getRmeasure();

private:

    /* Kalman filter variables */

    double Q_angle; // Process noise variance for the accelerometer

    double Q_bias; // Process noise variance for the gyro bias

```



```

double R_measure; // Measurement noise variance - this is actually the variance of the measurement
noise

double angle; // The angle calculated by the Kalman filter - part of the 2x1 state vector
double bias; // The gyro bias calculated by the Kalman filter - part of the 2x1 state vector
double rate; // Unbiased rate calculated from the rate and the calculated bias - you have to call getAngle
to update the rate

double P[2][2]; // Error covariance matrix - This is a 2x2 matrix
double K[2]; // Kalman gain - This is a 2x1 vector
double y; // Angle difference
double S; // Estimate error
};

#endif

//*****************************************************************************
Kalman.cpp ***** */

```



```

#include "Kalman.h"

Kalman::Kalman()
{
    /* We will set the variables like so, these can also be tuned by the user */

    Q_angle = 0.001;
    Q_bias = 0.003;
    R_measure = 0.03;
    angle = 0; // Reset the angle
    bias = 0; // Reset bias
}

```

$P[0][0] = 0;$ // Since we assume that the bias is 0 and we know the starting angle (use setAngle), the error covariance matrix is set like so - see:

http://en.wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical

```

P[0][1] = 0;
P[1][0] = 0;
P[1][1] = 0;
}

double Kalman::getQangle()
{
    return Q_angle;
}

double Kalman::getQbias()
{
}

```



```
return Q_bias;

}

double Kalman::getRmeasure()

{

    return R_measure;

}

void Kalman::setRmeasure(double newR_measure)

{

    R_measure = newR_measure;

}

void Kalman::setQbias(double newQ_bias)

{

    Q_bias = newQ_bias;

}

void Kalman::setQangle(double newQ_angle)

{

    Q_angle = newQ_angle;

}

double Kalman::getRate()

{

    return rate;

}

void Kalman::setAngle(double newAngle)
```



```

{
    angle = newAngle;
}

double Kalman::getAngle(double newAngle, double newRate, double dt)
{
    // KasBot V2 - Kalman filter module - http://www.x-firm.com/?page_id=145
    // Modified by Kristian Lauszus
    // See my blog post for more information:
    // http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it

    // Discrete Kalman filter time update equations - Time Update ("Predict")
    // Update xhat - Project the state ahead
    /* Step 1 */

    rate = newRate - bias;
    angle += dt * rate;

    // Update estimation error covariance - Project the error covariance ahead
    /* Step 2 */

    P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
    P[0][1] -= dt * P[1][1];
    P[1][0] -= dt * P[1][1];
    P[1][1] += Q_bias * dt;
}

```



```

// Discrete Kalman filter measurement update equations - Measurement Update ("Correct")

// Calculate Kalman gain - Compute the Kalman gain

/* Step 4 */

S = P[0][0] + R_measure;

/* Step 5 */

K[0] = P[0][0] / S;

K[1] = P[1][0] / S;

// Calculate angle and bias - Update estimate with measurement zk (newAngle)

/* Step 3 */

y = newAngle - angle;

/* Step 6 */

angle += K[0] * y;

bias += K[1] * y;

// Calculate estimation error covariance - Update the error covariance

/* Step 7 */

P[0][0] -= K[0] * P[0][0];

P[0][1] -= K[0] * P[0][1];

P[1][0] -= K[1] * P[0][0];

P[1][1] -= K[1] * P[0][1];

return angle;

```



```
}
```

```
***** encoder.h *****
```

```
#ifndef ENCODER_H
```

```
#define ENCODER_H
```

```
#include <arduino.h>
```

```
#include "moteur.h"
```

```
#define interruptPinG 2
```

```
#define interruptPinD 3
```

```
static int vitessD=0;
```

```
static int vitessG=0;
```

```
void init_encodeur();
```

```
static void encoderD();
```

```
static void encoderG();
```

```
void getVitess(int &moteurD, int &moteurG);
```

```
int comparaisonMoteur();
```

```
#endif
```

```
***** encoder.cpp *****
```

```
#include "encoder.h"
```



```

void init_encodeur()
{
pinMode(interruptPinG, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPinG), encoderG, HIGH);
pinMode(interruptPinD, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPinD), encoderD, HIGH);
sei();
}

static void encoderD()
{
static volatile byte state = LOW;
static int compteurD = 0;
static byte passe = state;
static double timer = micros();

state =! passe;

if(state != passe)
{
passe = state;
compteurD++;
}
}

```



```

if( micros() - timer > 20000 ) // 20ms

{
    //Serial.println("tour en 20ms D: ");

    //Serial.println(compteurD);

    timer = micros();

    vitessD = compteurD;

    compteurD = 0;
}

}

static void encoderG() // /\ IMPOSSIBLE TO DISPLAY VALUE TX is working.....

{
    static volatile byte state = LOW;

    static int compteurG = 0;

    static byte passe = state;

    static double timer = micros();

    state =! passe;

    if(state != passe)

    {
        passe = state;

        compteurG++;

    }
}

```



```
if( micros() - timer > 20000 ) // 20ms  
{  
    timer = micros();  
    vitessG = compteurG;  
    compteurG = 0;  
}  
}
```

```
int comparaisonMoteur()  
{  
/*  
Serial.println("VD : | VG : | dif: ");  
Serial.print(vitessD);  
Serial.print(" | ");  
Serial.print(vitessG);  
Serial.print(" | ");  
Serial.println(vitessD - vitessG );  
*/  
return (vitessD - vitessG );  
}
```

```
void getVitess(int &moteurD, int &moteurG)
```



```
{  
moteurD = vitessD;  
moteurG = vitessG;  
}
```

```
***** pid.h *****
```

```
#ifndef _monpid_H  
  
#define _monpid_H  
  
#include <arduino.h>  
  
#include "utilitaire.h"
```

```
#define Ki 3.5 // Ki pour avoir une réponse exacte en peu de temps (3)  
#define Kd 940 // Kd qui permet de rendre le système plus stable (888)  
#define Kp 13 // Kp afin d'améliorer le temps de réponse du système (12)
```

```
#define MAX 240  
  
#define MIN 73
```

```
*
```

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

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

*/



void getPid(int consigne, float mesure, double &commande, byte tillon); // PID==true MPU || pid == false PWM

//void getPidCorrection(int consigne, int mesure, double &commande, byte tillon); // PID full PWM

#endif

```

***** pid.cpp *****

```

#include "monPid.h"

void getPid(int consigne, float mesure, double &commande, byte tillon) // true == MPU, FALSE == PWM

{
    static unsigned long startTime = 0 ; // millis function
    static int somme_erreurs=0, erreur precedente=0; // les variables constante
    int erreur=0, variation_erreur=0;
    double TEMPOcommande=0;
    erreur = consigne - mesure;
    somme_erreurs += erreur;

    if( (millis() - startTime) > tillon ) // fin d echantillonnage

```



```

[

startTime = millis();

variation_erreur = erreur - erreur_precedente;

TEMPOcommande = Kp * erreur + Ki * somme_erreurs + Kd * variation_erreur;

/* Serial.print("pid MPU : ");

Serial.println(TEMPOcommande);*/



if(TEMPOcommande < 0)

{

    TEMPOcommande = mapping(TEMPOcommande, -62.10, 0, MAX * -1, MIN* -1); // adaptation du coef de
pid en 65-255 PWM

    if(TEMPOcommande < (MAX * -1) ) TEMPOcommande = MAX * -1; // vitesse MAX EN NEGATIF -255
    if(TEMPOcommande > (MIN* -1)) TEMPOcommande = MIN * -1; // // MIN ///
}

else

{

    TEMPOcommande = mapping(TEMPOcommande, 0, 95.4, MIN, MAX); // adaptation du coef de pid en
65-255 PWM

    if(TEMPOcommande > MAX) TEMPOcommande = MAX; // vitesse max
    if(TEMPOcommande < MIN) TEMPOcommande = MIN; // vitesse min
}

//Serial.print("pid MPU : ");

```



```

//Serial.println(TEMPOcommande);

commande = TEMPOcommande;

somme_erreurs = 0;

erreur precedente = erreur;

return ;

}

erreur precedente = erreur ;

return ;

}

/***************************************************************************************************** moteur.h *****/
#ifndef _moteur_H
#define _moteur_H

#include <arduino.h>
#include "pins_arduino.h"
#include "wiringprivate.h" // to use cbi and sbi #reference

#endif __cplusplus

extern "C" {

#endif

void init_moteur(); // declaration des entre sorties moteurs

```



```

void avant();           // fonctions de stabilisation du robot

void arriere();

void power_off();

float f_droit(float origin);      // fonctions de déplacement du robot

float f_gauche(float origin);    // peut modifier le % de puissance

float f_avant(float origin);    // et le setPoint ( point d équilibre)

float f_arriere(float origin);

float f_arret(float origin);

static char commandeD = 0; // % de vitesse de direction propre au fichier source

static char commandeG = 0; // % de vitesse de direction propre au fichier source

static char vitesseD = 80; // vitesse par défaut propre au fichier source

static char vitesseG = 80; // vitesse par défaut propre au fichier source

void change_vitesse(bool motor, char vite); // 1 = D, 0 = G

// https://playground.arduino.cc/Main/TimerPWMCheatsheet source des TIMER et des pins PWM

void monPWM(uint8_t pin, int val);

void moteurD(byte sense); // 1 = avant; 0 = arrière

void moteurG(byte sense); // 1 = avant; 0 = arrière

```



```

#define __cplusplus
}

#endif

#endif

/***** moteur.c *****/
#include "moteur.h"

void init_moteur()
{
    pinMode(5,OUTPUT);      // on declare toutes nos entrées sorties
    pinMode(6,OUTPUT);
    pinMode(10,OUTPUT);
    pinMode(9,OUTPUT);

    power_off(); // on met tt les port a 0
}

void avant() // 2 moteurs sens de rotation avant
{
    moteurD(1);
    moteurG(1);
}

```



```

}

void arriere() // 2 moteurs sens de rotation arriere

{
    moteurD(0);

    moteurG(0);

}

void power_off()

{
    monPWM(5,0);

    monPWM(6,0);

    monPWM(10,0);

    monPWM(9,0);

}

float f_droit(float origin)

{
    commandeD = 1; // vitesse de direction gauche élevé

    commandeG = 0.8;

    return origin;
}

float f_gauche(float origin)

{
    commandeD = 0.8; // vitesse de direction gauche élevé

    commandeG = 1;
}

```



```

return origin;

}

float f_avant(float origin)

{

commandeD = 1; // vitesse de direction gauche élevé

commandeG = 1;

return origin + 9.3;

}

float f_arriere(float origin)

{

commandeD = 1; // vitesse de direction gauche élevé

commandeG = 1;

return origin - 7.2;

}

float f_arret(float origin) // fonction casi inutile.....

{

commandeD = 1; // vitesse de direction gauche élevé

commandeG = 1;

return origin;

}

void change_vitesse(bool motor, char vite)

{

if(motor) vitesseD = vite * commandeD;           // 1 = D, 0 = G

```



```
else vitesseG = vite * commandeG;
```

```
}
```

```
void moteurD(byte sense)
```

```
{
```

```
    if(sense == 1)      // avant
```

```
{
```

```
        monPWM(5,vitesseD);
```

```
        monPWM(6,0);
```

```
}
```

```
    else          // arriere
```

```
{
```

```
        monPWM(5,0);
```

```
        monPWM(6,vitesseD);
```

```
}
```

```
}
```

```
void moteurG(byte sense)
```

```
{
```

```
    if(sense == 1)      // avant
```

```
{
```

```
        monPWM(10,vitesseG );
```

```
        monPWM(9,0);
```

```
}
```

```
    else          // arriere
```



```

}

monPWM(10,0);

monPWM(9,vitesseG);

}

}

void monPWM(uint8_t pin, int val)

{

// We need to make sure the PWM output is enabled for those pins

// that support it, as we turn it off when digitally reading or

// writing with them. Also, make sure the pin is in output mode

// for consistency with Wiring, which doesn't require a pinMode

// call for the analog output pins.

if (val == 0)

{

digitalWrite(pin, LOW);

}

else if (val == 255)

{

digitalWrite(pin, HIGH);

}

else

{

```



```

switch(digitalPinToTimer(pin))

{
#ifndef TCCR0 && defined(COM00) && !defined(__AVR_ATmega8__)
case TIMER0A:

// connect pwm to pin on timer 0

sbi(TCCR0, COM00);

OCR0 = val; // set pwm duty

break;

#endif

#ifndef TCCR0A && defined(COM0A1)
case TIMER0A:

// connect pwm to pin on timer 0, channel A

sbi(TCCR0A, COM0A1);

OCR0A = val; // set pwm duty

break;

#endif

#ifndef TCCR0A && defined(COM0B1)
case TIMER0B:

// connect pwm to pin on timer 0, channel B

sbi(TCCR0A, COM0B1);

OCR0B = val; // set pwm duty

```



```

break;

#endif

#if defined(TCCR1A) && defined(COM1A1)

case TIMER1A:

// connect pwm to pin on timer 1, channel A 5555555555555555

sbi(TCCR1A, COM1A1);

OCR1A = val; // set pwm duty

break;

#endif

#if defined(TCCR1A) && defined(COM1B1)

case TIMER1B:

// connect pwm to pin on timer 1, channel B

sbi(TCCR1A, COM1B1);

OCR1B = val; // set pwm duty

break;

#endif

#if defined(TCCR1A) && defined(COM1C1)

case TIMER1C:

// connect pwm to pin on timer 1, channel B

sbi(TCCR1A, COM1C1);

```



```
OCR1C = val; // set pwm duty  
break;  
#endif  
  
#if defined(TCCR2A) && defined(COM2B1)  
case TIMER2B:  
// connect pwm to pin on timer 2, channel B  
sbi(TCCR2A, COM2B1);  
OCR2B = val; // set pwm duty  
break;  
#endif  
  
#if defined(TCCR3A) && defined(COM3A1)  
case TIMER3A:  
// connect pwm to pin on timer 3, channel A  
sbi(TCCR3A, COM3A1);  
OCR3A = val; // set pwm duty  
break;  
#endif  
  
#if defined(TCCR3A) && defined(COM3B1)  
case TIMER3B:  
// connect pwm to pin on timer 3, channel B
```



```

sbi(TCCR3A, COM3B1);

OCR3B = val; // set pwm duty

break;

#endif

#if defined(TCCR4A) && defined(COM4B1)

case TIMER4B:

// connect pwm to pin on timer 4, channel B

sbi(TCCR4A, COM4B1);

OCR4B = val; // set pwm duty

break;

#endif

#if defined(TCCR4C) && defined(COM4D1)

case TIMER4D:

// connect pwm to pin on timer 4, channel D

sbi(TCCR4C, COM4D1);

#if defined(COM4D0) // only used on 32U4

cbi(TCCR4C, COM4D0);

#endif

OCR4D = val; // set pwm duty

break;

#endif

```



```

case NOT_ON_TIMER:

default:

break;

}

}

}

***** mpu6050.h *****/
#ifndef _MPU6050_H
#define _MPU6050_H

#include <arduino.h>

#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter
#include "I2C.h"

#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg instead - please read:
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf

static Kalman kalmanX; // Create the Kalman instances

static Kalman kalmanY;

/* IMU Data */

static double accX, accY, accZ;

static double gyroX, gyroY, gyroZ;

```



```

static int16_t tempRaw;

static double gyroXangle, gyroYangle; // Angle calculate using the gyro only
static double compAngleX, compAngleY; // Calculated angle using a complementary filter
static double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

static uint32_t timer;
static uint8_t i2cData[14]; // Buffer for I2C data

// TODO: Make calibration routine

void init_MPU();
void loop_MPU();

double getY();
double getX();
double get_accX();
double get_accY();

// ***** POLAIRE FUNCTION *****
float polaire_getR(float x, float y);    // pour eviter le decalage de valeur pendant la lecture de x et y
float polaire_getTeta();

```



#endif

```
***** mpu6050.h *****
```

```
#ifndef _mpu6050_H
```

```
#define _mpu6050_H
```

```
#include <arduino.h>
```

```
#include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter
```

```
#include "I2C.h"
```

```
#define RESTRICT_PITCH // Comment out to restrict roll to ±90deg instead - please read:
```

```
http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf
```

```
static Kalman kalmanX; // Create the Kalman instances
```

```
static Kalman kalmanY;
```

```
/* IMU Data */
```

```
static double accX, accY, accZ;
```

```
static double gyroX, gyroY, gyroZ;
```

```
static int16_t tempRaw;
```

```
static double gyroXangle, gyroYangle; // Angle calculate using the gyro only
```

```
static double compAngleX, compAngleY; // Calculated angle using a complementary filter
```



```

static double kalAngleX, kalAngleY; // Calculated angle using a Kalman filter

static uint32_t timer;

static uint8_t i2cData[14]; // Buffer for I2C data

// TODO: Make calibration routine

void init_MPU();

void loop_MPU();

double getY();

double getX();

double get_accX();

double get_accY();

// **** POLAIRE FUNCTION ****

float polaire_getR(float x, float y); // pour eviter le decalage de valeur pendant la lecture de x et y

float polaire_getTeta();

#endif

/********************************************* mpu6050.cpp *****/
#include "mpu6050.h"

void init_MPU()

```



```

{
    Wire.begin();

    TWBR = ((F_CPU / 400000L) - 16) / 2; // Set I2C frequency to 400kHz

    i2cData[0] = 7; // Set the sample rate to 1000Hz - 8kHz/(7+1) = 1000Hz
    i2cData[1] = 0x00; // Disable FSYNC and set 260 Hz Acc filtering, 256 Hz Gyro filtering, 8 KHz sampling
    i2cData[2] = 0x00; // Set Gyro Full Scale Range to ±250deg/s
    i2cData[3] = 0x00; // Set Accelerometer Full Scale Range to ±2g
    while (i2cWrite(0x19, i2cData, 4, false)); // Write to all four registers at once
    while (i2cWrite(0x6B, 0x01, true)); // PLL with X axis gyroscope reference and disable sleep mode

    while (i2cRead(0x75, i2cData, 1));
    if (i2cData[0] != 0x68) { // Read "WHO_AM_I" register
        //Serial.print(F("Error reading sensor"));

        while (1);
    }

}

delay(100); // Wait for sensor to stabilize
/* Set kalman and gyro starting angle */
while (i2cRead(0x3B, i2cData, 6));
accX = (i2cData[0] << 8) | i2cData[1];

```



```

accY = (i2cData[2] << 8) | i2cData[3];
accZ = (i2cData[4] << 8) | i2cData[5];

// Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25 and eq. 26

// atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2

// It is then converted from radians to degrees

#ifndef RESTRICT_PITCH // Eq. 25 and 26

double roll = atan2(accY, accZ) * RAD_TO_DEG;
double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;

#else // Eq. 28 and 29

double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
double pitch = atan2(-accX, accZ) * RAD_TO_DEG;

#endif

kalmanX.setAngle(roll); // Set starting angle
kalmanY.setAngle(pitch);
gyroXangle = roll;
gyroYangle = pitch;
compAngleX = roll;
compAngleY = pitch;

timer = micros();

}


```



```

void loop_MPU()

{
    /* Update all the values */

    /*while (*i2cRead(0x3B, i2cData, 14)/*);

    accX = ((i2cData[0] << 8) | i2cData[1]);

    accY = ((i2cData[2] << 8) | i2cData[3]);

    accZ = ((i2cData[4] << 8) | i2cData[5]);

    tempRaw = (i2cData[6] << 8) | i2cData[7];

    gyroX = (i2cData[8] << 8) | i2cData[9];

    gyroY = (i2cData[10] << 8) | i2cData[11];

    gyroZ = (i2cData[12] << 8) | i2cData[13];

    double dt = (double)(micros() - timer) / 1000000; // Calculate delta time

    timer = micros();

    // Source: http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf eq. 25 and eq. 26

    // atan2 outputs the value of -π to π (radians) - see http://en.wikipedia.org/wiki/Atan2

    // It is then converted from radians to degrees

    // la droite ou la gauche (le roll)

    // l'avant ou l'arrière (le pitch)

#define RESTRICT_PITCH // Eq. 25 and 26

    double roll = atan2(accY, accZ) * RAD_TO_DEG;

    double pitch = atan(-accX / sqrt(accY * accY + accZ * accZ)) * RAD_TO_DEG;

#else // Eq. 28 and 29

```



```

double roll = atan(accY / sqrt(accX * accX + accZ * accZ)) * RAD_TO_DEG;
double pitch = atan2(-accX, accZ) * RAD_TO_DEG;

#endif

double gyroXrate = gyroX / 131.0; // Convert to deg/s

double gyroYrate = gyroY / 131.0; // Convert to deg/s

#ifndef RESTRICT_PITCH

// This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees

if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {

    kalmanX.setAngle(roll);

    compAngleX = roll;

    kalAngleX = roll;

    gyroXangle = roll;

} else

    kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter

if (abs(kalAngleX) > 90)

    gyroYrate = -gyroYrate; // Invert rate, so it fits the restricted accelerometer reading

kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);

#else

// This fixes the transition problem when the accelerometer angle jumps between -180 and 180 degrees

if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {

```



```

kalmanY.setAngle(pitch);

compAngleY = pitch;

kalAngleY = pitch;

gyroYangle = pitch;

} else

kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // Calculate the angle using a Kalman filter

if (abs(kalAngleY) > 90)

gyroXrate = -gyroXrate; // Invert rate, so it fits the restricted accelerometer reading

kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // Calculate the angle using a Kalman filter

#endif

gyroXangle += kalmanX.getRate() * dt; // Calculate gyro angle using the unbiased rate

gyroYangle += kalmanY.getRate() * dt;

compAngleX = 0.93 * (compAngleX + gyroXrate * dt) + 0.07 * roll; // Calculate the angle using a Complimentary
filter

compAngleY = 0.93 * (compAngleY + gyroYrate * dt) + 0.07 * pitch;

// Reset the gyro angle when it has drifted too much

if (gyroXangle < -180 || gyroXangle > 180)

gyroXangle = kalAngleX;

if (gyroYangle < -180 || gyroYangle > 180)

gyroYangle = kalAngleY;

}

```



```

double getY()
{
    return gyroYangle;
}

double getX()
{
    return gyroXangle;
}

double get_accY()
{
    return accY;
}

double get_accX()
{
    return accX;
}

float polaire_getR(float x, float y)      // double sqrt(double a);
{
    return sqrt( (x*x) + (y*y)) ;
}

float polaire_getTeta() //      double sqrt(double a);
{
    float x=getX(); // pou eviter le decalage de valeur pendant la lecture de x et y
}

```



```
return (getX() / polaire_getR(getX(),getY()) ); // teta = x / R  
}
```

```
***** serialRX.h *****
```

```
#ifndef SERIALRX_H_INCLUDED
```

```
#define SERIALRX_H_INCLUDED
```

```
#include <avr/io.h>
```

```
#include <avr/interrupt.h>
```

```
#include <util/delay.h>
```

```
#include <stdlib.h>
```

```
#include "arduino.h"
```

```
#include "utilitaire.h" // led debug reception
```

```
***** REGISTRE UART ATMEGA328p *****
```

```
#define UARTB UCSR0B
```

```
#define UARTC UCSR0C
```

```
#define UARTA UCSR0A
```

```
#define BAUD UBRR0L
```

```
#define BAUD2 UBRR0H
```

```
#define TXEN TXENO
```

```
#define RXEN RXENO
```



```

#define enableUartRX RXC0
#define UartIsEmpty UDRE0
#define UartRegister UDR0

#define UMSEL0 UMSEL00
#define UMSEL1 UMSEL01
#define UCSZ_0 UCSZ00
#define UCSZ_1 UCSZ01
#define UCSZ_2 UCSZ02

***** REGISTRE UART ATMEGA328p *****

***** REGISTRE UART ATMEGA32U4 *****

#define UARTB UCSR1B
#define UARTC UCSR1C
#define UARTA UCSR1A
#define BAUD UBRR1L
#define BAUD2 UBRR1H
#define TXEN TXEN1
#define RXEN RXEN1

#define enableUartRX RXC1
#define UartIsEmpty UDRE1
#define UartRegister UDR1

```



```

#define UMSEL0 UMSEL10
#define UMSEL1 UMSEL11
#define UCSZ_0 UCSZ10
#define UCSZ_1 UCSZ11
#define UCSZ_2 UCSZ12
//********************************************************************* REGISTRE UART ATMEGA32U4 *****/
#define F_CPU 16000000UL
///#define BAUD_PRESCALE ((F_CPU)/(BAUD*16UL)-1)      // ARDUINO all
#define BAUD_PRESCALE (((F_CPU + (USART_BAUDRATE * 8UL)) / (USART_BAUDRATE * 16UL)) - 1) // ARDUINO micro pro

unsigned char uart_recieve (void);
void uart_transmit (char data);
void UART_Init(long USART_BAUDRATE);
char get_buffer();
static char buffeur;
/*
    registre UART A
        -Bit 7 – RXC: USART Receive Complete
        -Bit 6 – TXC: USART Transmit Complete

```



- Bit 5 – UDRE: USART Data Register Empty
- UDRE0 USART Data Register Empty. Set when the UDR0 register is empty and new data can be transmitted

registre UART B

- Bit 7 – RXCIE: RX Complete Interrupt Enable
- Bit 6 – TXCIE: TX Complete Interrupt Enable
- Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable
- Bit 4 – RXEN: Receiver Enable
- Bit 3 – TXEN: Transmitter Enable

registre UART C

- USART Mode Select 1 and 0. UMSEL01 and UMSEL00
- UCSZ01,UCSZ00,USART Character Size 1 and 0. Used together with UCSZ20 to set data frame size

*/

#endif

```
***** serialRX.cpp *****
```

```
#include "serialRX.h"
```



```

void UART_Init(long USART_BAUDRATE)

{
    // USART initialization

    UCSR0A= 0x00;           // Clear the USART status register

    UARTB |= ( (1<<RXEN) | (1<<TXEN) | (1<<RXCIE0) ); // Enable Receiver and Transmitter REGISTER B (one wire )

    UARTC |= ( (0<<UMSEL0)|(0<<UMSEL1) | (0<<UCSZ_2) | (1<<UCSZ_1) | (1<<UCSZ_0));

        /* UMSEL0.. asynchronous (00), synchronous (01) and master SPI (11).8-bit data

        UCSZ20 UCSZ01 and UCSZ00 control the data size. Possible sizes are 5-bit (000

        6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111)*/

    BAUD = BAUD_PRESCALE;      /* Load lower 8-bits of the baud rate */

    BAUD2 = (BAUD_PRESCALE >> 8); /* Load upper 8-bits*/

    sei();

}

// function to send data

void uart_transmit (char data)

{
    while(!UARTA & (1<<UartIsEmpty));           // wait while register is free

    UartRegister = data; // here 10 means decimaldata;           // load data in the register

}

// function to receive data

unsigned char uart_recieve (void)

{

```



```

while(!(UARTA) & (1<<enableUartRX));           // wait while data is being received

return UartRegister;                           // return 8-bit data

}

char get_buffer()
{
    return buffeur;
}

// #define USART0_RX_vect _VECTOR(20) /* USART Rx Complete */

ISR( USART_RX_vect )
{
    cli();

    led_power(true);

    buffeur = uart_recieve ();

    sei();
}

```

utilitaire.h

```

#ifndef _utilitaire_H
#define _utilitaire_H

```



```

#define __cplusplus

extern "C" { // pour coder en C dans un langage C++ (prore a arduino j imagine)

#endif

#define LED DDB3 // pin 16 led to debug

#include <stdbool.h>

#include <avr/io.h>

void led_init();

void led_power(bool power);

// value X, entre min, entre max, sortie min, sortie max

float mapping(float x, int in_min, float in_max, float out_min, float out_max);

void croissant(float *tableau );

float moyenne(float *tableau );

#endif __cplusplus

}

#endif

#endif

```

***** utilitaire.c *****

```

#include "utilitaire.h"

float mapping(float x, int in_min, float in_max, float out_min, float out_max)

```



```

{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void croissant(float *tableau )
{
    // ma taille : sizeof(measurePoint)/sizeof(int)

    float BUFFTableau[(sizeof(tableau)/sizeof(int))]={0}; // buffer au dimension du tableau en parametre

    for( int i=0; i < (sizeof(tableau)/sizeof(int)); i++) // lecture du tableau

    {
        int score = 0; // init lvl case a 0

        for( int s=0; s < (sizeof(tableau)/sizeof(int)); s++) // test du lvl de la variable sur toutes les cases

        {
            if(tableau[i] > tableau[s]) // si case est > a case suivante alors lvl++
            {
                score++;
            }
        }

        BUFFTableau[score] = tableau[i]; // rangement de la case en fonction de son score
    }

    tableau = BUFFTableau; // on transfert le buffer dans notre tableau
}

```



```

float moyenne(float *tableau )

{
    float resultat=0;

    for(int i=0;i<(sizeof(tableau)/sizeof(int));i++)
    {
        resultat += tableau[i];
    }

    return resultat / (sizeof(tableau)/sizeof(int)); // return somme tableau / taille tableau
}

void led_init()

{
    DDRB |= (1<< LED); // led is output

    PORTB &= ~(1<<LED);
}

void led_power(bool power)

{
    if(power) PORTB |= (1<<LED);// allume led sinon eteint
    else PORTB &= ~(1<<LED);
}

***** wiring_private.h *****
/*
wiring_private.h - Internal header file.

Part of Arduino - http://www.arduino.cc/

```



Copyright (c) 2005-2006 David A. Mellis

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General
Public License along with this library; if not, write to the
Free Software Foundation, Inc., 59 Temple Place, Suite 330,
Boston, MA 02111-1307 USA

*/

```
#ifndef WiringPrivate_h
```

```
#define WiringPrivate_h
```

```
#include <avr/io.h>
```



```

#include <avr/interrupt.h>
#include <stdio.h>
#include <stdarg.h>
#include "Arduino.h"
#ifndef __cplusplus
extern "C"{
#endif
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif
uint32_t countPulseASM(volatile uint8_t *port, uint8_t bit, uint8_t stateMask, unsigned long maxloops);

#define EXTERNAL_INT_0 0
#define EXTERNAL_INT_11
#define EXTERNAL_INT_2 2
#define EXTERNAL_INT_3 3
#define EXTERNAL_INT_4 4
#define EXTERNAL_INT_5 5
#define EXTERNAL_INT_6 6
#define EXTERNAL_INT_7 7

```



```

#ifndef __AVR_ATmega1280__ || defined(__AVR_ATmega2560__ || defined(__AVR_ATmega128RFA1__ || 
defined(__AVR_ATmega256RFR2__)) || \
defined(__AVR_AT90USB82__ || defined(__AVR_AT90USB162__ || defined(__AVR_ATmega32U2__ || 
defined(__AVR_ATmega16U2__ || defined(__AVR_ATmega8U2__))

#define EXTERNAL_NUM_INTERRUPTS 8

#elif defined(__AVR_ATmega1284__ || defined(__AVR_ATmega1284P__ || defined(__AVR_ATmega644__ || 
defined(__AVR_ATmega644A__ || defined(__AVR_ATmega644P__ || defined(__AVR_ATmega644PA__))

#define EXTERNAL_NUM_INTERRUPTS 3

#elif defined(__AVR_ATmega32U4__) // ça c est notre carte ^^

#define EXTERNAL_NUM_INTERRUPTS 5

#else

#define EXTERNAL_NUM_INTERRUPTS 2

#endif

typedef void (*voidFuncPtr)(void);

#ifndef __cplusplus
} // extern "C"
#endif
#endif

```

PROGRAMME TRANSFERT DES INFORMATIONS :

```

***** main.ino *****
#include "rgb.h"

```



```

#include "serialTX.h"

#include "spi_slave.h"

#include "bluetooth.h"

String t = "spi"; // default configuration (spi other mode)

void setup()

{

UART_Init(115200);

led_init(); // help to debug

start_rgb();

if (t == "spi") spi_init();

else bluetooth_init();

char commande = '5', pastCommande = '9';

led_power(false);

while(1)

{

continued_rgb(); // capteur de distance

if ( (t == "spi") && (get_Distance() > 4) )

{

commande = get_SpiData(); // send data ESP

led_power(false);

}

else if((t == "bluetooth") && (get_Distance() > 4) )

{

```





```
commande = bluetooth_loop();           // send data bluetooth

led_power(false);

}

else if(get_Distance() < 4)

{

commande = '3';

led_power(false);

}

else if(get_Distance() > 4)

{

commande = '5';

led_power(false);

}

if(commande != pastCommande)

{

pastCommande = commande;

uart_transmit(commande);

led_power(true);

}

}

}

***** bluetooth.h *****

#ifndef BLUETOOTH_H_INCLUDED
```



```

#define BLUETOOTH_H_INCLUDED

#include <SoftwareSerial.h>

static SoftwareSerial HC06(11,12); // (RX, TX) (pin Rx BT, pin Tx BT)

void bluetooth_init();

char bluetooth_loop();

#endif

***** bluetooth.cpp *****

#include "bluetooth.h"

void bluetooth_init()
{
    HC06.begin(9600);
}

char bluetooth_loop()
{
    static char buffeur = 5; // default = stop

    if(HC06.available())
    {
        buffeur = HC06.read();
    }
}

```



```
    return buffeur;  
}
```

```
*****          rgb.h          *****
```

```
#ifndef RGB_H_INCLUDED
```

```
#define RGB_H_INCLUDED
```

```
#include <Wire.h>
```

```
#include <Arduino.h>
```

```
#include "utilitaire.h"
```

```
#define APDS9960_ADR 0x39
```

```
#define APDS9960_ID 0xAB
```

```
static String SERIAL_STRING;
```

```
static unsigned char DATA_U,DATA_D,DATA_L,DATA_R;
```

```
static unsigned char OLD_U,OLD_D,OLD_L,OLD_R;
```

```
static unsigned char work;
```

```
static unsigned char U_PEAK_END_FLAG,D_PEAK_END_FLAG,L_PEAK_END_FLAG,R_PEAK_END_FLAG;
```

```
static unsigned char STATUS_UD,STATUS_LR;
```

```
static unsigned char OLD_STATUS_UD,OLD_STATUS_LR;
```

```
static unsigned char DISP_FLAG;
```



```

static unsigned char NOISE_LEVEL = 2;

static unsigned char DECIDE_FLAG;

static unsigned int PHASE_COUNTER;

static unsigned int U_PEAK,D_PEAK,L_PEAK,R_PEAK;

static int distance=0;

static void I2C_WRITE(unsigned char REG_ADR, unsigned char DATA);

static unsigned char I2C_READ(unsigned char REG_ADR);

static void RESET_VARIABLE(void);

static String DISP_DIR(void);

static void DATA_SYORI(void);

extern void start_rgb();           //void setup

extern String continued_rgb();     // void loop

extern int get_Distance();         //void setup

#endif

/*******************************************          rgb.cpp          *****/
#include "rgb.h"

```



```

=====
//*****



static unsigned char I2C_READ(unsigned char REG_ADR)
{
    Wire.beginTransmission(APDS9960_ADR);
    Wire.write(REG_ADR);
    Wire.endTransmission(false);
    Wire.requestFrom(APDS9960_ADR, 1);
    return Wire.read();
}

static void I2C_WRITE(unsigned char REG_ADR, unsigned char DATA)
{
    Wire.beginTransmission(APDS9960_ADR);
    Wire.write(REG_ADR);
    Wire.write(DATA);
    Wire.endTransmission(false);
}

//*****



extern void start_rgb()
{

```



```

Wire.begin();           //I2C BEGIN

work = I2C_READ(0x92);    //READ REGISTER 0x92 (ADPS9960's ID)

if (work == APDS9960_ID)

{

    Serial.println("Found APDS-9960 : ID = 0xAB on Register Address 0x92");

    Serial.println("GESTURE SENSOR START");

}

else Serial.println("APDS-9960 NOT FOUND");

I2C_WRITE(0x80, B01000101); //POWER ON<0>, GESTURE ENABLE<6>, PROXIMITY DETECT
ENALBE<2>,AEN=0

I2C_WRITE(0x90, B00110000); //Gesture LED Drive Strength 300%(max)

I2C_WRITE(0xA3, B01100100); //Reserve0, Gain x8(11), LED Drive 100mA(00), Wait Time see under number
//111=39.2mS 110=30.8mS 101=22.4mS 100=14.0mS 011=8.4mS 010=5.6mS 001=2.8ms 000=0mS

I2C_WRITE(0xA4, 70);      //U MINUS OFFSET

I2C_WRITE(0xA5, 0);       //D MINUS OFFSET

I2C_WRITE(0xA7, 10);      //L MINUS OFFSET

I2C_WRITE(0xA9, 34);      //R MINUS OFFSET

I2C_WRITE(0xAB, B00000001); //GIEN off<1>(INTERRUPT DISABLE), GMODE ON<0>

RESET_VARIABLE();

}

extern void reseting()

{

```



```

I2C_WRITE(0x80, B01000101); //POWER ON<0>, GESTURE ENABLE<6>, PROXIMITY DETECT
ENALBE<2>,AEN=0

I2C_WRITE(0x90, B00110000); //Gesture LED Drive Strength 300%(max)

I2C_WRITE(0xA3, B01100100); //Reserve0, Gain x8(11), LED Drive 100mA(00), Wait Time see under number
//111=39.2mS 110=30.8mS 101=22.4mS 100=14.0mS 011=8.4mS 010=5.6mS 001=2.8ms 000=0mS

I2C_WRITE(0xA4, 70);      //U MINUS OFFSET
I2C_WRITE(0xA5, 0);       //D MINUS OFFSET
I2C_WRITE(0xA7, 10);      //L MINUS OFFSET
I2C_WRITE(0xA9, 34);      //R MINUS OFFSET

I2C_WRITE(0xAB, B00000001); //GIEN off<1>(INTERRUPT DISABLE), GMODE ON<0>
RESET_VARIABLE();

}

```

```

extern int get_Distance()

{
    return distance;
}

```

```

extern String continued_rgb()

{
    String posi = "nul";
    work = I2C_READ(0xAE); //READ GESTUR FIFO LEVEL REGISTER
    if (work != 0)          //IF FIFO HAS SOME DATA
    {

```





```
DATA_U = I2C_READ(0xFC);
DATA_D = I2C_READ(0xFD);
DATA_L = I2C_READ(0xFE);
DATA_R = I2C_READ(0xFF);
```

```
distance = mapping((DATA_D+DATA_L+DATA_R+DATA_U)/4, 0, 255, 10, 0);
```

```
if ((DATA_U > NOISE_LEVEL) && (DATA_D > NOISE_LEVEL) && (DATA_L > NOISE_LEVEL) && (DATA_R >
NOISE_LEVEL)) //NOISE CANCEL
{
    DATA_SYORI();      //
    PHASE_COUNTER++; //
    DISP_FLAG = 1;     //
}
else
{
    if (DISP_FLAG)
    {
        DISP_FLAG = 0;
        posi = DISP_DIR();
    }
    RESET_VARIABLE();
    return posi;
}
```



```
 }  
 }  
  
//*****
```

```
static void RESET_VARIABLE(void)
```

```
{  
    PHASE_COUNTER = 0;
```

```
    U_PEAK = 0;
```

```
    D_PEAK = 0;
```

```
    L_PEAK = 0;
```

```
    R_PEAK = 0;
```

```
    OLD_U = 0;
```

```
    OLD_D = 0;
```

```
    OLD_L = 0;
```

```
    OLD_R = 0;
```

```
    U_PEAK_END_FLAG = 0;
```

```
    D_PEAK_END_FLAG = 0;
```

```
    L_PEAK_END_FLAG = 0;
```

```
    R_PEAK_END_FLAG = 0;
```

```
    STATUS_UD = 0;
```

```
    STATUS_LR = 0;
```

```
    OLD_STATUS_UD = 0;
```

```
    OLD_STATUS_LR = 0;
```



```

SERIAL_STRING = "";

DISP_FLAG = 0;

DECIDE_FLAG = 0;

}

//-----

static String DISP_DIR(void)

{

if (!(SERIAL_STRING == ""))

{

    //Serial.println(SERIAL_STRING);

    return SERIAL_STRING;

}

else

    return "nul";

}

//-----


static void DATA_SYORI(void)

{

if (DATA_U > OLD_U)      //IF NEW_DATA > OLD_DATA_BUFFER(APPROACH TO PEAK)

{

    OLD_U = DATA_U;          //SAVE NEW_DATA TO OLD_DATA_BUFFER

    U_PEAK = PHASE_COUNTER;  //PEAK_PHASE RENEWAL
}

```



```

U_PEAK_END_FLAG = 0;           //STILL PEAK or APPROACH TO PEAK

}

else

{

    U_PEAK_END_FLAG = 1;           //PEAK WAS GONE

}

//*****



if (DATA_D > OLD_D)

{

    OLD_D = DATA_D;

    D_PEAK = PHASE_COUNTER;

    D_PEAK_END_FLAG = 0;

}

else

{

    D_PEAK_END_FLAG = 1;

}

//*****



if (DATA_L > OLD_L)

{

    OLD_L = DATA_L;

    L_PEAK = PHASE_COUNTER;

    L_PEAK_END_FLAG = 0;
}

```



```

}

else

{

    L_PEAK_END_FLAG = 1;

}

//*****



if (DATA_R > OLD_R)

{

    OLD_R = DATA_R;

    R_PEAK = PHASE_COUNTER;

    R_PEAK_END_FLAG = 0;

}

else

{

    R_PEAK_END_FLAG = 1;

}

//*****



if (U_PEAK_END_FLAG && D_PEAK_END_FLAG && L_PEAK_END_FLAG && R_PEAK_END_FLAG) //IF ALL PEAK WAS GONE

{

    DECIDE_FLAG = 0;

    if ((U_PEAK > D_PEAK) & (U_PEAK >= L_PEAK) & (U_PEAK >= R_PEAK)) //U_PEAK WAS LAST

    {

        SERIAL_STRING = "DOWN";

```





```
DECIDE_FLAG = 1;

}

if ((D_PEAK > U_PEAK) & (D_PEAK >= L_PEAK) & (D_PEAK >= R_PEAK)) //D_PEAK WAS LAST

{

SERIAL_STRING = "UP";

DECIDE_FLAG = 1;

}

if ((L_PEAK >= U_PEAK) & (L_PEAK >= D_PEAK) & (L_PEAK > R_PEAK)) //L_PEAK WAS LAST

{

SERIAL_STRING = "RIGHT";

DECIDE_FLAG = 1;

}

if ((R_PEAK >= U_PEAK) & (R_PEAK >= D_PEAK) & (R_PEAK > L_PEAK)) //R_PEAK WAS LAST

{

SERIAL_STRING = "LEFT";

DECIDE_FLAG = 1;

}

if (!DECIDE_FLAG)SERIAL_STRING = "NONE"; //CAN'T DECIDE

}

}

***** serialTX.h *****

#ifndef SERIALTX_H_INCLUDED
#define SERIALTX_H_INCLUDED
```



```

#include <avr/io.h>

#include <util/delay.h>

#include <stdlib.h>

***** REGISTRE UART ATMEGA328p *****

#define UARTB UCSR0B

#define UARTC UCSR0C

#define UARTA UCSR0A

#define BAUD UBRR0L

#define BAUD2 UBRR0H

#define TXEN TXENO

#define RXEN RXENO

#define enableUartRX RXCO

#define UartIsEmpty UDRE0

#define UartRegister UDR0

#define UMSEL0 UMSEL00

#define UMSEL1 UMSEL01

#define UCSZ_0 UCSZ00

#define UCSZ_1 UCSZ01

#define UCSZ_2 UCSZ02

***** REGISTRE UART ATMEGA328p *****

```



```
***** REGISTRE UART ATMEGA32U4 *****
```

```
#define UARTB UCSR1B
```

```
#define UARTC UCSR1C
```

```
#define UARTA UCSR1A
```

```
#define BAUD UBRR1L
```

```
#define BAUD2 UBRR1H
```

```
#define TXEN TXEN1
```

```
#define RXEN RXEN1
```

```
#define enableUartRX RXC1 // I know what is her NAME
```

```
#define UartIsEmpty UDRE1 // The logic is better of GOOGLE ^^°
```

```
#define UartRegister UDR1
```

```
#define UMSEL0 UMSEL10
```

```
#define UMSEL1 UMSEL11
```

```
#define UCSZ_0 UCSZ10
```

```
#define UCSZ_1 UCSZ11
```

```
#define UCSZ_2 UCSZ12
```

```
***** REGISTRE UART ATMEGA32U4 *****/
```

```
#define F_CPU 16000000UL /* Define frequency here its 8MHz */
```

```
// ARDUINO all #define BAUD_PRESCALE ((F_CPU / 6 / USART_BAUDRATE) - 1)
```



```
#define BAUD_PRESCALE (((F_CPU + (USART_BAUDRATE * 8UL)) / (USART_BAUDRATE * 16UL)) - 1) //  
ARDUINO micro pro
```

// BAUD_PRESCALE is the value that we have to load in UBRR register to set defined baud rate.

```
unsigned char uart_recieve (void);  
  
void uart_transmit (uint16_t data);  
  
void UART_Init(long USART_BAUDRATE);  
  
static char buffeur;
```

```
/*
```

registre UART A

- Bit 7 – RXC: USART Receive Complete
- Bit 6 – TXC: USART Transmit Complete
- Bit 5 – UDRE: USART Data Register Empty
- UDRE0 USART Data Register Empty. Set when the UDR0 register is empty and new data can be transmitted

registre UART B

- Bit 7 – RXCIE: RX Complete Interrupt Enable
- Bit 6 – TXCIE: TX Complete Interrupt Enable
- Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable
- Bit 4 – RXEN: Receiver Enable
- Bit 3 – TXEN: Transmitter Enable



registre UART C

- USART Mode Select 1 and 0. UMSEL01 and UMSEL00
- UCSZ01,UCSZ00,USART Character Size 1 and 0. Used together with UCSZ20 to set data frame size

*/

#endif

***** serialTX.cpp *****

```
#include "rs232TX.h"
```

```
void UART_Init(long USART_BAUDRATE)
```

```
{
```

```
// USART initialization
```

```
UARTB |= ( (0<<RXEN) | (1<<TXEN)); // Enable Receiver and Transmitter REGISTER B (one wire )
```

```
UARTC |= ( (0<<UMSEL0)|(0<<UMSEL1) | (0<<UCSZ_2) | (1<<UCSZ_1) | (1<<UCSZ_0));
```

```
/* UMSEL0.. asynchronous (00), synchronous (01) and master SPI (11).8-bit data
```

```
UCSZ20 UCSZ01 and UCSZ00 control the data size. Possible sizes are 5-bit (000)
```

```
6-bit (001), 7-bit (010), 8-bit (011) and 9-bit (111)*/
```

```
BAUD = BAUD_PRESCALE; /* Load lower 8-bits of the baud rate */
```

```
BAUD2 = (BAUD_PRESCALE >> 8); /* Load upper 8-bits*/
```



```

}

// function to send data

void uart_transmit (uint16_t data)

{
    while(!(UARTA & (1<<UartIsEmpty)));           // wait while register is free

    UartRegister = data; // here 10 means decimaldata;           // load data in the register

}

// function to receive data

unsigned char uart_recieve (void)

{
    while(!(UARTA & (1<<enableUartRX)));           // wait while data is being received

    return UartRegister; // return 8-bit data

}

// #define USART_RX_vect _VECTOR(18) /* USART Rx Complete */

***** spi_slave.cpp *****
#ifndef SPI_H_INCLUDED

```



```

#define SPI_H_INCLUDED

#include <avr/io.h>

#include <avr/interrupt.h>

/***************************************************************************** atmega 328p *****/
#define SS DDB2

#define MISO DDB4 // atmega 328p

#define MOSI DDB3

#define SCK DDB5

/***************************************************************************** atmega 328p *****/
/****** atmega 32u4 *****/
#define SS DDB6 // why not...

#define MISO DDB3 //atmega 32u4

#define MOSI DDB2

#define SCK DDB1

/***************************************************************************** atmega 32u4 *****/
#define DDR_SPI DDRB

static char dataEcho = 5;

void spi_init();

char get_SpiData();

#endif

/***************************************************************************** spi_slave.cpp *****/

```



```

#include "spi_slave.h"

void spi_init()
{
    DDR_SPI &= ~( (1<<MOSI)|(1<<MISO)|(1<<SS)|(1<<SCK));

    // Define the following pins as output

    DDR_SPI |= (1<< MISO);

    SPCR = ((1<<SPE)|           // SPI Enable
             (1<<SPIE)|        // SPI Interrupt Enable
             (0u<<DORD)|       // Data Order (0:MSB first / 1:LSB first)
             (0<<MSTR)|        // Master/Slave select
             (0<<SPR1)|(0<<SPR0)| // SPI Clock Rate
             (0<<CPOL)|        // Clock Polarity (0:SCK low / 1:SCK hi when idle)
             (0<<CPHA));       // Clock Phase (0:leading / 1:trailing edge sampling)

    //SPI status register

    SPSR |= 0b00000000;

    sei(); // run interrupt
}

char get_SpiData()
{

```



```

return dataEcho;
}

ISR(SPI_STC_vect)
{
    // interrupt fonction type void

    cli();          // Disables all interrupts by clearing the global interrupt mask

    if(!(PINB & (1<<SS))) //while SS Low

    {
        SPDR = 0;      // send value

        while(!(SPSR & (1<<SPIF))); //wait SPI transfer complete

        dataEcho = SPDR; // get data du registre SPDR
    }

    sei();          // Enables interrupts by setting the global interrupt mask
}

```

```

/***************************************************************************************************** utilitaire.h *****/
#ifndef _utilitaire_H
#define _utilitaire_H

#ifndef __cplusplus

extern "C" { // pour coder en C dans un langage C++ (prore a arduino j imagine)

#endif

```



```

#define LED DDB1      // pin 9 led to debug

#include <stdbool.h>
#include <avr/io.h>

void led_init();
void led_power(bool power);

int mapping(float x, int in_min, int in_max, int out_min, int out_max); // value X, entre min, entre max, sortie min,
sortie max

void croissant(float *tableau );
float moyenne(float *tableau );

#endif __cplusplus
}

#endif
#endif

***** utilitaire.c *****

#include "utilitaire.h"

int mapping(float x, int in_min, int in_max, int out_min, int out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

```



```

void croissant(float *tableau )

{
    // ma taille : sizeof(measurePoint)/sizeof(int)

    float BUFFTableau[(sizeof(tableau)/sizeof(int))]={0}; // buffer au dimension du tableau en parametre

    for( int i=0; i < (sizeof(tableau)/sizeof(int)); i++) // lecture du tableau

    {
        int score = 0; // init lvl case a 0

        for( int s=0; s < (sizeof(tableau)/sizeof(int)); s++) // test du lvl de la variable sur toutes les cases

        {
            if(tableau[i] > tableau[s]) // si case est > a case suivante alors lvl++

            {
                score++;
            }
        }

        BUFFTableau[score] = tableau[i]; // rangement de la case en fonction de son score
    }

    tableau = BUFFTableau; // on transfert le buffer dans notre tableau
}

float moyenne(float *tableau )

```



```

{
float resultat=0;

for(int i=0;i<(sizeof(tableau)/sizeof(int));i++)

{
    resultat += tableau[i];

}

return resultat / (sizeof(tableau)/sizeof(int)); // return somme tableau / taille tableau
}

void led_init()

{
DDRB |= (1<< LED); // led is output

PORTB &= ~(1<<LED);

}

void led_power(bool power)

{
if(power) PORTB |= (1<<LED);// allume led sinon eteint

else PORTB &= ~(1<<LED);

}

```

PROGRAMME APPLICATION MOBILE :

```
***** main_application.java *****
```



```
package  
com.example.ne  
otxt.projet ;  
  
import android.annotation.SuppressLint ;  
importer android.bluetooth.BluetoothAdapter ;  
importer android.bluetooth.BluetoothDevice ;  
importer android.bluetooth.BluetoothSocket ;  
import android.content.Intent ;  
import android.graphics.Color ;  
import android.os.Bundle ;  
import android.support.v7.app.AppCompatActivity ;  
import android.view.MotionEvent ;  
import android.view.View ;  
import android.widget.Button ;  
importer android.widget.Toast ;  
import java.io.ByteArrayOutputStream ;  
import java.io.IOException ;  
import java.io.OutputStream ;  
import java.util.Set ;  
import java.util.UUID ;
```

```
La classe publique MainActivity étend AppCompatActivity {  
    // Version HC06  
    private final Chaîne DEVICE_ADDRESS = " 98: D3: 31: FC: 28: 97 " ; // Adresse MAC du  
    module Bluetooth  
    privé finale UUID PORT_UUID = UUID . fromString ( "  
00001101-0000-1000-8000-00805F9B34FB " );
```



```

// adafruit

/* private final Chaîne DEVICE_ADDRESS = "E0: 57: BE: 0C: DB: 6C"; // Adresse MAC du
module Bluetooth
    UUID final privé PORT_UUID = UUID.fromString
("00001101-0000-1000-8000-00805f9b34fb");
    // ou 00000000-0000-1000-8000-00805f9b34fb
*/
péphérique BluetoothDevice privé ;
prise BluetoothSocket privée ;
private OutputStream outputStream ;

Button forward_btn, forward_left_btn, forward_right_btn, reverse_btn,
bluetooth_connect_btn;

Commande de chaîne ; // Variable de chaîne qui va stocker la valeur à transmettre au
module Bluetooth

@SuppressLint ( " ClickableViewAccessibility " )
@Passer outre
void protégé onCreate ( Bundle savedInstanceState ) {
    super . onCreate ( savedInstanceState );
    setContentView ( R . layout . activity_main );

outputStream = new ByteArrayOutputStream ( 1024 );

// déclaration des variables de bouton
forward_btn = ( Bouton ) findViewById ( R . id . forward_btn );
forward_btn . setBackgroundColor ( couleur . GRIS );

```



```

forward_left_btn = ( Bouton ) findViewById ( R . id . forward_left_btn);
forward_left_btn . setBackgroundColor ( couleur . GRIS );
forward_right_btn = ( Bouton ) findViewById ( R . id . forward_right_btn);
forward_right_btn . setBackgroundColor ( couleur . GRIS );
reverse_btn = ( Bouton ) findViewById ( R . id . reverse_btn);
reverse_btn . setBackgroundColor ( couleur . GRIS );
bluetooth_connect_btn = ( Bouton ) findViewById ( R . id . bluetooth_connect_btn);
bluetooth_connect_btn . setBackgroundColor ( couleur . GRIS );

// Code OnTouchListener pour le bouton Suivant (appui long sur le bouton)
forward_btn . setOnTouchListener ( nouveau View . OnTouchListener () {

    @Passer outre

    boolean public onTouch ( Voir v , événement MotionEvent ) {

        si ( événement . getAction () == MotionEvent . ACTION_DOWN ) //
MotionEvent.ACTION_DOWN est lorsque vous maintenez un bouton vers le bas
        {

            commande = " 1 " ;

            essayer

            {

                outputStream . écrire (commande . getBytes ()) ; // transmet la valeur de la
commande au module Bluetooth
            }

            catch ( IOException e )

            {

                e . printStackTrace ();
            }
        }
    }
}

```



```

    }

    else si (event . getAction () == MotionEvent . ACTION_UP ) //
MotionEvent.ACTION_UP est lorsque vous relâchez un bouton
    {

        commande = " 5 " ;

        essayer

        {

            outputStream . écrire (commande . getBytes ());

        }

        catch ( IOException e)

        {

            e . printStackTrace ();

        }

    }

    return false ;

}

});

// Code OnTouchListener pour le bouton de marche arrière (appui long sur le bouton)
reverse_btn . setOnTouchListener ( nouveau View . OnTouchListener () {

    @Passer outre

    publique booléen OnTouch ( Voir v , MotionEvent événement )

    {

        if (event . getAction () == MotionEvent . ACTION_DOWN )

        {

            commande = " 3 " ;

            essayer

            {

                outputStream . écrire (commande . getBytes ());

            }

            catch ( IOException e)

```



```

    {
        e . printStackTrace ();
    }
}

else if (event . getAction () == MotionEvent . ACTION_UP )
{
    commande = " 5 " ;
    essayer
    {
        outputStream . écrire (commande . getBytes ());
    }
    catch ( IOException e )
    {
    }
}

return false ;
}

});

// Code OnTouchListener pour le bouton avant gauche (bouton appui long)
forward_left_btn . setOnTouchListener ( nouveau View . OnTouchListener () {
    @Passer outre
    publique booléen OnTouch ( Voir v , MotionEvent événement )
    {
        if (event . getAction () == MotionEvent . ACTION_DOWN )
        {
            commande = " 2 " ;
            essayer
            {
                outputStream . écrire (commande . getBytes ());
            }
        }
    }
}

```



```

        }

        catch ( IOException e)
        {
            e . printStackTrace ();
        }
    }

    else if (event . getAction () == MotionEvent . ACTION_UP )
    {
        commande = " 5 " ;
        essayer
    {
        outputStream . écrire (commande . getBytes ());
    }
    catch ( IOException e)
    {
    }
}

return false ;
}

});

// Code OnTouchListener pour le bouton avant droit (bouton appui long)

forward_right_btn . setOnTouchListener ( nouveau View . OnTouchListener () {

    @Passer outre

    publique booléen OnTouch ( Voir v , MotionEvent événement )
    {
        if (event . getAction () == MotionEvent . ACTION_DOWN )
        {
            commande = " 4 " ;

```



```

    essayer

    {

        outputStream . écrire (commande . getBytes ());

    }

    catch ( IOException e)

    {

        e . printStackTrace ();

    }

}

else if (event . getAction () == MotionEvent . ACTION_UP )

{

    commande = " 5 " ;

    essayer

    {

        outputStream . écrire (commande . getBytes ());

    }

    catch ( IOException e)

    {

        e . printStackTrace ();

    }

}

return false ;

}

});

// Bouton qui connecte l'appareil au module Bluetooth lorsqu'il est pressé

bluetooth_connect_btn . setOnClickListener ( nouveau View . OnClickListener () {

    @Passer outre

    public void onClick ( Voir v ) {

```



```

    if (BTinit ())
    {
        BTconnect ();
    }
}

// Initialise le module Bluetooth

Booléen public BTinit ()
{
    booléen trouvé = faux ;

    BluetoothAdapter bluetoothAdapter = BluetoothAdapter . getDefaultAdapter ();

    si ( ! bluetoothAdapter . isEnabled ()) // vérifie si Bluetooth est activé. Sinon, le
    programme demandera la permission à l'utilisateur pour l'activer
    {

        Intention enableAdapter = nouvelle intention ( BluetoothAdapter .
        ACTION_REQUEST_ENABLE );
        startActivityForResult (enableAdapter, 1234 );
        essayer

        {

            Discussion . dormir ( 50 );

        }

        catch ( InterruptedException e)

        {

            e . printStackTrace ();

        }

    }
}

```



```

Définissez < BluetoothDevice > bondedDevices = bluetoothAdapter . getBondedDevices
();

si (bondedDevices . isEmpty ()) // chèques pour les périphériques Bluetooth associés
{
    Toast . maketext (getApplicationContext (), " S'il vous plaît coupler le premier
dispositif " , Toast . LENGTH_SHORT ) . montrer();
}
autre
{
    pour ( itérateur BluetoothDevice : bondedDevices)
    {
        si (iterator . getAddress () . equals ( DEVICE_ADDRESS ))
        {
            device = itérateur;
            trouvé = vrai ;
            rompre ;
        }
    }
    retour trouvé;
}

booléen public BTconnect ()
{
    booléen connecté = vrai ;
    essayer
    {
        socket = périphérique . createRfcommSocketToServiceRecord ( UUID : fromString ( "
00001101-0000-1000-8000-00805F9B34FB " )); // Crée un socket pour gérer la connexion
sortante
    }
}

```



```

socket . relier();

Toast . maketext (getApplicationContext (), " Connexion à un périphérique Bluetooth
avec succès ", Toast . LENGTH_LONG ) . montrer();
}

catch ( IOException e)

{

e . printStackTrace ();

connecté = faux ;

}

si (connecté)

{

essayer

{

outputStream = socket . getOutputStream () // obtient le flux de sortie de la
socket
}

catch ( IOException e)

{

e . printStackTrace ();

}

}

retour connecté;
}

@Passer outre

void protégé onStart ()

{
super . onStart ();
}
}

```



```
***** android_interface.xml *****
```

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:background="@drawable/color"
    tools:context="com.example.neotxt.projet.MainActivity">

    <Button
        android:id="@+id/forward_btn"
        android:layout_width="125dp"
        android:layout_height="80dp"
        android:layout_alignStart="@+id/bluetooth_connect_btn"
        android:layout_below="@+id/imageView"
        android:layout_marginTop="24dp"
        android:text="@string/avancer" />

    <Button
        android:id="@+id/forward_left_btn"
        android:layout_width="125dp"
        android:layout_height="80dp"
        android:layout_below="@+id/forward_btn"
        android:layout_marginEnd="20dp"
        android:layout_toStartOf="@+id/forward_btn"
        android:text="@string/avancer_gauche" />

    <Button
        android:id="@+id/forward_right_btn"
        android:layout_width="125dp"
```



```
    android:layout_height="80dp"
    android:layout_below="@+id/forward_btn"
    android:layout_marginStart="15dp"
    android:layout_toEndOf="@+id/forward_btn"
    android:text="@string/avancer_droite" />

<Button
    android:id="@+id/reverse_btn"
    android:layout_width="125dp"
    android:layout_height="80dp"
    android:layout_alignStart="@+id/bluetooth_connect_btn"
    android:layout_below="@+id/bluetooth_connect_btn"
    android:layout_marginTop="8dp"
    android:text="@string/reculer" />

<Button
    android:id="@+id/bluetooth_connect_btn"
    android:layout_width="125dp"
    android:layout_height="80dp"
    android:layout_below="@+id/forward_btn"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="11dp"
    android:text="@string/bluetooth" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="600dp"
    android:layout_height="100dp"
    android:layout_alignEnd="@+id/forward_right_btn"
    android:layout_alignParentTop="true"
    android:src="@drawable/mainbot" />

<ImageView
    android:id="@+id/imageView2"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_below="@+id/reverse_btn"
```



```
    android:layout_centerHorizontal="true"
    android:layout_marginTop="44dp"
    android:src="@drawable/winky" />

</RelativeLayout>
```

PROGRAMME IHM :

```
***** myudp.h *****
#ifndef MYUDP_H
#define MYUDP_H

#include <QObject>
#include <QUdpSocket>
#include <iostream>
#include <QString>

using std::string;

class MyUDP : public QObject
{
    Q_OBJECT
public:
    explicit MyUDP(QObject *parent = nullptr);
    void HelloUDP (char data);

//explicit MyUDP(QObject *parent = 0);
signals:
public slots:
    void readyRead();

private:
    QUdpSocket *socket;
```



```

};

#endif // MYUDP_H

#ifndef WIDGET_H

*****widget.h*****



#define WIDGET_H

#include <QWidget>
#include <QPushButton>
#include <QDebugStateSaver>
#include <QNetworkReply>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QUrl>
#include <QtNetwork>
#include <QLabel>
#include <QMessageBox>
#include <QByteArray>
#include <QLabel>
#include <QTimer>

#include "myudp.h"

class MaFenetre : public QWidget
{
    Q_OBJECT
    QPushButton *BoutonHaut, *BoutonBas, *BoutonGauche, *BoutonDroit, *BoutonArret;

public:
    MaFenetre(QWidget *parent = 0);

private:
    QLabel *label;
}

```



```

bool erreurTrouvee;
QNetworkAccessManager *m;
 QTimer *timer;
 MyUDP client;

public slots:
 void chargement();
 void afficher();
 void messageErreur(QNetworkReply::NetworkError);

private slots:
//declaration des boutons qu'on vas utiliser
 void boutonHaut();
 void boutonBas();
 void boutonDroit();
 void boutonGauche();
 void boutonArret();
};

#endif // WIDGET_H

***** main.cpp *****

#include "widget.h"
#include "myudp.h"

#include <QCoreApplication>
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MaFenetre w;
    w.show();

    return a.exec();
}

```



```
***** myudp.cpp *****
```

```
#include "myudp.h"

MyUDP::MyUDP(QObject *parent) : QObject(parent)
{
    // create a QUDP socket
    socket = new QUdpSocket(this);

    socket->bind(QHostAddress::LocalHost, 2100);

    connect(socket, SIGNAL(readyRead()), this, SLOT(readyRead()));
}

void MyUDP::HelloUDP(char datai)
{
    QByteArray Data;
    Data.append(datai);

    socket->writeDatagram(Data, QHostAddress::("192.168.0.177") , 2100);
    //socket->writeDatagram(Data, QHostAddress("192.168.43.108"), 2100);
}

void MyUDP::readyRead()
{
    // when data comes in
    QByteArray buffer;
    buffer.resize(socket->pendingDatagramSize());

    QHostAddress sender;
    quint16 senderPort;

    socket->readDatagram(buffer.data(), buffer.size(), &sender, &senderPort);

    qDebug() << "Message port: " << senderPort;
    qDebug() << "Message: " << buffer;
```



```

}

***** widget.cpp *****

#include "widget.h"
#include "ui_widget.h"

MaFenetre::MaFenetre(QWidget *parent)
: QWidget(parent), m(new QNetworkAccessManager())
{
    m->setParent(this);
    erreurTrouvee = false;

    label=new QLabel(this);//création du label qui contiendra l'image
    label->move(40, 50);//on le déplace

    timer = new QTimer(this);//création du timer qui en appellent le slot créera une vidéo
    timer->start(40);
    connect(timer, SIGNAL(timeout()), this, SLOT(changement));//on le connecte

    BoutonHaut = new QPushButton("bouton haut", this);//creation du bouton haut
    BoutonHaut->setGeometry(999, 580, 80, 70); // Dimensions du bouton. // x , y , size ? , size ?
    connect(BoutonHaut, SIGNAL(clicked()), this, SLOT(boutonHaut()));

    BoutonBas = new QPushButton("bouton bas", this);//creation du bouton haut
    BoutonBas->setGeometry(999, 720, 80, 70); // Dimensions du bouton.
    connect(BoutonBas, SIGNAL(clicked()), this, SLOT(boutonBas()));

    BoutonDroit = new QPushButton("bouton droit", this);//creation du bouton droit
    BoutonDroit->setGeometry(1080, 650, 80, 70); // Dimensions du bouton.
    connect(BoutonDroit, SIGNAL(clicked()), this, SLOT(boutonDroit()));

    BoutonGauche = new QPushButton("bouton gauche", this);//creation du bouton gauche
    BoutonGauche->setGeometry(920, 650, 80, 70); // Dimensions du bouton.
    connect(BoutonGauche, SIGNAL(clicked()), this, SLOT(boutonGauche()));

    BoutonArret = new QPushButton("Arret", this);//creation du bouton quitter
    BoutonArret->setGeometry(999, 650, 80, 70); // Dimensions du bouton.

```



```

connect(BoutonArret, SIGNAL(clicked()), this, SLOT(boutonArret())); // Connecter les signaux et les slots
}

void MaFenetre::chargement()//slot qui charge l'image tout les 40 millièmes de seconde
{
    const QUrl url = QUrl("http://192.168.43.1:8080/shot.jpg"); //URL de l'IMAGE et nom de la page web qui affiche
    l'image
    //Adresse de la came du lycée 192.168.0.90
    const QNetworkRequest requete(url); //On crée notre requête

    QNetworkReply *r = m->get(requete);

    QObject::connect(r,
    static_cast<void(QNetworkReply::*)(QNetworkReply::NetworkError)>(&QNetworkReply::error),this,&MaFenetre::m
    essageErreur);//connection au message d'erreur si il y en à une

    QObject::connect(r, SIGNAL(finished()), this, SLOT(afficher()));//Dès que l'image est chargé on l'affiche en
    mettant à jour le QLabel
}

void MaFenetre::afficher()
{

    QNetworkReply *r = qobject_cast<QNetworkReply*>(sender());//on récupère l'image

    QByteArray data = r->readAll();//on la met dans un QByteArray

    QImage image;
    image.loadFromData(data);//on la charge car on ne peut pas l'afficher directement ( format de compression
    comme jpg...)

    label->setFixedSize(image.size());
    label->setPixmap(QPixmap::fromImage(image));//On rafraichi le QLabel avec la nouvel image

    r->deleteLater();
}

void MaFenetre::messageErreur(QNetworkReply::NetworkError)

```



```

{
    // On récupère la reply
    QNetworkReply *r = qobject_cast<QNetworkReply*>(sender());

    // Si une erreur a déjà été trouvée, on delete la reply et on ne fait rien d'autre
    if ( erreurTrouvee ){
        r->deleteLater();
        return; // Return pour ne pas traiter l'erreur
    }

    // On indique qu'il y a eu une erreur
    erreurTrouvee = true;

    QMessageBox::critical(this, "Erreur", "Erreur lors du chargement de l'image. Vérifiez votre connexion ou vérifiez que votre caméra est branché.<br /><br /> Code de l'erreur : <br /><em>" + r->errorString() + "</em>");
    close();

    r->deleteLater();
}

//fonction des boutons
void MaFenetre::boutonHaut()
{
    qDebug() << "Bouton Haut";
    client>HelloUDP('1');
    //emit signal(1);
}

void MaFenetre::boutonBas()
{
    qDebug() << "Bouton bas";
    client>HelloUDP('2');
}

void MaFenetre::boutonGauche()
{
    qDebug() << "Bouton gauche";
}

```



```

client>HelloUDP('3');

}

void MaFenetre::boutonDroit()
{
    qDebug() << "Bouton droit";
    client>HelloUDP('4');
}

void MaFenetre::boutonArret()
{
    qDebug() << "Arreter";
    client>HelloUDP('5');
}

```

Programme module wifi :

```

***** wifi.cpp *****
#include "wifi.h"
#include <SPI.h>
#define CS 15 // définit la ligne de sélection de puce pour le contrôle manuel

void setup () {
    // mettez votre code d'installation ici, pour l'exécuter une fois:
    En série. commencer ( 115200 );
    En série. println ( " coucou tua " );
    wifi_connection(); // lance le serveur

    pinMode (CS, OUTPUT); // configure la ligne en sortie
    digitalWrite (CS, HIGH); // Initialise la ligne CS à HIGH
    SPI. commencer (); // Initialise le module SPI -> configure les lignes MOSI, MISO et
    CLOCK

    /* Configure le bus SPI comme suit

```



1. Vitesse du bus SPI = 1 MHz
 2. Data Out = Du bit MSB ---> Au bit LSB
 3. Mode de données = SPI MODE0 * /
 SPI.beginTransaction (SPISettings (1000000 , MSBFIRST, SPI_MODE0));
 }

Boucle vide ()
 {
 // Serial.println (get_wifi_message());
 digitalWrite (CS, LOW); // Tirer CS Line Low
 SPI.transfert (get_wifi_messageChar ()); // Envoie un octet (0x02) à l'esclave ie Arduino UNO
 delayMicroseconds (10); // Donne un peu de temps à l'esclave pour traiter / faire quelque chose avec les données reçues
 digitalWrite (CS, HIGH); // Tirer CS Line High
 }

***** wifi.h *****

```
# ifndef WIFI_H
# define WIFI_H

# include < arduino.h >

# include " ESP8266WiFi.h "

# include < WiFiUdp.h >

# include < WiFiClient.h >

# include < ESP8266WebServer.h >

static const char * ssid = " SosoMobile " ; // id et mdp de connexion

static const char * mot de passe = " mainbot123 " ; //
```





```
WiFiUDP statique UDPTTestServer;           //  objet UDP

static unsigned int UDPPort = 2100 ;    //  port de com

static const int packetSize = 20 ;      //  taille des messages (on peut le remplacer par un
define)

static byte packetBuffer [packetSize]; //  tableau de réception des données

extern Chaîne wifi_connection ();

extern Chaîne get_wifi_message (); // get msg wifi

extern char get_wifi_messageChar ();

#endif

***** main *****

#include " wifi.h "
#include < SPI.h >

#define CS 15 // définit la ligne de sélection de puce pour le contrôle manuel

void setup () {

// mettez votre code d'installation ici, pour l'exécuter une fois:

En série. commencer ( 115200 );

En série. println ( " coucou tua " );

wifi_connection ();                      // lance le serveur

pinMode (CS, OUTPUT);                   // configure la ligne en sortie

digitalWrite (CS, HIGH);                // Initialise la ligne CS à HIGH
```



```

SPI.begin(); // Initialise le module SPI -> configure les lignes MOSI, MISO et
CLOCK

/* Configure le bus SPI comme suit

1. Vitesse du bus SPI = 1 MHz
2. Data Out = Du bit MSB --> Au bit LSB
3. Mode de données = SPI MODE0 */

SPI.beginTransaction ( SPISettings ( 1000000 , MSBFIRST, SPI_MODE0));

}

Boucle vide ()

{
// Serial.println (get_wifi_message ());

digitalWrite (CS, LOW); // Tirer CS Line Low

SPI.transfer ( get_wifi_messageChar () ); // Envoie un octet (0x02) à l'esclave ie
Arduino UNO

delayMicroseconds ( 10 ); // Donne un peu de temps à l'esclave pour traiter / faire
quelque chose avec les données reçues

digitalWrite (CS, HIGH); // Tirer CS Line High
}

```

