Étude et réalisation – Semestre 3

Mesure de l’angle : Accéléromètre et gyroscope

IUT De Nantes

BIZON Alexis; METAYER Simon

PEI

2017

Étude et réalisation – Semestre 3

Mesure de l’angle : Accéléromètre et gyroscope

Table des matières :

[1. Table des Illustrations : 2](#_Toc472441111)

[2. Introduction, Cahier des charges : 3](#_Toc472441112)

[3. Étude du filtre : 4](#_Toc472441113)

[3.1. Introduction : 4](#_Toc472441114)

[3.2. Représentation du filtre passe-bas : 4](#_Toc472441115)

[3.3. Réponse du filtre passe-bas à des sinus : 5](#_Toc472441116)

[3.4. Représentation du filtre passe-haut : 7](#_Toc472441117)

[3.5. Réponse du filtre passe-haut à des sinus : 8](#_Toc472441118)

[3.6. Réponse Impulsionnelle du filtre passe-bas : 10](#_Toc472441119)

[3.7. Réponse impulsionnelle du filtre passe-haut : 11](#_Toc472441120)

[3.8. Représentation du filtre global : 12](#_Toc472441121)

[3.9. Réponse du filtre global à des signaux quelconques : 13](#_Toc472441122)

[3.10. Représentation du filtre global en code Matlab : 15](#_Toc472441123)

[3.11. Réponse du filtre global en code Matlab à des signaux quelconques : 16](#_Toc472441124)

[4. Mise en œuvre des deux capteurs séparés : 17](#_Toc472441125)

[4.1. Introduction : 17](#_Toc472441126)

[4.2. Mise en œuvre de l’accéléromètre : 18](#_Toc472441127)

[4.3. Mise en œuvre du gyroscope : 20](#_Toc472441128)

[4.4. Combinaison des deux solutions : 21](#_Toc472441129)

[5. Mise en œuvre de la centrale inertielle : 21](#_Toc472441130)

[5.1. Présentation : 21](#_Toc472441131)

[5.2. Code étudié : 22](#_Toc472441132)

[5.3. Modification du code : 25](#_Toc472441133)

[5.4. Tests et conclusion : 28](#_Toc472441134)

[6. Asservissement du gyropode : 29](#_Toc472441135)

[6.1. Commande des moteurs : 29](#_Toc472441136)

[6.2. Asservissement des moteurs en vitesse et position : 31](#_Toc472441137)

[6.3. Construction du gyropode 33](#_Toc472441138)

[6.4 Passage du gyropode en temps réel 36](#_Toc472441139)

[7. Conclusion : 39](#_Toc472441140)

[8. Remerciements : 39](#_Toc472441141)

[9. Fourni avec le compte rendu : 39](#_Toc472441142)

# Table des Illustrations :

[Figure 1 : Représentation graphique du filtre passe-bas 4](#_Toc472441143)

[Figure 2 : Réponse à un sinus de 0.032Hz du filtre passe-bas 5](#_Toc472441144)

[Figure 3 : Réponse à un sinus de 0.32Hz du filtre passe-bas 6](#_Toc472441145)

[Figure 4 : Réponse à un sinus de 3.2Hz du filtre passe-bas 7](#_Toc472441146)

[Figure 5 : Représentation graphique du filtre passe-haut 7](#_Toc472441147)

[Figure 6 : Réponse à un sinus de 0.032Hz du filtre passe-haut 8](#_Toc472441148)

[Figure 7 : Réponse à un sinus de 0.32Hz du filtre passe-haut 9](#_Toc472441149)

[Figure 8 : Réponse à un sinus de 3.2Hz du filtre passe-haut 10](#_Toc472441150)

[Figure 9 : Réponse impulsionnelle du filtre passe-bas 11](#_Toc472441151)

[Figure 10 : Réponse impulsionnelle du filtre passe-haut 11](#_Toc472441152)

[Figure 11 : Représentation graphique du filtre global 12](#_Toc472441153)

[Figure 12 : réponse au filtre global avec ses paramètres initiales à des signaux quelconques 13](#_Toc472441154)

[Figure 13 : Réponse du filtre global avec un dt de 1e-3 à des signaux quelconques 14](#_Toc472441155)

[Figure 14 : Réponse du filtre global avec k = 0.1 et 1-k=0.9 à des signaux quelconques 14](#_Toc472441156)

[Figure 15 : Représentation du filtre global en code Matlab 15](#_Toc472441157)

[Figure 16 : simulation du filtre global en code matlab 16](#_Toc472441158)

[Figure 17 : Mesure de l’angle avec deux axes 18](#_Toc472441159)

[Figure 18 : Relevé du signal filtré et du signal issu du gyroscope 28](#_Toc472441160)

[Figure 19 : Relevé du signal filtré et du signal issu de l’accéléromètre 28](#_Toc472441161)

[Figure 20 : Synoptique asservissement du gyropode 29](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441162)

[Figure 21 : Schéma électronique de la commande des moteurs 29](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441163)

[Figure 22 : Schéma bloc de l’asservissement des moteurs 31](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441164)

[Figure 23 : Liste du matériel pour la réalisation du gyropode 33](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441165)

[Figure 25 : Schéma d’implantation de la carte de pilotage du gyropode 33](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441166)

[Figure 25 : Réalisation finale du gyropode 33](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441167)

[Figure 26 : Schéma électronique de la carte de pilotage du gyropode 34](#_Toc472441168)

[Figure 27 : Affichage LCD 34](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441169)

[Figure 28 : Variation V(angle) théorique en x² 35](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441170)

[Figure 29 : Variation V(angle) Reelle en abs(cos(x/8)) 35](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441171)

[Figure 30 : Synoptique de Trampoline 36](file:///C:\Users\simon\Desktop\Revue_de_projet\revue_de_projet.docx#_Toc472441172)

# Introduction, Cahier des charges :

Pour le module d’Etude et Réalisation 3, nous allons étudier les aspects électroniques et calculatoires liés au calcul de l’angle d’un gyropode. Pour ce faire, nous aurons à notre disposition une centrale inertielle équipée d’un accéléromètre et d’un gyroscope que nous devrons utiliser en complémentarités suivant leurs défauts intrinsèques. Le programme devra être implanté dans une carte de type Arduino, pour ensuite être couplé à un système d’asservissement moteur réalisé par un autre groupe pour enfin gérer complètement un gyropode. Pour atteindre cet objectif, nous aurons à passer par plusieurs étapes intermédiaires.

Cette étude à réaliser en binôme comporte 3 parties principales 1 partie optionnelle et 1 partie rajoutée par nos soins :

1. Une étude théorique d’un filtre, composé d’un passe bas et d’un passe haut complémentaires, à l’aide de logiciels de simulation tels que Matlab et Simulink, nécessaire pour obtenir la valeur précise d’un angle par rapport à une référence avec un accéléromètre et un gyroscope en fonction de leurs défauts intrinsèques
2. La mise en œuvre des capteurs séparés : un accéléromètre [ADXL 335](http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf) et un gyroscope [ENC-03R](https://www.elecrow.com/download/ENC-03.pdf) couplés à une carte [Arduino Due](https://www.arduino.cc/en/Main/ArduinoBoardDue) pour une étude détaillée de leur fonctionnement et une première ébauche de code de récupération d’angle.
3. Une étude de documentation approfondie sur un appareil plus complet : la centrale inertielle [MPU 6050](http://playground.arduino.cc/Main/MPU-6050) pour obtenir une valeur d’angle plus précise qu’avec les capteurs séparés précédemment utilisés.

La mise en œuvre de la centrale inertielle ([MPU 6050](http://playground.arduino.cc/Main/MPU-6050)) : accéléromètre et gyroscope, avec implémentation du filtrage des données nécessaire dans le microcontrôleur intégré à la centrale et récupération des valeurs par bus I2C sur une carte Arduino adaptée ([Arduino Due](https://www.arduino.cc/en/Main/ArduinoBoardDue) ou [Arduino Mega](https://www.arduino.cc/en/Main/ArduinoBoardMega2560) …)

1. Optionnel : L’insertion physique dans un système complet de cette centrale pour l’asservissement d’un gyropode (normalement avec un autre groupe).
2. Optionnel ++ : Passer tout le code en temps réel avec Trampoline pour plus de sécurité et d’efficacité.

# Étude du filtre :

## Introduction :

Pour étudier les différents filtres, nous allons utiliser le logiciel Matlab avec le module Simulink pour rendre l’étude plus visuelle. Ce logiciel nous permet d’assembler des « boîtes » contenants chacune une fonction (par exemple la fonction de transfert d’un filtre) et ainsi répondre à un cahier des charges. Nous appliquons des stimuli en entrée du système pour étudier les réponses temporelles et fréquentielles en sortie du même système.

## Représentation du filtre passe-bas :

Nous allons étudier les réponses temporelles d’un filtre passe-bas à des sinus de fréquence différentes. Le système est représenté par l’assemblage des fonctions en Figure 1.



Figure 1 : Représentation graphique du filtre passe-bas

La fonction de transfert décrite par le précédent système nous est fournie et est :

Où :

K => le coefficient du filtre

αn => l’angle calculé

αn-1 => l’angle calculé précédemment

αacc, n => l’angle renvoyé par l’accéléromètre.

On retrouve donc dans la partie supérieure du filtre le signal multiplié par 1-k, suivi d’un additionneur qui prend comme seconde variable la sortie précédemment échantillonnée multiplié par k.

A partir de la transformée en Z, de la fréquence d’échantillonnage ( ), et du coefficient k, nous pourrions être capable de définir la fréquence de coupure définit par l’équation précédemment donnée.

Ne sachant pas utiliser la transformée en z, la formule suivante nous est donnée :

Avec k = 0.98, on obtient donc fc = 0.0032 fe

Si fe vaut 1e-2 on obtient fc = 0.32Hz.

## Réponse du filtre passe-bas à des sinus :

### Sinus de 0.032Hz :

On applique maintenant en entré du système un sinus de 0.032Hz soit dix fois moins élevé que la fréquence de coupure précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 2. On n’oublie pas d’initialiser la période d’échantillonnage dt à 1e-2 dans Matlab.

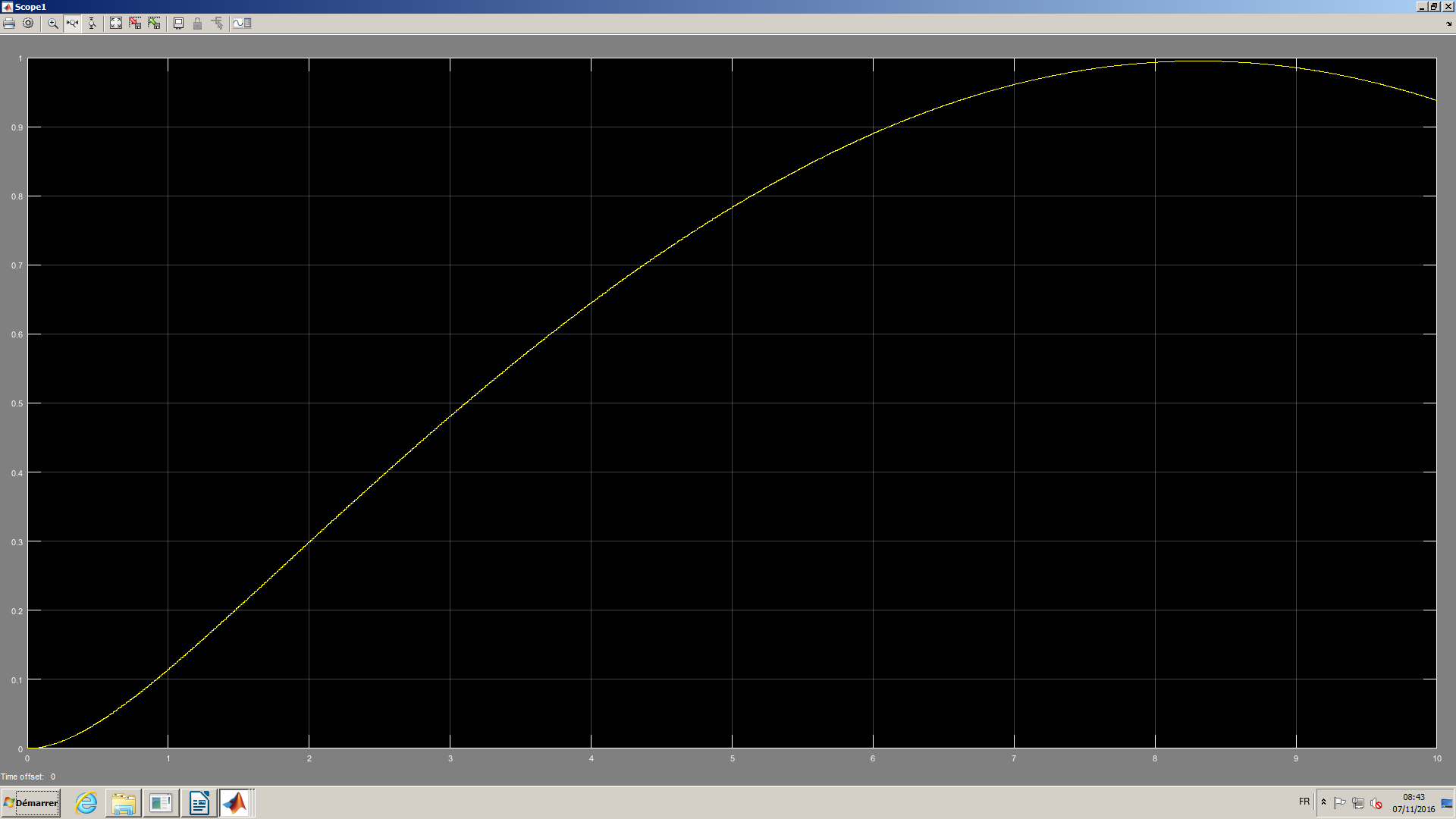


Figure 2 : Réponse à un sinus de 0.032Hz du filtre passe-bas

On observe donc un sinus de 0.032Hz d’amplitude 1V en sortie du système, ce qui correspond exactement au signal appliqué en entré dudit système.

On en déduit donc que pour cette fréquence, le système ne modifie pas du tout le signal, ce qui est normal car nous sommes à une fréquence dix fois plus basse que la fréquence de coupure du filtre.

### Sinus de 0.32Hz :

On applique maintenant en entré du même système un sinus de 0.32Hz qui est la fréquence de coupure du filtre précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 3.

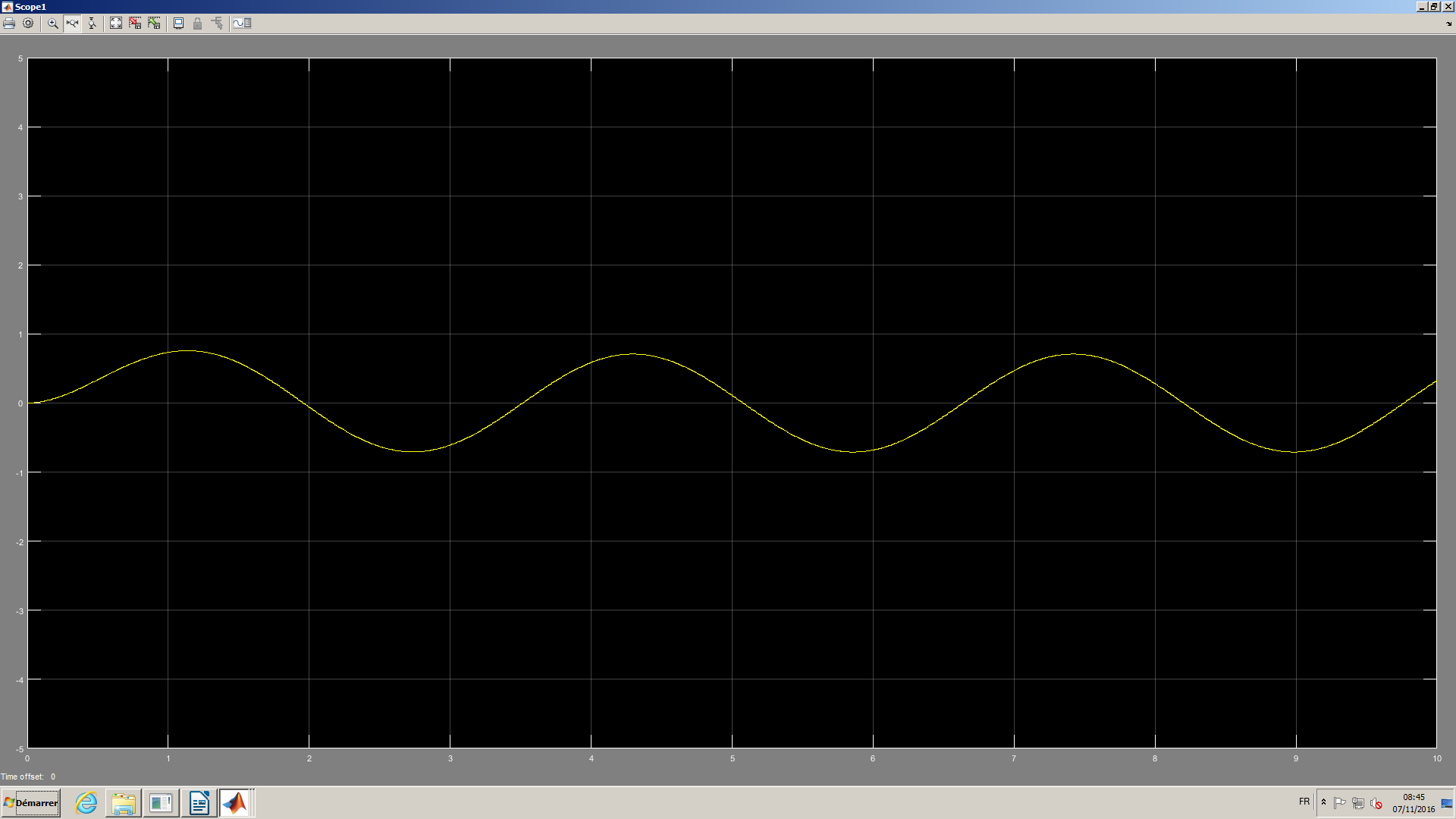


Figure 3 : Réponse à un sinus de 0.32Hz du filtre passe-bas

On observe donc un sinus de 0.32Hz d’amplitude 0.707V en sortie du système, ce qui correspond au signal appliqué en entrée du système atténué de 3dB.

On en déduit donc que pour cette fréquence, le système atténue le signal d’entrée du système de 3dB, ce qui est normal car nous sommes à la fréquence de coupure, et qu’à cette fréquence, un filtre du premier ordre a une atténuation de 3dB.

### Sinus de 3.2Hz :

On applique maintenant en entré du même système un sinus de 3.2Hz qui est dix fois plus grande que la fréquence de coupure du filtre précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 4.

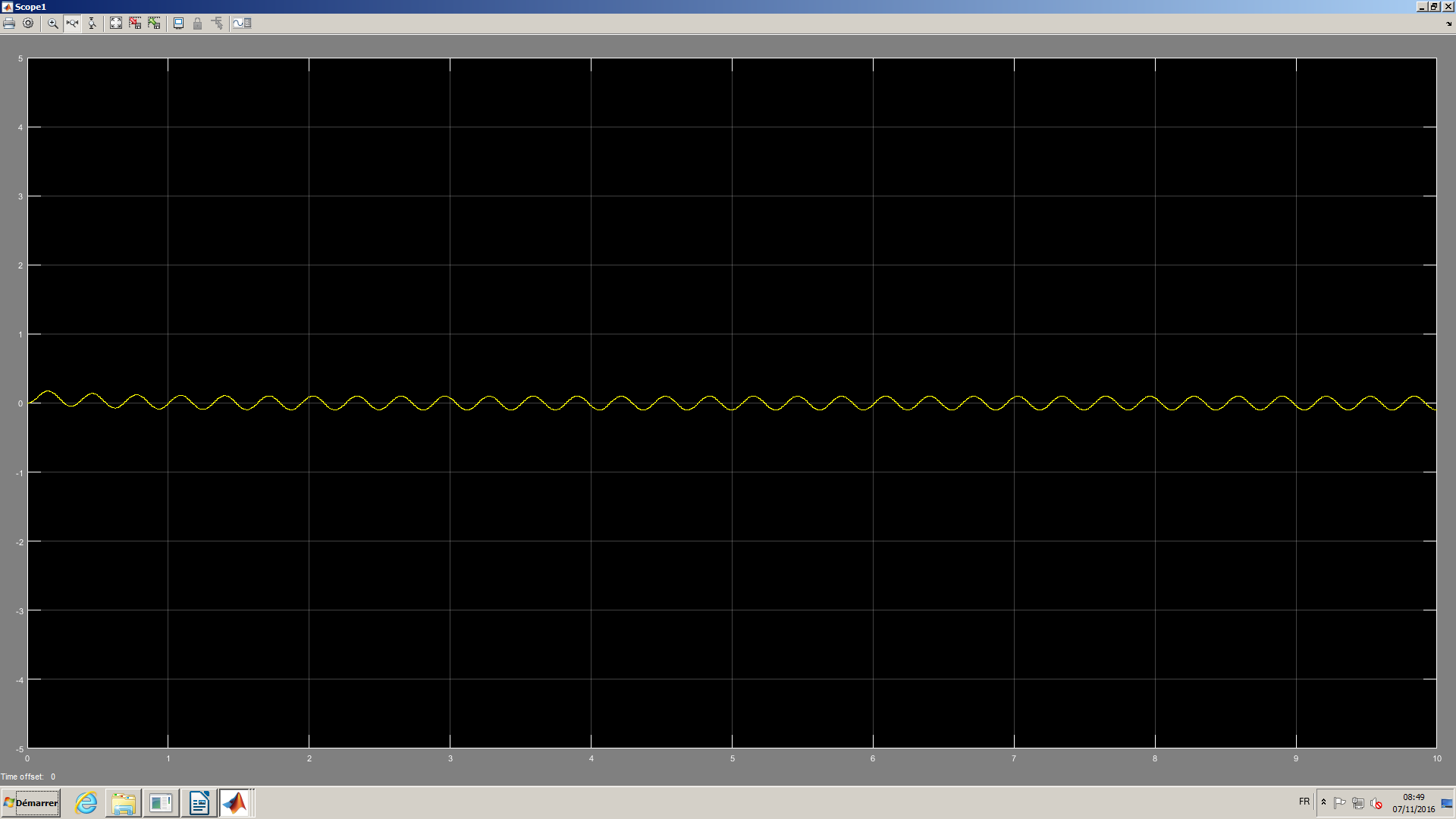


Figure 4 : Réponse à un sinus de 3.2Hz du filtre passe-bas

On observe donc un sinus de 3.2Hz d’amplitude 0.1V en sortie du système, ce qui correspond au signal appliqué en entrée du système divisé par 10.

On en déduit donc que pour cette fréquence, le système atténue le signal d’entrée du système de 20dB, ce qui est normal car nous sommes à une fréquence 10 fois supérieure à la fréquence de coupure, et qu’à cette fréquence, un filtre passe-bas du premier ordre a une atténuation de 20dB.

Ces différentes analyses nous permettent de valider le fonctionnement passe-bas du premier ordre du système numérique qui nous a été proposé d’étudier.

## Représentation du filtre passe-haut :

Nous allons étudier les réponses temporelles d’un filtre passe-haut à des sinus de fréquence différentes. Le système est représenté par l’assemblage des fonctions en Figure 5.

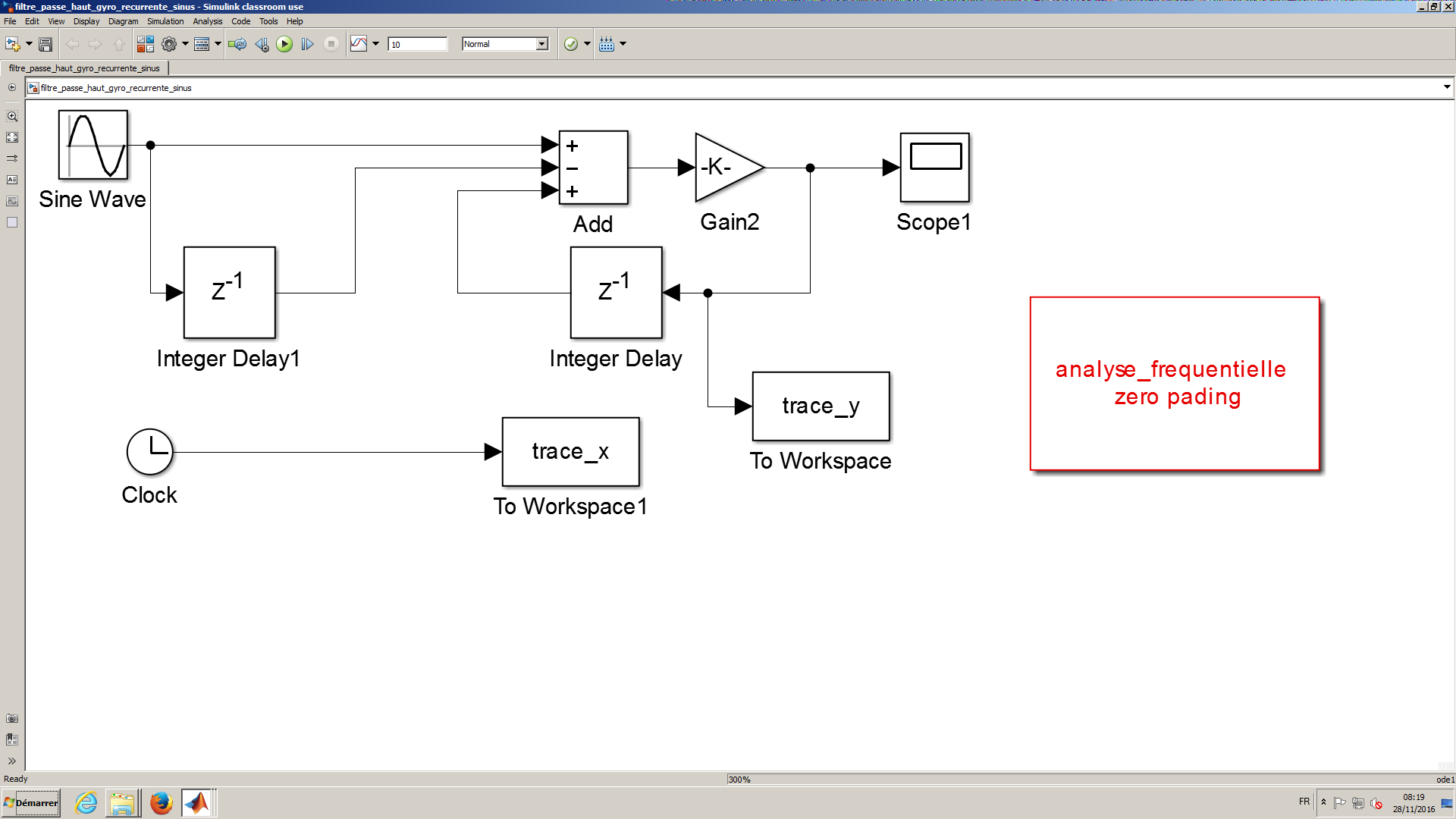


Figure 5 : Représentation graphique du filtre passe-haut

La fonction de transfert décrite par le précédent système nous est fournie et est :

On peut voir le terme comme la différence de l’angle donnée par le gyroscope entre deux instants et, soit. On obtient alors la nouvelle équation :

Où :

K => le coefficient du filtre

αn => l’angle calculé

αn-1 => l’angle calculé précédemment

αgyr, n => l’angle renvoyé par le gyroscope

αgyr, n => l’angle renvoyé par le gyroscope précédemment

A partir des mêmes outils que pour le filtre passe-bas, on peut trouver la fréquence de coupure du filtre. Après calculs, on trouve la même fréquence de coupure que le filtre passe-bas, soit 0.32Hz.

## Réponse du filtre passe-haut à des sinus :

### Sinus de 0.032Hz :

On applique maintenant en entrée du système un sinus de 0.032Hz qui est dix fois plus petite que la fréquence de coupure du filtre précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 6.

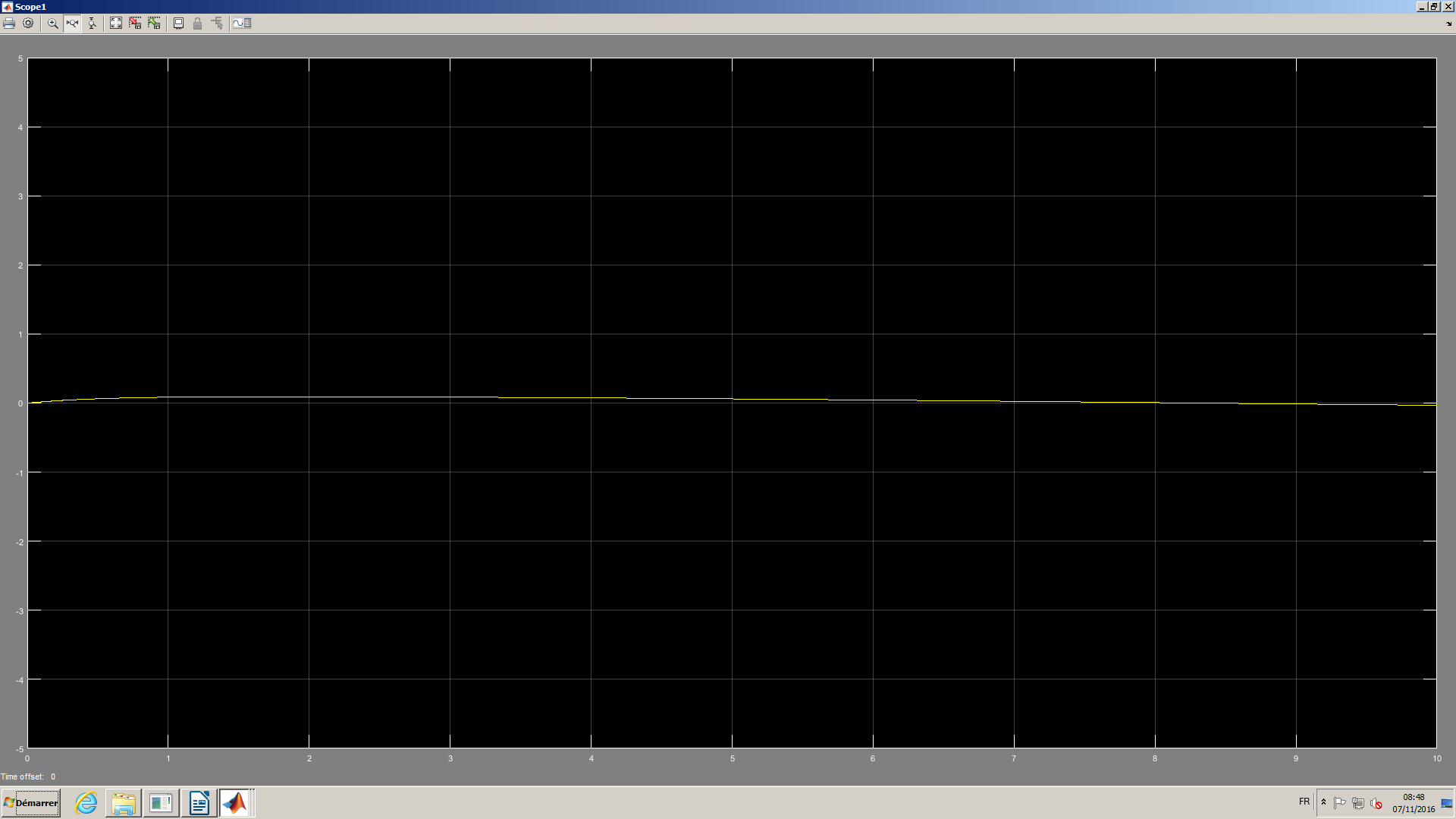


Figure 6 : Réponse à un sinus de 0.032Hz du filtre passe-haut

On observe donc un sinus de 0.032Hz d’amplitude 0.1V en sortie du système, ce qui correspond au signal appliqué en entrée du système divisé par 10.

On en déduit donc que pour cette fréquence, le système atténue le signal d’entrée du système de 20dB, ce qui est normal car nous sommes à une fréquence 10 fois inférieure à la fréquence de coupure, et qu’à cette fréquence, un filtre passe-haut du premier ordre a une atténuation de 20dB.

### Sinus de 0.32Hz :

On applique maintenant en entré du même système un sinus de 0.32Hz qui est la fréquence de coupure du filtre précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 7.

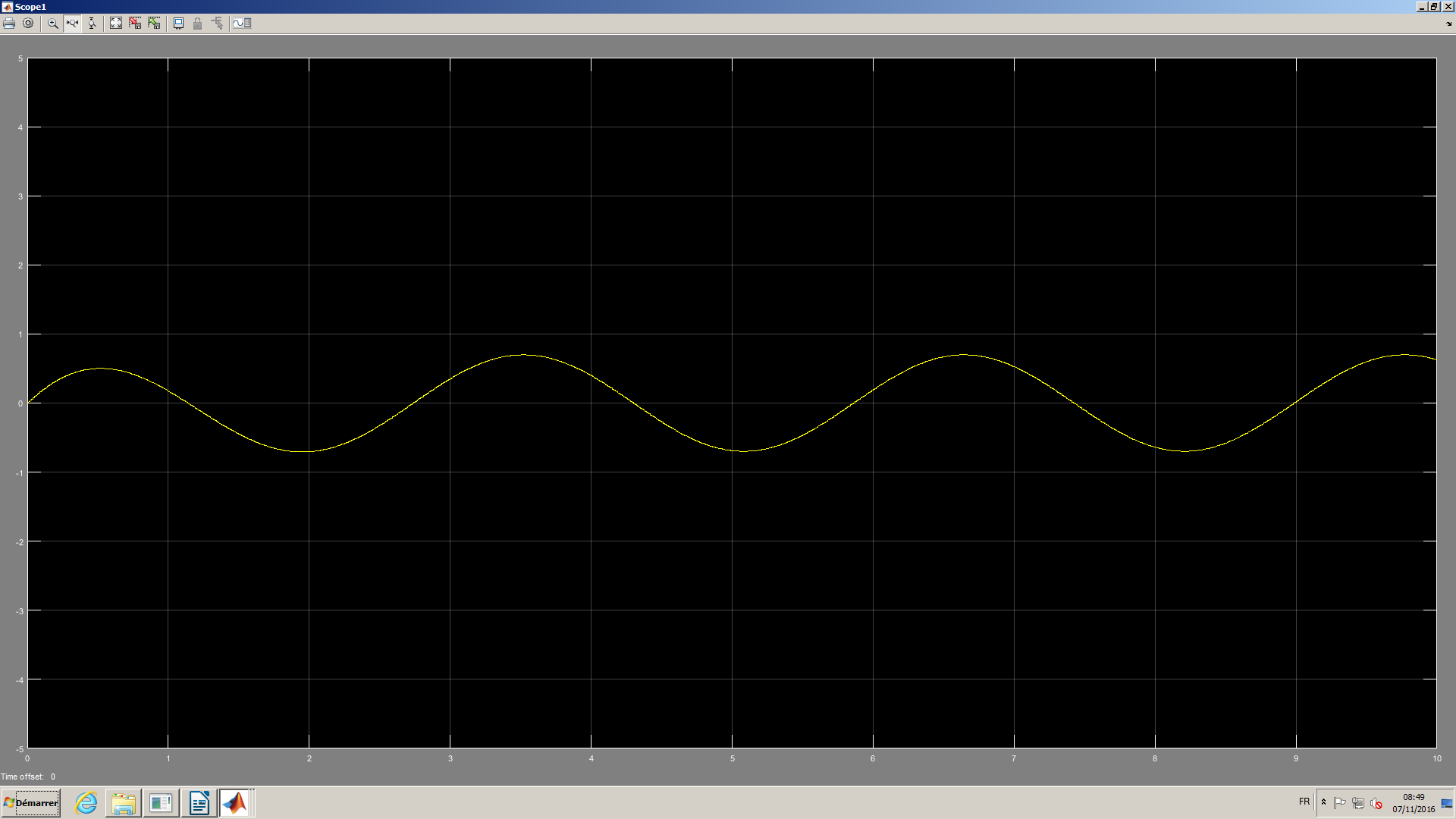


Figure 7 : Réponse à un sinus de 0.32Hz du filtre passe-haut

On observe donc un sinus de 0.32Hz d’amplitude 0.707V en sortie du système, ce qui correspond au signal appliqué en entrée du système atténué de 3dB.

On en déduit donc que pour cette fréquence, le système atténue le signal d’entrée du système de 3dB, ce qui est normal car nous sommes à la fréquence de coupure, et qu’à cette fréquence, un filtre du premier ordre a une atténuation de 3dB.

### Sinus de 3.2Hz :

On applique maintenant en entré du système un sinus de 3.2Hz qui est dix fois plus grande que la fréquence de coupure du filtre précédemment calculée qui était de 0.32Hz. La réponse temporelle du filtre à ce signal est représentée en Figure 8.

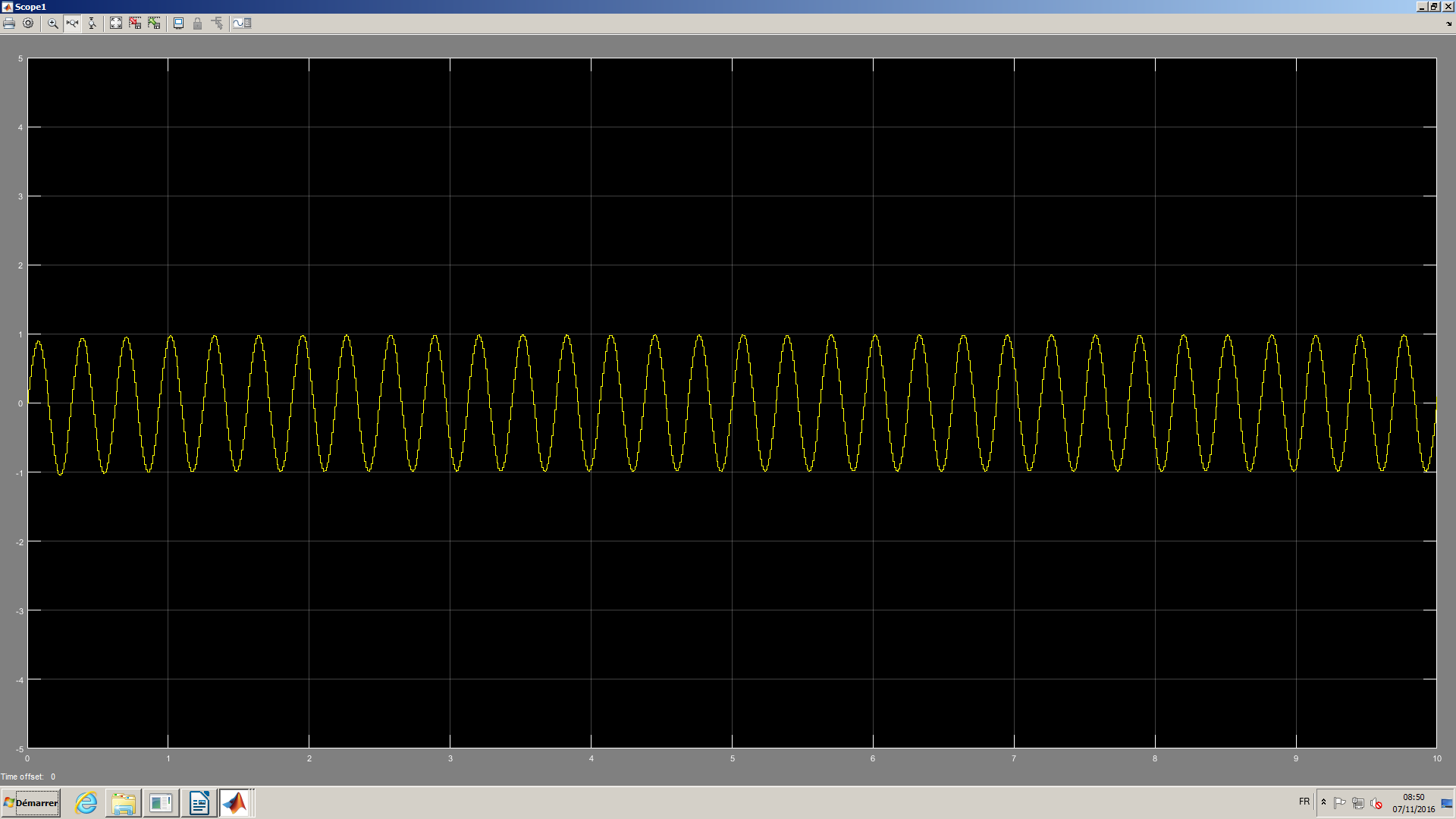


Figure 8 : Réponse à un sinus de 3.2Hz du filtre passe-haut

On observe donc un sinus de 3.2Hz d’amplitude 1V en sortie du système, ce qui correspond exactement au signal appliqué en entré dudit système.

On en déduit donc que pour cette fréquence, le système ne modifie pas du tout le signal, ce qui est normal car nous sommes à une fréquence dix fois plus basse que la fréquence de coupure du filtre.

Ces différentes analyses nous permettent de valider le fonctionnement passe-haut du premier ordre du système numérique qui nous a été proposé d’étudier.

On observe également une parfaite symétrie entre les deux systèmes autour de la fréquence de coupure de 0.32Hz.

## Réponse Impulsionnelle du filtre passe-bas :

On souhaite maintenant obtenir la réponse impulsionnelle du filtre passe-bas. Pour cela, on applique une pseudo impulsion de Dirac en entrée du système (pseudo, car physiquement, une impulsion de Dirac est impossible à réaliser) pour ensuite analyser sa réponse fréquentielle. Cette réponse en fréquence représente le comportement fréquentiel du système. Cette réponse est présentée en Figure 9 pour le filtre passe-bas.

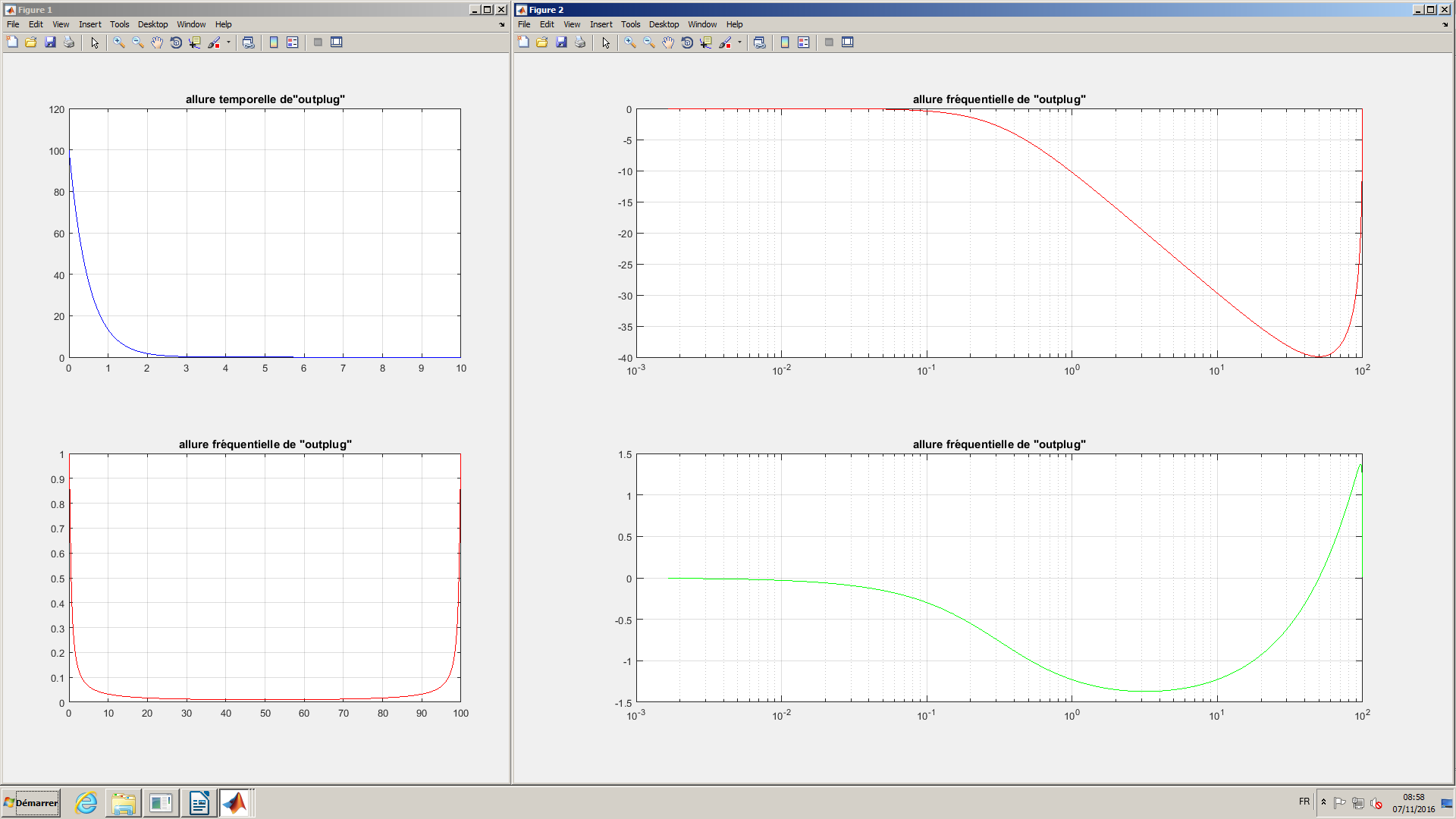


Figure 9 : Réponse impulsionnelle du filtre passe-bas

La courbe en haut à droite représente le comportement fréquentiel en amplitude du système. On remarque qu’elle correspond bien à un système du type passe-bas du premier ordre, sauf sur la fin, qui est dû au pas d’échantillonnage.

## Réponse impulsionnelle du filtre passe-haut :

Pour obtenir la réponse impulsionnelle du filtre passe-haut, on applique la même méthode qu’utilisée précédemment pour analyser le filtre passe-bas. Cette réponse est présentée en Figure 10.

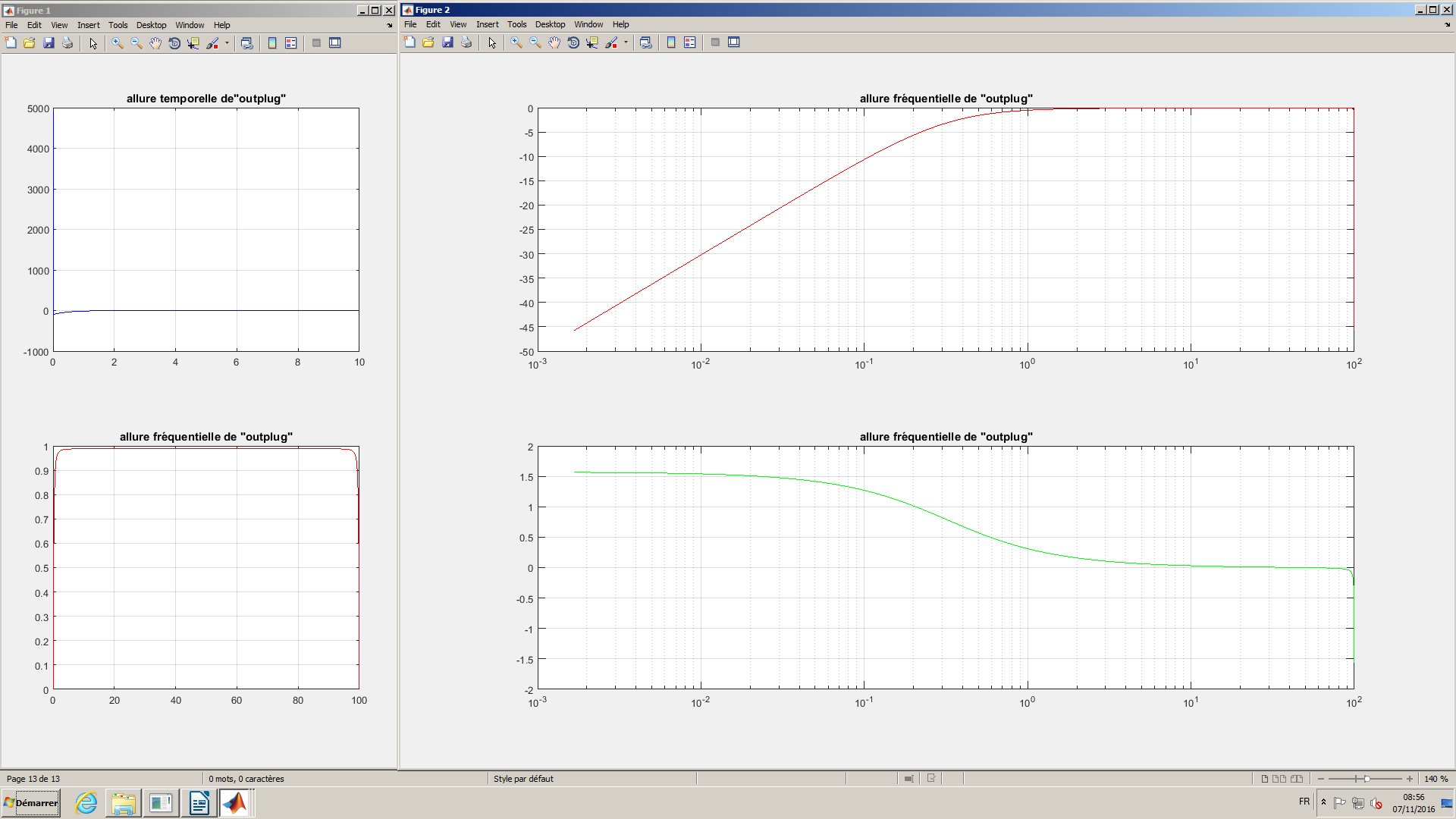


Figure 10 : Réponse impulsionnelle du filtre passe-haut

La courbe en haut à droite représente le comportement fréquentiel en amplitude du système. On remarque qu’elle correspond bien à un système du type passe-haut du premier ordre.

La symétrie observée dans l’analyse précédente est ici confirmée.

## Représentation du filtre global :

Nous allons étudier les réponses temporelles système complet à des signaux quelconques pouvant provenir des capteurs que nous cherchons à mettre en œuvre. Le système est représenté par l’assemblage des fonctions en Figure 11.

****

Figure 11 : Représentation graphique du filtre global

La fonction de transfert décrite par le précédent système nous est fournie et est :

Où :

K => le coefficient du filtre

αn => l’angle calculé

αn-1 => l’angle calculé précédemment

αacc, n => l’angle renvoyé par l’accéléromètre

Vgyr, n => la vitesse angulaire renvoyée par le gyroscope

∆ => le pas de simulation

La fréquence de coupure est la même que précédemment, soit 0.32Hz. C’est autour de cette fréquence que l’on observera une symétrie du comportement passe-bas et passe-haut.

## Réponse du filtre global à des signaux quelconques :

On applique maintenant en entrée du système des signaux quelconques pouvant être fournis par les capteurs. Ces signaux, ainsi que la réponse du système est présentée sur la Figure 12 où les couleurs de chaque signal sont :

Bleu => Angle renvoyé par l’accéléromètre

Rouge => Angle renvoyé par le gyroscope

Violet => Angle réel

Jaune => Angle calculé par le système



Figure 12 : réponse au filtre global avec ses paramètres initiales à des signaux quelconques

On observe que le signal calculé est très proche du signal réel, le système répond bien au cahier des charges.

On réduit maintenant le pas d’échantillonnage du système en concevant les autres paramètres, ainsi que les couleurs définissants les signaux. Lesdits signaux sont présentés en Figure 13.



Figure 13 : Réponse du filtre global avec un dt de 1e-3 à des signaux quelconques

On remarque que le signal de sortie est plus bruité que le signal précédemment analysé, mais il est plus centré sur le signal réel. On en déduit donc que réduire le pas d’échantillonnage équivaut à réduire la fréquence de coupure du système.

On réduit maintenant le coefficient k à une valeur de 0.1. On observe le résultat en Figure 14.



Figure 14 : Réponse du filtre global avec k = 0.1 et 1-k=0.9 à des signaux quelconques

On remarque que l’on obtient maintenant le signal émanent de l’accéléromètre, c’est-à-dire que le système ne se comporte plus que comme un passe haut. Donc diminuer k réduit aussi la fréquence de coupure du système.

On en déduit que les réglages déterminés au début nous permettent de répondre au cahier des charges, soit un k à 0.98, et un pas d’échantillonnage de 1e-3.

## Représentation du filtre global en code Matlab :

Un code Matlab nous est maintenant proposé pour décrire le fonctionnement du système. Ce code est présenté en Figure 15.

Clc

clear all

close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% simulation du filtrage des angles accelerometre et de la vitesse gyro

% entree: donnees capteurs dans fichier capteur\_position.mat

% capteur\_position[1,:] vecteur temps

% capteur\_position[2,:] angle accelerometre

% capteur\_position[3,:] angle gyroscope

% capteur\_position[4,:] angle reel

% capteur\_position[5,:] vitesse gyro

%sortie: angle

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% variables globales données capteur

% angle

load capteur\_position

% initialisation

angle(1)=0;

dt=capteur\_position(1,2)-capteur\_position(1,1);

% boucle simulant la boucle infinie et la procédure d’IT

for i=1:length(capteur\_position(1,:))

t(i)=capteur\_position(1,i);

angle\_gyro(i)=capteur\_position(3,i);

angle\_reel(i)=capteur\_position(4,i);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%procedure sous IT cadence à dt

%angle\_accelero=acquisition\_accelero();

angle\_accelero(i)=capteur\_position(2,i);

%vitesse\_gyro=acquisition\_gyro();

vitesse\_gyro(i)=capteur\_position(5,i);

%calcul de l’angle

angle(i+1)=0.98\*(angle(i)+vitesse\_gyro(i)\*dt)+0.02\*angle\_accelero(i);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

plot(t,angle\_accelero,’c’); hold on

plot(t,angle\_reel,’m’);

plot(t,angle\_gyro,’r’);

plot(t,angle(1:end-1),’y’);

grid

Figure 15 : Représentation du filtre global en code Matlab

On remarque dans ce code une partie servant à récupérer les valeurs fournies par les différents capteurs (exemple : angle\_accelero(i)=capteur\_position(2,i); ) puis, un calcul mettant en œuvre tous ces paramètre, qui est le filtre lui-même : angle(i+1)=0.98\*(angle(i)+vitesse\_gyro(i)\*dt)+0.02\*angle\_accelero(i); .

Puis, toutes les courbes sont affichées grâce à différentes commandes plot.

## Réponse du filtre global en code Matlab à des signaux quelconques :

La réponse du filtre précédemment décris en code Matlab aux mêmes stimuli que pour le filtre décris en blocs fonctionnels est présenté en Figure 16.

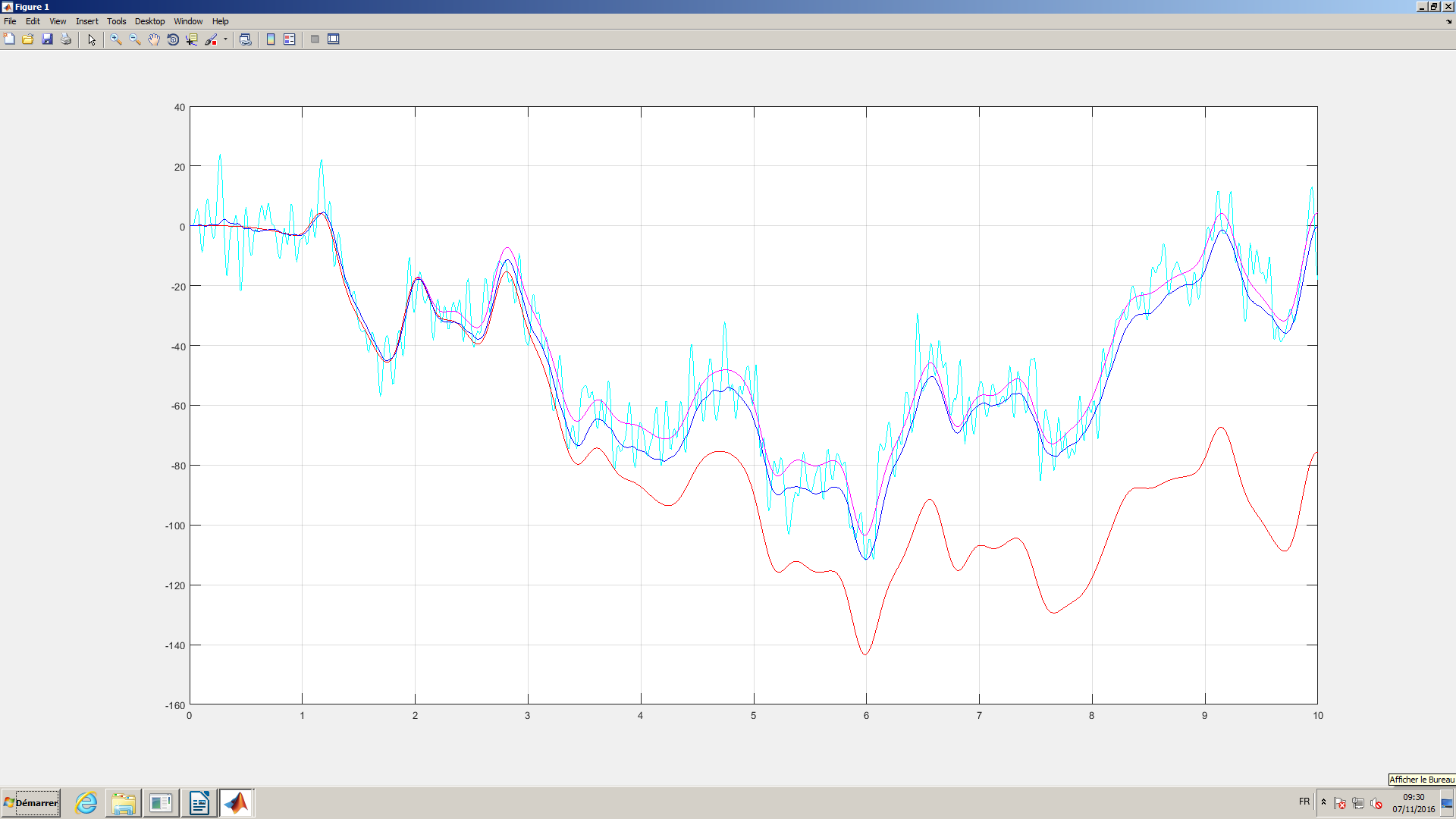


Figure 16 : simulation du filtre global en code matlab

On remarque que le résultat obtenue est similaire à celui obtenu grâce à la description en blocs fonctionnels avec les bons réglages de k et de dt.

Nous validons le fonctionnement du filtre global en environnement de simulation Matlab.

# Mise en œuvre des deux capteurs séparés :

## Introduction :

Dans cette partie, nous cherchons à mettre en œuvre un accéléromètre puis un gyroscope séparément pour pouvoir lire différentes valeurs, et ainsi calculer l’angle d’inclinaison de cela.

Ces manipulations nous permettrons de constater les qualités et les défauts de ces deux capteurs. Ainsi, nous pourrons observer les comportements en haute fréquences et basses fréquences de ces deux capteurs

Nous chercherons ensuite à les combiner pour obtenir un résultat proche de la réalité, grâce au filtre que nous avons pu étudier dans la partie précédente (

Étude du filtre :).

## Mise en œuvre de l’accéléromètre :

### Présentation :

L’accéléromètre mis en œuvre dans notre étude est l’[ADXL 335](http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf). Cet accéléromètre nous renvoie sous forme de valeurs analogiques l’accélération sur trois axes : x, y et z. à partir de l’accélération présente sur x, et celle présente sur y, sachant que la gravité de la terre impose une accélération de 1G perpendiculaire à sa surface, nous somme capable de connaitre l’angle d’inclinaison du dispositif par rapport à la surface terrestre. Ce principe est illustré par la Figure 17.

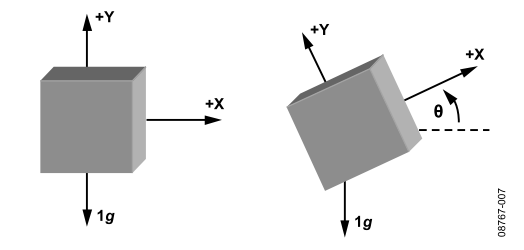


Figure 17 : Mesure de l’angle avec deux axes

Après différents calcul trigonométriques, nous obtenons la relation suivante :

C’est cette dernière relation que nous implémenterons dans notre programme. Notons qu’une seconde technique n’utilisant qu’un seul axe existe, mais nous avons ici décidé de ne pas l’étudier.

### Etude de la documentation technique :

Nous cherchons maintenant différents paramètres de l’accéléromètre dans sa [documentation](http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL335.pdf). Ainsi, nous trouvons les valeurs suivantes :

Tension maximum d’alimentation : 3.6V

Sensibilité pour une alimentation de 3.3V : 330mV/g

### Codage des fonctions de traitement :

Pour pouvoir mettre en forme les informations transmises par l’accéléromètre, nous utilisons un microcontrôleur intégré sur une carte [Arduino Due](https://www.arduino.cc/en/Main/ArduinoBoardDue). Cette carte intègre un cœur de calcul ARM M3. Pour utiliser le capteur, nous proposons le programme suivant qui affiche les valeurs renvoyées en volts, puis en G, et enfin l’angle calculé.

// Variables et constantes Globales ////////////////////////////////////////////////////////////////////////////////////

const int RES\_ADC **=** 1024**;** // Définition de la résolution de l'ADC

const float MAIN\_VCC **=** 3.3**;** // Définition de la tension d'alimentation de référence générale

const int buttonPin **=** 2**;** // Constante de la pin d'un bouton utilisé pour lancer la calibration par la suite

int buttonState**;** // Etat courant du bouton précédemment définit

// ACCELEROMETRE Variables et constantes ////////////////////////////////////////////////////////////////////////////////////

const int ACC\_ST\_PIN**=**22**;** // Constante de la pin reliée à la pin test de l'accéléromètre

const int ACC\_X\_PIN **=** A0**;** // Constante de la pin récupérant l'accélération en X

const int ACC\_Y\_PIN **=** A1**;** // Constante de la pin récupérant l'accélération en Y

const int ACC\_Z\_PIN **=** A2**;** // Constante de la pin récupérant l'accélération en Z

int AccxRawMin **=** 512**;** // Variable utilisée pour stocker la valeur Min atteint par X pendant la calibration

int AccxRawMax **=** 512**;** // Variable utilisée pour stocker la valeur Max atteint par X pendant la calibration

int AccyRawMin **=** 512**;** // Variable utilisée pour stocker la valeur Min atteint par Y pendant la calibration

int AccyRawMax **=** 512**;** // Variable utilisée pour stocker la valeur Max atteint par Y pendant la calibration

int AcczRawMin **=** 512**;** // Variable utilisée pour stocker la valeur Min atteint par Z pendant la calibration

int AcczRawMax **=** 512**;** // Variable utilisée pour stocker la valeur Max atteint par Z pendant la calibration

// INITIALISATIONS ///////////////////////////////////////////////////////////////////////////////////////////////////////

void setup**()** **{**

pinMode**(**ACC\_ST\_PIN**,** OUTPUT**);** // Mise en sortie de la pin reliée à la pin test de l'accéléromètre

digitalWrite**(**ACC\_ST\_PIN**,**LOW**);**

pinMode**(**ACC\_X\_PIN**,** INPUT**);** // Mise en entrée de la pin récupérant l'accélération en X

pinMode**(**ACC\_Y\_PIN**,** INPUT**);** // Mise en entrée de la pin récupérant l'accélération en Y

pinMode**(**ACC\_Z\_PIN**,** INPUT**);** // Mise en entrée de la pin récupérant l'accélération en Z

pinMode**(**buttonPin**,** INPUT**);** // Mise en Entrée du bouton utilisé pour lancer manuellement la calibration de l'accéléromètre

Serial**.**begin**(**9600**);** // Initialisation de la connexion serial (9600 baud)

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

**}**

// MAIN LOOP //////////////////////////////////////////////////////////////////////////////////////////////////////////////

void loop**()** // MAIN LOOP

**{**

buttonState **=** digitalRead**(**buttonPin**);** // Lecture de l'état du bouton

//(Par manque de moyen nous utiliserons un fil branché tour à tour sur le 3.3v ou le GND)

**if(**buttonState **==** LOW**)** // si fil branché au GND

**{**

Serial**.**println**(**" Acceleration en X (Volts) : " **+** String**((**getValueAxe**(**ACC\_X\_PIN**)/**1024**)\***3.3**)** **+** " V" **);**

Serial**.**println**(**" Acceleration en Y (Volts) : " **+** String**((**getValueAxe**(**ACC\_Y\_PIN**)/**1024**)\***3.3**)** **+** " V" **);**

Serial**.**println**(**" Acceleration en X (g) : " **+** String**(**getAccValue**(**ACC\_X\_PIN**))** **+** " g" **);**

Serial**.**println**(**" Acceleration en Y (g) : " **+** String**(**getAccValue**(**ACC\_Y\_PIN**))** **+** " g" **);**

Serial**.**println**(**" Angle mesure (accelerometre) : " **+** String**(**getAccAngle**())** **+** " deg" **);**

Serial**.**println**();**Serial**.**println**();** // Espace d'affichage

delay**(**pas**);**

**}**

**else** **if(**buttonState **==** HIGH**)** // Si fil branché au 3.3v

**{**

Serial**.**println**(**"Veuillez faire prendre toutes les positions possibles à l'accéléromètre"**);**

AccCalibrate**();** // Calibration de l'accéléromètre

Serial**.**println**(**"Calibration en cours"**);**

**}**

**}**

// Fonctions de l'ACCELEROMETRE ///////////////////////////////////////////////////////////////////////////////

void AccCalibrate**()**// Calibration de l'accéléromètre (Détermination des valeurs maximum et minimum prise par tous les axes (de 0 à 1023)

**{**

int AccxRaw **=** getValueAxe**(**ACC\_X\_PIN**);** // Variable très brève contenant la valeur actuelle présente sur l'axe X (de 0 à 1023)

int AccyRaw **=** getValueAxe**(**ACC\_Y\_PIN**);** // Variable très brève contenant la valeur actuelle présente sur l'axe Y (de 0 à 1023)

int AcczRaw **=** getValueAxe**(**ACC\_Z\_PIN**);** // Variable très brève contenant la valeur actuelle présente sur l'axe Z (de 0 à 1023)

**if(**AccxRaw **<** AccxRawMin**)** AccxRawMin **=** AccxRaw**;** // si la valeur présente sur l'axe X est plus petite que la valeur min elle devient la nouvelle valeur min

**if(**AccxRaw **>** AccxRawMax**)** AccxRawMax **=** AccxRaw**;** // si la valeur présente sur l'axe X est plus grande que la valeur max elle devient la nouvelle valeur max

**if(**AccyRaw **<** AccyRawMin**)** AccyRawMin **=** AccyRaw**;** // si la valeur présente sur l'axe Y est plus petite que la valeur min elle devient la nouvelle valeur min

**if(**AccyRaw **>** AccyRawMax**)** AccyRawMax **=** AccyRaw**;** // si la valeur présente sur l'axe Y est plus grande que la valeur max elle devient la nouvelle valeur max

**if(**AcczRaw **<** AcczRawMin**)** AcczRawMin **=** AcczRaw**;** // si la valeur présente sur l'axe Z est plus petite que la valeur min elle devient la nouvelle valeur min

**if(**AcczRaw **>** AcczRawMax**)** AcczRawMax **=** AcczRaw**;** // si la valeur présente sur l'axe Z est plus grande que la valeur max elle devient la nouvelle valeur max

**}**

int getValueAxe**(**int axe**)** // Récupération de la valeur (moyennée sur 10 valeurs) présente sur un axe (de 0 à 1023)

**{**

int val **=**0**;**

**for(**int i**=**0**;**i**<**10**;**i**++)** //Boucle de moyennage : val = Val1+Val2+...+Val10)/10

**{**

val**+=** analogRead**(**axe**);** // réalisation de la somme

**}**

**return** val**/**10 **;** // on renvoie la valeur divisée par 10

**}**

float getAccValue**(**int axe**)** // Récupération de la valeur présente sur un axe en G

**{**

/\* La fonction map sert à tracer une droite :

\* On attribue 2 valeurs arbitraires en ordonnée (ici -1000 et 1000)

\* Et on attribue, aux valeurs correspondantes en abscisses, un Min et un Max.

\* La fonction map trace donc la droite correspondante

\* On demande ensuite à la fonction map de placer un point un en abscisses et de nous donner sa correspondance en ordonnée (entre -1000 et 1000) suivant la droite

\* Son prototype : map(ValAPasserEnOrdonnee,AbscisseMin,AbscisseMax,OrdonneeMin,OrdonneeMax);

\*/

long scaled**;**

**if(**axe **==** ACC\_X\_PIN**)**scaled **=** map**(**getValueAxe**(**axe**),** AccxRawMin**,** AccxRawMax**,** **-**1000**,**1000**);**

**else** **if(**axe **==** ACC\_Y\_PIN**)**scaled **=** map**(**getValueAxe**(**axe**),** AccyRawMin**,** AccyRawMax**,** **-**1000**,**1000**);**

**else** **if(**axe **==** ACC\_Z\_PIN**)**scaled **=** map**(**getValueAxe**(**axe**),** AcczRawMin**,** AcczRawMax**,** **-**1000**,**1000**);**

float val **=** scaled**/**1000.0**;**

**return** val**;**

**}**

float getAccAngle**()** // Renvoi de l'angle en degré calculé sur 2 axes de l'accéléromètre

**{**

**return** **((**atan**(**getAccValue**(**ACC\_X\_PIN**)/**getAccValue**(**ACC\_Y\_PIN**))\*(**180**/**PI**)));** // angle en deg = arctan(ACCx(g) / ACCy(g) ) \* 180/pi

**}**

Après avoir flashé le microcontrôleur avec ce code, nous avons pu effectuer des tests :

-En faisant bouger l’accéléromètre sur ses axes, nous constatons qu’il y a bien variation des valeurs affichées.

-En faisant tourner l’accéléromètre de 90°, nous constatons que la valeur de l’angle calculé passe d’environ 1° à environ 87°, ce qui est concluant vis à vie de la précision de nos tests.

Nous ne constatons aucune variations hautes fréquences, car dans la fonction d’acquisition, nous faisons un moyennage sur 10 valeurs, ce qui créé un volant d’inertie sur variation d’accélération.

Nous validons le bon fonctionnement de l’accéléromètre seul.

## Mise en œuvre du gyroscope :

### Présentation :

Le gyroscope mis en œuvre dans notre étude est l’[ENC-03R](https://www.elecrow.com/download/ENC-03.pdf), à qui l’on a adjoint le circuit driver SEN05091P pour fonctionner. Les deux circuits sont rassemblés sur une même [CARTE](http://wiki.seeedstudio.com/wiki/Grove_-_Single_Axis_Analog_Gyro). Ce gyroscope nous renvoie sous forme de valeur analogique la vitesse angulaire selon deux axes, en l’occurrence, les axes x et y. à partir de cette vitesse, nous somme capable de trouver l’angle d’inclinaison en effet, l’angle n’est autre que la primitive de la vitesse angulaire. Nous proposons donc l’équation récurrente suivante pour calculer l’angle d’inclinaison à partir des informations fournies par le gyroscope :

Où delta est le pas d’échantillonnage.

C’est cette équation que nous mettrons en œuvre dans notre programme.

### Codage des fonctions de traitement :

Pour traiter le signal émis par le gyroscope, nous utilisons la même carte que précédemment avec l’accéléromètre. Pour réaliser ces fonctions, nous nous sommes inspiré de fonctions déjà existantes présentent sur ce [site](http://www.seeedstudio.com/wiki/Grove_-_Single_Axis_Analog_Gyro). Nous proposons le programme suivant pour afficher la valeur renvoyé par le gyroscope e l’angle calculé :

// Variables et constantes Globales ////////////////////////////////////////////////////////////////////////////////////

const int RES\_ADC **=** 1024**;** // Définition de la résolution de l'ADC

const float MAIN\_VCC **=** 3.3**;** // Définition de la tension d'alimentation de référence générale

// GYROSCOPE Variables et constantes ////////////////////////////////////////////////////////////////////////////////////

const int pas **=** 10**;** // Constante qui définit la durée du delay du main loop en ms (l'intervalle de temps dt)

const int GYRO\_PIN **=** A4**;** // Constante de la pin utilisée pour récupérer la valeur de la vitesse angulaire

float delta **=** **(**float**(**pas**)/**1000**);**

float gyroReferenceValue**=**0**;**

float sum**=**0**;**

// INITIALISATIONS ///////////////////////////////////////////////////////////////////////////////////////////////////////

void setup**()** **{**

pinMode**(**GYRO\_PIN**,** INPUT**);** // Mise en Entrée de la pin récupérant la vitesse angulaire du gyroscope

Serial**.**begin**(**9600**);** // Initialisation de la connexion serial (9600 baud)

initGyro**();** // Fonction d'initialisation du gyroscope (immobile) : gyroReferenceValue=(Val1+Val2+...+Val1000)/1000

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

**}**

// MAIN LOOP //////////////////////////////////////////////////////////////////////////////////////////////////////////////

void loop**()** // MAIN LOOP

**{**

Serial**.**println**(**" Valeur gyro : " **+** String**(**getGyroValue**())** **+** " deg/s" **);**

Serial**.**println**(**" Angle mesure (gyroscope) : " **+** String**(**getGyroAngle**())** **+** " deg" **);**

Serial**.**println**();**Serial**.**println**();** // Espace d'affichage

delay**(**pas**);**

**}**

// Fonctions du GYROSCOPE /////////////////////////////////////////////////////////////////////////////////

void initGyro**()** // Fonction d'initialisation du gyroscope (immobile) : gyroReferenceValue=(Val1+Val2+...+Val1000)/1000

**{**

Serial**.**println**(**"Ne touchez pas au Gyro : Calibration en cours"**);**

sum**=**0**;** // RAZ de la sum

**for(**int i**=**0**;**i**<**1000**;**i**++)** // On somme 1000 valeur de la vitesse en rad/s du gyro

**{**

sum **+=** analogRead**(**GYRO\_PIN**);** // Addition en chaine

delay**(**5**);** // Attente de 5ms

**}**

gyroReferenceValue **=** sum**/**1000.0**;** // On ajoute la valeur moyennée comme valeur de référence (fonctionne seulement si le gyro était immobile)

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

**}**

double getGyroValue**()** // Fonction de récupération de la valeur actuelle du gyro en deg/sec

**{**

**return** **((**double**)(**analogRead**(**GYRO\_PIN**)-**gyroReferenceValue**)\***4930.0**)/**1023.0**/**0.67**;**

**}**

double getGyroAngle**()** // Fonction de récupération de l'angle ( Intègre simplement la vitesse )

**{**

static double angle **=** 0**;**

angle **=(**getGyroValue**()\***delta**)** **+** angle**;** // On intègre la vitesse angulaire pour obtenir l'angle : An = An-1 + An \* dt

**return** angle**;** // Renvoi de l'angle en deg

**}**

Après avoir flashé le microcontrôleur avec ce code, nous avons pu effectuer des tests :

-En faisant tourner le gyroscope sur lui-même nous constatons qu’il y a bien variation des valeurs affichées.

-En faisant tourner gyroscope de 90°, nous constatons que la valeur de l’angle calculé passe d’environ 3° à environ 91°, ce qui est concluant vis à vie de la précision de nos tests. Nous constatons une dérive de l’angle qui est certainement dû à la présence d’une composante continue que nous intégrons.

Nous validons le bon fonctionnement du gyroscope seul.

## Combinaison des deux solutions :

### Le problème posé :

On a pu remarquer grâce aux tests précédemment effectué que le gyroscope présenté une dérive sur l’angle qui est calculé. Alors que l’accéléromètre est censé présenter des oscillations en hautes fréquences que nous avons pu supprimer en faisant une valeur moyenne.

On se propose donc d’intégrer le filtre précédemment étudié dans Matlab dans notre code pour obtenir un angle plus proche de la réalité.

### Codage du filtre :

Nous proposons l’implémentation suivante du filtre étudié basé sur les fonctions vus dans les précédents items :

// CALCUL FINAL ////////////////////////////////////////////////////////////////////////////////////////////////////////

float calculAngle**()** // Calcul de l'angle utilisant : l'accéléromètre ET le gyroscope ET le filtre associé

**{**

static double angle **=**0**;**

angle **=** **(**0.98 **\*** **(**angle **+** getGyroValue**()** **\*** delta **)** **+** 0.02 **\*** getAccAngle**());**

**return** angle**;**

**}**

Après avoir testé cette fonction en y faisant appel tous les dt, et en affichant les résultats, nous observons qu’il n’y a pas de dérive du signal quand l’état est stable, et qu’il n’y a pas d’ondulations hautes fréquences parasite quand le dispositif est mis en rotation.

Nous validons le fonctionnement général.

# Mise en œuvre de la centrale inertielle :

## 5.1. Présentation :

Nous allons maintenant chercher à mettre en œuvre une solution regroupent les deux capteurs dans un même packaging, une centrale inertielle. La centrale qui sera étudiée est la [MPU 6050](https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf). Cette centrale inertielle nous permet de récupérer les accélérations sur les trois axes comme nous le permettaient l’accéléromètre précédemment étudié. Nous avons également accès à la vitesse angulaire entre les axes x et y, comme nous pouvions le faire avec le gyroscope.

La liaison entre la centrale inertielle et le microcontrôleur se fait via un bus i2c, qui est un bus série avec un fil d’horloge et un fil de données. C’est en allant écrire et lire dans différents registres de la centrale via ledit bus que nous pourrons avoir accès aux différentes informations désirées. Il est à noter que la centrale est elle aussi dotée d’un DSP (Digital Signal Processor) lui permettant d’effectuer un pré traitement sur les informations envoyées. Ainsi elle peut filtrer les informations issues des capteurs en utilisant notamment un filtre de Kalman.

Dans cette partie, nous n’étudierons pas la partie communication qui a préalablement été développée est mise dans une [bibliothèque](https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/I2Cdev) mise à notre disposition. Nous disposons également d’une seconde [bibliothèque](https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050) nous permettant d’avoir directement les valeurs issues des capteurs de la centrale inertielle en faisant abstraction de la couche communication.

Notre étude du module se décomposera donc en plusieurs parties, une partie consistant en l’étude d’un code déjà développé nous permettant d’obtenir les différentes valeurs, puis nous modifierons ce code pour pouvoir réaliser l’affichage des angles mesurés, puis de celui calculé, et enfin nous effectuerons des tests pour visualiser ces angles.

Il est à noter que le code qui nous a été proposé était adapté à une architecture de type AVR, alors que nous sommes sur une architecture ARM, nous avons donc dû effectuer plusieurs modifications qui seront abordées dans la seconde partie pour palier à certains disfonctionnement.

## 5.2. Code étudié :

Nous allons maintenant étudier le code qui nous est proposé pour acquérir des données depuis la centrale inertielle. Ce code se décompose en plusieurs fichiers, car il intègre la gestion des moteurs et des codeurs rotatifs des moteurs. Nous n’étudieront ici seulement deux fichiers, gyro.ino qi est le programme principal, et angle.cpp qui contient les fonctions de calcul d’angle.

### 5.2.1. gyro.ino :

#define FILTRE\_COMP 1

#include "angle.h"

#include "SerialChart.h"

#include "serie.c"

#include "codeurs.h"

#include "moteurs.h"

#include <DueTimer.h>

//#include <FlexiTimer2.h>

#define CADENCE\_MS (10)

const double R **=** 0.05**;** //5 cm

//angle du robot (en radian)

double xiref **=** 0**;**

float consigne **=** 0**;**

float KP **=** 250**;**

float KI **=** 0**;**

void setup**()** **{**

Serial**.**begin**(**19200**);**

angleInit**();**

initCodeur**();**

initMotors**();**

FlexiTimer2**::**set**(**CADENCE\_MS**,** 1 **/** 1000.**,** itAsservissement**);** // rÃ©solution timer = 1 ms

FlexiTimer2**::**start**();**

**}**

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

angleAcquisition**();**

// if(millis() > 10000) xiref = PI;

**}**

//sans Serial.print(..), il faut environ 200us pour la boucle d'asservissement.

void itAsservissement**()**

**{**

//long date = micros();

int vitesseD**,** vitesseG**;** //ticks sur les 80 dernieres ms.

getVitesse**(**vitesseD**,**vitesseG**);**

double omegaDroit **=** 1000.**\***vitesseD**/(**8**\***CADENCE\_MS**)\***DEG\_TO\_RAD**;** //en rad/s (si 10ms)

double omegaGauche **=** 1000.**\***vitesseG**/(**8**\***CADENCE\_MS**)\***DEG\_TO\_RAD**;** //en rad/s (si 10ms)

const double vxref **=** consigne**;**

const double vxmes **=** **(**omegaDroit **+** omegaGauche**)** **\*** R **/** 2**;**

const double gyroRate **=** getGyroRate**();**

//P sur l'orientation du robot (direction): angle Xi

//expÃ©rimentalement, un tour => delta de 1800 ticks sur les codeurs

double P\_xi **=** KP**;**

const double xiMes **=** **(**getPosition**(**DROITE**)** **-** getPosition**(**GAUCHE**))\*(**2**\***PI**)/**1800**;** //en rad.

const double commande\_xi **=** P\_xi **\*** **(**xiref **-** xiMes**);**

PI sur la vitesse d'avance

const double Kpvx **=** **-**400**;**

const double Kivx **=** **-**600**;**

static double I\_vx **=** 0.0**;**

const double P\_vx **=** Kpvx **\*** **(**vxref **-** vxmes**);**

const double commande\_vx **=** P\_vx **+** I\_vx**;**

I\_vx **=** I\_vx **+** Kivx **\*** **(**CADENCE\_MS**/**1000.**)** **\*** **(**vxref **-** vxmes**);**

correcteur vertical

**=>** par rapport Ã  la vitesse de rotation et l'angle du robot.

const double Kpomega **=** **-**20**;** //correcteur proportionnel vitesse de rotation

s'il y a une instabilitÃ© avec oscillations avec le systÃ¨me qui va vers l'avant**/**arriÃ¨re

augmenter ce gain**:**

const double Kiomega **=** **-**1000**;** //correcteur proportionnel angle de rotation

const double gyroRateRef **=** 0.0**;** //consigne de vitesse de rotation.

const double AngleRef **=** 0.0**;** //consigne d'angle par rapport Ã  la verticale

const double P\_omega **=** Kpomega **\*** **(**gyroRateRef **-** gyroRate**);**

const double angle **=** getCompAngle**();**

const double I\_omega **=** Kiomega **\*** **(**AngleRef **-** angle**);**

const double commande\_omega **=** P\_omega **+** I\_omega**;**

const double commandeDroit **=** commande\_vx **-** commande\_omega **+** commande\_xi**;**

const double commandeGauche **=** commande\_vx **-** commande\_omega **-** commande\_xi**;**

setMotorDroit**(**commandeDroit**);**

setMotorGauche**(**commandeGauche**);**

Serial**.**println**(**angle**\***RAD\_TO\_DEG**);**

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

Serial**.**print**(**commande\_omega**);**

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

Serial**.**println**(**commande\_vx**);**

**}**

La première chose que nous observons dans ce code, c’est l’appel à une classe FlexiTimer, après recherches, il s’avère que cette classe, permettant de réaliser des interruptions sur alarme, n’est utilisable uniquement sur des architectures AVR. Nous avons donc dû trouver son équivalent pour ARM, la classe DueTimer. L’intégration de celle-ci sera vue dans la prochaine partie.

Dans la boucle principale, nous constatons qu’une seule fonction est appelée, la fonction permettant de faire l’acquisition de l’angle. Nous verrons la composition de cette fonction dans la partie concernant angle.cpp d’où elle est issue. La dernière fonction déclarée dans cette page est l’interruption de l’asservissement, cette fonction est appelée toute les 10 ms pour effectuer des calculs nécessitant une très grande précision temporelle.

En premier lieu, cette fonction permet de récupérer la vitesse de rotation de chaque moteur en rad/s, à travers l’appel à une fonction issue de codeur.c, puis un simple calcul de mise en forme de la valeur renvoyée par ladite fonction.

Ensuite, un simple calcul basé sur la vitesse angulaire de ces deux moteurs et le rayon des roues nous permet de récupérer la vitesse moyenne de déplacement du gyropode en m/s. Cette valeur est stockée dans la variable vxmes.

L’étape suivante consiste à calculer la rotation du gyropode autour de l’axe vertical. Pour cela nous faisons appel à une fonction nous renvoyant la position des deux codeurs des roues, puis un calcul de mise en forme pour homogénéiser le résultat à des radians. Par exemple, si le moteur droit a fait 3 tours, et le gauche en a effectué 2, on sait que le gyropode a fait un tour complet, soit 2π rad. La ligne suivante nous permet d’obtenir la commande sur cette rotation, en faisant le produit de l’erreur, obtenue grâce à la différence entre la consigne et la mesure, et du gain d’un correcteur P, définis constant au début du programme (250).

Ensuite, un correcteur PI (proportionnel intégral) est appliqué sur la commande de vitesse. On commence par calculer l’erreur comme pour la partie précédente, on la multiplie par un simple gain, pour finir par lui appliquer l’action intégrale à travers la variable I\_vx qui est déclarée comme étant static pour se mettre à jours à partir de sa précédente valeurs et d’autres paramètres.

Pour finir, il ne nous reste plus qu’à asservir le gyropode en rotation autour de l’axe horizontal grâce au calcul de l’angle d’inclinaison, et la vitesse d’inclinaison. On commence par calculer la commande en vitesse angulaire autour de cet axe à partir de l’erreur de vitesse multiplié par un gain. Puis, nous calculons la commande en position avec la même technique en récupérant l’angle à partir de la centrale inertielle. La récupération de l’angle et de la vitesse angulaire est expliquée dans la partie angle.cpp. On remarque ici que le correcteur est de type PID (Proportionnel Intégral Dérivée).

### 5.2.2. angle.cpp :

#include <Wire.h>

#include <I2Cdev.h>

#define FILTRE\_COMP 1

#include "Wire.h"

#include "I2Cdev.h"

#include "MPU6050.h"

#include "angle.h"

#include <avr/interrupt.h>

#ifdef FILTRE\_KALMAN

#include <Kalman.h>

Kalman kalman**;**

static double gyroUbRate**,** gyroUbRatef**=**0**,** kalmAngle **=** 0**;**

static double tau\_gyro **=** 0.75**;**

#endif

#ifdef FILTRE\_COMP

double compAngle**;**

double a **=** 0.98**;**

#endif

double accAngle**;** //acceleration brute

double gyroRate**;** //gyro brut

static unsigned long date**;** //pour le calcul du dt

double gyroYzero**;** //offset gyro.

MPU6050 mpu**;**

bool checkMinMax**(**int16\_t **\***array**,** uint8\_t length**,** int16\_t maxDifference**)** **{** // Used to check that the robot is laying still while calibrating

int16\_t min **=** array**[**0**],** max **=** array**[**0**];**

**for** **(**uint8\_t i **=** 1**;** i **<** length**;** i**++)** **{**

**if** **(**array**[**i**]** **<** min**)**

min **=** array**[**i**];**

**else** **if** **(**array**[**i**]** **>** max**)**

max **=** array**[**i**];**

**}**

**return** max **-** min **<** maxDifference**;**

**}**

bool calibrateGyro**()** **{**

int16\_t gyroYbuffer**[**25**];**

**for** **(**uint8\_t i **=** 0**;** i **<** 25**;** i**++)** **{**

gyroYbuffer**[**i**]** **=** mpu**.**getRotationY**();**

delay**(**10**);**

**}**

**if** **(!**checkMinMax**(**gyroYbuffer**,** 25**,** 2000**))** **{**

**return** 1**;**

**}**

**for** **(**uint8\_t i **=** 0**;** i **<** 25**;** i**++)**

gyroYzero **+=** gyroYbuffer**[**i**];**

gyroYzero **/=** 25**;**

**return** 0**;**

**}**

void angleInit**()**

**{**

Wire**.**begin**();**

mpu**.**initialize**();**

#ifdef FILTRE\_KALMAN

kalman**.**setAngle**(**0**);** // Set starting angle

#endif

**while** **(**calibrateGyro**());** // Run again if the robot is moved while calibrating

date **=** millis**();**

**}**

void angleAcquisition**()**

**{**

int16\_t accX **=** mpu**.**getAccelerationX**();**

int16\_t accZ **=** mpu**.**getAccelerationZ**();**

accAngle **=** atan2**((**double**)**accX **,** **(**double**)**accZ **);**

int16\_t gyroY **=** mpu**.**getRotationY**();**

gyroRate **=** **-(((**double**)**gyroY **-** gyroYzero**)** **/** 131**)** **\*** DEG\_TO\_RAD**;**

const double dt **=** **(**millis**()-**date**)/**1000.**;**

static float angletestgyro**;**

angletestgyro**=** **(**gyroRate**\***dt**)** **+** angletestgyro**;**

#ifdef FILTRE\_KALMAN

// Calculate the angle using a Kalman filter

const double tmpKalman **=** kalman**.**getAngle**(**accAngle**,** gyroRate**,** dt**);**

gyroUbRate **=** kalman**.**getRate**();**

//vitesse gyro sans biais.

const double \_\_gyroUbRatef **=** tau\_gyro**\***gyroUbRatef**+(**1**-**tau\_gyro**)\***gyroUbRate**;**

const uint8\_t oldSREG **=** SREG**;** //enregistre l'Ã©tat (donc des its).

cli**();** //supprime les interruptions

kalmAngle **=** tmpKalman**;**

gyroUbRatef **=** \_\_gyroUbRatef**;**

SREG **=** oldSREG**;** //reaffectation de l'Ã©tat d'interruption precedent

#endif

#ifdef FILTRE\_COMP

const double tmpComp **=** a**\*(**compAngle **+** dt **\*** gyroRate**)** **+** **(**1**-**a**)** **\*** accAngle**;**

const uint8\_t oldSREG2 **=** SREG**;** //enregistre l'Ã©tat (donc des its).

cli**();** //supprime les interruptions

compAngle **=** tmpComp**;**

SREG **=** oldSREG2**;** //reaffectation de l'Ã©tat d'interruption precedent

#endif

delay**(**10**);**

date **=** millis**();**

**}**

double getGyroRate**()**

**{**

volatile double tmp**;**

cli**();**

#ifdef FILTRE\_KALMAN

tmp **=** gyroUbRatef**;**

#else

tmp **=** 0**;**

#endif

sei**();**

**return** tmp**;**

**}**

double getCompAngle**()**

**{**

volatile double tmp**;**

cli**();**

#ifdef FILTRE\_COMP

tmp **=** compAngle**;**

#else

tmp **=** 0**;**

#endif

sei**();**

**return** tmp**;**

**}**

double getRawGyro**()**

**{**

volatile double tmp**;**

cli**();**

tmp **=** gyroRate**;**

sei**();**

**return** tmp**;**

**}**

double getRawAngle**()**

**{**

volatile double tmp**;**

cli**();**

tmp **=** accAngle**;**

sei**();**

**return** tmp**;**

**}**

double getKalmanAngle**()**

**{**

volatile double tmp**;**

cli**();**

#ifdef FILTRE\_KALMAN

tmp **=** kalmAngle**;**

#else

tmp **=** 0**;**

#endif

sei**();**

**return** tmp**;**

**}**

## 5.3. Modification du code :

En essayant de compiler le code précédent, nous nous sommes rendus compte qu’il était incompatible avec l’architecture matérielle mise à notre disposition. En effet, les différentes fonctions comportaient des appels vers des fonctions directement issues du Framework AVR, alors que nous étions sur une plateforme de type ARM. Il nous a donc fallut effectuer quelques changements, comme la librairie pour les timer, ou encore les arrêt et mise en services des interruptions matérielles. Ces différentes modifications se trouvent présentées dans le code suivant.

### 5.3.1. gyro.ino :

#include "angle.h"

#include "Plotter.h"

#include <DueTimer.h>

#define CADENCE\_MS (10)

void setup**()** **{**

Serial**.**begin**(**19200**);**

angleInit**();**

Timer3**.**attachInterrupt**(**itAsservissement**);**

Timer3**.**start**(**CADENCE\_MS**\***1000**);** // Calls every 10ms

**}**

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

angleAcquisition**();**

**}**

void itAsservissement**()**

**{**

long date **=** micros**();**

const double gyroRate **=** getGyroRate**();**

const double gyroRateRef **=** 0.0**;** //consigne de vitesse de rotation.

const double AngleRef **=** 0.0**;** //consigne d'angle par rapport Ã  la verticale

const double angle **=** getCompAngle**();**

**}**

### 5.3.2. angle.cpp:

#include <Wire.h>

#include <I2Cdev.h>

#include "MPU6050.h"

#include "angle.h"

double compAngle**;**

double a **=** 0.98**;**

double accAngle**;** //acceleration brute

double gyroRate**;** //gyro brut

static unsigned long date**;** //pour le calcul du dt

double gyroYzero**;** //offset gyro.

MPU6050 mpu**;**

bool checkMinMax**(**int16\_t **\***array**,** uint8\_t length**,** int16\_t maxDifference**)** **{** // Used to check that the robot is laying still while calibrating

int16\_t min **=** array**[**0**],** max **=** array**[**0**];**

**for** **(**uint8\_t i **=** 1**;** i **<** length**;** i**++)** **{**

**if** **(**array**[**i**]** **<** min**)**

min **=** array**[**i**];**

**else** **if** **(**array**[**i**]** **>** max**)**

max **=** array**[**i**];**

**}**

**return** max **-** min **<** maxDifference**;**

**}**

bool calibrateGyro**()** **{**

int16\_t gyroYbuffer**[**25**];**

**for** **(**uint8\_t i **=** 0**;** i **<** 25**;** i**++)** **{**

gyroYbuffer**[**i**]** **=** mpu**.**getRotationY**();**

delay**(**10**);**

**}**

**if** **(!**checkMinMax**(**gyroYbuffer**,** 25**,** 2000**))** **{**

**return** 1**;**

**}**

**for** **(**uint8\_t i **=** 0**;** i **<** 25**;** i**++)**

gyroYzero **+=** gyroYbuffer**[**i**];**

gyroYzero **/=** 25**;**

**return** 0**;**

**}**

void angleInit**()**

**{**

Wire**.**begin**();**

mpu**.**initialize**();**

**while** **(**calibrateGyro**());** // Run again if the robot is moved while calibrating

date **=** millis**();**

**}**

void angleAcquisition**()**

**{**

int16\_t accX **=** mpu**.**getAccelerationX**();**

int16\_t accZ **=** mpu**.**getAccelerationZ**();**

accAngle **=** atan2**((**double**)**accX **,** **(**double**)**accZ **);**

int16\_t gyroY **=** mpu**.**getRotationY**();**

gyroRate **=** **-(((**double**)**gyroY **-** gyroYzero**)** **/** 131**)** **\*** DEG\_TO\_RAD**;**

const double dt **=** **(**millis**()-**date**)/**1000.**;**

static float angletestgyro**;**

angletestgyro**=** **(**gyroRate**\***dt**)** **+** angletestgyro**;**

const double tmpComp **=** a**\*(**compAngle **+** dt **\*** gyroRate**)** **+** **(**1**-**a**)** **\*** accAngle**;**

compAngle **=** tmpComp**;**

delay**(**10**);**

//Serial.print(angletestgyro\*RAD\_TO\_DEG);

//Serial.print(",");

Serial**.**print**(**accAngle**\***RAD\_TO\_DEG**);**

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

Serial**.**println**(**compAngle**\***RAD\_TO\_DEG**);**

date **=** millis**();**

**}**

double getGyroRate**()**

**{**

volatile double tmp**;**

tmp **=** 0**;**

**return** tmp**;**

**}**

double getCompAngle**()**

**{**

volatile double tmp**;**

tmp **=** compAngle**;**

**return** tmp**;**

**}**

double getRawGyro**()**

**{**

volatile double tmp**;**

tmp **=** gyroRate**;**

**return** tmp**;**

**}**

double getRawAngle**()**

**{**

volatile double tmp**;**

tmp **=** accAngle**;**

**return** tmp**;**

**}**

## 5.4. Tests et conclusion :

Pour valider le bon fonctionnement de notre système, nous procédons à des relevés de l’angle issu du gyroscope et du filtre, puis de l’accéléromètre et du filtre. Ces caractéristiques sont présentées en Figure 18 et Figure 19.

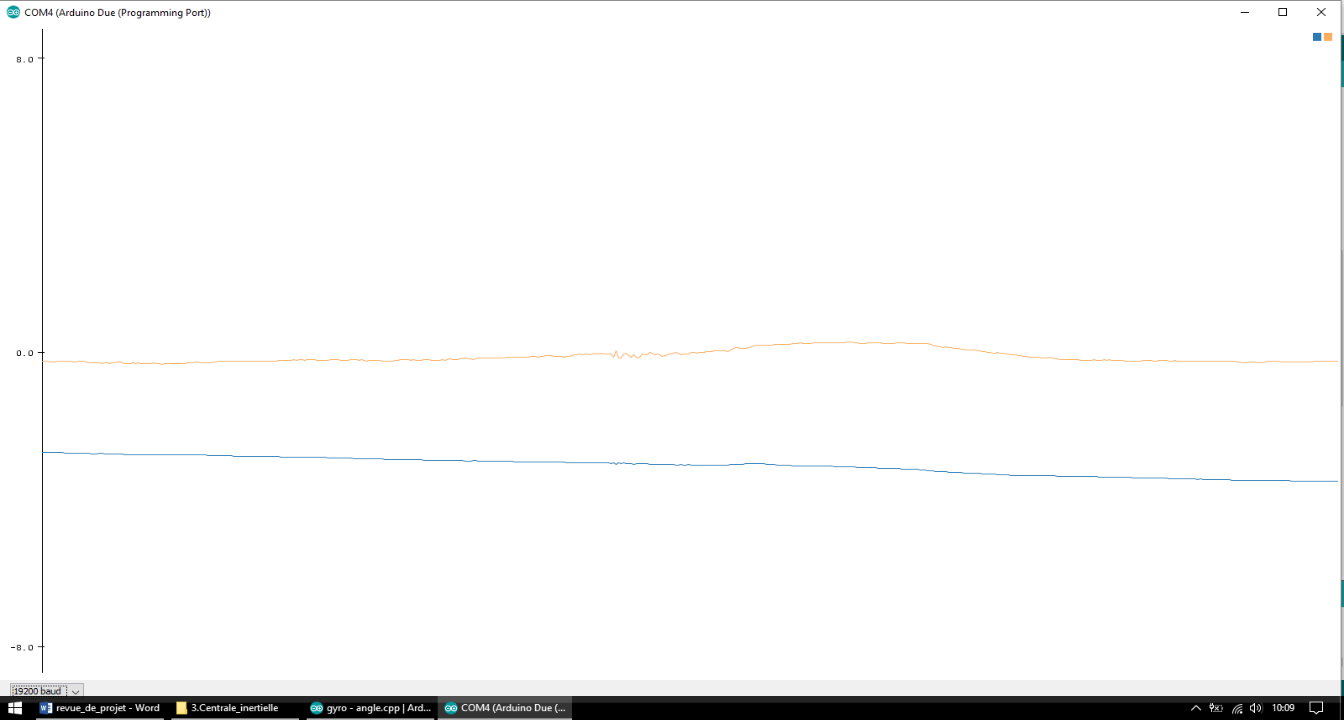


Figure 18 : Relevé du signal filtré et du signal issu du gyroscope

Sur la Figure 18, La trace jaune représente le signal filtré, alors que la trace bleu représente le signal issu du gyroscope. Sur cette derniere trace on observe bien la dérive du gyroscope du à une intégration d’une composante continue dans le procédé, alors que sur la trace jaune, on remarque bien que cette dérive a été très atténuée, ce qui nous convient parfaitement.

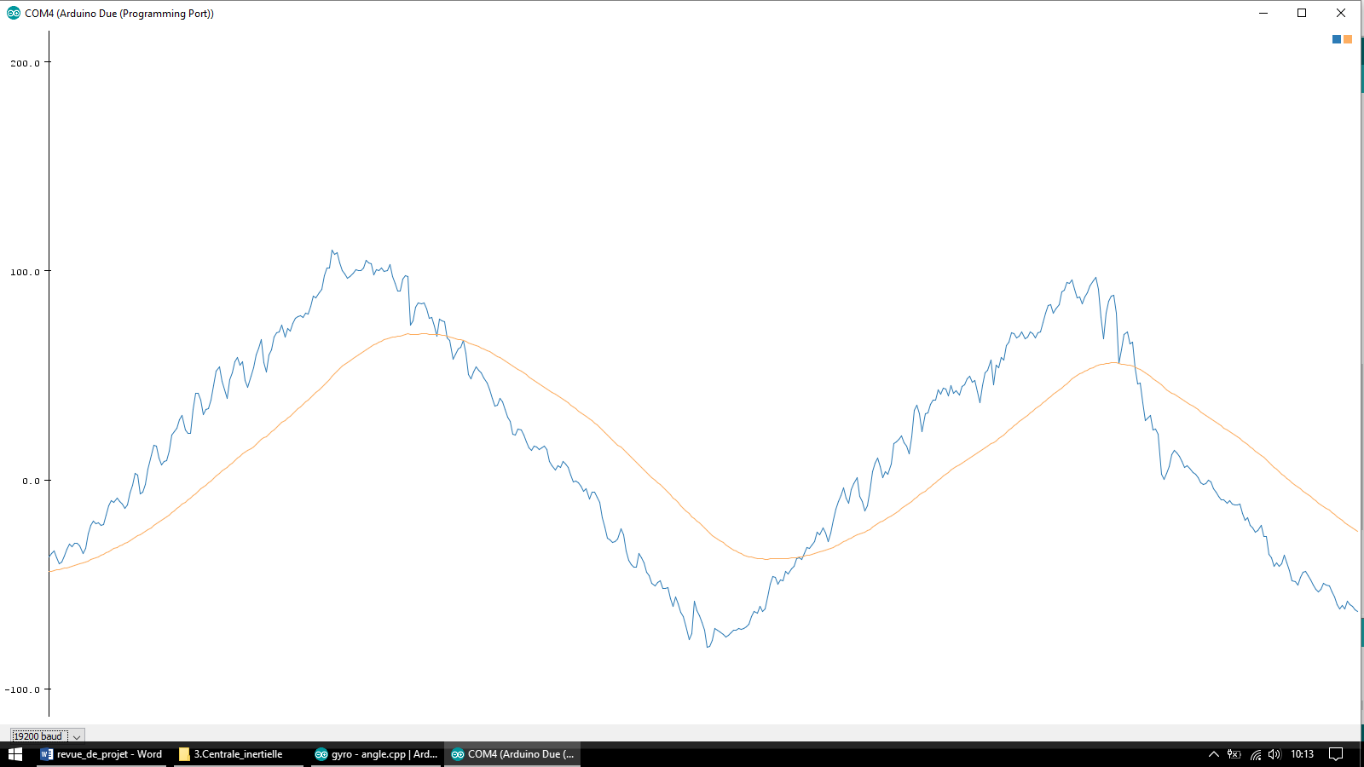


Figure 19 : Relevé du signal filtré et du signal issu de l’accéléromètre

Sur la Figure 19, la trace jaune représente le signal filtré, alors que la trace bleue représente le signal issu de l’accéléromètre. On remarque bien sur ce dernier signal les oscillations hautes fréquences qui pollues notre signal et qui pourrait remettre en cause la fiabilité de notre système. Tandis que sur le signal jaune, ces « pollutions » ont été supprimés grâce à un filtrage passe bas. Néanmoins, nous remarquons que notre filtre a introduit un retard dans le signal, comme tous filtre.

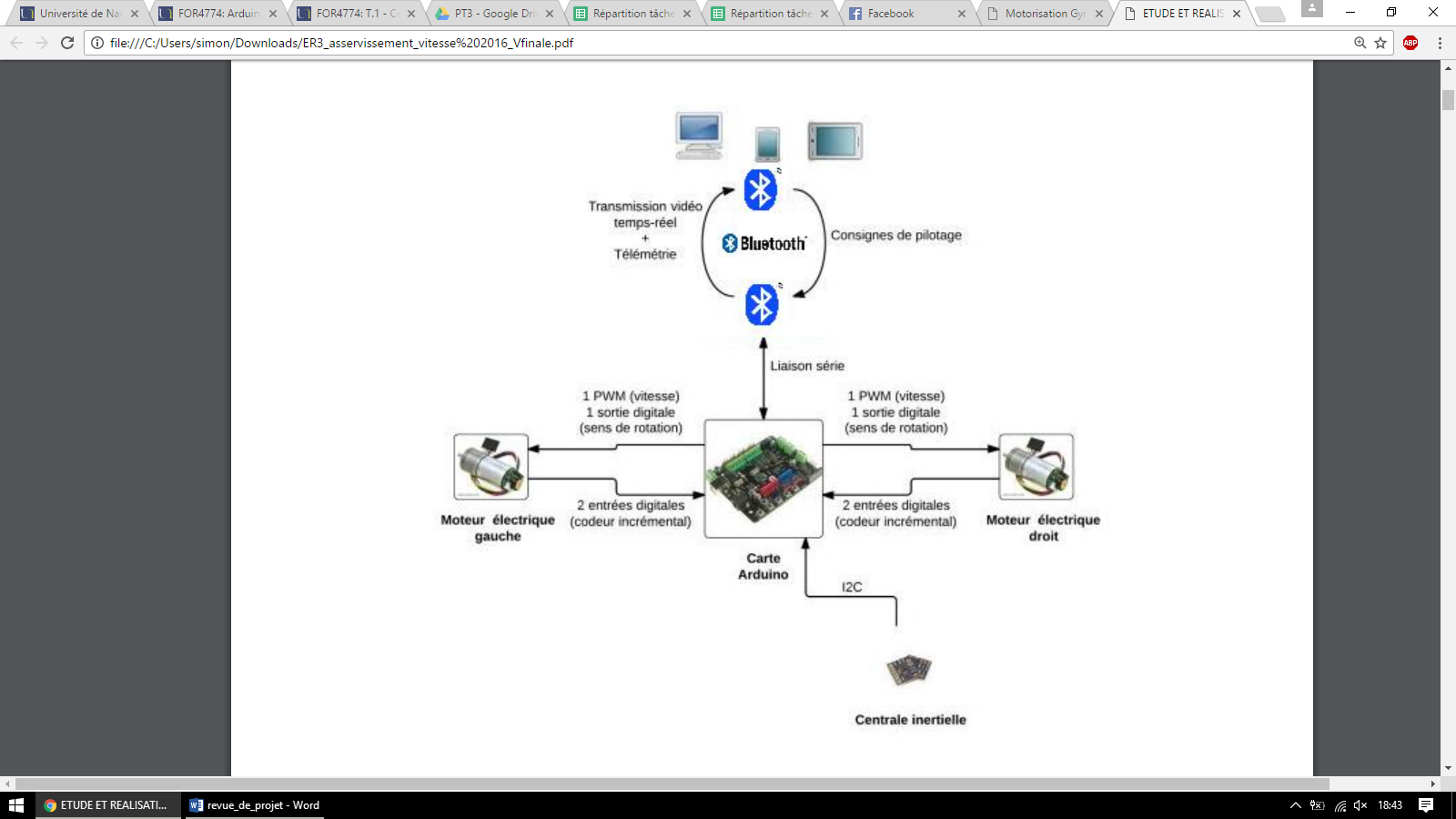
La viabilité de notre système est donc validée par les précédents tests. Il ne nous reste donc plus qu’à asservir le gyropode avec ces valeurs calculées, ce que nous nous proposons de faire dès maintenant.

# Asservissement du gyropode :

Il était normalement prévu que nous conjuguions nos efforts avec un groupe en charge de l’asservissement du gyropode pour la suite du projet. Seulement aucun autre groupe n’était prêt en même temps que nous. Nous avons donc choisi de nous occuper nous-même de l’asservissement et par la même occasion de la réalisation du robot gyropode.

La commande du gyropode nécessite de pouvoir maitriser le déplacement. Pour cela il faut réaliser l’asservissement de vitesse des 2 moteurs. Pour la programmation en C ou C++, on utilisera l’IDE Arduino. L’objectif final est de pouvoir piloter un gyropode.

Figure 20 : Synoptique asservissement du gyropode



Carte réalisée

## Commande des moteurs :

### 6.1.1 Cahier des charges

Pour commencer nous avons dû récupérer le schéma électronique de la commande des moteurs pour développer notre propre commande. Nous avons également dû nous renseigner sur les moteurs pour connaitre la résolution des codeurs ainsi que leurs principales caractéristiques.

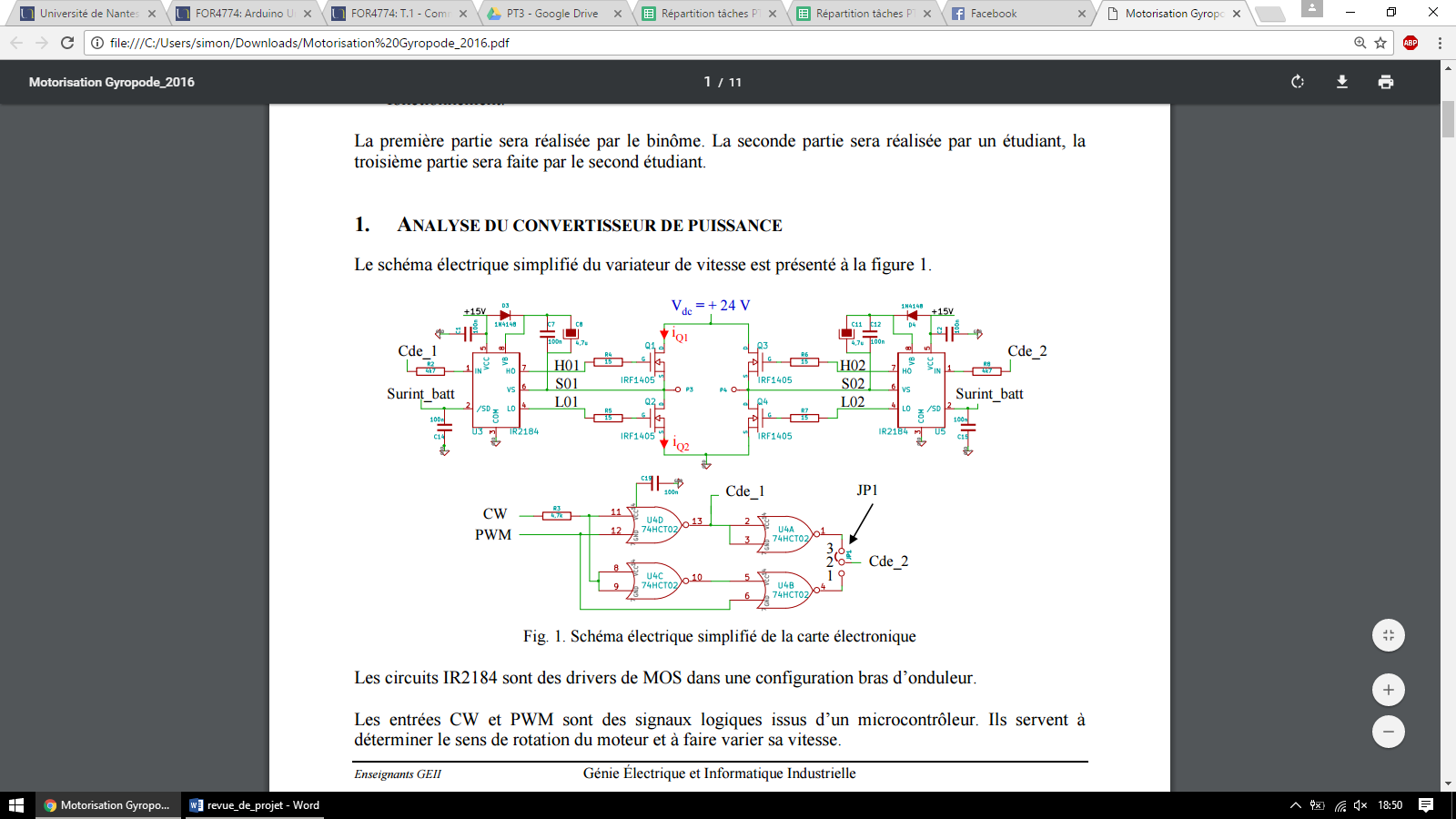


Figure 21 : Schéma électronique de la commande des moteurs

Nous avons, à partir de ses informations, d’un code réalisé par nos soins en TP de 1ère année concernant la commande moteur et pour finir d’un squelette de code fourni au groupe en charge de la commande des moteurs, réalisé le code ci-dessous.

### 6.1.2 Code réalisé

#include "codeurs.h"

volatile int \_\_positionD **=** 0**;**

volatile int \_\_positionG **=** 0**;**

int deltaDroit**[**8**]** **=** **{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**};**

int sommeDroit **=** 0**;**

unsigned char indexDroit **=** 0**;**

long previousPosDroit **=** 0**;**

int deltaGauche**[**8**]** **=** **{**0**,**0**,**0**,**0**,**0**,**0**,**0**,**0**};**

int sommeGauche **=** 0**;**

unsigned char indexGauche **=** 0**;**

long previousPosGauche **=** 0**;**

void initCodeur**()**

**{**

pinMode**(**IND1**,** INPUT**);**

pinMode**(**IND2**,** INPUT**);**

pinMode**(**ING1**,** INPUT**);**

pinMode**(**ING2**,** INPUT**);**

**}**

long getPosition**(**int sens**)**

**{**

**if** **(**sens**==**DROITE**)** **return** \_\_positionD**;**

**else** **if(**sens**==**GAUCHE**)** **return** \_\_positionG**;**

**}**

void getVitesse**(**int **&**droit**,** int **&**gauche**)**

**{**

const long currentPosDroit **=** getPosition**(**DROITE**);**

const int newDeltaDroit **=** **(**currentPosDroit**-**previousPosDroit**);**

deltaDroit**[**indexDroit**]** **=** newDeltaDroit**;**

sommeDroit **+=** newDeltaDroit**;** //ajoute la nouvelle valeur

indexDroit **=** **(**indexDroit**+**1**)** **%** 8**;**

sommeDroit **-=** deltaDroit**[**indexDroit**];** //supprime la plus ancienne

previousPosDroit **=** currentPosDroit**;**

droit **=** **(**sommeDroit**);**

const long currentPosGauche **=** getPosition**(**GAUCHE**);**

const int newDeltaGauche **=** **(**currentPosGauche**-**previousPosGauche**);**

deltaGauche**[**indexGauche**]** **=** newDeltaGauche**;**

sommeGauche **+=** newDeltaGauche**;** //ajoute la nouvelle valeur

indexGauche **=** **(**indexGauche**+**1**)** **%** 8**;**

sommeGauche **-=** deltaGauche**[**indexGauche**];** //supprime la plus ancienne

previousPosGauche **=** currentPosGauche**;**

gauche **=** **(**sommeGauche**);**

**}**

/\* 1ere interruption moteur droit \*/

ISR **(**INT0\_vect**){**

**if(**digitalRead**(**IND1**)** **&&** digitalRead**(**IND2**))** \_\_positionD**++;**

**};**

/\* 2eme interruption moteur droit \*/

ISR **(**PCINT0\_vect**){**

**if(**digitalRead**(**IND2**)** **&&** digitalRead**(**IND1**))** \_\_positionD**--;**

**};**

/\* 1ere interruption moteur gauche \*/

ISR **(**INT1\_vect**){**

**if(**digitalRead**(**ING1**)** **&&** digitalRead**(**ING2**))** \_\_positionG**++;**

**};**

/\* 2eme interruption moteur gauche \*/

ISR **(**PCINT2\_vect**){**

**if(**digitalRead**(**ING2**)** **&&** digitalRead**(**ING1**))** \_\_positionG**--;**

**};**

À la fin du code précédent, nous remarquons l’utilisation d’interruptions sur les moteurs. Ces interruptions sont reliées aux signaux A et B des codeurs présents sur chacun des moteurs et nous permettent de connaitre la position de l’arbre du moteur en fonction du front de l’un des signaux et du niveau de l’autre. Ces interruptions ne font que modifier les variables globales de positions gauche et droite

En utilisant un timer pour appeler une fonction toutes les x ms, on peut aisément calculer la vitesse des moteurs en faisant le rapport entre la variation de position et la variation de temps.

### 6.1.3 Conclusion

Les différentes fonctions précédemment vues nous permettent donc d’obtenir la position et la vitesse de chacun des moteurs pour pouvoir asservir le gyropode. En effet, comme nous pourrons le voir dans le schéma bloc présenté dans la partie suivante, la vitesse et la position de chaque moteur entre en compte dans l’asservissement en vitesse linéaire et en rotation autour de y du gyropode.

## Asservissement des moteurs en vitesse et position :

### 6.2.1 Cahier des charges

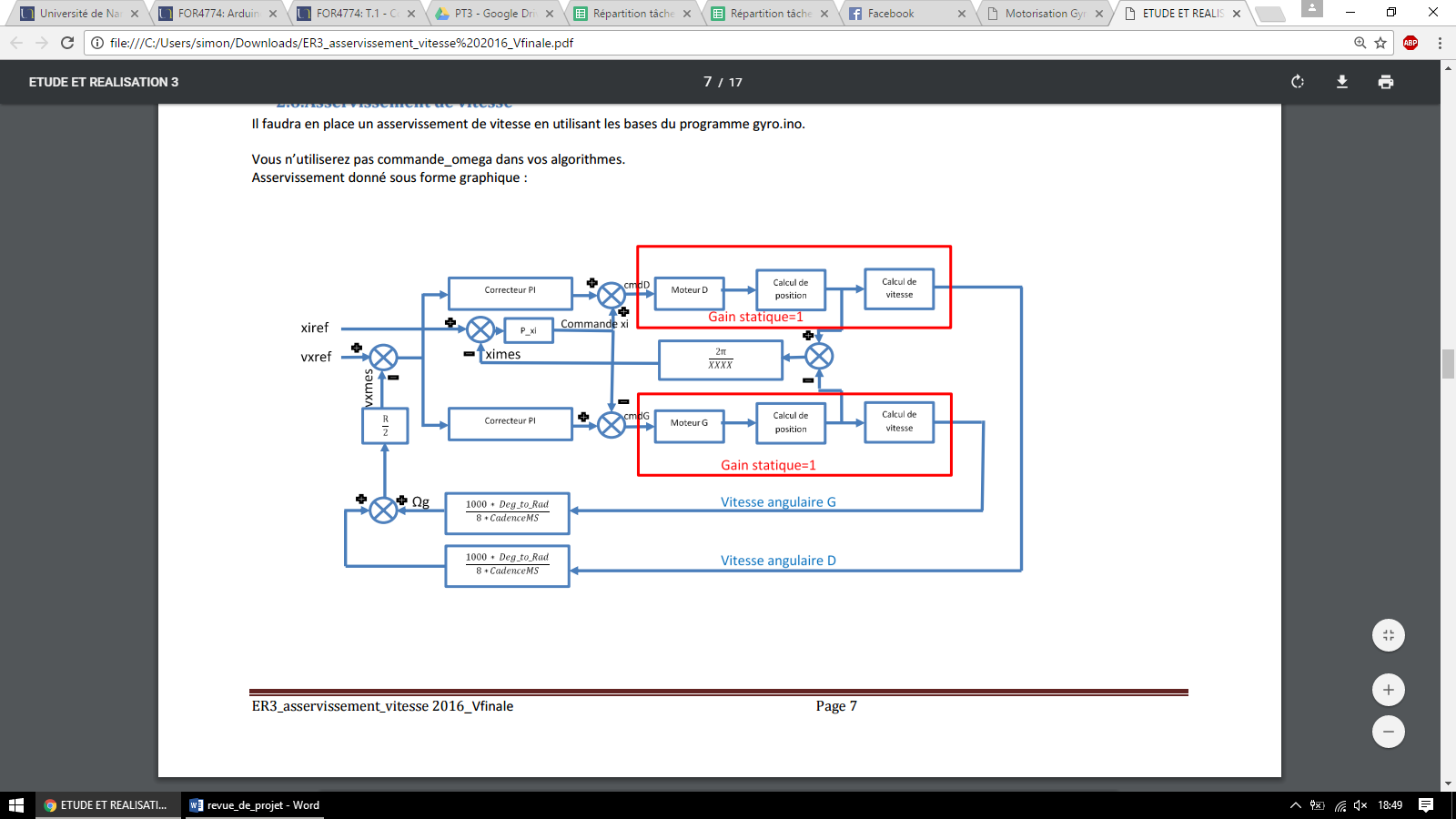


Figure 22 : Schéma bloc de l’asservissement des moteurs

Le schéma bloc précédent nous montre le comportement de l’asservissement du gyropode en vitesse linéaire et en rotation autour de y.

On y retrouve tous d’abord les deux moteurs, sur lesquels nous prenons la position pour calculer l’erreur en rotation, puis la vitesse qui nous permet de calculer l’erreur en vitesse. Un correcteur PI est appliqué sur la vitesse, alors qu’un simple correcteur P est appliqué sur la rotation. La commande en vitesse est la même pour les deux moteurs, alors que la commande en rotation est opposée pour les deux moteurs.

Dans l’asservissement final, nous ajouterons aussi l’asservissement en rotation autour de x grâce à la vitesse angulaire du gyropode autour de cette axe et l’angle de ce dernier entre lui et la terre.

### 6.2.2 Code réalisé

#include "define.h" // Fait appel à un define.h ou toutes les constantes sont définies

double angle**=**0**;** // Variable globale de l’angle

void setup**()** **{**

Serial**.**begin**(**115200**);** // Initialisation du serial

pinMode**(**12**,** OUTPUT**);** // pin utilisée pour une led témoin sur le robot

pinMode**(**11**,** OUTPUT**);** // idem

digitalWrite**(**12**,**1**);** // mise à 1 de la led témoin

digitalWrite**(**11**,**1**);** // idem

Serial**.**println**(**"Init\_lcd"**);** // affichage en serial (infos de debug )

initLcd**();**

displayCalibration**();** // affichage sur le LCD avec mise en forme

Serial**.**println**(**"Init\_angle"**);**// affichage en serial (infos de debug )

initAngle**();**

Serial**.**println**(**"Init\_codeurs"**);**// affichage en serial (infos de debug )

initCodeur**();**

Serial**.**println**(**"Init\_moteurs"**);**// affichage en serial (infos de debug )

initMotors**();**

Serial**.**println**(**"Init\_timer"**);**// affichage en serial (infos de debug )

FlexiTimer2**::**set**(**CADENCE\_MS**,** 1 **/** 1000.**,** itAsservissement**);** // résolution timer = 1 ms

FlexiTimer2**::**start**();**

Serial**.**println**(**"Inits\_ALL\_OK"**);**

displayWelcome**();**

**}**

void loop**()**

**{**

angleAcquisition**();**

displayAngle**(**angle**);** // affichage sur le LCD avec mise en forme

displayClock**();** // affichage sur le LCD avec mise en forme

//Serial.println(angle\*RAD\_TO\_DEG);

**}**

//sans Serial.print(..), il faut environ 200us pour la boucle d'asservissement.

void itAsservissement**()**

**{**

int vitesseD**,** vitesseG**;** //ticks sur les 80 dernieres ms.

getVitesse**(**vitesseD**,**vitesseG**);**

double omegaDroit **=** 1000.**\***vitesseD**/(**8**\***CADENCE\_MS**)\***DEG\_TO\_RAD**;** //en rad/s (si 10ms)

double omegaGauche **=** 1000.**\***vitesseG**/(**8**\***CADENCE\_MS**)\***DEG\_TO\_RAD**;** //en rad/s (si 10ms)

const double vxref **=** CONS\_VIT**;**

const double vxmes **=** **(**omegaDroit **+** omegaGauche**)** **\*** R **/** 2**;**

const double gyroRate **=** getGyroRate**();**

//correcteur P sur l'orientation du robot (direction): angle Xi

//expérimentalement, un tour => delta de 1800 ticks sur les codeurs

double P\_xi **=** KP**;**

const double xiMes **=** **(**getPosition**(**DROITE**)** **-** getPosition**(**GAUCHE**))\*(**2**\***PI**)/**RES\_COD**;** //en rad.

//angle du robot (en radian)

double xiref **=** CONS\_ORI**;**

const double commande\_xi **=** P\_xi **\*** **(**xiref **-** xiMes**);**

//PI sur la vitesse d'avance

static double I\_vx **=** 0.0**;**

const double P\_vx **=** KP\_VX **\*** **(**vxref **-** vxmes**);**

const double commande\_vx **=** P\_vx **+** I\_vx**;**

I\_vx **=** I\_vx **+** KI\_VX **\*** **(**CADENCE\_MS**/**1000.**)** **\*** **(**vxref **-** vxmes**);**

//correcteur vertical

// => par rapport à la vitesse de rotation et l'angle du robot.

const double gyroRateRef **=** CONS\_VIT\_OMEGA**;** //consigne de vitesse de rotation.

const double AngleRef **=** CONS\_ORI\_OMEGA**;** //consigne d'angle par rapport à la verticale

const double P\_omega **=** KP\_OMEGA **\*** **(**gyroRateRef **-** gyroRate**);**

angle **=** getCompAngle**();**

const double I\_omega **=** KI\_OMEGA **\*** **(**AngleRef **-** angle**);**

const double commande\_omega **=** P\_omega **+** I\_omega**;**

const double commandeDroit **=** commande\_vx **-** commande\_omega **+** commande\_xi**;**

const double commandeGauche **=** commande\_vx **-** commande\_omega **-** commande\_xi**;**

setMotorDroit**(**commandeDroit**);**

setMotorGauche**(**commandeGauche**);**

**}**

### 6.2.3 Conclusion

Dans cette partie, nous avons surtout utilisé le code qui nous a été fourni ainsi que les fonctions que nous avions déjà codées. Néanmoins, nous avons enlevé toutes les constantes d’asservissements du fichier principal pour les déclarer dans un fichier à part afin d’aider à la compréhension du programme.

Nous avons également pu remarquer que l’asservissement mis en œuvre dans le programme était totalement différent du schéma bloc présent dans le dossier de présentation du projet donné aux groupes s’occupants de l’asservissement.

## Construction du gyropode

### 6.3.1 Cahier des charges



Afin de tester notre centrale inertielle et également l’asservissement moteur associé que nous venons de coder, nos professeurs nous ont proposé la réalisation d’un « mini » gyropode maquette utilisée pour les tests d’asservissement, affichage LCD, avec extensions I2C possibles pour commande Bluetooth, le tout programmable avec une petite interface USB/serial Arduino.

### 6.3.2 Réalisation

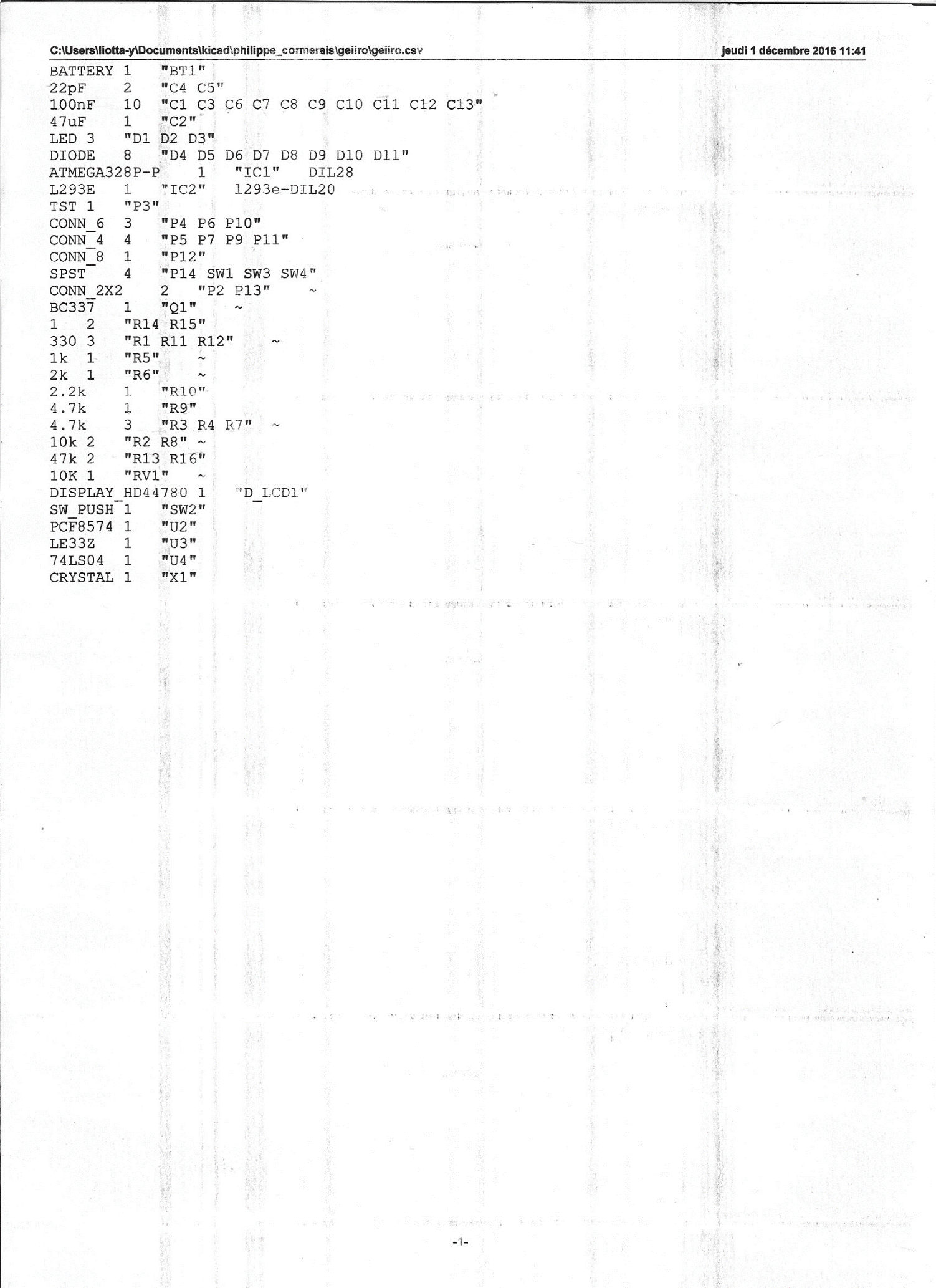


Figure 23 : Liste du matériel pour la réalisation du gyropode

Pour la partie réalisation, nous avons simplement suivi à la lettre les consigne du développeur de la maquette Mr Philippe Cormerais. Nous nous sommes également inspiré du prototype de notre gyropode.

Pour la partie technique ce gyropode est principalement constitué d’une [Atmega328P](http://www.atmel.com/devices/ATMEGA328P.aspx) qui communique avec un afficheur LCD par le biais d’un contrôleur I2C. Nous y retrouvons également une puissante alimentation à découpage 12v/3.3V. Et enfin un pont en H pour la commande des moteurs. Les autres composants sont principalement présents pour adapter des tensions, pour protéger le système ou pour d’autres raisons mineures (découplage etc).

Pour finir, il est important de préciser que lors de la réalisation des tests de la maquette, nous avons réalisé une mauvaise manipulation ; nous avons envoyé le 12v de la batterie directement dans le bus I2C, ce qui a eu pour effet de griller le contrôleur de bus I2C de l’afficheur LCD ainsi que les broches associées de l’[Atmega328p](http://www.atmel.com/devices/ATMEGA328P.aspx). Nous avons dû les remplacer tous deux.

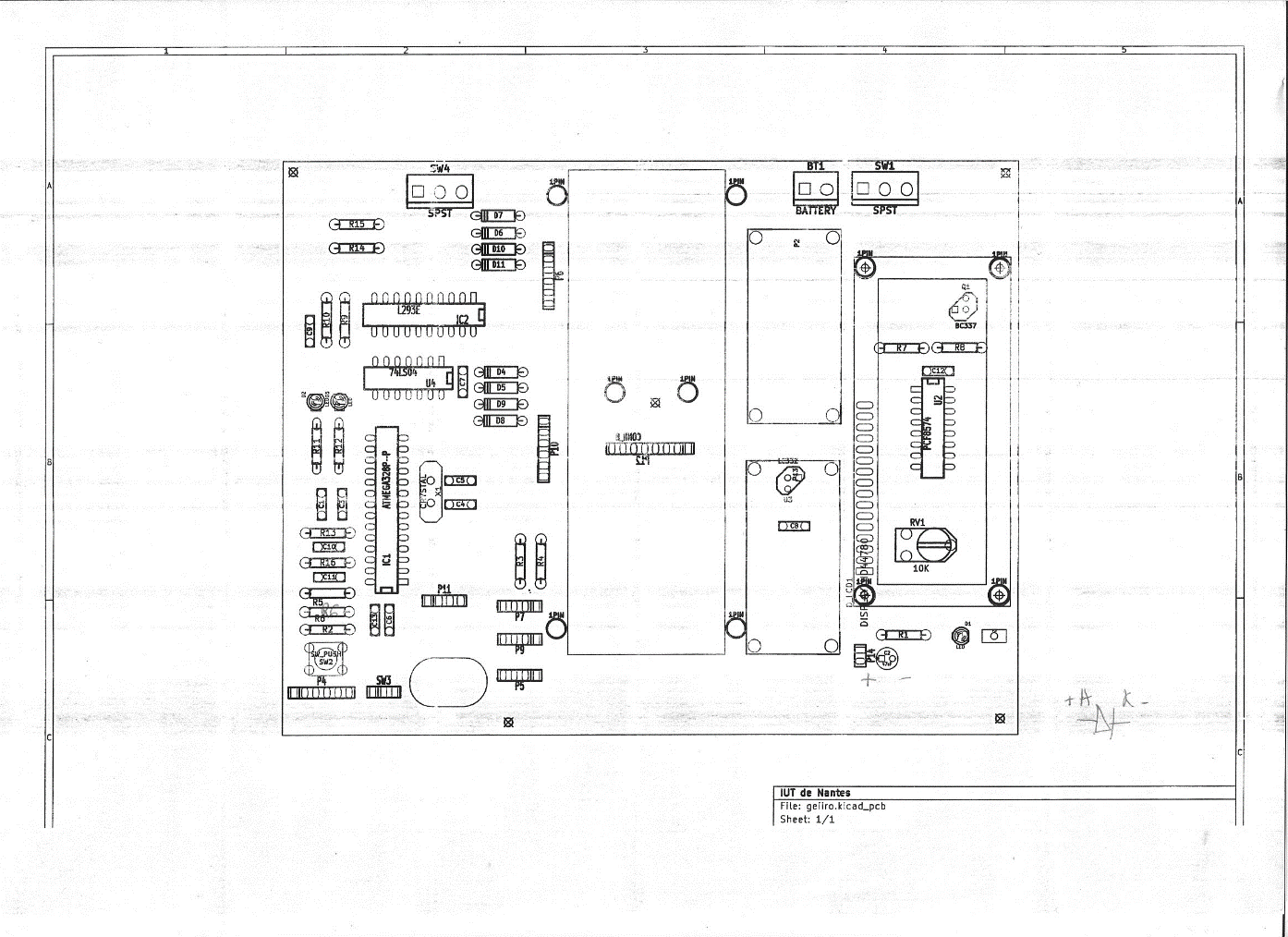


Figure 25 : Schéma d’implantation de la carte de pilotage du gyropode

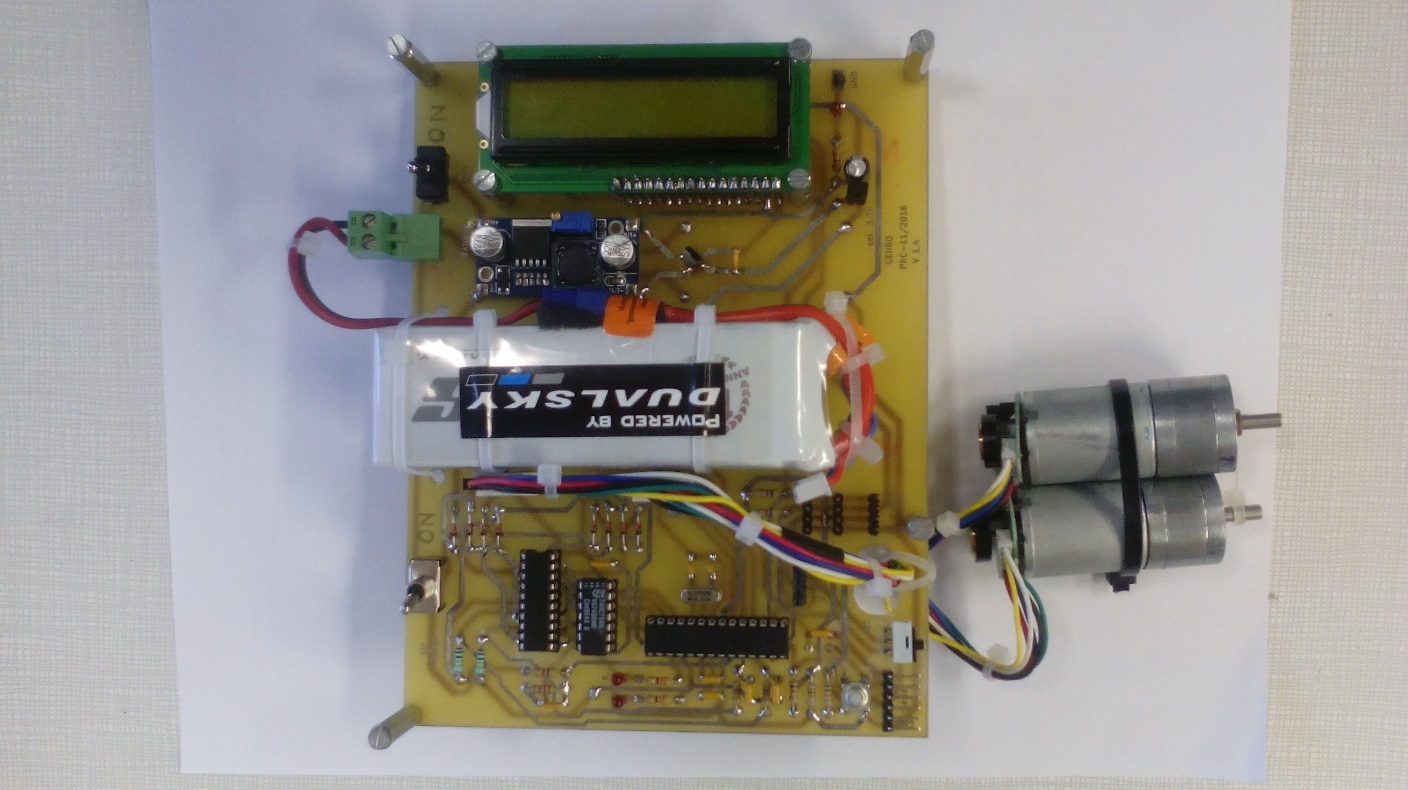


Figure 25 : Réalisation finale du gyropode

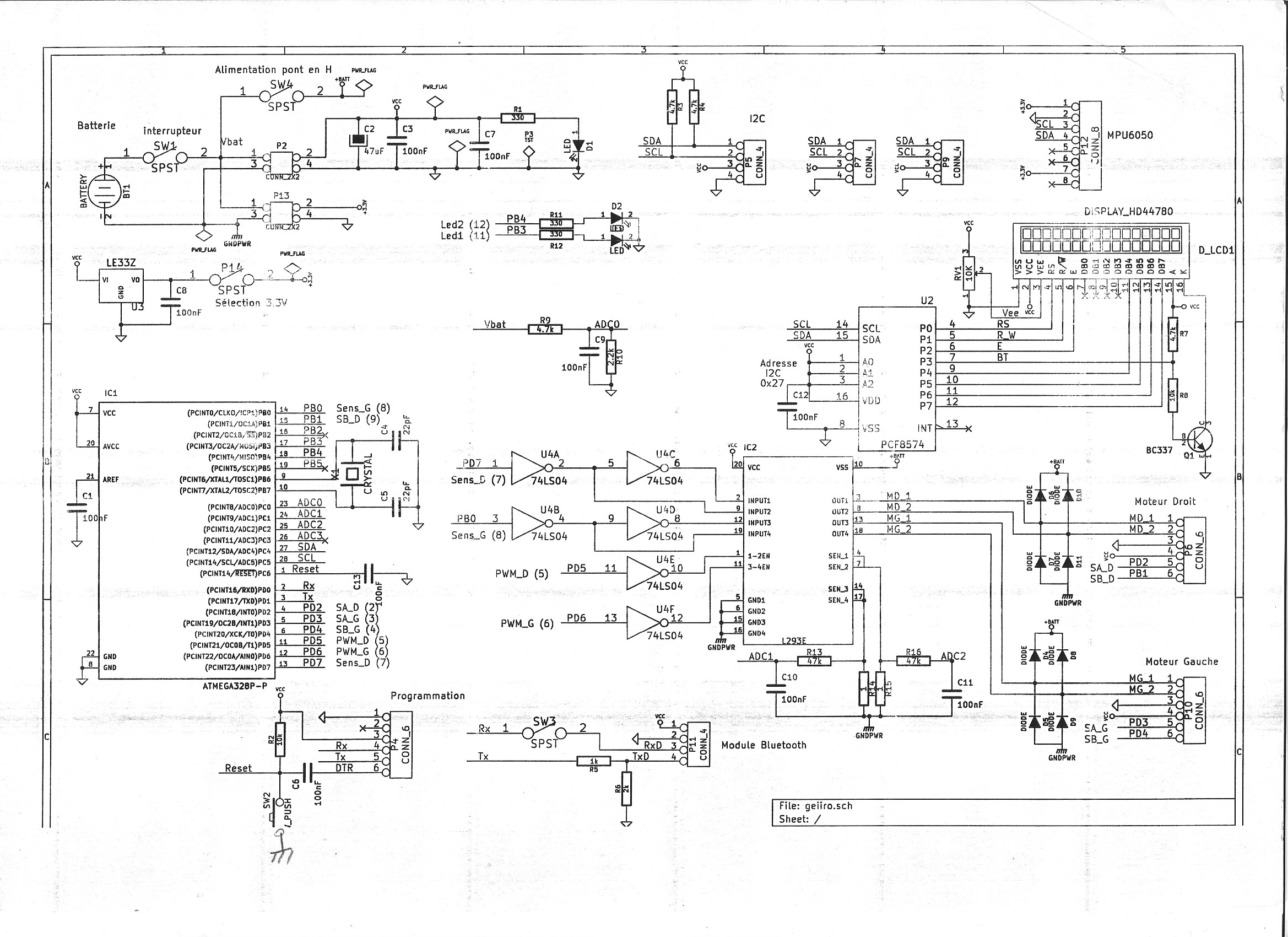
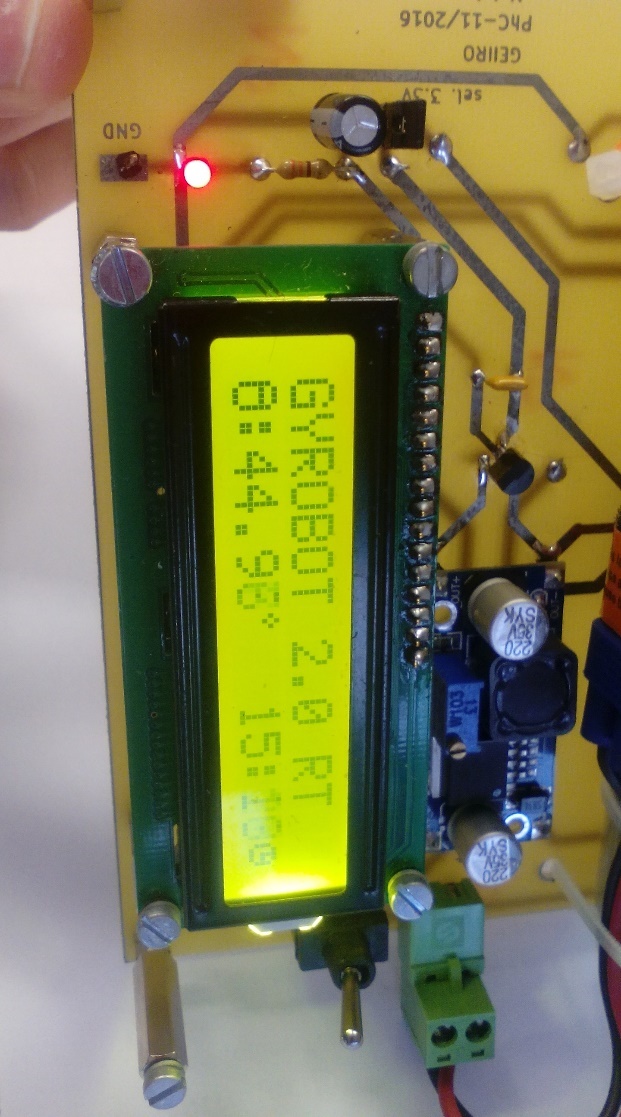
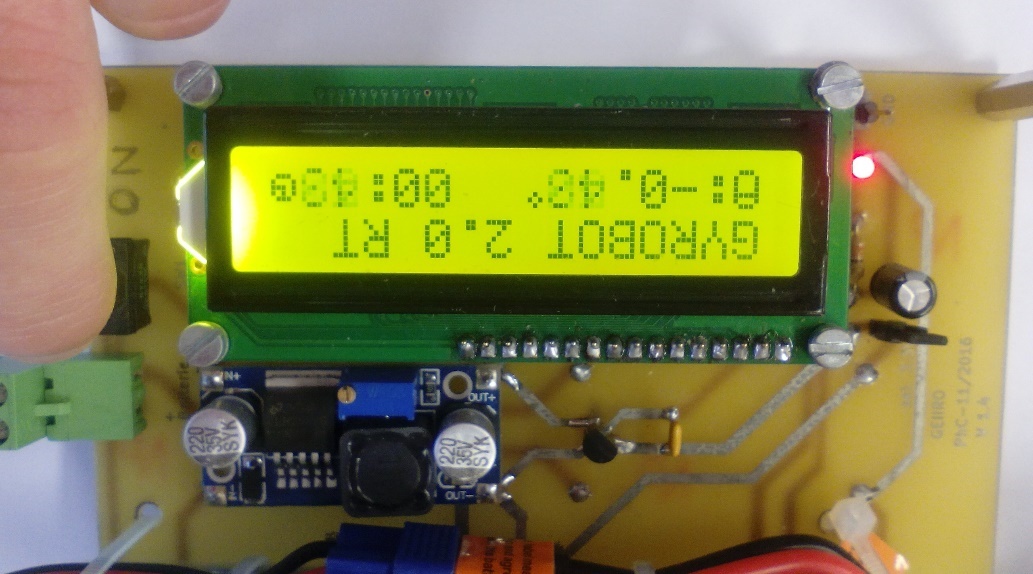


Figure 26 : Schéma électronique de la carte de pilotage du gyropode

### 6.3.3 Test d’affichage LCD

A des fins de finissions (et un peu d’amusement aussi) nous avons réalisé un affichage LCD complexe et sympathique pour tout le fonctionnement du robot.



1

2

3

4

Figure 27 : Affichage LCD

Lors de l’allumage ou d’un reset du gyropode celui-ci entre dans une phase de calibration avec l’affichage en conséquence. (Ici en 1). Juste après (ici en 2), un message d’accueil chaleureux s’affiche pendant 2 secondes avec les principaux contributeurs du projet. (Les petites horloges et les cœurs ont été récupérée dans la librairie de l’afficheur LCD) Suite à cette procédure d’accueil (environ 3 secondes en tout) le « GYROBOT » entre en action (ici en 3) On observe le nom du robot, ainsi que sa version en fixe ; ici 2.0 RT pour Real Time. En fait cet affichage était prévu pour la version temps réel mais celle-ci n’a jamais été implémentée dans le robot.

Au-dessous de message fixe on observe deux informations : l’angle et une horloge. Cette dernière sert juste de témoins de temps de fonctionnement. L’angle quant à lui est symbolisé par un Thêta que nous avons réalisé nous-mêmes avec une simple matrice binaire. Cet angle est affiché en degré et nous avons également réalisé le « ° » nous-mêmes. Pour finir nous observons en 4 un angle de 45° qui confirme le bon fonctionnement de l’afficheur avec une précision inférieure à 1° malgré une fréquence de rafraichissement du LCD très élevé

### 6.3.4 Suite des tests, difficultés rencontrées

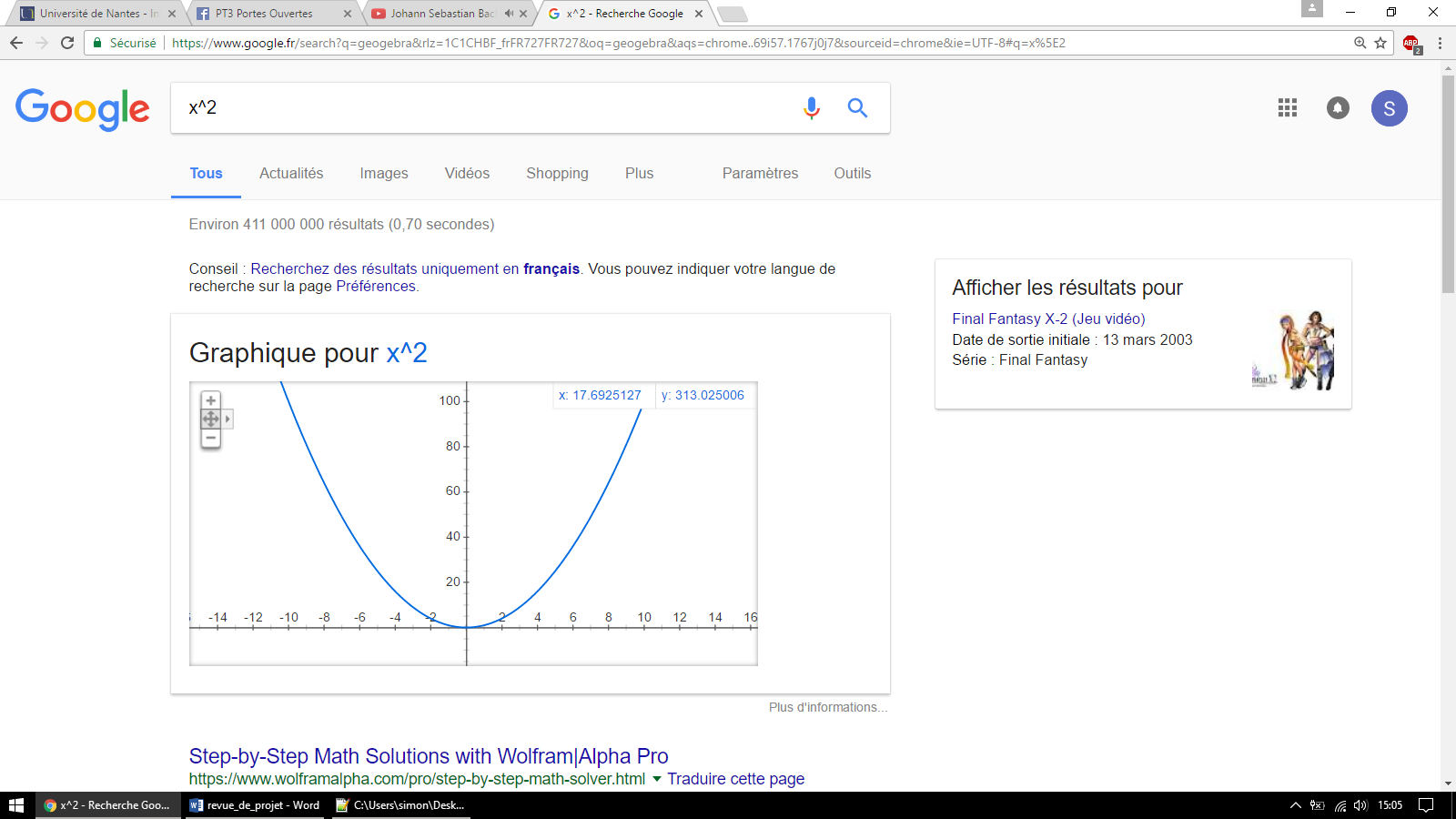
Le gyropode dans son ensemble fonctionne très bien, les moteurs fonctionnent et leur vitesse est correctement variable. L’afficheur I2C LCD fonctionne parfaitement. Et l’affichage de l’angle sur l’afficheur LCD est spectaculairement précis de -170° à 170° soit bien au-delà de sa plage de fonctionnement classique. L’angle reste également précis et correctement calibré même si on impose des variations brutales et répétés au gyropode (donc fonctionnel en haute fréquence sur l’accéléromètre) ou des périodes d’inactivités prolongés (donc la dérive intrinsèque du gyroscope est correctement compensée par le filtre complémentaire.)

Seulement il nous manque la carcasse métallique du gyropode ainsi que les roues pour attester réellement du bon fonctionnement.

Nous avons observé un seul problème de fonctionnement sur le gyropode :

La réaction des moteurs à la variation d’angle devrait normalement ressembler (de très loin) à un x². Soit plus on s’éloigne de de l’angle 0 plus les moteurs devraient tourner vite pour ramener le gyro vers un angle nul :

Figure 28 : Variation V(angle) théorique en x²



Vitesse des moteurs (non à l’échelle)

Variation d’angle (deg)

Seulement, la vraie courbe que nous observons ressemble à abs(cos(x/8))  (déterminée empiriquement):



Vitesse des moteurs (non à l’échelle)

Variation d’angle (deg)

Figure 29 : Variation V(angle) Reelle en abs(cos(x/8))

Soit une vitesse non nulle quand l’angle est nul (illogique car le robot est sensé se considérer comme stable à 0deg) et des moteurs qui s’arrêtent à des valeurs précises symétriques périodiques : … -36, -12, 12, 36 … Notre gyropode se considère donc comme stable aux alentours de ses points.

### 6.3.5 Conclusion

Nous observons donc un dysfonctionnement évident qui pourrais venir de beaucoup de facteurs. Seulement sans carcasse et sans roues nous ne pouvions pas réellement observer les réactions du robot et observer si celui-ci était capable de se stabiliser autour de ses points ou non et ainsi pouvoir identifier clairement les facteurs de dysfonctionnement.

## 6.4 Passage du gyropode en temps réel

### 6.4.1 Présentation du temps réel

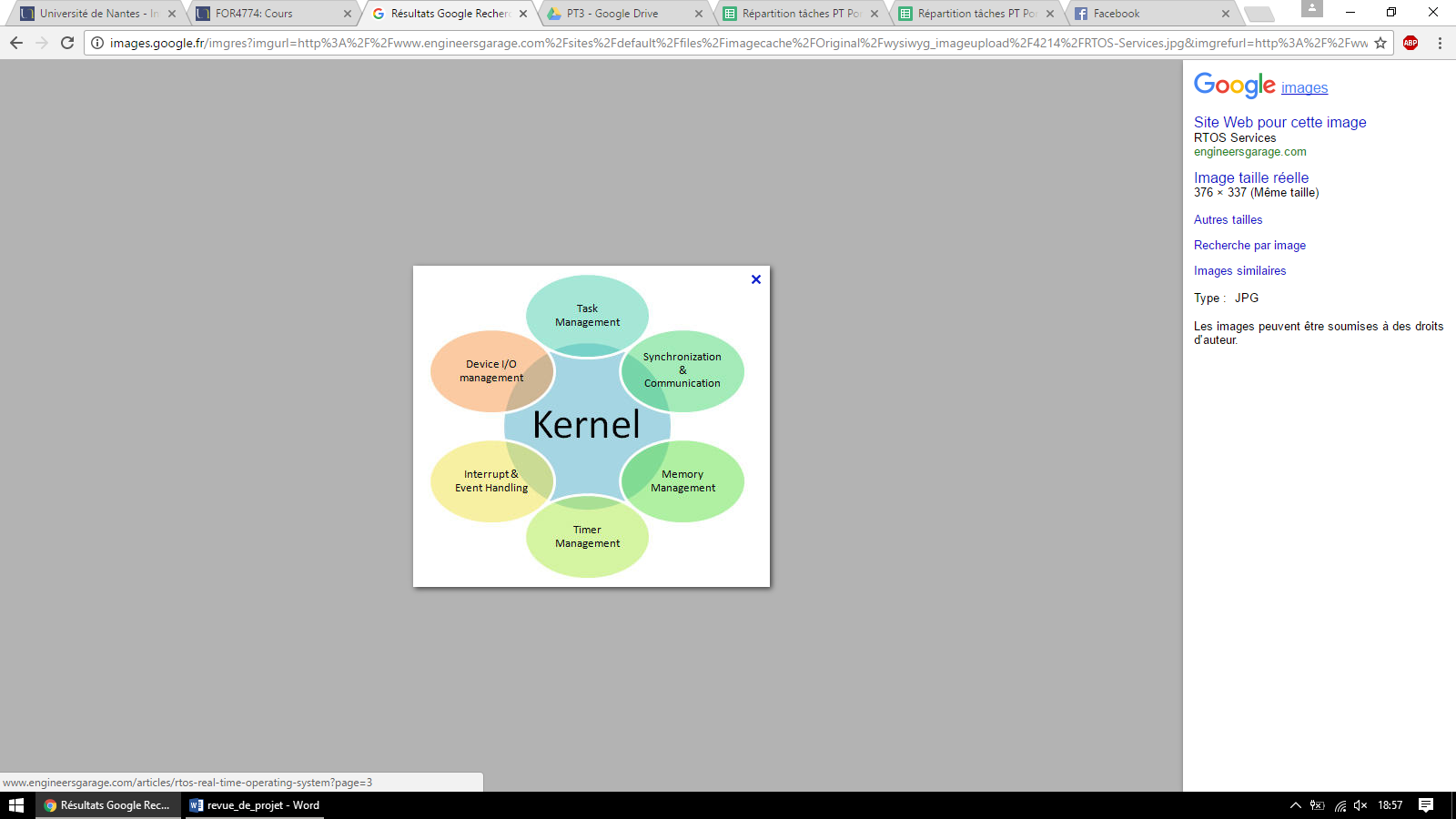
[Trampoline](http://trampoline.rts-software.org/?lang=fr) est un [système d’exploitation temps-réel](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d'exploitation_temps_r%C3%A9el) (ou exécutif) à source ouvert qui, une fois certifié, pourrait être conforme à la spécification [OSEK/VDX](https://fr.wikipedia.org/wiki/OSEK/VDX). Trampoline est développé principalement par des membres de l’équipe Systèmes Temps-Réel de l’[IRCCyN](http://www.irccyn.ec-nantes.fr/fr/) (Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou et Yvon Trinquet).

Figure 30 : Synoptique de Trampoline

Trampoline est distribué sous la licence [LGPL v2.1](http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html). Trampoline fonctionne maintenant sur 7 plateformes :

* Système POSIX
* Système POSIX avec Viper 2
* Infineon C166 (projet KEIL)
* Freescale HCS12 (merci à Geensys)
* Freescale/IBM PowerPC
* ARM (olimex-lpc-e2294 ; simtec-eb675001 ; NXT2.0 ; SAMD21)
* Atmel 8 bits AVR

### 6.4.2 Objectifs

Le passage en temps réel a pour but de séparer efficacement les différentes taches internes du gyropode et de les cadencer avec exactitude. Exemple : grâce à trampoline nous allons pouvoir séparer proprement les taches qui :

* Affiche le temps de fonctionnement du gyropode
* Calcule l’angle
* Affiche cet angle
* Calcule l’asservissement moteur

Et les appeler aux bon intervalles précis et requis pour chacune des taches.

Le second avantage est de rendre ses tâches préemptibles et priorisables. Ainsi l’affichage du temps de fonctionnement du gyropode (non vital) pourra être interrompu si un calcul d’angle ou d’asservissement est nécessaire à cet instant (beaucoup plus vital). Cela rend notre système beaucoup plus fiable et performant.

Enfin, trampoline rend notre système beaucoup plus adaptable. Exemple : si nous souhaitons ajouter un module Bluetooth sur notre robot pour renvoyer des données en temps réel sur un PC ou même contrôler le gyropode avec des ordres de vitesse et de direction. Il nous suffit de rajouter une alarme avec une période adaptée, de définir si la communication Bluetooth est plus vitale que l’asservissement, les calculs d’angles ou que l’affichage (c’est assez évident) et de définir une nouvelle tâche qui exécutera le code associé. Tout ceci sans toucher aucune ligne de code de l’ancien système. Toutes les taches sont parfaitement indépendantes et sont juste exécutée en fonction de leur période d’activation et de priorité.

Ainsi on passe de (par priorité) :

calculAngleTask < asservissementTask < displayAngleTask < displayTimeTask

à:

calculAngleTask < asservissementTask < **BluetoothTask** < displayAngleTask < displayTimeTask

Nous sommes donc capables d’intercaler de manière propre, n’importe quelle fonction dans notre système sans en altérer le fonctionnement.

### 6.4.3 Code réalisé

Code OIL réalisé et ajouté au projet pour passer le tout en temps réel. Les autres fichiers (en C/C++) ont été un peu modifiés dans leur syntaxe d’entête ex : TASK(calculAngleTask) mais le contenu principal n’est pas changé.

/\* Running the following example:

\* just call goil a first time using the command line:

\* goil --target=avr/arduino --templates=../../../../goil/templates/ blink.oil

\* you may have to update ARDUINO section (l.20) for your configuration.

\*

\* Then a Makefile is generated. Just type:

\* make -s

\* To clean the project, call:

\* make clean

\* It will remove all generated files, except the Makefile.

\* to flash to the Arduino:

\* make -s flash

\* It will use the PORT defined in the ARDUINO part.

\*/

////////////////////////////// Definitions systems du code OIL //////////////////////

OIL\_VERSION **=** "2.5" **:** "test" **;**

CPU test **{**

OS config **{**

STATUS **=** STANDARD**;**

ARDUINO **=** TRUE **{**

SERIAL **=** TRUE**;** //use the serial interface

PORT **=** "/dev/tty.usbmodem1411"**;** //serial port (flash)

**};**

BUILD **=** TRUE **{**

TRAMPOLINE\_BASE\_PATH **=** "../../../.."**;**

APP\_NAME **=** "gyroRTS"**;**

APP\_SRC **=** "gyroRTS\_V2.cpp"**;**

APP\_SRC **=** "angle.cpp"**;**

APP\_SRC **=** "codeurs.cpp"**;**

APP\_SRC **=** "lcd.cpp"**;**

APP\_SRC **=** "moteurs.cpp"**;**

CPPCOMPILER **=** "avr-g++"**;**

COMPILER **=** "avr-gcc"**;**

LINKER **=** "avr-gcc"**;**

ASSEMBLER **=** "avr-gcc"**;**

SYSTEM **=** PYTHON**;**

**};**

SYSTEM\_CALL **=** TRUE**;**

**};**

APPMODE std**{};**

////////////////////////////// Definitions des alarmes /////////////////////////////

ALARM asservissementAlarm **{** // Alarme de déclenchement de l’asservissement T=10ms

COUNTER**=** SystemCounter**;**

ACTION **=** ACTIVATETASK **{**

TASK **=** asservissementTask**;**

**};**

AUTOSTART **=** TRUE **{**

ALARMTIME **=** 10**;**

CYCLETIME **=** 10**;**

APPMODE **=** std**;**

**};**

**};**

ALARM displayAngleAlarm **{** // Alarme de déclenchement de l’affichage LCD de l’angle T=100ms

COUNTER**=** SystemCounter**;**

ACTION **=** ACTIVATETASK **{**

TASK **=** displayAngleTask**;**

**};**

AUTOSTART **=** TRUE **{**

ALARMTIME **=** 100**;**

CYCLETIME **=** 100**;**

APPMODE **=** std**;**

**};**

**};**

ALARM displayTimeAlarm **{** // Alarme de déclenchement de l’affichage du temps de fonctionnement : T=1000ms (format 00:00 en seconds)

COUNTER**=** SystemCounter**;**

ACTION **=** ACTIVATETASK **{**

TASK **=** displayTimeTask**;**

**};**

AUTOSTART **=** TRUE **{**

ALARMTIME **=** 1000**;**

CYCLETIME **=** 1000**;**

APPMODE **=** std**;**

**};**

**};**

ALARM calculAngleAlarm **{** // Alarme de déclenchement du calcul de l’angle T=10ms

COUNTER**=** SystemCounter**;**

ACTION **=** ACTIVATETASK **{**

TASK **=** calculAngleTask**;**

**};**

AUTOSTART **=** TRUE **{**

ALARMTIME **=** 10**;**

CYCLETIME **=** 10**;**

APPMODE **=** std**;**

**};**

**};**

////////////////////////////// Definitions des taches périodiques //////////////////////

TASK calculAngleTask **{** // Tache de calcul de l’angle

PRIORITY **=** 1**;**

AUTOSTART **=** FALSE**;**

ACTIVATION **=** 1**;**

SCHEDULE **=** FULL**;**

STACKSIZE **=** 256**;**

**};**

TASK asservissementTask **{** // Tache de l’asservissement

PRIORITY **=** 2**;**

AUTOSTART **=** FALSE**;**

ACTIVATION **=** 1**;**

SCHEDULE **=** FULL**;**

STACKSIZE **=** 256**;**

**};**

TASK displayAngleTask **{** // Tache de l’affichage de l’angle

PRIORITY **=** 5**;**

AUTOSTART **=** FALSE**;**

ACTIVATION **=** 1**;**

SCHEDULE **=** FULL**;**

STACKSIZE **=** 256**;**

**};**

TASK displayTimeTask **{** // Tache de l’affichage du temps de fonctionnement

PRIORITY **=** 6**;**

AUTOSTART **=** FALSE**;**

ACTIVATION **=** 1**;**

SCHEDULE **=** FULL**;**

STACKSIZE **=** 256**;**

**};**

////////////////////////////// Definitions des ISR ///////////////////////////////////////////

ISR ISRMoteurDroitA **{** // Interruption du moteur Droit A

CATEGORY **=** 2**;**

PRIORITY **=** 20**;**

STACKSIZE **=** 256**;**

SOURCE **=** "INT0\_vect"**;**

**};**

ISR ISRMoteurDroitB **{** // Interruption du moteur Droit B

CATEGORY **=** 2**;**

PRIORITY **=** 20**;**

STACKSIZE **=** 256**;**

SOURCE **=** "PCINT0\_vect"**;**

**};**

ISR ISRMoteurGaucheA **{** // Interruption du moteur Gauche A

CATEGORY **=** 2**;**

PRIORITY **=** 20**;**

STACKSIZE **=** 256**;**

SOURCE **=** "INT1\_vect"**;**

**};**

ISR ISRMoteurGaucheB **{** // Interruption du moteur Gauche B

CATEGORY **=** 2**;**

PRIORITY **=** 20**;**

STACKSIZE **=** 256**;**

SOURCE **=** "PCINT1\_vect"**;**

**};**

**};**

### 6.4.4 Conclusion, problèmes rencontrés

Notre code OIL ainsi que l’ensemble du projet se compile correctement, et semble fonctionner. Seulement nous ne sommes pas parvenus à flasher notre gyropode avec ce dernier. En effet, en donnant notre code temps réel à Michaël Briday, l’un des principaux contributeurs au projet trampoline, qui possède un environnement de développement parfaitement opérationnel pour utiliser trampoline, notre projet semblait se compiler, mis à part le fait qu’il fallait ajouter toutes les bibliothèques d’arduino à l’environnement oil.

Mais pour obtenir un environnement de développement similaire, il nous fallait emprunter un chemin semé d’embuches. Il se trouve que le compilateur Goil ne fonctionne correctement uniquement sur les système Unix, il nous fallait donc créer une machine virtuelle sous Ubuntu, avec les problèmes de compatibilités pour faire passer les ports com à la VM. Ensuite, Il nous fallait compiler le compilateur Goil, car il n’existe pas en précompilé, mais les ordres de compilations sont écrits en Python, langage de programmation qui nous était totalement inconnu jusqu’à ce jour. Mais, suite à une mise à jour des sources datant du 30 décembre 2016, une erreur c’était glissée dans ces dernière, il nous était donc impossible de compiler le compilateur. La fin du projet approchant, nous avons donc décidé de nous arrêter ici pour le passage en temps réel, néanmoins, vous pourrez trouver le code correspondant en annexe pour une utilisation ultérieure.

Cette partie est résumée par l’équation suivante :

# Conclusion :

Ce projet nous as permis d’assimiler nombre de nouveaux concepts. Tout d’abord en traitement du signal, filtrage numérique et en calculs différentiels numériques avec l’acquisition et le calcul de l’angle. Nous avons également pu nous confronter à un BUS nouveau (ou presque) ; le bus I2C pour communiquer avec la centrale inertielle. Par la suite, nous avions beaucoup pris d’avance sur les autres groupes par conséquent nous n’avons pas pu nous associer avec un groupe d’asservissement. Nous avons dû le réaliser nous-même et mettre en œuvre nos compétences concernant les moteurs et l’asservissement numérique vu en première et deuxième année. Ensuite nous avons réalisé le petit gyropode pour nous servir de maquette et tester notre système. Nous avons dû utiliser nos compétences en réalisations techniques et en tests de fonctionnement. Pour finir nous avons voulu optimiser notre système en passant le tout sur la plateforme OS temps réel trampoline pour rendre le système plus performant, fiable et adaptable, et dans l’optique d’intégrer le Bluetooth et d’autres fonctions comme l’affichage du temps de fonctionnement sur l’afficheur LCD. Cette dernière étape nous as permis de mettre en œuvre les différents outils acquis dans le module temps réel du semestre 3 ainsi que des capacités avancés de programmation (C,C++,OIL,Python), de génération et gestion de machines virtuelles ou encore de programmation sous linux malgré que nous n’ayons pas pu finaliser cette partie.

# Remerciements :

Nous tenons à remercier Mr. Bruno Feuvrie, notre référent pour ce projet, pour les conseils qu’il a pu nous fournir vis à vie de l’utilisation de la centrale inertielle, et du traitement des informations en étant issues. Nous remercions également Mr. Michaël Briday pour son aide très précieuse dans le passage de notre projet en temps réel. Nous adressons également nos remerciements à Mr. Philipe Cormerais, responsable du module, qui a bien voulu répondre à nos questions envers le projet dans sa globalité et pour le développement de la maquette du gyropode.

# Fourni avec le compte rendu :

* Le code final du gyropode réalisé par nos soins avec :
  + Asservissements moteurs
  + Affichage LCD en I2C
  + Calcul d’angle et filtrage
* Le même code (ou presque) en version 2.0 Temps réel :
  + Tache d’asservissement
  + Tache d’affichage LCD
  + Tache de calcul d’angle
  + Tache d’affichage du temps de fonctionnement
  + Code OIL associé