télécom saint-étienne

# Contrôleur d'attitude

SABIR Ilyass et YU Jean-François

Rapport technique de projet d'électronique

# Table des matières

# 1 INTRODUCTION:

Ce projet s'inscrit dans le cadre du module "Projets d'application électronique" pendant le 6ème semestre, qui permet aux étudiants de découvrir plus de technologies et outils, ainsi de pratiquer leurs savoirs techniques. L'idée devant ce projet est réaliser un contrôleur d'attitude en utilisant une carte AltiMu10 qui va déterminer le positionnement du robot dans l'espace en temps réel, et une carte Teensy qui permettra de récupérer les données.

D'une manière plus générale, le développement des objets en mouvement (les robots, les drones...) nécessitent le contrôle de leurs mouvements de manière la plus efficace possible, notamment pour récupérer les données numériques caractérisant son mouvement (accélération, vitesse angulaire. . . ), ce qui nous permet aussi de contrôler le mouvement à distance, visualiser les variations rapides qu'on ne peut pas les détecter en tant qu'être humain, contrôler automatiquement le mouvement (par des algorithmes qui traitent les données récupérables).

Malheureusement puisque le mouvement d'un objet peut varier la distance entre le capteur des données et ce qui va récupérer ces données (le temps de retard augmente) ce qui perturbe en plus le contrôle instantané des données, sans oublier le problème de stabilisation des données vu que les données récupérables d'une côté varient rapidement (instables) et d'autre côté contient une partie de bruit non utile, ce qui oblige à utiliser des filtres spécifiques pour stabiliser les données le maximum possible.

Ce rapport présente une démarche de travail qu'on a fait durant toutes les séances de TP, et explication des solutions techniques mises en place.

**Figure 1.** Exemples d'utilisation des contrôleurs d'attitudes

# 2 CAHIER DE CHARGE:

Le but de ce projet est de réaliser un contrôleur d'attitude destiné à une installation sur un robot roulant de type «rover». Ce système permet de visualiser la position du robot dans l'espace en temps réel : lacet, roulis, tan-gag ainsi que sa position par rapport au nord. Le capteur, une carte AltiMu10 qui intègre un accéléromètre, un altimètre, un gyromètre et un magnétomètre, sera associé à une carte Teensy qui permettra de récupérer les données. La datavisualisation de la position du robot se fera avec Processing.

# 3 Présentation de la carte Teensy:

Les cartes de développement Teensy sont des cartes électroniques embarquant un microcontrôleur et la connectique nécessaire pour travailler avec, exactement comme un Arduino ou une STM32. Il existe plusieurs modèles de Teensy: LC, 3.2, 3.5, 3.6 Tous ces modèles sont prévus pour être utilisés de la même manière qu'un Arduino, n'importe quel code écrit pour un Arduino pourra fonctionner sur une Teensy (la réciproque est fausse).

Dans notre projet on a utilisé une carte Teensy 3.2

## 3.1 Les principales caractèristiques de la carte Teensy 3.2

Dans le tableau suivant on représente les principales caractéristiques des cartes Teensy 3.2:

| CARTE | Teensy 3.2 |
|---|---|
| Taille (mm) | 36x23 |
| Fréquence d'horloge (MHz) | 72 |
| Overclock (MHz) | 96 |
| Mémoire Flash (kilo-octets) | 256 |
| RAM (kilo-octets) | 64 |
| EEPROM (octets) | 2048 |
| Canaux DMA | 16 |
| E/S numériques | 34 |
| E/S numériques sur connecteurs | 24 |
| Entrées analogiques | 21 |
| Sorties analogiques | 1 |
| Broches Touch Sensing | 12 |
| Timers | 12 |
| Broches PWM | 12 |
| Ports USB | 1 |
| Ports série (RX, TX) | 3 |
| SPI (MOSI, MISO, SCK, CS) | 1 |
| I2C (SDA, SCL) | 2 |
| CAN (CANRX, CANTX) | 1 |
| Sorties audio | 2 |
| SD Card | 0 |
| Ethernet | 0 |

**Figure 2.** Les principales caractéristiques des cartes Teensy, extraite du site : https://www.locoduino.org/spip.php?article24

## 3.2 Les principaux composantes de la carte Teensy 3.2

La carte de développement Teensy 3.2 présente un microprocesseur ARM Cortex de 32 bits, capable de traiter des applications de calcul intensives, et un chargeur d'amorçage en RAM Flash[1], permettant une programmation directe via un port USB : aucun programmateur externe n'est nécessaire.

Vous pouvez programmer la Teensy dans votre éditeur en C ou avec le complément logiciel Teensyduino pour IDE Arduino™ et écrire des esquisses Arduino™ pour la Teensy. Le processeur de la Teensy peut aussi accéder à l'interface USB et émuler n'importe quel dispositif USB. Il convient donc très bien aux projets USB-MIDI et autres HID[2]. Le processeur 32 bits fournit de nombreuses autres fonctionnalités, p. ex. plusieurs canaux DMA, plusieurs convertisseurs A/N haute résolution et même une interface audio I2S[3] numérique. Il est en outre doté de quatre temporisateurs distincts et d'un retardateur. Tous les connecteurs ont une capacité d'interruption.

La Teensy offre une tension système de 3,3 V jusqu'à 250 mA pour d'autres appareils. La Teensy 3.2 accepte 5 V aux entrées numériques, mais seulement 3,3 V aux entrées purement analogiques.
La figure suivante présente les principales composantes de la carte Teensy 3.2:

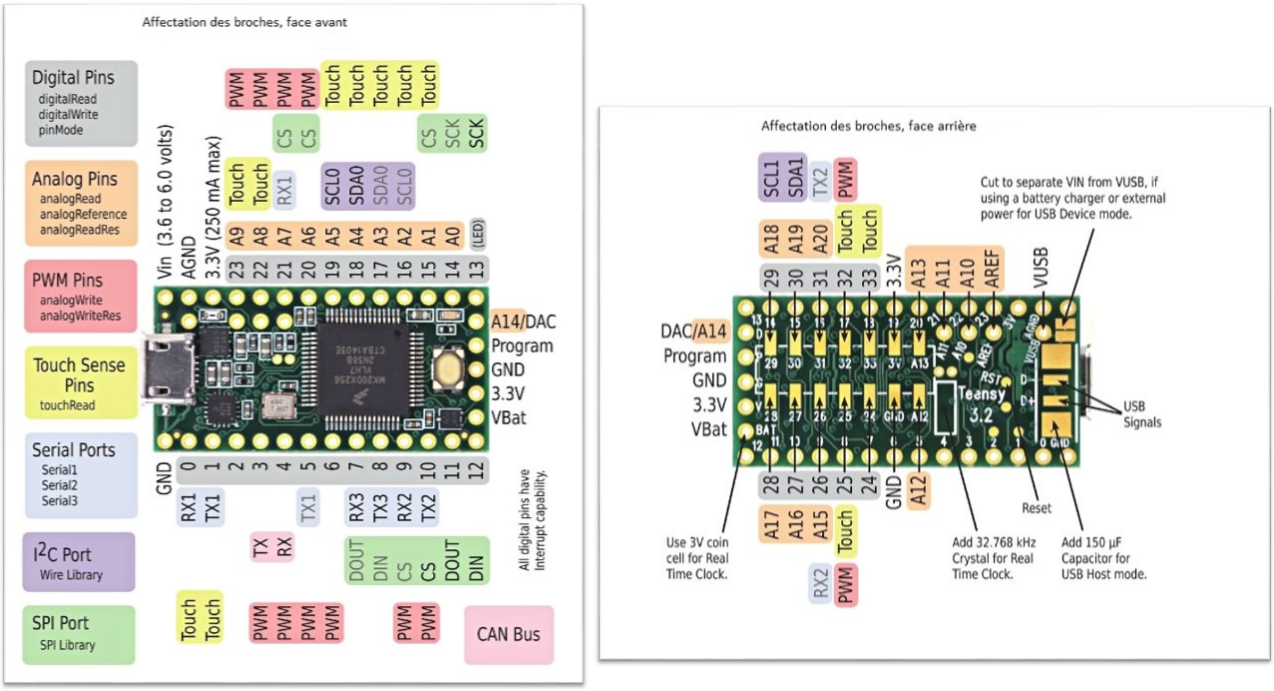


**Figure 3.** Les principaux composantes de la carte Teensy 3.2

# 4 PRÉSENTATION DE L'ALTIMU10:

1. La mémoire flash est un type de mémoire non volatile qui convient parfaitement aux appareils à piles, mobiles et portables ou en remplacement d'un disque dur dans les ordinateurs portables. La mémoire flash peut être effacée très rapidement par des moyens électriques.

2. HID : signifie un périphérique d'interface humaine (en anglais Human Interface Device).

3. I2S (également appelé Inter-IC Sound, Integrated Interchip Sound, ou IIS) est un standard d'interface électrique de type serial bus pour connecter des matériels audios numériques ensemble.

AltIMU-10 v4 est un capteur qui combine un baromètre numérique LPS25H, un gyroscope 3 axes L3GD20H et un accéléromètre 3 axes LSM303D et un magnétomètre 3 axes pour former une unité de mesure inertielle (IMU en anglais) et altimètre. Ces capteurs sont d'excellents circuits intégrés, Ils fonctionnent également à des tensions inférieures à 3,6 V, ce qui peut rendre l'interfaçage difficile pour les microcontrôleurs fonctionnant à 5 V. L'AltIMU-10 v4 résout ces problèmes en incorporant une électronique supplémentaire, y compris un régulateur de tension et un circuit de décalage de niveau, tout en conservant la taille globale, aussi compact que possible.

**Figure 4.** L'AltIMU-10 v4

## 4.1 Le fonctionnement d'un baromètre numérique LPS25H

Le baromètre numérique mesure la pression atmosphérique à l'aide de deux capteurs capacitifs : un capteur de pression et un capteur de température. En effet, la variation de la température entraîne une dilatation du capteur de pression qu'il faut prendre en compte lors de la mesure. Ces deux capteurs sont reliés à un circuit oscillateur qui délivre un signal. La fréquence de ce signal varie en fonction de la pression.

Ce signal est ensuite traité par un micro-processeur qui calcule la valeur de la pression en tenant compte de la température et délivre les informations sous forme numérique. Des courbes d'étalonnage sont enregistrées dans la mémoire du micro-processeur pour tenir compte des caractéristiques de chaque capteur.

La carte Teensy contient un baromètre numérique LPS25H qui a le même principe de fonctionnemt qu'on a indiqué au-dessus.

Pour voir plus d'informations sur le baromètre numérique LPS25H, vous pouvez voir sa datasheet :

https://www.pololu.com/file/0J761/LPS25H.pdf

## 4.2 Le fonctionnement d'un gyroscope 3 axes L3GD20H

Un gyroscope est un appareil qui exploite le principe de la conservation du moment cinétique en physique. Cette loi fondamentale de la mécanique veut qu'en l'absence de couple appliqué à un solide en rotation autour d'un de ces axes principaux, celui-ci conserve son axe de rotation invariable. Lorsqu'un couple est appliqué à l'appareil, il provoque une précession ou une nutation du solide en rotation.

Les gyroscopes sont utilisés comme capteur de position angulaire, alors que les gyromètres sont des capteurs de vitesse angulaire. Le gyroscope donne la position angulaire (selon un ou deux axes uniquement) de son référentiel par rapport à un référentiel inertiel (ou galiléen).

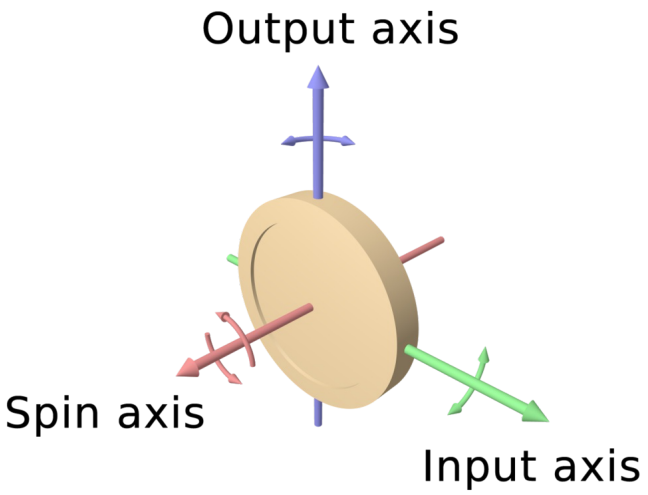


**Figure 5.** Les 3 axes d'un gyroscope

#### 4.2.1 Les équations fondamentales décrivant le comportement du gyroscope est :

$$\vec{\tau} = \frac{d\vec{L}}{dt}$$

où les vecteurs $\vec{\tau}$ et $\vec{L}$ sont respectivement le moment sur le gyroscope et son moment cinétique.

Dans le cas de l'approximation gyroscopique où la vitesse de rotation $\omega$ est élevée, on approxime $\vec{L}$ par $I.\vec{\omega}$, le scalaire $I$ étant son moment d'inertie, et $\vec{\omega}$ son vecteur vitesse angulaire. L'équation devient :

$$\vec{\tau} = I\frac{d\vec{\omega}}{dt}$$

Il découle de cela qu'un couple $\vec{\tau}$ appliqué perpendiculairement à l'axe de rotation, et donc perpendiculaire à $\vec{L}$, provoque un déplacement perpendiculaire à $\vec{L}$. Ce mouvement est appelé **précession**. La vitesse angulaire de la précession $\overrightarrow{\Omega_P}$ est donnée par :

$$\vec{\tau} = \overrightarrow{\Omega_P} \wedge \vec{L}$$

Voici datasheet de gyroscope 3 axes L3GD20H:

https://www.pololu.com/file/0J731/L3GD20H.pdf

### 4.3 Le fonctionnement d'un accéléromètre 3 axes LSM303D

Un accéléromètre est un capteur qui permet de mesurer l'accélération non gravitationnelle linéaire d'un objet (mobile ou fixe). On parle d'accéléromètre même lorsqu'il s'agit en fait de 3 accéléromètres qui calculent les accélérations linéaires selon 3 axes orthogonaux.

Le principe de la plupart des accéléromètres est basé sur la loi fondamentale de la dynamique, en effet il consiste en l'égalité entre la force d'inertie de la masse sismique du capteur et une force de rappel appliquée à cette masse.

Voici la datasheet de l'accéléromètre 3 axes LSM303D:

https://www.pololu.com/file/0J703/LSM303D.pdf

### 4.4 Le fonctionnement d'un altimètre

L'altimètre barométrique mesure une différence de pression atmosphérique entre le niveau de référence, fixé par l'utilisateur, et le niveau de l'altimètre. Le niveau de référence choisi, correspondant à l'affichage « 0 », dépend de la phase du vol et est généralement celui de l'aérodrome de départ ou de destination ou celui du niveau de la mer.

L'altimètre barométrique est étalonné suivant la variation de la pression atmosphérique selon l'altitude dans une atmosphère standard.

En prenant le niveau de la mer comme altitude de référence h0, et en prenant pour l'atmosphère un état moyen défini par l'atmosphère normalisée type OACI[4], on obtient la formule internationale du nivellement barométrique :

$$p(h) = 1013.25\left(1 - \frac{0.0065.h}{288.15}\right)^{5.255} \text{ hPa}$$

### 4.5 les connections d'AltIMU_10 v4

Pour faire fonctionner **l'AltIMU_10 v4**, il faut au minimum 4 connections :

- VIN : entre 2,5 et 5,5 V
- GND : doit être relié à 0 V
- SCL et SDA : qui doivent être connecté à un I²C bus au même niveau logique que VIN

La liaison entre SDA et SDC est faite par le protocole $I^2C$ [5] (7 bits d'adressage + 1 bit R/W (Lecture-Ecriture)).

## 5 Le filtre Kalman:

Comme on a déjà dit, un des outils les plus importants qu'on a utilisé dans notre projet est le filtre de Kalman pour filtrer les données bruitées.

Le filtre de Kalman est un filtre de réponse impulsionnelle infinie qui estime des états d'un système dynamique à partir d'une série de mesures incomplètes ou bruitées. Le filtre a été inventé par le mathématicien et l'automaticien américain Rudolf Kalman[6].

### 5.1 Le fonctionnement du filtre Kalman en contexte discret

On va représenter ici, le principe de fonctionnement du filtre Kalman dans le cas d'un système discret, le cas général sera donné dans l'ANNEXE.

Le filtre de Kalman en contexte discret est un estimateur récursif[7].

L'état du filtre est représenté par 2 variables :

✓ $\hat{x}_{k|k}$ : l'estimation de l'état à l'instant $k$;

✓ $\boldsymbol{P_{k|k}}$ : La matrice de covariance de l'erreur[8].

4. OACI : Organisation de l'aviation civile internationale

5. $I^2C$ : (signifie : Inter-Integrated Circuit, en anglais) est un bus informatique qui a émergé de la « guerre des standards » lancée par les acteurs du monde électronique. Il permet de relier facilement un microprocesseur et différents circuits, notamment ceux d'un téléviseur moderne : récepteur de la télécommande, réglages des amplificateurs basses fréquences, tuner, horloge, gestion de la prise péritel, etc.

6. Rudolf Emil Kalman (en hongrois Kálmán Rudolf Emil) (19 mai 1930 à Budapest - 2 juillet 2016) est un mathématicien et un automaticien américain d'origine hongroise, ingénieur en électrotechnique de formation, connu pour l'invention du filtre de Kalman.

7. la récursivité signifie que pour estimer l'état courant, seules l'estimation de l'état précédent et les mesures actuelles sont nécessaires. L'historique des observations et des estimations n'est ainsi pas requis.

8. La matrice de cvariance de l'erreur mesure la précision de l'état estimé.

Le filtre de Kalman a deux phases distinctes : **Prédiction** et **Mise à jour**. Dans l'étape de **mise à jour**, les observations de l'instant courant sont utilisées pour corriger l'état prédit dans le but d'obtenir une estimation plus précise.

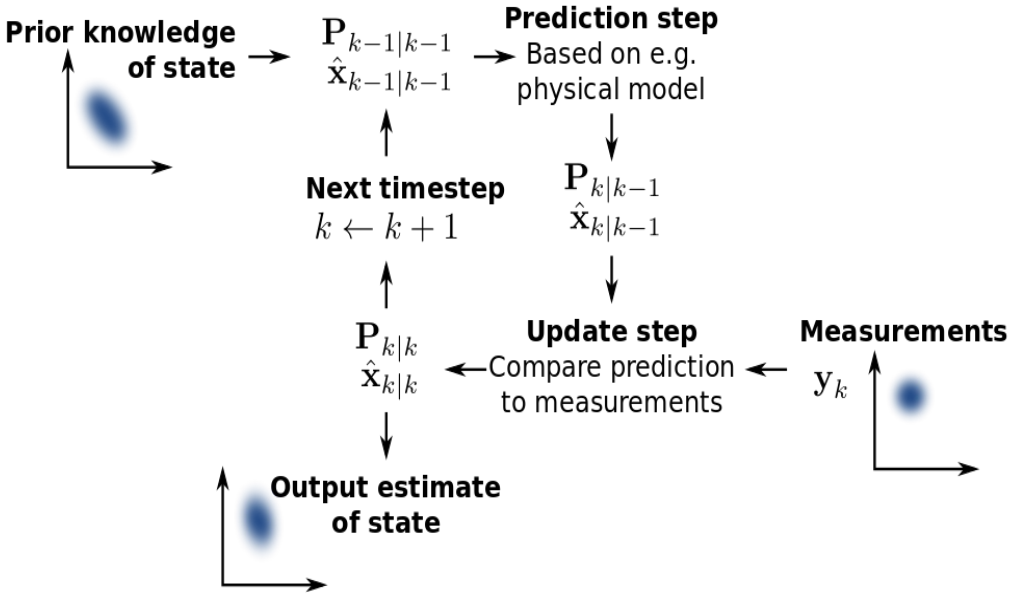


**Figure 6.** Le principe de filtre Kalman

### 5.1.1 La phase de prédiction :

La phase de **prédiction** utilise l'état estimé de l'instant précédent pour produire une estimation de l'état courant.

Les équations d'état sont données par:

$$\hat{x}_{k|k} = F_k \hat{x}_{k-1|k-1} + \boldsymbol{B_k} u_k \text{ (état prédit)}$$

$$\boldsymbol{P_{k|k-1}} = \boldsymbol{F_k P_{k-1|k-1} F_k^T} + \boldsymbol{Q_k} \text{ (estimation prédite de la covariance)}$$

avec :

- $\boldsymbol{F_k}$: matrice qui relie l'état précédent k-1 à l'état actuel $k$.
- $u_k$: entrée de commande.
- $\boldsymbol{B_k}$: matrice qui relie l'entrée de commande u à l'état $x$.
- $\boldsymbol{P_{k|k-1}}$: matrice d'estimation *a priori* de la covariance de l'erreur.
- $\boldsymbol{Q_k}$: matrice de covariance du bruit du processus.

### 5.1.2 La phase de mise à jour :

Dans l'étape de **mise à jour**, les observations de l'instant courant sont utilisées pour corriger l'état prédit dans le but d'obtenir une estimation plus précise.

Les équations d'état sont données par:

$$\hat{y}_k = z_k - \boldsymbol{H_k} \hat{x}_{k|k-1} \text{ (innovation)}$$

$$S_k = \boldsymbol{H_k P_{k|k-1} H_k^T} + \boldsymbol{R_k} \text{ (covariance de } l\text{'innovation)}$$

$$K_k = \boldsymbol{P_{k|k-1} H_k^T S_k^{-1}} \text{ (gain de Kalman optimal)}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + \boldsymbol{K_k} \hat{y}_k \text{ (état mis à jour)}$$

$$\boldsymbol{P_{k|k}} = (\boldsymbol{I - K_k H_k}) \boldsymbol{P_{k|k-1}} \text{ (covariance mise à jour)}$$

avec :

$z_k$: observation ou mesure du processus à l'instant $k$.

$\boldsymbol{H_k}$: matrice qui relie l'état $x_k$ à la mesure $z_k$.

$\boldsymbol{P_{k|k}}$: matrice d'estimation *a posteriori* de la covariance de l'erreur.

$\boldsymbol{R_k}$: matrice de covariance du bruit de mesure.

$\boldsymbol{I}$: matrice identité aux dimensions adéquates.

La formule de la mise à jour de la covariance est valide uniquement pour un gain de Kalman optimal. L'utilisation d'autres valeurs de gains nécessite des formules plus complexes (voir l'ANNEXE).

# 6 Le code Arduino

Le code qu'on a utilisé sous Arduino est le suivant :

```
#include <Wire.h>
#include <L3G.h> //gyroscope
#include <LPS.h> // baromètre numérique
#include <LSM303.h> //accéléromètre sur 3 axe
#include <SimpleKalmanFilter.h> //filtre de Kalman

L3G gyro; //variable de type L3G (gyroscope)
LSM303 compass; //variable de type LSM303 (accélérateur de 3 axe)
LPS ps;  //varibale de type LPS (pression)
SimpleKalmanFilter pressureKalmanFilter(1, 1, 0.01);
/* 1:Incertitude de mesuree
   1: Incertitude d'estimation
   0.01: bruit de processus */

char report[80]; //chaine de caractère de 80 caractères.

void setup() {
      // initialise la communication série à 9600 bits par seconde
     Serial.begin(9600);
     Wire.begin();// rejoindre le bus I2C
     //gyroscope
     //test d'initialisation du gyroscope.
     if (!gyro.init())
     {
          Serial.println("Erreur sur le gyroscope!");
          while(1);
  }
     gyro.enableDefault();
     //boussole
     compass.m_min = (LSM303::vector<int16_t>){-3029, -2238, -3193};
     compass.m_max = (LSM303::vector<int16_t>){+2812, +3110, +2183};
     //voir quelle IMU est connectée
     compass.init();
     compass.enableDefault();

     //Mesure de la pression
     if (!ps.init())
     {
          Serial.println("Erreur sur le detecteur de pression");
```

```
            while (1);
        }
        ps.enableDefault();

    }

    void loop() {
      //On affiche les données du gyroscope
      gyro.read();
      float x_est1 = gyroKalmanFilter.updateEstimate((int)gyro.g.x);
      float x_f = skf.updateEstimate(x_est1);
      float y_est1 = gyroKalmanFilter.updateEstimate((int)gyro.g.y);
      float y_f = skf.updateEstimate(y_est1);
      float z_est1 = gyroKalmanFilter.updateEstimate((int)gyro.g.z);
      float z_f = skf.updateEstimate(z_est1);




      Serial.print("Gyroscope : ");

      Serial.print(" X: ");
      Serial.print(gyroKalmanFilter.updateEstimate((int)gyro.g.x));
      Serial.print("/");
      Serial.print((int)gyro.g.x);
      Serial.print("/");
      Serial.print(x_f);
      Serial.print("\t");

      Serial.print(" Y: ");
      Serial.print(gyroKalmanFilter.updateEstimate((int)gyro.g.y));
      Serial.print("/");
      Serial.print((int)gyro.g.y);
      Serial.print("/");
      Serial.print(y_f);

      Serial.print("\t");

      Serial.print(" Z: ");
      Serial.println(gyroKalmanFilter.updateEstimate((int)gyro.g.z));
      Serial.print("/");
      Serial.println((int)gyro.g.z);
      Serial.print("/");
      Serial.print(z_f);

      Serial.print("\n");

    //On affiche les données de la boussole
      compass.read();

      float heading = compass.heading();
      Serial.print(heading);
```

```
//On affiche l'altitude
  float pression = ps.readPressureMillibars();
  float altitude = ps.pressureToAltitudeMeters(pression);
  //calcule la valeur estimée avec Kalman Filter
  float estimated_altitude = pressureKalmanFilter.updateEstimate(altitude);
  Serial.print("Altitude : ");
  Serial.print(estimated_altitude);

  Serial.print("\n——————————————————\n");

  delay(100);
}
```

# 7 Processing

Processing est un environnement de développement libre (sous licence GNU GPL), créé par Benjamin Fry et Casey Reas, deux artistes américains. Processing est le prolongement « multimédia » de Design by numbers, l'environnement de programmation graphique développé par John Maeda au Media Lab du Massachusetts Institute of Technology.

## 7.1 Le code de Processing

Pour récupérer et visualiser les données du Arduino sur Processing, on a fait le code suivant :

```
import processing.serial.*;
Serial myPort;
float valeur_recuX;
float valeur_recuY;
float valeur_recuZ;

void setup()
{
    size(1000,1000); // Taille de la fenetre
    myPort = new Serial(this, "COM57", 9600); // Teensy connecté au port COM57
    background(255,255,255); // Fond en blanc
    noStroke(); // pas de bordure
    rectMode(CENTER);

}

void loop()
{
    fill(0,0,0); // rectangle blanc
    if ( myPort.available() > 0) // si le port marche
    {
        // On récupère les différentes valeurs angulaire selon la syntaxe choisie (teensy)
        valeur_recuX = port_de_com.readStringUntil(',')
        valeur_recuY = port_de_com.readStringUntil(',')
        valeur_recuZ = port_de_com.readStringUntil('\n')
    }
```

```
// On teste sur l'angle X avec un seul rectangle

float c = cos(valeur_recuX);
translate(width/2 , height/2);
rotate(c);
rect(500,500,100,200);
```

## 7.2 l'Affichage sur Processing

On a sur la fenêtre de processing le positionnement de la carte AltIMU_10 v4 dans le plan (Oxy):

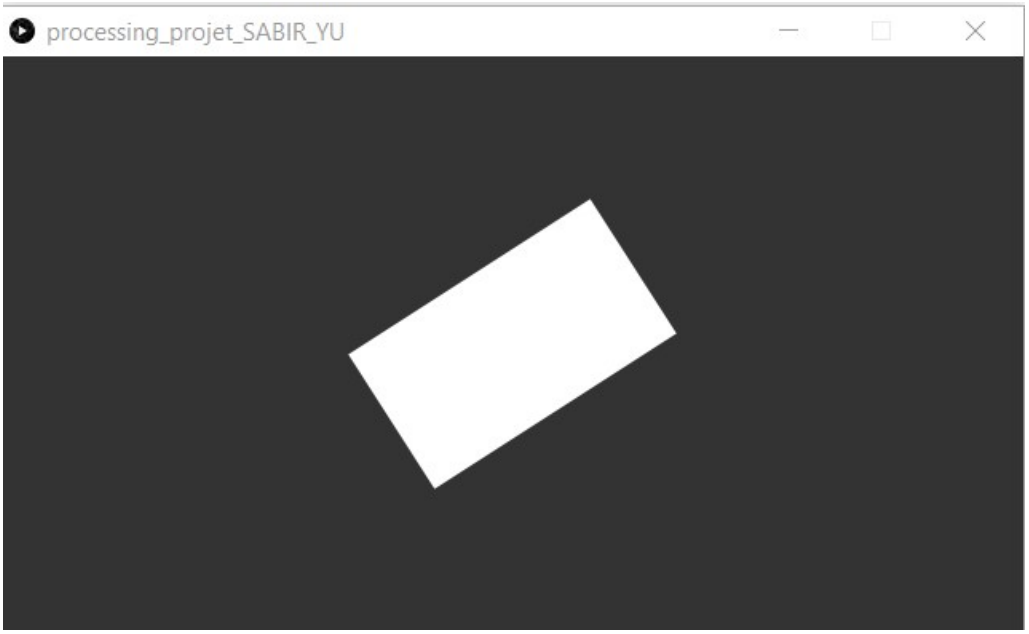


**Figure 7.** l'affichage de positionnement de l'AltIMU_10 v4 dans le plan horisontal

# 8 ANNEXES:

## 8.1 Le filtre de Kalman étendu

Le principe d'un filtre de Kalman étendu est très simple. Tout d'abord, les équations d'état et les équations liant l'état précédent à l'instant suivant qui étaient linéaires dans le cas du filtre de Kalman classique sont maintenant non linéaire. Il est donc impossible de l'écrire sous forme matricielle.

On remplace donc les équations

$$\begin{cases} Y = H.X + B \\ \widehat{X_k}^+ = A.\hat{X}_k \end{cases} \text{ par } \begin{cases} Y = h(X, B) \\ \widehat{X_k}^+ = f(\widehat{X_k}^+) \end{cases}$$

On est donc obligé d'appliquer ces équations non linéaires pour le calcul de la prédiction et la mise à jour du vecteur d'état. Les équations de Kalman deviennent donc :

**La phase de prédiction**

$$\hat{X}_k^+ = f(\hat{X}_k)$$

$$P_k^+ = A.P_k A^T + Q$$

**La phase de mise à jour**

$$K_{k+1} = P_k^+ .H_{k+1}^T .(R_{k+1} + H_{k+1}.P_k^+ .H_{k+1}^T)^{-1}$$

$$P_{k+1} = (I - K_{k+1}.H_{k+1}).P_k^+$$

$$X_{k+1} = \hat{X}_k^+ + K_{k+1}.(y_{k+1} - h(\hat{X}_k^+, 0))$$

Vous remarquerez que pour mettre à jour le vecteur d'état, on utilise l'équation d'état non linéaire avec un bruit nul (second paramètre de la fonction **h** à 0).

Il reste un problème en ce qui concerne le calcul de la covariance de l'erreur et du gain de Kalman. En effet, on utilise toujours les matrices d'observation et de transition **H** et**A**. Pour pouvoir utiliser ces formules, il faut donc que l'on linéarise localement les fonctions **h** et **f**. On obtient donc les matrices d'observation et de transition en prenant les matrices des dérivés partielles des équations non linéaires.

$$H_k = \left.\frac{\partial h}{\partial X}\right|_{\hat{X}_k^+}$$

$$A_k = \left.\frac{\partial f}{\partial X}\right|_{\hat{X}_{k-1}}$$

Avec ces Jacobiennes, il est donc possible d'appliquer le filtre de Kalman tel que défini ci-dessus. Il suffit de recalculer les matrices aux dérivées partielles à chaque nouveaux échantillons à traiter et d'utiliser ces matrices dans les équations.

Par contre, on se rend bien compte que l'on linéarise localement les équations afin d'appliquer le filtre de Kalman. Cette linéarisation est locale, ce qui entraine donc une convergence locale du filtre de Kalman étendu. Ce filtre ne garantis donc pas une convergence globale (à l'inverse du filtre de Kalman classique). La stabilité d'un EKF est donc plus difficile à garantir et dépend souvent de sa bonne initialisation.

## 8.2 La classe Kalman :

```
#ifndef _Kalman_h
#define _Kalman_h

class Kalman {
public:
    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_appli-
cation.2C_technical
        P[0][1] = 0;
        P[1][0] = 0;
        P[1][1] = 0;
```

```
};
// 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) {
    /* 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;
};
void setAngle(double newAngle) { angle = newAngle; }; // Used to set angle, this should be set as the starting angle
double getRate() { return rate; }; // Return the unbiased rate

/* These are used to tune the Kalman filter */
void setQangle(double newQ_angle) { Q_angle = newQ_angle; };
void setQbias(double newQ_bias) { Q_bias = newQ_bias; };
void setRmeasure(double newR_measure) { R_measure = newR_measure; };

double getQangle() { return Q_angle; };
double getQbias() { return Q_bias; };
double getRmeasure() { return R_measure; };

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
```

# 9 Webographie:


[1]: https://club-intech.minet.net/index.php/Teensy

[2]: https://www.locoduino.org/spip.php?article24

[3]:https://www.conrad.fr/p/carte-de-developpement-joy-it-teensy-32-usb-entwicklerboard-convient-pour-cartes-arduino-arduino-1-pcs-1656378

[4]: http://www.robotpark.com/AltIMU-10-v4-Gyro-Accelerometer-Compass-and-Altimeter-L3GD20H-LSM303D-and-LPS25H-Carrier

[5]: https://fr.wikipedia.org/wiki/Filtre_de_Kalman

[6]: http://www.ferdinandpiette.com/blog/2011/05/le-filtre-de-kalman-etendu-principe-et-exemple/

[7]:https://github.com/TKJElectronics/Example-Sketch-for-IMU-including-Kalman-filter/blob/master/IMU/MPU6050/Kalman.h

[8]: http://www1.ucam.ac.ma/cneree/Barometrenumerique.pdf

[9]: https://www.pololu.com/file/0J761/LPS25H.pdf

[10]: https://fr.wikipedia.org/wiki/Altim%C3%A8tre#:~:text=Un%20altim%C3%A8tre%20barom%C3%A9trique%20mesure%20la,base%20utilis%C3%A9%20dans%20les%20a%C3%A9ronefs.

[11]: https://fr.wikipedia.org/wiki/Processing