ETUDE DE LA STABILITÉ D'UN DRONE DE LIVRAISON DE MARCHANDISES EN ZONE URBAINE

Sommaire

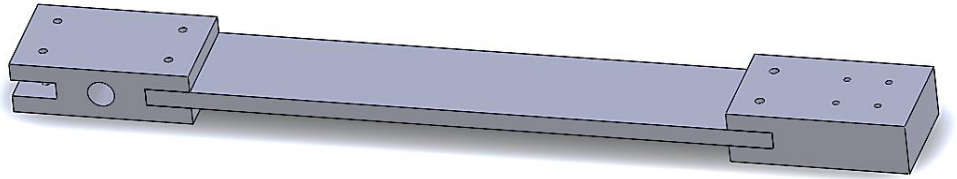
https://blog.route4me.com

Problématique et enjeux

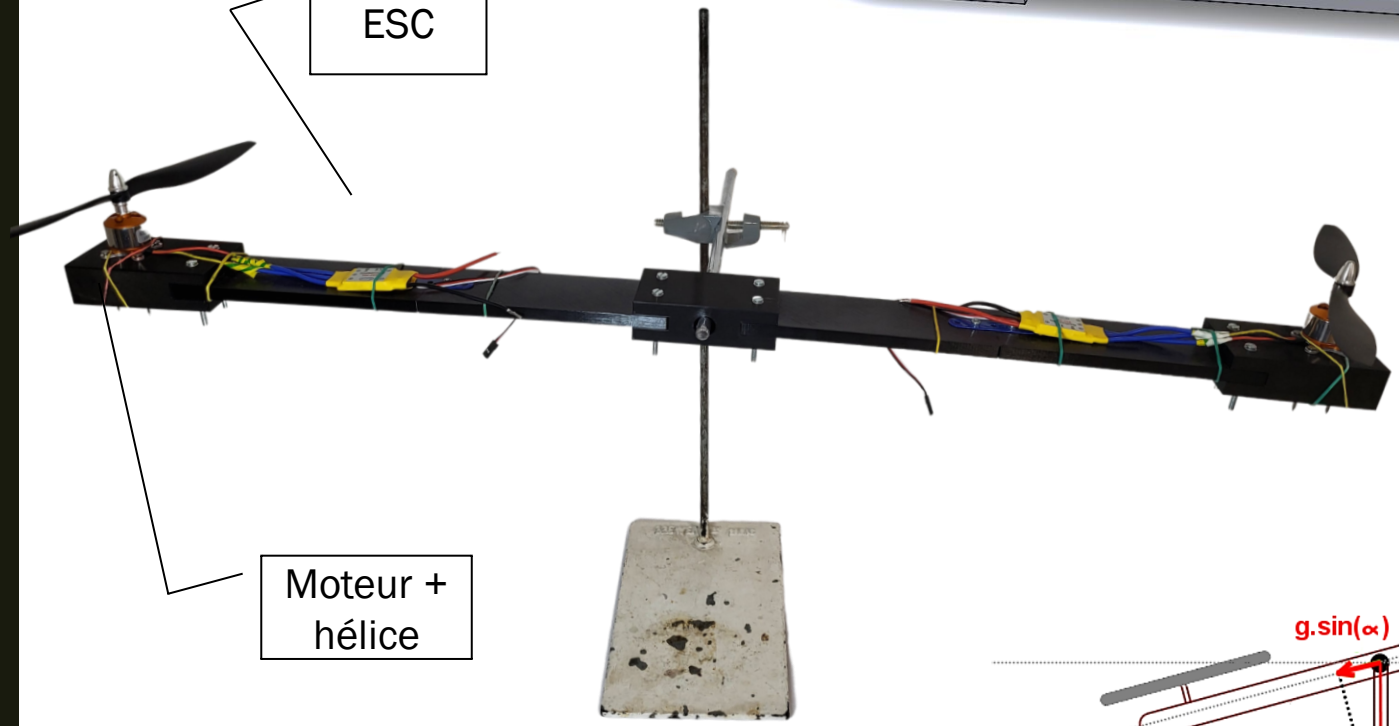
https://dronexl.co

Comment optimiser l'asservissement d'un drone de livraison afin d'assurer la stabilité de la rotation autour de son axe de roulis, dans le cadre de transport de marchandises ?

Présentation du modèle

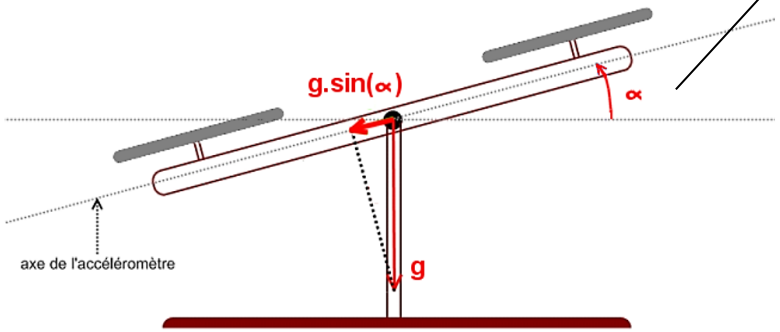


ESC



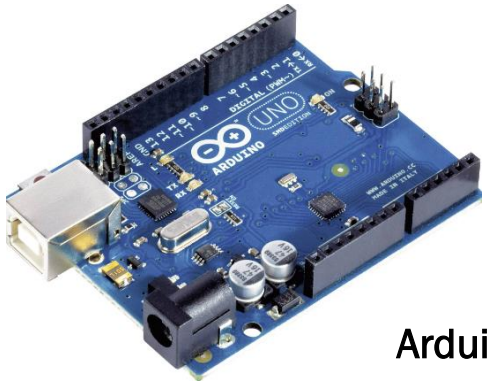
Moteur + hélice

Paramètre de l'asservissement angulaire



https://wiki.electrolab.fr

Présentation du modèle



Arduino UNO

Acquérir

MPU-6050

Traiter

Communiquer

Alimenter

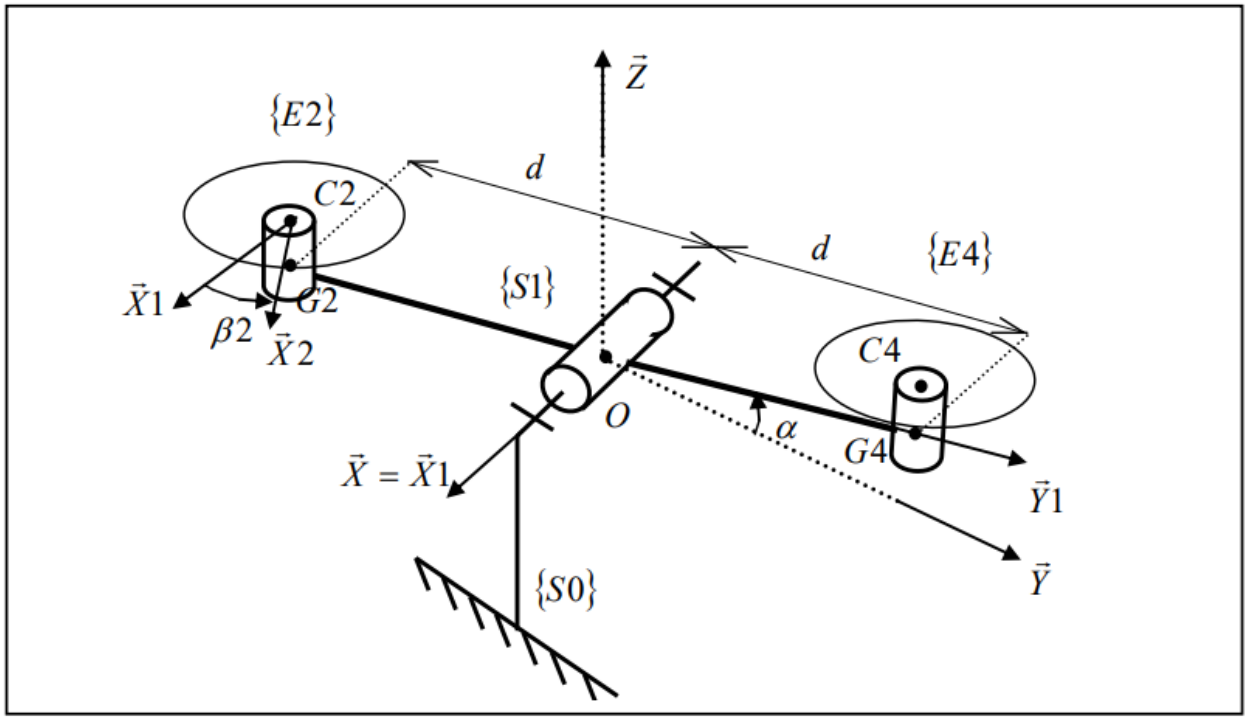
Distribuer

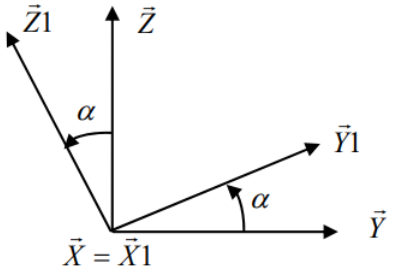
Convertir

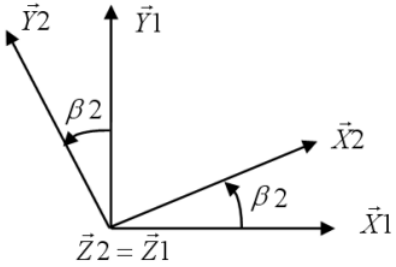
ESC (Electronic Speed Controller)

Moteur

Etude dynamique



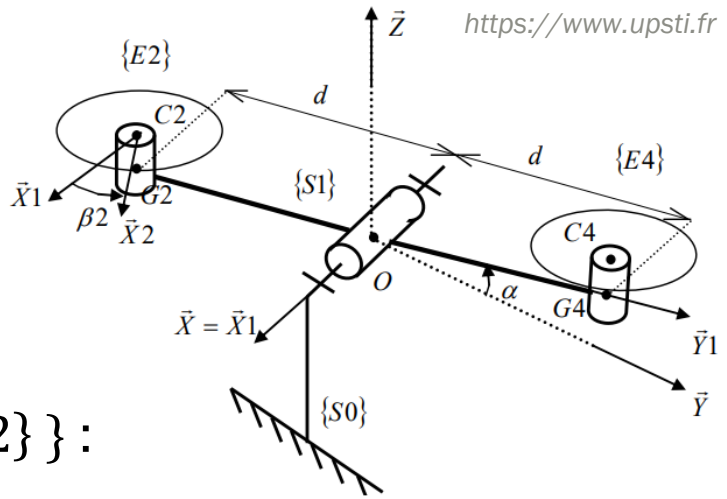




https://www.upsti.fr

Hypothèse : liaisons parfaites

Etude dynamique



<u>Théorème du moment dynamique</u> en O selon $\overrightarrow{X_1}$ appliqué à l'ensemble $\{\{S1\}+\{E4\}+\{E2\}\}$:

$$\sum \overrightarrow{M^O}.\overrightarrow{X_1} = d.F_d - d.F_g$$

$$\vec{\delta}_{O(B|R_0)} = (2(I_{xx}+md^2)+I_{S1}).\ddot{\alpha}.\overrightarrow{X_1} - I_{zz}(\dot{\beta}_2+\dot{\beta}_4)\dot{\alpha}.\overrightarrow{Y_1} + I_{zz}(\ddot{\beta}_2+\ddot{\beta}_4).\overrightarrow{Z_1}$$

→ $$\boldsymbol{\ddot{\alpha} = \frac{d\,(F_d - F_g)}{2(I_{xx}+md^2)+I_{S1}}}$$

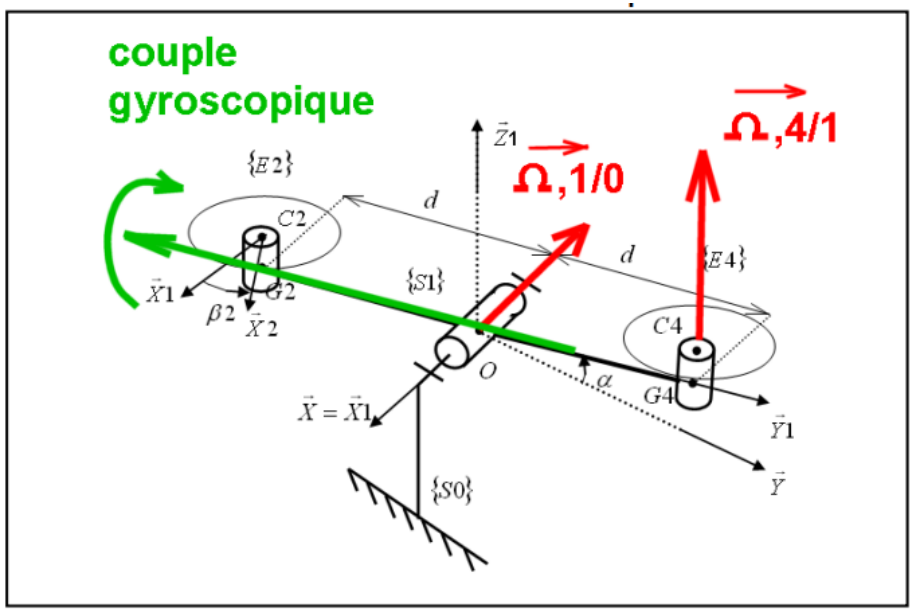
Etude dynamique

$$R = (2(I_{xx} + m.d^2) + I_{S1})\ddot{\alpha}$$

$$L = I_{zz}(\ddot{\beta}_2 + \ddot{\beta}_4)$$

$$C = -I_{zz}(\dot{\beta}_2 + \dot{\beta}_4)\dot{\alpha}$$

https://www.upsti.fr



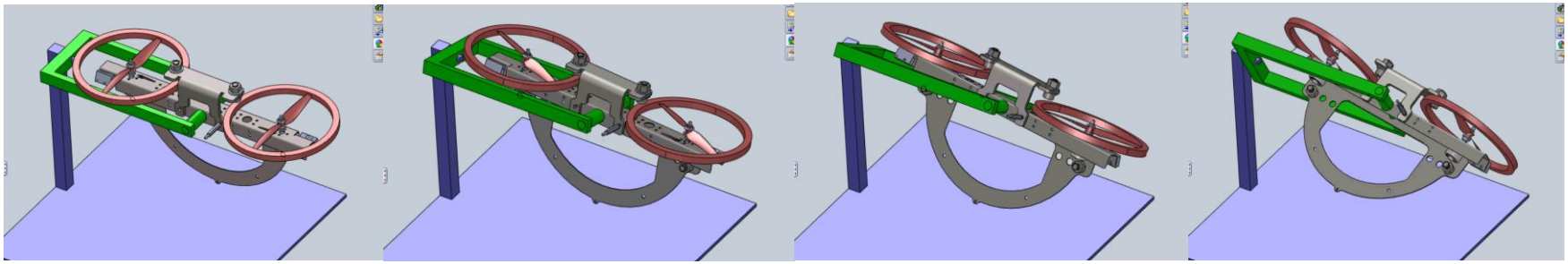
R : « moment des effets dynamiques axe de Roulis » $(\vec{X}_1)$

L : « moment des effets dynamiques axe de Lacet » $(\vec{Z}_1)$

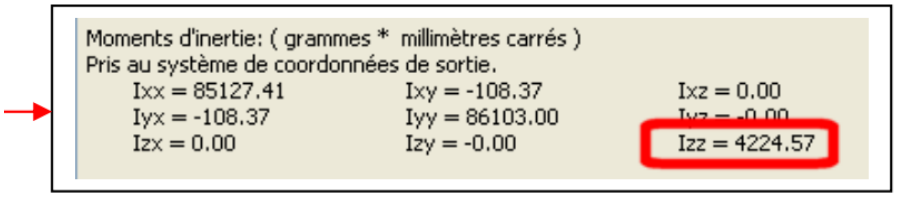
C : « moment des effets dynamiques de Coriolis » / « couple gyroscopique » $(\vec{Y}_1)$

Etude dynamique

Analyse de l'influence de l'effet gyroscopique sur la stabilité :



Vitesse des moteurs : 2500 tr/min
Vitesse de rotation du balancier : 20 tr/min
Inertie Izz :

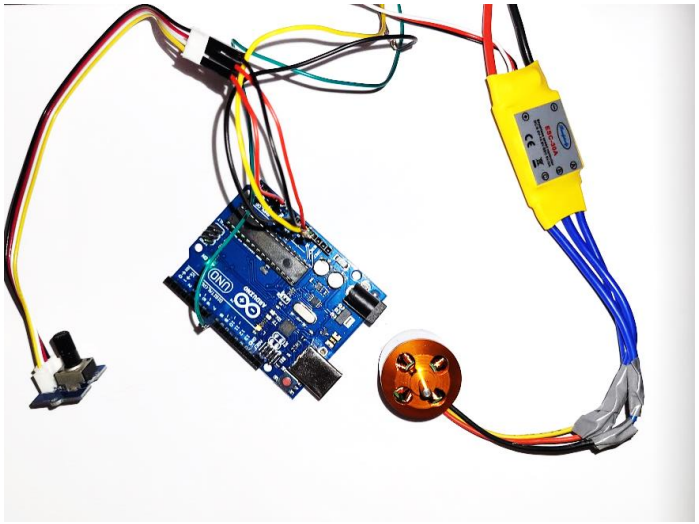


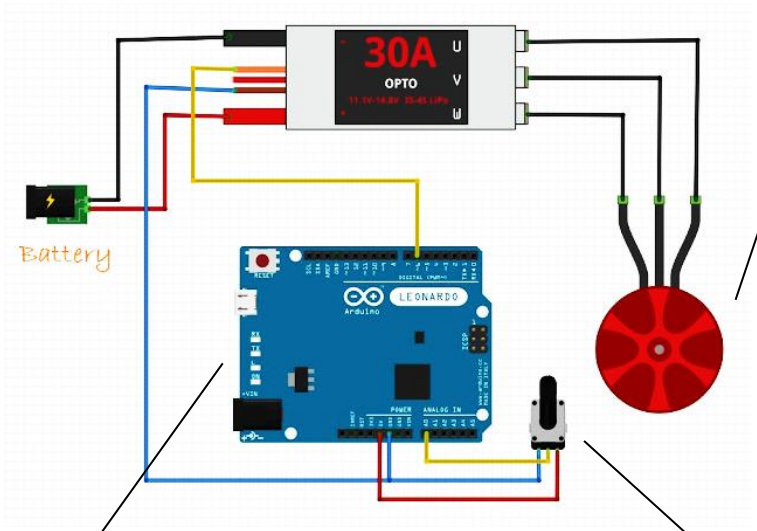
$$C = -I_{zz}\left(\dot{\beta}_2 + \dot{\beta}_4\right)\dot{\alpha} = \mathbf{0,5.10^{-2}\ N.m}$$

$$\overrightarrow{M_{moteurs}}.\overrightarrow{X_1} = d\left(F_d - F_g\right) = \mathbf{6,45.10^{-2}\ N.m}$$

Couple gyroscopique
<u>13x plus faible</u>
→ négligeable

Asservissement et simulations





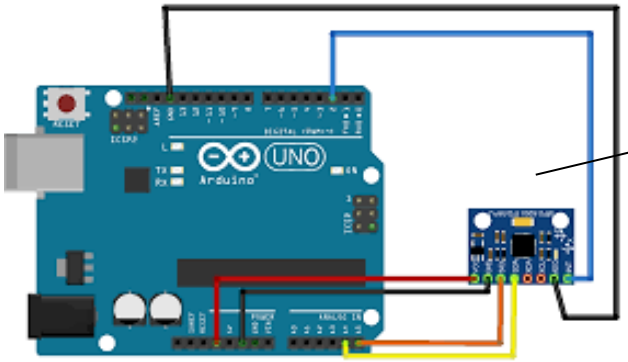
ESC

Moteur

Carte Arduino

Potentiomètre

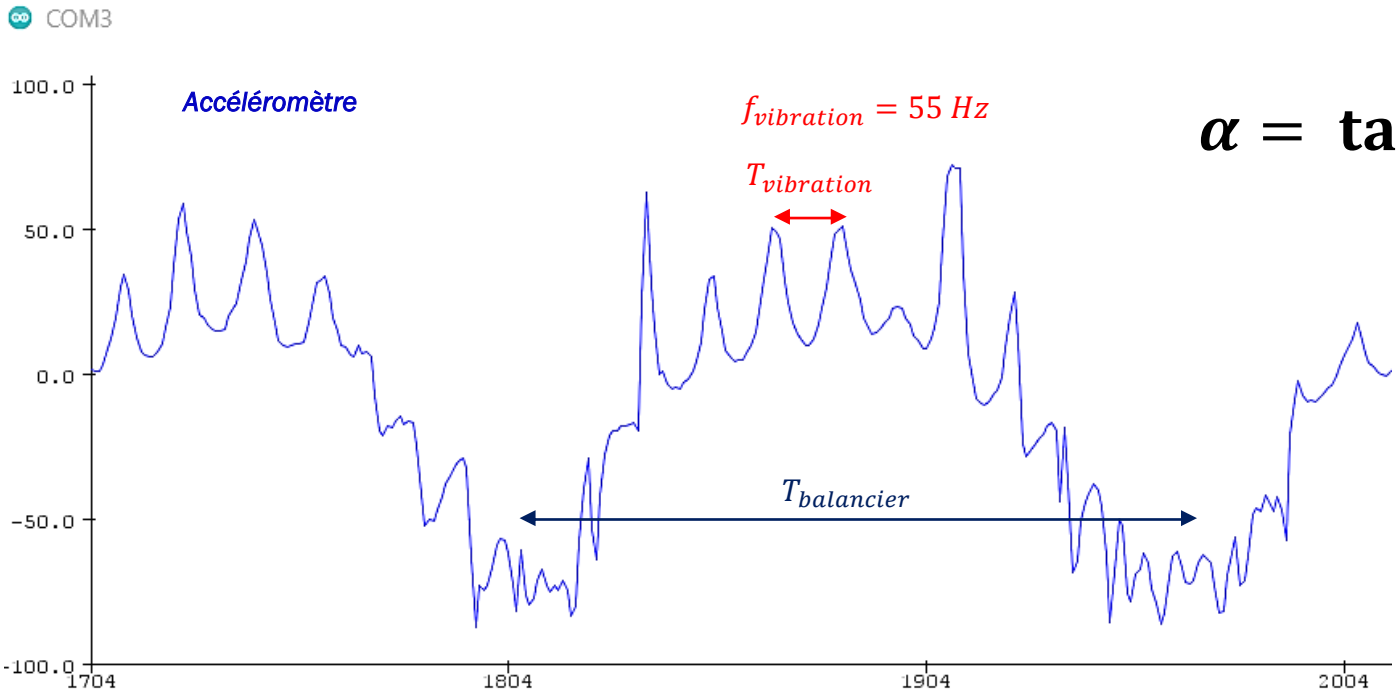
Asservissement et simulations



MPU6050
(Accéléromètre)

Formule d'Euler :

$$\alpha = \tan^{-1}\frac{g_Y}{g_X}$$



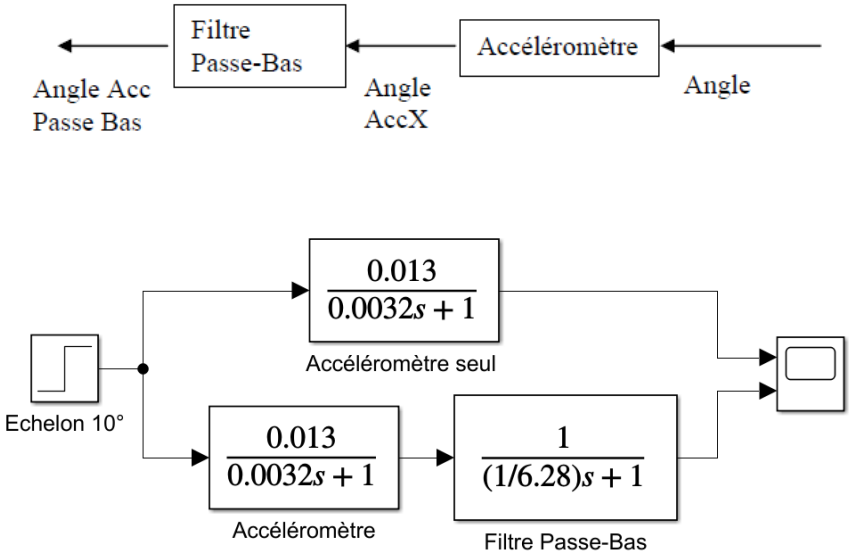
Asservissement et simulations

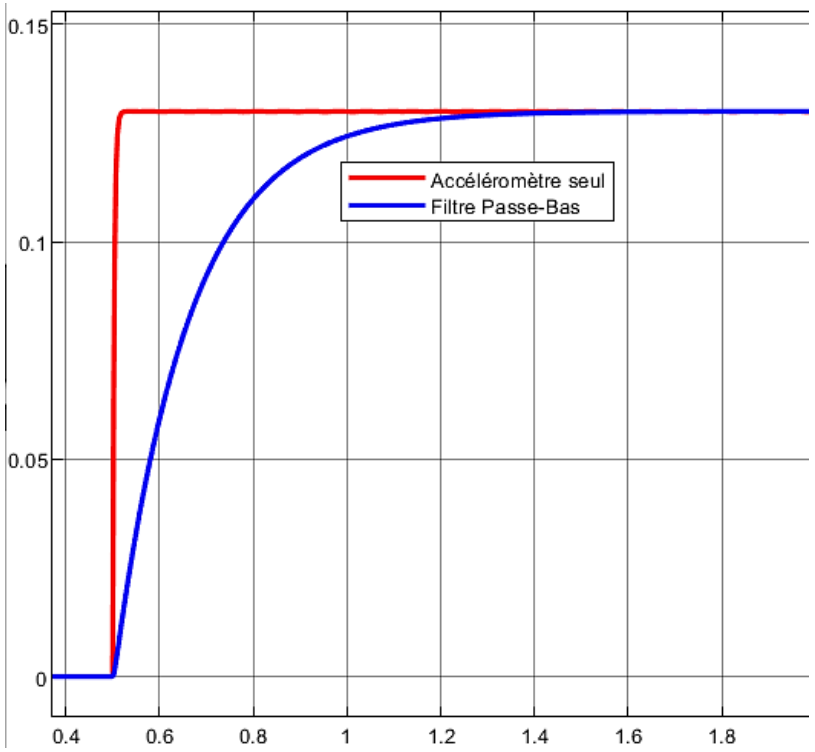
Accéléromètre : réponse juste mais lente

Problème : accélération/vibration du drone perçue comme une inclinaison → drone instable

Solution : filtre passe-bas

Inconvénient : $t_{r5\%}$ rallongé → perte de rapidité



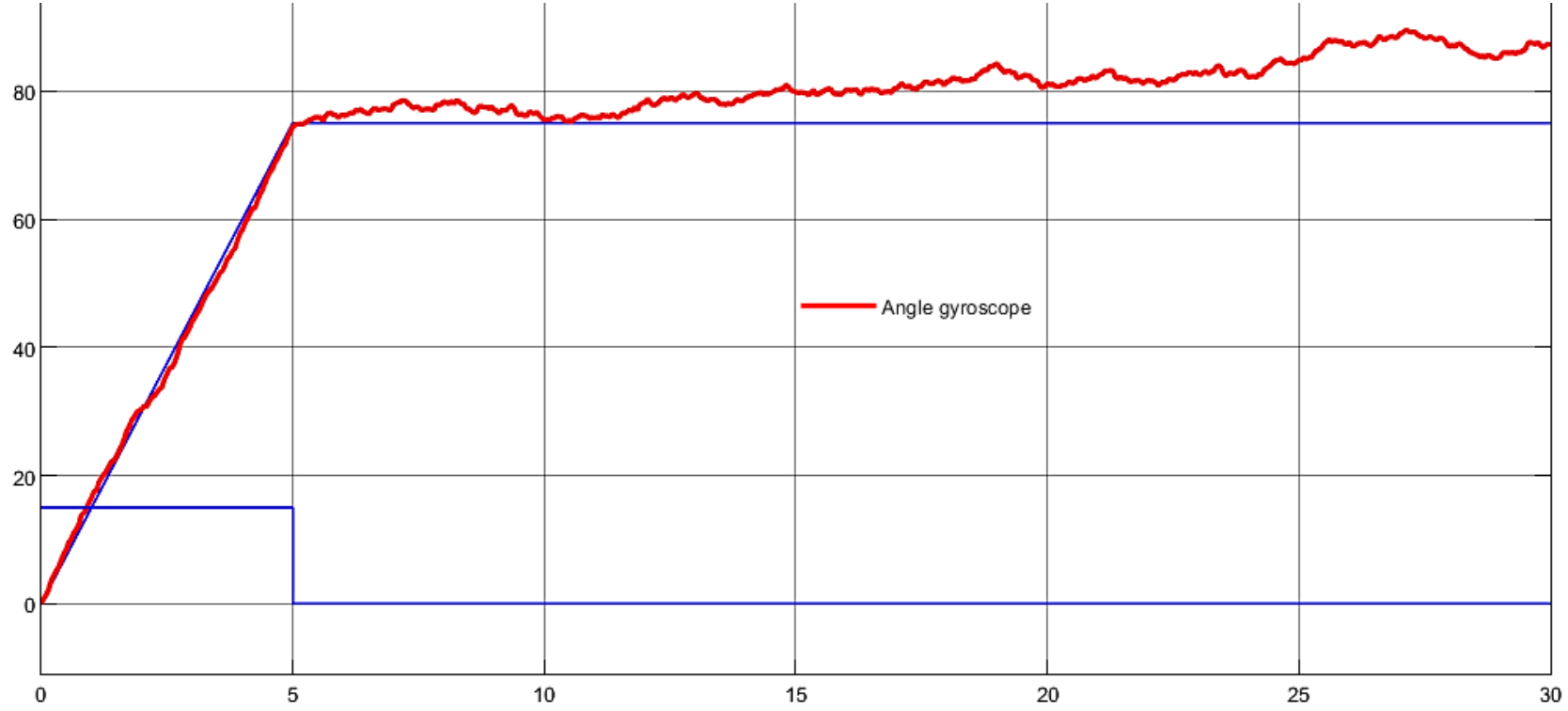


Asservissement et simulations

Gyroscope : réponse rapide

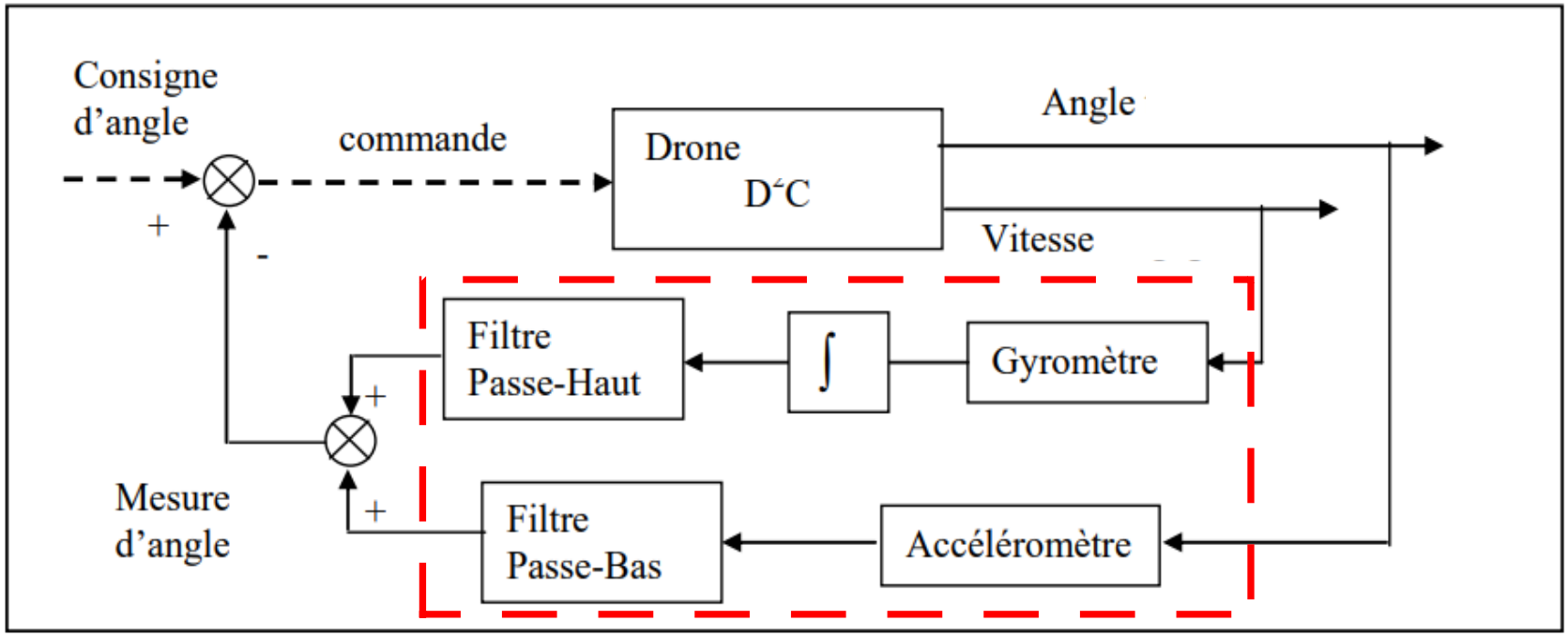
Problème : vitesse angulaire intégrée → amplification des imprécisions → **dérive du gyroscope**

Solution : filtre passe-haut + mise en parallèle de l'accéléromètre

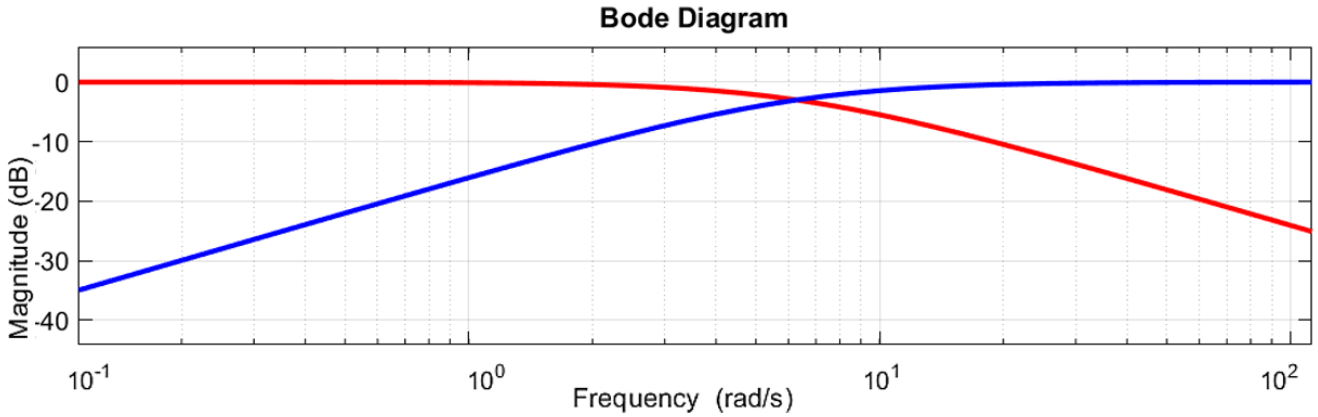


Asservissement et simulations

https://www.upsti.fr



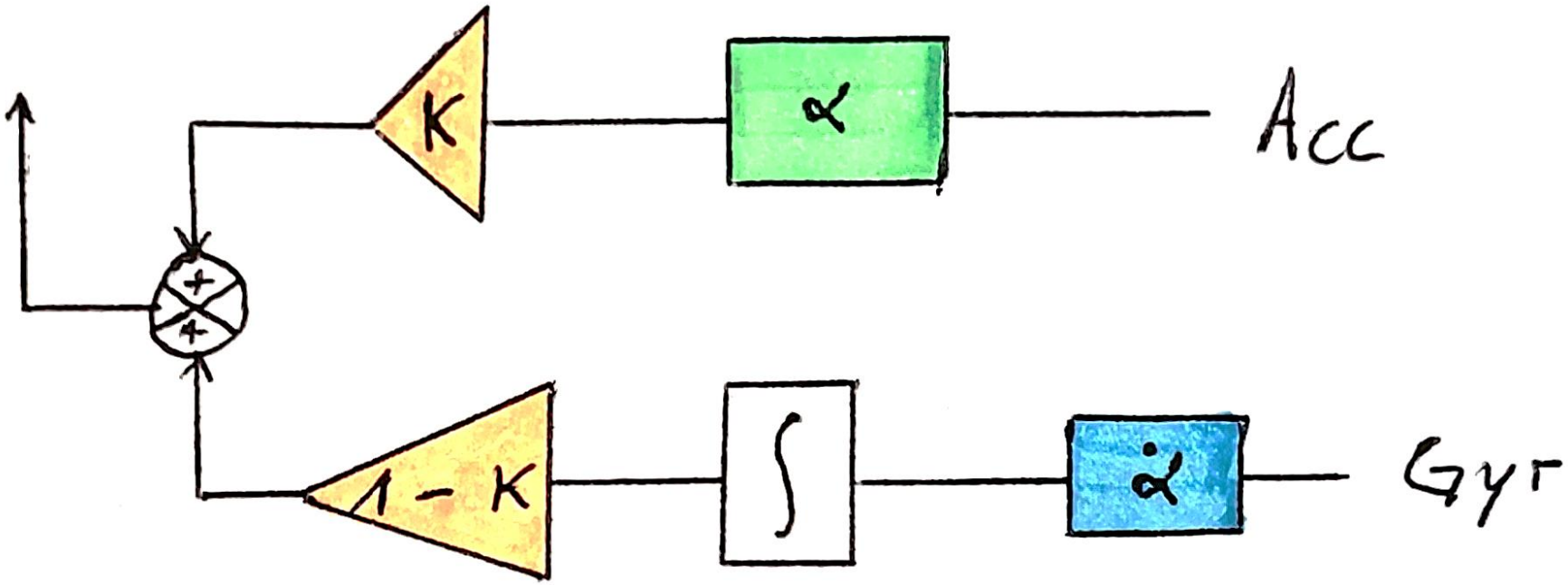
Filtre complémentaire :



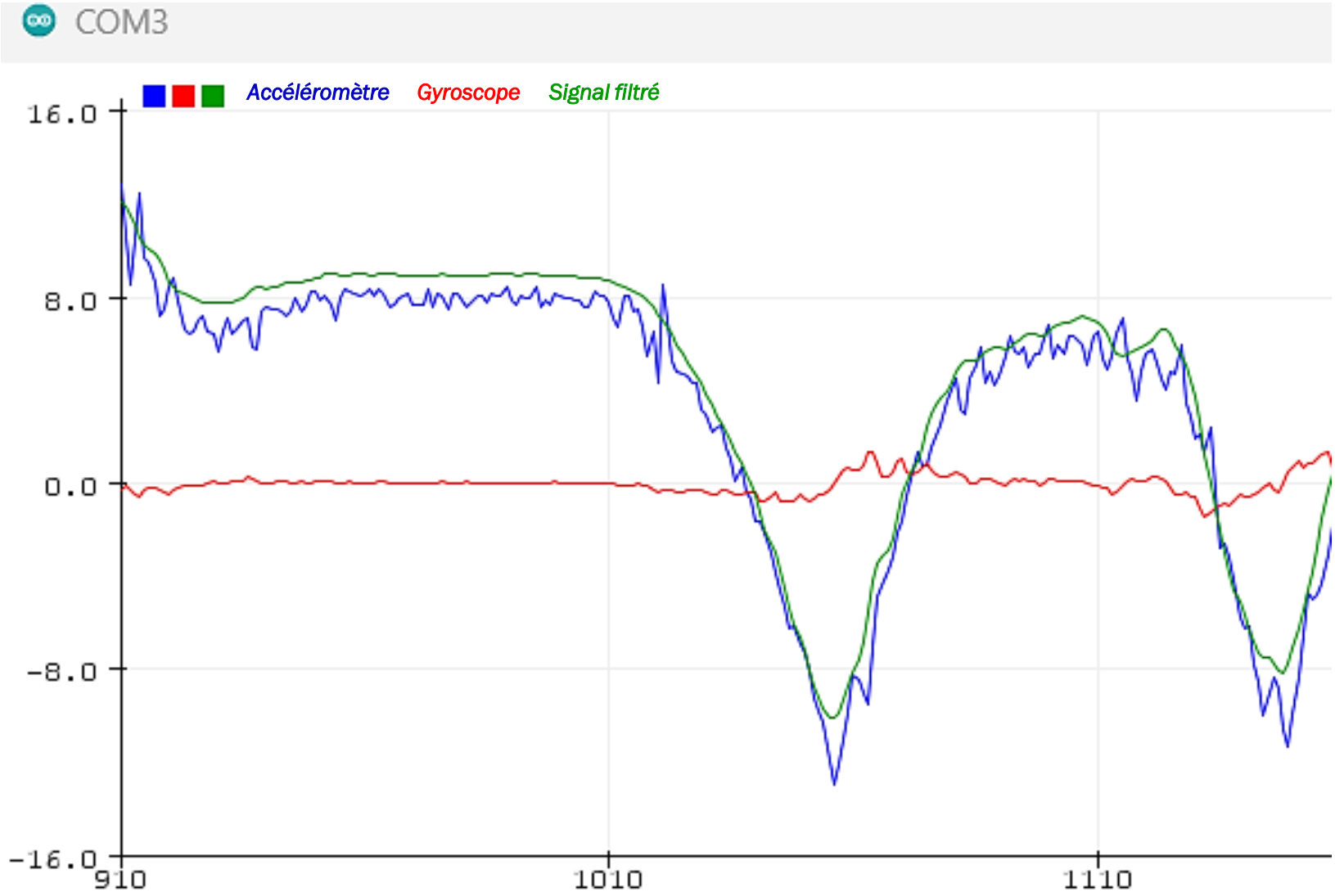
Asservissement et simulations

Implémentation Arduino du filtre complémentaire :

$$angle = 0.98.(angle + gyroData.\Delta t) + 0.02.accData$$

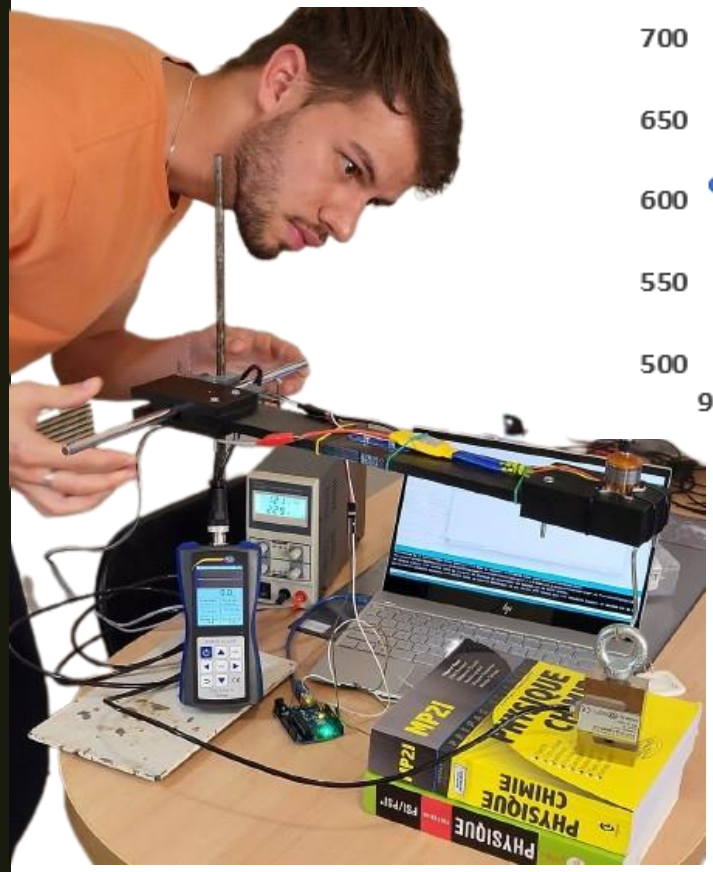


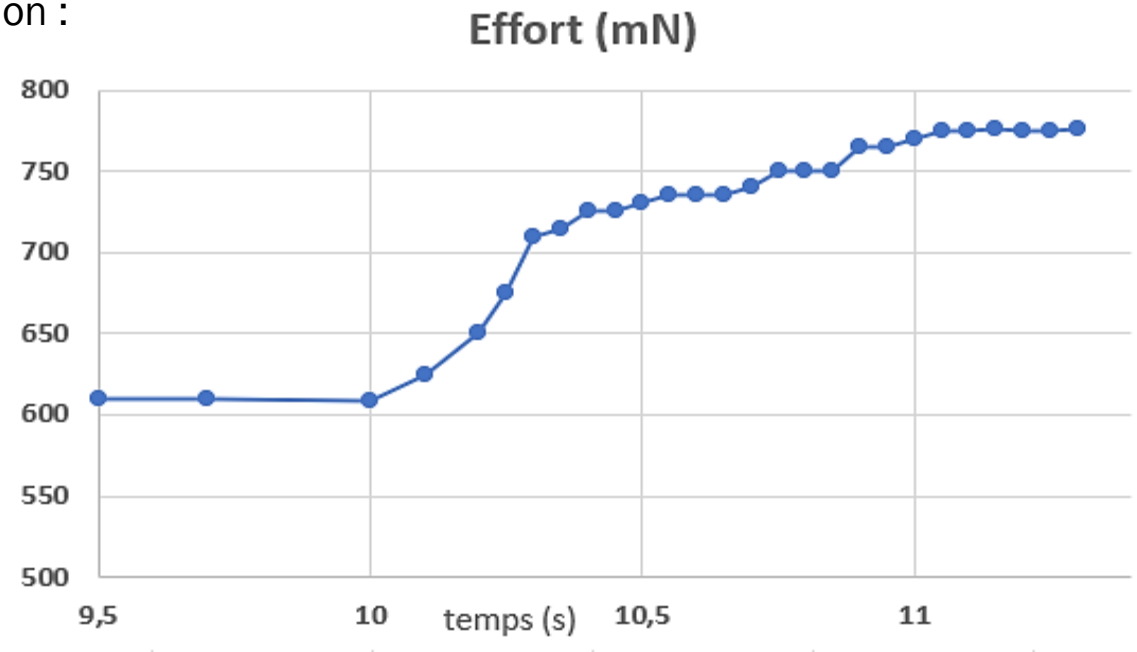
Asservissement et simulations



Asservissement et simulations

Modélisation de la motorisation :
un premier ordre



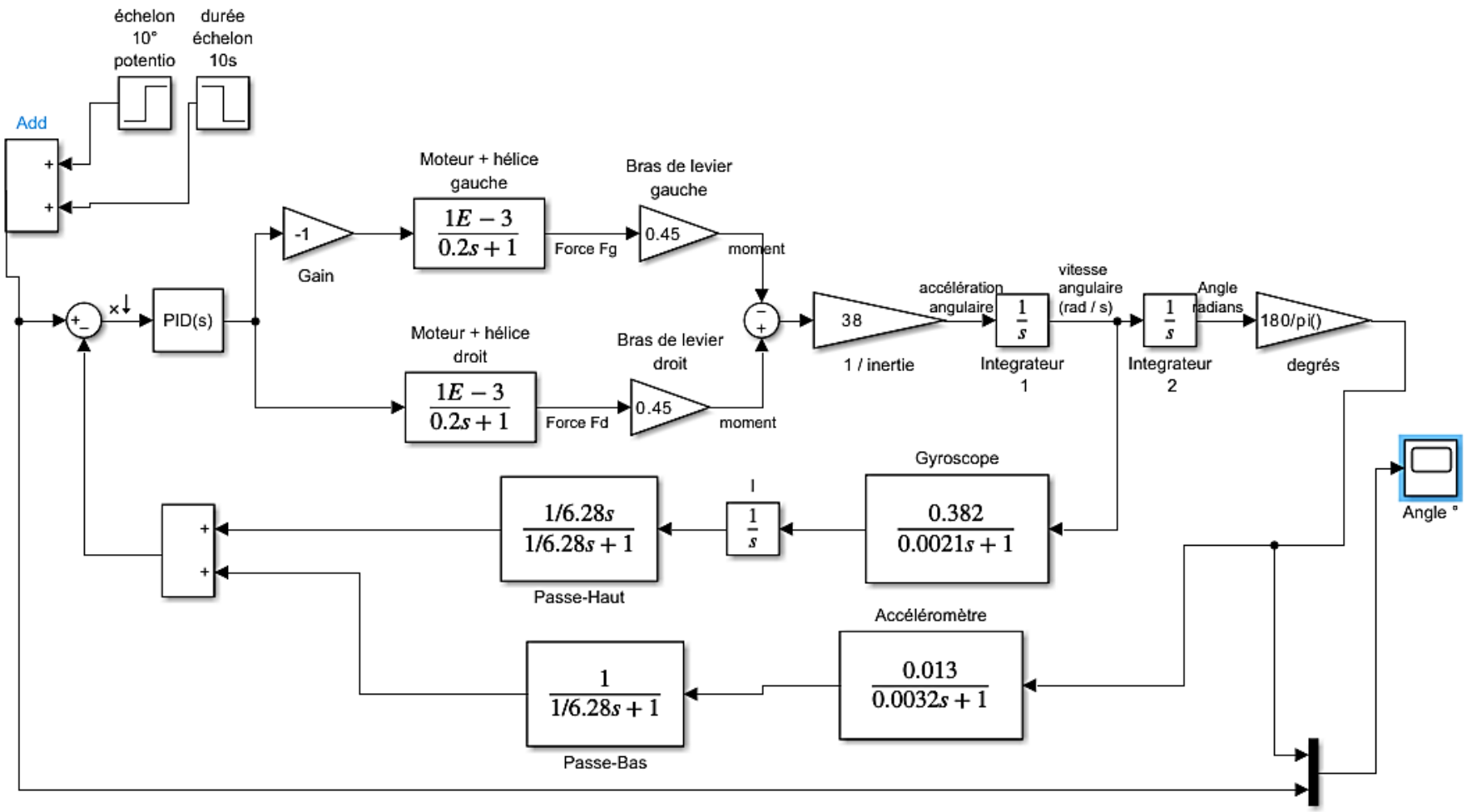


$$F_m(p) = \frac{Km}{1 + \tau_m . p}$$

$$Km = \frac{\Delta F}{\Delta F_c} = \mathbf{1,06\ mN}$$

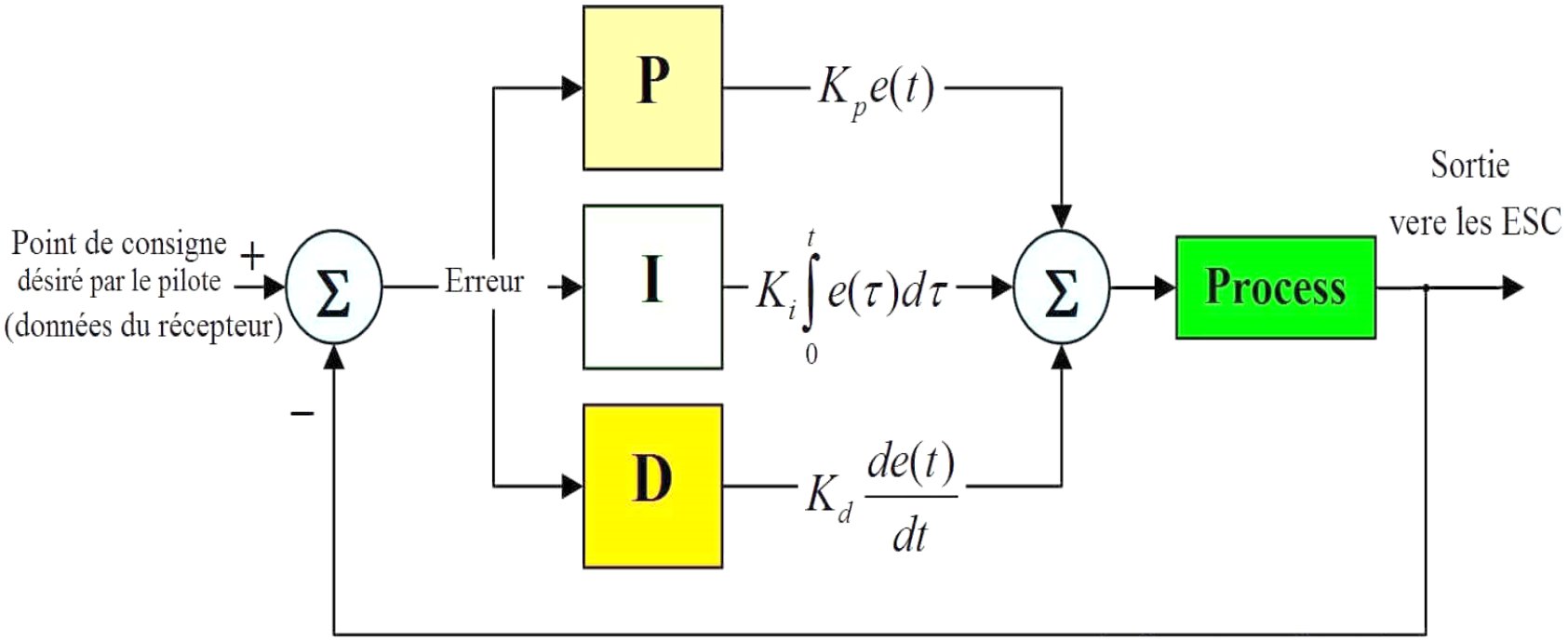
$$\tau_m = \frac{t_{r5\%}}{3} = \mathbf{0,2\ s}$$

Asservissement et simulations



Asservissement et simulations

Choix du correcteur : **PID**

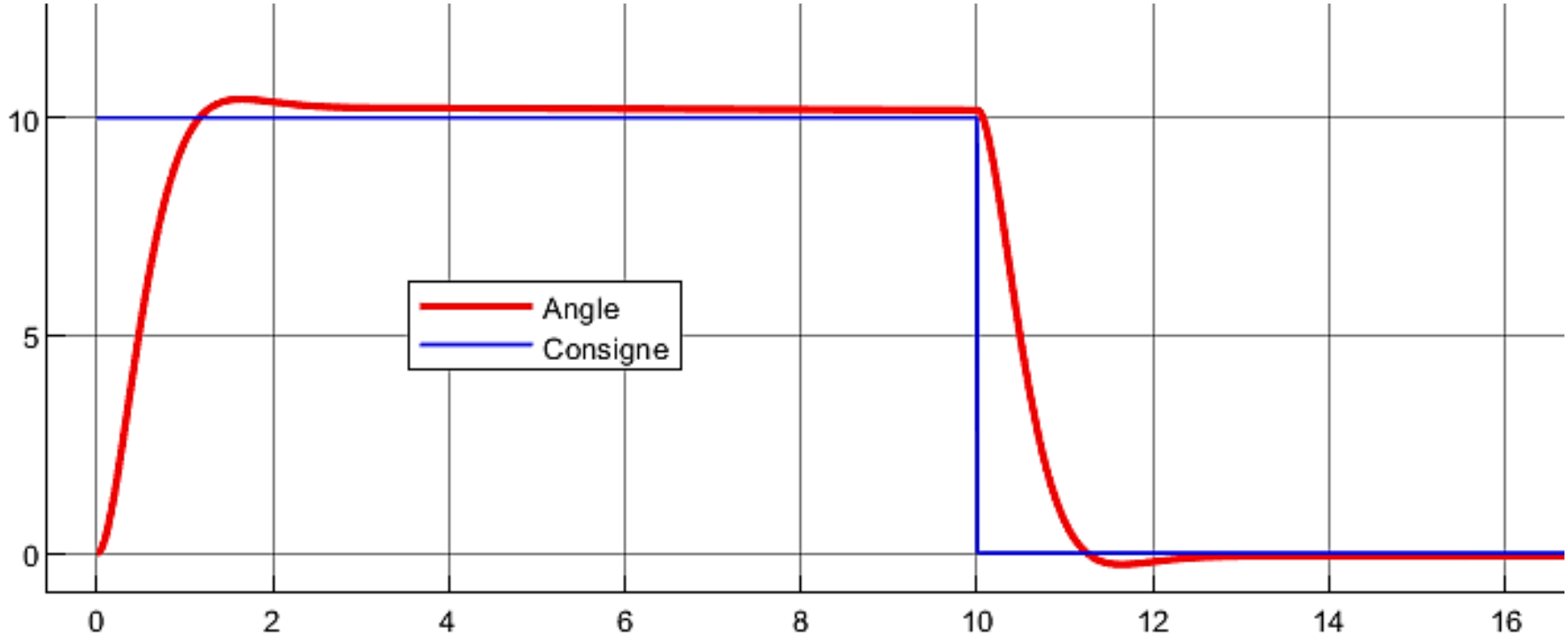


https://wikimemoires.net

Asservissement et simulations

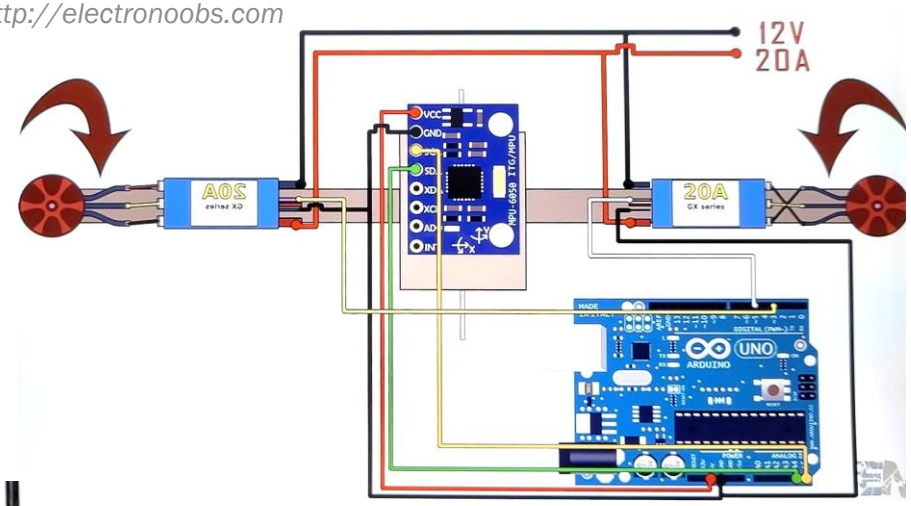
Détermination des constantes PID :

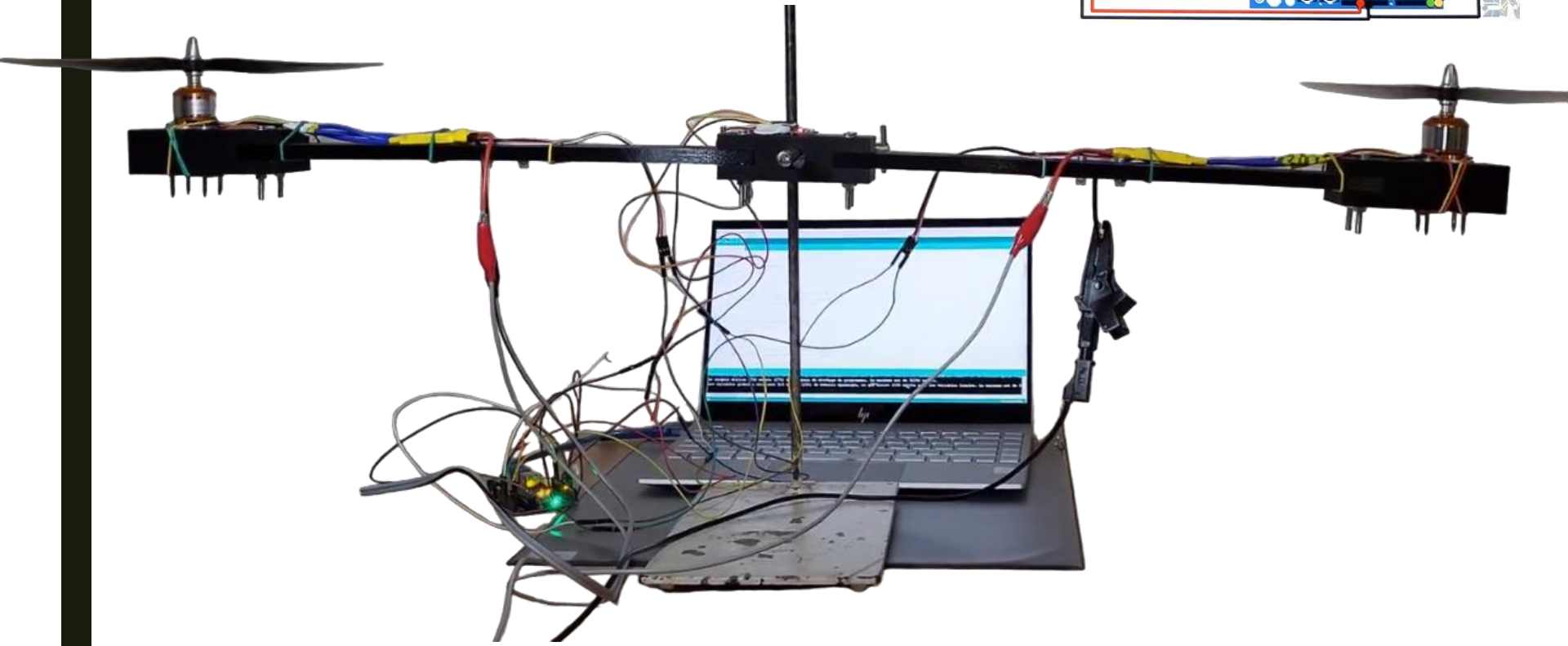
Proportional (P):	0.0002709467664342 92
Integral (I):	2.9372592021963e-06
Derivative (D):	0.00555324255393534



Résultats expérimentaux

http://electronoobs.com





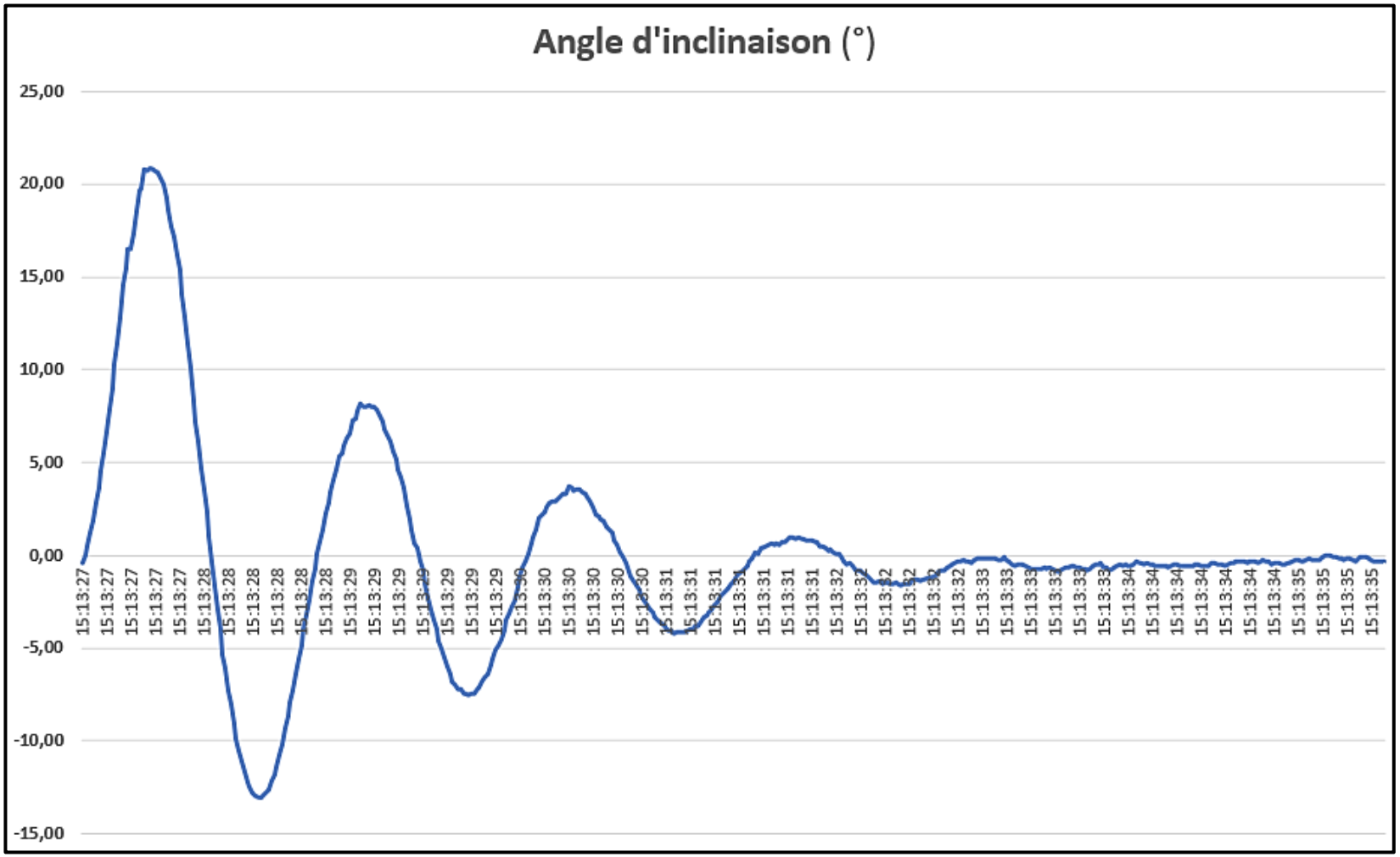
Résultats expérimentaux

PID_balance_arduino §

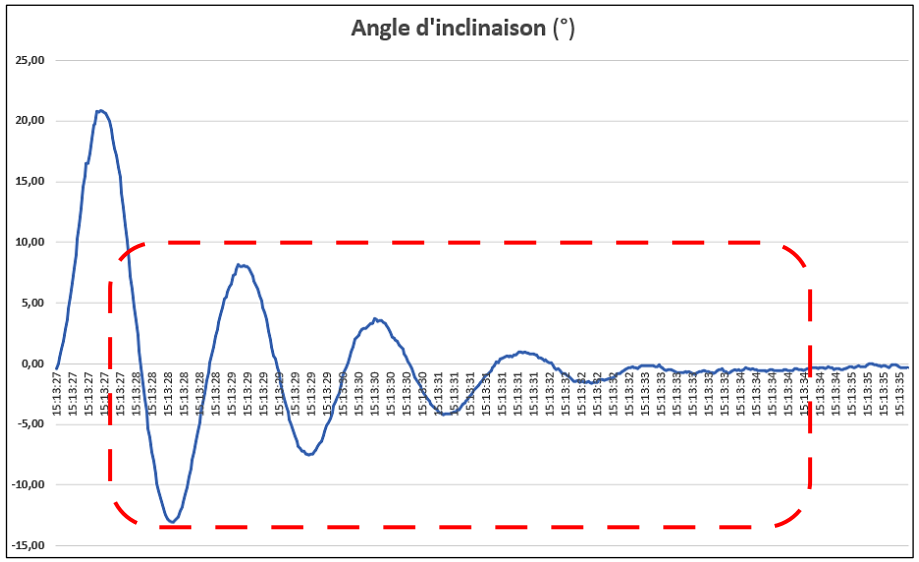
```
float pid_d=0;
////////////////// Constantes PID /////////////////
double kp=0.000270946766;
double ki=2.937259202e-6;
double kd=0.00555324255;
///////////////////////////////////////////////
double throttle=1100; // Valeur initiale de poussée des moteurs
float desired_angle = 0; // Consigne de position
```

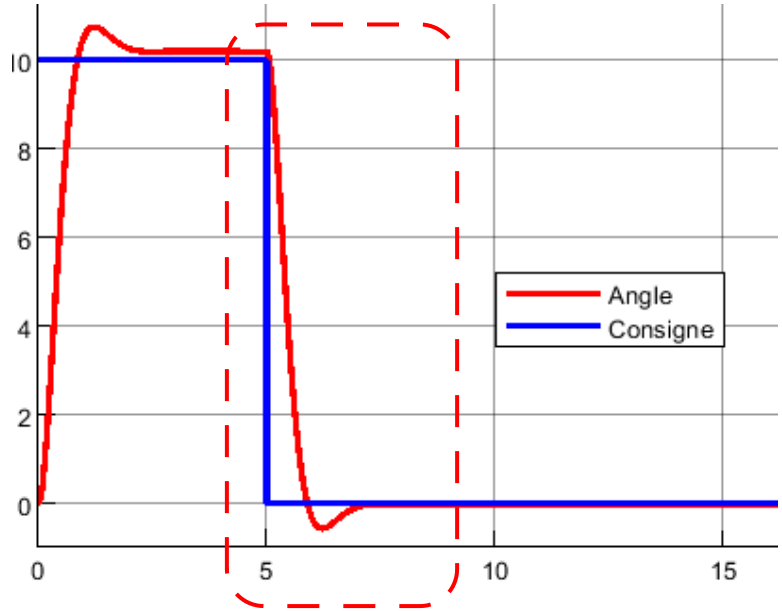
```
////////////////// Filtre complémentaire /////////////////
   /*---X axis angle---*/
   Total_angle[1] = 0.98 *(Total_angle[1] + Gyro_angle[1]*elapsedTime)
   + 0.02*Acceleration_angle[1];

    Serial.print(("DATA,TIME,"));
    Serial.println(Total_angle[1]);
//////////////////// P I D ////////////////////////////
error = Total_angle[1] - desired_angle; // Calul de l'erreur
pid_p = kp*error; // Effet proportionnel
/* L'intégrale doit intervenir lorsque on est près de la consigne : |error|<3*/
if(-3 <error <3)
{
  pid_i = pid_i+(ki*error); // Effet intégrale
}
/* On calcule la dérivée de l'erreur */
pid_d = kd*((error - previous_error)/elapsedTime); // Effet dérivée
PID = pid_p + pid_i + pid_d;  // On somme les 3 pour calculer le PID
```

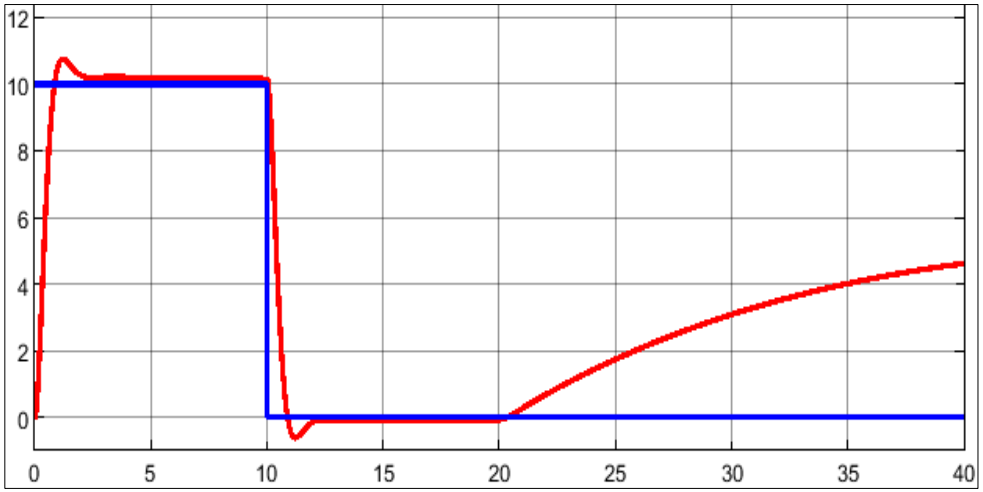
Résultats expérimentaux

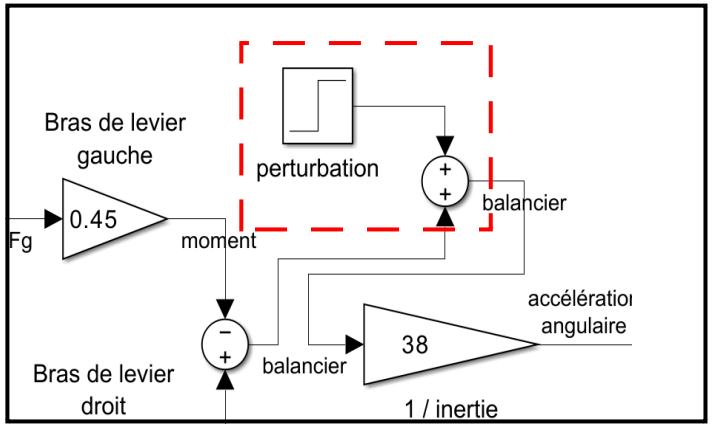


Conclusion et perspective





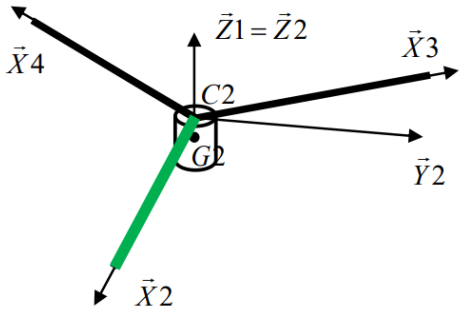


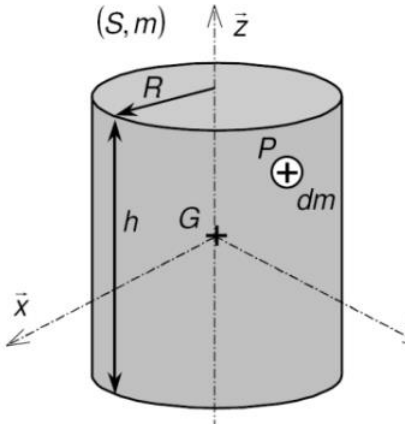


https://dronexl.co

MERCI

ANNEXE





$$[I]_{rotor\,moteur,G_2} = \begin{pmatrix} \frac{m_r R^2}{4} + \frac{m_r h^2}{12} & 0 & 0 \\ 0 & \frac{m_r R^2}{4} + \frac{m_r h^2}{12} & 0 \\ 0 & 0 & \frac{m_r R^2}{2} \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_2)}$$

$$[I]_{pale2,C_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \int x^2.dm & 0 \\ 0 & 0 & \int x^2.dm \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_2)}$$

$$\int x^2.dm = \frac{m_p}{l}\int_0^l x^2 dx = m_p \frac{l^2}{3}$$

$$[I]_{pale2,C_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & m_p \frac{l^2}{3} & 0 \\ 0 & 0 & m_p \frac{l^2}{3} \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_2)}$$

ANNEXE

$$C_h = 2m_p\frac{l^2}{3} = m_h\frac{l^2}{3}$$

On utilise le théorème de Huygens :

$$[I]_{hélice,G2} = [I]_{\substack{hélice,C2\\cdg}} + [I]_{m\in C,G2} \qquad \overrightarrow{G_2C_2} = a\,\vec{Z}_1$$

Donc :

$$[I]_{hélice,G2} = \begin{pmatrix} A_h & 0 & 0 \\ 0 & B_h & 0 \\ 0 & 0 & C_h \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_1)} + m_h \begin{pmatrix} a^2 & 0 & 0 \\ 0 & a^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_1)}$$

$$[I]_{hélice,G2} = \begin{pmatrix} A_h + m_h.a^2 & 0 & 0 \\ 0 & B_h + m_h.a^2 & 0 \\ 0 & 0 & C_h \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_1)}$$

$$[I]_{\{E2\},G2} = [I]_{rotor\,moteur,G2} + [I]_{hélice,G2}$$

$$[I]_{\{E2\},G2} = \begin{pmatrix} \frac{m_r R^2}{4} + \frac{m_r h^2}{12} + A_h + m_h.a^2 & 0 & 0 \\ 0 & \frac{m_r R^2}{4} + \frac{m_r h^2}{12} + B_h + m_h.a^2 & 0 \\ 0 & 0 & \frac{m_r R^2}{2} + C_h \end{pmatrix}_{(\vec{x}_2,\vec{y}_2,\vec{z}_2)}$$

ANNEXE

$$[I]_{\{E2\},G2} = [I]_{rotor\,moteur,G2} + [I]_{hélice,G2}$$

$$[\boldsymbol{I}]_{\{\boldsymbol{E2}\},\boldsymbol{G2}} = \begin{pmatrix} \frac{\boldsymbol{m_r R^2}}{\boldsymbol{4}} + \frac{\boldsymbol{m_r h^2}}{\boldsymbol{12}} + \boldsymbol{A_h} + \boldsymbol{m_h a^2} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \frac{\boldsymbol{m_r R^2}}{\boldsymbol{4}} + \frac{\boldsymbol{m_r h^2}}{\boldsymbol{12}} + \boldsymbol{B_h} + \boldsymbol{m_h a^2} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \frac{\boldsymbol{m_r R^2}}{\boldsymbol{2}} + \boldsymbol{C_h} \end{pmatrix} = \begin{pmatrix} Ixx & 0 & 0 \\ 0 & Ixx & 0 \\ 0 & 0 & Izz \end{pmatrix}$$

$$[I]_{\{S1\},O} = \begin{pmatrix} \frac{m_{S1}}{12}((2d)^2+h^2) & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix} = \begin{pmatrix} I_{S1} & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix} =$$

$$\begin{pmatrix} 260.10^{-4}\ kg.m^2 & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix}$$

ANNEXE

$\overrightarrow{OG_2} = -d \cdot \vec{Y}_1$ par $\overrightarrow{OG_4} = +d \cdot \vec{Y}_1$; et β_2 par β_4

$$\vec{\sigma}_{G2(E2/R_0)} == \begin{pmatrix} Ixx & 0 & 0 \\ 0 & Ixx & 0 \\ 0 & 0 & Izz \end{pmatrix} . \begin{pmatrix} \dot{\alpha} \\ 0 \\ \dot{\beta}_2 \end{pmatrix}_{(\vec{x}_1,\vec{y}_1,\vec{z}_1)} = \begin{pmatrix} Ixx.\dot{\alpha} \\ 0 \\ Izz.\dot{\beta}_2 \end{pmatrix}_{(\vec{x}_1,\vec{y}_1,\vec{z}_1)}$$

$$\vec{\delta}_{G_2(E2/R_0)} = \frac{d}{dt}(\vec{\sigma}_{G_2(E2/R_0)})_0 = Ixx.\ddot{\alpha}.\vec{X}_1 + Izz.\ddot{\beta}_2.\vec{Z}_1 - Izz.\dot{\beta}_2.\dot{\alpha}.\vec{Y}_1$$

$$\vec{\delta}_{O(E2/R_0)} = \vec{\delta}_{G_2(E2/R_0)} + \overrightarrow{OG_2} \wedge m.\vec{\Gamma}^{G_2}_{E2/R_0} \text{ avec } \vec{V}^{G_2}_{E2/R_0} = -d.\dot{\alpha}.\vec{Z}_1 \text{ et}$$

$$\vec{\Gamma}^{G_2}_{E2/R_0} = -d.\ddot{\alpha}.\vec{Z}_1 + d.\dot{\alpha}^2.\vec{Y}_1$$

On trouve : $\boxed{\vec{\delta}_{O(E2/R_0)} = (Ixx + m.d^2).\ddot{\alpha}.\vec{x}_1 - Izz.\dot{\beta}_2.\dot{\alpha}.\vec{y}_1 + Izz.\ddot{\beta}_2.\vec{z}_1}$

$$\vec{\delta}_{O(E4/R_0)} = (Ixx + m.d^2).\ddot{\alpha}.\vec{x}_1 - Izz.\dot{\beta}_4.\dot{\alpha}.\vec{y}_1 + Izz.\ddot{\beta}_4.\vec{z}_1$$

{ S1 }est un solide en rotation autour de l'axe $\left(O\vec{X}_1\right)$ fixe ; donc :

$$\boxed{\vec{\delta}_{O(S1/R_0)} = I_{S1}.\ddot{\alpha}.\vec{x}_1}$$

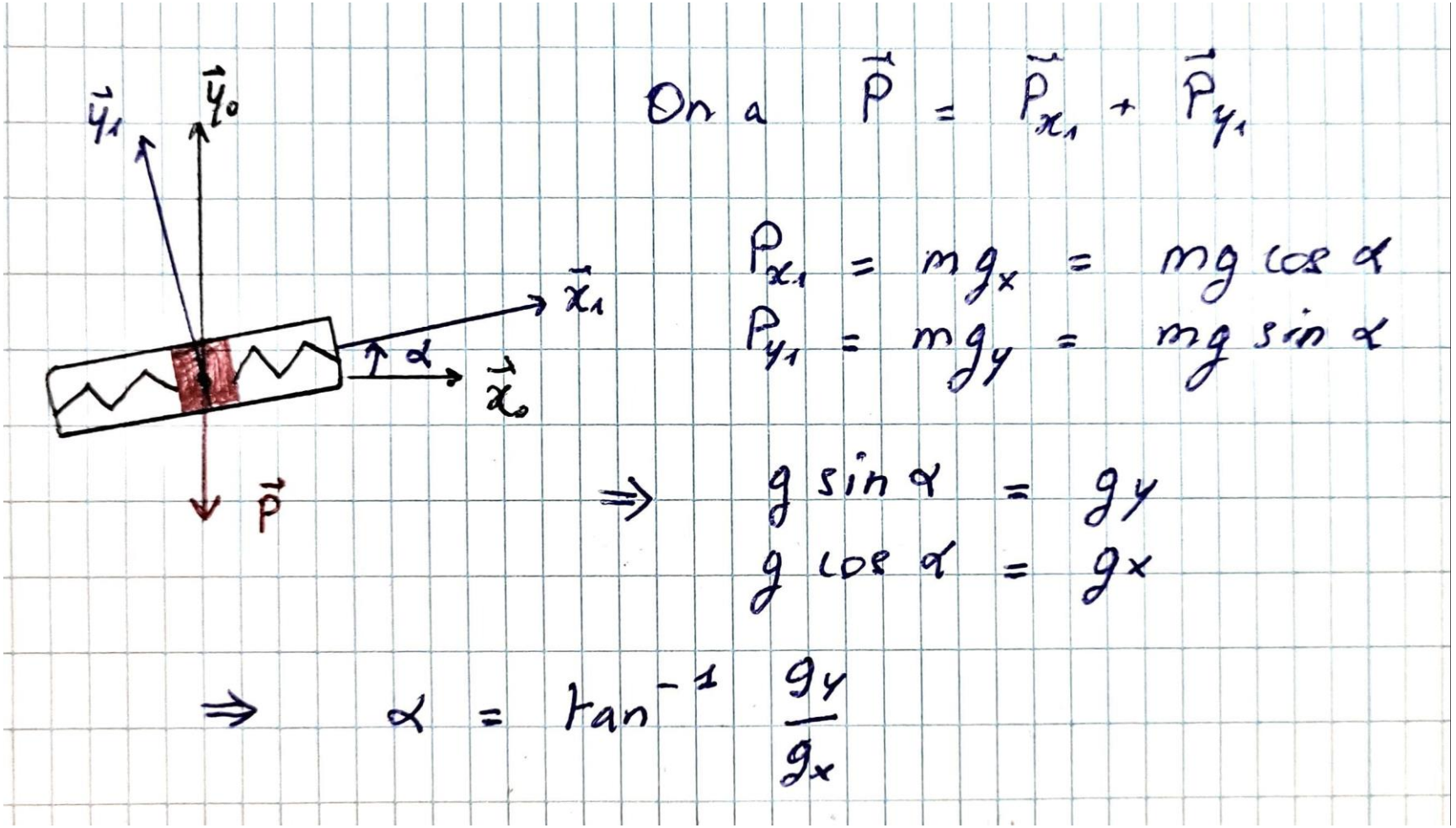
ANNEXE

Calibration de l'ESC

```
#include<Servo.h>
Servo esc; //Création d'une classe servo avec le nom esc

void setup()
{
esc.attach(8); // ESC connecté sur PIN8 de Arduino
esc.writeMicroseconds(1000); //initialiser le signal à 1000 (= 0 tr/min)
Serial.begin(9600);
}

void loop()
{
int val; //Création d'une variable val
val= analogRead(A0); // val = Valeur du potentiomètre connecté sur PIN A0
val= map(val, 0, 1023,1000,2000); // val est comprise entre 0 et 1023,
                                  // nous convertissons cette plage
                                  // entre 1000 et 2000 pour l'ESC
esc.writeMicroseconds(val); //envoi de val comme signal à ESC
}
```

ANNEXE

Formule d'Euler :



ANNEXE

Lecture de l'angle α avec MPU-6050

```
#include<Wire.h>

const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;  //déclarations des variables :
int minVal=265;                        //accéléromètre, température et gyroscope
int maxVal=402;
double x;   //double : équivalent à float
double y;
double z;
double r;

void setup(){
  Wire.begin();   //initialise la bibliothèque Wire
  Wire.beginTransmission(MPU_addr);   //on spécifie l'addresse
  Wire.write(0x6B);
  Wire.write(0);    // initialise à 0 (réveille la MPU6050)
  Wire.endTransmission(true);
  Serial.begin(9600);       //moniteur série
}
```

```
void loop(){
  Wire.beginTransmission(MPU_addr); //connexion a MPU-6050
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);
  AcX=Wire.read()<<8|Wire.read();
  AcY=Wire.read()<<8|Wire.read();
  AcZ=Wire.read()<<8|Wire.read();
    int xAng = map(AcX,minVal,maxVal,-90,90);
    int yAng = map(AcY,minVal,maxVal,-90,90);
    int zAng = map(AcZ,minVal,maxVal,-90,90);

       x= RAD_TO_DEG * (atan2(-yAng, -zAng));
       if (x<0){
        r=x+ 180;   //angles compris entre
       } else{      //  -180° et 180°
        r=x-180;
       }

    Serial.print("AngleX= ");
    Serial.println(r);

    delay(100);   // à diminuer pour plus de précision
}
```

ANNEXE

Filtre complémentaire :

$$\alpha = H_{pbas}(p).\alpha_{acc} + H_{phaut}(p).\alpha_{gyr}$$
$$= \frac{1}{1+T_C.p}.\alpha_{acc} + \frac{T_C.p}{1+T_C.p}.\alpha_{gyr}$$
$$= \frac{1}{1+T_C.p}.\alpha_{acc} + \frac{T_C.p}{1+T_C.p}.\frac{1}{p}.\omega_{gyr}$$

$$\boxed{K = \frac{T_S}{T_C + T_S}}$$

$$\leftrightarrow (1+T_C.p).\alpha = \alpha_{acc} + T_C\omega_{gyr}$$

$$\leftrightarrow \alpha(t) + T_C\frac{d\alpha(t)}{dt} = \alpha_{acc}(t) + T_C\omega_{gyr}(t)$$

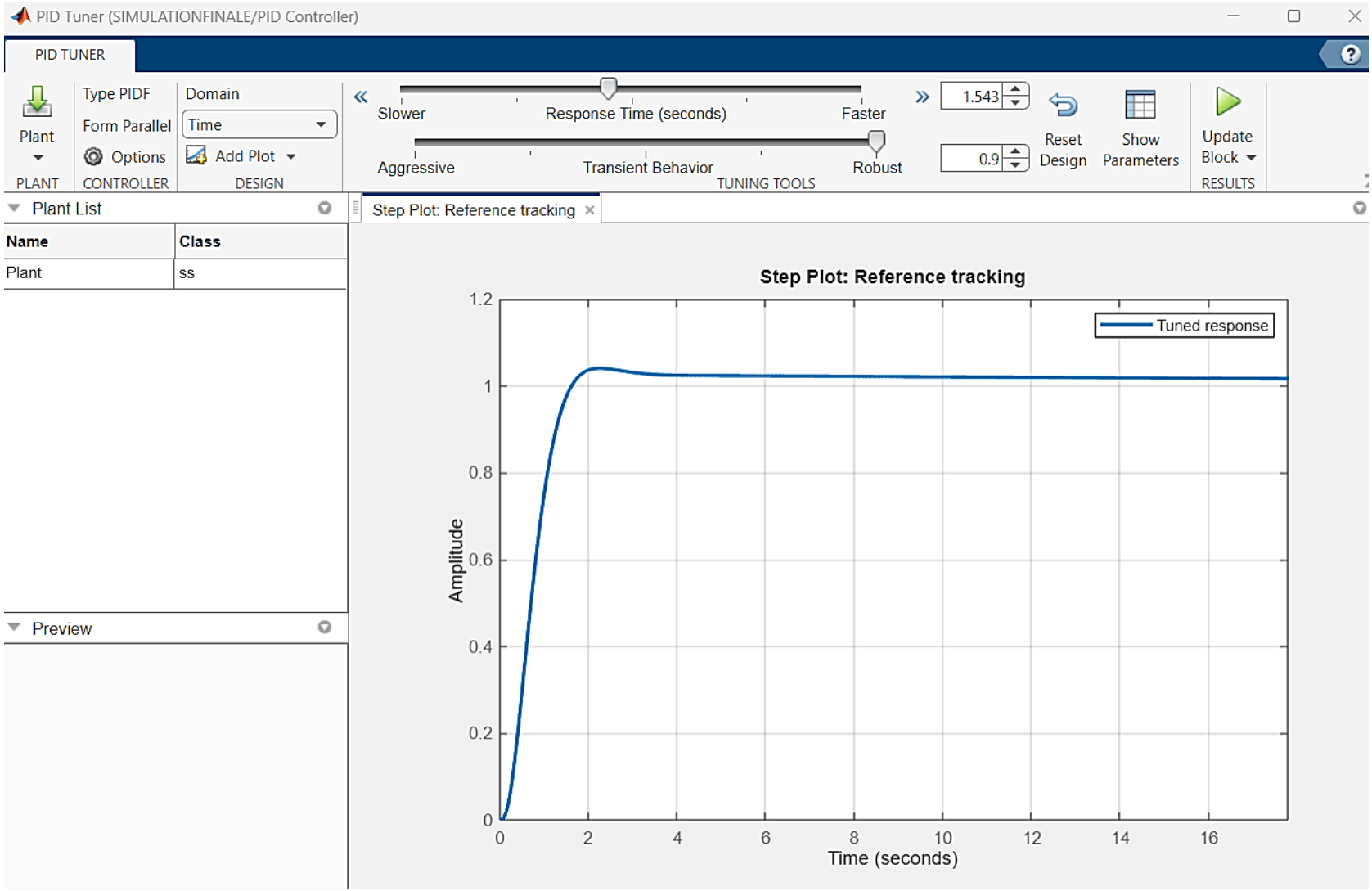
$$T_C = \frac{1}{6{,}28}\ s$$
$$T_S = 0{,}0032$$

$$\rightarrow \alpha(n) + T_C\frac{\alpha(n) - \alpha(n-1)}{T_S} = \alpha_{acc}(n) + T_C\omega_{gyr}(n)$$

$$\rightarrow \alpha(n) = \frac{T_C}{T_C+T_S}\alpha(n-1) + \frac{T_S}{T_C+T_S}\alpha_{acc}(n) + \frac{T_C.T_S}{T_C+T_S}\omega_{gyr}(n)$$

$$\rightarrow \alpha(n) = (1-K).[\alpha(n-1) + T_S\omega_{gyr}(n)] + K.\alpha_{acc}(n)$$

ANNEXE


PID Tuner (SIMULATIONFINALE/PID Controller)
PID TUNER
Plant
Type PIDF
Form Parallel
Options
Domain
Time
Add Plot
PLANT
CONTROLLER
DESIGN
Slower
Response Time (seconds)
Faster
Aggressive
Transient Behavior
Robust
TUNING TOOLS
1.543
0.9
Reset Design
Show Parameters
Update Block
RESULTS
Plant List
Name
Class
Plant
ss
Preview
Step Plot: Reference tracking
Step Plot: Reference tracking
Tuned response
Amplitude
Time (seconds)
0
0.2
0.4
0.6
0.8
1
1.2
0
2
4
6
8
10
12
14
16

ANNEXE

PID_balance_arduino

```
#include <Wire.h>
#include <Servo.h>
Servo right_prop;
Servo left_prop;
/* La MPU6050 fournit des données en 16 bits donc on crée des variables 16 int */
int16_t Acc_rawX, Acc_rawY, Acc_rawZ,Gyr_rawX, Gyr_rawY, Gyr_rawZ;
/* Création des variables nécéssaires */
float Acceleration_angle[2];
float Gyro_angle[2];
float Total_angle[2];
float elapsedTime, time, timePrev;
int i;
float rad_to_deg = 180/3.141592654;
float PID, pwmLeft, pwmRight, error, previous_error;
float pid_p=0;
float pid_i=0;
float pid_d=0;

////////////////// Constantes PID /////////////////
double kp=0.000270946766;
double ki=2.937259202e-6;
double kd=0.00555324255;
///////////////////////////////////////////////////
```

```
double throttle=1100; // Valeur initiale de poussée des moteurs
float desired_angle = 0; // Consigne de position

void setup() {
  Wire.begin();
  Wire.beginTransmission(0x68); // Connexion à MPU6050
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
  Serial.println(F("CLEARDATA"));
  Serial.println(F("LABEL,Temps,Angle"));

  right_prop.attach(3); // Moteur droit connecté au PIN3
  left_prop.attach(5);  // Moteur gauche connecté au PIN5

  time = millis();
  /*On établit la connexion avec les ESC*/
  left_prop.writeMicroseconds(1000);
  right_prop.writeMicroseconds(1000);
  delay(7000);
}
```

```
void loop() {
    timePrev = time;  // Temps précedent enregistré
    time = millis();  // Lecture du temps actuel
    elapsedTime = (time - timePrev) / 1000; // Calcul de dt en secondes

////////////////// Calcul de l'inclinaison ////////////////

     Wire.beginTransmission(0x68); // Connexion à l'accéléromètre
     Wire.write(0x3B); //Ask for the 0x3B register- correspond to AcX
     Wire.endTransmission(false);
     Wire.requestFrom(0x68,6,true);
     Acc_rawX=Wire.read()<<8|Wire.read(); //each value needs two registres
     Acc_rawY=Wire.read()<<8|Wire.read();
     Acc_rawZ=Wire.read()<<8|Wire.read();

    /*/// On va calculer les angles grâce aux équations d'Euler///*/
    /* Pour obtenir les valeurs de l'accélération dans l'unité de "g"
     il faut diviser les valeurs lues de la MPU6050 par 16384 d'après la fiche technique.
     Puis on convertit les rad en degré.
     On apllique les formules d'Euler*/
     /*---X---*/
     Acceleration_angle[0] = atan((Acc_rawY/16384.0)/sqrt(pow((Acc_rawX/16384.0),2)
     + pow((Acc_rawZ/16384.0),2)))*rad_to_deg;
```

```
    /*---Y---*/
    Acceleration_angle[1] = atan(-1*(Acc_rawX/16384.0)/sqrt(pow((Acc_rawY/16384.0),2)
    + pow((Acc_rawZ/16384.0),2)))*rad_to_deg;

  Wire.beginTransmission(0x68); // Connexion au gyroscope
  Wire.write(0x43);
  Wire.endTransmission(false);
  Wire.requestFrom(0x68,4,true);
  Gyr_rawX=Wire.read()<<8|Wire.read();
  Gyr_rawY=Wire.read()<<8|Wire.read();

  /*Pour obtenir les valeurs du gyroscope en deg/s
  il faut les diviser par 131 d'après la fiche technique*/
  /*---X---*/
  Gyro_angle[0] = Gyr_rawX/131.0;
  /*---Y---*/
  Gyro_angle[1] = Gyr_rawY/131.0;

//////////////// Filtre complémentaire ////////////////
  /*---X axis angle---*/
  Total_angle[1] = 0.98 *(Total_angle[1] + Gyro_angle[1]*elapsedTime)
  + 0.02*Acceleration_angle[1];
```

```
    Serial.print(("DATA,TIME,"));
    Serial.println(Total_angle[1]);

//////////////////// P I D ////////////////////////////

error = Total_angle[1] - desired_angle; // Calul de l'erreur

pid_p = kp*error; // Effet proportionnel

/* L'intégrale doit intervenir lorsque on est près de la consigne : |error|<3*/
if(-3 <error <3)
{
  pid_i = pid_i+(ki*error); // Effet intégrale
}

/* On calcule la dérivée de l'erreur */
pid_d = kd*((error - previous_error)/elapsedTime); // Effet dérivée

PID = pid_p + pid_i + pid_d;  // On somme les 3 pour calculer le PID

/*Les valeurs min et max des ESC sont 1000 et 2000
donc PID varie de -1000 à 1000 (car si on est à 2000 on enlève au max 1000) */
if(PID < -1000)
```

```
{
  PID=-1000;
}
if(PID > 1000)
{
  PID=1000;
}

/*On ajoute le PID à la poussée*/
pwmLeft = throttle + PID;
pwmRight = throttle - PID;

if(pwmRight < 1000)
{
  pwmRight= 1000;
}
if(pwmRight > 2000)
{
  pwmRight=2000;
}
if(pwmLeft < 1000)
{
  pwmLeft= 1000;
}
if(pwmLeft > 2000)
{
  pwmLeft=2000;
}

left_prop.writeMicroseconds(pwmLeft); // On envoie le signal aux moteurs
right_prop.writeMicroseconds(pwmRight);
previous_error = error;
}
```