

Thème : Jeux, sports

Stabilisateur d'image cardan GIMBAL



PLAN

1 Introduction

2 Problématique

3 Objectifs

4 Conclusion

3.1 Modélisation du système
et choix du moteur

3.2 Détecter les mouvements indésirables
de la caméra

3.3 Optimisation et choix du matériau de
la plateforme porte-caméra

3.4 Réalisation d'un prototype

INTRODUCTION





PROBLÉMATIQUE

Comment atténuer les vibrations et les mouvements indésirables des caméras d'action ,qui entraîne des images floues ou saccadées, en temps réel afin d'obtenir une image fluide ?

OBJECTIFS



01

Modélisation du système et choix du moteur

02

Détecter les mouvements indésirables de la caméra

03

Optimisation et choix du matériau de la plateforme porte-caméra

04

Réalisation d'un prototype

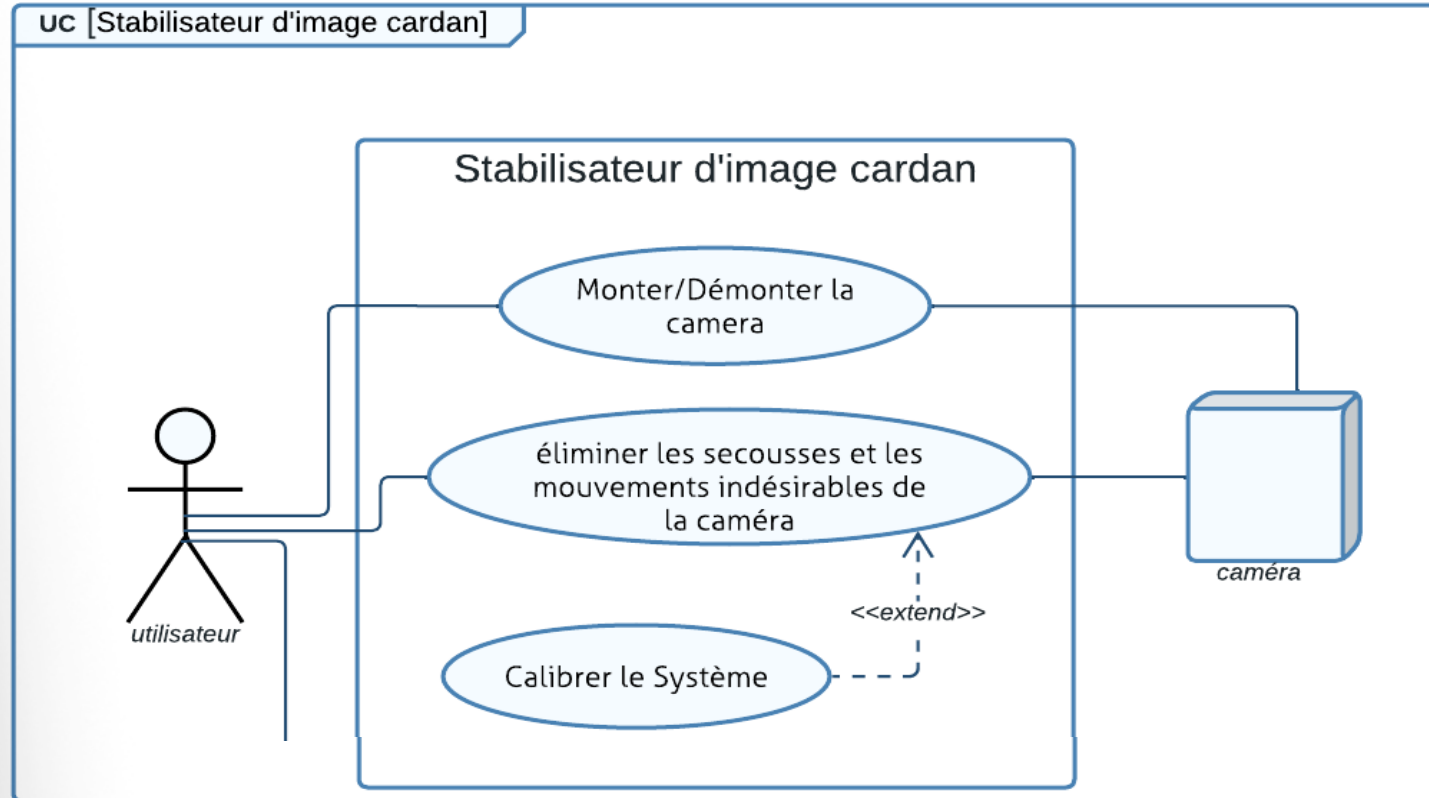
OBJECTIF 1

**Modélisation du système
et choix du moteur**

01 Modélisation du système et choix du moteur

Représentations SysML

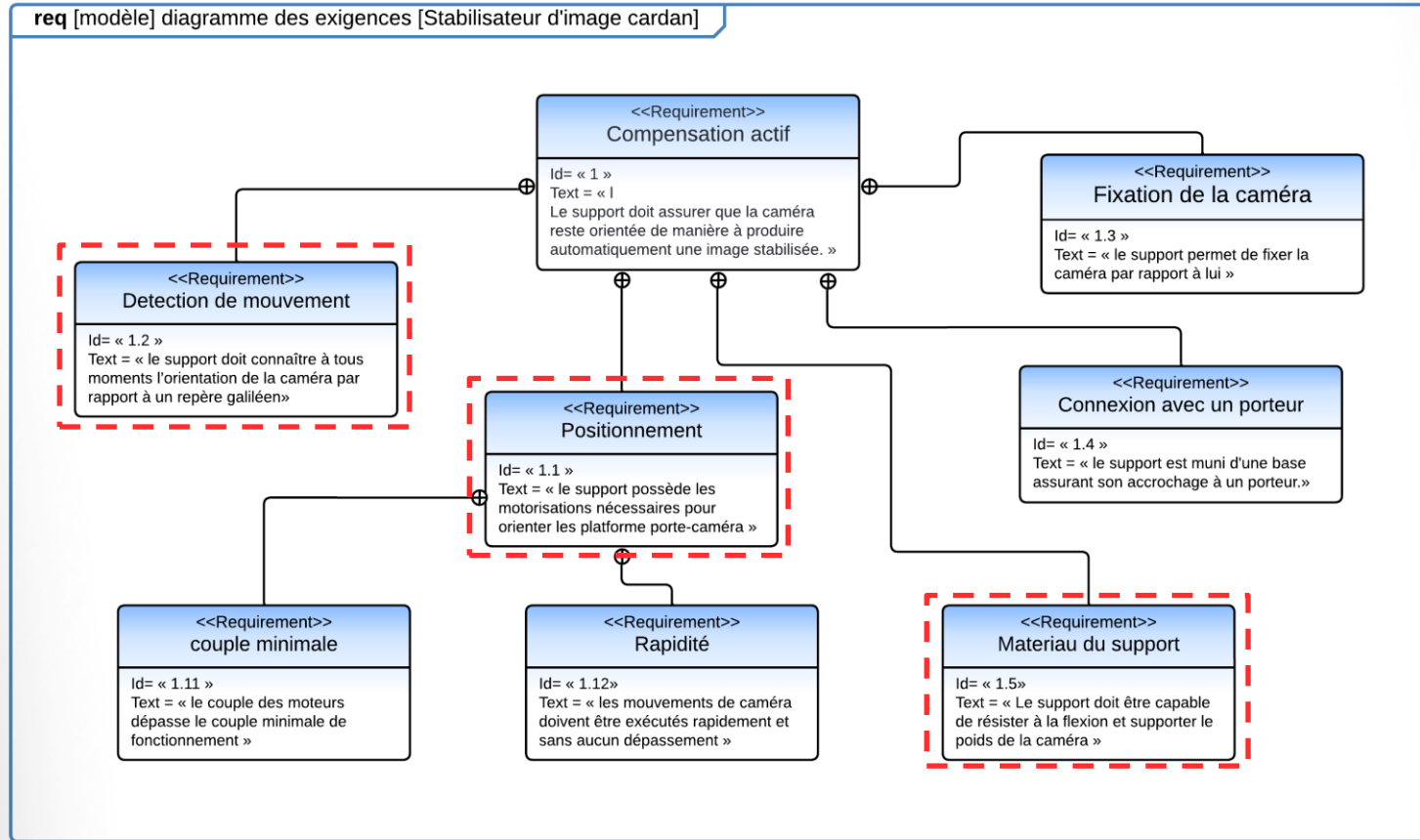
Figure 1: diagramme cas d'utilisation



01 Modélisation du système et choix du moteur

Représentations SysML

Figure 2: diagramme des exigences



01 Modélisation du système et choix du moteur

Représentations SysML

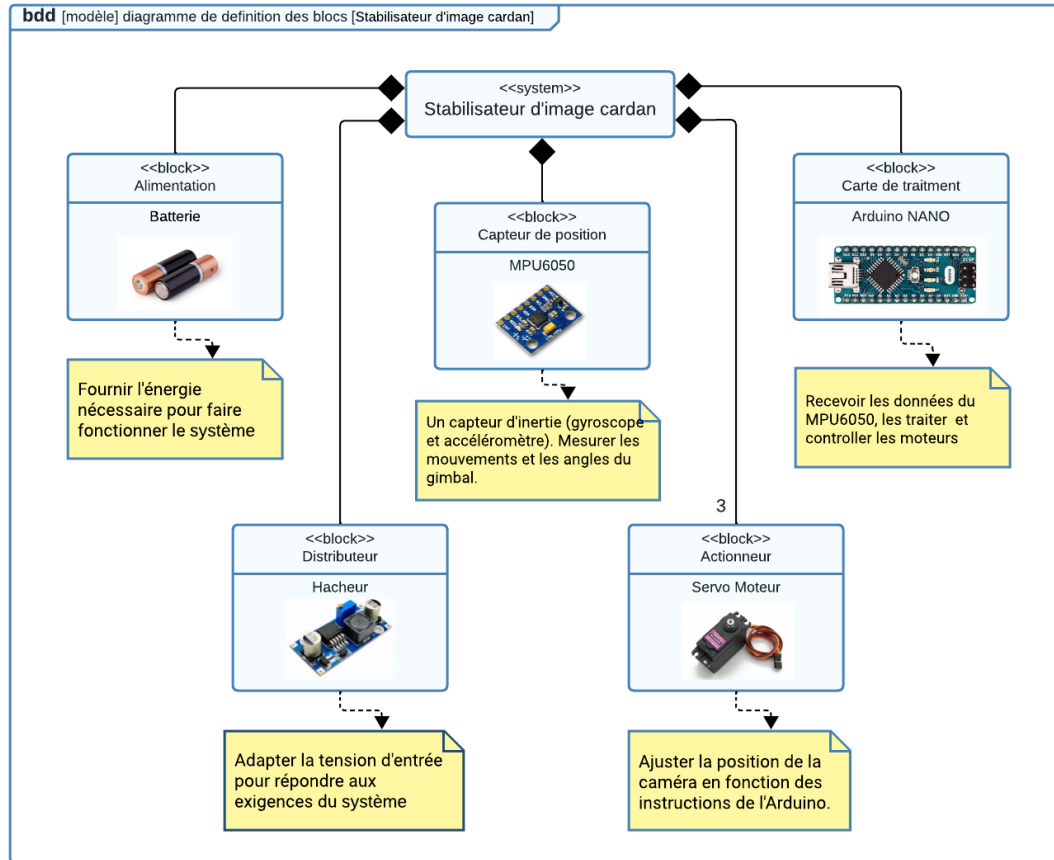


Figure 3:
diagramme de
definition des blocs

01 Modélisation du système et choix du moteur

Modélisation

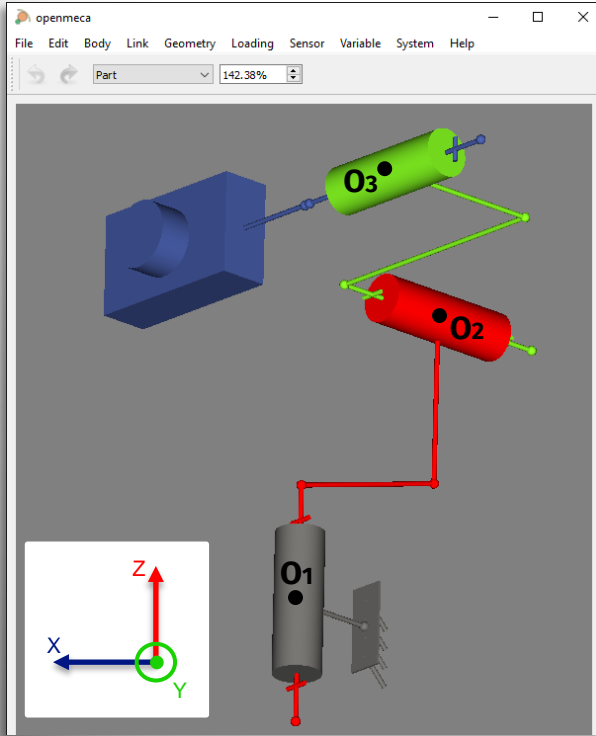


Figure 5: schéma cinématique

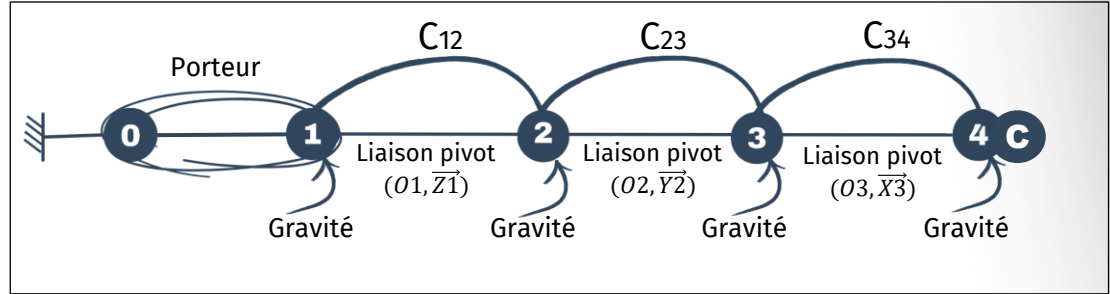


Figure 6: graphe de liaisons

$m = 3 \Rightarrow$ On exige 3 moteurs

HYPOTHÈSES :

- On néglige les effets du frottement dans les joints du gimbal
- Les liaisons sont parfaits
- On suppose que les masses sont concentrées aux centres de gravité de la caméra et des plateformes

01 Modélisation du système et choix du moteur

Modélisation

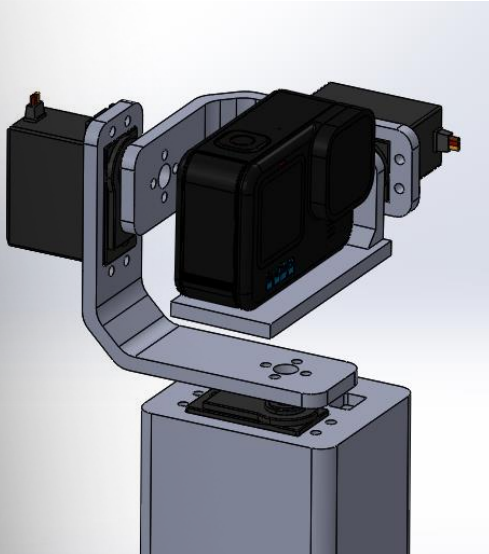
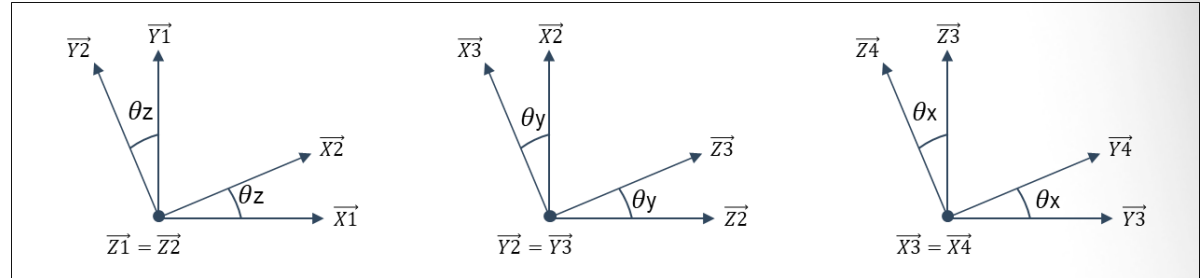


Figure 7: modelisation (solidworks)

Figure 8: figures plans

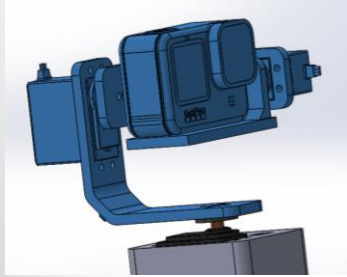


Couple moteur	Ensemble isolé	Théoreme utilisé	Justifications de l' isolement et du théoreme utilisé
C12	2+3+4+C	Moment dynamique en O1 en projection sur $\vec{Z1}$	$\vec{M}(O1,1 \rightarrow 2) \cdot \vec{Z1} = 0$
C23	3+4+C	Moment dynamique en O2 en projection sur $\vec{Y2}$	$\vec{M}(O2,2 \rightarrow 3) \cdot \vec{Y2} = 0$
C34	4+C	Moment dynamique en O3 en projection sur $\vec{X3}$	$\vec{M}(O3,3 \rightarrow 4 + c) \cdot \vec{X3} = 0$

01 Modélisation du système et choix du moteur

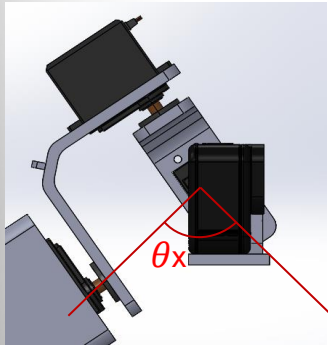
Calcul de C12 (Théorie)

Ensemble isolé : E=2+3+4+C



Moments of inertia: (grams * square millimeters)
Taken at the output coordinate system.

lxx = 1752325.83	lxy = 41919.14	lxz = -81302.28
lyx = 41919.14	lyy = 1522642.49	lyz = -326777.38
lzx = -81302.28	lzy = -326777.38	lzz = 815580.52



$\theta \in [30^\circ ; 120]$
 $\Rightarrow \theta_{\max} = 90^\circ$

$$\vec{M}(O1,1 \rightarrow 2). \vec{Z1} = 0$$

$$\vec{M}(O1,pesanteur \rightarrow 2). \vec{Z1} = \overrightarrow{O_1 G_2} \wedge m_2. g \vec{Z4}$$

$$\vec{M}(O2,pesanteur \rightarrow 3). \vec{Y2} = \overrightarrow{O_2 G_3} \wedge m_3. g \vec{Z4}$$

$$\vec{M}(O3,pesanteur \rightarrow 4 + c). \vec{X3} = \overrightarrow{O_3 G_{4+c}} \wedge m_{4+c}. g \vec{Z4}$$

$$\vec{M}(O1,moteur \rightarrow E). \vec{Z1} = C_{12}$$

$$\vec{\delta}(E/1). \vec{Z1} = I_{zz}. \dot{\theta} \vec{z}$$

Center of mass: (millimeters)
X = 0.00
Y = -36.35
Z = 15.30

Center of mass: (millimeters)
X = -22.54
Y = 15.65
Z = 0.00

Center of mass: (millimeters)
X = 43.40
Y = -4.29
Z = 7.75

TMD:

$C_{12} = 0.049 \text{ N.m}$

01 Modélisation du système et choix du moteur

Calcul de C12 (Simulation)

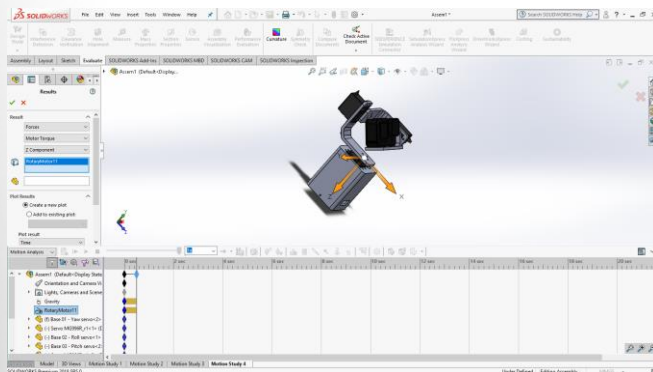
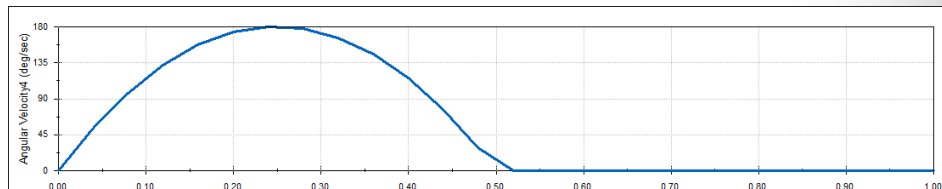
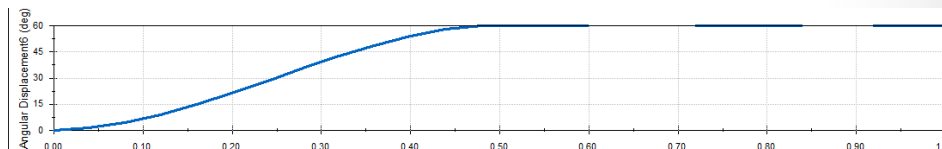


Figure 9: Analyse du mouvement (SW)

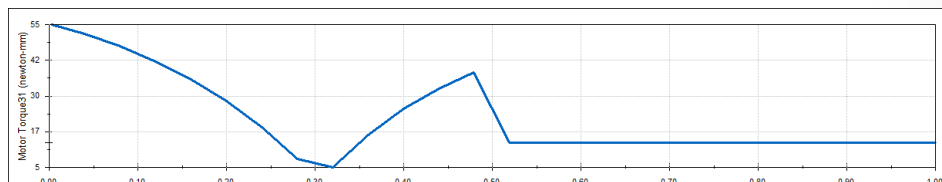
Vitesse angulaire :



Position angulaire :



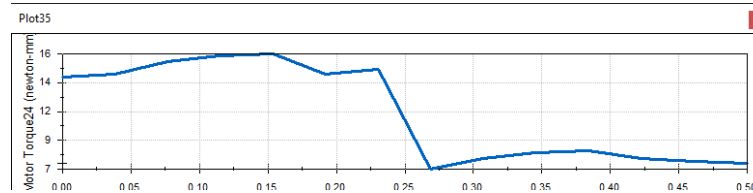
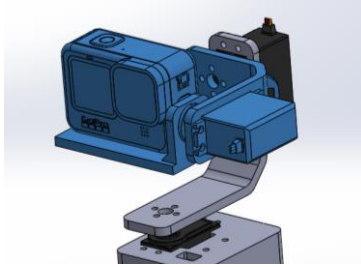
Couple moteur



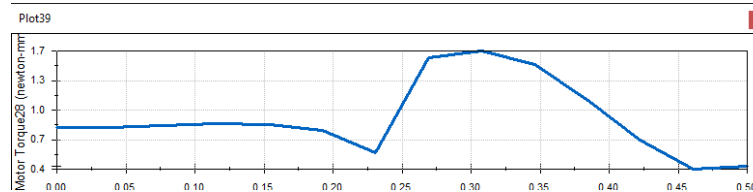
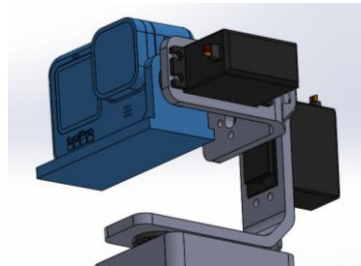
01 Modélisation du système et choix du moteur

Calcul de C23 et C34 (Simulation)

Ensemble isolé : E=3+4+C



Ensemble isolé : E=4+C



01 Modélisation du système et choix du moteur

Choix des moteurs et conclusion



- **Modèle** : SG90
- **Dimensions** : 22 x 11,5 x 27 mm
- **Poids** : 9g
- **Vitesse** : 0.12 sec/60° sous 4.8V
- **Couple** : 1.2Kg/cm sous 4.8V
- **Tension** : 4.8V – 6V



- **Modèle** : MG995
- **Dimensions** : 40,7 x 19,7 x 42,9 mm
- **Poids** : 55g
- **Vitesse** : 0.14 sec/60° sous 6.0V
- **Couple** : 6V 10.5 kg/cm
- **Tension** : 3,0V – 7,2V



- **Modèle** : MG996R
- **Dimensions** : 40,7 x 19,7 x 42,9 mm
- **Poids** : 55g
- **Vitesse** : 0.16 sec/60° sous 6.0V
- **Couple** : 6V 11kg/cm
- **Tension** : 3,0V – 7,2V

<<Requirement>>
Positionnement

Id= « 1.1 »

Text = « le support possède les motorisations nécessaires pour orienter les plateforme porte-caméra »



OBJECTIF 2

**Détecter les mouvements
Indésirables de la caméra**

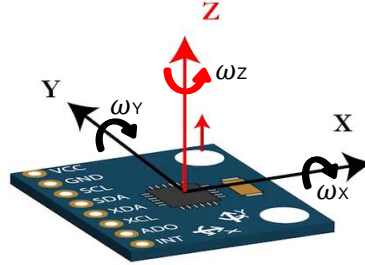
02 Détecter les mouvements indésirables de la caméra

Acquisition des angles (Théorie)

MPU6050



Gyromètre

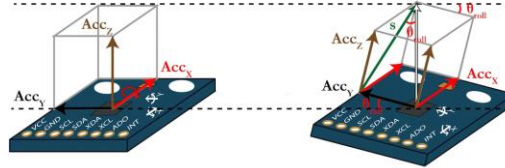


$$\theta = \int_0^{k.T} \omega . dt$$

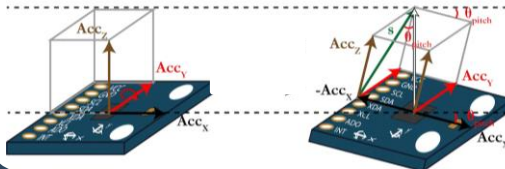
$$\theta(k) = \theta(k-1) + \omega(k) \times T$$



Accéléromètre



$$\theta_x = \tan^{-1} \left(\frac{-Acc_X}{\sqrt{Acc_Y^2 + Acc_Z^2}} \right)$$

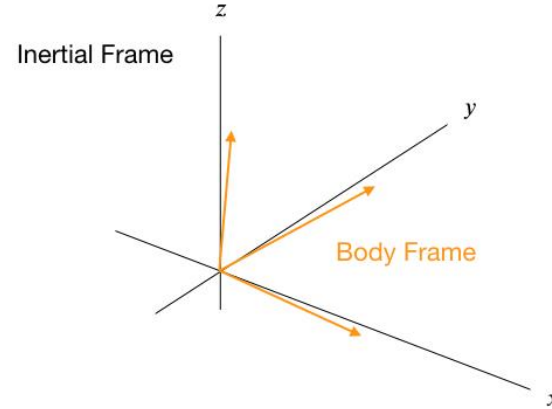
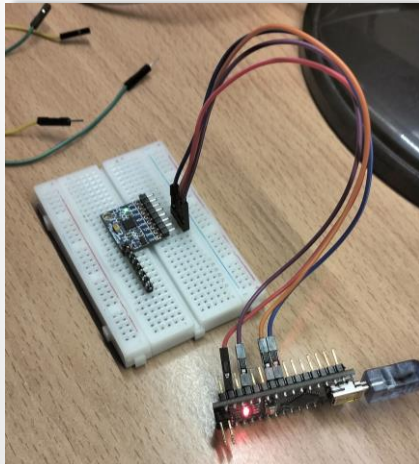
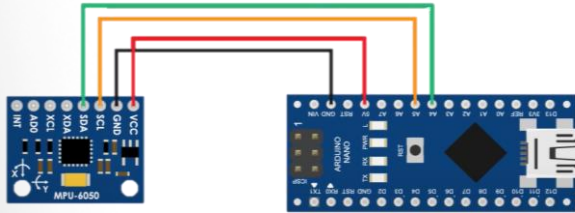


$$\theta_y = \tan^{-1} \left(\frac{Acc_Y}{\sqrt{Acc_X^2 + Acc_Z^2}} \right)$$

02 Détecter les mouvements indésirables de la caméra

Calibrage du capteur

MONTAGE EXPÉRIMENTAL 1



```
calculate_IMU_error()
```

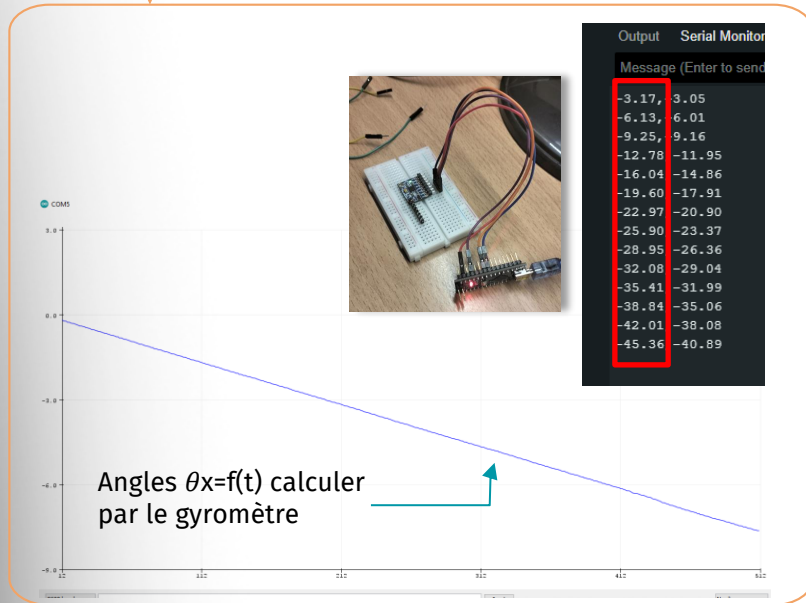
```
277  
278 GyroX = Wire.read() << 8 | Wire.read();  
279 GyroY = Wire.read() << 8 | Wire.read();  
280 GyroZ = Wire.read() << 8 | Wire.read();  
281  
282 // Sum all readings  
283 GyroErrorX += (GyroX / 131.0);  
284 GyroErrorY += (GyroY / 131.0);  
285 GyroErrorZ += (GyroZ / 131.0);  
286 c++;  
287  
288 // Divide the sum by 200 to get the error  
289 GyroErrorX /= 200;  
290  
291 Serial.println("GyroErrorX: " + String(GyroErrorX));  
292 Serial.println("GyroErrorY: " + String(GyroErrorY));  
293 Serial.println("GyroErrorZ: " + String(GyroErrorZ));  
294  
295 }
```

AccErrorX: -0.65
AccErrorY: -0.87
GyroErrorX: -1.46
GyroErrorY: -1.52
GyroErrorZ: 0.33

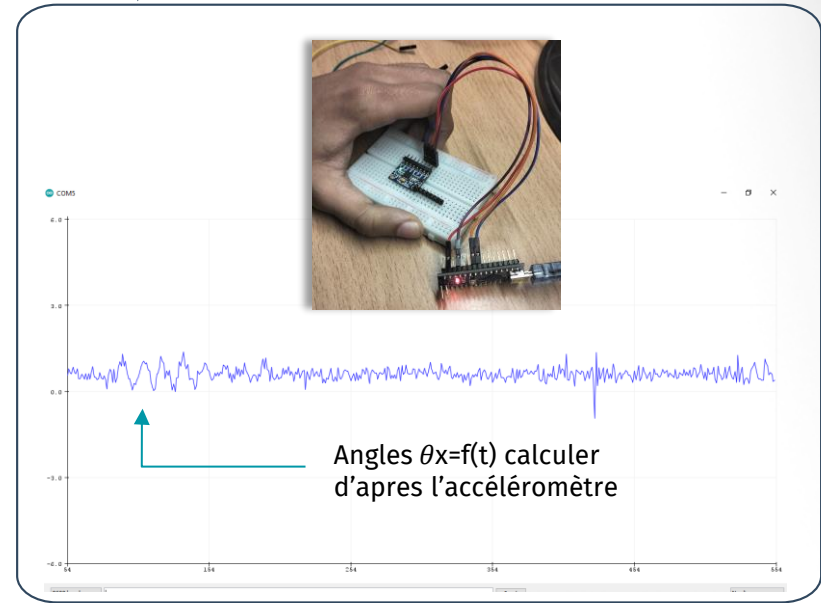
02 Détecter les mouvements indésirables de la caméra Acquisition des angles (Experience)



Gyromètre



Accéléromètre



02 Détecter les mouvements indésirables de la caméra

Filtrage (passe bas accéléromètre)

$$acc_{filtre}[n] = \alpha acc_{filtre}[n-1] + (1 - \alpha) acc[n]$$

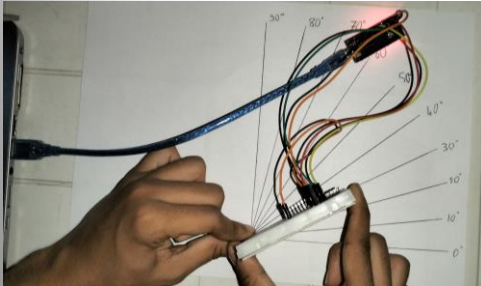
$$\Rightarrow \alpha = \frac{\Delta t}{\tau + \Delta t}$$

$$\Rightarrow \Delta t = \frac{1}{f_s}$$

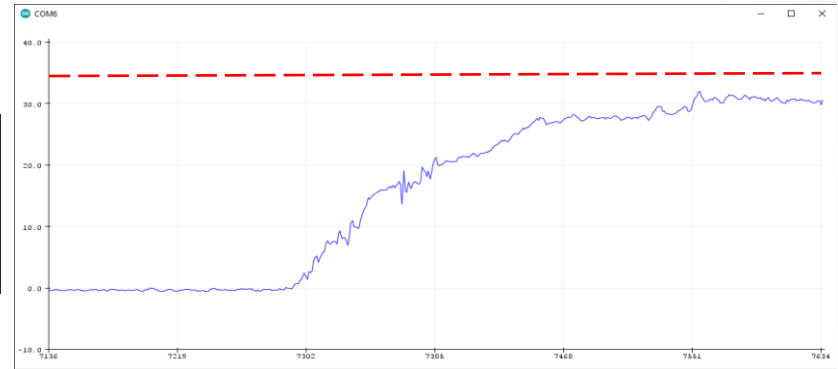
$$\Rightarrow \tau = 2\pi f_s$$

$$\Rightarrow f_s = 100 \text{ Hz}$$

Angle réelle : 35°

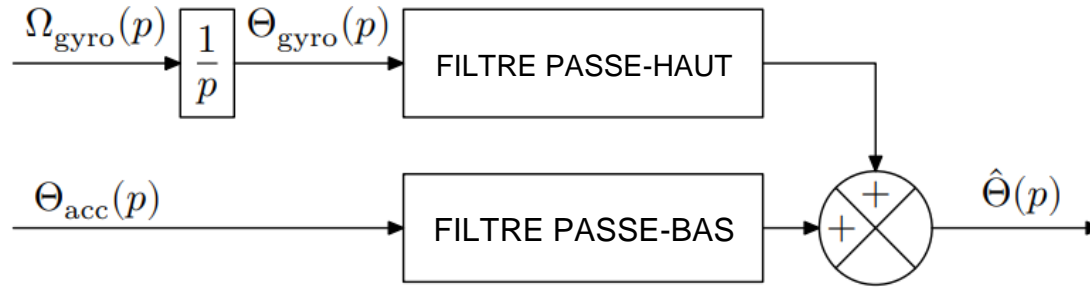


```
AccAngleX (Low-pass filter): -31.84  
AccAngleX (Low-pass filter): -31.81  
AccAngleX (Low-pass filter): -31.87  
AccAngleX (Low-pass filter): -31.78  
AccAngleX (Low-pass filter): -31.84  
AccAngleX (Low-pass filter): -31.82  
AccAngleX (Low-pass filter): -31.94
```



02 Détecter les mouvements indésirables de la caméra

Filtrage (par fusion)



Filtre complémentaire: $\theta = \frac{T}{1+T} \theta_{\text{acc}} + \frac{1}{1+T} (\theta_{\text{gyro}} + \Omega \cdot T)$

On pose : $\alpha = \frac{1}{1+T}$

$$\theta = (1 - \alpha) \theta_{\text{acc}} + \alpha [\theta_{\text{gyro}}(k-1) + \Omega_{\text{gyro}}(k) \times kT]$$

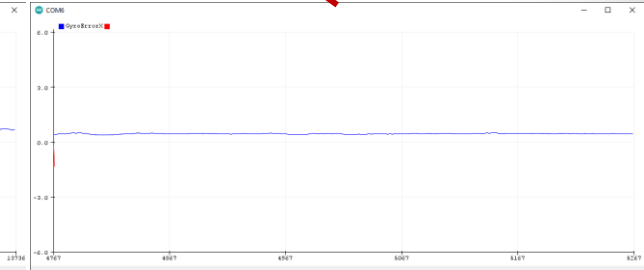
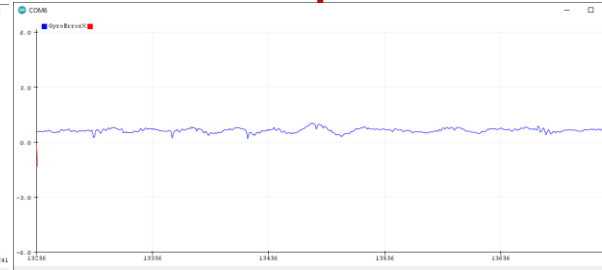
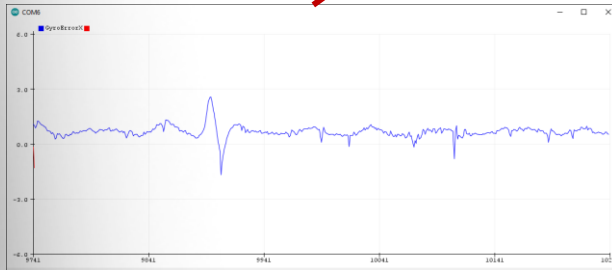
02 Détecter les mouvements indésirables de la caméra

Choix de α

On calcule la valeur moyenne des angles obtenus pour chaque α et on calcule l'erreur :

$$\varepsilon = \frac{|\theta_{\text{réel}} - \theta_{\text{mesuré}}|}{\theta_{\text{réel}}}$$

Valeur de α	0.65	0.76	0.84	0.94	0.96	0.98
ε	1.46	0.21	0.13	0.03	0.02	0.06

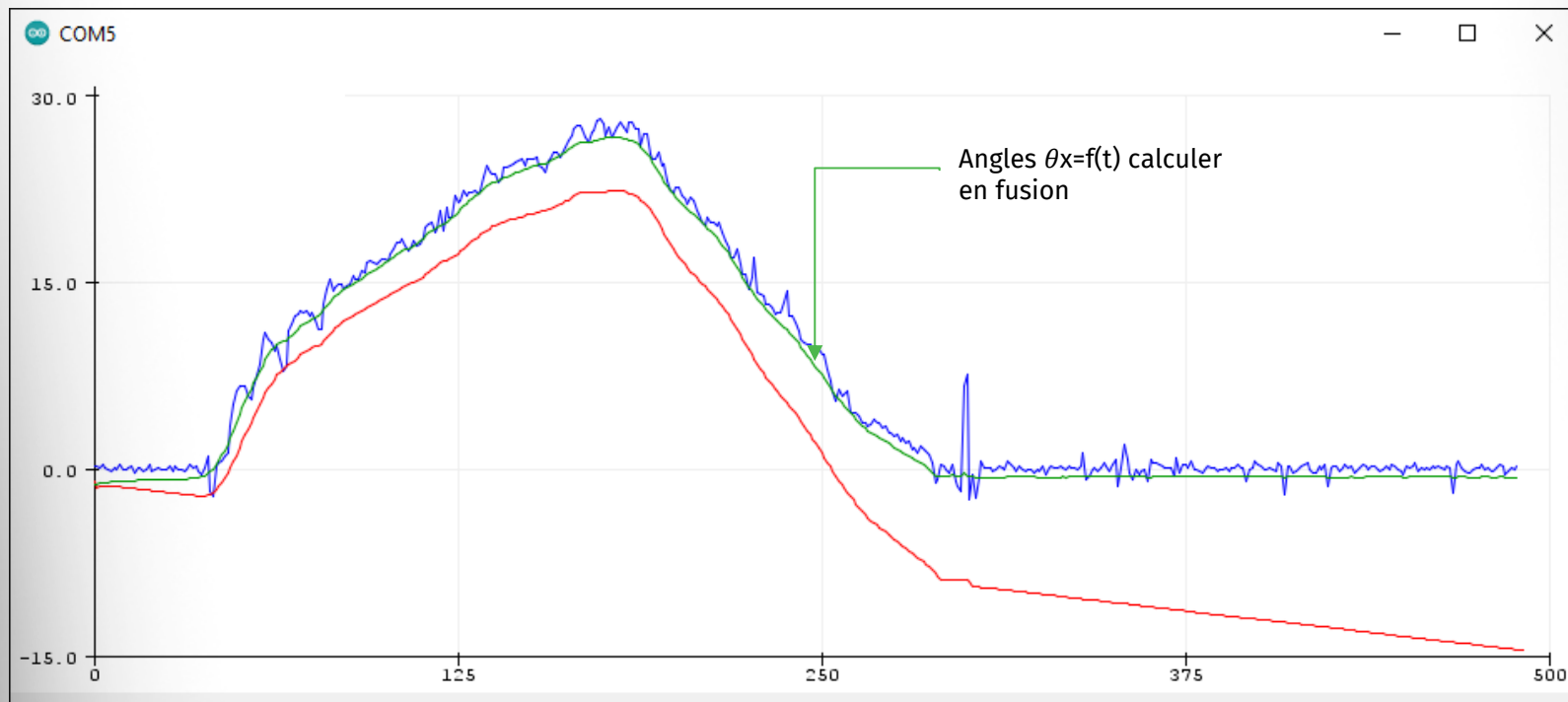


Bon choix de α : $\in [92 ; 98]$

02 Détecter les mouvements indésirables de la caméra

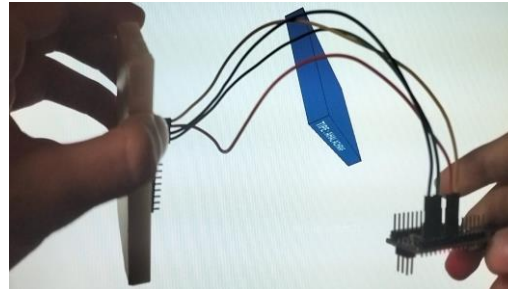
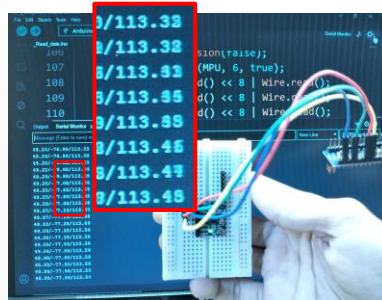
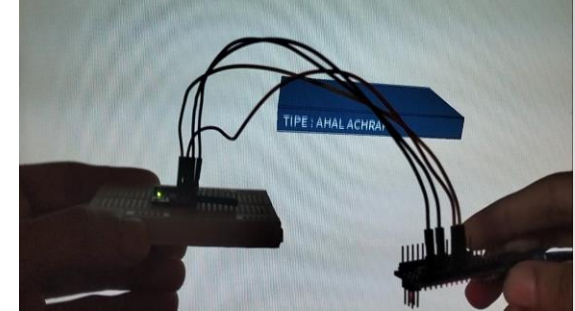
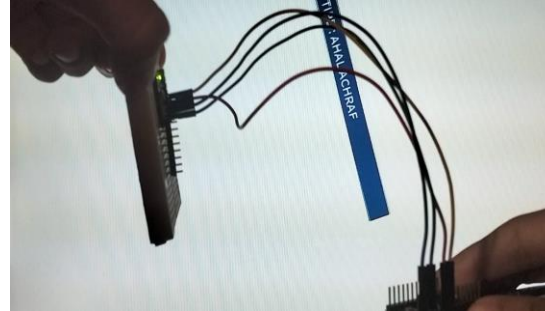
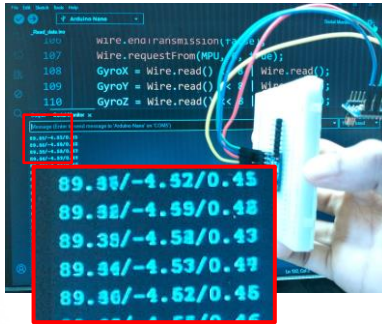
Filtrage (par fusion)

Figure 10: sortie du filter complémentaire

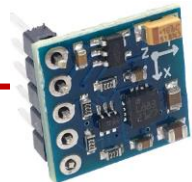


02 Détecter les mouvements indésirables de la caméra

Résultats et simulation



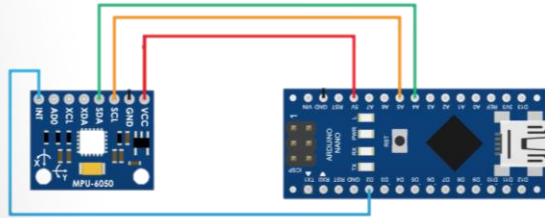
Remarque : Le driftage de l'angle autour de l'axe z persiste



02 Détecter les mouvements indésirables de la caméra

DMP (Traitement numérique du mouvement)

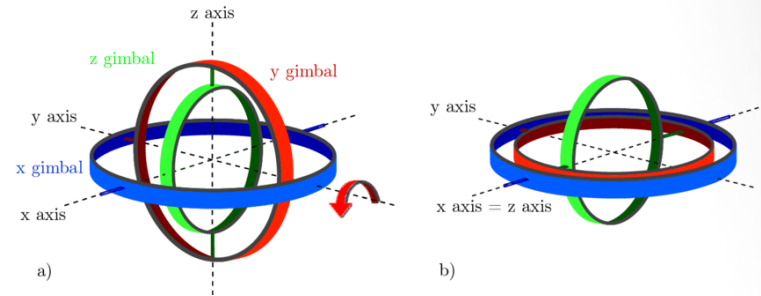
MONTAGE EXPÉRIMENTAL 2



l'IMU MPU6050 contient un DMP (Digital Motion Processor), qui calcule les résultats en termes de quaternions.

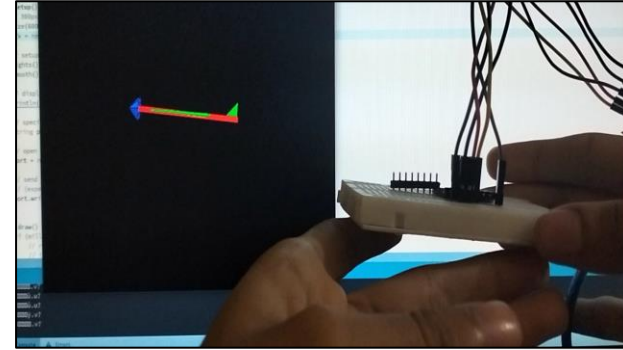
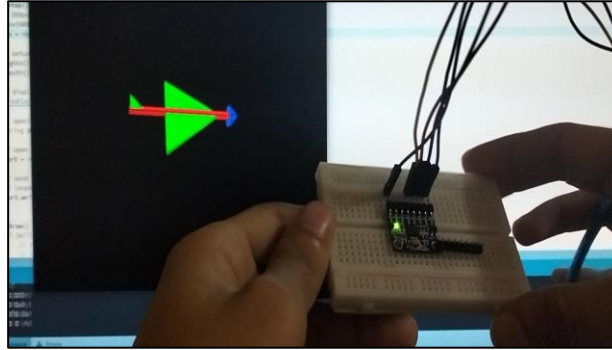
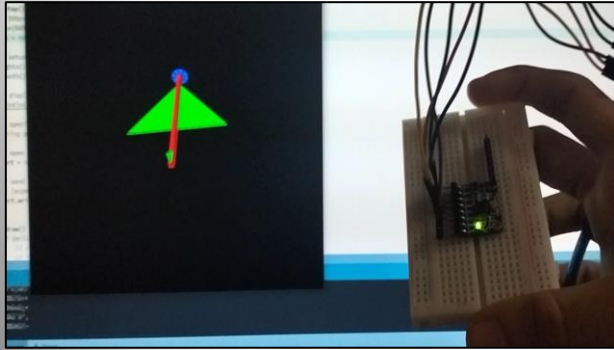
AVANTAGES :

- Réduire la charge de travail sur le microcontrôleur externe.
- Éviter le gimbal lock



02 Détecter les mouvements indésirables de la caméra

Simulation et conclusion



Le DMP assure une précision plus élevée que celle obtenue par la fusion des données via un filtre complémentaire.

<<Requirement>>

Détection de mouvement

Id= « 1.2 »

Text = « le support doit connaître à tous moments l'orientation de la caméra par rapport à un repère galiléen »



OBJECTIF 3

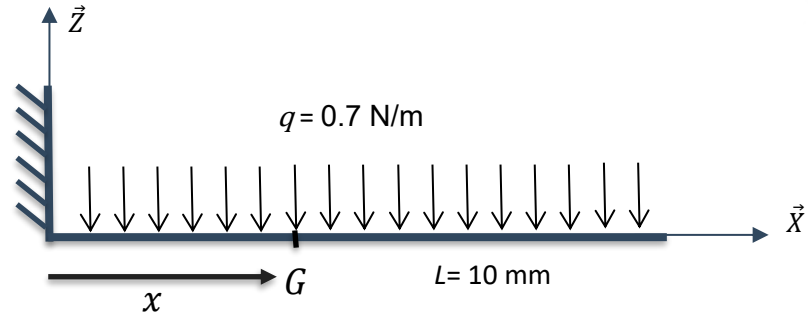
**Optimisation et choix du
Matériau de la plateforme
Porte-camera**

03 Optimisation et choix du matériau de la plateforme porte-camera

Modélisation de la sollicitation

HYPOTHÈSES :

- On suppose que le matériau de la poutre est continu.
- La surface à l'extrémité est encastree.
- On suppose que la masse de la camera est répartie uniformément sur la poutre



03 Optimisation et choix du matériau de la plateforme porte-camera

Torseur de cohésion

$$\{Coh\} = \left\{ \begin{array}{cc} 0 & 0 \\ 0 & Mf_y \\ T_z & 0 \end{array} \right\}_{(\vec{x}, \vec{y}, \vec{z})} = \left\{ \begin{array}{c} -q.(L-x) \vec{Z} \\ -q.\frac{(L-x)^2}{2} \vec{Y} \end{array} \right\}_{(\vec{x}, \vec{y}, \vec{z})}$$

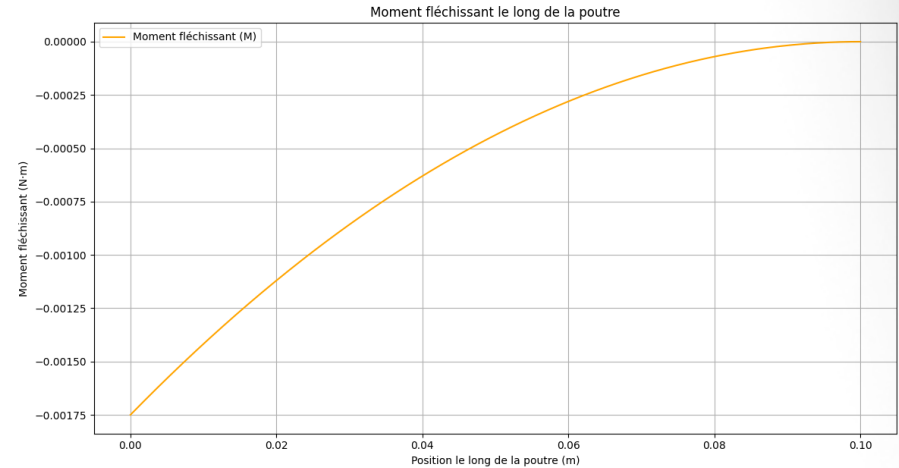
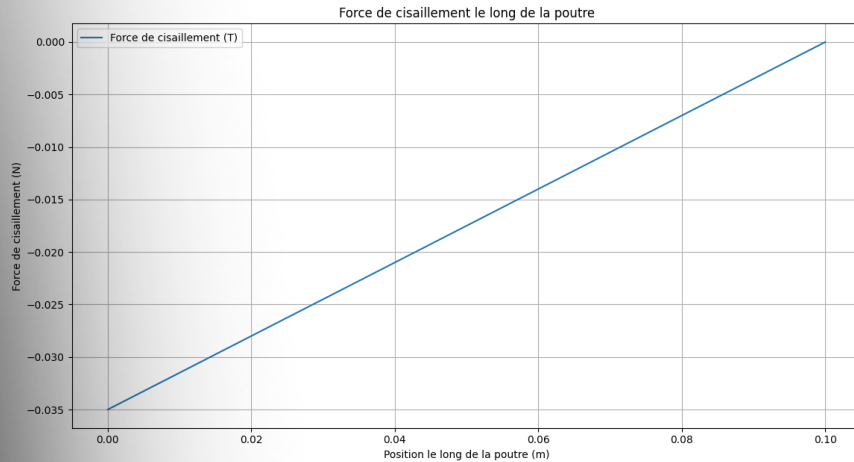







Figure 11: représentation graphique du torseur de cohésion

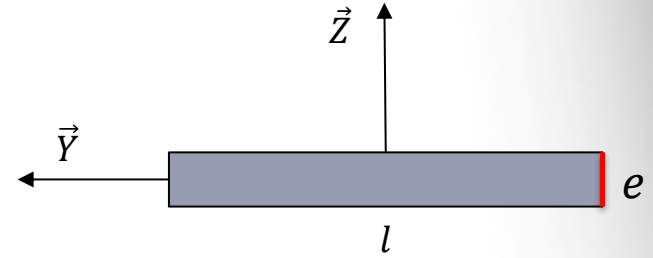
03 Optimisation et choix du matériau de la plateforme porte-camera

Démarche

	FONCTION	résister à un moment fléchissant dû à la charge répartie sur la poutre
	CONTRAINTES	la limite élastique doit être supérieure à 140 MPa + Durable dans l'eau
	OBJECTIF	minimiser la masse du support m
	VARIABLES LIBRES	l'épaisseur e
	VARIABLES IMPOSÉES	l'effort, la longueur L et la largeur l

03 Optimisation et choix du matériau de la plateforme porte-camera

Indice de performance



La masse : $m = \rho \cdot V = \rho \cdot L \cdot l \cdot e$

La contrainte maximale : $\sigma_{\max} = \frac{M_{f_{y_{\max}}}}{I_{G_y}} \cdot \gamma_{\max} = 3 \cdot q \cdot \frac{L^2}{l \cdot e^2} = \sigma_e$

(O) : Fonction objectif

(M) : paramètre propre
au matériau

$$m = \sqrt{3} \cdot \sqrt{q} \cdot \frac{\rho}{\sqrt{\sigma_e}} \cdot L^2 \sqrt{l}$$

(F) : Paramètre(fixe)

(G) : Caractéristique
géométrique

Diagramme d'ashby

Indice de performance : $I = \frac{\sigma_e^{\frac{1}{2}}}{\rho} \Rightarrow \log \sigma_e = 2 \log \rho + \log I$

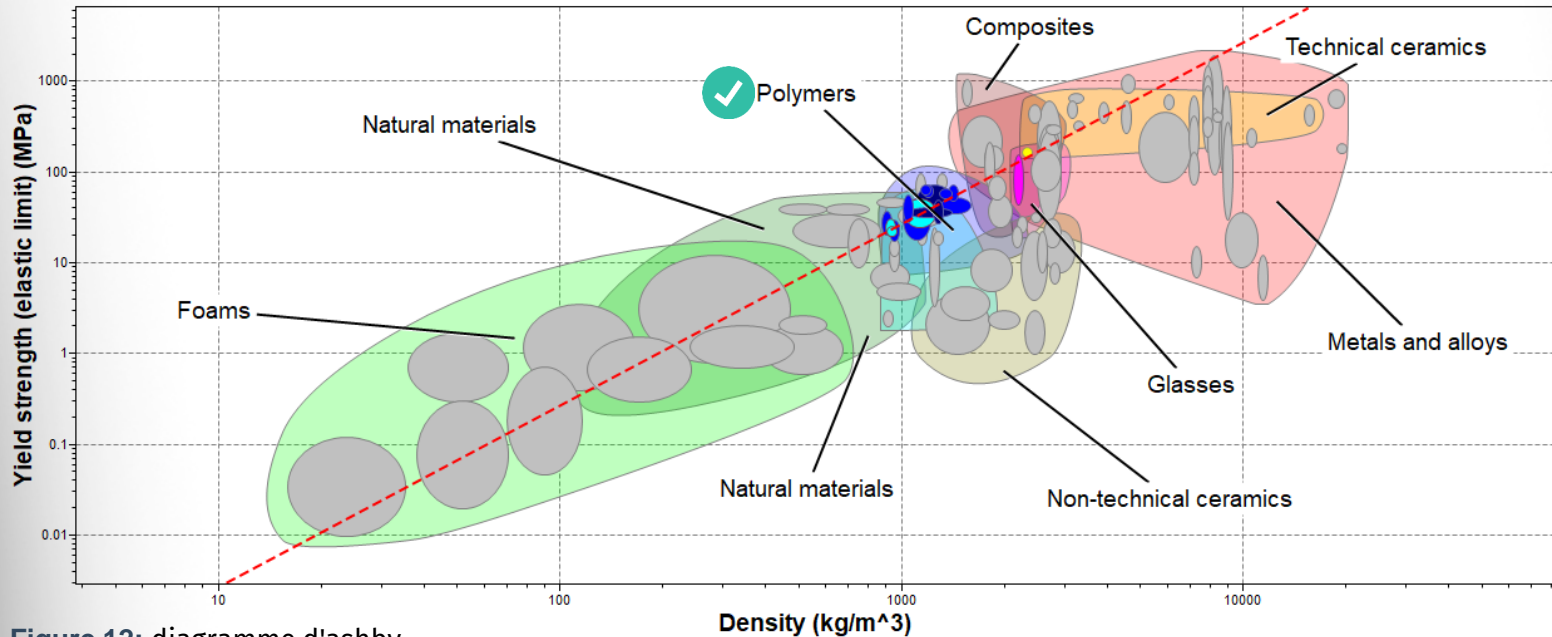


Figure 12: diagramme d'ashby

03 Optimisation et choix du matériau de la plateforme porte-camera

Choix du matériau

3. Résultats : 17 validées sur 100

Afficher : Fiches passant toutes les étapes

Classer par : Étape 1 : Yield strength (elastic limit) (MPa)

Nom	Yield strength (elast
Polyethylene (PE)	17.9 - 29
Acrylonitrile butadiene styrene (A...	18.5 - 51
Polyisoprene rubber (IIR)	20 - 25
Natural rubber (NR)	20 - 30
Polypropylene (PP)	20.7 - 37.2
Polyurethane	25 - 51
Phenolics (PH)	27.6 - 49.7
Polystyrene (PS)	28.7 - 56.2
Polyester (UP)	33 - 40
Polyvinylchloride (tpPVC)	35.4 - 52.1
Epoxies (EP)	36 - 71.7
Silica glass	45 - 155
Polyoxymethylene (Acetal, POM)	48.6 - 72.4
Polymethyl methacrylate (Acrylic, ...	53.8 - 72.4
Polyethylene terephthalate (PET)	56.5 - 62.3
Polycarbonate (PC)	59 - 70
Silicon	160 - 180


Polycarbonate (PC)

Disposition : All properties Afficher/Masquer

Polymers and elastomers > Polymers > Thermoplastics >

Description

Image



Caption

1. Personal computer casing made of polycarbonate. © Chris Lefteri 2. Polycarbonate is tough and impact-resistant: hence its use in hard hats and helmets, transparent roofing and riot shields.

Le polycarbonate, reconnu pour sa résistance aux chocs, est utilisé dans diverses applications telles que les casques de sécurité, les toitures transparentes et les boucliers anti-émeutes.

Détermination de l'épaisseur minimale

Condition de resistance : $\sigma_{\max} = \frac{M_{f_{y_max}}}{I_{G_y}} \cdot \gamma_{\max} \leq \frac{\sigma_e}{s}$

Polycarbonate (PC) 59 - 70

$$\Rightarrow e \geq \left(3 \cdot q \cdot \frac{s \cdot L^2}{l \cdot \sigma_e} \right)^{1/2}$$

Données :

$$q = 0.7 \text{ N/m}$$

$$L = 10 \text{ cm}$$

$$l = 3.6 \text{ cm}$$

$$s = 3$$

$$\Rightarrow e \geq 0.1722 \text{ mm}$$

$$e = 1 \text{ mm}$$

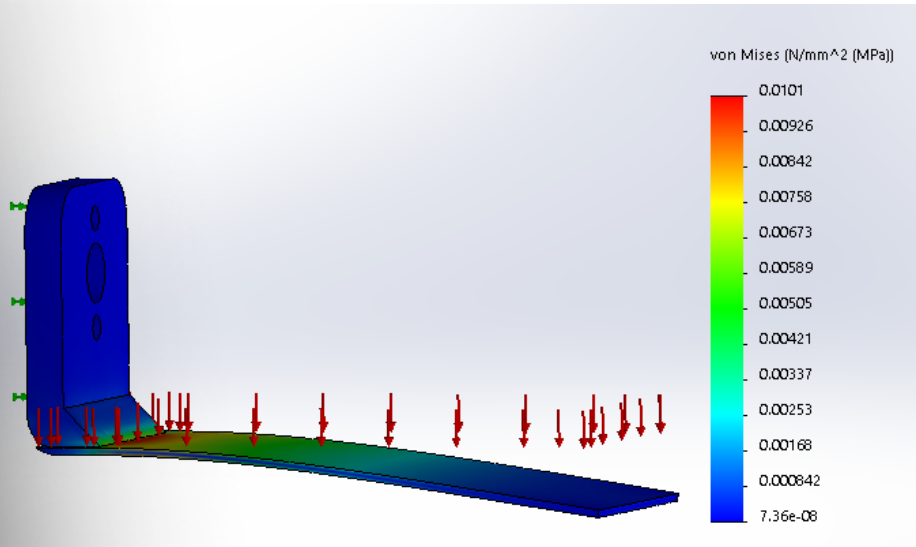


03 Optimisation et choix du matériau de la plateforme porte-camera

Simulation et conclusion

<<Requirement>> Matériau du support	
Id= « 1.5 »	✓
Text = « Le support doit être capable de résister à la flexion et supporter le poids de la caméra »	

Figure 13: simulation Xpress (SW)



Le polycarbonate :

- résiste au moment fléchissant + résiste aux chocs
- Une structure légère permet des mouvements plus fluides
- résiste aux variations de température + résiste à l'eau
- offre un excellent rapport qualité-prix

OBJECTIF 3

Réalisation d'un prototype

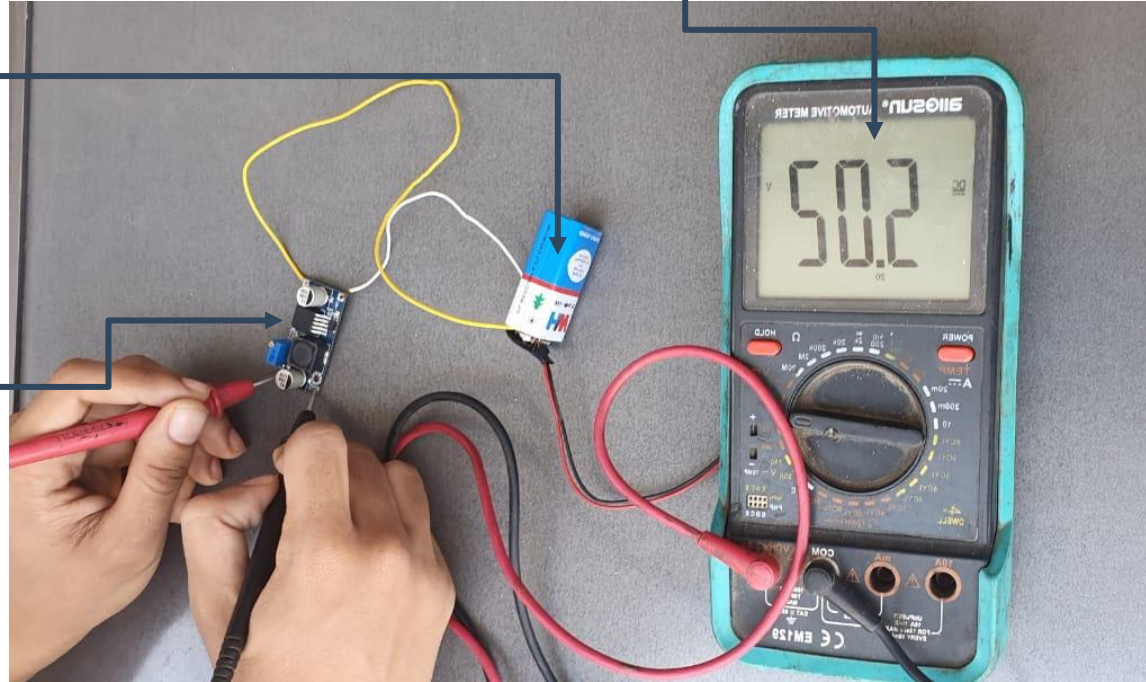
04 Réalisation d'un prototype

Regulation de tension



Batterie (9V)

Hacheur LM2596



04 Réalisation d'un prototype

Cablage

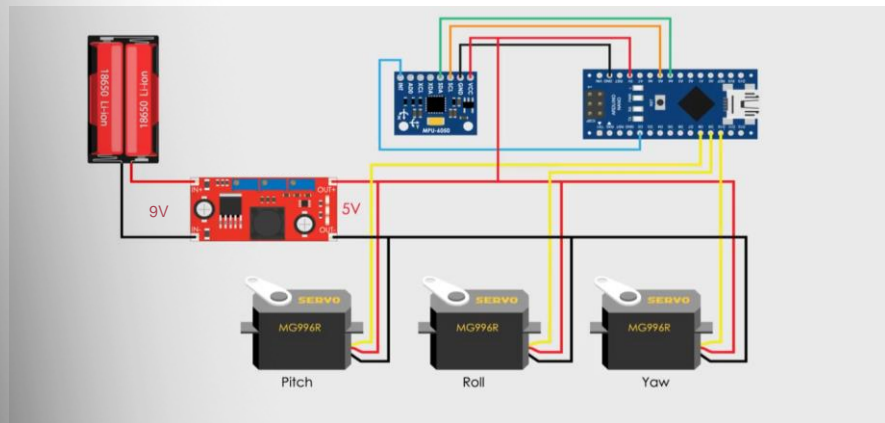
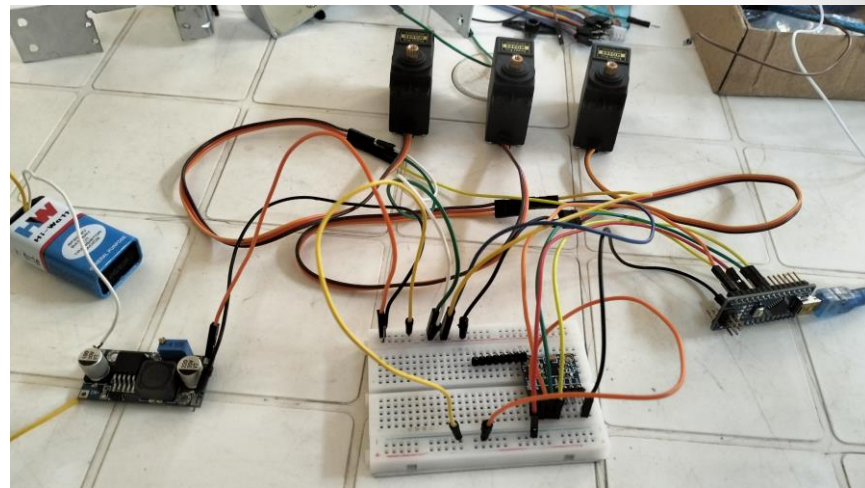


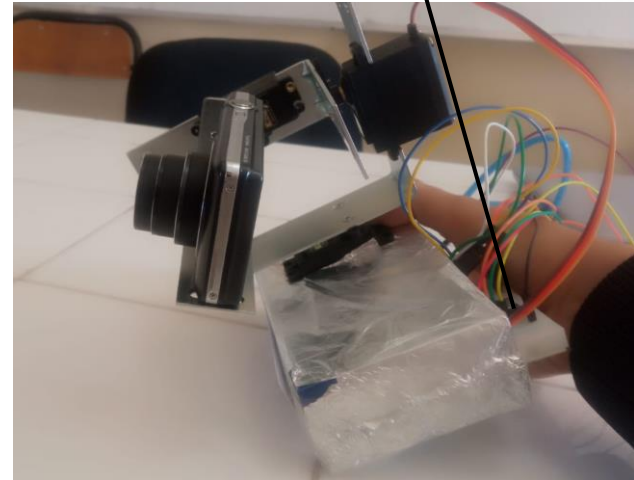
Figure 14: circuit du Gimbal



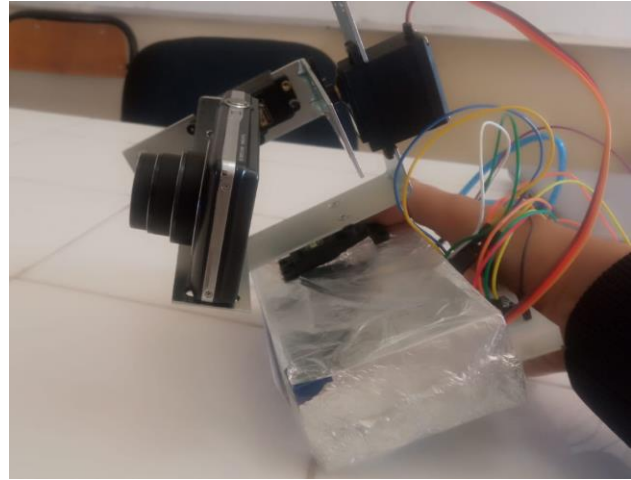
04 Réalisation d'un prototype

Prototype

MPU-6050



CONCLUSION



Annexe

Calibrage du capteur

```
void calculate_IMU_error() {  
  
    // Read accelerometer values 200 times  
    while (c < 200) {  
        Wire.beginTransmission(MPU);  
        Wire.write(0x3B);  
        Wire.endTransmission(false);  
        Wire.requestFrom(MPU, 6, true);  
  
        AccX = (Wire.read() << 8 | Wire.read()) / 16384.0;  
        AccY = (Wire.read() << 8 | Wire.read()) / 16384.0;  
        AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0;  
  
        // Sum all readings  
        AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI));  
        c++;  
    }  
  
    // Divide the sum by 200 to get the error value  
    AccErrorX = AccErrorX / 200;  
    c = 0;  
  
    // Read gyro values 200 times  
    while (c < 200) {  
        Wire.beginTransmission(MPU);  
        Wire.write(0x43);  
        Wire.endTransmission(false);  
        Wire.requestFrom(MPU, 6, true);  
  
        GyroX = Wire.read() << 8 | Wire.read();  
  
        // Sum all readings  
        GyroErrorX = GyroErrorX + (GyroX / 131.0);  
        c++;  
    }  
  
    // Divide the sum by 200 to get the error value  
    GyroErrorX = GyroErrorX / 200;  
  
    // Print the error values on the Serial Monitor  
    Serial.print("AccErrorX: ");  
    Serial.println(AccErrorX);  
    Serial.print("GyroErrorX: ");  
    Serial.println(GyroErrorX);  
}
```



Arduino IDE

Annexe Filtrage (passe bas accéléromètre)



Arduino IDE

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

// Filtre passe-bas
float accX, accY, accZ;
float accAngleX, accAngleY;
float accAngleX_f, accAngleY_f;
float alpha = 0.5; // Coefficient du filtre passe-bas

unsigned long lastTime, currentTime;
float elapsedTime;

void setup() {
  Serial.begin(115200);
  Wire.begin();
  mpu.initialize();
  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed");
    while (1);
  }

  // Calibration du gyroscope
  lastTime = millis();
}

void loop() {
  currentTime = millis();
  elapsedTime = (currentTime - lastTime) / 1000.0;
  lastTime = currentTime;

  // Lire les valeurs du MPU6050
  mpu.getAcceleration(&accX, &accY, &accZ);

  // Calcul des angles avec l'accéléromètre
  accAngleX = atan(accY / sqrt(pow(accX, 2) + pow(accZ, 2))) * 180 / PI;
  accAngleY = atan(-accX / sqrt(pow(accY, 2) + pow(accZ, 2))) * 180 / PI;

  // Appliquer le filtre passe-bas
  accAngleX_f = alpha * accAngleX_f + (1 - alpha) * accAngleX;
  accAngleY_f = alpha * accAngleY_f + (1 - alpha) * accAngleY;

  // Afficher les résultats
  Serial.print("AccAngleX (Filtre passe-bas): "); Serial.print(accAngleX_f); Serial.print(" ");
  Serial.print("AccAngleY (Filtre passe-bas): "); Serial.println(accAngleY_f);

  delay(10);
}
```

Annexe

Filtrage (par fusion)

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

```
const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;
```

```
void setup() {
  Serial.begin(19200);
  Wire.begin(); // Initialize communication
  Wire.beginTransmission(MPU); // Start communication with MPU6050
  Wire.write(0x6B); // Talk to the register 6B
  Wire.write(0x00); // Make reset - place a 0 into the 6B register
  Wire.endTransmission(true); // End the transmission

  // Call this function if you need to get the IMU error values for your module
  calculate_IMU_error();
  delay(20);
}
```

```
void loop() {
  // == Read accelerometer data == //
  Wire.beginTransmission(MPU);
  Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
```

```
AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
```

```
// Calculating Roll and Pitch from the accelerometer data
accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) + 0.95; // AccErrorX
accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.38; // AccErrorY
```

```
// == Read gyroscope data == //
previousTime = currentTime; // Previous time is stored before the actual time read
currentTime = millis(); // Current time actual time read
elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to get seconds
```

```
Wire.beginTransmission(MPU);
Wire.write(0x43); // Gyro data first register address 0x43
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2 registers
```

```
GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divide first the raw va
GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
```

```
// Correct the outputs with the calculated error values
GyroX = GyroX - 1.38; // GyroErrorX
GyroY = GyroY - 1.49; // GyroErrorY
GyroZ = GyroZ + 0.43; // GyroErrorZ
```

```
// Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by seconds (s) to get th
gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
gyroAngleY = gyroAngleY + GyroY * elapsedTime;
yaw = yaw + GyroZ * elapsedTime;
```

```
// Complementary filter - combine accelerometer and gyro angle values
gyroAngleX = 0.96 * gyroAngleX + 0.04 * accAngleX;
gyroAngleY = 0.96 * gyroAngleY + 0.04 * accAngleY;
```

```
roll = gyroAngleX;
pitch = gyroAngleY;
```

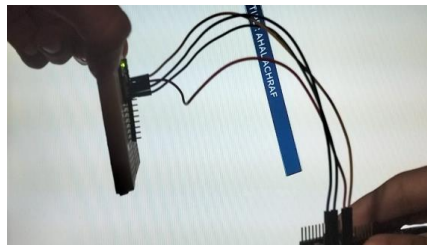
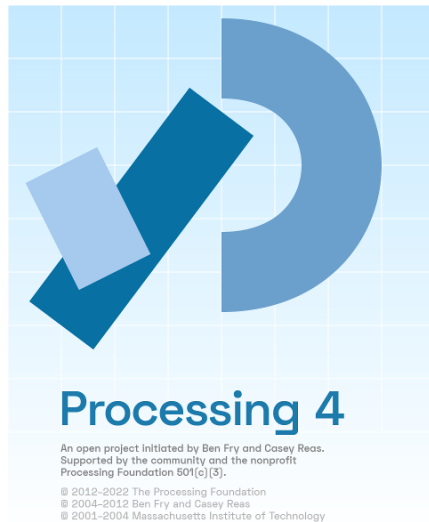
```
// Print the values on the serial monitor
Serial.print(roll);
Serial.print("/");
Serial.print(pitch);
Serial.print("/");
Serial.println(yaw);
}
```



Arduino IDE

Annexe Filtrage (par fusion)

```
1 import processing.serial.*;
2
3 Serial myPort;
4 String data = "";
5 float roll, pitch, yaw;
6
7 void setup() {
8   size(1600, 900, P3D);
9   myPort = new Serial(this, "COM5", 19200);
10  myPort.bufferUntil('\n');
11 }
12
13 void draw() {
14   frameRate(30); // Limit frame rate to 30 fps
15   translate(width/2, height/2, 0);
16   background(233);
17   textSize(22);
18   text("Roll: " + int(roll) + "      Pitch: " + int(pitch), -100, 265);
19   rotateX(radians(-pitch));
20   rotateZ(radians(-roll));
21   rotateY(radians(yaw));
22   textSize(30);
23   fill(0, 76, 153);
24   box(386, 40, 200);
25   textSize(25);
26   fill(255, 255, 255);
27   text("TIPE : AHAL ACHRAF", -183, 10, 101);
28 }
29
30 void serialEvent (Serial myPort) {
31   data = myPort.readStringUntil('\n');
32   if (data != null) {
33     data = trim(data);
34     String items[] = split(data, '/');
35     if (items.length > 1) {
36       roll = float(items[0]);
37       pitch = float(items[1]);
38       yaw = float(items[2]);
39     }
40   }
41 }
```



Annexe Tracage du torseur de cohésion

```
import numpy as np
import matplotlib.pyplot as plt

# Paramètres
L = 0.1 # Longueur de la poutre en mètres
w = 0.35 # Charge répartie en N/m

# Discrétisation de la poutre
x = np.linspace(0, L, 500)

# Calcul de la force de cisaillement (T)
T = - w * (L - x)

# Calcul du moment fléchissant (M)
M = - (w/2) * (L - x)**2

# Tracer la courbe de la force de cisaillement (T)
plt.figure()
plt.plot(x, T, label='Force de cisaillement (T)')
plt.xlabel('Position le long de la poutre (m)')
plt.ylabel('Force de cisaillement (N)')
plt.title('Force de cisaillement le long de la poutre')
plt.grid(True)
plt.legend()
plt.show()

# Tracer la courbe du moment fléchissant (M)
plt.figure()
plt.plot(x, M, label='Moment fléchissant (M)', color='orange')
plt.xlabel('Position le long de la poutre (m)')
plt.ylabel('Moment fléchissant (N·m)')
plt.title('Moment fléchissant le long de la poutre')
plt.grid(True)
plt.legend()
plt.show()
```

