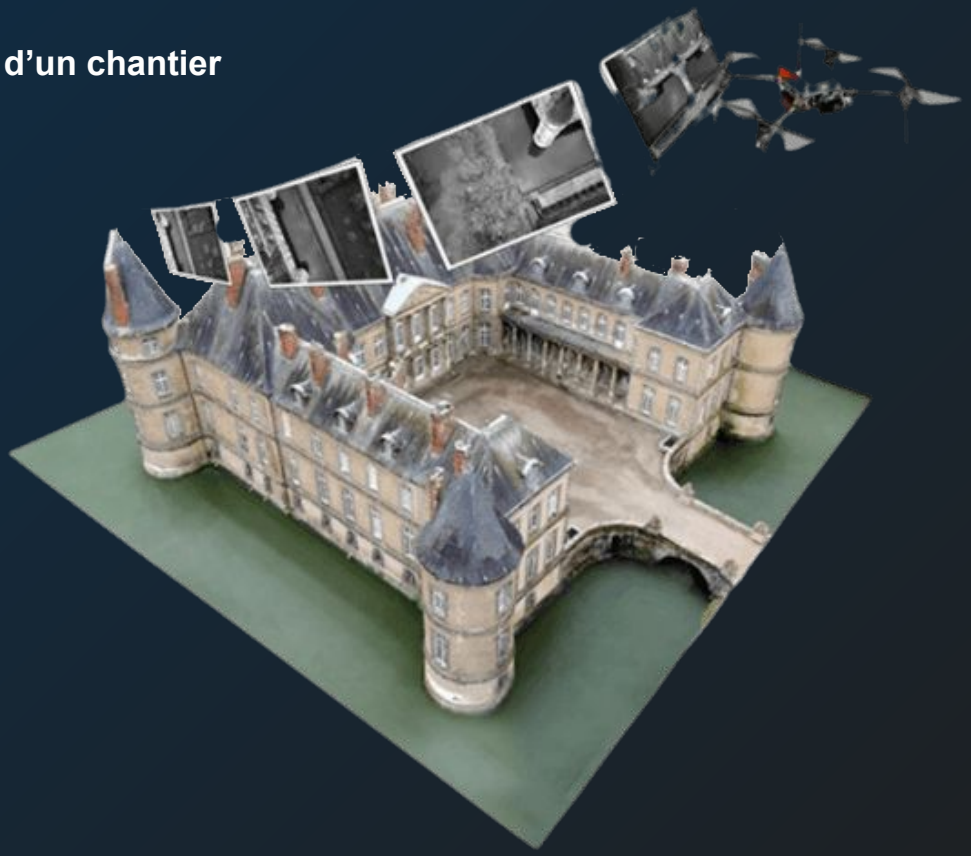
Asservissement d’un drone

Peut-on se passer d’une centrale inertielle ?

Introduction

La photogrammétrie pour évaluer l'avancement d'un chantier nécessite

- **Haute résolution d'Image**
- **Netteté des images**
  - **Stabilité de la nacelle**
- **Chevauchement des images**
  - **Fréquence d'acquisition**



Introduction

La photogrammétrie pour évaluer l'avancement d'un chantier nécessite

Haute résolution d'Image

Netteté des images

Stabilité de la nacelle

Chevauchement des images

Fréquence d'acquisition

Asservissement du drone sur ses trois axes à partir de la donnée angulaire

Acquisition via une centrale inertielle

Ajustement de la puissance des moteurs

Introduction

La photogrammétrie pour évaluer l'avancement d'un chantier nécessite

Haute résolution d'Image

Netteté des images

Stabilité de la nacelle

Chevauchement des images

Fréquence d'acquisition

Asservissement du drone sur ses trois axes à partir de la donnée angulaire

Détermination via analyse d'images

Ajustement de la puissance des moteurs

Introduction

Détection d’angle dans une image

- Approche naïve et itérations
- Difficultés et Limitations
- Implémentation en Python

Asservissement du drone

- Modélisation Physique
- Analyse du correcteur PID
- Implémentation Arduino

Conclusions

Détection d'Angle dans une Image

Corrélation d'Image

Image maître

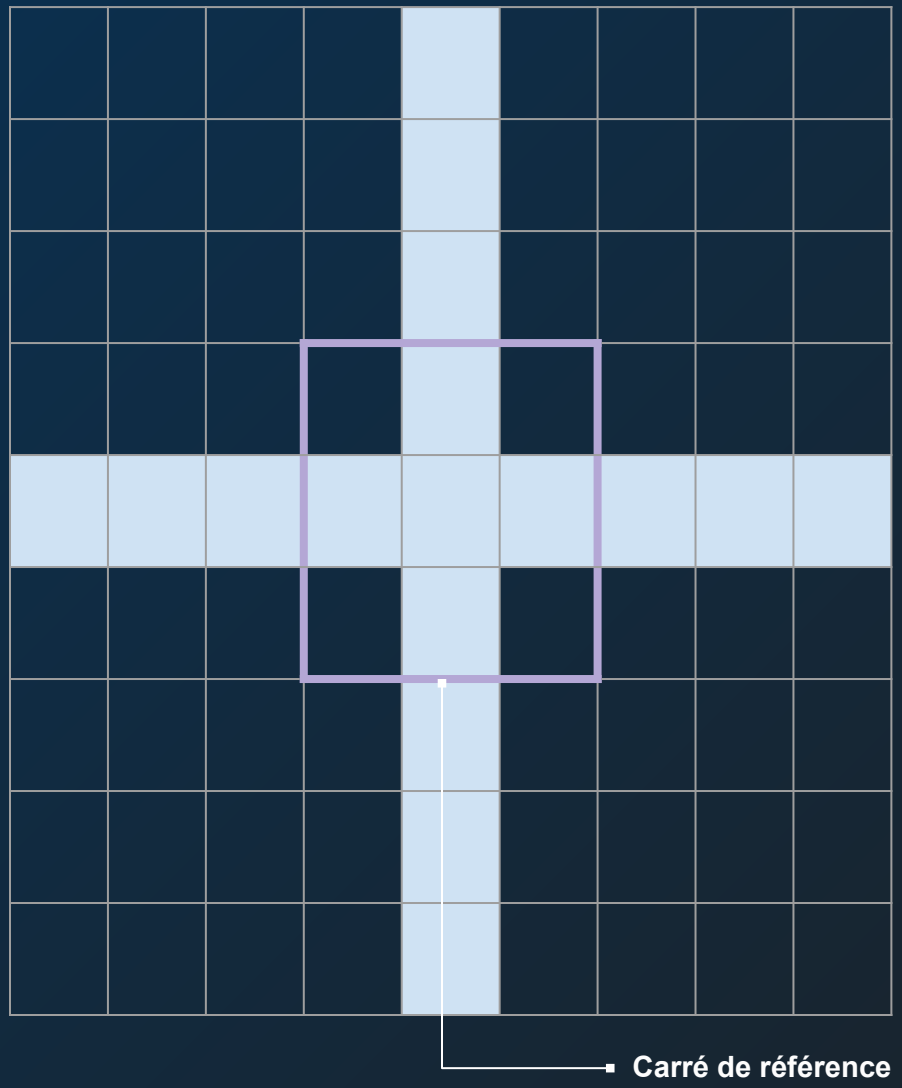
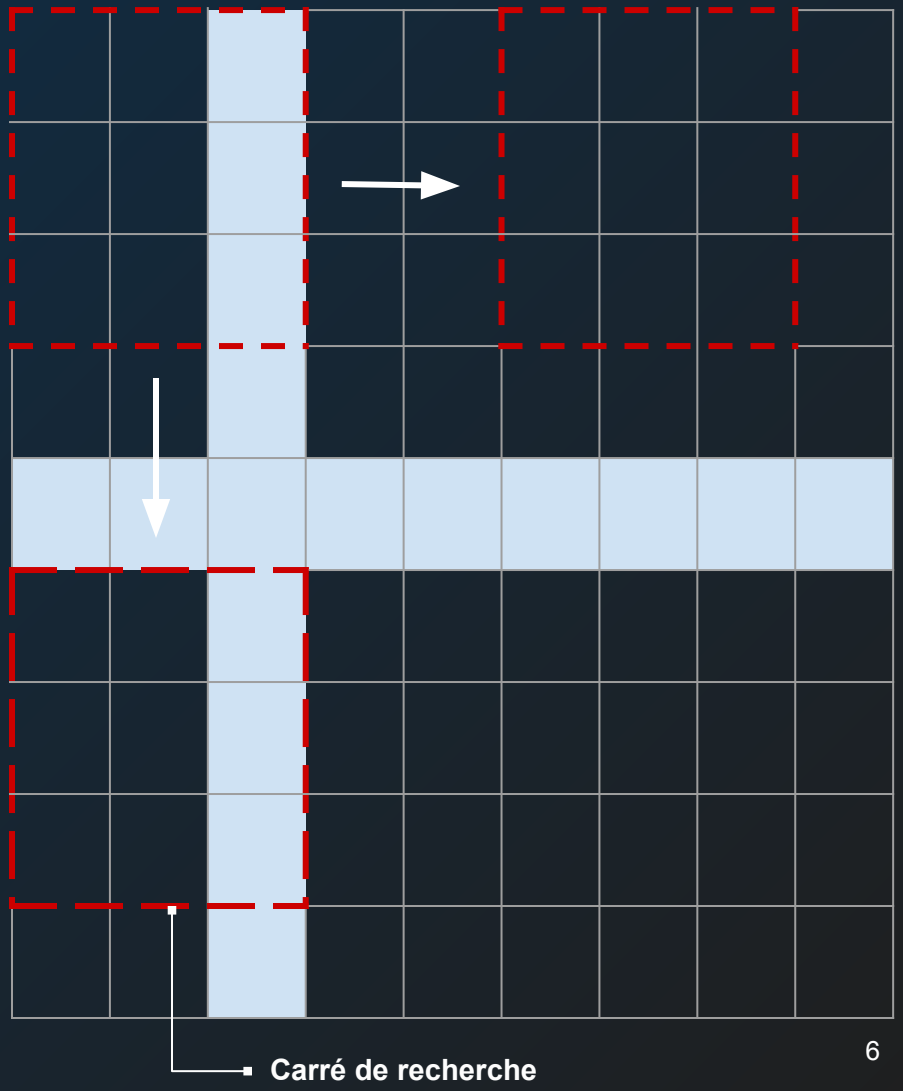


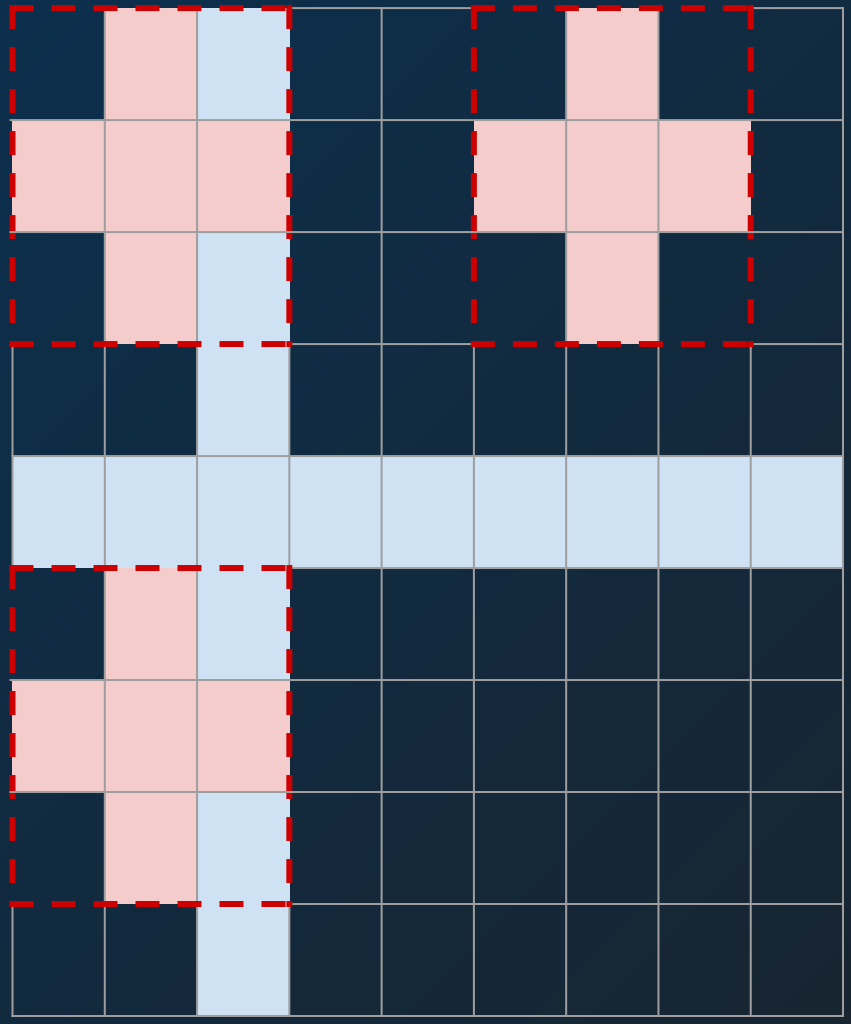
Image esclave



Détection d'Angle dans une Image

Corrélation d'Image

Calcul de corrélation

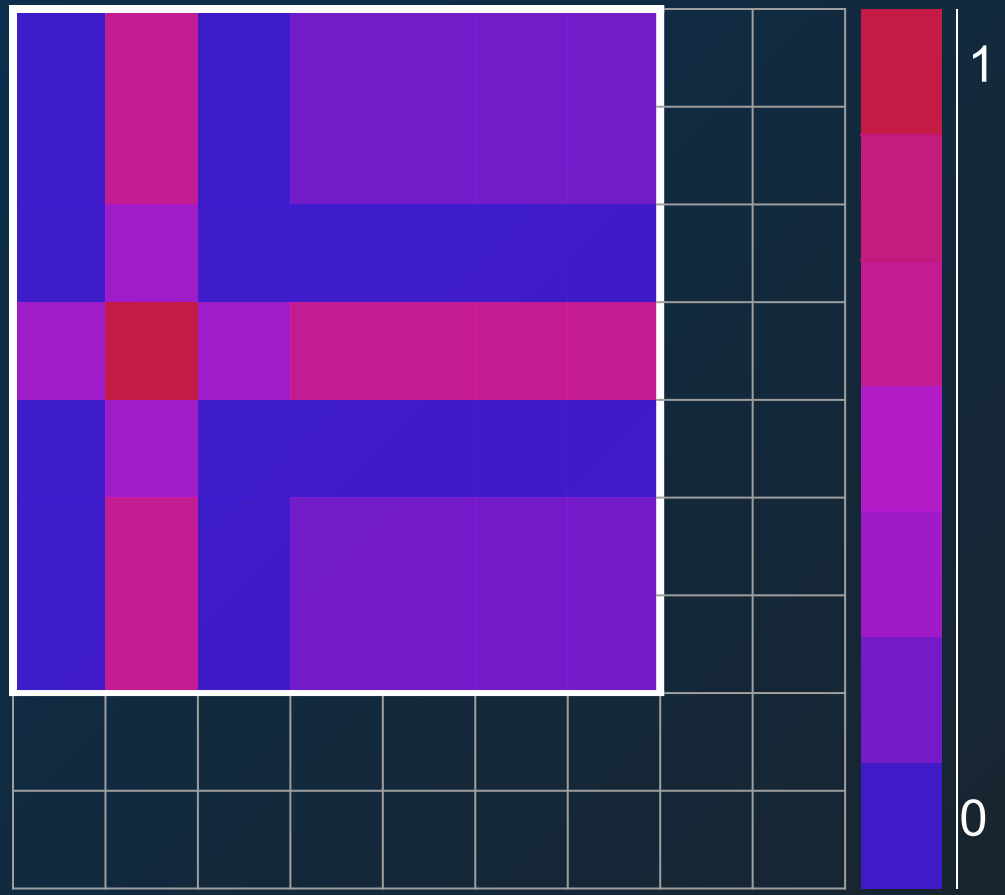


Matrice de corrélation

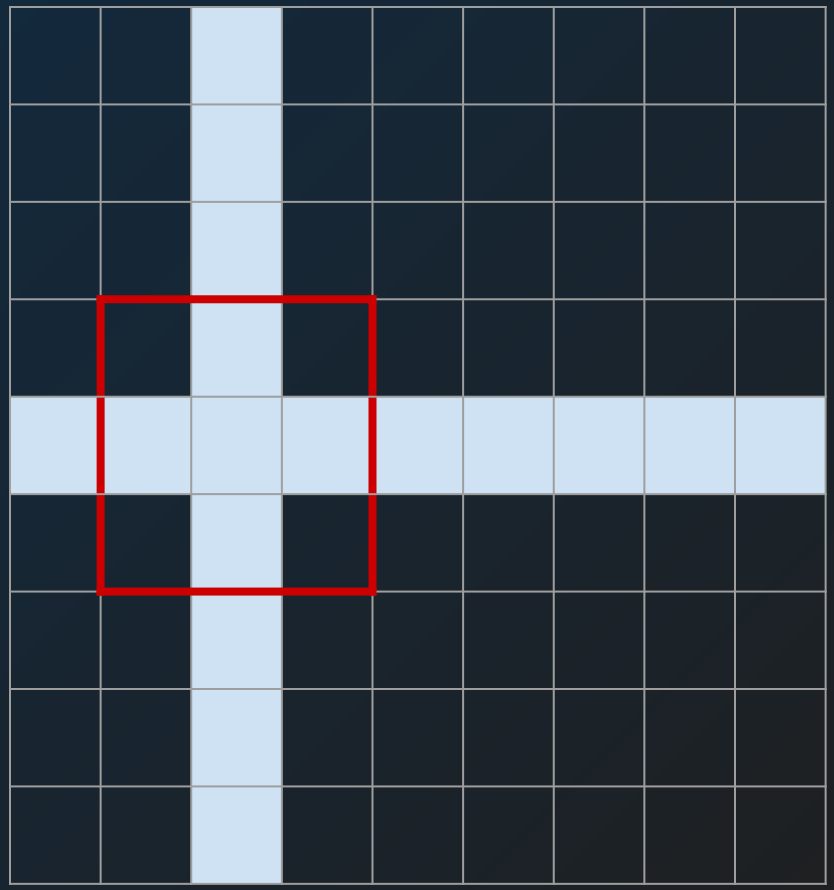
Détection d'Angle dans une Image

Corrélation d'Image

Matrice de corrélation

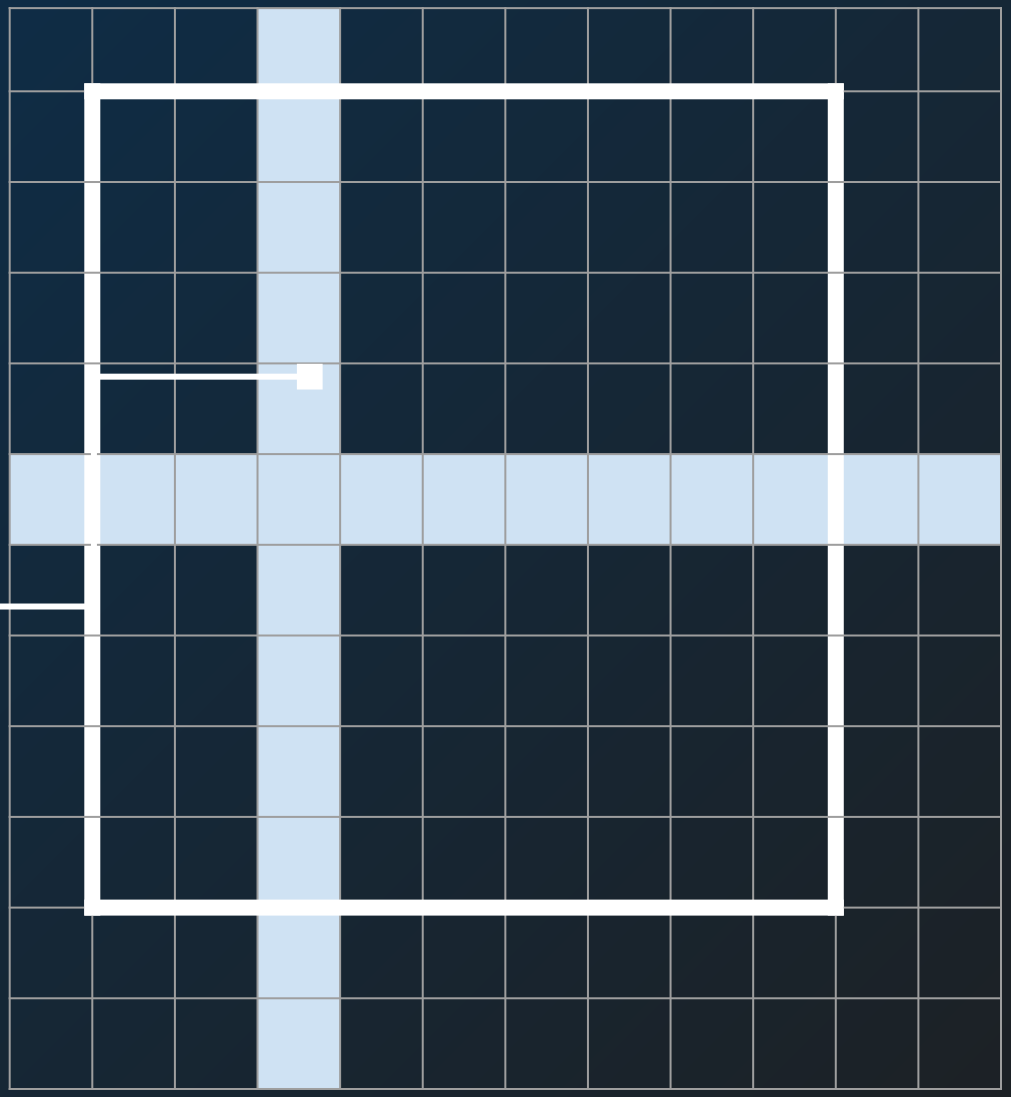


Corrélation Trouvée



Détection d'Angle dans une Image

Corrélation d'Image



Zone de recherche

Détection d'Angle dans une Image

Corrélation d'image - Implémentation

Préparer les Images :

- Conversion en niveaux de gris
- Pivoter l'image d'un angle connu

Définir les différents paramètres :

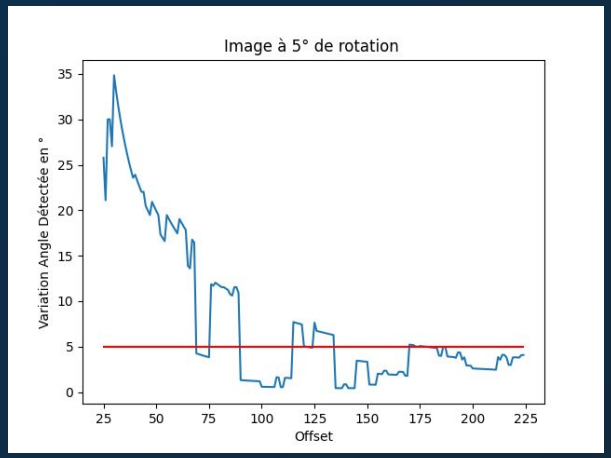
- Dimensions du carré de référence
- Position du carré de référence
- Dimension de la zone de recherche

Rotation de 5°

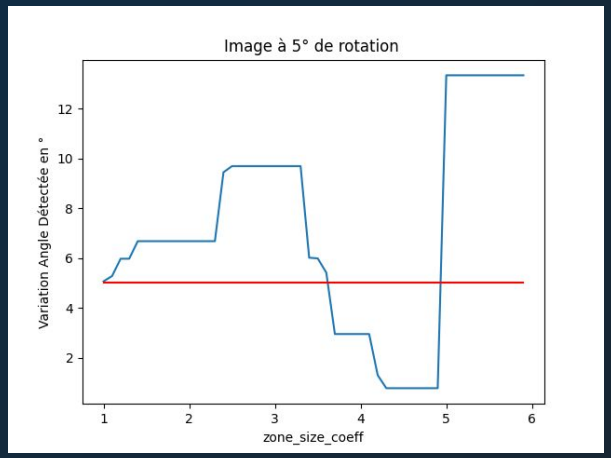
Détection d'Angle dans une Image

Optimisation des paramètres (img 4K)

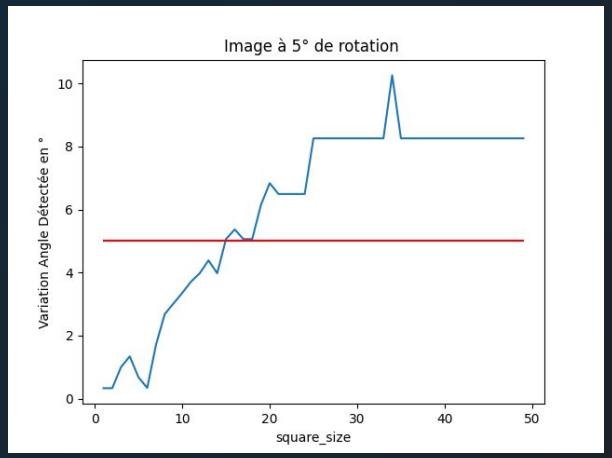
Position du carré de référence (CR)



Dimensions de la zone de recherche (ZR)



Dimensions du carré de référence (CR)



Analyse

- CR éloignée du centre
- ZR 5x plus grande que le CR
- CR de 15px

Note

Malgré l'optimisation des paramètre l'estimation reste toutefois aléatoire

Détection d'Angle dans une Image

Cas particulier du ciel

Le ciel est une zone, récurrente, éloignée du centre *mais* uniforme…

Problème identifié :

Le CR doit être unique dans la ZR

Solution envisagée :

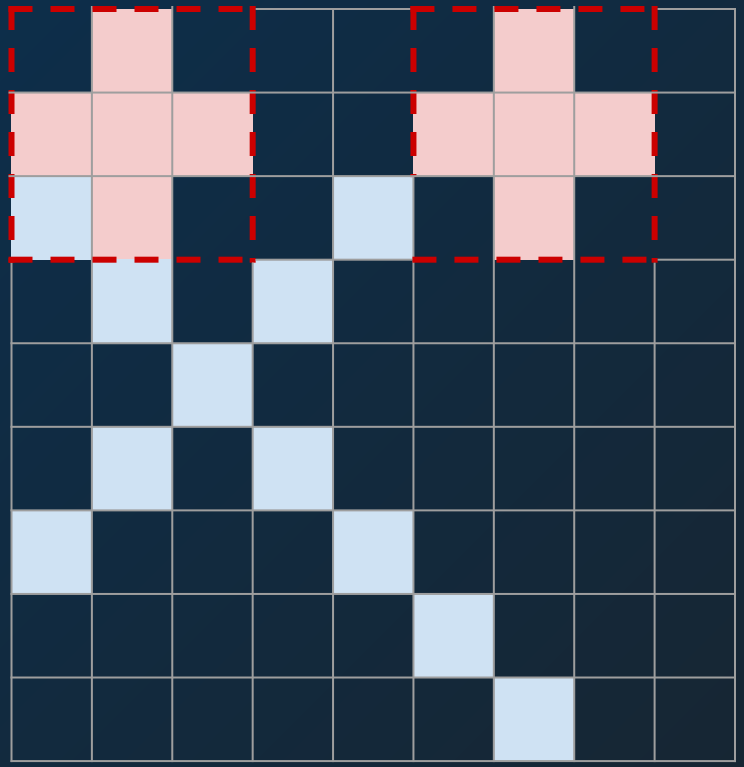
Sélection le CR dans une zone où la variance est la plus élevée

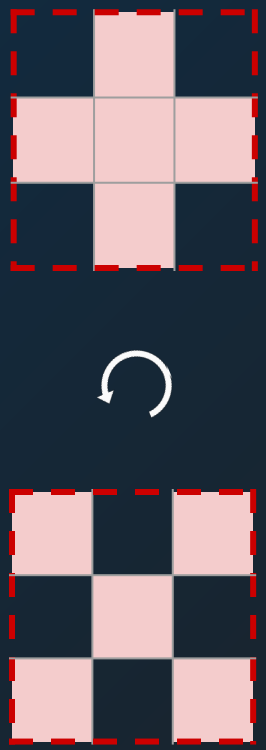
Détection d'Angle dans une Image

Une rotation… ça tourne !

Calcul de corrélation après rotation

Après rotation du CR

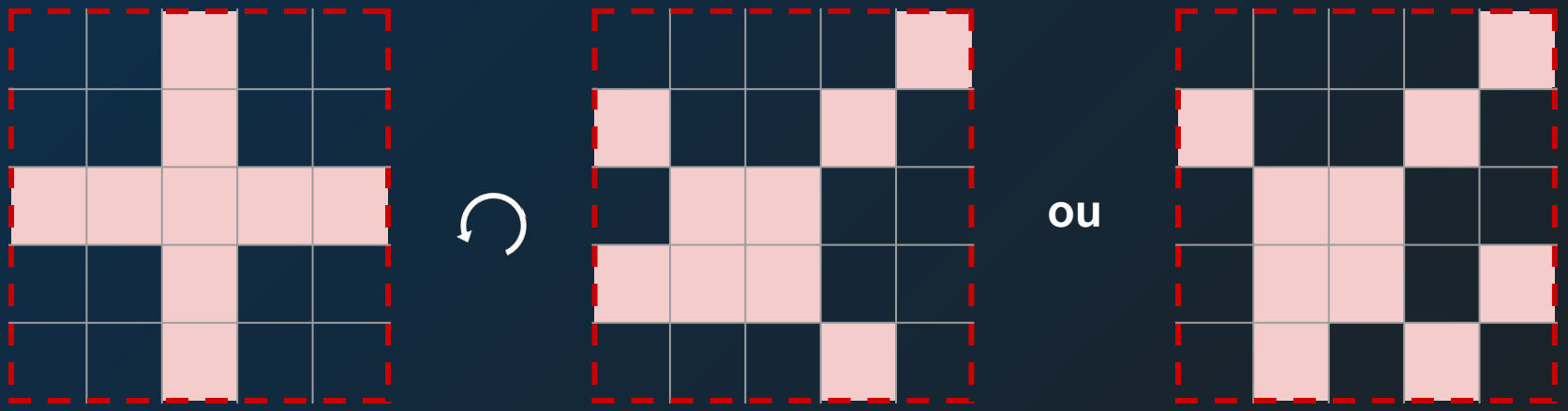




Détection d’Angle dans une Image

INFORMATIQUE CCP 2023

Une rotation du fait de l'échantillonnage déforme le CR



Note

Cr = Zone Haute Variance donc pas de régression linéaire possible

Détection d’Angle dans une Image

Implémentation

Convertir en Niveau de gris

Zone de Haute Variance

Pivoter l’image esclave de différents angles

Retenir l’angle donnant l’image la plus ressemblante

```
def calc_angle_diff(image1, image2, CR_size,
precision):
    img1 = np.array(image1.convert("L"))
    img2 = np.array(image2.convert("L"))

    # Coordonnées de la zone de variance max
    mvi, mvj = max_var_zone(img1, CR_size)

    max_corr = 0.0
    b_angle = 0.0

    # Recherche de l'angle donnant la meilleure
corrélation
    for angle in np.arange(0, 6, precision):
        rotated_img2 = ndimage.rotate(img2, angle)

        # Calcul du score de corrélation
        corr = np.corrcoef(img1[mvi:mvi+CR_size,
mvj:mvj+CR_size], rotated_img2[mvi:mvi+CR_size,
mvj:mvj+CR_size]).sum()

        if corr > max_corr:
            max_corr = corr
            b_angle = angle

    return b_angle
```

Détection d'Angle dans une Image

Implémentation

Convertir en Niveau de gris

ITU-R 601-2 luma transform :

$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \text{(Luminance)}$$
$$U' = (B' - Y') \cdot 0.565 \text{(Chrominance)}$$
$$V' = (R' - Y') \cdot 0.713 \text{(Chrominance)}$$

Zone de Haute Variance

$$variance = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^2}{N}$$

Pivoter l'image esclave de différents angles

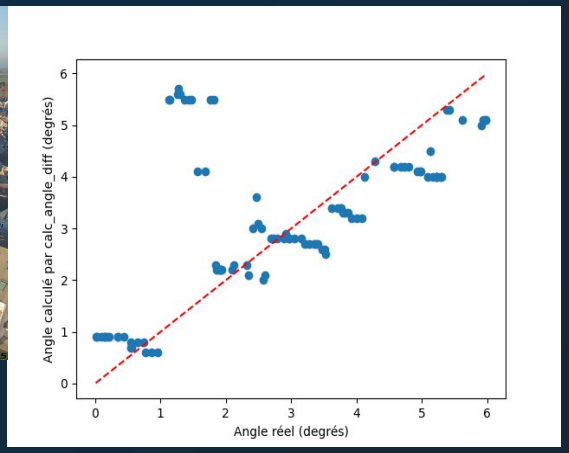
$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

Retenir l'angle donnant l'image la plus ressemblante

$$\rho_{XY} = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2 \cdot \sum_{i=1}^{n} (Y_i - \bar{Y})^2}}$$

Détection d’Angle dans une Image

Résultats



Ecart Moyen - 0.91°

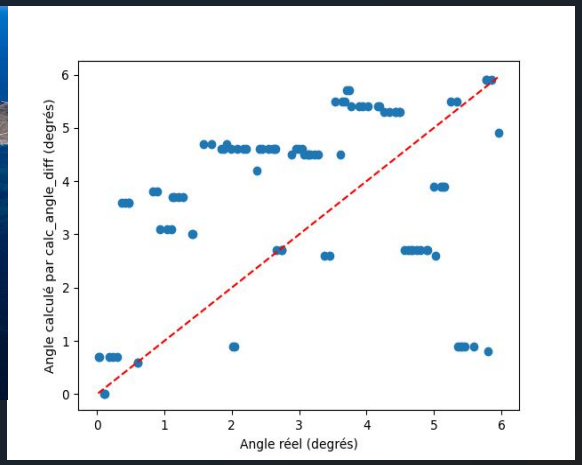
Nombre de points - 100

Dimensions de l’image - 960×539

Ecart Moyen - 1.71°

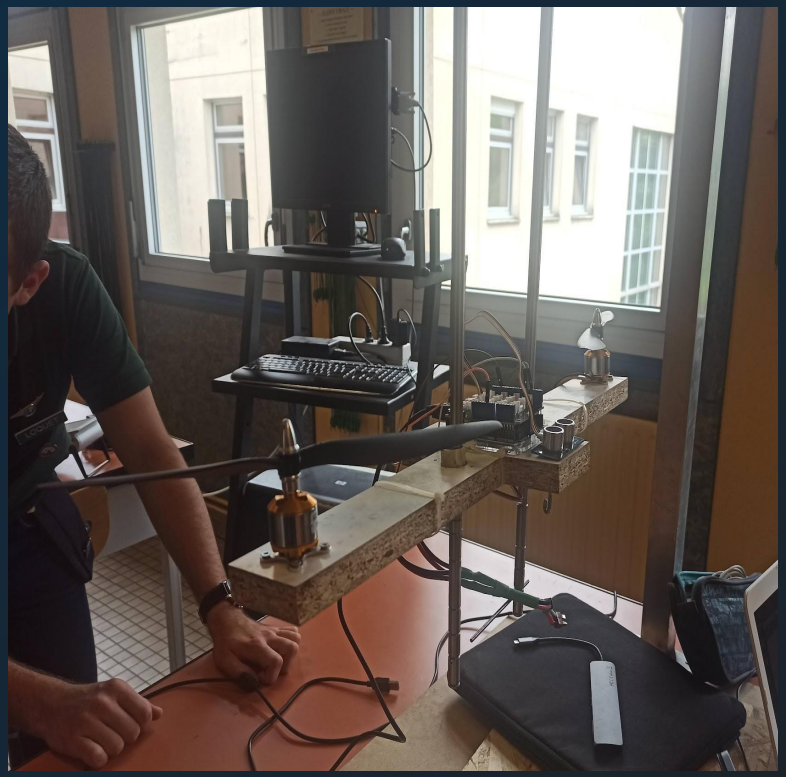
Nombre de points - 100

Dimensions de l’image - 2560×1705



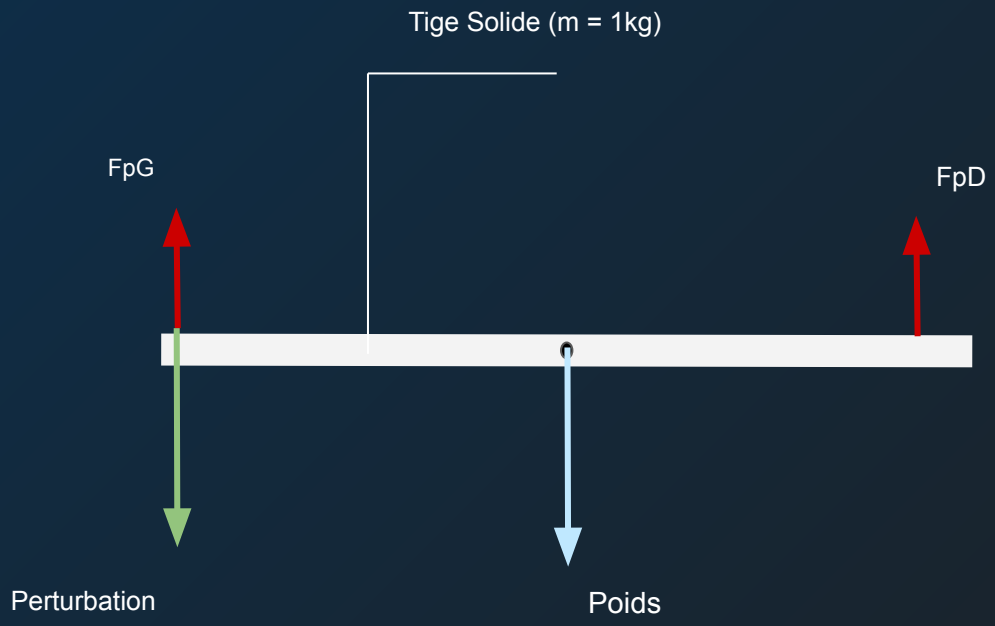
Asservissement du drone

Modélisation Physique



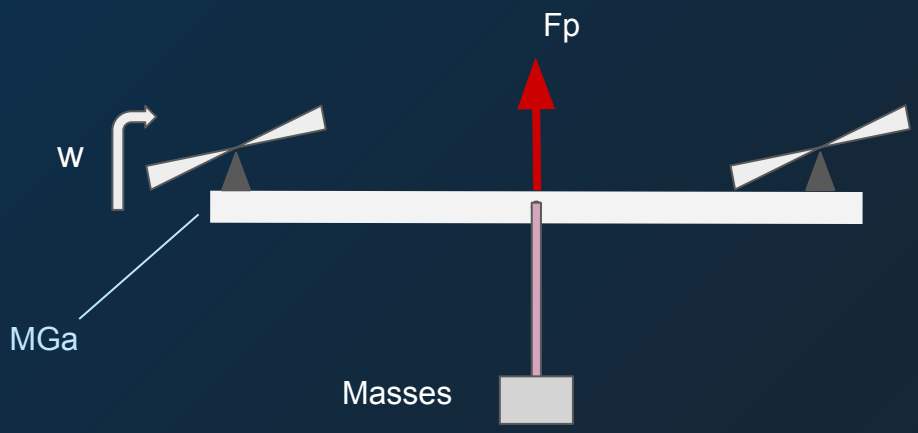
BILAN DES FORCES

Forces de poussée
Poids
Perturbation

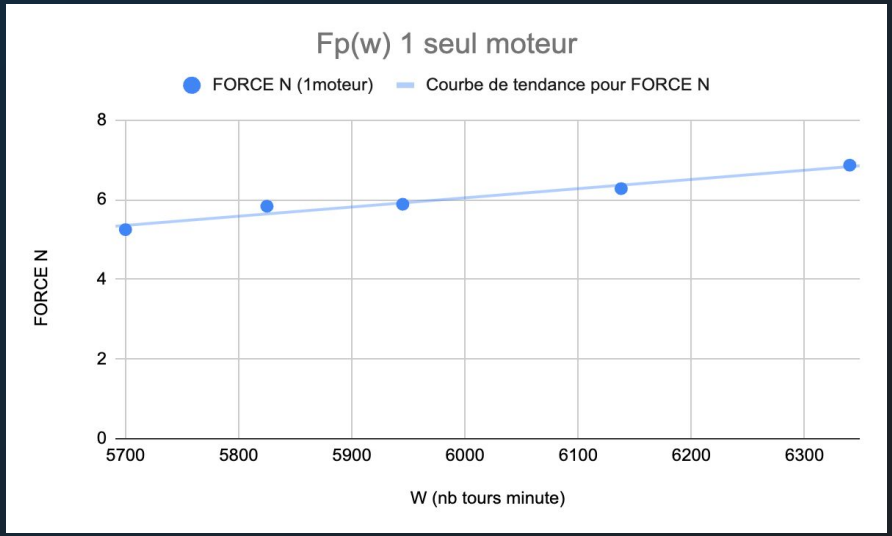


Asservissement du drone

Détermination Expérimentale des Forces de Poussée



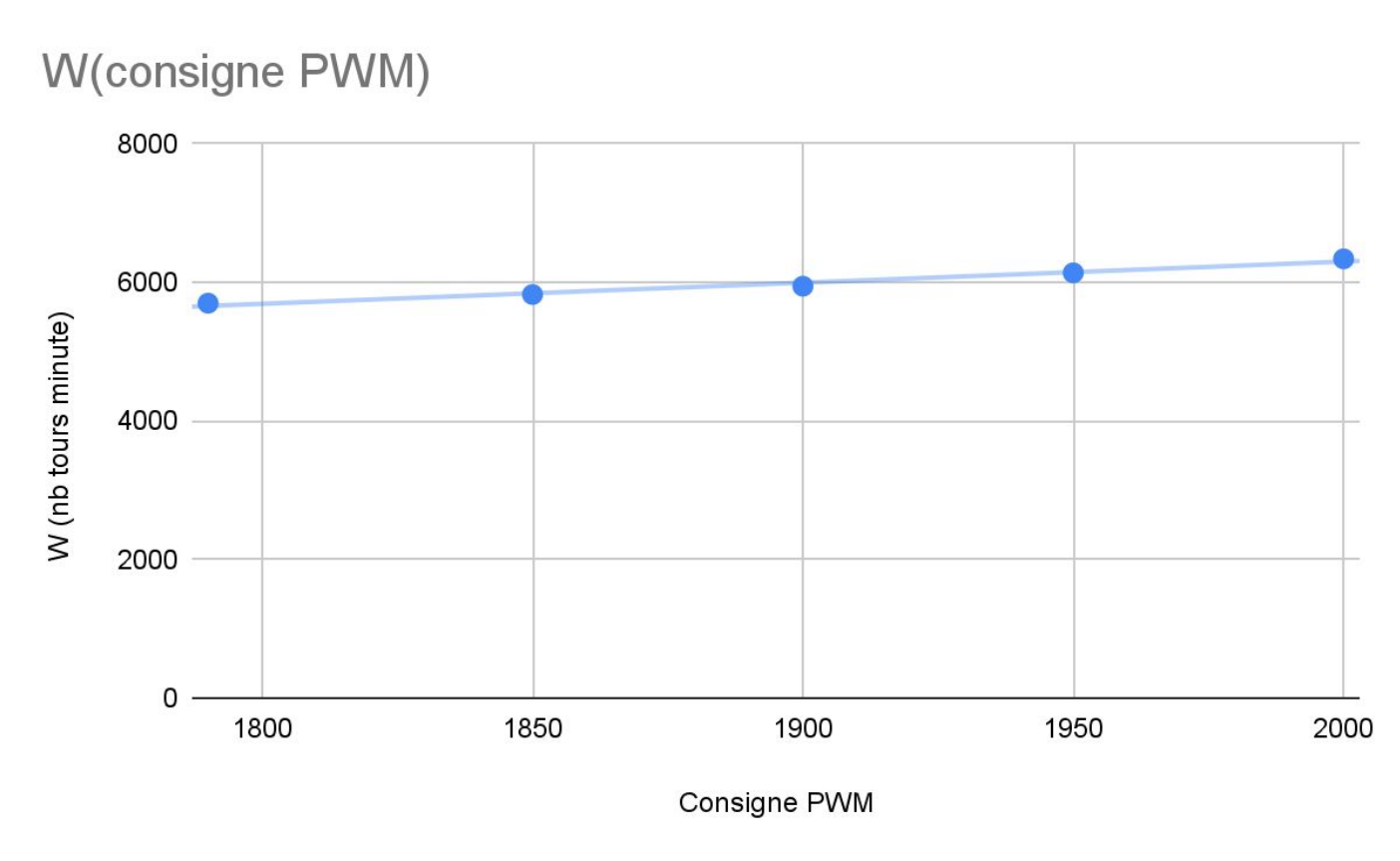
Poids vol stationnaire	1,4	1,28	1,2	1,19	1,07
W (nb tours minute)	6340	6138	5945	5825	5700



$$F_p(w) = 0.001 \cdot w$$

Asservissement du drone

Détermination Expérimentale des Forces de Poussée



Consigne PWM	2000	1950	1900	1850	1790
W (nb tours minute)	6340	6138	5945	5825	5700

$$w = 3.14 \cdot C_{PWM}$$

Asservissement du drone

Équation dynamique du drone

Théorème de l'Énergie Cinétique

$$\Delta E_c = P_{\text{int}} + P_{\text{ext}}$$

cad

$$J \cdot \frac{d^2\alpha}{dt^2} \cdot \frac{d\alpha}{dt} = F_p \cdot r_{\text{drone}} \cdot \frac{d\alpha}{dt}$$

où

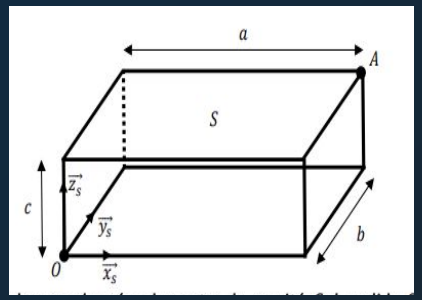
J est le moment d'inertie du drone autour de l'axe de roulis

α est l'angle du drone sur l'axe de roulis

Transformation de Laplace

$$\boxed{H_{drone}(p) = \frac{\alpha}{F_p} = \frac{r_{drone}}{J \cdot p^2}}$$

Calcul du moment d'inertie J



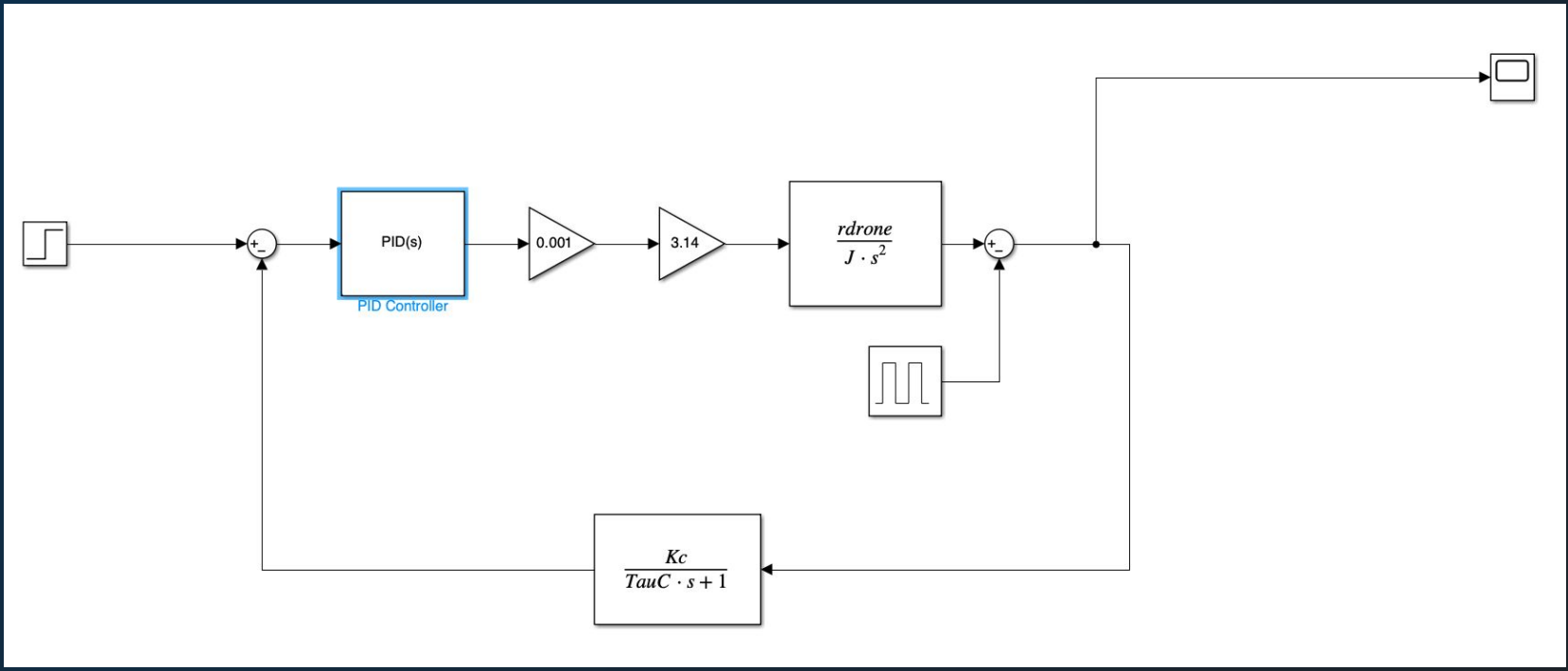
$$V = abc; \quad m = \rho abc; \quad dV = dxdydz;$$

$$I(G,S) = \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{bmatrix}_G$$

$$\boxed{B = \frac{m}{12}(a^2 + b^2)}$$

Asservissement du drone

Schéma Bloc

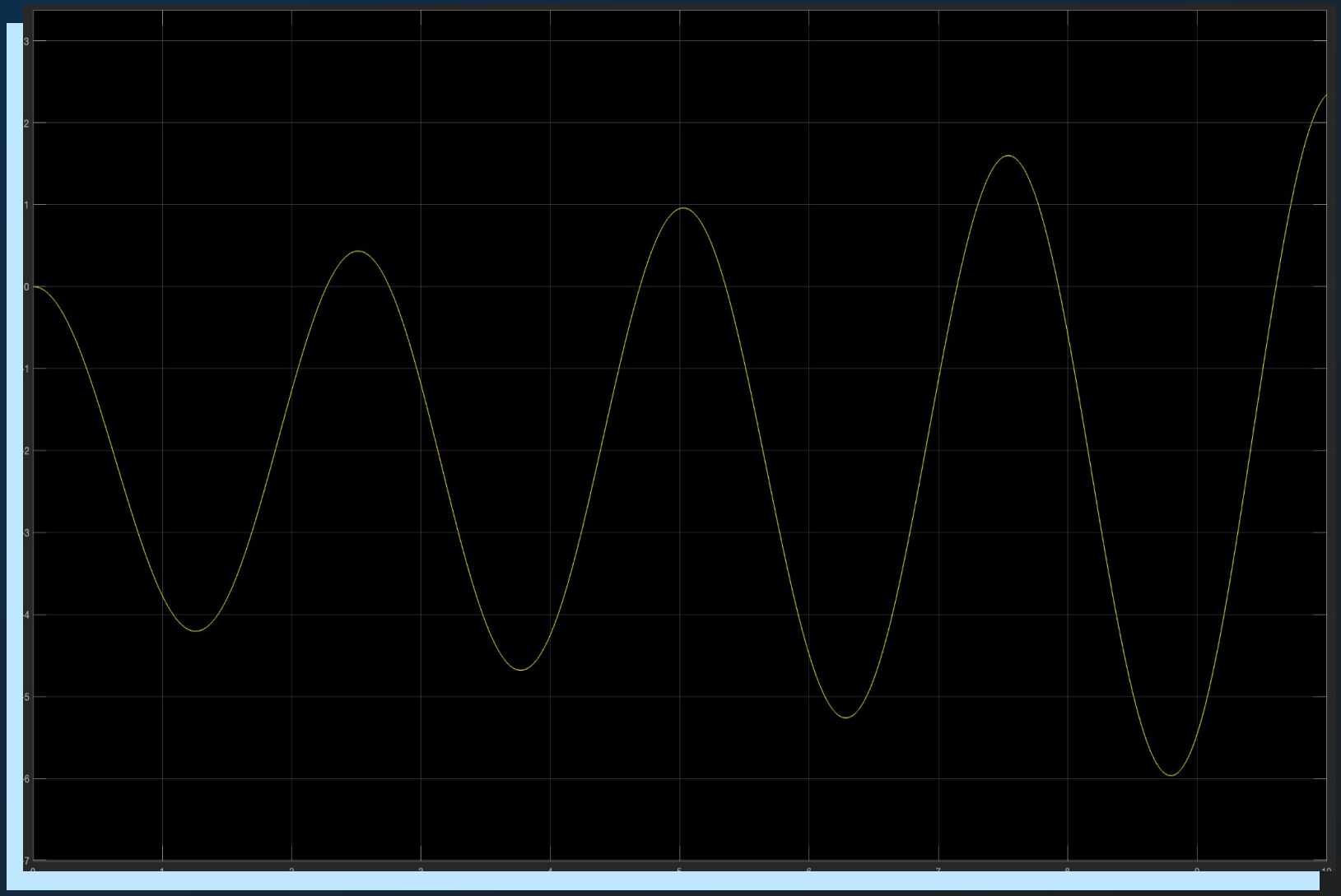


$$PID(p) = K_p \left(1 + \frac{1}{\tau_i p} + \tau_d p\right)$$

Asservissement du drone

Simulation MatLab

Boucle Fermée - U = -2°



Asservissement du drone

Etude Stabilité en Boucle Fermée

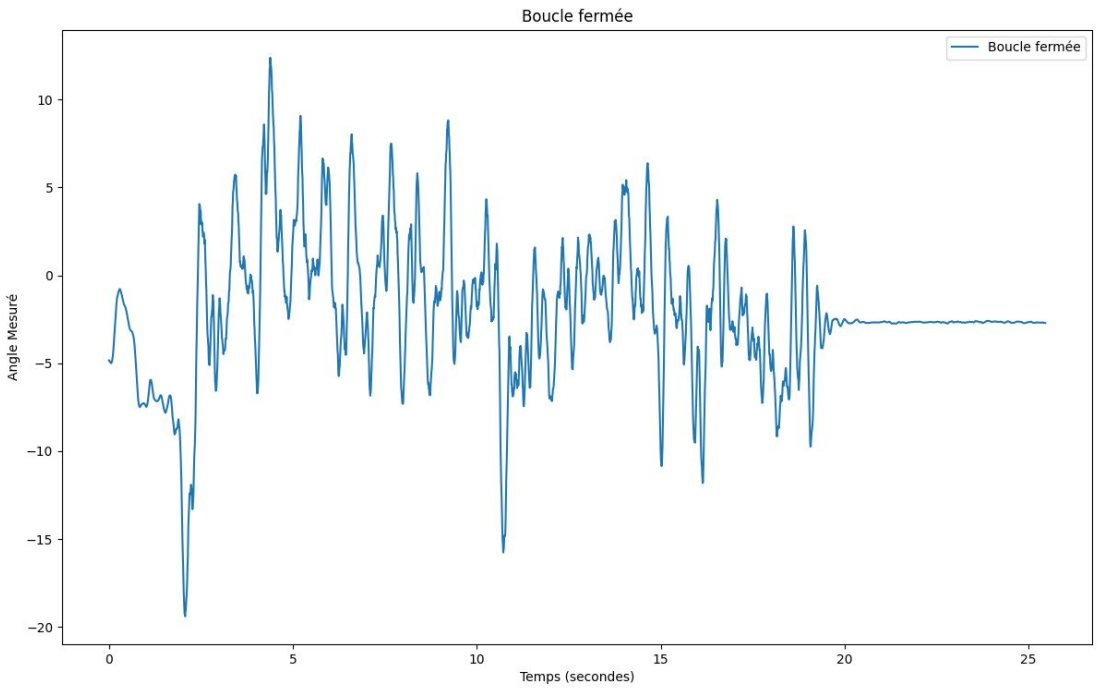
PID :

Kp = 1

Ki = 0

Kd = 0

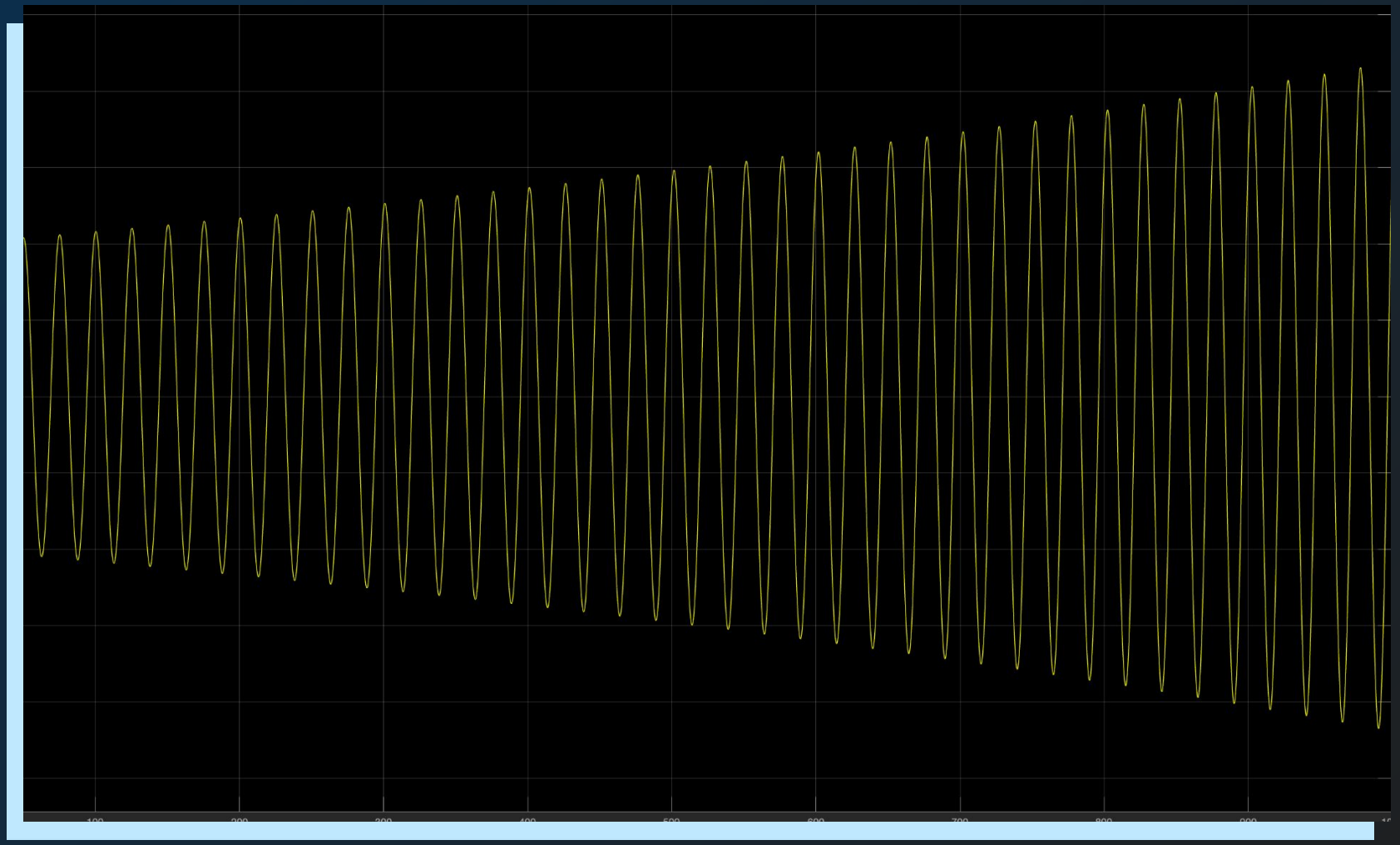
Consigne : 0°



Asservissement du drone

Simulation MatLab

Correcteur proportionnel Kp = 0.8 - U = -2°



Méthode de Ziegler-Nichols impossible…

Asservissement du drone

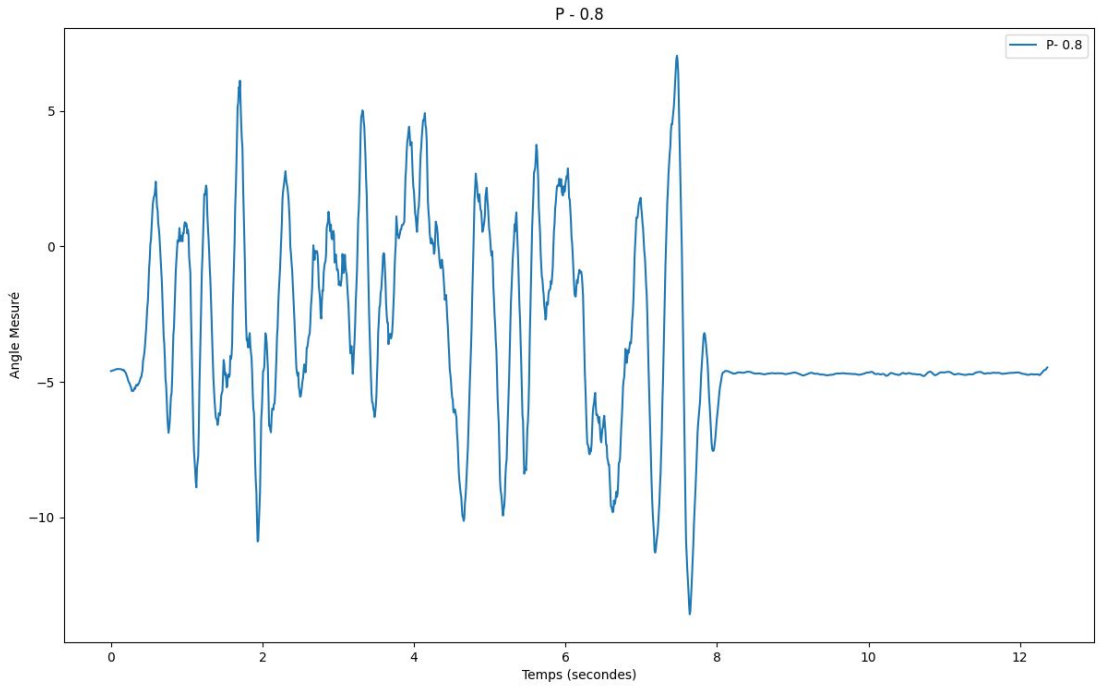
Réglage PID - Méthode Ziegler-Nichols

Recherche **Kp Limite**

PID :

Kp = 0.8
Ki = 0
Kd = 0

Consigne : 0°



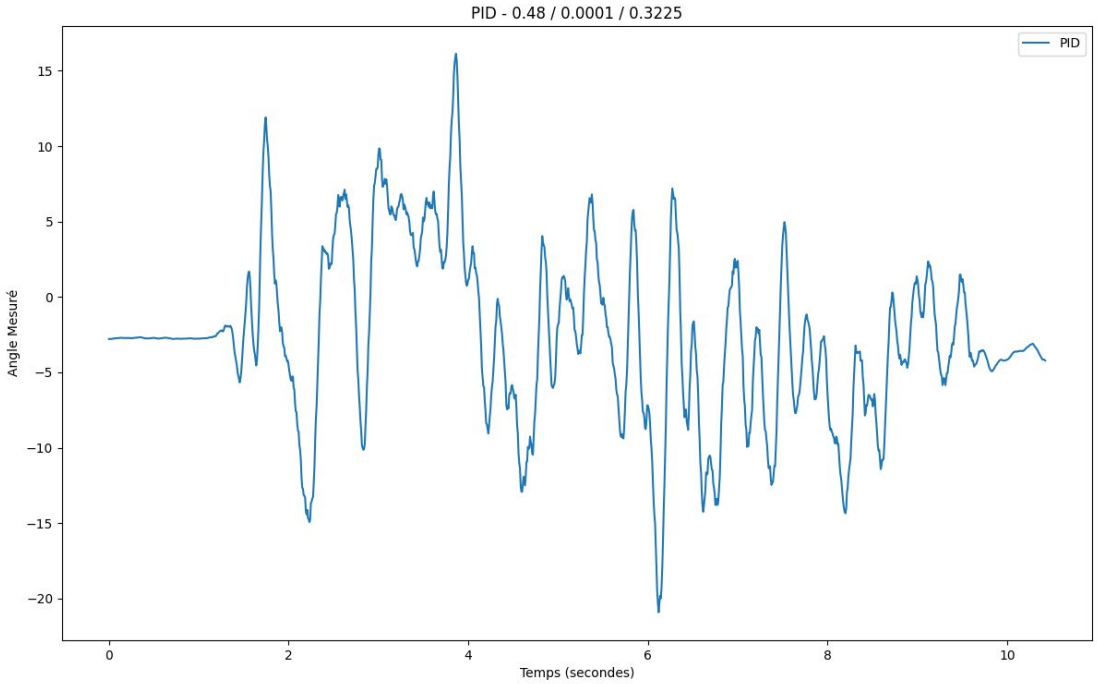
Asservissement du drone

Réglage PID - Méthode Ziegler-Nichols

Calculs Kp, Ki, Kd :

Kp = 0,6Kplim
Ti = 0,5Tosc et Ki =1/Ti
Td = 0,125Tosc et Kd = Td

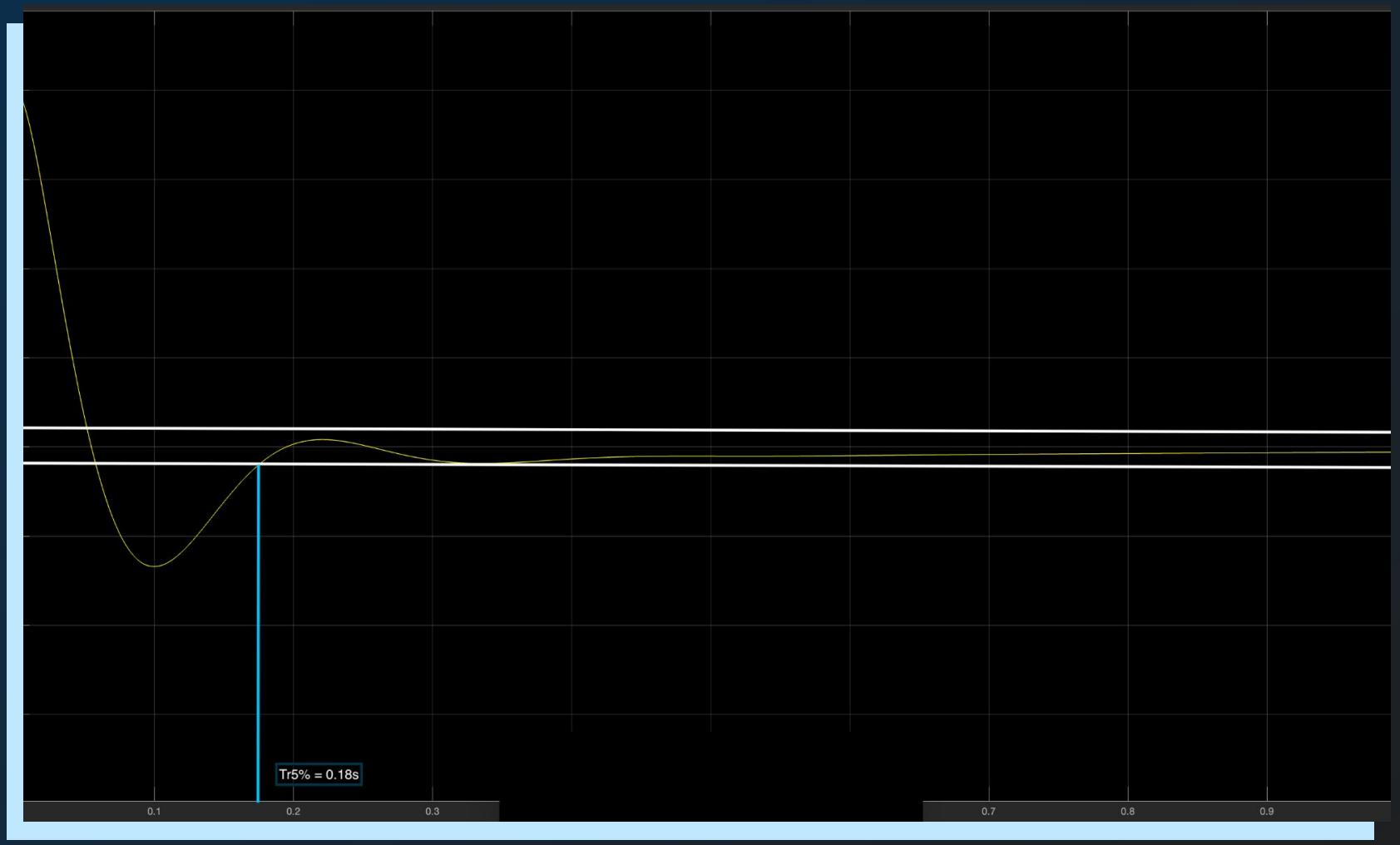
Consigne : 0°



Asservissement du drone

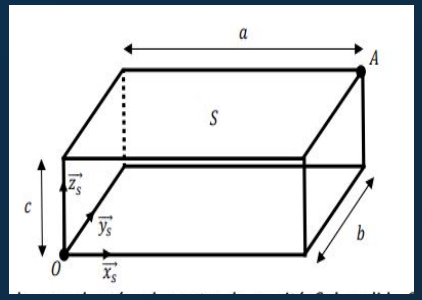
Simulation MatLab

Correcteur PID Kp = 5 / Ki = ,0.01 / Kd = 4 - U = -2°



Annexe 1

Calcul du moment d'inertie J



$$V = abc; \quad m = \rho abc; \quad dV = dxdydz;$$

$$I(G,S) = \begin{bmatrix} A & -F & -E \\ -F & B & -D \\ -E & -D & C \end{bmatrix} G, B_s = \cdots = \begin{bmatrix} A & 0 & 0 \\ 0 & B & 0 \\ 0 & 0 & C \end{bmatrix}$$

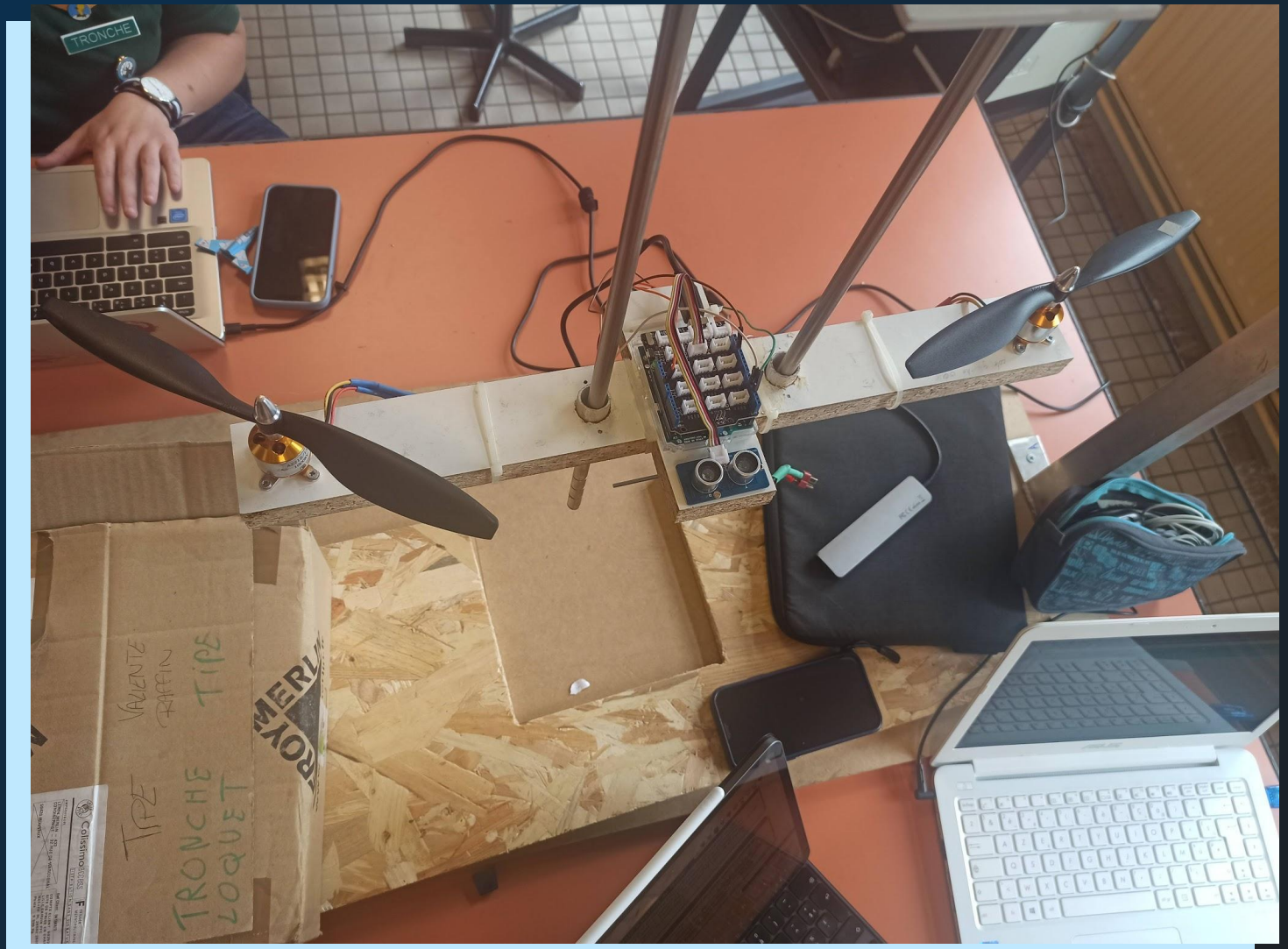
$$B = \int_S (x^2 + z^2)\, dm \quad = \int_S x^2 \rho\, dV + \int_S z^2 \rho\, dV$$

Les bornes de l'intégrale doivent être centrées sur le point G :

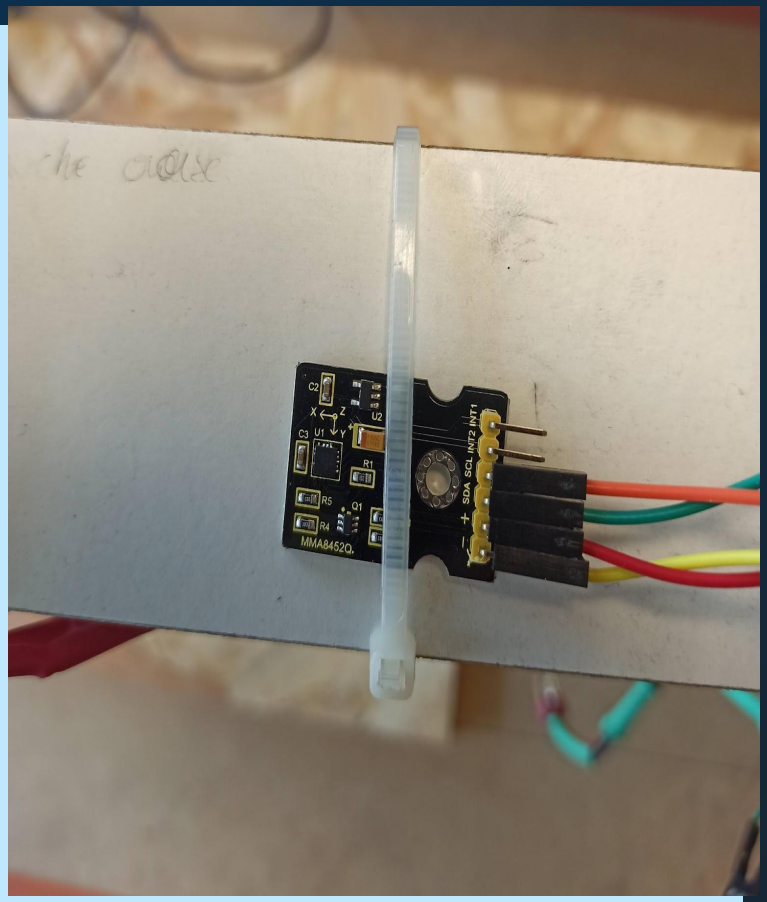
$$B = \rho \int_{-a/2}^{a/2} x^2 dx \int_{-b/2}^{b/2} dy \int_{-c/2}^{c/2} dz + \rho \int_{-a/2}^{a/2} dx \int_{-b/2}^{b/2} dy \int_{-c/2}^{c/2} z^2 dz$$

$$B = \rho \frac{bc}{3} [x^3] \frac{\frac{a}{2}}{\frac{-a}{2}} - rho \frac{ab}{3} [z^3] \frac{\frac{c}{2}}{\frac{-c}{2}} = \boxed{\frac{m}{12}(a^2 + c^2)}$$

Annexe 2.1 (Maquette Drone)



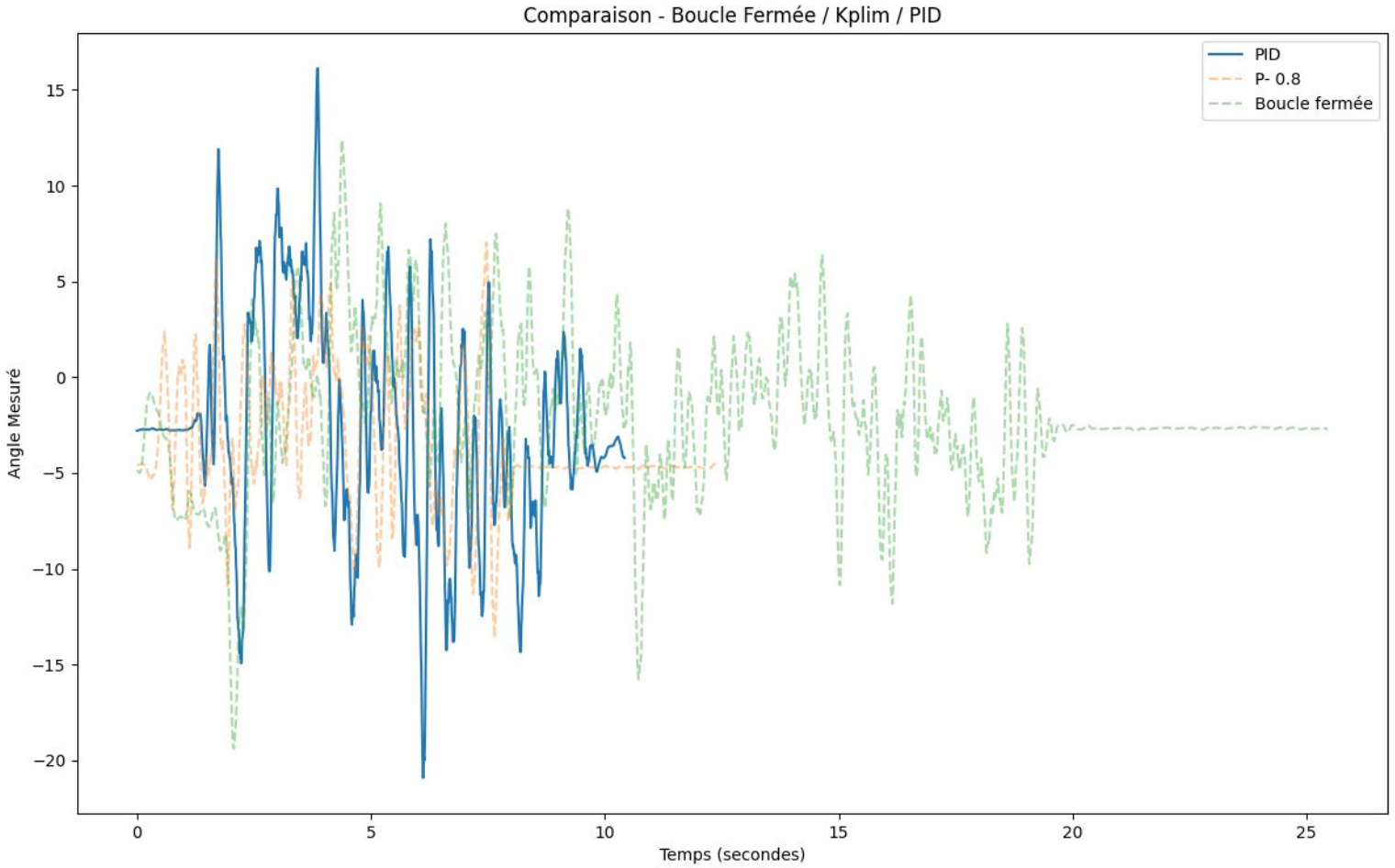
Annexe 2.2 (Maquette Drone)



MMA8452Q 3Axis Accelometer Sensor

Mesure via tachymètre

Annexe 3


Comparaison - Boucle Fermée / Kplim / PID
PID
P- 0.8
Boucle fermée
Angle Mesuré
Temps (secondes)
15
10
5
0
−5
−10
−15
−20
0
5
10
15
20
25

Annexe 4.1 (Fonctions_Utiles.py)

```
from math import *
import numpy as np
from PIL import Image

def show_position_on_image(position, gm,
square_size, color=255):
    # Afficher la position de référence
sur l'image
    # position (w, h)
    try:
        for i in range(square_size):
            for y in range(square_size):

gm[position[1]+i][position[0]+y] = color
    except Exception as e:
        print("error", e)
    return gm
```

```
def compare_matrix(m1, m2):
    for i in range(len(m1)):
        for y in range(len(m1[1])):
            if m1[i][y] != m2[i][y]:
                return False
    return True
```

Annexe 4.2 (Fonctions_Utiles.py)

```python
def correlation_score(master_square, slave_square):
    n = len(master_square)
    master_flattened = [item for sublist in master_square for item in sublist]
    slave_flattened = [item for sublist in slave_square for item in sublist]

    # Calcul des moyennes
    master_mean = sum(master_flattened) / n**2
    slave_mean = sum(slave_flattened) / n**2

    # Calcul des écarts par rapport à la moyenne
    master_deviation = [x - master_mean for x in master_flattened]
    slave_deviation = [x - slave_mean for x in slave_flattened]

    # Calcul des produits des écarts
    product = sum(master_deviation[i] * slave_deviation[i] for i in range(n**2))

    # Calcul des carrés des écarts
    master_deviation_square = sum(x**2 for x in master_deviation)
    slave_deviation_square = sum(x**2 for x in slave_deviation)

    # Calcul du coefficient de corrélation de Pearson
    correlation = product / (master_deviation_square * slave_deviation_square)**0.5

    return correlation
```

Annexe 4.2 (Fonctions_Utiles.py)

```
def find_on_slave(master_square, gm_slave, zone):
    """
    Fonction qui trouve un carré de référence (master_square) de pixels sur
    une matrice d'image (gm_slave)
    gm_slave est la matrice en niveaux de gris de l'image esclave
    zone est un tuple ((w, h), (w, h)) de pixels en haut à gauche et en bas à droite
    qui définit la zone de recherche du carré de référence
    """
    square_size = len(master_square)
    top_left, bottom_right = zone[0], zone[1]

    best_find = (float('inf'), None, None)  # correlation_score, position du coin supérieur gauche et carré
d'esclave correspondant

    # Déplacement dans la portion de l'image
    for width in range(top_left[0], bottom_right[0] - square_size):
        for height in range(top_left[1], bottom_right[1] - square_size):
            slave_square = [gm_slave[height + i][width:width+square_size] for i in range(square_size)]
            cs = correlation_score(master_square, slave_square)
            if cs < best_find[0]:
                best_find = (cs, (width, height), slave_square)

    return best_find
```

Annexe 4.3 (Fonctions_Utiles.py)

```
def find_search_zone(min_offset, square_size, zone_size_coeff, gm):
    """
    Fonction qui trouve la zone ayant la variance de gris la plus élevée
    Une zone très disparate au centre n'est pas utile car
l'approximation de l'angle serait trop mauvaise
    On cherche donc une zone après un +-min_offset
    """
    highest_variance = 0
    best_find_position = None

    zone_size = square_size * zone_size_coeff
    security = zone_size * 3  # Éviter les problèmes de bord

    # Partie supérieure de l'image
    for h in range(security, int(len(gm)/2) - min_offset, zone_size):
        for w in range(security, int(len(gm[0])) - security, zone_size):
            mat = []
            for hi in range(h, h + square_size):
                for val in gm[hi][w:w + zone_size]:
                    mat.append(val)
            variance_score = np.var(mat)
            if variance_score != 0.0:
                print(variance_score)
            if variance_score > highest_variance:
                highest_variance = variance_score
                best_find_position = (w, h)
```

Annexe 4.4 (Fonctions_Utiles.py)

```
        # Partie inférieure de l'image
        for h in range(int(len(gm)/2) + min_offset, len(gm) - security, zone_size):
            for w in range(security, int(len(gm[0])) - security, zone_size):
                mat = []
                for hi in range(h, h + square_size):
                    for val in gm[hi][w:w + zone_size]:
                        mat.append(val)
                variance_score = np.std(mat)
                if variance_score > highest_variance:
                    highest_variance = variance_score
                    best_find_position = (w, h)

        return best_find_position, square_size, zone_size_coeff
```

Annexe 4.5 (Fonctions_Utiles.py)

```python
def find_angle_variation(square_size, zone_size_coeff, search_zone, gm_master, gm_slave):
    """
    Calcule la variation d'angle entre deux images
    square_size : Définit un carré de référence (int)
    zone_size_coeff : Coefficient de taille de la zone (int)
    gm_master : Matrice de l'image maîtresse (np.array())
    gm_slave : Matrice de l'image esclave (np.array())
    """
    # Coordonnées du centre de la zone de recherche (w, h)
    w, h = search_zone
    center_position = (w + int(square_size / 2), h + int(square_size / 2))  # [width][height]

    # Zone de recherche
    zone_size = int(square_size * zone_size_coeff)
    search_zone = ((center_position[0] - zone_size, center_position[1] - zone_size),
                   (center_position[0] + zone_size, center_position[1] + zone_size))

    # Définition du carré de référence
    master_square = [None for _ in range(square_size)]
    for h in range(center_position[1], center_position[1] + square_size):
        master_square[h - center_position[1]] = gm_master[h][center_position[0]:center_position[0] + square_size]
```

Annexe 4.6 (Fonctions_Utiles.py)

```
rslt = find_on_slave(master_square, gm_slave, search_zone)

   # Afficher la zone de recherche sur l'esclave
   gm_slave = show_position_on_image(search_zone[ 0], gm_slave, 3,
255)
   gm_slave = show_position_on_image(search_zone[ 1], gm_slave, 3,
255)
   # Afficher la position du carré esclave sur l'esclave
   gm_slave = show_position_on_image(rslt[ 1], gm_slave, 5, 100)

   image_from_array = Image.fromarray(np.array(gm_slave))
   image_from_array.show()

   # Position du coin supérieur gauche du carré de référence
   TopLeftPx = (center_position[ 0], center_position[ 1])  # (w, h)
   return angle_variation(TopLeftPx, rslt[ 1])
```

Annexe 5 (main.py)

```
from PIL import Image
import numpy as np
from Fonctions_utiles import *

image_link =
"/Users/rodrigueleitao/Downloads/TIPE/Corrélation
d'Image/Test
Fonctions/find_search_zone/Test_Zone_Variance.jpg
"
#in gray scale
g_master = Image.open(image_link).convert('L')

#image matrix
gm_master = np.array(g_master)




from matplotlib import pyplot as plt

x = [angle for angle in range(2, 5)]
y = []
```

```
for angle in x :
    g_slave = g_master.rotate(angle)
    gm_slave = np.array(g_slave)

    search_zone, square_size, zone_size_coeff =
find_search_zone(10, 25, 2, gm_slave)
    y.append(find_angle_variation(square_size,
zone_size_coeff, search_zone,  gm_master,
gm_slave))




plt.title("Approximation de la rotation d'angle
sur l'image")
plt.xlabel("Angle Réel en °")
plt.ylabel("Variation Angle Détectée en °")
plt.plot(x,y)

plt.plot(x, x, '-r', label='Real Image Rotation')


plt.show()
```