L'extraction de la trajectoire d'une balle de tennis de table à partir d'une seule caméra

Anas TIBATI
N°SCEI : 21447

Table des matières

Table des matières

Positionnement du problème

Figure – Match de tennis de table au sein de l'établissement

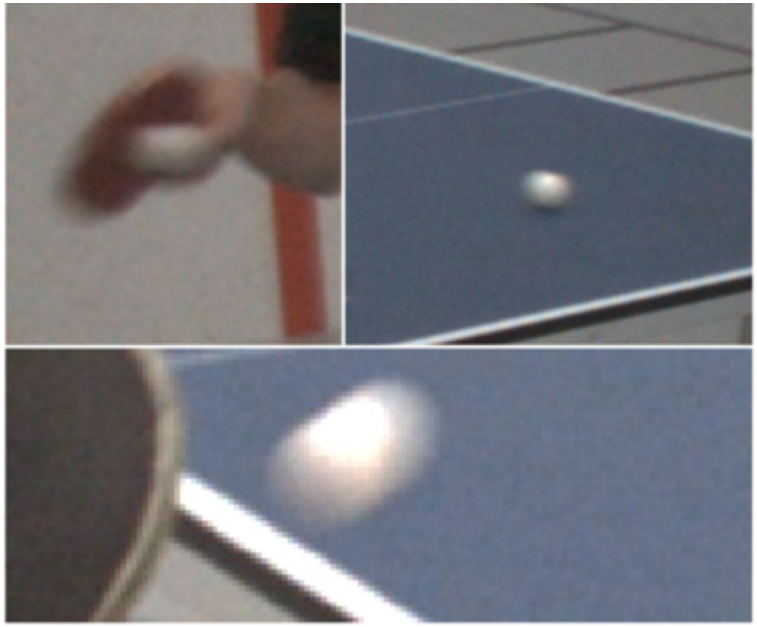


Figure – Illustrations de difficultés liées à la détection d'objet

Positionnement du problème

Comment peut-on donc extraire une trajectoire aussi précise que possible d'une balle de tennis de table à partir d'une seule caméra ?

Déroulement du TIPE

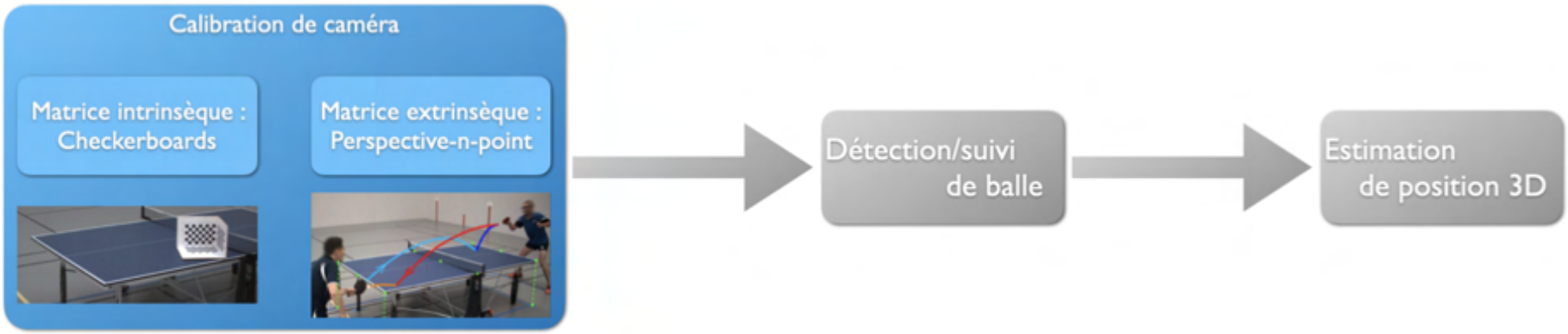


Figure – L'enchainement des étapes du TIPE

Table des matières

La calibration d'une caméra

Principe

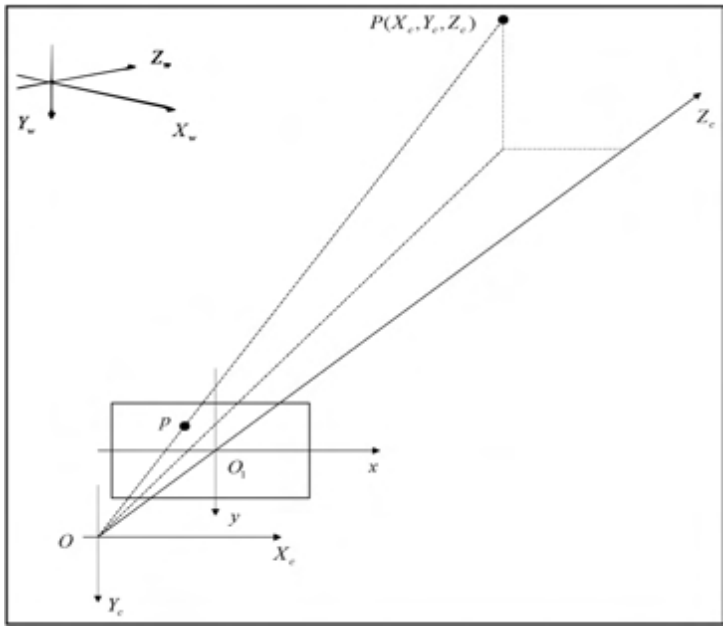


Figure – Les coordonées d'un point P dans le repère caméra

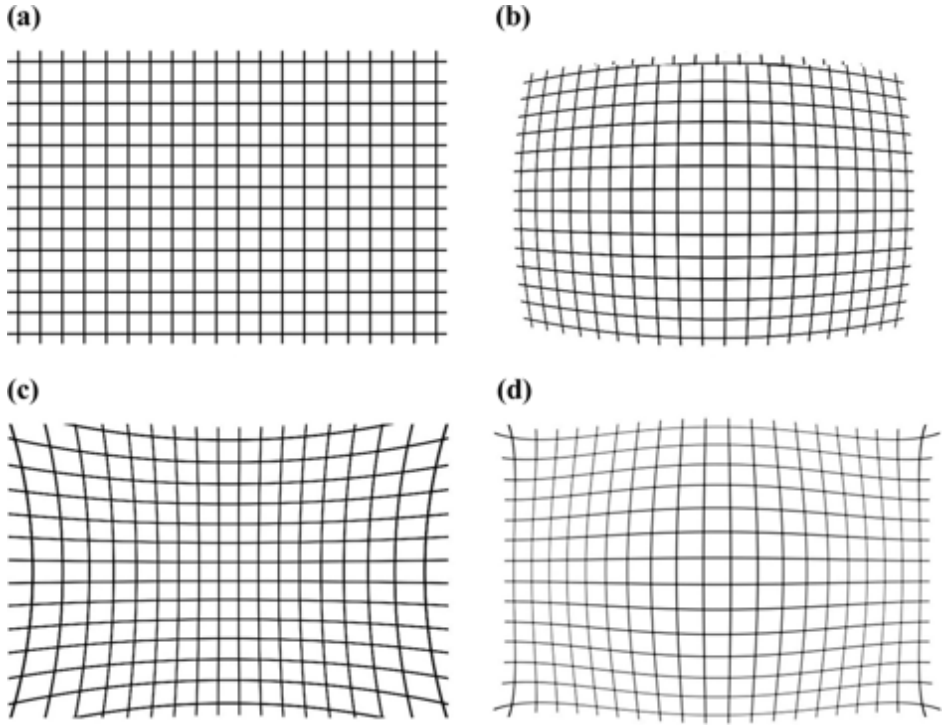
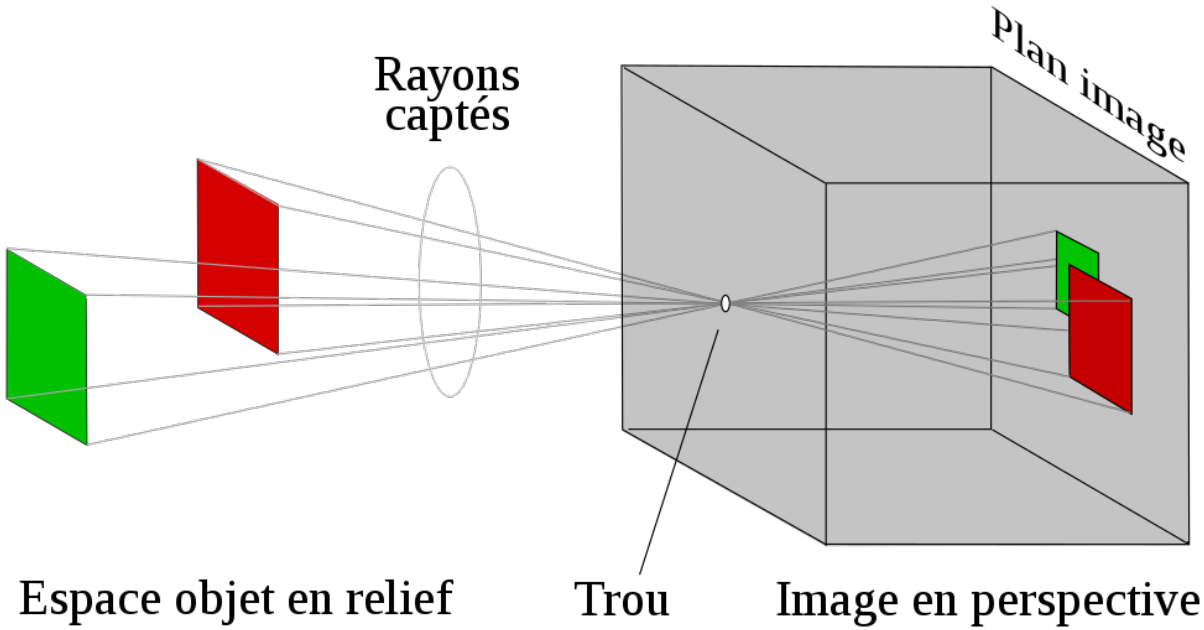


Figure – Différents types de distortion

La calibration d'une caméra

Modélisation

- On choisit le modèle du sténopé :



La calibration d'une caméra

Modélisation

- Il convient de préciser les repères orthonormés nécessaires pour modéliser le fonctionnement de la caméra :

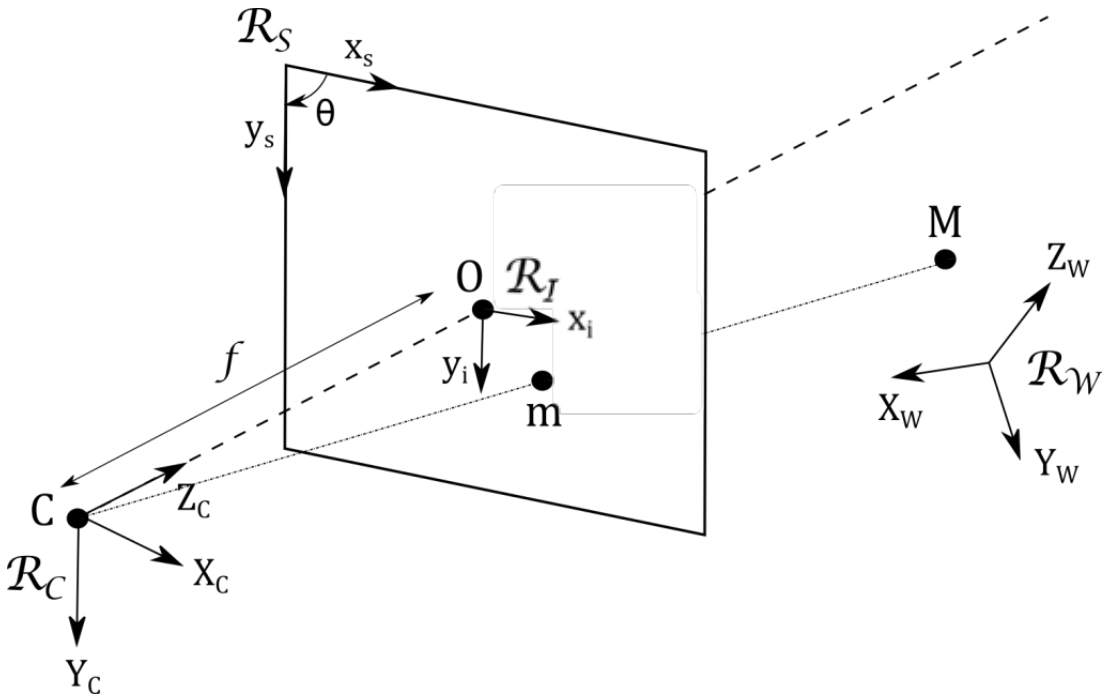
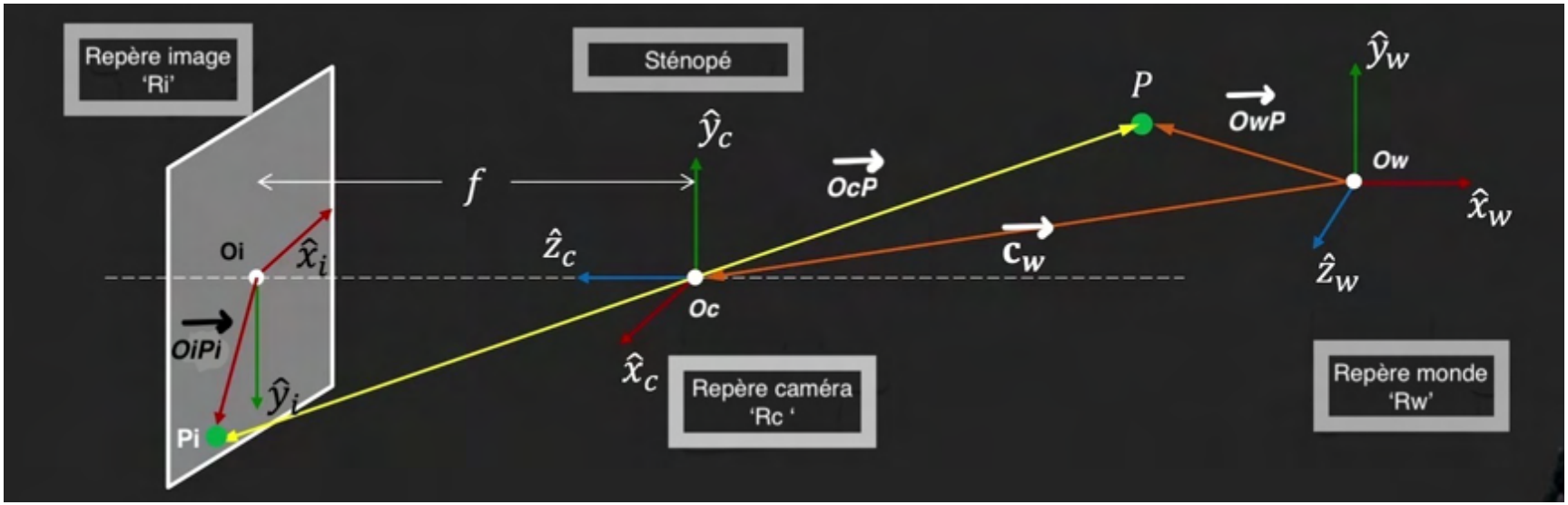


Figure – Repères associés à l'étalonnage de la caméra

La Calibration d'une caméra

Modélisation



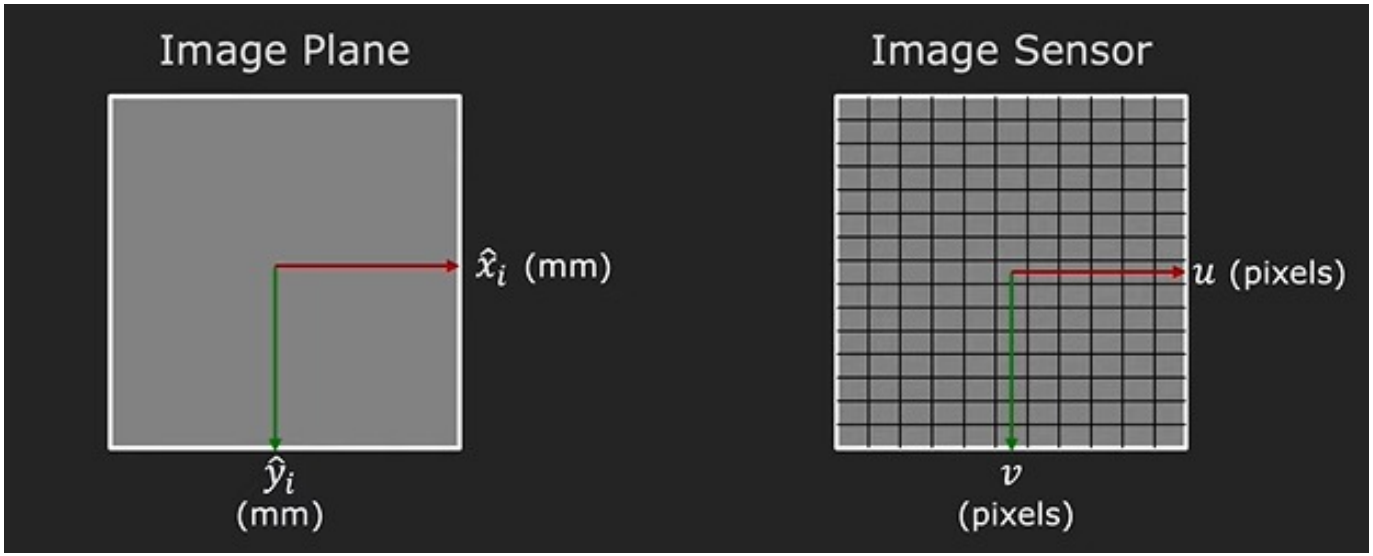
Résultats géométriques

Les résultats de la projection perspective s'expriment : $\frac{x_i}{f} = \frac{x_c}{z_c}$ et $\frac{y_i}{f} = \frac{y_c}{z_c}$

La calibration d'une caméra

Modélisation

- Sur le plan image :



- $u = m_x\, x_i = m_x\, f\, \frac{x_c}{z_c} + o_x = f_x\, \frac{x_c}{z_c} + o_x$
- $v = m_y\, y_i = m_y\, f\, \frac{y_c}{z_c} + o_y = f_y\, \frac{y_c}{z_c} + o_y$

La calibration d'une caméra

Modélisation

En coordonnées homogènes :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} z_c\, u \\ z_c\, v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x\, x_c + z_c\, o_x \\ f_y\, y_c + z_c\, o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$$

La calibration d'une caméra

Matrice de projection

- Matrice de calibration : K= $\begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$

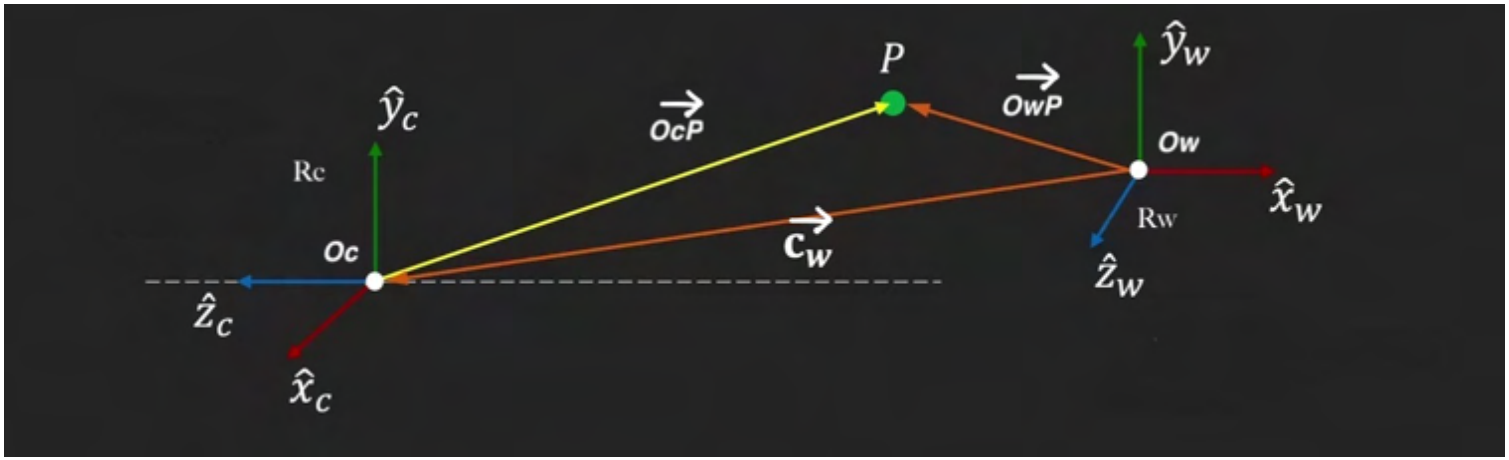
- **La matrice intrinsèque** convertit les points du système de coordonnées de la caméra au système de coordonnées des pixels.

- Elle est de la forme : M_{int}=[K|0]= $\begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

- Ainsi la relation entre les deux coordonnées du point P et sa projection dans le plan s'écrit : $\tilde{\mathbf{u}} = M_{int}\tilde{\mathbf{x}}_c$

La calibration d'une caméra

Matrice de projection



- On peut alors exprimer la transformation du système de coordonnées de l'espace de travail au système de coordonnées de la caméra :
  - $\vec{O_cP} = R.\left(\vec{O_wP} - \vec{c}_w\right) = R.\vec{O_wP} - R.\vec{c}_w = R.\vec{O_wP} + \vec{t}$
  - *avec* : R$= \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{23} & r_{32} & r_{33} \end{bmatrix}$ et $\vec{t} = -R.\vec{c}_w$

La calibration d'une caméra

Matrice de projection

- **La matrice extrinsèque** est une matrice de transformation du système de coordonnées de l'espace de travail au système de coordonnées de la caméra

- Elle est de la forme : $M_{ext} = \begin{bmatrix} R_{3\times3} & \mathbf{t} \\ 0_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{23} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

La calibration d'une caméra

Matrice de projection

Conclusion sur la matrice de projection

- Finalement on définit donc la matrice de projection P : $P = M_{int}\, M_{ext}$
- On a donc : $\tilde{\mathbf{u}} = P\, \tilde{\mathbf{x}}_w$

La calibration d'une caméra

Méthode de Zhang

Définition

La méthode de Zhang pour le calibrage de la caméra repose sur l'utilisation d'un motif planaire avec des coins bien détectables pour estimer les paramètres de la caméra.

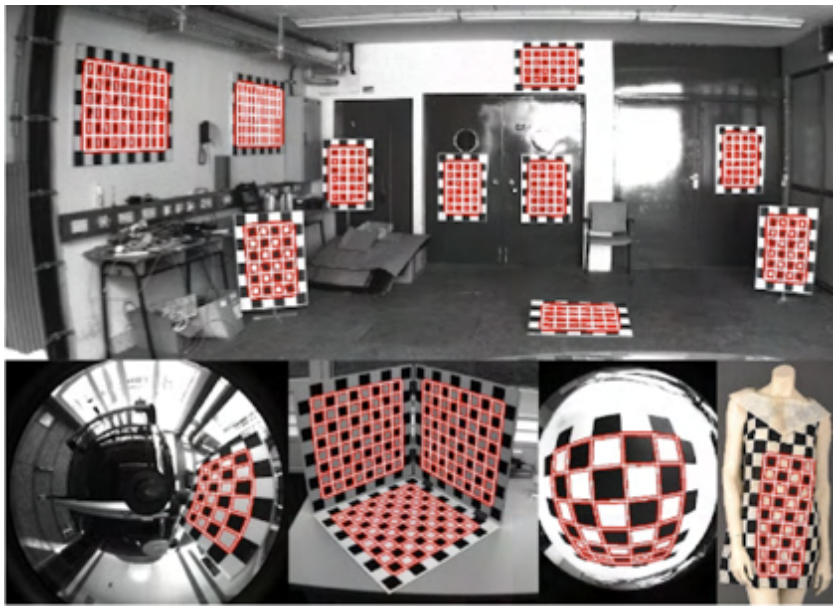


Figure – Illustration de scènes composées de plusieurs mires de calibration

La calibration d'une caméra

Méthode de Zhang

Etapes de la méthode de Zhang

1. Acquisition des images de calibration
2. Détection des coins du motif
3. Extraction des coordonnées des coins
4. Estimation des paramètres intrinsèques

La calibration d'une caméra

Résultats expérimentaux

1. Acquisition des images de calibration

Figure – Illustration de scènes d'acquisition des images

La calibration d'une caméra

Résultats expérimentaux

1. Détection des coins du motif

Figure – Détetction des coins par le code Python

La calibration d'une caméra

Résultats expérimentaux

Des résultats théoriques aux résultats expérimentaux

- $M_{int} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

$$\rightarrow \begin{bmatrix} 2,8412 \times 10^3 & 0 & 1,9386 \times 10^3 & 0 \\ 0 & 2,8317 \times 10^3 & 1,4777 \times 10^3 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Table des matières

Forces aérodynamiques

Effet de Magnus

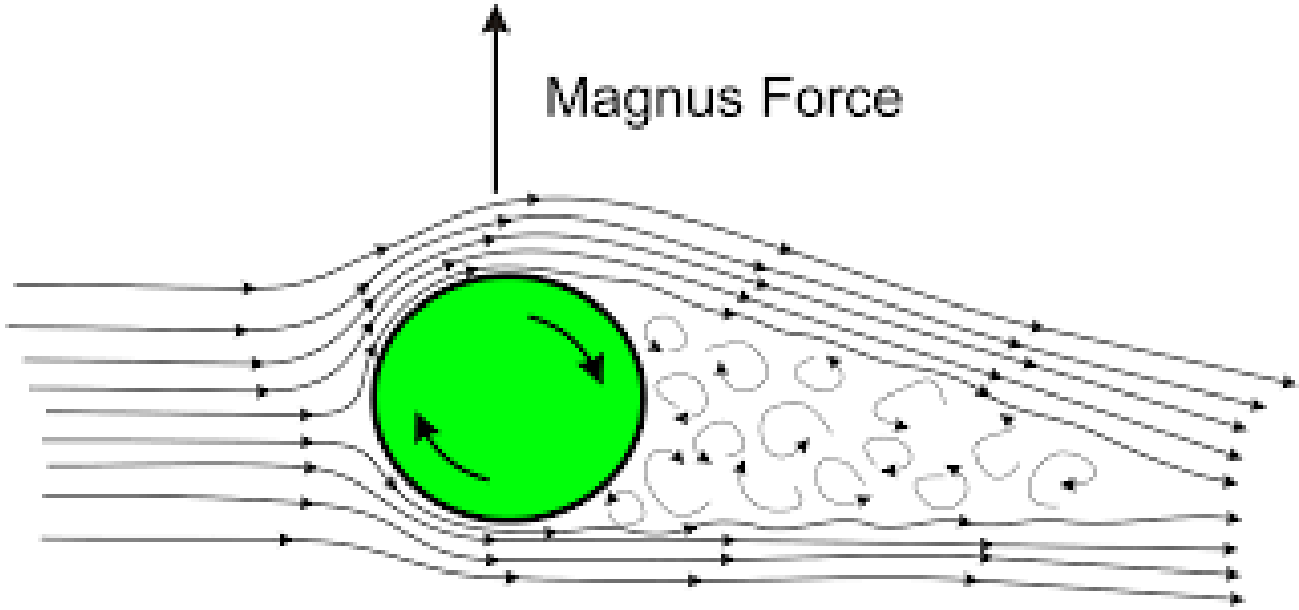


Figure – Représentation du flux d'air lié à la rotation d'une balle

Forces aérodynamiques

Force de trainée

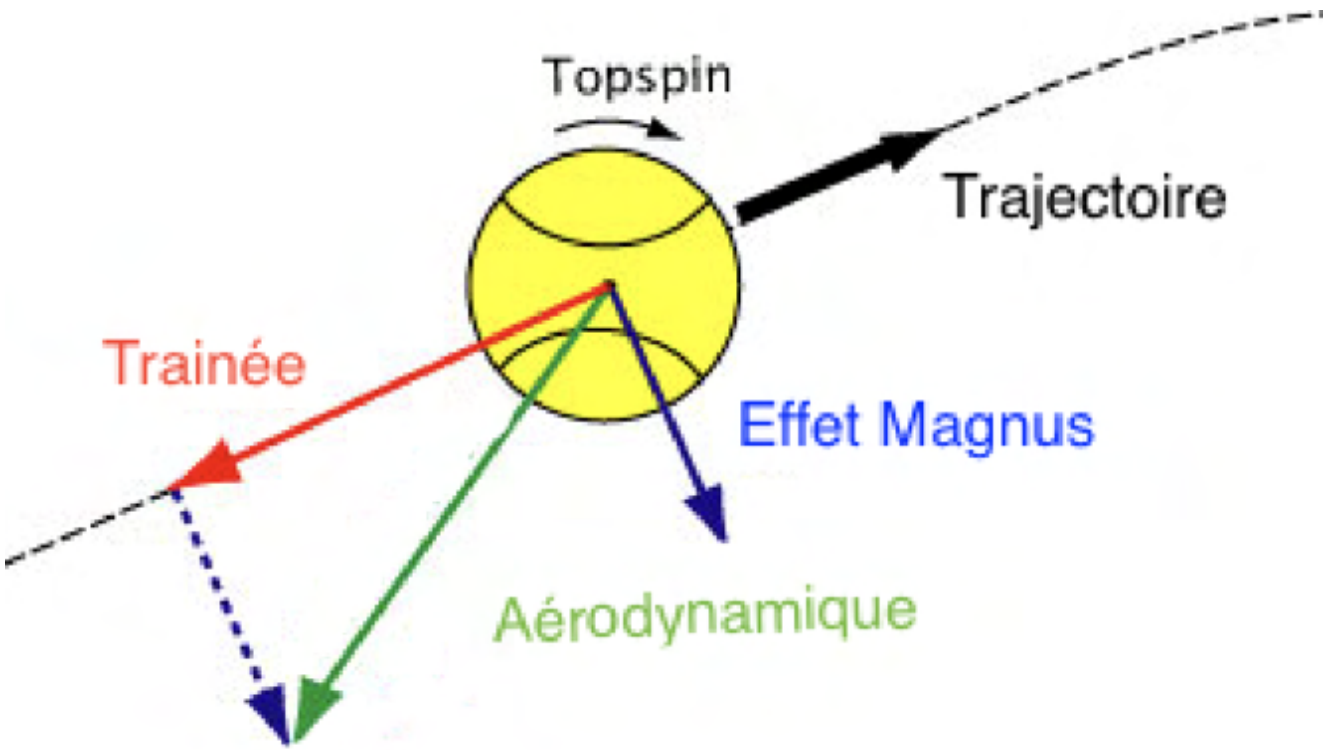


Figure – Les forces aérodynamiques appliquées à la balle

Etude théorique

Expression des forces

Formules théoriques

- $\overrightarrow{F_t} = \frac{-1}{2} C_D \rho A V \overrightarrow{V}$
- $\overrightarrow{F_m} = S_0 \, \vec{\omega} \wedge \overrightarrow{V}$
- On a aussi : $||\overrightarrow{F_m}|| = \frac{1}{2} C_L \rho A V^2$
- Avec : $C_L = \frac{2 S_0 \omega}{\rho A V}$

Valeurs numériques

- Pour notre balle de tennis de table, dans l'air ambiant, on a :
  - $C_D = 0.4$
  - $C_L = 0.7$

Equations de mouvement

Application du principe fondamental de la dynamique

- Par application du principe fondamental de la dynamique on retrouve : $m \frac{\mathrm{d}V}{\mathrm{d}t} = F_g + F_t + F_m$ (1)

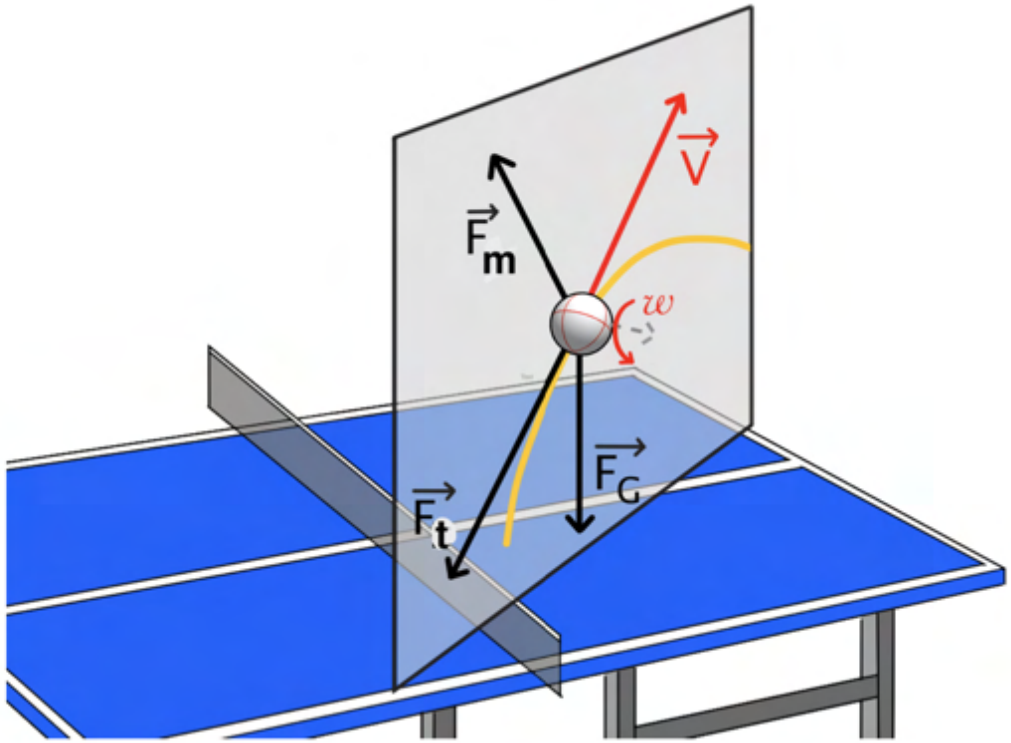


Figure – Schématisation des forces appliquées sur la balle

Equations de mouvement

Application du principe fondamental de la dynamique

- Afin de diminuer le nombre d'équations, on étudiera le mouvement de la balle par rapport au nouveau repère défini $\mathcal{R}'$

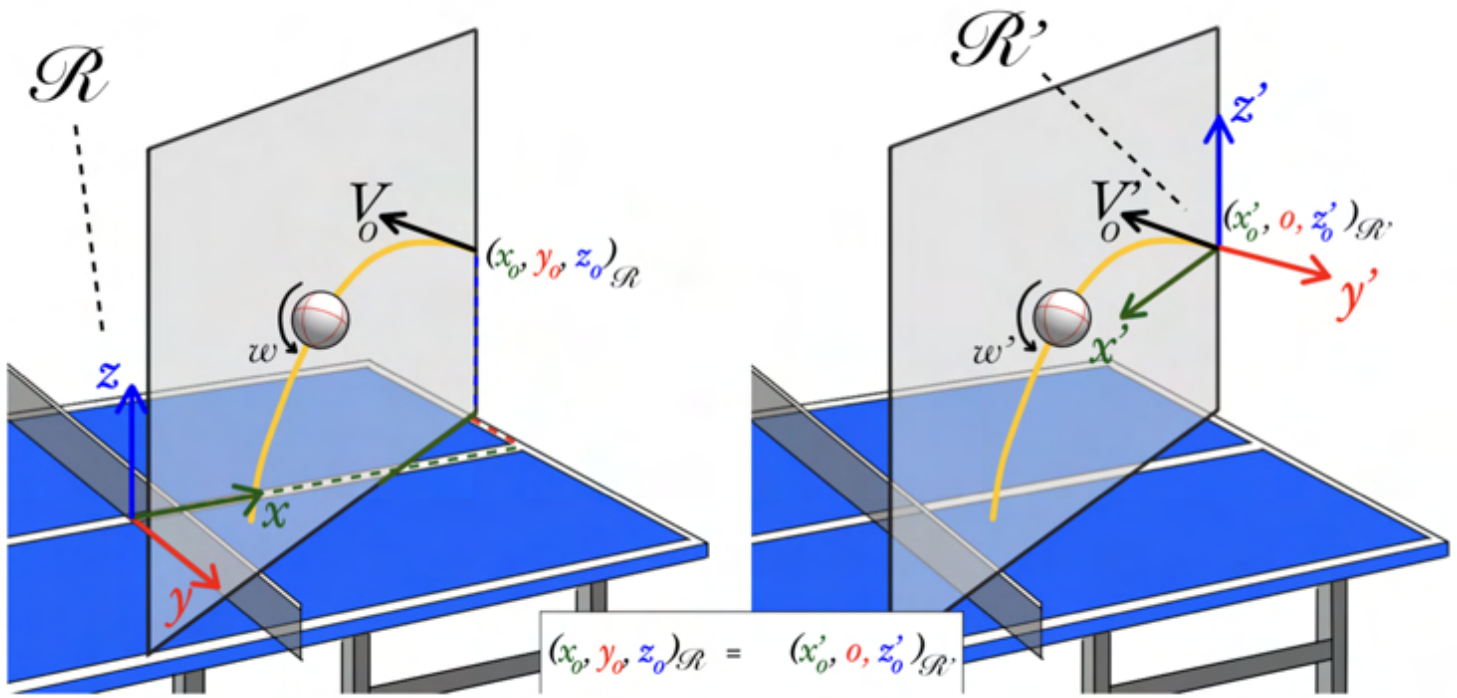


Figure – Repère choisi pour l'étude : $\mathcal{R}'$

Equations de mouvement

Application du principe fondamental de la dynamique

- En rapportant l'étude au repère $\mathcal{R}'$ défini précédemment avec V(t) la norme du vecteur vitesse, θ(t) l'angle du vecteur vitesse avec l'axe horizontal.
- On retrouve alors par projection de (1) :
  - $\frac{d^2x'(t)}{dt^2} = \frac{-1}{2m}\rho AV'(t)^2(C_D cos(\theta'(t)) + C_L(t)sin(\theta'(t)))$
  - $\frac{d^2z'(t)}{dt^2} = -g - \frac{1}{2m}\rho AV'(t)^2(C_D sin(\theta'(t)) + C_L(t)cos(\theta'(t)))$

Résolution numérique et modélisation

- En connaissant, pour t=0, tous les paramètres cinématiques et les coordonnées de la balle, il est possible d'estimer de manière itérative les positions x'(t) et z'(t) du centre de masse de la balle le long des trajectoires , sous la supposition que $\omega = \omega_0$ reste constante.

Résolution numérique et modélisation

- Essayons de représenter la trajectoire d'une balle pour chacun de ces trois types de coups :

Coup	Translation (m/s)		Rotation (rotations/s)	
	min	max	min	max
Top Spin	10,00	16,66	30,00	70,00
Contre-Attaque	4,17	10,00	10,00	20,00
Poussette	1,38	5,55	-15,00	0,00

Figure – Plages de données des vitesses de translation et de rotation pour chaque type de coup

Résolution numérique et modélisation

- Afin de générer la séquence de positions successives, la position initiale de la balle est prise à :
  - : x_0 = -1.5m (juste en dehors de la table de longueur 2.743m)
  - : z_0 = 25cm (au dessus de la table)

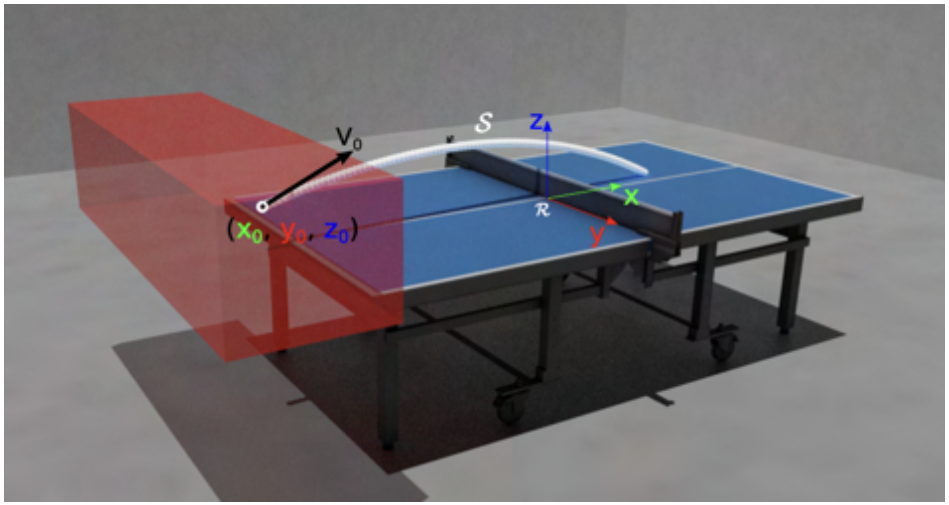
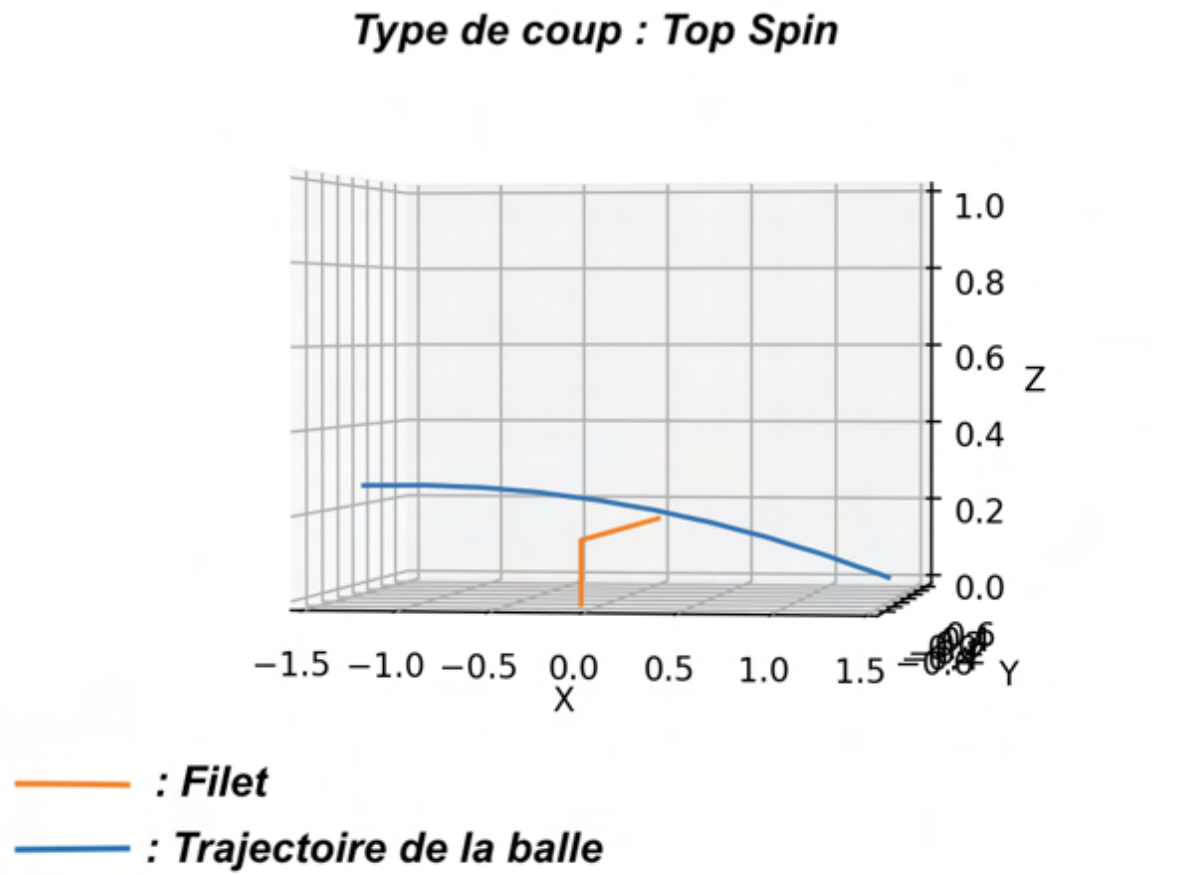


Figure – Représentation de la zone d'initialisation de la trajectoire

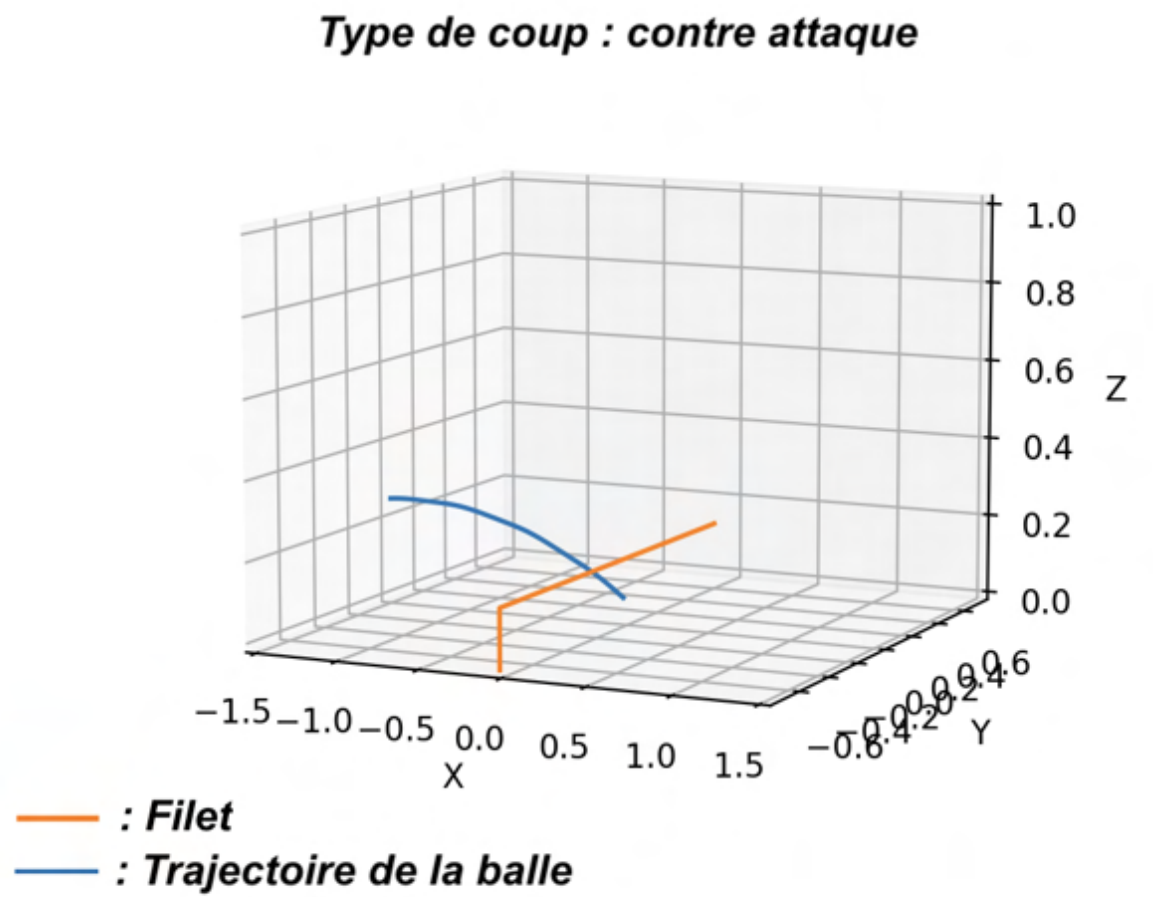
Résolution numérique et modélisation

Type de coup : 1. Top Spin



Résolution numérique et modélisation

Type de coup : 2. Contre Attaque



Résolution numérique et modélisation

Type de coup : 3. Poussette

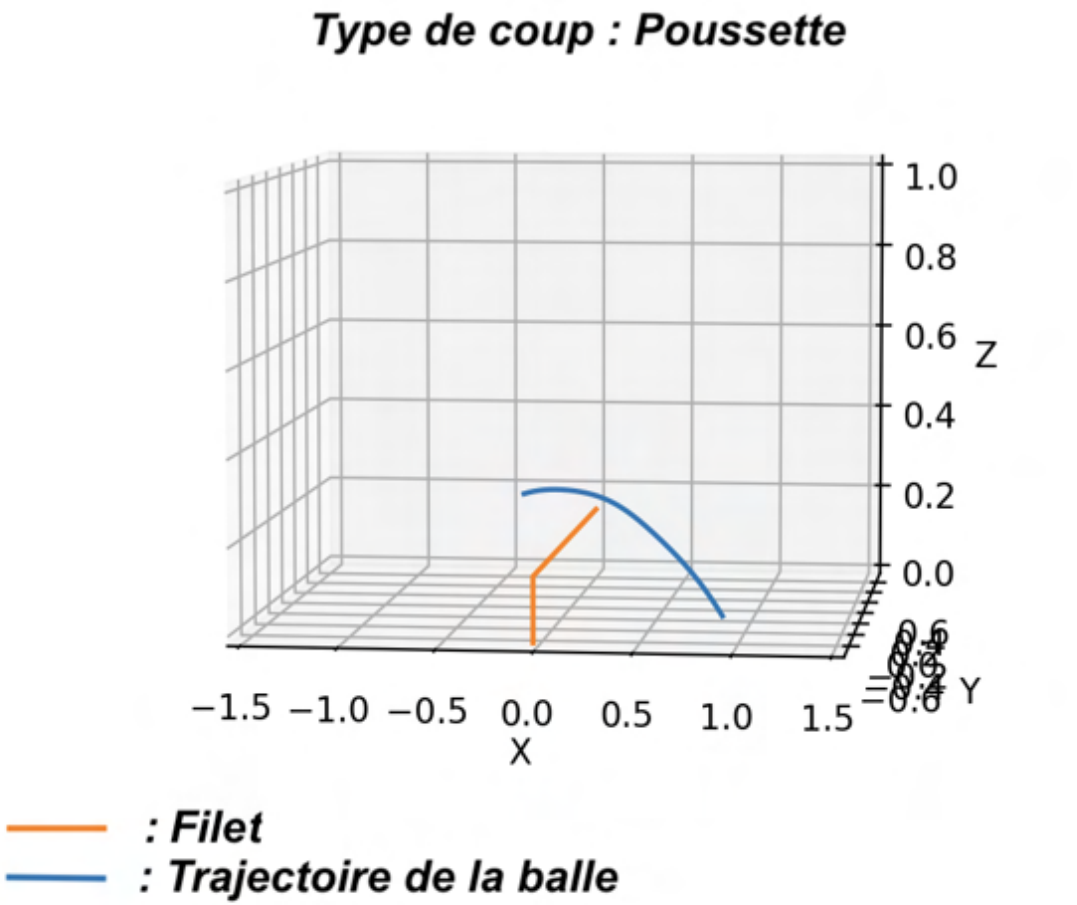


Table des matières

Modélisation de la trajectoire de la balle

- Maintenant vient l'étape de la détection de la balle sur une séquence vidéo :

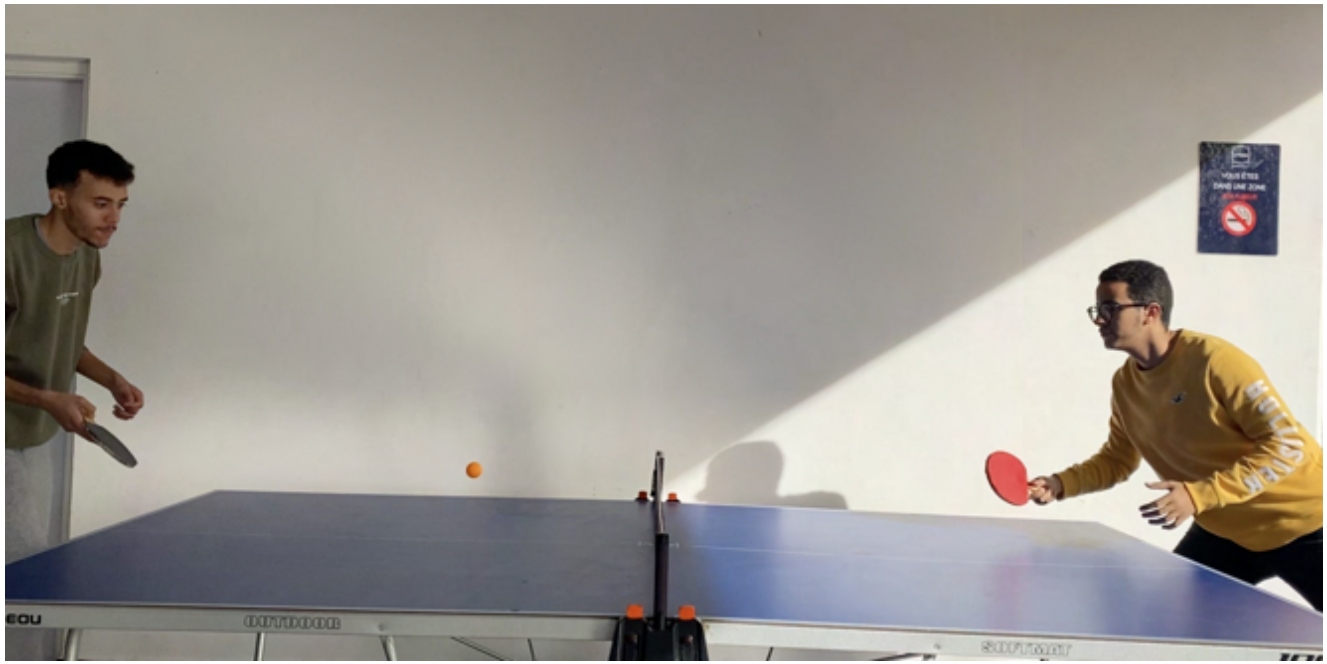


Figure – Image prise de la séquence de l'échange à traiter

Modélisation de la trajectoire de la balle

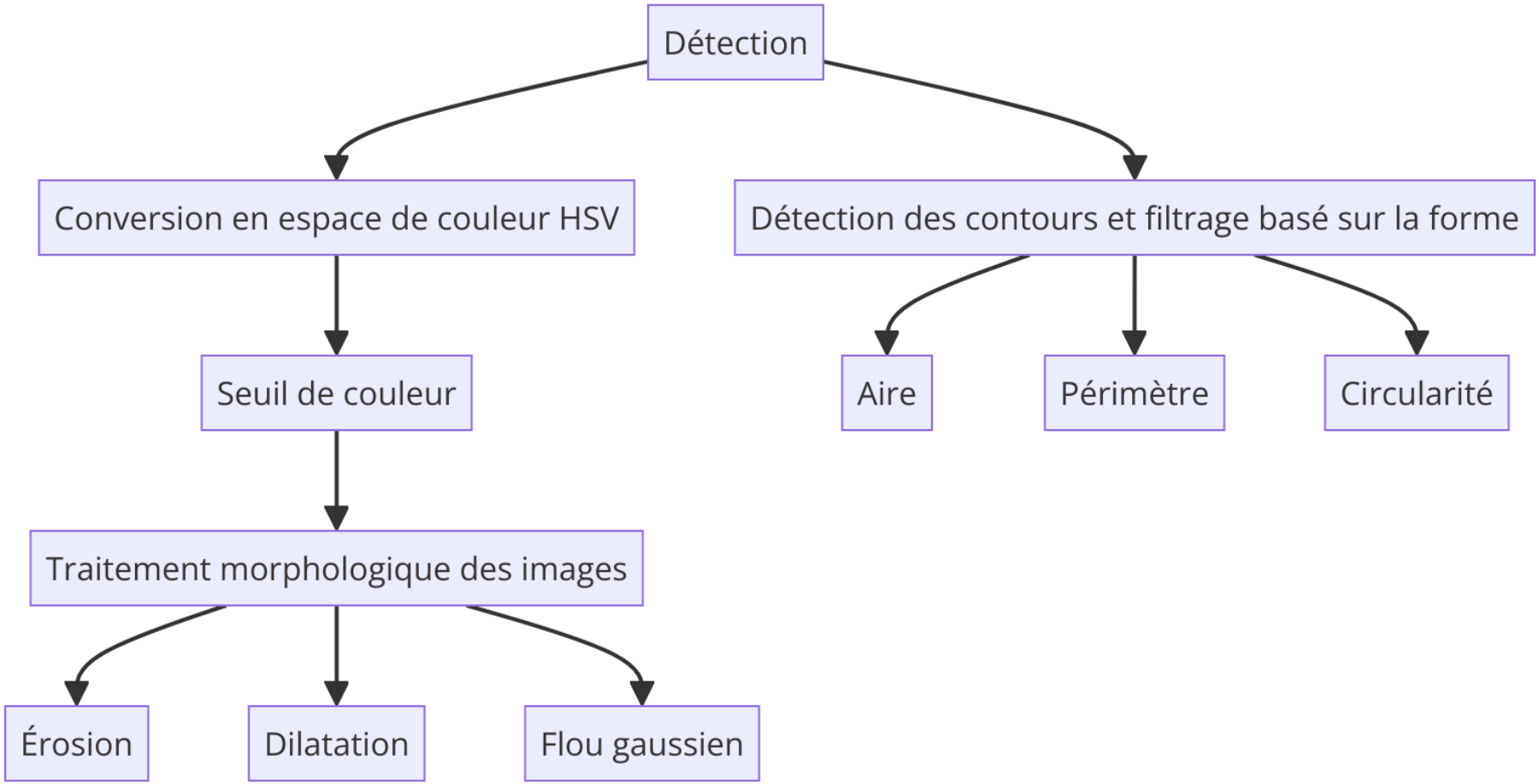
Dispositif d'acquisition

Figure – Caméra positionnée pour l'acquisition de la séquence

Figure – L'emplacement de la caméra par rapport à la table

Modélisation de la trajectoire de la balle

Etapes de la détection de la balle



Modélisation de la trajectoire de la balle

Résultat de la détection de la balle

Figure – Exemple de détection de la balle

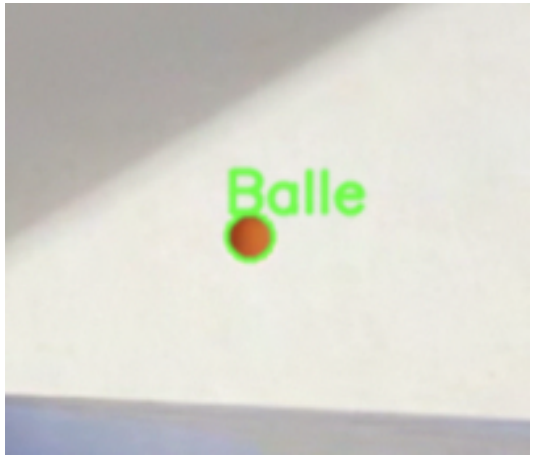
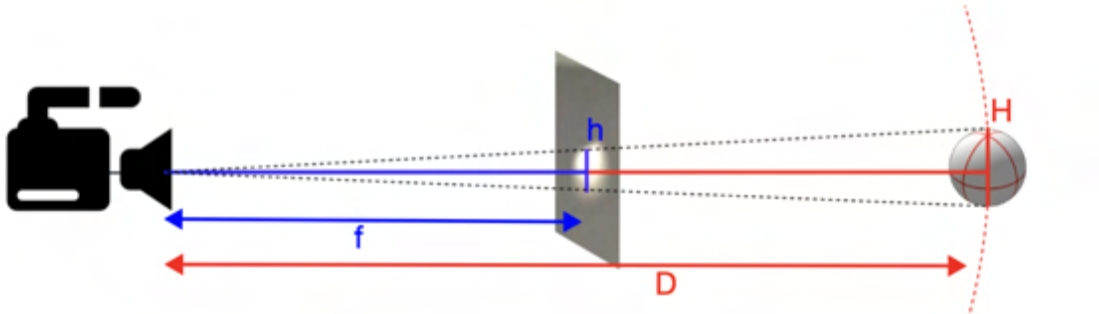


Figure – Zoom sur la balle détectée

Modélisation de la trajectoire de la balle

Passage des coordonnées 2D aux coordonnées 3D

- Pour obtenir la succession des positions 3D de la balle au cours du temps, nous devons obtenir la distance caméra-balle en vision monoculaire.



- Étant donnée H la taille réelle de la balle (soit 4.00 cm), h la taille apparente de la balle dans le domaine image, f la distance focale, et $f_{dev} = \sqrt{(x - x_0)^2 + (y - y_0)^2}$ la distance en pixels dans l'image entre le point principal et le centre de la balle, la distance caméra-balle D est calculée suivant l'homothétie : $D = \frac{H}{h} * \sqrt{f^2 + f_{dev}^2}$

Modélisation de la trajectoire de la balle

Problèmes retrouvés lors de l'étape de la détection

Figure – Exemple de mauvaise détection

Figure – Zoom sur l'aberration

Modélisation de la trajectoire de la balle

Méthode RANSAC

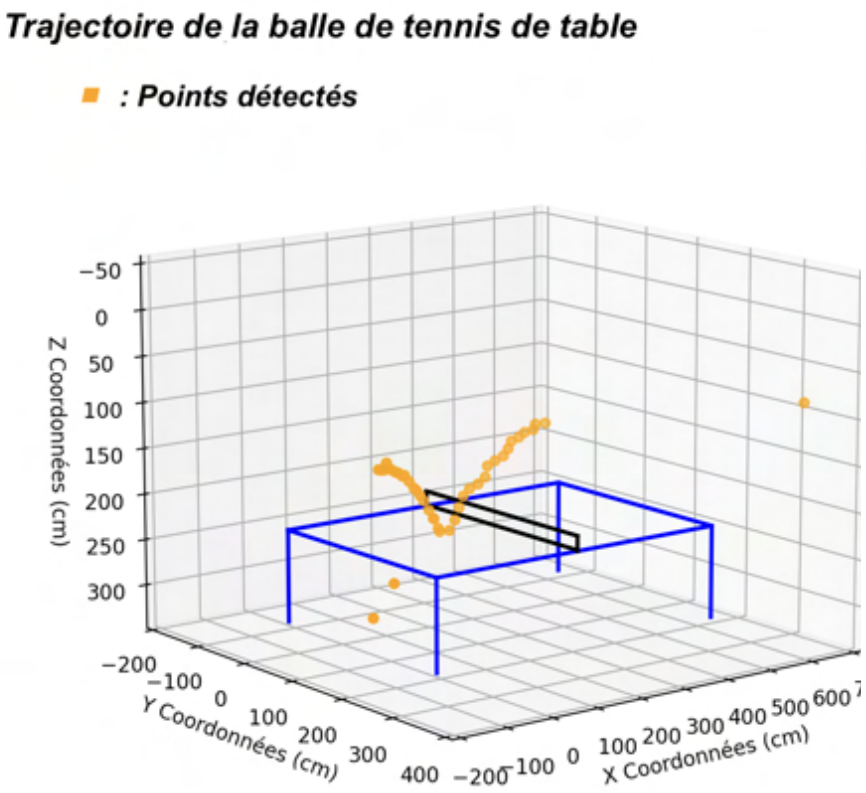
- Pour pallier ce problème, nous optons pour la méthode RANSAC utilisée lorsque l'ensemble de données observées peut contenir des valeurs aberrantes.

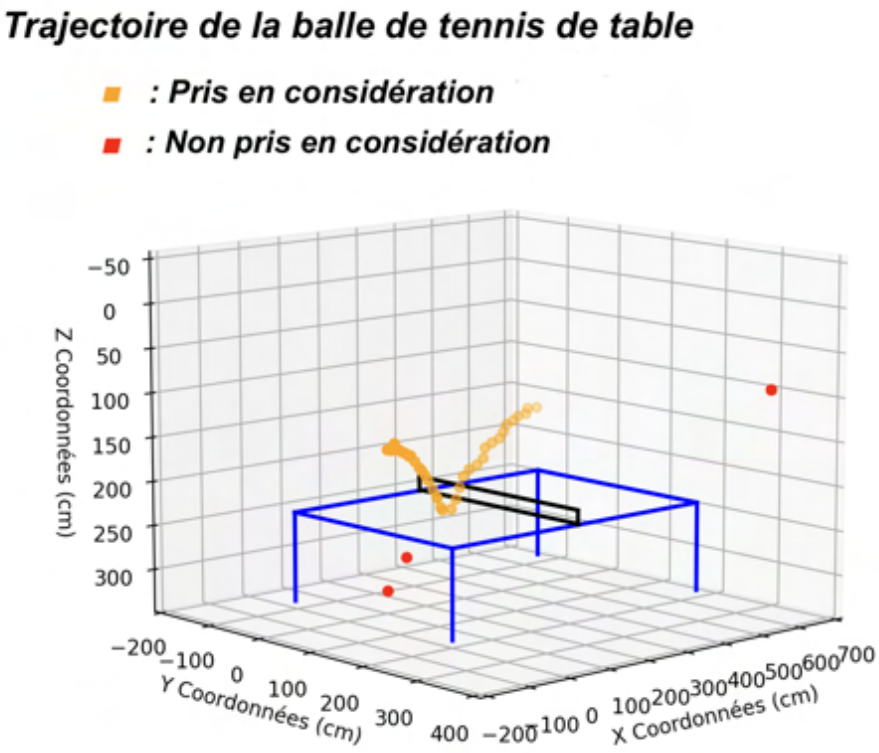
Figure – Un jeu de données avec de nombreuses valeurs aberrantes pour lesquelles une ligne doit être ajustée

Figure – Ligne ajustée avec la méthode RANSAC, les valeurs aberrantes n'ont aucune influence sur le résultat

Modélisation de la trajectoire de la balle

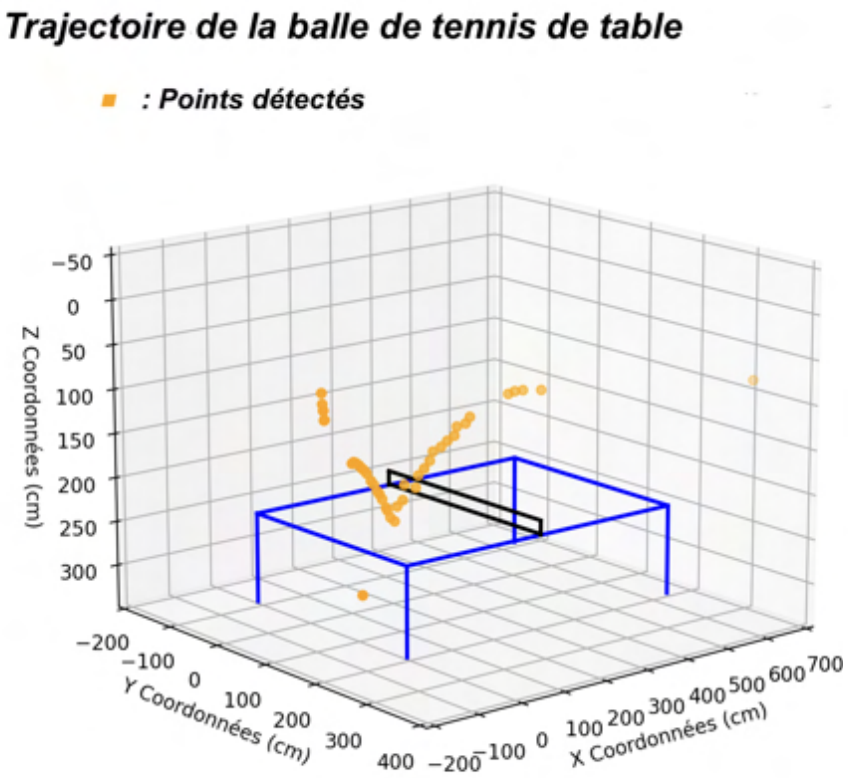
Méthode RANSAC





Modélisation de la trajectoire de la balle

Méthode RANSAC



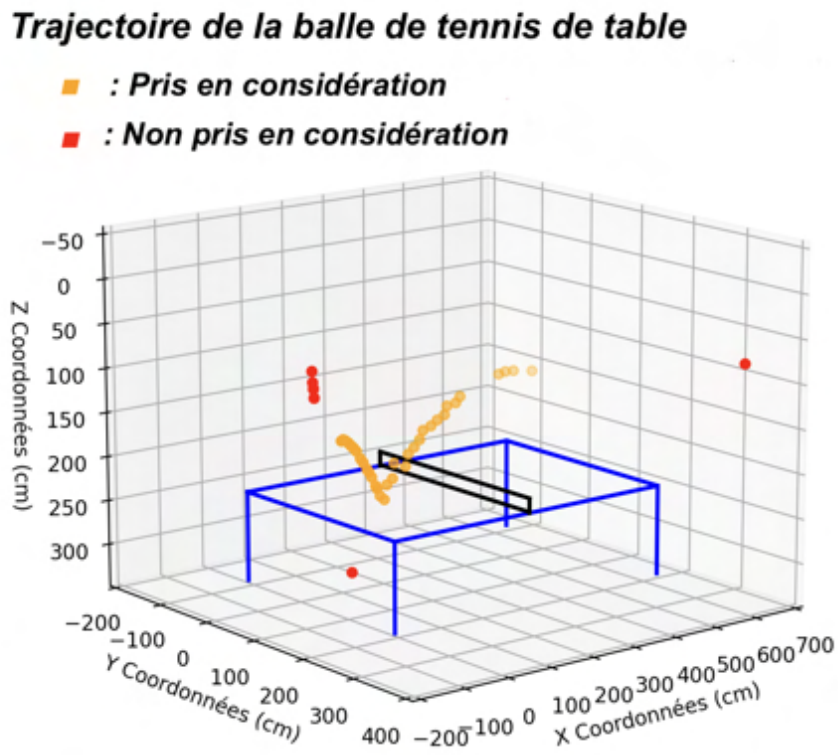


Table des matières

Méthode de la vision stéréoscopique

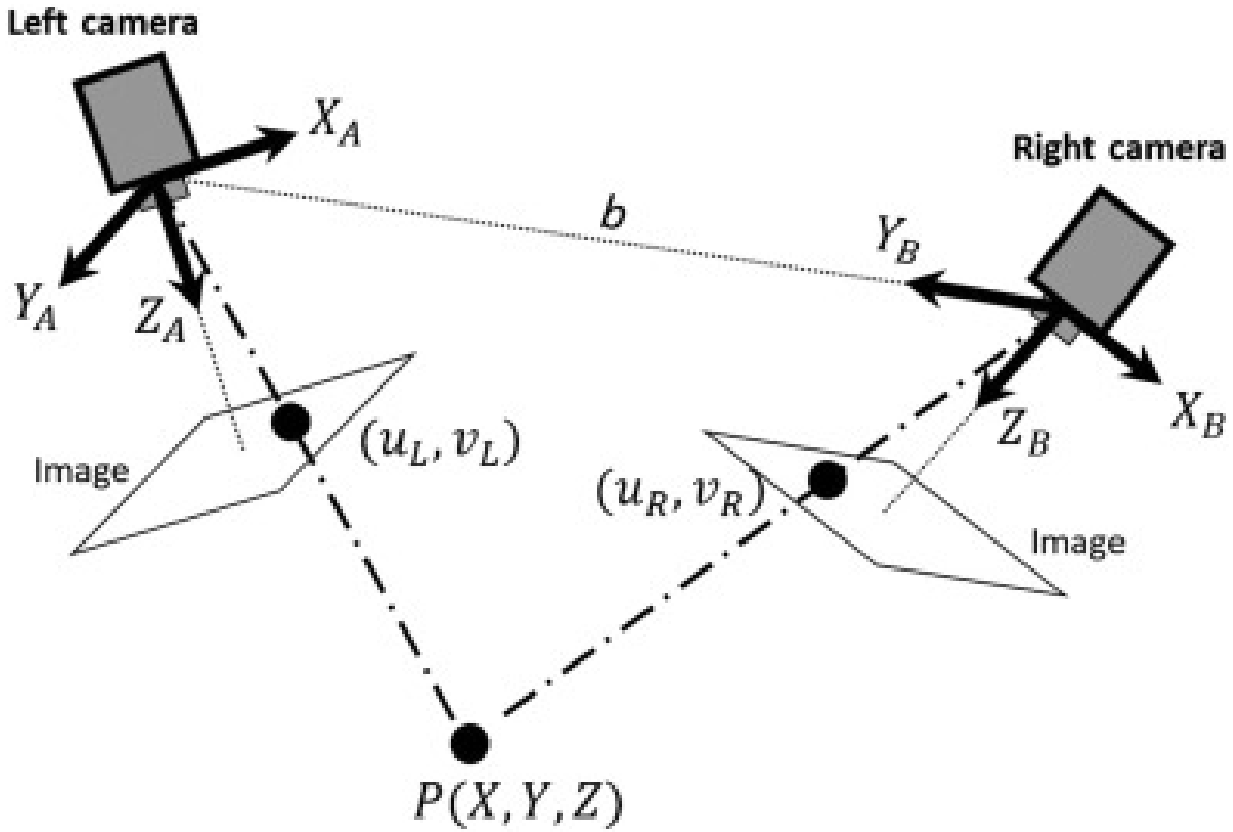


Figure – Principe de la vision stéréoscopique

Méthode de la vision stéréoscopique

1ère étape : Stéréocalibration

Figure – Image de la caméra de gauche

Figure – Image de la caméra de droite

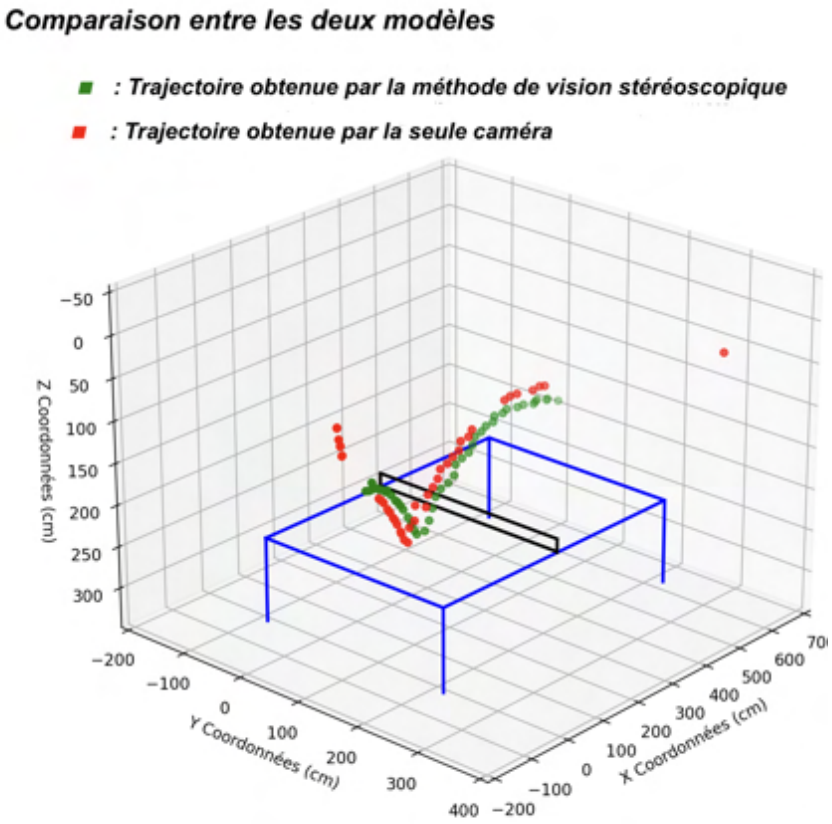
Méthode de la vision stéréoscopique

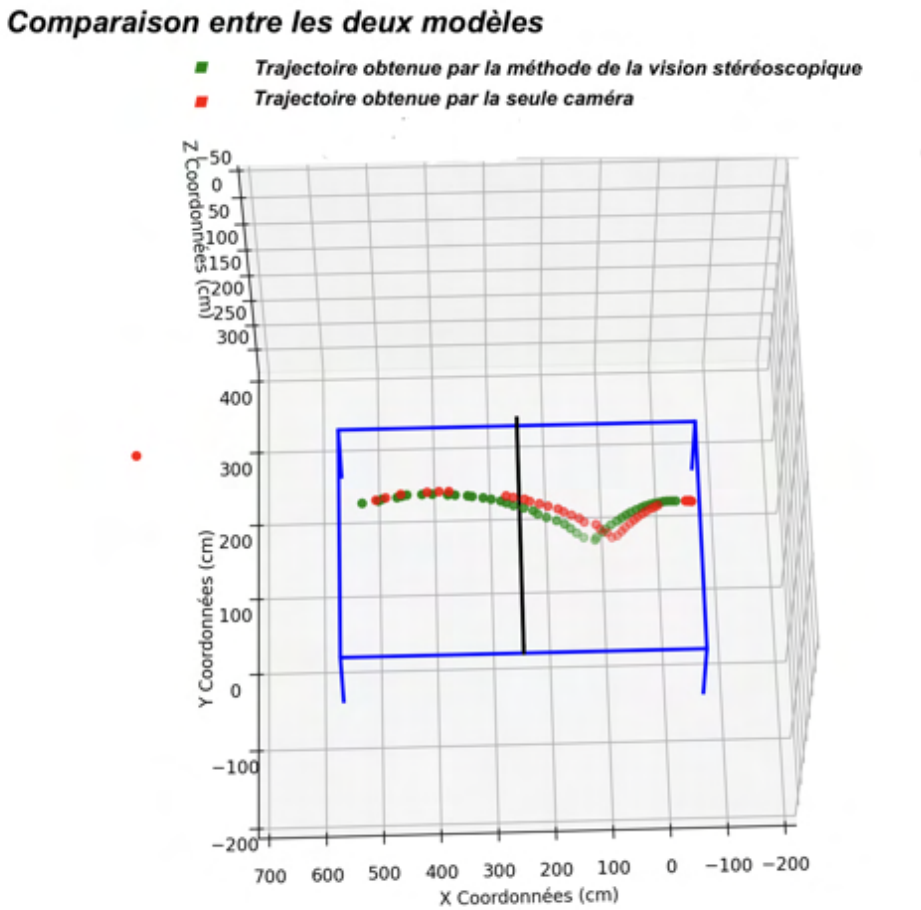
Etapes suivantes

1. **Capture vidéo à l'aide des deux caméras synchronisées**
2. **Rectification des images**
3. **Correspondance Stéréo**
4. **Estimation de la profondeur**
5. **Suivi de la balle**
6. **Extraction de la trajectoire**

Table des matières

Comparaison des deux modèles





Merci pour votre attention

Table des matières

Annexe

Code de calibrage de la caméra

```
import numpy as np
import cv2
import os
import glob
# Définir le nombre de points dans le motif de calibration
num_corners_horizontal = 9  # Nombre de coins selon l'axe
horizontal
num_corners_vertical = 6    # Nombre de coins selon l'axe
vertical

# Préparer les points objets comme (0,0,0), (1,0,0),
(2,0,0) ...., (8,5,0)
object_points = np.zeros((num_corners_horizontal *
num_corners_vertical, 3), np.float32)
object_points[:,:2] = np.mgrid[0:num_corners_horizontal,
0:num_corners_vertical].T.reshape(-1, 2)

# Tableaux pour stocker les points d'objet et les points
d'image de toutes les images d'étalonnage
object_points_list = []  # Points 3D dans l'espace du monde
réel
image_points_list = []   # Points 2D dans le plan image

# Importer les images
#image_files = ['/Users/mac//Desktop//AHMED3//test2.jpg',]
image_files =glob.glob('/Users/mac//Desktop//CALIB/*.jpeg')
# Fonction de prétraitement de l'image
def preprocess_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.equalizeHist(gray)
    return gray

# Lire les images
image = cv2.imread(image_file)
```

Annexe

Code de calibrage de la caméra

```
# Lire les images
image = cv2.imread(image_file)

# Prétraitement de l'image
gray = preprocess_image(image)

# Retrouver les coins de l'échiquier
ret, corners = cv2.findChessboardCorners(gray,
(num_corners_horizontal, num_corners_vertical), None)

    # Si des coins sont trouvés, ajoutez des points d'objet
et des points d'image
    if ret:
        object_points_list.append(object_points)
```

1

```
        image_points_list.append(corners)

        # Dessiner et afficher les coins
        cv2.drawChessboardCorners(image,
(num_corners_horizontal, num_corners_vertical), corners,
ret)
        cv2.imshow('Coins de l\'échéquier', image)
        cv2.waitKey(500)  # Afficher l'image pendant 500
millisecondes
cv2.destroyAllWindows()
```

Annexe

Code de calibrage de la caméra

```
cv2.destroyAllWindows()

# Vérifier que des points ont été trouvés
if len(object_points_list) == 0 or len(image_points_list)
== 0:
    print("Erreur : Aucun point d'objet ou point d'image
n'a été trouvé.")
else:
    # Performer la calibration de la caméra
    ret, camera_matrix, distortion_coeffs, rvecs, tvecs =
cv2.calibrateCamera(
        object_points_list, image_points_list,
gray.shape[::-1], None, None
    )

    # Les paramètres intrinsèques
    print("paramètres intrinsèques (Matrice Camera):")
    print(camera_matrix)

    # Les coefficients de distortion
    print("\nCoefficients de distortion:")
    print(distortion_coeffs)

    # Les paramètres extrinsèques (vecteurs de rotation et
de translation) pour chaque image de calibration
    print("\nparamètres extrinsèques :")
    for i in range(len(rvecs)):
        print(f"Image {i+1}:")
        print("Vecteur Rotation:")
        print(rvecs[i])
        print("Vecteur Translation:")
        print(tvecs[i])
```

Annexe

Code de traçage des courbes (Etude théorique)

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
# Constantes
rh = 1.2 # densité de l'air
A = np.pi * (0.02) ** 2 # Surface de friction de la sphére
Cd = 0.4  # le coefficient de frottement estimé
g = 9.81  # Intensité de la pesanteur
m = 0.0027  # masse de la balle
#omega = 2*np.pi*40 # vitesse angulaire initiale (rad/s)
#omega = 2*np.pi*15
omega = 2*np.pi*(-7)
CL=0.7  # coefficient du lift


# Fonction pour résoudre le système d'équations différentielles
def F(Y, t):
    theta, x, z, dtheta, dx, dz = Y
    V = np.sqrt(dx ** 2 + dz ** 2) # norme de V(Vx,Vz)
    d2x = (-1 / (2 * m)) * rh * A * V ** 2 * (Cd * np.cos(theta) + CL * np.sin(theta))
    d2z = -g - (1 / (2 * m)) * rh * A * V ** 2 * (Cd * np.sin(theta) + CL * np.cos(theta))
    return [dtheta, dx, dz, 0, d2x, d2z]
```

Annexe

Code de traçage des courbes (Etude théorique)

```
# Conditions Initiales
#Y0 = [10 * np.pi / 180, -1.5, 0.25,omega,13, 0.25]
#Y0 = [10 * np.pi / 180, -1.5, 0.25,omega,6.5, 0.3]
Y0 = [10 * np.pi / 180, -1.5, 0.25,omega,3.5, 0.5]

# Tableau de temps
t = np.linspace(0, 0.5, 10000000)

# Résolution du système
Y = odeint(F, Y0, t)
theta, x, z = Y[:,0],Y[:,1],Y[:,2]

# Le tracage de la trajectoire
plt.plot(x, z)
plt.xlabel('x (m)')
plt.ylabel('z (m)')
plt.title('Trajectoire du projectile Poussette')
plt.show()
```

Annexe

Code de détection de la balle

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Paramètres de calibration de la caméra
camera_matrix = np.array([[2841, 0, 1938], [0, 2831, 1477],
[0, 0, 1]], dtype=np.float32)
distortion_coeffs = np.array([0.1, -0.05, 0.001, 0.001,
0.0])

# Paramètres de la caméra et de la balle
focale_mm = 26
taille_balle_mm = 40
taille_capteur_mm = 35
resolution_horizontale = 1920
focale_pixels = (focale_mm / taille_capteur_mm) *
resolution_horizontale


def pixel_to_3d(x_pixel, y_pixel, distance, camera_matrix):
    cx, cy = camera_matrix[0, 2], camera_matrix[1, 2]
    fx, fy = camera_matrix[0, 0], camera_matrix[1, 1]
    x_normalized = (x_pixel - cx) / fx
    y_normalized = (y_pixel - cy) / fy
    z = distance
    return x_normalized * z, y_normalized * z, z

# Capture vidéo
cap = cv2.VideoCapture('//Users//mac//Desktop//
VID_ECHANGE1.MOV')
trajectory_points = []
start_time = 5.05  # Temps de début en secondes
end_time = 5.8  # Temps de fin en secondes
frame_rate = 30
cap.set(cv2.CAP_PROP_POS_MSEC, start_time * 1000)
```

Annexe

Code de détection de la balle

```
while True:
    ret, frame = cap.read()
    if not ret or cap.get(cv2.CAP_PROP_POS_MSEC) >=
end_time * 1000:
        break

    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    lower_orange = np.array([5, 50, 50])
```

1

```
    upper_orange = np.array([15, 255, 255])
    mask = cv2.inRange(hsv, lower_orange, upper_orange)
    mask = cv2.erode(mask, None, iterations=1)
    mask = cv2.dilate(mask, None, iterations=9)
    mask = cv2.GaussianBlur(mask, (9, 9), 0)
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    for contour in contours:
        temp_area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)
        if perimeter == 0:
            continue
        circularity = 4 * np.pi * (temp_area / (perimeter *
perimeter))

        if 500 < temp_area < 6000 and 0.7 < circularity <
1.2:
```

Annexe

Code de détection de la balle

```
            (x, y), radius =
cv2.minEnclosingCircle(contour)
            if radius > 6:
                cv2.circle(frame, (int(x), int(y)),
int(radius), (0, 255, 0), 2)
                cv2.putText(frame, "Balle", (int(x -
radius), int(y - radius)), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 0), 2)

                diametre_pixels = 2 * radius
                distance_cm = (focale_pixels *
taille_balle_mm) / (diametre_pixels * 10)
                cv2.putText(frame, f"Distance:
{distance_cm:.2f} cm", (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 0, 255), 2)

                x_3d, y_3d, z_3d = pixel_to_3d(x, y,
distance_cm, camera_matrix)
                trajectory_points.append((x_3d, y_3d,
z_3d))

                break  # Sortir de la boucle après avoir
trouvé la première balle pour simplifier

    cv2.imshow('Frame', frame)

    if cv2.waitKey(int(1000/frame_rate)) & 0xFF ==
ord('q'):
```

2

Annexe

Code de détection de la balle

```
        break

cap.release()
cv2.destroyAllWindows()

# Tracé de la trajectoire
trajectory_points = np.array(trajectory_points)

# Tracer la trajectoire réelle
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(trajectory_points[:, 0], trajectory_points[:,
2], trajectory_points[:, 1], color='orange', label='Points
détectés')

# Dimensions de la table de tennis de table en cm
table_length = 274
table_width = 152
table_height = 76
net_height = 15.25

# Dessiner la surface de la table
x_table = [250-table_length/2, 250+table_length/2,
250+table_length/2, 250-table_length/2, 250-table_length/2]
z_table = [100-table_width/2, 100-table_width/2,
100+table_width/2, 100+table_width/2, 100-table_width/2]
y_table = [226, 226, 226, 226, 226]
ax.plot(x_table, z_table, y_table, color='blue',
linewidth=2)

# Dessiner les pieds de la table
```

Annexe

Code de détection de la balle

```
y1 = [226, 226+100]
x1 = [250-table_length/2, 250-table_length/2]
z1 = [100-table_width/2, 100-table_width/2]
x2 = [250-table_length/2, 250-table_length/2]
z2 = [100+table_width/2, 100+table_width/2]
x3 = [250+table_length/2, 250+table_length/2]
z3 = [100-table_width/2, 100-table_width/2]
x4 = [250+table_length/2, 250+table_length/2]
z4 = [100+table_width/2, 100+table_width/2]

ax.plot(x1, z1, y1, color='blue', linewidth=2)
ax.plot(x2, z2, y1, color='blue', linewidth=2)
ax.plot(x3, z3, y1, color='blue', linewidth=2)
ax.plot(x4, z4, y1, color='blue', linewidth=2)
```

3

```
# Dessiner le filet
net_z = [100-table_width/2, 100+table_width/2]
net_x = [250, 250]
net_y = [225-net_height, 225-net_height]
net_y2 = [225, 225]
ax.plot(net_x, net_z, net_y, color='black', linewidth=2)
ax.plot(net_x, net_z, net_y2, color='black', linewidth=2)
```

Annexe

Code de détection de la balle

```
# Dessiner les bordures verticales du filet
net_vertical_z1 = [100-table_width/2, 100-table_width/2]
net_vertical_x1 = [250, 250]
net_vertical_y1 = [226-net_height, 226]
net_vertical_z2 = [100+table_width/2, 100+table_width/2]
ax.plot(net_vertical_x1, net_vertical_z1, net_vertical_y1,
color='black', linewidth=2)
ax.plot(net_vertical_x1, net_vertical_z2, net_vertical_y1,
color='black', linewidth=2)

ax.set_xlim([-200, 700])  # Longueur de la table / 2
ax.set_ylim([-200, 400.25])  # Largeur de la table / 2
ax.set_zlim([-50, 340])  # Plage de hauteur raisonnable en
cm pour la trajectoire de la balle
ax.set_xlabel('X Coordonnées (cm)')
ax.set_ylabel('Y Coordonnées (cm)')
ax.set_zlabel('Z Coordonnées (cm)')
plt.title('Trajectoire 3D de la balle de Tennis de Table')
ax.legend(loc='lower left', bbox_to_anchor=(0, 0))  #
Déplacer la légende en bas à gauche en dehors du graphique
plt.legend()
plt.show()
```

Annexe

Code de calibration : méthode vision stéréoscopique

```
import numpy as np
import cv2
import glob

# Définir les dimensions de l'échiquier
chessboard_size = (9, 6)
square_size = 1.0  # Taille d'une case dans l'unité de votre choix
(par ex., mètres)

# Préparer les points d'objets basés sur le motif de l'échiquier
objp = np.zeros((chessboard_size[0]*chessboard_size[1], 3),
np.float32)
objp[:, :2] = np.mgrid[0:chessboard_size[0],
0:chessboard_size[1]].T.reshape(-1, 2)
objp *= square_size

# Tableaux pour stocker les points d'objets et les points d'images de
toutes les images
objpoints = []  # Points 3D dans l'espace réel
imgpoints_left = []  # Points 2D dans le plan image pour la caméra
gauche
imgpoints_right = []  # Points 2D dans le plan image pour la caméra
droite

# Charger les images de calibration pour les deux caméras
images_left = glob.glob('/Users/mac//Desktop//AHMED3//normal/*.jpeg')
images_right = glob.glob('/Users/mac//Desktop//ANAS3//normal/*.jpeg')

# Assurer que les deux répertoires contiennent le même nombre
d'images
assert len(images_left) == len(images_right), "Nombre d'images
différent pour les caméras gauche et droite."
```

Annexe

Code de calibration : méthode vision stéréoscopique

```
    # Trouver les coins de l'échiquier
    retL, cornersL = cv2.findChessboardCorners(grayL,
chessboard_size, None)
    retR, cornersR = cv2.findChessboardCorners(grayR,
chessboard_size, None)

    # Si trouvé, ajouter les points d'objets et les points d'images
(après les avoir affinés)
    if retL and retR:
        objpoints.append(objp)

        criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
        cornersL = cv2.cornerSubPix(grayL, cornersL, (11, 11), (-1,
-1), criteria)
```

1

```
        imgpoints_left.append(cornersL)

        cornersR = cv2.cornerSubPix(grayR, cornersR, (11, 11), (-1,
-1), criteria)
        imgpoints_right.append(cornersR)
```

Annexe

Code de calibration : méthode vision stéréoscopique

```
        cv2.waitKey(100)

cv2.destroyAllWindows()

# Calibration de la caméra individuelle
retL, mtxL, distL, rvecsL, tvecsL = cv2.calibrateCamera(objpoints,
imgpoints_left, grayL.shape[::-1], None, None)
retR, mtxR, distR, rvecsR, tvecsR = cv2.calibrateCamera(objpoints,
imgpoints_right, grayR.shape[::-1], None, None)

# Calibration stéréo
flags = cv2.CALIB_FIX_INTRINSIC
criteria_stereo = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 1e-6)
retS, mtxL, distL, mtxR, distR, R, T, E, F = cv2.stereoCalibrate(
    objpoints, imgpoints_left, imgpoints_right,
    mtxL, distL, mtxR, distR, grayL.shape[::-1],
    criteria=criteria_stereo, flags=flags
)

# Rectification stéréo
R1, R2, P1, P2, Q, roi1, roi2 = cv2.stereoRectify(
    mtxL, distL, mtxR, distR, grayL.shape[::-1], R, T, alpha=0
)

# Sauvegarder les résultats de calibration
np.savez('stereo_calibration.npz', mtxL=mtxL, distL=distL, mtxR=mtxR,
distR=distR, R=R, T=T, E=E, F=F, R1=R1, R2=R2, P1=P1, P2=P2, Q=Q)

print("Calibration stéréo terminée et résultats sauvegardés.")
```