Élaboration d'une alternative viable au jugement visuel (par définition) arbitraire des arbitres

Épreuve de TIPE

Session 2024 -1 10975

Jeux, Sports

Table des matières

Nécessité de l'étude

- Arbitrage
- Étude des sports

Problématique

Comment déterminer la trajectoire de la balle lors de compétitions sportives?

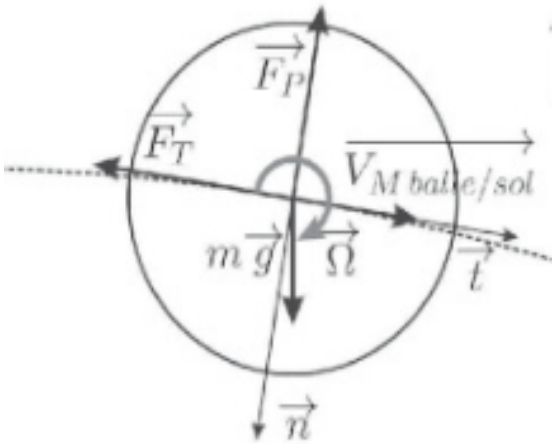
Problématique

Comment déterminer la trajectoire de la balle lors de compétitions sportives?

- Étude théorique
- VAR #2
- Méthode ondulatoire
- Conciliation des méthodes
- Correction des erreurs
- Conclusion

Étude théorique

Mise en équation



Méthode SI: Ballon

BAM: $\vec{P} = -mg\vec{e_z}$

$\vec{F_P} = \frac{1}{2}\mu A C_L v(t) w(t) \vec{e_{z'}}$

$\vec{F_T} = \frac{1}{2}\mu A C_X S v(t) w(t) \vec{e_{x'}}$

$w : t \to w_0 e^{-t/\tau}$

Figure 1: Forces exercées

Mise en équation:

$$\frac{d^2x}{dt^2} = K_2 \frac{dx}{dt} w(t) cos(\alpha(t)) sin(\beta(t)) - K_1 (\frac{dx}{dt})^2 cos(\alpha(t)) cos(\beta(t))$$

$$\frac{d^2y}{dt^2} = K_2 \frac{dy}{dt} w(t) sin(\alpha(t)) sin(\beta(t)) + K_1 (\frac{dy}{dt})^2 sin(\alpha(t)) cos(\beta(t))$$

$$\frac{d^2z}{dt^2} = K_2 \frac{dz}{dt} w(t) cos(\beta(t)) - K_1 (\frac{dz}{dt})^2 sin(\beta(t)) - g$$

$$\alpha(t) = arctan(\frac{|\frac{dy}{dt}|}{|\frac{dx}{dt}|}); \ \beta(t) = arctan(\frac{|\frac{dx}{dt}|}{|\frac{dz}{dt}|})$$

Étude théorique

Coefficients

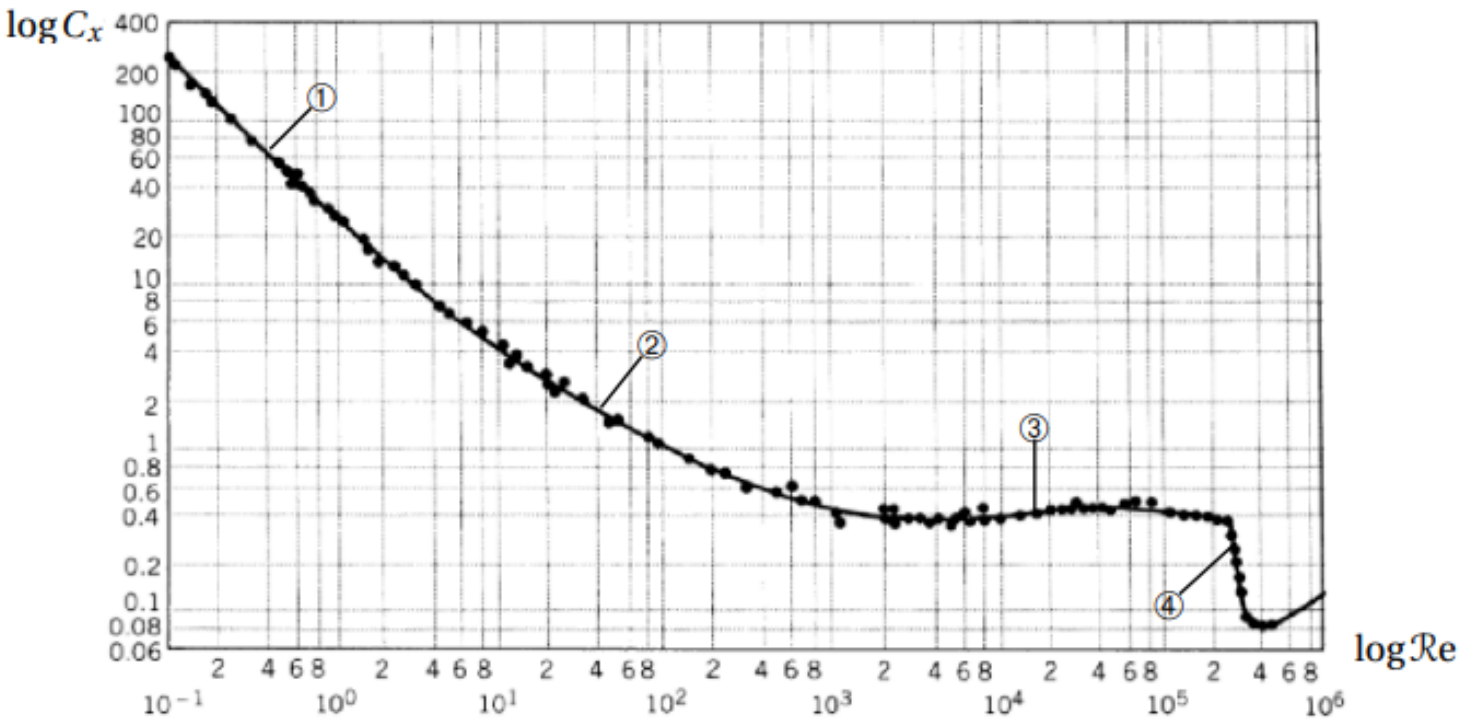


Figure 2: Cx(Re)

Étude théorique

Limites du modèle

Méthode d'Euler + scipy

```
atan_yx = np.arctan2(np.abs(y_pos - y_prev), np.abs(x - x_prev))
atan_xz = np.arctan2(np.abs(x - x_prev), np.abs(z - z_prev))

dvx_dt = -K1 * vx**2 * np.cos(atan_yx) * np.cos(atan_xz) + K2 * vx * np.cos(atan_yx) * np.sin(atan_xz) * w_t
dvy_dt = K1 * vy**2 * np.sin(atan_yx) * np.cos(atan_xz) + K2 * vx * np.sin(atan_yx) * np.sin(atan_xz) * w_t
dvz_dt = -K1 * vz**2 * np.sin(atan_xz) + K2 * vz * w_t * np.cos(atan_xz) - g
```

```
x_sol = solution.y[0]
y_sol = solution.y[1]
z_sol = solution.y[2]
```

Figure 3: Code 1

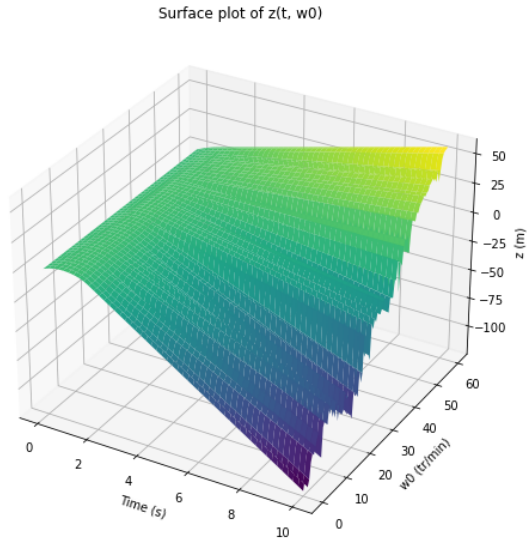


Figure 4: Trajectoires possibles

Trilatération

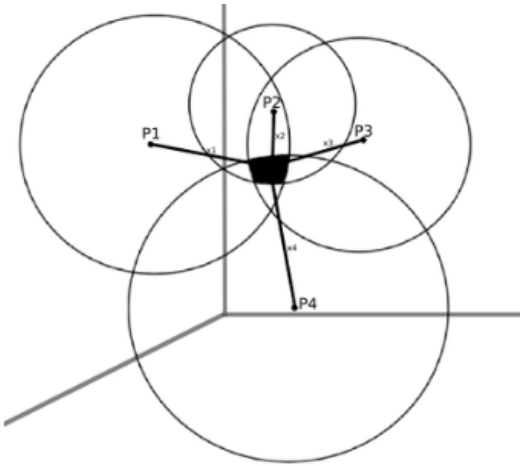


Figure 5: Principe

```
def residus(p, positions, distances, incertitudes):
    return ((np.sqrt(np.sum((positions - p)**2, axis=1)) - distances) / incertitudes)

x0 = np.average(positions, axis=0, weights=1.0 / incertitudes**2)

res = least_squares(residus, x0, args=(positions, distances, incertitudes))

return res.x
```

Figure 6: Code

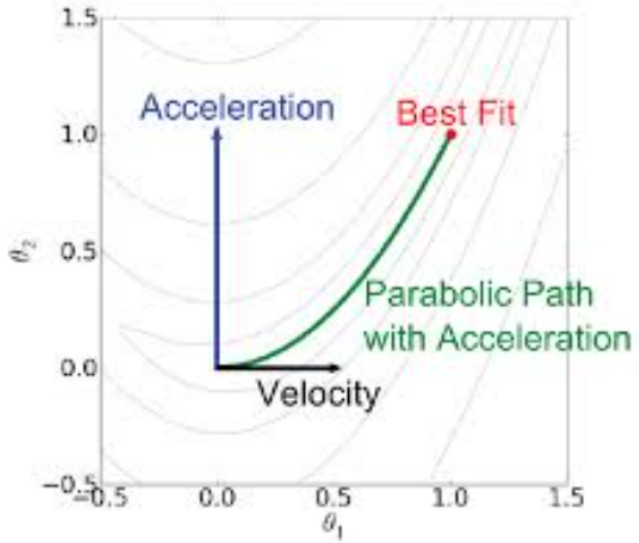
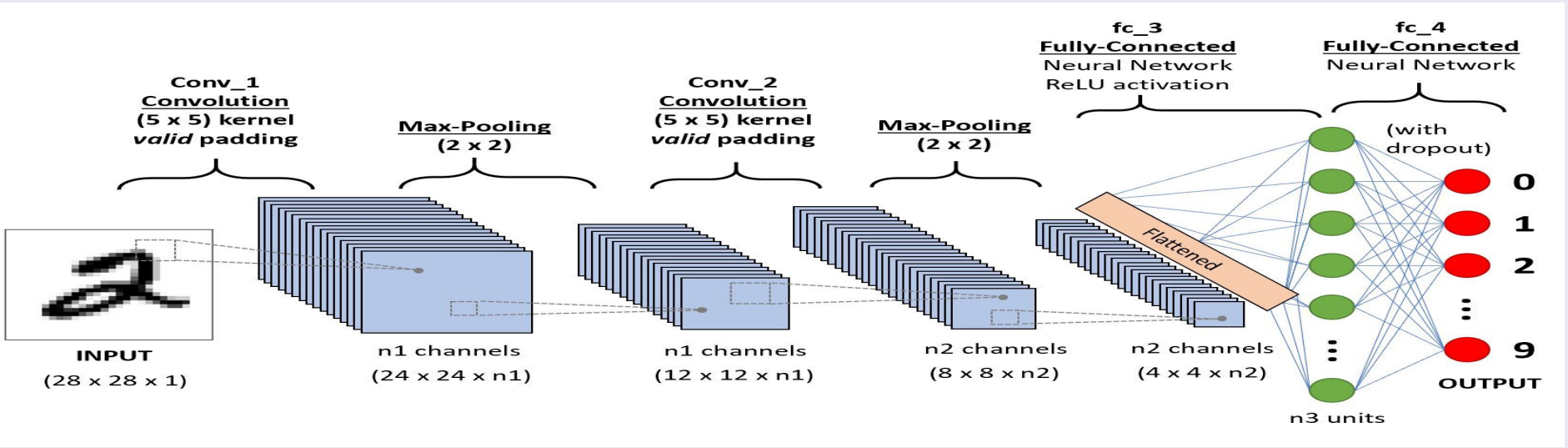


Figure 7: Principe

Architecture



- Vitesse
- Préentraîné
- $>$ Dirichlet

VAR 2

IA

```
if etiquette == 'sports ball' and (confiance > 0.7 and abs(x1-x2)*95/100 <= abs(y1-y2) <= abs(x1-x2)*105/100):
    Res_dist.append(abs(y1-y2))
```

Figure 8: Code

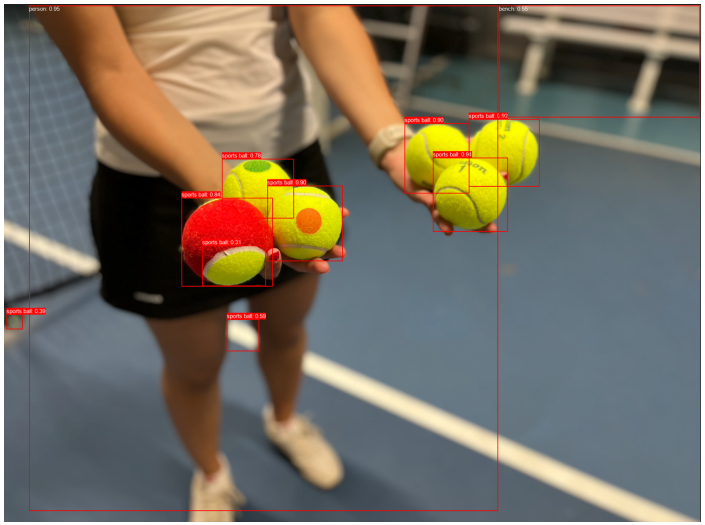


Figure 9: Test

Méthode ondulatoire

Principe

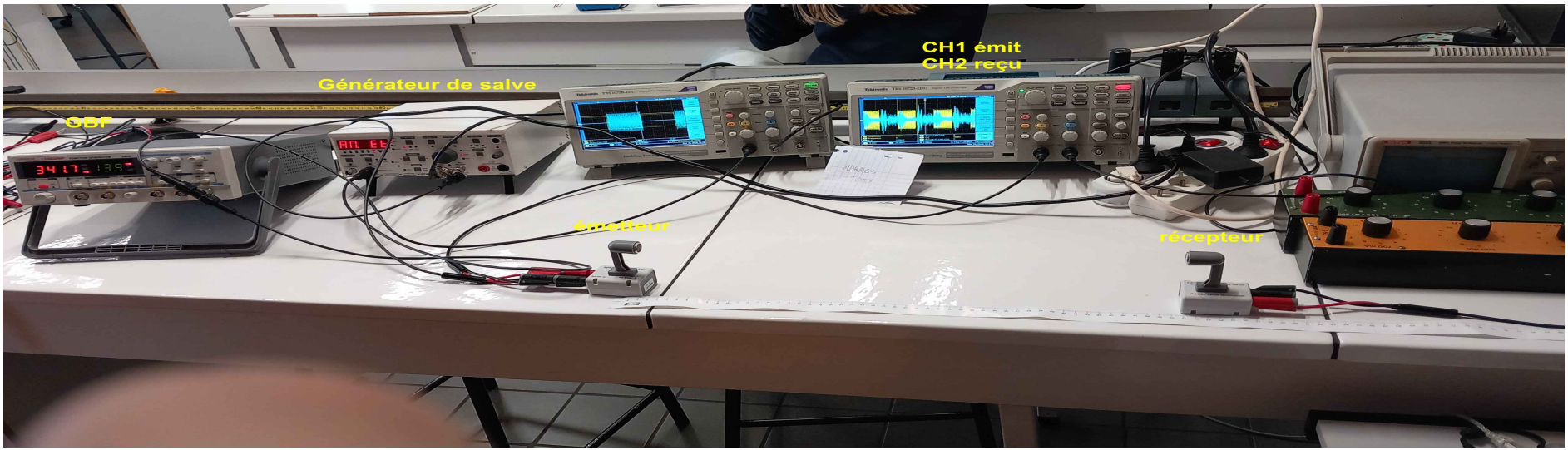


Figure 10: Montage 1

- Intérêt du générateur de salve
- Mise en place réelle

Principe

Redressement $\longrightarrow$ *Variations* $\longrightarrow$ *Pics*
Pont de diode $\longrightarrow$ *Passe – bas actif* $\longrightarrow$ *Montage Dérivateur*
Valeur absolue $\longrightarrow$ *Passe – bas numérique* $\longrightarrow$ *Seuil*

Paramètres du filtre

Gain

$\rightarrow$ *Sans intérêt*

fréquence de coupure

$\rightarrow$ *Critère de Shanon et tatonage*

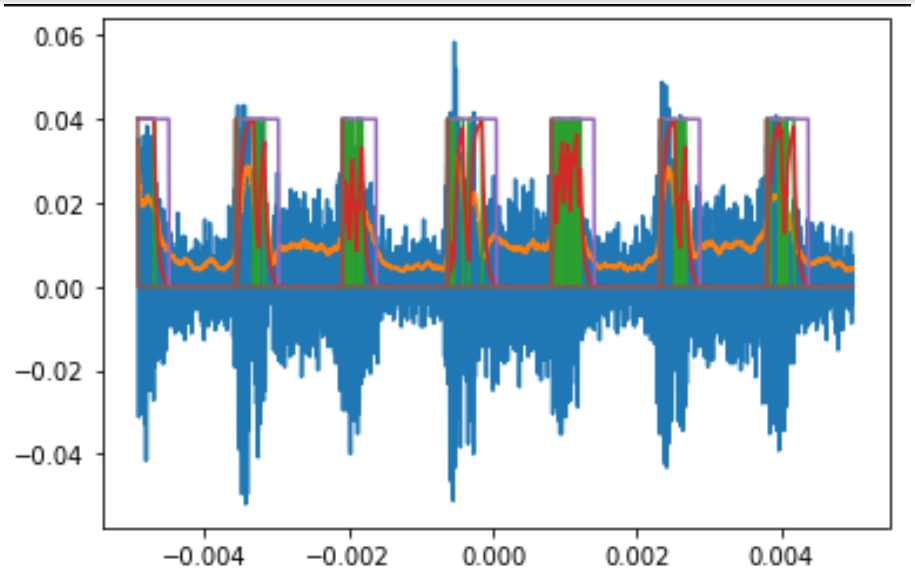


Figure 11: Précision: 95 %

Méthode ondulatoire

Alternative plus rapide

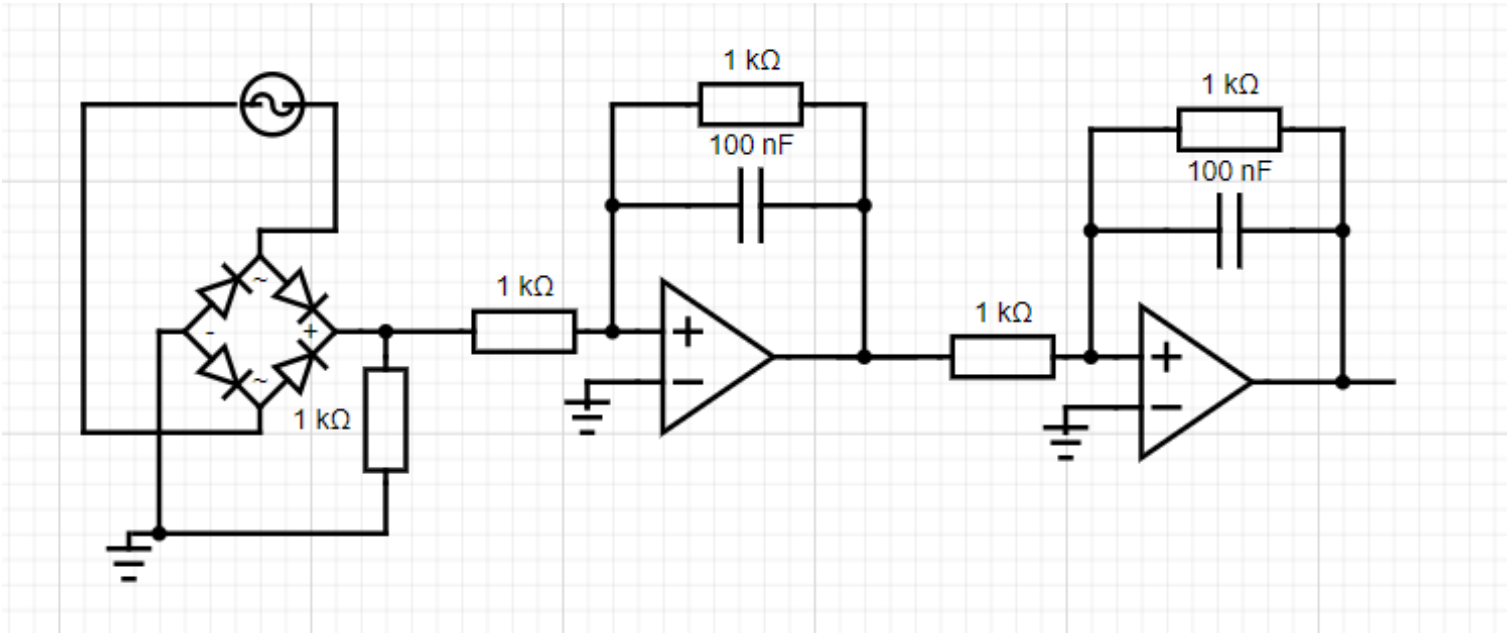


Figure 12: Circuit

Cumulation des méthodes

Comparaison

Théorie
- Imprécis
- Instantané

VAR
- Précis
- Lent

Onde
- $\pm$*Précis*
- $\pm$*Rapide*

KNN

Résultats: 61 CSV

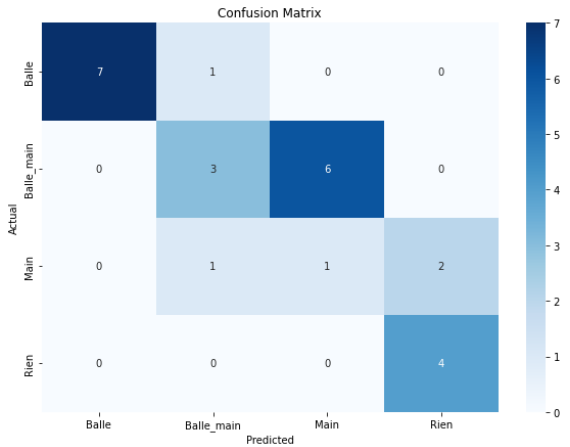


Figure 13: Général
Précision: 60%

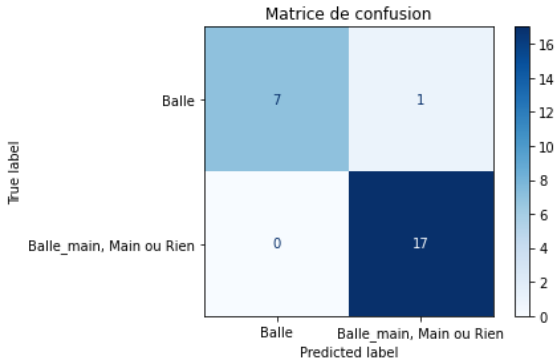


Figure 14: Soldat
Précision: 95%

```
for i in range(len(y_pred)):
    if probas[i, 0] < thresh:
        y_pred[i] = 3
```

Figure 15: Code

Conclusion

Mise en place

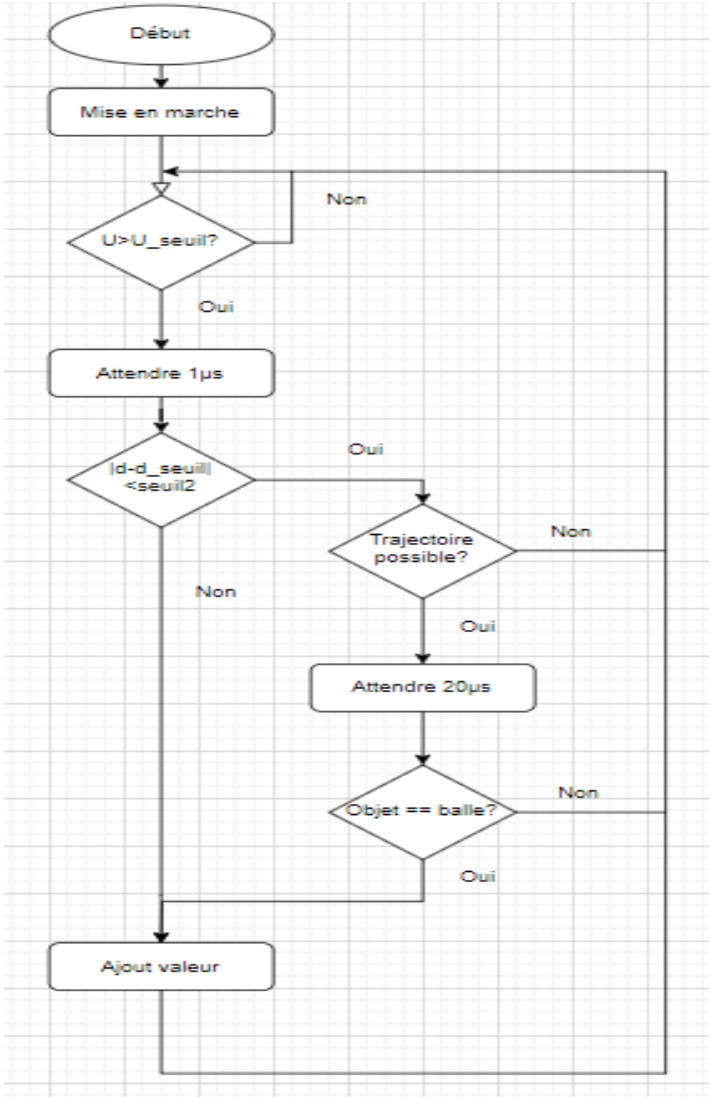


Figure 16: Fonctionnement

Conclusion

Résultats

Résultats	Limites
Efficace	Eléments extérieurs
Rapide	Objets externes
Peu coûteux	X
Envisageable techniquement	X

Figure 16: Tableau conclusif

Annexe

Extraction

```python
def read_csv_files(folder_path):
    signals = []
    for root, dirs, files in os.walk(folder_path):
        for file_name in files:
            if file_name.endswith('.CSV'):
                file_path = os.path.join(root, file_name)
                try:
                    with open(file_path, 'r') as file:
                        reader = csv.reader(file)
                        signal = []
                        for i, row in enumerate(reader):
                            if i >= 18:  # Commence à lire à partir de la ligne 19
                                if row and len(row) > 4 and row[4].strip():  # Vér
                                    try:
                                        amplitude = float(row[4].strip())
                                        signal.append(amplitude)
                                    except ValueError:
                                        pass
                        if signal:  # Ajouter le signal uniquement s'il n'est pas
                            signals.append(np.array(signal))
                except Exception as e:
                    print(f"Error reading {file_path}: {e}")
    return signals

# Charger les données et les étiquettes
```

Annexe

Importation

```python
from PIL import Image, ImageDraw, ImageFont
from ultralytics import YOLO

# Charger le modèle YOLO
modele = YOLO('yolov8m.pt')

# Chemin de l'image
chemin_image = 'C:/Users/Tommy/Desktop/Tennispallon-valinta2.png'

# Charger l'image
image = Image.open(chemin_image)

# Effectuer une prédiction
resultats = modele.predict(source=chemin_image)

# Créer un objet ImageDraw
dessin = ImageDraw.Draw(image)

# Charger une police (optionnel)
try:
    police = ImageFont.truetype("arial.ttf", 15)
except IOError:
    police = ImageFont.load_default()

# Parcourir les résultats et dessiner les boîtes de délimitation
for resultat in resultats:
    for boite, score, cls in zip(resultat.boxes.xyxy, resultat.boxes.conf, resultat.boxes.cls):
        x1, y1, x2, y2 = map(int, boite)
        etiquette = modele.names[int(cls)]
        confiance = score

        dessin.rectangle([x1, y1, x2, y2], outline="red", width=2)
        texte = f'{etiquette}: {confiance:.2f}'
        taille_texte = dessin.textsize(texte, police)
        emplacement_texte = (x1, y1 - taille_texte[1] if y1 - taille_texte[1] > 0 else y1)
        dessin.rectangle([emplacement_texte, (x1 + taille_texte[0], y1)], fill="red")
        dessin.text(emplacement_texte, texte, fill="white", font=police)

# Sauvegarder l'image avec les boîtes de délimitation
chemin_sortie = 'C:/Users/Tommy/Desktop/Tennispallon-valinta2_output.png'
image.save(chemin_sortie)

# Afficher l'image (optionnel)
image.show()
```

Annexe

Recherche

```python
if data.size > 0:
    max_length = max(len(signal) for signal in data)
    data = np.array([np.pad(signal, (0, max_length - len(signal)), 'constant') for signal in data])

    # Diviser les données pour l'entraînement et le test
    X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

    # Créer et entraîner le modèle KNN
    knn = KNeighborsClassifier(n_neighbors=3)  # Choisissez n_neighbors selon vos besoins
    knn.fit(X_train, y_train)

    # Prédire sur les données de test
    y_pred = knn.predict(X_test)

    # Évaluer le modèle
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy * 100:.2f}%')

    # Prédire une nouvelle donnée
    def predict_new_signal(signal):
        signal = np.pad(signal, (0, max_length - len(signal)), 'constant')
        signal = signal.reshape(1, -1)
        prediction = knn.predict(signal)
        for key, value in labels.items():
            if value == prediction:
                return key

    # Exemple d'utilisation pour prédire un nouveau signal
    # new_signal = np.array([...])  # Remplacer par les données du nouveau signal
    # print(predict_new_signal(new_signal))
else:
    print("Data array is empty. Exiting...")
```

Annexe

Calcul

```python
class SystemWithMemory:
    def __init__(self, initial_conditions, w0):
        self.prev_positions = np.array(initial_conditions[:3])  # Store initial positions
        self.w0 = w0

    def system(self, t, y):
        x, y_pos, z, vx, vy, vz = y

        # Get previous positions
        x_prev, y_prev, z_prev = self.prev_positions

        w_t = w(t, self.w0)
        atan_yx = np.arctan2(np.abs(y_pos - y_prev), np.abs(x - x_prev))
        atan_xz = np.arctan2(np.abs(x - x_prev), np.abs(z - z_prev))

        dvx_dt = -K1 * vx**2 * np.cos(atan_yx) * np.cos(atan_xz) + K2 * vx * np.cos(atan_yx) * np.sin(atan_xz) * w_t
        dvy_dt = K1 * vy**2 * np.sin(atan_yx) * np.cos(atan_xz) + K2 * vx * np.sin(atan_yx) * np.sin(atan_xz) * w_t
        dvz_dt = -K1 * vz**2 * np.sin(atan_xz) + K2 * vz * w_t * np.cos(atan_xz) - g

        # Update previous positions
        self.prev_positions = np.array([x, y_pos, z])

        return [vx, vy, vz, dvx_dt, dvy_dt, dvz_dt]
```

Table des matières

Table des matières

Table des matières

Table des matières

Table des matières