
# TP Estimation de Pose 3D par Transfert Learning avec ResNet50.

**CAIPA PRIETO Julian Andres**

École Nationale Supérieure des Arts et Métiers

Computer Vision

Enseignant : Fakhreddine Ababsa

PA10

Campus de Paris, France

2024

# Table des matières

# Table des figures

# Table des codes

# 1 Introduction

Dans ce travail pratique, nous abordons l'estimation de la pose 3D, une problématique clé en vision par ordinateur, particulièrement utile dans les domaines de la localisation et de la navigation. Ce projet met en œuvre une approche par transfert d'apprentissage (Transfer Learning) en utilisant ResNet50, un modèle pré-entraîné, afin d'estimer la pose d'une caméra (position et orientation) à partir d'images issues du jeu de données Office de 7-Scenes.

Le processus suit une méthodologie incrémentale : préparation des données, configuration et entraînement du modèle, prédiction et évaluation des erreurs. Chaque étape vise à renforcer la compréhension des concepts fondamentaux de l'estimation de pose et à analyser les performances du modèle. Les statistiques des erreurs seront explorées et visualisées à travers des outils graphiques (histogrammes, boxplots), et un modèle de mélange de Gaussiennes sera utilisé pour approcher la distribution des erreurs de pose.

L'objectif final est de fournir une compréhension approfondie de l'estimation de pose 3D tout en exploitant les avantages du transfert d'apprentissage pour traiter des problèmes complexes avec des données limitées.

# 2 Objectifs

— Apprendre les bases de l'estimation de pose 3D.
— Comprendre le concept de transfert learning en utilisant ResNet50 pour la régression.
— Évaluer la précision des estimations de pose et analyser les erreurs.
— Utiliser des outils statistiques et des visualisations (histogrammes, boxplots).
— Appliquer des modèles de mélange de Gaussiennes pour approximer les erreurs de pose.

# 3 Contextualisation

L'estimation de pose 3D est essentielle dans la vision par ordinateur, en particulier pour la localisation et la navigation. Le but de ce TP est d'apprendre à estimer la pose d'une caméra (position et orientation) en utilisant un modèle pré-entraîné (ResNet50) sur le jeu de données Office de 7-Scenes (Figure 1). Une démarche incrémentale sera adoptée, depuis la préparation des données à l'analyse des erreurs de pose, en passant par l'entraînement et l'évaluation du modèle (2).

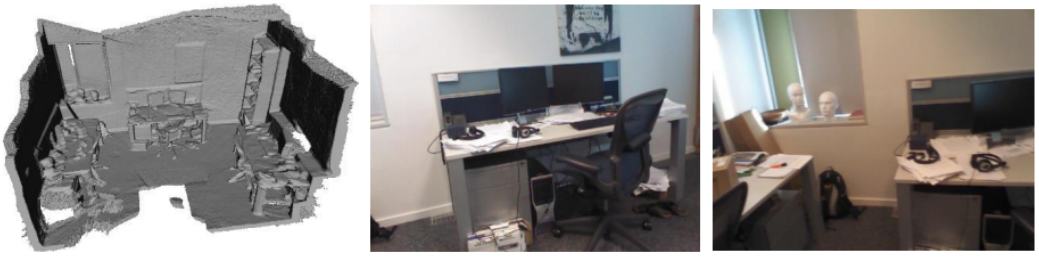

FIGURE 1 – Exemples de la BD Office – 7-Scenes (2).

La dynamique du TP sera d'abord d'analyser tout le travail et de répondre aux questions posées (2) avec une petite base de données d'images simple, puis après avoir analysé la performance du code proposé et les capacités du réseau à estimer la pose, des bases de données plus grandes seront planifiées pour fournir des modèles plus robustes, et le surentraînement et le meilleur obtenu entre la sélection originale et l'augmentation donnée seront analysés. La base de données initiale tient compte des recommandations fournies par le «Office de 7-Scènes», où il est mentionné que les séquences 1, 3, 4, 5, 8 et 10 sont destinées à l'entraînement et les séquences 2, 6, 7 et 9 à la validation. Par conséquent, pour une petite base d'images de départ, la séquence 1 est utilisée pour l'entraînement et la séquence 2 pour la validation, chacune avec 1000 données composites (c'est-à-dire 1000 images et 1000 fichiers de pose).

# 4 Analyse de base et réponse aux questions

## 4.1 Étape 1 : Préparation des données

But de l'étape : Chargement des données d'entraînement et de test depuis le jeu de données Office de 7-Scènes.

À cette étape, les données sont simplement chargées à partir des dossiers de séquences et préparées pour le travail. Comme mentionné ci-dessus, pour le développement initial du travail et des questions, seules les premières et deuxièmes séquences sont travaillées.

Une fonction est créée, dans laquelle un cycle est généré pour parcourir le dossier et stocker dans des tableaux les informations relatives aux images et aux poses de la caméra. Les matrices de pose sont interprétées à partir du fichier .txt comme des matrices 4x4, puis les valeurs d'orientation et de position sont extraites comme les valeurs associées respectives, en interprétant la matrice 4x4 comme une matrice de transformation standard, c'est-à-dire une matrice de bloc avec une orientation 3x3 et un vecteur de colonne 3x1 comme vecteur de position.

```
def load_pose_data(folder):
    images = []
    poses = []

    for file in sorted(os.listdir(folder)):
        if file.endswith(".pose.txt"):
            pose_path = os.path.join(folder, file)
            pose_matrix = np.loadtxt(pose_path).reshape((4, 4))

            position = pose_matrix[:3, 3]
            rotation = pose_matrix[:3, :3].flatten()

            pose_vector = np.concatenate((position, rotation))
            poses.append(pose_vector)

            img_name = file.replace(".pose.txt", ".color.png")
            img_path = os.path.join(folder, img_name)
            images.append(img_path)

    return images, np.array(poses)
```

Code 1 – Préparation des données.

**Questions :**

- **Comment sont structurées les matrices de pose dans 7-Scenes, et que représentent les trois premières lignes et colonnes ?** Les matrices de pose dans 7-Scenes sont des matrices de transformation homogènes 4×4. Ces matrices décrivent la pose de la caméra dans une scène 3D, c'est-à-dire sa position et son orientation, et sont construites comme des matrices de blocs.
  - Les trois premières lignes et colonnes $(33)$ forment la matrice de rotation, qui définit l'orientation de la caméra par rapport au système de coordonnées du monde.
  - La quatrième colonne contient le vecteur de translation $(X, Y, Z)$, qui indique la position de la caméra dans l'espace 3D.
  - La dernière ligne est $[0, 0, 0, 1]$, utilisée pour maintenir l'homogénéité des matrices pour les transformations.

    Enfin, la matrice de blocs se présente comme suit :

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{1}$$

- **Pourquoi aplatissons-nous la matrice de rotation ?** Aplanir la matrice de rotation simplifie sa représentation sous forme de vecteur, qui peut être traité plus facilement par les modèles d'apprentissage.

  La matrice de rotation $R$ a des dimensions 3×3, ce qui signifie qu'elle contient 9 valeurs. En l'aplatissant, nous la transformons en un vecteur unidimensionnel de taille 9 $(R_{11}, R_{12}, \ldots, R_{33})$. Après, en concaténant ce vecteur avec le vecteur de translation $(X, Y, Z)$, nous obtenons un vecteur de pose de 12 dimensions qui combine la position et l'orientation.

## 4.2 Étape 2 : Configuration de ResNet50 pour l'estimation de pose

But de l'étape : Adaptation du modèle ResNet50 pour effectuer une régression en modifiant les couches finales.

L'idée de cette étape est de charger le réseau neuronal ResNet50, un réseau neuronal profond préentraîné sur le jeu de données ImageNet, et d'utiliser le concept de Transfer Learning pour l'adapter à la tâche de régression. Le code charge simplement le réseau depuis Keras, en retirant la couche de classification supérieure, puis une nouvelle couche est générée comme sortie du réseau, de type dense avec 12 sorties. Le modèle est créé en superposant ces couches, puis compilé, en utilisant la fonction de perte MSE, ainsi que la métrique MAE et l'optimiseur ADAM.

```
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))

x = base_model.output
x = Flatten()(x)
output_layer = Dense(12, activation='linear')(x)

model = Model(inputs=base_model.input, outputs=output_layer)

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
```

Code 2 – Configuration ResNet50.

Il faut prendre en compte les considérations suivantes pour mieux comprendre le fonctionnement du code. Dans la base_model, les poids et la forme de l'entrée sont spécifiés (224x224 pixels avec 3 canaux RGB). La couche de sortie utilise une activation linéaire, car des prédictions continues sont attendues, sans plage spécifique. En ce qui concerne la compilation, ADAM est utilisé comme optimiseur (Adaptive Moment Estimation), car il ajuste dynamiquement le taux d'apprentissage pour chaque poids en fonction du gradient moyen accumulé et de l'amplitude du gradient le plus récent. Il converge rapidement et gère bien les gradients bruyants et petits. La fonction de perte utilisée est MSE (Mean Squared Error), idéale pour les tâches de régression où l'on souhaite minimiser les grandes erreurs dans des valeurs continues, car elle pénalise de manière disproportionnée les erreurs les plus grandes, aidant ainsi le modèle à se concentrer sur elles. Enfin, comme métrique, MAE (Mean Absolute Error) est utilisé, car c'est la plus intuitive à comprendre, il s'agit simplement d'une différence moyenne absolue et elle est facile à analyser à travers le modèle.

**Questions :**

- **Pourquoi utilise-t-on le transfert Learning ici ? Quels sont les avantages ?** Nous utilisons le Transfer Learning car ResNet50 a été préentraîné sur un grand ensemble de données (ImageNet) qui possède des caractéristiques génériques utiles pour analyser des images. Les avantages sont les suivants :
  - Gain de temps et de ressources : Nous n'avons pas besoin de former le réseau depuis zéro.
  - Meilleures performances : Les caractéristiques apprises sur ImageNet (comme les bords, les textures) sont réutilisées, améliorant la précision même avec un ensemble de données limité.
  - Convergence rapide : L'entraînement est plus rapide car nous ajustons uniquement les couches finales.
- **Pourquoi modifie-t-on la dernière couche pour obtenir 12 sorties ?** Le modèle original de ResNet50 est conçu pour la classification (1000 catégories sur ImageNet), c'est pourquoi il a une couche dense finale avec 1000 sorties. Pour notre cas, nous avons besoin de 12 sorties pour représenter le vecteur de pose, 3 valeurs pour la position $(X, Y, Z)$ et 9 valeurs pour la rotation (matrice aplatie $R$ de 3×3).

## 4.3 Étape 3 : Entraînement du modèle

But de l'étape : Entraînement du modèle sur les données d'entraînement et valider sur les données de test.

Dans cette étape, l'entraînement du modèle est effectué en utilisant les poses et images déjà préparées, les hyperparamètres d'entraînement sont sélectionnés et l'historique de l'entraînement est sauvegardé pour une utilisation ultérieure. Au préalable, dans l'étape 1, les images ont été sauvegardées sous forme de vecteurs de directions, c'est pourquoi elles sont maintenant récupérées en tant qu'images, redimensionnées pour être adaptées au réseau (224x224 pixels), converties en tableaux et normalisées.

```
def preprocess_images(image_paths):
    images = []
    for path in image_paths:
        img = load_img(path, target_size=(224, 224))
        img_array = img_to_array(img)
        img_array = img_array / 255.0
        images.append(img_array)
    return np.array(images)

train_images_processed = preprocess_images(train_images)
test_images_processed = preprocess_images(test_images)
```

Code 3 – Prétraitement des images.

Une fois les images préparées, les hyperparamètres sont choisis. Dans ce cas, 15 époques ont été choisies pour observer quand le modèle cesse de changer de manière significative, et 10 échantillons par lot ont été sélectionnés, un nombre faible, afin de rendre le modèle plus robuste. Cela peut rendre l'exécution plus lente, mais étant donné la quantité de données, cette sélection a été faite. Un élément de type «CheckPoint» de Keras est également créé pour stocker le meilleur modèle obtenu au cours de l'entraînement pour une utilisation future.

```
epochs = 15
batch_size = 10
checkpoint = ModelCheckpoint("best_model.keras", monitor="val_loss",
save_best_only=True, verbose=1)
```

Code 4 – Hyperparamètres et Checkpoint.

Enfin, l'entraînement du modèle est lancé et l'historique de l'entraînement est sauvegardé pour une analyse ultérieure.

```
history = model.fit(
    x=train_images_processed,
    y=train_poses,
    validation_data=(test_images_processed, test_poses),
    epochs=epochs,
    batch_size=batch_size,
    callbacks=[checkpoint],
    verbose=1)

import pickle
with open("training_history.pkl", "wb") as file:
    pickle.dump(history.history, file)

print("Training completed and history saved!")
```

Code 5 – Entraînement du modèle.

**Questions :**

- **Quels paramètres influent sur le temps d'entraînement et la qualité de la convergence ?** Divers facteurs influent :
  - Hyperparamètres : Plus d'epochs permettent au modèle d'apprendre mieux, mais peuvent aussi entraîner un surajustement si elles sont trop nombreuses. Par contre, peu d'epochs peuvent entraîner un modèle sous-entraîné. En tant que la taille du batch, des tailles petites permettent des mises à jour fréquentes, mais augmentent le temps d'entraînement et des tailles grandes réduisent le temps d'entraînement, mais peuvent rendre la convergence plus difficile.
  - Taux d'apprentissage : Trop élevé peut empêcher le modèle de converger correctement, mais un taux très bas peut rendre l'apprentissage trop lent.
  - Qualité des données : Plus de données et plus de diversité dans l'ensemble d'entraînement améliorent la convergence.
  - Architecture du modèle : Modèles plus complexes nécessitent plus de temps d'entraînement, mais peuvent mieux s'ajuster aux données complexes, en ce cas, celui du ResNet50.
- **Quelle fonction de perte est appropriée pour ce type de régression, et pourquoi ?** Nous utilisons MSE (Mean Squared Error) car elle est idéale pour les tâches de régression, car elle mesure directement la différence quadratique entre les poses réelles et prédites et pénalise plus les grandes erreurs que les petites erreurs, ce qui aide le modèle à se concentrer sur les grandes erreurs et à les corriger.

## 4.4 Étape 4 : Prédiction et Calcul des Erreurs

But de l'étape : Effectuer des prédictions sur les données de test et calculer les erreurs de position et d'orientation.

Dans cette étape, avec le modèle déjà entraîné, on l'utilise pour effectuer une prédiction et analyser ses erreurs ainsi que son comportement lors de la prédiction des positions, qui est son objectif principal. Pour cela, la prédiction est d'abord effectuée avec les données de test, et l'idée est d'étudier les erreurs de position et d'orientation de manière séparée, la première par norme euclidienne des vecteurs, et la seconde par norme matricielle des matrices, une extension à $n-dimensions$ de la norme euclidienne, et ces erreurs sont ensuite stockées dans des vecteurs.

```
predicted_poses = model.predict(test_images_processed)

real_positions = test_poses[:, :3]
predicted_positions = predicted_poses[:, :3]

real_rotations = test_poses[:, 3:].reshape(-1, 3, 3)
predicted_rotations = predicted_poses[:, 3:].reshape(-1, 3, 3)

position_errors = np.linalg.norm(real_positions - predicted_positions, axis=1)

orientation_errors = [
    np.linalg.norm(real - pred, ord='fro')  # Frobenius norm
    for real, pred in zip(real_rotations, predicted_rotations)]
```

Code 6 – Utilisation du modèle pour prédire la pose.

**Questions :**

- **Pourquoi utilisons-nous une distance Euclidienne pour la position et une norme matricielle pour l'orientation ?** Nous utilisons la distance euclidienne pour la position et la norme matricielle pour l'orientation en raison de la forme de ces variables. Alors que la position est un vecteur colonne de coordonnées $(X, Y, Z)$, l'orientation est représentée sous forme de matrice de rotation $3x3$, ce qui nécessite l'utilisation de la norme matricielle pour calculer la différence, c'est-à-dire pouvoir calculer les valeurs d'erreur. Cela est dû au fait que tout est extrait d'une matrice de transformation homogène, comme expliqué dans la première étape.
- **Quelle est la différence entre l'erreur de position et l'erreur d'orientation en termes de calcul ?** Les différences en termes de calcul sont assez considérables, bien que dans le code, la même fonction soit utilisée (norme de numpy), et le système détecte simplement l'espace vectoriel dans lequel on travaille, puis il lui est indiqué la méthode lorsque l'espace vectoriel est matriciel (matrices 3x3), dans ce cas, la norme matricielle de Frobenius. Les calculs sont donc effectués comme suit :
  - Pour la position, on utilise la norme euclidienne classique des vecteurs, définie pour un vecteur de position réel (X, Y, Z) et la position prédite (X', Y', Z') comme :

$$d = \sqrt{(X - X')^2 + (Y - Y')^2 + (Z - Z')^2} \tag{2}$$

— Pour l'orientation, on utilise la norme matricielle de Frobenius, qui mesure la magnitude d'une matrice en sommant les carrés de tous ses éléments et en prenant la racine carrée. Elle est similaire à la norme euclidienne, mais appliquée aux matrices, idéale pour mesurer les différences entre matrices. Il existe plusieurs types de normes matricielles et leurs descriptions sont complexes et élaborées en termes de calcul. La manière la plus simple de calculer cette norme pour différencier la distance entre deux matrices est d'utiliser la méthode de la norme de Frobenius, et les autres méthodes de calcul sont laissées de côté dans ce rapport, car elles ne sont pas d'intérêt (4). Elle est définie comme suit :

$$\|R_{\text{reel}} - R_{\text{pred}}\|_F = \sqrt{\sum_{i,j} (R_{\text{reel}}[i,j] - R_{\text{pred}}[i,j])^2} \tag{3}$$

## 4.5 Étape 5 : Analyse Statistique des Erreurs

But de l'étape : Analyse de la précision des estimations de pose en calculant des statistiques sur les erreurs.

Dans cette étape, les erreurs générées lors de l'étape précédente seront simplement analysées statistiquement, c'est-à-dire en calculant les statistiques typiques et en les présentant sous forme de tableaux pour faciliter leur visualisation. Pour cela, le code suivant est utilisé :

```
position_stats = {
    "Mean": np.mean(position_errors),
    "Std Dev": np.std(position_errors),
    "Median": np.median(position_errors),
    "Min": np.min(position_errors), "Max": np.max(position_errors)}
orientation_stats = {
    "Mean": np.mean(orientation_errors),
    "Std Dev": np.std(orientation_errors),
    "Median": np.median(orientation_errors),
    "Min": np.min(orientation_errors), "Max": np.max(orientation_errors)}
position_table = pd.DataFrame([position_stats], index=["Position Errors"])
orientation_table = pd.DataFrame([orientation_stats],
index=["Orientation Errors"])

print("\nPosition Error Statistics:")
print(position_table)
print("\nOrientation Error Statistics:")
print(orientation_table)
```

Code 7 – Statistiques typiques.

Et le résultat des statistiques d'erreur est :

```
Position Error Statistics:
                    Mean   Std Dev    Median      Min       Max
Position Errors  0.110717  0.052116  0.106582  0.00717  0.288382

Orientation Error Statistics:
                       Mean   Std Dev    Median       Min       Max
Orientation Errors  0.163938  0.072127  0.149937  0.030789  0.406634
```

FIGURE 2 – Statistiques typiques comme tableaux.

**Questions :**

- **Que signifient les différentes mesures statistiques pour l'analyse de la précision des poses ?**
  - Moyenne : Représente l'erreur moyenne, basse indique que le modèle a, en général, de bonnes performances.
  - Écart-type (Std Dev) : Indique la dispersion des erreurs, faible signifie que les erreurs sont plus proches de la moyenne (comportement cohérent).
  - Médiane : Représente la valeur centrale de l'ensemble des erreurs, elle est utile en présence de valeurs atypiques, car elle n'est pas influencée par celles-ci comme la moyenne.
  - Minimum et maximum : Aident à identifier les limites extrêmes des erreurs, utiles pour détecter d'éventuelles valeurs atypiques ou des problèmes spécifiques dans certaines prédictions.
- **Quelle est l'importance de l'écart-type dans cette analyse ?** L'écart-type est crucial car il mesure la consistance du modèle, Si elle est élevée, cela signifie que le modèle a des performances incohérentes, avec certaines erreurs beaucoup plus grandes que d'autres. Si l'écart-type est faible, les erreurs sont regroupées autour de la moyenne, ce qui suggère que le modèle est fiable.

## 4.6 Étape 6 : Visualisation des Erreurs avec des Graphiques

But de l'étape : Visualisation des distributions des erreurs à l'aide d'histogrammes et de boxplots.

Dans cette étape, l'objectif est simplement d'obtenir visuellement les données des erreurs obtenues afin de les analyser plus facilement et de tirer des conclusions sur les prédictions du modèle. Comme outils, l'histogramme et le diagramme en boîte (boxplot) sont utilisés. Le code pour les tracer est présenté ci-dessous :

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(position_errors, bins=20, color='blue', alpha=0.7)
plt.title("Histogram of Position Errors")
plt.xlabel("Position Error (Euclidean Distance)")
plt.ylabel("Frequency")

plt.subplot(1, 2, 2)
plt.hist(orientation_errors, bins=20, color='green', alpha=0.7)
plt.title("Histogram of Orientation Errors")
plt.xlabel("Orientation Error (Frobenius Norm)")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 5))

plt.boxplot([position_errors, orientation_errors], vert=True,
patch_artist=True, labels=["Position", "Orientation"])
plt.title("Boxplot of Errors")
plt.ylabel("Error Value")
plt.grid(True)

plt.show()
```

Code 8 – Histogramme et boxplots.

Et les résultats sont les suivantes :

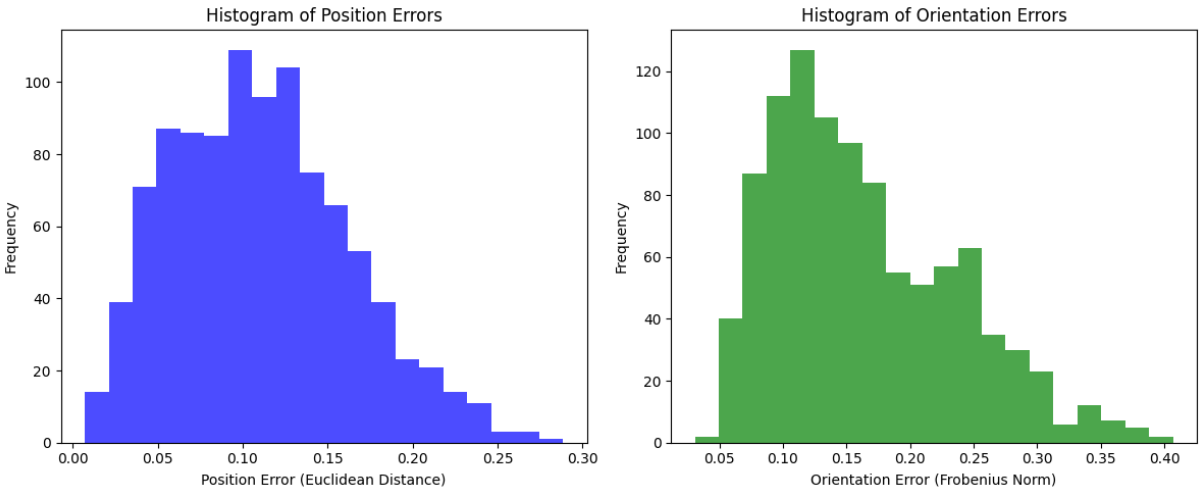


FIGURE 3 – Histogrammes de position et orientation.

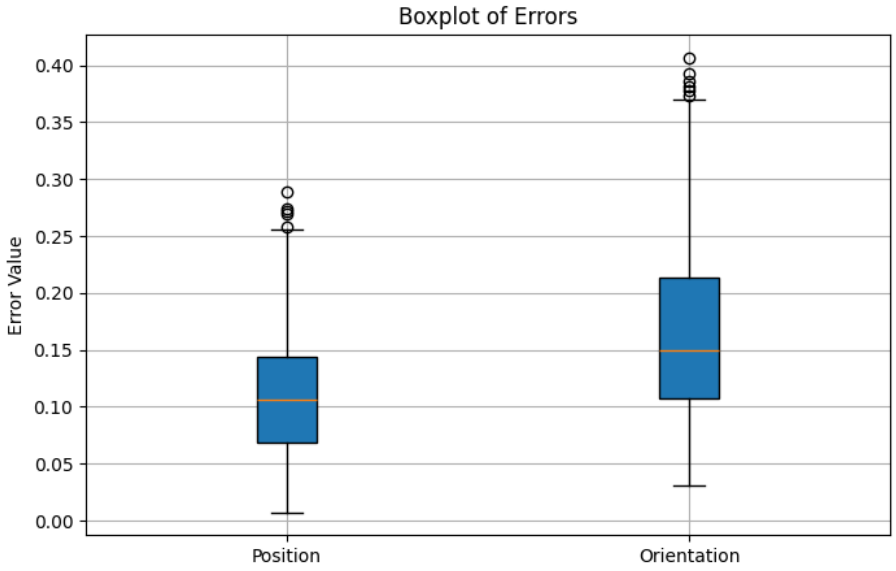


FIGURE 4 – Boxplot de position et orientation.

**Questions :**

- **Que révèlent les histogrammes sur la distribution des erreurs ?** Les histogrammes montrent que la majorité des erreurs de position et d'orientation sont faibles, les erreurs de position en sa majorité se situe entre 0,05 et 0,15, avec une fréquence élevée autour de 0,1. Les erreurs d'orientation ont plus de dispersion, se concentrant entre 0,05 et 0,20. La forme en cloche suggère que le modèle a de bonnes performances dans la majorité des cas, avec peu d'erreurs élevées.
- **Les boxplots montrent-ils des valeurs atypiques ? Si oui, que signifient-elles ?** Oui, les deux boxplots montrent des valeurs atypiques. Pour les erreurs de position, les valeurs atypiques (> 0,25) représentent des cas où le modèle a plus de difficulté à prédire la position, probablement en raison des scènes plus complexes ou faible représentativité dans l'entraînement. Pour les erreurs d'orientation, les valeurs atypiques (> 0,35)

indiquent des prédictions incorrectes de l'orientation, ce qui pourrait être causé par des changements brusques dans la pose ou une grande variabilité dans les images.

## 4.7 Étape 7 : Approximation des Erreurs par un Mélange de Gaussiennes

But de l'étape : Approximation de la distribution des erreurs par un mélange de Gaussiennes.

Dans cette étape, l'objectif est d'utiliser une analyse plus complexe que l'histogramme simple pour observer les erreurs de position et comprendre de manière plus détaillée comment le modèle se comporte. Pour cela, nous utiliserons un outil connu sous le nom de Modèle de Mélange Gaussien (GMM), qui approxime les erreurs et montre le résultat sur l'histogramme pour interpréter les résultats de manière plus détaillée. Avant de montrer le code, et étant donné qu'il s'agit de l'étape la plus cruciale impliquant des concepts peu connus, il est expliqué au préalable comment fonctionne le GMM, de quoi il s'agit et comment prendre des décisions à ce sujet.

Un Gaussian Mixture Model (GMM) est un modèle probabilistique qui représente la distribution des données comme une combinaison de multiples distributions gaussiennes (ou normales). Cela signifie qu'au lieu de supposer que toutes les données proviennent d'une seule distribution, le GMM divise les données en sous-groupes, chacun représenté par une gaussienne, chaque gaussienne ayant une moyenne, une variance et un poids. Ce modèle est idéal pour les problèmes où les erreurs ne sont pas distribuées de manière homogène, comme dans ce cas, où l'histogramme, bien que relativement uniforme, montre qu'il existe des cas de valeurs distribuées (en analysant à la fois la queue de l'histogramme et les valeurs atypiques et les quartiles dans le boxplot). La moyenne et la variance sont des valeurs statistiques connues et leur représentation est habituelle. Quant au poids, il s'agit d'une valeur qui nous indique comment chaque gaussienne contribue à l'ensemble des données, par exemple, si une gaussienne a un poids élevé ($w = 0.7$), cela signifie qu'elle explique 70% des données. Il est donc possible de créer une gaussienne pondérée, qui additionne de manière pondérée toutes les gaussiennes obtenues, formant ainsi une courbe gaussienne totale du modèle qui est plus fidèle à ce qui se passe réellement, étant plus précise que l'histogramme qui généralise et interprète toutes les erreurs comme égales, alors que dans la réalité, les causes des erreurs varient et ont tendance à se regrouper en sous-groupes. Par exemple, dans certains cas, un sous-groupe peut indiquer une erreur récurrente due à une condition compliquant la prise de données, comme des positions de la caméra difficiles à interpréter pour le modèle (3).

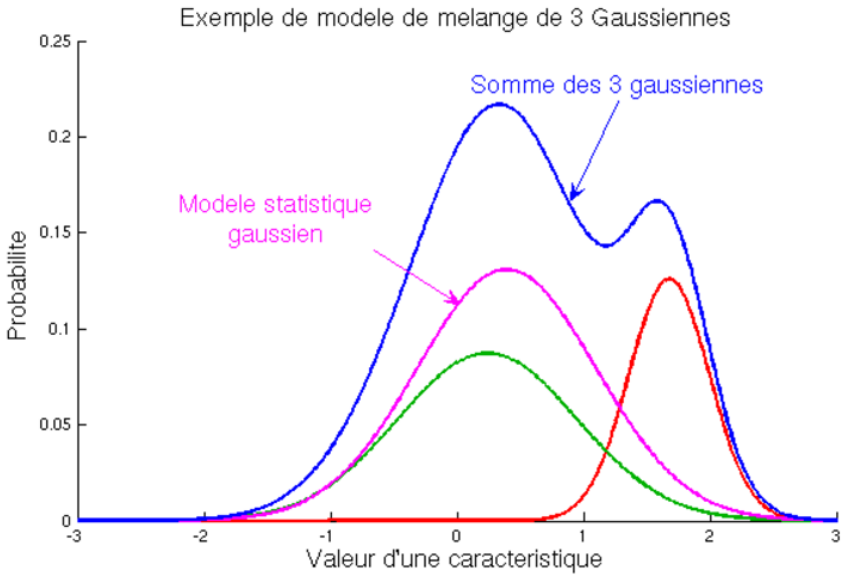


FIGURE 5 – GMM (1).

Une autre chose clé dans le GMM est le nombre de composants gaussiens, qui indique au GMM combien de gaussiennes il va générer, c'est-à-dire en combien de sous-groupes il va regrouper les erreurs avant de faire la pondération. L'idée étant de conserver la simplicité du modèle, on privilégie un faible nombre de composants, car cela évite que le GMM ne s'ajuste trop aux données spécifiques, ce qui pourrait conduire à un surajustement, c'est-à-dire qu'il deviendrait trop spécifique à certaines valeurs et incapable de travailler avec d'autres, similaire au surajustement que l'on observe avec les réseaux. Il est possible de choisir visuellement le nombre de composants en observant les sous-groupes existants, en examinant l'histogramme, où l'on peut voir un groupe initial d'erreurs faibles, une distribution intermédiaire assez élevée et une queue relativement éloignée. Cependant, il existe des méthodes pour choisir le nombre de composants du GMM, la plus courante et la plus utilisée pour les modèles simples étant le Critère d'Information Bayesien (BIC), qui pénalise la complexité et favorise des modèles simples, et qui est parfait pour ce cas en raison de la simplicité des erreurs que l'on peut observer dans l'histogramme, qui est assez uniforme. Bien que ce ne soit pas pertinent dans ce rapport, dans l'Annexe 1, se trouve le code permettant de calculer le nombre optimal de composants gaussiens à l'aide de la méthode BIC et de générer le graphique associé. En utilisant le BIC, il a été trouvé que le nombre idéal de composants pour le GMM est de 2 composants gaussiens (3).

Avec le concept de GMM maintenant clair, voici le code qui permet de réaliser le graphique :

```
gmm = GaussianMixture(n_components=n_components, random_state=42)
position_errors_reshaped = position_errors.reshape(-1, 1)

gmm.fit(position_errors_reshaped)

x = np.linspace(min(position_errors), max(position_errors), 500
).reshape(-1, 1)

gaussian_curves = np.exp(gmm.score_samples(x))

plt.figure(figsize=(10, 6))

plt.hist(position_errors, bins=20, density=True, color='blue',
alpha=0.6, label="Histogram of Position Errors")

for i in range(n_components):
    plt.plot(x, gmm.weights_[i] *
    np.exp(gmm._estimate_weighted_log_prob(x)[:, i]),
    label=f"Gaussian {i+1}")

plt.plot(x, gaussian_curves, color='red', linestyle='--',
label="GMM Combined Curve")

plt.title("Gaussian Mixture Model (GMM) for Position Errors")
plt.xlabel("Position Error (Euclidean Distance)")
plt.ylabel("Density")
plt.legend()
plt.show()
```

Code 9 – GMM.

Et les résultats sont les suivantes :

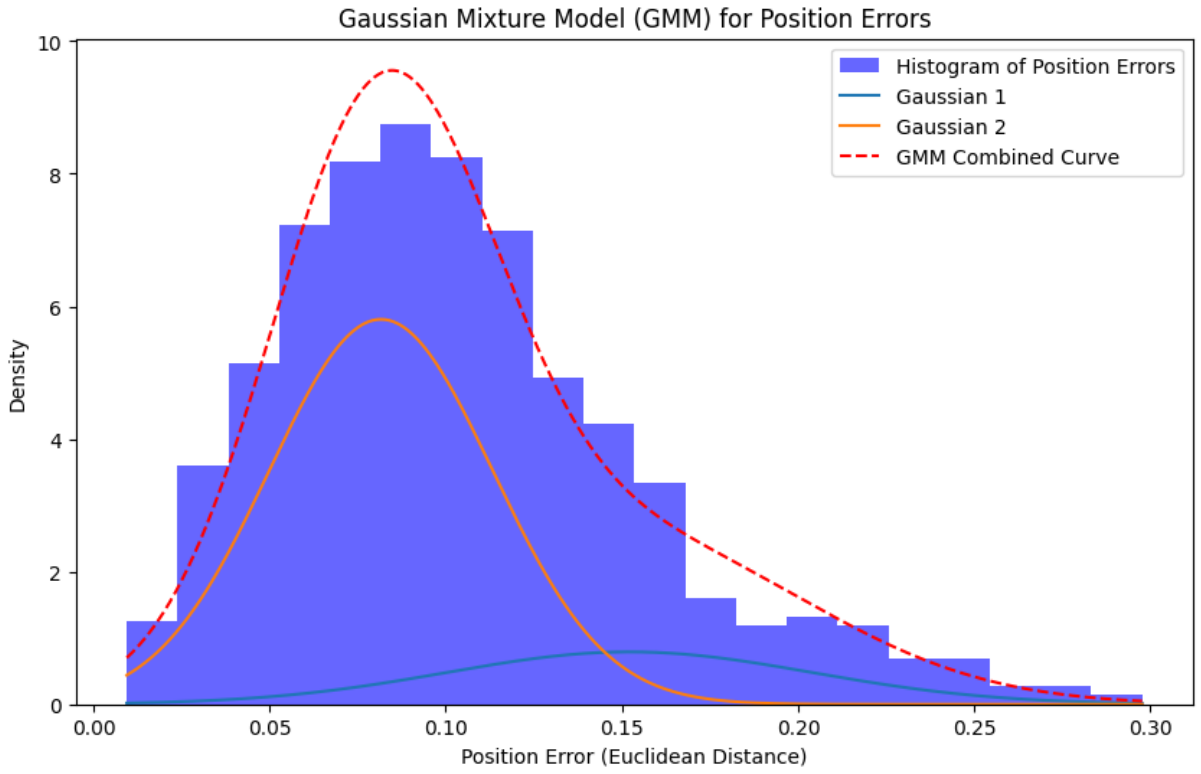


FIGURE 6 – GMM de position.

**Questions :**

- **Pourquoi un modèle de mélange gaussien peut-il mieux modéliser les erreurs qu'une seule distribution normale ?** Un modèle de mélange gaussien (GMM) est plus flexible qu'une seule distribution normale, car il peut représenter des distributions plus complexes ayant plusieurs modes ou sous-groupes. Par exemple, dans ce cas, les erreurs de position peuvent provenir de différentes conditions (comme des scènes faciles ou des angles difficiles), ce qui génère des regroupements distincts dans les données, et une seule distribution normale suppose que toutes les erreurs suivent une unique distribution symétrique, ce qui est souvent une simplification excessive.
- **Que signifient les composants du mélange dans le contexte des erreurs de pose ?** Chaque composant gaussien représente un sous-groupe d'erreurs avec des caractéristiques similaires. Le premier composant gaussien (avec une petite variance) pourrait représenter les cas où le modèle prédit avec une haute précision. Les autres composants (avec des moyennes plus élevées ou des variances plus grandes) pourraient représenter les cas où le modèle a plus de difficultés, comme dans des scènes ambiguës ou des angles de caméra complexes.

En conclusion, les résultats montrent que le modèle est capable d'estimer la pose 3D de manière fiable, avec une faible erreur moyenne et une dispersion contenue. L'implémentation du GMM comme outil d'analyse a permis une compréhension plus approfondie des caractéristiques des erreurs, mettant en évidence la robustesse du modèle face aux conditions difficiles des données de test.

# 5 Autres modèles avec plus de données pour l'entraînement et la validation

En tant qu'étude complémentaire, il a été décidé de réaliser et de tester le modèle avec une plus grande quantité de données. Un test a été effectué, en ajustant le modèle aux séquences 1 et 3 pour l'entraîner, et aux séquences 2 et 6 pour le valider. Comme hyperparamètres, 15 epochs et une taille de batch de 32 ont été choisis, une valeur typique étant une puissance de 2, utilisée parce qu'avec l'augmentation des données, le modèle devient plus lent et il est nécessaire de privilégier sa vitesse en augmentant la taille du batch. Les graphiques suivants ont été obtenus à partir de cette expérience :

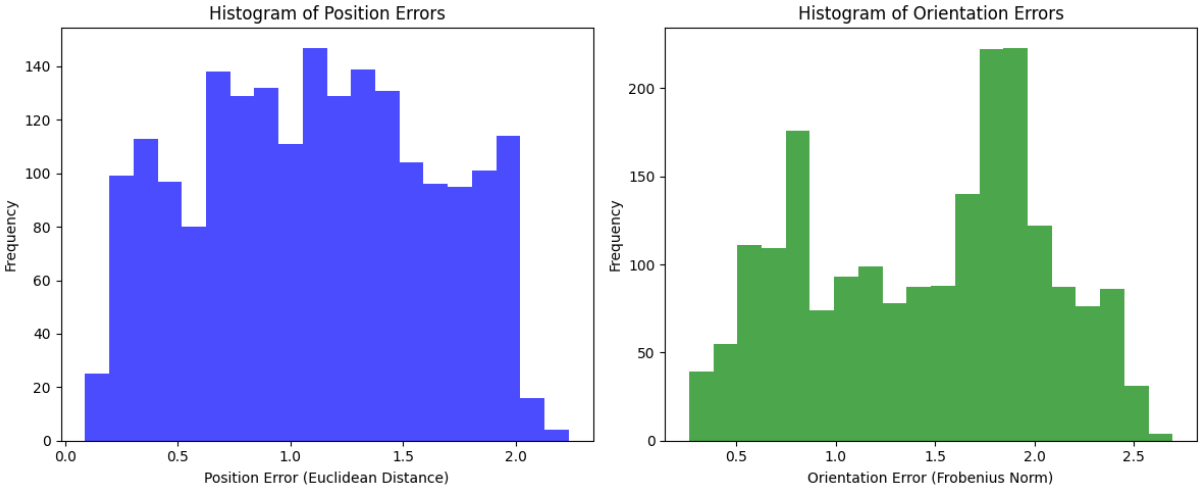


FIGURE 7 – Histogrammes de position et orientation, deuxième modèle.

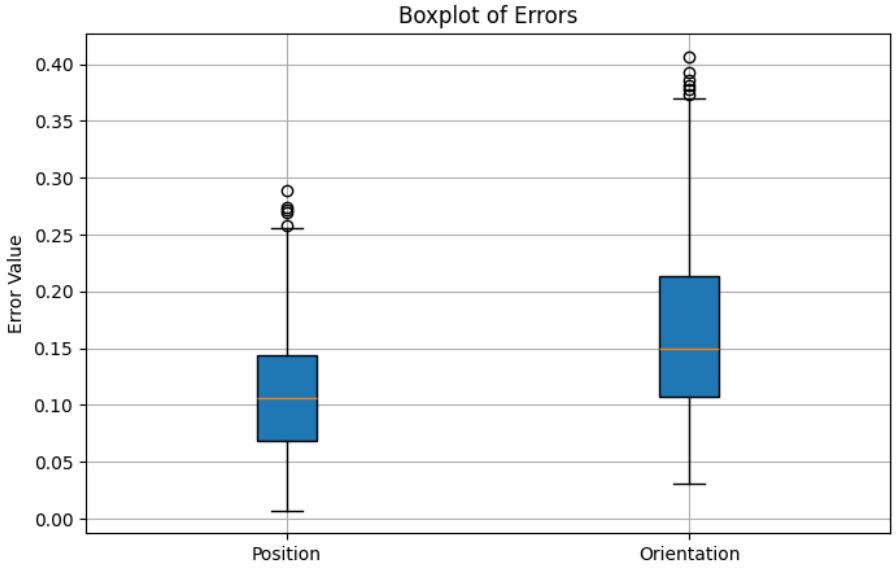


FIGURE 8 – Boxplot de position et orientation, deuxième modèle.

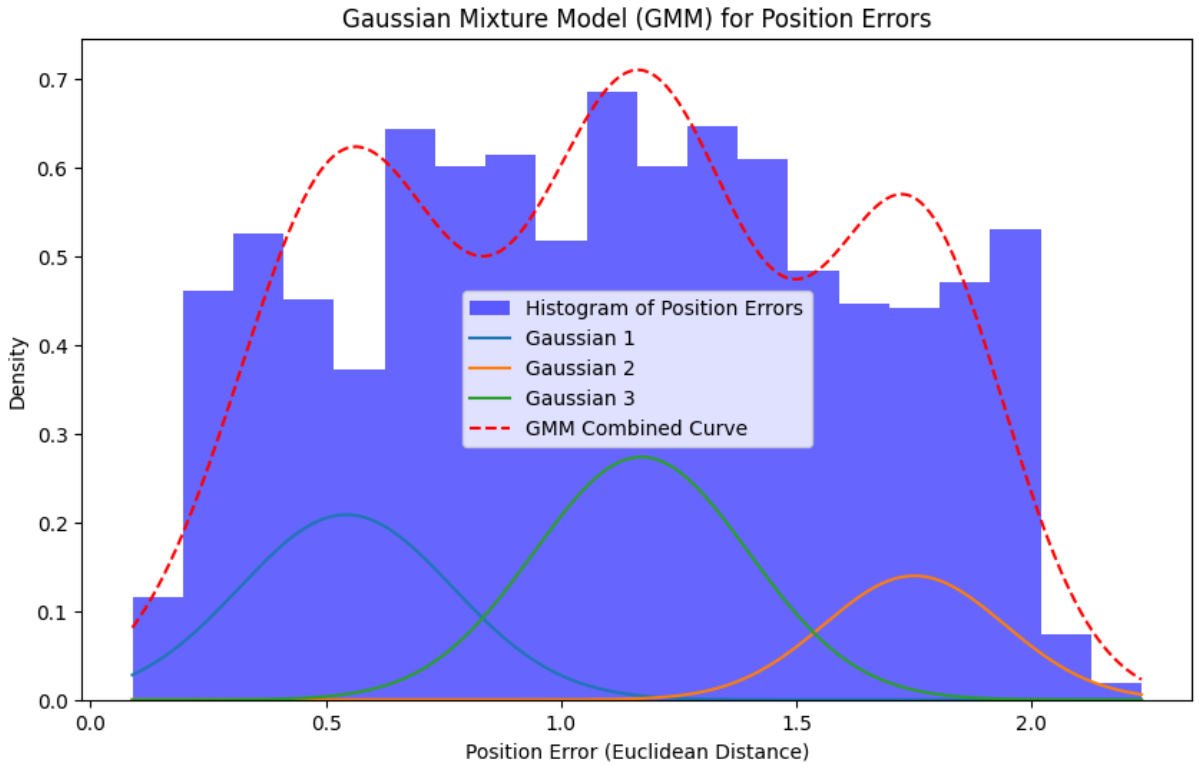


FIGURE 9 – GMM de position, deuxième modèle.

En observant les graphiques, on peut voir à la fois dans les histogrammes et dans le GMM que le modèle a considérablement augmenté l'erreur et que les distributions d'erreur ne sont plus aussi uniformes que dans le premier cas étudié dans le rapport. Cela nous indique que ce modèle n'est pas très performant et peut présenter des erreurs telles que du surapprentissage (overfitting) ou d'autres erreurs affectant sa capacité de prédiction.

Par la suite, il a été décidé d'augmenter encore la quantité de données, en utilisant toutes les séquences d'entraînement et de validation, avec des hyperparamètres de 10 epochs et une taille de batch de 32, afin de favoriser la vitesse du modèle. De plus, dans les deux modèles précédents, on a observé qu'à partir de 5-6 epochs, aucun changement significatif dans le **val_loss** n'était remarqué avant bien plus tard, ce qui est computationnellement coûteux en temps d'exécution et la différence d'amélioration n'est pas assez marquée. C'est pourquoi, avec l'augmentation des données, il a été décidé de réduire le nombre d'epochs. Les résultats suivants ont été obtenus :

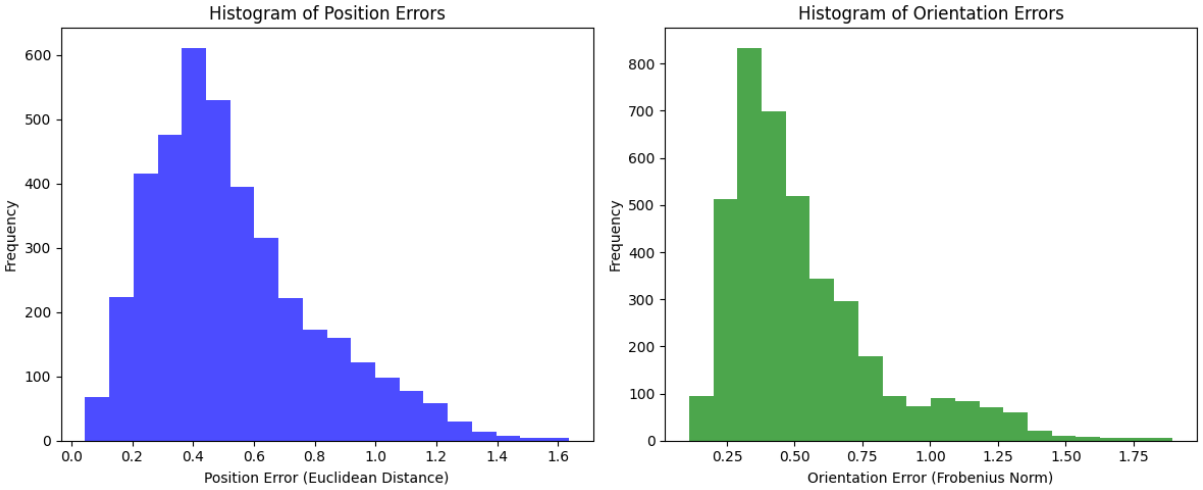


FIGURE 10 – Histogrammes de position et orientation, troisième modèle.

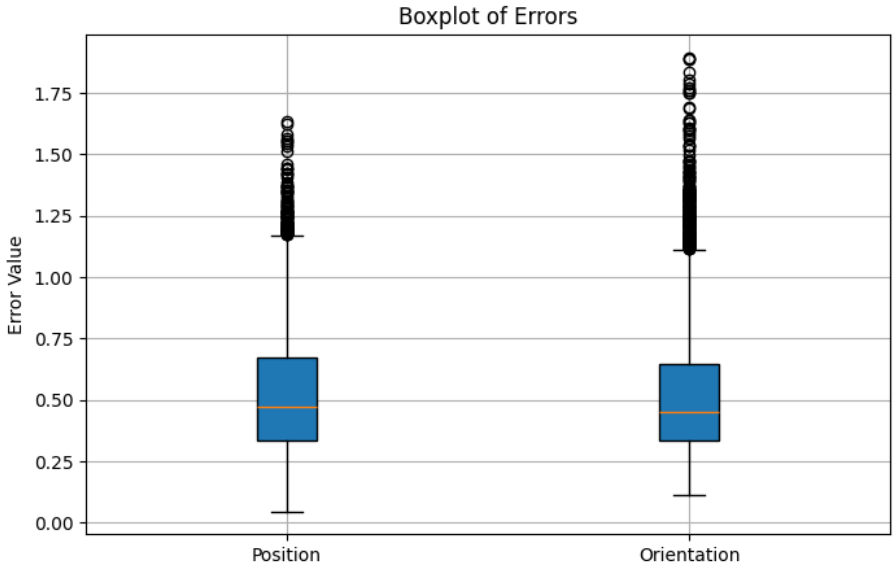


FIGURE 11 – Boxplot de position et orientation, troisième modèle.

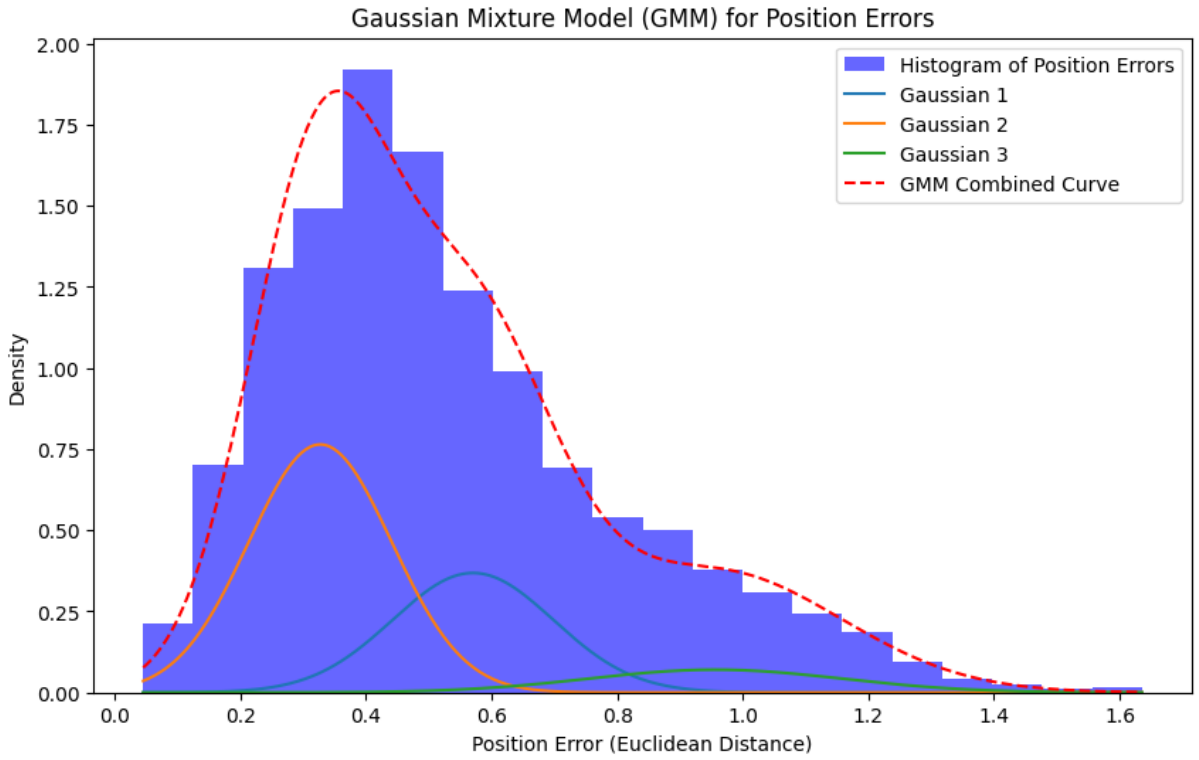


FIGURE 12 – GMM de position, troisième modèle.

Les résultats du troisième modèle montrent effectivement un bon modèle, qui ajuste mieux les données que le second. Cependant, en comparant ce modèle avec le premier exécuté, on remarque que les erreurs dans ce troisième modèle sont plus largement réparties dans l'histogramme et qu'il y a plus de valeurs atypiques dans le boxplot. Cela signifie que les erreurs obtenues sont plus grandes, ce qui est tout à fait prévisible en raison du changement des hyperparamètres et du fait qu'il y a maintenant plus de données à analyser, ce qui augmente la complexité du modèle. Cette complexité est facilement détectable dans le GMM, car les distributions sont désormais plus uniformes, ce qui permet d'étudier plus précisément les erreurs grâce à la quantité accrue de données pour former les sous-groupes.

Comme recommandation et travail futur, l'idée serait d'améliorer le troisième modèle afin d'améliorer ses performances, en réduisant la taille du batch à 16 ou même 8, en augmentant le nombre d'epochs si nécessaire, et ces modifications des hyperparamètres devraient l'améliorer. Il est également possible d'améliorer davantage le modèle en appliquant des stratégies de régularisation sur le réseau, une validation croisée pour éviter la dépendance de la division des données entre entraînement et validation, un augmentation des données en utilisant des rotations et des transformations des données déjà présentes (puisque ce modèle utilise toutes les données disponibles), et enfin, analyser d'autres métriques pour vérifier si le modèle s'améliore réellement ou si de la mémoire et du temps de calcul sont gaspillés, ou pire encore, si un surajustement (overfitting) se produit.

# 6 Conclusions

Le projet a permis d'explorer l'entraînement d'un modèle de régression basé sur ResNet-50 pour la prédiction de poses 3D. L'utilisation de données bien structurées et d'une augmentation contrôlée des échantillons a montré que le modèle atteint une performance robuste, sans signes de surapprentissage.

Les résultats des histogrammes et des boxplots confirment que les erreurs de position et d'orientation sont faibles et bien distribuées. De plus, l'utilisation du modèle de mélange gaussien (GMM) a permis de mieux modéliser la répartition des erreurs, soulignant la qualité du modèle et sa capacité à capturer la complexité des données.

Pour aller plus loin, des améliorations possibles incluent l'ajustement des hyperparamètres (batch size, epochs) ou l'exploration d'architectures plus avancées pour affiner la précision, tout en maintenant une généralisation optimale.

# Bibliographie

[1] Notions abordées dans les démonstrations metiss. *Institute for Research in Computer Science and Random Systems IRISA* (2024). Accès :1 de Décembre de 2024.

[2] Ababsa, F. Estimation de pose 3d par transfert learning avec resnet50. analyse des erreurs de pose. travaux pratiques. *TP Analyse de Pose* (2024).

[3] Martinez Castillo, D. Modelos de mezclas gaussianas como clasificadores en el contexto de machine learning. *MONOGRAFÍA DE TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE MATEMÁTICO PROYECTO CURRICULAR DE MATEMÁTICAS. Universidad Distrital Francisco José de Caldas. Bogotá D.C., Colombia.* (2022).

[4] Wikipedia. Matrix norm. Accès :1 de Décembre de 2024.

# Annexe 1 : Code Python pour calculer le nombre optimal de composants gaussiens à l'aide de la méthode BIC

Le code suivant présente comment appliquer le critère d'information bayésienne BIC, en plus de tracer le processus et présenter le nombre de composants gaussiens approprié pour le GMM.

```
bic_scores = []
n_components_range = range(1, 10)

for n_components in n_components_range:
    gmm = GaussianMixture(n_components=n_components, random_state=42)
    gmm.fit(position_errors.reshape(-1, 1))
    bic_scores.append(gmm.bic(position_errors.reshape(-1, 1)))

plt.figure(figsize=(10, 6))
plt.plot(n_components_range, bic_scores, label="BIC", marker='o', color='blue')
plt.xlabel("Number of Components")
plt.ylabel("BIC Score")
plt.title("BIC for GMM")
plt.legend()
plt.grid(True)
plt.show()

optimal_components_bic = n_components_range[np.argmin(bic_scores)]
print(f"Optimal number of components (BIC): {optimal_components_bic}")
```