# Travail pratique sur un modèle de patte d'oiseau

CARPENTIER Lucas COURTOIS Thibault SYSMER 3A

Responsable : M. Hugel 27 octobre 2024

# Introduction

Ce TD s'inspire des travaux de recherche présentés dans l'article suivant :

> Vincent Hugel, Rémi Hackert, et Anick Abourachid, *Kinematic Modeling of Bird Locomotion from Experimental Data*, IEEE Trans. Rob., 2011.

Dans ces travaux, des données 3D issues de radiocinématographie d'oiseaux ont été utilisées pour modéliser la locomotion des pattes d'oiseaux en robotique. Grâce à ces données expérimentales, il est possible de déterminer un modèle de patte d'oiseau capable de reproduire fidèlement les trajectoires naturelles observées chez les oiseaux étudiés comme la caille et la sarcelle. L'objectif de ce TD est d'implémenter un programme Python permettant de commander les articulations d'une patte d'oiseau de manière à reproduire le plus fidèlement possible les trajectoires mesurées sur un oiseau réel, en particulier les trajectoires du pied, de la cheville et éventuellement du genou.

Dans ce cadre, nous allons :

— **Modéliser la jambe robotisée** en utilisant des articulations représentant la hanche, le genou et la cheville ;

— **Optimiser les angles des articulations** afin de minimiser l'écart entre la trajectoire cible (mesurée sur les oiseaux réels) et la trajectoire calculée des segments de la patte robotisée ;

— **Appliquer des méthodes d'optimisation numérique**, en utilisant un algorithme de minimisation, pour ajuster les angles initiaux et les suivre sur une trajectoire prédéfinie.

Ce TP nous permettra de mettre en pratique des concepts de cinématique inverse, d'optimisation de trajectoire et de contrôle des articulations dans un contexte biomécanique.

# 1 Calcul des commandes articulaires et initialisation

Dans cette section, nous allons analyser le calcul des commandes articulaires, en particulier le processus de cinématique inverse utilisé pour ajuster les angles des articulations de la hanche, du genou et de la cheville.

## 1.1 Initialisation des positions articulaires

Pour commencer, nous initialisons les positions des articulations par rapport à la hanche. Ces positions sont calculées en prenant les coordonnées relatives du genou, de la cheville et du pied par rapport à la hanche. Ces valeurs servent de référence pour les étapes suivantes de calcul.

```
# Initialisation des points articulaires par rapport    la hanche
K0 = np.array([xKnee[ind_start]-xHip[ind_start],
               yKnee[ind_start]-yHip[ind_start],
               zKnee[ind_start]-zHip[ind_start]])

A0 = np.array([xAnkle[ind_start]-xHip[ind_start],
               yAnkle[ind_start]-yHip[ind_start],
               zAnkle[ind_start]-zHip[ind_start]])

F0 = np.array([xFoot[ind_start]-xHip[ind_start],
               yFoot[ind_start]-yHip[ind_start],
               zFoot[ind_start]-zHip[ind_start]])
```

Listing 1 – Initialisation des points articulaires

## 1.2 Variation des coordonnées du pied

La variation de la position du pied, notée `deltaFoot` est calculée pour chaque échantillon de la trajectoire mesurée. Cette variation est utilisée pour ajuster les angles des articulations dans la prochaine étape.

```
# Premi re variation des coordonn es du pied
deltaFoot = np.array([xFoot[ind_start+1]-xFoot[ind_start],
                      yFoot[ind_start+1]-yFoot[ind_start],
                      zFoot[ind_start+1]-zFoot[ind_start]])
```

Listing 2 – Calcul de la variation des coordonnées du pied

## 1.3 Cinématique directe

La cinématique directe permet de calculer la position des articulations à partir des angles actuels des articulations. Le modèle calcule les positions du genou, de la cheville et du pied en fonction des angles de la hanche, du genou et de la cheville.

```
# Calcul de la position des articulations via la cin matique directe
K, A, F = MyLegLModel.calc_KneeAnkleFootCoord(qHipRoll, qHipIncl, qKnee, qAnkle)
```

Listing 3 – Calcul de la cinématique directe

## 1.4 Calcul des matrices jacobiennes

La matrice jacobienne permettent de relier les variations de positions des articulations aux variations des angles articulaires. Cela permet d'estimer la variation des angles nécessaires pour atteindre une nouvelle position.

```
# Calcul des matrices jacobiennes (genou, cheville, pied)
JK, JA, JF = MyLegLModel.calc_JacobianKneeAnkleFoot(qHipRoll, qHipIncl, qKnee,
    qAnkle)
```

Listing 4 – Calcul des matrices jacobiennes

### 1.5 Utilisation de la pseudo-inverse

La pseudo-inverse de la matrice jacobienne notée $J^+$, est utilisée pour résoudre le problème de la cinématique inverse. En multipliant la pseudo-inverse par la variation des coordonnées du pied (`deltaFoot`), nous pouvons obtenir la variation des angles nécessaires pour ajuster la position des articulations.

```
# Calcul de la pseudo-inverse pour la cin matique inverse
Jp = LA.pinv(JF)

# Calcul des variations d'angles
Dq = np.dot(Jp, deltaFoot)
```

Listing 5 – Utilisation de la pseudo-inverse

### 1.6 Ajustement des commandes articulaires

Une correction supplémentaire est apportée par le vecteur $Z$, qui permet de stabiliser les ajustements angulaires en réduisant l'amplitude des corrections. Le paramètre $\nu$ est utilisé pour pondérer l'influence du vecteur $Z$ sur les ajustements finaux des angles.

```
# Calcul du vecteur Z pour ajuster les commandes articulaires
MaddZ = np.eye(4) - np.dot(Jp, JF)
Z = np.dot(MaddZ, np.array([qHipRoll, qHipIncl, qKnee, qAnkle]))

# Mise    jour des commandes articulaires avec le vecteur Z
nu = 1.0
qHipRoll = Dq[0] - nu * Z[0]
qHipIncl = Dq[1] - nu * Z[1]
qKnee = Dq[2] - nu * Z[2]
qAnkle = Dq[3] - nu * Z[3]
```

Listing 6 – Ajustement des commandes articulaires

Ces étapes nous ont permis de calculer la première coordonnée de la trajectoire. Il est désormais nécessaire de répéter ce processus pour l'ensemble des points afin d'obtenir la trajectoire complète calculée pour chaque articulation.

## 2 Calcul itératif de la trajectoire complète

Après avoir calculé la première coordonnée de la trajectoire nous devons répéter le processus pour l'ensemble des points afin d'obtenir une trajectoire complète. La boucle suivante permet de mettre à jour les angles des articulations à chaque étape de la trajectoire mesurée et de calculer les variations nécessaires pour minimiser l'écart entre la position calculée et la position cible.

### 2.1 Position suivante et variation des coordonnées

La première étape dans chaque itération consiste à calculer la position suivante du pied (`Footnext`) en fonction des coordonnées relatives à la hanche. Ensuite, la variation de la position du pied.

```
# Position suivante sur la trajectoire mesur e
Footnext = np.array([xFoot[i] - xHip[i], yFoot[i] - yHip[i], zFoot[i] - zHip[i]])

# Variation de la position du pied (deltaFoot)
deltaFoot = np.array([
    Footnext[0] - FootCurrent[0],
    Footnext[1] - FootCurrent[1],
    Footnext[2] - FootCurrent[2]
])
```

Listing 7 – Calcul de la position suivante et variation des coordonnées

Ici `Footnext` représente les nouvelles positions du pied et de la cheville respectivement. La variation `deltaFoot` est calculée en prenant la différence entre la nouvelle position et la position actuelle.

## 2.2 Matrices jacobiennes et pseudo-inverse

Une fois la variation de position calculée nous devons mettre à jour les matrices jacobiennes des articulations. Ensuite nous pouvons calculer la pseudo-inverse. Cela permet d'ajuster les angles d'articulation en fonction de la position cible.

```
# Calcul des matrices jacobiennes
JK, JA, JF = MyLegLModel.calc_JacobianKneeAnkleFoot(qHipRoll, qHipIncl, qKnee,
    qAnkle)

# Calcul de la pseudo-inverse des matrices jacobiennes
Jp = LA.pinv(JF)
```

Listing 8 – Calcul des matrices jacobiennes et pseudo-inverse

La pseudo-inverse (`Jp`) est utilisée pour résoudre le problème de cinématique inverse. Grâce à cela, nous pouvons calculer les variations d'angles nécessaires pour corriger la trajectoire.

## 2.3 Ajustement des angles et correction avec le vecteur $Z$

Après avoir calculé les matrices jacobiennes, nous devons mettre à jour les angles en fonction des variations de position.

```
# Calcul des variations d'angles
Dq = np.dot(Jp, deltaFoot)

# Calcul du vecteur Z pour ajuster les corrections angulaires
MaddZ = np.eye(4) - np.dot(Jp, JF)
Z = np.dot(MaddZ, np.array([qHipRoll, qHipIncl, qKnee, qAnkle]))

# Mise    jour des angles d'articulation
qHipRoll = qHipRoll + Dq[0] - nu * Z[0]
qHipIncl = qHipIncl + Dq[1] - nu * Z[1]
qKnee = qKnee + Dq[2] - nu * Z[2]
qAnkle = qAnkle + Dq[3] - nu * Z[3]
```

Listing 9 – Ajustement des angles avec le vecteur Z

Dans ce passage, les variations des angles (`Dq`) sont calculées à partir de la pseudo-inverse et de la variation de position. Le vecteur $Z$ est utilisé pour stabiliser les corrections en réduisant l'amplitude des ajustements.

## 2.4 Mise à jour des positions et stockage des résultats

Enfin, nous recalculons les positions des articulations (genou, cheville, pied) à l'aide de la cinématique directe. Ces positions sont ensuite stockées pour être utilisées lors de l'optimisation de la trajectoire complète.

```
# Calcul des nouvelles positions des articulations avec la cin matique directe
KCurrent, ACurrent, FootCurrent = MyLegLModel.calc_KneeAnkleFootCoord(qHipRoll,
    qHipIncl, qKnee, qAnkle)

# Stockage des positions et des angles calcul s
Kq[i] = np.squeeze(KCurrent[0:3]).tolist()
Aq[i] = np.squeeze(ACurrent[0:3]).tolist()
Fq[i] = np.squeeze(FootCurrent[0:3]).tolist()

qH1[i] = qHipRoll
qH2[i] = qHipIncl
qK[i] = qKnee
qA[i] = qAnkle
```

Listing 10 – Mise à jour des positions et stockage

Les positions actuelles (`KCurrent`, `ACurrent`, `FootCurrent`) sont mises à jour et stockées dans les tableaux `Kq`, `Aq`, `Fq`, afin de pouvoir reconstruire la trajectoire complète. Cela permet donc de calculer l'ensemble de la trajectoire. Voyons à quoi celle-ci ressemble :

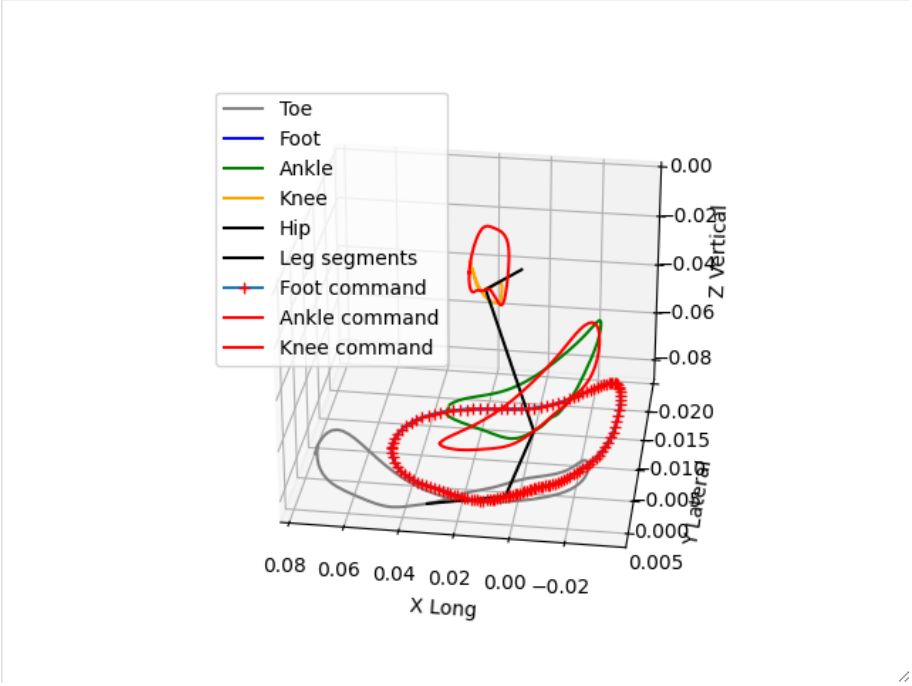


FIGURE 1 – Trajectoires des différentes parties de l'articulation dans l'espace 3D.

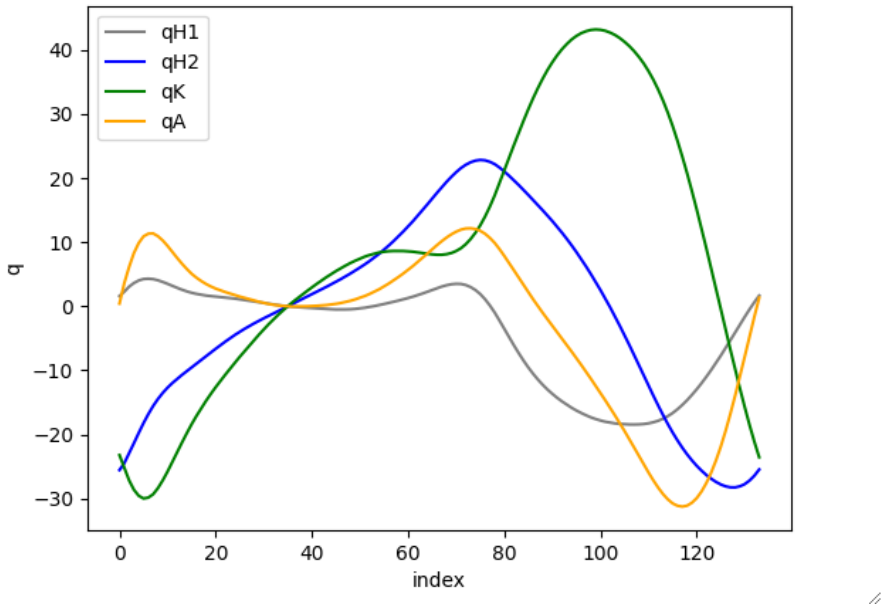


FIGURE 2 – Évolution des angles articulaires en fonction du temps (initialisation 1).

Les trajectoires obtenues montrent des écarts entre la trajectoire calculée et la trajectoire réelle observée. Cela suggère que notre modèle n'a pas parfaitement estimé les composantes. De plus, nous remarquons que la trajectoire calculée est sensible au vecteur initial $x_0$. Par exemple, en utilisant $x_0 = [90, 90, 90, 90]$ (contre $x_0 = [116, 55, -93, 67]$ auparavant), la trajectoire et les angles articulaires se modifient comme suit :

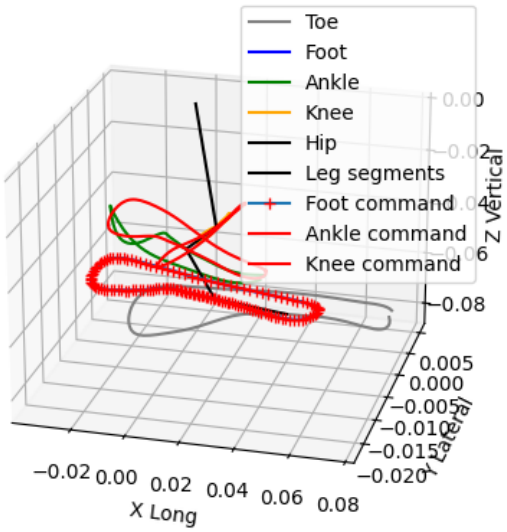


FIGURE 3 – Trajectoires des différentes parties de l'articulation avec une nouvelle initialisation.

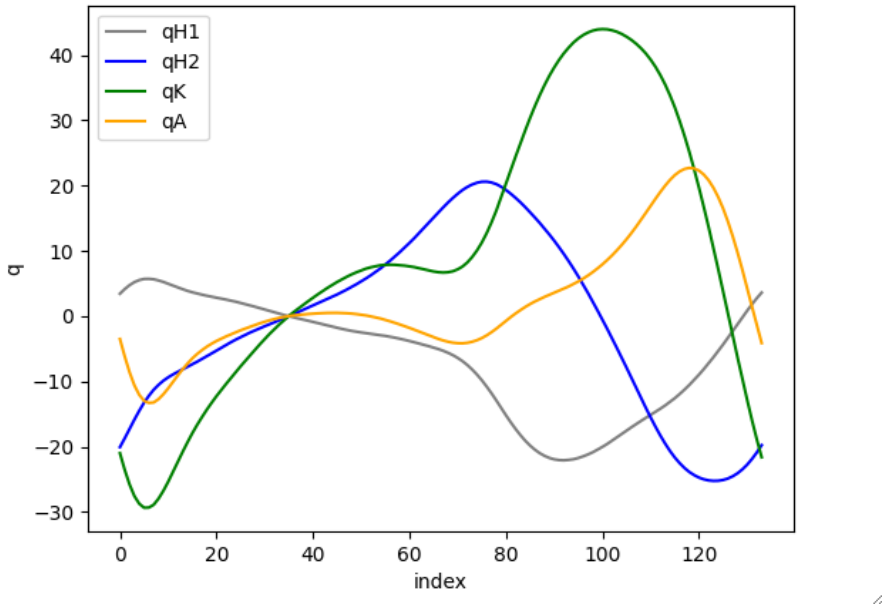


FIGURE 4 – Évolution des angles articulaires en fonction du temps (initialisation 2).

Nous observons que la trajectoire calculée dépend fortement des valeurs initiales de $x_0$, ce qui est conforme au principe de l'optimisation. L'étape suivante consistera à trouver les meilleures conditions initiales minimisant l'erreur de calcul.

# 3 Optimisation du calcul de la trajectoire

## 3.1 Fonction de coût

La fonction de coût est au cœur de l'optimisation du modèle cinématique. Elle permet de mesurer l'erreur entre la position calculée de la cheville (*AnkleCurrent*) et la position cible suivante (*AnkleNext*) qui est elle déterminée à partir des données mesurées. La fonction de coût est formulée pour minimiser cette erreur, qui est représentée sous forme de distance euclidienne entre les deux positions, comme suit :

$$\text{cost+} = \sqrt{(AnkleNext - ACurrent)^2}$$

L'erreur est calculée pour chaque point de la trajectoire, et la somme de toutes ces erreurs forme le coût total de la trajectoire. L'objectif de l'optimisation est de trouver les angles qui minimisent ce coût. Nous obtiendrions alors la trajectoire qui correspond le mieux possible à la trajectoire mesurée. Pour notre trajectoire précédente, l'erreur est de 0.5925246445575614. Le but est donc de la minimiser.

## 3.2 Optimisation avec SciPy

Pour minimiser la fonction de coût définie, nous avons recours à l'optimiseur de SciPy, en particulier la fonction `minimize()` qui utilise un algorithme d'optimisation. L'optimisation commence par un vecteur initial des angles articulaires $x_0 = [q_{\text{hip\_incl\_0}}, q_{\text{hip\_incl\_1}}, q_{\text{ankle\_0}}, q_{\text{ankle\_1}}]$. Ce vecteur

est ensuite ajusté à chaque itération de l'optimisation afin de minimiser la fonction de coût. Voici comment l'optimisation est mise en œuvre :

$$\text{result} = \text{minimize}(\text{cost\_function}, x_0, \text{method} =' BFGS')$$

Dans ce cas, `cost_function` est la fonction de coût que nous avons définie précédemment. L'optimiseur itère sur les valeurs d'angles de la hanche et de la cheville, à la recherche d'un ensemble d'angles qui minimise l'écart entre la position calculée et la position réelle mesurée.

À la fin de l'optimisation, l'optimiseur retourne un ensemble d'angles qui correspond à la meilleure estimation possible pour minimiser l'erreur de trajectoire. Après avoir appliqué la méthode d'optimisation avec la fonction de coût décrite précédemment, l'algorithme a convergé vers une solution optimale. Voici les résultats fournis par l'optimiseur `scipy.optimize.minimize` :

```
message: CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
  success: True
   status: 0
      fun: 0.37819398748770616
        x: [ 1.403e+02  7.726e+01  1.162e+02  1.126e+02]
      nit: 14
      jac: [-5.185e-06 -1.832e-07 -6.217e-07 -2.887e-07]
     nfev: 90
     njev: 18
 hess_inv: <4x4 LbfgsInvHessProduct with dtype=float64>
```

L'optimisation a abouti avec succès, comme indiqué par le message *CONVERGENCE* et `success: True`. Voici quelques détails des résultats :

- La valeur finale de la fonction de coût, `fun`, est de 0.3782. Cela représente l'erreur résiduelle minimale entre les trajectoires mesurées et calculées après optimisation.
- Le vecteur optimal des angles articulaires `x` obtenu est :

  $$x = [140.3°, 77.26°, 116.2°, 112.6°]$$

  Ces valeurs représentent les angles optimisés pour la hanche et la cheville qui minimisent l'erreur entre les positions mesurées et calculées de la jambe.
- Le nombre d'itérations effectuées par l'algorithme est de 14 (`nit: 14`), avec 90 évaluations de la fonction de coût (`nfev: 90`) et 18 évaluations de son gradient (`njev: 18`).
- Le vecteur du gradient projeté (`jac`) est proche de zéro, ce qui indique que l'algorithme a trouvé un minimum local.

Essayons alors de recalculer la trajectoire à partir de ces paramètres initiaux :

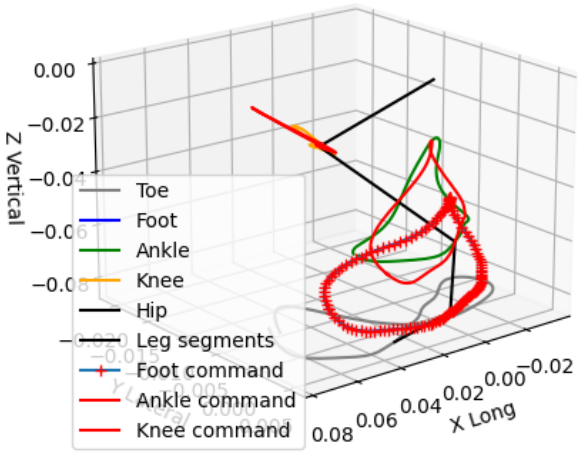


FIGURE 5 – Évolution des angles articulaires en fonction du temps (initialisation optimisée).

Nous pouvons remarquer que les courbes de commandes et mesurée sont nettement plus proches que précédemment. Cela montre bien que la sélection des paramètres initiaux est importante pour la convergence de notre méthode. Nous pouvons observer que les courbes des commandes calculées et mesurées sont nettement plus proches qu'auparavant. Cela démontre clairement l'importance de la sélection des paramètres initiaux pour assurer la convergence de notre méthode. Une meilleure initialisation permet de réduire l'erreur entre les trajectoires, ce qui conduit à une optimisation plus efficace et à des résultats plus précis. Ces résultats soulignent que l'optimisation des conditions initiales est cruciale pour atteindre un contrôle articulaire optimal.

# Conclusion

Dans ce travail, nous avons implémenté une méthode permettant de modéliser et de commander une patte robotique inspirée des mouvements naturels observés chez les oiseaux. En utilisant des données 3D issues de la radiocinématographie, nous avons pu déterminer les positions relatives des articulations de la hanche, du genou, de la cheville et du pied. Nous avons ensuite appliqué des méthodes de calcul géométrique pour estimer les angles d'articulations nécessaires à la reconstitution des trajectoires mesurées.

Le cœur de notre approche réside dans l'utilisation de la cinématique inverse et des matrices jacobiennes pour ajuster les commandes des articulations en fonction des variations des positions. Nous avons pu observer que la sélection des paramètres initiaux joue un rôle déterminant dans la qualité de la trajectoire calculée. Cela nous a conduit à appliquer des méthodes d'optimisation pour minimiser l'erreur entre les trajectoires calculées et mesurées.

Les résultats obtenus montrent une nette amélioration de la précision des trajectoires lorsque des conditions initiales optimisées sont utilisées. Cependant, il est important de noter que les résultats restent sensibles à ces paramètres initiaux et que des méthodes d'optimisation supplémentaires, telles que les algorithmes génétiques, pourraient offrir de meilleures performances en minimisant encore davantage l'erreur.