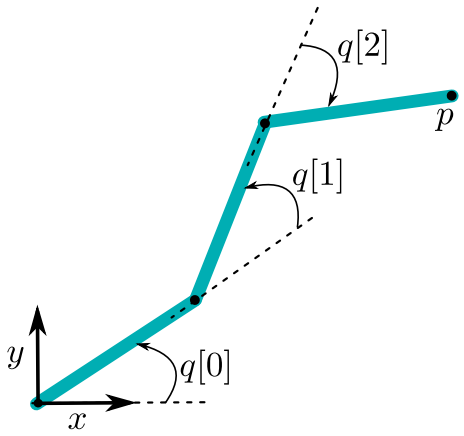
TP : introduction à la modélisation et à la commande en robotique

Infos pratiques :
— À faire de préférence en binôme.
— Il n'y a pas de document à rendre. L'évaluation sera faite durant le TP, et dépendra en partie de l'ordre d'arrivée des groupes aux différents "checkpoints" (en tenant compte de l'aide nécessaire pour y arriver).

1 Cinématique inverse d'un bras articulé plan

Le fichier `inverse_kin.py` constitue votre point de départ.

On considère un bras articulé plan constitué de N segments de taille l. Ses configuration son définies par le vecteur des angles relatifs q. On veut effectuer avec ce bras des mouvements décrits par des configurations successives dont le nombre est égal à la variable `ITERATIONS`.



Deux matrices `midpoints_x` et `midpoints_y`, chacune de taille $(N+1) \times ITERATIONS$, sont utilisées pour stocker toutes les positions des points intermédiaires du bras articulé (en incluant l'origine comme premier point, et le point p, c'est-à-dire l'extrémité du robot, comme $(N+1)$-ième et dernier point). Par exemple, à l'itération `count`, la position selon x du i-ième point sera égale à `midpoints_x[i, count]`. À cette même itération les coordonnées du point p sont :

`(midpoints_x[N, count],midpoints_x[N, count])`

Checkpoint 1 : initialisez le vecteur q avec la valeur de 0.2 (rad) pour tous les (N) angles, et modifiez dans la boucle principale ("`for count in range(ITERATIONS)...`") le calcul des valeurs de `midpoints_x` et `midpoints_y`, afin que la position initiale affichée soit correcte.

Dans la boucle principale, l'objectif est d'itérer des calculs de valeurs articulaires pour faire suivre au point p une trajectoire désirée. Pour cela on va utiliser la méthode de la cinématique inverse. À chaque itération, la vitesse désirée du point p est donnée par :

```
v_des = p_target(count) - p.
```

Pour trouver une vitesse articulaire q' adéquate, on passe par la pseudo-inverse $J^{\#}$: $q' = J^{\#} v\_des$. Ce calcul est effectué par la ligne de code suivante :

```
q_prime = np.dot(np.linalg.pinv(J),v_des).
```

Une fois que cette vitesse articulaire est déterminée, on met à jour la valeur de q :

```
q = q + delta/(np.linalg.norm(q_prime) + 1e-6) * q_prime.
```

Ici `delta` correspond au temps écoulé (qui doit être très petit pour que l'approximation au premier ordre soit valable) entre deux configurations successives.

Checkpoint 2 : implémentez le calcul de la Jacobienne. Le mouvement obtenu est circulaire. Ajustez les divers paramètres temporels de façon à ce que ce mouvement soit relativement fluide à l'écran, et testez plusieurs valeurs de N.

Checkpoint 3 : la fonction `p_target(i)` définit le mouvement désiré. Modifiez cette fonction afin de faire converger la position du point p vers une position particulière donnée, puis faites bouger le point p selon une ligne droite, et enfin faites-lui décrire un mouvement de la forme du symbole ∞ (indice : lemniscate de Bernoulli).

Checkpoint 4 : la projection sur le noyau de la Jacobienne permet d'exploiter la redondance. La matrice de projection est $I - JJ^{\#}$. Un exemple d'utilisation est le suivant : si q_0 est une "'configuration préférentielle", on peut tenter d'exécuter une tâche tout en restant aussi proche que possible de cette configuration. Pour cela, le calcul de q' est modifié comme suit : $q' = J^{\#} v\_des + (I - JJ^{\#})(q_0 - q)$. Implémentez cette modification avec q_0 égale à la configuration où tous les angles sont nuls. Observez la différence de comportement pour le suivi de trajectoire.

2 Le pendule double

Consultez la page web suivante :

```
http://en.wikipedia.org/wiki/Double_pendulum
```

Vous y trouverez une description des équations du mouvement du pendule double, calculées à partir du Lagrangien du système. L'expression de ce Lagrangien est :

$$L = \frac{1}{2}m(\dot{x}_1^2 + \dot{y}_1^2 + \dot{x}_2^2 + \dot{y}_2^2) + \frac{1}{24}ml^2(\dot{\theta}_1^2 + \dot{\theta}_1^2) - mg(y_1 + y_2)$$

Puis :

$$L = \frac{1}{6}ml^2(\dot{\theta}_2^2 + 4\dot{\theta}_1^2 + 3\dot{\theta}_1\dot{\theta}_2 cos(\theta_1 - \theta_2)) + \frac{1}{2}mgl(3\cos\theta_1 + \cos\theta_2)$$

Le fichier `double_pendulum.py` est un squelette de code permettant la simulation dynamique du pendule double. L'état du système y est décrit par le vecteur `state` (de taille 4), tel que `state[0]`$= \theta_1$, `state[1]`$= p_{\theta_1}$, `state[2]`$= \theta_2$, `state[3]`$= p_{\theta_2}$, p_{θ_1} et p_{θ_2} étant deux grandeurs particulières (des moments généralisés) permettant de simplifier les calculs.

Checkpoint 5 : à partir des équations de la page Wikipédia, implémentez la dynamique du pendule double dans la fonction `derivs` qui calcule les dérivées de θ_1, p_{θ_1}, θ_2 et p_{θ_2} en fonction de l'état courant. Lancez alors la simulation dynamique, qui devrait correspondre au mouvement chaotique du double pendule libre. Ajustez `dt` pour faire correspondre le temps de l'animation et le temps réel (si `dt` est trop petit l'animation peut être lente).

Lorsque le système est actionné (c'est-à-dire lorsqu'il y a des moteurs aux deux articulations), on peut injecter des couples articulaires ("torques") dans l'équation d'Euler-Lagrange de la façon suivante :

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} = (\tau_1 \;\; \tau_2),$$

avec $\theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}$, et $\tau = \begin{pmatrix} \tau_1 \\ \tau_2 \end{pmatrix}$ le vecteur des torques ou couples articulaires. Cette expression conduit à mettre à jour `derivs` en ajoutant simplement les torques (c'est-à-dire les valeurs `t[0]` et `t[1]` renvoyées par la fonction `torque`) dans le calcul des dérivées de p_{θ_1} et p_{θ_2} (`d[1]` et `d[3]` dans le code).

On désire commander le pendule double en couple par retour d'état, et stabiliser ce dernier autour de la position $\theta_{des} = \begin{pmatrix} 45^\circ \\ 45^\circ \end{pmatrix}$ (avec les mêmes conventions d'angles que sur la page Wikipedia). Pour cela on propose dans un premier temps d'utiliser une commande "proportionnelle", c'est-à-dire de la forme $\tau = -k(\theta - \theta_{des})$ avec $k \in \mathbb{R}^+$.

Checkpoint 6 : modifiez la fonction `derivs` de façon à intégrer les torques correctement, et implémentez cette loi de commande proportionnelle. Il n'est pas possible d'obtenir un résultat satisfaisant, mais en ajustant k on peut observer que le système tente de se stabiliser autour de la configuration désirée.

On décide alors de rajouter un terme de commande dérivée, et plus précisément un terme de commande basé sur les moments généralisés (qui dépendent principalement des vitesses angulaires). La nouvelle loi de commande s'écrit : $\tau = -k(\theta - \theta_{des}) - q\mathbf{p}$, avec $q \in \mathbb{R}^+$ et $\mathbf{p} = \begin{pmatrix} p_{\theta_1} \\ p_{\theta_2} \end{pmatrix}$.

Checkpoint 7 : Implémentez cette loi de commande, en ajustant k et q pour obtenir le meilleur résultat possible.

Avec ce contrôleur proportionel-dérivé, il y a nécessairement un décalage entre la configuration obtenue et celle désirée (à cause de l'effet de la gravité). Connaissant l'équation du mouvement de ce système, nous pouvons très simplement ajouter un terme de "feed-forward" à la commande pour compenser les effets de la gravité (il s'agit simplement de rajouter à τ les valeurs qui compensent directement la dynamique naturelle du pendule double).

Checkpoint 8 : rajoutez le terme de feed-forward de compensation de la gravité, et tentez à nouveau d'implémenter d'optimiser les valeurs de k et q afin de converger vers la position désirée le plus rapidement possible. Utilisez également cette méthode pour stabiliser le pendule double en position verticale.

Checkpoint subsidiaire 1 : faites une animation qui démontre le caractère chaotique du mouvement du pendule inverse libre.

Checkpoint subsidiaire 2 : utilisez la cinématique inverse et le contrôleur mis au point pour suivre des trajectoires courbes (cercle, lemniscate, etc.) avec l'extrémité du pendule.