Comment découvrir son corps ?

Lucas Schwab

Mars - Aout 2020

Contents

Remerciements

Introduction

Dans le but de générer une approximation d'un modèle inverse, nous avons suivis un certain apprentissage qui a pour but d'approvisionner une base de donnée qui contient des postures du robot et la position 3D associé au bout de ce robot dans cette posture. Le modèle inverse utilisé est assez simple. Quand il doit donner une posture du robot pour atteindre un but donné, il séléctionne le point le plus proche dans la base et retourne la posture associée. Logiquement, plus cette base est pleine et plus les points couvre l'espace atteignable par le robot, plus notre modèle inverse sera de bonne qualité. Pour remplir la base j'ai suivi une simple méthode d'apprentissage qui consiste à faire:

- Un certain nombre de "Motor Babling" qui consiste à donner une posture aléatoire au robot, tout simplement ici en donnant un angle aléatoire à chacun de ses moteurs, dans la limite de ceux-ci. Cette étape peut permetre à elle seule d'approximer un modèle inverse pour un robot simple (comme un robot 2D avec seulement 2 ou 3 moteurs). Mais cependant l'espace atteignable par le robot n'est pas entièrement visité lorsque la complexité du robot augmente. S'il y a plus de moteurs, il y a aussi plus de chance que le robot soit recourbé sur lui même.

- Puis un certain nombre de "Goal Babling". Pour éviter de ne visiter que l'espace proche du robot (quand il est recourbé sur lui-même), une technique consiste à générer un but à atteindre plutôt qu'une posture aléatoire à essayer. Il y a donc plusieurs algorithmes pour choisir efficacement un but. La donnée qui sera ajouté à la base dans cette étape sera une posture légèrement modifié aléatoirement qui correspond à celle retourné par le modèle inverse pour atteindre le but généré.

Modèle inverse

La première partie à mettre en place est le modèle inverse. Comme il suffit de séléctionner le point le plus proche dans la base de donnée, il faut mettre un place en Nearest-Neighbor. Celui-ci doit répondre à plusieurs contraintes:

- La recherche d'un Nearest Neighbor doit être rapide : Utiliser une méthode naive qui calcule une à une la distance entre le but et tous les points de la base n'est pas envisagable, car dans l'execution de cet apprentissage, le modèle inverse est utilisé à chaque étape de "Goal Babling". Pour maximiser l'efficacité des résultats, un maximum d'étape de goal babling est à faire.

- L'ajout d'un point à la base de donnée doit être possible et rapide : Chaque étape de "Goal Babling" ajoute un point à la base du modèle inverse. Il faut donc que cet ajout soit faisable et si possible rapide.

Le programme est en python, et est basé sur ce projet ce projet qui utilise une bibliothèque appelé "learners". Celle-ci permet l'ajout d'un point dans la base, mais est difficile d'utilisation et, selon l'installation, peut ne pas être rapide.
Une autre bibliothèque assez utilisé pour le Nearest Neighbor est "Sklearn" avec la méthode du KDTree. Cette bibliothèque est assez facile d'utilisation et d'installation, est aussi rapide pour la recherche d'un Nearest Neighbor mais ne permet pas d'inserer facilement un point dans la base. En effect, le KDTree utilise une architecture d'arbre binaire pour ranger les points. Chaque noeud contient un point de la base est un hyper-plan qui coupe l'espace en deux, créant ainsi deux sous-espaces qui correspondent aux deux branches de ce noeud. Pour une utilisation rapide de cet arbre binaire, il est important de le garder équilibré. Ceci est facile à la création de l'arbre, mais cet équilibre est impossible à garantir lors de l'ajout d'un point sans procéder à un rééquilibrage total de l'arbre. C'est pourquoi Sklearn ne propose simplement pas cette méthode.
J'utilise donc une bibliothèque appellée "Rtree" qui utilise la méthode du RTree et est basé sur la librairie "libspatialindex" écrite en C. Le Rtree utilise des rectangles englobant, qui regroupe les points de la base en plusieurs paquets. La recherche d'un voisin proche est simplifié par la structure de l'arbre et donc rapide. Les données sont organisé spécialement pour le stockage sur disque comme une base de donnée.

Algorithmes utilisés

Le modèle inverse ne change pas dans cet apprentissage. Il reste donc trois parties de l'apprentissage qu'il faut créer:

- La génération aléatoire d'une posture dans l'étape de "Motor Babling".
- La génération d'un but dans l'étape de "Goal Babling".
- La modification de la posture retourné par le modèle inverse pour le but généré précédemment.

J'ai utilisé des valeurs aléatoires uniformes pour générer une posture dans le Motor Babling, et aussi une valeur aléatoire uniforme dans un petit interval pour modifier la posture donnée par le modèle inverse. Il reste la génération d'un but à faire, et il existe une multitude de manière de faire. La distribution finale de ces but doit remplir au maximum l'espace atteignable par le robot, sans pouvoir connaître cet espace. Voici les algorithme que j'ai mis en place dans ce projet.

Agnostic goal generation

La génération agnostique de but n'utilise aucun paramètre présumé sur le robot (comme sa taille). A chaque itération l'algorithme garde en mémoire les coordonnées maximales que peut atteindre le robot dans chacun des axes, et génère un point dans cet interval avec un facteur qui permet un peu d'exploration. Le résultat est ainsi une zone de génération de but très proche de la zone atteignable par le robot.

Frontier strategy

La génération agnostique donnera un volume en forme de pavé droit, et dans notre cas la zone atteignable par le robot ressemble plus à une sphère. C'est à cette problématique que répond la stratégie de frontière. Cette stratégie consiste à discrétiser l'espace en carré pour la 2D ou en cube pour la 3D pour determiner quelle cellule (carré / cube) a déjà été visité et s'en servir pour la génération d'un nouveau but. Le nouveau but est généré avec une probabilité p dans des cellules déjà visitées et avec une probabilité 1-p dans une cellule non visitée.
Pour savoir quelle cellule non visitée est choisie, une position déjà atteinte par le robot est selectionnée aléatoirement. Ensuite, une direction est choisie aléatoirement. Cet algorithme va suivre le vecteur déterminé par la position et la direction, et va se déplacer dans l'espace jusqu'à atteindre une cellule qui n'a pas encore été visitée.

Evaluation des algorithmes

Pour pouvoir comparer l'efficacité de certains