



alpha*ai*

L'EDUCATION À L'INTELLIGENCE ARTIFICIELLE

ALGORITHME DES K PLUS PROCHES VOISINS

PREREQUIS

CONNAISSANCES

PYTHON

- Créer une fonction et renvoyer une valeur avec *return*.
- Importer une bibliothèque.
- Parcourir un tableau avec une boucle *for*.
- Prendre une décision avec une condition *if/else*.
- Afficher du texte dans la console avec la fonction *print*.
- Rédiger des affectations et équations.

MATHEMATIQUES

Calculer une distance entre deux points $a(x_1, y_1)$ et $b(x_2, y_2)$ avec la formule :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

LOGICIEL

Avoir le logiciel AlphaI installé sur un ordinateur.

MATERIEL

Un ordinateur avec une carte Wi-Fi pour communiquer avec le robot.

Un robot AlphaI.

INTRODUCTION

Le but de ce TP est de programmer le robot à différentes étapes pour lui permettre d'éviter les murs de son arène en utilisant l'algorithme des K plus proches voisins. Après la mise en place de l'environnement, vous serez guidés pour la création de cet algorithme. Les données d'entrée sont deux zones de la caméra qui donnent la luminosité à gauche et à droite du robot, ce qui lui permet de connaître sa proximité avec un mur, ainsi que son orientation. Les nouvelles données seront comparées avec celles enregistrées pour la prise de décision en autonomie. Si l'entraînement est cohérent, le comportement du robot le sera également.

ETAPES DU TP

Le TP se déroule en deux parties :

- I. Découvrir l'algorithme des K plus proches voisins en exécutant la version incluse dans le logiciel.
- II. Programmer soi-même l'algorithme.

DECOUVERTE DES K PLUS PROCHES VOISINS DANS LE LOGICIEL

PHASE DE PREPARATION



→ Mettez en place une arène : idéalement avec un sol blanc et des murs rouges ou noirs.

ATTENTION : POUR CE SCENARIO, AFIN D'OBTENIR LE MEILLEUR CONTRASTE POSSIBLE, METTEZ UNE SURFACE BLANCHE POUR L'ARENE, ET VEILLEZ A AVOIR UN ECLAIRAGE UNIFORME SUR L'ARENE.

→ Allumez le robot (l'interrupteur se trouve en dessous). Il effectue un petit mouvement lorsqu'il est prêt. Connectez-vous au robot soit en Wi-Fi, soit en Bluetooth (le Bluetooth est préférable lorsqu'il y a plus de 4 robots, ou si de nombreux réseaux Wi-Fi sont déjà présents dans la salle).

Connexion Wi-Fi	Connexion Bluetooth
<ul style="list-style-type: none">- Connectez l'ordinateur au Wi-Fi du robot : cherchez le réseau Wi-Fi qui commence par ALPHAI et se termine par le numéro de votre robot ; le mot de passe est identique au nom du wifi- Dans le menu Outils, vérifiez que « wifi » est coché.	<ul style="list-style-type: none">- Dans le menu Connexions > Bluetooth, sélectionnez dans la liste le nom (numéro) de votre robot.- Si votre robot n'est pas dans la liste, cliquez sur « Mon robot n'est pas dans la liste » et suivez les instructions. Il est alors ajouté à la liste où vous pouvez le sélectionner.

→ Lancez le logiciel en cliquant sur l'icône « **AlphaI** » sur le Bureau ou depuis le menu démarrer.

→ Cliquer sur le bouton « connexion »  pour vous connecter au robot. Vous voyez apparaître son niveau de batterie . Vérifiez que le robot est bien chargé avant de passer à la suite.

→ Dans le menu Paramètres / Charger les paramètres d'exemple, chargez la configuration « Apprentissage supervisé – KNN caméra ».

PHASE D'ENTRAINEMENT

PHASE DE TEST

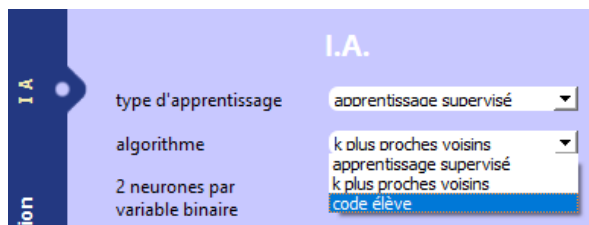
- Après un court apprentissage, réactivez l'autonomie



Si le robot est bien entraîné, il saura éviter les bords par lui-même. Si au contraire il est mal entraîné ou la luminosité dans l'arène n'est pas homogène, les zones de couleur sur le graphique auront des contours plus irréguliers et le robot fera plus d'actions inappropriées.

PROGRAMMER L'ALGORITHME DES K PLUS PROCHES VOISINS

- Désactivez le bouton « Autonome ».
- Dans l'onglet IA, sélectionnez **Algorithme :** `code élève`, cela ouvrira une nouvelle fenêtre vous demandant de nommer le fichier.
- Après lui avoir donné un nom, le fichier apparaît dans l'explorateur Windows : ouvrez-le dans votre éditeur de code préféré.



Ce code présente 3 fonctions déjà existantes : *init*, *learn* et *take_decision*. Ces fonctions seront appelées par le logiciel principal. Seule la fonction *take_decision* est importante pour ce TP.

- Remarquez que quand vous vous déconnectez, un mini-robot simulé apparaît en bas à droite. Si vous le voulez, vous pouvez faire les questions ci-dessous avec ce mini-robot et revenir au vrai robot à la fin du TP ; mais vous pouvez aussi faire tout le TP en restant connecté au vrai robot : comme vous préférez !


PROGRAMMER LE ROBOT DIRECTEMENT

Avant de programmer une intelligence artificielle, commençons par comprendre le principe de la fonction *take_decision*. Vous allez pouvoir modifier cette fonction pour changer le comportement du robot : après chaque modification, sauvegardez votre code et cliquez sur le bouton « Réinitialiser l'IA ». Cette étape est à répéter pour chaque modification du code : **sauver puis recharger le code**.



La fonction *take_decision* accepte un paramètre *sensors* de type *list[float]* et sa valeur de retour doit être un nombre entier (*int*).

La liste *sensors* contient **deux** nombres décimaux qui représentent la vision du robot. Le premier nombre représente l'**intensité lumineuse** dans la partie gauche de l'image et le deuxième nombre représente l'intensité lumineuse dans la partie droite. La valeur de retour de la fonction doit être un nombre entier correspondant à une action possible pour le robot (tourner à gauche, tourner à droite, ou aller tout droit).

Q1. Ajoutez l'instruction `print(sensors)` à l'intérieur de la fonction. Sauvegardez, cliquez « Réinitialiser l'IA » dans le logiciel, et démarrez le robot. La valeur de `sensors` s'affiche maintenant dans la console (cliquer dans la barre des tâches sur l'icône pour afficher la console). Quelle est à peu près la valeur de `sensors` quand le robot est face à un mur ? quand il n'est pas face à un mur ? quand il y a un mur à sa gauche ou à sa droite ? Dans quel intervalle se trouvent donc les valeurs de `sensors` ? 

Q2. Pour l'instant, la fonction `take_decision` renvoie la valeur 0. Quelle est l'action correspondant à ce nombre ? En modifiant cette valeur, trouver les nombres correspondants aux actions *aller tout droit* et *tourner à droite*.

Q3. Utilisez le paramètre `sensors` pour programmer un comportement cohérent du robot : « s'il n'y a pas de mur, je vais tout droit ; s'il y a un mur, je me tourne du côté le plus lumineux. »

PROGRAMMATION DE L'ALGORITHME

CALCUL DE DISTANCE

Programmez une nouvelle fonction `distance(a,b)`. Ses paramètres `a` et `b` sont deux listes de deux nombres décimaux (`float`), représentant les coordonnées (`x` ; `y`) d'un point du graphique. Sa valeur de retour doit être la **distance euclidienne entre ces deux points** (de type `float`).

Pour rappel, si $a = [a_1, a_2]$ et $b = [b_1, b_2]$

Alors la distance euclidienne entre ces deux points est : $d = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2}$.

La racine carrée peut être utilisée en Python en important la bibliothèque `math` (ou `numpy`) en haut du fichier :

```
import math
```

et s'écrit ainsi :

```
y = math.sqrt(x)
```

L'opérateur carré peut être fait de la façon suivante :

```
y = x**2
```

Q4. Programmez la nouvelle fonction `distance`. Puis pour la tester, écrivez **en-dessous** :

```
# compute distance
a = [0, 0]
b = [1, 2]
print("distance", distance(a, b))
```

Sauvez, chargez le code avec « Réinitialiser l'IA » et notez ci-dessous le résultat qui s'affiche dans la console :

CALCUL DE TOUTES LES DISTANCES

Maintenant que vous disposez d'une fonction pour calculer une distance, écrivez une nouvelle fonction *all_distances* qui prend en paramètres un point *a* et une liste de points *train_sensors*, qui représente les données d'entraînement du robot. Cette fonction *all_distances* doit renvoyer la **liste des distances** entre le point *a* et chacun des points contenus dans la liste *train_sensors*. Pensez à utiliser la fonction *distance* pour calculer une distance entre deux points.

Si la liste *train_sensors* est de longueur $n = \text{len}(\text{train_sensors})$, alors la valeur de retour doit aussi être une liste de longueur *n*.

Q5. Programmez la fonction *all_distances*, puis testez-la en ajoutant le code suivant en bas du fichier :

```
# compute all distances
a = [0.4, 0.6]
train_sensors = [[0, 0], [0, 1], [1, 0]]
distance_list = all_distances(a, train_sensors)
print('distances to data', distance_list)
```

Rechargez le code et recopiez votre résultat :

TROUVER LE PLUS PETIT ELEMENT D'UN TABLEAU

Créez une fonction s'appelant *find_minimum* prenant comme unique paramètre une liste de nombres *dlist* (de type *list[float]*) et renvoyant l'**indice du premier plus petit élément** (de type *int*).

Q6. Programmez et testez *find_minimum* avec le code suivant :

```
# minimum in a list
idx_min = find_minimum(distance_list)
```

```
print('index of minimum', idx_min)
```

Notez le résultat :

LE PLUS PROCHE VOISIN

Maintenant que nous avons toutes les fonctions nécessaires, nous allons pouvoir créer la fonction *nearest_neighbor_decision* qui accepte 3 paramètres : *train_sensors*, une liste de points représentant les données d'entraînement du robot ; *train_decisions*, une liste d'entiers représentant les actions associées à chacun des points de *train_sensors* ; et un point *a*, représentant les valeurs de luminosité fournies par la caméra. On rappelle qu'un point est représenté par une liste de deux nombres décimaux qui sont ses coordonnées.

La fonction *nearest_neighbor_decision* doit (en 3 lignes) :

- 1) Calculer les distances entre le point *a* et chacun des points de la liste *train_sensors*.
- 2) Trouver l'indice de la plus petite de ces distances.
- 3) Renvoyer l'action correspondant au point d'entraînement le plus proche du point *a*. Sa valeur de retour est donc un entier correspondant à un code d'action (tout droit, à droite ou à gauche).

Q7. Programmez et testez *nearest_neighbor_decision* avec le code suivant :

```
# KNN
a = [0.4, 0.6]
train_sensors = [[0, 0], [0, 1], [1, 0]]
train_decisions = [1, 2, 0]
decision = nearest_neighbor_decision(train_sensors, train_decisions, a)
print('KNN', decision)
```

Recopiez le résultat :

UTILISATION DE VOTRE ALGORITHME AVEC LE ROBOT

Félicitations, vous avez programmé vous-mêmes l'algorithme des K plus proches voisins dans le cas K=1. Il ne vous reste plus qu'à l'utiliser dans le programme pour entraîner le robot.

Pour cela, recopiez les lignes ci-dessous pour reprogrammer la fonction *take_decision* pour prendre les bonnes décisions, mais également *learn* pour se rappeler des données d'entraînement *train_sensors* et *train_decisions* qui seront créées au fur et à mesure dans le programme principal :

```
train_sensors = train_decisions = None

def learn(X_train, y_train):
    global train_sensors, train_decisions
    train_sensors, train_decisions = X_train, y_train
    loss = 0
```



```

    return loss

def take_decision(sensors):
    if train_sensors is None:
        return 0
    return nearest_neighbor_decision(train_sensors, train_decisions,
sensors)

```

Et à présent, vous pouvez entraîner votre robot !



1. Rechargez votre code



2. Désactivez l'autonomie du robot en cliquant sur l'icône
3. Dirigez le robot en utilisant les flèches du clavier. Tournez quand le robot est proche des murs et allez tout droit sinon.

Les tableaux *train_sensors* et *train_decisions* sont remplis automatiquement par le logiciel à chaque fois que vous donnez un ordre au robot en appuyant sur une flèche.



4. Après un court apprentissage, réactivez l'autonomie
5. Le robot navigue seul dans l'arène. Évite-t-il les murs par lui-même ?

POUR ALLER PLUS LOIN

UTILISER UNE VRAIE IMAGE CAMERA

Pour pouvoir utiliser une image de meilleure résolution, il faut modifier notre fonction *distance*. En effet, celle-ci permet actuellement de calculer la distance entre deux points ayant chacun 2 coordonnées (puisque nous travaillons avec des images de 2 pixels). Il faut donc maintenant pouvoir calculer la distance entre deux images ayant un nombre plus grand de pixels (ce nombre de pixels devant être le même pour les deux images). Pour cela, on peut par exemple utiliser la fonction suivante :

```

def distance(image1, image2):
    nombre_pixels = len(image1)
    somme_carres = 0
    for i in range(nombre_pixels):
        somme_carres = somme_carres + (image1[i] - image2[i])**2
    return math.sqrt(somme_carres)

```

Prendre le temps de bien lire et comprendre cette fonction. Quel est son prototype ?

Grâce à cette fonction, vous pouvez maintenant organiser une course de robots en utilisant leur caméras ! (Choisissez par exemple dans l'onglet Capteurs la résolution de caméra 16x12 et le précalcul « Luminance – Jaune/Bleu – Vert/Rouge »).



K PLUS PROCHES VOISINS

L'algorithme que vous avez écrit utilise le plus proche voisin pour prendre sa décision. Modifiez votre code pour qu'il prenne en compte la majorité des décisions prises entre un nombre $k > 1, k \in \mathbb{N}$. La difficulté réside dans la modification de la fonction trouvant le minimum. Il faut maintenant qu'elle en trouve k et non juste 1.

INTRODUCTION AUX RESEAUX DE NEURONES

Chargez les paramètres d'exemple « Apprentissage supervisé, navigation avec caméra » et relancez un apprentissage. Quelles différences existe-il entre les deux algorithmes ?

CONCLUSION

Notions apprises :

Apprentissage Supervisé : En donnant au robot un jeu de données cohérent et un algorithme, il a été capable de généraliser sa prise de décision à des situations qu'il ne connaissait pas encore en se rapprochant des actions qu'il a mémorisées.

Algorithme du plus proche voisin : En mettant en œuvres des algorithmes classiques comme le calcul d'une distance ou la recherche d'un minimum dans une liste, nous découvrons qu'il est possible de programmer soi-même un programme de *machine learning*.