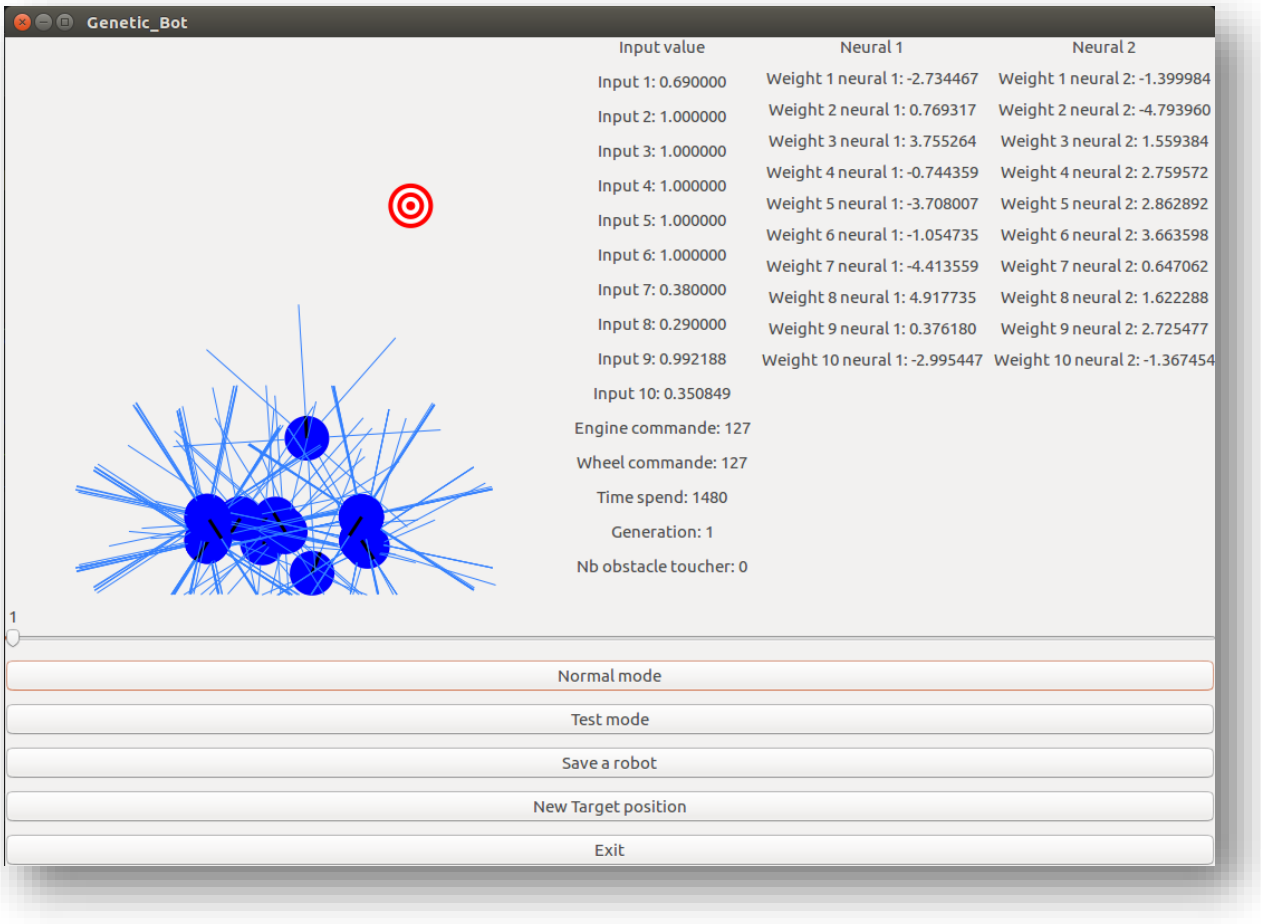
BLANC Swan Sorbonne Université

GENETIC BOT

Projet Linux



Objectif

L'objectif de ce projet est de réaliser un simulateur qui entrainera, le plus rapidement possible, un réseau de neurones servant à piloter un robot pour atteindre une cible, tout en évitant des obstacles.

Algorithme Utilisé

J'ai fait le choix d'utiliser un algorithme génétique pour permettre d'entrainer le réseau de neurones.

Pourquoi ce choix ? Etant donné le poids des entrées du réseau de neurones, celui-ci peut prendre un très grand nombre de valeurs différentes. Les tester une par une prendrait un temps infini. L'utilisation d'un algorithme génétique permet relativement rapidement de trouver la bonne combinaison de valeur et ainsi de trouver une solution approchée à notre problème.

Architecture

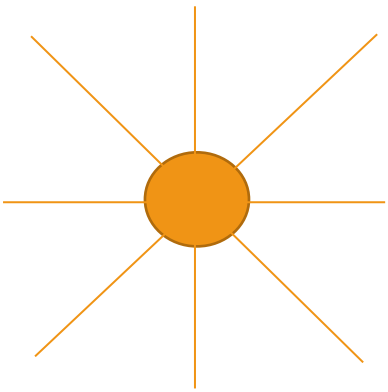
L'architecture globale du programme est une architecture MVC. Modèle Vu Contrôleur.

Model

Robot

Packages Robot

Un robot est composé de 8 capteurs laser placés de la manière suivante :



Un Robot peut avancer, reculer ou tourner. Son déplacement aura une accélération pour atteindre la vitesse désirée.

Réseau de neurones

Packages Neural

Chaque robot a un réseau de neurones d'une seule couche dont chacune contient 2 neurones : un neurone pour avancer ou reculer et un neurone pour tourner.

Le réseau a 10 entrées (les 8 capteurs laser + la vitesse du robot + la distance entre le robot et la cible).

Pourquoi le robot a t-il accès à la distance avec la cible ?

Pour éviter que les robots obtiennent un comportement de déplacement aléatoire.

Pourquoi le robot a t-il accès à sa vitesse ?

Pour que le réseau de neurones puisse gérer l'accélération du robot.

Algorithme génétique

Packages Genetic

ADN d'un robot

Comme il a été précisé précédemment, le réseau est composé de 10 entrées. Chacune ayant 2 neurones, nous obtenons 20 chromosomes qui représenterons le poids des entrées du réseau. Nous rajouterons 2 bias node à l'ADN. Celui-ci aura donc 22 chromosomes.

Attribution des scores

L'attribution des scores se fait sur plusieurs critères :

Le robot a t-il atteint l'objectif ? Le cas échéant, on lui attribue un score de 100 points moins le temps mis en seconde pour l'atteindre (j'ai mis arbitrairement 1 point par seconde).

Si le robot n'a pas atteint son objectif, trois critères sont appliqués :

1. Si le robot ne bouge pas, 12 points lui sont attribués, moins le nombre d'obstacles qu'il a touchés.
2. Si le robot tourne en rond, 25 points lui sont attribués, moins le nombre d'obstacles qu'il a touchés.
3. Si le robot ne parvient pas à se déplacer normalement, 50 points lui sont attribués, moins le nombre d'obstacles qu'il a touchés.

Sélection

Il existe 2 types de sélection :

1. Si aucun robot n'atteint l'objectif, une sélection se fait avec la méthode de rejet sur toute la population.
2. Si un ou plusieurs robots atteignent l'objectif, une sélection se fait sur le ou les robots concernés, avec la même méthode de rejet.

Croisement

Le croisement est simple. Il s'agit de choisir une partie de l'ADN du robot A de taille aléatoire puis une partie de l'ADN du robot B de taille aléatoire, et de les croiser.

Exemple :

À : 1 | 4 | **10 | 20 | 13 | 2**

B : **2 | 7** | 8 | 9 | 15 | 23

AB : 2 | 7 | 10 | 20 | 13 | 2

Mutation

La mutation est également relativement simple. Pour chaque chromosome du fils créé, un chiffre aléatoire est tiré entre 0 et 1. Si ce chiffre est inférieur à une constante, un chiffre aléatoire est attribué au chromosome.

Choix et amélioration

Pour la partie algorithme génétique, j'ai dû faire un choix particulier concernant la sélection. Si un ou des robots parviennent à toucher la cible, je sélectionne les robots concernés. La génération suivante descendra alors du ou des robots qui ont réussi. J'ai fait ce choix, car trop souvent, un ou deux robots atteignent l'objectif mais, au moment du croisement avec les autres robots n'ayant pas réussi, ils enregistrent une perte d'efficacité. Le plus souvent, il(s) se remet(ent) à tourner en rond.

Par contre, par la suite, la qualité de mutation est très importante car elle permettra de garder une diversité dans la population. Les résultats montrent que cette méthode permet d'affiner les recherches mais fait converger assez vite la population sur la même solution.

Il est possible dans le code, de choisir le mode de sélection normal en modifiant la constante « MODE_COPIE » dans le fichier constante.h. Concernant les améliorations possibles, l'attribution des scores mériterait d'être optimisée.

En effet, une méthode courante n'analyse pas assez finement le comportement de chaque robot. A plusieurs reprises, plusieurs robots se retrouvent avec le même score alors que leur comportement est différent.

Communication inter-processus

Package Workers

Le problème d'un algorithme génétique est la population. Pour que l'algorithme trouve une solution le plus rapidement et le plus précisément possible, il faut que la population soit grande. Le problème c'est que chaque robot doit mettre à jour un petit nombre de valeurs toutes les 10ms. Ce temps de calcul cumulé sur une population assez grande (exemple 20) ralentit considérablement la simulation.

C'est pourquoi, j'ai opté pour un calcul parallélisé. L'objectif est de démarrer N threads (exemple: Pour mon ordinateur j'en ai mis 8, car mon processeur comporte 8 cœurs, ce qui permet de faire un véritable calcul parallèle). Chacun des threads va se répartir les robots à calculer. L'idée est de créer une pile qui va contenir la population. Chaque thread prendra un robot dans la pile pour éviter les situations de concurrence, avec l'utilisation d'un mutex pour protéger la pile. Une fois qu'un thread aura fini ses calculs, il le signalera au programme principal à l'aide d'un sémaphore. Quand tous les threads auront fini, le programme principal reprendra son fonctionnement.

Vidéo explicative sur le code

Pour faciliter la compréhension globale du code, j'ai fait le choix de réaliser une vidéo. Des images sont quelquefois plus efficaces que de longues explications.

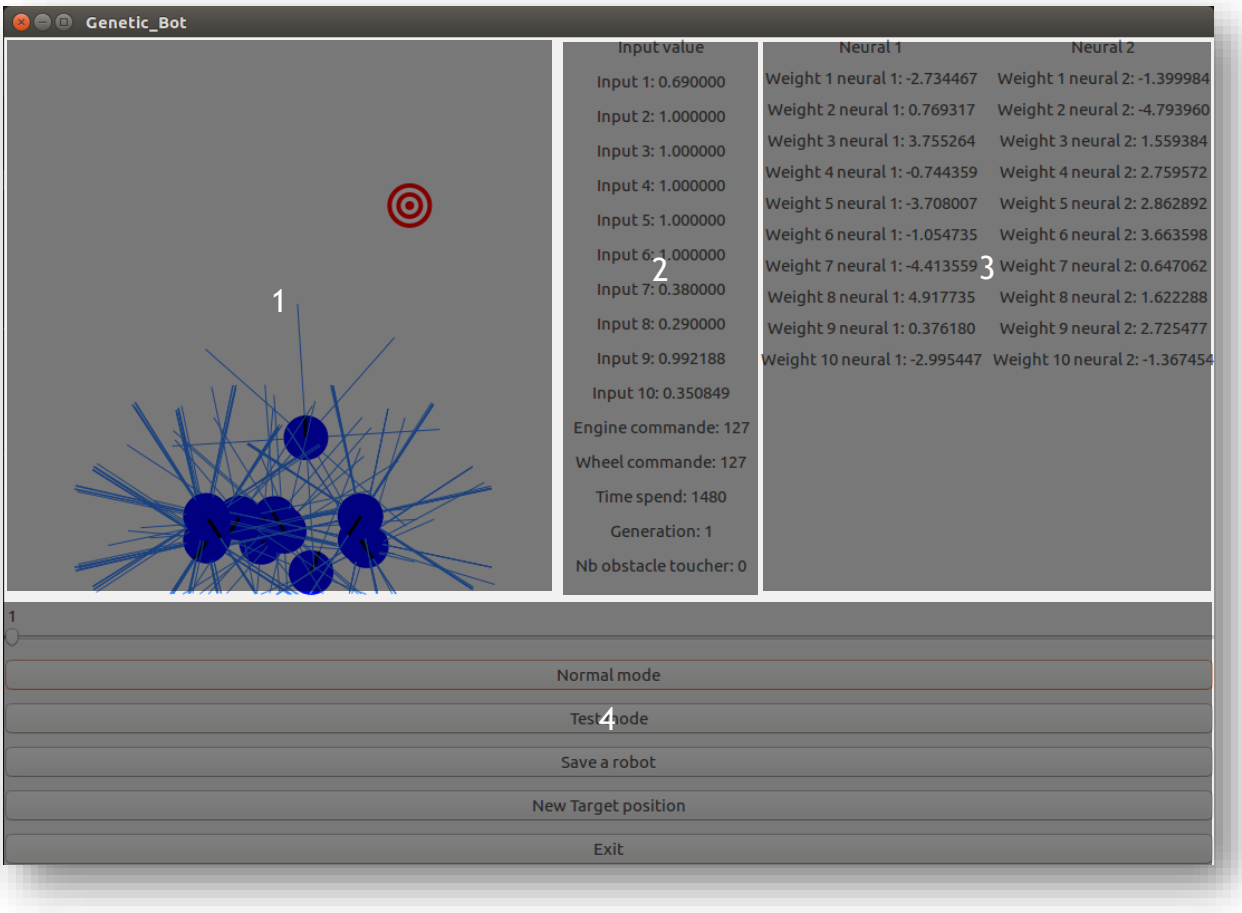
https://drive.google.com/open?id=1WYcbi0yWIaVqZRnUaqWMzNFD7pRsRY9f

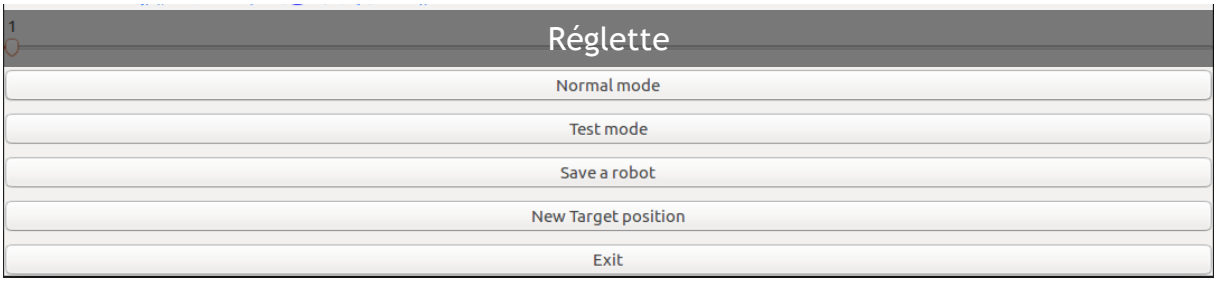
Programme

Interface graphique

*V*oici les différentes parties de l'interface graphique :

1. Partie affichage simulation
2. Affichage des valeurs du réseau de neurones du robot sélectionné
3. Affichage des valeurs du poids des entrées du réseau de neurones du robot
4. Panneau de commande





- Réglette :
  Permet de choisir un robot de la population et de visualiser les informations des capteurs ainsi que du réseau de neurones.
- Normale mode :
  Une population est générée à laquelle est appliqué un algorithme génétique. À chaque appui ça recommence de 0 la simulation.
- Test mode :
  Permet de charger un ou plusieurs robots dans le simulateur pour le(s) tester.
- Save a robot :
  Fonctionne uniquement pendant le mode normal. Permet de sauvegarder un robot choisi.
- New target position :
  Permet de changer la position de la cible. En mode test ça reset les robots chargés est recommence le test
- Exit :
  Quitte le simulateur

Vidéo explicative sur l'utilisation de l'interface

Comme l'interface est relativement simple, une courte vidéo permettra de gagner du temps sur l'utilisation.

https://drive.google.com/open?id=1j4IPwt5t_o6sAyyPHedUWDtauAvAu3l_

Conclusion

La réalisation de ce projet a demandé beaucoup de temps. Non pas du fait de la taille du code mais à cause de la partie technique des réseaux de neurones, des techniques de croisement ou de sélection des algorithmes génétiques, etc.

L'objectif de réalisation du code était de le rendre le plus facilement réutilisable, même si toutes les fonctionnalités des packages implémentés ne soient pas toutes utilisées dans le cadre de ce projet. Ce choix permettra de l'améliorer rapidement ou de changer complètement la situation sans trop de travail.