GADEYNE Guillaume

Compte rendu de projet

Embedded Systems and Robotics

Simulation de “FRED” sur VRep
Guidage par réseau de neuron

Monsieur Patrick HENAFF
09/06/2019

Sommaire.

Présentation du sujet.

Dans le cadre de notre projet 2A “FRED au CEA”, nous avons réalisé un prototype de base robotique mobile, ayant pour objectif de se déplacer de façon autonome. La gestion des déplacements du robot n’ayant pas été gérée cette année, j’ai décidé de m’y intéresser dans le cadre de ce projet.

Les objectifs initiaux étaient :

- modélisation d’un robot avec tourelle motrice et directrice avant
- guidage par réseau de neurone vers un point cible
- suivie d’une ligne discrétisée par une liste de points

Les étapes suivies ont été :

- modélisation et tests de contrôle via python
- adaptation du code présenté en cours à notre cas
- tests vrep avec différents gradients selon les coordonnées du robot et de la cible
- tests vrep avec un “modèle” d’apprentissage
- étude du problème avec scikit-learn
- tests vrep avec un réseau de scikit-learn
- retour à l’online-training

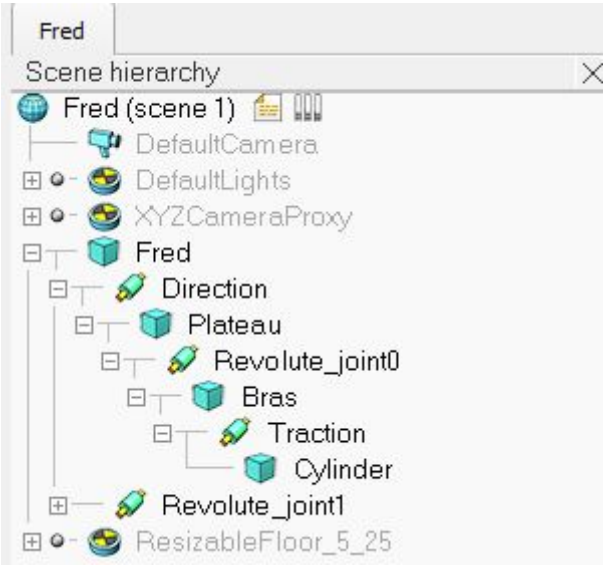
Description du travail effectué.

- modélisation et tests de contrôle via python.

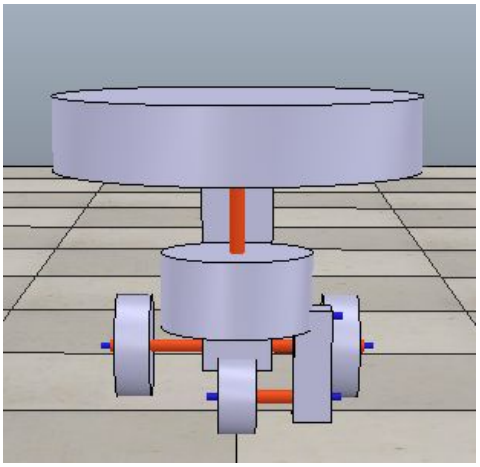
Le robot a été simulé sous VRep de manière très schématique. J'ai jugé suffisant de créer un modèle du même ordre de grandeur que le nôtre, et possédant :

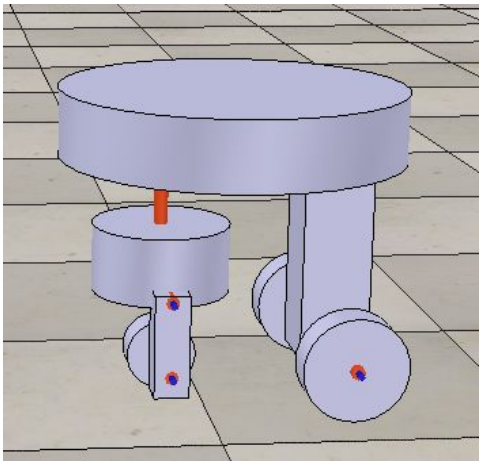
- un plateau qui fait office de châssis
- une tourelle directrice avant
- une roue motrice avant
- deux roues libres arrières

La hiérarchie VRep est la suivante :



Quelques vues du modèle :





J’ai ajusté les poids des différents éléments pour corriger quelques problèmes, notamment:

- oscillation du plateau lors du déplacement (de haut en bas)
- tendance à partir du côté “lourd”

- Adaptation du code présenté en cours à notre cas.

Pour utiliser le code “app-el-pioneer-vrep”, plusieurs modifications évidentes ont été faites :

- modification des noms de scènes et d’objets dans le code
- des méthodes de la classe de simulation du robot

- Tests vrep avec différents gradients selon les coordonnées du robot et de la cible.

Pour ce qui est du réseau de neurone, j’ai d’abord essayé de garder en entrée du réseau la différence entre la position du robot et de la cible, en x et y, sans angle. Ce dernier choix était motivé par le fait que je ne souhaitait pas demander au robot d’arriver sur cible avec un angle particulier : l’objectif final étant de suivre une ligne imaginaire, si celle-ci est bien discrétisée, le robot devrait réussir à la suivre en restant tangent à celle-ci. Les sorties sont les vitesses de rotation des jointures de direction et de traction.

Pour ce qui est du gradient, j’ai essayé différentes formules pour l’odométrie du robot, mais le résultat ne changeait pas : pas d’apprentissage.

(Pas de vidéo mais ici, le robot tournait en rond).

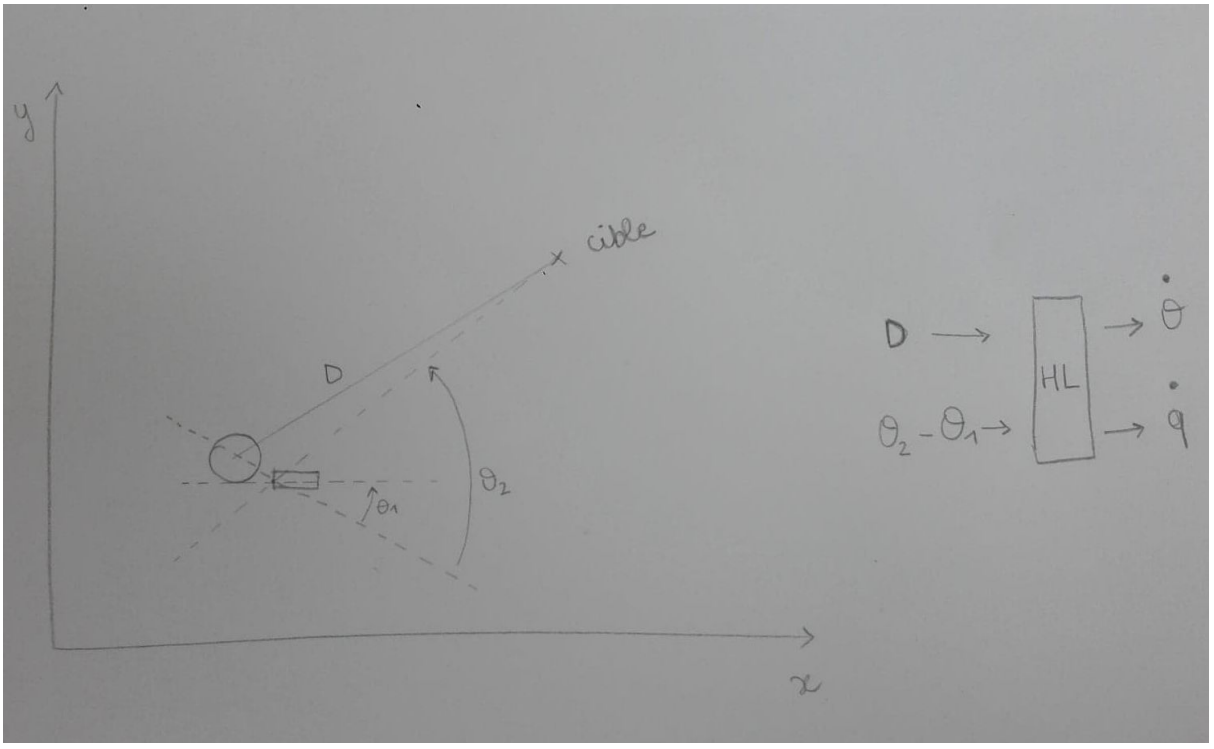
Modification supplémentaire apportée : la tourelle ne peut se situer qu’entre -60° et 60° par rapport à l’axe du robot.

- Tests vrep avec un “modèle” d’apprentissage.

J’ai ensuite voulu essayer de faire apprendre un comportement précis au robot via l’online training. Pour ce faire, j’ai prix les entrées suivantes pour le réseau :

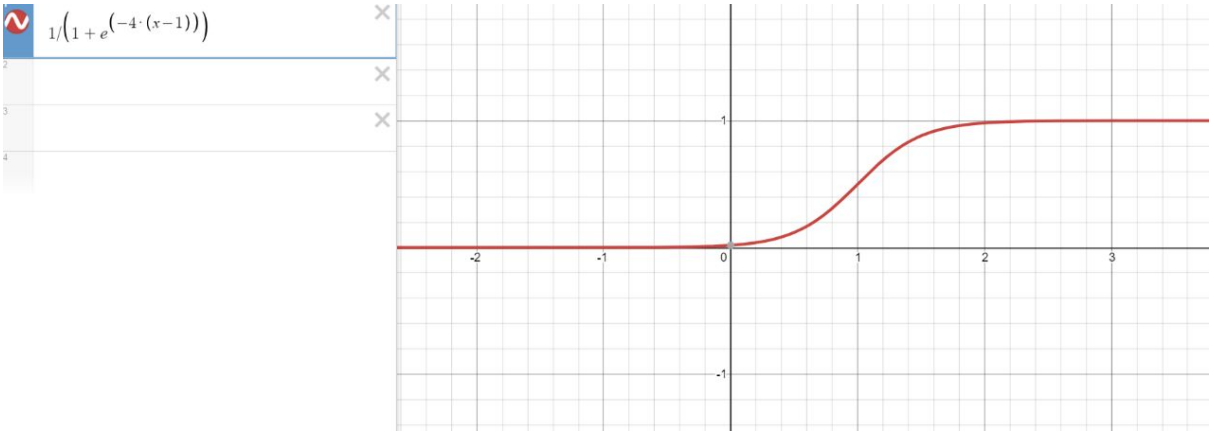
- la différence entre l’angle axe/roue et l’angle axe/cible
- la distance à la cible

Schéma des entrées :



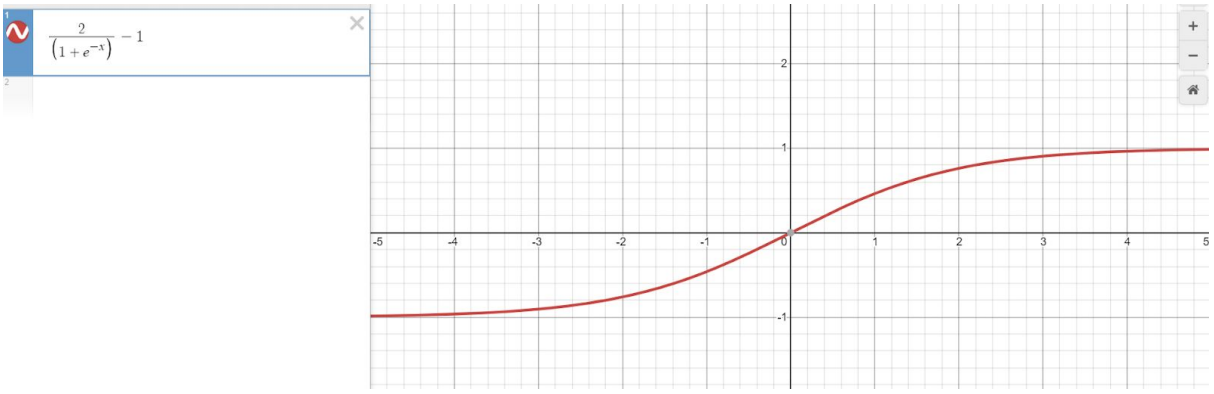
Le modèle choisi est simple :

- ralentir quand on s'approche de la cible (moins de deux mètres) selon une sigmoïde :



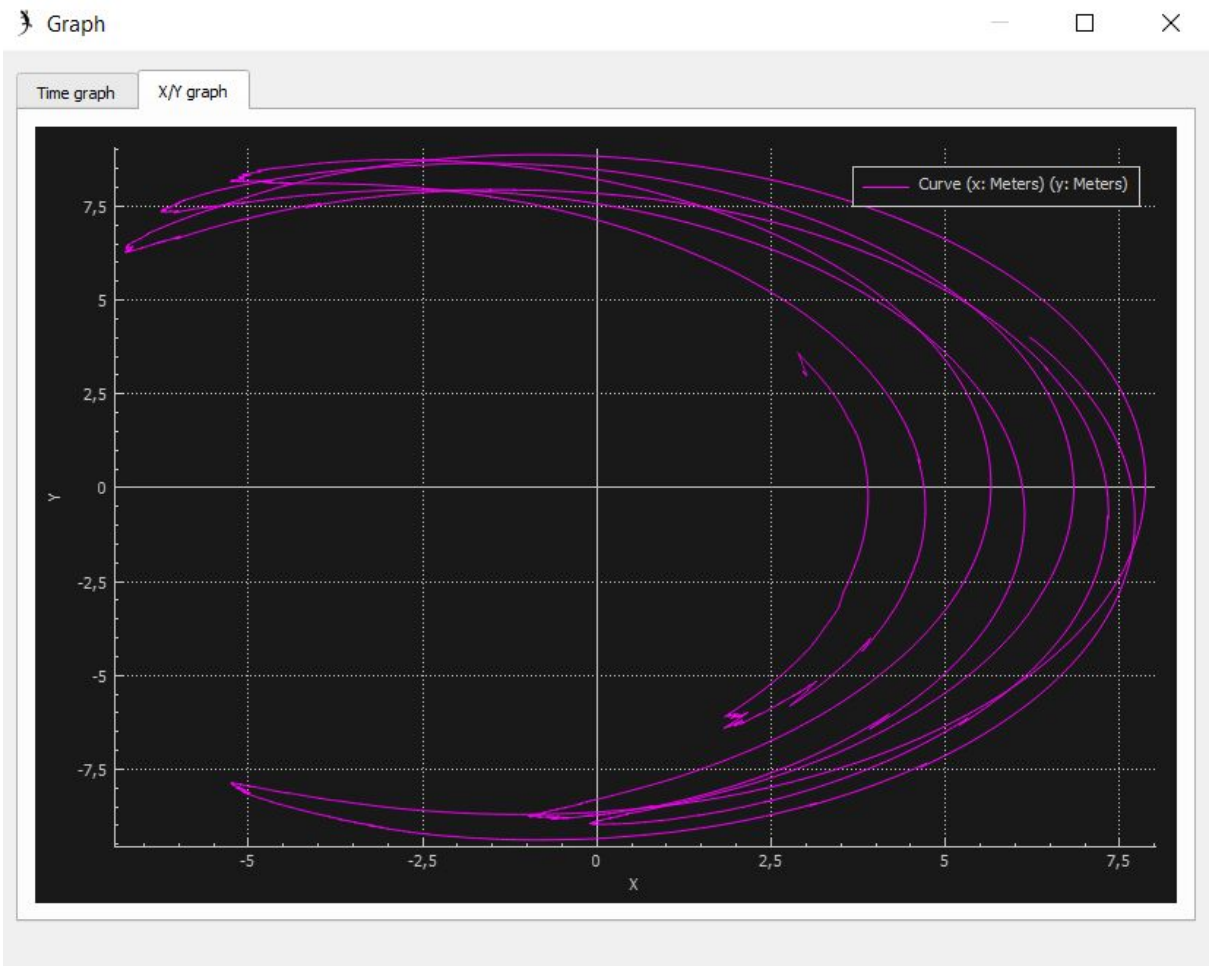
J'ai ensuite ajouté un terme multiplicatif supplémentaire : 1 / (1+ abs(diff_theta)), de manière à ce que le robot ralentisse quand il ne prend pas le bon cap.

- tourner dans le sens horaire si la différence d'angle est positive, et dans le sens antihoraire si elle est négative



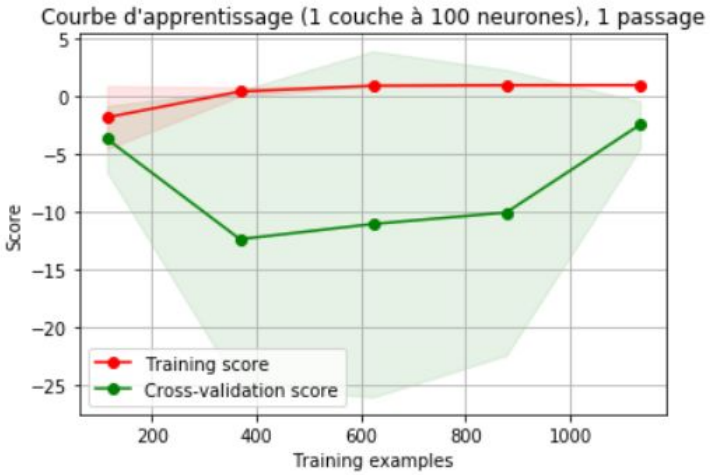
Les valeurs du gradient étaient donc (valeur de sortie - valeur souhaitée) pour les deux sorties. Ce code a marché quelque fois de la bonne manière, mais généralement, le robot se contentait de tourner en rond. J'en ai donc conclu un non apprentissage.

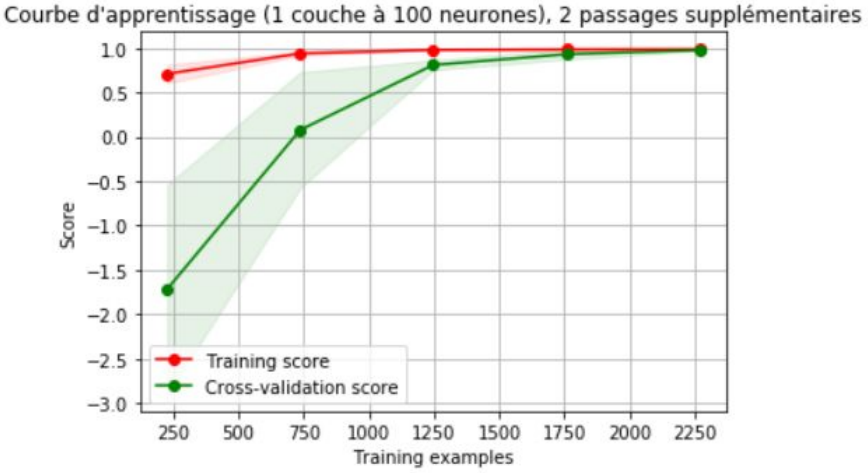
Cependant, ce n'était pas le problème, et l'exemple vidéo (1) aurait dû m'aider à le comprendre. La route suivie était la suivante :



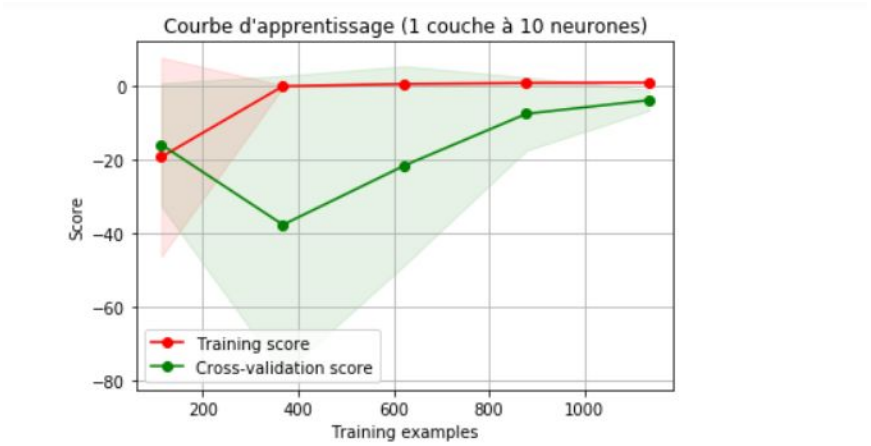
- étude du problème avec scikit-learn.

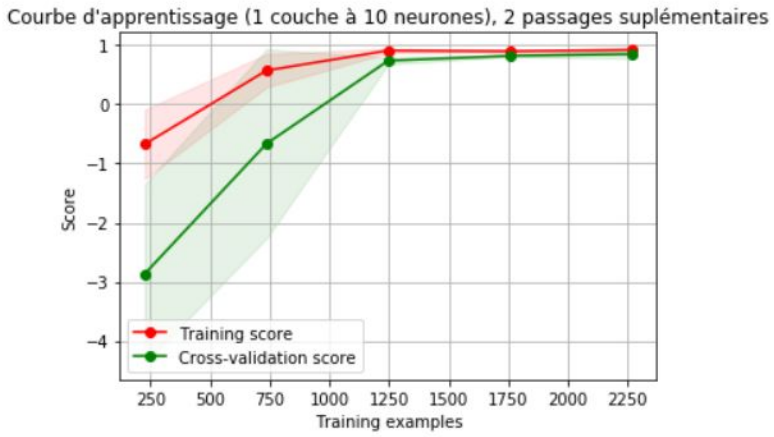
Ne sachant pas comment résoudre mon problème, j'ai choisi de réaliser l'étude du problème via jupyter grâce à la bibliothèque scikit-learn (code en annexe), avec un apprentissage via base d'exemple préalable. Les différentes étapes sont détaillées dans le code, je ne joint donc que les courbes d'apprentissage :





L’apprentissage est plus lent avec 10 neurones dans la couche cachée, mais l’erreur finale semble suffisamment faible :





L’apprentissage ne semble pas être un problème ici, j’ai donc pensé que le problème dans la version précédente venait du fait que sans entraînement préalable, les valeurs d’entrée varient trop fortement pour que l’apprentissage converge.

- Tests vrep avec un réseau de scikit-learn.

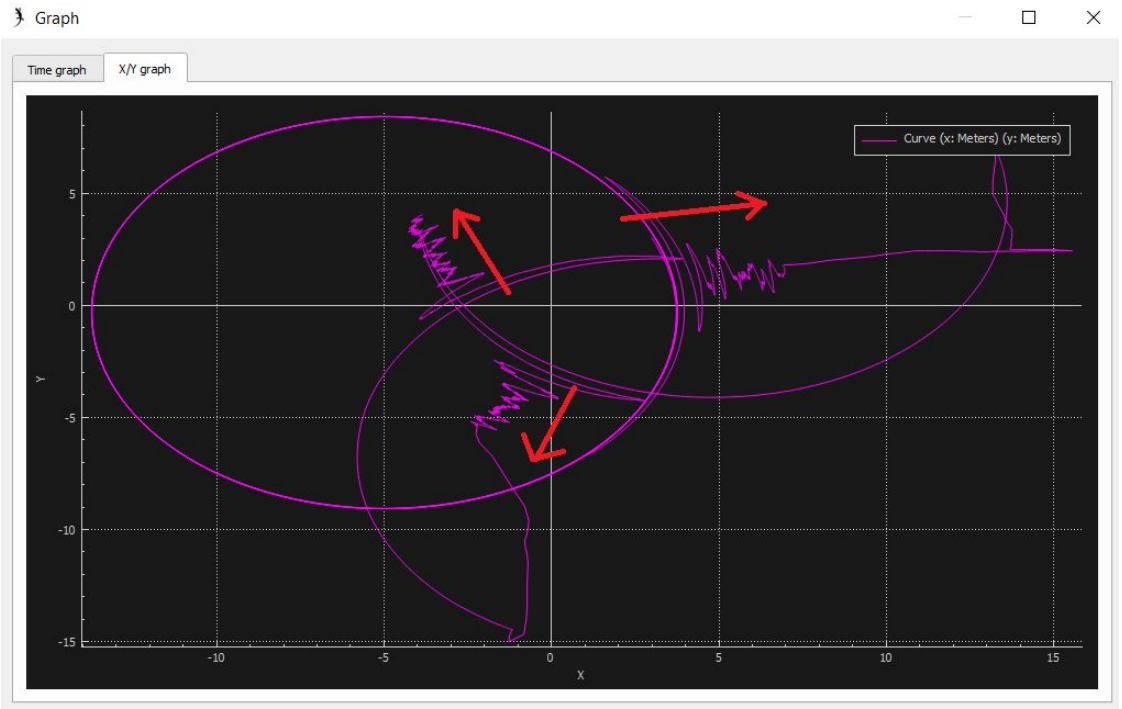
Pour confirmer cette théorie, j’ai essayé une version du code ou je n’utilise pas le réseau de neurone présenté ni la backpropagation : tout est fait avec la bibliothèque scikit-learn.

L’apprentissage est fait au préalable, avec 3 passages complets dans la base d’exemple.

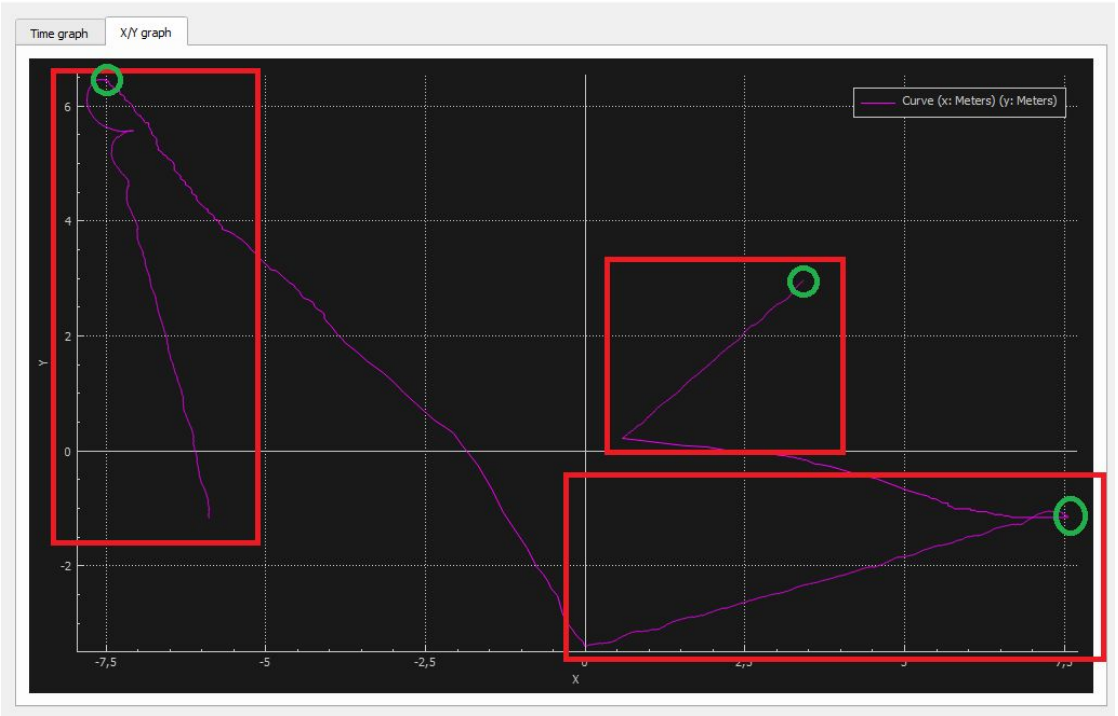
Les tests étaient ici aussi peu concluant, avec de très grandes oscillations, alors que l’apprentissage ne devrait pas poser de problème : j’ai donc ici compris (un peu tard) que le problème venait des valeurs de gains utilisés pour le contrôle.

J’ai donc pu modifier le gain et trouver des valeurs permettant d’avoir un comportement correcte, illustré par la vidéo 2.

Graph correspondant (le déplacement manuel du robot décale le graph, comme on peut le voir sur la vidéo) :



Après plusieurs tests de gain, on arrive à un comportement satisfaisant (départ en vert, cible en (-5, -5)) :

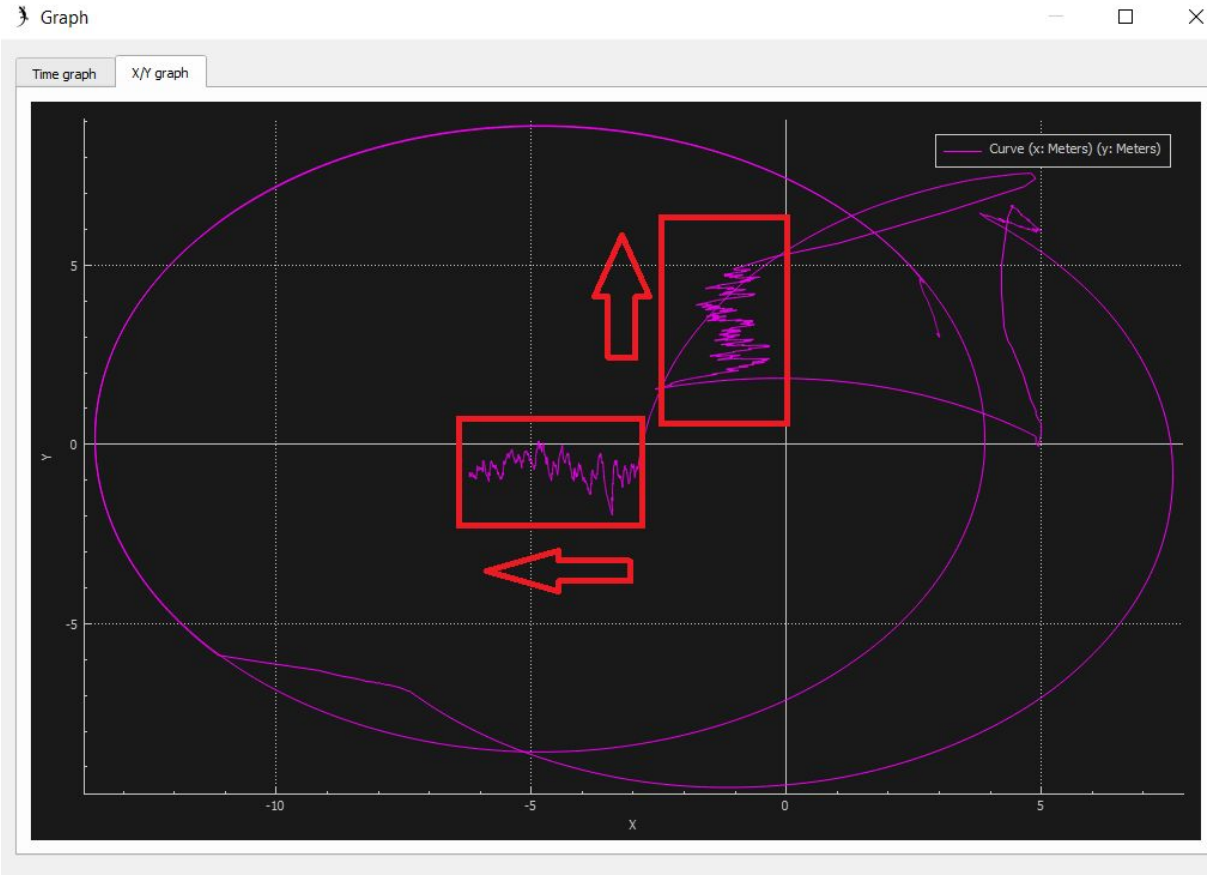


Note : les graphs obtenus pendant la capture présentent des cercles car le robot commence à tourner pendant que le code se lance (la capture prend pas mal de ressources).

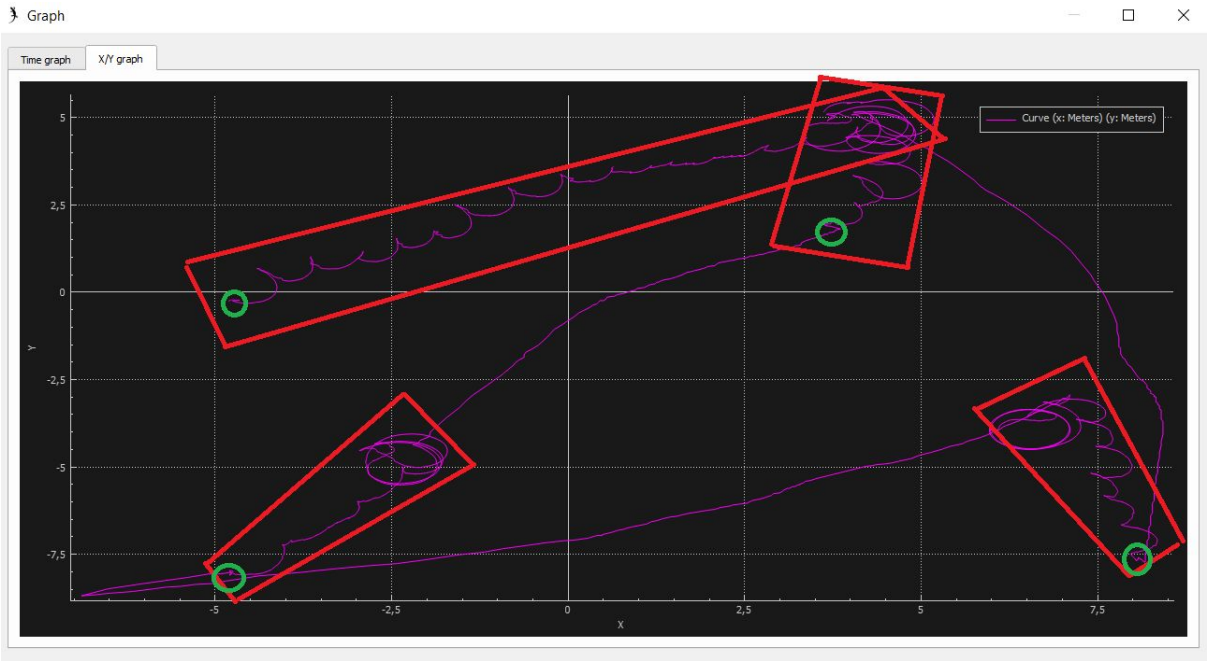
- Retour à l’online training.

Après avoir corrigé le problème de gain avec le code scikit-learn, j’ai pu conclure que pendant l’online-training précédent, les cas qui “marchaient bien” étaient ceux où les poids étaient pas trop mal initialisés avec une différence d’angle faible. Le cas précédemment présenté dans le rapport correspond pour moi à la situation ou le gain sur la direction est trop faible comparé à celui sur la motricité, ce qui fait que le robot ne tourne que quand la différence d’angle est suffisamment élevée pour ralentir le robot et lui laisser le temps de faire son demi-tour.

Ici, le graph obtenu dans la vidéo (3)



Un autre graph, avec d’autres gains :



Critiques et remarques.

Concernant le modèle : étant donné qu’il n’est pas conforme à notre prototype que ce soit en terme de dimension, de poid, et d’autres caractéristiques techniques, il ne serait pas possible d’effectuer l’entraînement sous simulation pour ensuite tester le réseau en cas réel. Toutefois cet exercice m’aura permis de cerner les différences de comportement pendant l’apprentissage entre notre prototype et le pioneer.

Concernant le réseau utilisé : étant donné que le gradient ne dépend pas de la position du robot, mais plutôt d’une courbe de valeurs de sortie que je voulais atteindre, l'émergence de nouveaux comportements est impossible. L’exercice pourrait presque être résumé à : approximer une fonction continue dérivable à l’aide d’un perceptron monocouche. C’était cependant intéressant. De plus, je n’ai, lors de mon étude, influé que sur deux paramètres du réseau : le nombre d’entrées et le nombre de neurones dans la couche cachée.

Concernant les objectifs initiaux : les soucis rencontrés m’ont pris beaucoup de temps à résoudre, ce qui ne m’a pas permis de m’intéresser aux problèmes de suivi de droite. Cependant une solution rapide et efficace pourrait être mise en place : passsage au point suivant dans la liste quand la distance au point actuel passe sous un certain seuil.

Conclusions et perspectives.

En conclusion, je suis satisfait d'avoir réussi à aboutir à un modèle qui permet au robot de se déplacer à un point cible même si les ambitions initiales étaient plus élevées.

Les perspectives futures pour ce projet pourraient être :

- gradients pour la backpropagation dépendants de la position du robot et de son orientation
- étude de l'influence des autres paramètres du réseau (coefficient d'apprentissage, moment de propagation, …)
- ajout d'un angle désiré avec éventuellement une certaine tolérance
- suivi d'une ligne discrétisée par une ligne de points.