REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IVA01/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Image et Vie Artificielle (IVA)**

La neuro-évolution pour le contrôle de robot à pattes

Par :

HADDANA ZAKARIA

Soutenu le 26/06/2022 devant le jury composé de :

Djerou Leila	professeur	Président
Akrour Djouher	MCB	Rapporteur
Boucetta Mebarek	MAA	Examinateur

Année universitaire 2021-2022

Résumé

La robotique existe depuis l'antiquité. Le mot lui-même vient du mot grec "robotos" qui signifie "travailler seul" ou "faire des choses par soi-même". Aujourd'hui, la robotique joue un rôle important dans les secteurs de la fabrication, de la santé, de l'armée, des transports et autres. Avec l'essor de l'intelligence artificielle (IA), les chercheurs ont tenté de concevoir des robots capables d'accomplir des tâches difficile pour les humains. Dans notre travail nous nous sommes basé sur les robots quadrupède. Se sont des robots marcheurs à quatre pattes. Nous avons conçu notre robots simulé dans l'environnement de simulatio Gazebo. Afin de contrôler le robot nous avons utilisé la neuro-évolution qui est une approche souvent utilisée pour le contrôle de robots simulés. Premièrement nous avons utilisé les réseaux de neurones comme contrôleur et nous avons utilisé les algorithmes génétiques pour évoluer les poids des réseaux de neurones.

Mots clés : Gazebo,neuro-évolution,quadrupède

Abstract

Robotics has existed since antiquity. The word itself comes from the Greek word "robotos" which means "to work alone" or "to do things by oneself". Today, robotics plays an important role in manufacturing, healthcare, military, transportation and other industries. With the rise of artificial intelligence (AI), researchers have tried to design robots capable of performing tasks that are difficult for humans. In our work we have based ourselves on quadruped robots. They are four-legged walking robots. We designed our simulated robots in the Gazebo simulation environment. In order to control the robot we used neuro-evolution which is an approach often used for the control of simulated robots. First we used neural networks as a controller and we used genetic algorithms to evolve the weights of the neural networks.

Key words : Gazebo,neuro-evolution,quadruped

ملخص

اليوم تلعب الروبوتات دورا مهما في التصنيع و الرعاية الصحية و الجيش و النقل وغيرها من الصناعات مع ظهور الذكاء الإصطناعي , حاول الباحثون تصميم روبوتات قادرة على أداء المهام التي يصعب على البشر القيام بها ,إعتمدنا في عملنا على روبوتات الرباعية ,هم روبوتات المشي بأربع أرجل لقد صممنا روبوتنا الخاص وذلك في بيئة المحاكاة جازيبو . من أجل التحكم في الروبوت إستخدمنا التطور العصبي وهونهج يستخدم غالبا للتحكم في محاكاة الروبوتات .في البداية إستخدمنا الشبكات العصبية كوحدة تحكم و إستخدمنا الخوارزميات الجينية لتطوير أوزان الشبكات العصبية .

كلمات مفتاحية: جازيبو, تطور العصبي, روبوتات الرباعية

Remerciements

Nous voulons exprimer par ces quelques lignes de remerciements à la fin pour ce modeste travail, une grande gratitude à :

Nous commençons par remercier **Akrour Djouher** qui nous a fait l'honneur d'être notre mentor pour son suivi et son énorme soutien et toute l'aide dont elle dispose amené.

Adresser également nos sincères remerciements aux membres des jurys pour avoir bien voulu revoir et juger ce travail.

Nous ne laissons pas passer cette opportunité, sans remercier le chef de service **Tibermacine okba**.

Enfin, nous remercions tous ceux qui nous ont aidés de près ou de loin pour que notre travail soit à la hauteur.

Table des matières

Table des figures

Première partie

Introduction générale

Les robots quadrupèdes sont de plus en plus populaires dans notre vie quotidienne. Ils sont utilisés dans diverses industries telles que l'agriculture, l'armée et la santé. Le robot quadrupède est un robot à quatre pattes qui marche sur ses jambes. Le robot quadrupède a été étudié et développé dans plusieurs centres de recherche. Ces robots à quatre pattes ont généralement des pattes à trois articulations. Chaque articulation a deux degrés de liberté (DOF). Les DOF sont les angles entre les articulations. Ces articulations permettent au robot d'avancer, de reculer, de gauche ou de droite.

Nous trouvons dans la littérature plusieurs méthodes qui permettent de contrôler les robots. Le principe ici est de faire bouger les jointures en appliquant des forces de rotations. Les chercheurs utilisent généralement la neuro-évolution qui est une approche qui se base sur l'évolution des réseaux de neurones artificiels par les algorithmes génétiques. Ce type d'approche se situe dans le type d'apprentissage par renforcement. Notre objectif principal dans ce mémoire est donc de faire en sorte que le robot quadrupède apprenne à marcher automatiquement à longue distance avec des approches d'apprentissage, dans notre cas, nous utilisons un réseau de neurones artificiels pour contrôler les mouvements du robot.[3]

Les réseaux de neurones artificiels (ANN) s'inspirent de la façon dont les neurones communiquent entre eux dans notre cerveau. Les ANN sont conçus pour imiter la capacité du cerveau à reconnaître des modèles et à résoudre des problèmes. Ils sont également connus sous le nom de réseaux de neurones artificiels car ils sont composés de plusieurs couches de nœuds interconnectés. Chaque nœud représente un neurone dans le cerveau. Les algorithmes génétiques (AG) quant à eux sont des modèles informatiques inspirés de l'évolution biologique. Ils imitent la sélection naturelle par mutation aléatoire et recombinaison. Les AG sont largement utilisés dans les problèmes d'optimisation tels que l'apprentissage automatique, la planification, la robotique et la finance.[3]

Nous utilisons un simulateur de robots appelé Gazebo comme environnement de simulation. Gazebo est un logiciel de modélisation 3D qui nous permet de créer des modèles réalistes de bâtiments, de ponts et d'autres structures. Le programme est devenu très populaire au fil des ans, notamment parce qu'il offre un large éventail de fonctionnalités et d'outils concernant la simulation de robots.

Nous structurons notre travail dans ces trois chapitres :

Dans le **premier chapitre** : dans ce chapitre, nous parlerons des robots quadrupède ainsi que leurs types, dont chaque type a un domaine de travail et un rôle principale. Nous parlerons aussi des simulateur robotiques tridimensionnels.

Dans le **deuxième chapitre** : dans ce chapitre, nous parlerons des contrôleurs robotiques et plus précisément la neuro-évolution. Nous présenterons les réseaux de

neurones artificiel et ses modèles mathématiques ainsi que ses types et enfin nous présenterons les un rappel des algorithmes génétiques.

Dans le **troisième chapitre** : dans ce chapitre nous parlerons du système de conception et de l'implémentation de notre travail, comment nous avons implémenté notre programme pour faire marcher le robot automatiquement en utilisant des approches d'apprentissage, nous parlons de la façon dont le robot marche et comment il met à jour son mouvement.

Chapitre 1

Robotique et environnement de simulation

1.1 Introduction

L'implantation de robots dans les ateliers a mis en évidence différents problèmes liés à l'utilisation et à la gestion des sites robotisés dont le manipulateur est l'un des composants. Afin de contribuer à la maîtrise de ces problèmes, différents outils ont été développés. L'objectif de ces outils est d'apporter une aide à l'utilisateur ou au concepteur pour résoudre les différents problèmes qui se posent lors des différentes phases de son travail. Ces problèmes concernent par exemple, le choix du robot en fonction des tâches visées, l'implantation de celui-ci dans son site, les méthodes de programmation de tâches. Ces différents problèmes peuvent être résolus ou potentiellement traités par l'utilisation de systèmes de **CAO** robotique. En effet, ces systèmes offrent de puissants outils graphiques qui permettent de traiter facilement certains des problèmes cités. Ils permettent en outre, au moyen de simulateurs graphiques, de programmer et de simuler les tâches. Ainsi, les problèmes d'accessibilité de la tâche et d'évitement de collisions peuvent être vérifiés lors de la simulation ce qui permet de réduire la phase de vérification sur le site réel[6].

1.2 Le robot

Un robot est un moteur mécatronique (alliant dynamique, électronique et bureautique) concu pour fixer infailliblement des taches imitant ou reproduisant, entre un ceinture chaste, des actions humaines. La conception de ces systémes est l'outil d'une érudition architecte, confinement de l'automatisme nommé robotique.

Il existe deux types de robots [17]

— Le robot assistant sans forme humaine
— Le robot androïde ou animale

1.2.1 Historique

Le terme robot apparaît pour la première fois dans la pièce de théâtre (science-fiction) R. U. R. (Rossum's Universal Robots), écrite en 1920 par l'auteur Karel Čapek1. Le mot a été créé par son frère Josef à partir du mot tchèque **(robota)** qui signifie (travail, besogne, corvée).
Les premiers robots industriels apparaissent, malgré leur coût élevé, au début des années 1970. Ils sont destinés à exécuter certaines tâches répétitives, éprouvantes ou toxiques pour un opérateur humain : **peinture** ou **soudage** des carrosseries automobiles. Aujourd'hui, l'évolution de l'électronique et de l'informatique permet de développer des robots plus précis, plus rapides ou avec une meilleure autonomie. Industriels, militaires ou spécialistes chirurgicaux rivalisent d'inventivité pour mettre au point des robots assistants les aidant dans la réalisation de tâches délicates ou dangereuses. Dans le même temps apparaissent des robots à usages domestiques : aspirateur, tondeuses, etc.
L'usage du terme **(robot)** s'est galvaudé pour prendre des sens plus larges : **automate distributeur**, **dispositif électro-mécanique** de forme humaine ou animale, logiciel servant d'adversaire sur les plateformes de jeu, bot informatique .[17]

1.3 Morphologie des robots

Le robot est doté d'un squelette avec un ou plusieurs membres ainsi que d'un ordinateur qui fait office de cerveau. Il est aussi doté d'articulations qui relie deux segments et qui permet de faire des rotations dans 3 axes.
Les articulations du robot consultent les composants mobiles du robot qui provoquent des mouvements relatifs entre les hyperliens adjacents. Dans ce contexte, des hyperliens sont utilisés pour consulter les contributeurs inflexibles reliant les articulations pour un fonctionnement propre et correct. Cela va révéler que le bras de robot peut utiliser un mélange d'hyperliens et d'articulations pour une meilleure fonctionnalité. Il est essentiel de s'assurer que les joints du manipulateur fonctionnent en conséquence pour s'assurer qu'il y aura précision et exactitude même comme dans une application.
Les articulations du robot sont également appelées axes. Ils sont essentiels et disponibles en ver-sion accessible pour s'assurer que le mouvement du bras n'est pas limité ou gêné. En savoir plus sur les articulations du robot vous aidera à décider du type de bras de robot dont vous avez be-soin pour travailler en fonction des

responsabilités dont vous avez besoin.

Le degré de liberté fait essentiellement référence au nombre de façons dont un objet peut se dé-placer. Maintenant, en termes simples, il existe 2 moyens fondamentaux de mouvement :

— Translation
— Rotation

La **translation** peut être décomposée en 3 types de mouvements :

— En avant en arrière
— Gauche droite
— Haut-Bas

La **rotation** peut également être décomposée en 3 types de mouvements :

— Terrain
— Embardée
— Rouler

Ce sont les 6 degrés de liberté. Vous pouvez obtenir une meilleure compréhension à partir de-cediagramme.[16]

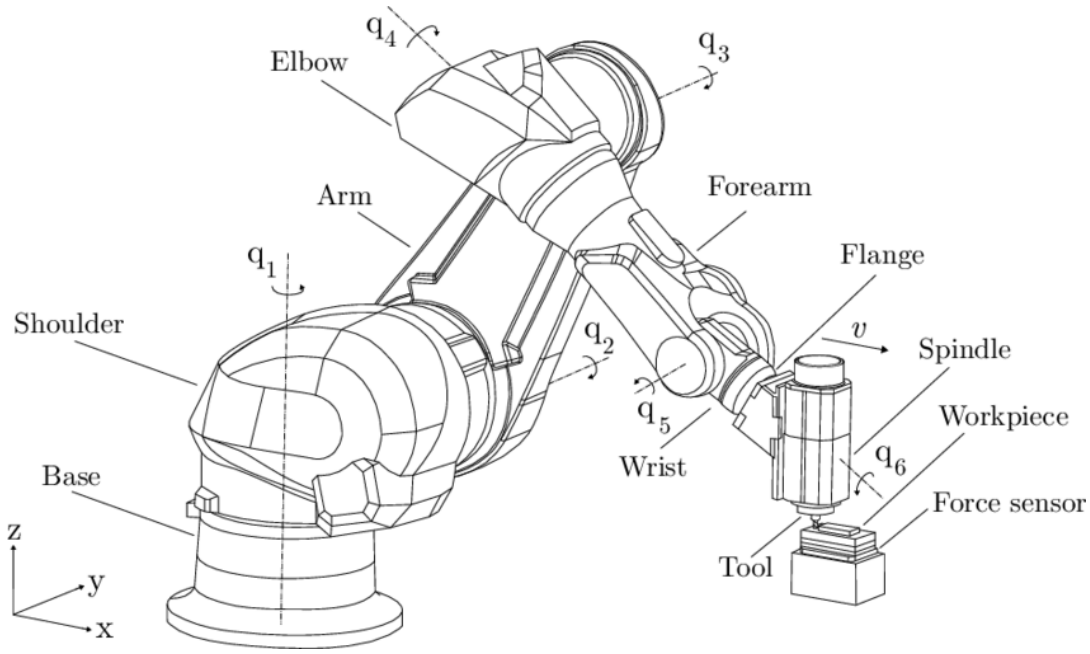


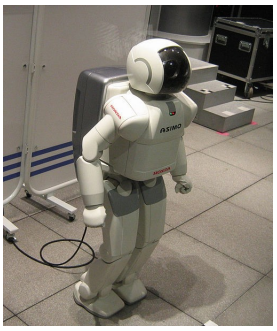
FIGURE 1.1 – Diagramme de 6 axes de Robot[8]

Selon le nombre et la manière de composition de segments, d'articulations et de degrés de liberté différentes formes de robots peuvent se former. Nous citons a titre d'exemple :

— Robots serpent
— Hexapode

— Les Robots mobiles autonomes (**AMR**)
— Les véhicules à guidage automatique (**AGV**)
— Les bras robotiques
— Les cobots
— Les robots humanoïdes et Les hybrids
— Le robot Titan
— Le robot oscillator
— Le robot Dochiba
— Le robot Tohoko
— Le robot Big dog de Boston Dynamics

(a) Robot serpent

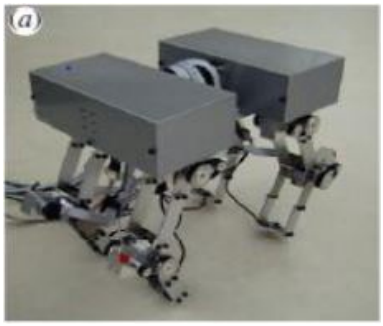


(b) Robot humanoide

(c) Robot hybrid

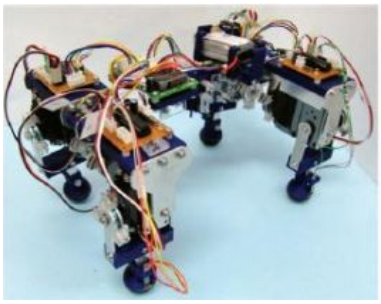
(d) Robot chien

(e) Robot big dog

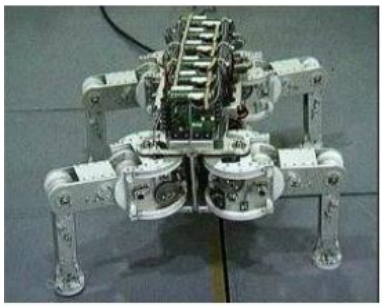


(f) Robot Dochiba

FIGURE 1.2 – Exemples de différentes formes de robots



(a) Robot oscillator



(b) Robot Titan

FIGURE 1.3 – Exemples de différentes formes de robots

1.4 Le Rôle/Objectif des robots

Les robots sont construits entre verser des travaux dangereuses et pénibles et quelquefois équivalent altruistes parmi l'type. L'avantage, c'est qu'ils accomplissent ces différentes fonctions d'une lèvre davantage ouverte et le font de canalisation inoccupé. Depuis les années 70, les robots sont devenus mobiles, possèdent un mannequin embarqué, sont pourvus d'une caméra et, le encore innombrable, ils sont capables de disputer.
à cause l'école, les robots servent à provenir des offices répétitives supposé que les suite de création sont aisément cohérent à des modifications. Ils offrent :

- Régularité entre production des charges.
- Pas de besoin de échiné ou de grippe.
- opposition aux ambiance décisif (gaz nocifs, températures extrêmes, radiations).

FIGURE 1.4 – Robot industrielle

Dans les transports les robots permettent de transporter de biens ou des personnes sur des trajets prédéfinis ou Programmés.

- Flexibilité d'exploitation (24h/24, 7j/7, gestion optimisée de flotte).
- Réduction des coûts d'exploitation (pas de chauffeur, moins d'erreurs humaines).
- Investissements limités (Véhicules plus simples, Infrastructure légère).
- Pas de pollution atmosphérique.

— Pas de bruit.

FIGURE 1.5 – Robot de transport

à cause le pelote curative les robots on peut appointer des atrophie à écart et ressources des nomination en 3D en HD ce qui fournie une élégance de style et élection de la ville d'interposition. On peut de même façonner des hérédité mécanisée des gestes en supprimant les tremblements.

FIGURE 1.6 – Robot médicale

Dans les domaines militaires, les robots autonomes ou commandés à distance sont utilisés pour des opération de reconnaissance, d'attaque ou de déminage

— Pas de problème moral en cas de capture ou destruction.
— Mise en réserve en tant de paix.
— Pas de formation longue et couteuse (mise à jour).
— Réduction des couts de fabrication de l'armement.

Les robots humanoïdes quant eux ils ont un rôle très important puisque ils ressemblent à l'être humain. Cette forme de robotique est souvent la principale motivation des roboticiens. Ils pourraient être employés dans les tâches qui demandent une

FIGURE 1.7 – Robot militaire

étroite collaboration avec l'homme. La ressemblance à l'être humain est un facteur essentiel de l'acceptation de la machine par l'homme.

— Assistance à la personne, travail collaboratif avec les humains.
— compétent de filer, passer/exécuter les escaliers.
— voir les visages, langue, subir la représentant.

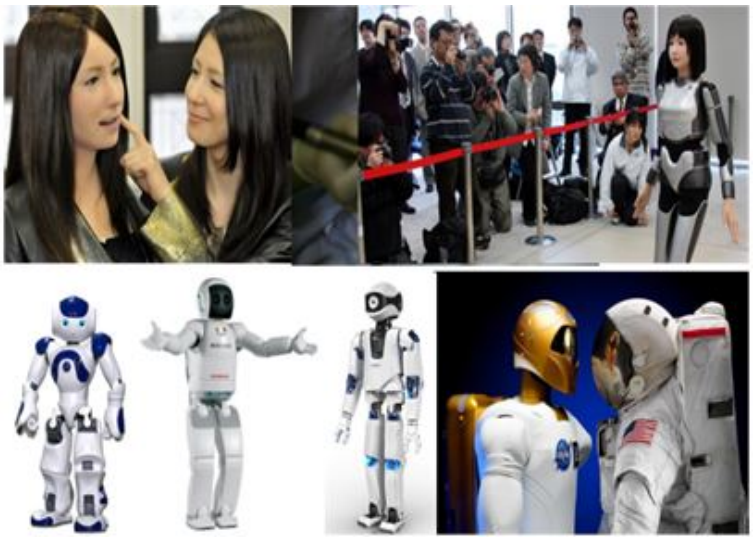


FIGURE 1.8 – Robot Humanoid

1.5 Le robot quadrupède

La quadrupédie est un mode de locomotion terrestre par lequel un organisme se meut préférentiellement sur quatre membres. Le terme de quadrupède ou quadripède désigne les espèces d'animaux qui marchent à quatre pattes.[17]

Les robots quadrupèdes peuvent être utilisé et appliqué dans :

— la robotique industrielle.
— la robotique domestique.
— la robotique militaire.
— la robotique sociale.

— la robotique scientifique
par exemple pour l'exploration de l'espace (aérobot), des fonds marins (robots sous-marins au-tonomes), dans les laboratoires d'analyse (robotique de laboratoire), etc.
— la robotique de transport(de personnes et de marchandises)
par exemple **ROPITS** (Robot for Personal Intellingent Transport System), Robosoft, Ro-boCourier, etc.

1.6 Simulateur 3D

D'abord, pour bien saisir ces affirmations, il importe de comprendre ce qu'est la simulation 3D et à quelle fin elle peut être utilisée. La simulation 3D est une visualisation animée du comportement de la cellule robotique sur ordinateur. Le simulateur 3D est l'outil logiciel qui permet d'assembler et de tester l'environnement de production avant sa mise en marche.

Par le passé, la conception des cellules robotiques était faite uniquement à partir des logiciels de CAO . Le designer positionnait le robot au meilleur de sa connaissance sans pouvoir tester les programmes robots. Une fois le système robotisé en place, nous nous retrouvions souvent avec des robots qui n'avaient pas la capacité d'atteindre la pièce correctement. La simulation permet entre autres d'éviter ces problèmes en offrant la possibilité de tester les programmes robots virtuellement avant qu'il ne soit trop tard. La simulation vient donc influencer la modélisation de la cellule.

La simulation 3D est une visualisation animée du comportement de la cellule robotique sur ordinateur. Le simulateur 3D est l'outil logiciel qui permet d'assembler et de tester l'environnement de production avant sa mise en marche.[11]

1.6.1 Environnement de simulation 3D

Le simulateur rend possible la modélisation et la planification du système robotique complet en un environnement tridimensionnel. Le simulateur comporte généralement une ou plusieurs librairies de robots. Une fois les modèles robotique importés, il permet de sélectionner et de tester le robot qui a la bonne portée pour l'application désirée. C'est-à-dire que le simulateur peut littéralement être utilisé pour essayer un robot. Dans le cas où le robot n'est pas en mesure d'atteindre certains points, un autre robot de plus longue portée peut être sélectionné dans la librairie et se superposer à celui qui ne fonctionne pas. Les programmes peuvent être transférés au nouveau robot et poursuivre le développement.[11]

1.6.2 Objectif

Le simulateur rend possible la programmation hors ligne de robots par opposition à la programmation en mode point à point. Par le passé, les robots ne se programmaient qu'en mode point à point, c'est-à-dire que le robot devait être déplacé avec la manette de programmation à chaque position où il devait passer pour que la position puisse être enregistrée.

Cela avait premièrement pour effet d'arrêter la production pour l'enseignement des trajectoires robots (ce qui peut prendre de plusieurs heures à plusieurs jours).

Deuxièmement, la programmation devait être réalisée seulement lorsque toutes les pièces étaient fabriquées et installées dans leurs gabarits de maintien.

Avec la simulation 3D, la programmation robot peut être réalisée complètement dans l'environnement virtuel en temps masqué par rapport à la fabrication et l'installation. De plus, dans la plupart des simulateurs, il est beaucoup plus rapide et précis de déplacer le robot qu'avec la manette de programmation en mode point à point. Il en résulte des gains de temps de mise en route importants.[11]

1.7 Le contrôleur

Le contrôle de robots est souvent présenté comme un problème à deux niveaux :

— un niveau « haut » associé aux aspects de raisonnement ou encore d'intelligence artificielle .

— un niveau « bas » associé aux aspects d'exécution et plus précisément aux problèmes de commande des actionneurs.

Cette perfection argent que conceptuellement alléchante ne permet pas de revaloir mémoire du niveau d'enchevêtrement élevé comme les aspects de réflexion et d'perpétration et la prôner des problèmes complexes est de surtout en principalement accommodée à cause l'échafaudage de comportements aimables laquelle l'repliement peut contraindre à la achèvement de comportements plus complexes.

1.8 Conclusion

Dans ce chapitre nous avons présenter brièvement la robotique, son historique et ces applications. Nous avons présenté les types de robots et plus particulièrement les robots quadrupède qui sera le type choisi dans notre travail. Enfin nous avons défini un environnement de simulation tridimensionnel, ces objectifs et ces caractéristiques. Dans le prochain chapitre nous allons présenter la robotique évolutionnaires et les systèmes de contrôles de robots a base des réseaux de neurones.

Chapitre 2

La Neuro-Évolution

2.1 Introduction

Dans ce chapitre on va aborder les systèmes de contrôle conçus pour contrôler les robots quadrupèdes. Dans notre travail, nous allons opter pour la neuro-évolution. C'est-à-dire que nous allons utiliser les réseaux de neurones artificiels comme contrôleur et les algorithmes génétiques comme méthode d'apprentissage.

Un réseau neuronal artificiel est un système dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques. Par ailleurs, les algorithmes génétiques sont des techniques d'optimisation utilisées dans ce cas pour l'apprentissage des réseaux de neurones afin de contrôler les robots.

2.2 Réseaux de neurones artificiels

Les réseaux de neurones artificiels (RNA) se sont des systèmes dont la conception est à l'origine schématiquement inspirée du fonctionnement des neurones biologiques, et qui par la suite s'est rapproché des méthodes statistiques.

C'est un calcul (ou algorithme), généralement réalisé à l'aide d'un ordinateur, dont le résultat reproduit ou prévoit aussi fidèlement que possible, le comportement de n'importe quel processus en fonction des facteurs qui déterminent ce comportement.[10]

2.2.1 Historique

Les études des réseaux de neurones artificiels (RNA) datent depuis les années 1940. Grâce aux développements des recherches sur le cerveau et la disponibilité des outils de simulation, les chercheurs étudiaient des ensembles de neurones formels interconnectés. Ces réseaux, déjà développés à l'époque, permettaient d'effec-

tuer quelques opérations logiques simples. Jusqu'aux années 1980, la recherche était freinée par la limitation théorique du perceptron.

Peu après cette époque, Hopfield lança de nouveau en 1982 la recherche dans ce domaine après avoir montré l'analogie entre les RNA et les systèmes physiques. Après les années 1990, quelques travaux scientifiques ont vu le jour dans le domaine de la robotique, parmi ces applications, on trouve la commande des systèmes d'entraînement et des systèmes de positionnement de haute performance.[4]

2.2.2 Perceptron multi couches

Le perceptron multicouche (**MLP**) est un type de réseau neuronal artificiel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (**feedforward**).

Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche dite de **sortie** étant les sorties du système global. Le perceptron a été inventé en 1957 par Frank Rosenblatt au Cornell Aeronautical Laboratory. Dans cette première version le perceptron était alors mono-couche et n'avait qu'une seule sortie à laquelle toutes les entrées étaient connectées.[15]

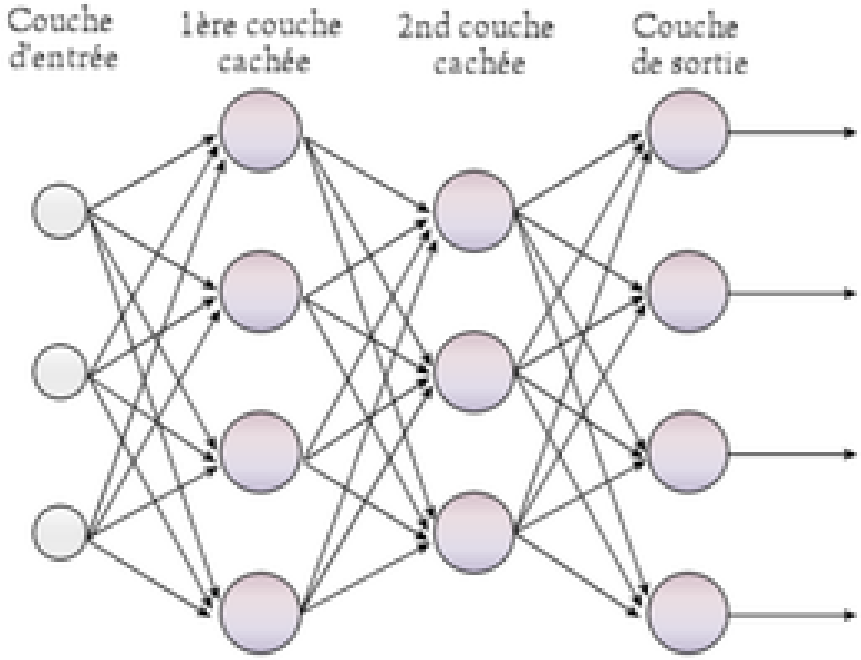


FIGURE 2.1 – Vue simplifiée d'un réseau artificiel de neurones

2.2.2.1 Neurone formel

Le neurone formel est conçu comme un automate doté d'une fonction de transfert qui transforme ses entrées en sortie selon des règles précises. Par exemple, un neurone somme ses entrées, compare la somme résultante à une valeur seuil, et répond en émettant un signal si cette somme est supérieure ou égale à ce seuil (modèle ultra-simplifié du fonctionnement d'un neurone biologique).

2.2.2.2 Fonction d'activation

Dans le domaine des réseaux de neurones artificiels, la fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel. Le terme de "fonction d'activation" vient de l'équivalent biologique : "potentiel d'activation", seuil de stimulation qui, une fois atteint, entraîne une réponse du neurone.

La fonction d'activation est souvent une fonction non linéaire. Un exemple de fonction d'activation est la fonction de Heaviside, qui renvoie tout le temps 1 si le signal en entrée est positif, ou 0 s'il est négatif.[14]

Les fonctions d'activation sont utilisées selon leurs caractéristiques :

- **Non-linéarité** :Quand une fonction est non linéaire, un réseau neuronal à 2 couches peut être considéré comme un approximateur de fonction universel. **Note** : La fonction identité a l'effet inverse, rendant un réseau neuronal multicouches équivalent à un réseau neuronal à une mono-couche.
- **Partout différentiable** : Cette propriété permet de créer des optimisations basées sur les gradients.
- **Étendue** : Quand la plage d'activation est finie, les méthodes d'apprentissage basées sur les gradients sont plus stables (impact sur un nombre de poids limités). Quand la plage est infinie, l'apprentissage est généralement plus efficace (impact sur davantage de poids).
- **Monotone** : Lorsque la fonction est monotone, la surface d'erreur associée avec un modèle monocouche est certifié convexe.
- **Douce** (dérivée monotone) : Les fonctions à dérivée monotone ont été montrées comme ayant une meilleure capacité à généraliser dans certains cas. Ces fonctions permettent d'appliquer des principes comme le **rasoir d'Ockham**.
- **Identité** en 0 (quand) : Ces fonctions permettent de faire un apprentissage rapide en initialisant les poids de manière aléatoire. Si la fonction ne converge pas vers l'identité en 0, alors un soin spécial doit être apporté lors de l'initialisation des poids.

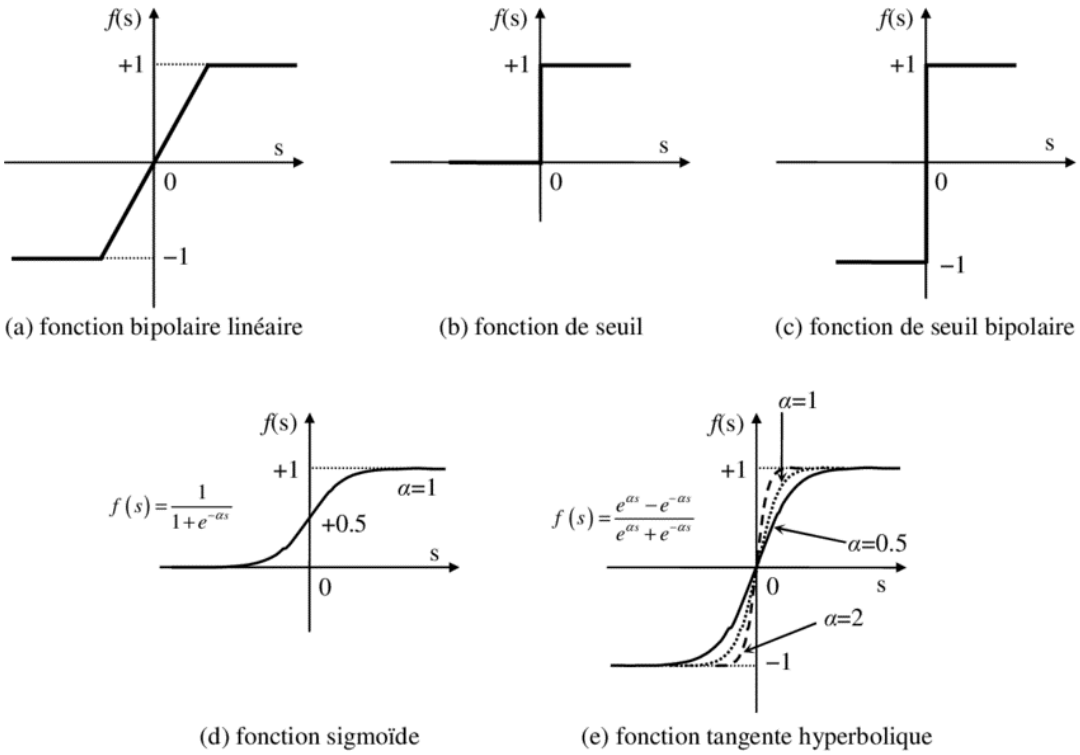


FIGURE 2.2 – Neurone formel et fonction d'activation

2.2.3 Types des réseaux de neurones

— **Réseaux feedforward** : Le réseau neuronal feedforward est un type particulier de réseau neuronal artificiel précoce reconnu pour sa simplicité de conception. Le réseau neuronal à action directe comprend une couche d'entrée, des couches cachées et une couche de sortie. Les informations voyagent généralement dans une seule direction - de la couche d'entrée à la couche de sortie - et ne reviennent en aucun cas en arrière.

— **Réseaux récurrents** : Un réseau de neurones récurrent est une sorte de réseau de neurones composé de boucles permettant de stocker des données dans le réseau. En bref, les réseaux de neurones récurrents utilisent leur raisonnement d'études précédentes pour raconter les événements qui approchent. Les modèles récurrents sont précieux pour leur capacité à séquencer des vecteurs, ce qui ouvre l'API à l'exécution de tâches plus complexes.

2.3 Apprentissage

Il y a plusieurs manières pour faire apprendre un réseau de neurones artificiel. En robotique, on utilise généralement l'apprentissage par renforcement (ou Reinforcement Learning en anglais, abrégé RL). Ce dernier est un domaine de l'apprentissage automatique qui consiste à rechercher une politique optimale à suivre pour un agent en fonction de son environnement.

Un agent est l'entité qui prend le choix d'effectuer une action (α) parmi l'espace d'action A, agissant ainsi dans l'environnement. En fonction de l'action choisie et de son environnement, l'agent se voit remettre une récompense (R_t) et un nouvel état dans lequel il se trouve à l'étape suivante.[5]

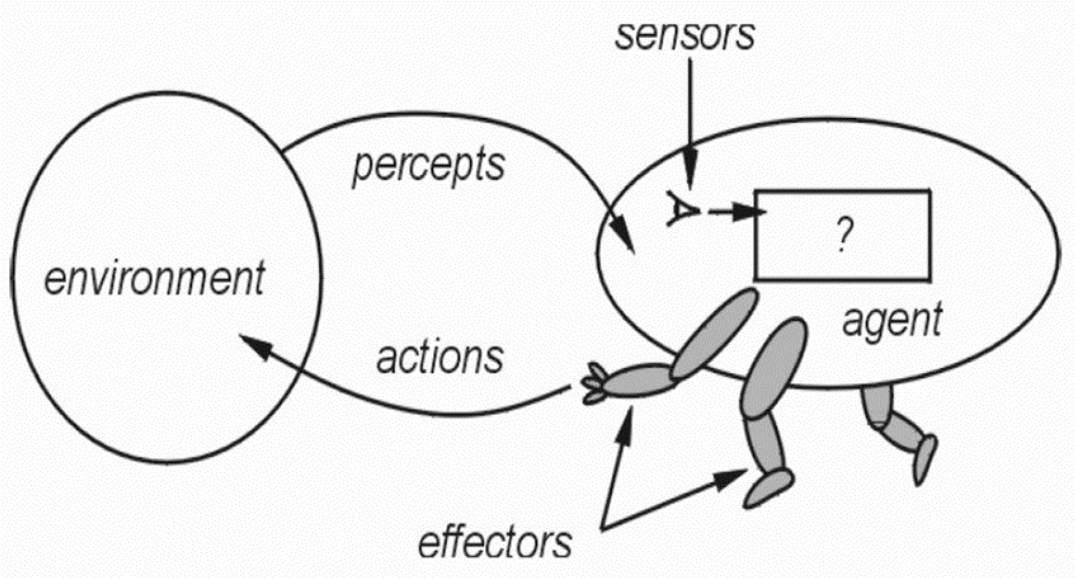


FIGURE 2.3 – Interaction agent-environnement dans un processus de décision markovien

Il existe de nombreux algorithmes de RL, celui que nous avons décidé d'utiliser est les algorithmes génétiques pour ses performances élevées sur les applications courantes de ce domaine.

2.3.1 Algorithme génétique

Les algorithmes génétiques sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de la génétique et de l'évolution naturelle : croisements, mutations, sélection, etc.

Les algorithmes génétiques ont déjà une histoire relativement ancienne, puisque les premiers travaux de John Holland sur les systèmes adaptatifs remontent à 1962 [9]. L'ouvrage de David Goldberg [7] a largement contribué à les vulgariser.

Un algorithme génétique recherche le ou les extrema d'une fonction définie sur un espace de données. Pour l'utiliser, on doit disposer des cinq éléments suivants :

1. Un principe de codage de l'élément de population. Cette étape associe à chacun des points de l'espace d'état une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. Le choix du codage des données conditionne le succès des algorithmes génétiques. Les codages binaires ont été très employés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs, pour l'optimisation de problèmes à variables continues.
2. Un mécanisme de génération de la population initiale. Ce mécanisme doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur tout le domaine de recherche.
3. Une fonction à optimiser. Celle-ci prend ses valeurs dans R + et est appelée fitness ou fonction d'évaluation de l'individu. Celle-ci est utilisée pour selectionner et reproduire les meilleurs individus de la population.
4. Des opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recompose les gènes d'individus existant dans la population, l'opérateur de mutation a pour but de garantir l'exploration de l'espace d'état.
5. Des paramètres de dimensionnement : taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérateurs de croisement et de mutation. Le principe général du fonctionnement d'un algorithme génétique est représenté sur la figure .

On commence par engendrer une population d'individus de façon aléatoire. Pour passer d'une génération k à la génération k + 1, les trois opérations suivantes sont répétées pour tous les éléments de la population k. Des couples de parents P1 et P2 sont sélectionnés en fonction de leurs adaptations. L'opérateur de croisement leur est appliqué avec une probabilité Pc (généralement autour de 0.6) et engendre des couples d'enfants C1 et C2. D'autres éléments P sont sélectionnés en fonction de leur adaptation. L'opérateur de mutation leur est appliqué avec la probabilité Pm (Pm est généralement très inférieur à Pc) et engendre des individus mutés P 0 . Les enfants (C1,C2) et les individus mutés P 0 sont ensuite évalués avant insertion dans la nouvelle population (la figure 2.4 présente le cas où les enfants et les individus mutés remplacent les parents).

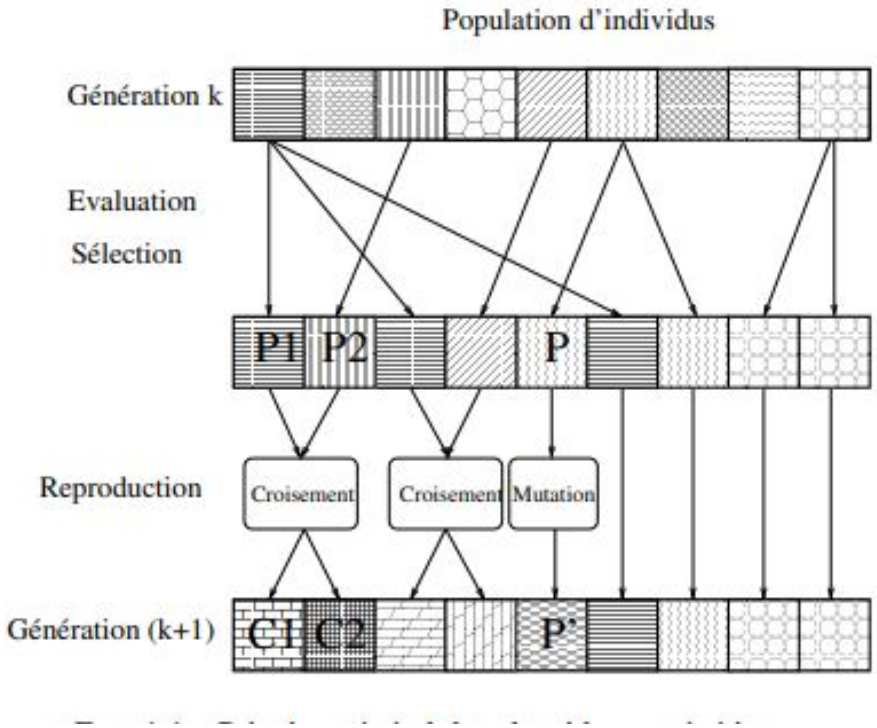


FIGURE 2.4 – Principe générales des algorithmes génétiques

Différents critères d'arrêt de l'algorithme peuvent être choisis :

- Le nombre de générations que l'on souhaite exécuter peut être fixé a priori. C'est ce que l'on est tenté de faire lorsque l'on doit trouver une solution dans un temps limité.
- L'algorithme peut être arrêté lorsque la population n'évolue plus ou plus suffisamment rapidement.

2.4 Neuro Evolution

La neuro évolution est une technique d'apprentissage automatique qui applique des algorithmes évolutifs pour construire des réseaux de neurones artificiels, s'inspirant de l'évolution des systèmes nerveux biologiques dans la nature. Comparée à d'autres méthodes d'apprentissage par réseau de neurones, la neuro évolution est très générale ; elle permet un apprentissage sans cibles explicites, avec seulement peu de rétroaction, et avec des modèles neuronaux et des structures de réseau arbitraires. La neuro évolution est une approche efficace pour résoudre les problèmes d'apprentissage par renforcement et est le plus souvent appliquée dans la robotique évolutive et la vie artificielle.[12]

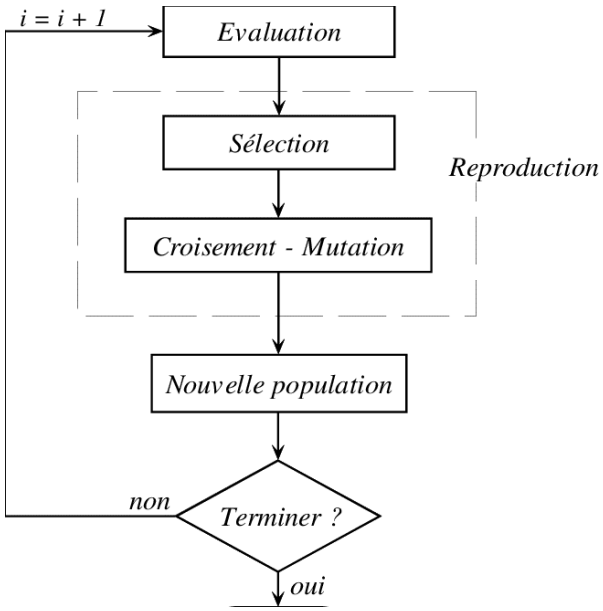


FIGURE 2.5 – Les étapes d'un algorithme génétique

2.5 État de l'art

Dans ce qui suit nous allons citer quelques travaux cités dans la littérature concernant le contrôle de robots quadrupèdes. La direction du corps associée à une trajectoire assignée (position, vitesse, orientation) et l'équilibre (accélération angulaire du corps devant rester à l'intérieur d'une plage déterminée par la dynamique du robot).[13]

— (Kyrre Glette) ce chercheur a simulé et coordonné la demarche du robot avec capteurs, parmi les capteurs qu'il a utilisés TIGOTAE Capteure.

— (Neil vaughan) ce chercheur se base sur une comparaison entre L'algorithme benchmark et l'algorithme ANN et s'appuyait advantage Sur la technique de EVC.

2.6 Conclusion

Nous avons consacré ce chapitre à la neuroévolution. Nous avons donc présenté le concept général de la neuroévolution. Nous avons défini les réseaux de neurones artificiels et expliqué le perceptron multicouche et le neurone formel. Nous avons cité les différents types des réseaux de neurones. Par ailleurs, nous avons présenté les algorithmes génétiques comme méthode d'apprentissage. Dans le prochain chapitre nous allons présenter notre modèle de contrôle de robot et son intégration dans le simulateur de robot Gazebo.

Chapitre 3

Conception et implémentation du Système

3.1 Introduction

Ce chapitre est consacré à la présentation de la conception de notre système ainsi que l'implémentation. Nous allons parler sur l'environnement de simulation de notre robot quadrupède et la machine utilisée ainsi que les capteurs utilisés dans travail. Nous allons parler aussi du contrôleur utilisé sous forme d'un réseaux de neurones artificiels.

3.2 Environnement de simulation

Nous avons utilisé le simulateur Gazebo. Né avec le nom de Gazebo Project, est un simulateur 3D, cinématique, dynamique et multi-robot permettant de simuler des robots articulés dans des environnements complexes, intérieurs ou extérieurs, réalistes et en trois dimensions. Il s'agit d'un programme open source distribué sous licence Apache 2.0, utilisé dans les domaines de la recherche en robotique et en intelligence artificielle. Gazebo a été une composante du projet Player de 2004 à 2011, et en 2011 est devenu un projet indépendant.

3.2.1 Etapes pour installer gazebo

Il y a deux étapes rapides pour installer Gazebo et se sont des commandes lignes que nous tapons dans le terminal

première :

```
curl -sSL http ://get.gazebosim.org — sh
```

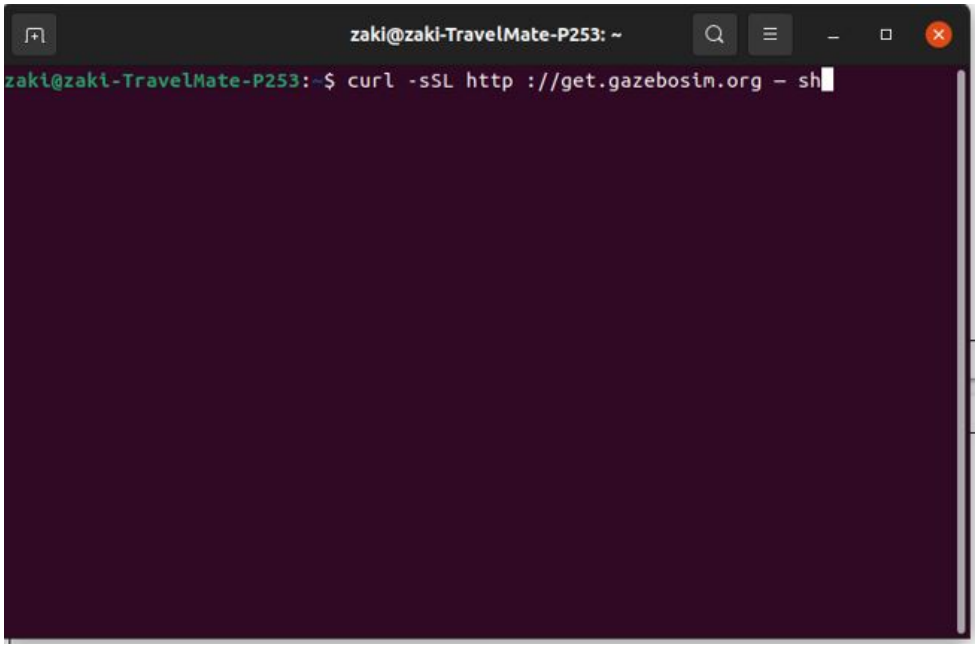


FIGURE 3.1 – Etape pour install gazebo

deuxième :

```
gazebo
```

FIGURE 3.2 – lancement de gazebo

3.2.2 Gazebo

Gazebo [Koenig and Howard, 2004] est un simulateur multi-robots open source qui a été crée par Andrew Howard et son étudiant Nate Koenig au laboratoire USC Robotics Research Laboratory en 2004. Il est capable de simuler dans des environnements tridimensionnels réalistes une population de robots, capteurs et objets. Il

est l'un des simulateurs 3D les plus populaires et a une communauté d'utilisateurs trés active .
Ce simulateur peut intégrer plusieurs moteurs physiques ; 'a savoir **ODE** (par défaut), Bullet, Simbody et DART. On peut alors utiliser le moteur physique que nous souhaitons en changeant uniquement un seul paramétre. Le système de visualisation par défaut est OpenGL , mais on peut utiliser le moteur de rendu graphique 3D OGRE . Ce dernier fournit un rendu 3D réaliste, un éclairage de haute qualité,ainsi que les ombres et les textures.
Gazebo est séparé en deux parties, un serveur et un client. Le serveur gére le moteur physique et la génération des données des capteurs, tandis que le client est responsable de la visualisation et de l'interface graphique.[2] L'architecture de Gazebo est illustrée dans la figure ci-dessous.

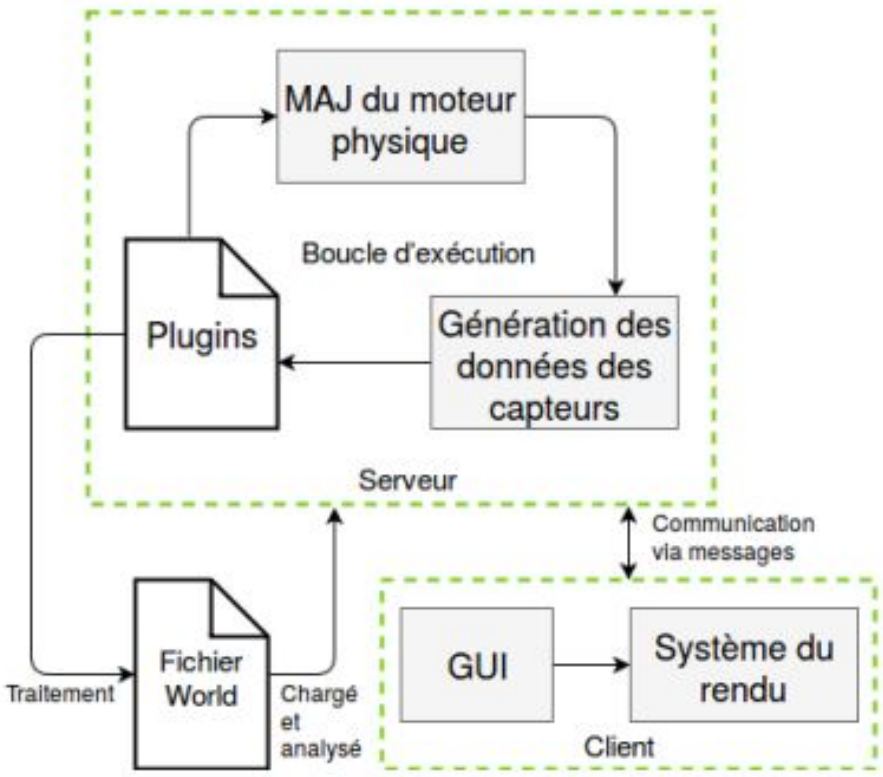


FIGURE 3.3 – Architecture de simulateur gazebo

Les éléments impliqués dans l'exécution d'une simulation Gazebo sont :

— **Le fichier (world) :** Il s'agit d'un fichier Extensible Markup Language (XML) de format Simulation Description Format (SDF) qui est analysé par le serveur. Il permet de spécifier les propriétés globales de la scène simulée, tels que le moteur physique utilisé, la force de gravité et l'éclairage. Dans ce fichier on peut aussi spécifier les modèles et les plugins qui vont etre chargés dans l'environnement.

— **Les fichiers (model) :** Ces fichiers de format SDF décrivent en XML les modèles (robots) qui doivent etre simulés dans l'environnement. Un modèle est une collection de liens, de jointures, de capteurs et de plugins. Les corps sont connectés les uns aux autres par des jointures. Les jointures sont controlées par des plugins. Il est possible d'importer différents formats de modèles pour construire des environnements encore plus réalistes.

— **Les plugins :** Un plugin est un morceau de code (C++) qui est compilé en tant que bibliothéque partagée pour modifier la simulation en temps réel. Ce code peut etre attaché a des modèles, des capteurs, a l'environnement ou au simulateur lui-meme. Par exemple, un plugin modèle pourra permettre de controler les jointures d'un robot. Les plugins fournissent aux utilisateurs un accés direct aux propriétés physiques des (models) et aux bbibliothèque e Gazebo via des classes standard C++.

3.2.2.1 Ces Caractéristiques

Gazebo est un logiciel libre financé en partie par Willow Garage qui peut etre reconfiguré, développé et modifié. Il est compatible avec ROS et Player. Gazebo peut etre exécuté à partir de ROS et il est possible d'utiliser les API de ce dernier pour controler les robots dans les simulations, c'est-à-dire envoyer et recevoir des données de ceux-ci.

Ce logiciel permet de faire des simulations réalistes de la physique des corps rigides. Les robots peuvent interagir avec le monde (ils peuvent ramasser et pousser des objets, rouler et glisser sur le sol) et inversement (ils sont affectés par la gravité et peuvent se heurter à des obstacles dans le monde). Pour le faire, Gazebo utilise de multiples moteurs physiques tels que Open Dynamics Engine (ODE), Bullet, Symbody ou DART.

Il existe la possibilité de développer et de simuler ses propres modèles de robot (URDF) et de les charger au moment de l'exécution. De plus, il est possible de créer des scénarios de simulation (mondes), en modifiant les caractéristiques des contacts avec le sol, des obstacles et meme des valeurs de gravité dans les trois dimensions. Il est également possible de faire varier les caracté0ristiques de contact de chaque lien individuellement.[2]

3.2.2.2 Ces composants

Gazebo contient divers plug-ins pour ajouter des capteurs au modèle du robot et les simuler, tels que des capteurs d'odométrie (GPS et IMU), de force, de contact,

de laser et des caméras stéréos. Gazebo possède un immense catalogue de robots dont la majorité des robots commerciaux.

Gazebo a les éléments suivants :

— Simulation dynamique, avec accés à plusieurs moteurs physiques hautes performances, notamment ODE, Bullet, Simbody et DART.
— Graphiques 3D avancés, en utilisant OGRE, vous pouvez générer des environnements et rendre des formes réalistes, des textures, des lumiéres, des ombres, etc.
— Des capteurs et bruit dont les données sont trés intéressantes. — Extensions, afin que les développeurs puissent personnaliser les robots, les capteurs et les environnements de controle grace à son API.
— De nombreux modéles de robots, y compris PR2, Pioneer2 DX, iRobot Create, TurtleBot, ou créez le votre en utilisant SDF.
— En outre, il dispose également de fonctions pour le réseau, le cloud et des outils pour la ligne de commande, ce qui facilite l'introspection dans la simulation et le controle.[1]

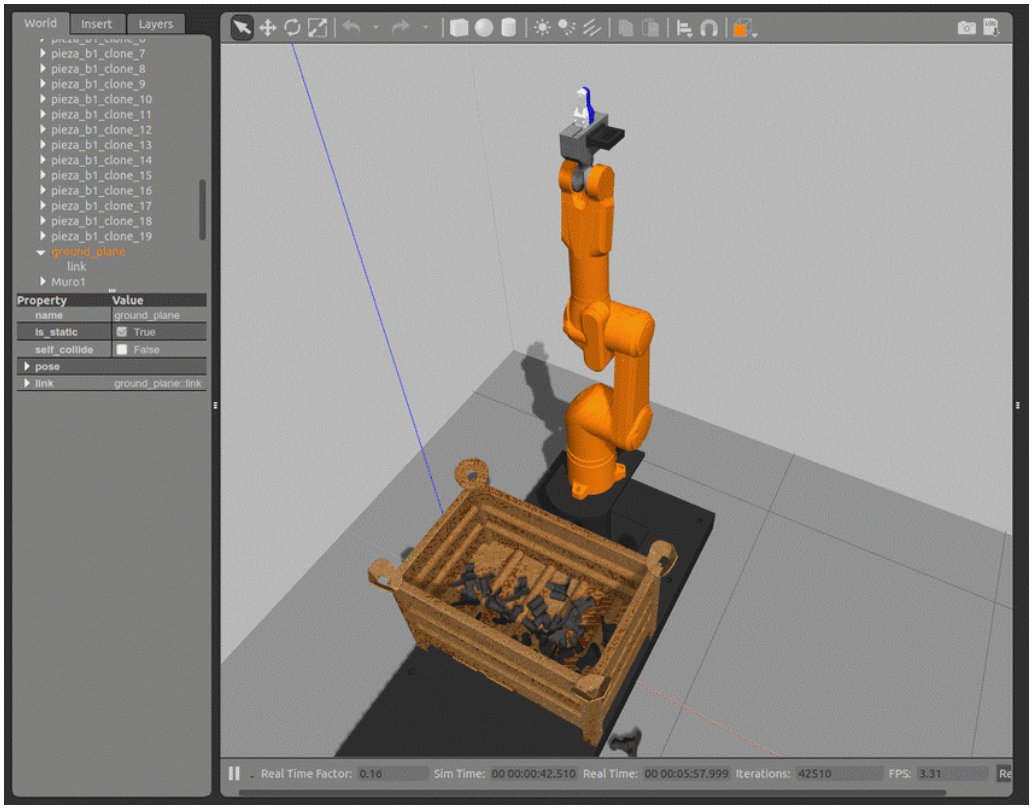


FIGURE 3.4 – simulateur gazebo

3.2.3 Language utilisé

Comme un language de programmation on a utilisé le language C++ pour une structure claire aux programmes et permet de réutiliser le code

3.2.4 Les caracteristiques de la machine

Les caracteristiques de la machine pour la simulation et l'apprentissage de robot

- une version de system d'exploitation ubuntu 21.04
- un CPU i7 7eme .
- un GPU Nvidia GTX 1660 Ti
- une RAM de 12 GB

3.3 Le robot quadrupède

Les robots quadrupèdes sont des robots qui possèdent quatre pattes. Dans cette section on va présenter la structure de notre robot simulé sur le simulateur multi-robot Gazebo.

3.3.1 la structure du robot

Dans notre travail le modèle de robot est sous forme XML, il se compose d'une partie principale avec quatre pattes où chaque patte se compose de trois segments de tailles éguales liés par des jointure de rotation. Dans le fichier XML on trouve les informations physiques de notre modèle :

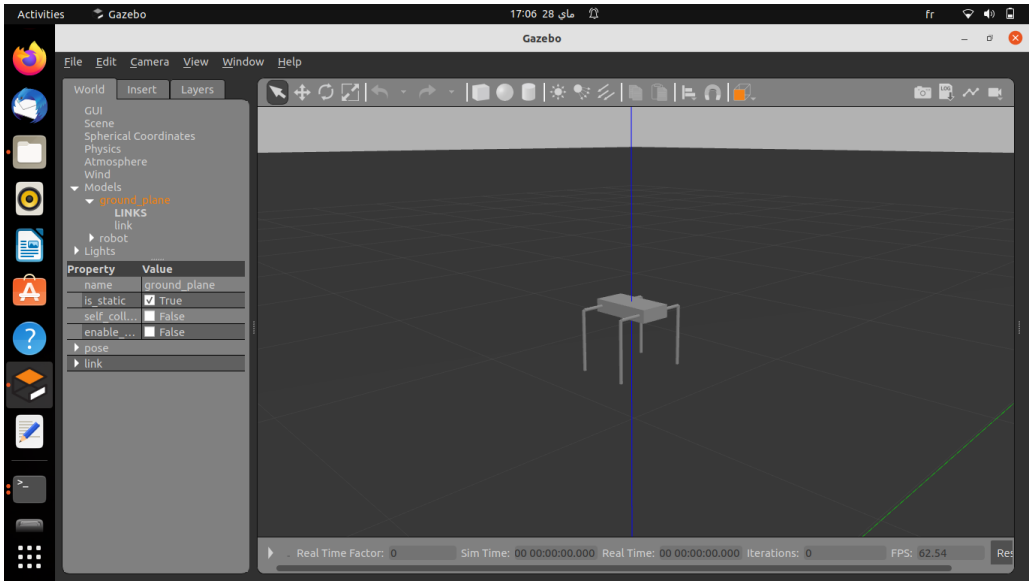


FIGURE 3.5 – quadruped Model

3.3.1.1 structure de torse

Le torse du robot quadrupède est la composante centrale de toute la structure du robot. Quatre jambes sont connectées ensemble à travers le torse.

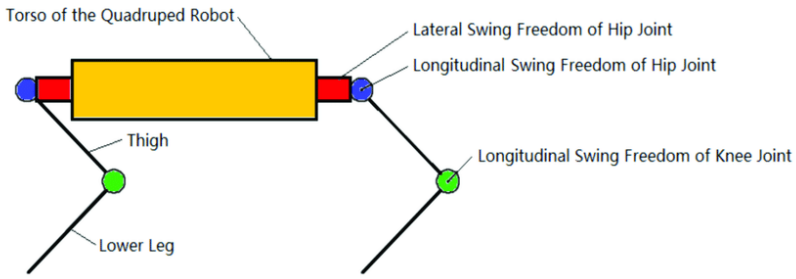


FIGURE 3.6 – torse de quadruped robot

3.3.1.2 structure des jambes

Trois classes de structure des jambes sont courantes dans le monde naturel, ils sont : Plantigrade, Digitigrade et Unguligrade.

3.3.1.3 Segment et Jointure

Comme nous le voyons dans la figure qui illustre la structure du robot, nous avons un robot quadrupède, qui possède 12 segments et 8 jointures liés au torse.

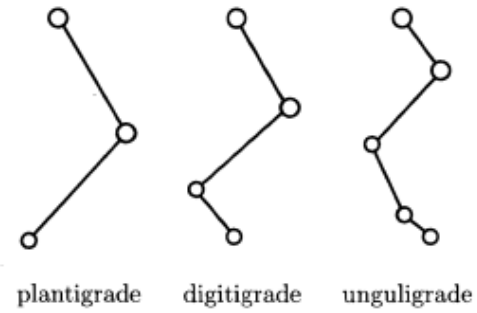


FIGURE 3.7 – Plantigrade, Digitigrade et Unguligrade Les pattes postérieures montraient en omettant les membres du système squelettique qui sont en contact avec le sol en se tenant debout

FIGURE 3.8 – quadruped robot segment et jointure

3.3.1.4 Formats de description des simulations

Le format de description de simulation (SDF) a été développé à l'origine dans le cadre du simulateur Gazebo avec des programmes robotiques scientifiques à l'esprit. Il s'agit d'un format XML qui décrit des objets et des environnements pour la simulation, la visualisation et le contrôle robotiques. Le SDF permet des descriptions précises d'un robot, composé d'attributs cinématiques et dynamiques, de capteurs, de propriétés de surface, de textures et de frottements articulaires. De plus, le SDF donne les informations nécessaires des outils pour décrire avec précision les environnements riches et complexes souvent nécessaires à une simulation réaliste .

Les modèles SDF de notre robot incluent la description complète de tous les liens

et jointures. Un lien est associé à un corps du modèle et contient plusieurs éléments comme indiqué ci-dessous :

- Collision : L'élément de collision établit une encapsulation géométrique utilisée pour la détection de collision. L'encapsulation peut prendre plusieurs formes, mais pour le travail effectué dans ce travail, l'encapsulation est soit : une boîte rectangulaire (base), un cylindre (jambes et genoux), ou une sphère (pieds).
- Capteur : Un élément capteur est généralement ajouté à un lien pour permettre la collecte de données. Dans notre cas, nous utilisons des capteurs de position et de vitesse.

Les jointures complètent les liens en spécifiant la connexion entre deux liens dans un schéma parent et enfant. Il précise également les propriétés essentielles telles que l'axe de rotation, les limites articulaires et la dynamique articulaire.[3]

3.4 Les capteurs utilisés

Gazebo fournit des modèles de capteurs les plus utilisés. Pour exemple, le Global Positioning System (GPS), l'Inertial Measurement Unit (IMU), les lasers et les caméras. Ces capteurs sont parfaits et observent le monde extérieur sans exposer de bruits. Cependant, pour avoir des simulations proches du réel, il est possible d'ajouter explicitement des bruits aux données . Avoir de tels capteurs intégrés dans le simulateur, nous permet de les utiliser sans pour autant en développer. Chaque module pourra ainsi être spécialisé et comporter un capteur donné différent. Par exemple un module doté d'une caméra jouera le rôle d'un œil et le module doté d'un laser jouera le rôle d'une oreille.[3] Les capteurs se sont des outils qui permettent aux robots d'interagir avec le monde extérieur. En fonction des données perçues, ils vont prendre des décisions et agir selon leurs environnements et leurs états internes. Théoriquement, plus le robot possède de capteurs, plus il est conscient de ce qui se passe autour de lui. Il prendra donc les bonnes décisions.[3]

3.5 Paramètres physiques

Gazebo fournit pour tous objets simulés de nombreux attributs physiques. Cependant, il est nécessaire d'initialiser ces attributs par des valeurs se rapprochant le plus de la réalité. Les attributs que nous trouvons les plus importants à ajuster correctement sont les suivants :

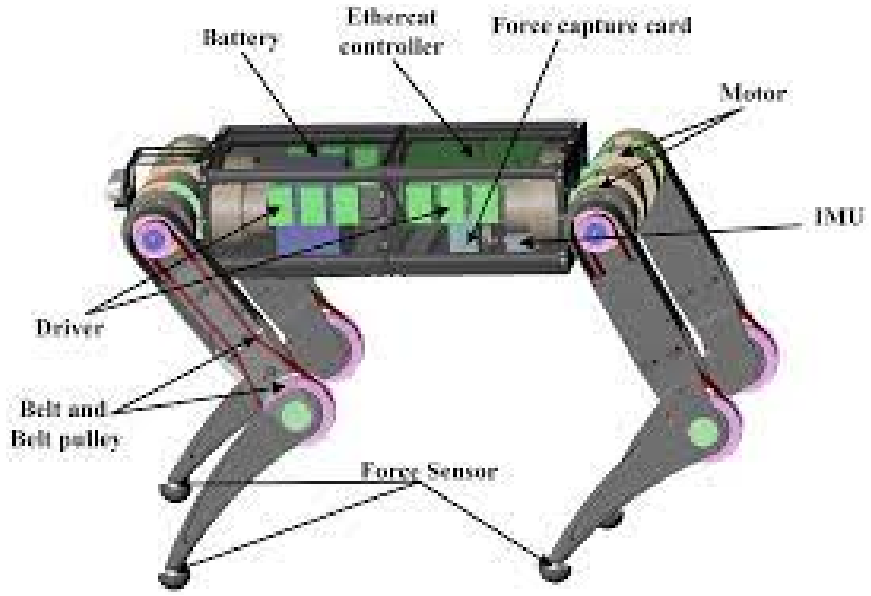


FIGURE 3.9 – capteurs intégrer

— **Les dimensions du segment :** elles vont de 5 cm à 10 cm.
— **La masse du segment :** : la masse est calculée à partir du volume du segment et de sa masse volumique. Nous avons choisi la masse volumique d'eau. En fait, la masse des segments doit être aussi légère que possible. Ainsi, le segment pourra supporter le poids des autres segments connectés à lui.
— **Les frottements :**si elle est appliquée sur une surface d'un segment, cette force opposée peut éviter le comportement de glissement du robot et, si appliquée au niveau des jointures, elle évite les mouvements de vibrations irréalistes. Nous avons fixé le frottement de la surface du segment à 0.5 et nous avons fixé les frottements au niveau des jointures à 0.2.
— **L'amortissement des jointures :** en fonction de la vitesse de la jointure, l'amortissement permet de dissiper l'énergie. Cela peut éviter les mouvements de rebondissement. Avec les expérimentations, nous l'avons mis à 0.02.
— **La vitesse des jointures :** : la vitesse maximale autorisée pour une articulation est de 5 rad/s. Cette valeur est déterminée par expérimentations.
— **Le couple des jointures :** : le couple ou le moment d'une force (torque en anglais) représente la force requise permettant à une jointure d'effectuer une rotation et soulever l'ensemble des segments qui lui sont liés sans pour autant atteindre la rupture de la jointure. Après plusieurs tests, nous avons fixé une limite de 1.75 Nm.[3]

3.6 Le contrôleur

Comme dit précédemment, le contrôleur utilisé dans ce travail était un ANN. Les ANN sont des modèles mathématiques du cerveau organique, et ils sont utilisés pour imiter la machine nerveuse des êtres organiques.

L'ANN contrôle le robot à l'aide d'un réseau neuronal prédictif avec une fonction d'activation de tangente hyperbolique. Le réseau neuronal a neuf entrées ; ils représentent les données qu'on récupère des capteurs, à savoir les positions de toutes les joinutres, et la 9ème entrée est un neurone qui représente la position générale du robot par rapport au repère de on environnement. Les informations sont diffusées dans le réseau de neurones via la fonction feedforward, les valeurs des neurones de sortie sont les forces que nous appliquons par la suite aux jointures. Le vecteur de sortie possède donc 8 sorties.

L'équation ci-dessous donne la sortie du réseau de neurones :

$$O_j = tanh \sum_{i=1}^{t} W_{ij} J_j$$

où O_j est la valeur de sortie de jointure j, W_{ij} représente le poids qui relie le neurone d'entrée i avec le neurone de sortie j, et J_j est la position articulaire actuelle de jointure j.[3]

3.7 Implementation

Nous expliquerons dans cette section les méthodes et les approches utilisées pour l'apprentissage de notre robot. Nous avons utilisé dans notre projet les 7 fichier suivant :

— **fichier (controller.cc)**

Nous implémentons dans ce fichier les méthodes utilisées pour contrôler le robot et le faire bouger et fonctionner.

Nous commençons par charger le robot en format SDF dans l'environnement de simulation Gazebo par la fonction Load. Nous chargeons ensuite les jointures du robot mis en un seul vecteur.

```
void Controller::Load(physics::ModelPtr _parent, sdf::ElementPtr _sdf )
{
        this->model = _parent;
        vector<robotController> cont;
        int j=0;

        joints = this->model->GetJoints();



        //vector<annController> anns;

        //for(int i=0; i<anns.size(); i++)
        //      anns[i].ann()=nbrlayers;


        double xi, yi; xf, yf;

        vector<double> robotInfo;

        this->updateConnection = event::Events::ConnectWorldUpdateBegin(
                boost::bind(&Controller::OnUpdate, this, _1));


}
```

FIGURE 3.10 – fonction pour charger variables

Une fois les variables et les paramètres concernant le robot (joints, positions, vecteurs) sont chargé dans l'environnement, nous passons à la fonction OnUpdate.
Cette fonction est la fonction principale qui permet de mettre à jours les paramètres du robot. Elle va donc boucler tout au long de la simulation pour gérer le mouvement du robot. On commence comme montré dans la figure ci-dessous de récupérer les données issues des capteurs. Premièrement on récupère la position du robot dans l'axe X par rapport au repère de l'environnement.
Nous récupérons ensuite les positions de toutes les jointures du robot qui ont été chargé précédemment dans un vecteur 'Joints'. Les positions ici représente les angles en radian.

```
void Controller::OnUpdate(const common::UpdateInfo & )
{



  for( int i=0; i<joints.size(); i++)
  {
     double p = this->joints[i]->Position(0);

     robotInfo.push_back(p);

  }
```

FIGURE 3.11 – fonction pour gère le mouvement du robot

```
double p = this->joints[i]->Position(0);
```

FIGURE 3.12 – les informations de robot

Ensuite on a crée un vecteur (robotInfo) [**robotInfo.push_back(p)**]. Ce vecteur permet d'enregistrer les informations liées avec le robot comme les positions de jointures et la position de robot.

Après cela, ce vecteur qui représente les entrées du contrôleur, va être passé au réseau de neurones avec la fonction **feedforward**. Cette dernière nous retourne un vecteur de sorties. nous le récupérons dans un vecteur nommé 'output'. L'instruction qui permet de réaliser cela est présenté ci-dessous.

vector $<$ *double* $>$ output = cont[j].ann.feedForward(robotInfo)

le vecteur **output** obtenu à partir de la fonction **feedforward** représente les forces qui vont être appliquées aux jointures dans le prochain pas de temps. Ainsi on obtiendra les nouvelles position des jointures. Nous utilisons la fonction **setforce** afin d'appliquer la force au niveau de chaque jointure. Cela est fait comme suit

```
for(unsigned int i=0; i<joints.size();i++)
{
this->joints[i]->SetForce(0, output[i]  );
}
```

FIGURE 3.13 – fonction applique dans les sorties de feedforward

Cette procédure est répété à chaque pas du temps durant toute la simulation. vers la fin du temps d'évaluation qui est fixé à 1000 itération, nous calculons le fitness du contrôleur. Il représente la distance de marche du robot durant cette période. Mais avant le calcul des distances, la première chose à faire est de calculer les coordonnées de **X** et **Y** au début et à la fin du temps d'évalaution avec ces étapes :

premier etapes est calcule XI et YI initiales

```
for(int i=0;i<1000*(j+1)){
                  j++ ;
                  }

      if(i==0)
      {
      xi= this->model->WorldPose().Pos().X();
      yi=this->model->WorldPose().Pos().Y();
      }
```

FIGURE 3.14 – calcule de coordonnées

deuxième etape calcule XF et YF finale

```
for(unsigned int i=0; i<joints.size();i++)
{
this->joints[i]->SetForce(0, output[i]  );
}

  if(i==1000*(i+1)){
  xf= this->model->WorldPose().Pos().X();
      yf=this->model->WorldPose().Pos().Y();
```

FIGURE 3.15 – calcule de coordonnées

Après avoir fait tout cela, nous calculons la distance euclidienne avec cette instruction

double distance = calculerdist(xi, yi, xf, yf)

qui est une fonction qui nous renvoie la distance de marche du robot,sa implementation est comme ça :

```
void controller::calculdist(double Xi,double Yi,double Xf,double Yf)
{
   double Distance = hypot((Xf-Xi),(Yf-Yi));

}
```

FIGURE 3.16 – fonction pour calcule distance de marche

que la fonction **hypot** calcule le **SQRT** de $(XF - Xi)^2$ et $(YF - Yi)^2$

— **fichier (controller.hh)**

Nous incluons dans ce fichier des variables

```
vector<int > nbrlayers={9, 7, 5, 8};

struct annController
{
        ANN ann();
        double fitness;
};
```

FIGURE 3.17 – structure et variables utilise

— **fichier (extra.hh)**

ce fichier est créé pour gérer les paramètres du robot et l'algorithme génétique comme :

```
double round(double value)
{
  return std::round( value * 1000.0 ) / 1000.0;
}
```

FIGURE 3.18 – function pour gérer la manupilation de robot

```
string time()
{
  time_t t = time(0);   // get time now
struct tm * now = localtime( & t );

std::stringstream stime;
stime  << now->tm_mday << '-'
     << (now->tm_mon + 1) << '-'
 << (now->tm_year + 1900) << "--"
     << now->tm_hour << '-'
     << now->tm_min << '-'
     << now->tm_sec ;
return stime.str();
}
```

FIGURE 3.19 – function pour gérer le temps de marche

— **fichier (word.word)**

Dans ce fichier, nous implémentons le design de notre robot avec xml dans un environnement de simulation se composant d'un terrain plat et d'une source de lumière.

```
<joint type="revolute" name="torso_hip">
        <pose>0 0 0.0175 0 0 0</pose>
        <child>hip</child>
        <parent>torso</parent>
        <axis>
                <xyz>0 0 1</xyz>
                <limit>
                        <lower>0</lower>
                        <upper>0s</upper>
                </limit>
        </axis>
</joint>
```

FIGURE 3.20 – implementation de torse do robot

```
<joint type="revolute" name="thigh_shank">
        <pose>0 0 0.025 0 0 0</pose>
        <child>shank</child>
        <parent>thigh</parent>
        <axis>
                <xyz>0 1 0</xyz>
                <limit>
                        <lower>-1.57075</lower>
                        <upper>1.57075</upper>
                        <effort>1000</effort>
                        <velocity>0.5</velocity>
                </limit>
        </axis>
</joint>
```

FIGURE 3.21 – implementation de cuisse do robot

— **fichier (ANN.hh)**
— **fichier (ANN.cc)**

Cette classe contient un constructeur et un destructeur :

ANN : :ANN(vector ≺ int ≻ ANNinfo) : nbrLayer (ANNinfo.size()),
nbrInNeuron (ANNinfo[0]), nbrOutNeuron (ANNinfo.back())
fitness = 0 ;
for (int i=1 ; i ≺ ANNinfo.size() ; i++)
network.push_back(strLayer(ANNinfo[i] , ANNinfo [i-1])) ;

```
ANN::~ANN()
/*
I have to find a way to delete complete memory
*/
{
        network.clear();
}
```

FIGURE 3.22 – destructeur de class ANN

```
vector<double> ANN::getWeights()
{
        vector<double> weights;

        for (int i=0; i<network.size();i++)//for each layer
                for (int j=0; j<network[i].layer.size(); j++)//for each neuron
                        for (int k=0; k<network[i].layer[j].weight.size(); k++)//for each weight

                                weights.push_back(network[i].layer[j].weight[k]);// bias is included

        return weights;
}
```

FIGURE 3.23 – fonction getweights de classe ANN

```
vector <double>  ANN::feedForward(vector <double> input)
/*
spread the input signal in to the network and produce an output signal
here we do not store neurons's outputs
*/
{
        vector <double> inputLayer = input;
        double v;
        int indcInput;
        vector <double> output;

        if(input.size() != nbrInNeuron)
        {
                std::cerr << "WARNING (ANN::feedForward):  neuron nbr ininput layer\n";
                return input ;
        }
        for(int i=0; i<network.size(); i++)// for each layer (without input layer)
        {
                if (i != 0)
                        input = output;
                output.clear();
                for (int j=0; j<network[i].layer.size(); j++) // for each neuron of layer i
                {
                        v=0;
                        indcInput=0;
                        for(int k=0; k<network[i].layer[j].weight.size()-1; k++) // for each weight of
neuron j
                                v = v + (network[i].layer[j].weight[k] * input[indcInput++]);
                        v= v + network[i].layer[j].weight.back(); // add the bias
                        output.push_back(activationFct(v, ActFunType));
                }
```


FIGURE 3.24 – fonction de feedforward implémentassions de classe ANN

3.8 Apprentissage

3.8.1 Les Algorithmes génétiques

3.8.1.1 La fonction d'évaluation

Le système d'évolution va récompenser les contrôleurs qui semblent répondre le mieux à la fonction objectif. Si la sélection des robots est basée sur cette dernière, il est nécessaire de bien la définir pour obtenir les résultats que nous souhaitons.

La fonction d'évaluation est donc la distance euclidienne parcourue par le robot afin de déterminer sa capacité à effectuer un déplacement. Les deux positions initiale et finale sont les positions isobarycentres du robot. Les robots sont évalués dans l'environnement pendant 1000 pas de temps de simulation[3].

3.8.1.2 Paramètres de l'étude

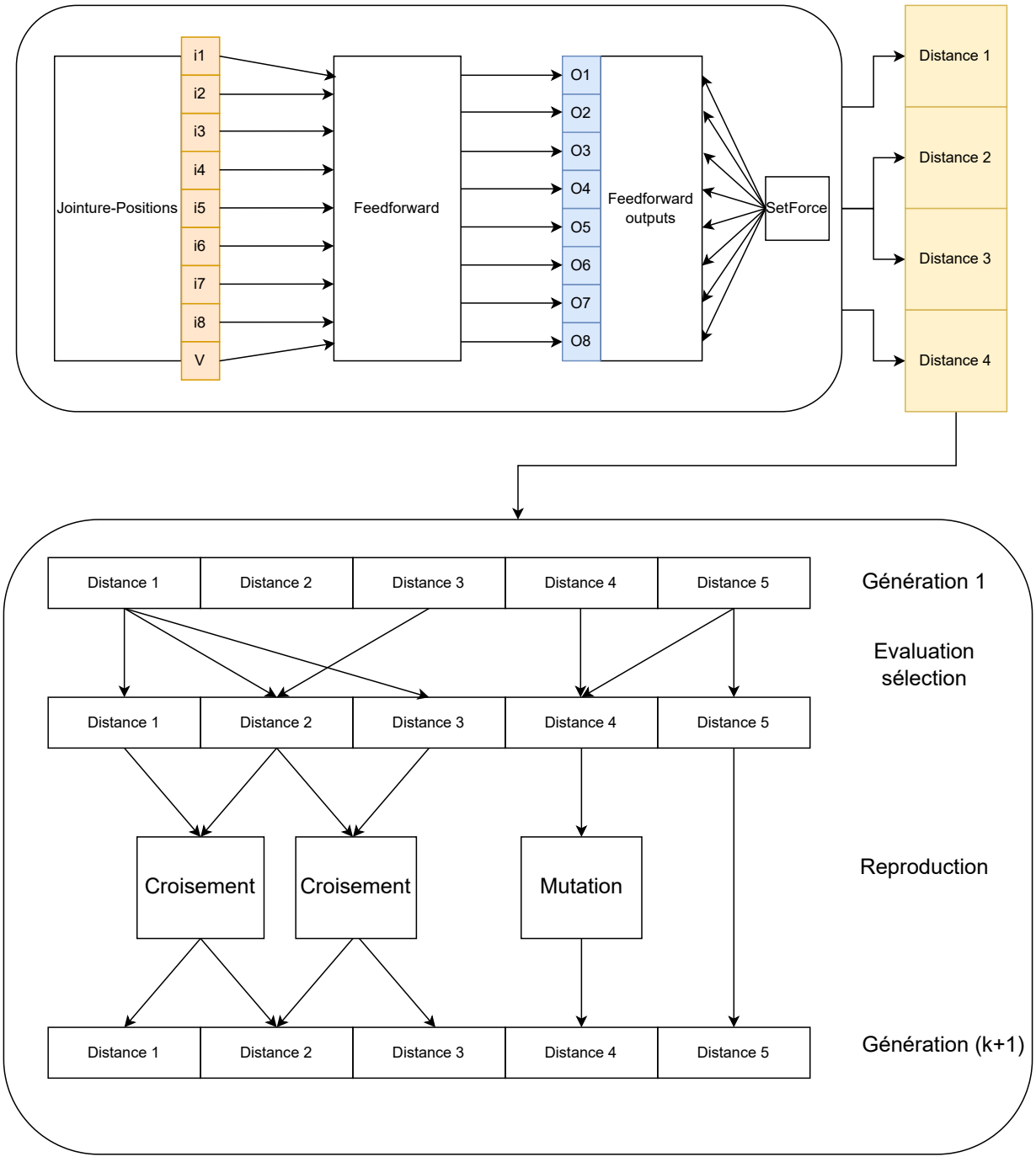
Le tableau ci-dessous représentent les paramètres à utiliser pour l'évolution :

Paramètre	Valeur
Taille de la population	50
Nombre de génération	100
Taux de croisement	35%
Taux de mutation	75%
Élitisme	10
Méthode de sélection	Sélection par tournoi avec 7 concurrents

FIGURE 3.25 – Les paramètres définis pour le système d'évolution[3]

3.8.2 Approche utilisée

Le diagramme présenté dans la figure ci-dessous résume notre approche, qui se base sur l'évolution des neurones à travers lequel nous voulons trouver le meilleure contrôleur à travers les générations pour la marche du robot. On a mis la distance dans le vecteur et on cherche les deux meilleures distances avec l'application de sélection. Après cela on cherche et on supprime les plus mauvais avec une distances qui a 0, et ont applique sur les contrôleurs permettant des distances moyennes le croisement et la mutation .


Jointure-Positions
i1
i2
i3
i4
i5
i6
i7
i8
V
Feedforward
O1
O2
O3
O4
O5
O6
O7
O8
Feedforward outputs
SetForce
Distance 1
Distance 2
Distance 3
Distance 4
Distance 1
Distance 2
Distance 3
Distance 4
Distance 5
Génération 1
Evaluation sélection
Distance 1
Distance 2
Distance 3
Distance 4
Distance 5
Croisement
Croisement
Mutation
Reproduction
Distance 1
Distance 2
Distance 3
Distance 4
Distance 5
Génération (k+1)

3.9 Conclusion

Nous avons montré dans ce chapitre la conception et l'implémentation de notre travail de mémoire. Nous avons conçu un robot quadrupède avec le format SDF et nous l'avons simulé dans le simulateur Gazebo. Nous avons utilisé un réseau de neurones artificiel pour contrôler ces jointures afin qu'il puisse effectuer des déplacements. Les algorithmes génétiques on été ensuite appliqués pour réaliser l'apprentissage.

Deuxième partie

Conclusion générale

Dans ce mémoire, nous avons présenté notre travail concernant la conception d'une approche de contrôle en se basant sur la neuro-évolution. Le contrôleur était un réseau de neurones artificiel avec 9 neurones dans la couche d'entrées et 8 neurones dans la couche de sortie. Le réseaux de neurone possède deux couches cachées avec 10 neurones et 5 neurones respectivement. L'objectif était de contrôler un robot quadrupède pour se déplacer dans son environnement. Pour cela il était nécessaire de faire une étape d'apprentissage pour faire apprendre aux robots de réaliser des mouvements lui permettant le déplacement. Le type d'apprentissage utilisé est l'apprentissage par renforcement avec les algorithmes génétiques. Le robot était simulé dans un environnement tridimensionnel avec le simulateur physique Gazebo.

Nous proposons comme travaux futures d'améliorer l'algorithme de contrôle ainsi que l'apprentissage pour obtenir de meilleurs résultats. Nous proposons aussi d'améliorer la conception du robot simulé en modifiant les dimensions des segments et les positions des jointures.

Bibliographie

[1] Le controle des robots. URL https ://www.futurasciences.com/tech/dossiers/robotique-icub-robots-service-1143/page/4/ .

[2] Gazebo : simulateur de robot dans un monde ouvert. Accessed :2022-03-07. URL https ://www.linuxadictos.com/fr/gazebo-simulador-robots.html.

[3] D. C.-B. S. S. S. D. N. [Akrour et al., 2017] Akrour and Luga. Joint evolution of morphologies and controllers for realistic modular robots. (2017).

[4] H. CHAOUI. CONCEPTION ET COMPARAISON DE LOIS DE COMMANDE ADAPTATIVE A BASE DE R ‘ ESEAUX DE NEURONES ´ POUR UNE ARTICULATION FLEXIBLE AVEC NON-LINEARIT ´ E DURE. D´ecembre 2002. URL https://depot-e.uqtr.ca/id/eprint/4019/1/000102241.pdf.

[5] R. S. S. et A. G. Barto. Reinforcement Learning :An Introduction, 2nd Ed. The MIT Press. 2018.

[6] J.-P. et S. Zeghloul. Robotique. Aspects fondamentaux, Modélisation mécanique. *CAO robotique – Commande*, Pari+ 9s 1994.

[7] D. Goldberg. Genetic Algorithms in Search. Optimization and Machine Learning. Reading MA Addison Wesley. 1989.

[8] H. Hoai Nam, E. Riviere, and O. Verlinden. Multibody modelling of a flexible 6-axis robot dedicated to robotic machining. 2018. URL https://www.researchgate.net/publication/326106790_Multibody_modelling_of_a_flexible_6-axis_robot_dedicated_to_robotic_machining/citation/download.

[9] J. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association of Computing Machinery*, 1962.

[10] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 1982. URL

https://fr.wikipedia.org/wiki/Réseau$_de_neurones_artificiels$cite$_ref$ − $GDT_1 - 0$.

[11] i. Jérôme Pouliot. La simulation 3D, le point de départ pour un projet robotique à succès ! *Centre de Robotique et de Vision Industrielles*, ON1 DÉCEMBRE 2010. URL https://magazinemci.com/2010/12/01/la-simulation-3d-le-point-de-depart-pour-un-projet-robotique-a-succes/.

[12] J. Lehman and R. Miikkulainen. Neuroevolution. *Scholarpedia*, 8(6) :30977, 2013. doi : 10.4249/scholarpedia.30977. revision #137053.

[13] P. Lessard. Contr^ole de la d´emarche de Headus robot quadrup'ede dynamique. 2002.

[14] G. Papy. Le mot de fonction a ´et´e introduit par Leibniz en 1694. Math´ematique moderne. *()*, 1968. URL https://fr.wikipedia.org/wiki/Fonction$_(mathématiques)$.

[15] M. Parizeau. Réseaux de Neurones. *(Le perceptron multicouche et son algorithme de retropropagation des erreurs)*, 2004. URL https://fr.wikipedia.org/wiki/Perceptron$_multicouche$.

[16] S. Paul. MSc in. *Artificial Intelligence Robotic*, 2020. URL https://www.quora.com/What-is-the-degree-of-freedom-in-robotics.

[17] K. Čapek (trad. Jan Rubes). R.U.R. *Rossum's Universal Robots,Éditions de La Différence*, 2011. URL https://fr.wikipedia.org/wiki/Robotcite$_ref - 1$.