UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE

"τ-HYPERNEAT: RETARDOS DE TIEMPO EN UNA RED HYPERNEAT PARA APRENDIZAJE DE CAMINATAS EN ROBOTS CON EXTREMIDADES MÓVILES"

OSCAR ANDRÉ SILVA MUÑOZ

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO, MENCIÓN COMPUTADORES

PROFESOR GUIA: **MARÍA JOSÉ ESCOBAR SILVA.**

PROFESOR CORREFERENTE: **FERNANDO AUAT CHEEIN.**

Agradecimientos

Resumen

Abstract

Glosario

Índice general

1. INTRODUCCIÓN Y OBJETIVOS

El Proyecto"τ-HyperNEAT: Retardos de Tiempo en una Red HyperNEAT para Aprendizaje de Caminatas en Robots con Extremidades Móviles" pretende incorporar conceptos temporales en una red neuronal HyperNEAT incluyendo retardos de tiempo adicionales a los pesos en las conexiones entre neuronas, permitiendo así generar caminatas en robots con distinta cantidad de extremidades móviles (ver Figura 1.1), a través de simulaciones en entornos virtuales, de forma más óptima y obteniendo resultados más cercanos a comportamientos encontrados en la naturaleza.

Fig. 1.1: Robots con distinto número de extremidades móviles.

1.1. OBJETIVOS DEL PROYECTO

Objetivo 1 Proponer un red neuronal usando HyperNEAT que incluya retardos de tiempo en sus conexiones.

Objetivo 2 Desarrollar el software necesario para manejar el entorno de simulación

a usar en el trascurso del proyecto.

Objetivo 3 Poner a prueba la nueva red neuronal en tareas de aprendizaje y realizar benchmarks para determinar su desempeño.

Objetivo 4 Usar la nueva red neuronal en tareas de aprendizaje de caminatas en robots con extremidades móviles.

1.2. TRABAJOS A DESARROLLAR

El proyecto se inicia en base a estudios e implementaciones previas de redes NEAT y HyperNEAT para la generación de caminatas en robots con extremidades móviles en entornos virtuales de simulación, con las cuales se obtuvieron resultados exitosos. A partir de esto es que se plantea la incorporación de retardos de tiempo a una red HyperNEAT de forma de implementar τ-HyperNEAT computacionalmente. Luego se debe comparar el desempeño de τ-HyperNEAT versus el desempeño de su predecesor, el cual se espera que sea mejor, para luego entrenar y generar caminatas en los robots. La correcta generación y evolución de caminatas en un entrenamiento está sujeta a una función de desempeño en base a las variables observadas en el robot, por lo que se debe realizar un estudio exhaustivo de cuál es la función de desempeño que mejor describe a una correcta caminata. Para el desarrollo de los entrenamientos de caminatas en los robots se debe implementar un modelo para cada robot en un entorno de simulación con el fin de observar las caminatas generadas y emular correctamente las dinámicas que se presentarían en un entorno real de forma de lograr traspasar posteriormente los resultados obtenidos a los robots reales. Además se debe desarrollar el software necesario para la comunicación con el programa de simulaciones, el cual soporta comunicación por sockets, y que este permita poder exportar directamente el trabajo al entorno real.

1.3. EVALUACIONES A REALIZAR

Una vez obtenidas las caminatas con τ-HyperNEAT resta comparar los resultados con los anteriormente obtenidos solo con HyperNEAT, tanto en el aspecto visual final de las caminatas obtenidas, como además en la evolución de dichas caminatas a lo largo del proceso de entrenamiento medida a través de las variables aplicadas para la calificación del desempeño de manera de validar el mejor funcionamiento de τ-HyperNEAT. Luego se debe comparar cuan influyentes fueron los retardos de tiempo incluidos en la red HyperNEAT observando la estructura y conexiones de la red τ-HyperNEAT finalmente obtenida para obtener las conclusiones del trabajo propuesto. Finalmente se debe evaluar el desempeño final obtenido usando la red final τ-HyperNEAT sobre los robots en el entorno real para comprobar el buen funcionamiento de las caminatas en ellos.

1.4. RESULTADOS ESPERADOS

Al culminar el Proyecto "τ-HyperNEAT: Retardos de Tiempo en una Red HyperNEAT para Aprendizaje de Caminatas en Robots con Extremidades Móviles", se espera poder obtener caminatas naturales y armónicas en robots con extremidades móviles de manera más óptima a las obtenidas solo con una red HyperNEAT, y lograr reproducir los mismos resultados en las plataformas robóticas reales emulando correctamente las dinámicas de los sistemas. De manera más general se pretende obtener una red neuronal más robusta y eficiente que permita resolver problemas reales con dependencias temporales. Además se espera generar un software robusto que permita una correcta comunicación con el entorno de simulación a usar para proveer esta herramienta a proyectos futuros en donde se requiera de emular sistemas reales complejos.

1.5. TÓPICOS A TRATAR

Las redes neuronales artificiales son un campo muy importante dentro de la Inteligencia Artificial. El estudio de las redes neuronales ha estado muy en boca durante las últimas décadas y su uso ha sido de gran relevancia para la solución de problemas difíciles de resolver mediante técnicas algorítmicas convencionales. Uno de estos problemas de difícil solución es la generación de caminatas en robots con extremidades móviles.

Con el objetivo de intentar emular movimientos más naturales en los robots estudiados es que investigadores han propuesto técnicas de neuroevolución, lo cual es una forma de aprendizaje de máquina que usa algoritmos evolutivos para entrenar a una red neuronal. Junto con esto y la implementación de algoritmos genéticos en redes neuronales, una de las líneas más prometedoras de la Inteligencia Artificial, se han logrado obtener los resultados esperados para este problema. Sin embargo lo que se busca en esta memoria es profundizar aún más en la solución a este problema e incorporar todos estos conceptos en una red neuronal adicionando variables temporales al modelo incorporando retardos de tiempo en cada una de las conexiones de la red.

Junto con el problema mismo que es la generación de caminatas en robots con extremidades móviles esta la tarea de simular virtualmente el modelo de cada robot, ya que la mayoría de las veces realizar pruebas en plataformas reales es inalcanzable, por elevados costos de adquisición de los equipos; muy poco práctico ya que requiere de una constante y prolongada intervención de personas; o muy peligroso, ya que cualquier problema o error podría incurrir en el deterioro del equipo o inclusive podría atentar contra la seguridad de las mismas personas que realizan los experimentos. Aun es más difícil poder traspasar los resultados del estudio realizado en un modelo simulado virtualmente al modelo real de forma directa. Es por esto que esta memoria contempla la implementación de una herramienta de software que permita trabajar con modelos virtualizados y reales de forma transparente para el usuario, de manera de dar facilidades para la implementación del modelo en un entorno virtual y luego

poder exportar todo el trabajo realizado y los resultados al entorno real con el solo hecho de ajustar los parámetros de trabajo.

Para el trabajo de simulación en el área de la robótica existen variadas opciones con distintos niveles de dificultad y costo de uso dependiendo del público objetivo para el cual está pensado. Es por esto que para el desarrollo de esta memoria se propone el uso de una herramienta de fácil acceso, tanto por el nivel de conocimiento que requiere su uso como su accesibilidad de descarga y sencillo manejo, con el objetivo de que el software a realizar este al alcance de uso de cualquier persona. Esto busca acercar a las personas a trabajar en el área de la robótica incitándolas con herramientas de fácil acceso y manejo.

1.6. TRABAJOS RELACIONADOS CON LOS TEMAS A TRATAR

En el área de redes neuronales que hacen uso de neuroevolución y algoritmos genéticos se puede observar el trabajo realizado por el investigador Kenneth O. Stanley, siendo el primer artículo de interés el que relata el desarrollo de NEAT [2], Neuroevolución a través del Aumento de Topologías, el cual supera en pruebas comparativas a redes con topologías fijas en tareas de aprendizaje reforzado. Stanley asume que el aumento en la eficiencia se debe al uso de un método de cruce entre diferentes topologías, a la clasificación por especies de redes diferenciadas por su topología y cambios en ella, y el crecimiento incremental a partir de una estructura mínima.

Una continuación del desarrollo de NEAT es HyperNEAT [1], Hipercubo basado en Neuroevolución a través del Aumento de Topologías, igualmente desarrollado por Stanley, el cual emplea una codificación indirecta llamada conectivo Compositional Pattern Producing Network (conectivo CPPNs), que puede producir un patrón de conectividad con simetrías y esquemas repetidos interpretado por el patrón espacial generado dentro de un hipercubo. La ventaja de este enfoque es que es posible explotar la geometría de la tarea mediante el mapeo de sus regularidades en la topología de

la red, desplazando con ello la dificultad del problema lejos de la dimensionalidad de este hacia la estructura misma del problema.

En el área de generación de caminatas en robots con extremidades móviles existe una vasta cantidad de investigaciones relacionadas tanto con el uso de HyperNEAT, algoritmos genéticos en general u otro tipo de técnicas que se expondrán a continuación.

Investigadores de la Universidad de Cornell el año 2004 publicaron "Evolving Dynamic Gaits on a Physical Robot" [3], en donde formularon un algoritmo genético para entrenar un controlador de lazo abierto para la generación de una caminata en un robot conformado por dos plataformas Stewart, evolucionando en búsqueda de optimizar su velocidad y su patrón de movimiento garantizando al mismo tiempo el ritmo de estos.

Otros investigadores del Centro de Investigación Ames, de la NASA, el año 2005 publicaron "Autonomous Evolution of Dynamic Gaits with Two Quadruped Robot" [4], en donde relatan cómo han desarrollado un algoritmo de evolución para generar caminatas dinámicas en dos robos cuadrúpedos, OPEN-R y ERS-110 de la marca Sony, midiendo el desempeño de las caminatas con los distintos sensores incorporados en ellos.

En el 2009, el investigador Jeff Clune junto a otros publicaron "Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding" [5], en donde demuestra cómo es posible desarrollar caminatas en robos cuadrúpedos sin realizar un trabajo manual para resolver el problema usando HyperNEAT.

En el 2011, investigadores de la Universidad de Cornell junto a un investigador de la Universidad de Chile publicaron "Evolving Robot Gaits in Hardware: the HyperNEAT Generative Encoding Vs. Parameter Optimization" [6], en donde presentan la investigación de variados algoritmos para la generación automática de caminatas sobre un robot cuadrúpedo, en donde se comparan dos clases, los de búsqueda local de modelos de movimientos parametrizados y la evolución de redes neuronales artificiales usando HyperNEAT. Aquí concluyeron que las caminatas desarrolladas con Hyper-

NEAT fueron considerablemente mejores a las desarrolladas con los otros métodos de búsqueda local parametrizada, y produjo caminatas casi nueve veces más rápidas que caminatas generadas a mano.

En el año 2013, un grupo de investigadores de la Universidad de Cornell, Universidad de Oslo y Universidad de Wyoming publicaron en conjunto "Evolving Gaits for Physical Robots with the HyperNEAT Generative Encoding: The Benefits of Simulation" [7], en el cual plantean y confirman la hipótesis de que los resultados de las caminatas generadas con HyperNEAT en un entorno simulado superan con creces a las generadas en un entorno real.

Recientemente investigadores de la Universidad de la Coruña han propuesto una extensión del algoritmo NEAT propuesto por Stanley llamado τ-NEAT [8] debido a que NEAT no es siempre fiable cuando existen manejos de variables temporales dentro de la tarea a solucionar debido a la falta de elementos temporales explícitos dentro de la topología de la red NEAT. Es por esta razón que el algoritmo τ-NEAT propuesto incluye la posibilidad de incluir retardos variables en las conexiones de la red NEAT, afectando tanto a las conexiones directas como recurrentes de la red.

El software de simulación que se usará para el desarrollo de esta memoria se llama V-REP [9], Virtual Robot Experimentation Plataform, desarrollado por Coppelia Robotics GmbH en Zurich, Suiza, el cual posee versiones tanto pagas como gratuitas. La versión educacional de este software es completamente gratuita y posee todas las funcionalidades del programa completo. Este software posee una API desarrollada en una gran variedad de lenguajes de programación, permitiendo así manejar todos los aspectos del programa y la simulación desde el exterior a través de sockets. Esta API será usada por la herramienta de software a desarrollar para el trabajo con los experimentos tanto en el entorno virtual como en el real.

Con respecto a la herramienta de software que se implementará para manejar trasparentemente las plataformas robóticas en el entorno de simulación y entorno real no se conoce implementación alguna disponible, por lo que se cree será una gran contribución a trabajos futuros.

2. ALTERNATIVAS DE SOLUCIÓN PARA EL DESARROLLO DEL PROYECTO

En este capítulo se darán a conocer las alternativas de solución consideradas para llevar a cabo el Proyecto "τ-HyperNEAT: Retardos de Tiempo en una Red HyperNEAT para Aprendizaje de Caminatas en Robots con Extremidades Móviles", concentrándose en cómo se logrará la generación de algoritmos de caminata usando herramientas de Inteligencia Artificial (IA) en un entorno de simulación virtual. El programa encargado de la generación de caminatas deberá hacer uso de una herramienta externa para la comunicación con los robots a manipular tanto en el entorno virtual como en el real, tal como se muestra en la Figura 2.1

Fig. 2.1: Diagrama de Flujo del Sistema Completo para la Generación de Caminatas.

Si bien el desarrollo de este proyecto de memoria es bastante acotado debido a lo específico que es el tema, se presentarán otras opciones con las cuales es posible la generación de caminatas en robots con extremidades móviles: Deep Learning en

redes neuronales, Reinforcement learning, además del método que usa HyperNEAT para la implementación de τ-HyperNEAT, propuesto para este proyecto. Con respecto al entorno virtual de simulación a usar, se optará por el software V-REP, y no se indagará en otras alternativas ya que se posee un gran manejo y experiencia en el uso de este software. Para la librería encargada de manejar los robots tanto en el entorno virtual como en el real se optará por una implementación en lenguaje C++ al igual que para la implementación de la herramienta de IA.

2.1. ALTERNATIVAS DE SOLUCIÓN

2.1.1. ALTERNATIVA N° 1: RETARDOS DE TIEMPO EN UNA RED HYPERNEAT

HyperNEAT, Hipercubo basado en el Aumento de Topologías, es un generador de codificación que evoluciona redes neuronales artificiales haciendo uso de los principios del algoritmo de Neuroevolución basado en el Aumento de Topologías (NEAT). Es una novedosa técnica para la evolución de redes neuronales de gran escala usando las regularidades geométricas del problema descritas por la red. Esta herramienta actualmente ha sido implementada para trabajo multiplataforma en lenguaje C++.

HyperNEAT básicamente es formado por dos redes neuronales: la red principal llamada substrato, y una segunda red neuronal NEAT usada para generar los pesos de las conexiones en el substrato. La topología del substrato es fija (no evoluciona), y está relacionada con la existencia de restricciones espaciales. El substrato está conformado por un número fijo de neuronas y capas: una capa de entrada, una capa de salida y capas intermedias; donde cada neurona de la red tiene una posición espacial asignada.

Dado un cierto conexionado realizado por la red NEAT en el substrato, se verifica el desempeño de la red HyperNEAT en una tarea dada y se le asigna a dicha red NEAT una calificación en relación directa con la efectividad de la red HyperNEAT en dicha tarea. Luego de calificar a un número determinado de redes NEAT, el algoritmo

de NEAT hace evolucionar dichas redes generando otras nuevas, con el objetivo de alcanzar mejores desempeños al utilizarlas para generar el conexionado del substrato de HyperNEAT. Esto se realiza ciclicamente hasta alcanzar el desempeño deseado de la red HyperNEAT.

A partir de la implementación de HyperNEAT es posible implementar el algoritmo propuesto llamado τ-HyperNEAT, el cual usa una red secundaria NEAT que no solo proporciona el peso de la conexión entre dos nodos, sino que además entrega el tiempo de retardo entre la conexión.

VENTAJAS Y DESVENTAJAS DE LA ALTERNATIVA

De acuerdo a lo antes expuesto, la alternativa de diseñar τ-HyperNEAT mediante la implementación anterior de HyperNEAT presenta las siguientes características:

- Al tener la implementación de HyperNEAT y por ende NEAT en C++, es posible realizar la implementación de τ-HyperNEAT sin mayor complicación.
- Al no usar ninguna librería externa para la implementación de las herramientas antes descritas permite la incorporación de τ-HyperNEAT en cualquier proyecto sobre sistemas operativos Windows, Linux o Mac OS.
- No existe implementación conocida de τ-HyperNEAT en particular, por lo que implementarla sería entrar en áreas inexploradas de la investigación.

Las siguientes alternativas expuestas no son factibles en base a la idea principal de este proyecto, que es precisamente incorporar retardos de tiempo en las conexiones de una red HyperNEAT para la generación de caminatas en robots con extremidades móviles, pero que si cumplirían con el objetivo de la generación de caminatas propiamente tal.

2.1.2. ALTERNATIVA N° 2: DEEP LEARNING EN REDES NEURONALES

Deep Learning [10] es una rama de Machine Learning basado en un conjunto de algoritmos que intentan modelar abstracciones de alto nivel de datos mediante el uso de varias capas de procesamiento con estructuras complejas, o de otra manera compuestas de múltiples trasformaciones no lineales. Deep Learning es parte de una familia más amplia de métodos de Machine Learning basado en la representación de los datos de aprendizaje. Una observación se puede representar de muchas maneras. Una de las promesas de Deep Learning está en remplazar funciones realizadas manualmente con eficientes algoritmos para aprendizaje de características y extracción de características jerárquica de manera no-supervisada o semi-supervisada.

La investigación en esta área intenta tomar mejores representaciones y crear modelos para aprender estas representaciones a partir de datos no etiquetados a gran escala. Algunas de las representaciones están inspiradas por los avances en neurociencia y se basan libremente en la interpretación de los patrones de procesamiento y comunicación de información en un sistema nervioso, tales como la codificación neuronal que intenta definir una relación entre varios estímulos y las respuestas neuronales asociados en el cerebro.

El aprendizaje profundo es una clase de algoritmos de aprendizaje automático que:

- Utiliza una cascada de muchas capas de unidades de procesamiento no lineal para la extracción de características y transformación. Cada capa sucesiva utiliza la salida de la capa anterior como entrada. Los algoritmos pueden ser supervisados o sin supervisión y las aplicaciones incluyen análisis de patrones (sin supervisión) y clasificación (supervisado).

- Se basa en el aprendizaje de múltiples niveles de características o de las representaciones de los datos (no supervisado). Características de nivel superior se derivan de características de nivel inferior para formar una representación jerárquica.

- Son parte del campo de aprendizaje automático más amplio de representaciones de aprendizaje de datos.
- Aprende múltiples niveles de representaciones que corresponden a diferentes niveles de abstracción; los niveles forman una jerarquía de conceptos.

Los algoritmos de aprendizaje profundo contrastan con los algoritmos de aprendizaje poco profundo por el número de transformaciones aplicadas a la señal mientras se propaga desde la capa de entrada a la capa de salida. Cada una de estas transformaciones incluye parámetros que se pueden entrenar como pesos y umbrales. No existe un estándar de facto para el número de transformaciones (o capas) que convierte a un algoritmo en profundo, pero la mayoría de investigadores en el campo considera que aprendizaje profundo implica más de dos transformaciones intermedias.

VENTAJAS Y DESVENTAJAS DE LA ALTERNATIVA

De acuerdo a lo antes expuesto, la alternativa de realizar la generación de caminatas mediante la implementación de Deep Learning presenta las siguientes características:

- Permite varios niveles de abstracción que ayudarían a los robots utilizados para generar algoritmos de caminata.
- Generalmente es usado para trabajar con elementos visuales para la extracción de características y no para la generación de caminatas, por lo que su uso en dicha área no ha sido muy explorado.
- Se cree que el aumento de las capas en una red neuronal usando Deep Learning aumente el grado de complejidad de la red volviendo el problema de generación de caminatas aún más complejo.

2.1.3. ALTERNATIVA N° 3: APRENDIZAJE REFORZADO

Cuando hablamos de Aprendizaje Reforzado (RL por su nombre en inglés, Reinforcement Learning) [11], nos referimos a un área de estudio dentro del Aprendizaje de Máquinas, donde el objetivo principal es resolver problemas de decisiones secuenciales modelados como Procesos de Decisión Markovianos (MDP). Sin embargo, por mucho tiempo se ha ampliado el espectro de aplicaciones a áreas como por ejemplo Robótica o Teoría de Control Automático.

En términos simples, el objetivo del aprendizaje por refuerzos es maximizar la recompensa esperada a largo plazo, en un entorno (inicialmente) desconocido mediante la búsqueda de una secuencia óptima de acciones a tomar para cierto problema. Mientras el agente (quien aprende) decide qué acciones tomar, debe hacer un balance de dos objetivos: explotar lo que ya se ha aprendido para evitar caer en soluciones que reporten una menor utilidad, o explorar nuevas soluciones que eventualmente permitan obtener una mayor utilidad a futuro. Este compromiso usualmente se conoce como el dilema de exploración-explotación, el que ha sido extensamente tratado en la literatura, teniendo métodos de exploración indirecta (como exploración de Boltzmann) que explora todo el espacio de estado-acciones al hacer una asignación del tipo probabilista a las diferentes acciones posibles, y métodos de exploración directa que usan información estadística obtenida en experiencias pasadas. Un problema de aprendizaje reforzado, formulado como un MDP está compuesto por (S, A, T, R) donde:

- S: corresponde al conjunto de todos los estados posibles.
- A: denota el conjunto de todas las acciones que el agente puede ejecutar.
- T: $S \times A \times S \rightarrow [0, 1]$ es una función de transición de estado, la que asigna una probabilidad de que el agente en el estado s ejecutando a sea trasladado al estado s'.
- R: $S \times A \rightarrow \Re$ corresponde a una función escalar (real) de recompensas.

- π: $S \rightarrow A$ es un mapeo de estados a acciones, describe la política (secuencia de acciones) que el agente tomará en cierto estado.

La Figura 2.2 muestra un esquema de la estructura que tiene un problema de aprendizaje reforzado.

Fig. 2.2: Esquema utilizado en el contexto de aprendizaje reforzado.

VENTAJAS Y DESVENTAJAS DE LA ALTERNATIVA

De acuerdo a lo antes expuesto, la alternativa de realizar la generación de caminatas mediante la implementación de Aprendizaje Reforzado presenta las siguientes características:

- Al tener un aprendizaje constante, la caminata se adapta de forma automática en caso de que la estructura del robot cambie o el tipo de terreno varíe.
- Los métodos tabulares (basados en tablas) son imprácticos en caso de que el espacio de estados discreto sea muy grande (imposible de utilizar en el caso continuo, necesitando aproximador de funciones como redes neuronales).
- El espacio de estados discreto aumenta de forma considerable conforme el número de extremidades o el número de grados de libertad por cada extremidad aumenta.

2.1.4. ENTORNO VIRTUAL DE SIMULACIÓN: VIRTUAL ROBOT EXPERIMENTATION PLATAFORM (V-REP)

V-REP (véase la Figura 2.3) es un simulador compatible con Windows, Mac y Linux, el cual permite el modelado de un sistema completo o de solo ciertas componentes, como sensores, mecanismos, engranajes u otros en poco tiempo. El programa de control de un componente puede estar unido a un objeto ligado o a una escena para modelarlo de manera similar a la realidad. Esta plataforma puede ser usada para controlar partes de hardware, desarrollar algoritmos, crear simulaciones de automatizaciones de fábricas o para demostraciones educativas.

Fig. 2.3: Imagen publicitaria del software de simulación V-REP.

V-REP puede ser usado de dos formas, vía script interno programado en Lua, o vía script externo con la ayuda de la API entregada por el programa, la cual se encuentra en C/C++, Python, Java, Urbi, Lua, Matlab y Octave. Su última versión data del 17 de mayo del 2016.

2.2. ALTERNATIVA SELECCIONADA

En esta sección se seleccionará una de las tres alternativas de solución que se han expuesto para el Proyecto de Titulación en sección anterior, utilizando para ello

diversos criterios que permitirán determinar la conveniencia de una u otra alternativa.

2.2.1. CRITERIOS DE SELECCIÓN

De acuerdo a lo expuesto anteriormente, se encontraron tres opciones que permiten generar caminatas en robots con extremidades móviles. Sin embargo lo acotado del problema a solucionar hará que los criterios presentados a continuación estén bastante sesgados para favorecer a la solución principal propuesta con anterioridad. Los criterios a considerar son los siguientes:

- **Cantidad de publicaciones en donde se use la alternativa para la generación de caminatas:** Mientras más publicaciones, información y referencias se tenga del trabajo realizado con la alternativa a elegir será más fácil obtener los conocimientos necesarios para implementar la solución.

- **Simplicidad en la programación:** Puesto que el tiempo para realizar el proyecto es limitado, una programación fácil ayuda a ocupar este recurso en forma óptima. Esto también puede verse afectado por el conocimiento previo que se tenga de la alternativa a elegir.

- **Conocimientos previos de la alternativa a elegir:** Conocimientos previos del tema a trabajar facilitaran en gran medida la implementación de la solución a efectuar.

- **Complejidad computacional de la alternativa a elegir:** Una variable importante a considerar es la complejidad computacional que puede tener la alternativa a elegir, lo cual puede limitar en gran medida a la generación de las caminatas debido a los retardos de procesamiento que se puedan generar.

La ponderación relativa que tienen cada uno de los criterios en la decisión final de la alternativa escogida, de acuerdo a las condiciones, recursos y tiempo del que se dispone son las siguientes:

Criterio de Selección	Porcentaje de Relevancia
Cantidad de bibliografía	30 %
Simplicidad en la programación	15 %
Conocimientos previos	35 %
Complejidad computacional	20 %
Porcentaje Total	100 %

Tab. 2.1: Ponderación de cada uno de los criterios de evaluación

Se ha dado un mayor porcentaje de relevancia a los conocimientos previos que se puedan tener de la alternativa a elegir debido a que daría más facilidad para comprender y solucionar el problema por su mejor manejo. Luego sigue la cantidad de bibliografía disponible, debido a que es fundamental poder informarse de los desarrollos realizados por otros investigadores con el fin de nutrir el trabajo a realizar y para efectuar comparaciones de desempeño. Posteriormente le sigue la complejidad computacional, punto realmente crítico si es que se desea obtener buenos resultados de las simulaciones. Finalmente, la simplicidad en la programación tiene cierta relevancia y se debe tomar en consideración, sin embargo no es un criterio demasiado crítico, ya que se dispone de basto conocimiento de ello.

2.2.2. EVALUACIÓN DE LAS ALTERNATIVAS

Para evaluar cada una de las alternativas se utilizará el siguiente sistema de puntuación:

Sistema de Puntuación				
Muy Deficiente	Deficiente	Aceptable	Bueno	Óptimo
0.1 - 0.2	0.3 - 0.4	0.5 - 0.6	0.7 - 0.8	0.9 - 1

Tab. 2.2: Sistema de puntuación utilizado para evaluar las alternativas

La puntuación de cada alternativa se detalla a continuación.

Retardos de tiempo en una red HyperNEAT (τ-HyperNEAT)

- **Cantidad de publicaciones en donde se use la alternativa para la generación de caminatas: Puntuación = 0.9** (Se pueden encontrar un gran número de publicaciones del tema).
- **Simplicidad en la Programación: Puntuación = 0.8** (Se posee un buen manejo de programación en C++ para cualquiera de las alternativas, pero al tener mayor manejo del tema se facilita aún más la programación).
- **Conocimientos previos de la alternativa a elegir: Puntuación = 0.9** (Se posee conocimientos bastos en el tema debido a trabajos realizados previamente).
- **Complejidad computacional de la alternativa a elegir: Puntuación = 0.7** (La complejidad computacional se ve perjudicada por tratarse de dos redes involucradas en el cálculo de la alternativa, sin embargo no perjudica mayormente al problema en sí).

Deep learning en redes neuronales

- **Cantidad de publicaciones en donde se use la alternativa para la generación de caminatas: Puntuación = 0.5** (Cantidad de bibliografía promedio).
- **Simplicidad en la Programación: Puntuación = 0.7** (Se posee un buen manejo de programación en C++ para cualquiera de las alternativas, pero no se posee gran manejo del tema).
- **Conocimientos previos de la alternativa a elegir: Puntuación = 0.4** (Se poseen conocimientos regulares del tema).
- **Complejidad computacional de la alternativa a elegir: Puntuación = 0.8** (No posee gran complejidad computacional manteniendo baja la cantidad de capas

de la red).

Aprendizaje reforzado

- **Cantidad de publicaciones en donde se use la alternativa para la generación de caminatas: Puntuación = 0.4** (Se encuentran un numero de publicaciones bajo el promedio).
- **Simplicidad en la Programación: Puntuación = 0.5** (Se posee un buen manejo de programación en C++ para cualquiera de las alternativas, pero no se posee ningún manejo del tema, complicando la tarea de programar los algoritmos).
- **Conocimientos previos de la alternativa a elegir: Puntuación = 0.6** (Se posee muy poco conocimiento del tema).
- **Complejidad computacional de la alternativa a elegir: Puntuación = 0.4** (Debido a la gran complejidad de los robots por su cantidad de grados de libertad implica que exista un espacio de estados discretos muy grande volviendo la solución impráctica).

En la Tabla 2.3 se evalúa cada una de las alternativas, de acuerdo a la ponderación que tiene cada criterio. Ella muestra, que con una puntuación total del 84.5 %, la opción más conveniente es la de Retardos de tiempo en una red HyperNEAT (τ-HyperNEAT), luego con un 55.5 % la de Deep learning en redes neuronales y finalmente con un 48.5 % la de Aprendizaje reforzado. **Con todo, la alternativa seleccionada para la generación de caminatas en robots con extremidades móviles es Retardos de tiempo en una red HyperNEAT (τ-HyperNEAT)**.

Criterio de Selección	Puntuación		
	τ-HyperNEAT	Deep L.	Reinforcement L.
Cantidad de bibliografía	0.9	0.5	0.4
Simplicidad en la programación	0.8	0.7	0.5
Conocimientos previos	0.9	0.4	0.6
Complejidad computacional	0.7	0.8	0.4
Puntuación Total	84.50 %	55.50 %	48.50 %

Tab. 2.3: Tabla de Evaluación de cada alternativa

3. BASE TEÓRICA

En este capítulo se presentará una introducción teórica de la herramienta de neuroevolución HyperNEAT y sus componentes, necesaria para la implementación de ?-HyperNEAT.

3.1. NEAT

Neuroevolución a través del Aumento de Topologías (NEAT) [2] usa algoritmos genéticos para hacer evolucionar la topología y los pesos entre conexiones de una red neuronal de acuerdo a una función de desempeño. Debido al fundamento que tienen los algoritmos genéticos, NEAT hace evolucionar la red neuronal en base al individuo más fuerte (de mejor desempeño), teniendo una gran probabilidad de crear una nueva generación de individuos o redes neuronales, y así preservar la especie.

Una de las principales ventajas de NEAT, comparada con otros tipos de redes neuronales, es que la estructura inicial de la red (topología) no es necesaria, pero es encontrada a través del proceso de neuroevolución del algoritmo mismo. Cada red NEAT es codificada por un genoma, como se muestra en la Figura 3.1, en donde la red es representada por Genes de Nodo y Genes de Conexión, representando a neuronas y conexiones respectivamente. Cada Nodo o Conexión tiene un único numero Id de innovación, el cual se preserva si el Nodo o Conexión es removido (como resultado de la evolución) o creado nuevamente.

Fig. 3.1: Ejemplo de Genoma usado en NEAT para codificar la red neuronal mostrada a su derecha.

La evolución del genoma que forma la red NEAT es impulsado por su desempeño dada una función de desempeño determinada. La función de desempeño determina la probabilidad de reproducción de un genoma en específico. Los hijos generados por el mecanismo de reproducción conservará los nodos y conexiones del padre y aleatoriamente cambiará los pesos de sus conexiones. Si una conexión solo proviene de uno de los padres, esta es heredada instantáneamente. Una especie es obtenida cuando un grupo de redes neuronales comparten similitudes entre ellas, esto es cuando, la distancia entre ellos está bajo un umbral (el cual depende de la topología de la red y los pesos de las conexiones). Redes neuronales pertenecientes a una misma especie tienen una alta probabilidad de convertirse en padres de una nueva población de redes neuronales. Similarmente a los algoritmos genéticos, la variabilidad en el proceso de reproducción en redes NEAT es dado por mutaciones. Aquí las mutaciones permiten la creación de nuevos nodos y la modificación de conexiones entre nodos (crear, eliminar o modular los pesos entre conexiones).

3.2. CPPN

falta aca

3.3. HYPERNEAT

Una extensión de NEAT, es el algoritmo de codificación generativa Hypercubo basado en Neuroevolución a través del Aumento de Topologías (HyperNEAT) [1]. HyperNEAT es formado por dos redes neuronales: la red principal llamada substrato, y una segunda red neuronal NEAT usada para generar los pesos de las conexiones en el substrato. La topología del substrato es fija (no evoluciona), y está relacionada con la existencia de restricciones espaciales. El substrato está conformado por un número fijo de neuronas y capas: una capa de entrada, una capa de salida y capas intermedias; donde cada neurona de la red tiene una posición espacial asignada.

Las conexiones entre neuronas del substrato son obtenidas a través de la segunda red neuronal implementada por NEAT. La red NEAT recibe como entrada la posición espacial de los nodos que van a ser conectados, representadas como coordenadas de una matriz abstracta (véase la Figura 3.2). La salida de la red NEAT corresponde al peso entre la conexión entre esas dos neuronas. Siguiendo esta implementación, HyperNEAT toma ventaja de las simetrías y regularidades en la geometría del substrato para resolver el problema planteado con sus restricciones espaciales.

Información adicional puede ser usada como entrada de la red para realzar las características geométricas de la red implementada, como por ejemplo la distancia euclidiana entre neuronas.

Fig. 3.2: Algoritmo HyperNEAT formado por dos redes neuronales distintas. La red principal 'Substrato' conforma la red con información estructural y puede ser formada por muchas capas. Las conexiones de pesos entre neuronas en el Substrato son obtenidas usando la red neuronal secundaria implementada por NEAT.

El beneficio de un cierto set de conexiones en la implementación de HyperNEAT es evaluada por una función de desempeño. El resultado del desempeño es posteriormente pasado a la red secundaria NEAT con el fin de evolucionar los pesos de las conexiones entre nodos. Se elige entonces el conjunto de pesos de conexiones dado el mejor valor de desempeño obtenido como una solución del problema de optimización.

Bibliografía

[1] STANLEY, K.O., D'AMBROSIO, D., and GAUSI, J. (2009). "A hypercube-based encoding for evolving large-scale neural networks". Artificial Life, 15(2):185-212.

[2] STANLEY, K.O., and MIIKKULAINEN, R. (2002). "Evolving neural networks through augmenting topologies". Evolutionary Computation, 10(2):99-127.

[3] ZYKOV, V., BONGARD, J., and LIPSON, H. (2004). "Evolving Dynamic Gaits on a Physical Robot", Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO.

[4] HORNBY, G.S., TAKAMURA, S., YAMAMOTO, T., and FUJITA M. (2005). "Autonomous Evolution of Dynamic Gaits with Two Quadruped Robots".

[5] CLUNE, J., BECKMAN, B.E., OFRIA, C., and PENNOCK, R.T. (2009). "Evolving coordinated quadruped gaits with the HyperNEAT generative encoding", In Proceedings of the IEEE Congress on Evolutionary Computing.

[6] YOSINSKI, J., CLUNE, J., HIDALGO, D., NGUYEN, S., ZAGAL, J.C., and LIPSON, H. (2011). "Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization", In Proceedings of the 20th European Conference on Artificial Life.

[7] LEE, S., YOSINSKI, J., GLETTE, K., and CLUNE, J. (2013). "Evolving Gaits for Physical Robots with the HyperNEAT Generative Encoding: The Benefits of Simulation".

[8] CAAMAÑO, P., BELLAS, F., and DURO, R. (2014). "τ-NEAT: Initial experiments in precise temporal processing through neuroevolution", International Joint Conference on Neural Networks.

[9] Virtual Robot Experimentation Plataform, Coppelia Robotics, Switzerland. http://www.coppeliarobotics.com/

[10] Deep learning, From Wikipedia, the free enciclopedia. https://en.wikipedia.org/wiki/Deep_learning

[11] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, volume 1. Cambridge Univ Press, 1998.