Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Desarrollo de algoritmos no supervisados para información sensorial en robótica autónoma

Estudante: Rubén Ares García
Dirección: Francisco Javier Bellas Bouza
Dirección: Óscar Fontenla Romero

A Coruña, setembro de 2019.

A mi familia y amigos.

Agradecimientos

Gracias a mi familia por su apoyo incondicional a lo largo de todo la etapa académica..
Agradezco a mis tutores, por hacer posible este trabajo.
Y gracias a Isabel por creer en mí cuando ni yo mismo lo hacía.

Resumen

Este trabajo de investigación pertenece al campo de la robótica autónoma y busca el desarrollo de algoritmos que, utilizando técnicas de clasificación no supervisada, permitan realizar un análisis previo de datos de información sensorial. Esta información será aportada por los agentes autónomos en lo que se vaya a integrar el sistema. Este proceso permitiría facilitar enormemente el actual proceso de aprendizaje de modelos existente en robots basados en MDB, los cuales aprenden modelos de mundo utilizando directamente algoritmos evolutivos sobre el conjunto de datos obtenidos de su entorno en tiempo real. El sistema realizaría una agrupación sobre los datos obtenidos, creando diferentes conjuntos de entrenamiento relacionados espacialmente. Estos conjuntos serían aprendidos posteriormente de forma paralela por el agente autónomo, el cual dispondría de un modelo específico para conjuntos de datos diferentes. Gracias a estos modelos el robot podría adaptarse de forma más eficiente a las diferentes situaciones que enfrente en su entorno.

A lo largo de este documento se detallará como se han desarrollado las diferentes etapas y componentes que integran el proyecto y se realizará un análisis global de los resultados obtenidos para evaluar la utilidad de los elementos desarrollados.

Abstract

This research work belongs to the field of autonomous robotics and seeks the development of algorithms that, using unsupervised classification techniques, allow a prior analysis of sensory information data. This information will be provided by the autonomous agents in which the system will be integrated. This process would greatly facilitate the current process of learning models existing in MDB-based robots, which learn world models directly using evolutionary algorithms on the set of data obtained from their environment in real time. The system would group the data obtained by creating different spatially related training sets. These sets would be subsequently learned in parallel by the autonomous agent, which would have a specific model for different data sets. Thanks to these models, the robot could adapt more efficiently to the different situations it faces in its environment.

Throughout this document it will be detailed how the different stages and components that make up the project have been developed and a global analysis of the results obtained to evaluate the usefulness of the elements developed.

Palabras clave:

- robótica
- cognición
- aprendizaje online
- aprendizaje supervisado
- clustering
- ogpc
- matlab
- mdb

Keywords:

- robotics
- cognition
- online learning
- supervised learning
- clustering
- ogpc
- matlab
- mdb

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Motivación

Uno de los principales problemas a los que se enfrenta la robótica autónoma es lograr una correcta toma de decisiones frente a una situación y entorno cambiante, buscando cumplir un objetivo determinado. Esta situación implica la necesidad de dotar al robot de capacidades cognitivas que le permitan actuar de una forma eficiente y efectiva ante situaciones desconocidas. Esto es algo que debido a su complejidad no puede ser solucionado por las estructuras y sistemas clásicos de control. La robótica cognitiva [1] busca suplir esta nueva necesidad utilizando el estudio de la cognición animal y humana para lograr un comportamiento inteligente y adaptativo en los robots.

Esta rama de la robótica no solo busca la obtención de agentes más inteligentes, sino también aportar nuevas herramientas que permitan estudiar las propias capacidades cognitivas humanas a partir del aprendizaje de estos robots, creando una relación de simbiosis entre ambos campos.

Durante años la robótica ha intentado dotar a los robots de autonomía utilizado estructuras de control cada vez más complejas y elaboradas directamente implementadas por el diseñador, obteniendo sistemas poco inteligentes, sumamente limitados por su diseño inicial y con una capacidad de adaptación muy escasa. Debido a la gran complejidad alcanzada en estos sistemas, se optó por utilizar aproximaciones diferentes, buscando una mayor similitud con la biología animal y humana, con el objetivo de dotar al robot de una autonomía mayor a lo largo de su ciclo vital.

Una de las bases de la robótica cognitiva es la creación de modelos que permitan procesos de razonamiento interno. La robótica del desarrollo cognitivo (CDR por sus siglas en inglés) [1] [2], obtiene estos modelos en tiempo real utilizando memorias que almacenan datos reales obtenidos de la interacción del robot con su entorno. Este TFG busca aportar una alternativa para mejorar este proceso de aprendizaje en tiempo real a través del uso de clustering online,

ya que el modelado directo de los datos obtenidos por el robot es complejo, dada su falta de contextualización.

1.2 Objetivos

El objetivo de este TFG es desarrollar un método de aprendizaje de modelos de mundo dentro la arquitectura cognitiva MDB[3], que se aplica en sistemas robóticos reales. Estos modelos de mundo se representan mediante redes neuronales[4], que deben ser ajustadas en tiempo de ejecución (on-line) a partir de la información sensorial que el robot adquiere. La complejidad del espacio de estados suele ser muy alta, por lo que es necesario desarrollar técnicas que permitan simplificar el aprendizaje. En este sentido, se plantea desarrollar un método on-line de clasificación no supervisada que analice los datos sensoriales y cree agrupaciones en función de sus características espaciales. Sobre estas agrupaciones se realiza el aprendizaje de los modelos de mundo[5], asumiendo que será más sencillo aprender un conjunto de datos con similitudes espaciales. Por tanto, el método de aprendizaje a desarrollar combina técnicas de clasificación no supervisada con técnicas supervisadas de entrenamiento de redes neuronales, y debe funcionar en tiempo de ejecución. En primer lugar, se desarrollará el entrenamiento de redes offline a partir de conjuntos clasificados mediante técnicas también offline con el objetivo de obtener una referencia para los resultados buscados en el posterior entrenamiento online del sistema.

1.3 Organización de la memoria

Para la estructura de esta memoria se optó por dividirla en los siguientes capítulos:

- **Capítulo 1, Introducción:** : Se explica el contexto del trabajo junto a los objetivos a alcanzar.
- **Capítulo 2, Fundamentos teóricos:** Descripción de los conceptos de aprendizaje sobre los que se sustenta el proyecto.
- **Capítulo 3, Antecedentes:** Sistemas basados en MDB y aprendizaje por refuerzo.
- **Capítulo 4, Fundamentos tecnológicos:** Descripción de las herramientas empleadas para cumplir los objetivos planteados.
- **Capítulo 5, Metodología:** Conjunto de tareas de ingeniería realizadas en el proyecto, describiendo cómo se llevaron a cabo.
- **Capítulo 6, Desarrollo:** Arquitectura global del sistema a desarrollar.
- **Capítulo 7, Pruebas de Funcionamiento:** Descripción de los experimentos realizados y análisis de los resultados obtenidos en ellos.
- **Capítulo 8, Conclusiones:** Análisis de los conocimientos adquiridos y el trabajo desarrollado.

También se añaden los siguientes apéndices:

- **Apéndice A, Acrónimos y siglas:** Significado de los acrónimos y siglas que aparecen en esta memoria.
- **Bibliografía:** Conjunto de referencias utilizadas.

Capítulo 2

Fundamentos teóricos

En este capítulo se describen los conceptos en los que se basa este proyecto, el aprendizaje online y técnicas de clustering.

2.1 Aprendizaje por refuerzo

El aprendizaje por refuerzo [6] es un área del aprendizaje automático cuyo objetivo principal es permitir a un robot conocer el comportamiento óptimo en una situación concreta a partir de un proceso de prueba y error. Este aprendizaje utiliza una serie de elementos entre los que se encuentran:

- Un conjunto de estados que representan los valores de las variables, externas o internas, que participan en el proceso a ejecutar.
- Una serie de acciones que el robot puede realizar.
- Una función que genere estas acciones en función del estado, conocida como "política".

Ejemplificando estos elementos en un experimento concreto compuesto por un brazo robótico que debe atrapar una pelota en movimiento, los estados estarían conformados por la velocidad y posición de la pelota, así como la propia posición del robot. Las acciones estarían descritas por los giros y velocidades que puede aplicar a sus motores y, finalmente, la política sería la encargada de relacionar ambos elementos para atrapar la pelota de forma correcta. Un cuarto elemento en este aprendizaje lo compondría el sistema de recompensas, el objetivo global se centraría en optimizar este último componente. En este ejemplo propuesto sería atrapar la pelota el que conformaría la recompensa principal a optimizar por el sistema. Además, podrían contemplarse otros aspectos con diferente grado de importancia para componer la recompensa completa, como, por ejemplo, la distancia recorrida por la pelota antes de ser atrapada o la consumición de energía.

En este proyecto se utilizará un método de aprendizaje por refuerzo basado en modelos. La mayoría de estos métodos se basan en el proceso de decisión de Markov, (partially-observable Markov decision process o POMDP)[5] que gobierna la relación entre los estados y acciones antes mencionados. Mientras que muchas de las investigaciones en el campo del aprendizaje por refuerzo no utilizan modelos, en el caso de la robótica cognitiva es básico, se busca aprender modelos explícitos con los que guiar el aprendizaje de la política.

2.2 Aprendizaje Supervisado

Los modelos internos que posee el sistema MDB se caracterizan por aprenderse de forma supervisada y en tiempo real, generalmente mediante redes neuronales. Estos modelos buscan obtener, a partir de una serie de datos observados por el robot y el estado actual en el que se encuentra, una estimación del estado sensorial en el instante siguiente. Su aprendizaje se realiza a partir de la comparación de las predicciones de salidas de las redes neuronales y las salidas reales, obteniendo así el error de predicción que permitirá modificar y ajustar adecuadamente los parámetros utilizados.

Como hemos dicho, las técnicas utilizadas de aprendizaje supervisado en este proyecto se componen de redes neuronales artificiales (Artificial Neural Networks o ANN)[4] que consisten en un modelo matemático que busca simular el comportamiento biológico de las neuronas.

Una red neuronal se identifica por una serie de características:

- Topología: Consiste en la organización y disposición de las neuronas que componen una red, concretamente el número de capas, número de neuronas en cada una de estas capas, grado de conectividad y tipo de conexión entre ellas.
- Mecanismo de aprendizaje: El aprendizaje de una neurona se define como los cambios que experimenta en el valor del peso de sus conexiones a otras neuronas.
- Representación de la información de entrada y salida: Dependiendo de la naturaleza de los datos de entrada y salida de una red, estas reciben una clasificación diferente. Si ambas componentes son analógicas la red se considerará una red continua mientras que si ambas utilizan valores discretos, la red será discreta. Existen también redes híbridas en las que la salida será discreta mientras que la entrada es continua.

En este proyecto se utilizan redes de diversas topologías con el objetivo de comparar cual aporta resultados más adecuados para resolver los problemas planteados. Como mecanismo de aprendizaje se utilizarán diferentes algoritmos populares en los problemas de regresión y ampliamente tratados en investigación, y como representación utilizaremos siempre redes continuas.

2.3 Clustering

El clustering es una técnica clave en la minería de datos, consiste en la división de un conjunto en grupos o clústers en los que los elementos dentro de un mismo clúster son similares o cercanos entre sí y diferentes o lejanos a elementos pertenecientes a otros clústers. Estos métodos son no supervisados, a diferencia de los métodos de aprendizaje supervisados no requieren un valor o etiqueta de referencia en la salida para realizar su aprendizaje, sino que trata al conjunto de entradas como una serie de variables aleatorias y construye un modelo a partir de ellas. Para realizar la clasificación, estos métodos se basan en la detección de características y correlaciones entre los datos y suelen requerir tiempos de entrenamiento menores a los métodos supervisados.

Dentro de la técnica de clustering existen numerosas variaciones y especializaciones, centradas en diferentes aspectos y objetivos dentro de la propia división de conjuntos. El proyecto se centrará concretamente en el clustering de partición tanto en su versión offline o clásica (K-means[7]) como en una de sus implementaciones online (OGPC[8])

2.3.1 Clusterting de partición offline

Se describe el clustering de partición offline como el conjunto de algoritmos de agrupamiento que utilizan un conjunto de datos inicial concreto (batch) para obtener un vector que represente el centro del clúster. Uno de los principales algoritmos de clustering es K-means, el cual consta de tres pasos diferenciados compuestos por:

- Inicialización: A partir de la selección de un número determinado de grupos (k) se establece un centroide por grupo que representará al conjunto de datos, la selección inicial de estos centroides puede seguir múltiples tipos de estrategias, desde métodos heurísticos hasta selecciones puramente aleatorias.

- Asignación de elementos a los centroides: Los elementos pertenecientes al batch se asignan entre los centroides disponibles por grado de cercanía, obteniendo progresivamente conjuntos de datos diferenciados.

- Actualización de centroides: La posición del centroide inicial se actualiza en función de los nuevos elementos que se van asignando a cada conjunto.

Mientras que la inicialización solo se produce al inicio de la aplicación del algoritmo, las otras dos etapas suceden secuencialmente hasta que se cumpla la condición de parada del k-means, normalmente esta condición se satisface al llegar a una distancia mínima umbral de movimiento de los centroides en su actualización

2.3.2 Clustering de partición online

El clustering online, a diferencia del offline, busca obtener una agrupación dinámica del conjunto de datos, adaptando iterativamente la distribución de los clusters adaptándolos a partir de los datos obtenidos progresivamente por el sistema con el objetivo de obtener agrupaciones más adecuadas y acordes a la situación en la se desenvuelva en cada momento. Este método permite realizar la clasificación en tiempo real sin disponer de todos los datos a priori.

OGPC

El OGPC (Online Graded Possibilistic Clustering) [8] es una aproximación de clustering online que busca aportar una solución a la clasificación de flujos de datos multidimensionales que pueden presentar tanto cambios sutiles a lo largo del tiempo, como grandes cambios repentinos y permitir adaptarse a su comportamiento de forma adecuada, para ello propone una combinación de elementos de entrenamiento batch, a partir de una ventana deslizante de elementos que se mantiene simultáneamente para analizar las tendencias del flujo de datos, así como estrategias online a partir de patrones.

En este algoritmo de clustering se utiliza el modelo de gradiente probabilistco c-Means que es un método de soft clustering, lo cual implica que la pertenencia a un clustering puede ser parcial, representadas por funciones de pertenencia representadas por valores reales y no enteros.

Dado un modelo de agrupación entrenado, es decir, un conjunto de centroides y un conjunto de anchuras de agrupación βj, explotamos las propiedades de las membresías posibles para evaluar el grado de atípica. Definimos valores atípicos como la pertenencia de una observación al concepto de "ser un valor atípico" con respecto a un modelo de agrupamiento dado. A diferencia de otros enfoques basados en el análisis de las distancias patrón-centroide, el modelo de posibilidades graduadas utilizado en este trabajo proporciona una medida directa de valores atípicos. Proponemos medir la masa total de membresía a los clústeres ζ, que, por definición del modelo de posibilidad graduada, no necesariamente es igual a 1, y medir si y cuánto es menor que 1. Cuantitativamente, definimos un índice Ω como sigue:

$$\Omega(x_l) = max\{1-\zeta_l, 0\}$$

El valor atípico se puede modular mediante una elección apropiada de α. Los valores bajos corresponden a un rechazo de valores atípicos más agudo, mientras que los valores más altos implican un clúster más amplio y, por lo tanto, menor rechazo. Para α= 1, el modelo se vuelve probabilístico y pierde la capacidad de identificar o rechazar valores atípicos. Observamos que utilizando la función:

$$\zeta_l = \Sigma_j u_l j \in (0,c)$$

Sin embargo:

- los valores $\zeta > 1$ son típicos de regiones bien cubiertas por centroides.
- El valor $\zeta >> 1$ es muy improbable para buenas soluciones de agrupamiento sin muchos grupos superpuestos.
- El valor $\zeta << 1$ caracteriza regiones no cubiertas por centroides, y cualquier observación hay un valor atípico.

El índice Ω se define como el complemento de uno de ζ, con valores negativos recortados como no interesantes. El índice de valor atípico es una medida puntual, pero puede integrarse para medir la frecuencia de los valores atípicos. Para una toma de decisiones nítida, un punto puede ser etiquetado como atípico cuando Ω excede algún umbral. Por lo tanto, es fácil contar la proporción (frecuencia) de valores atípicos sobre un conjunto dado de puntos de sonda S. Sin embargo, aprovechamos el hecho de que Ω expresa un concepto difuso, y en lugar de simplemente contar la frecuencia, podemos medir una densidad superior $\rho \in [0,1]$ definido de una de las siguientes maneras:

$$\rho M = \frac{1}{|S|} \sum_{l \in S} \Omega(x_l)$$

La densidad ρM cuenta tanto para la frecuencia como para la intensidad, o el grado de anomalía, de valores atípicos. Un número elevado de valores atípicos límite es equivalente a un número menor de valores atípicos más fuertes, siempre que su valor medio sea el mismo. Para dar más énfasis al caso en el que algunas observaciones tienen un valor atípico más alto, se puede utilizar una definición alternativa:

$$\rho RMS = \frac{1}{|S|} \sqrt{\sum_{l \in S} (\Omega(x_l))^2}$$

2.4 Aprendizaje online

El aprendizaje máquina representa uno de los principales campos en los que se ha centrado la inteligencia artificial en los últimos tiempos. Lograr que un sistema consiga aprender a realizar una tarea es crucial y es por ello que se han desarrollado numerosos algoritmos para conseguir este objetivo de una forma eficaz y sencilla.

Los paradigmas tradicionales se basan en la utilización de una batería de datos de la que se dispone a priori que, a través de un algoritmo de aprendizaje, es utilizada para entrenar

un modelo. Estos sistemas proporcionan buenos resultados para una gran cantidad de problemas, pero también posee una serie de limitaciones, las cuales se hacen considerablemente relevantes en problemas en tiempo real.

El aprendizaje online [9] busca solucionar las deficiencias del paradigma clásico al ser aplicado a problemas a tiempo real, principalmente:

- Adaptación ante nuevos datos: Mientras que los aprendizajes offline son entrenados y desplegados para su posterior uso, un aprendizaje online puede adaptarse a los nuevos datos recibidos e ir actualizando el modelo acordemente, algo completamente necesario en un sistema que trabaje en tiempo real y proporciona un flujo constante de información.
- Reducción de coste de reentrenamiento: Un algoritmo offline requiere un reentrenamiento completo para la incorporación de nuevos datos al modelo, mientras que el online puede incorporarlos a medida que se reciben.
- Reducción del coste de almacenamiento: Al no requerir mantener en memoria enormes conjuntos de datos con los que realizar el entrenamiento.

Este tipo de aprendizaje será el utilizado en este proyecto debido fundamentalmente a la falta de datos a priori, ya que se irán obteniendo en tiempo real a medida que el robot interactúa con su entorno. Esto permitirá al robot desarrollar la capacidad de realizar un proceso de adaptación en tiempo real, a medida que nuestro sistema reciba nuevos datos puede incorporarlos a su modelo y realizar un aprendizaje más adecuado en cada etapa de su ciclo de funcionamiento.

2.4.1 Descenso del gradiente

El descenso de gradiente[10] es uno de los algoritmos más populares para la optimización de redes neuronales. Su fundamento es la miminización de una función objetivo a partir de los parámetros de un modelo actualizando estos en la dirección opuesta al gradiente de la función objetivo, siguiendo la dirección de la pendiente de la superficie creada por la función objetivo hasta alcanzar un mínimo local.

El algoritmo de descenso de gradiente posee diferentes variantes, todas mantienen una estructura común y se diferencian en la cantidad de datos utilizados para computar el gradiente de la función objetivo. A más datos utilizados se obtiene una mayor precisión, aumentando el tiempo necesario para realizar una actualización,

Para el problema a desarrollar en este proyecto, la velocidad de aprendizaje es un aspecto clave para el funcionamiento deseado del sistema, por lo que se opta por la utilización de una versión mucho más rápida de este algoritmo, sacrificando precisión.

2.4.2 Descenso del gradiente estocástico

A diferencia de su implementación clásica, el algoritmo de descenso de gradiente estocástico[10] (SGD) realiza una actualización de sus parámetros por cada componente entrenada (x) para una etiqueta (y).

Esta actualización continua permite al SGD saltar a mínimos locales nuevos y potencialmente mejores, aunque también llega a complicar la convergencia de la función en el mínimo global por esta misma razón. Aunque se ha demostrado que ante la disminución de la tasa de aprendizaje gradual, SGD muestra el mismo comportamiento de convergencia que el algoritmo de descenso de gradiente clásico llegando a obtener el mínimo global.

Capítulo 3

Antecedentes

3.1 MDB

El aprendizaje de modelos que se utilizará en este proyecto se basa fundamentalmente en la arquitectura cognitiva Multilevel Darwinist Brain (MDB)[3] que consiste en una aproximación evolutiva para dotar a los robots de capacidad de adaptación a lo largo de todo su ciclo vital que le permitirá ir adquiriendo conocimiento a medida que interactúa con su entorno buscando cumplir unos objetivos determinados. Para lograrlo MDB aporta nuevos elementos al modelo cognitivo clásico:

- Estructuras de comportamiento: En lugar de utilizar una sola acción como en el modelo cognitivo, utiliza un nuevo concepto basado en múltiples acciones o secuencias de acciones.

- Elementos de memoria: El robot necesita tanto una memoria a corto plazo como una a largo plazo para aprender nuevos comportamientos.

La principal diferencia de la arquitectura MDB frente a otros modelos cognitivos es la forma en la que obtiene el conocimiento, utilizando para ello técnicas evolutivas que proporcionan modelos complejos debido a la propia naturaleza del proceso, puesto que al interaccionar con el mundo real, está expuesto a situaciones e incluso objetivos cambiantes a lo largo de su vida. A diferencia de otros problemas comunes en la robótica donde el objetivo es optimizar la respuesta del agente en un instante concreto, MDB busca una forma de, a partir de las experiencias vividas, realizar un aprendizaje gradual y continuo, siendo capaz de generalizar estas situaciones y obtener modelos robustos y reutilizables que puedan ser utilizados en situaciones similares futuras.

Inicialmente se implementaron algoritmos evolutivos clásicos y las redes neuronales utilizadas consistían en modelos de perceptrón de dos capas, obteniendo resultados satisfactorios

para entornos sencillos. Posteriormente se hizo patente la necesidad de modelos más avanzados para afrontar escenarios complejos. Una de las razones es que debido a la propia naturaleza de los algoritmos evolutivos, estos tienden a generar poblaciones muy similares entre sí, algo útil en otros escenarios de la robótica en la que el problema a resolver está más acotado, sin embargo, para un objetivo tan dinámico y cambiante como es otorgar un aprendizaje autónomo a un robot, esto es un limitación importante. Es por esto que el nuevo modelo evolutivo a utilizar debería ser capaz de aportar a la población resultante un componente de diversidad que permita su fácil adaptación a escenarios completamente diferentes. Para lograr esta tarea son necesarias dos componentes de memoria diferentes:

- STM: Short-Term Memory, la memoria a corto plazo, es la encargada de almacenar la información obtenida por el robot al interactuar con su entorno y es esta la información utilizada para generar e intentar predecir en los modelos del sistema. La principal característica de esta memoria es su capacidad limitada y las implicaciones derivadas de esto, ya que hace necesaria la sustitución continua de los datos almacenados para permitir la obtención de los nuevos que permitan continuar el aprendizaje, la realización de esta operación recae en una estrategia externa de reemplazo que permita una adaptación dinámica que pueda ceñirse a las necesidades cambiantes del agente. Profundizando en esta estrategia diferenciamos cuatro componentes necesarias:

  - El punto temporal en el que se obtiene la muestra (T) con el objetivo de priorizar muestras más antiguas a la hora de sustituirlas y mantener el aprendizaje enfocado a la situación actual.
  - La distancia entre muestras (D) compuesto por la distancia Euclídea entre los vectores de acción y percepción. Buscando obtener muestras generales del espacio evitando focalizar un punto concreto que limite la capacidad de aprendizaje del robot.
  - La complejidad de la muestra (C), el sistema prioriza muestras con una complejidad superior, medida a partir del error obtenido por el modelo actual.
  - La relevancia de la muestra (R), considerando una muestra muy relevante aquella que a pesar de presentar unos errores iniciales muy altos, los modelos consiguen aprenderla correctamente.

  Una obtenidos estos cuatro factores, cada muestra es almacenada y etiquetada con el resultado correspondiente (L) obtenido mediante una función aplicada sobre todos ellos, L = f(T,D,C,R) .

- LTM: Long-Term Memory, la memoria a largo plazo, es la encargada de almacenar el conocimiento del sistema, en la arquitectura MDB este conocimiento está representa-

do por los modelos y el comportamiento. La estrategia seguida por la LTM consiste en guardar aquellos modelos que funcionan correctamente, es decir, aquellos que permiten predecir los contenidos de la STM adecuadamente. En este caso, el modelo se guarda conjuntamente con los contenidos de la STM en el momento de su ejecución, para así mantener el contexto en el que funcionó correctamente. Con el objetivo de mantener este proceso actuando de forma eficiente, el sistema compara los modelos correctos entre sí para saber si son equivalentes y, en ese caso, evita guardarlos. Esta comprobación se realiza comparando cada modelo sobre el contexto del otro y se observa el grado de similitud que comparten.

Es necesario un tercer componente, la interacción de memoria, que administre ambas memorias y permita una "comunicación" para el correcto funcionamiento del sistema, con el objetivo de evitar situaciones problemáticas como por ejemplo, cuando la memoria LTM detecta un cambio en el entorno, comprobando que los modelos almacenados fallan continuamente ante los nuevos datos recogidos por la STM, es necesario purgar la información almacenada en la STM con el objetivo de obtener los datos nuevos y generar modelos y comportamientos adecuados.

3.2 Aprendizaje de modelos mediante MDB

Para ilustrar el funcionamiento del sistema MDB podemos observar el esquema de la figura 3.1 donde aparecen representado el modelo cognitivos y sus elementos:

- Modelo de mundo (W): Es la función que permite, a partir de unas entradas sensoriales externas (e) y una acción (a) ejecutada en un instante determinado (t), evaluar las consecuencias que esta genera en el instante siguiente (t+1), las cuales son representadas por las entradas sensoriales externas en este nuevo instante. Ejemplos de sensaciones externas para el robot serían distancias, ángulos respecto a elementos del entorno, etc.

- Modelo interno (I): Análogamente al modelo de mundo. Función que permite evaluar las consecuencias de una acción ejecutada en un instante determinado respecto a las sensaciones internas del agente (i), obteniendo así las sensaciones internas en el instante siguiente. Algún ejemplo de sensación interna podría ser el nivel de batería disponible o temperatura de CPU.

- Modelo de satisfacción (W): Es la función que permite obtener la satisfacción del robot a partir de las entradas sensoriales (tanto externas como internas) en un instante determinado. Es este modelo el que se encarga de predecir como de útil será una acción para el agente.

El elemento básico que utilizan estos modelos se conocen como par acción-percepción, el cual se compone de:

- Las entradas sensoriales en un instante (t)
- Acción realizada en ese instante (t)
- Entradas sensoriales en el instante siguiente (t+1)
- Satisfacción obtenida con esa acción

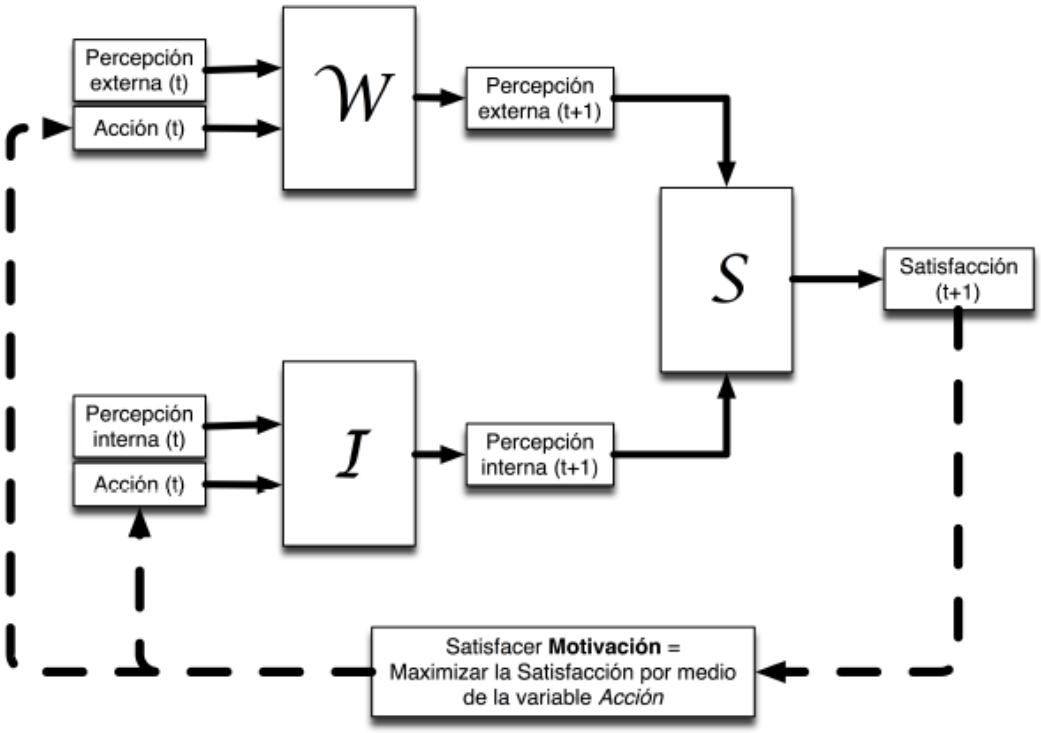

Figura 3.1: Esquema del aprendizaje de modelos

El proceso de aprendizaje de estos modelos funciona de la siguiente forma:

- Inicialmente el robot obtiene un nuevo par acción-percepción que almacena en su memoria STM, usando una estrategia de reemplazo.
- El agente aprende el modelo mediante la utilización de un algoritmo evolutivo.
- Se almacena este modelo aprendido en la memoria LTM con el objetivo de reutilizarlo en situaciones posteriores.

En este proceso no existe un análisis de la información sensorial obtenida, que será el desarrollo planteado en este TFG, con el objetivo de proporcionar una agrupación previa de estos datos que permita facilitar el aprendizaje de los modelos.

Capítulo 4

Fundamentos tecnológicos

En este capítulo se describen y justifican las herramientas tecnológicas utilizadas a lo largo de este proyecto.

4.1 Matlab

La herramienta seleccionada para el desarrollo del todo el proyecto es el software Matlab, el cual consiste en un sistema de cómputo numérico el cual dispone de un entorno de desarrollo integrado y un lenguaje de programación interpretado propio (lenguaje M) .

Se optó por esta herramienta debido a múltiples razones: Matlab era una herramienta utilizada a lo largo del grado en numerosas ocasiones, por lo que el equipo de desarrollo poseía un conocimiento previo del lenguaje, el entorno y varias toolbox disponibles. Muchas de las funciones y algoritmos utilizados, especialmente todo lo relacionado con las redes neuronales ya se encuentran implementadas y documentadas en Matlab, permitiendo un ahorro de tiempo significativo en el desarrollo de los experimentos. La amplia comunidad de desarrolladores que posee Matlab permite un fácil acceso a funciones e implementaciones que sería inviable encontrar en otros lenguajes. La experiencia previa en las funciones de representación de datos en Matlab permitió agilizar el desarrollo de métodos que permitieran analizar los resultados que, de tener que desarrollarse en otro lenguaje desconocido, hubieran incrementado el tiempo empleado en el proyecto considerablemente

4.1.1 Deep Learning Toolbox

Esta toolbox sirve como marco de desarrollo en el campo de las redes neuronales y el deep learning, aportando modelos y algoritmos que permiten utilizar e implementar de forma sencilla sistemas de aprendizaje máquina, así como múltiples herramientas gráficas con las que visualizar y monitorear las ejecuciones y los progresos de los sistemas implementados.

- Función'Adapt': Esta función permite el entrenamiento online de una red de neuronas artificiales. La función toma entradas de series temporales X, objetivos T. La red se adapta en respuesta a los datos de la serie temporal, luego la red actualizada se devuelve con las salidas Y, los errores E.
- Función 'trainscg': Función que permite el entrenamiento de una red neuronal mediante el algoritmo de gradiente descendiente estocástico. Será el principal algoritmo de entrenamiento utilizado en las redes neuronales presentes a lo largo del proyecto.
- Función OnlineGradedPossibilisticClustering [11]: Es la implementación del algoritmo OGPC descrito anteriormente, el cual nos permitirá realizar un clustering online de un conjunto de datos.

Capítulo 5

Metodología

Este capítulo se compone de la metodología utilizada durante el desarrollo del proyecto. Al tratarse de un proyecto de investigación y no un trabajo clásico de ingeniería existen una serie de diferencias en el desarrollo del mismo respecto a estos últimos

5.1 Roles en el proyecto

Los participantes en el proyecto se componen de:

- *Equipo de desarrollo*, compuesto por Rubén Ares García, es el encargado de realizar todo el desarrollo de los componentes software utilizados en el proyecto.
- *Equipo de investigadores*, compuesto por Francisco Javier Bellas Bouza y Oscar Fontenla Romero

5.2 Estructuración del proyecto

Este proyecto se basa en la realización de diferentes procesos de análisis, agrupación (clustering online y offline) y entrenamiento de sistemas mediante redes neuronales con el objetivo de demostrar que la utilización de técnicas no supervisadas puede facilitar la selección de conjuntos de datos para su posterior aprendizaje mediante un robot autónomo.

Para llevar a cabo su desarrollo se optó por la realización de iteraciones consecutivas en las que poder analizar los resultados obtenidos y variar o profundizar en las técnicas utilizadas a partir de ellos.

5.2.1 Iteración

En cada iteración se estipulan una serie de tareas a realizar, en un tiempo determinado cuyos resultados se analizarán al final de esta de cara a desarrollar la siguiente iteración del

proyecto

5.3 Gestión del proyecto

Para llevar a cabo la gestión del proyecto, debido a que solo era necesario realizar un control de las distintas tareas y no coordinar más de un desarrollador se utilizó la herramienta Trello [12] debido a su interfaz intuitiva y facilidad de uso. En la figura 5.1 podemos observar el diagrama de Gant con el transcurso de las diferentes etapas del proyecto

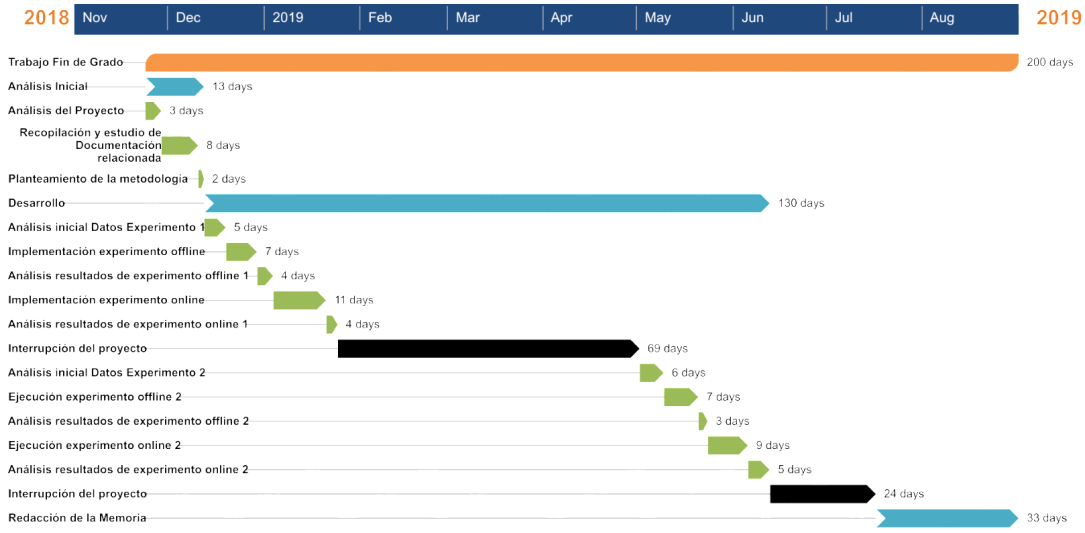

Figura 5.1: Cofiguración ejemplo aprendizaje modelos

5.3.1 Estimación

La estimación se llevó a cabo considerando el proceso a desarrollar, teniendo en cuenta que el equipo de desarrollo poseía unas nociones básicas en el lenguaje empleado (Matlab) y existían conocimiento previo por parte de los investigadores sobre qué configuraciones de aprendizaje se adecuarían mejor a los datos.

El tiempo de dedicación al proyecto fue variable a lo largo de su ejecución debido a motivos laborales, con interrupciones a lo largo de su desarrollo

5.3.2 Planificación

Las iteraciones que conforman el proyecto son las siguientes:

- **Iteración 1**: Análisis del dominio de datos y tratamiento inicial.
- **Iteración 2**: Desarrollo del entrenamiento offline del experimento 1

- **Iteración 3**: Desarrollo del entrenamiento online del experimento 1
- **Iteración 4**: Desarrollo del entrenamiento offline del experimento 2
- **Iteración 5**: Desarrollo del entrenamiento online del experimento 2

5.3.3 Recursos

El proyecto consta de dos recursos humanos representados por:

- Equipo de desarrollo: Rubén Ares García.
- Directores/Investigadores: Francisco Javier Bellas Bouza y Óscar Fontenla Romero

5.3.4 Gestión de riesgos

Debido al desconocimiento de la viabilidad de las técnicas utilizadas para obtener los resultados deseados en el desarrollo del proyecto, existía un considerable riesgo en la gestión del proyecto.

Clasificación de riesgos

Los riesgos identificados en este proyecto se muestran en siguiente la tabla:

Código	Descripción	Probabilidad	Impacto	Exposición
R1	Problemas con la tecnología	Media	Media	Media
R2	Desarrollo incorrecto	Media	Baja	Media
R3	Mala estimación en la planificación	Alta	Alta	Alta

Tabla 5.1: Riesgos identificados en el proyecto.

Prevención

Los riesgos con una probabilidad de exposición Alta deben ser analizados independientemente con el objetivo de ser evitados.

- **R3 - Mala estimación en la planificación**: La falta de experiencia unida al desconocimiento del dominio del problema por parte del equipo de desarrollo plantea un alto peligro de evaluar incorrectamente el tiempo necesario para desarrollar cada iteración, por lo que se buscará realizar una planificación individual de cada iteración con el fin de miminizar esta posibilidad al máximo

Seguimiento sobre los riesgos de exposición media

Debido a que los riesgos con una exposición media pueden convertirse en riesgos de alta exposición, se realizará un seguimiento que controle dichos riesgos:

- **R1 - Problemas con la tecnología**: Se utilizará la documentación disponible así como la ayuda aportada por el equipo de investigadores

Capítulo 6
Desarrollo

Actualmente dentro del MDB, en el aprendizaje de modelos de mundo a partir de los pares acción-percepción(AP), no hay análisis de los datos que llegan, y en muchos casos el modelado se vuelve demasiado complejo. Lo que buscamos en este TFG es una división automática del espacio de búsqueda tratando de encontrar patrones espaciales en los datos que hagan que sea más simple aprender varios modelos que aprender uno solo como hasta ahora. Realizaremos una serie de experimentos con el objetivo de comprobar si esto es posible y puede aportar una ayuda a los sistemas descritos.

6.1 Arquitectura global del sistema de aprendizaje de modelos

El funcionamiento del sistema se resume en la siguiente secuencia, representada esquemáticamente en la figura 6.1:

- El robot realiza una acción en el entorno recibiendo un nuevo par acción-percepción(AP), que será uno de los vectores utilizados como dato de entrada en nuestros entrenamientos.
- Estos pares se almacenan en una Short-Term-Memory (STM), utilizando una estrategia de reemplazo concreta. En este proyecto, la estrategia será seguir un criterio estrictamente temporal (sustituyendo pares más antiguos).
- Se realiza el aprendizaje de modelo de mundo utilizando los pares presentes en la STM, utilizando un algoritmo de entrenamiento online (algoritmo de descenso de gradiente estocástico), en cada iteración se reutiliza el modelo anterior logrando un aprendizaje incremental.
- Paralelamente, estos mismos pares AP se guardan en una memoria diferente que permita sobre la cual se realizará clustering progresivamente con los nuevos pares obtenidos.

- El clustering será, inicialmente, de dos clases y cuando se haya obtenido una clasificación suficientemente diferenciada, se entrenarán dos modelos paralelos, uno para cada cluster y se comparará el nivel de error obtenido con el modelo único aprendido a partir de la STM.
- Si estos dos modelos paralelos aportan una mejora respecto al original, pasarán a gestionarse dos STM paralelas y el algoritmo de clustering decidiría a cual mandar cada nuevo vector.
- En este caso el sistema de clustering probará a realizar una separación en tres clases para comprobar si puede mejorar aún más el error.
- El criterio de parada de esta subdivisión será inicialmente tres clases, aunque podría establecerse en más.

En el siguiente fragmento de pseudo-codigo podemos observar el proceso de clasificación y entrenamiento de las redes online utilizadas en los experimentos descritos a continuación.

```
Inicio
    Desde posicion = 1 Hasta FinDataSet Con Paso 1 Hacer
        clase = clasificar par(posicion)

    Si clase = 1
      Almacenar par en Memoria 1
    Sino
      Almacenar par en Memoria 2
    Fin Si

    Si tamaño(memoria 1) >= 100
      Añadir memoria 1 a Conjunto de entrenamiento 1
      Entrenar red 1(Conjunto de entrenamiento 2)
    Fin Si

    Si tamaño(memoria 2) >= 100
      Añadir memoria 2 a Conjunto de entrenamiento 2
      Entrenar red 2(Conjunto de entrenamiento 2)
    Fin Si
     FinDesde
Fin
```

En la figura 6.2 podemos observar el estado del sistema en el siguiente paso a la utilización de dos modelos paralelos. El sistema mantendría simultáneamente dos STM con dos modelos, unos para cada clase, mientras paralelamente buscaría realizar un clustering en tres clases y generar un modelo de mundo para cada una al obtener suficientes muestras. Una vez entrenados los modelos se realizaría la comparativa entre los errores obtenidos por los dos modelos

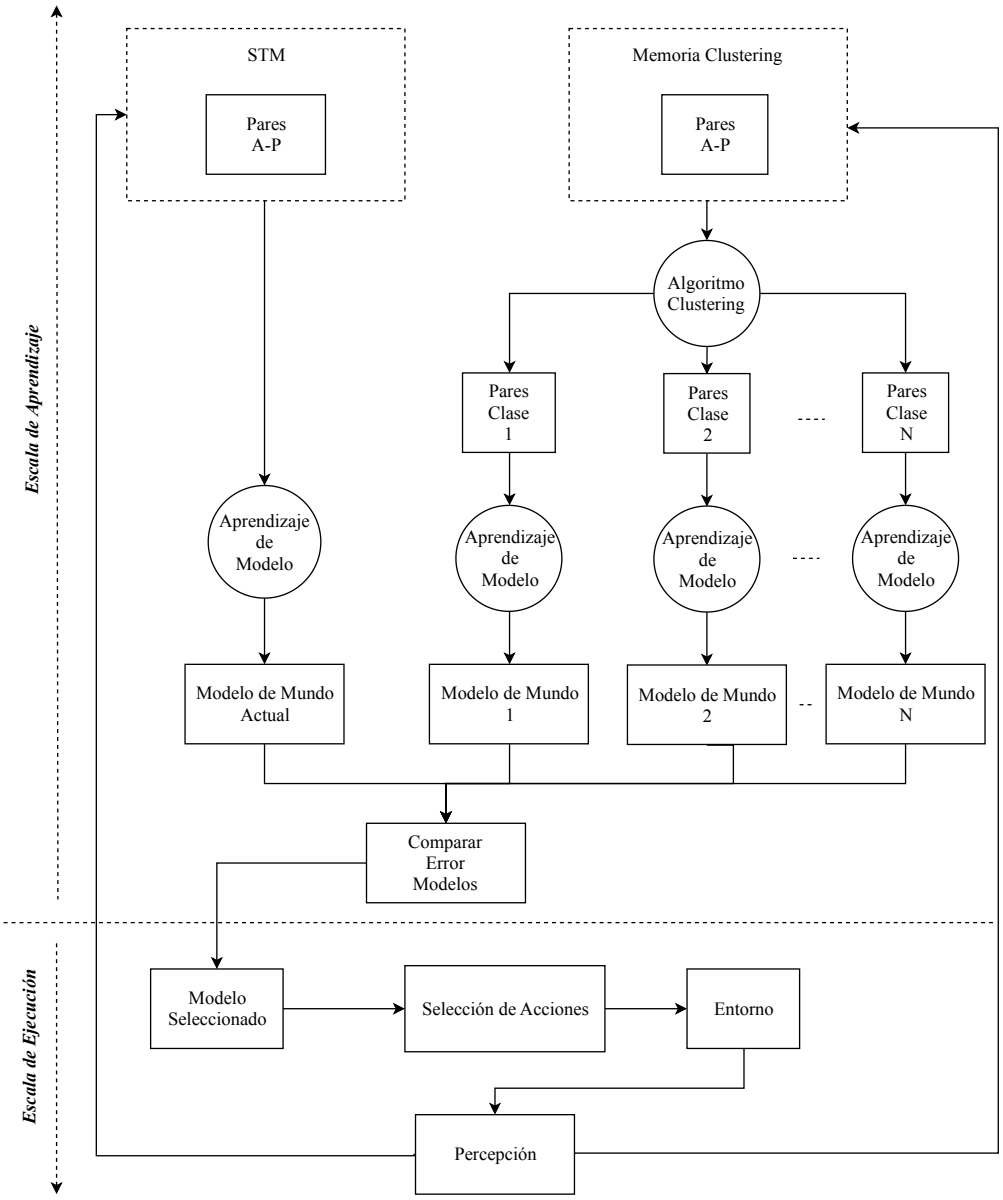

Figura 6.1: Esquema del aprendizaje de modelos

utilizados y los tres nuevos generados. En caso de conseguir mejores resultados con estos últimos, el sistema pasaría a mantener tres modelos simultáneos, y continuaría el proceso hasta alcanzar el criterio de parada establecido.

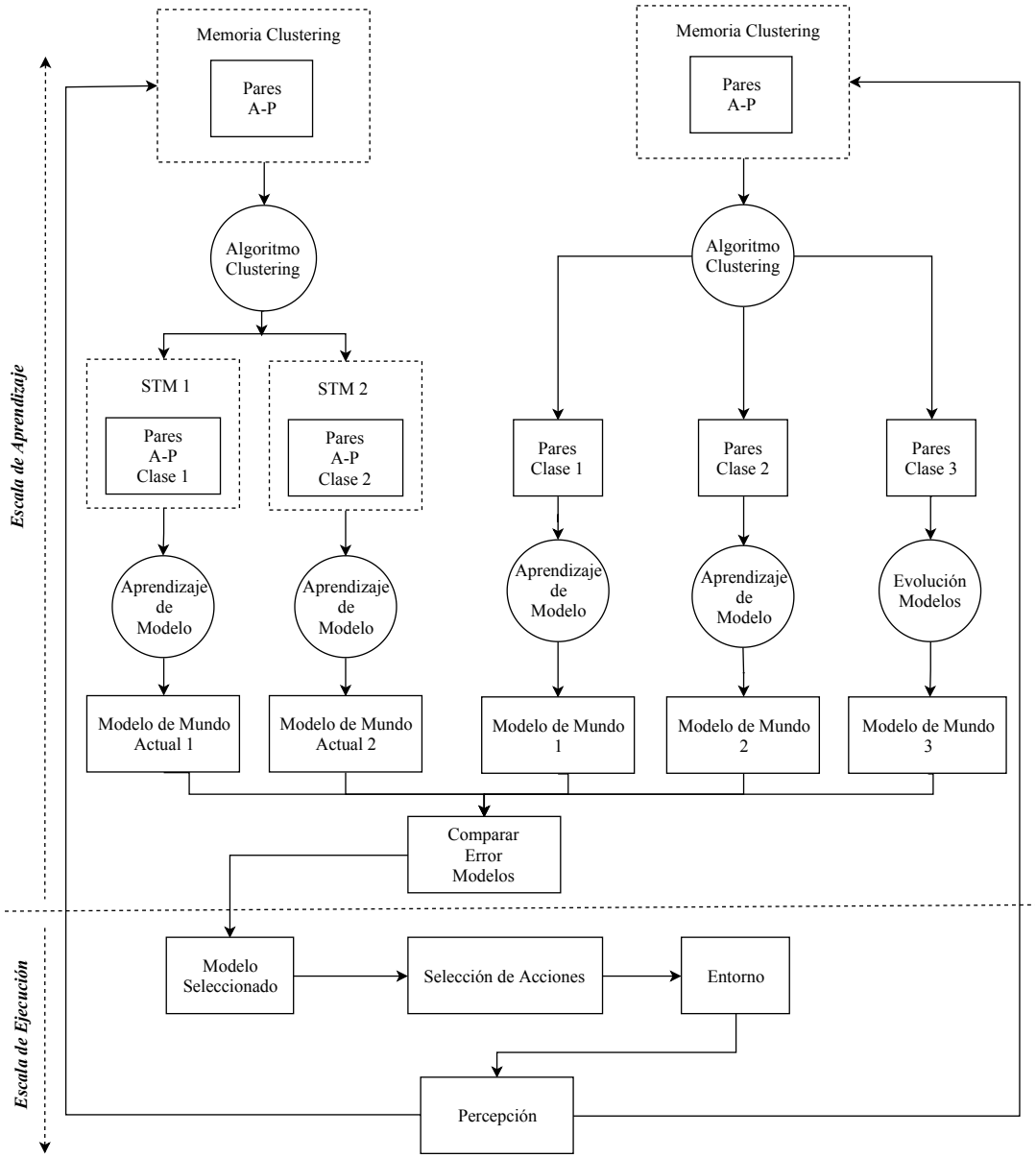

Figura 6.2: Cofiguración ejemplo aprendizaje modelos

Capítulo 7

Pruebas de funcionamiento

En este apartado se detallan los experimentos realizados con el fin de comprobar la viabilidad del método propuesto de cara a su posible implementación en un sistema real. Las pruebas consistirán en el análisis y clasificación de diferentes datos de ejecuciones reales de robots autónomos, los cuales intentaremos agrupar de forma no supervisada, y posteriormente entrenar redes con los diferentes conjuntos obtenidos y comparar el error obtenido con redes entrenadas sobre el conjunto completo de datos sin agrupar.

En el diagrama 7.1 podemos observar representada la secuencia que realizará el algoritmo propuesto en su funcionamiento en tiempo real.

Los elementos del conjunto de datos se procesan secuencialmente uno a uno simulando el comportamiento real del robot en el que obtendría estos datos sensoriales interactuando con el entorno. El algoritmo de clustering los clasifica en clase 1 o clase 2 y lo almacena en su respectiva memoria temporal. Esta memoria temporal se llena hasta alcanzar un determinado numero de elementos y es entonces cuando se añaden sus elementos al conjunto de entrenamiento. Cada vez que este conjunto de entrenamiento recibe nuevos datos, se utiliza para reentrenar la red online y se prueba con un conjunto de test elaborado previamente. Finalmente, recopilamos los resultados obtenidos para valorar el funcionamiento del sistema.

7.1 Experimento 1

Este experimento consiste en un robot que realice la colocación de un objeto en una posición predeterminada sin tener conocimiento previo del entorno. El objetivo es utilizar los datos que obtiene el robot en tiempo real para generar uno o varios modelos de mundo. El método propuesto intentará agrupar la información sensorial obtenida y entrenar modelos específicos para cada agrupación de datos. Para su ejecución se emplea un robot Robobo [13], el cual se basa en una base móvil que porta un Smartphone el cual provee al robot con funcionalidades de alto nivel. Además del robot, se utiliza una mesa con un “área objetivo” marcada

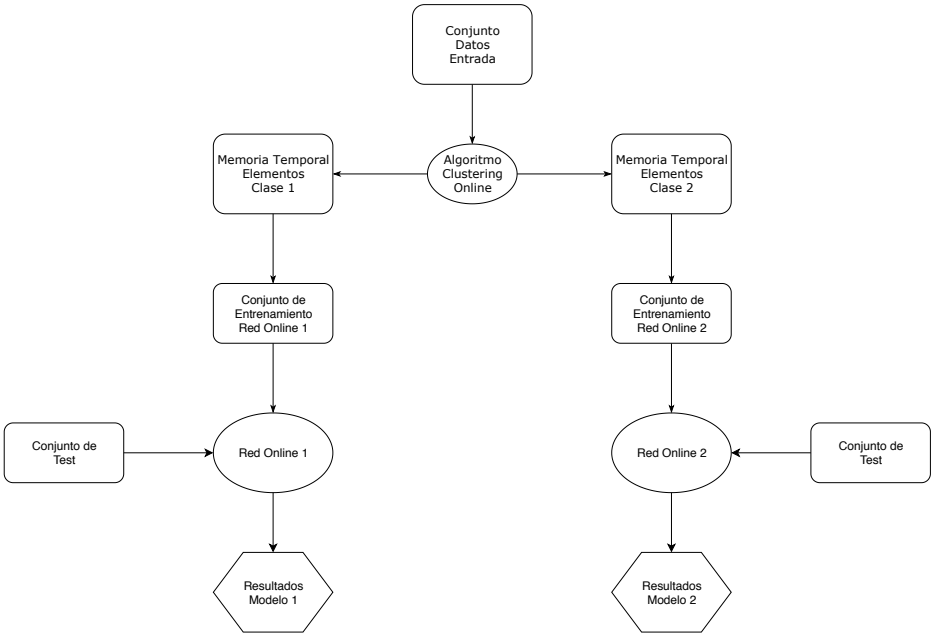

Figura 7.1: Esquema del aprendizaje de modelos

en violeta, un cilindro rojo móvil y dos balizas, una verde y otra azul a cada lado del área objetivo. Este escenario puede verse representado en la figura 7.2. El robot comienza siempre con el cilindro enganchado en la pinza y su objetivo será depositarlo en el área objetivo utilizando dos sensores que proporcionan la distancia (en metros) a cada una de las balizas.

Para este experimento se dispone de una colección de datos con las siguientes componentes de entrada:

- *distance1(t)* es la distancia entre la baliza verde y el robot.
- *distance2(t)* es la distancia entre la baliza azul y el robot.
- *orientation* es el ángulo en el que va a desplazarse en t.

Y dos componentes de salida:

- *distance1(t+1)* es la distancia a la que está el robot de la baliza verde en el momento t+1 tras haberse desplazado en el ángulo *orientation*.
- *distance2(t+1)* es la distancia a la que está el robot de la baliza azul en el momento t+1 tras haberse desplazado en el ángulo *orientation*.

Por lo tanto, el modelo de mundo a obtener se compondría de tres entradas y dos salidas.

7.1.1 Análisis Inicial

Con el objetivo de establecer una tipología de red adecuada y conocer la complejidad del conjunto de datos, se ha realizado un análisis inicial del problema. Para ello, disponemos de un conjunto de dos mil vectores(pares AP) conformados por las cinco componentes obtenidas en la ejecución real de este experimento.

En primer lugar se aplica un preprocesado sobre el conjunto completo de datos que consiste en una normalización de los valores entre 0 y 1 con el objetivo de mantener los resultados en una escala estándar para su posterior análisis.

Para realizar el entrenamiento de la red se opta por la utilización del algoritmo de gradiente descendiente estocástico que, a diferencia de la aproximación clásica del algoritmo de gradiente descendiente, es considerablemente más rápido y nos permitirá encontrar más fácilmente los errores mínimos en nuestros entrenamientos.

Utilizamos el error medio cuadrático del conjunto de entrenamiento y de test, así como la desviación típica de estos para comparar el rendimiento de las diferentes ejecuciones. Cada resultado se obtiene a partir de la aplicación de un validación cruzada de 10 iteraciones sobre cada una de las arquitecturas propuestas. Esta técnica consiste en distribuir el conjunto de datos totales en 10 paquetes de datos diferentes. En cada uno de los experimentos un elemento

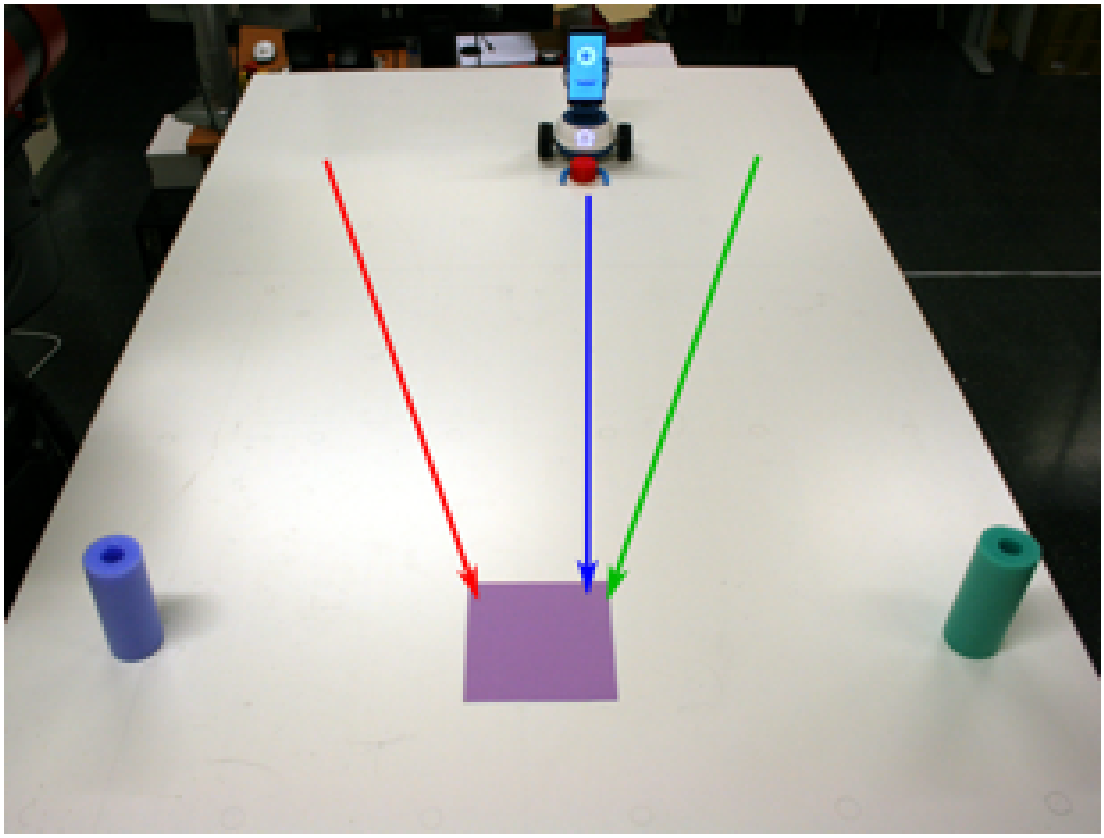

Figura 7.2: Escenario del experimento. Configuración inicial ejemplo del experimento

Validación cruzada				
	Conjunto salida 1		Conjunto salida 2	
Topología	Error entrenamiento	Error test	Error entrenamiento	Error test
Red 3	3.03E-04	3.16E-04	3.19E-04	3.23E-04
Red 5	2.58E-04	2.81E-04	2.80E-04	2.92E-04
Red 10	2.12E-04	2.33E-04	2.32E-04	2.57E-04
Red 3-3	2.99E-04	3.15E-04	3.13E-04	3.25E-04
Red 5-5	2.28E-04	2.50E-04	2.33E-04	2.51E-04
Red 10-10	1.65E-04	2.25E-04	1.83E-04	2.22E-04

Tabla 7.1: Resultados entrenamiento inicial

formará parte del conjunto de test una sola vez y del conjunto de entrenamiento nueve veces. Una vez ejecutados los procedimientos en las diez diferentes distribuciones, se calcula la media de rendimiento de todas ellas, obteniendo así el resultado final que utilizaremos en nuestras comparativas. En cuanto a las arquitecturas utilizadas, se opta por una variedad de redes de una y dos capas de neuronas ocultas, compuestas cada una por tres, cinco y diez neuronas. Esto nos permite observar el comportamiento del conjunto ante redes con una menor o mayor complejidad.

Creamos seis redes neuronales prealimentadas independientes sobre cada conjunto de datos con diferentes arquitecturas: tres redes con una sola capa oculta compuesta por tres, cinco y diez neuronas respectivamente y tres redes con dos capas ocultas compuestas por tres, cinco y diez neuronas cada una. La distribución del conjunto de entrenamiento-test-validación será en porcentajes 70-15-15 respectivamente. En la Tabla 7.1 se observan los resultados del error medio cuadrático de las diferentes redes para cada uno de los conjuntos.

La notación utilizada para representar las arquitecturas de red representa cada capa con un dígito con el número de neuronas que la componen, por lo que la 'Red 3' indica una sola capa con tres neuronas ocultas mientras que la 'Red 10-10' representa una red con dos capas de diez neuronas en la primera capa y diez neuronas en la segunda

Las diferencias entre los resultados de regresión entre las redes de ambos conjuntos son muy similares, siendo ligeramente peores los obtenidos en las redes del conjunto que utiliza la salida dos. Podemos suponer que esta salida es más problemática que la salida uno, por lo que centraremos las pruebas siguientes en este conjunto de datos. Podemos observar la distribución de la salida de la red de tamaño 10-10 respecto a la salida real en la figura 7.3

7.1.2 Funcionamiento offline

Partiendo del conjunto de datos seleccionado, se procede a aplicar un clustering de dos y tres clases sobre el conjunto de datos, utilizando el algoritmo K-means, con el objetivo de

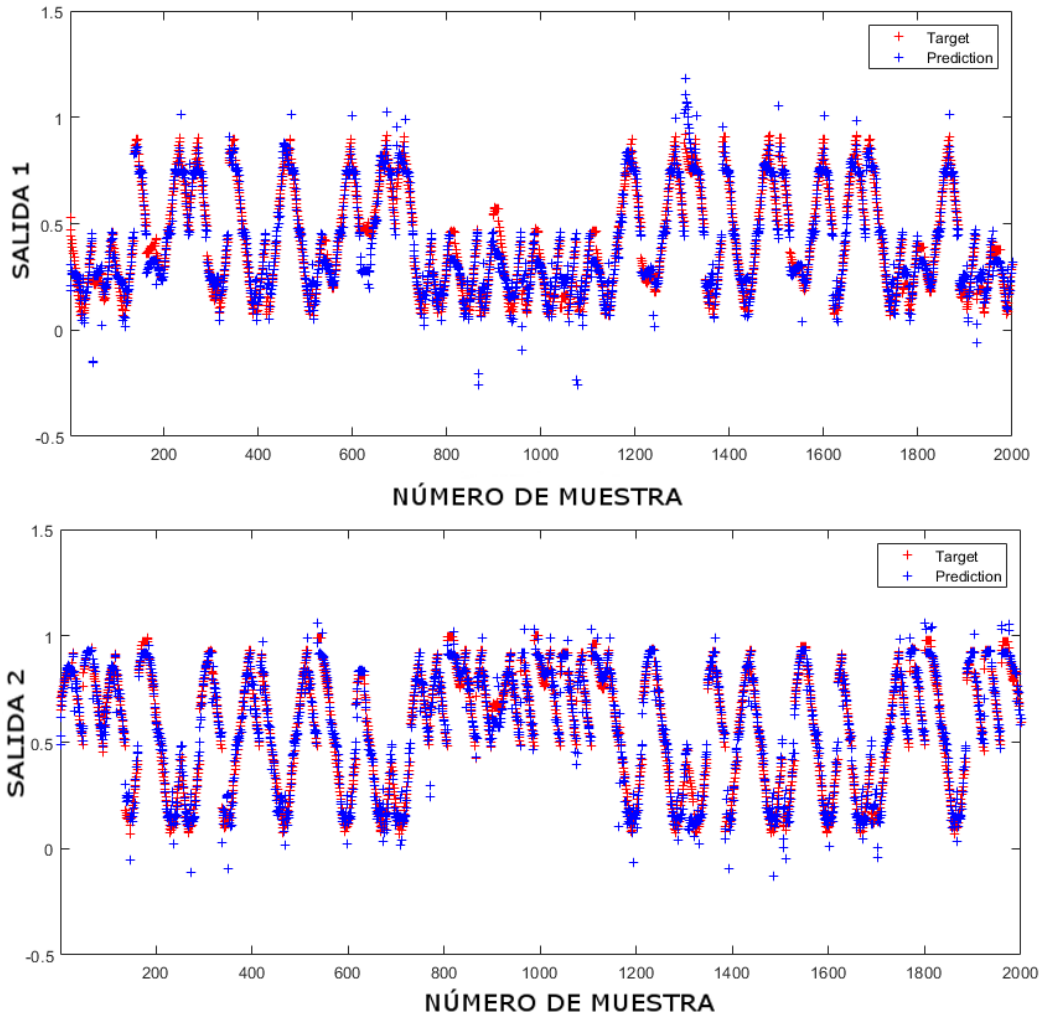

Figura 7.3: Distribución de la salida real frente a la salida de la red neuronal

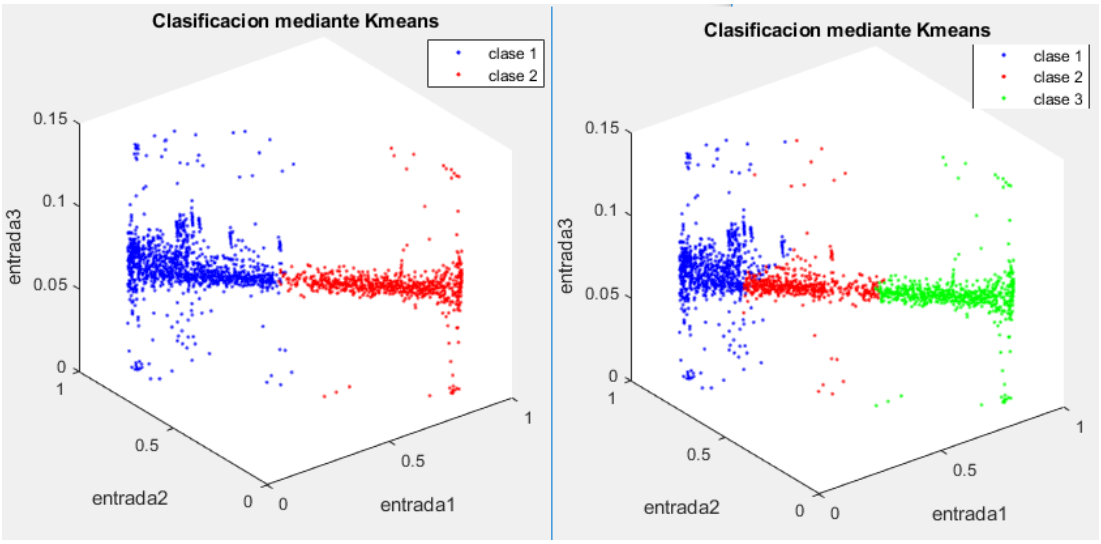

Figura 7.4: Distribución subconjuntos clustering mediante K-means

obtener subconjuntos similares y mejorar así la capacidad de las redes para obtener un entrenamiento óptimo.

Se observa gráficamente en la figura 7.4 como el clustering subdivide el conjunto en dos y tres clases de similar tamaño, manteniendo los conjuntos de entrenamiento equilibrados en cuanto a número de muestras a entrenar. Para evaluar el entrenamiento sobre los subconjuntos se utilizan las mismas métricas que en el experimento anterior, errores cuadráticos y desviaciones medias tras la aplicación de la validación cruzada de diez ejecuciones. En cuanto a las arquitecturas utilizadas, realizaremos pruebas con la configuración de red que mejor resultado obtuvo en el análisis inicial (tabla 7.1), es decir, una sola capa oculta con diez neuronas. Obtenemos así los resultados mostrados en la Tabla 7.2 para cada uno de los subconjuntos entrenados, incluyendo también los resultados de aplicar el entrenamiento sobre el conjunto completo de datos sin clustering previo para realizar la comparativa.

Con estos resultados podemos observar que el error de entrenamiento disminuye considerablemente tanto en todos los subconjuntos de clustering, que el error de test mejora ligeramente en los subconjuntos del clustering en dos clases y empeora en los de tres.

A continuación observamos la gráfica de la salida real frente a la predicción de red para la red entrenada con el conjunto completo de datos (Figura 7.5) y la misma representación para los dos subconjuntos del clustering en dos clases (7.6).

Se observa que la gráfica muestra una clara división entre ambas agrupaciones. La primera red clasifica los elementos de clase 1, en la que se agrupan mayoritariamente los vectores con salidas reales comprendidas entre los valores [0,0.5], mientras que la red que clasifica los elementos de la clase 2, agrupa aquellos elementos cuya salida real abarca los valores entre

	No clustering	Clustering 2 clases		Clustering 3 clases		
		Clase 1	Clase 2	Clase 1	Clase 2	Clase 3
Error Entrenamiento	1.85E-04	1.42E-04	1.38E-04	1.25E-04	9.67E-05	1.35E-04
Error Test	2.35E-04	2.08E-04	2.15E-04	2.21E-04	2.78E-04	2.78E-04
Desviación Entrenamiento	9.66E-06	6.97E-06	7.08E-06	9.78E-06	5.78E-06	1.13E-05
Desviación Test	3.68E-05	6.97E-06	9.03E-05	1.71E-04	2.15E-04	2.02E-04

Tabla 7.2: Resultados entrenamiento sobre conjunto completo y subconjuntos clustering

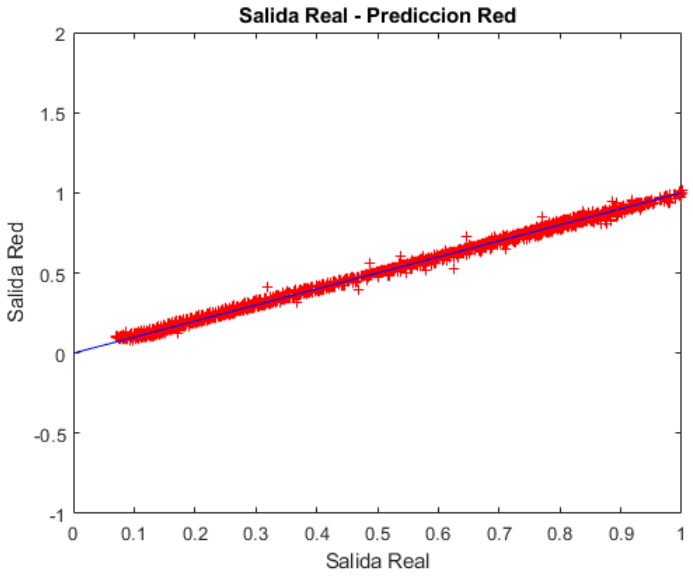

Figura 7.5: Representación salida real frente a predicción de red conjunto completo

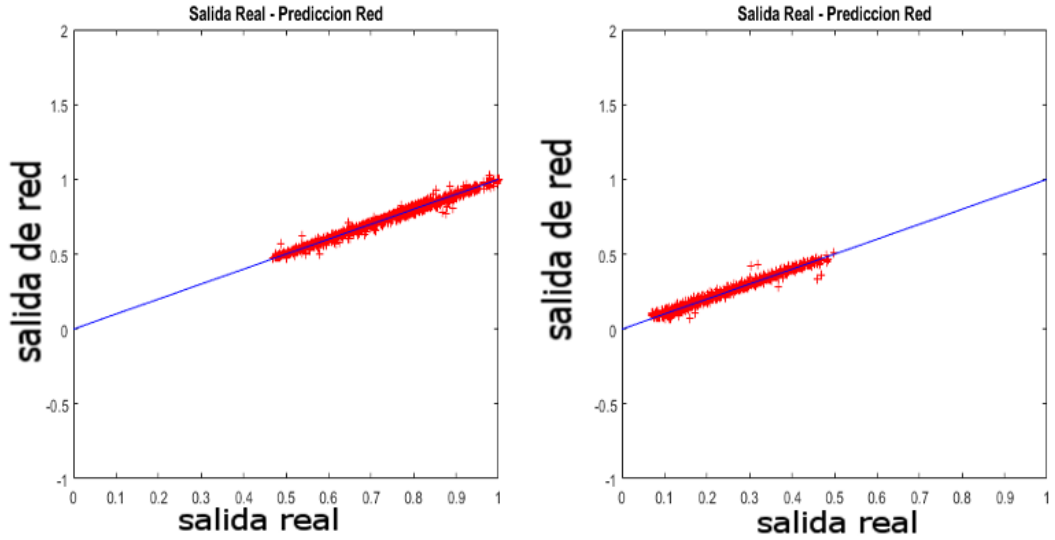

Figura 7.6: Representación salida real frente a predicción de red subconjuntos clustering 2 clases

[0.5, 1]. En la 7.7 se muestra la misma representación para los tres entrenamientos de los subconjuntos del clustering para tres clases, donde los tres conjuntos se distribuyen claramente en tres rangos de salida diferenciados para cada clase, [0.7,1] , [0.4,0.7] y [0.1,0.4].

Aparentemente, ambas agrupaciones, tanto en dos como en tres clusters , mejoran ligeramente el error respecto al conjunto sin agrupación, pero no son suficientemente representativos para afirmar cuál de ambos es más adecuado para el problema a resolver.

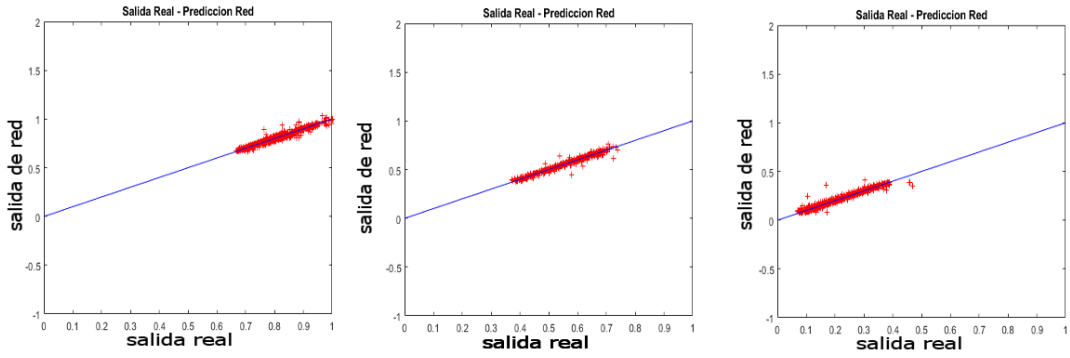

Figura 7.7: Representación salida real frente a predicción de red subconjuntos clustering 2 clases

Topología	Error entrenamiento	Error test	Desviación entrenamiento	Desviación test
Red 5-5	6.82E-03	4.32E-03	3.57E-03	2.12E-03
Red 10-10	6.39E-03	2.80E-03	3.21E-03	6.84E-04
Red 15-15	6.09E-03	2.95E-03	2.07E-03	7.22E-04

Tabla 7.3: Resultados entrenamiento online sobre conjunto completo de datos

7.1.3 Funcionamiento online

Una vez comprobada la ligera mejoría obtenida de aplicar el clustering previo en métodos de entrenamiento supervisado offline, se procede a realizar pruebas con una aproximación online, obteniendo en primer lugar los resultados del entrenamiento con redes neuronales online sobre el conjunto completo de datos, y posteriormente utilizando un clustering dinámico previo que permita mejorar los resultados previos. En esta prueba se utilizará el mismo conjunto de dos mil elementos utilizado en las pruebas offline. Aplicamos un entrenamiento online utilizando la función 'adapt' de matlab que permite entrenar una red incrementalmente, adaptando sus pesos en función de las nuevas entradas que va recibiendo dinámicamente. Dado que esta función realiza internamente el entrenamiento secuencial y obtenemos la medida de error una vez ha finalizado el entrenamiento completo, optamos por realizar una modificación en el proceso, entrenando la red en rangos de 100 elementos, de modo que entrenamos una red en diferentes etapas (e.g. rango 1-100, 1-200...1-4500) manteniendo un mismo conjunto de test de 200 elementos. De esta forma podemos observar el estado del error de entrenamiento en las diferentes etapas, no solamente cuando ya se han procesado todos los datos del experimento.

En la Tabla 7.3 se observan los resultados del error cuadrático medio y desviación tras aplicar este entrenamiento con el conjunto completo de datos sobre tres redes neuronales diferentes, la primera con dos capas de cinco neuronas ocultas, la segunda con dos capas de diez neuronas ocultas y la última con dos capas de quince neuronas ocultas.

Se observa que los resultados varían ligeramente de una arquitectura a otra, siendo la red de diez neuronas en sus capas la que posee un error de test más bajo, por lo que será la seleccionada para realizar las pruebas de entrenamiento en los subconjuntos de clustering. Para comprobar que un clustering dinámico puede mejorar los resultados obtenidos en el entrenamiento sobre el conjunto completo de datos, utilizaremos el algoritmo OGPC para separarlo en subconjuntos sobre los que aplicar las redes online independientes. En la figura 7.8 observamos la clasificación obtenida tras aplicar el algoritmo sobre el conjunto completo de datos.

Se observa como el clustering en dos clases divide el conjunto en dos mitades de similar

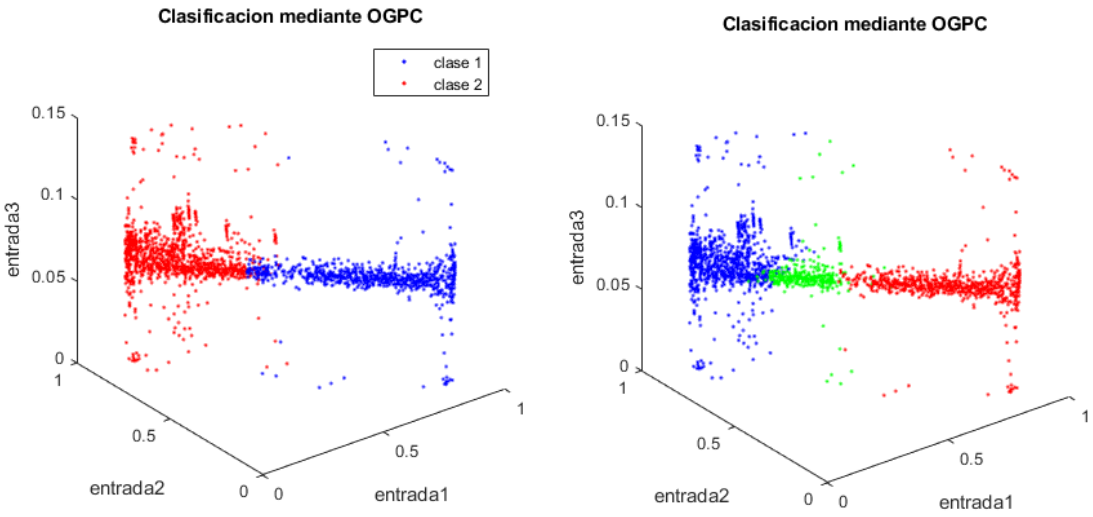

Figura 7.8: Agrupaciones mediante clustering OGPC

	No clustering	Clustering 2 clases		Clustering 3 clases		
		Clase 1	Clase 2	Clase 1	Clase 2	Clase 3
Error Entrenamiento	6.39E-03	5.58E-03	4.46E-03	3.03E-03	4.05E-03	6.26E-03
Error Test	2.80E-03	2.02E-03	1.99E-03	1.94E-03	1.82E-03	2.82E-03
Desviación Entrenamiento	3.21E-03	2.89E-03	1.91E-03	9.07E-04	1.17E-03	1.84E-03
Desviación Test	6.84E-04	6.25E-04	3.18E-04	5.40E-04	5.79E-04	3.51E-04

Tabla 7.4: Resultados Redes entrenamiento online + clustering

tamaño al igual que el algoritmo K-means en las pruebas offline. En cuanto a la división en tres clases, realiza una clasificación de la tercera clase considerablemente más pequeña que sus otros dos subconjuntos. Aplicamos el entrenamiento online sobre los conjuntos obtenidos por OGPC utilizando dos redes independientes de forma paralela para el clustering en dos clases y tres redes para el clustering de tres conjuntos. En la 7.4 representamos los resultados obtenidos tras realizar los entrenamientos mencionados para los subconjuntos divididos por OGPC, así como los resultados obtenidos anteriormente en la red entrenada con el conjunto completo de datos.

Puede observarse como el clustering en dos clases mejora considerablemente la actuación de la red neuronal tanto para los elementos de la clase 1 como los de la clase 2. De forma similar

los elementos de estas dos clases en la división en tres conjuntos del clustering obtienen un resultado parecido, mientras que la red entrenada con el subconjunto de clase 3 mantiene un error prácticamente igual al obtenido por la red entrenada con el conjunto completo de elementos.

Ante la discreta mejora del método planteado optaremos por la realización de un segundo experimento en un problema más complejo que nos permita evaluar la mejora en un nuevo conjunto de datos diferente.

7.2 Experimento 2

Como se muestra en la 7.9, el experimento utiliza una mesa de 2,40 x 1,20 metros frente a la cual se sitúa el robot Baxter. En algún lugar de la mesa hay una pelota roja y tenemos un brazo robótico que tiene que ir a por la pelota para posteriormente meterla en una caja (hay más cosas en el entorno, pero no son relevantes) . Los sensores que tenemos son la distancia (en metros) de la mano a la pelota y el ángulo de la mano a la pelota (en radianes). La única actuación que se puede hacer es decidir en qué ángulo (en radianes) se va a mover la mano, lo que hará que la mano se mueva en el ángulo dado 0,05 metros (es decir 5 centímetros).

Para este experimento se dispone de una colección de datos con las siguientes componentes de entrada:

- *distance(t)* es la distancia entre la pelota y el brazo en el instante t
- *angle(t)* es el ángulo respecto a la pelota en el instante t
- *angleact(t)* es el ángulo en el que va a desplazarse en t

Y dos componentes de salida:

- *distance(t+1)* es la distancia a la que está el robot de la pelota en el momento t+1 tras haberse movido 0,05m en el ángulo *angleact(t)*
- *angle(t+1)* es el ángulo en el que está el robot respecto a la pelota en el momento t+1 tras haberse movido.

Se han obtenido datos de la ejecución real de este experimento y se buscará aprenderlos de manera más eficiente a como se realizaba hasta este momento, con el objetivo de aportar una mejora en el desempeño final del robot.

7.2.1 Análisis inicial

Estos datos fueron obtenidos en el montaje anterior, mediante movimientos aleatorios del brazo del robot. Este conjunto de datos constituye una representación realista de lo que el modelo de mundo del MDB debería aprender en este caso.

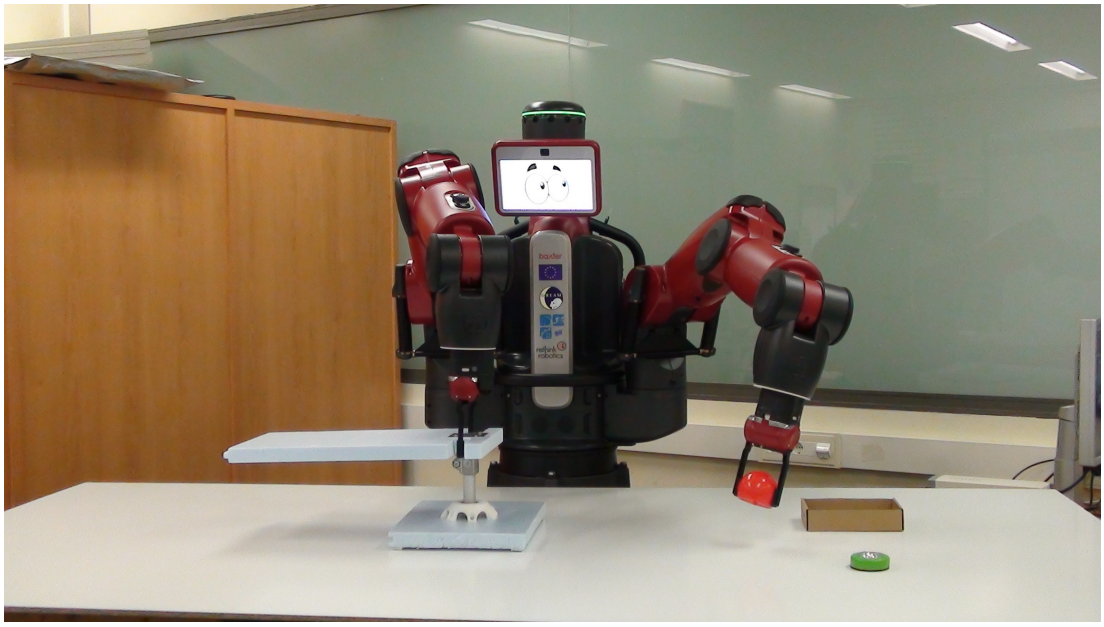

Figura 7.9: Robot empleado en el experimento con el diseño experimental empleado.

Al igual que en el caso anterior, vamos a tratar de aprender estos datos con un algoritmo on-line de entrenamiento supervisado, pero antes vamos a usar un entrenamiento off-line (descenso de gradiente) para conocer los niveles de error a los que podríamos llegar, así como qué salida es más problemática.

Se opta por realizar pruebas de entrenamiento iniciales que permitan saber si alguna de las salidas (distance t+1 y angle(t+1)) es más problemática que la otra y centrar en ella las sucesivas pruebas que realizaremos.

En primer lugar se aplica un preprocesado sobre el conjunto completo de datos que consiste en una normalización de los valores entre 0 y 1 con el objetivo de mantener los resultados en una escala estándar para su posterior análisis.

Creamos dos redes neuronales prealimentadas independientes con una arquitectura compuesta por tres capas de neuronas ocultas, con tres neuronas en la primera capa y cinco neuronas en la segunda y tercera. Utilizando en ambas el algoritmo de gradiente descendente implementado en matlab por la función 'trainsgd' , como entrada de las redes neuronales utilizamos, para la primera red, el conjunto de datos compuesto por las tres componentes de entrada (distance(t). angle(t) y angleact(t)) y la salida 1 (distance t+1), mientras que la segunda red utilizaremos las mismas componentes de entrada y cambiando la salida por angle(t+1). La distribución del conjunto de entrenamiento-test-validación será en porcentajes 70-15-15 respectivamente. En la Tabla 7.5 aparecen representados los resultados del error medio cuadrático de ambas redes para los conjuntos de Entrenamiento, Test y Validación mientras que en la Figura 7.10 se muestra la representación gráfica del valor de predicción de la red y la

Red	Error Entrenamiento	Error Test	Error Validación
Red Salida 1	0.0809	0.0721	0.0772
Red Salida 2	0.3038	0.217	0.3973

Tabla 7.5: Resultados entrenamiento redes inicial.

Topología	Error Entrenamiento	Error Test	Desviación Entrenamiento	Desviación Test
Red 3	0.0052312	0.0078949	0.0012051	0.0080843
Red 5	0.0045337	0.0058284	0.0012562	0.0040738
Red 10	0.0034126	0.0063539	0.00074064	0.005807
Red 3-3	0.0035396	0.0057108	0.00084213	0.0056315
Red 5-5	0.0026644	0.0051031	0.00096246	0.0042171
Red 10-10	0.0012109	0.0070562	0.00070123	0.0092459

Tabla 7.6: Resultados entrenamiento redes sobre conjunto de elementos completo

salida real del vector para cada muestra del conjunto de entrenamiento.

Podemos deducir de estos resultados que debido al error superior, y la distribución de los resultados en la red de la salida 2, que este conjunto es más problemático para su correcto entrenamiento con redes neuronales, por lo que centraremos las siguientes pruebas en el conjunto de datos formado por las tres entradas comunes y la salida angle(t+1). Así podremos analizar si el método propuesto proporciona una ventaja práctica real en un caso complejo.

7.2.2 Funcionamiento offline

La primera prueba a ejecutar consiste en una serie de entrenamientos sobre redes de neuronas artificiales prealimentadas, con el algoritmo de gradiente descendiente estocástico y diferentes arquitecturas, con el objetivo de obtener la distribución de neuronas más adecuada para el conjunto de datos seleccionado en el análisis inicial.

De estos resultados podemos deducir que la red con arquitectura 5-5 aporta los mejores resultados de error cuadrático medio para el conjunto de test, por lo que la utilizaremos en las pruebas sucesivas e intentaremos alcanzar dicho error en las pruebas online. Además, podemos ver gráficamente en la figura 7.11 el comportamiento del error cuadrático medio para ambos conjuntos, entrenamiento y test, a lo largo de los 1000 epochs de entrenamiento de la red 5-5 seleccionada.

También se muestra la representación de las salida real esperada frente a la predicción realizada por la red en la Figura 7.12.

De cara a comprobar preliminarmente si el método puede ser útil en este experimento,

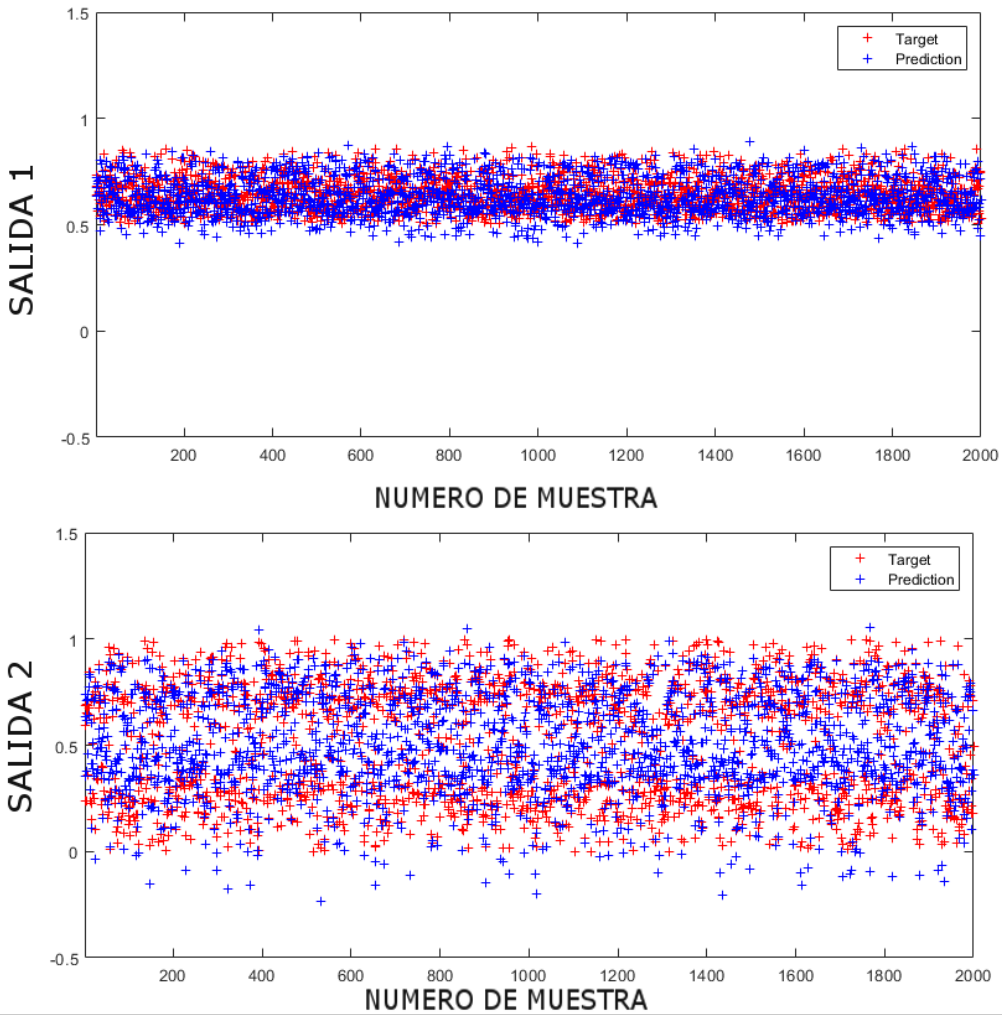

Figura 7.10: Representación Predicción-Salida Real entrenamiento redes salida 1 y salida 2

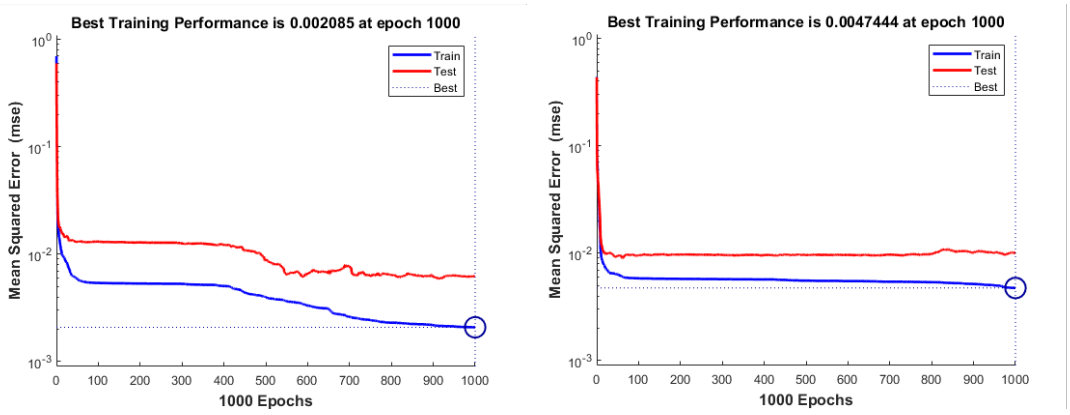

Figura 7.11: Representación gráfica rendimiento del entrenamiento de la red 5-5

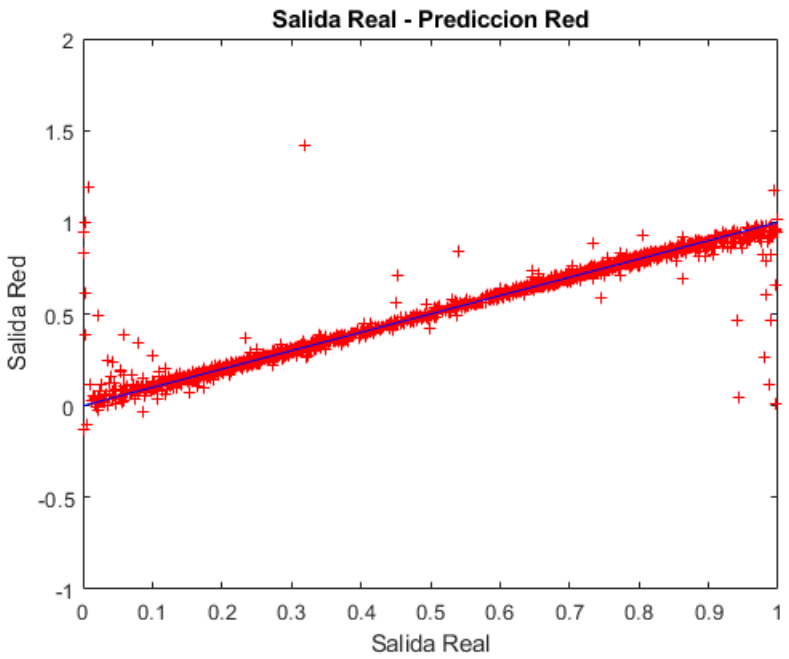

Figura 7.12: Representación gráfica salida real frente a salida de la red

se ha realizado una prueba de clustering off-line sobre este aprendizaje. Optamos por la utilización de un algoritmo de clustering no supervisado clásico, el Kmeans, implementado en matlab mediante la función 'kmeans'. Clasificando el conjunto completo de datos en dos y tres clases, obtenemos los agrupamientos mostrados en la Figura 7.13

A partir de la clasificación obtenida mediante K-means, realizamos una subdivisión en conjuntos de datos en función de la clase designada, por un lado dos conjuntos para la clase 1 y clase 2 de la ejecución en dos clusters de kmeans, se observa que se divide el conjunto completo en dos subconjuntos de tamaño similar. Por otro lado, obtenemos tres grupos para la clase 1, clase 2 y clase 3 en la ejecución en tres clusters donde se obtiene un conjunto similar para la clase dos al obtenido en el primer clustering, y es el conjunto de la clase 1 el que se subdivide en este caso en clase 1 y clase 3. Con estos conjuntos crearemos paralelamente redes neuronales con dos capas de cinco neuronas ocultas cada una y mismo algoritmo de gradiente descendiente conjugado utilizado anteriormente sobre cada conjunto de datos. Una vez realizados los entrenamientos observamos la comparativa entre la salida real y la salida predicha por la red para los subconjuntos del clustering de dos clases dibujados en la figura 7.14

Se observa que la gráfica muestra una clara división entre ambas agrupaciones, la primera red clasifica los elementos de clase 1, en la que se agrupan mayoritariamente los vectores con salidas reales comprendidas entre los valores [0,0.5] mientras que la red que clasifica los

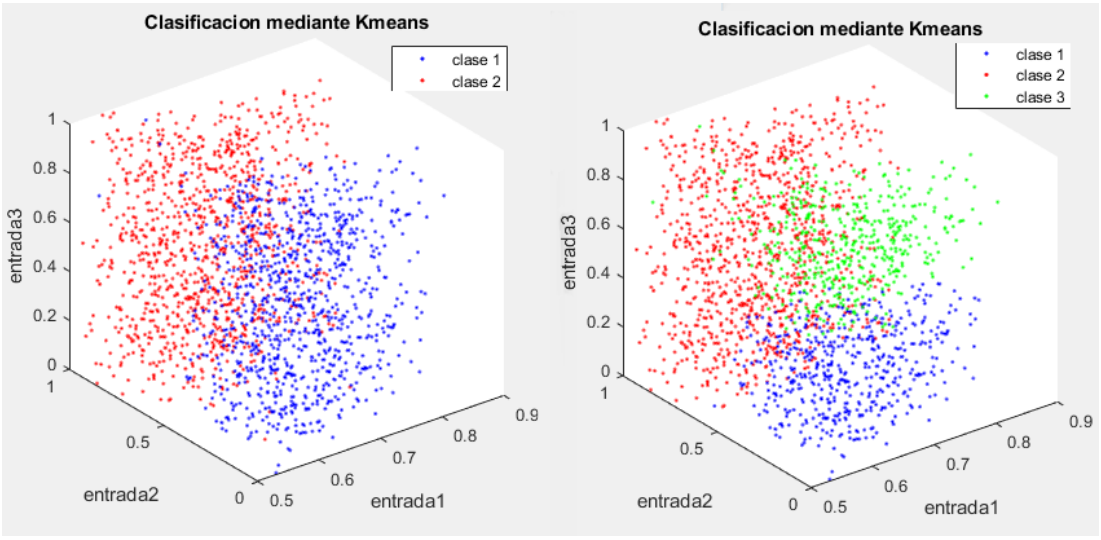

Figura 7.13: Agrupaciones del conjunto de datos mediante K-means

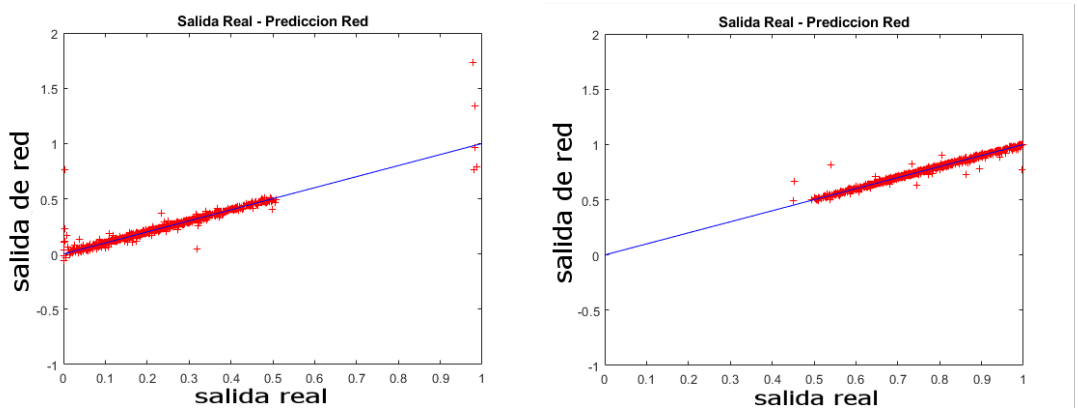

Figura 7.14: Salida real frente a predicción de red clustering en 2 clases

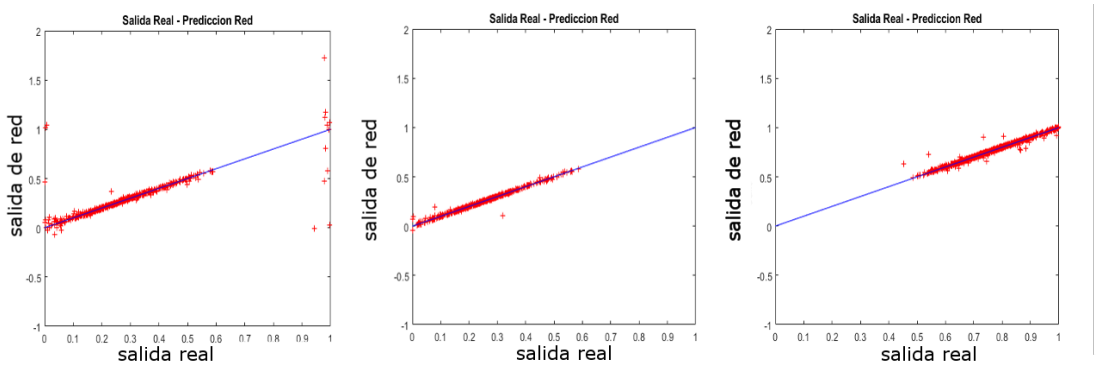

Figura 7.15: Salida real frente a predicción de red clustering en 3 clases

	No clustering	Clustering 2 clases		Clustering 3 clases		
		Clase 1	Clase 2	Clase 1	Clase 2	Clase 3
Error Entrenamiento	2.664E-03	6.589E-04	1.725E-04	8.683E-04	1.4901E-04	9.682E-05
Error Test	5.103E-03	1.331E-03	2.401E-04	1.044E-02	1.9705E-04	1.946E-04
Desviación Entrenamiento	9.624E-04	1.194E-03	7.022E-05	9.581E-04	8.463E-05	5.431E-05
Desviación Test	4.217E-03	2.038E-03	2.461E-04	1.317E-02	1.233E-04	2.963E-04

Tabla 7.7: Resultados entrenamientos redes sobre subconjuntos de clustering

elementos de la clase 2, agrupa aquellos elementos cuya salida real abarca los valores entre [0.5, 1]. En la Figura 7.15 se muestra la misma representación para los tres entrenamientos de los subconjuntos del clustering para tres clases donde, a diferencia de los conjuntos anteriores, la clase 1 y la clase 2 comparten elementos comprendidos en el rango [0, 0.5]

Para analizar cuál de ambos clustering obtiene unos mejores resultados al aplicar las redes neuronales utilizamos sus métricas de error medio cuadrático para conjunto de entrenamiento y test así como sus desviaciones medias (Tabla 7.7)

En esta tabla se resumen de los experimentos realizados. Comprobamos cómo la predicción de las redes sobre los subconjuntos obtenidos mediante kmeans mejora considerablemente respecto a los entrenamientos realizados directamente sobre el conjunto completo de datos tanto en el error de entrenamiento como en el error de test. En cuanto a las diferencias entre los clusterings de dos y tres clases se observa como la clase 1 del clustering de tres clases posee un error de test mucho mayor que el resto de subconjuntos, lo que podría indicar que esta agrupación es prescindible y sería suficiente con una clasificación en dos clases.

Topología	Error Entrenamiento	Error Test	Desviación Entrenamiento	Desviación Test
Red 5-5	8.66E-03	8.19E-03	1.58E-03	1.47E-03
Red 10-10	9.55E-03	7.93E-03	1.44E-03	1.73E-03
Red 15-15	9.29E-03	7.89E-03	1.37E-03	2.11E-03

Tabla 7.8: Resultados redes sobre conjunto completo de datos

7.2.3 Funcionamiento online

Para realizar una aproximación más fiel al comportamiento que deberá tener el sistema al integrarse con MDB, se realizarán una serie de pruebas utilizando entrenamientos online de redes neuronales, primero sobre un conjunto de datos directamente y posteriormente, aplicando un proceso de clasificación previo mediante clustering dinámico. En esta prueba se utilizará un conjunto formado por cinco mil elementos formados por las mismas componentes tratadas en las pruebas anteriores.

De forma análoga al Experimento 1, utilizaremos el mismo procedimiento para el entrenamiento online, con la función 'adapt' y su aplicación por rangos progresivos.

En la Tabla 7.8 se observan los resultados del error cuadrático medio y desviación tras aplicar este entrenamiento sobre tres redes neuronales diferentes, la primera con dos capas de cinco neuronas ocultas, la segunda con dos capas de diez neuronas ocultas y la última con dos capas de quince neuronas ocultas.

Podemos observar como estos resultados son peores con los errores obtenidos en las pruebas offline, algo esperable debido a la propia naturaleza de ambos procesos. En el entrenamiento online no contamos con todos los datos desde un primer momento, si no que debemos ir adaptando el modelo a los datos recibidos en cada iteración, causando unos errores mucho mayores, especialmente en las primeras ejecuciones del proceso. Se observa que los resultados varían muy ligeramente de una arquitectura a otra, siendo la red de quince neuronas en sus capas la que posee un error de test más bajo, por lo que será la seleccionada para realizar las pruebas de entrenamiento en los subconjuntos de clustering. Posteriormente se realiza un clustering dinámico con el objetivo de mejorar los resultados obtenidos aplicando el entrenamiento sobre el conjunto completo de datos. Como implementación utilizaremos el algoritmo OGPC. Aplicaremos el clustering sobre el conjunto completo de datos en dos pruebas paralelas, dividiendo el conjunto en dos clases en la primera y en tres en la segunda, se muestra la distribución de los datos tras esta clasificación utilizando las tres componentes de entrada para representarlos. En esta Figura 7.16 observamos la clasificación obtenida tras aplicar el algoritmo sobre el conjunto completo de datos.

Se observa como el clustering en dos clases divide claramente el conjunto en dos mitades

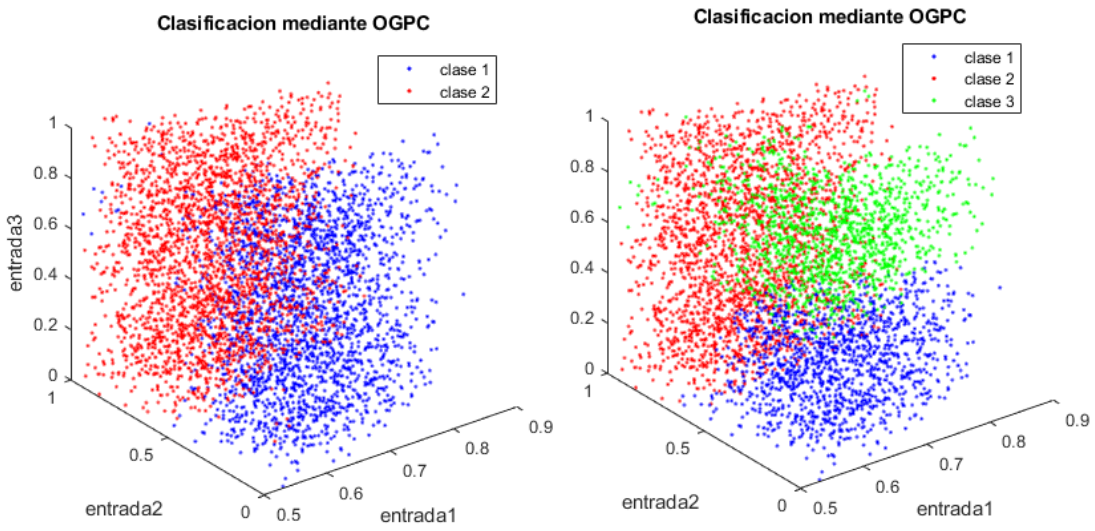

Figura 7.16: Agrupaciones mediante clustering OGPC

diferenciadas, similar a la clasificación obtenida anteriormente con el algoritmo k-means en las pruebas offline (Figura 7.13). En cuanto a la división en tres clases, realiza una clasificación del conjunto designado como "clase 2" muy similar al clustering en dos clases y divide el resto de datos entre las dos clases restantes. Para simular el proceso que realizaría el robot en un entorno real, realizamos una ejecución secuencial de OGPC y vamos acumulando los elementos según su clase designada. Cuando recibimos cien elementos de una misma clase reentrenamos la red online de quince neuronas ocultas en cada una de sus dos capas con estos nuevos datos, actualizando así el modelo para adaptarlo a los nuevos datos obtenidos. Aplicamos el entrenamiento online sobre los conjuntos obtenidos por OGPC utilizando dos redes independientes de forma paralela para el clustering en dos clases y tres redes para el clustering de tres conjuntos. En la Tabla 7.9 representamos los resultados obtenidos tras realizar los entrenamientos mencionados para los subconjuntos realizados por OGPC, así como los resultados obtenidos anteriormente en la red entrenada con el conjunto completo de datos.

Puede observarse como, tanto en el clustering en dos clases como el de tres muestra un error menor en la clase dos respecto al entrenamiento sobre el conjunto completo sin clustering, mientras que la clase uno empeora ligeramente, lo cual nos indica que es este conjunto el que posee una mayor dificultad para nuestro algoritmo. En este segundo experimento se ha obtenido una mejora más relevante en la utilización del desarrollo planteado que en el primer experimento realizado, lo cual nos lleva a la conclusión de que este método puede servir de gran ayuda en el aprendizaje de modelos en tiempo real.

	No clustering	Clustering 2 clases		Clustering 3 clases		
		Clase 1	Clase 2	Clase 1	Clase 2	Clase 3
Error Entrenamiento	1.14E-02	1.30E-02	7.32E-03	2.74E-02	2.73E-03	1.11E-02
Error Test	8.07E-03	9.50E-03	5.57E-03	1.63E-02	1.55E-03	8.07E-03
Desviación Entrenamiento	2.06E-03	4.45E-03	4.39E-03	4.95E-03	7.23E-04	2.06E-03
Desviación Test	3.58E-03	2.68E-03	1.70E-03	8.17E-03	2.89E-04	3.58E-03

Tabla 7.9: Resultados Redes entrenamiento online + clustering

Capítulo 8

Conclusiones

8.0.1 Futuros Desarrollos

Los experimentos mostrados en este proyecto han sido desarrollados con el objetivo de demostrar la posibilidad de mejorar los resultados y facilitar el aprendizaje de estos por parte de un sistema autónomo utilizando técnicas de clasificación no supervisada. Aún cuando esto queda demostrado en los experimentos aportados. También sería interesante el planteamiento de otros algoritmos de clustering online diferentes a los tratados en este proyecto con el objetivo de evaluar sus características respecto a los ya planteados, mayor eficacia, velocidad o menos consumo de recursos, características muy relevantes en el sistema final de aplicación.

8.0.2 Lecciones aprendidas

A lo largo de la realización del proyecto se han afianzado conceptos estudiados durante el grado y obtenido una serie de nuevos conocimientos. Técnicamente se ha profundizado considerablemente en la programación con Matlab, fundamentalmente debido a la escasa formación obtenida durante el grado, fue necesario realizar una amplia formación en el software apoyado en la documentación suministrada por el programa y por los tutores del proyecto. Las necesidades del proyecto han llevado a un amplio conocimiento de muchas de las funciones de Matlab, especialmente a las pertenecientes a su 'Deep Learning Toolbox'. Aspectos aprendidos en el grado como las buenas prácticas de programación han facilitado considerablemente la realización de un código modular, reutilizable y escalable que han permitido un desarrollo más ágil de los desarrollos realizados durante el proyecto. En definitiva realizar este proyecto me ha permitido comprender como funciona un trabajo de investigación y todas las dificultades inherentes de estos, las cuales me han permitido descubrir claras carencias profesionales debo suplir a lo largo de mi carrera profesional.

8.0.3 Conclusiones

La principal ventaja de esta aproximación se obtiene cuando en caso de que el espacio de búsqueda se pueda agrupar, lo haríamos automáticamente, de modo que tenemos modelos específicos para zonas del entorno. Esto hace que el aprendizaje de nuevos modelos como combinación de modelos básicos sea mucho más sencilla. Las técnicas empleadas en los experimentos analizados durante este proyecto han aportado resultados satisfactorios que confirman la premisa inicial de obtener una mejora en la capacidad de clasificación del sistema, a pesar de ser una mejora discreta en estos casos.

Gracias al bajo coste computacional de estas técnicas no supervisadas, sería rentable añadir el sistema de clasificación previa a los datos sensoriales obtenidos en lugar de realizar directamente un aprendizaje supervisado. Puedo confirmar la posible viabilidad del método, quedando trabajo por hacer para poder ser aplicado online en un sistema real, principalmente en lo referente a pruebas para seleccionar el número de clusters, en qué casos concretos compensaría o no esta distribución previa, etc

Apéndices

Lista de acrónimos

MDB: *Multilevel Darwinist Brain*

CDR: *Cognitive Development Robotics*

ANN: *Artificial Neural Network*

POMDP: *Partially-observable Markov decision process*

SGD: *Stochastic Gradient Descent*

STM: *Short Term Memory*

LTM: *Long Term Memory*

par AP: *par Acción-Percepción*

Bibliografía

[1] G. Lakemeyer and H. Levesque, "Cognitive robotics," *Foundations of Artificial Intelligence*, vol. 3, no. 23, pp. 869–866, 12 2008.

[2] Y. K. H. I. T. I. Y. Y. M. O. C. Y. Minoru Asada, Koh Hosoda, "Cognitive developmental robotics: A survey," *IEEE Transactions on autonomous mental development*, vol. 1, pp. 12–34, 04 2009.

[3] A. F. D. S. Francisco Bellas, Richard J Duro, "Multilevel darwinist brain (mdb): Artificial evolution in a cognitive architecture for real robots," *IEEE Transactions on autonomous mental development*, vol. 2, pp. 340–354, 10 2010.

[4] "Five algorithms to train a neural network," https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network, 2015.

[5] J. B. Hamrick, "Analogues of mental simulation and imagination in deep learning," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 8–16, 10 2019.

[6] J. Kober, J. Andrew Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, pp. 1238–1274, 09 2013.

[7] B. HH., "Clustering methods: A history of k-means algorithms," 2007.

[8] S. R. Amr Abdullatif, Francesco Masulli, "Cognitive developmental robotics: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, p. e1258, 07 2018.

[9] J. L. Doyen Sahoo and P. Zhao, "Online learning: A comprehensive survey," *SMU Technical Report*, no. 1, pp. 1–100, 10 2018.

[10] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv*, pp. 1–14, 1 2016.

[11] "Ogpc implementation," https://es.mathworks.com/matlabcentral/fileexchange/64318-onlinegradedpossibi-listicclustering.

[12] "Trello," https://trello.com/.

[13] B. F. et al., "The robobo project: Bringing educational robotics closer to real-world applications," *Advances in Intelligent Systems and Computing*, vol. 630, 2018.