eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniaritza Goi Eskola Teknikoa
Escuela Técnica Superior de Ingeniería
Bilbao

Trabajo Fin de Máster

# Resolución en tiempo real del problema cinemático directo del robot paralelo 3PRS mediante redes neuronales RBF

---

Director: Eloy Irigoyen
Autor: Alejandro Solozabal

2016-2017

# Índice

# Resumen

Castellano:

En este trabajo se presenta una solución en tiempo real al Problema Cinemático Directo (DKP) del robot paralelo 3-PRS haciendo uso de una Red Neuronal Artificial (RNA). Este problema consiste en determinar la posición del efector en función de los actuadores del robot. Esto implica resolver un sistema de ecuaciones no lineales sin solución analítica. Generalmente, se hace uso de métodos numéricos que obtienen un gran grado de precisión pero que implican un coste computacional muy alto. En este trabajo se hará uso de las redes neuronales de Función de Base Radial (RBF) para resolver el DKP disminuyendo el tiempo de ejecución y reduciendo el coste computacional.

English:

The work here proposed present a real-time solution for the Forward Kinematics Problem (FKP) of the parallel robot 3-PRS using an Artificial Neural Network (ANN). The problem of determining the position of the effector depending on the position of the robot actuators implies the resolution of a non linear equation system without an analytic solution. Generally, is solved using numerical methods which obtain high precision but at high computational cost. For this purpose Radial Based Functions (RBF) neural networks will be used in order to solve the FKP reducing the computational time an therefore the computational cost.

Euskera:

Gradu amaierako lan honetan 3-PRS errobotaren problema zinematiko zuzenari (DKP), denbora errealean irtenbide bat bilatzen diogu, neurona-sare artifizialaren (ANN) bitartez. Arazo honen mamia errobotaren eragingailuarekin eralazio zuzena duen efektorearen posizioa zehaztean datza. Honetarako erantzun analitikorik gabeko ekuazio ez-lineal sistema bat ebatzi behar da. Normalean zenbaki metodo batzuk erabiltzen dira horretarako, zehazpen altuko erantzunak lortuz, baina gastu konputazional oso altu baten truke. Lan honetan oinarri erradialeko funtzioen neuronal-sareak erabiliko dira errobotaren arazo zinematiko zuzena ebazteko, exekuzio denbora eta kostu konputazionala murriztuz.

# Índice de figuras

# Índice de tablas

# Lista de acrónimos

**RNA** Red Neuronal Artificial

**DKP** Problema Cinemático Directo

**MSE** Error Cuadrático Medio

**MLP** Multilayer Perceptron

**RBF** Radial Basis Function

**MJS** MATLAB Job Scheduler

# 1. Memoria

Esta memoria contiene un primer apartado de introducción al trabajo realizado. A continuación, se enumerarán los objetivo principales y secundarios propuestos. Seguidamente, se expondrá el beneficio que aporta y el estado del arte. Por último se expondrá una breve introducción teórica a los dos campos que abarca este trabajo: los robots paralelos y las RNA.

## 1.1. Introducción

Hoy en día, la industria tiende a requerir procesos de producción más eficientes y automatizados, el robot paralelo se introdujo con este objetivo. Está formado por una estructura, que al contrario que un robot serie, contiene múltiples cadenas cinemáticas. Esta disposición permite una mayor respuesta dinámica, cualidad esencial para desarrollar las actividades requeridas en la industria como por ejemplo el *pick-and-place*.

El problema que presenta este tipo de robot es la dificultad de determinar la posición del efector en función de los actuadores, a esto se le denomina Problema Cinemático Directo (DKP). En los robot serie debido a su construcción con una sola cadena cinemática este problema se resuelve de manera directa analíticamente. Por el contrario, en los robots paralelos para resolver el DKP se debe resolver un sistema de ecuaciones no lineales únicas para cada tipo de estructura y que no tienen solución analítica por lo que se hace uso de métodos numéricos para resolverlas. El más utilizado es el método iterativo de Newton-Rapson. Este método iterativo requiere un tiempo y coste computacional alto.

Las redes neuronales artificiales (RNA) empiezan a ser utilizadas como solución a este problema. Estas estructuras, que emulan de manera artificial un sistema neuronal, permite llegar a una solución aproximada con mayor rapidez y con menos coste computacional. Las estructuras de redes neuronales más utilizadas son las MLP y las RBF, pero también aparecen otro tipo de estructuras como las *Wavelet neural networks*, soluciones híbridas o también, agrupaciones de redes neuronales.

En este trabajo se presenta una solución del DKP para el robot 3PRS. Este robot posee 3 cadenas cinemáticas idénticas que permiten posicionar el efector con 3 grados de libertad: un desplazamiento vertical y dos rotaciones. La red neuronal utilizada es la red RBF, este tipo de red neuronal hace uso de funciones de activación de base radial aportando a la red de un funcionamiento diferente al tipo de red MLP, mayormente utilizada.

## 1.2. Objetivo y alcance del trabajo

El objetivo principal de este trabajo es la entrenar una red RBF que resuelva la el DKP del robot paralelo 3PRS cumpliendo los objetivos de error y mejorando el tiempo de ejecución del método iterativo Newton-Rapson, que es de 200 ms en la plataforma GEME 2000.

Los objetivos de error que se proponen son los siguientes:

- Error máximo en el cálculo de la posición en Z del efector menor que $10^{-4}$ m.
- Error máximo en el cálculo de los ángulos del efector ($\theta_x$ y $\theta_y$) menor que 0.1 grados.

También se tiene como objetivo el realizar una comparación entre las redes MLP y las redes utilizadas en este trabajo, las RBF.

Los objetivos secundarios o específicos que se han de completar para realizar el objetivo principal del trabajo son los siguientes:

- Realizar una introducción teórica a las RNA.
- Estudiar en profundidad el tipo de red RBF.
- Realizar el entrenamiento de la red.
- Implementar la red en LabView.
- Analizar los resultados en la plataforma de tiempo real.

## 1.3. Beneficios que aporta el trabajo

El beneficio que aporta este trabajo es el de comprobar la validez de las redes RBF para resolver problemas cinemáticos como el propuesto en este trabajo. La sustitución de algoritmos iterativos por redes neuronales permite una mayor rapidez en el computo de la solución. Esto posibilita que el control sobre el robot se realice en un intervalo menor y así mejorar las características dinámicas del actuador. También se realiza una comparación teórica y práctica entre el tipo de redes MLP y RBF permitiendo determinar las características y deficiencias de cada una de ellas.

## 1.4. Estado del arte

Desde la inserción del robot paralelo en la industria, la resolución del DKP ha sido el único laste de este tipo de actuadores. Los métodos numéricos utilizados para la resolución de este problema exigían tiempos de control elevados debido a su fuerte carga computacional. Para mejorar la respuesta dinámica de este tipo de actuadores era necesario encontrar sustitutos a estos métodos de cálculo tradicionales.

A partir de 2005 empiezan a surgir numerosos estudios dirigidos en aplicar redes neuronales en la resolución de problemas cinemáticos de robots paralelos. Para diversos tipos de robots paralelos se plantean soluciones basadas, sobre todo, en redes MLP con resultados positivos [6, 12, 4]. Mas adelante, se realizan estudios para ver la validez de otro tipos de redes como las RBF [24, 23], las Wavenet [16] o agrupaciones de redes [9].

Siguiendo esta linea, en este trabajo se platea desarrollar y analizar el comportamiento de las redes RBF para resolver el DKP del robot paralelo 3PRS, observando no solo los objetivos de error plateados si no también su carga computacional.

## 1.5. Introducción teórica

En este apartado se realiza una breve introducción a los dos campos que abarca este trabajo, que son el robot paralelo y las RNA. Se explicará lo que es un robot paralelo, las ventajas y desventajas que tienen sobre un robot serie, su problema cinemático. Por ultimo se describirá específicamente la estructura del robot 3PRS. En lo relativo a las RNA, expondremos lo que es una RNA, tipos de redes existentes, topologías y métodos de entrenamiento.

### 1.5.1. Robot Paralelo

Hoy en día los robot paralelo son ampliamente utilizados en la industria, tienen características que les distingue de los robot serie convencionales y aportan un grado más de eficiencia en los procesos de fabricación.

Un robot paralelo consiste en una plataforma móvil, donde se ubica el efector, conectada a una base a través de una serie de brazos dispuestos en paralelo, a las cuales se les llama cadenas cinemáticas. Al contrario, un robot serie consiste en una serie de brazos unidos en serie desde la base al efector. Gracias a la estructura del robot paralelo pueden

realizar tareas en un espacio de trabajo mas reducido y con mayor agilidad.

En las siguiente imágenes se muestra un ejemplo de cada tipo de robot manipulador:


(a) Robot Paralelo (b) Robot Serie

Figura 1: Tipos de Robot manipulador

**Ventajas y Desventajas de robot paralelo**

A continuación se enumeran las principales ventajas que presenta el robot paralelo, citadas por Salgado en [19]:

- Posee una alta relación carga/peso.
- Alta rigidez estructural.
- Mayor precisión, los errores en la posición de las articulaciones inciden con el mismo grado al error total, no se agregan articulación a articulación como en el caso del robot serie.
- Mayor respuesta dinámica, este tipo de manipulador alcanza altas velocidades y aceleraciones.

Las desventajas que presenta son las siguientes:

- Espacio de trabajo reducido.
- Calibración compleja.
- Cinemática compleja, requiere un alto tiempo de computación.

### Configuraciones de robot paralelo

Debido al gran número de variables que existen en la estructura de un robot paralelo no es posible hacer una clasificación clara. Se pueden agrupar los distintos robot paralelo existentes según las siguientes características:

- Espacio de trabajo en un solo plano o volumen de 3 dimensiones
- Número de grados de libertad: hasta 6 (3 translaciones y 3 rotaciones)
- Configuración de las cadenas cinemáticas (PRS,RPS,etc..)

Información recogida de [21]

### Problema dinámico

En este tipo de robot no es posible resolver analíticamente el Problema Cinemático Directo o *DKP*, esto es, no se puede calcular de manera analítica la posición del efector en función de la posición de los actuadores. Debido a esto, se utilizan métodos numéricos iterativos para resolver la posición del efector en cada intervalo de control, requiriendo costes de computación elevados y elevando el intervalo de control más de lo deseado. [25]

### Robot Paralelo 3PRS

El robot paralelo 3PRS posee 3 cadenas cinemáticas idénticas compuestas cada una por una junta prismática P, una junta de revolución R y una junta esférica S. Los actuadores se encuentran en la junta prismática. Este tipo de robot paralelo provee al sistema de 3 grados de libertad, permitiendo un desplazamiento en Z y dos rotaciones al efector.

En la siguiente figura se muestra un ejemplo de la estructura del robot 3PRS:

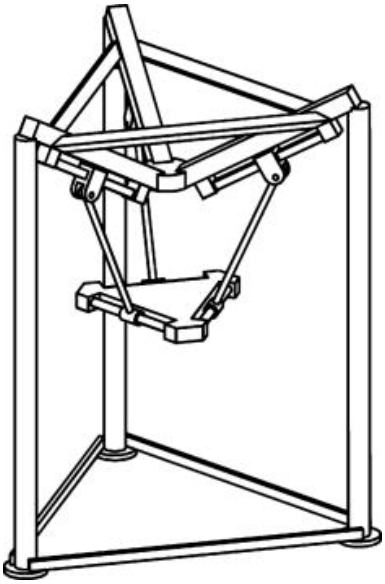


Figura 2: Robot paralelo 3PRS

### 1.5.2. Redes Neuronales Artificiales

Las redes neuronales artificiales son estructuras lógicas que emula la manera de procesar del cerebro. La parte lógica del cerebro se compone de neuronas conectadas entre si por las que circulan los impulsos eléctricos desde las entradas del sistema, neuronas receptoras, o entradas realimentadas, los recuerdos o la memoria a corto plazo, y las salidas, neuronas motoras o memoria. Las conexiones entre las neuronas se realizan por medio del axón, donde el impulso que se propaga de una neurona a otra a través de una conexión sináptica química.

Una RNA es una aproximación artificial de este sistema biológico. Está formado por los mismos componentes y mismas características:

- Neuronas de entrada. Están situadas en la primera capa de la red neuronal. Estas neuronas son las encargadas de introducir los valores de entrada a la red. Puedes ser también entradas realimentadas cuyo valor de entrada depende salidas previas de la red neuronal o de neuronas de capas intermedias.
- Conexiones entre neuronas. Conecta las neuronas de cada capa con las neuronas de la siguiente capa. En esta cada unión se multiplica el valor del impulso por un factor o peso.
- Neuronas. En cada una de ellas se realizan dos tareas: agrupa los valores de entrada ponderados y definir la salida a través de una de una función de activación. Estas neuronas se organizan en capas conectadas de manera sucesiva. Estas capas de denominan capas ocultas o *hidden layers*.

- Neuronas de salida. Se sitúan en la ultima capa de la red neuronal. Las salidas de estas neuronas son las salidas de la red neuronal.

### Unidad neuronal

Cada neurona tiene la tarea de agrupar los impulsos entrantes a ella, ponderarlos por un factor o peso, agregarle un *offset* o *bias* y por último modificar el valor resultante con un función de activación.

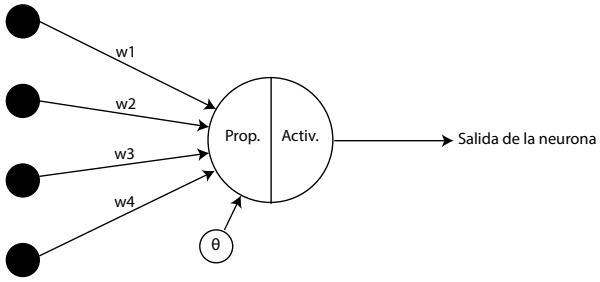



Figura 3: Representación de una neurona

La regla de propagación más utilizada consiste en sumar cada entrada ponderada por un peso y añadirle el *offset* de la neurona. La ecuación que define este tipo de propagación es la siguiente:

$$y_k(t) = \sum_j w_{jk}(t)x_j(t) + \theta_k(t)$$

Siendo $y_k$ la salida de la regla de propagación, $w_{jk}$ los pesos de las conexiones entre las neuronas $j$ y $k$, $x_j$ el valor de las entradas y $\theta_k$ el offset.

### Función de activación

Una vez agregadas las entradas de la neurona se modifica el valor resultante aplicándole una función de activación. Esta función de activación puede ser de 3 tipos:

- Función signo. Utilizada para solo obtener dos estados en la salida de la neurona. Se utiliza este tipo de función de activación en la ultima capa de redes que tienen como salida una señal binaria.

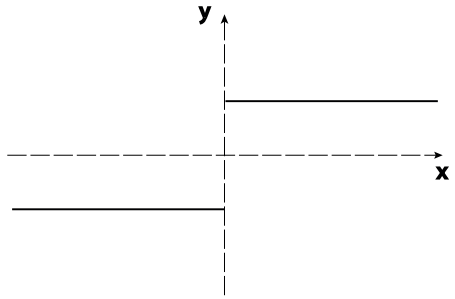
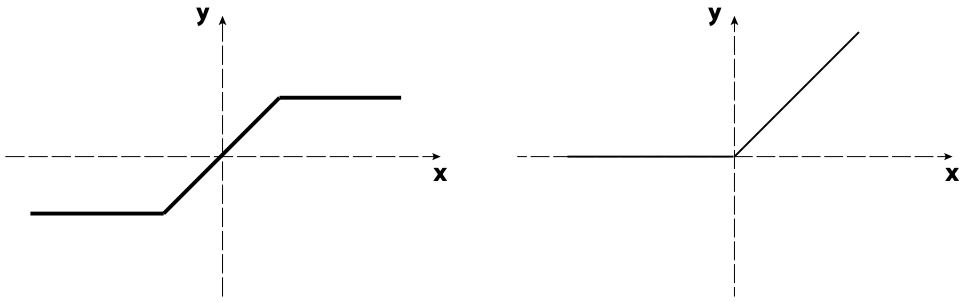



Figura 4: Función signo

- Lineal. La salida de la neurona es linealmente proporcional a las entradas. Se utiliza para resolver problemas que presentan un comportamiento de $1^{er}$ orden.




(a) Función lineal (b) Función Relu

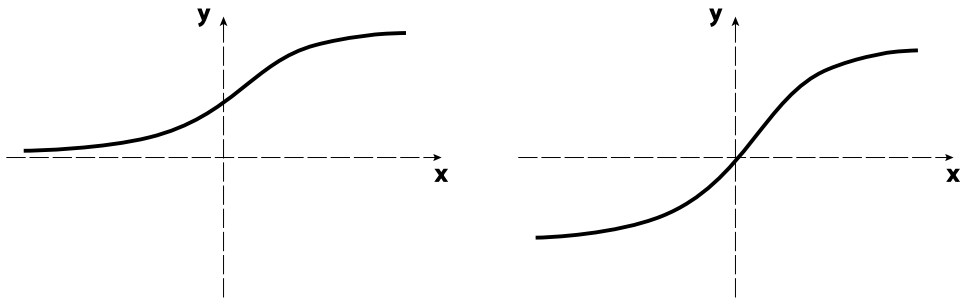
Figura 5: Funciones de activación lineales

- No lineal. Utilizada para resolver problemas que presentan un comportamiento no lineal. Las funciones mas utilizadas son la función sigmoide y tangente hiperbólica.

  Sus funciones son las siguientes:

$$p(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$




(a) Función sigmoide (b) Función tangente hiperbólica

Figura 6: Funciones de activación no lineales

- Funciones de base radial. Son utilizadas en las redes neuronales de base radial. La función mas utilizada es la función de distribución gausiana.

  Está formulada de la siguiente manera:

$$y(r_h) = e^{(-r_h^2/\sigma_h^2)}$$

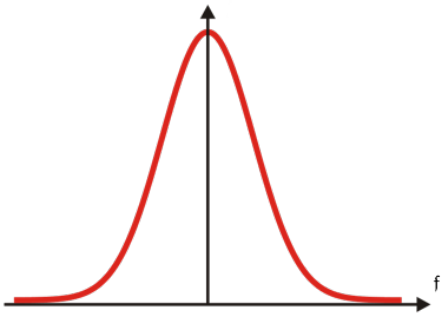



Figura 7: Función gausiana

**Topologías**

De la diversidad de tipos de redes neuronales que existen en ingeniería de control los dos grandes grupos de redes que se utilizan son: las redes *Feed-Fordward Network* y *Recurrent Network*.

En las redes *Feed-Fordward Network* los valores fluyen desde la entrada hasta la salida de la red sin que haya ningún tipo de realimentación. Las aplicaciónes más directas de este tipo de topología es la clasificación y regresión. El ejemplo más destacado de este tipo de redes es la *Multilayer Perceptron*. Este tipo de redes poseen una estructura como la representada en la siguiente imagen:

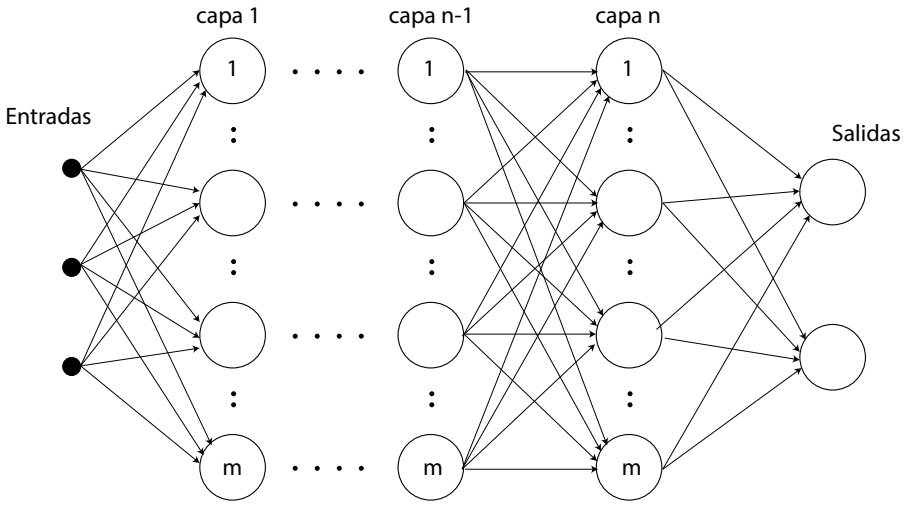



Figura 8: *Feed-Fordward Network*

En las redes redes recurrentes o *RNN* existen conexiones de neuronas con capas previas de la red. Esta realimentación permite a la red obtener comportamientos mas complejos sin aumentar el tamaño de la misma. Son utilizadas para resolver problemas con un comportamiento dinámico en el tiempo. Las dos redes recursivas más importantes son la red Jordan y la red Elman.

En la red Jordan la realimentación se realiza desde la capa de salida de la red. La red posee entradas que tiene por valor salidas anteriores de la red. El esquema de este tipo de red se representa en la siguiente imagen:

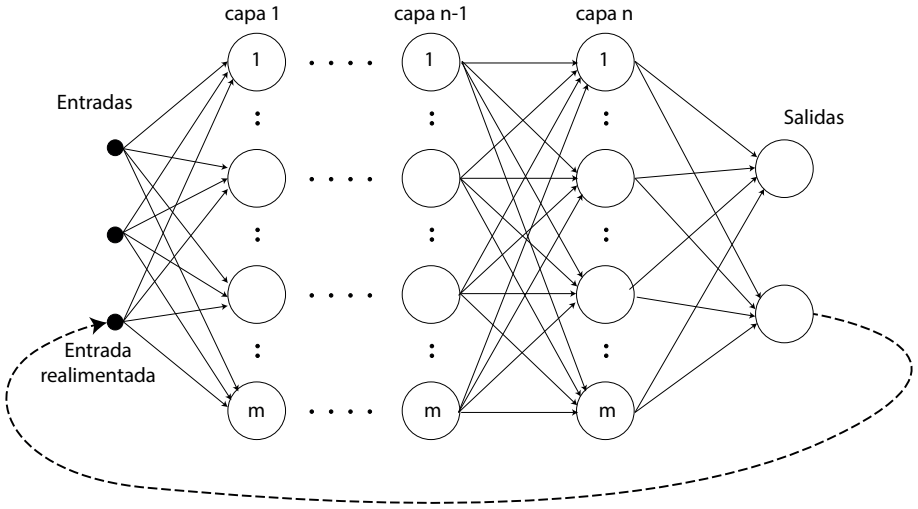



Figura 9: *Jordan Network*

En las redes del tipo Elman la realimentación se realiza desde una capa oculta. El

esquema de este tipo de red se representa en la siguiente imagen:

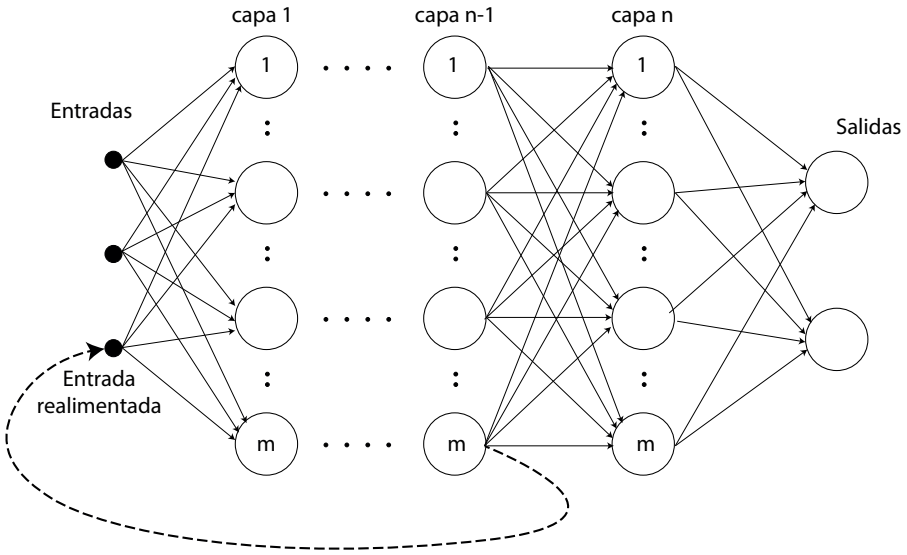



Figura 10: *Elman Network*

### Métodos de entrenamiento

En una red con una estructura ya definida los parámetros que se pueden modificar son los pesos de cada conexión y los *offsets* de las neuronas. En un principio estos parámetros se inicializan con un valor aleatorio. Mediante un método de aprendizaje variaremos estos parámetros para que la salida de la red sea lo mas próxima a la deseada. Esto es, buscar en un espacio multidimensional el mínimo de la una función error o coste de manera iterativa. Existen dos grandes métodos de entrenamiento: supervisados, no supervisados:

- Entrenamiento supervisado o asociativo. Consiste en realizar el entrenamiento de la red a partir de ejemplos, ante vectores de entrada se especifica los vectores de salida deseado. A partir de estos ejemplos la red puede llegar a aprender la solución a problema planteado. El ejemplo mas destacado de este tipo de entrenamiento es el algoritmo *Back-Propagation*.
- Entrenamiento no supervisado o auto-asociativo. Este tipo de algoritmos son utilizados en problemas de clasificación. Este tipo de algoritmos entrenan la red a partir de vectores de entrada sin conocer la salida deseada del problema, puntuando el grado de acierto que se obtiene en cada ejecución.

En la elaboración de este trabajo se hará uso de de un tipo de entrenamiento supervisado.

### Algoritmo de entrenamiento: *Back-Propagation*

El algoritmo más utilizado para el entrenamiento supervisado de redes es el llamado *back-propagation*. Este algoritmo aplica la técnica del descenso por gradiente teniendo en cuenta la propagación del error desde la salida de la red a las capas ocultas.

El entrenamiento de una red consiste buscar un mínimo en la función de error $E(w_1, w_2..w_n)$. El método del descenso por gradiente es capaz de reducir en cada iteración el error en el espacio multidimensional definido por la función error moviéndose en ella según su gradiente. Este espacio tendrá tantas dimensiones como pesos tenga la red.

En la siguiente imagen recogida de [10] se muestra un espacio de error de dos dimensiones:

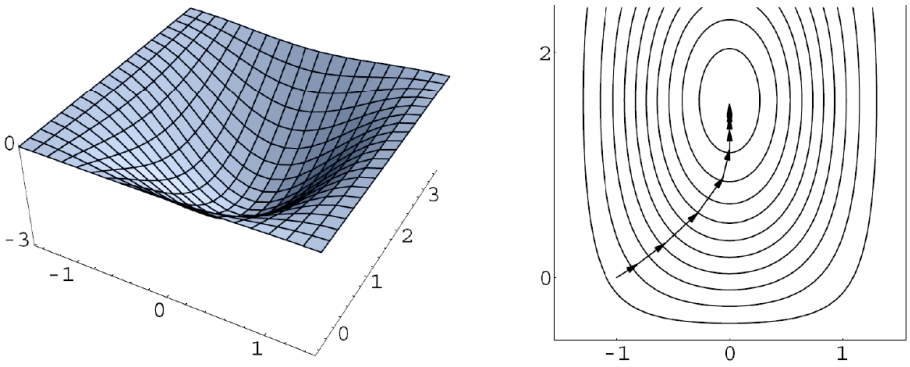



Figura 11: Función error de 2 dimensiones

En cada iteración los pesos se actualizaran según la siguiente ecuación:

$$w_n(i) = w_n(i-1) * paso * \frac{\partial E}{\partial w_n} \tag{3}$$

La técnica de *back-propagation* nos permite calcular la incidencia de cada peso en el error total de la red, esto es, la derivada de la función error respecto a cada peso. Observando la propagación del error desde la salida podemos descomponer la derivada total del error en múltiples derivadas:

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial out2} * \frac{\partial out2}{\partial in2} * \frac{\partial in2}{\partial out1} * \frac{\partial out1}{\partial w_1} \tag{4}$$

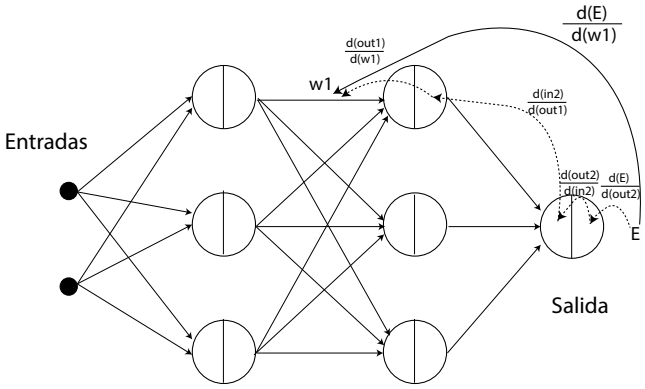



Figura 12: *Back-Propagation*

Sobre este algoritmo se han desarrollado los métodos de entrenamiento que se usan comúnmente para el desarrollo de redes. En el caso de las redes RBF debido a sus características utiliza otro método de entrenamiento que se explicará en la tarea 3, donde se hará un estudio especifico de las redes MLP y las RBF.

# 2. Metodología

En esta apartado del documento se va a explicar la metodología seguida para elaborar el trabajo. En los siguientes subapartados están explicadas las fases y tareas realizadas. Por ultimo se mostrara un diagrama de Gantt donde se podrá apreciar de manera visual el desarrollo del trabajo.

## 2.1. Descripción de fases y tareas

Este trabajo se realizo en un total de 43 semanas. Se inició en la semana del 5 de diciembre de 2016 y finalizó en la semana del 25 de septiembre de 2017.

El trabajo se ha dividido en 7 tareas agrupadas en 3 fases. A continuación se expondrá de cada tarea: el objetivo, el equipamiento utilizado, el tiempo invertido, los procedimientos seguidos y los resultados obtenidos.

### 2.1.1. Fase 1: Estudio Teórico

Esta primera fase consiste en realizar una introducción al trabajo planteado. Se realizaron 3 tareas cuyos objetivos son realizar una introducción teórica a los dos campos que abarca el trabajo, realizar una búsqueda bibliográfica especifica del problema planteado y por ultimo centrarse en analizar en profundidad las redes RBF y las diferencias que tienen con las MLP.

#### 2.1.1.1 Tarea 1: Estudio de los fundamentos de Redes Neuronales Artificiales y del Robot Paralelo

*Objetivo:*

El objetivo de esta tarea es formar una base teórica en los dos ámbitos del proyecto: las RNA y el robot paralelo. En lo relativo a RNA es necesario tener una visión global de lo que son las redes neuronales: su fundamento, que aplicaciones tienen, las arquitectura existente y los distintos métodos de entrenamiento. En el ámbito de robots paralelo, conocer las diferencias entre robot serie y paralelo, el problema dinámico, y específicamente

la estructura del robot 3PRS.

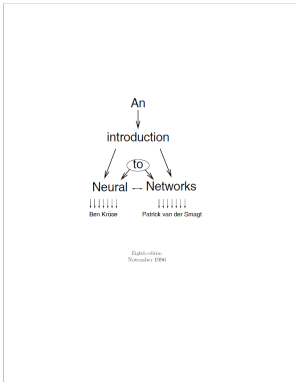
*Duración:*

La duración de esta tarea fue de 8 semanas. Se inicio en la semana 1 del proyecto y finalizo en la semana 8.

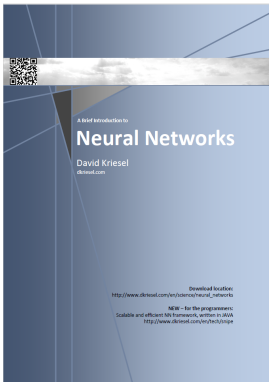
*Equipamiento:*

Se hizo uso de los siguientes libros de introducción a las redes neuronales:

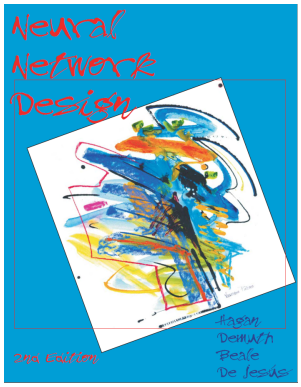
- *An Introducction to Neural Networks* de Ben Krose y Patrick van der Smagt [11].
- *A Brief Introduction to Neural Network* de David Kriesel [10].
- *Neural Network Design* de Martin T. Hagan [7].
- *Neural Networks* de Raúl Rojas [17].



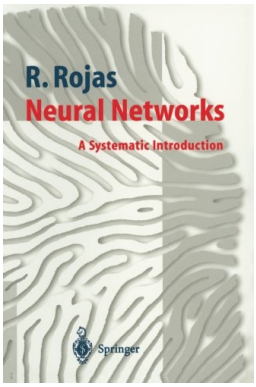

(a) *An Introducction to Neural Networks*




(b) *A Brief Introduction to Neural Network*




(c) *Neural Network Design*

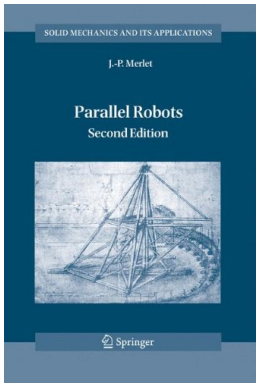



(d) *Neural Networks*

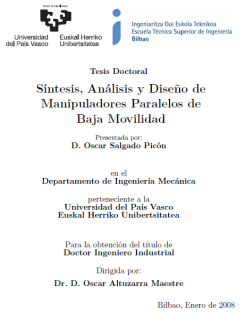
Figura 13: Bibliografía introductoria a la RNA

Para la realizar la introducción a los robots paralelos se hizo uso de:

- *Parallel Robots* de J.P Merlet [14].
- *Síntesis, Análisis y Diseño de Manipuladores Paralelos de Baja Movilidad* de D. Oscar Salgado Picón [19].




(a) *Parallel Robots*




(b) *Síntesis, Análisis y Diseño de Manipuladores Paralelos de Baja Movilidad*

Figura 14: Bibliografía introductoria a el Robot Paralelo

*Procedimiento:*

Para la elaboración de la introducción a las RNA se hizo uso principalmente del libro *An Introducction to Neural Networks* de Ben Krose y Patrick van der Smagt [11]. Es el libro más indicado para introducirse en la teoría de las RNA, posee una distribución de capítulos fácil de seguir y una explicaciones más sencillas que [10], [7] y [17]. También se hizo uso de estos últimos libros para complementar las explicaciones de los temas difíciles de comprender de [11].

Para el campo de los robot paralelos se hizo uso tanto del libro *Parallel Robots* de de J.P Merlet [14] como de la tesis *Síntesis, Análisis y Diseño de Manipuladores Paralelos de Baja Movilidad* de D. Oscar Salgado Picón [19]. Se utilizaron los capítulos de introducción de ambos documentos para obtener una idea global de los robot paralelos y también capítulos específicos para conocer las diferentes configuraciones de robot paralelos que existen, los problemas dinámicos que plantean y las diferencia que poseen con los robot serie.

*Resultados:*

De esta tarea se obtuvo una base teórica de los dos ámbitos que trata el trabajo. Además se realizó la introducción teórica del apartado 1.5 del documento.

#### 2.1.1.2 Tarea 2: Búsqueda bibliográfica

*Objetivo:*

El objetivo de esta tarea es realizar un búsqueda especifica de artículos relacionados que puedan ayudar en la elaboración del trabajo. Ya sean artículos que implementen redes neuronales MLP o RBF para dar solución al problema dinámico directo o inverso de cualquier tipo de robot paralelo, o mas específicos que utilicen solo las redes RBF o que utilicen cualquier tipo de red sobre el robot 3PRS.

*Duración:*

La duración de esta tarea fue de 3 semanas. Se inicio en la semana 8 del proyecto y finalizo en la semana 11.

*Equipamiento:*

Para realizar el objetivo de la tarea se hizo uso de los siguientes buscadores:

- Springer `https://link.springer.com/`
- IEEE Xplorer `http://ieeexplore.ieee.org`
- Scopus `https://www.scopus.com`
- Google Scholar `https://scholar.google.es/`

*Procedimiento:*

Utilizando los buscadores listados se realizaron 3 búsquedas distintas:

1. Una búsqueda general incluyendo cualquier tipo de RNA utilizada para resolver el DKP o IKP de cualquier robot paralelo.
2. Una búsqueda especifica que solo utilice redes RBF en la resolución del DKP o IKP de cualquier robot paralelo.
3. Una búsqueda especifica aplicada solamente al robot paralelo 3PRS haciendo uso de cualquier tipo de RNA.

*Resultados:*

A continuación se muestran los resultados de las 3 búsquedas descritas en el anterior apartado:

- Búsqueda 1: Cualquier RNA + cualquier robot paralelo + DKP/IKP
  - Neural Network Solution For The Forward Kinematics Problem Of A Redundant Hydraulic [6]
  - NN-based solution of forward kinematics of 3DOF parallel spherical manipulator [12]
  - Neural Network Solution for Forward Kinematics Problem of HEXA Parallel Robot [5]
  - A neural-network committee machine approach to the inverse kinematics problem solution of robotic manipulators [9]
  - Real-time solution of the forward kinematics for a parallel haptic device using a numerical approach based on neural networks [13]
  - Solving the Forward Kinematics of Cable-Driven Parallel Robots with Neural Networks and Interval Arithmetic [20]
  - An improved hybrid method for forward kinematics analysis of parallel robots [8]
  - Direct Kinematics Solution of 2-(6UPS) Hybrid Manipulator based on Neural Network [15]
  - Application of neural network training in forward kinematics simulation for a novel modular hybrid manipulator with experime [16]
- Búsqueda 2: Redes RBF + cualquier robot paralelo + DKP/IKP
  - Neural Networks Approaches for Computing [18]
  - Kinematic analysis of a novel 3-DOF actuation redundant parallel [24]
  - Neural Network Solutions of Forward Kinematics for 3RPS Parallel Manipulator [1]
  - A New Algorithm for Solving Inverse Kinematics of Robot Based on BP and RBF Neural Network [23]
  - Direct Kinematics Solution of 3-RRR Robot by Using Two Different Artificial Neural Networks [2]
- Búsqueda 3: Cualquier RNA + robot paralelo 3PRS + DKP/IKP
  - Workspace generation of the 3-PRS parallel robot based on the NN [3]
  - A 3-PRS Parallel Manipulator Control Based on Neural Network [22]
  - Real Time Parallel Robot Direct KinematicProblem Computation using Neural Networks [25]

De estas búsquedas se extrae que el uso de las RNA para resolver problemas cinemáticos es un foco de estudio en la actualidad. Las redes MLP y RBF son las más utilizadas para resolver este tipo de problemas, aunque también se plantean soluciones con otro tipo de redes como las *Wavelet* [16] o conjuntos de redes [9].

#### 2.1.1.3 Tarea 3: Estudio especifico de las redes MLP y RBF

*Objetivo:*

El objetivo de esta tarea es el estudio en profundidad de los tipos de redes neuronales MLP y RBF. Se estudiará la estructura de los dos tipos de redes, las características de cada una, sus algoritmos de entrenamiento y las ventajas y desventajas de cada una de ellas.

*Equipamiento:*

En esta tarea se hace uso de los libros introductorios a las redes neuronales listados en la tarea 1:

- *An Introducction to Neural Networks* de Ben Krose y Patrick van der Smagt [11].
- *A Brief Introduction to Neural Network* de David Kriesel [10].
- *Neural Network Design* de Martin T. Hagan [7].
- *Neural Networks* de Raúl Rojas [17].

Como ampliación en el tema de redes RBF también se hizo uso de la explicación del capitulo 6 del manual de Matlab *Neural Network Toolbox™ User's Guide*.

*Duración:*

La duración de esta tarea fue de 4 semanas. Se inicio en la semana 11 del proyecto y finalizo en la semana 15.

*Procedimiento:*

El procedimiento seguido en esta tarea es la lectura de los capítulos referentes a redes MLP y RBF de los libros citados en el apartado de equipamiento de la tarea

*Resultados:*

A continuación expone lo aprendido de las redes neuronales MLP y RBF.

**Red MLP**

Las redes MLP surgieron a partir de la necesidad de ampliar las posibilidades del tipo

de red *Perceptron*, red de una sola neurona que solo podía dar solución a problemas muy sencillos.

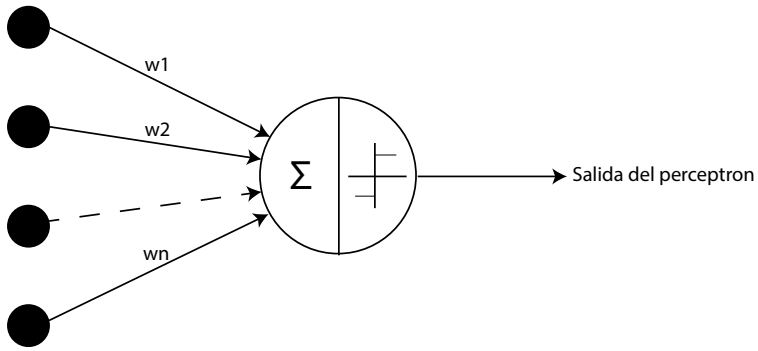



Figura 15: *Perceptron*

Este tipo de red de una sola neurona, agrupaba los valores de entrada ponderados por un peso y mediante una función de activación signo daba una respuesta binaria al problema. Este tipo de red se entrenaba de mediante un sencillo algoritmo. Esta red no era capaz de resolver problemas con un mínimo de complejidad, por ejemplo emular una puerta lógica XOR.

Añadiendo más neuronas a la red *Perceptron* se crearon las redes MLP. Redes que puede cualquier número de neuronas distribuidas en un número ilimitado de capas ocultas. Gracias a este aumento en la complejidad de la estructura, esta red es capaz de resolver problemas mucho mas complejos que la red *Perceptron*.

Las redes MLP poseen una estructura *feed forward*, los impulsos de entrada se propagan desde la entrada a la salida sin ningún tipo de realimentación. Tampoco puede tener conexione entre neuronas de la misma capar o de capas que no sean adyacentes.

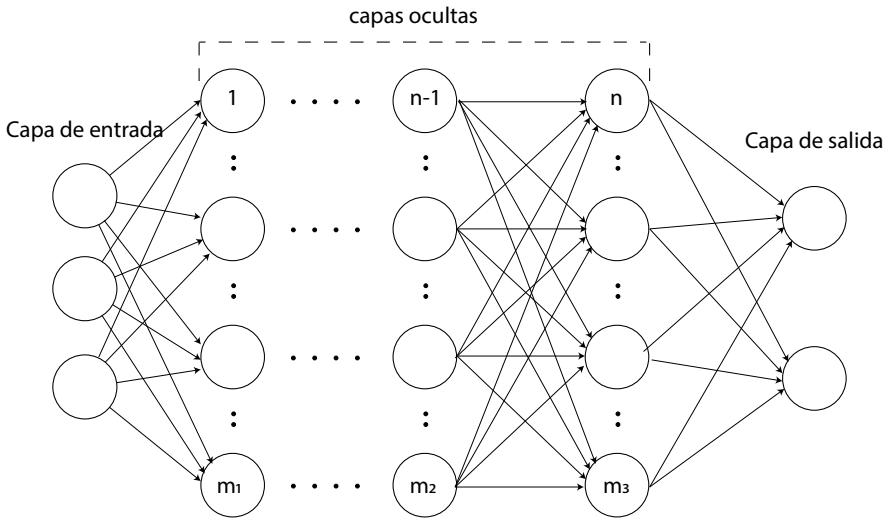



Figura 16: *Estructura de la red MLP*

A diferencia que la red *Perceptron* en las redes MLP la función de propagación además de agrupar las salidas ponderadas por los pesos también suma un offset de neurona. Las funciones de activación aparte de la función signo también pueden ser de los siguientes tipos: lineal, Relu, sigmoidea o tangente hiperbólica entre otras. Las funciones de activación puedes variar entre capas para adecuarse la red al comportamiento deseado.

Debido al aumento en la complejidad de la red los algoritmos de entrenamiento también son más complejos. Todos ellos parten del concepto de descenso de gradiente. Algunos ejemplos de estos algoritmos son: *Levenberg-Marquardt*, *Bayesian regularization backpropagation* y *Scaled Conjugate Gradient*.

La mayoría de algoritmos tiene un parámetro para regular velocidad de aprendizaje, este parámetro es denominado *learning rate*. Este parámetro ajusta el grado de variación que tendrán los parámetros de la red en cada iteración. Un valor bajo de este parámetro hará que el entrenamiento de la red se ralentice y tarde demasiadas iteraciones hasta que la red converja a un valor mínimo de error. Un valor demasiado alto provocara que los parámetros no converjan y 'salten' de manera continuada haciendo que la el error 'salte' alrededor de un mínimo sin llegar a acercarse a él. En la siguiente imagen se muestra este concepto:

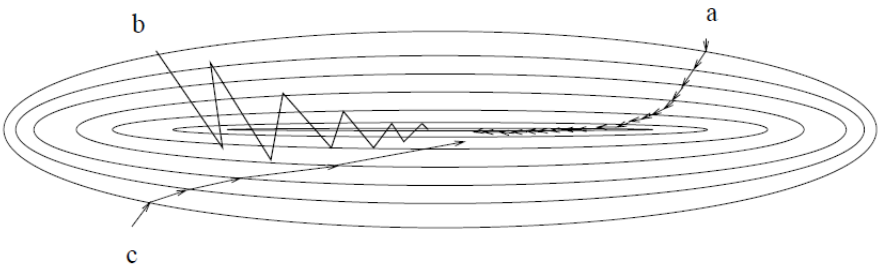



Figura 17: Efecto del parámetro *learning rate*. a) *Learning rate* bajo, b) *Learning rate* óptimo, c) *Learning rate* alto. Imagen obtenida de [11]

El número de unidades y capas ocultas también es un parámetro de la red que se debe escoger cuidadosamente. Aunque una red con mas neuronas ocultas tenga más potencial para resolver problemas complejos, si el problema es simple y el número de capas y neuronas ocultas es alto se obtendrá un comportamiento de la red demasiado complejo que puede dar valores bajo de error en las muestras de entrenamiento pero no sera capaz de emular el comportamiento general del problema y obtendrán malos resultados en las muestras de test.

A continuación se muestra una imagen donde se aprecia el concepto anterior:

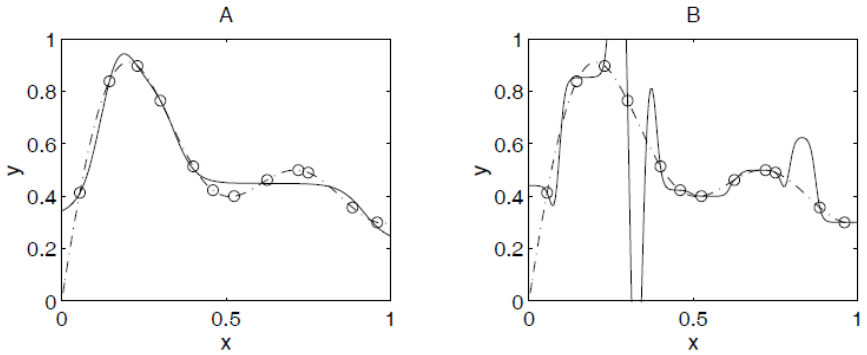



Figura 18: Efecto del número de neuronas de las capas ocultas en el rendimiento de la red. a) 5 neuronas en la capa oculta, b) 20 neuronas en la capa oculta. Imagen obtenida de [11]

Los algoritmos de entrenamiento no garantizan que se llegue al mínimo absoluto de la solución. Dependiendo de la complejidad del algoritmo serán capaces en mayor o mejor medida de evitar converger a mínimos relativos. La solución de este problema es realizar múltiples entrenamientos variando en cada uno de ellos los parámetros iniciales de la red, así el camino que seguirá el algoritmo de aprendizaje será distinto.

Debido a la complejidad de estos algoritmos y la necesidad de repetir los entrenamien-

tos, el tiempo de entrenamiento de una red MLP es grande. Este tiempo varía en función del numero de neuronas por capa y sobre todo por el número de capas de la capa oculta. Una vez entrenada la red el tiempo de ejecución es reducido. La ejecución de este tipo de redes se reduce a multiplicaciones matriciales y pasos por las funciones de activación.

**Red RBF**

Las redes RBF se desarrollaron en la década de los 80 por Michael J. D. Powell como solución a un problema interpolación exacta en un espacio multidimensional. Este tipo de redes al igual que las MLP pueden ser utilizadas en problemas de aproximación y reconocimiento de patrones. Poseen una estructura y cualidad diferentes a las MLP, explicadas a continuación.

Las redes de RBF componen la solución mediante la agrupación de funciones de base radial. La característica de este tipos de funciones es que su salida disminuye con la distancia al punto central. La función mas utilizada en las RBF es la distribución de gauss [7] pero existen otras funciones:

- Función distancia: $\phi(r) = d$
- Función multicuadrática inversa: $\phi(r) = \frac{1}{\sqrt{1+(\epsilon r)^2}}$

Estas dos funciones poseen las siguientes desventajas respecto a la distribución de gauss: la función distancia no es capaz de generar un curva aunque se agrupen un gran numero de funciones y la función multicuadratica inversa posee un offset que puede ser indeseado.

Por ello la función de distribución de Gauss es la mas utilizada, se muestra a continuación:

$$\phi(r) = e^{(-\epsilon r)^2}$$

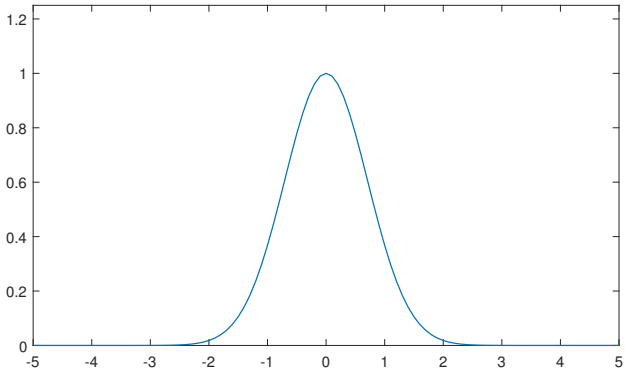



Figura 19: Función de distribución de gauss

Posee una estructura *feed forward* al igual que las redes MLP. La estructura de la red es la siguiente: una de entrada, una capa de base radial y una capa lineal. En la siguiente figura se representa un red RBF:

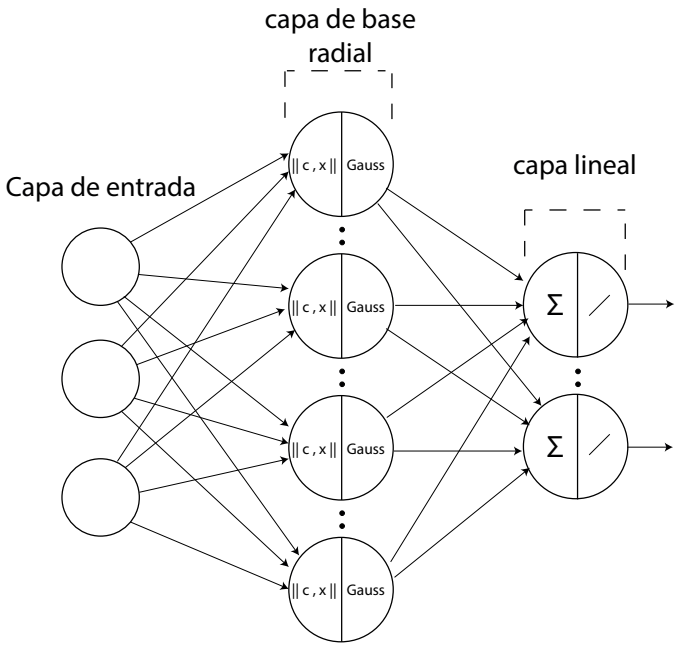



Figura 20: *Estructura de la red RBF*

Al igual que la capa de entrada de cualquier red neuronal, posee el mismo numero de neuronas que dimensiones tenga el vector de entrada y tendrán como salida el valor de los vectores de entrada.

La capa de base radial es la capa intermedia de la red. Posee unas funciones de pro-

pagación y activación muy diferentes a las utilizadas en redes MLP.

La función de propagación consiste en el calculo de la distancia euclidea entre los vectores de entrada y los pesos. En el caso de las redes RBF, los pesos de esta capa se denominan centros ya que serán los centros de los de las funciones gaussianas que forman cada neurona.

$$y_k(t) = ||c, x||$$

La función de activación es la función de base racial comentada anteriormente. Como se ha dicho, la función de distribución gausiana es la mas utilizada por lo que de aquí en adelante solo hará referencia a ella cuando se hable de función de activación en las redes RBF.

La capa de lineal tiene el mismo número de neuronas como dimensiones de salida tenga la red. Tiene como objetivo agrupar las salidas de las neuronas de la capa anterior multiplicadas por sus pesos y sumarle el offset de la neurona.

A diferencia de las redes MLP, las redes RBF solo dan un salida en el espacio de entrenamiento. Las redes MLP pueden ser utilizadas para extrapolar comportamientos fuera del espacio de entrenamiento ya que sus neuronas obtienen una salida en todo el espacio. Al contrario, las neuronas de las redes RBF tienen como salida un valor nulo si el vector de entrada se encuentra lejos de los centros de las neuronas que estarán ubicados en el espacio de entrenamiento.

La segunda diferencia con las redes MLP, es que las neuronas de las redes RBF solo se activan de manera local. En cambio el solapamiento de las neuronas de las redes MLP es total. Esto implica que si se requiere cubrir un espacio de entrada amplio el numero de neuronas RBF sera mucho mayor que el de una red MLP.

Los método de entrenamiento son sencillos, esto se debe a la localidad de las neuronas RBF. En la tarea 4.a se explican en detalle los algoritmos que dispone matlab para el entrenamiento de este tipos de redes.

Los tiempos de entrenamiento son reducidos gracias a la sencillez de los algoritmos. Por el contrario, los tiempos de ejecución son mayores ya a que por norma general una red RBF va a necesitar más neuronas para resolver el mismo problema que las redes MLP debido a la localidad, ya comentada, de las neuronas RBF.

**Diferencias entre redes MLP y RBF**

A continuación se resumen las principales diferencia entre los dos tipos de redes:

- Estructura multicapa de las MLP y estructura fija de 3 capas de la RBF
- Funciones propagación: basadas en la suma ponderada de las MLP y basadas en el calculo de la distancia euclidea en las RBF.
- Funciones de activación: lineal, Relu, signo, sigmoidea o tangente hiperbólica de las MLP y funciones de base radial en las RBF.
- Localidad de las neuronas RBF y globalidad de las neuronas MLP.
- Tiempos de ejecución de redes MLP menores que las redes RBF.
- Tiempos de entrenamiento de redes RBF menores que las redes MLP.

### 2.1.2. Fase 2: Entrenamiento de la red

En esta fase se preparará y se realizará el entrenamiento de la red. Para cumplir este objetivo se realizaron 2 tareas: una para introducirse al entorno de desarrollo de RNAs de Matlab y otra para realizar el propio entrenamiento de la red.

#### 2.1.2.1 Tarea 4: Introducción al entorno de desarrollo de RNA en Matlab

Esta tarea se divide en 3 subtareas. La primera subtarea tiene como objetivo familiarizar se con el entrorno de desarrollo de RNA de Matlab, la segunda modificar la función de entrenamiento para facilitar el entrenamiento en batería y la última subtarea realizar una primera resolución de un problema ejemplo mediante RNA.

**Tarea 4.a: Familiarización con el Toolbox de Matlab de RNA**

*Objetivo:*

El objetivo de esta tarea es realizar un primer contacto con el Toolbox de Matlab de RNA para determinar que herramientas dispone para el desarrollo de redes y cual es la que más conviene utilizar para el desarrollo del trabajo.

*Equipamiento:*

Se ha hecho uso de los siguientes elementos para el desarrollo de la tarea:

- Matlab R2016a
- Neural Network Toolbox
- Documentación del toolbox: *Neural Network Toolbox™ Getting Started Guide*, *Neural Network Toolbox™ Reference* y *Neural Network Toolbox™ User's Guide*.
- Webinars y codigos de ejemplo del toolbox albergados en la dirección `https://es.mathworks.com/products/neural-network.html`

*Duración:*

La duración de esta tarea fue de 4 semanas. Se inicio en la semana 15 del proyecto y finalizo en la semana 19.

*Procedimiento:*

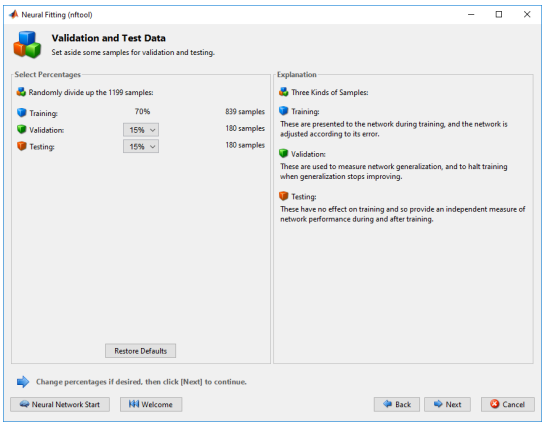
En primer lugar para tener una visión global del funcionamiento del toolbox se hizo uso tanto de los Webinar como de los códigos de ejemplo de la web de matlab. En ellos se muestra el uso de de la interfaz gráfica para el desarrollo de redes que incluye el toolbox. Haciendo un uso de esta interfaz se observaron las limitaciones que tiene. A continuación, utilizando los documentos de ayuda *Neural Network Toolbox™ Getting Started Guide*, *Neural Network Toolbox™ Reference* y *Neural Network Toolbox™ User's Guide* se analizaron las función generales que incluye el toolbox y la especificas para el desarrollo de redes RBF.

*Resultados:*

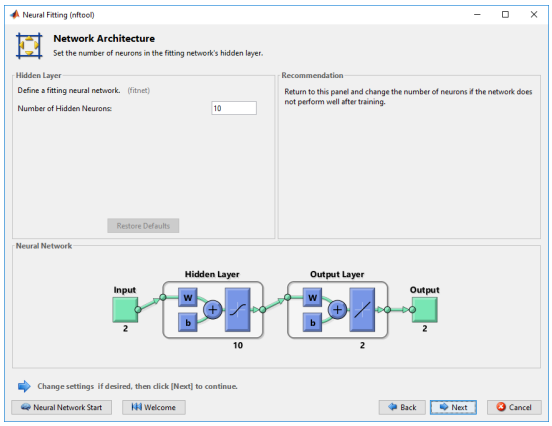
Los conocimientos que se han obtenido en la realización de esta tarea se exponen a continuación:

**Interfaz gráfica de desarrollo de RNA**

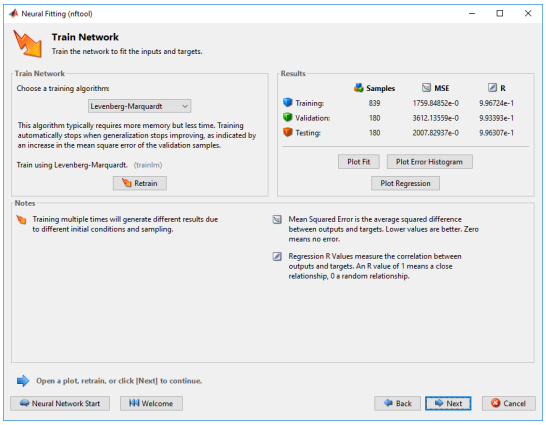
La interfaz gráfica incluida en el toolbox posee la siguiente apariencia:




(a) División entre vectores de entrenamiento y de test




(b) Selección del número de neuronas




(c) Selección del algoritmo de entrenamiento


(d) Visualización del resultado del entrenamiento

Figura 21: Interfaz gráfica de desarrollo de redes del *Neural Network Toolbox*

Gracias a esta interfaz podemos entrenar de una manera sencilla RNAs para resolver problemas como: análisis de grupos, ajuste de curvas, reconocimiento de patrones o predicción en series temporales. Antes de realizar el entrenamiento se puedes configurar varios parámetros como son: el número de neuronas de la capa oculta (solo admite una capa oculta), el porcentaje de vectores de entrada que se usaran como vectores de test y el algoritmo de entrenamiento. Una vez realizado el entrenamiento nos mostrara los resultados: número de iteraciones, el error cuadrático medio, el gradiente de la curva de error en cada iteración, etc.

Esta interfaz tiene una serie de limitaciones que impiden que pueda ser utilizada para resolver todo tipo de problemas como son: la falta de poder modificar el número de capas ocultas ,y las funciones de activación y propagación, como también el poder realizar series

de entrenamientos y no quedarte con la primera solución a la que haya llegado al algoritmo de entrenamiento que puede ser un mínimo local.

Esta interfaz puede ser útil a la hora de solucionar problemas de poca complejidad, pero si el problema es de gran envergadura es necesario utilizar las funciones del toolbox.

**Funciones generales**

La función más general para crear un red neuronal es `network`. En esta función se puede especificar la dimensión de entrada de la red, el número de capas, la configuración los offsets de las capas y las conexiones entre capas ocultas y de salida.

```
net = network(numInputs,numLayers,biasConnect,inputConnect,layerConnect,
                outputConnect)
```

También existen funciones para problemas específicos con parámetros de configuración más acotados, como son `feedforwardnet`, `fitnet`, `patternnet` y `cascadeforwardnet`.

```
net = feedforwardnet(hiddenSizes,trainFcn)
net = fitnet(hiddenSizes,trainFcn)
net = patternnet(hiddenSizes,trainFcn,performFcn)
net = cascadeforwardnet(hiddenSizes,trainFcn)
```

La función `feedforwardnet` se utiliza para definir redes MLP, `fitnet` y `patternnet` son funciones similares pero modificadas para adecuarse a los problemas de ajuste de curvas y reconocimiento de patrones. Por último, La función `cascadeforwardnet` permite crear redes con conexiones entre capas no adyacentes.

Todas las funciones nombradas retornan un objeto `network` ya configurado. Este objeto describe el comportamiento de la red: las dimensiones de entrada y salida, el número de neuronas de las capas. las conexiones entre neuronas, las funciones de propagación y activación y también el algoritmo de entrenamiento. Además contiene los valores de la red como pesos y offsets, que antes de entrenar la red no tendrán valores significativos.

La función utilizada para entrenar la red es `train`. Esta función ejecutará el algoritmo de entrenamiento descrito en el objeto `network`.

```
[net,tr] = train(net,X,T,Xi,Ai,EW)
```

A parte de esta función existen otras para entrenar la red con otro algoritmo que no sea el descrito en el objeto `network`. Estas funciones son:

- `trainlm`: algoritmo *Levenberg-Marquardt*
- `trainbr`: algoritmo *Bayesian regularization backpropagation*
- `trainscg`: algoritmo *Scaled Conjugate Gradient*
- `trainrp`: algoritmo *Resilient Backpropagation*
- `trainbfg`: algoritmo *BFGS Quasi-Newton*
- `traincgb`: algoritmo *Conjugate Gradient with Powell/Beale Restarts*
- `traincgf`: algoritmo *Fletcher-Powell Conjugate Gradient*
- `traincgp`: algoritmo *Polak-Ribiére Conjugate Gradient*
- `trainoss`: algoritmo *One Step Secant*
- `traingdx`: algoritmo *Variable Learning Rate Backpropagation*

Para calcular el valor de rendimiento, siendo el más común el Error Cuadrático Medio (MSE), definido en la red se hace uso de la función `perform`:

```
perf = perform(net,t,y,ew)
```

Para ejecutar la red se llama al objeto `network` pasándole los vectores de entrada, retornando los vectores de salida.

```
output = net(input)
```

**Funciones especificas para redes RBF**

Las funciones de configuración y entrenamiento para redes RBF son diferentes a las mostradas. Debido a las características de la red , descritas en la tarea 3, su modo de configuración y entrenamiento es diferente. Las funciones de entrenanimento son `newrbe` y `newrb`. Tienen los siguientes parámetros de entrada y de salida:

```
net = newrbe(P,T,SPREAD)
net = newrb(P,T,GOAL,SPREAD)
```

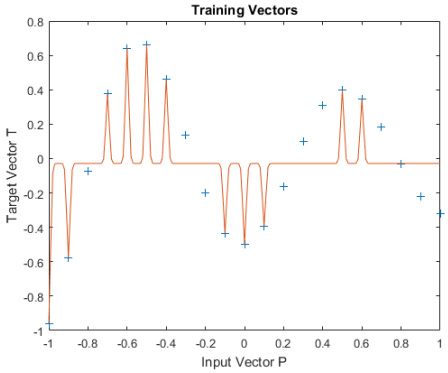
La función `newrbe` configura y entrena la red RBF con el mismo número de neuronas que vectores de entrada. Solamente pasando por parámetro los vectores de entrada, *target* y *spread* (anchura de las gaussianas), nos retorna una red en la cual las neurona tendrán como centros los valores de entrada, cuyos anchuras serán el valor el parámetro de entrada spread, y sus alturas serán los valores del vector target y los offsets nulos. Para obtener un valor de spread óptimo que permita que el valor de error sea mínimo se deberá hacer un barrido en este parámetro.

Este método es inviable si queremos realizar el entrenamiento sobre un dominio amplio ya que tendríamos el mismo número de neuronas que de vectores de entrada, haciendo que la ejecución de la red sea ridículamente lenta.

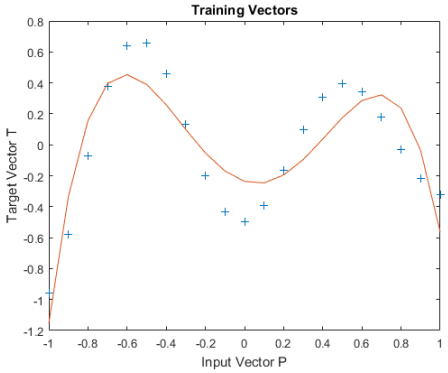
La función `newrb` configura y entrena la red RBF añadiendo neuronas en cada iteración del entrenamiento hasta llegar al objetivo *goal*. El algoritmo utilizado para la selección de los parámetros de las neuronas es el siguiente:

1. Estimular la red con todos los vectores de entrada.
2. Buscar el vector de entrada que genere mas error.
3. Se añade a la red una nueva neurona con el centro igual al vector de entrada del paso anterior.
4. Se ajusta el offset y la altura de las neuronas para minimizar el error.

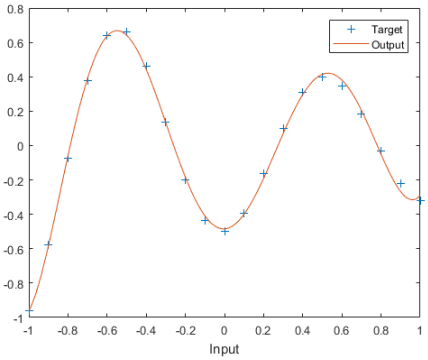
Al igual que la función anterior, al hace uso de ésta también se debe hacer un barrido en el parámetro *spread* para minimizar el error de la red. Un valor de *spread* bajo no permite que la salida de las neuronas se solapen creando el efecto ilustrado en la figura 22.a. Por el contrario si el valor de *spread* es demasiado alto se producirá un efecto de sobre-solapamiento entre neuronas mostrado en la figura 22.b. Gracias a un barrido en el parámetro *spread* podemos obtener un valor óptimo consiguiendo un comportamiento de la red igual al deseado, figura 22.c.




(a) Efecto de *Underlapping* entre neuronas




(b) Efecto de *Overlapping* entre neuronas




(c) *Spread* óptimo

Figura 22: Efecto del valor del parámetro *spread* en redes RBF. Imagenes obtenidas de www.mathworks.com

La principal ventaja del uso de esta última función en comparación con `newrbe` es que el número de neuronas que obtendrá la red una vez entrenada será menor, mejorando así el coste computacional a la hora de ejecutar la red. La única desventaja es que el tiempo de entrenamiento es más elevado.

Para realizar el entrenamiento de la red objetivo del trabajo se hará uso de una modificación de la función `newrb`.

### Tarea 4.b: Modificación de la función de entrenamiento

*Objetivo:*

El objetivo principal es modificar la función `newrb` incluida en el toolbox para facilitar y mejorar los entrenamientos en batería. Como objetivo secundario se intentará realizar una modificación del algoritmo de entrenamiento que permita la creación de redes con parámetro de *spread* distintos.

*Equipamiento:*

Para el desarrollo de esta tarea se hizo uso del software Matlab con su toolbox Neural Network.

*Duración:*

La duración de esta tarea fue de 2 semanas. Se inicio en la semana 19 del proyecto y finalizo en la semana 21.

*Procedimiento:*

En primer lugar para realizar el objetivo principal de facilitar y mejorar los entrenamientos en batería. Para ello se eliminaran las salidas de información (*plots* y *prints*) de entrenamiento. Gracias a esto se conseguirá una mejora en el tiempo de entrenamiento ya que estas salidas consumen recursos y no son útiles para un entrenamiento en batería donde lo importante es el resultado final.

También se modificará el código para dar salida a las redes intermedias que se generan al ejecutar la función. Se modificaran los parámetros de entrada para que se pueda especificar las redes con un número de neuronas determinado de las que se quieren obtener los objetos `network`.

Se realizará una mejora que permitirá reducir el numero de vectores de entrada utilizados en la selección de centros, permitiendo realizar el entrenamiento reduciendo la cantidad memoria utilizada.

Por ultimo, se realizará el objetivo secundario que consiste en permitir que el método de entrenamiento pueda operara con distintos valores de *spread*. Para ello se deberá entender el algoritmo y su implementación. También se realizara una prueba para verificar si esta modificación mejora el comportamiento de la red.

*Resultados:*

Tras realizar las modificaciones descritas en el procedimiento la declaración de la función final tiene el siguiente aspecto:

```
[net,performance] = newrb_GICI(p,t,f,spread,num_neuronas)
```

A continuación se explica el significado de los parámetros de entrada de la función:

- `p`: Matriz de RxQ. Q vectores input con R dimensiones.
- `t`: Matriz de RxS. Q vectores target con S dimensiones.
- `f`: Número entero por el cual es dividido los vectores de entrenamiento para reducir el uso de memoria en la selección de centros de las neuronas.
- `spread`: Escalar o Vector de 1xN con los valores de ensanchamiento de las neuronas.
- `num_neuronas`: Escalar o Vector de 1xM número de neuronas de las cuales se quiere obtener la red neuronal.

Los parámetros de salida son los siguientes:

- `net`: Celda de 1xM con las M redes cuyo número de neuronas está especificado en la variable de entrada `num_neuronas`.
- `performance`: Escalar o vector de 1xM con los valores cuadráticos medios obtenidos al final del entrenamiento de las redes incluidas en `net`.

### Tarea 4.c: Resolución de un problema ejemplo con redes RBF

*Objetivo:*

El objetivo de esta tarea es aprender a resolver un problema mediante redes RBF haciendo uso de la función desarrollada en la tarea anterior. También se comprobará la validez de la mejora realizada en la función de entrenamiento que permite el uso varios valores de *spread*.

*Equipamiento:*

Para desarrollar esta tarea se hizo uso, a parte de Matlab y su Toolbox de Neural Network, un problema ejemplo incluido en los *Neural Network Datasets* llamado `engine-dataset`.

*Duración:*

La duración de esta tarea fue de 4 semanas. Se inicio en la semana 19 del proyecto y finalizo en la semana 23.

*Procedimiento:*

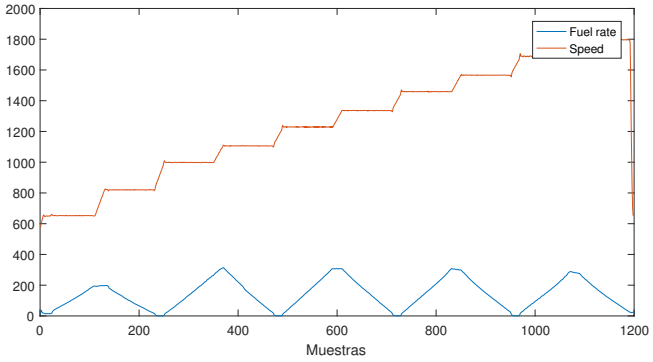
Para resolver el problema ejemplo se seguirán los siguientes pasos:

1. Dividir las muestras del *dataset* en muestras de entrenamiento y muestras de test.
2. Definir con que parámetros se va a realizar los barridos en los entrenamientos.
3. Realizar un entrenamiento inicial con un amplio rango en los parámetros definidos en el anterior paso para conocer en que rango valores se encuentra los valores óptimos.
4. Realizar un segundo entrenamiento con un barrido fino sobre los valores óptimos de los parámetros
5. Una vez obtenida la batería de redes se deberá ejecutar las redes con la muestras de test para obtener los valores de rendimiento.
6. Por ultimo, haciendo uso de los valores de rendimiento se seleccionara la mejor red.

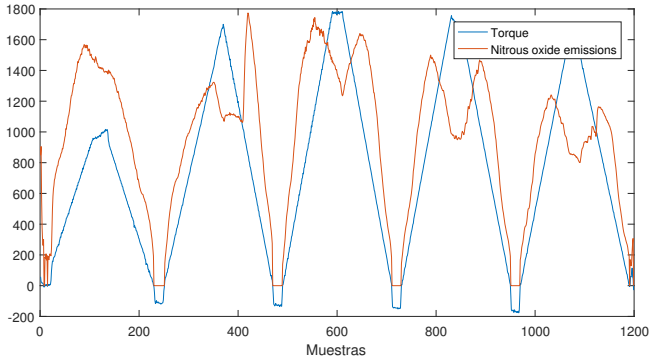
Para verificar la mejora de en la función de entrenamiento que permite varios valores de *spread*, se comparará los valores de rendimiento obtenidos al entrenar las redes con un solo valor de *spread* con las redes entrenadas haciendo uso de varios valores de *spread*.

*Resultados:*

El problema consiste en entrenar una red RBF para resolver la relación entre el consumo y la velocidad de un coche con el par y las emisiones de $NO_2$. En la siguiente imagen se exponen las gráficas con las muestras de entrada y de salida del *dataset* del problema:




(a) Muestras de entrada




(b) Muestras de salida

Figura 23: *Dataset* del toolbox de RNA de Matlab: *engine_dataset*

En primer lugar se realizó la división de las muestra, utilizando el 50 % de las muestras para entrenamiento y el otro 50 % para test. Debido a que el numero de muestras no es muy grande no se pudo hacer uso de un porcentaje mayor de las muestras de entrenamiento sin comprometer la validación.

Para resolver este problema se realizara barridos en los parámetros de la red *spread* y numero de neuronas. En un primer entrenamiento se dedujo que los valores óptimos de estos parámetros se encontraban en el rango de [10,100] neuronas y [10,150] de *spread*.

Una vez realizado el entrenamiento final en el rango de parámetros seleccionado se obtuvo el siguientes resultados en función del numero de neuronas de la red:

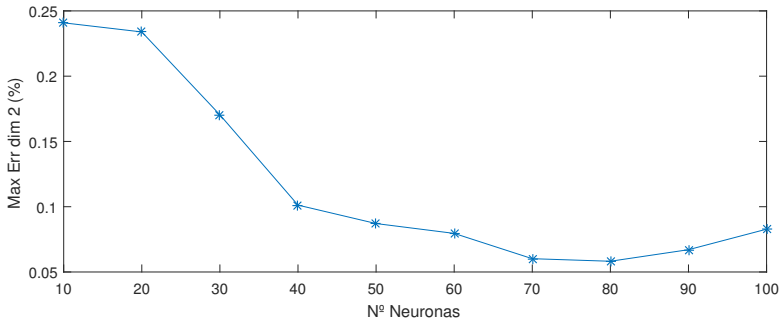



Figura 24: Rendimiento de las redes en función del numero de neuronas

Fijándonos en la salida de emisiones de $NO_2$, que es la dimensión más compleja de ajustar, se obtiene que la la red que mejor rendimiento tiene es la red con un número de neuronas de 80 y un *spread* de 120.

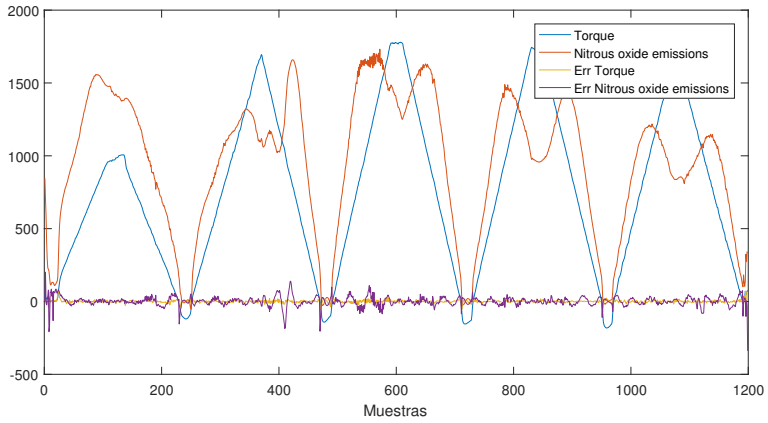



Figura 25: Salida de la red neuronal seleccionada

Para verificar la mejora de en la función de entrenamiento que permite varios valores de *spread*, se realizo el mismo procedimiento de entrenamiento pero utilizando varios valores de *spread*. Se utilizaron `[0.1 0.2 .5 1 2 5 10] .* spread` para generar los valores de *spread* de cada iteración.

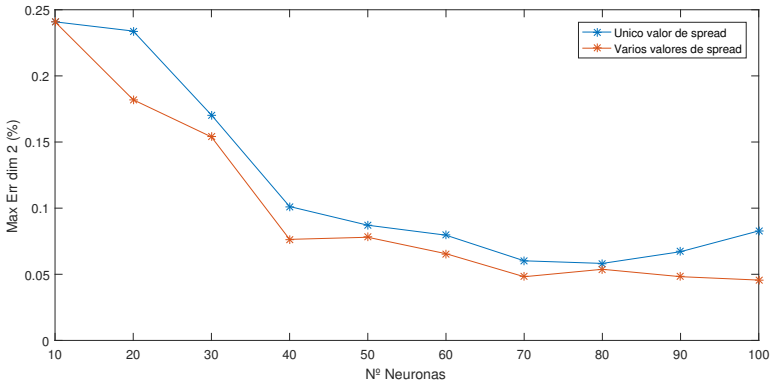



Figura 26: Comparación del rendimiento de red obtenido haciendo uso de varios valores de *spread*

En la anterior gráfica se observa que permitiendo que la red tome varios valores de spread se obtiene una mejora en el rendimiento. Esto es debido a que el ajuste se puede realizar mejor si componiendo la solución con funciones de distribución de gauss de distintas anchuras. Debido a que la modificación no se realizo a partir de una formulación matemática si no a partir de la lógica del algoritmo no se puede afirmar que este comportamiento de mejora de la solución se pueda obtener siempre.

#### 2.1.2.2 Tarea 5: Entrenamiento de la red

*Objetivo:*

El objetivo de esta tarea es realizar el entrenamiento de la red RBF para resolver el objetivo del trabajo, el cual es la resolución del problema cinemático del robot paralelo 3PRS. Se expondrán las diferentes configuraciones de red que se plantean y se determinara la mejor.

*Equipamiento:*

-Dataset de entrenamiento -Cluster de ordenadores del laboratorio -Parallel computing toobox

*Duración:*

La duración de esta tarea fue de 10 semanas. Se inicio en la semana 23 del proyecto y finalizo en la semana 33.

*Procedimiento:*

En primer lugar se implementará el código de Matlab que permitirá realizar los entrenamientos en batería haciendo uso del *cluster*. Gracias a esto, se podrá realizar entrenamientos en paralelo bajando el tiempo total dedicado a cada tanda de entrenamientos.

El siguiente paso será realizar el código que permita extraer los resultados de cada batería de entrenamientos. Con ellos se extraerá parámetros de rendimiento de cada red permitiendo seleccionar la mejor.

A continuación realizarán los propios entrenamientos haciendo uso del dataset proporcionado por el Grupo de Investigación de Control Inteligente. Se seguirá un procedimiento que permitirá determinar los rangos de parámetros de red óptimos. Debido la dificultad de resolver el problema del DKP, se proponen diferentes configuraciones de redes para resolver cada dimensión de salida por separado. Se realizarán entrenamiento para 3 configuraciones:

1. Una red 3x3. Con esta configuración se resolverán las 3 dimensiones del problema con una única red.
2. Dos redes. Una red 3x1 y otra red 3x2. Se revolverá con una red la dimensión Z, dimensión que contiene más complejidad, y con la otra red las dimensiones $\theta_x$ y $\theta_y$ .

3. Tres redes 3x1. Cada dimensión de salida se resuelve por separado.

Por último, se analizarán los resultados obtenidos, se seleccionarán las mejores redes de cada configuración y se seleccionará la configuración que obtenga mejores resultados.

*Resultados:*

**Preparación para el entrenamiento en el cluster**

Haciendo uso de la herramienta *Admin Center* de matlab , figura 27, se generó un nuevo perfil MATLAB Job Scheduler (MJS) para los entrenamientos. Este perfil se genero con el nombre de 'RBF' y se le otorgaron para realizar los entrenamiento un total de 6 *cores* del total del *cluster*. Una vez generado el perfil MJS ya se podrán enviar las tareas o *tasks* de entrenamiento al cluster.

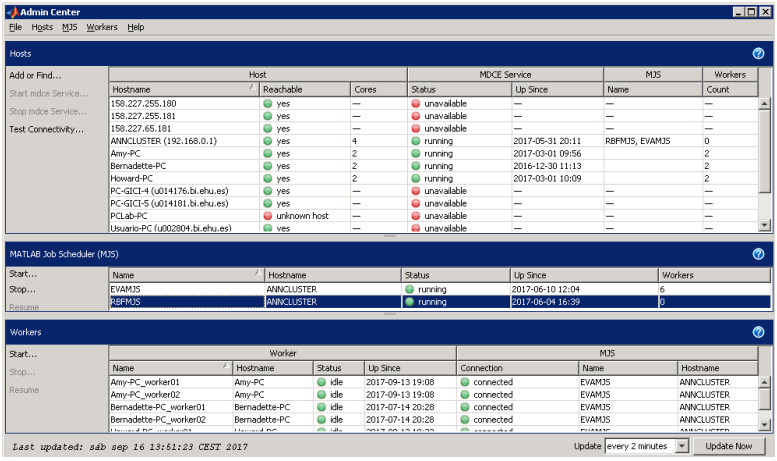



Figura 27: Configuración del perfil MSJ

Para enviar los *jobs* al *cluster* se hace uso de las siguientes funciones incorporadas en el toolbox *Parallel Computing*:

- `parcluster`: para la selección del perfil MJS.
- `createJob`: para la creación de un nuevo trabajo dentro del perfil MJS seleccionado.
- `createTask`: para el envío de tareas al los cores disponibles dentro del perfil MJS.

Para el envío de las tareas se realiza a través de una función que deberá incorporar el código que se desea que cada core ejecute. Se definió una función con el nombre `task_entrenamiento` con los siguientes parámetro de entrada y salida

```
[ net , perf , perf_test ] = task_entrenamiento (input_train,target_train,
```

```
input_test,target_test,f,\textit{spread},num_neuronas)
```

Gracias a esta función se pudo realizar los barridos en los parámetros de red de una manera sencilla. La función retorna, a parte del los objeto *net*, los valores de rendimiento definidos en para la resolución del problema que son los valores de error máximo en las 3 dimensiones de salida.

### Dataset de entrenamiento

El dataset utilizado en los entrenamiento fue realizado y cedido por el Grupo de Investigación de Control Inteligente. Este dataset contiene un conjunto de 38.886 vectores de entrenamiento y 349.980 vectores de test. Estos vectores recorren de manera uniforme todo el espacio de salida. Los vectores de entrenamiento serán utilizados para entrenar las redes y con los vectores de test se obtendrán los valores de rendimiento de la red.

A continuación, se muestra el aspecto de los vectores de entrada y salida de entrenamiento:

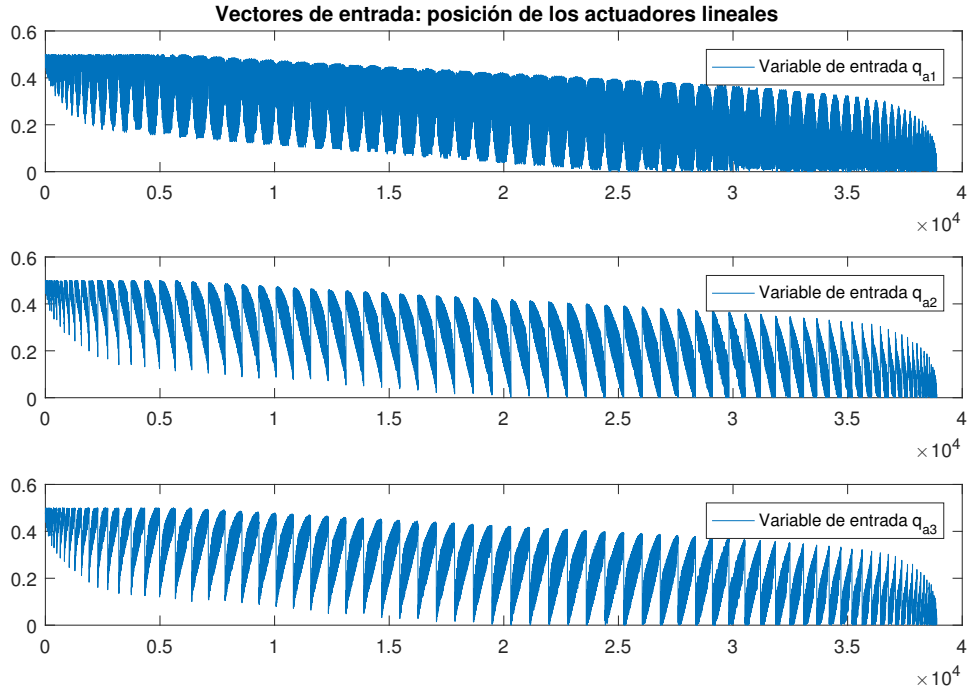



Figura 28: Vectores de entrenamiento de entrada

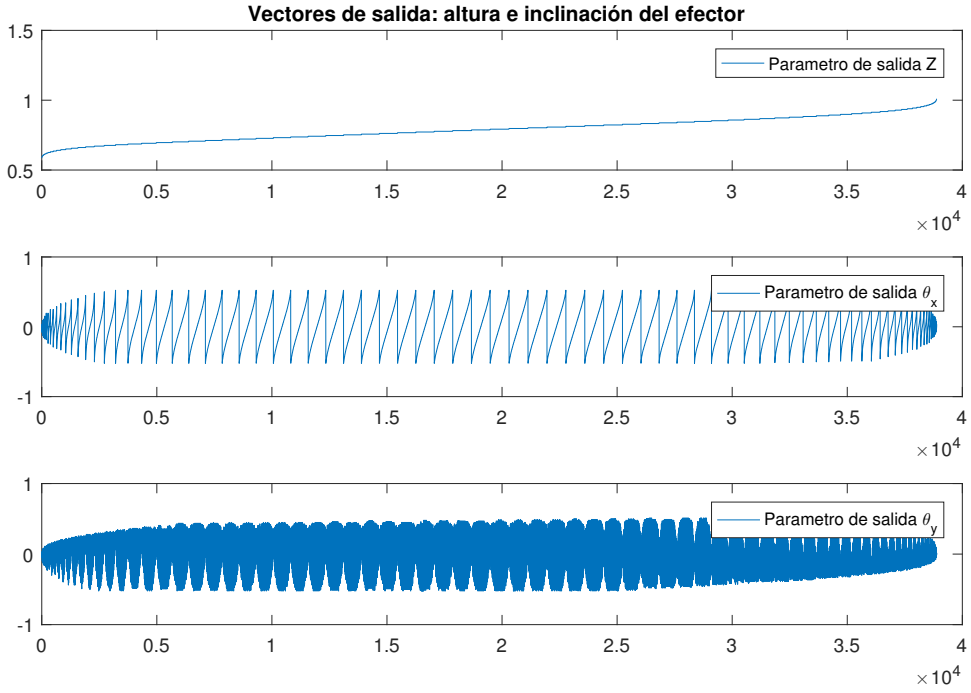



Figura 29: Vectores de entrenamiento de salida

### Tandas de entrenamientos

Una vez generado el código que permita realizar y extraer la información de la batería de entrenamiento se procedió a realizar los propios entrenamiento. Se completaron un total de 14 entrenamientos:

- 7 baterías de entrenamientos para la configuración una red 3x3.
- 4 baterías de entrenamientos para la configuración una red 2x3 más una red 1x3.
- 3 baterías de entrenamientos para la configuración 3 redes de 1x3.

Los parámetros que se hicieron variar en los entrenamientos fueron: numero de neuronas y el *spread*. También se comprobó si existía una mejora de rendimiento haciendo uso de varios valores de *spread*.

En la siguiente tabla se enumeran los entrenamiento que variaciones se hicieron en cada uno:

| | ***Nº neuronas*** | ***spread Base*** | ***spread Factor*** | ***F*** | ***Dim. de salida*** |
|---|---|---|---|---|---|
| **1** | 5:5:150 | 0.5:0.005:2 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **2** | 5:5:150 | 0.5:0.005:2 | [0.5 1 2] | 10 | Z,$\theta_x$,$\theta_y$ |
| **3** | 210:10:300 | 0.2:0.1:0.4 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **4** | 10:10:300 | 0.2:0.05:0.5 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **5** | 10:10:300 | 0.3:0.02:0.5 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **6** | 10:10:300 | 0.2:0.01:1 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **7** | 10:10:200 | 0.3:0.01:2 | 1 | 10 | Z,$\theta_x$,$\theta_y$ |
| **8** | 10:10:200 | 0.3:0.01:2 | 1 | 10 | Z |
| **9** | 10:10:200 | 0.2:0.01:1 | [0.2 0.5 1 2 5] | 10 | Z |
| **10** | 10:10:200 | 0.3:0.005:0.6; | 1 | 10 | Z |
| **11** | 10:10:200 | 0.1:0.1:10; | 1 | 10 | $\theta_x$ |
| **12** | 10:10:200 | 0.1:0.02:1.5; | 1 | 10 | $\theta_x$ |
| **13** | 10:10:200 | 0.1:0.02:1.5; | 1 | 10 | $\theta_y$ |
| **14** | 10:10:200 | 0.1:0.02:1.5; | 1 | 10 | $\theta_x$,$\theta_y$ |

Tabla 1: Entrenamientos realizados

*Spread Factor* hacen referencia a las anchuras que se van a utilizar en cada entrenamiento. Si *spread factor* se trata de un solo numero solo se utilizara una única anchura definida por *spread base*. En cambio, si *spread factor* es un vector de valores las anchuras que se utilizaran en el entrenamiento serán el valor de *spread* base por el vector de *spread factor* obteniendo un conjunto de valores de anchuras.

El valor F hace referencia parámetro de entrada de la función de entrenamiento `newrb_GICI`, que era el numero por el cual se dividían los vectores de entrada que eran utilizados para la selección de centros debido al problema de memoria. Con un valor de 10 se pudieron realizar los entrenamientos en los PC del cluster.

La columna Dimensiones de salidas indica para que dimensiones de salida se entreno la red.

En los entrenamientos 3 y 9 se llego a la conclusión que la modificación realizada en la tarea 4.b que permitía el uso de varios valores de *spread* no mejoraba el rendimiento de las redes, por lo que no se profundizó en utilizar esa característica.

**Resultados de los entrenamientos**

El parámetro definido para medir el rendimiento de la red es el error máximo en cada uno de las dimensiones de salida. Se define como objetivo los siguientes valores:

- Error máximo menor que $10^-4$ de error en Z
- Error máximo menor que 0.1 grados en $\theta_x$ y $\theta_y$

A continuación se muestran los resultados que obtiene las redes de cada configuración:

1. Configuración red 3x3:

   En la siguiente gráfica se muestran los valores de error máximo que obtienen las redes entrenadas en función del número de neuronas:

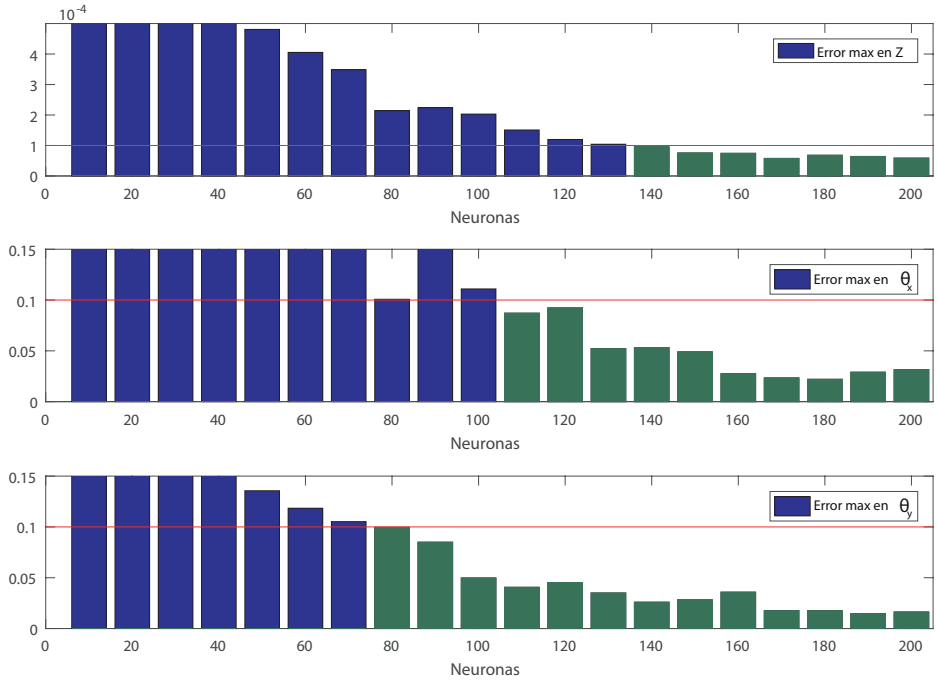



Figura 30: Error máximo en función del número de neuronas de la red 3x3

La red que cumple los objetivos de error es la red de 140 neuronas. En las siguientes gráficas se muestran las salidas la red seleccionada de 140 neuronas:

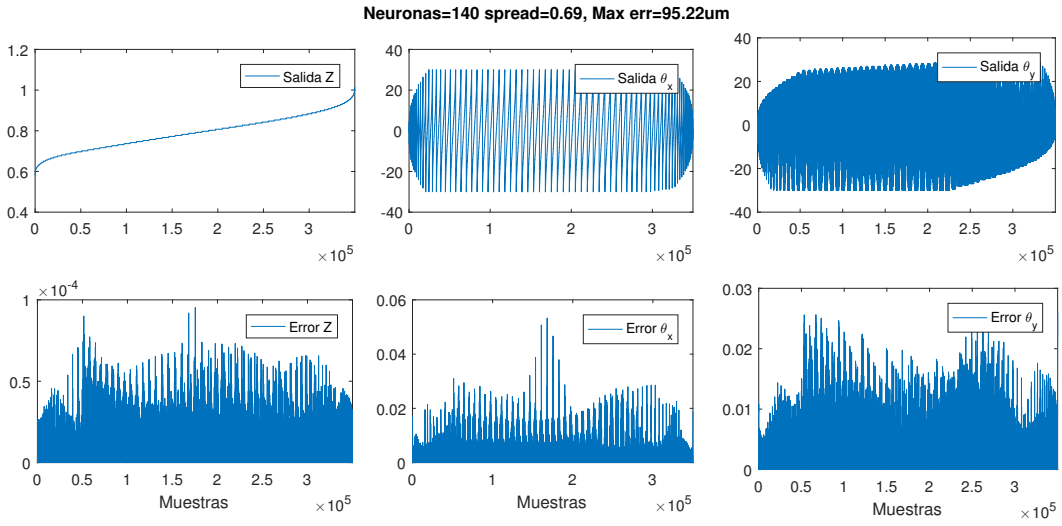



Figura 31: Salida de la red seleccionada para la configuración 1

2. Configuración red 3x2 más una red 3x1:

   En esta configuración tenemos dos redes: una encargada de resolver Z y otra encargada en resolver los dos ángulos $\theta_x$ y $\theta_y$.

   En la siguiente gráfica se muestra los valores error máximo en Z obtenidos por la red 3x1 en función del numero de neuronas:

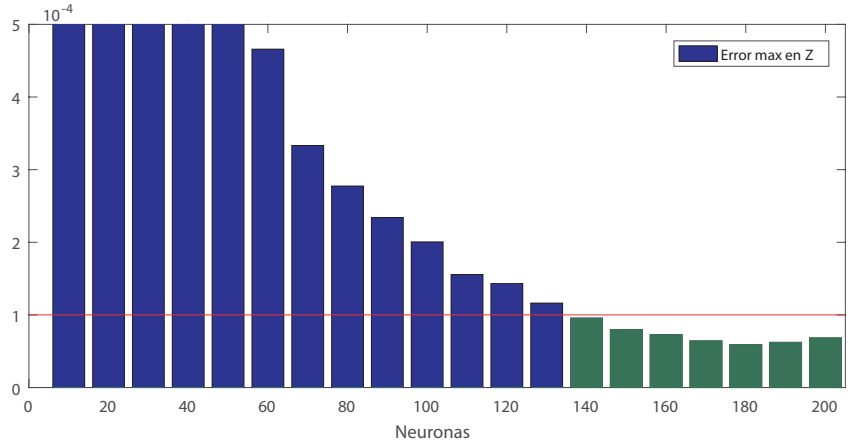



Figura 32: Error máximo en función del número de neuronas de la red 3x1 en Z

La red que cumple el objetivo de error en Z es la red 140 neuronas. La salida y el error que obtiene la red es el siguiente:

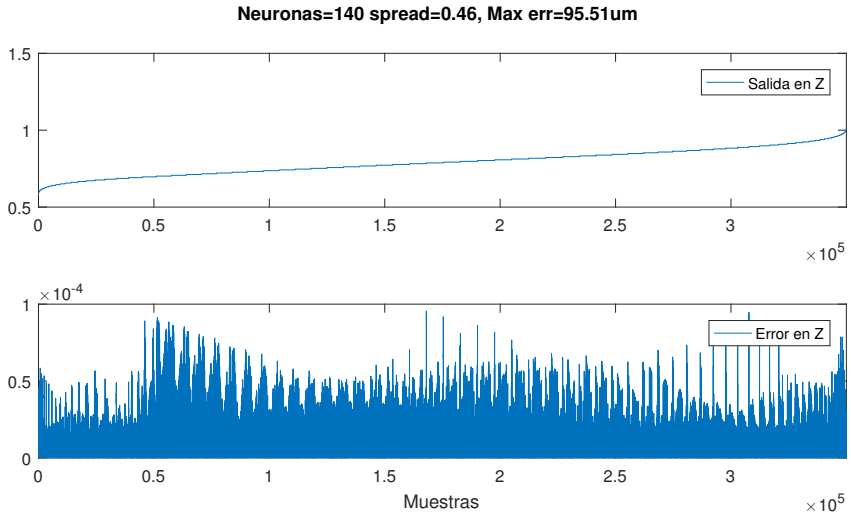



Figura 33: Salida de la red seleccionada para la configuración 3x1 en Z

Para resolver los ángulos $\theta_x$ y $\theta_y$ se utilizara la red 3x2. Los resultados que obtienen se muestran en la siguiente gráfica:

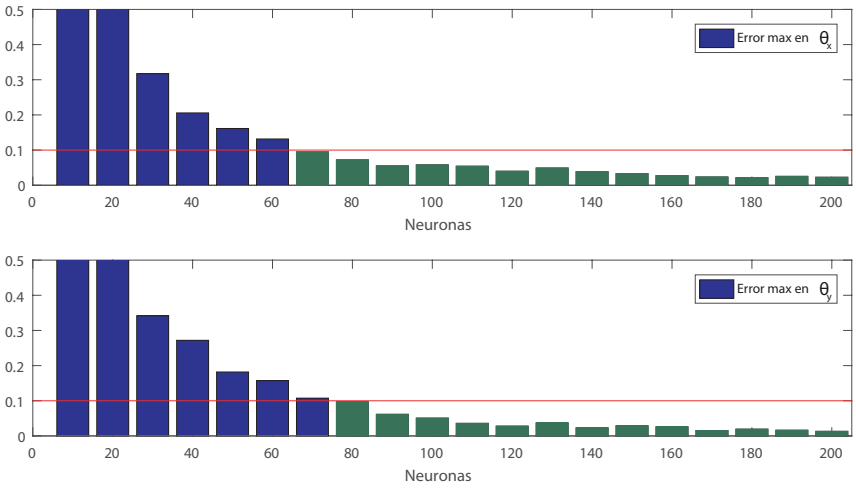



Figura 34: Error máximo en función del número de neuronas de la red 3x2 en $\theta_x$ y $\theta_y$

Se cumple el objetivo de error para los dos ángulos a partir de las 80 neuronas. La salida de la red de 80 neuronas es la siguiente:

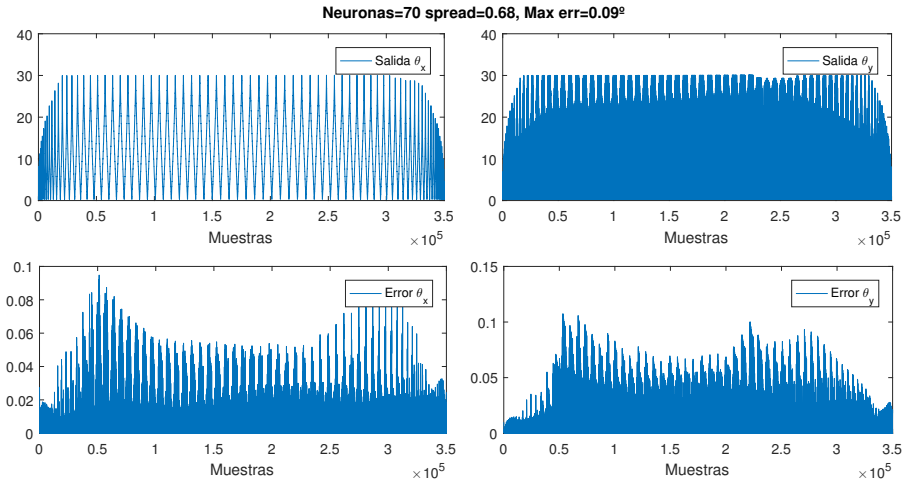



Figura 35: Salida de la red seleccionada 3x2 en $\theta_x$ y $\theta_y$

3. Configuración 3 redes 3x1:

   En esta configuración se resolverá cada dimensión de salida por separado.

   La solución de la dimensión Z esta ya resuelta por la red 3x1 de la configuración anterior.

   Los resultados que se obtiene en $\theta_x$ con una red 3x1 son los siguientes:

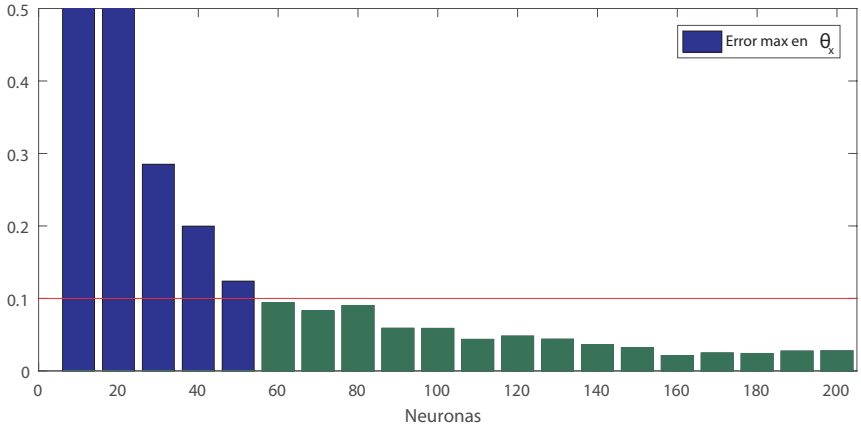



Figura 36: Error máximo en función del número de neuronas de la red 3x1 en $\theta_x$

Se cumple el objetivo de error para $\theta_x$ con 60 neuronas, la salida de esta red se muestra en la siguiente imagen:

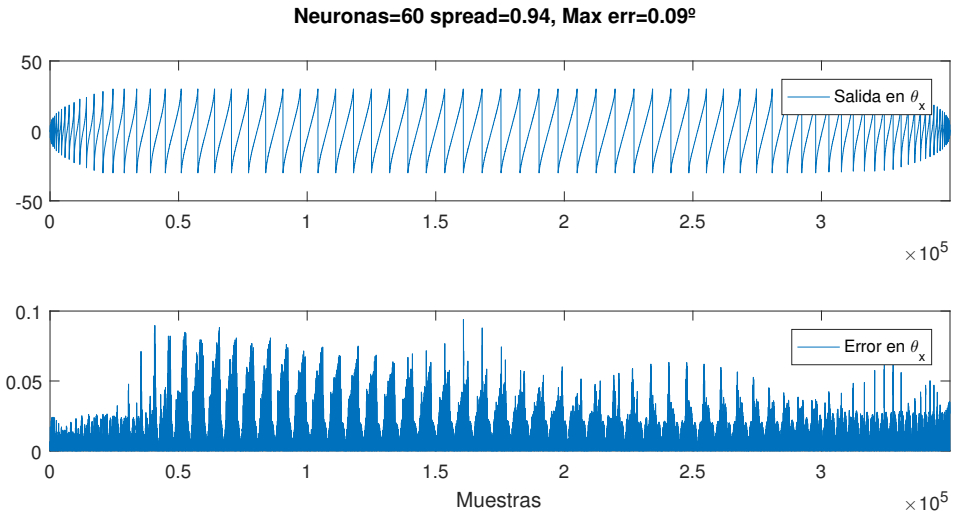



Figura 37: Salida de la red seleccionada 3x1 en $\theta_x$

Los resultados que se obtiene en $\theta_y$ con una red 3x1 son los siguientes:

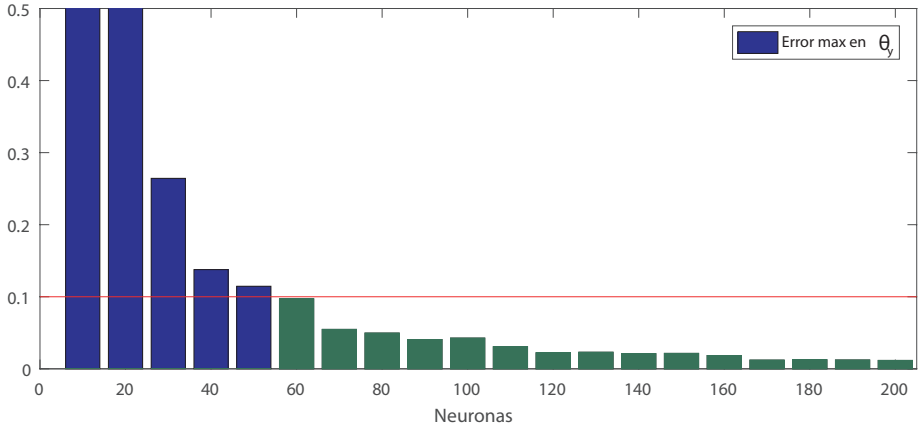



Figura 38: Error máximo en función del número de neuronas de la red 3x1 en $\theta_y$

Se cumple el objetivo de error para $\theta_y$ con 60 neuronas, la salida de esta red se muestra en la siguiente imagen:

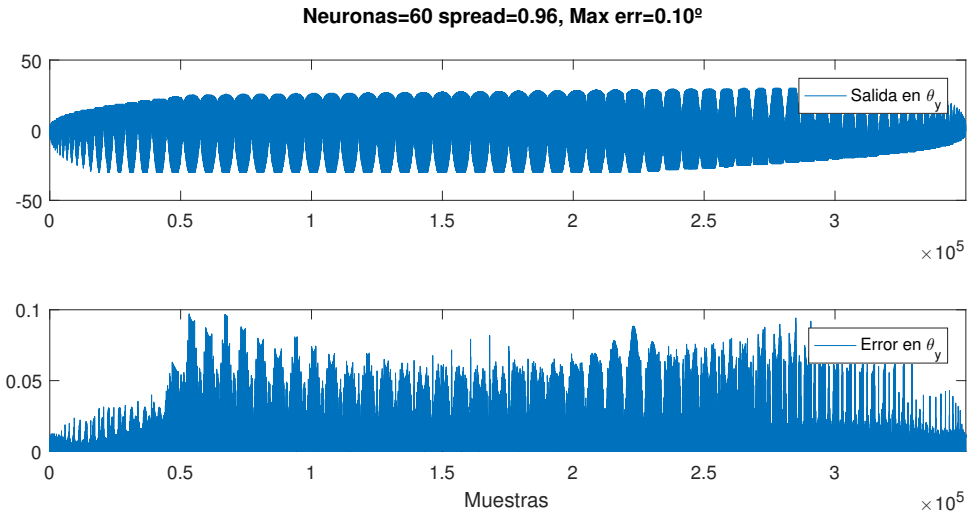



Figura 39: Salida de la red seleccionada 3x1 en $\theta_y$

**Análisis de los resultados**

En todas las configuraciones se consigue cumplir los objetivos de error con un número de neuronas relativamente bajo. Para poder determinar la mejor configuración se comparará el coste computacional que requiere cada una de ellas. Para ello se calculará el número de neuronas requeridas por cada configuración y con ello el número de operaciones matemáticas.

El número de neuronas requeridas para cumplir con los objetivos de error son las siguientes:

1. Configuración red 3x3: total de 140 neuronas.
2. Configuración red 3x1 más 3x2: total de 220 neuronas.
3. Configuración 3 redes 3x1: total de 260 neuronas.

El número de operaciones requeridas por cada configuración son las siguientes:

| | **Conf. 1** | **Conf. 2** | **Conf. 3** |
|---|---|---|---|
| ***Nº Multiplicaciones*** | 980 | 1180 | 1300 |
| ***Nº Sumas*** | 1120 | 1400 | 1560 |
| ***Nº Raíces*** | 140 | 220 | 260 |
| ***Nº Pasos por función*** | 140 | 220 | 260 |

Tabla 2: Número de operaciones matemáticas en cada configuración

Observando la tabla anterior se puede deducir que la configuración con menor coste

computacional es la 1º. Esto se debe a que el mayor coste proviene del calculo de la distancia euclídea que se debe realizar en cada neurona, y tanto por el calculo debido a las múltiples dimensiones de salida que tenga.

Por ejemplo, añadir una dimensión de salida más a una red 3x3 solo supondría una multiplicación y una suma más por neurona. En cambio, si esa dimensión adicional se resuelve con una red aparte supondría por neurona: 5 multiplicaciones, 6 sumas, una raíz y un paso por función ya que se tendría que resolver de nuevo la distancia euclídea de cada vector de entrada.

### 2.1.3. Fase 3: Implementación de la red

En esta fase se realizará la implementación de la red en la plataforma de tiempo real. Para ello, en la tarea 6, se realiza un modelado en LabView de la misma. Por ultimo, en la tarea 7 se analizarán los resultados obtenidos.

#### 2.1.3.1 Tarea 6: Implementación de la RNA en LabVIEW

*Objetivo:*

El objetivo de esta práctica es implementar la seleccionada en la plataforma de tiempo real y comprobar los tiempos de calculo.

*Equipamiento:*

En esta tarea se hizo uso del dataset de validación, realizado por el Grupo de Investigación de Control Inteligente. Este dataset contiene 2.836.219 muestras, dos ordenes mayor que el utilizado en el entrenamiento.

También se hizo uso de la plataforma de tiempo real GEME 2000.

*Duración:*

La duración de esta tarea fue de 9 semanas. Se inicio en la semana 33 del proyecto y finalizo en la semana 43.

*Procedimiento:*

Para realizar el objetivo de la tarea primero de deberá entender en su totalidad el computo que realiza cada neurona de la red para dar las salidas. Una vez entendido esto se dispondrá a realizar el modelado de la red en Labview. Tras una primera implementación de la red, se procederá a la mejorar del modelo para reducir operaciones y así bajar los tiempos de ejecución. Por ultimo, se comprobarán los tiempos de ejecución de modelo en la plataforma de tiempo real GEME 2000, realizado por Asier Zubizarreta del grupo de investigación *Systems Control and integration Research Group* (GCIS).

*Resultados:*

El computo que realiza cada neurona RBF es el siguiente:

1. Se calcula la distancia euclídea entre los valores de entrada y los centros de la neurona.
2. Se multiplica por el valor de *spread* o anchura.
3. Se pasa por la función gausiana.
4. se multiplica por cada altura y se suma el offset (la suma del offset solo se realiza una vez).

En la siguiente imagen se muestra de manera gráfica este flujo de computo:

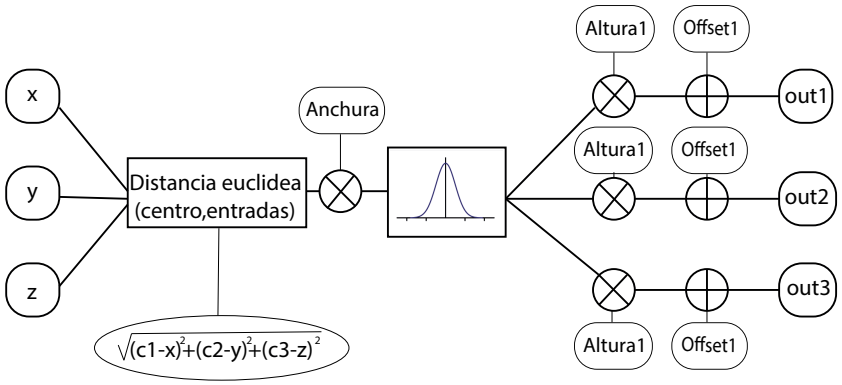



Figura 40: Esquema del computo de cada fila de neuronas

Partiendo del modelo de computo anterior se realiza el primer modelo en Labview, se muestra en la siguiente imagen:

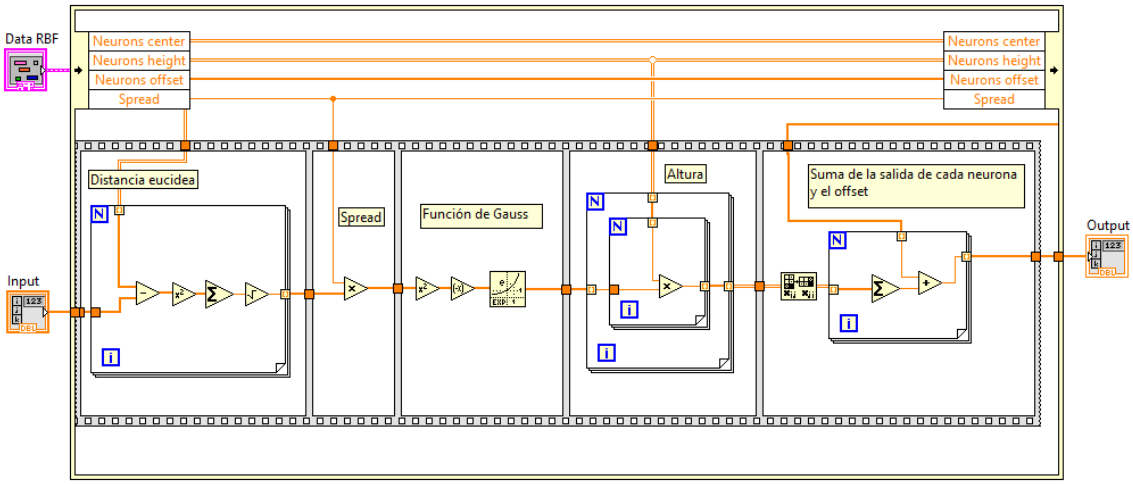



Figura 41: Modelo original en Labview

Los resultados que obtienen en la plataforma de tiempo real son los siguientes:

| | ***Tiempo medio*** ($\mu s$) | ***Tiempo máximo*** ($\mu s$) |
|---|---|---|
| **Dist. euclídea** | 127.4 | 146 |
| **Función Gauss** | 34.3 | 49 |
| **Alturas** | 590.8 | 732 |
| **Suma y offset** | 9.8 | 27 |
| **Total** | 774.1 | 921 |

Tabla 3: Tiempos de ejecución del modelo original en la plataforma T-R

Se observa que la parte de la red que más coste computacional tiene es el computo de las alturas. Esto es debido a la poca eficiencia computacional que supone el doble bucle de esta parte del modelo. Debido a esto se realiza un segundo modelo sustituyendo este doble bucle por un multiplicación matricial.

En la siguiente imagen se muestra la modificación del modelo:

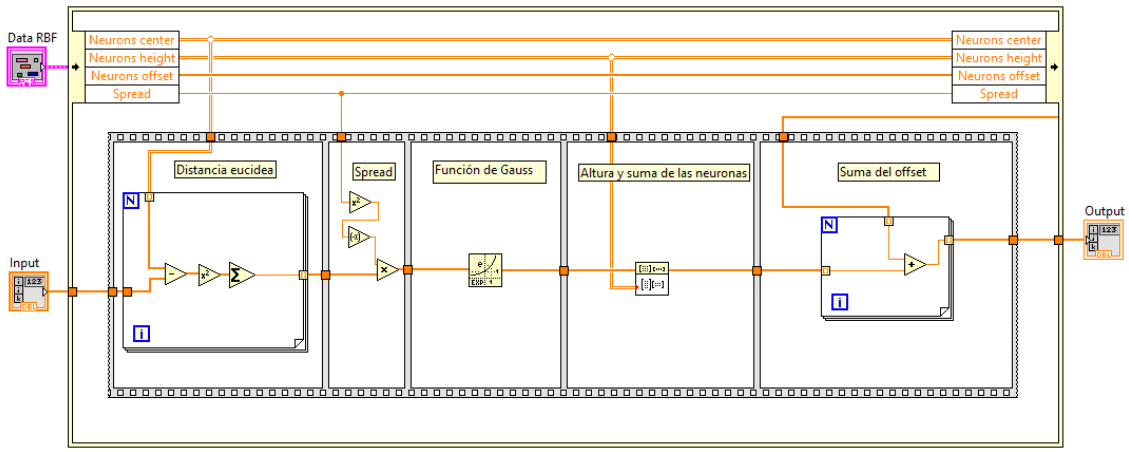



Figura 42: Modificación del modelo

A parte de la incorporación de la multiplicación matricial también se elimino las operaciones de raíz y potencia del calculo de la distancia euclídea. Con estas modificaciones los resultados que se obtienen son los siguientes:

| | ***Tiempo medio*** ($\mu s$) | ***Tiempo máximo*** ($\mu s$) |
|---|---|---|
| **Dist. euclídea** | 104.2 | 119 |
| **Función Gauss** | 29.0 | 42 |
| **Alturas** | 14.8 | 30 |
| **Suma y offset** | 3.9 | 15 |
| **Total** | 159.1 | 197 |

Tabla 4: Tiempos de ejecución del modelo modificado

Observando los tiempos de ejecución de este modelo modificado, obtenemos que la parte con mas peso computacional es el cálculo de la distancia euclídea. Se obtiene un tiempo medio de ejecución de 159.1 $\mu s$.

#### 2.1.3.2 Tarea 7: Comparación de los resultados con redes MLP

*Objetivo:*

El objetivo de esta tarea es comparar los resultados en este trabajo con los resultados descritos en el articulo *Real Time Parallel Robot Direct Kinematic Problem Computation using Neural Networks* [25] realizado por Asier Zubizarreta, Mikel Larrea, Eloy Irigoyen y Itziar Cabanes. En este articulo se presenta una solución al DKP del robot 3PRS haciendo uso de redes neuronales MLP.

*Equipamiento:*

Para realizar la comparativa de esta tarea se hizo uso del articulo *Real Time Parallel Robot Direct Kinematic Problem Computation using Neural Networks* [25].

*Duración:*

La duración de esta tarea fue de 1 semana. Se desarrolló durante la semana 43, la ultima del proyecto.

*Procedimiento:*

Para realizar esta comparativa se hará uso únicamente del dato de tiempo de ejecución, ya que es el dato que determina el rendimiento de la red que permite mejorar el tiempo de control del robot.

*Resultados:*

En el articulo [25] consiguen cumplir los objetivos de error con una red MLP de 55 neuronas. Este numero de neuronas es mucho menor que el conseguido con el tipo de red RBF en este trabajo , que es de 140 neuronas RBF. Esto se debe a la característica de localidad de las neuronas RBF, descrita en la tarea 3. Las neuronas RBF solo tienen salida en un dominio reducido dentro del espacio de la solución. Por ello para dar respuesta a todo el espacio se debe utilizar un numero de neuronas mayor.

En la siguiente imagen se muestras los tiempos medios requeridos para hacer el calculo del DKP por la red MLP, RBF y por el método iterativo de Newton-Rapson. El dato de tiempo de ejecución del método iterativo está extraído de [25].

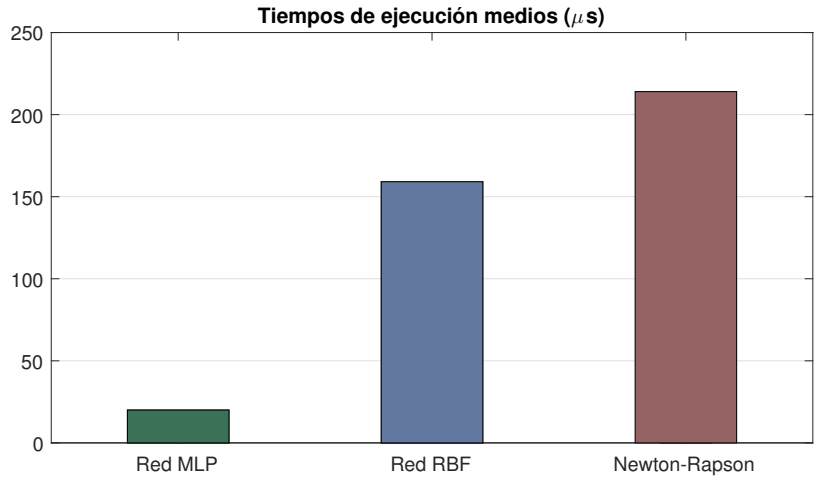



Figura 43: Comparación entre tiempos de ejecución

La red RBF desarrollada en este trabajo presenta una mejora en el tiempo de ejecución en comparación con el método iterativo Newton-Rapson, sin embargo no consigue llegar al rendimiento excepcional de las redes MLP.

## 2.2. Diagrama de Gantt

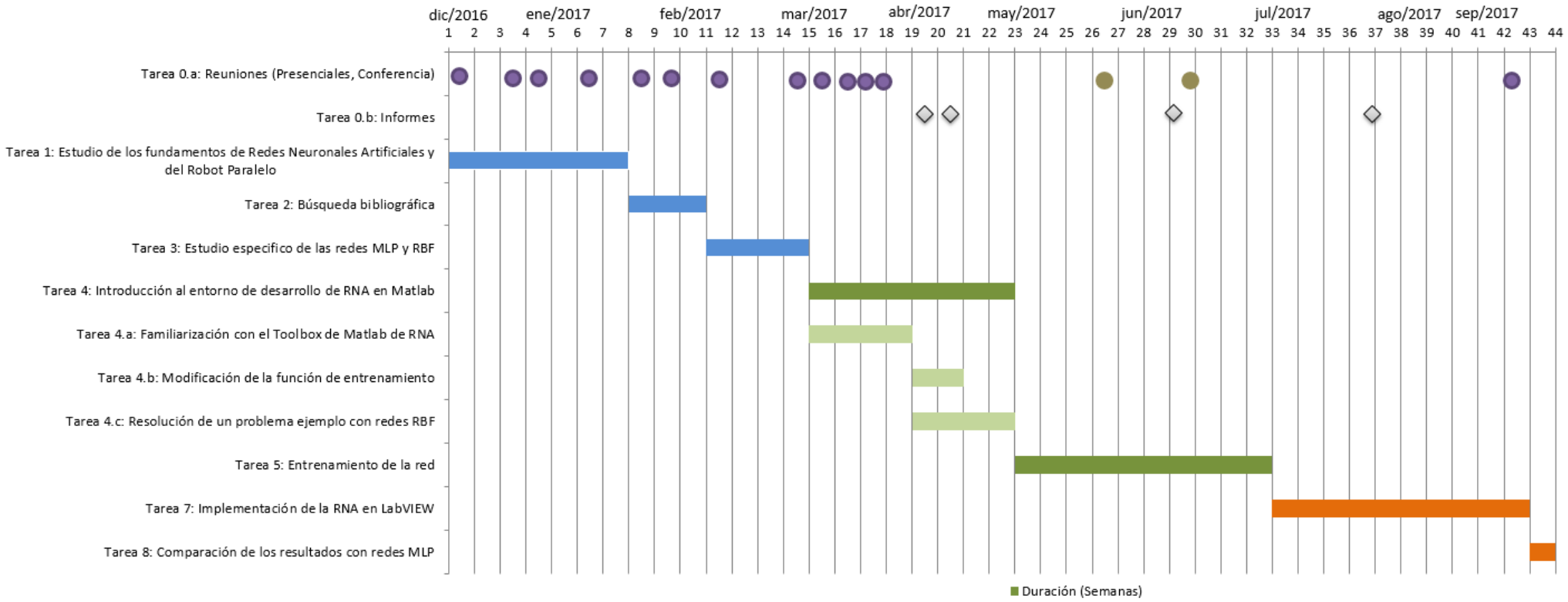



Figura 44: Diagrama de Gantt

# 3. Aspectos Económicos

## 3.1. Desglose de gastos

En el desglose de gastos se van a contabilizar dos apartados: el coste por recursos humanos y el coste por los recursos materiales. Por ultimo, se indicara el coste total del trabajo.

### 3.1.1. Recursos Humanos

En este trabajo han participado 3 personas:

- Eloy Irigoyen: Doctor en Ingeniería Industrial como Director del trabajo.
- Asier Zubizarreta: Doctor en Robótica y Sistemas de Control Automáticos como participante del trabajo.
- Alejandro Solozabal: Graduado en Ingeniería de Telecomunicación como desarrollador del trabajo.

En la siguiente tabla se muestra el desglose por horas imputadas en cada tarea:

| | ***A.Solozabal*** | ***E.Irigoyen*** | ***A.Zubizarreta*** |
|---|---|---|---|
| **Tarea 0.a** | 15 | 15 | - |
| **Tarea 0.b** | 16 | 2 | - |
| **Tarea 1** | 84 | 4 | - |
| **Tarea 2** | 36 | 4 | - |
| **Tarea 3** | 48 | 2 | - |
| **Tarea 4** | 96 | 4 | - |
| **Tarea 5** | 120 | 2 | 2 |
| **Tarea 7** | 180 | 2 | 6 |
| **Tarea 8** | 12 | 2 | - |
| **Total** | **607** | **37** | **8** |

Tabla 5: Desglose de horas por tarea

En la siguiente tabla se muestra el coste total imputado al trabajo por cada participante:

| | | *€/hora* | *Total de horas* | *Coste total* |
|---|---|---|---|---|
| **Ingeniero Senior** | **E.I.** Director | 60 | 37 | 2.220 € |
| | **A.Z.** Participante | 60 | 8 | 480 € |
| **Ingeniero Junior** | **A.S.** Desarrollador | 30 | 607 | 18.210 € |
| | | | **Total** | **20.910 €** |

Tabla 6: Coste en RR.HH.

El coste total imputable al trabajo en RR.HH. es de 20.910 €.

### 3.1.2. Recursos Materiales

En este apartado se va a contabilizar el coste de los recursos materiales utilizados en el proyecto. En la siguiente tabla se muestra los recursos fungible que fueron utilizados:

| | *Coste €* |
|---|---|
| **Libro de introducción a las RNA** | 146,35 |
| **Material de oficina** | 20 |
| **Total** | 164.35 € |

Tabla 7: Material Fungible

La siguiente tabla muestra el coste de los recursos materiales amortizables que se hicieron uso en el trabajo:

| | *Precio* | *Periodo de uso* | *Tiempo de vida* | *Coste imputable* |
|---|---|---|---|---|
| **Matlab** | 900 | 9 meses | 60 meses | 135 € |
| **GEME 2000** | 2000 | 1 mes | 60 meses | 33,3 € |
| **PCs Cluster** | 1500 | 3 meses | 60 meses | 75 € |
| | | | **Total** | 243,3 € |

Tabla 8: Coste de los recursos amortizables

### 3.1.3. Coste total

En la siguiente tabla se muestra el coste total del desarrollo del trabajo:

| | *Importe (€)* |
|---|---|
| **Recursos Humanos** | 20.910 € |
| **Recursos Materiales Amortizables** | 243,3 € |
| **Recursos Materiales Fungibles** | 164,35 € |
| **Base imponible** | 21.318 € |
| **I.V.A. (21 %)** | 4476,70 € |
| **Coste Total** | 25.794 € |

Tabla 9: Coste total del proyecto

EL coste total imputable al trabajo es de 25.794 €.

# 4. Conclusiones

Con la elaboración de este trabajo se han podido cumplir los objetivos presentados en la memoria, que era el de desarrollar una red neuronal RBF capaz de resolver el DKP del robot paralelo 3PRS cumpliendo los objetivos de error propuestos y mejorando el tiempo de ejecución del método iterativo Newton-Rapson.

Gracias a la comparativa teórica y practica realizada en el trabajo entre las redes RBF y las redes MLP se ha podido observar las ventajas y deficiencias de cada una. Las redes RBF presenta una mayor facilidad en el entrenamiento permitiendo llegar a la solución mas rápido que las redes MLP. Por el contrario debido a la localidad de las neuronas RBF se ha de hacer uso de un número de neuronas mayor que en las redes MLP, aumentado el coste computacional a la hora de ejecutar la red.

La principal desventaja que tiene las redes RBF es el calculo de la distancia euclídea. Este calculo requiere de un número de operaciones elevado por neurona , que sumado al mayor número de neuronas que requieren para obtener la solución hacen que la red sea lenta de ejecutar en comparación con las redes MLP.

# Bibliografía


[1] *Neural Network Solutions of Forward Kinematics for 3RPS Parallel Manipulator*, 2013.

[2] H. S. Bidokhti and J. Enferadi. Direct kinematics solution of 3-rrr robot by using two different artificial neural networks. In *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pages 606–611, Oct 2015.

[3] Xiang Cheng, Y. M. Huang, Z. M. Fan, and J. H. Su. Workspace generation of the 3-prs parallel robot based on the nn. In *Proceedings. International Conference on Machine Learning and Cybernetics*, volume 4, pages 2087–2089 vol.4, Nov 2002.

[4] M. Dehghani, M. Ahmadi, A. Khayatian, M. Eghtesad, and M. Farid. Neural network solution for forward kinematics problem of hexa parallel robot. In *2008 American Control Conference*, pages 4214–4219, June 2008.

[5] Ali Ghasemi, Mohammad Eghtesad, and Mehrdad Farid. Neural network solution for forward kinematics problem of cable robots. *Journal of Intelligent & Robotic Systems*, 60(2):201–215, 2010.

[6] A. Ghobakhloo and M. Eghtesad. Neural network solution for the forward kinematics problem of a redundant hydraulic shoulder. In *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, pages 6 pp.–, Nov 2005.

[7] Martin T. Hagan, Howard B. Demuth, and Mark Beale. *Neural Network Design*. PWS Publishing Co., Boston, MA, USA, 1996.

[8] Iman Kardan and Alireza Akbarzadeh. An improved hybrid method for forward kinematics analysis of parallel robots. *Advanced Robotics*, 29(6):401–411, 2015.

[9] Raşit Köker, Tarık Çakar, and Yavuz Sari. A neural-network committee machine approach to the inverse kinematics problem solution of robotic manipulators. *Engineering with Computers*, 30(4):641–649, 2014.

[10] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[11] Ben Kröse and Patrick van der Smagt. *An introduction to Neural Networks*. 1993.

[12] Temei Li, Qingguo Li, and S. Payendeh. Nn-based solution of forward kinematics of 3dof parallel spherical manipulator. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1344–1349, Aug 2005.

[13] Guanyang Liu, Yan Wang, Yuru Zhang, and Zheng Xie. Real-time solution of the forward kinematics for a parallel haptic device using a numerical approach based on neural networks. *Journal of Mechanical Science and Technology*, 29(6):2487–2499, 2015.

[14] J. P. Merlet. *Parallel Robots*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[15] Arash Rahmani, Ahmad Ghanbari, and Mehran Mahboubkhah. Direct kinematics solution of 2-(6ups) hybrid manipulator based on neural network. *Journal homepages: http://www. jweet. science-line. com*, 4(1):21–28, 2015.

[16] Arash Rahmani and Ahmand Ghanbari. Application of neural network training in forward kinematics simulation for a novel modular hybrid manipulator with experimental validation. *Intelligent Service Robotics*, 9(1):79–91, 2016.

[17] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1996.

[18] H Sadjadian, HD Taghirad, and A Fatehi. Neural networks approaches for computing the forward kinematics of a redundant parallel manipulator. *International Journal of Computational Intelligence*, 2(1):40–47, 2005.

[19] Oscar Salgado. *Síntesis, Análisis y Diseño de Manipuladores Paralelos de Baja Movilidad*. PhD thesis, ETSI Bilbao, 2008.

[20] Valentin Schmidt, Bertram Müller, and Andreas Pott. *Solving the Forward Kinematics of Cable-Driven Parallel Robots with Neural Networks and Interval Arithmetic*, pages 103–110. Springer Netherlands, Dordrecht, 2014.

[21] Andrés Vivas. Robótica paralela aplicaciones industriales, modelado y control. *Universidad del Cauca*, 2007.

[22] Qingsong Xu and Yangmin Li. *A 3-PRS Parallel Manipulator Control Based on Neural Network*, pages 757–766. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[23] T. Yuan and Y. Feng. A new algorithm for solving inverse kinematics of robot based on bp and rbf neural network. In *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, pages 418–421, Sept 2014.

[24] Dan Zhang and Jianhe Lei. Kinematic analysis of a novel 3-dof actuation redundant parallel manipulator using artificial intelligence approach. *Robotics and Computer-Integrated Manufacturing*, 27(1):157 – 163, 2011.

[25] Asier Zubizarreta, Mikel Larrea, Eloy Irigoyen, and Itziar Cabanes. *Real Time Parallel Robot Direct Kinematic Problem Computation Using Neural Networks*, pages 285–295. Springer International Publishing, Cham, 2015.