UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO DEL SISTEMA DE CONTROL DE UN MOTOR DE C.C. MEDIANTE REDES NEURONALES

AUTOR: LAURA LLOPIS IBOR

TUTORA: MARINA VALLÉS MIQUEL

COTUTOR: JOSÉ LUÍS NAVARRO HERRERO

Curso académico: 2017-2018

Agradecimientos

A mi tutor, por su valiosa guía,
a mi compañero, por su incalculable apoyo
y a mis padres, por insistirme.

Resumen

En este proyecto se plantea la aplicación de las redes neuronales para diseñar el sistema de control de un motor de corriente continua, estudiando los resultados y comparándolos con otros métodos de control tradicional y de lógica borrosa.

La posibilidad de entrenar las redes neuronales permite diseñar un control inteligente que incluya información de las zonas no lineales del sistema, lo que resultaría imposible utilizando controles tradicionales.

Durante el estudio se ha observado un bajo nivel de ruido y oscilaciones suaves, siendo por tanto un control poco agresivo para el motor. A pesar de la naturaleza adaptativa de las redes neuronales, estas carecen de robustez frente a cambios bruscos del modelo a controlar, requiriéndose un nuevo entrenamiento para eliminar el error de posición. Así mismo, se plantean una lista de alternativas de controles basados en las redes neuronales a partir de las conclusiones sacadas.

La metodología a seguir ha sido la creación de una interfaz gráfica utilizando el programa MATLAB, aprovechado las *toolboxes* de manejo de redes neuronales que facilita, con el fin de visualizar y configurar los datos recogidos del motor de corriente continua para diseñar su sistema de control. Posteriormente se ha contrastado con los métodos de control Proporcional-Integral-Derivativo y lógica borrosa.

Resum

En este projecte es planteja l'aplicació de les xarxes neuronals per a dissenyar el sistema de control d'un motor de corrent contínua, estudiant els resultats i comparant-los amb altres mètodes de control tradicionali lógica borrosa.

La possibilitat d'entrenar les xarxes neuronals permet dissenyar un control intel·ligent que incloga informació de les zones no lineals del sistema, el qual resultaria impossible utilitzant controls tradicionals.

Durant l'estudi s'ha observat un baix nivell de soroll i oscil·lacions suaus, sent per tant un control poc agressiu per al motor. A pesar de la naturalesa adaptativa de les xarxes neuronals, estes no tenen robustesa enfront canvis bruscos del model a controlar, requerint-se un nou entrenament per a eliminar l'error de posició.

Així mateix, es plantegen una llista d'alternatives de controls basats en les xarxes neuronals a partir de les conclusions tretes.

La metodologia a seguir ha sigut la creació d'una interfície gràfica utilitzant el programari MATLAB, aprofitant les *toolboxes* de maneig de xarxes neuronals que facilita, a fi de visualitzar i configurar les dades arreplegades del motor de corrent continu per a dissenyar el seu sistema de control. Posteriorment s'ha contrastat amb els mètodes de control Proporcional-Integral-Derivatiu i lògica borrosa.

Abstract

In this project, the application of neural networks to design a control system for a DC motor has been approached, studying the results and comparing them with other traditional control methods and fuzzy logic.

The possibility of training neural networks allows for the design of an intelligent control method that includes information of the non-linear zones of the system, which would prove impossible otherwise if more traditional controls were used.

During the study period, low noise level and soft oscillations have been observed, resulting in a less aggressive control for the motor. Despite the adaptive nature of neural networks, these lack sturdiness against abrupt model changes, making the requirement of a new training to eliminate the position error a must.

Likewise, a list of neural network-based control alternatives are proposed based on the conclusions acquired.

The following methodology has been the creation of a graphic interface using the MATLAB software. The neural network handling toolboxes the program provides have been utilized with the purpose of visualizing and configuring the data acquired from the continuous current motor to design its control system. Subsequently, the Proportional-Integral-Derivative and fuzzy logic control methods have been contrasted with others.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

MEMORIA

AUTOR: LAURA LLOPIS IBOR

TUTORA: MARINA VALLÉS MIQUEL

COTUTOR: JOSÉ LUÍS NAVARRO HERRERO

Curso académico: 2017-2018

ÍNDICE

1. Introducción

El control es una disciplina de la ingeniería que mediante el modelado de sistemas dinámicos pretende conseguir que se alcance una referencia deseada. Estos modelos son adaptaciones lineales de procesos no lineales, dando lugar a discrepancias.

Al utilizar sistemas de control tradicional, tal como controladores Proporcional-Integral-Derivativo (PID), estas discrepancias dificultan el control en las zonas no lineales, y es por tanto el objetivo de este documento el estudio de un control inteligente mediante redes neuronales.

Como apoyo en la realización de este estudio, se ha desarrollado una interfaz con la que acceder, visualizar y manipular los datos obtenidos. El control se realizará sobre un motor de corriente continua.

1.1. Motivación

La motivación principal para la elección de este proyecto ha sido la de obtener conocimientos sobre las redes neuronales, materia que no se ha estudiado durante el grado cursado, pero accesible con los conocimientos adquiridos en este, especialmente en el campo del *machine learning* y la inteligencia artificial.

Dado el interés en el área de programación impartida durante el curso y el deseo de ampliar los conocimientos sobre el control de sistemas, este proyecto se ha presentado como la oportunidad ideal para combinar ambas disciplinas.

1.2. Objetivos

El objetivo principal de este proyecto es el de realizar el diseño del control mediante redes neuronales de un motor de corriente continua. Se pretende estudiar los resultados obtenidos y compararlos con otros tipos de control para evaluar el éxito de las redes neuronales y sus posibles aplicaciones.

1.3. Metodología

Para conseguir los objetivos propuestos, ha sido imprescindible un estudio previo sobre las redes neuronales, el cual se incluye en el siguiente apartado como base teórica de este proyecto.

La herramienta empleada para diseñar el sistema de control y el manejo de las redes es el software MATLAB, por sus *toolboxes* y librerías preparadas para el manejo de redes neuronales y funciones de transferencia, y porque permite al usuario diseñar una interfaz donde visualizar los resultados y configurar de parámetros de diseño.

2. Antecedentes

2.1. Conocimiento teórico previo

Las redes neuronales están basadas en las redes neuronales biológicas, son un sistema computacional formado por elementos interconectados que procesan información. Estos elementos reciben el nombre de neuronas, y están organizados en capas comunicadas entre sí. Cada neurona está compuesta (Fig. 1) por:

- **Entradas:** Es la información que se procesa en la neurona. Según las conexiones entre las capas, esta información puede ser externa o realimentada de la salida del sistema, adaptándose a los procesos dinámicos.
- **Pesos:** Los pesos son valores reales, positivos o negativos, que multiplican las entradas de las neuronas. Estos valores son aleatorios previamente a su entrenamiento, cambiando para obtener la salida deseada.
- **Función de transferencia:** Combina el producto de cada entrada con su peso, cuyo resultado será procesado por la función de activación.
- **Función de activación:** Es una función que devuelve la salida de la neurona. Según el objetivo de la red neuronal, esta puede ser de tipo discreto (verdadero o falso) o continuo.

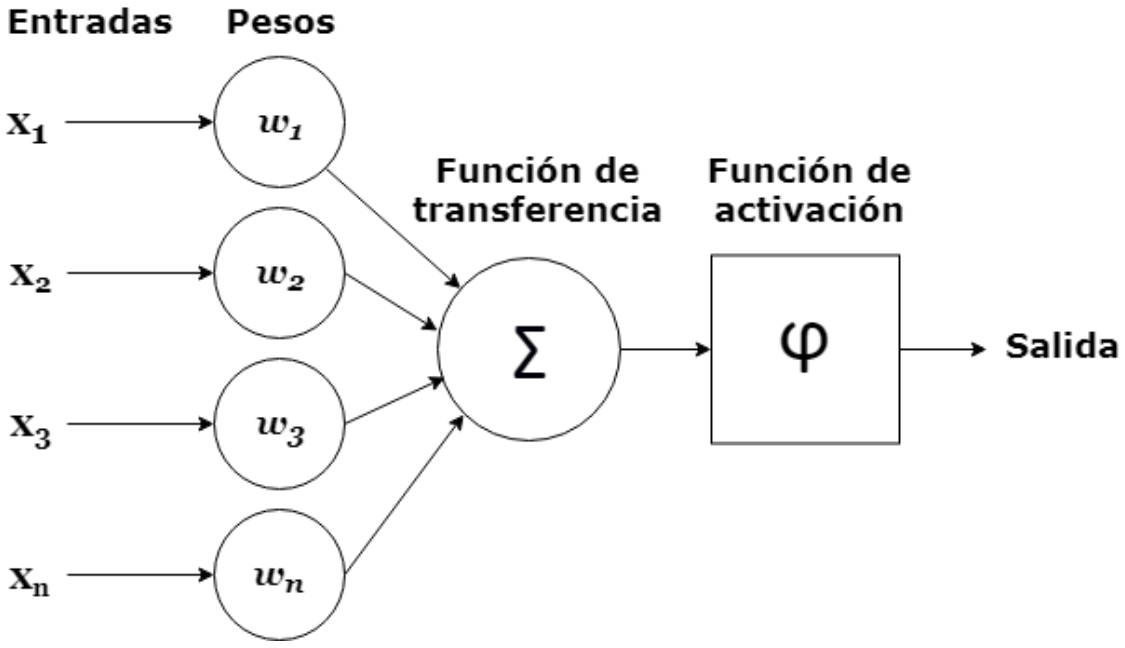

Fig. 1: Estructura interna de una neurona artificial, donde la información de entrada se multiplica por los pesos, este producto se combina mediante la función de transferencia, y se calcula la salida a través de la función de activación.

Cada una de las capas en las que se organizan las neuronas está enlazada con otras capas, formando la red neuronal y clasificándose según su posición en esta: capa de entrada, capa oculta o capa de salida.

Las redes neuronales se someten a un aprendizaje durante el cual se modifican los valores de los pesos reiteradamente hasta que se minimice el error respecto a la salida deseada o se cumpla

una de las condiciones de parada definidas previamente. Este entrenamiento permite hacer uso de las redes neuronales en el ámbito de la agrupación, regresión, previsión y clasificación.

En este proyecto concretamente, se utilizan las redes neuronales para que el control sobre las zonas no lineales sea más preciso que un control tradicional, ya que el entrenamiento al que se las somete permite incluir información de dichas zonas.

2.1. Software empleado

MATLAB es el software empleado para diseñar y evaluar los resultados del control por redes neuronales, ya que dispone de herramientas específicas para el entrenamiento e implementación de redes, facilidad para el procesamiento de datos y librerías relacionadas con los sistemas de control.

Las *toolboxes* son las herramientas en forma de librerías que agrupan funciones de un ámbito específico, la mayoría de las cuales son fácilmente manejables gracias a su interfaz gráfica. En este proyecto, se han empleado las siguientes *toolboxes: Graphical User Interface* (GUI), *Neural Network*, *System Identification*, *Fuzzy Logic* y *Control System*:

- **GUI [1]:** Se emplea la herramienta GUIDE para el diseño de la interfaz gráfica mediante el editor de diseño y paralelamente se genera el código que construye la interfaz, el cual se puede modificar para programar el comportamiento de la aplicación.
- ***Neural Network* [2]:** Contiene herramientas con las que crear, manejar y entrenar redes neuronales. Además, incluye la aplicación nnstart, que ayuda al usuario a elegir la estructura de la red más apropiada.
- ***System Identification* [3]:** Se emplea la herramienta IDENT para abrir la aplicación que permite la identificación de modelos lineales.
- ***Fuzzy Logic* [4]:** Es la herramienta utilizada para el control, diseño y análisis de sistemas basados en la lógica *fuzzy* o borrosa.
- ***Control System* [5]:** Contiene los recursos necesarios para el control, diseño y análisis de sistemas de control lineales.

2.2. Elementos del laboratorio

En este apartado se describen los diferentes elementos del laboratorio que se emplean en la realización de este proyecto. Se ha elegido controlar un motor de corriente continua por su disponibilidad y por tener dos no linealidades: la zona muerta y la de saturación. Además, junto a este se empleará una tarjeta de adquisición de datos (DAQ) conectada a un ordenador personal y los cables necesarios para realizar las conexiones (Fig. 2). A continuación, se analizan con más detalle las características más importantes de estas herramientas:

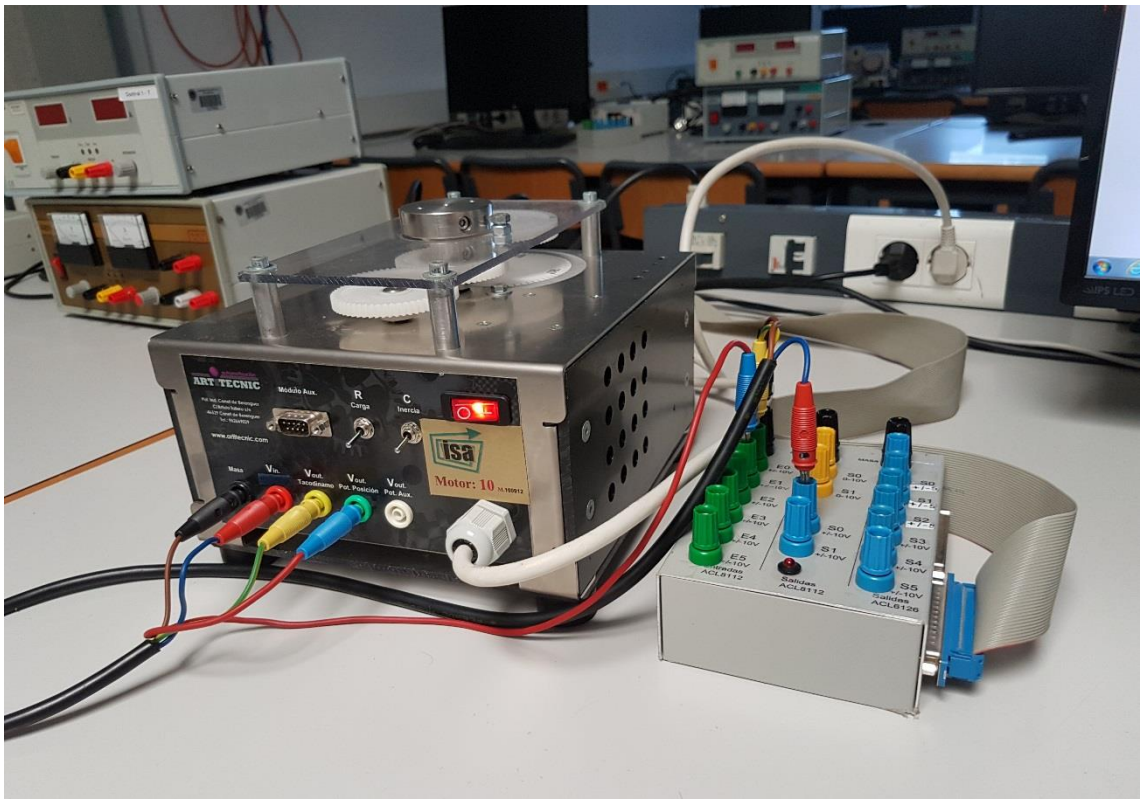

Fig. 2: Motor de corriente continua (izquierda) conectado mediante conectores banana a la DAQ (derecha).

Motor de corriente continua: Es un motor de corriente continua de marca Artitecnic con un rango de entrada de ±10 V. En el mismo modulo se incluyen sensores de posición y velocidad, cuyos rangos de salida son inferiores a ±10 V. Además, consta de dos interruptores desde los que se puede cambiar la configuración interna del motor: uno que añade resistencia adicional (R) y otro que añade inercia al motor (C), dando lugar a 4 configuraciones diferentes en función de cuáles de los interruptores estén activados. El rango de velocidad del motor medido por la tacodinamo ha sido calculador para la máxima y la mínima entrada de voltaje, resultando en ±6 V (Fig. 3).

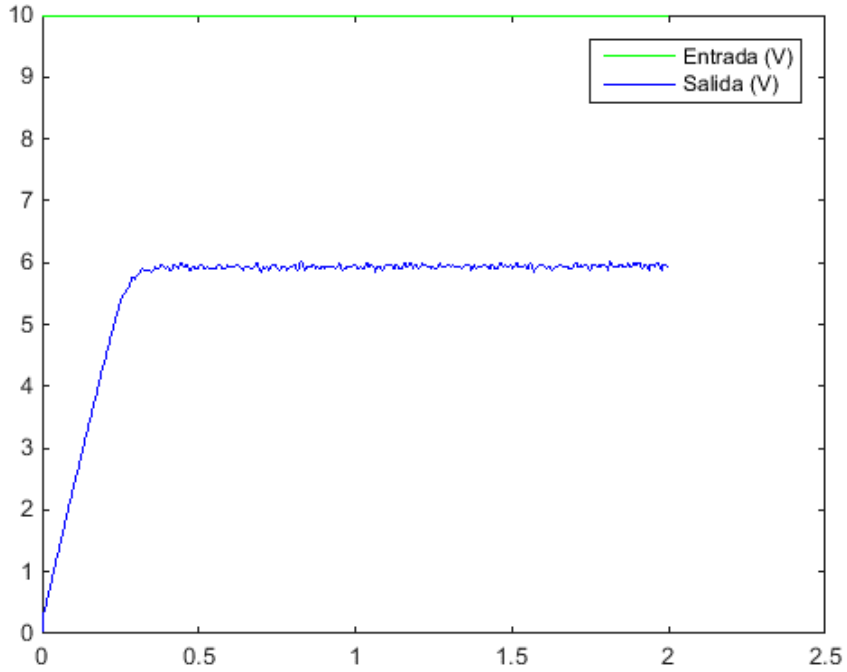

Fig. 3: Respuesta estabilizada en 6 V del motor ante el máximo escalón de entrada positivo.

Tarjeta de adquisición de datos [6]: DAQ de 12 bits y 16 canales. Dado que tiene 12 bits, la resolución de entrada cuando se leen voltajes de ±10 V viene dada por la ecuación (Ec. 1), donde 20 V es el rango de entrada y 4096 es el número de combinaciones posibles máximas para 12 bits:

$$Resolución = \frac{VinputMAX - VinputMIN}{Nº\ de\ combinaciones} = \frac{10\ V - (-10\ V)}{2^{12}\ combinaciones} = 4.88\ mV \quad (1)$$

Además, para obtener salidas bipolares se ha introducido un circuito adicional junto a la DAQ. Este circuito transforma las salidas unipolares de 0 V a 10 V en bipolares de ±10 V, para que se corresponda con la entrada del motor. Por tanto, para el cálculo del voltaje de salida, se tendrá que tener en cuenta mediante la siguiente ecuación (Ec. 2):

$$Vsalida = (Vconfigurado - 5) * 2 \quad (2)$$

Así, cuando se configuren 5 V en la salida de la DAQ, sustituyendo en la ecuación anterior, la salida real será de 0 V (Ec. 3):

$$Vsalida = (5 - 5) * 2 = 0\ V \quad (3)$$

El acceso a la DAQ se realiza mediante funciones de MATLAB, que permiten tanto la adquisición de datos como la configuración de la salida.

Para los diferentes experimentos, se ha interconectado la DAQ con el motor de corriente continua, de forma que se leen los valores de ambos sensores a través de los canales 0 y 1, y se controla la entrada del motor a partir de la salida bipolar 0 (Fig. 4). En la gran mayoría de los experimentos se ha usado el motor con ambas modificaciones desactivadas (hacia abajo), excepto para las diferentes pruebas ante cambios en el modelo donde se ha llegado a probar el comportamiento de los diferentes controles en cada una de las configuraciones.

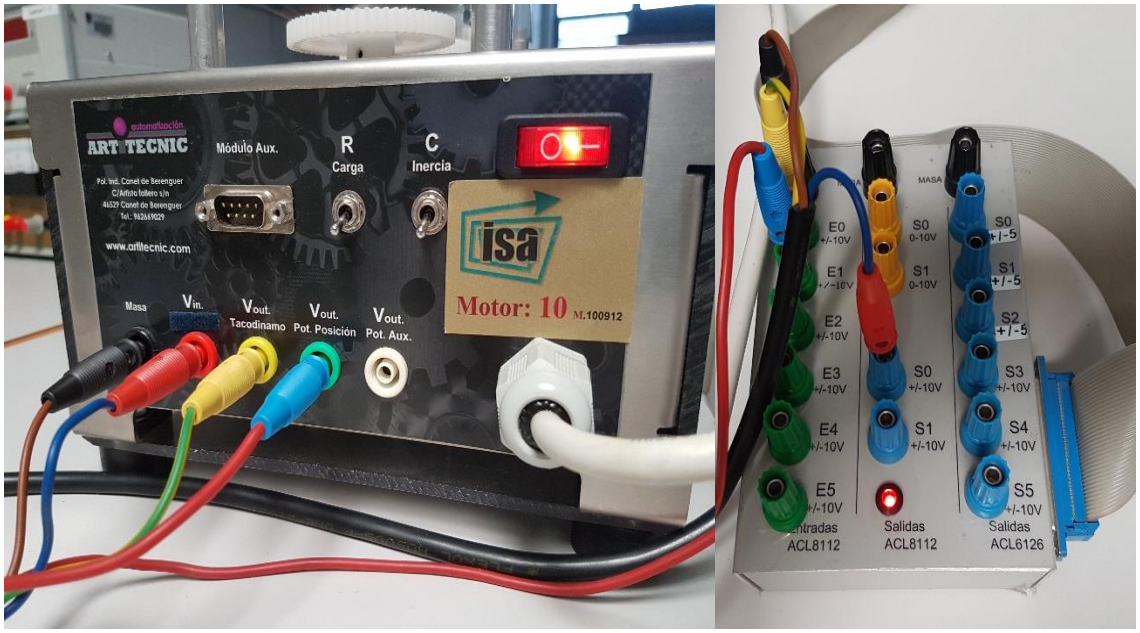

Fig. 4: Frontal del motor con los interruptores para carga e inercia adicional, el cableado que conecta con la DAQ, y los puertos utilizados en ésta (entradas analógicas y salida analógica bipolar). Se conectan Vin con la salida bipolar (S0), los sensores de velocidad y posición las entradas bipolares (E0 y E1).

3. Interfaz de usuario

Se diseña una interfaz gráfica para facilitar el estudio y el diseño del control mediante redes neuronales.

3.1. Estructura de la interfaz de usuario

Para el diseño de la interfaz de usuario se ha empleado el entorno de desarrollo de interfaces de MATLAB, GUIDE. Este entorno proporciona herramientas para diseñar interfaces de usuario mediante un editor de diseño.

Se requiere acceder a los datos de salida del motor y tener la opción de guardarlos y cargarlos, visualizar y editar las variables gráficamente y botones que ejecuten el código correspondiente a cada etapa de diseño. Por lo tanto, la estructura de la interfaz consta de una pantalla principal donde visualizar resultados, un panel de configuración y varios botones para alternar entre las diferentes opciones de la aplicación (Fig. 5).

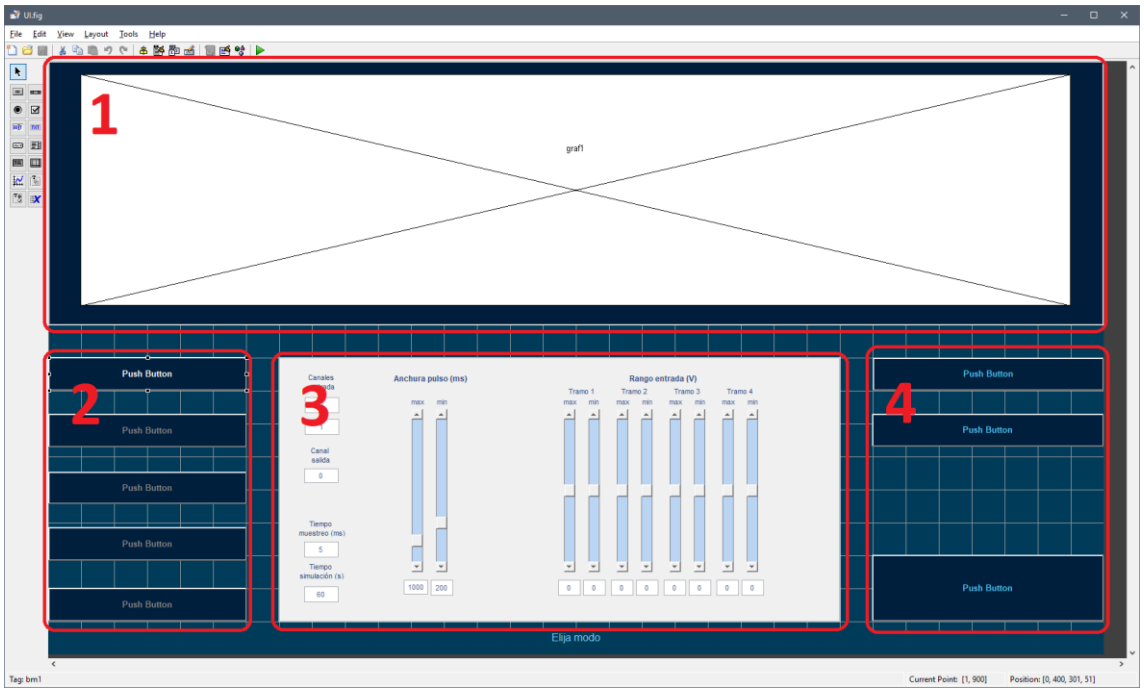

Fig. 5: Estructura de la interfaz. 1. Visualización de resultados en forma de gráficas en la pantalla principal. 2. Botones para alternar entre los diferentes modos. 3. Panel para configuración de parámetros. 4. Botones de carga, guardado y ejecución.

Se plantea que esta interfaz alterne entre diferentes modos, donde cada uno se encargue de una etapa del diseño y de la evaluación del control. Para cada modo, el panel de configuración se adaptará mostrando diferentes elementos internos.

3.2. Modos de la interfaz de usuario

Para el control del motor mediante redes neuronales se requerirá recoger datos del motor para entrenar un modelo del motor en redes neuronales, el cual será una versión digital del comportamiento del motor formado por neuronas para facilitar su control. Seguidamente se definirá un modelo de referencia, que será el comportamiento para el que se entrenará el motor. Finalmente, se entrenará la red neuronal de control a partir de los datos del modelo del motor y del modelo de referencia.

Por tanto, la interfaz de usuario contiene cinco modos diferentes, dada la necesidad de configurar cada uno individualmente. Estos modos guían al usuario y facilitan el uso de la aplicación. Su programación se detalla en el apartado posterior. Los modos a incluir son los siguientes: Datos motor, Modelo motor, Modelo referencia, Modelo control y Ejecución control.

3.2.1. Datos motor

En este modo se facilita la toma de datos para el diseño del modelo del motor, generando la entrada utilizada para recoger los valores de velocidad y posición del motor. Para ello, desde el panel (Fig. 6) se configuran los canales de entrada y salida de datos, el periodo de muestreo, el tiempo total, y mediante una serie de sliders y cajas de texto interconectadas mediante código, se eligen los rangos de entrada y anchos de pulso con la que se genera la señal transmitida al motor para generar los datos.

Los rangos de entrada están limitados para que coincidan con los del motor, al igual que la anchura de pulso para que el motor tenga tiempo de estabilizarse. La señal de entrada se divide en cuatro tramos con rangos configurables para entrenar diferentes zonas de trabajo del motor.

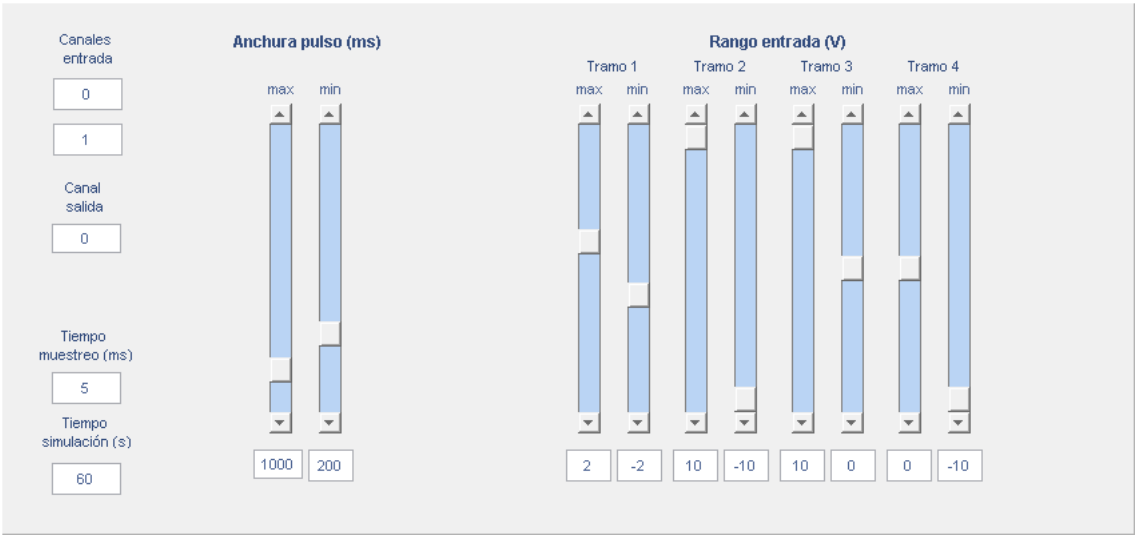

Fig. 6: Panel de configuración en el modo “Datos motor”, donde se eligen los canales de entrada y salida de la DAQ (izquierda), el tiempo de muestreo, el tiempo de simulación y una serie de sliders para configurar la anchura de pulso mínima y máxima y el rango de entrada.

3.2.2. Modelo motor

Este modo es el correspondiente a la generación del modelo del motor en redes neuronales o mediante función de transferencia. Para ello se dispone de una serie de sliders con unos límites programados para la configuración del modelo en red neuronal, y botones para ejecutar un asistente para identificar la función de transferencia o importar una previamente guardada de un archivo de extensión .mat (Fig. 7).

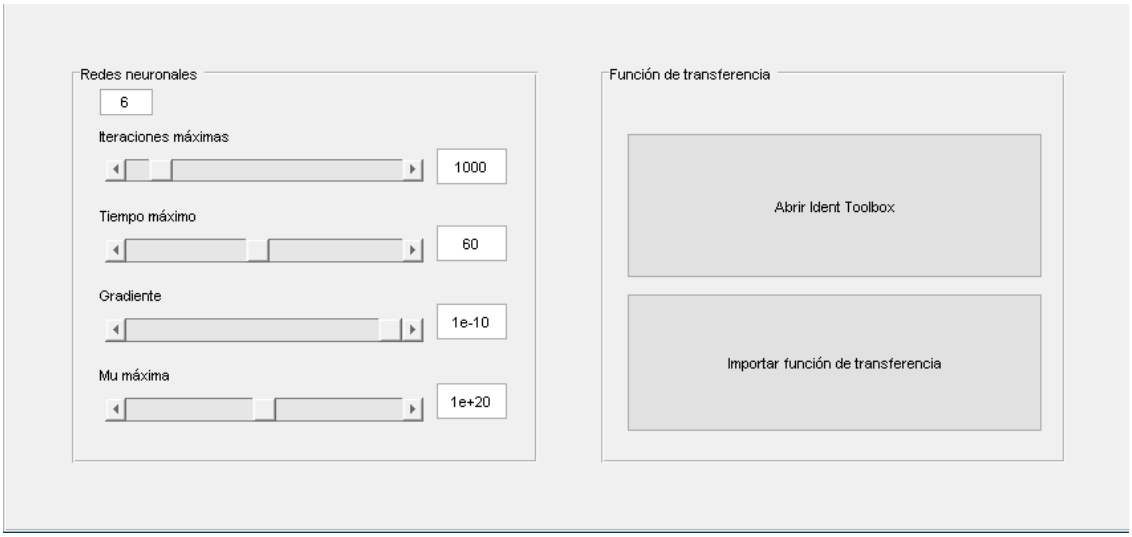

Fig. 7: Panel de configuración en el modo "Modelo motor", dividida en dos paneles secundarios: a la izquierda se configuran los parámetros del entrenamiento de la red mediante sliders y una entrada de texto para seleccionar el número de neuronas; a la derecha se elige cómo identificar la función de transferencia mediante dos botones.

3.2.3. Modelo referencia

Recoge las variables para la configuración del modelo de referencia: orden de la función de transferencia que actualiza dos líneas correspondientes al numerador y el denominador donde introducirla manualmente, así como el rango de referencia de la velocidad del motor para el entrenamiento, el cual no deberá sobrepasar el rango de velocidades del motor. En este modo se visualiza el comportamiento deseado ante una entrada de escalón y modificar las variables para adecuar el modelo a las capacidades reales del motor (Fig. 8).

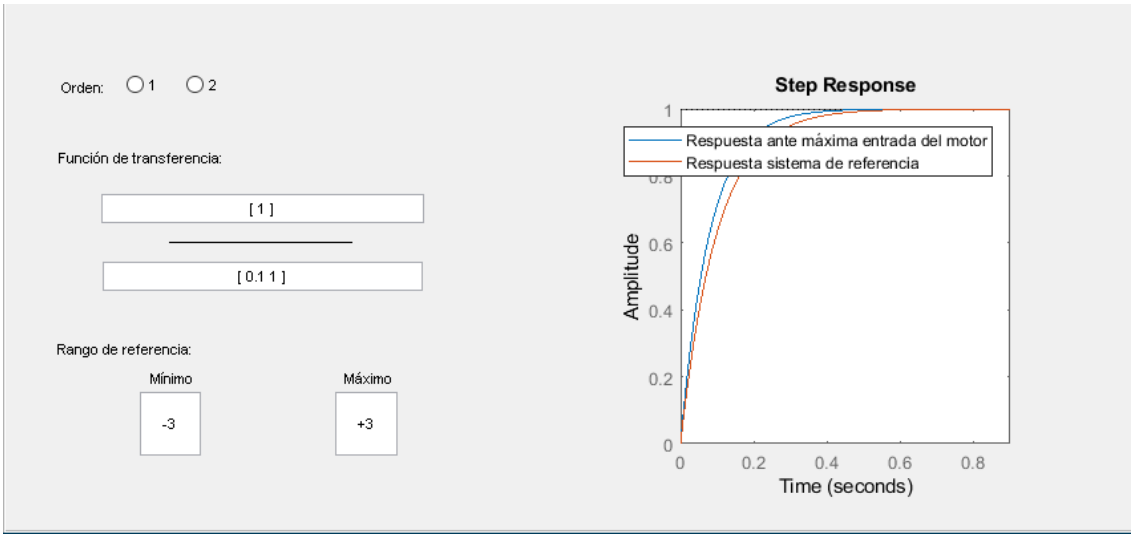

Fig. 8: Panel de configuración en el modo "Modelo referencia", a la izquierda se configura la función de transferencia mediante entradas manuales y se introduce el rango de referencia para el que se entrenará la red. A la derecha se visualiza la respuesta ante escalón y se compara con la respuesta ante entrada máxima.

3.2.4. Modelo control

Este modo se utiliza para entrenar la red neuronal de control, y por tanto incluye sliders para configurar el entrenamiento dentro de unos límites previamente fijados (Fig. 9).

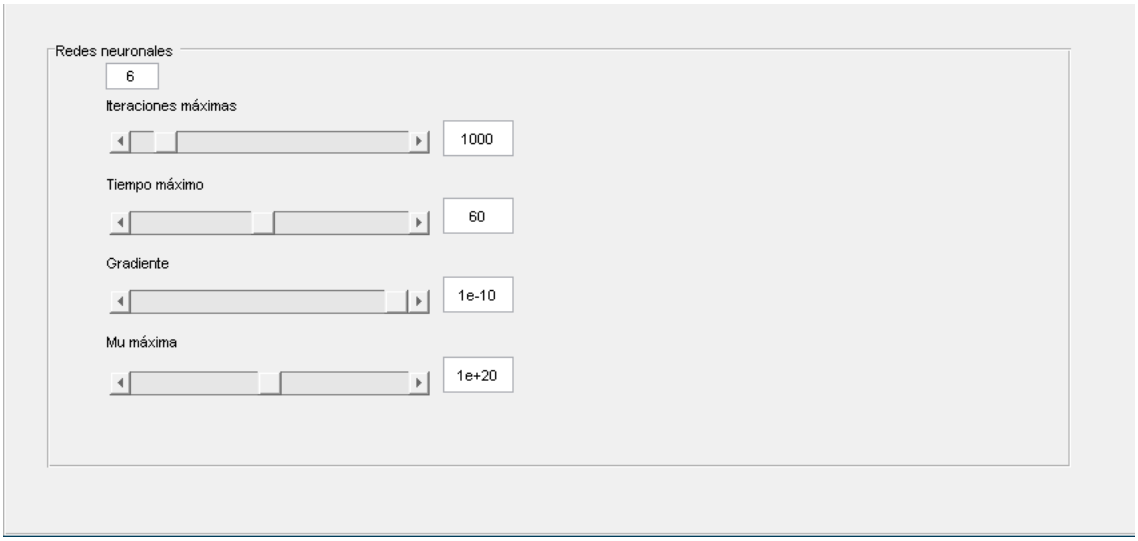

Fig. 9: Panel de configuración en el modo "Modelo control", a la izquierda se configuran los parámetros para el entrenaimeinto del modelo del motor.

3.2.5. Ejecución control

El último modo permite elegir qué tipo de control se desea probar, para poder comparar otros métodos de control como el control PID y el borroso. También permite editar estos controles y incluye opciones diferentes para la ejecución, como habilitar o no un filtro para la acción de control, o la elección de las referencias que deberá seguir el motor durante la ejecución (Fig. 10).

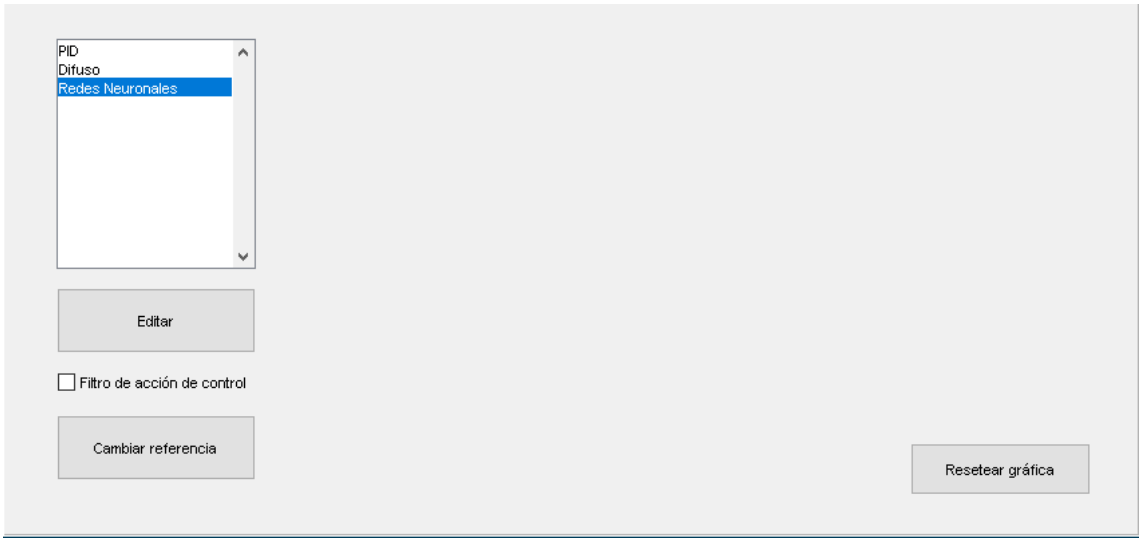

Fig. 10: Panel de configuración en el modo "Ejecución control". A la izquierda se elige el tipo de control a ejecutar, a elegir entre PID, borroso y redes neuronales. Mediante botones se puede elegir editar los dos primeros controles, cambiar la referencia o resetear la gráfica.

A continuación, se describe la función de edición del control PID y la del control borroso, junto con la del cambio de referencia para los tres tipos de controles:

Editar PID: Se pide que se edite, importe o introduzca manualmente (Fig. 11) los parámetros del control discretizado, ya sea introduciendo el valor de la ganancia (valor a) y la posición del cero (valor b) (Fig. 12), o mediante la herramienta *PID Tuner*.

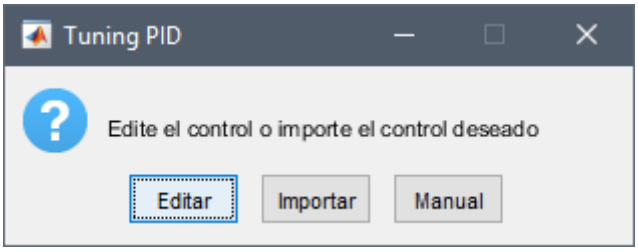

Fig. 11: Ventana de elección de modo de edición del control PID.

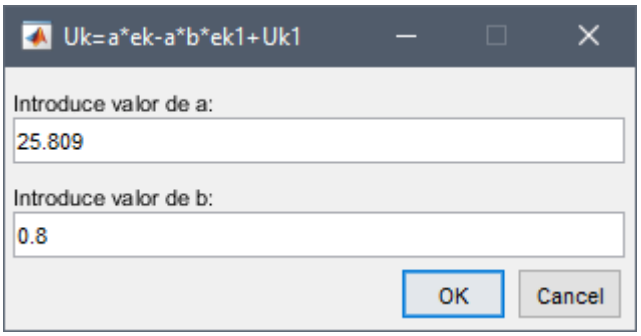

Fig. 12: Ejemplo de ventana de introducción de los valores a y b para la ganancia y posición del cero en un control PID.

Editar control borroso: Se accede a una aplicación que proporciona MATLAB para la edición del control borroso. El usuario puede editar los métodos de implicación, agregación y defuzzificación, además de las funciones de pertenencia de la entrada y salida, y las reglas de implicación (Fig. 13).

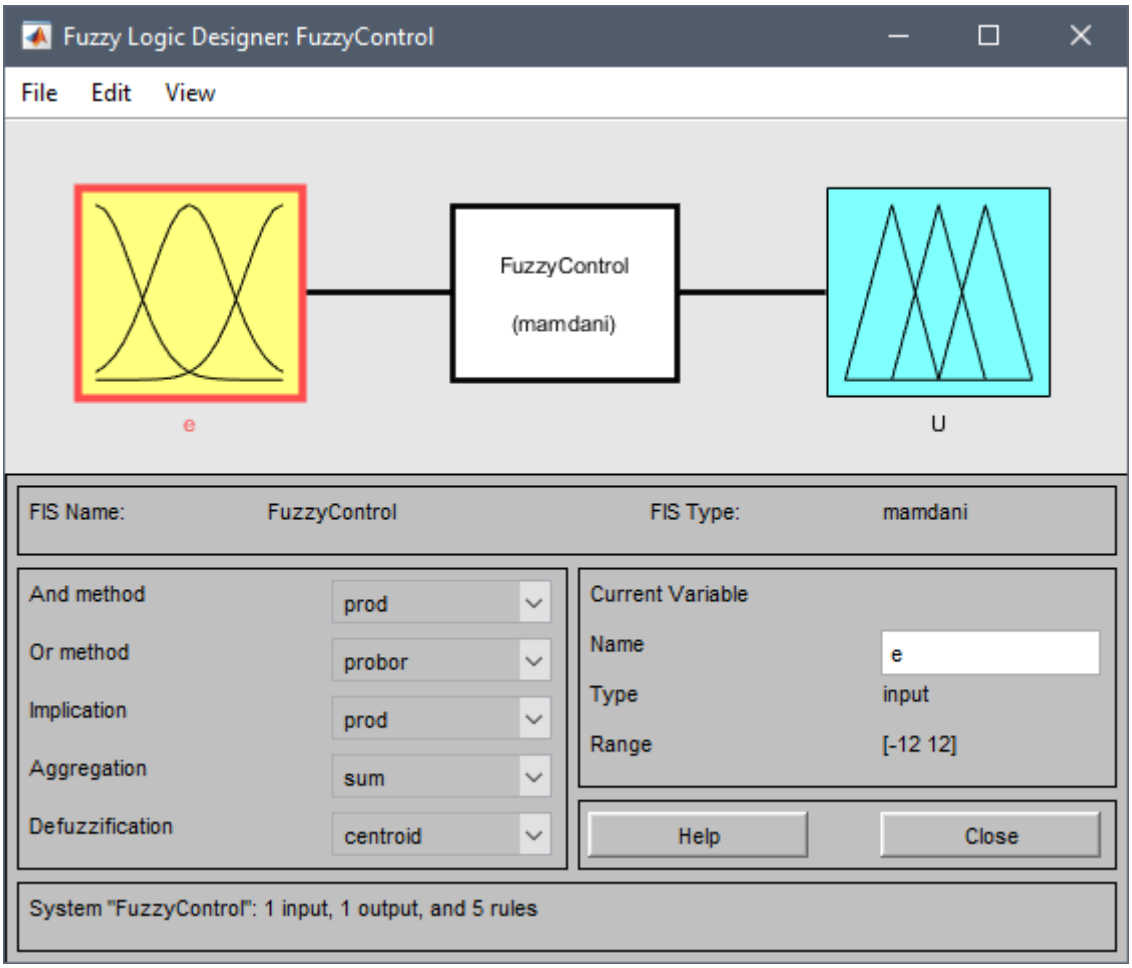

Fig. 13: Ventana de aplicación donde se edita el control borroso. Se elige el método (izquierda), la entrada, la salida y las reglas (esquema superior).

Referencia: Esta opción permite editar la referencia de cualquiera de los tres modos de control, se edita la velocidad en unidades de voltaje y el tiempo en milisegundos que se desea que cumpla el motor (Fig. 14).

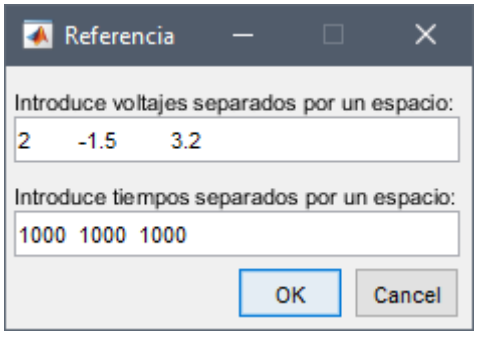

Fig. 14: Ventana de edición de la referencia, donde se introducen manualmente los valores de la velocidad (superior) y el tiempo (inferior).

3.3. Diseño de la interfaz de usuario

Para el diseño de la aplicación se ejecuta el entorno GUIDE, mediante el comando *guide* en MATLAB. Este comando carga una ventana emergente que permite elegir entre crear una UI completamente nueva, o cargar una previamente guardada (Fig. 15).

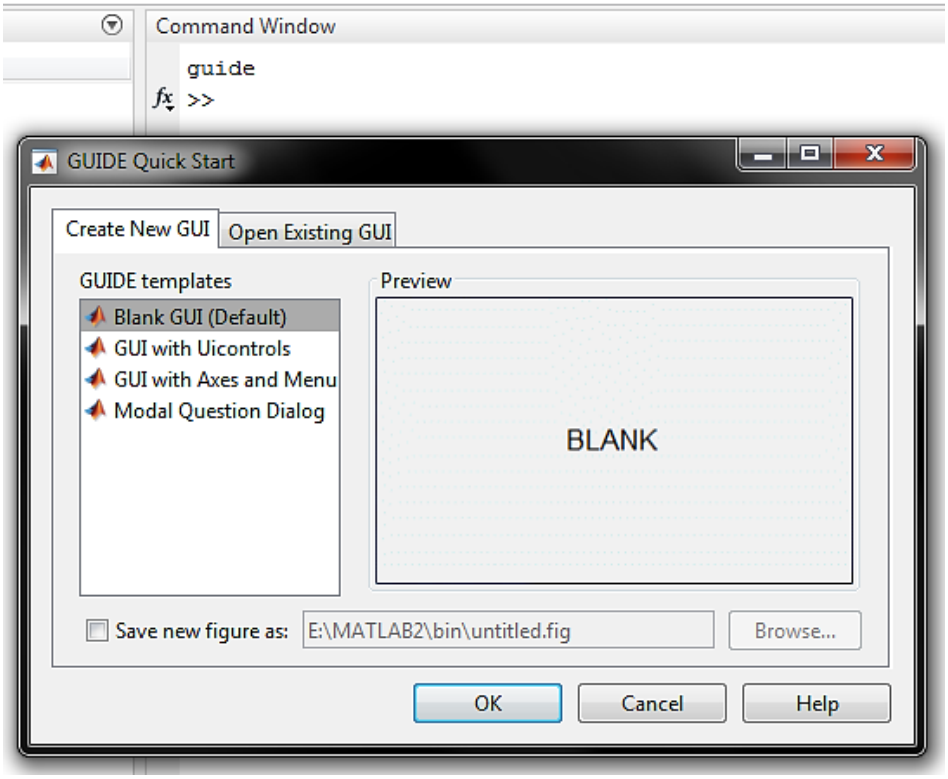

Fig. 15: Menú emergente tras la ejecución del comando *guide*, donde se selecciona una plantilla base para la interfaz.

Una vez abierto, la ventana principal de la interfaz puede ser redimensionada, de acuerdo a la resolución de la pantalla donde se vaya a utilizar. Para ello, se hace clic derecho sobre el elemento en cuestión, o desde el *Object Browser* (Fig. 16), y se accede a las propiedades del objeto. A continuación, se permite fijar la posición inicial y el tamaño de la interfaz correspondientes a la variable *Inner Position*. Se sigue este mismo procedimiento para editar cada uno de los elementos de la interfaz. Además, para facilitar la colocación de los objetos en la interfaz, se puede activar la cuadrícula desde el menú de *Tools*. Los elementos a implementar serán los siguientes:

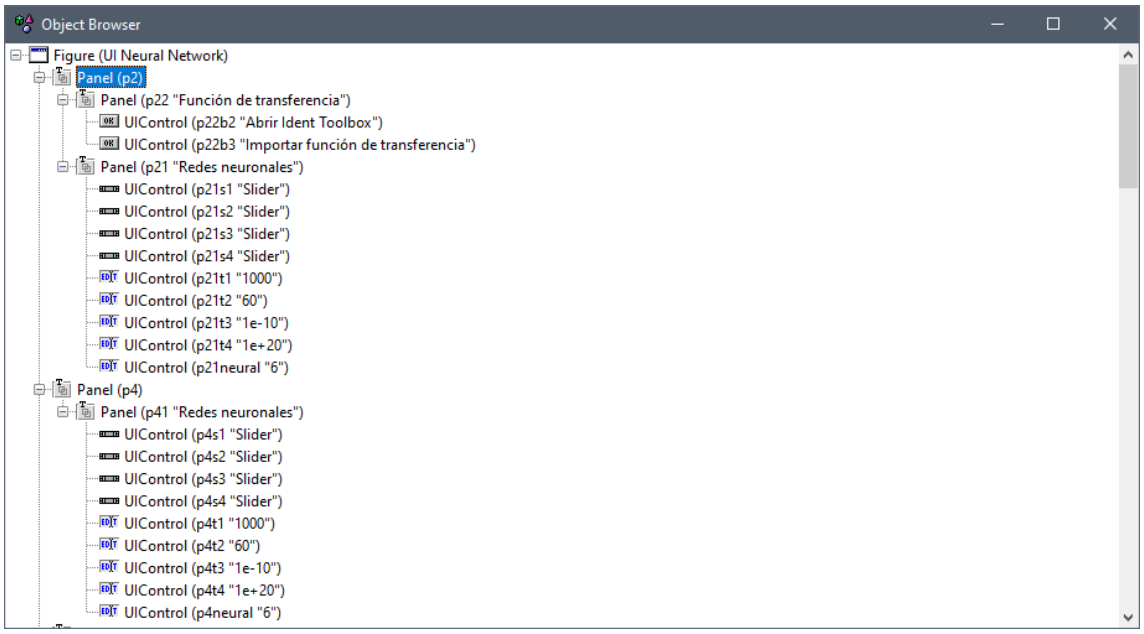

Fig. 16: Object Browser, menú desplegable con las propiedades y lista de todos los objetos presentes en la interfaz.

Panel gráfico: Para obtener una pantalla donde visualizar resultados se utiliza el objeto *Axis,* que funciona como una *Figure* de MATLAB. Para que estos ejes puedan ser utilizados, deben configurarse como la figura activa, empleando su manejador *handles* y utilizando el comando *set(gcf,'CurrentAxes',handles.graf1);*. Una vez configurados, estos ejes se muestran empleando el comando *plot,* pudiendo también acceder a sus propiedades y modificar el formato de la gráfica (Fig. 17).

```
grafl =

  Axes (grafl) with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
        Position: [9.6000 2.2308 300.2000 27]
           Units: 'characters'

  Show all properties

                    ALim: [0 1]
                ALimMode: 'auto'
   ActivePositionProperty: 'position'
        AmbientLightColor: [1 1 1]
             BeingDeleted: 'off'
                      Box: 'off'
                BoxStyle: 'back'
              BusyAction: 'queue'
            ButtonDownFcn: ''
                     CLim: [0 1]
                CLimMode: 'auto'
           CameraPosition: [0.5000 0.5000 9.1603]
       CameraPositionMode: 'auto'
             CameraTarget: [0.5000 0.5000 0.5000]
         CameraTargetMode: 'auto'
```

Fig. 17: Fragmento de la lista de propiedades del objeto *Axis "graf1"*.

Botones de cambio de modo y utilidades: Los botones de cambio de modo se implementan mediante objetos *PushButton*. Este tipo de objetos genera, además de su código de inicialización, el código del evento producido al hacer clic sobre ellos. En esta parte del código es donde programa la parte correspondiente a la función de cada uno. Los botones se han alineado para que el espaciado entre elementos sea el mismo. utilizando la herramienta *Align objects* (Fig. 18) con los elementos seleccionados, el cual abre una ventana emergente donde se puede elegir entre alineación o distribución en cada dimensión, o si se prefiere establecer una separación uniforme utilizando una distancia elegida por el usuario.

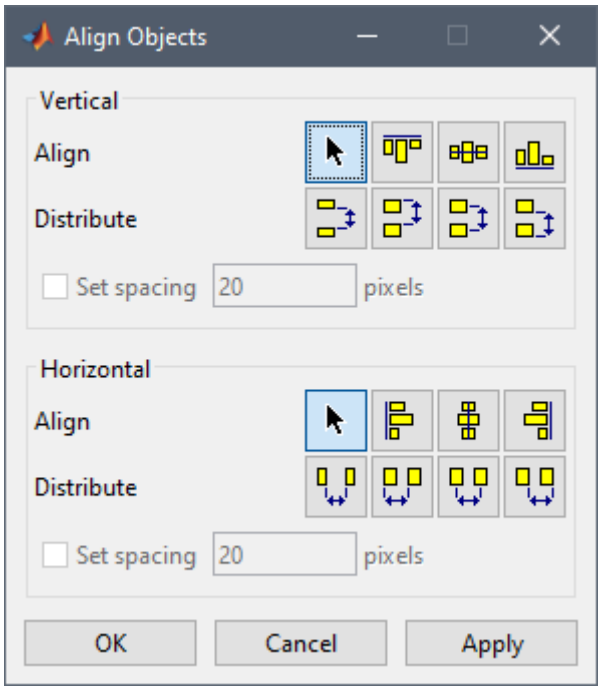

Fig. 18: Panel de configuración *Align Objects*.

Panel de configuración: Se ha creado un panel para cada modo de la aplicación, de forma que se muestra en cada momento el correspondiente al modo actual. Para ello se ha utilizado el objeto *Panel*, que es un contenedor donde se pueden incluir otros elementos, y al hacerlo no visible, dejan de mostrarse todos a la vez conforme se cambie a un modo u otro. Cada modo consta de diferentes objetos según se requieran, pero los elementos más utilizados son los *Static text*, que muestran un texto al usuario, los *Edit text*, que pueden recoger entradas de texto y los *Slider*, que permiten entrada de datos desde el ratón.

3.4. Programación de la interfaz de usuario

Para implementar la interfaz con código de MATLAB se aprovecha el código generado automáticamente (Fig. 19). Este código se puede modificar para programar el comportamiento de la aplicación.

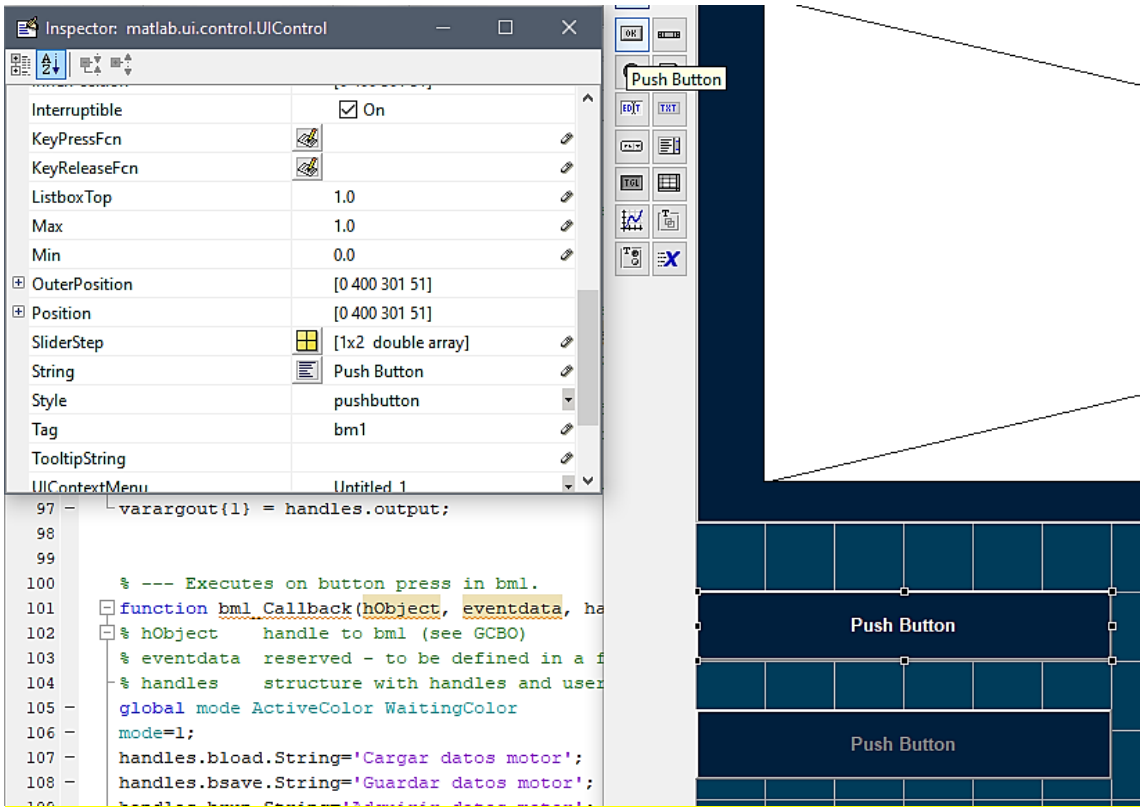

Fig. 19: *Push Button* (derecha) seleccionado, su panel de propiedades (superior izquierda) y el código generado correspondiente a *Push Button* (inferior izquierda).

La programación de la interfaz de usuario se realiza sobre el archivo .m generado por el programa GUIDE. La primera operación del programa es inicializar las variables necesarias para la ejecución de la aplicación (Fig. 20), esto se implementará en la función de *Opening* de la interfaz. Los elementos a inicializar:

```
46
47    % --- Executes just before UI is made visible.
48    function UI_OpeningFcn(hObject, eventdata, handles, varargin)
49    % This function has no output args, see OutputFcn.
50    % hObject    handle to figure
51    % eventdata  reserved - to be defined in a future version of MATLAB
52    % handles    structure with handles and user data (see GUIDATA)
53    % varargin   command line arguments to UI (see VARARGIN)
54
55    % Choose default command line output for UI
56    handles.output = hObject;
57
58    % Update handles structure
59    guidata(hObject, handles);
60    set(gcf,'CurrentAxes',handles.grafl);
```

Fig. 20: Fragmento del código que se ejecuta al inicializar la interfaz.

Variables globales: Estas variables son las que guardarán los elementos empleados en distintas funciones, como los modelos del motor, tiempo de muestreo, algunos colores predefinidos, modo de funcionamiento, etc. Se inicializan mediante unos valores predefinidos o cargadas desde los archivos necesarios, ya que podrían resultar en mensajes de error si están vacías desde el principio.

Textos: Para poder acceder a los textos con mayor facilidad e implementar aplicaciones en múltiples idiomas, los *strings* de los textos se cargan al inicio de la aplicación.

Se han programado las funciones de evento de cada uno de los elementos, modificando el código para que realice la función deseada. Estas funciones son las denominadas *Callback* y están asignadas a cada elemento en particular. Se pueden añadir *Callback* en función del tipo de objeto que estemos tratando. Por ejemplo, un objeto *PushButton* puede tener cinco tipos de *Callback*, como pueden ser el de creación del botón o el de clicado. Para generar el código para cada uno, se puede acceder a estos entrando en las propiedades del objeto o desde el menú *Callback* disponible al hacer clic derecho sobre el objeto. Algunos objetos tienen un tipo de *Callback* yo implementado al crearlos, pero la mayoría no están implementados por defecto.

Al pulsarlos los botones de modo se cambia la variable de modo del programa, modificando los textos de los botones de utilidades. Además, se deben manejar los paneles para que solo se muestre el del modo correspondiente, y marcar el modo actual resaltando el color del botón pulsado.

Dado que todos los modos requieren de algún elemento previo, el único botón habilitado al principio será el del modo más básico, y los demás se irán habilitando para que sea posible elegir el siguiente modo. Los elementos restantes de la programación dependen en gran medida de los modos de la aplicación, alternando su función en cada uno o siendo visibles solo en un modo, por tanto, se explicarán desde su modo concreto.

Paneles de configuración: Se deben programar los elementos internos de estos paneles si requieren algún comportamiento especial. Para evitar confusiones, se ha decidido establecer el nombre de los elementos en función del panel al que corresponden y del tipo de elemento que son, de forma que por ejemplo p3b1 es el primer botón del panel 3. A continuación se detallan los elementos de cada panel y su programación:

- **Panel de datos motor:** Este panel consta de varios *Edit text* para recoger los valores de las variables. Para acceder a estos valores, es importante tener en cuenta que se guardan como *strings*, y por tanto será necesario convertirlos a enteros para su posterior utilización mediante la función *str2num.* El valor de estos *edit text* se tomará al pulsar el botón de ejecutar, de forma que si hay algún valor incorrecto en ellos se generará el error correspondiente. También se emplean los *sliders* con un *edit text* asociado (Fig. 21). Para interconectar estos elementos, se utilizan sus *Callbacks*, de forma que al actualizar la posición de los *sliders* se genera el texto correspondiente en el *edit* y a la inversa.

```
523
524     % --- Executes on slider movement.
525     function samax_Callback(hObject, eventdata, handles)
526     % hObject    handle to samax (see GCBO)
527     % eventdata  reserved - to be defined in a future version of MATLAB
528     % handles    structure with handles and user data (see GUIDATA)
529
530     % Hints: get(hObject,'Value') returns position of slider
531     %        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
532
533 -   global tref
534 -   tref(2) = round(handles.samax.Value,0);
535 -   handles.tamax.String = num2str(tref(2));
536
537     function tamax_Callback(hObject, eventdata, handles)
538     % hObject    handle to tamax (see GCBO)
539     % eventdata  reserved - to be defined in a future version of MATLAB
540     % handles    structure with handles and user data (see GUIDATA)
541
542     % Hints: get(hObject,'String') returns contents of tamax as text
543     %        str2double(get(hObject,'String')) returns contents of tamax as a double
544
545 -   global tref
546 -   tref(2) = str2num(handles.tamax.String);
547 -   handles.samax.Value = tref(2);
548
```

Fig. 21: Fragmento del código que conecta un *slider* y una caja de texto conectados.

- **Panel de modelo de motor:** Este panel está subdividido en dos paneles. El primero consta de dos grupos de *sliders* y *edit texts* interconectados, cuyos límites están fijados por unos valores previamente definidos. El segundo panel consta de dos botones, el primero abre la utilidad IDENT de MATLAB y proporciona unas instrucciones para la importación de datos para esta. El segundo permite importar una función de transferencia de un modelo ya identificado previamente.

- **Panel de modelo de referencia:** En este panel se configura la función de transferencia que se usará como referencia, que se puede configurar en dos *edit texts*. Además, dos botones superiores permiten definir el orden de esta función, y otro objeto de tipo Axis permite visualizar la respuesta ante un escalón unitario de esta función de transferencia y compararla con un máximo previamente establecido, para no forzar el motor y obtener mejores resultados.

- **Panel de modelo de control:** Este panel tiene el mismo funcionamiento que el primer subpanel del modo Modelo motor.

- **Panel de ejecución de control:** Este panel consta de un elemento *Listbox* donde elegir el método de control, y un botón que en función del método elegido lo edita de forma diferente. Para PID, acciona un diálogo que permite editar la función de transferencia a partir de varias acciones disponibles: Editar, Importar o Manual. Para el control borroso y para redes neuronales no es necesario porque ya ha sido previamente configurado.

Botones de utilidad: Los botones de utilidad son los botones básicos para el manejo de la interfaz y sirven para guardar progresos y cargarlos posteriormente, y para ejecutar el programa. Cambian su operación en función del modo en el que se encuentre el programa. La programación de estos botones se puede definir como:

- **Botón de cargar:** Este botón ejecuta una ventana emergente creada con el comando *inputdlg* en la que cargar un archivo previamente guardado del tipo que trabaje el modo en el que esté actualmente. Es decir, si está en el modo de Modelo motor, sobrescribirá la variable *MotorNet* al cargar el archivo.mat, actualizando así los datos a los del modelo guardado. Además, se encarga de actualizar la gráfica con la información cargada y desbloquea el botón del modo siguiente, ya que hay datos listos.

- **Botón de guardar:** Al igual que el botón anterior, este botón hace visible una ventana emergente que pide un nombre de archivo. En este archivo se guardará la variable correspondiente al modo en el que se encuentre para cargarla posteriormente (Fig. 22). Cabe destacar que en el modo correspondiente a la ejecución del control, no se guarda una variable, sino la figura visualizada al ejecutar el control.

```
319
320
321     % --- Executes on button press in bsave.
322     function bsave_Callback(hObject, eventdata, handles)
323     % hObject    handle to bsave (see GCBO)
324     % eventdata  reserved - to be defined in a future version of MATLAB
325     % handles    structure with handles and user data (see GUIDATA)
326 -   global mode
327 -   switch mode
328 -       case 1
329 -           ans = inputdlg('Nombre del archivo a guardar:','Datos motor',[1 40],{'MotorData.mat'});
330 -           save(cell2mat(ans),'MotorData')
331 -           handles.tstate.String='Datos del motor guardados';
332 -       case 2
333 -           ans = inputdlg('Nombre del archivo a guardar:','Modelo motor',[1 40],{'MotorNet.mat'});
334 -           save(cell2mat(ans),'MotorNet')
335 -           handles.tstate.String='Datos del modelo del motor guardados';
336 -       case 3
337 -           ans = inputdlg('Nombre del archivo a guardar:','Modelo referecia',[1 40],{'RefModel.mat'});
338 -           save(cell2mat(ans),'RefModel')
339 -           handles.tstate.String='Datos del modelo de referencia guardados';
340 -       case 4
341 -           ans = inputdlg('Nombre del archivo a guardar:','Modelo control',[1 40],{'ControlNet.mat'});
342 -           save(cell2mat(ans),'ControlNet','PreviousNet')
343 -           handles.tstate.String='Datos del modelo de control guardados';
344 -       case 5
345 -           ans = inputdlg('Nombre del archivo a guardar:','Figura',[1 40],{'Figura.fig'});
346 -           savefig(cell2mat(ans));
347 -           handles.tstate.String='Figura guardada';
348 -   end
349
```

Fig. 22: Fragmento del código de la función de guardar con una estructura *switch case*.

- **Botón de ejecución:** Este botón es el que se encarga de la ejecución del código importante del programa, del que se hablará a continuación. Después de ejecutar estas partes de código, se encarga de mostrar la información en los ejes y habilitar el botón del modo siguiente.

3.5. Programación de los modos de la interfaz de usuario

En este apartado se trata la programación de los modos que se ejecutan al pulsar el botón de ejecución en cada uno éstos. Se pasa a analizar cada modo distinto por separado:

3.5.1. Datos motor

Para obtener los datos del motor, es necesario actualizar las variables necesarias a partir de los valores incluidos en el panel. Con estos datos, se ejecuta el archivo test.m . Este archivo se encarga de crear un vector en el que estará la entrada para el motor, generado a partir de la función *skyline* y los datos anteriores. Con esta entrada y los parámetros anteriores se ejecuta control.m, que empleando funciones de la librería de la DAQ, consigue la conexión con el motor. control.m consta de tres periodos de operación:

- Start, en la que se inicializan las variables necesarias y se fija la salida de la DAQ a 5 V para que la salida del motor sea 0 V
- Control, la que consigue aplicar las entradas incluidas en el vector generado con anterioridad al motor mientras se registran las salidas.
- End, donde se devuelve al motor al estado de reposo y se devuelven los datos registrados a la variable deseada.

3.5.2. Modelo motor

En este caso, se emplea la función motorModel.m utilizando los valores recogidos anteriormente del motor. Esta función crea una red neuronal con los parámetros recogidos en el panel, y acto seguido la prepara y entrena para “parecerse” al motor, mediante la modificación de los valores de los pesos de sus neuronas.

3.5.3. Modelo referencia

En este modo simplemente se modificará una variable en la que se almacena la función de transferencia que se usará de referencia, y se mostrará en las gráficas para poder ajustarla adecuadamente.

3.5.4. Modelo control

Este modo hace empleo de la función controlModel.m, que a su vez usa los datos del modelo de referencia y el modelo del motor. Con estos datos se crea una red neuronal de cuatro capas. Las dos primeras son el modelo de control que se pretende entrenar, y la tercera y la cuarta capas se crean con las mismas neuronas que el modelo del motor, y se actualizan los valores de sus pesos para que actúe como una copia exacta de esta, ya que para entrenar las dos primeras capas es necesario seguir este procedimiento. Además, se configura para que estas neuronas no sean reentrenadas y se prepara la segunda red. Con esto se empieza el entrenamiento, en el que se pretende minimizar el error entre el modelo de referencia y el conjunto del control y el modelo del control (Fig. 23). Finalmente, se crea una nueva red, esta vez de una capa donde se actualizan los valores para coincidir con los entrenados en el control mediante este proceso.

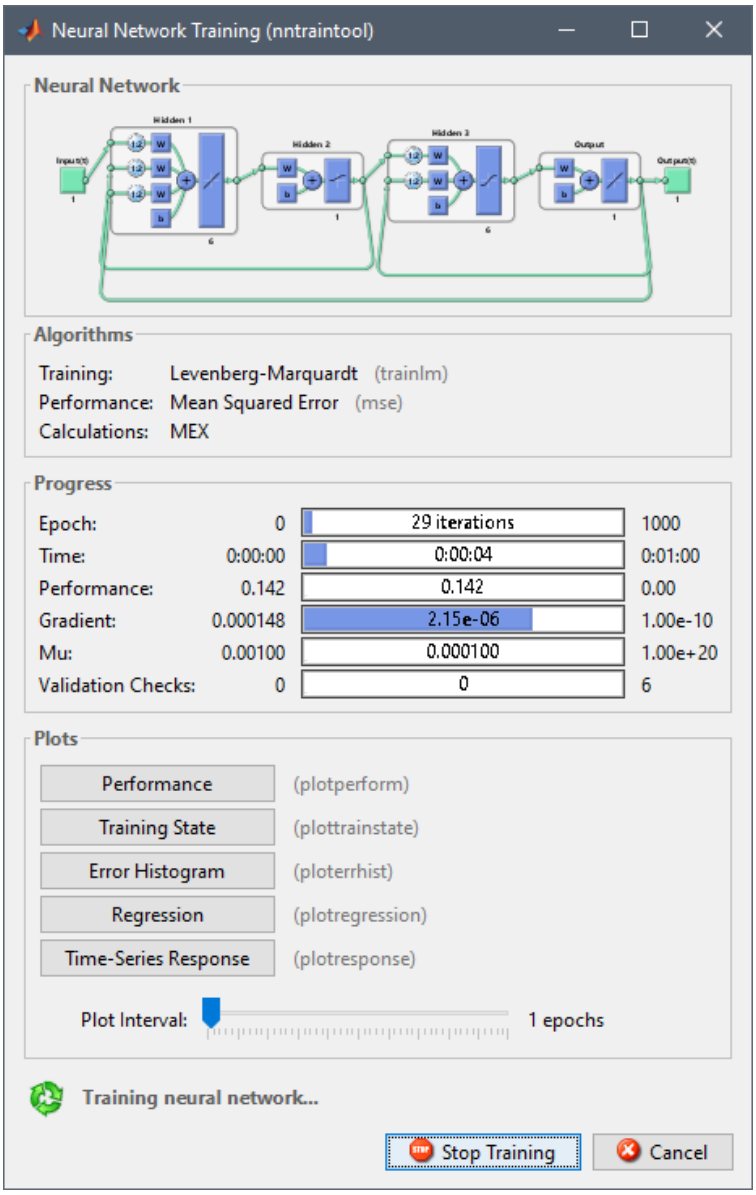

Fig. 23: Panel de entrenamiento de la red neuronal, con un esquema de las cuatro capas y del estado del aprendizaje.

3.5.5. Ejecución control

En el último modo, se pretende probar el funcionamiento de los controles deseados, mediante el uso de la función testControl.m, que incluye los parámetros necesarios para configurar todo, incluido el tipo de control deseado. Esta función configura las entradas y llama a runControl.m, que es la encargada del control. Funciona análogamente a control.m, mediante los tres estados (Start, Control, End), pero en este caso el cálculo de la acción de control viene dado por el tipo de control elegido. Para el caso de PID la acción de control se calcula a partir de la discretización del controlador PID. Si el control elegido es de tipo borroso, se utiliza *evalfis*, una función que evalúa el controlador borroso y actualiza la acción de control en función de la salida de este. Por último, para la red neuronal se utiliza la propia red, en la que se incluyen la salidas y referencias anteriores y se evalúa para determinar la acción de control. Todos estos modos incluyen un posterior control para saturar la acción de control y la posibilidad de usar un filtro que limite los incrementos de la acción de control a un 10%, haciendo función de paso bajo.

4. Control del motor de CC

El control de un motor de corriente continua puede ser abordado utilizando diversos métodos, en función del comportamiento que se desee tener, la robustez y las especificaciones. En este proyecto se implementará el control de velocidad de este tipo de motor desde tres perspectivas diferentes para compararlas y establecer las ventajas y desventajas en este proceso en particular y extraer conclusiones generales.

Para ello, se evalúa la facilidad con la que puede ser implementado el control, el comportamiento del control en condiciones normales y la robustez del control ante cambios en la planta.

4.1. Control mediante redes neuronales

Para el control del motor mediante redes neuronales, se necesita primero entrenar un modelo en redes neuronales del motor, para mediante este poder entrenar el control en redes neuronales. Los pasos a seguir en este proceso serían por tanto los siguientes:

4.1.1. Modelo del motor con redes neuronales

En este apartado se trata el entrenamiento de la red neuronal que actúa como modelo. Para el entrenamiento de esta red, se debe primero decidir qué tipo de datos se han de usar. Cuanta mayor sea la cantidad de datos, mejor se entrenará la red neuronal, mientras que para las zonas en las que falten datos el modelo puede contener imprecisiones, por tanto, se debe trabajar con rangos amplios de valores si se quiere un modelo preciso en general, aunque esta dará lugar a que se penalicen menos los errores en la zona muerta y por tanto sea más difícil tener un buen resultado para esa zona.

Con el objetivo de decidir la estructura de la red neuronal y obtener el código base para implementar la función motorModel.m, se ha abierto la *toolbox* de redes neuronales (Fig. 24) de MATLAB, mediante el comando *nnstart*. Este paso solo se lleva a cabo durante el diseño de la interfaz, y por tanto no es necesario repetir este proceso. Como en este caso se pretende modelar un sistema dinámico, se utiliza la aplicación de *Time series*. El tipo de red a elegir será una red NARX, ya que el estado de un sistema dinámico evoluciona con el tiempo, y este permite almacenar el estado y realimentarlo para obtener nuevos valores.

Esta estructura contará con dos capas: una primera capa oculta de cálculo con dos retardos para almacenar la información del estado, cuya función de activación es una sigmoide, y una segunda capa cuya función de activación permite combinar esta información de manera lineal y obtener una sola salida. También se puede configurar qué porcentaje de datos se utilizan para entrenamiento, validación y test, o el algoritmo de entrenamiento, aunque en este caso no se han modificado.

Mediante esta *toolbox*, y empleando los datos del motor, se ha generado el código base utilizando la función de explortación código de MATLAB, que permite el entrenamiento de esta red neuronal. La elección del número de neuronas determina las posibilidades de nuestra red neuronal. Un número demasiado limitado de neuronas provocará inflexibilidad en el modelo, y por tanto menor capacidad para adaptarse a este. En cambio, un número alto de neuronas

puede provocar demasiada variabilidad en el modelo en el caso de no tener suficientes datos, y provocar comportamientos anómalos indeseados en condiciones normales del sistema.

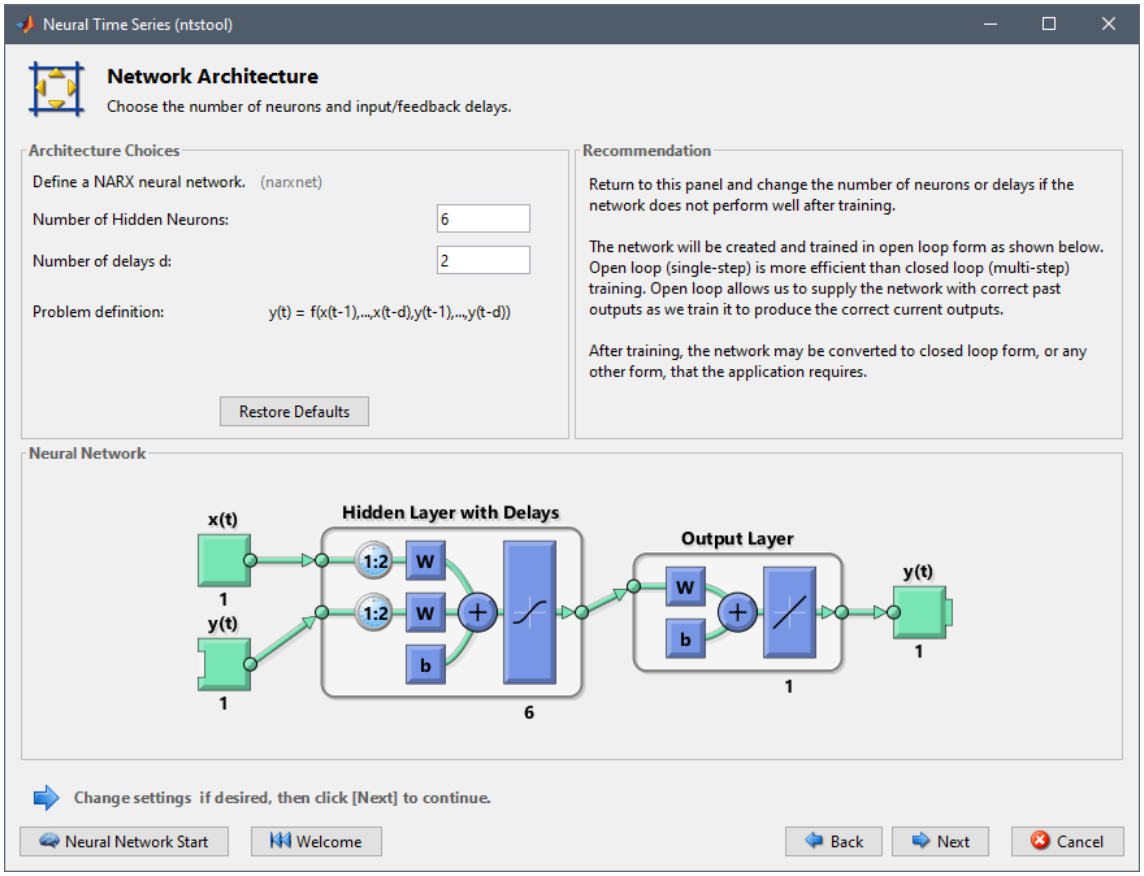

Fig. 24: Panel de configuración del entrenamiento de la red neuronal generado por el comando *nnstart.* Se pide al usuario el número de neuronas y de retardos, y se le muestra un esquema de las capas de la red neuronal.

Este entrenamiento se llevará a cabo mediante el algoritmo de Levenberg-Marquardt [7] (predeterminado de MATLAB para este tipo de redes). Dicho algoritmo de aprendizaje es normalmente el método más rápido de *backpropagation* [8], y trata de modificar los pesos para obtener unas determinadas salidas a partir de un conjunto de entradas. Para ello se sigue el siguiente proceso:

1. Primero se inicializan los pesos de las neuronas. Es aconsejable que se inicialicen con valores aleatorios próximos a cero, pero sin ser cero para evitar errores en la primera iteración.
2. Después se procede a pasar las entradas a través de la red. Estas entradas se ven multiplicadas por los pesos, y operadas de modo distinto hasta llegar a calcular, mediante la función de activación, las diferentes salidas.
3. En este paso, se calcula el error total cometido respecto a la referencia deseada (Fig. 25).

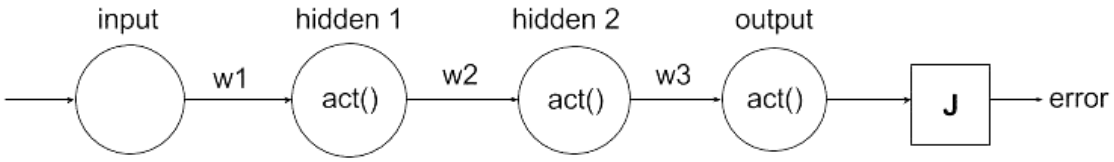

Fig. 25: Propagación del error desde la entrada [9].

4. Este es el paso donde se comienza a entrenar la red. Se calcula cuál es el nivel en el que afecta cada una de las neuronas de la última capa al error total, y mediante un conjunto de operaciones con derivadas, se obtiene el gradiente para cada uno de los pesos y la aproximación a la matriz Hessiana, que depende de mu (*momentum update*). Si mu es cero se está empleando el método de Newton, que es una versión más rápida para la actualización de los pesos. Si mu es grande, se ralentiza el descenso del gradiente y por tanto se pretende que el valor de mu sea bajo para simplificar el proceso. El valor de mu se incrementará si el rendimiento no mejora.
5. Finalmente, se vuelve a operar hasta llegar a la primera capa, y posteriormente se actualizan todos los pesos de acuerdo a la aproximación a la matriz Hessiana calculada (Fig. 26).

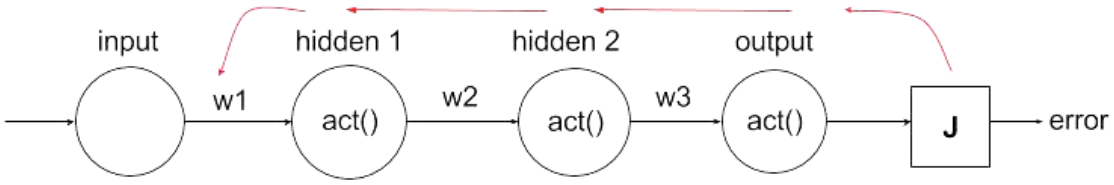

Fig. 26: Propagación inversa del error, seguido del cálculo del gradiente de los pesos que se utilizará para la actualización de pesos [9].

Este es un proceso iterativo, y por tanto el entrenamiento llegará a su fin cuando uno de los parámetros de parada de entrenamiento sea alcanzado. Estos parámetros son los siguientes, y se pueden visualizar en la interfaz que muestra MATLAB durante el entrenamiento de la red neuronal (Fig. 27):

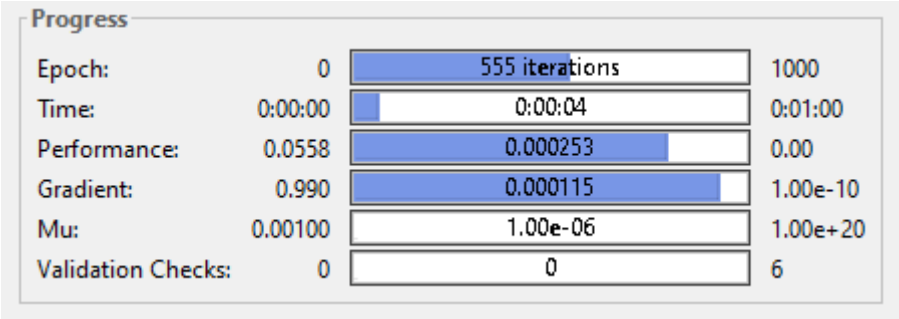

Fig. 27: Parámetros de parada de entrenamiento visualizados durante el entrenamiento de la red neuronal que actuará como modelo del motor.

- **Número máximo de iteraciones del algoritmo:** Para limitar el número de ejecuciones del proceso iterativo y poner fin a este entrenamiento si no se alcanza ninguna de las otras condiciones.
- **Máximo tiempo de entrenamientos:** Similar al anterior método, pero limitando el tiempo durante el que se entrena la red.
- **Rendimiento de la red llevado hasta el objetivo:** Al principio del entrenamiento se fija un objetivo de rendimiento de la red. El rendimiento viene dado por la suma cuadrática de los errores cometidos entre los datos del motor y la salida de la red neuronal (Fig. 28).

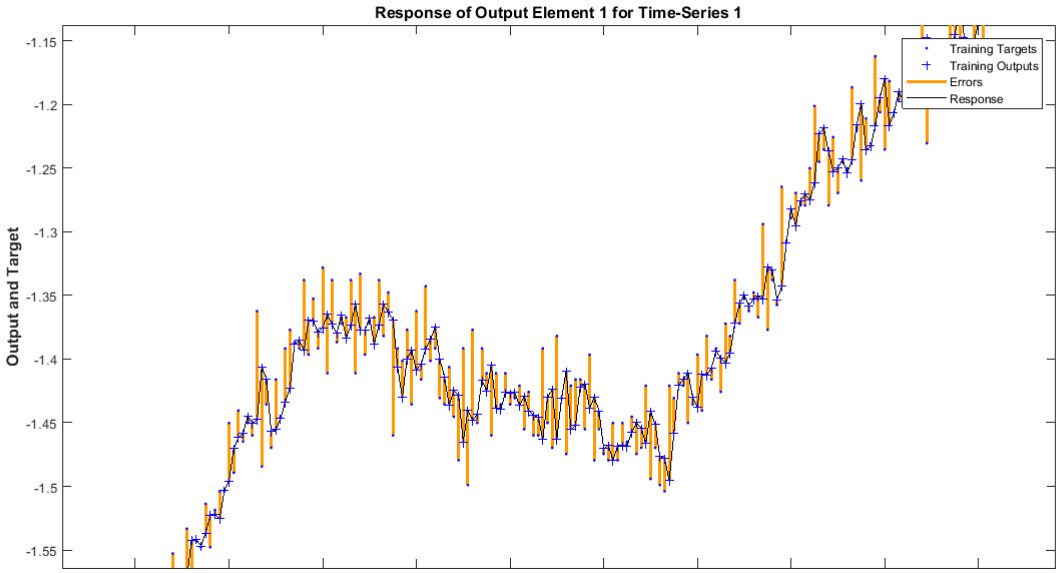

Fig. 28: Visualización de los errores cometidos (líneas naranjas) entre los datos reales (puntos azules) y la salida de la red neuronal (cruces azules/línea negra) mediante la herramienta.

Durante el entrenamiento este rendimiento mejorará con las iteraciones, por lo general de manera muy pronunciada al principio y ralentizándose hasta alcanzar un mínimo de error. En el caso de que este mínimo sea local, debido a una configuración de las neuronas que localmente no mejore los resultados, podrá darse el caso en el que pasado un tiempo logre superarse hasta llegar a un nuevo mínimo error (Fig. 29).

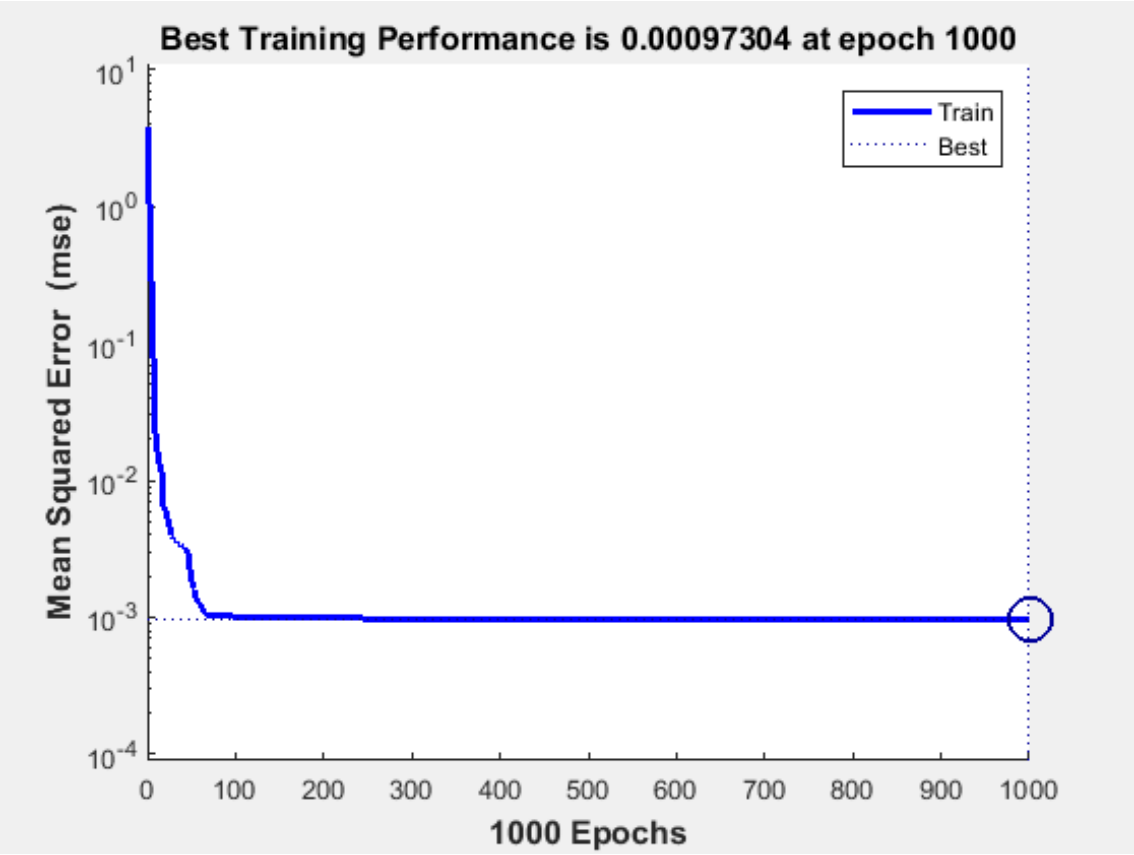

Fig. 29: Gráfica de la evolución del rendimiento, el cual se estabiliza en 10^{-3} unidades de error cuadrático. Alrededor de 50 iteraciones se produce un error mínimo local, deja de mejorar alrededor de las 75 iteraciones y se estabiliza alrededor de las 100.

- **Gradiente de rendimiento:** El gradiente es la diferencia de rendimiento entre dos iteraciones del algoritmo. En el caso de que este gradiente sea próximo a cero, indicará un mínimo de error, ya que no se estará mejorando este resultado.
- **Mu supera el máximo:** Mu es un parámetro de control que afecta directamente al error de convergencia, si este parámetro excede unos límites, se parará el entrenamiento.
- **El rendimiento de validación es peor que la última mejor validación:** Este caso viene indicado por un aumento en el error cuadrático en las validaciones, lo que provocará el cese del entrenamiento al repetirse en un número concreto de ocasiones y se volverá al último resultado en el cual el error hubiese bajado (Fig. 30).

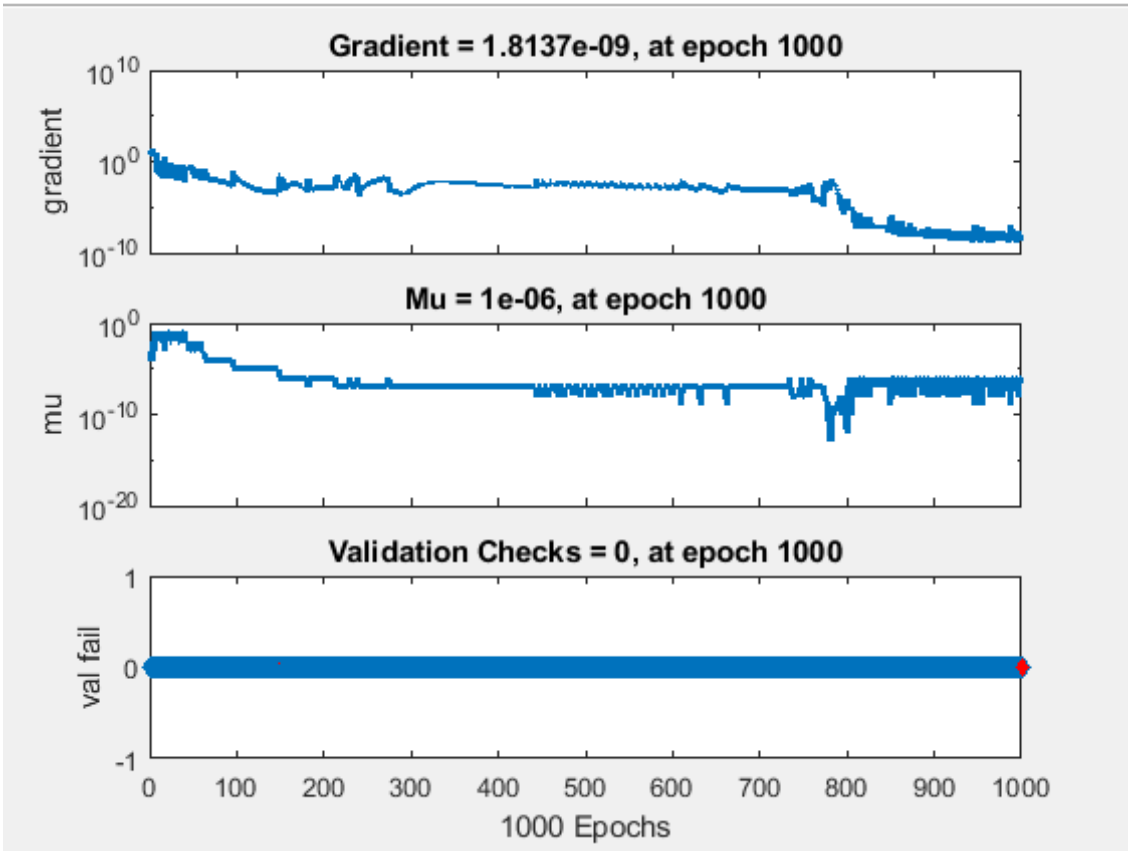

Fig. 30: De arriba abajo, gráfica de la evolución temporal del gradiente, mu y las comprobaciones mediante datos de validación.

Una vez integrado en la aplicación, y configurados los parámetros de entrenamiento, se puede observar que la red neuronal entrenada con los mismos datos no será exactamente igual en varias iteraciones, debido a la que la inicialización aleatoria de los pesos da lugar a diferentes entrenamientos y resultados. Por esto es importante guardar el resultado de un buen entrenamiento o fijar el valor inicial de los pesos para obtener un solo resultado. Un ejemplo de un entrenamiento exitoso del motor, sería el mostrado en la siguiente gráfica, con el motor configurado con resistencia adicional, donde se puede observar una coincidencia alta en el comportamiento del modelo respecto a los datos de entrenamiento del motor (Fig. 31).

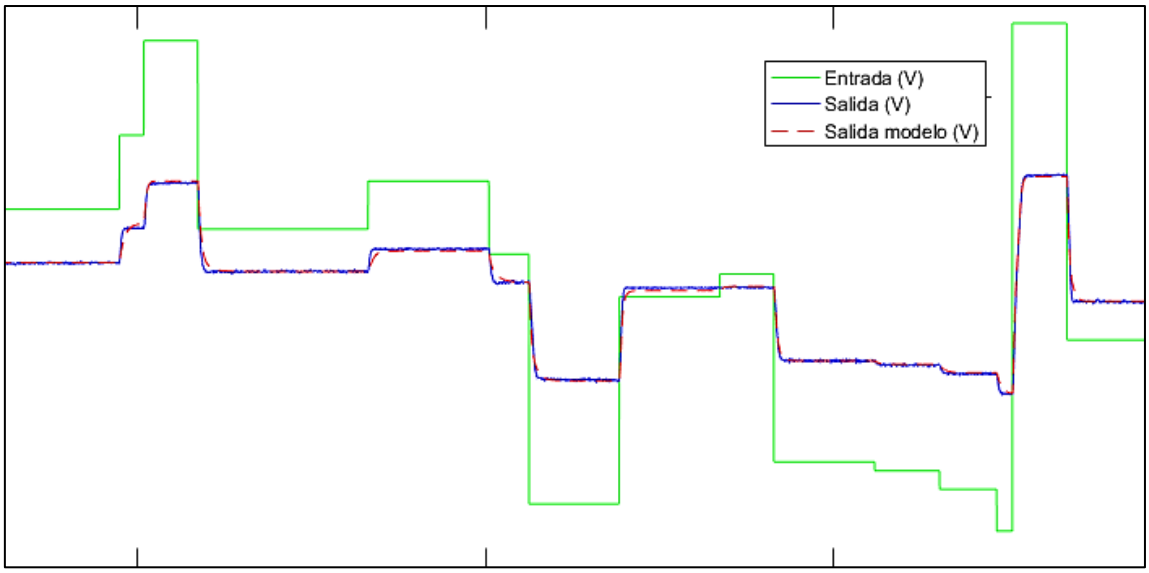

Fig. 31: Respuesta temporal del motor (azul) ante una entrada en *skyline* (verde) y respuesta ante esta misma entrada de la red neuronal entrenada (rojo discontinuo).

4.1.2. Control del motor con redes neuronales

Antes de poder entrenar el control junto a esta red neuronal, se deben obtener datos de un experimento sobre el modelo de referencia deseado. Este modelo se obtiene a partir de la función *skyline*, que utiliza el mismo tipo de datos de entrada que en la identificación del proceso. Además, debe tener en cuenta las capacidades reales del proceso, y por tanto, para poder comparar se analiza la respuesta del sistema para la máxima entrada (Fig. 32).

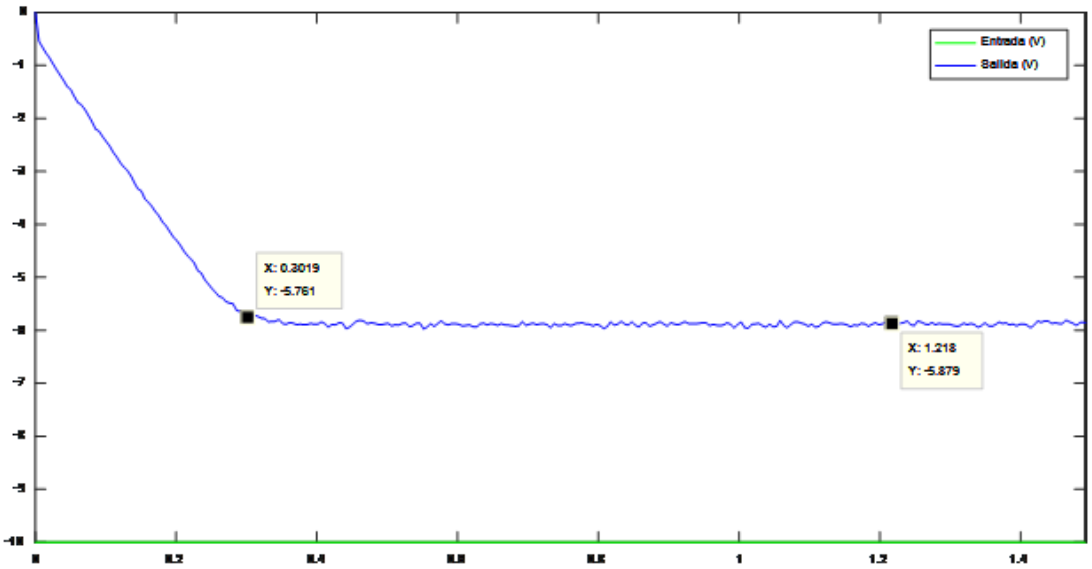

Fig. 32: Respuesta del motor ante el máximo escalón de entrada negativo. Al construir el modelo de referencia se tendrá en cuenta que el tiempo de establecimiento ante un escalón máximo no puede ser menor al de esta respuesta.

Una vez obtenido el modelo de referencia, se ha de configurar la estructura para entrenar la red neuronal encargada del control. Esta estructura se compondrá por dos bloques principales, el segundo de los cuales ya ha sido configurado con anterioridad. Estos bloques son: la red de control a entrenar y el modelo del motor, el cual se copia en este segundo bloque para utilizarlo en el entrenamiento (Fig. 33). El nuevo bloque se sitúa a continuación de las entradas, y contiene una primera capa con dos retardos, encargada de guardar la información del estado y procesarla, y una segunda capa encargada de combinar los resultados de la capa anterior.

Durante el proceso, se bloquearán los pesos de las dos últimas capas, correspondientes al modelo del motor, y tan solo se entrenarán la primera y la segunda. Además, para conseguir que la acción de control esté limitada entre 0 y 10 (-10 y 10 a la salida de la DAQ), se implementa una función de tipo *logsig*, que limita la activación de la neurona entre 0 y 1 y se multiplican los pesos de entrada por 10, consiguiendo así el resultado equivalente a tener el rango deseado en la acción de control.

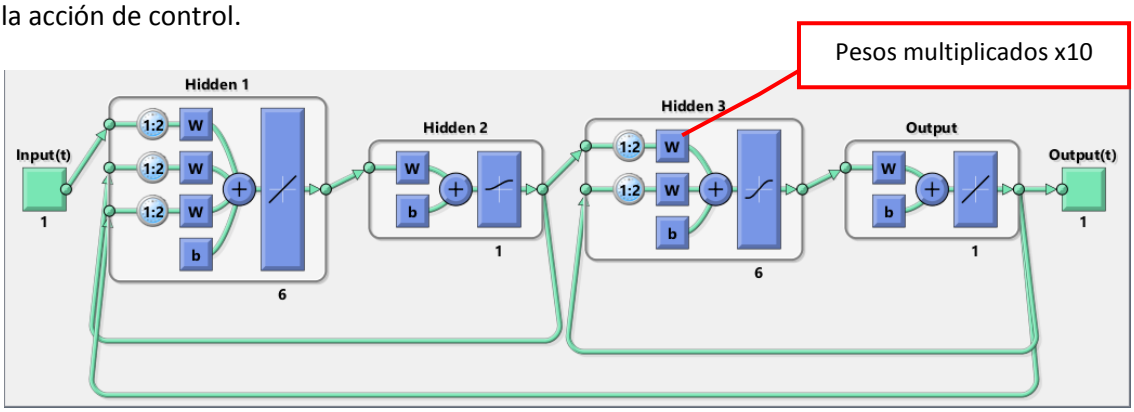

Fig. 33: Representación de la red neuronal del control (capas 1 y 2, dos primeros bloques) más el modelo del control (capas 3 y 4, dos últimos bloques).

Finalmente se procede al entrenamiento de esta red. En el entrenamiento se ha de tener en cuenta que si las acciones de control mediante las que se consigue el control son demasiado oscilantes, dará problemas en el control real posterior como vibraciones fuertes al llegar a la referencia, inestabilidad en el control o un mayor desgaste del motor debido a los cambios constantes (Fig. 34).

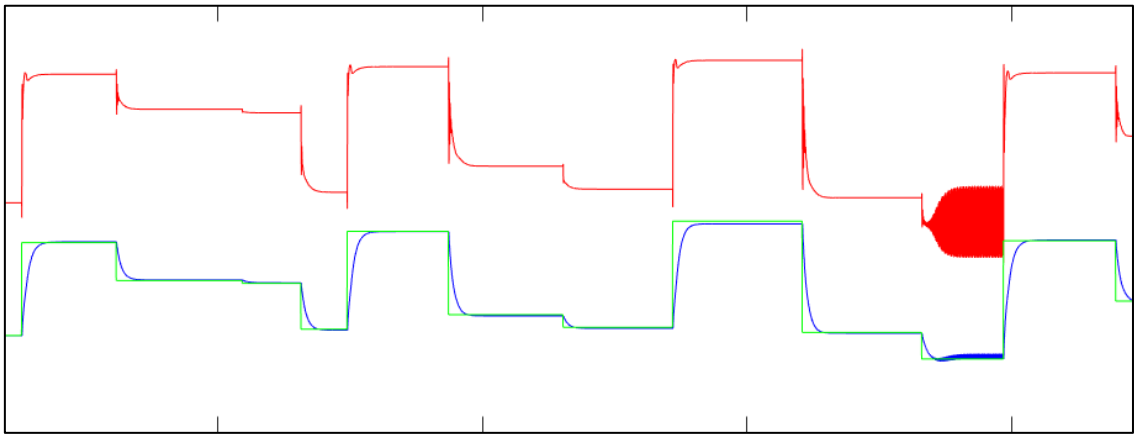

Fig. 34: Respuesta temporal simulada del sistema empleando el control de redes neuronales. La referencia de velocidad (verde) es seguida por el motor (azul) haciendo uso de las acciones de control (rojo) proporcionadas por el control. Se muestran oscilaciones puntuales en la acción de control (derecha) que pueden provocar problemas en el motor o en el control, y por tanto se deberá reentrenar para eliminarlas.

4.2. Control mediante PID

A continuación, se ha implementado el control mediante PID para posteriormente comparar estos resultados con los del control mediante redes neuronales.

4.2.1. Identificación de la función de transferencia del modelo

El primer paso para implementar un control de tipo PID de un motor de corriente continua es la obtención de un modelo del motor mediante función de transferencia. Para ello, primero se deben obtener datos del motor. Para la identificación dinámica de un proceso, se propone generar un tren de escalones a la entrada y observar la respuesta temporal del sistema a la salida, y mediante la herramienta IDENT elegir unos datos de validación y otros de entrenamiento para identificar este modelo. A continuación, se detalla este procedimiento:

Obtención de datos: Los datos se generarán a partir de un tren de escalones para los diferentes rangos de entrada del motor. Dado que el rango de entrada del motor es de ±10 V, se propone comprobar tanto el funcionamiento desde paro como el funcionamiento al cambiar dirección o velocidad. Además, al estar controlado desde un ordenador, el proceso será discreto y por tanto requiere la elección de un periodo de muestreo. La elección de este periodo de muestreo viene dada por la frecuencia de la señal a medir. En este caso, al estar trabajando con velocidad del motor no se llegará a frecuencias muy altas, y por tanto el periodo de muestreo de 0.005 segundos será más que suficiente para evitar problemas de *aliasing*. El resultado de la obtención de datos es el que se puede comprobar en la siguiente figura (Fig. 35).

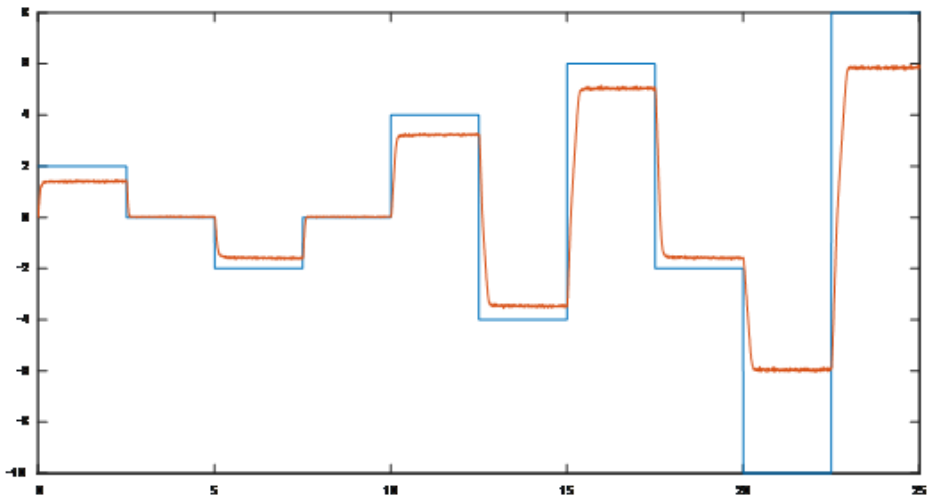

Fig. 35: Respuesta del motor (naranja) ante una serie de escalones de diferentes rangos (azul).

Procesado de datos: Para la correcta identificación, se ha de procesar esta entrada como escalones independientes en incrementos de acción de control, para ello se ejecuta un sencillo código que trocea la entrada y hace que el valor inicial de cada trozo sea cero. En este código puede observarse como para generar cada trozo, se parte del trozo anterior y se le resta su valor inicial:

```
for i=1:N-1
      it(i,:)=t((i-1)*l+l/2:i*l+l/2)-t((i-1)*l+l/2);
      iu(i,:)=ref2(2,(i-1)*l+l/2:i*l+l/2)-ref2(2,(i-
1)*l+l/2);
      iy(i,:)=ref2(3,(i-1)*l+l/2:i*l+l/2)-ref2(3,(i-
1)*l+l/2);
end
```

El resultado de este procesado se puede comprobar en la figura siguiente (Fig. 36), donde se visualizan los datos ya introducidos en la herramienta IDENT. Se puede comprobar cómo ahora ya no es una señal entera, sino muchos escalones a partir del incremento de la entrada que parten de valor nulo.

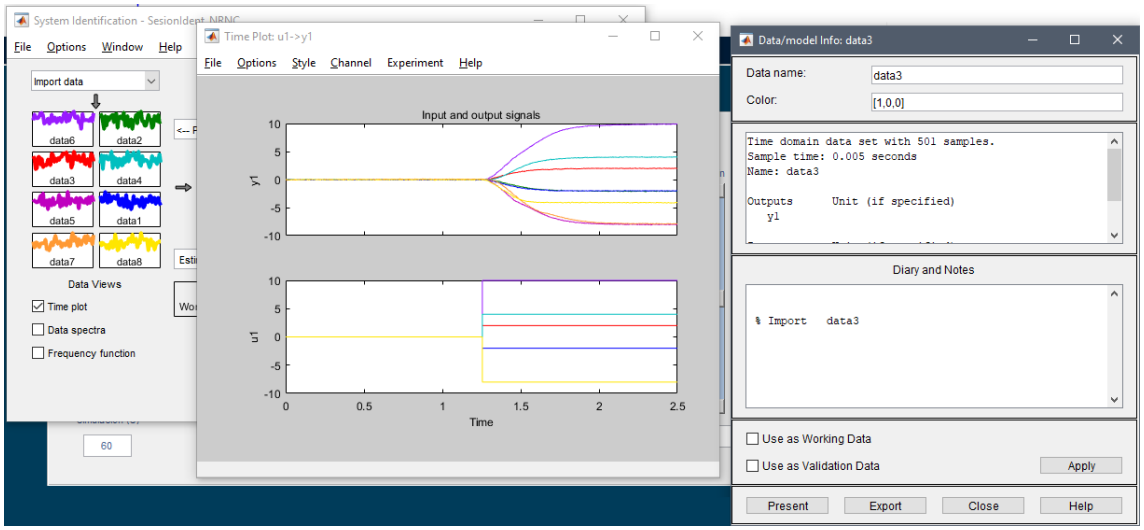

Fig. 36: Ventanas mostradas a partir de la aplicación de identificación de MATLAB. De izquierda a derecha: Ventana principal de la aplicación, con los diferentes datos importados. Respuesta temporal del motor ante los diferentes escalones de entrada. Ventana de visualización de la información de los datos importados.

Identificación: Se puede observar que el motor de corriente continua funciona como un proceso de primer orden en el caso de la velocidad, ya que se estabiliza tras un periodo de tiempo después de introducir un escalón, y no se observa ninguna sobreoscilación al estabilizarse. Se puede obtener una función de transferencia estudiando varias características de la respuesta ante escalón, como pueden ser la ganancia y la constante de tiempo.

Para el cálculo de la ganancia se ha de obtener el valor final después de aplicar el escalón, (Fig. 37) y para el cálculo de la constante de tiempo se debe identificar el tiempo de establecimiento. A partir de estos valores, se obtienen ganancia y constante de tiempo (Ec. 4, 5, 6) y con estas la función de transferencia (Ec. 7, 8).

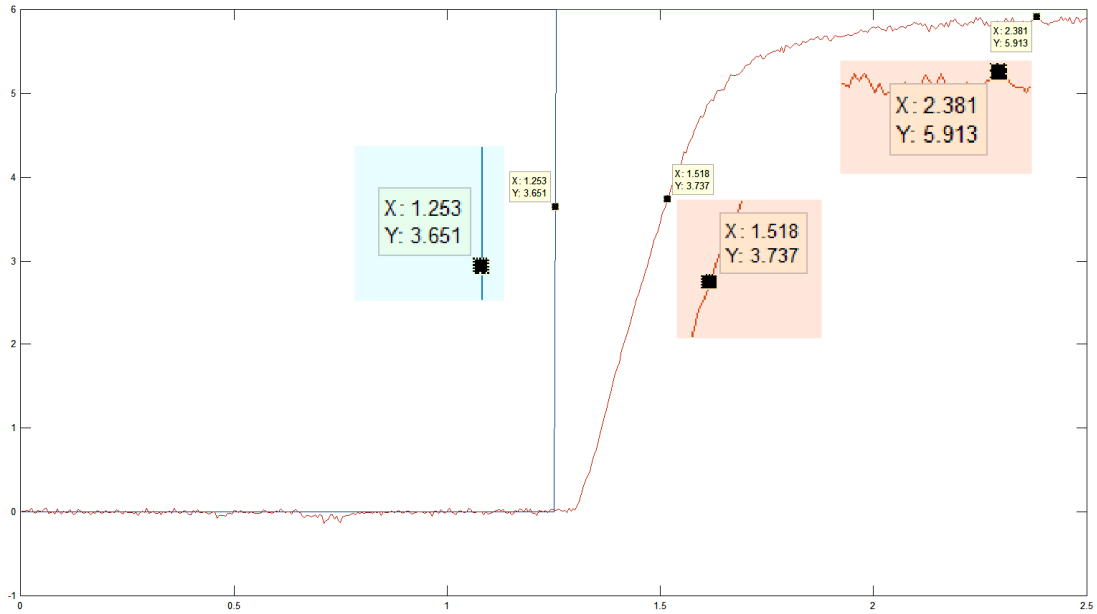

Fig. 37: Datos del tiempo inicial, el valor al 63.2% para el cálculo de la constante de tiempo y el valor final.

$$Ganancia = \frac{Valor\ final}{Entrada} = \frac{5.913\ V}{6\ V} = 0.9855 \tag{4}$$

$$V\ (\tau) = 63.2\%\ del\ valor\ final * Valor\ final = 3.737\ V \tag{5}$$

$$\tau = t(63.2\%) - t(0) \rightarrow\ \tau = 1.517\ s - 1.253\ s = 0.264\ s \tag{6}$$

$$G(s) = \frac{K}{1 + \tau * s} = \frac{0.9855}{1 + 0.264 * s} \tag{7}$$

discretizando mediante el comando *c2d(ft,0.005,'zoh')*

$$G(z) = \frac{0.0188283}{z - 0.9814} \tag{8}$$

Para validar la identificación de datos desde IDENT mediante función de transferencia, el primer paso que demanda el programa es introducir el número de polos y ceros para la identificación y si es un proceso continuo o discreto. Al ser un proceso de primer orden constará de tan solo un polo y ningún cero, y dado que se han empleado los datos de la DAQ, será de naturaleza discreta. También hay que destacar que no se observa ningún tipo de retardo, ya que la salida comienza a cambiar en el instante en el que la entrada cambia. Por tanto, se elige que identifique con solo un polo y ningún cero. Además, se puede fijar el retardo a cero en la estimación de la función de transferencia para que sea más correcta.

El resultado de la identificación de validará con otro conjunto de datos entero, ya que nunca se deben usar los datos de entrenamiento para la validación. Finalmente, cuando el resultado sea satisfactorio como para la (Fig. 38), se guardará la función de transferencia y procederá a la etapa de diseño.

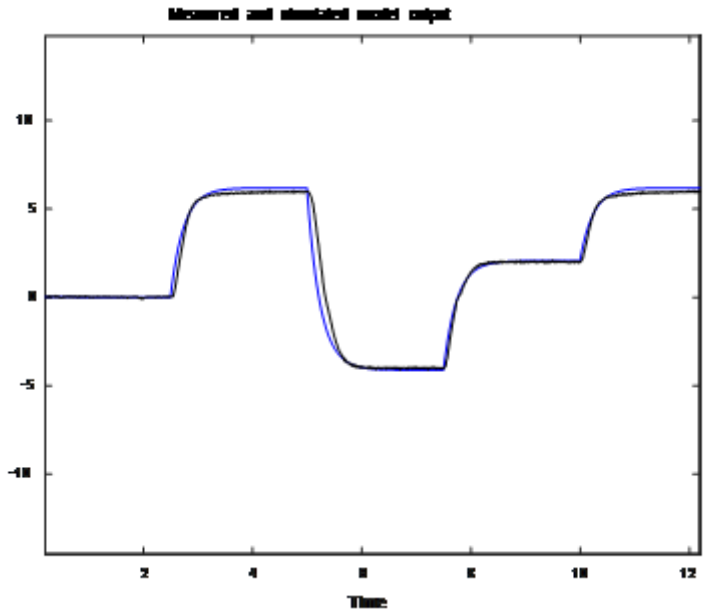

Fig. 38: Respuesta ante escalones del motor (negro) y modelo mediante función de transferencia.

El resultado obtenido mediante la herramienta IDENT concuerda con los cálculos manuales, resultando en dos funciones muy similares (Ec. 9):

$$G(z) = \frac{0.02145}{z - 0.9792} \tag{9}$$

4.2.2. Diseño del controlador PID

El diseño del PID se realiza mediante un proceso iterativo hasta conseguir el controlador más sencillo posible que consiga su propósito. Es por esto por lo que se prueban varias configuraciones, haciendo uso de los diferentes elementos del PID y de un filtro para la acción de control. Para todas estas pruebas, se limita la acción de control a ±10, el rango de entrada del motor.

Las primeras pruebas se realizan con un controlador de tipo P, dando como resultado un control rápido pero con error de posición, además de una acción de control muy poco estable incluso con la acción del filtro (Fig. 39 y Fig. 40). El filtro simplemente reduce los incrementos de la acción de control hasta un determinado factor, concretamente el empleado en este caso es del 10% del valor original, eliminando cambios bruscos en esta y actuando hasta cierto punto como un filtro paso bajo. Además, este tipo de control no fue capaz de seguir referencias muy cercanas a la zona muerta, ya que la acción de control no era suficientemente grande para salir de esta.

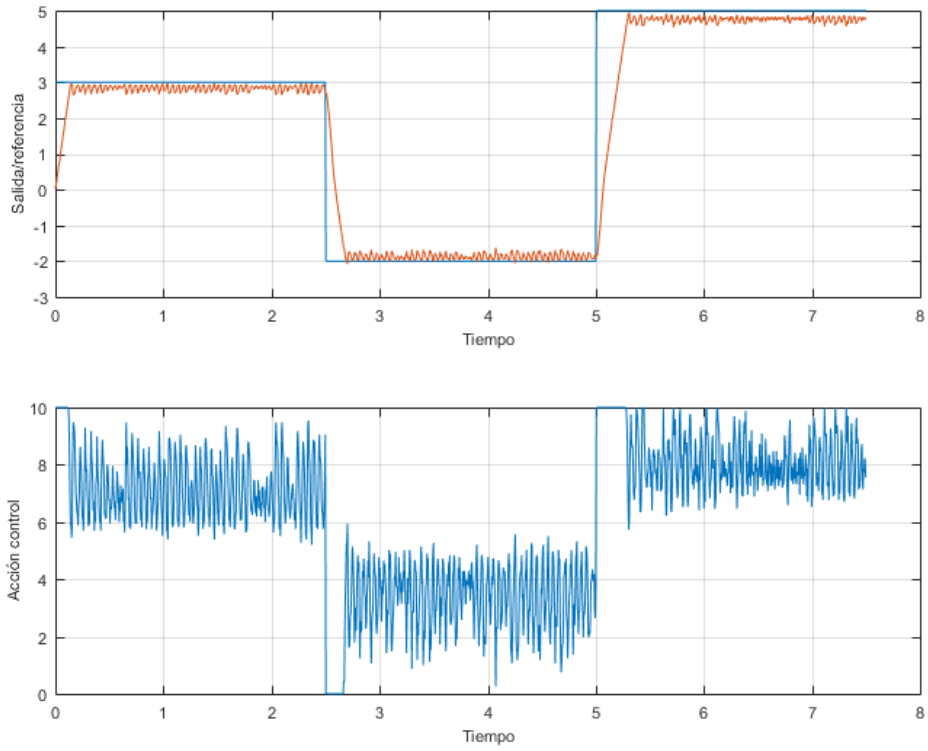

Fig. 39: Respuesta del motor (rojo, gráfica superior) con un control proporcional sin filtro en la acción de control (gráfica inferior, azul).

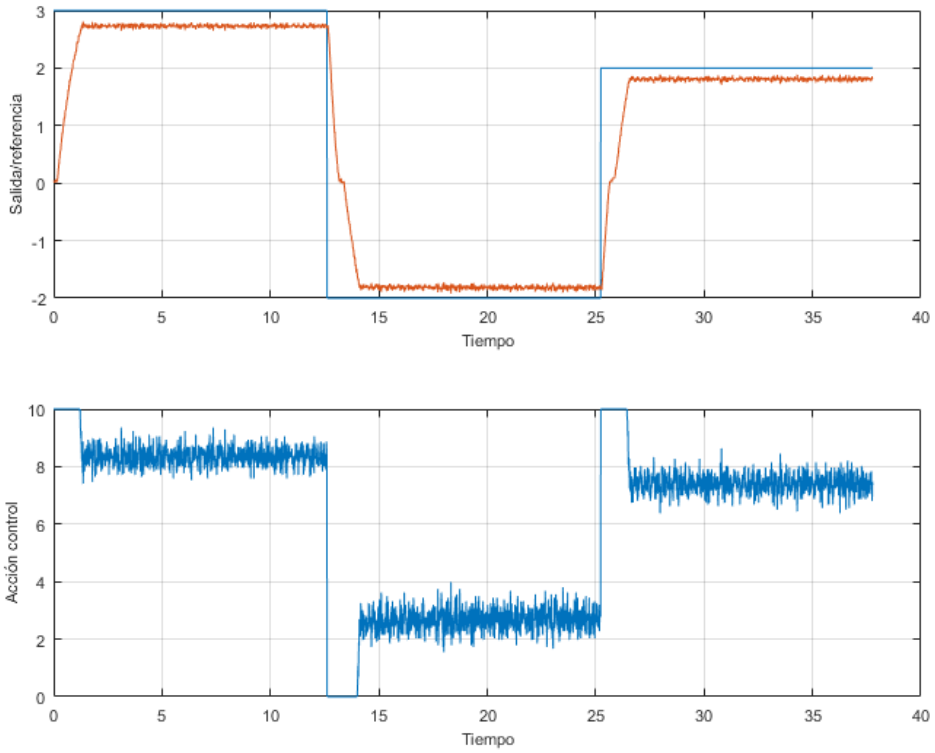

Fig. 40: Respuesta del motor (rojo, gráfica superior) con un control proporcional con filtro en la acción de control (azul, gráfica inferior).

Con objetivo de reducir el error de posición y reducir el problema con la zona muerta, se pasa a las pruebas con un controlador de tipo PI. Esto asegura que el error de posición sea nulo. En este caso, se puede comprobar (Fig. 41 y Fig. 42) que el error de posición es en efecto nulo, y que el filtro es en este caso más efectivo. También se observa en el caso del filtro una pequeña sobreoscilación, debido al efecto conocido como *windup* (Fig. 43), producido al acumularse error integral durante un tiempo hasta llegar a la referencia. Llegado a este punto, no parece necesaria la implementación de un controlador PID, ya que los resultados dinámicos parecen muy buenos, simplemente requiere la correcta elección de los parámetros.

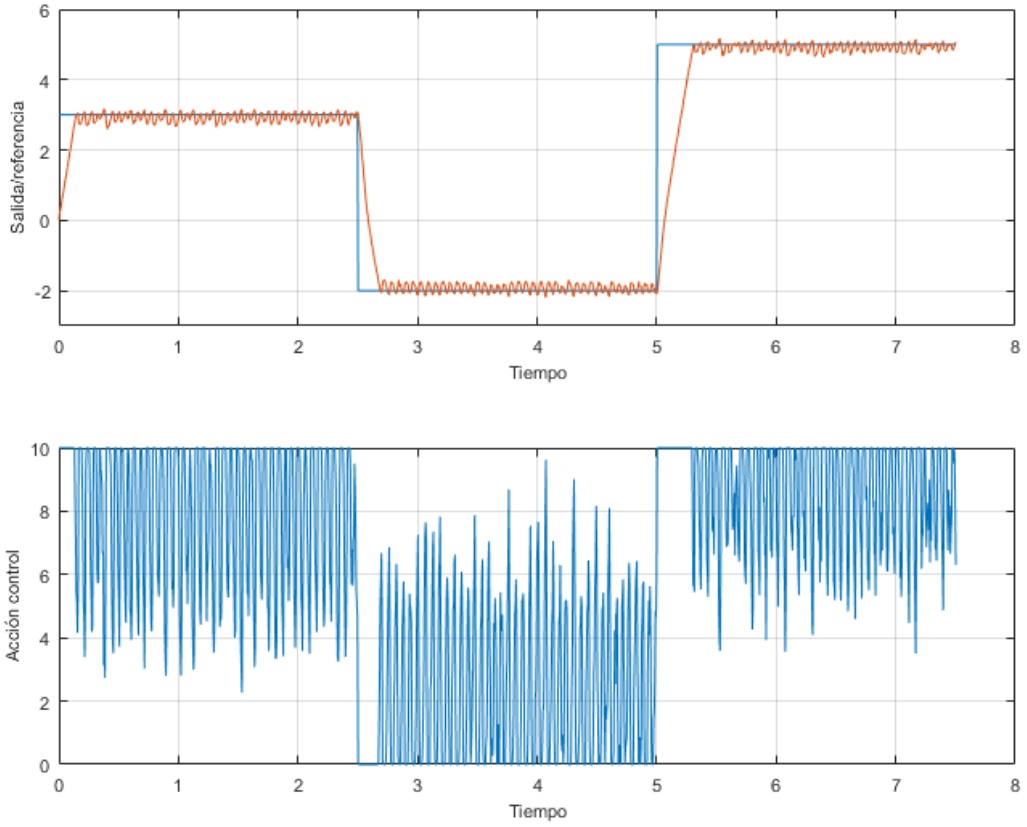

Fig. 41: Respuesta del motor con un control PI (rojo, gráfica superior) sin filtro en la acción de control (azul, gráfica inferior).

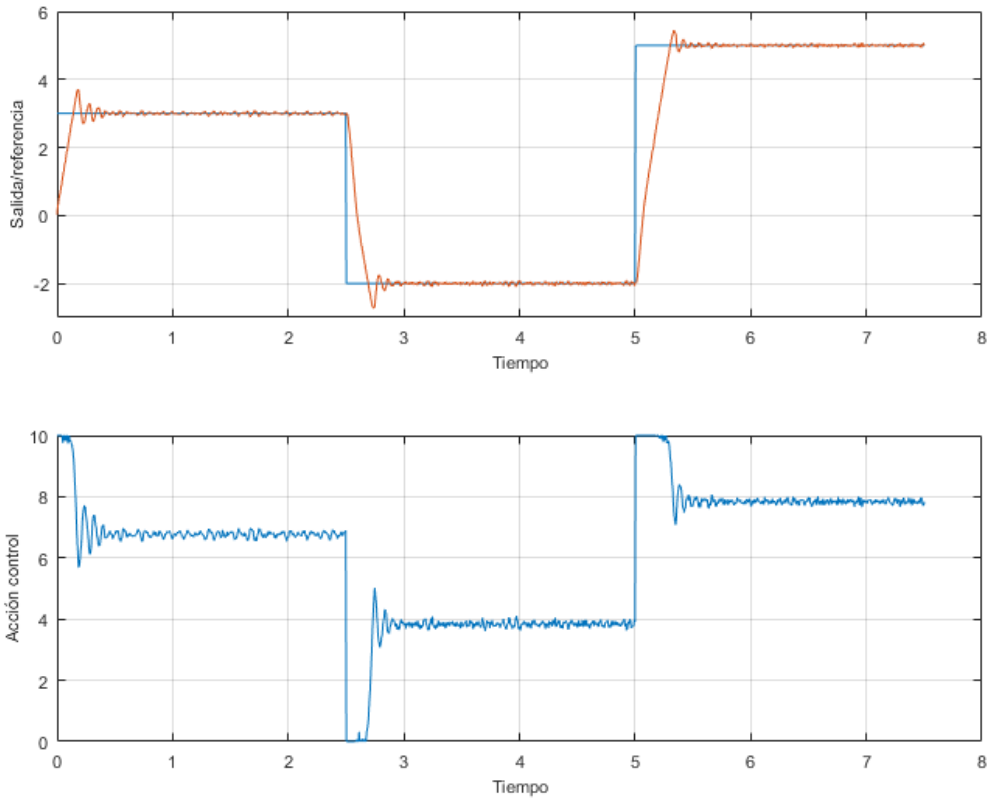

Fig. 42: Respuesta del motor con un control PI (rojo, gráfica superior) con filtro en la acción de control (azul, gráfica inferior).

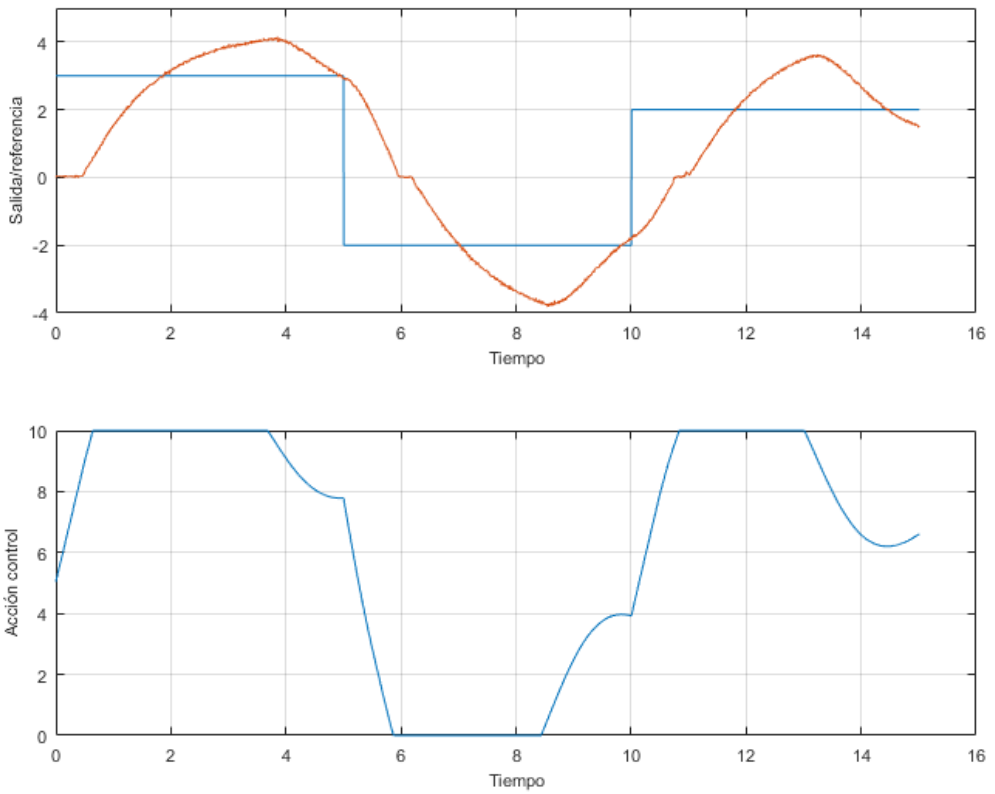

Fig. 43: Respuesta del motor controlado con un regulador PI. Se puede apreciar el efecto del *windup* debido a una acción integral demasiado grande.

Para el diseño de los parámetros se empleará la utilidad *PID Tuner*, con objetivo de diseñar un PI con un compromiso de velocidad y robustez. Esta herramienta dispone de varias opciones ajustables rápidamente, y con la facilidad de poder exportar el resultado del diseño para probarlo fácilmente (Fig. 44). Mediante el uso de esta herramienta y pruebas reales, se obtiene el control final, pudiendo realizar un ajuste fino de los parámetros dinámicos fácilmente.

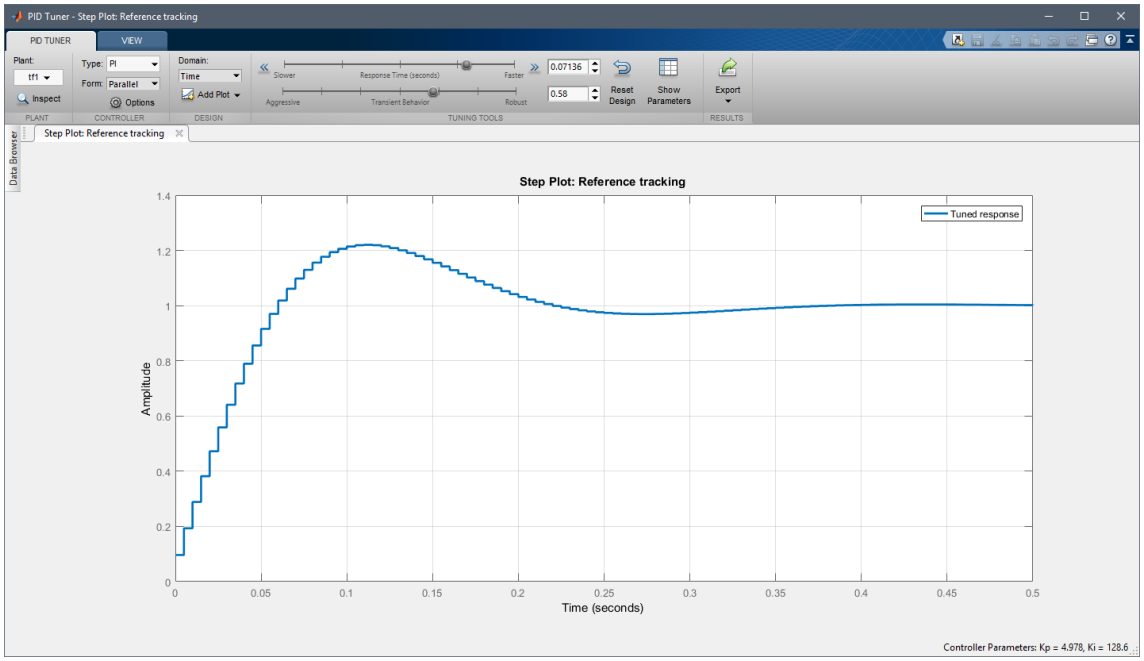

Fig. 44: Ventana de la aplicación *PID Tuner* de la *toolbox* de control lineal de MATLAB. Se pueden observar las diferentes opciones para el ajuste de la función de transferencia.

4.3. Control tipo *fuzzy*

Un control de tipo *fuzzy* o borroso, no se implementa a partir de modelos como tal, sino a partir del conocimiento previo sobre la planta. Se pueden plantear varios tipos de control borroso, dependiendo de las entradas y salidas elegidas. En este caso, se plantea un control de una entrada y una salida, la entrada será el error respecto a la referencia y la salida el incremento de acción de control que debe llevar a cabo.

El funcionamiento de este tipo de control se basa en conjuntos de valores arbitrarios, como por ejemplo, “mucho error” o “poco error”. Se clasifica la entrada mediante funciones de pertenencia que se corresponden con unos rangos de valores. Una misma entrada puede activar varias funciones de pertenencia simultáneamente, de forma que pertenezca a varios conjuntos. Mediante un conjunto de reglas se opera con estos valores para obtener varias salidas que se agrupan para dar la salida del control borroso utilizando las funciones de defuzificación.

Los elementos a diseñar en un control borroso son las funciones de pertenencia de las entradas y salidas, y los métodos empleados para el cálculo de las relaciones entre las pertenencias. En este caso nos vamos a centrar en la elección de las funciones de pertenencia.

Para la salida, se eligen cinco funciones de pertenencia, en función del sentido e intensidad de la acción de control. Las posibilidades serán: acción de control muy positiva, positiva, cero, negativa y muy negativa. Los rangos de estas acciones de control dependerán en gran medida del tiempo de establecimiento deseado y del tiempo de muestreo, ya que al ser incrementos de la acción de control la velocidad de esta se verá afectada por el tiempo de muestreo (Fig. 45). En el diseño final se considera alcanzar la máxima acción de control en 1 segundo, de forma que para un periodo de muestreo de 0.1s, se alcance en 10 muestreos de tiempo, por tanto, partiendo desde 5 V de salida (velocidad nula), hasta llegar a 10 V (velocidad máxima) se requieren 5 V, que divididos entre los 10 muestreos, dan como resultado un rango de ±0.5 V en los incrementos de la acción de control (Fig. 46).

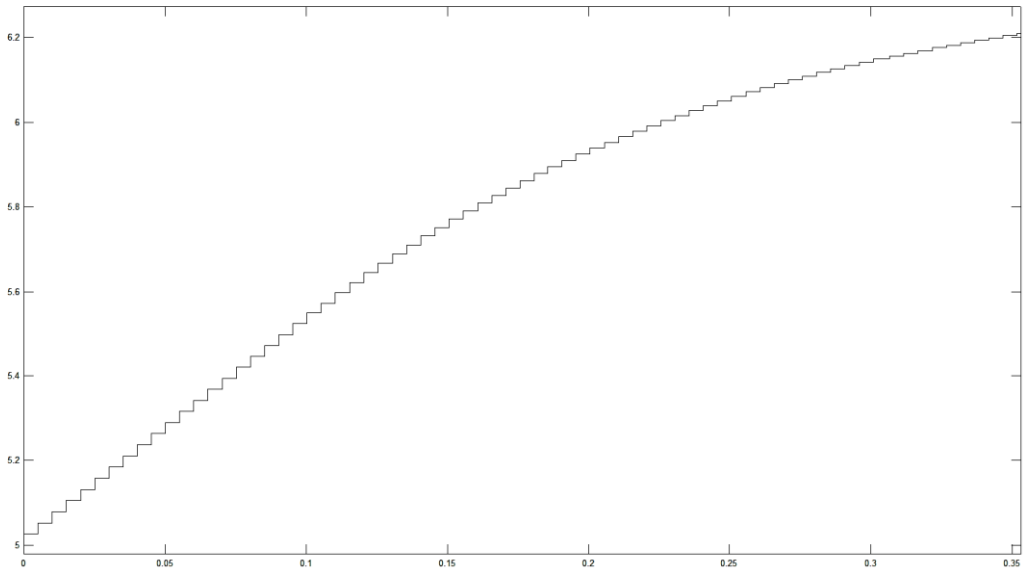

Fig. 45: Respuesta temporal simulada de un control de tipo borroso. Debido a que los incrementos de la acción de control en cada periodo de muestreo son muy limitados, la respuesta del sistema es lenta.

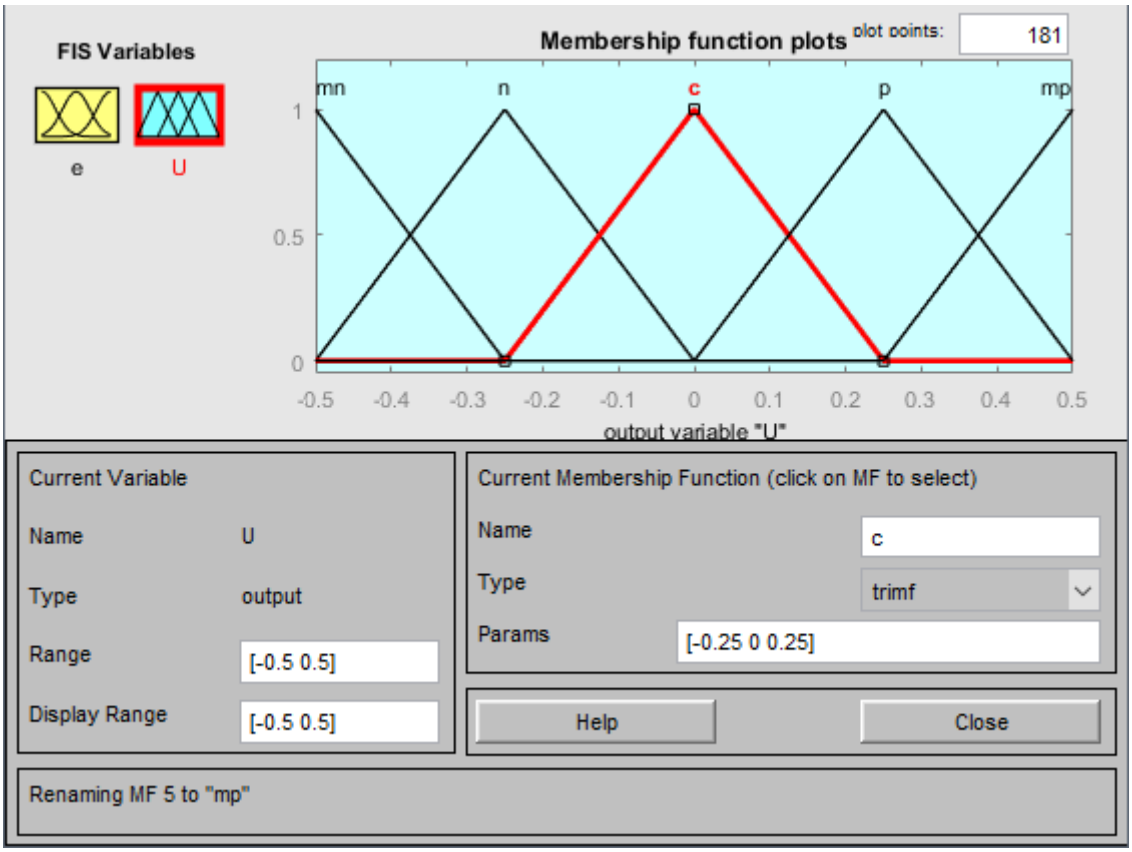

Fig. 46: Configuración de las funciones de pertenencia de las salidas del control borroso mediante la aplicación de MATLAB. El rango de incrementos de la acción de control variará entre -0.5 V y +0.5 V

Para la entrada también se eligen cinco funciones de pertenencia, de forma que las reglas de relación entre funciones de pertenencia de entradas y salidas sean fáciles de implementar. De igual manera que las salidas, las etiquetas se clasificarán desde mucho error positivo hasta mucho error negativo. En este caso, los rangos del error no dependen del tiempo de muestreo, así que sus rangos tampoco vienen condicionados por este. Además, en este caso se pretende conseguir una distinción entre rangos, empezando a considerar como mucho error diferencias mayores a 1 V en valor absoluto e incrementándose a partir de este, y como poco error entre 5 V y 0 V. (Fig. 47) Esto consigue crear una zona de mayor pendiente en los valores próximos a cero (Fig. 48), que es útil para prevenir comportamientos no deseados en las zonas próximas a la zona muerta del motor, ya que el comportamiento es más rápido en esta zona.

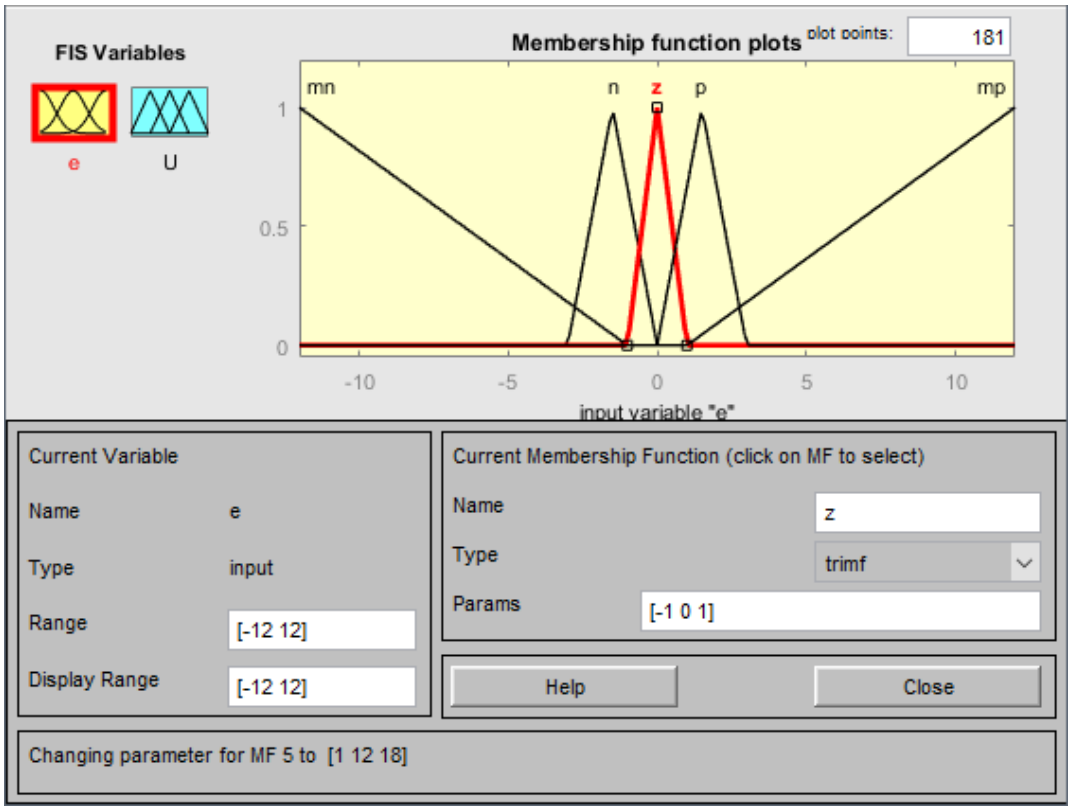

Fig. 47: Configuración de las funciones de pertenencia de las entradas del control borroso mediante la aplicación de MATLAB. Se proponen diferentes rangos en las entradas para conseguir acciones de control más fuertes que con rangos distribuidos linealmente en valores próximos a cero.

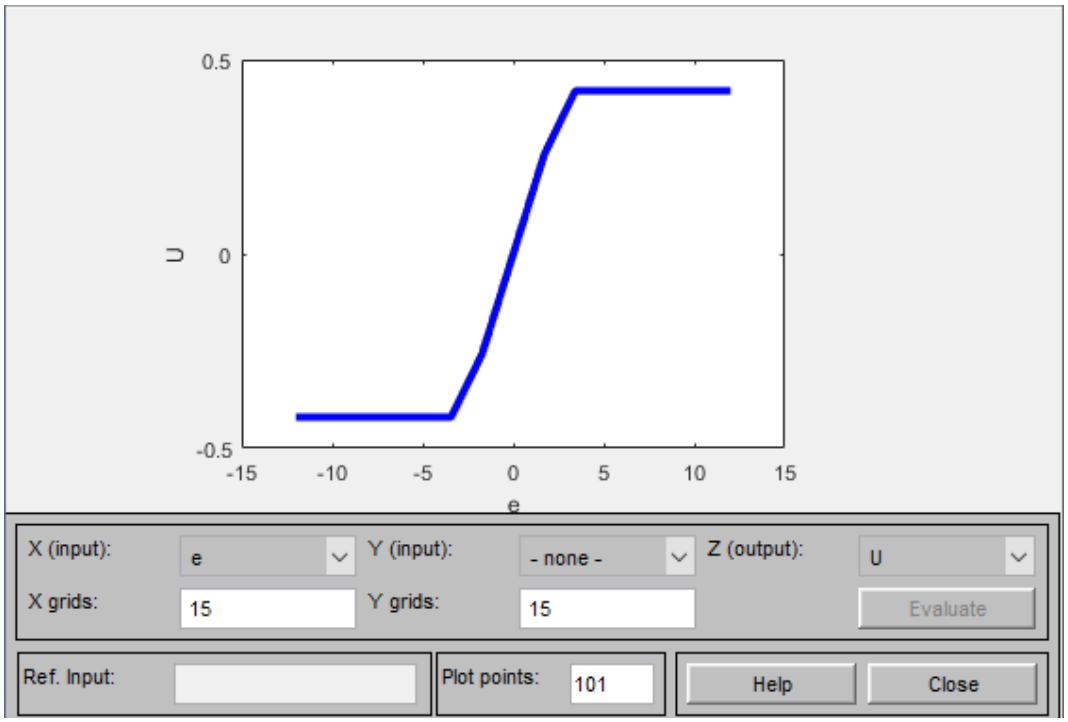

Fig. 48: Visualización de la respuesta del control borroso ante diferentes entradas de error. Se puede apreciar una mayor pendiente en valores de error próximos a 0.

Durante el estudio en el laboratorio, los resultados con esta configuración los resultados fueron satisfactorios, ya que las acciones de control son poco agresivas, aunque bastante rápidas. También se puede apreciar cómo se evita la zona muerta gracias a su mayor pendiente en esta zona de trabajo, dando lugar a una pequeña sobreoscilación en la acción de control (Fig. 49).

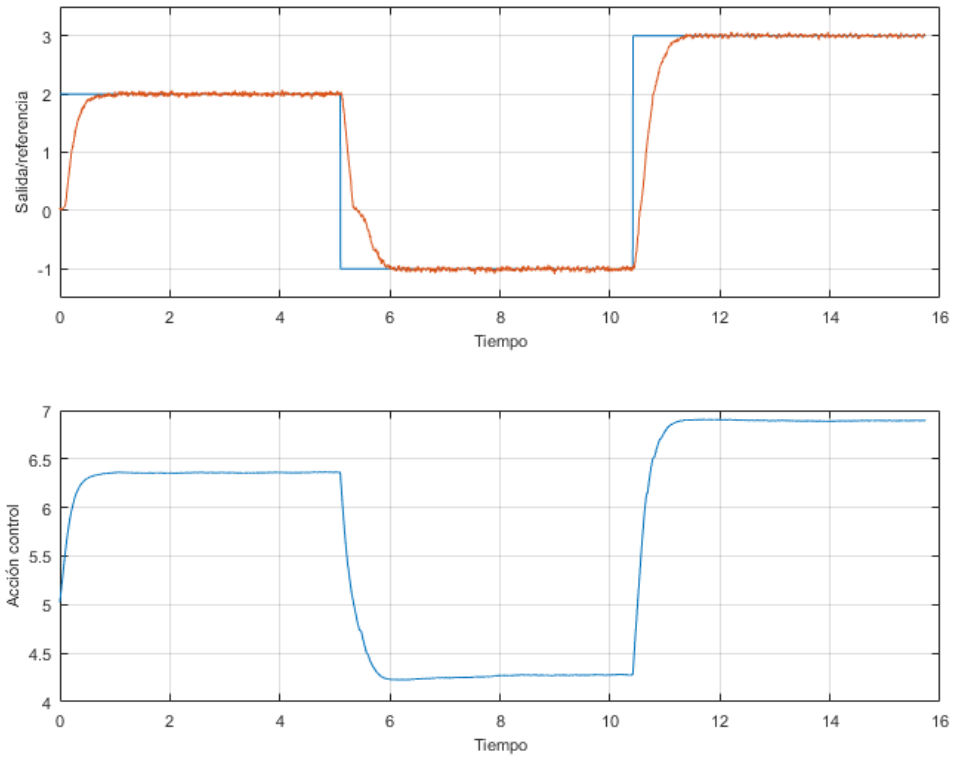

Fig. 49: Respuesta real del motor empleando el control borroso diseñado (Gráfica superior, naranja) ante una referencia de velocidad (Gráfica superior, azul). La acción de control (Gráfica inferior) es bastante rápida y suave, sin grandes oscilaciones excepto para superar la zona muerta a partir del segundo 5.

Hasta llegar a esta función final, se produjeron varios errores asociados a la no correcta selección de los parámetros. En la siguiente figura (Fig. 50) se puede observar un comportamiento muy lento ante errores pequeños, debido a que el rango de los incrementos de la acción de control era menor, provocando que tardase mucho en alcanzar la referencia. Además, el rango de error muy positivo estaba limitado a 5, provocando que en el primer tramo no se llegase a pertenecer a error muy positivo y por tanto se ralentizase aún más la acción de control. También se aprecia cierto efecto de *windup* en la tercera parte, ya que los incrementos negativos de la acción de control eran demasiado lentos para corregir este problema.

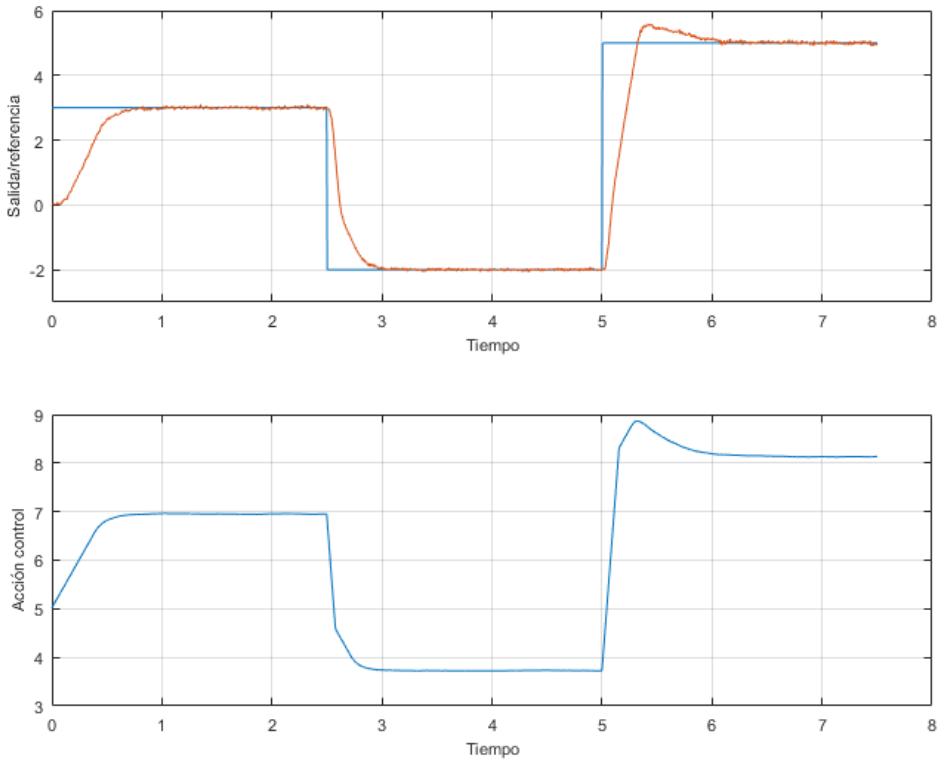

Fig. 50: Respuesta real del motor empleando un control borroso previo al final (Gráfica superior, Naranja) ante una referencia de velocidad (Gráfica superior, Azul). La respuesta es más lenta que la del control diseñado. La acción de control (Gráfica inferior) muestra comportamientos similares al *windup a partir del segundo 5.*

5. Conclusiones y trabajo futuro

5.1. Comparación de resultados de control

En este apartado se van a tratar las diferencias y similitudes entre el control mediante PID y el control mediante redes neuronales. Se han elegido comparar estos dos métodos porque el control borroso tal y como se ha planteado es muy parecido a un control de tipo P, aunque con algunas ventajas que ya se han mencionado antes como la posibilidad de variar la constante proporcional según la zona de trabajo. Para comparar los métodos de control, se han elegido los siguientes aspectos:

Modelado: En esta fase previa al control, es importante destacar varios aspectos diferenciales. Por una parte, la identificación y modelado de sistemas dinámicos mediante función de transferencia con varias zonas sustancialmente no lineales es especialmente complicado. Esto puede llevar al control PID a no trabajar correctamente en estas zonas. Por otro lado, las redes neuronales permiten entrenar cada zona por separado mientras se tengan suficientes datos de estas, pero el modelo puede fallar mucho al trabajar fuera de la zona para la que ha sido entrenado.

Desde un punto de vista práctico, en algunas situaciones no es viable obtener datos de la planta con un tren de escalones, debido a problemas del sistema o simplemente a que el proceso no debe ser detenido por necesidades industriales. Aunque es posible realizar la identificación mediante función de transferencia, es más complicado que el modelado mediante redes neuronales, que pueden ser entrenadas con los mismos datos del funcionamiento habitual del proceso.

Comportamiento dinámico: El comportamiento dinámico de ambos métodos es bastante similar, excepto en algunos aspectos concretos. En primer lugar, dada la estructura de las redes neuronales, existe un retardo que afecta al tiempo de respuesta del sistema. Esto hace que el tiempo de establecimiento sea ligeramente más lento en esta clase de control, aunque existen varias posibilidades para corregir esto si la referencia es conocida con antelación (Fig. 51).

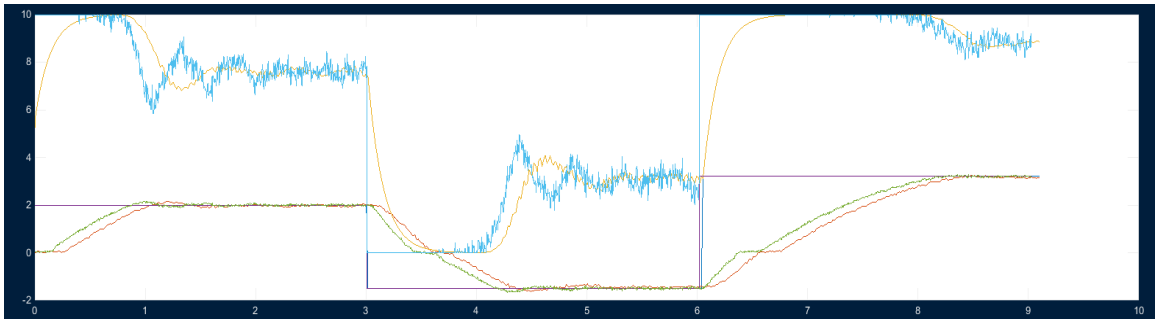

Fig. 51: Respuesta del motor controlado mediante PI (verde) y redes neuronales (naranja) siguiente una referencia de velocidad (morado). Se puede observar como la acción de control de las redes (amarillo) menos ruidosa que la del PI (azul).

Por otro lado, las acciones de control generadas han sido diferentes. El PID presenta acciones de control muy ruidosas, ampliamente afectadas por pequeños ruidos del sensor. Esto puede acabar afectando a la vida útil del proceso a la larga, y incluso con el efecto del filtro paso bajo se mantenía un considerable nivel de ruido. Las redes neuronales también han presentado este problema en determinados entrenamientos, pero una vez conseguido un buen entrenamiento del control, este ruido es mucho menor que el del PID (Fig. 52).

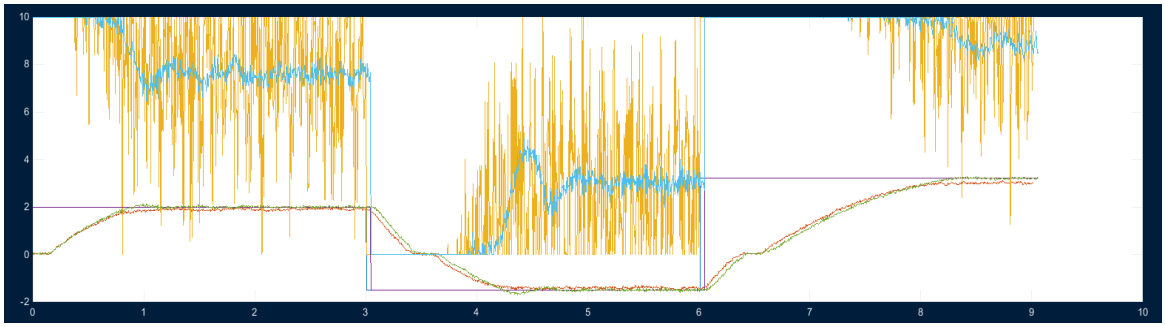

Fig. 52: Respuesta del motor controlado con un PI sin filtro (naranja) y con filtro (verde). Se observa que la acción de control (amarillo) es filtrada y se reducen sus oscilaciones en gran medida (azul).

Comportamiento en zonas no lineales: Para comprobar este aspecto del control, se ha probado el control en velocidades muy bajas, de modo que el control debe trabajar en voltajes próximos a la zona muerta del motor. En estas condiciones, el control PID consigue el control, pero presenta oscilaciones considerables. Estas son posiblemente debidas al efecto de la parte integral, que aumenta la acción de control hasta sobrepasar la zona muerta, pero acumulando más error hasta que se llega a este punto, de forma que sobrepasa la referencia. También es posible que sea consecuencia del ruido en la acción de control explicado en el apartado anterior.

Las redes neuronales en cambio con un entrenamiento general consiguen un mejor resultado, dado que el control está entrenado para llevar la salida hasta la referencia en estas condiciones. En esta figura se puede apreciar como las oscilaciones son del orden de un tercio comparadas con las del PID (Fig. 53).

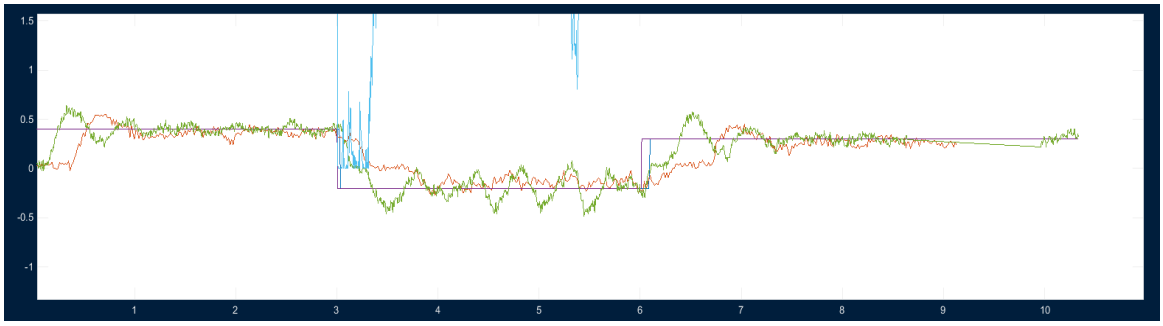

Fig. 53: Respuesta del motor controlado mediante PI (verde) y redes neuronales (naranja) siguiente una referencia de velocidad de una amplitud reducida (morado). Se observan observaciones reducidas en la respuesta ante redes frente a las grandes oscilaciones del PI.

Robustez del control: Este es el aspecto donde más destaca el PID respecto a las redes neuronales, y el aspecto que hace que sea uno de los controles más empleados a nivel industrial. Frente a un cambio del modelo, como añadir resistencia simulando un desgaste del motor, el PID es un método fiable para obtener controles estables. En cambio, las redes neuronales no consiguen compensar esta situación sin ser reentrenados para ello, y presentan error de posición al no esperar esta resistencia adicional y no tener ningún método para compensarla (Fig. 54).

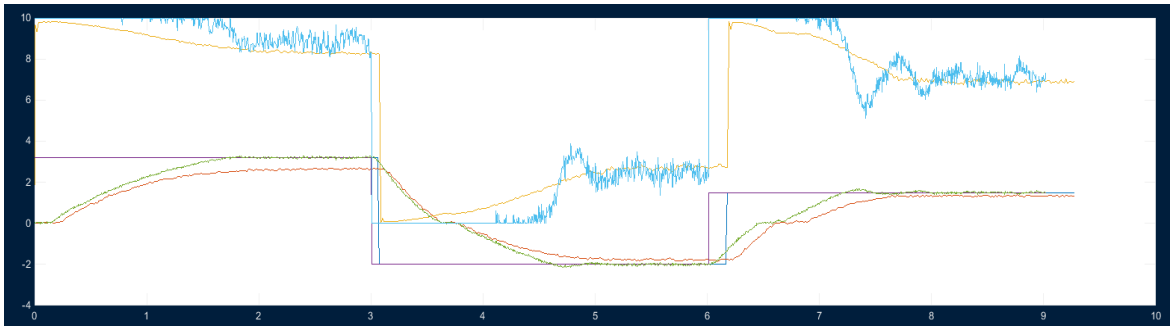

Fig. 54: Respuesta del motor controlado mediante PI (verde) y redes neuronales (naranja) siguiente una referencia de velocidad (morado) ante un cambio en el modelo. En el caso de las redes neuronales no se alcanza la referencia deseada. La acción de control de las redes (amarillo) no alcanza el nivel necesario para ello, frente a la mayor acción del PI (azul) gracias a su parte integral.

Ha sido posible realizar la implementación de un control de un motor de corriente continua mediante redes neuronales. Se han comparado sus características con los de un control tradicional como el control PID (Tabla 1):

	Modelado	**Comportamiento dinámico**	**Comportamiento en zonas no lineales**	**Robustez**
PID	Puede llegar a ser complejo e impreciso en las zonas no lineales	Ruido en la acción de control	Oscilaciones importantes	Comportamiento adecuado ante cambios en el modelo
Redes	Sencillo, pero impreciso fuera de la zona entrenada	Poco ruido en la acción de control, pero existen retardos	Oscilaciones suaves	Presenta error de posición al cambiar el modelo sin ser reentrenado

Tabla 1: Resultados al comparar control PID con control mediante redes neuronales para comportamiento dinámico, comportamiento en zonas no lineales y robustez.

La construcción de este modelo es fácilmente reproducible para diferentes motores, se requiere una gran cantidad de datos y para que el programa desarrollado entrene el modelo. Es por tanto un método rápido para conseguir modelos para un conjunto de motores, haciendo posible un control individualizado para cada motor aunque sean ligeramente diferentes, dada la naturaleza adaptativa de las redes neuronales.

Sin embargo, el control mediante redes neuronales puede ser insuficiente a nivel industrial, ya que su robustez, su capacidad de adaptarse a cambios en el modelo, no es destacable y presenta grandes problemas al ser usado fuera de su zona de entrenamiento. Sin embargo, demuestra potencial combinado con otros métodos o para usos complementarios al control, como la supervisión.

5.2. Trabajo futuro

Con objetivo de analizar en más profundidad el método de control basado en las redes neuronales, se plantean las siguientes alternativas de control y supervisión:

Implementación de control de posición del motor: Esta alternativa nos permitiría comprobar con mayor profundidad las capacidades de las redes neuronales para enfrentarse a la zona muerta del motor, puesto que en el control de posición se partiría de velocidad nula al cambiar de posición. Además, se probaría su comportamiento en lo referente a oscilaciones y estabilidad.

Implementación de control en un motor de alterna: Un camino alternativo sería el control de un motor de alterna mediante redes neuronales, pudiendo regular el par del motor y su velocidad de trabajo. También puede plantearse la detección y prevención de fallos en un motor de este tipo, ya que muchos se pueden monitorizar mediante sensores sencillos.

Implementación de un PID con parámetros controlados mediante redes neuronales: Otra posibilidad es combinar ambos métodos de forma que, al cambiar la zona de trabajo, cambien los parámetros del PID on-line, para conseguir los objetivos deseados en cada zona. Este método comprobaría las ventajas de robustez de este sistema y el aprendizaje para cada zona de las redes neuronales.

Aprendizaje de rutinas de posición basadas en el ejemplo del usuario: Se propone esta alternativa para el aprendizaje de trabajos repetitivos de posición guiados por el usuario, de forma que se entrene la red neuronal con varios ejemplos de posiciones, y esta se encargue de reconstruir la referencia a seguir en base a estos entrenamientos.

Mantenimiento del motor: Mediante el uso de redes neuronales es posible reconocer patrones en la salida para la detección del funcionamiento normal e irregular. Esto permite predecir el fallo del motor antes de que ocurra y realizar mantenimiento de este o substituirlo evitando pérdidas mayores.

6. Presupuesto

El coste total de este proyecto es de 12.457,00 € (sin IVA). El precio base de las horas trabajadas en el laboratorio se corresponde al de un técnico industrial, e incluyen la programación y diseño de la aplicación.

Descripción	Cantidad	Precio	Coste
Mano de obra			
H. Técnico Industrial	124	70,00 €	8.680,00 €
Software			
Ud. Software MATLAB (Licencia anual Educación)	1	250,00 €	250,00 €
Materiales			
Ud. Motor CC Artitecnic	1	2.500,00 €	2.500,00 €
Ud. Tarjeta adquisición de datos	1	215,00 €	215,00 €
Ud. Ordenador personal y periféricos	1	800,00 €	800,00 €
Ud. Cable conector Banana-Banana	4	3,00 €	12,00 €
Resumen			
Total Mano de obra			8.680,00 €
Total Software + Total Materiales			3.777,00 €
Total			**12.457,00 €**

7. Bibliografía y referencias

Base teórica

- Jyh-Shing Roger Jang, Chuen-Tsai Sun, Eiji Mizutani (1997). Neuro-Fuzzy and Soft Computing. A Computational Approach to Learning and Machine Intelligence.
- Colaboradores de Wikipedia. *Artificial neural network* [en línea]. Wikipedia, La enciclopedia libre, 2018 [fecha de consulta: Mayo del 2018]. Disponible en <https://en.wikipedia.org/wiki/Artificial_neural_network>

Información sobre MATLAB

- MathWorks. Información general sobre el software MATLAB [en línea]. Página oficial MATLAB, 2018 [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/products/matlab.html>

***Toolboxes* de MATLAB**

- [1] MathWorks. Información general sobre GUI en MATLAB [en línea]. Página oficial de la *toolbox* para la construcción de GUI en MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/discovery/matlab-gui.html>
- [2] MathWorks. Información general sobre la *toolbox* de redes neuronales MATLAB [en línea]. Página oficial sobre la *toolbox* de redes neuronales de MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/products/neural-network.html>
- [3] MathWorks. Información general sobre la *toolbox* de identificación de MATLAB [en línea]. Página oficial sobre la *toolbox* de identificación de MATLAB, 2018 [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/products/sysid.html>
- [4] MathWorks. Información general sobre la *toolbox* de lógica difusa de MATLAB [en línea]. Página oficial sobre la *toolbox* de lógica difusa de MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/products/fuzzy-logic.html>
- [5] MathWorks. Información general sobre la *toolbox* de sistemas de control de MATLAB [en línea]. Página oficial sobre la *toolbox* de sistemas de control de MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/products/control.html>

Información sobre la DAQ

- [6] ADLINK Technology. Página de compra del modelo ACL-8112 Series [en línea]. Página oficial de ADLINK Technology. MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://www.adlinktech.com/Products/Data_Acquisition/Data_Acquisition/ACL-8112_Series>

Información sobre el entrenamiento de redes neuronales

- [7] MathWorks. Explicación del funcionamiento del entrenamiento por defecto usado por MATLAB para redes neuronales y sus parámetros [en línea]. Página oficial sobre trainlm de MATLAB [fecha de consulta: Mayo del 2018]. Disponible en <https://es.mathworks.com/help/nnet/ref/trainlm.html>
- [8] Matt Mazur. Ejemplo del entrenamiento de una red neuronal mediante el algoritmo de *backpropagation* [en línea] A Step by Step Backpropagation Example [fecha de consulta: Mayo del 2018]. Disponible en <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>
- [9] Rohan Kapur. Redes neuronales y el algoritmo de *backpropagation* [en línea] Rohan & Lenny #1: Neural Networks & The Backpropagation Algorithm, Explained [fecha de consulta: Mayo del 2018]. Disponible en <https://ayearofai.com/rohan-lenny-1-neural-networks-the-backpropagation-algorithm-explained-abf4609d4f9d>

Anexo código

- Se adjunta el código de la interfaz gráfica en la carpeta *TFG* del CD adjunto.