UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE MÁSTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UNA GUI EN MATLAB PARA EL ESTUDIO Y CONTROL DE VARIADORES DE FRECUENCIA EN MOTORES ELÉCTRICOS

AUTOR: PAULA ABELLÁN OLIVARES

TUTOR: ÁNGEL SAPENA BAÑÓ

COTUTOR: RUBÉN PUCHE PANADERO

Curso Académico: 2019-20

AGRADECIMIENTOS

“Agradecer el apoyo de mi familia y
amigos en este año tan atípico”

RESUMEN

El presente documento describe el desarrollo de un interfaz GUI mediante el entorno de programación de Matlab, la cual permite controlar el modelo de un variador de velocidad para motores asíncronos. Mediante esta interfaz se podrán configurar los parámetros del motor, así como de las señales de entrada, de los modelos de conmutación del puente de conversión AC/DC y del filtro LC previo al motor. Los resultados se mostrarán mediante gráficas, con las cuales el usuario podrá interactuar para seleccionar la señal que se quiere graficar, ampliar dicha señal e incluso, controlar un cursor vertical que se desplace por la señal, indicando el instante de tiempo en el que se encuentra.

Palabras clave: Matlab, GUIDE, Simulink, GUI, variador de frecuencia, motor asíncrono.

RESUM

El present document descriu el desenvolupament d'una interfície GUI de l'entorn de programació de Matlab, amb la qual es controlarà el model d'un variador de freqüència per a motors asíncrons dissenyat en Simulink. Mitjançant aquesta interfície es podrà configurar els paràmetres del motor, així com dels senyals d'entrada, dels models de commutació del pont de conversió AC/DC i del filtre LC previ al motor. Els resultats es mostraran mitjançant gràfiques, amb les quals l'usuari podrà interactuar per a seleccionar el senyal que es vol graficar, ampliar aquest senyal i fins i tot, controlar un eix vertical que es desplaça per tot el senyal, indicant l'instant de temps en el qual ens trobem.

Paraules clau: Matlab, GUIDE, Simulink, GUI, variador de freqüència, motor asíncron.

RESUM

This document describes the development of a GUI interface of the Matlab programming environment, which controls the model of a variable speed drive for asynchronous motors. This interface will be used to configure the motor parameters, as well as the input signals, the switching models of the AC/DC conversion bridge and the LC filter. The results will be shown by means of graphics, with which the user can interact to select the signal to be graphed, amplify this signal and even control a vertical cursor that moves along the entire signal, indicating the instant of time.

Keywords: Matlab, GUIDE, Simulink, GUI, frequency inverter, asynchronous motor.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

Una de las asignaturas más comunes en el estudio de la ingeniería industrial, es el control de máquinas eléctricas. Esta asignatura recopila multitud de conceptos relacionados con el diseño, estudio y control de motores eléctricos, y es por ello que tiene una gran importancia en el sector industrial.

Este proyecto surge a raíz de las dificultades que muestran los alumnos para el entendimiento de algunos conceptos fundamentales en el estudio de las máquinas eléctricas. En este caso, el objetivo se ha centrado principalmente en el control de motores eléctricos a partir de variadores de frecuencia.

Con la intención de crear un método de aprendizaje más interactivo y visual, se ha creado una interfaz de usuario donde los alumnos podrán diseñar y visualizar los procesos que forman parte del control de una máquina eléctrica. En este proyecto, el control se basa en un variador de frecuencia trifásico con un puente inversor de dos niveles que puede ser modulado a partir de dos métodos: SPWM (*Senoidal Pulse Width Modulation*) y SVPWM (*Space Vectorial Pulse Width Modulation*). Estos dos métodos representan los criterios de modulación más utilizados en el control de puentes inversores trifásicos de 2 niveles. Mientras que el método PWM se encarga de modular cada una de las fases por separado a partir de la comparación entre una señal portadora y una moduladora, el método SVPWM está basado en vectores espaciales, donde se trabaja con un único modulador sobre el vector espacial que representa a la señal trifásica del conjunto.

Toda la configuración y parametrización de los componentes podrá ser manipulada por los alumnos desde la interfaz, donde también se podrán ir visualizando todos los procesos por los que pasa la señal hasta llegar a la alimentación de la máquina. Los resultados se han dispuesto en gráficos con la intención de fomentar un aprendizaje más dinámico y visual.

1.1. DESCRIPCIÓN

En el presente documento se va a describir el diseño, funcionamiento y finalidad de una interfaz GUI, creada a partir del software Matlab, para el estudio y modelado de variadores de frecuencia utilizados en el control de máquinas eléctricas, desde un punto de vista didáctico.

Para el desarrollo del proyecto se ha trabajado con los entornos de programación Simulink y GUIDE, complementarios al software Matlab. A partir de la herramienta Simulink se ha diseñado un modelo de variador de frecuencia con un motor eléctrico, donde se recogen los distintos

métodos de modulación para la etapa de conversión DC/AC implementada mediante un puente de IGBTs de 2 niveles. Por otro lado, mediante la interfaz creada de GUIDE se pueden introducir los datos de entrada de la señal, así como los valores del modelo del motor eléctrico y la modulación requerida. Una vez configurados los valores del modelo, este podrá arrancarse para, posteriormente, mostrar las señales resultantes en los diferentes gráficos de la interfaz. Para la programación de la herramienta se ha trabajado con el entorno de Matlab, donde se han definido los comandos a ejecutar en cada una de las funciones de control que engloban los bloques.

1.2. OBJETIVO

Esta interfaz se ha creado con fines académicos, con el **objetivo principal** de diseñar una plataforma interactiva y de aprendizaje para el alumno, donde poder comprender las características principales del control y modelación de las máquinas eléctricas, de una manera participativa y dinámica.

Para lograr este fin principal, se partirá de unos **objetivos parciales** que compondrán las fases de actuación del presente proyecto.

- Diseño de un modelo de variador de frecuencia para motores eléctricos a partir de la herramienta de simulación, Simulink. Este modelo debe englobar una etapa rectificadora de conversión AC/DC, un filtrado de continua, una etapa inversora DC/AC controlada por diferentes criterios de modulación, una etapa de filtrado de la señal de salida y un modelo de motor eléctrico.
- Creación de una interfaz de usuario que permita la configuración y parametrización de los diferentes bloques del modelo del variador; así como la visualización gráfica de la señal en cada una de las fases del proceso de control. El diseño de la aplicación se hará mediante la herramienta de interfaces, GUIDE.
- Relacionar el modelo de Simulink con la herramienta de GUIDE de manera que ambos puedan interactuar entre sí y se permita realizar toda la configuración y manipulación del modelo utilizando únicamente la interfaz de usuario.
- Programación de las funciones que controlan cada uno de los bloques diseñados en la interfaz, teniendo en cuenta que se tratan de funciones independientes que se ejecutarán únicamente en los instantes en los que el usuario esté interactuando con ese bloque.
- Añadir bloques para la manipulación de gráficas, con el fin de que el usuario pueda visualizar los resultados con la mayor comodidad posible. Para esta finalidad también se agregará una barra de herramientas a la interfaz, donde estarán disponibles multitud de opciones adicionales.
- Realización de ejemplos de manipulación de la interfaz, con la intención de explicar el correcto funcionamiento de la misma, así como la diversidad de herramientas disponibles.

1.3. ANTECEDENTES

La asignatura de máquinas eléctricas contiene multitud de conceptos importantes en el día a día de las ingenierías, sin embargo, puede llevar a contener conocimientos que resultan bastante complicados para el estudio de los alumnos. Es por ello, que se requiere de una herramienta educativa complementaria a la asignatura que sirva de ayuda para el estudio y comprensión del control de los motores eléctricos.

Las interfaces de usuario es una de las innovaciones tecnológicas más presente en el día a día actual de las personas. Es un medio de transmisión de información entre el hombre y la máquina muy visual y sencillo de interpretar, consiguiendo una productividad eficiente y cómoda.

Partiendo de esta idea de comodidad y claridad, se ha pensado en la idea de utilizar este tipo de herramientas con fines educativos, donde los alumnos puedan aprender y entender los conceptos a través de la interactuación con la interfaz.

CAPÍTULO 2. DESARROLLO

2.1. INTRODUCCIÓN

Con la intención de complementar los métodos de enseñanza actuales para el estudio de las máquinas eléctricas mediante métodos prácticos, se han barajado multitud de alternativas, donde destaca principalmente la distinción entre una herramienta física y material mediante bancos de prueba, o una opción virtual mediante simulaciones dinámicas.

Se ha descartado la alternativa de diseño de bancos de prueba físicos y se ha optado por la alternativa de una herramienta de simulación con la que poder diseñar un "banco de pruebas" virtual. Al crear una herramienta virtual se permite a los alumnos la posibilidad de disponer de un banco de ensayos en todo momento, en que poder simular diferentes criterios de modulación de una forma dinámica, rápida y sencilla. Además, cabe destacar las facilidades que aporta este método de ensayo, principalmente, a la hora de visualizar los resultados y estudiar los datos obtenidos. Desde una aplicación virtual es mucho más sencillo el proceso de toma de datos y gráfico de las señales que desde el modelo de un banco de pruebas físico, donde puede resultar bastante complicado alcanzar a medir señales o valores internos de ciertos componentes del modelo.

Finalmente, se ha tenido en cuenta que el objetivo principal del proyecto es la creación de una herramienta didáctica de soporte para la asignatura de máquinas eléctricas, por lo que no debe ser obligatoriamente un sustitutivo de los ensayos físicos, sino una herramienta extra que pueda utilizarse conjuntamente con una parte teórica y/o práctica.

2.1.1. Requisitos de la herramienta virtual

Para la selección de un software con el que diseñar y programar la herramienta de trabajo, se han considerado los requisitos más importantes a tener en cuenta a la hora de elegir entre los diferentes programas. A continuación, se enumeran todas las características que se han tenido en cuenta en la comparativa entre softwares.

- Fácil accesibilidad para todos los alumnos, por ejemplo, mediante el uso de licencias de estudiante que permitan utilizar las herramientas sin costes adicionales.
- Lenguaje programación e interfaz de funcionamiento conocidos por los alumnos, de tal forma que el uso que la herramienta resulte familiar al usuario y su manejo resulte lo más sencillo posible.
- Disponibilidad de diseño de interfaces de usuario.

- Disponibilidad de modelos predefinidos de motores eléctricos, así como de diferentes bloques de control y modulación de señales.
- Disponibilidad de bloques de medición y almacenamiento de señales.
- Diseño de las etapas de un variador frecuencia: etapa rectificadora, filtro de continua y etapa inversora.
- Capacidad de diseñar, mediante librerías y bloques internos, diferentes señales de modulación.
- Fácil interactuación entre la interfaz de usuario y el modelo de diseñado para el variador de frecuencia al ser herramientas internas de un mismo software.
- Configuración de diferentes parámetros del variador de frecuencia, así como de los filtros, a partir de la interfaz.
- Diseñar y configurar bloques gráficos donde plotear los resultados obtenidos en el modelo.

Con todos los requisitos enumerados, se van a barajar diferentes alternativas de software para el diseño y la programación del proyecto. Además, se va a estudiar tanto la posibilidad de trabajar un programa para el diseño del modelo del variador de frecuencia, y un programa distinto para el diseño y programación de la interfaz de usuario, como la posibilidad de trabajar con un único software capaz de contener herramientas para realizar ambos procesos.

2.1.2. Estudio de alternativas

Al seleccionar una alternativa virtual, se deberán diferenciar dos partes importantes dentro del proyecto, una parte donde se encuentre el modelo del variador de frecuencia el cual se va a parametrizar y simular, obteniendo unos resultados para su posterior estudio; y una interfaz de usuario con la que controlar y configurar el modelo, además de representar los resultados mediante múltiples gráficos.

Para la parte de creación del modelo dinámico del variador de frecuencia, se han estudiado tres posibles softwares que podrían realizar el diseño y la simulación del modelo requerido: PSIM, Simulink y PSCAD.

PSIM

Se trata de un software de simulación de circuitos electrónicos, especializado en la electrónica de potencia y, en accionamiento y control de motores eléctricos. Una de sus grandes ventajas es la diversidad de módulos adicionales, permitiendo así crear unos diseños mucho más completos y complejos. Cuenta con diferentes licencias, en función de la utilización educativa que sea requerida, pudiendo seleccionar licencias gratuitas para estudiantes.

Simulink

Este programa pertenece al software de cálculo Matlab, el cual cuenta con multitud de entornos y herramientas capaces de amoldarse a cualquier necesidad del usuario. La herramienta Simulink se basa en un entorno de diagramas de bloques con los que diseñar y simular gran diversidad de modelos dinámicos. El hecho de estar integrado en Matlab le permite incorporar algoritmos desde la propia consola de Matlab o, incluso, exportar resultados para un posterior análisis. Además, cabe destacar que, esta herramienta viene incluida en la licencia gratuita de estudiante que proporciona Matlab.

PSCAD

Se trata de una herramienta de diseño asistido por ordenador, especializada en el diseño y simulación de sistemas eléctricos. Cuenta con potentes paquetes de simulación que permiten análisis de redes tanto en corriente continua (CC), como en corriente alterna (CA). Destacan sus versiones gratuitas donde poder trabajar con el programa pese a tener ciertas limitaciones de herramientas.

Estos tres programas disponen de potentes herramientas de simulación con las que se podría diseñar un modelo de variador de frecuencia muy completo y con multitud de opciones. Sin embargo, la finalidad de este proyecto consiste en crear una herramienta que facilite a los estudiantes el entendimiento de los diferentes procesos de un variador de velocidad, visualizando los resultados gráficos que se obtienen tras pasar por cada una de las fases del modelo. Por ello, no se ha dado tanta importancia a la potencia de simulación de los programas sino a la variedad de alternativas que proporciona cada uno.

Por otra parte, es importante recordar que el objetivo principal no es que los alumnos aprendan a diseñar un modelo de control de motores eléctricos, sino que comprendan los resultados obtenidos del modelo en base a una teoría previa que se haya visto en clase. En por ello que se ha creado una interfaz de usuario donde el alumno pueda introducir unos datos de partida y observe los diferentes resultados en varios gráficos.

Nuevamente, para la creación de la interfaz de usuario se han barajado tres posibles softwares que podrían diseñar y programar la aplicación requerida: Visual Studio, Adobe XD y Guide.

Visual Studio

Se trata de un entorno de desarrollo integrado con multitud de lenguajes de programación diferentes. Una de sus herramientas más destacadas es la creación de aplicaciones de escritorio a partir del lenguaje de programación C#. La interfaz de trabajo para la creación de aplicaciones es muy cómoda y visual; además de completa, pues contiene una gran diversidad de bloques y propiedades con las que diseñar las aplicaciones de usuario. Este programa también dispone de licencias de estudiante con las que los alumnos pueden trabajar de forma totalmente gratuita.

Adobe XD

Es una herramienta del colectivo Creative Cloud Adobe, especializada en el diseño de aplicaciones e interfaces de usuario. A partir de Adobe XD se puede diseñar y programar las interfaces e, incluso, simular la experiencia del usuario. Parte de un diseño mucho más actualizado y moderno de bloques, pudiendo crear aplicaciones compatibles en diferentes equipos electrónicos: ordenadores, móviles, tablets… El conjunto Adobe permite descargar versiones limitadas para estudiantes, donde se puede manejar de forma gratuita la mayoría de sus herramientas internas.

Guide

Se trata, nuevamente, de una herramienta interna del software de cálculo Matlab. Mediante esta herramienta se permite crear aplicaciones de usuario con código preprogramado y configurable mediante la consola de comandos propia de Matlab. Cuenta con una interfaz de diseño de aplicaciones mediante bloques, desde los cuales también se podrá configurar el aspecto de la aplicación y la programación de la misma. Como se ha comentado anteriormente, Matlab cuenta con versiones de estudiante totalmente gratuitas con las que hacer uso de la mayoría de herramientas propias del software.

Cada uno de estos programas ofrece diferentes alternativas de programación, con multitud de bloques de diseño y lenguajes de programación propios. Se ha de tener en cuenta que esta interfaz debe interactuar internamente con el modelo del variador de frecuencia, de tal forma que el usuario se encargue de la parametrización y el estudio de resultado del modelo.

2.2. DETALLE DE LA SOLUCIÓN ADOPTADA

Finalmente, se ha seleccionado el software de Matlab por ser el denominador común entre los dos objetivos principales del proyecto. Dispone de una herramienta de simulación dinámica como es Simulink y, además, permite diseñar interfaces de usuario con las que interactuar sobre estos modelos, gracias a la herramienta GUIDE. Es cierto que el diseño de interfaces que permite GUIDE es más limitado que en otros softwares comentados, ya que el diseño puede parecer algo más “anticuado” y la diversidad de bloques es inferior; no obstante, al tratarse de una aplicación con fines educativos, el diseño no es uno de los requisitos prioritarios.

La herramienta Matlab es una de las más utilizadas a nivel educativo en multitud de universidades, lo que permite que sea un lenguaje de programación de los más conocidos por los alumnos y facilite la interacción entre el usuario y la interfaz. Por esta misma razón, Matlab dispone de licencias de estudiante, las cuales permiten a los alumnos la utilización de la mayoría de las herramientas internas de forma totalmente gratuita. Este hecho permitirá que todos los alumnos puedan disponer de la interfaz de simulación en todo momento, pudiendo trabajar con ella de forma paralela con la teoría y la práctica.

Una vez seleccionado el software de trabajo para la realización del proyecto, se van a describir los procedimientos que se han llevado a cabo en el diseño y programación tanto del modelo del variador de frecuencia en Simulink, como de la interfaz de control en GUIDE.

En este caso se ha desarrollado un modelo de variador de frecuencia con una señal trifásica de entrada, un puente de diodos rectificador, un bus de continua y un puente inversor de dos niveles. Este puente inversor de IGBTs podrá ser controlado por modulación SPWM y SVPWM. Una vez modulada la señal, se pasará por un filtro LC antes de alimentar al motor eléctrico. Ya con el modelo del variador diseñado, se debe almacenar en el entorno de Matlab todas las señales y variables que posteriormente se quieran manejar por la interfaz.

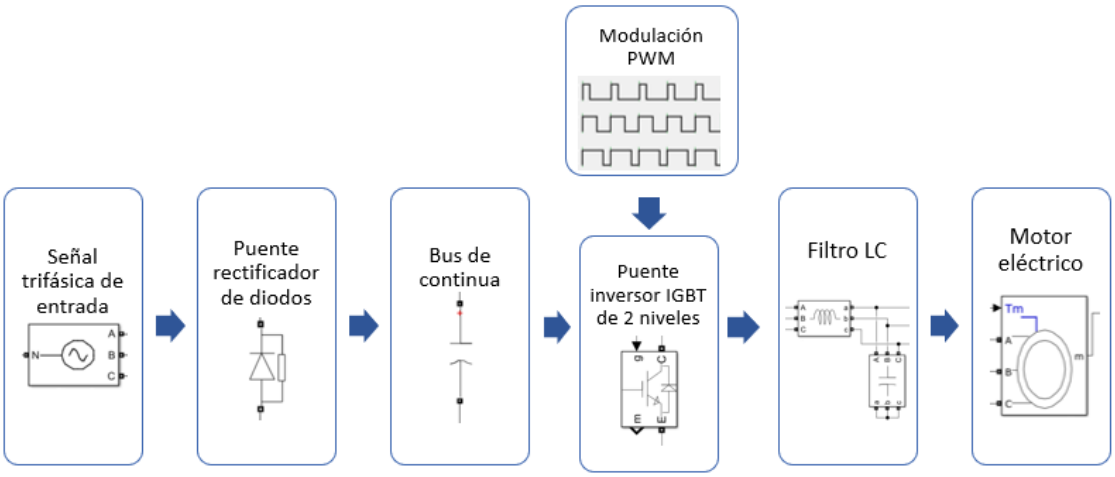


Diagrama 1. Secuencia de bloques del modelo del variador de frecuencia diseñado mediante Simulink

Para el diseño de la interfaz se han creado 3 bloques de secciones principales: parametrización del motor eléctrico asíncrono, configuración de señales de entrada, modulación y filtro, y ploteo de resultados. Una vez creado el diseño, se deben programar las funciones internas de todos los bloques que componen la interfaz. Muchos de estos bloques actuarán directamente sobre el modelo creado de Simulink, por lo que se tendrán que usar comandos específicos de la librería de Matlab.

Previamente a la explicación de todos los procesos descritos, se van a describir brevemente algunos conceptos como variador de frecuencia, modulación, Matlab, Simulink y GUIDE; por ser elementos clave en el desarrollo de este proyecto.

2.2.1. Variador de frecuencia

Un variador de frecuencia es un dispositivo electrónico capaz de modificar la frecuencia y amplitud de una señal de entrada. Normalmente, se compone de 4 etapas: rectificadora, bus de continua, inversora y modulación.

- **Etapa rectificadora:** Se encarga de convertir la tensión alterna de entrada en una tensión continua a partir de puentes de diodos, tiristores, IGBTs…
- **Bus de continua:** Se compone de 1 o 2 condensadores que tienen como fin suavizar la señal que se ha obtenido en la etapa rectificadora.
- **Etapa inversora:** Va a convertir la señal continua rectificada y filtrada, en una señal alterna con frecuencia y valor eficaz variables. Este proceso se suele realizar a través de puentes de IGBTs, controlados por pulsos con frecuencia configurable.
- **Etapa de modulación:** Se encarga de controlar los pulsos de los IGBTs que componen la etapa inversora. A partir de esta etapa se podrá configurar la frecuencia y amplitud deseadas en la señal de salida.

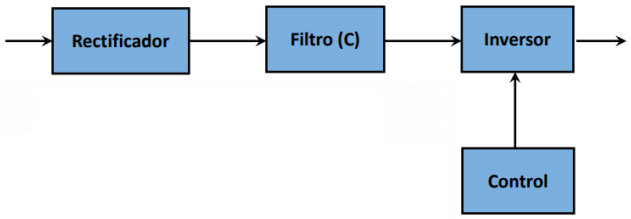


Diagrama 2. Secuencia de etapas en un variador de frecuencia.

2.2.2. Criterios de modulación

Los métodos de modulación que se desarrollan en este proyecto son SPWM (*Senoidal Pulse Width Modulation*) y SVPWM (*Space Vectorial Pulse Width Modulation*). Se han seleccionado estos dos métodos por ser los más utilizados hoy en día en multitud de procesos de control de transistores. En este caso, van a modular una etapa inversora de 2 niveles, por lo que se crearán 6 señales de control, una para cada transistor IGBT.

El principio de funcionamiento de la **modulación SPWM** se basa en la generación de una señal de pulsos a partir de la comparación de una señal senoidal moduladora que se configura con la frecuencia de salida deseada, y una señal triangular portadora; de tal manera que, cuando la señal senoidal sea mayor que la triangular, se generará un pulso de valor 1, y en el caso contrario será de valor 0.

En la Figura 1 se muestra el control SPWM creado en este proyecto implementado a partir de puertas lógicas (*PWM simple*).

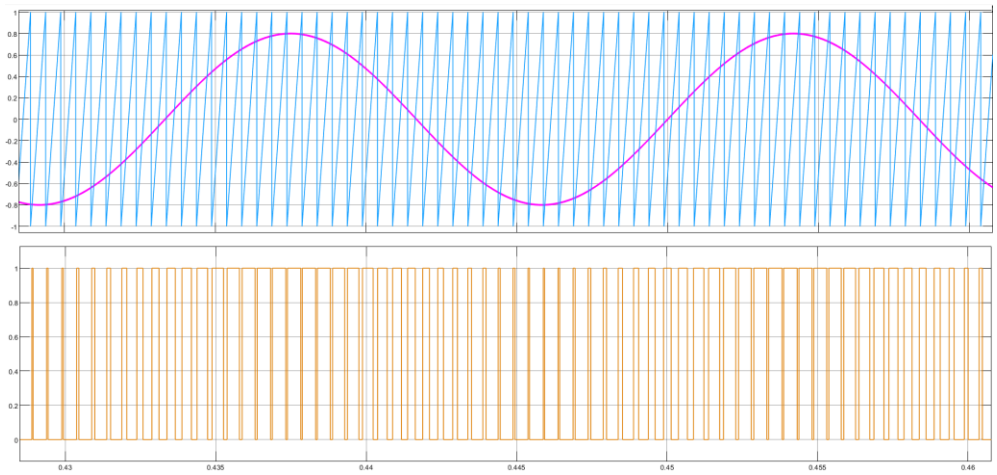
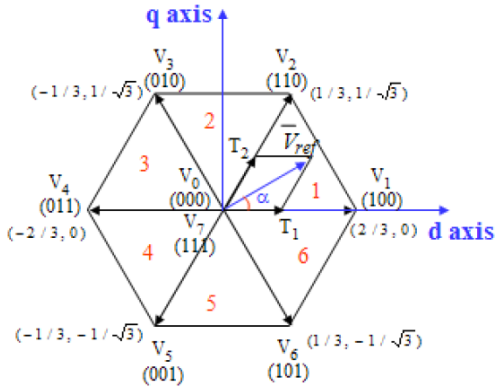


Figura 1. Control mediante modulación SPWM.

Por otro lado, la **modulación SVPWM** considera que una tensión senoidal se puede ver como un vector rotativo de amplitud y frecuencia constante. Considerando un puente de 6 IGBTs, se pueden considerar 6 secciones vectoriales de 60 grados cada una, por las cuales pasará la tensión trifásica de salida *Vref*. En la Figura 2 se observan los valores de la tensión en función de la combinación de IGBTs que se encuentren disparados o en conducción, de esta forma se posicionarían cada uno de los vectores en distintas secciones del hexágono.



Voltage Vectors	Switching Vectors			Line to neutral voltage			Line to line voltage		
	a	b	c	V_{an}	V_{bn}	V_{cn}	V_{ab}	V_{bc}	V_{ca}
V_0	0	0	0	0	0	0	0	0	0
V_1	1	0	0	2/3	-1/3	-1/3	1	0	-1
V_2	1	1	0	1/3	1/3	-2/3	0	1	-1
V_3	0	1	0	-1/3	2/3	-1/3	-1	1	0
V_4	0	1	1	-2/3	1/3	1/3	-1	0	1
V_5	0	0	1	-1/3	-1/3	2/3	0	-1	1
V_6	1	0	1	1/3	-2/3	1/3	1	-1	0
V_7	1	1	1	0	0	0	0	0	0

Figura 2. Tabla de valores y secciones del hexágono para la implementación del control mediante modulación SVPWM [1].

[1] ***Apuntes "Control del motor de inducción".*** Asignatura *Control de máquinas y accionamientos eléctricos*, Universidad Politécnica de Valencia.

2.2.3. Software Matlab

Es un sistema de cálculo numérico con interfaz de desarrollo y lenguaje de programación propios. Aparte de ser una potente máquina de cálculo, dispone de herramientas adicionales que le permiten diseñar y simular modelos mediante bloques (Simulink) y herramientas para la creación de interfaces de usuario (GUIDE). En la Figura 3 se muestra la pantalla de inicio del software Matlab.

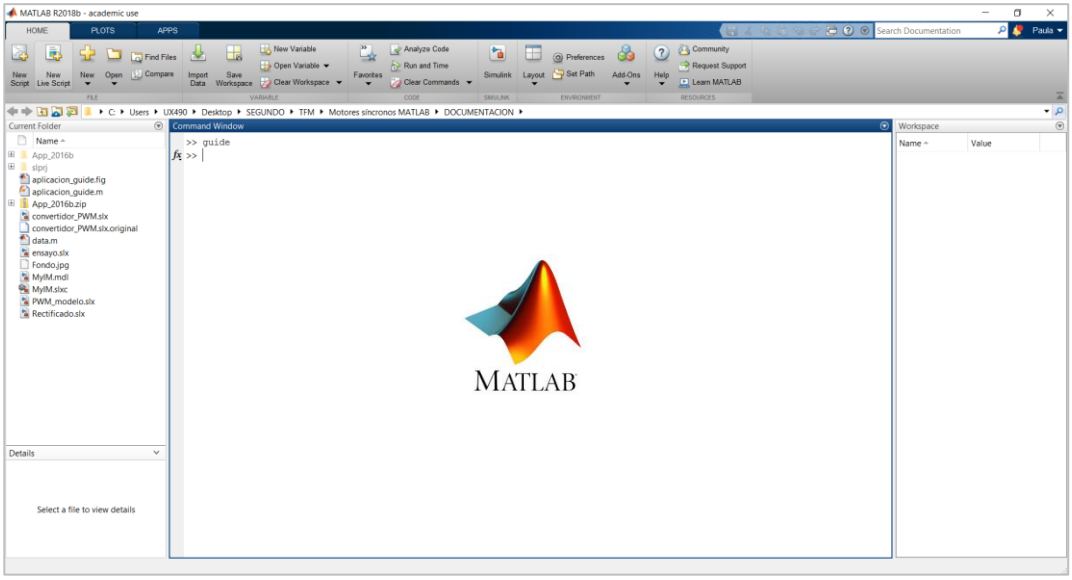


Figura 3. Pantalla inicial de Matlab.

2.2.4. Simulink

Entorno de programación visual que trabaja conjuntamente con Matlab. Se trata de un lenguaje de alto nivel que opera a partir de bloques para realizar el modelado, simulación y análisis de sistemas dinámicos. En la Figura 4 se muestra la pantalla de inicio de Simulink.

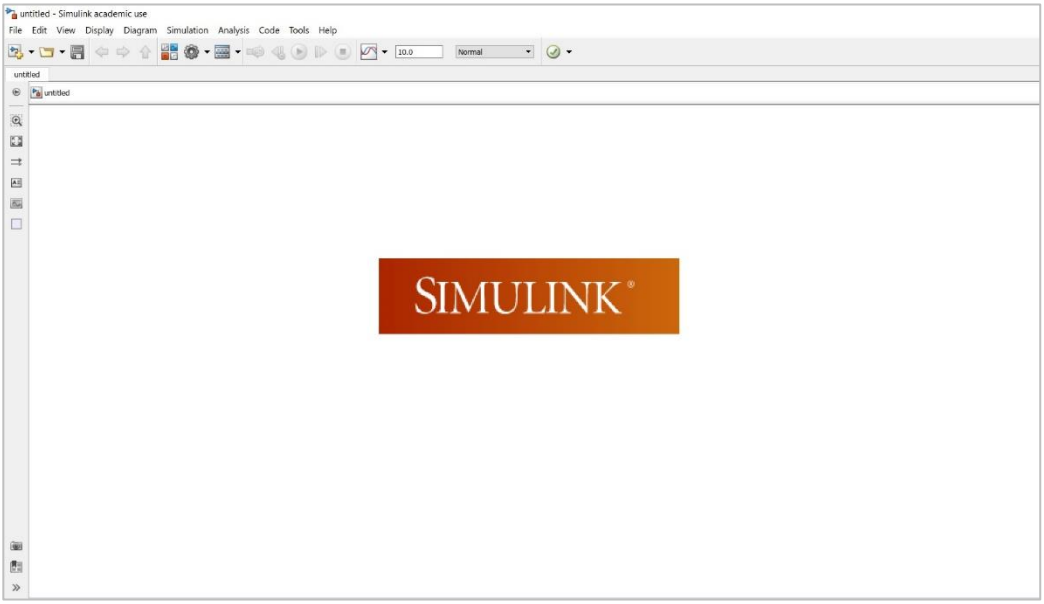


Figura 4. Pantalla inicial de Simulink.

2.2.5. GUIDE

GUIDE es un entorno de programación de interfaces de usuario disponible en el software Matlab. Esta plataforma se encarga de generar de forma automática el código Matlab de cada uno de los bloques, de esta forma, permite una programación más rápida y sencilla de las distintas funciones internas de los componentes. La Figura 5 muestra la pantalla inicial de GUIDE.

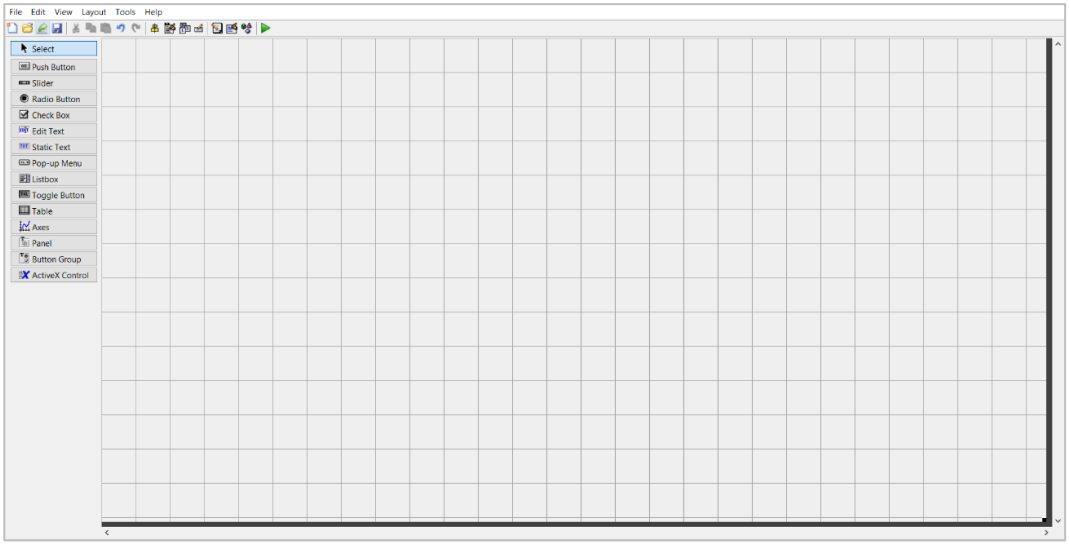


Figura 5. Pantalla de inicio para la creación de interfaces mediante GUIDE.

2.3. MODELO SIMULINK

Antes de comenzar con el desarrollo de la interfaz GUIDE del usuario, se ha diseñado el modelo del variador de frecuencia con las diferentes opciones de modulación para la etapa inversora. En este modelo se han agregado elementos para el acondicionamiento de la señal, como pueden ser condensadores y/o bobinas, disponibles para la configuración mediante la interfaz por parte del usuario. Finalmente, se han añadido todas las pestañas de configuración del modelo Simulink del motor para poder seleccionar la opción con la que se quiere trabajar.

En la Figura 6 se muestra una imagen del modelo de variador creado para Simulink.

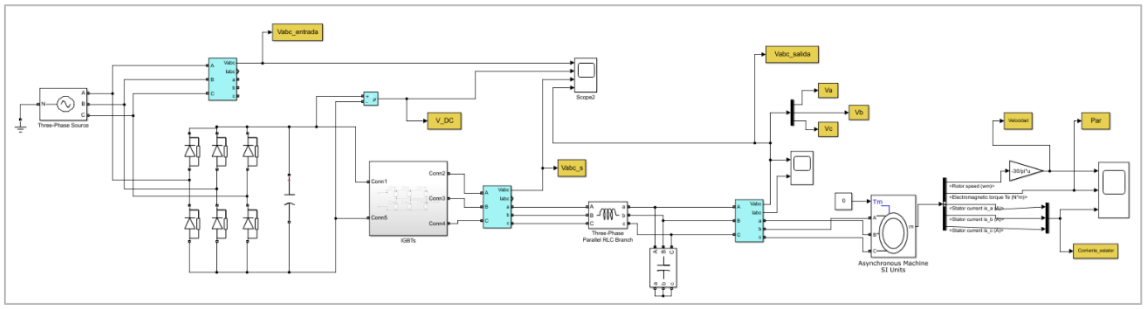


Figura 6. Modelo variador de frecuencia en Simulink.

Este modelo se compone de una etapa rectificadora AC / DC + filtro, configurada con un puente de diodos trifásico y un condensador encargado de estabilizar la señal DC de salida. Este conjunto se muestra con mayor claridad en la Figura 7.

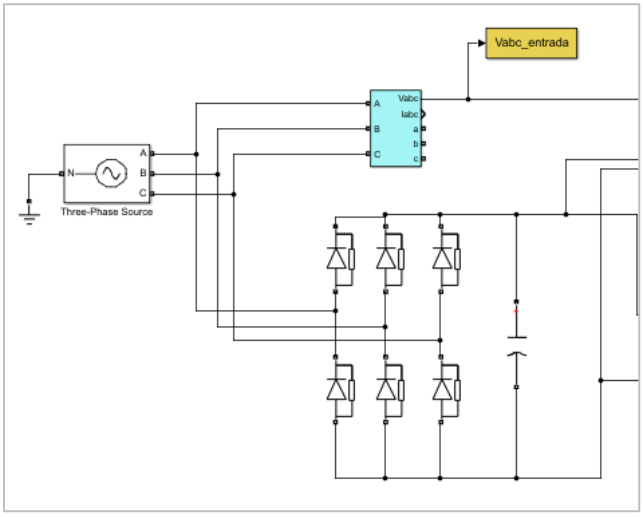


Figura 7. Etapa convertidora AC / DC trifásica.

A continuación de esta etapa se ha añadido una etapa inversora de dos niveles con transistores IGBTs (Figura 8), los cuales estarán controlados por diferentes señales de modulación: PWM diseñada, SPWM y SVPWM.

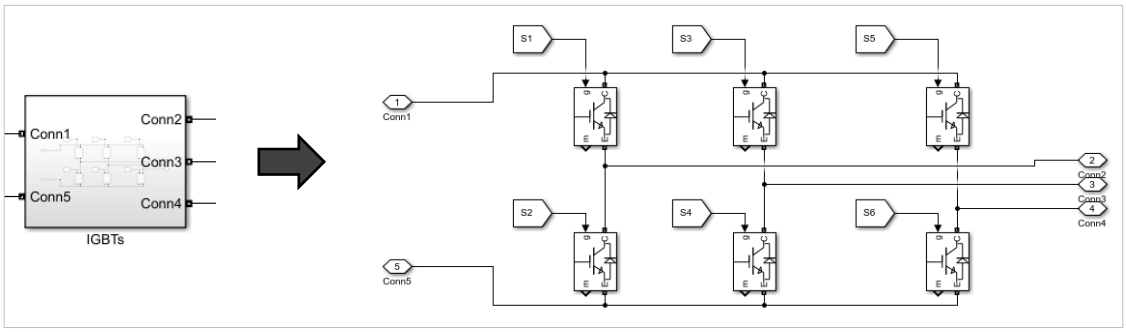


Figura 8. Puente de 6 IGBTs para modulación de 2 niveles.

Para crear los diferentes criterios de modulación se ha trabajado tanto con bloques operacionales de la librería Simulink, para la opción PWM diseñado; como con bloques de modulación de la librería Simscape (Figura 9a), para las opciones SPWM y SVPWM.

Las propiedades de modulación serán las mismas para todos los modelos y se controlará mediante interruptores la opción seleccionada en la interfaz, como se muestra en la Figura 9b.

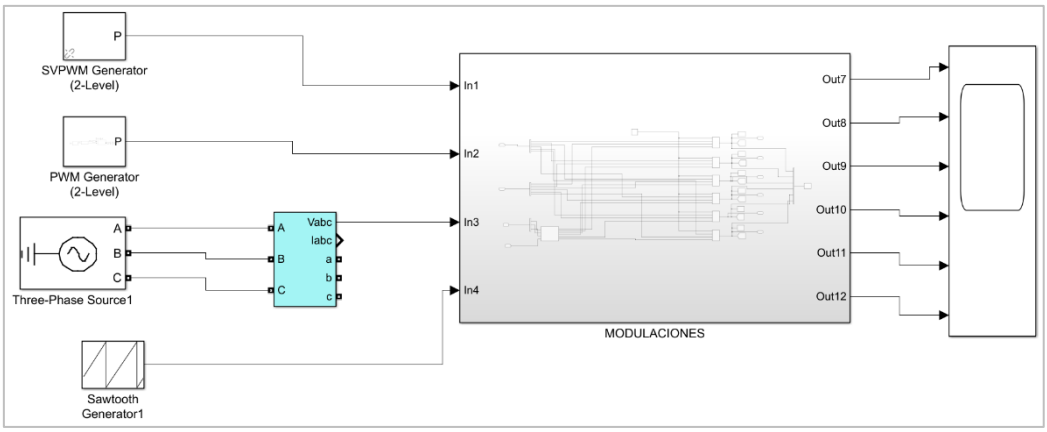


Figura 9a. Diseño de los 3 modelos para el control de la etapa inversora AC / DC.

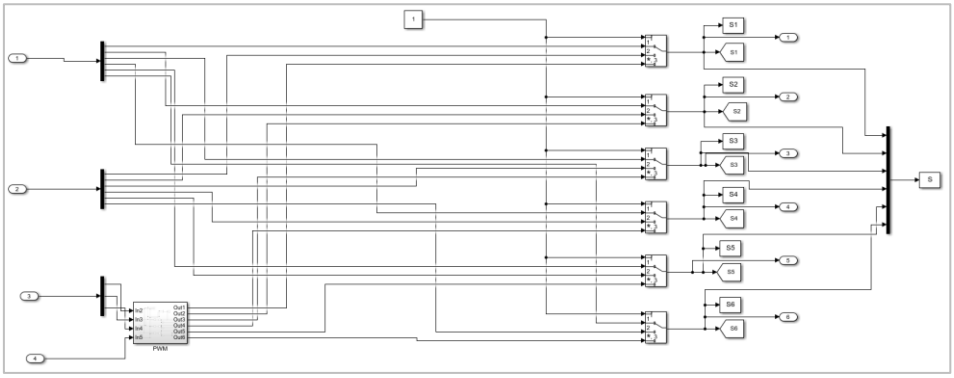


Figura 9b. Contenido del bloque *MODULACIONES* para la selección del criterio de modulación.

En el interior del bloque PWM se ha diseñado una modulación por ancho de pulso (PWM) mediante bloques operacionales, Figura 10. El proceso se realizará en cada una de las fases de la señal trifásica de entrada (moduladora) y consistirá en un sumatorio de las señales de entrada (moduladora y portadora), una comparación con el valor 0 y una inversión de la señal obtenida.

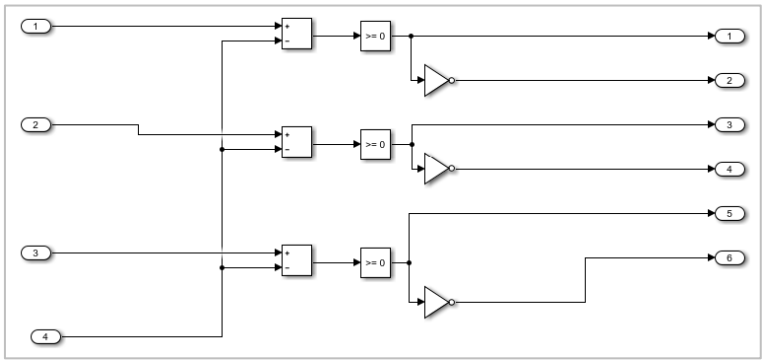


Figura 10. Modulación PWM diseñada mediante bloques operacionales.

La razón principal por la que se ha diseñado un PWM a partir de bloques lógicos, ha sido el hecho de poder aportar a los alumnos una visualización interna del funcionamiento de este tipo de control.

Para suavizar los posibles armónicos que hayan aparecido tras la etapa inversora, se ha añadido un filtro LC antes de conectar con el motor, apreciable en la Figura 11. Este filtro debe ajustarse a las condiciones de las señales de entrada y salida que se desean, por lo tanto, su parametrización estará disponible en la interfaz para que el usuario pueda configurar los valores de inductancia y capacidad.

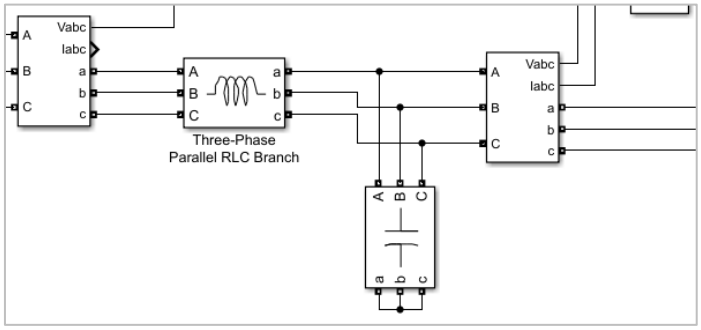


Figura 11. Filtro LC tras la etapa inversora.

Finalmente, se ha conectado a la salida del filtro un motor asíncrono trifásico (Figura 12) donde se podrá configurar el tipo de rotor, la magnitud de entrada que se debe superar para que el motor comience a funcionar, la referencia de entrada y todos los parámetros de trabajo.

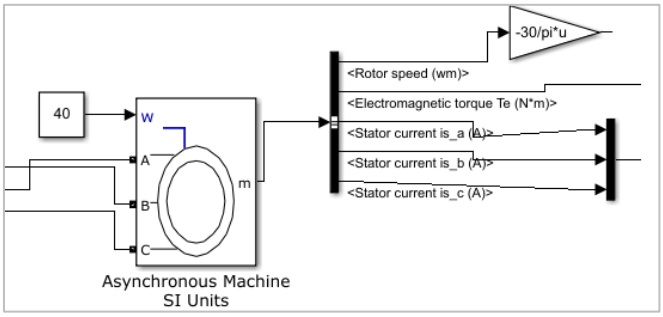


Figura 12. Motor asimétrico configurado mediante la interfaz.

A la salida del motor se ha medido la velocidad y el par de giro, así como la corriente que circula por el estator.

En cada una de las etapas del modelo del variador, del filtro y del motor, se han añadido elementos de medición de señales para poder estudiar los resultados que se han ido obteniendo.

La Figura 13 muestra los elementos de medición en color azul.

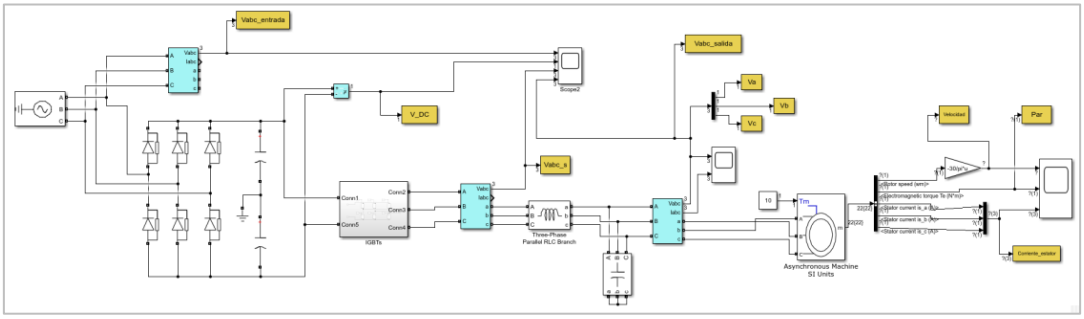


Figura 13. Señalización de los elementos de medición mediante color azul.

Estas mediciones se han almacenado en el *Workspace* de Matlab para poder trabajar con ellas posteriormente, mediante la interfaz. La utilización de referencias entre Simulink y la interfaz GUIDE se explicará en apartados posteriores.

En las Figuras 14a y b se ven resaltados en color amarillo aquellos bloques de almacenamiento de variables y señales en el *Workspace* de Matlab.

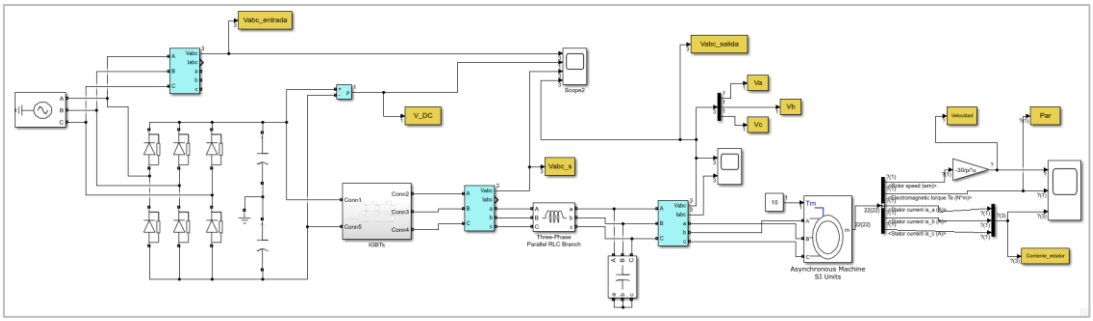


Figura 14a. Señalización en amarillo de las variables que se almacenan en Workspace de la parte del variador

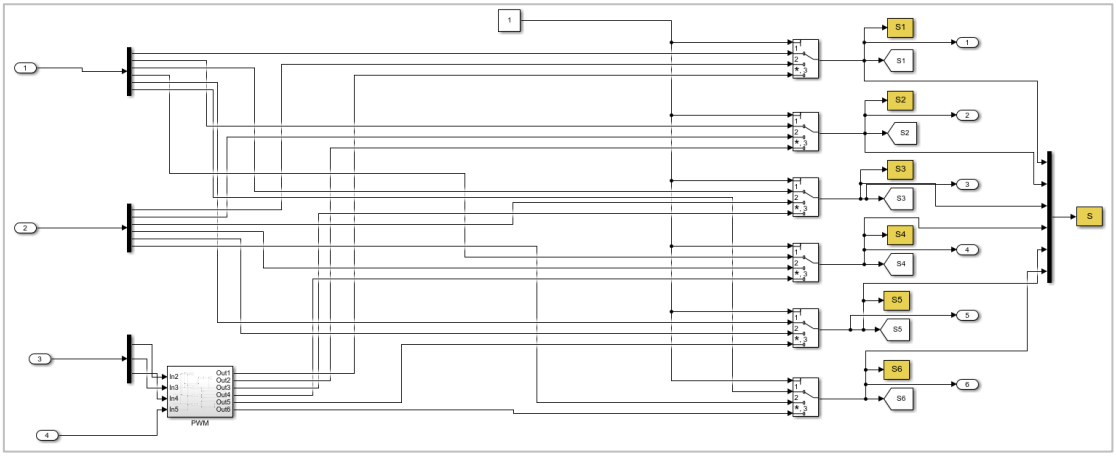


Figura 14b. Señalización en amarillo de las variables que se almacenan en Workspace de la parte de modulación.

2.4. MODELO INTERFAZ GUIDE

La interfaz se compone principalmente de botones, bloques editables y gráficos, con los cuales se podrán configurar las características de las señales de entrada y salida, del inversor, del filtro de la señal de salida y del motor; así como, analizar los resultados obtenidos e iniciar/parar la simulación. En la Figura 15 se puede visualizar el modelo creado para la interfaz GUIDE. En la sección *Anexos* se recopilan todos los pasos necesarios para diseñar y configurar los distintos bloques de la interfaz.

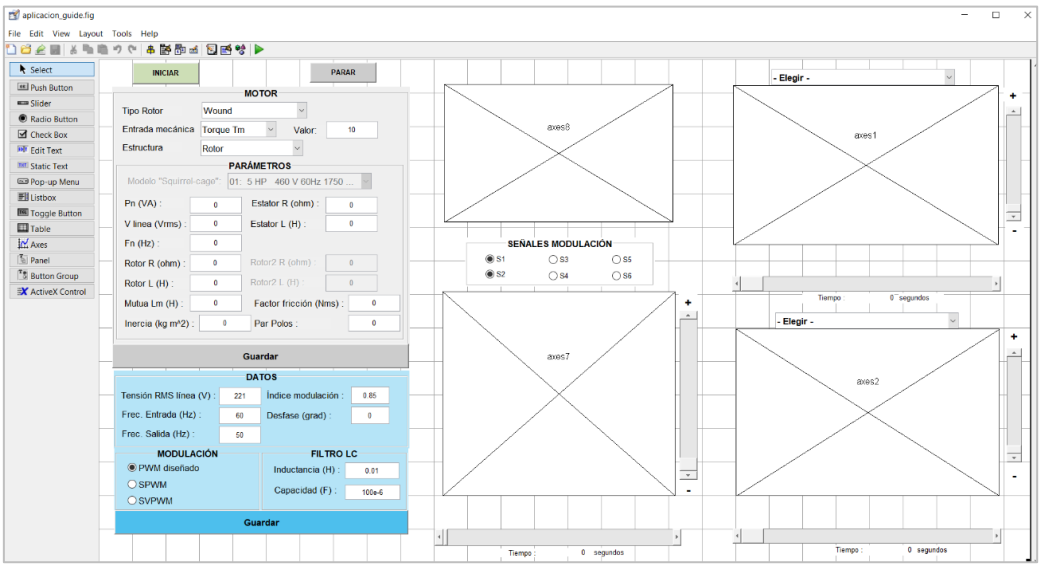


Figura 15. Diseño pantalla de inicio de la interfaz GUI.

2.5. PROGRAMACIÓN DE FUNCIONES MEDIANTE MATLAB

2.5.1. Tipos de funciones

Al crear una interfaz GUIDE se genera un archivo de código Matlab *.m*, donde se recogen las funciones de programación de inicialización de la interfaz, así como de los diferentes bloques diseñados dentro de la misma.

Dentro de esta ventana de código, aparecen diferentes tipos de funciones, de acuerdo al comportamiento de cada una de ellas.

- *function aplicacion_guide_OpeningFcn.* Esta función se ejecuta nada más arrancar la interfaz, de tal forma que el contenido se ejecute justo antes de abrir la aplicación y sea lo primero que se visualice al iniciar.
- *function varargout = aplicacion_guide_OutputFcn.* Esta función permite sacar por línea de comandos los resultados o valores ejecutados dentro de la misma.
- *function* "tag bloque"*_Callback.* Este tipo de función se crea automáticamente para cada uno de los bloques diseñados en la interfaz. En su interior se debe configurar la rutina asociada al bloque, de tal manera que se ejecute en el instante en el que se aplica una determinada acción sobre ese elemento.
- *function* "tag bloque"*_CreateFcn.* Este tipo de función se crea automáticamente al configurarse los parámetros de propiedades iniciales de los bloques. Su ejecución únicamente se lleva a cabo en el momento en que se inicializa el bloque.

Como se ha mencionado, la función *Callback* se ejecutará de acuerdo al accionamiento que tiene adjudicado cada tipo de bloque. De tal forma que se pueden distinguir los siguientes modos de ejecución:

- *function edit1_Callback.* Se ejecuta en el instante en que se termina de modificar el valor del bloque de texto editable, en este caso, del bloque *edit1.*
- *function pushbutton1_Callback.* Arranca en el momento en que se pulsa el botón, en este caso *pushbutton1.*
- *function radiobutton1_Callback.* Se ejecuta en el instante en que se selecciona la opción del bloque *radiobutton1*.
- *function popupmenu1_Callback.* Arranca cuando se selecciona alguna de las opciones configuradas en el *string* del bloque.
- *function slider1_Callback.* Se ejecuta en el momento en que se varía la posición del *slider*, bien sea seleccionando alguno de los botones avance/retroceso de los extremos, o bien desplazando la barra del *slider* con el ratón.

Los bloques *static text* y *axes* no tienen asociada una función de ejecución ya que, su objetivo principal es mostrar un contenido y, por lo tanto, se hará referencia a estos bloques dentro del resto de funciones.

Todas las funciones explicadas, se componen de unos datos de entrada que agrega Matlab de forma automática. Estas entradas permiten adquirir y configurar parámetros de los bloques desde las funciones, lo cual facilita la interactuación entre la interfaz y el modelo de Simulink.

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Figura 16. Función interna del botón pushbutton1.

La Figura 16 muestra la función interna del botón *Inicio*, donde se puede ver lo comentado anteriormente. Matlab, agrega comentarios de forma automática que pueden servir de ayuda para el usuario. Arriba de la función, se observa la acción de ejecución, en este caso actuará cuando se presione el botón. Una vez dentro de la función, se ven comentadas las diferentes entradas que componen la misma, donde se describe el sobrenombre del bloque (*hObject*), una entrada para futuras actualizaciones del software (*eventdata*) y una referencia a la estructura del elemento (*handles*).

2.5.2. Función de arranque de la interfaz

Para arrancar la interfaz, basta con pulsar el botón *run* de Matlab. En ese momento se ejecutarán dos partes del código, una sección no configurable de inicialización y una sección configurable que se ejecuta al iniciar la interfaz.

La sección no configurable es creada automáticamente por Matlab y se encarga de inicializar parámetros internos de la herramienta de GUI (Figura 17).

```
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @aplicacion_guide_OpeningFcn, ...
                   'gui_OutputFcn',  @aplicacion_guide_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

Figura 17. Sección no editable creada por Matlab que se ejecutará al arrancar la interfaz.

A continuación, se desarrolla la función de inicio de la aplicación, la cual si permite ser configurada por Matlab.

En este caso, se ha programado la carga del modelo del variador de frecuencia de Simulink y, por otro lado, se ha configurado una imagen de fondo para los elementos gráficos (Figura 18).

La carga del archivo ***convertidor_PWM*** es muy importante porque va a permitir que cada vez que se haga referencia a alguno de los bloques del modelo, o de las configuraciones internas de Simulink, la interfaz sepa a qué archivo se está refiriendo y sea capaz de encontrarlo.

Los comandos que se han configurado van a permitir cargar el archivo para trabajar de forma conjunta con la interfaz (***load_system***), buscar el archivo dentro de la memoria de la interfaz (***find_system***) y abrirlo automáticamente al iniciar (***open_system***).

```
load_system('convertidor_PWM');
find_system('Name','convertidor_PWM');
open_system('convertidor_PWM');
```

Además, se ha programado la carga de una imagen de fondo en los elementos gráficos. Para ello, se ha almacenado una imagen ***Fondo.jpg*** en un variable A, mediante la función ***imread***. Posteriormente, se ha cargado esa variable como imagen en cada uno de los gráficos utilizando el comando ***image***.

```
A= imread('Fondo.jpg');
image(handles.axes1,A);
image(handles.axes2,A);
```

Hay que destacar que, para que Matlab no tenga problemas de cargas con los archivos comentados, ambos se deben guardar en la misma ruta de desarrollo de la interfaz.

```
function aplicacion_guide_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to aplicacion_guide (see VARARGIN)
load_system('convertidor_PWM');
find_system('Name','convertidor_PWM');
open_system('convertidor_PWM');
A= imread('Fondo.jpg');
image(handles.axes1,A);
image(handles.axes2,A);
image(handles.axes7,A);
image(handles.axes8,A);
axis(handles.axes8,'off');
%Variables "globales"
handles.output = hObject;
handles.aux = 0;
handles.aux1 = 0;
handles.aux2 = 0;
guidata(hObject, handles);
```

Figura 18. Función editable que se ejecutará al arrancar la interfaz.

Finalmente, se han declarado a 0 unas variables globales que se usarán posteriormente en algunas funciones para el control de bloques. Estas variables almacenan un valor mientras la interfaz esté en marcha, y únicamente se inicializarán de nuevo cuando se vuelva a arrancar la aplicación; a diferencia del resto de variables que se generan dentro de las funciones, las cuales se inicializan cada vez que se entra a su respectiva función.

```
handles.aux = 0;
handles.aux1 = 0;
handles.aux2 = 0;
guidata(hObject, handles);
```

2.5.3. Función botón *INICIAR*

Para iniciar las simulaciones del modelo de variador de frecuencia, se ha creado el botón *INICIAR*. Una vez pulsado este botón, se ejecuta la función ***pushbutton1_Callback*** (Figura 19), donde se ha programado el arranque de la simulación en Simulink y, nuevamente, cargar las gráficas con su fondo inicial.

Para el arranque de la simulación, se ha trabajado con el comando *set_param*. Este comando permite configurar un parámetro en el archivo o elemento que se defina en el interior del paréntesis. Además, se debe especificar el apartado en el cual se quiere establecer ese valor de parámetro, y separarlo mediante coma. En este caso, el objetivo es arrancar la simulación de Simulink directamente a partir de la interfaz, para conseguirlo, se ha configurado la opción ***start***, dentro del parámetro ***Simulation Command*** del archivo ***Convertidor_PWM***.

```
set_param ('convertidor_PWM','SimulationCommand','start');
```

Por último, se volverá a cargar la imagen del fondo de las gráficas, para que en el momento que iniciemos una nueva simulación, se cambie a un fondo predeterminado y no se queden visualizadas las gráficas de la simulación anterior.

Nota: Cabe destacar que GUIDE configura funciones independientes para cada elemento, es por ello, que las variables que se definen en el interior de una función no están disponibles para otras funciones, como ya se ha comentado anteriormente.

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set_param ('convertidor_PWM','SimulationCommand','start');
A= imread('Fondo.jpg');
image(handles.axes1,A);
image(handles.axes2,A);
image(handles.axes7,A);
image(handles.axes8,A);
```

Figura 19.Función interna del bloque pushbutton1.

2.5.4. Función botón *PARAR*

De igual manera, se ha creado el botón *PARAR* para detener la simulación de Simulink desde la interfaz. Pulsando este botón se ejecuta la función ***pushbutton2_Callback*** (Figura 20), donde se ha programa la detención de la simulación del modelo del variador gracias al comado *set_param.* En este caso, también se quiere configurar el parámetro ***Simulation Command*** del archivo ***Convertidor_PWM***, pero ahora te introducirá el valor ***stop***.

```
set_param ('convertidor_PWM','SimulationCommand','stop');
```

```
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set_param ('convertidor_PWM','SimulationCommand','stop');
```

Figura 20. Función interna del bloque pushbutton2.

2.5.5. Funciones para la configuración del motor asíncrono

Para la configuración completa del bloque que simula el motor asíncrono *Asynchronous Machine SI Units* se han requerido diferentes funciones. Esta división de funciones se ha realizado principalmente porque algunos parámetros del motor son dependientes de otros, provocando que estos puedan habilitarse o deshabilitarse.

Las diferentes funciones que componen esta configuración se pueden dividir en una función para cada uno de los menús desplegables de la parte superior del bloque (***popupmenu***) y una función para el botón *GUARDAR.* Pese a que todos los bloques editables (***edit_text***) tienen una función propia que se activa cada vez que se configura el valor del bloque, se ha preferido agrupar los valores de todos los parámetros editables para poder ser modificados y ajustados conjuntamente antes de enviarlos al bloque del motor en Simulink.

Antes de continuar comentando las diferentes funciones, es importante destacar que para la configuración de los bloques y elementos de Simulink, se debe hacer referencia al nombre de la variable interna que tiene configurada cada parámetro según la programación de Simulink. Para visualizar estos nombres, habrá que abrir el editor de máscara/contorno del bloque y seleccionar la opción ***Mask / View Base Mask***, como se muestra en la Figura 21.

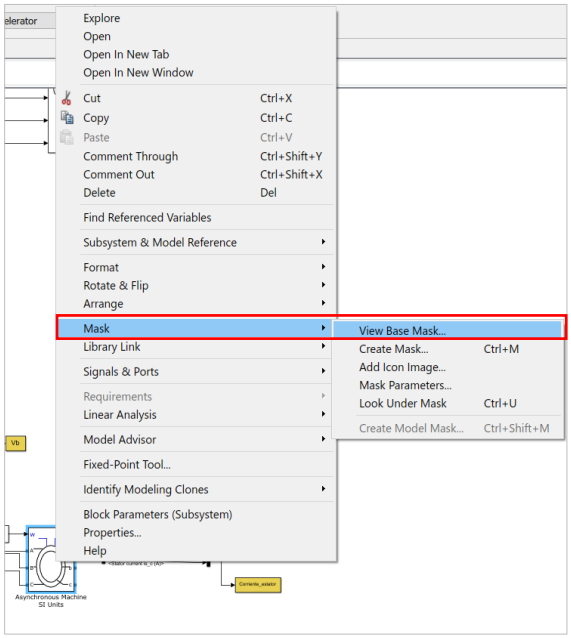


Figura 21. Selección ventana de edición de contorno del motor asíncrono.

A continuación, se muestra la ventana ***Mask Editor*** donde se debe seleccionar la pestaña ***Parameters & Dialog***. Dentro de esta pestaña aparecen todos los posibles parámetros del bloque, los cuales se muestran con el nombre de etiqueta (*prompt*) y el nombre interno al que se debe hacer referencia para configurarse de forma externa (*name*). En la Figura 22 se muestra la ventana descrita.

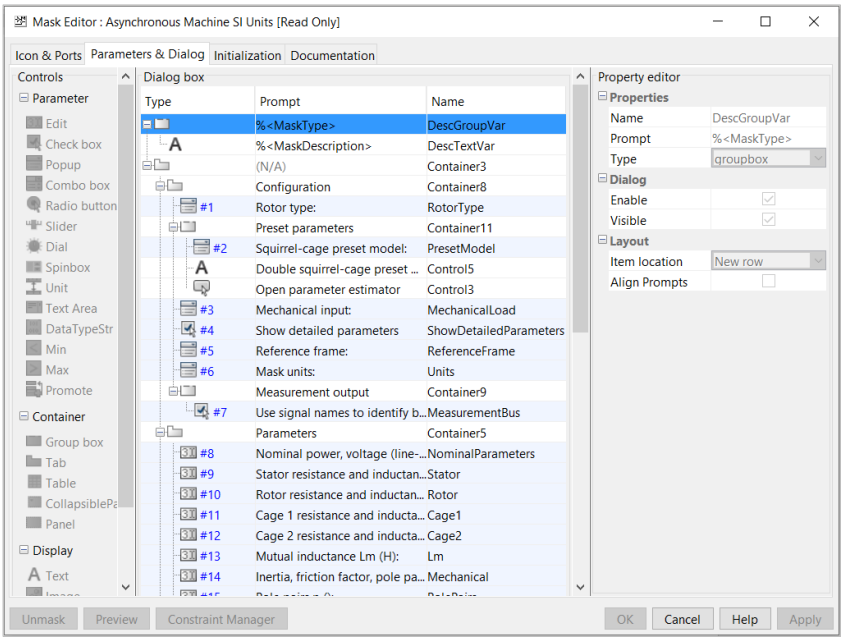


Figura 22.Ventana de edición de contorno.

2.5.5.1. Función menú desplegable *Tipo Rotor*

Con este bloque se puede seleccionar el tipo de rotor del motor asíncrono: *Wound, Squirrel Cage* o *Double Squirrel Cage*. Cada opción permite configurar unos parámetros característicos del motor, por lo tanto, se han tenido en cuenta todas las posibles opciones interactuando previamente con el bloque del motor en Simulink.

Una vez seleccionadas una de estas opciones, se ejecutará la función ***popupmenu1_Callback.*** En su interior se ha usado el comando ***get*** para almacenar el contenido. En este caso, se ha programado para almacenar el parámetro ***Value*** o valor del bloque, es decir, se guarda el número/posición de la opción seleccionada (*Wound* = 1, *Squirrel Cage* = 2 y *Double Squirrel Cage* = 3) y se pasa de una variable de celdas a una numérica mediante el comando ***cell2mat***.

```
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
```

A partir de la variable *Rotor* se han creado subfunciones condicionales a partir del comando ***if-else***, de esta forma se puede distinguir la opción seleccionada y actuar en consecuencia.

Dentro de cada condición, se realizarán dos operaciones principales: la inserción del propio parámetro en una variable string *R* para su posterior configuración en el bloque de Simulink, y la des/habilitación de los parámetros característicos del motor. Para la segunda operación se debe configurar el parámetro ***Enable*** de cada uno de los bloques de la interfaz que intervienen es este proceso; por ello, se han creado 3 variables *string* (*W, S, D)* que tendrán

un valor *On / Off*, dependiendo del caso. Los parámetros que dependen de la opción *Wound* se configurarán de acuerdo a la variable *W*, y lo mismo ocurrirá con el resto de las opciones, aquellos que dependan de la opción *Squirrel Case* se configurarán con la variable *S* y los que dependan de *Double Squirrel Cage* con la variable *D.*

```
if Rotor == 1
        R = 'Wound';
        W = 'on';
        S = 'off';
        D = 'off';
   elseif Rotor == 2
        R = 'Squirrel-cage';
        W = 'off';
        S = 'on';
        D = 'off';
    else
        R = 'Double Squirrel-cage';
        W = 'on';
        S = 'off';
        D = 'on';
    end
set_param('convertidor_PWM/Asynchronous Machine SI
Units','RotorType', R);
```

En el caso de seleccionar la opción *Wound*, se configurarán los parámetros del motor uno por uno, por lo tanto, se deshabilitarán las opciones correspondientes únicamente a los otros dos tipos de rotor: *Modelo "Squirrel Cage", Rotor2 R* y *Rotor2 L.* La Figura 23 muestra la relación entre los bloques diseñados en la interfaz y el bloque del motor en Simulink.

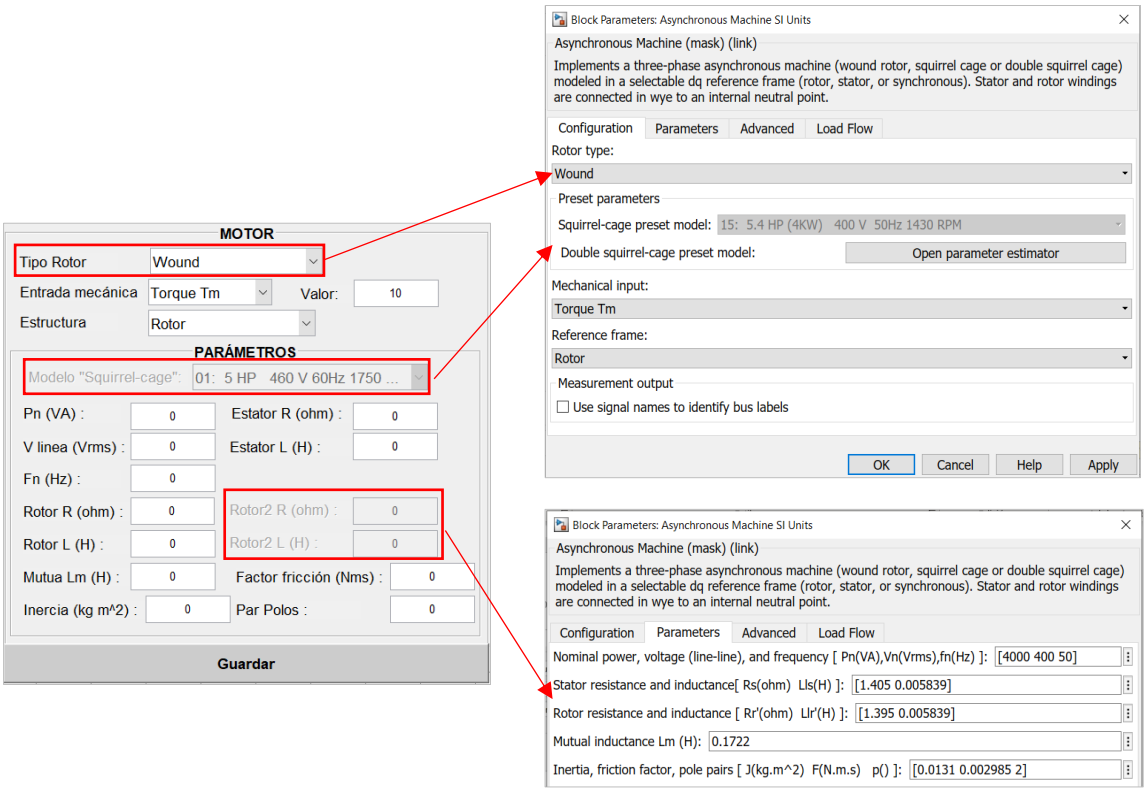


Figura 23. Parámetros deshabilitados al seleccionar la opción *Wound.*

Estos parámetros corresponden a los bloques *popupmenu4, text15, text19, text30, edit23* y *edit24.* Por lo tanto, seleccionando esta opción, esos bloques se pondrán a *Off* y se deshabilitarán. El resto de los bloques se configuran a partir de la variable *W* y se habilitarán. En la Figura 24 se muestra lo descrito anteriormente.

```
Rotor = cell2mat (AUX);
   if Rotor == 1
        R = 'Wound';
        W = 'on';
        S = 'off';
        D = 'off';
   elseif Rotor == 2
      R = 'Squirrel-cage';
      W = 'off';
      S = 'on';
      D = 'off';
    else
       R = 'Double Squirrel-cage';
       W = 'on';
       S = 'off';
       D = 'on';
    end
set_param('convertidor_PWM/Asynchronous Machine SI Units','RotorType', R);
%Parametros Squirrel-cage
set(handles.text15,'Enable',S);
set(handles.popupmenu4,'Enable',S);
%Parametros Double Squirrel-cage
set(handles.text29,'Enable',D);set(handles.text30,'Enable',D);
set(handles.edit23,'Enable',D);set(handles.edit24,'Enable',D);
%Parametros Wound y Double Squirrel-cage
set(handles.text18,'Enable',W);set(handles.text19,'Enable',W);set(handles.text20,'Enable',W);
set(handles.text21,'Enable',W);set(handles.text22,'Enable',W);set(handles.text23,'Enable',W);
set(handles.text24,'Enable',W);set(handles.text25,'Enable',W);set(handles.text26,'Enable',W);
set(handles.text27,'Enable',W);set(handles.text28,'Enable',W);set(handles.edit12,'Enable',W);
set(handles.edit13,'Enable',W);set(handles.edit14,'Enable',W);set(handles.edit15,'Enable',W);
set(handles.edit16,'Enable',W);set(handles.edit17,'Enable',W);set(handles.edit18,'Enable',W);
set(handles.edit19,'Enable',W);set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
set(handles.edit22,'Enable',W);
```

Figura 24. Señalización del funcionamiento de la opción *Wound* en el código de la función popupmenu1.

En el caso de seleccionar la opción *Squirrel Cage* los parámetros dependerán de un segundo menú desplegable, ***Modelo "Squirrel Cage"***. Por ello, el resto de los parámetros se deshabilitarán y solamente quedará habilitado ese bloque de la interfaz. En la Figura 25 se pueden ver los parámetros habilitados y deshabilitados al seleccionar esta opción.

Por último, seleccionando la opción ***Double Squirrel Cage***, se podrán configurar todos los parámetros del bloque, excepto el menú desplegable creado para un rotor de tipo *Squirrel Cage.* Nuevamente, en la Figura 25, se muestran los posibles parámetros a configurar si se selecciona esta opción.

Figura 25. Parámetros des/habilitados para las opciones *Squirrel Cage* (izquierda) y *Double squirrel-Cage* (derecha).

También se ha tenido en cuenta la posibilidad de que este botón se modifique después de hacer cambios en el resto de los parámetros, teniendo que respetar las condiciones que estos conlleven. En este caso, se ha tenido en cuenta la posibilidad de que el botón *Entrada mecánica* esté configurado como *Speed*, lo que provocaría que los parámetros *Inercia* y *Factor fricción* no fueran necesarios y haya que deshabilitarlos. Para ello, se ha añadido un condicionante al final de la función principal que mirará el valor que contiene el menú desplegable ***popupmenu5*** y ejecutará o no los comandos que se encuentran en su interior. En la siguiente sección se comentará más información de este parámetro *Speed.*

```
AUX5 = {get(handles.popupmenu5,'Value')};
speed = cell2mat (AUX5);
if speed == 2
    W='off';
    set(handles.text26,'Enable',W);set(handles.text27,'Enable',W);
    set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
end
```

Finalmente, en la Figura 26 se muestra la programación interna realizada para el conjunto completo de la función popupmenu1.

```
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
%CONFIGURACION DE ROTOR
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
   if Rotor == 1
       R = 'Wound';
       W = 'on';
       S = 'off';
       D = 'off';
   elseif Rotor == 2
      R = 'Squirrel-cage';
      W = 'off';
      S = 'on';
      D = 'off';
    else
       R = 'Double Squirrel-cage';
       W = 'on';
       S = 'off';
       D = 'on';
    end
set_param('convertidor_PWM/Asynchronous Machine SI Units','RotorType', R);
%Parametros Squirrel-cage
set(handles.text15,'Enable',S);
set(handles.popupmenu4,'Enable',S);
%Parametros Double Squirrel-cage
set(handles.text29,'Enable',D);set(handles.text30,'Enable',D);
set(handles.edit23,'Enable',D);set(handles.edit24,'Enable',D);
%Parametros Wound y Double Squirrel-cage
set(handles.text18,'Enable',W);set(handles.text19,'Enable',W);set(handles.text20,'Enable',W);
set(handles.text21,'Enable',W);set(handles.text22,'Enable',W);set(handles.text23,'Enable',W);
set(handles.text24,'Enable',W);set(handles.text25,'Enable',W);set(handles.text26,'Enable',W);
set(handles.text27,'Enable',W);set(handles.text28,'Enable',W);set(handles.edit12,'Enable',W);
set(handles.edit13,'Enable',W);set(handles.edit14,'Enable',W);set(handles.edit15,'Enable',W);
set(handles.edit16,'Enable',W);set(handles.edit17,'Enable',W);set(handles.edit18,'Enable',W);
set(handles.edit19,'Enable',W);set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
set(handles.edit22,'Enable',W);

AUX5 = {get(handles.popupmenu5,'Value')};
speed = cell2mat (AUX5);
if speed == 2
    W='off';
    set(handles.text26,'Enable',W);set(handles.text27,'Enable',W);
    set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
end
```

Figura 26. Función interna del bloque popupmenu1.

2.5.5.2. Función menú desplegable *Entrada mecánica*.

Mediante este menú se va a dar la posibilidad de configurar una entrada mecánica al motor de tipo par ***Torque Tm*** o tipo velocidad ***Speed w***. Para este caso, se han almacenado dos variables del bloque *popupmenu*. Por un lado, se ha guardado el parámetro *String* donde están configurados los posibles valores del bloque; por otro lado, se ha almacenado el valor (parámetro *Value*) que tiene el bloque en ese momento.

Dado que el parámetro *String* contiene una lista de valores, estos se van a guardar en formato matriz de celdas string. Para saber cúal es la opción seleccionada, se utiliza el valor numérico del parámetro *Value* para entrar a la matriz y buscar la opción correspondiente. La opción se almacenará en una variable tipo string que se podrá enviar al bloque del motor en Simulink.

```
Mec_matriz = cellstr(get(handles.popupmenu5,'String'));
Numero={get(handles.popupmenu5,'Value')};
Num = cell2mat (Numero);
Mechanical=string(Mec_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI
Units','MechanicalLoad', Mechanical)
```

Además, como se ha comentado anteriormente, se debe tener en cuenta si alguna de las opciones des/habilita parámetros del motor. Para este caso, los parámetros de *Inercia* y *Factor fricción* no son necesarios en una entrada de tipo *Speed w*.

Sin embargo, no bastará con habilitar o deshabilitar esas variables, también será necesario mirar el tipo de rotor con el que se va a trabajar para saber si se deben habilitar o no. Como se ha visto en la sección anterior, en el caso de haber seleccionado un rotor *Squirrel Cage*, estos parámetros deben permanecer deshabilitados. Por esta razón, se ha almacenado el valor del menú desplegable para el tipo de rotor y se ha usado un condicionante *if-else* que tendrá en cuenta tanto el valor del bloque *popupmenu5,* como del *popupmenu1.* La Figura 27 muestra la programación interna de esta función.

```
function popupmenu5_Callback(hObject, eventdata, handles)
%Almacenar valor Tipo Rotor
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
%Alamcenar los posibles valores de entrada mecanica en un matriz de string,
%posteriormente almacenar el valor para entrar en la matriz y coger el
%string que corresponde
Mec_matriz = cellstr(get(handles.popupmenu5,'String'));
Numero={get(handles.popupmenu5,'Value')};
Num = cell2mat (Numero);
Mechanical=string(Mec_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','MechanicalLoad', Mechanical);
if Num == 1 && Rotor ~= 2
    W="on";
else
    W ="off";
end
set(handles.text26,'Enable',W);set(handles.text27,'Enable',W);
set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
```

Figura 27. Función interna del bloque popupmenu5.

2.5.5.3. Función menú desplegable *Estructura*.

Mediante esta función ***popupmenu6*** se selecciona la estructura del motor eléctrico, pudiendo elegir entre las opciones: *Rotor, Stationary* y *Synchronous.* En este caso, ninguna de estas opciones condiciona los parámetros de configuración del motor, por lo tanto, será suficiente con crear una función que almacene el valor seleccionado del bloque de la interfaz y lo configure en el bloque del motor de Simulink. Estos pasos se harán de igual manera que se ha explicado en el apartado anterior, como se puede observar en la Figura 28.

```
function popupmenu6_Callback(hObject, eventdata, handles)
Estru_matriz = cellstr(get(handles.popupmenu6,'String'));
Numero={get(handles.popupmenu6,'Value')};
Num = cell2mat (Numero);
Estructura=string(Estru_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','ReferenceFrame', Estructura);
```

Figura 28. Función interna del bloque popupmenu6.

2.5.5.4. Función menú desplegable *Modelo "Squirrel Cage".*

Este bloque permite parametrizar el motor eléctrico únicamente cuando se haya seleccionado un tipo de motor *Squirrel Cage*. Dentro del bloque hay un listado de 15 opciones para configurar unos valores predeterminados en el motor, estas opciones son idénticas a las que permite seleccionar el bloque del motor en Simulink.

Para la programación de esta función ***popupmenu4*** se operará de igual manera que en los casos anteriores, se almacenará el valor seleccionado, se identificará con su respectiva etiqueta o nombre, y se enviará al bloque de Simulink. En este caso, además, hay que tener en cuenta que la primera de todas las opciones "*No*", permite parametrizar uno por uno los valores y no configurarlos mediante las alternativas predeterminadas. Por lo tanto, si se selecciona esta opción, se deberán volver a habilitar todos los bloques para la configuración de parámetros del motor, conforme se muestra en la Figura 29.

```
function popupmenu4_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
Rotor_matriz = cellstr(get(handles.popupmenu4,'String'));
Numero={get(handles.popupmenu4,'Value')};
Num = cell2mat (Numero);
Rotor=string(Rotor_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','PresetModel', Rotor);
if Num == 1
    W = 'on';
    %Parametros Wound y Double Squirrel-cage
    set(handles.text18,'Enable',W);set(handles.text19,'Enable',W);set(handles.text20,'Enable',W);set(handles.text2
    set(handles.text24,'Enable',W);set(handles.text25,'Enable',W);set(handles.text26,'Enable',W);set(handles.text2
    set(handles.edit12,'Enable',W);set(handles.edit13,'Enable',W);set(handles.edit14,'Enable',W);set(handles.edit1
    set(handles.edit18,'Enable',W);set(handles.edit19,'Enable',W);set(handles.edit20,'Enable',W);set(handles.edit2
    AUX5 = {get(handles.popupmenu5,'Value')};
    speed = cell2mat (AUX5);
    if speed == 2
        W='off';
        set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
    end
end
```

Figura 29. Función interna del bloque popupmenu4.

2.5.5.5. Función botón *Guardar* en la configuración de parámetros del motor.

A partir de este botón se ejecutará la función ***pushbuttom4,*** la cual se encarga de almacenar todos los valores configurados en la pestaña de parámetros y enviarlos al bloque del motor en Simulink. Antes de enviarlos al bloque del modelo del motor, se deberán modificar los parámetros para que el modelo pueda reconocerlos correctamente, ya que, algunos de estos se deben insertar de forma conjunta como un vector de parámetros o incluso, con una longitud de decimales determinada. Para realizar este paso, se han convertido todas las variables en tipo string y se han concatenado aquellas que requieran ser configuradas en forma de vector en el modelo.

```
P_n = get(handles.edit12,'String');
V_n = get(handles.edit13,'String');
F_n = get(handles.edit14,'String');
Nominal_parameters = strcat("[",P_n," ",V_n," ",F_n,"]");
```

Finalmente, se deberán crear unos condicionantes que tengan en cuenta el tipo de motor que ha sido seleccionado y qué variables estarán deshabilitadas para que sepa que esos parámetros no deben ser enviados al modelo. La programación diseñada se puede visualizar en la Figura 30.

```
function pushbutton4_Callback(hObject, eventdata, handles)
%CONFIGURACION PARAMETROS MOTOR
P_n = get(handles.edit12,'String');
V_n = get(handles.edit13,'String');
F_n = get(handles.edit14,'String');
Nominal_parameters = strcat("[",P_n," ",V_n," ",F_n,"]");
set_param('convertidor_PWM/Asynchronous Machine SI Units','NominalParameters',Nominal_parameters);
R_e = get(handles.edit15,'String');
L_e = get(handles.edit16,'String');
Estator_parameters = strcat("[",R_e," ",L_e,"]");
set_param('convertidor_PWM/Asynchronous Machine SI Units','Stator',Estator_parameters);
R_r = get(handles.edit17,'String');
L_r = get(handles.edit18,'String');
Rotor_parameters = strcat("[",R_r," ",L_r,"]");
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
if Rotor == 3
    R_r2 = get(handles.edit23,'String');
    L_r2 = get(handles.edit24,'String');
    Rotor2_parameters = strcat("[",R_r2," ",L_r2,"]");
    set_param('convertidor_PWM/Asynchronous Machine SI Units','Cage1',Rotor_parameters);
    set_param('convertidor_PWM/Asynchronous Machine SI Units','Cage2',Rotor2_parameters);
else
    set_param('convertidor_PWM/Asynchronous Machine SI Units','Rotor',Rotor_parameters);
end
L_m = get(handles.edit19,'String');
set_param('convertidor_PWM/Asynchronous Machine SI Units','Lm',L_m);
I = get(handles.edit20,'String');
F = get(handles.edit21,'String');
P = get(handles.edit22,'String');
mecanical_parameters = strcat("[",I," ",F," ",P,"]");
AUX = {get(handles.popupmenu5,'Value')};
Entrada = cell2mat (AUX);
if Entrada == 2
    set_param('convertidor_PWM/Asynchronous Machine SI Units','PolePairs',P);
else
    set_param('convertidor_PWM/Asynchronous Machine SI Units','Mechanical',mecanical_parameters);
end
constante = get(handles.edit28,'String');
set_param('convertidor_PWM/Constant','Value',constante);
```

Figura 30. Función interna del bloque pushbutton4.

2.5.6. Funciones para la configuración del variador de frecuencia.

Esta sección se encarga de configurar parámetros esenciales en el funcionamiento del variador de frecuencia, como son: las frecuencias de entrada y salida, la tensión de entrada, tipo de modulación y valores del filtro LC para la señal de salida.

Al igual que en la sección de configuración del motor, en este caso se va a dividir este bloque en una función para el botón *Guardar* y diferentes funciones para la selección del criterio de modulación: *PWM diseñado, SPWM* y *SVPWM*.

2.5.6.1. Funciones botones seleccionables *PWM diseñado, SPWM* y *SVPWM.*

Estos bloques se encuentran integrados en un panel de grupo (*button group*) de tal manera que solamente uno de ellos puede estar habilitado en todo momento, cuando sea preciso cambiar de opción, automáticamente se deshabilitara la anterior y se seleccionará esta nueva.

Estos bloques están controlados mediante la función ***radiobutton1, radiobutton2*** y ***radiobutton3.*** En el interior de cada función se usa un único comando para enviar el valor que debe contener la constante que se encarga de conmutar el multiplexor del modelo de Simulink para la elección de la señal que controlará el puente inversor de IGBTs (Figura 31).

```
% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
RB1 = {get(handles.radiobutton1,'Value')};
RB_1 = cell2mat (RB1);
if RB_1 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value', '3');
end

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
RB2 = {get(handles.radiobutton2,'Value')};
RB_2 = cell2mat (RB2);
if RB_2 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value','2');
end

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
if RB_3 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value','1');
end
```

Figura 31. Funciones internas de los bloques radiobutton1, radiobutton2 y radiobutton3.

En la Figura 32 se muestra la constante que configuran estos botones de selección, esta se encuentra dentro del bloque *Modulaciones*.

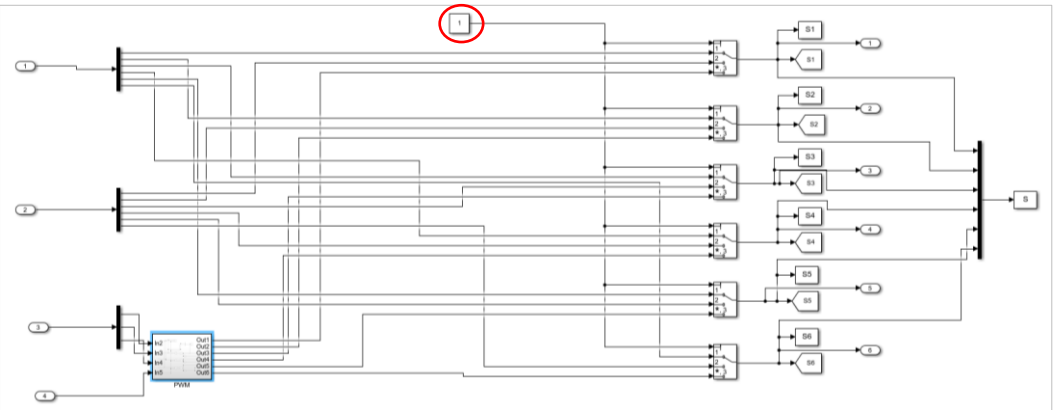


Figura 32. Constante para la selección de la señal de entrada en los multiplexores.

2.5.6.2. Función botón *Guardar* en la configuración de parámetros del variador.

Nuevamente, se utilizará una única función (Figura 33) con la que almacenar los valores configurados en los bloques ***edit_text***, para su posterior configuración en los bloques del modelo del variador.

En este caso, los parámetros modificados han de ir configurados en diferentes bloques del modelo. La frecuencia y tensión de entrada son parámetros configurados en la fuente de tensión trifásica (bloque *Three-Phase Source*) que simula la red de alimentación del variador; mientras que los valores de frecuencia de salida, índice de modulación y desfase, serán utilizados en los diferentes bloques que se encargan de crear las señales de pulsos para el control de los IGBTs (bloques *Three-Phase Source 1* y *Sawtooth Generator1* para la opción PWM diseñado, bloque *PWM Generator (2-Level)* para la opción SPWM y bloque *SVPWM Generator (2-Level)* para la opción de modulación SVPWM).

Por último, los valores para el filtro LC se configurarán en los bloques de inductancia y capacitancia trifásicas (bloques *Three-Phase Parallel RLC Branch* y *Three-Phase Parallel RLC Branch1*).

```
function pushbutton5_Callback(hObject, eventdata, handles)
%CONFIGURACION DE VALORES
F_entrada = get(handles.edit1,'String');
F_salida = get(handles.edit2,'String');
V_rms = get(handles.edit3,'String');
ma=get(handles.edit26,'String');
desfase = get(handles.edit28,'String');
L_filtro = get(handles.edit4,'String');
C_filtro = get(handles.edit5,'String');
%Configuración valores señal entrada
set_param('convertidor_PWM/Three-Phase Source','Frequency',F_entrada);
set_param('convertidor_PWM/Three-Phase Source','Voltage',V_rms);
%configuración valores modulación
%Modulación PWM mediante puertas logicas
mad=str2double(ma);
madd= mad*(3^(1/2))/(2^(1/2));
madds=string(madd);
set_param('convertidor_PWM/Three-Phase Source1','Voltage',madds);
set_param('convertidor_PWM/Three-Phase Source1','Frequency',F_salida);
set_param('convertidor_PWM/Sawtooth Generator1','Phase',desfase);
%Modulación PWM bloque
set_param('convertidor_PWM/PWM Generator (2-Level)','Freq',F_salida);
set_param('convertidor_PWM/PWM Generator (2-Level)','m',ma);
set_param('convertidor_PWM/PWM Generator (2-Level)','Phase',desfase);
%Modulación SVPWM bloque
SVPWM_param= strcat("[",ma," ",desfase," ",F_salida,"]");
set_param('convertidor_PWM/SVPWM Generator (2-Level)','ParUref',SVPWM_param);
%Configuración Filtro LC
set_param('convertidor_PWM/Three-Phase Parallel RLC Branch','Inductance',L_filtro);
set_param('convertidor_PWM/Three-Phase Parallel RLC Branch1','Capacitance',C_filtro);
```

Figura 33. Función interna del bloque pushbutton5.

2.5.7. Funciones para la representación gráfica de los resultados.

Para la representación de los resultados se han creado 3 secciones de gráficos, donde se podrá seleccionar la señal que se quiere plotear en cada una de ellas. También se han diseñado barras correderas (*sliders*) para simular herramientas de zoom en el eje horizontal y cursores vecticales. No obstante, en momentos donde se requiera de mayor precisión también se puede hacer uso de la barra de herramientas superior que tiene configurada la propia interfaz.

Como ya se comentó anteriormente, los bloques gráficos no tienen una propia función de ejecución ya que su misión consiste en la representación de datos, por lo tanto, se requiere de una función externa donde ejecutar el comando para graficar. En este caso, se programarán todas las funciones que afecten a la representación de datos, dentro de la función del menú desplegable donde se elige la señal a graficar. Es decir, los *sliders* también se ejecutarán dentro de esta función, de tal forma que, cada vez que se modifique la posición del slider, se ejecute la misma función que afecta a los gráficos y se represente rápidamente. Para configurar que un bloque se ejecute en una función diferente a la que tiene predeterminada por defecto, se debe configurar el nombre de la función que tiene asociada en el parámetro *Callback*, tal y como se muestra en la Figura 34.

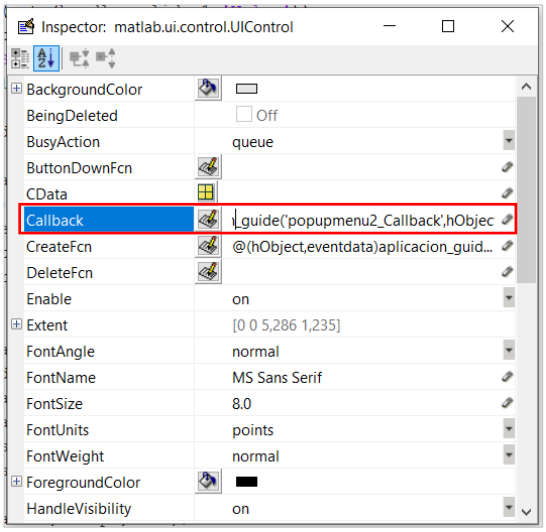


Figura 34. Declaración de la función de control del bloque.

Una vez referidos todos los bloques sobre la misma función ***popupmenu2*** (en el caso de los bloques de representación gráfica del lado superior derecho) se ha procedido con la programación de la representación gráfica de las distintas señales.

Como ya se comentó en el diseño del modelo del variador de frecuencia, todas las señales se van a almacenar en el *WorkSpace* de Matlab para que se pueda trabajar con ellas en las funciones internas de la interfaz. De esta forma, una vez se arranque una simulación se almacenarán los datos de cada señal en vectores de longitud igual a los incrementos de tiempo de simulación que haya generado Simulink. Además de los valores que conforman cada señal, interesa graficar el tiempo de simulación de las señales, el cual se usará siempre para plotear en el eje X. Para trabajar con todas estas señales en las funciones internas de la interfaz se guardarán en diferentes variables mediante el comando ***evalin.***

```
tiempo=evalin('base','tout');
V_DC=evalin('base','V_DC');
Vabc_entrada=evalin('base','Vabc_entrada');
```

Para configurar el *slider* que actuará de zoom en la gráfica, se ha parametrizado con un valor máximo bastante elevado, con la intención de que cada vez que se avance de posición el eje X del gráfico se divida por el valor del *slider* y el rango de valores representados vaya disminuyendo. Por lo tanto, dentro de la función únicamente será necesario almacenar el valor del *slider* en cada instante.

Una vez almacenados todos los valores anteriormente comentados, se programa un condicionante *if-else* para cada una de las posibles opciones a graficar. A continuación, se nombran las posibles señales a representar en el conjunto de las diferentes gráficas que hay diseñadas en la interfaz:

- Tensión Trifásica de Entrada
- Tensión Continua Convertida
- Tensión Trifásica de Salida Modulada
- Señal de Modulación
- Tensión Trifásica de Salida Filtrada
- Velocidad Motor
- Par Motor
- Corriente Estator Motor

Nota: Estas opciones se han repartido entre los 3 gráficos disponibles, con el fin de repetir la menor cantidad de código posible.

Dentro de cada condicionante se ha programado el ploteo de la señal correspondiente, así como la definición de valores y nombres de los ejes X e Y. También se hará referencia a los valores límite del eje X para que en el momento de usar el *slider zoom* se divida el rango de valores que haya en ese momento en el eje X, no el rango de valores de la señal (ya que estos no varían). Además, se incluye una leyenda que se modificará automáticamente en cada gráfico.

```
if Graf == 2
        plot(handles.axes1,tiempo,Vabc_entrada);
        lim =axis(handles.axes1);
        ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
        xlim(handles.axes1,[(lim(1)/zoom) (lim(2)/zoom)]);
        xlabel(handles.axes1,'Tiempo (s)');
        ylabel(handles.axes1,'Voltaje (V)');
        legend (handles.axes1,'a','b','c');
```

Para la configuración del *slider* que controla el cursor, se requiere del vector de datos *tiempo*; ya que, a partir de la longitud del vector se configurarán el valor máximo del *slider*, así como el tamaño de paso con el que se desplaza. Al estar configurados con el mismo tamaño, se podrá leer el valor que tiene el *slider* en todo momento y acceder con ese valor al vector *tiempo_celda* para saber en qué instante se encuentra y mostrarlo junto a la gráfica.

```
tout=evalin('base','tout');
set(handles.slider1,'Max',length(tout));
tiempo_celda = int32(get (handles.slider1,'Value'));
tiempo_valor = tout(tiempo_celda,1);
set(handles.text47,'String',tiempo_valor);
```

Finalmente, se grafica el cursor como una señal vertical en el instante indicado por el *slider*. Además, se ha creado una condición para que, en el momento que el cursor se quede fuera de rango por haber ampliado mucho la señal mediante zoom, el eje X se ajuste a un rango de valores donde el cursor pueda continuar desplazándose por la gráfica sin desaparecer. Para este proceso se ha utilizado una variable global capaz de almacenar el valor *tiempo_valor*, de esta forma se almacena el último valor del cursor y se parte del mismo para que, la próxima vez que se entre a la función, se sepa si se excedió el rango de valores del eje X.

```
aux1=handles.aux1;
if tiempo_valor>=(aux1+(lim(2)/zoom))
    aux1=tiempo_valor;
    handles.aux1=aux1;
    guidata(hObject,handles);
end
xlim(handles.axes1,[aux1 aux1+(lim(2)/zoom)]);
xline(handles.axes1,tiempo_valor,'--
','LineWidth',1.5,'DisplayName','Cursor');
```

En la Figura 35 se muestra la programación diseñada para la representación gráfica de las señales en la función *popupmenu2*.

```
function popupmenu2_Callback(hObject, eventdata, handles)
AUX2 = {get(handles.popupmenu2,'Value')};
Graf = cell2mat (AUX2);
%VALORES GRAFICAS
tiempo=evalin('base','tout');
V_DC=evalin('base','V_DC');
Vabc_entrada=evalin('base','Vabc_entrada');
Vabc_s=evalin('base','Vabc_s');
Vabc_salida=evalin('base','Vabc_salida');
S=evalin('base','S');
%CURSOR
tout=evalin('base','tout');
set(handles.slider1,'Max',length(tout));
tiempo_celda = int32(get (handles.slider1,'Value'));
tiempo_valor = tout(tiempo_celda,1);
set(handles.text47,'String',tiempo_valor);
%ZOOM
zoom =double(get(handles.slider3,'Value'));
%Modulacion SVPWM
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
Vreferencia=evalin('base','Vref');
vector_angulo=evalin('base','angulo');
angulo=vector_angulo(tiempo_celda,1);
   if Graf == 2
       plot(handles.axes1,tiempo,Vabc_entrada);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[(lim(1)/zoom) (lim(2)/zoom)]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   elseif Graf == 3
       plot(handles.axes1,tiempo,V_DC);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'Continua');
   elseif Graf == 4
       plot(handles.axes1,tiempo,Vabc_s);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   else
       plot(handles.axes1,tiempo,Vabc_salida);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   end

   aux1=handles.aux1;
if tiempo_valor>=(aux1+(lim(2)/zoom))
    aux1=tiempo_valor;
    handles.aux1=aux1;
    guidata(hObject,handles);
end
if tiempo_valor < (aux1+(lim(1)/zoom))
    aux1=tiempo_valor;
    handles.aux1=aux1;
    guidata(hObject,handles);
end
xlim(handles.axes1,[aux1 aux1+(lim(2)/zoom)]);
xline(handles.axes1,tiempo_valor,'--','LineWidth',1.5,'DisplayName','Cursor');
```

Figura 35. Función interna del bloque popupmenu2.

2.5.7.1. Selección de las señales de modulación a graficar

Como se ha comentado, se podrán plotear diferentes señales en cada una de las gráficas, de tal forma que, se ha configurado la gráfica *axes7* para la representación de las 6 señales de modulación de la etapa inversora de IGBTs.

Para la selección de cada una de las señales moduladoras, se han configurado 6 botones de selección *radio button*, con los cuales se podrá habilitar o deshabilitar la visualización de cada una de ellas. Para realizar esta operación, se almacenará el valor de cada uno de los botones y de las diferentes señales de modulación (*S1, S2, S3, S4, S5* y *S6*); de tal forma que, si alguno de estos botones se encuentra deshabilitado (valor 0), la señal se elimine de la gráfica.

```
S1=evalin('base','S1');
S1button={get(handles.radiobutton4,'Value')};
S1_button=cell2mat (S1button);
a = plot(handles.axes7,tiempo,S1,'DisplayName','S1');
legend (handles.axes7);
hold (handles.axes7,'on');
if S1_button==0
    delete(a);
end
```

Además, se ha añadido un cuarto gráfico *axes 8*, el cual va a tener la funcionalidad de mostrar las diferentes situaciones en las que se pueden encontrar los transistores IGTBTs del puente inversor. De tal manera que, en función del instante en que se encuentre el cursor del gráfico, se actualizará la imagen para que muestre los IGBTs que se encuentran en conducción en ese momento. Para ello, se ha leído el valor de cada una de las señales moduladoras en el instante que muestra el cursor; es decir, se ha leído la celda del vector que corresponde con ese instante de tiempo. Una vez leídos los valores, se utilizarán condicionantes para saber la imagen de conmutación de los transistores que le corresponde (Figura 36).

```
S1_on = S1(tiempo_celda,1);
S3_on = S3(tiempo_celda,1);
S5_on = S5(tiempo_celda,1);

if S1_on==1 && S3_on==1 && S5_on==1
    Ima=imread('111.jpg');
    image(handles.axes8,Ima);
```

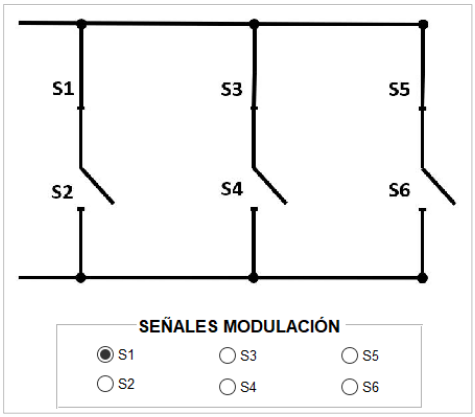


Figura 36. Ejemplo de una situación de modulación con los IGBTs 1,3 y 5 en conducción.

Por otro lado, se ha añadido una figura externa (Figura 37) que se encargará de representar el vector de referencia de la modulación SVPWM en el momento seleccionado por el cursor. Esta ventana únicamente aparecerá cuando se haya seleccionado la modulación mediante el control SVPWM, en la sección de botones para la configuración del variador de frecuencia.

```
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
Vreferencia=evalin('base','Vref');
vector_angulo=evalin('base','angulo');
angulo=vector_angulo(tiempo_celda,1);
Vref=Vreferencia(tiempo_celda,1);
if RB_3 ==1
    figure(1)
    x=cos(angulo)*Vref;
    y=sin(angulo)*Vref;
    c=compass(x,y);
    c.LineWidth=2;
end
```

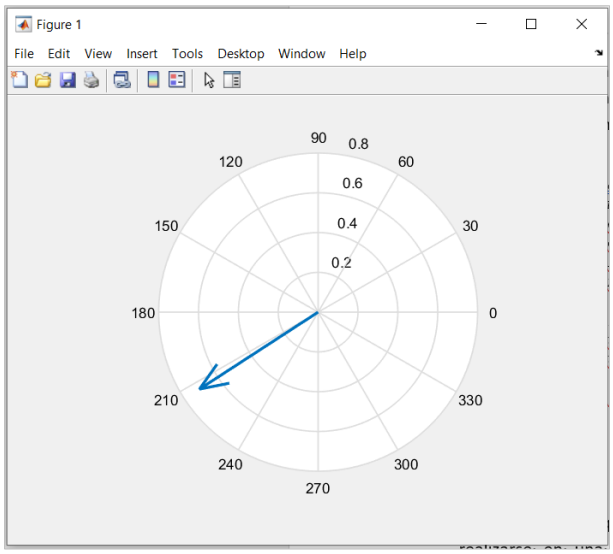


Figura 37. Ventana externa a la interfaz para la representación del vector de referencia en la modulación SVPWM.

Como ya se explicó en los apartados anteriores, estas operaciones deben realizarse en una única función, debido a que requieren unas de otras para el correcto funcionamiento de la gráfica. Por lo tanto, se deberán referenciar las funciones de los respectivos bloques a una única función, en este caso se ha seleccionado la del *slider6*.

CAPÍTULO 3. EJEMPLO DE APLICACIÓN DE LA INTERFAZ DE USUARIO

Se han realizado dos ejemplos donde visualizar el control de un motor eléctrico asíncrono mediante modulación SPWM y SVPWM, a partir de los cuales se podrá entender con mayor precisión los sistemas de control diseñados para un motor eléctrico asíncrono, además de entender de una forma más visual el funcionamiento de los diferentes bloques de la interfaz explicados en el desarrollo de ente proyecto

Para ambos ejemplo se ha configurado un motor de jaula de ardilla (*Squirrel Cage*) con un ajuste de parámetros por defecto: *5.4 HP (4KW) 400 V 50Hz 1430 RPM*. Por otro lado, para la entrada mecánica se ha configurado un par externo de 10 Tm (Figura 38).

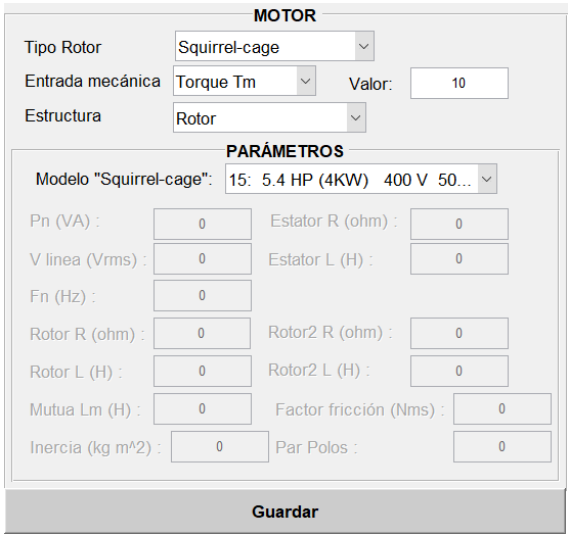


Figura 38. Configuración del motor eléctrico asíncrono.

3.1. MODULACIÓN SPWM

En este primer caso, se ha utilizado la **modulación SPWM** para el control del puente inversor del variador.

Con respecto a los valores de trabajo del variador, se ha seleccionado una tensión de entrada trifásica de línea igual a $480V_{RMS}$ y una frecuencia de entrada de 60 Hz. Sabemos por la configuración del motor, que este debe trabajar a una tensión de línea de $400V_{RMS}$ y una frecuencia de 50 Hz, por lo tanto, configuraremos la modulación y el filtrado para poder garantizar una alimentación del motor lo más óptima posible.

Para el cálculo del índice de modulación, se han utilizado las siguientes expresiones, de las cuales se ha obtenido un valor aproximado de **m_A = 0.962**.

$$V_{RMS\_salida} \times \sqrt{2} = \sqrt{3} \times m_A \times \frac{1}{2} V_{DC}$$

$$V_{DC} = V_{RMS\_entrada} \times \sqrt{2}$$

Para la etapa del filtro LC, se van a configurar unos valores de L = 20 mH y C = 0.1 mF. En la Figura 39 se muestran los parámetros configurados en la interfaz.

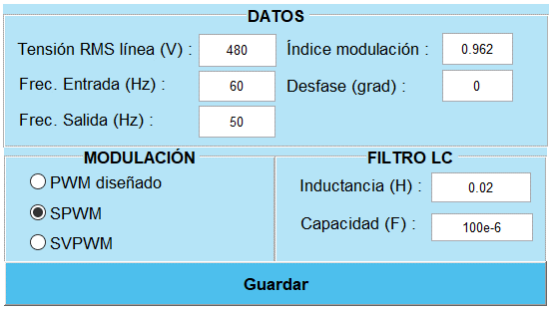


Figura 39. Configuración de los parámetros referentes al variador de frecuencia.

A partir de las gráficas se ha comprobado que los resultados obtenidos son los deseados para el control del motor asíncrono. En las siguientes gráficas, se muestran las señales correspondientes a la tensión trifásica de entrada (Figura 40a), la tensión continua rectificada (Figura 40b), la tensión trifásica obtenida tras la etapa inversora (Figura 40c) y la tensión trifásica filtrada antes de llegar a la entrada del motor (Figura 40d).

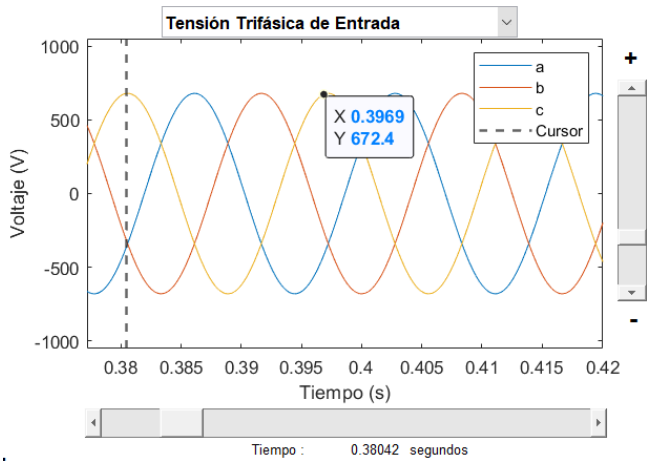


Figura 40a. Tensión trifásica de entrada al variador.

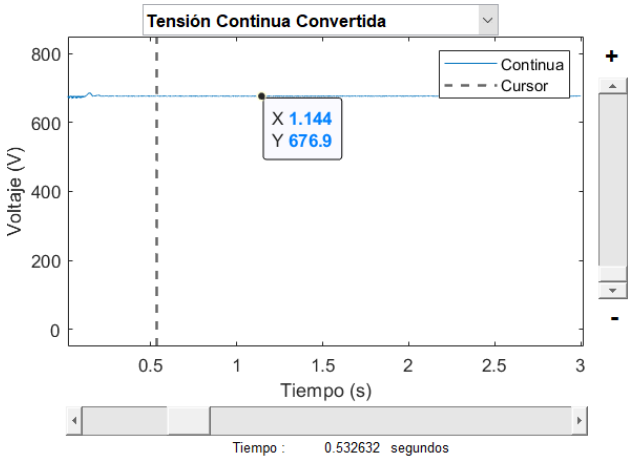


Figura 40b. Tensión continua rectificada

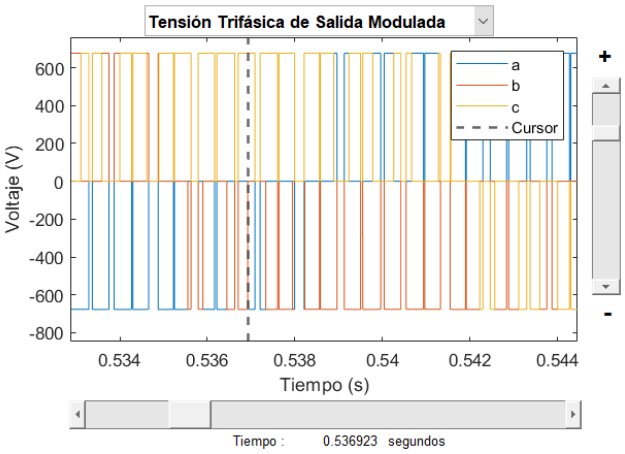


Figura 40c. Tensión trifásica modulada mediante SPWM.

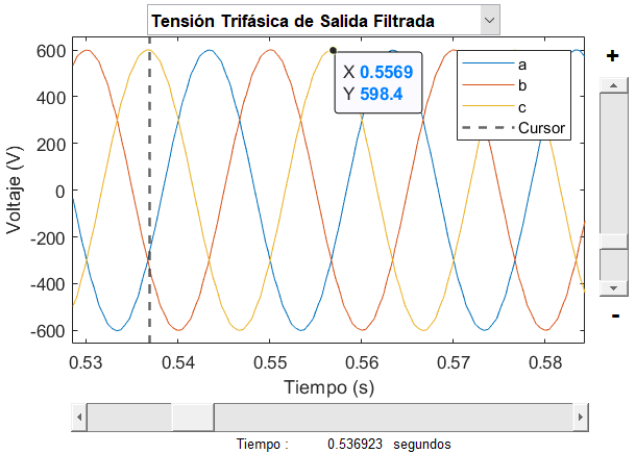


Figura 40d. Tensión trifásica tras el filtro LC.

Mediante la gráfica que muestra la señal de entrada se han obtenido los siguientes valores de tensión y frecuencia:

$$V_{línea} = 676.9\ V \rightarrow V_{línea}RMS = \frac{676.9}{\sqrt{2}} = 479\ V$$

$$F_{entrada} = \frac{1}{0.397 - 0.3804} = 60.2\ Hz$$

Igualmente, se han comprobado los valores de tensión y frecuencia obtenido en la señal de salida del variador, la cual se encargará de alimentar el motor eléctrico.

$$V_{línea} = 598.4\ V \rightarrow V_{línea}RMS = \frac{598.4}{\sqrt{2}} = 423.13\ V$$

$$F_{salida} = \frac{1}{0.5569 - 0.5369} = 50\ Hz$$

Las posibles desviaciones en los resultados prácticos frente a los teóricos podrían deberse a factores de precisión a la hora de tomar los datos. No obstante, se observa como los valores se ajustan perfectamente a los configurados en la interfaz.

Para la visualización de las señales de modulación SPWM, se ha utilizado la gráfica central de la interfaz y los diferentes botones de selección que habilitan/deshabilitan las señales a mostrar. En las siguientes gráficas se pueden visualizar cada una de ellas para un instante de tiempo determinado (Figuras 41a, b, c, d, e y f).

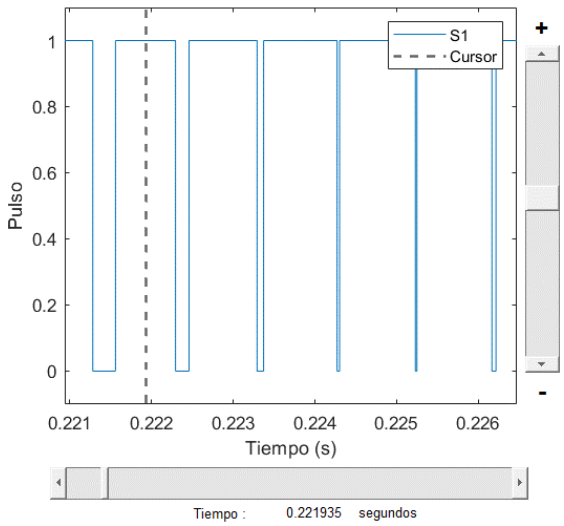


Figura 41a.Señal de control del transistor S1.

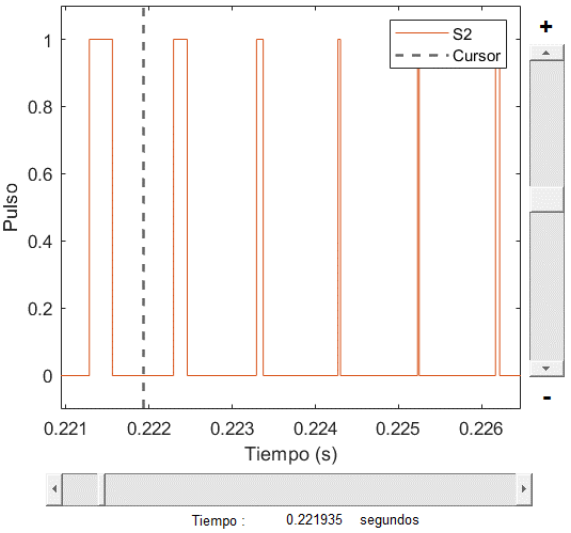


Figura 41b. Señal de control del transistor S2.

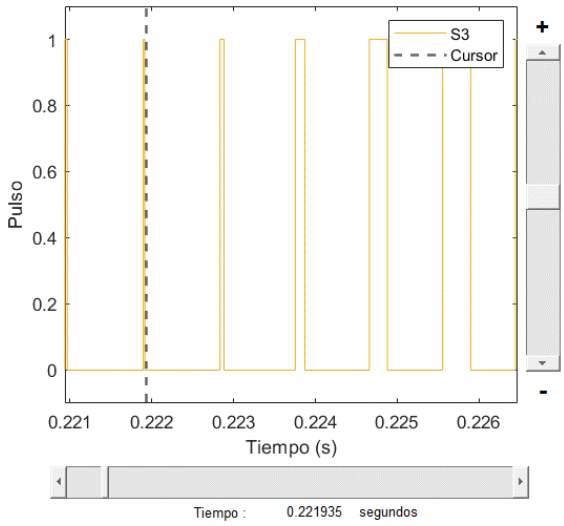


Figura 41c. Señal de control del transistor S3.

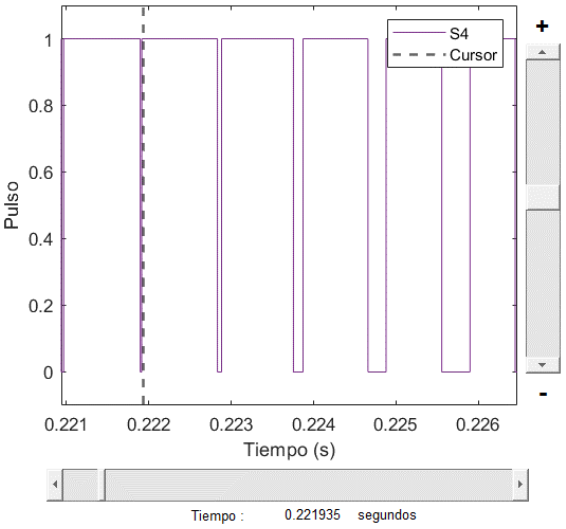


Figura 41d. Señal de control del transistor S4.

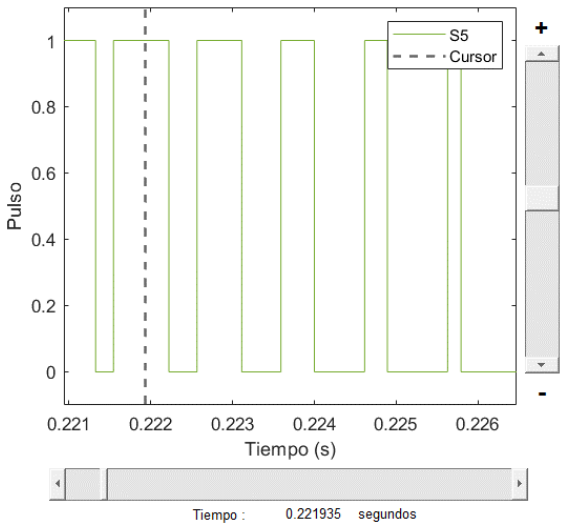


Figura 41e. Señal de control del transistor S5.

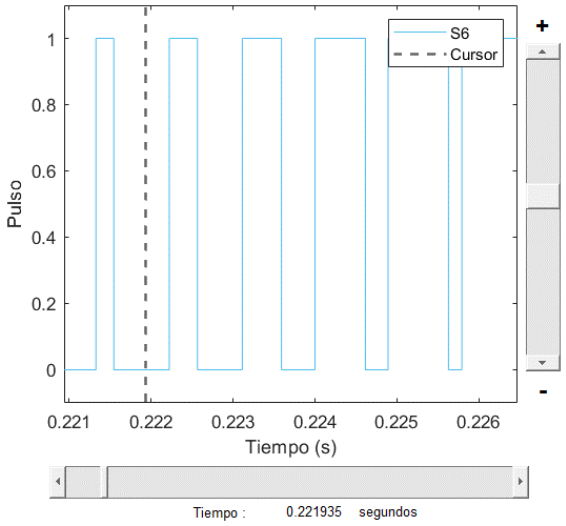


Figura 41f. Señal de control del transistor S6.

Para ese mismo instante se muestra, en la parte superior, el estado de conmutación de los transistores (figura 42).

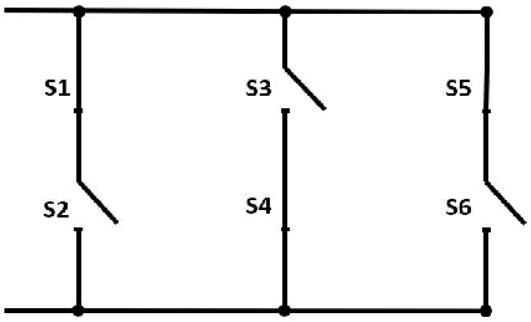


Figura 42. Representación de los transistores en conmutación.

Finalmente, se ha estudiado el funcionamiento del motor eléctrico mediante gráficos de velocidad de giro, par motor y corriente en el estator (Figuras 43a, b y c).

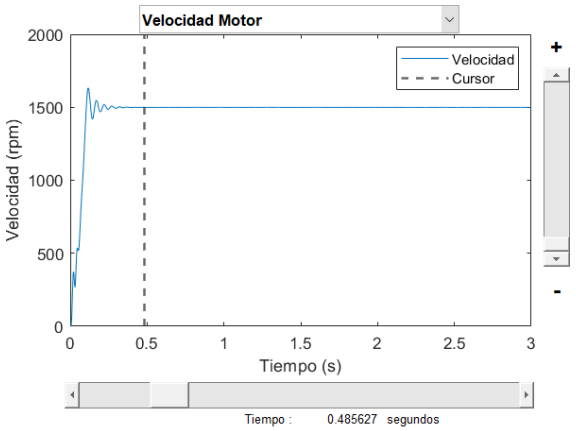


Figura 43a. Velocidad de giro del motor.

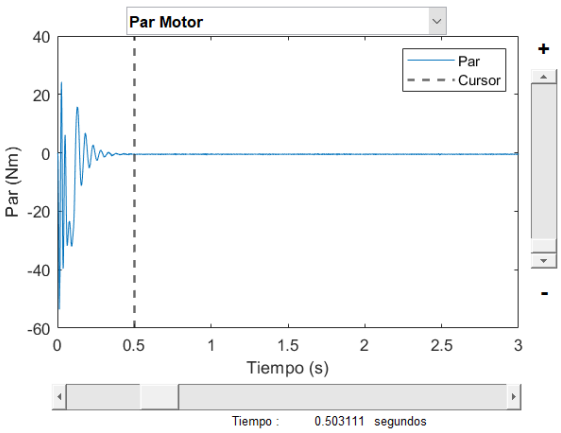


Figura 43b. Par motor.

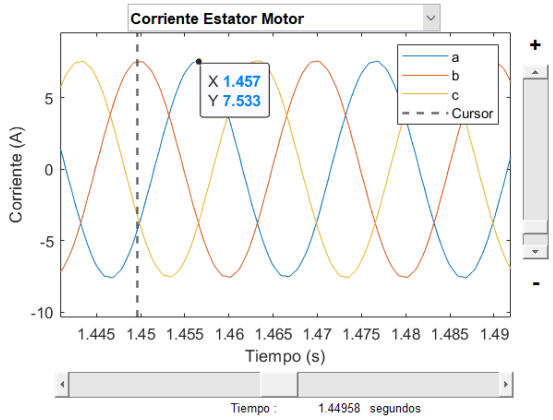


Figura 43c. Corriente trifásica que circula por el estator del motor.

A partir de la corriente que circula por el motor y la tensión con la que trabaja el mismo, se puede calcular la potencia consumida por la máquina eléctrica.

$$P_{motor} = V_{Linea} \times I_{Linea} = 598.4 \times 7.533 = 4507.74\ kW$$

3.2. MODULACIÓN SVPWM

Para el segundo caso, se ha utilizado la **modulación SVPWM** para el control del puente inversor del variador. Cabe destacar que con este tipo de modulación se consigue un valor eficaz de la tensión más elevado que en la modulación SPWM. Para demostrarlo, se ha configurado un valor distinto para la tensión eficaz de entrada 450 V_{RMS}, de tal forma que se pueda observar como con unos valores menores de tensión y modulación se consiguen resultados igualmente válidos para el correcto funcionamiento del variador de frecuencia.

Para el cálculo del índice de modulación, se han utilizado las siguientes expresiones, de las cuales se ha obtenido un valor aproximado de **m_A = 0.89**.

$$V_{RMS\_salida} \times \sqrt{2} = \sqrt{3} \times m_A \times \frac{1}{\sqrt{3}} V_{DC}$$

$$V_{DC} = V_{RMS\_entrada} \times \sqrt{2}$$

Para la etapa del filtro LC, nuevamente se van a configurar unos valores de L = 20 mH y C = 0.1 mF. En la figura 44 se muestran los parámetros configurados en la interfaz.

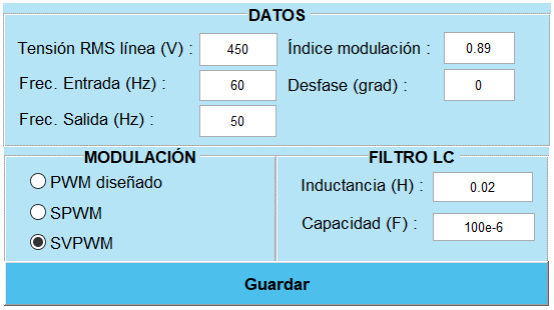


Figura 44. Configuración de los parámetros referentes al variador de frecuencia.

A partir de las gráficas se ha comprobado que los resultados obtenidos son los deseados para el control del motor asíncrono. En las siguientes gráficas, se muestran las señales correspondientes a la tensión trifásica de entrada (Figura 45a), la tensión continua rectificada (Figura 45b), la tensión trifásica obtenida tras la etapa inversora (Figura 45c) y la tensión trifásica filtrada antes de llegar a la entrada del motor (Figura 45d).

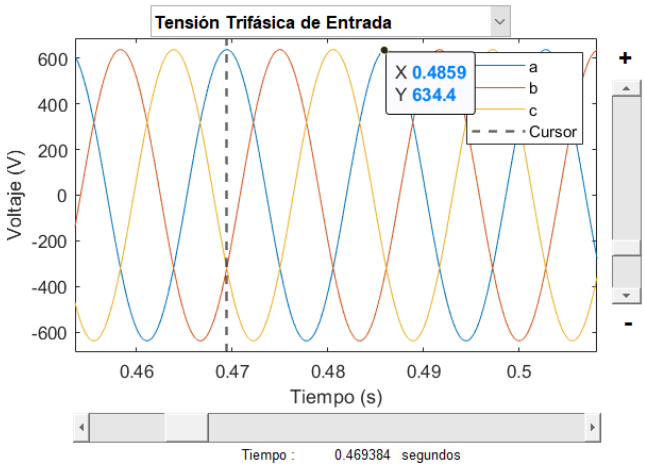


Figura 45a. Tensión trifásica de entrada al variador.

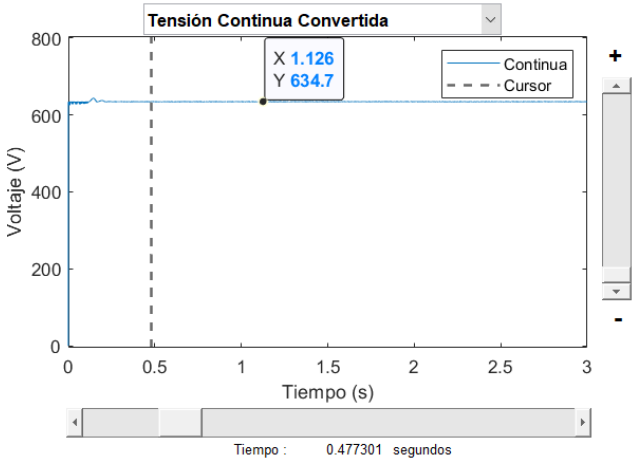


Figura 45b. Tensión continua rectificada

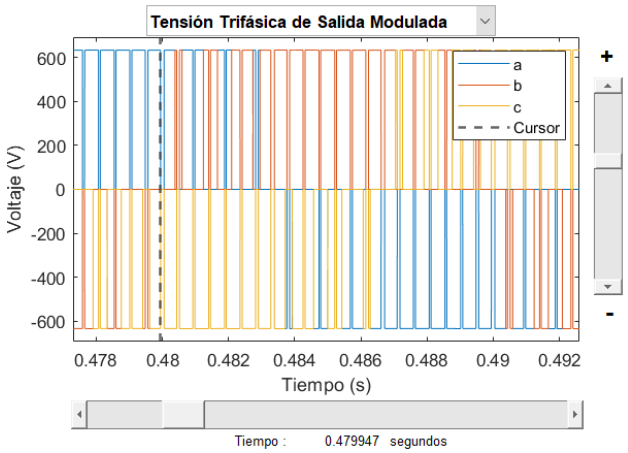


Figura 45c. Tensión trifásica modulada mediante SVPWM.

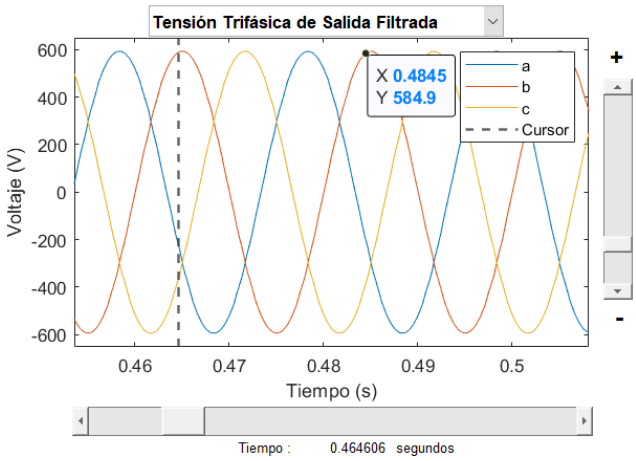


Figura 45d. Tensión trifásica tras por el filtro LC.

Mediante la gráfica que muestra la señal de entrada se han obtenido los siguientes valores de tensión y frecuencia:

$$V_{línea} = 636.3\,V \rightarrow V_{línea}RMS = \frac{636.6}{\sqrt{2}} = 449.93\,V$$

$$F_{entrada} = \frac{1}{0.4859 - 0.4693} = 60.2\,Hz$$

Igualmente, se han comprobado los valores de tensión y frecuencia obtenido en la señal de salida del variador, la cual se encargará de alimentar el motor eléctrico.

$$V_{línea} = 592.3\,V \rightarrow V_{línea}RMS = \frac{592.3}{\sqrt{2}} = 418\,V$$

$$F_{salida} = \frac{1}{0.4845 - 0.4646} = 50.2\,Hz$$

Resaltar que la modulación SVPWM se ajusta mejor a los valores de tensión de salida para la alimentación del motor, que la modulación SVPWM estudiada en el ejemplo anterior. No obstante, las pequeñas desviaciones en los resultados prácticos frente a los teóricos podrían deberse a factores de precisión a la hora de tomar los datos.

Para la visualización de las señales de modulación, se ha utilizado la gráfica central de la interfaz y los diferentes botones de selección que habilitan/deshabilitan las señales a mostrar. En las siguientes gráficas se pueden visualizar cada una de ellas para un instante de tiempo determinado (Figuras 46a, b, c, d, e y f).

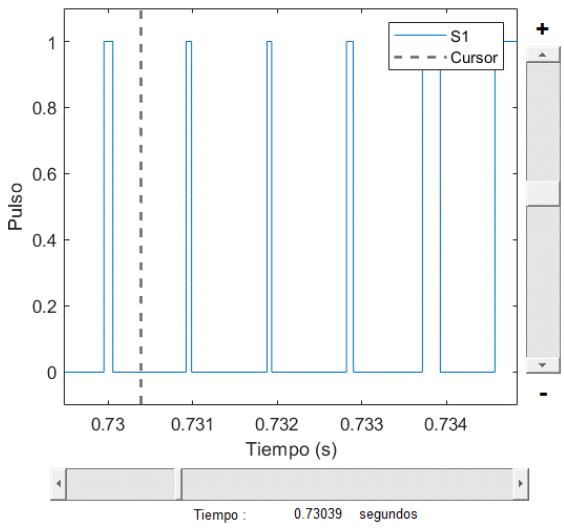


Figura 46a.Señal de control del transistor S1.

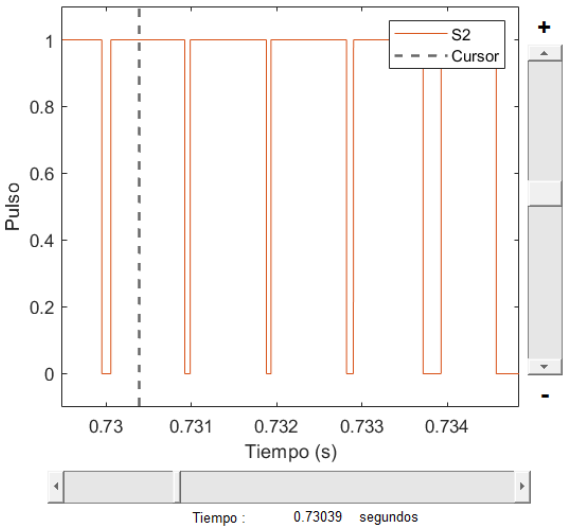


Figura 46b. Señal de control del transistor S2.

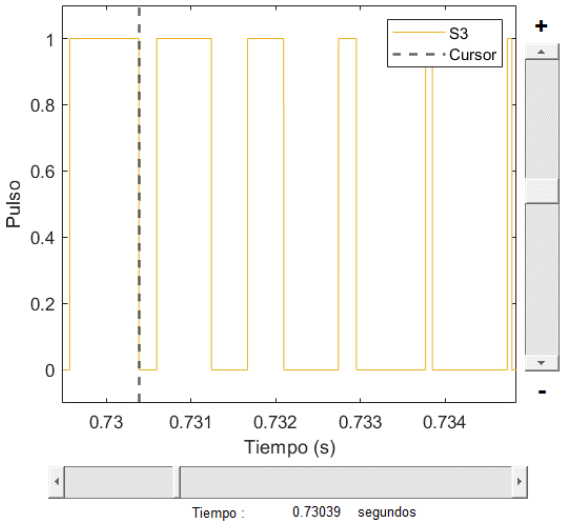


Figura 46c. Señal de control del transistor S3.

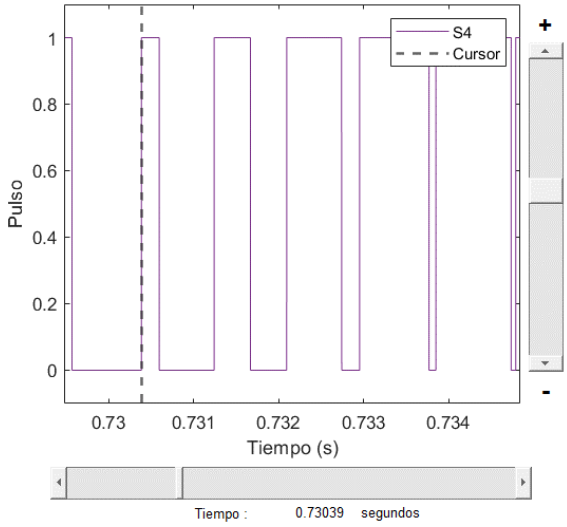


Figura 46d. Señal de control del transistor S4.

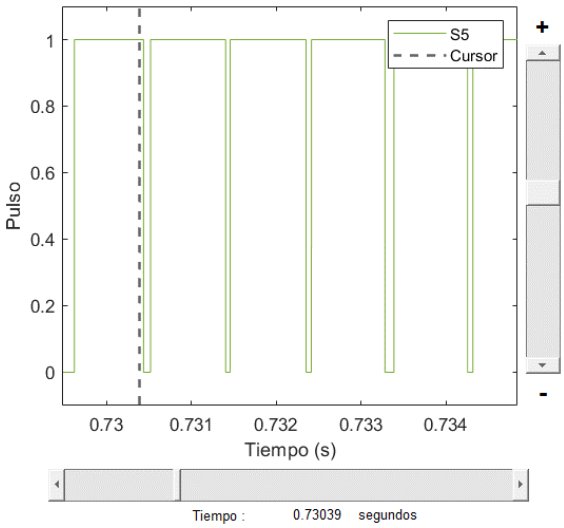


Figura 46e. Señal de control del transistor S5.

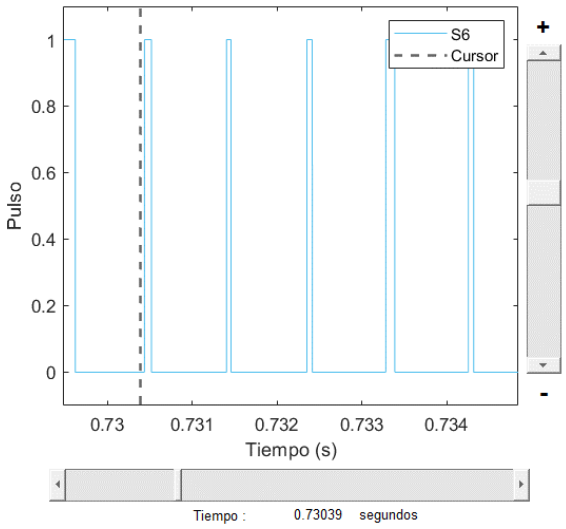


Figura 46f. Señal de control del transistor S6.

Para ese mismo instante se muestra, en la parte superior, el estado de conmutación de los transistores (Figura 47); así como el vector de referencia creado por el control SVPWM, representado a partir de una ventana externa para gráficos vectoriales (Figura 48).

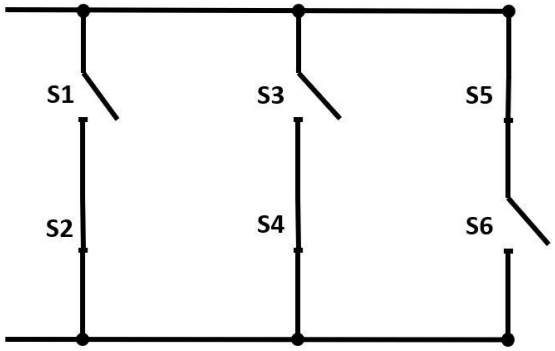


Figura 47. Representación de los transistores en conmutación.

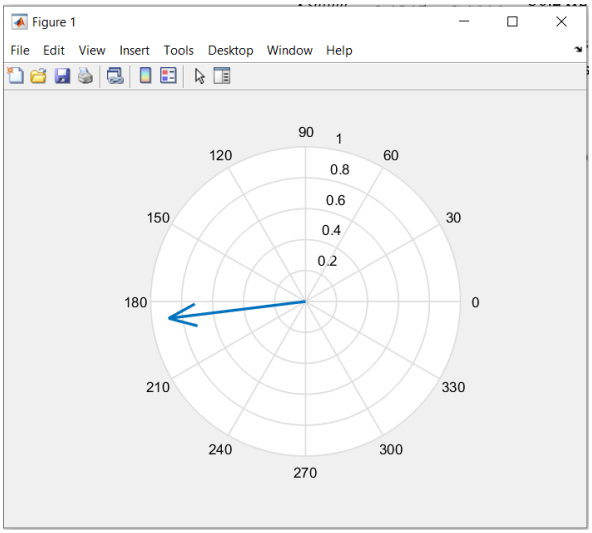


Figura 48. Ventana externa para la representación del vector referencia en el control SVPWM.

Finalmente, se ha estudiado el funcionamiento del motor eléctrico mediante gráficos de velocidad de giro, par motor y corriente en el estator (Figuras 49a, b y c).

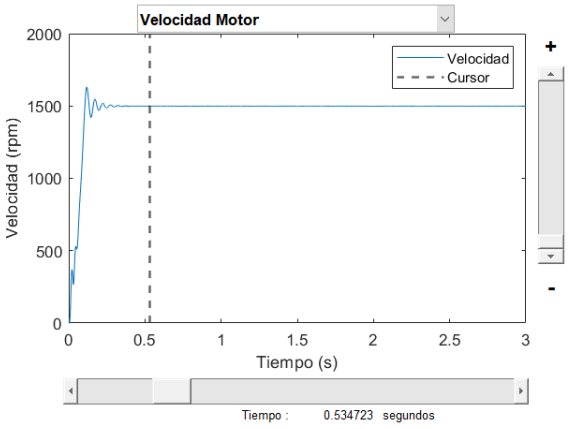


Figura 49a. Velocidad de giro del motor.

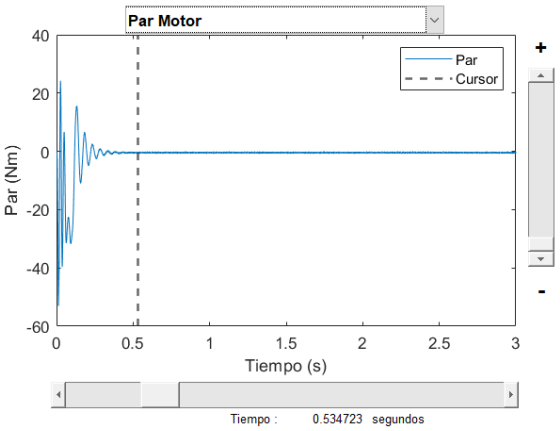


Figura 49b. Par motor.

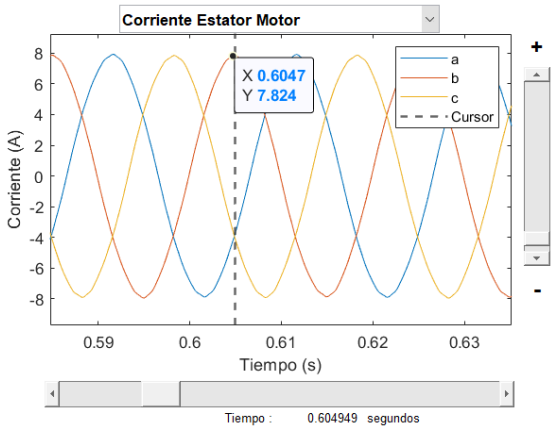


Figura 49c. Corriente trifásica que circula por el estator del motor.

A partir de la corriente que circula por el motor y la tensión con la que trabaja el mismo, se puede calcular la potencia consumida por la máquina eléctrica.

$$P_{motor} = V_{Linea} \times I_{Linea} = 592.3 \times 7.824 = 4634.15\ kW$$

CAPÍTULO 4. CONCLUSIONES

El control de las máquinas eléctricas es una herramienta imprescindible en el ámbito de aprendizaje de cualquier ingeniería industrial.

Este proyecto tenía como objetivo principal la creación de un método de aprendizaje interactivo y visual, de distintos sistemas de control y modulación en las máquinas eléctricas; de tal forma que, pudiera ayudar a los alumnos en el estudio y comprensión de aquellas asignaturas que traten con estas temáticas en su planificación educativa.

A la hora de desarrollar el modelo de control trifásico a partir de la herramienta de Simulink, se han considerado los dos sistemas de modulación más comunes para nivelación de dos niveles con puentes de IGBTs, como son la modulación SPWM y SVPWM.

Para alcanzar el objetivo principal del proyecto, así como los diferentes objetivos secundarios planteados, ha sido necesario familiarizarse con la herramienta de software de cálculo, Matlab. A partir de este programa, se ha podido realizar un diseño del modelo del variador de frecuencia mediante diferentes bloques de la herramienta Simulink; además de diseñar una interfaz de usuario con la herramienta GUIDE.

Mediante la herramienta GUIDE se han podido programar las diferentes funciones internas de la interfaz para obtener, finalmente, una aplicación de usuario con la que controlar y parametrizar un variador de frecuencia y ver los resultados obtenidos de una manera gráfica, facilitando así el aprendizaje de los alumnos.

Analizando el resultado obtenido tras la realización de este proyecto, se puede decir que se ha conseguido lograr el objetivo principal del trabajo, ya que se ha obtenido una interfaz de fácil interacción con el usuario, con la que poder parametrizar y estudiar modelos de control en motores eléctricos. Se ha diseñado un interfaz sencilla y completa, accesible para cualquier usuario que disponga del software Matlab, sea cual sea la licencia utilizada.

Además, se ha diseñado el modelo de un variador de frecuencia trifásico de 2 niveles accesible para el usuario al inicializarse automáticamente a partir de la interfaz. Mediante este modelo se pueden observar las diferentes fases por las que pasa la señal y que están parametrizadas totalmente por el usuario desde la aplicación.

Como conclusión final, se puede afirmar que el objetivo se ha alcanzado satisfactoriamente y que, se espera que tenga un uso académico satisfactorio en el aprendizaje de los alumnos en todas aquellas asignaturas donde sea necesario el control de las máquinas eléctricas.

De manera futura, se podría plantear trasladar esta aplicación creada mediante el software GUIDE, a un software más específico para la creación de interfaces de usuario, el cual podría ofrecer mayor diversidad de bloques para los gráficos de resultados, e incluso, aportar una apariencia más actual a la interfaz. Sin embargo, es algo que se debe meditar minuciosamente ya que se requiere de una interacción continua con el modelo de Simulink, y podría acarrear grandes dificultades.

CAPÍTULO 5. PRESUPUESTO

El presupuesto para llevar a cabo este proyecto se ha enfocado hacia el coste económico, temporal y personal, de una persona contratada especialmente para realizar este trabajo y con estos objetivos, principalmente, educativos.

Para los costes económicos se deberá tener en cuenta el desglose de horas dedicadas al proyecto, las cuales tendrán un sueldo acorde con la función del trabajador; así como las licencias de los diferentes softwares utilizados para la programación de la interfaz de usuario.

En la tabla 1 se muestra el desglose de horas dedicadas al proyecto, divididas por fases y objetivos.

Tabla 1. Desglose de tiempo empleado en la realización del proyecto

FASE	OBJETIVO	HORAS DEDICADAS
Creación modelo variador de frecuencia mediante Simulink	Diseño variador de frecuencia	10
	Diseño diferentes criterios de modulación	8
	Añadir elementos de medición y almacenamiento de resultados	3
Diseño de la interfaz de usuario mediante GUIDE	Diseño de los bloques que componen la interfaz	35
	Diferenciación entre secciones para la configuración de parámetros y la representación de los resultados	4
	Programación de las funciones internas de los diferentes bloques	180
	Programación de la interacción entre el modelo diseñado en Simulink y los bloques diseñados en la interfaz GUI	50
Análisis del proyecto	Realización de un ejemplo de variación de frecuencia y tensión de una señal mediante SVPWM	10
Informe del proyecto	Redacción de antecedentes, desarrollo y resultados del proyecto	50

A partir de esta tabla, se puede calcular el tiempo total dedicado al diseño, desarrollo, análisis y disertación del proyecto, resultando un total de **350 horas**.

Una vez conocidas las horas de trabajo, se debe tener en cuenta el salario asociado a profesionales de nivel "Licenciados y titulados de 2º y 3º ciclo universitario y analista" en el convenio colectivo nacional para empresas de la ingeniería y oficinas de estudios técnicos, declarado en el año 2020 del Boletín Oficial del Estado (BOE). El cual declara un valor salarial de 26.323,57 € anuales.

Realizando los cálculos a partir de las horas implicadas en el proyecto y las horas anuales legisladas por convenio en el Estatuto de los Trabajadores, el cual establece una jornada máxima de 1828 horas anuales, se obtiene un **coste económico salarial de 5.040,07 euros**.

$$Salario = \frac{26323{,}57\ €/año}{1828\ h/año} \times 350\ h = 5.040{,}07\ €$$

Además, se debe tener en cuenta el valor económico de la maquinaria de trabajo, en este caso, se sumará únicamente el coste del ordenador empleado para la realización del proyecto, el cual aportar unos **costes de 1050,00 euros**.

En este caso, se ha realizado un proyecto a partir de diferentes softwares de cálculo (Matlab) y ofimática (Office 365). Ambos programas tienen convenio con la Universidad Politécnica de Valencia y han permitido que se realice el trabajo a partir de licencias de estudiante, haciendo que el coste de licencias de trabajo sea de **0€** en el presupuesto.

Finalmente, se ha aplicado un porcentaje muy bajo de costes indirectos 2% ya que, en este caso, las posibles variaciones del presupuesto afectan ínfimamente en los costes al tener un número muy bajo de materiales implicados en el desarrollo del proyecto. Aplicando este porcentaje, se obtienen unos **costes presupuestados totales de 6211,88 euros**.

$$Presupuesto\ final = (5040{,}07 + 1050{,}00) \times 1{,}02 = 6211{,}88\ €$$

En la Tabla 2 se muestra un resumen del presupuesto desarrollado para el presente proyecto.

Tabla 2. Resumen de costes para el presupuesto del proyecto.

RESUMEN DE COSTES (€)	
PERSONAL	5040,07
MATERIAL	1050,00
INDIRECTO	121.80
TOTAL	6211,88 €

Para terminar, se ha planteado la alternativa de que este proyecto pueda llevarse a cabo fuera de la universidad, por ejemplo, en una consultoría de ingeniería donde se requiera de una formación de máquinas eléctricas a los trabajadores. En este caso, se deberá tener en cuenta el coste de las licencias para los softwares utilizados: Matlab y Office 365 (Tabla 3). Esto aporta unos costes extras de **145,37 euros**.

Tabla 3. Costes asociados a los softwares de cálculo y ofimática.

Software	Coste anual (€/año)	Coste proyecto (€)
Matlab	800,00	115,97
Office 365	202,80	29,40

De esta forma, los **costes materiales han aumentado hasta un total de 1195,37 euros**.

Nuevamente, se aplica un porcentaje de costes indirectos 2% ya que, se sigue manteniendo un número muy bajo de materiales implicados en el desarrollo del proyecto. Aplicando este porcentaje, se obtienen unos **costes presupuestados totales de 5111,65 euros**.

$$Presupuesto\ final = (5040{,}07 + 1195{,}37) \times 1{,}02 = 6360{,}15\ €$$

En la Tabla 4 se muestra un resumen del presupuesto desarrollado para el caso en que hubiera que sumar un coste de licencias al presupuesto inicial del proyecto.

Tabla 4. Resumen de costes para el presupuesto del proyecto.

RESUMEN DE COSTES (€)	
PERSONAL	5040,07
MATERIAL	1195,37
INDIRECTO	124,71
TOTAL	6360,15 €

A modo de conclusión, se puede comentar que la **diferencia entre ambos presupuestos redondea un valor de 148,27 euros**, un precio extra bastante asumible por cualquier empresa o institución que requiera de estos softwares para trabajar con el proyecto.

CAPÍTULO 6. BIBLIOGRAFÍA

- Ponce Cruz, Pedro & Sampé López, Javier (1971). ***Máquinas eléctricas y técnicas modernas de control.*** México D.F.
- Sanz Feito, J (2002). ***Máquinas eléctricas.*** Madrid.
- Sul, Seung-Ki (2011). ***Control of electric machine driver systems.*** Hoboken.
- Fraile Mora, Jesús (2011). ***Máquinas eléctricas.*** Madrid.
- Asignatura "Control de máquinas y accionamientos eléctricos" de la Universidad Politécnica de Valencia. Apuntes ***Control del motor de inducción.***
- Asignatura "Electrónica de potencia" de la Universidad Politécnica de Albacete. Apuntes ***Control y modulación.***
- Página web oficial del software Matlab. ***Mathworks.*** https://es.mathworks.com/
- Página web oficial de PSIM. ***Powersimtech***. https://powersimtech.com/products/psim/
- Página web oficial de PSCAD. ***Pscad***. https://www.pscad.com/software/pscad/overview
- Página web oficial de Microsoft Visual Studio. ***Visualstudio***. https://visualstudio.microsoft.com/es/vs/
- Página web oficial de Adobe XD. ***Adobe***. https://www.adobe.com/es/products/xd/prototyping-tool.html
- Manual básico de creación de una interfaz GUIDE. ***Crea una aplicación sencilla usando GUIDE.*** https://es.mathworks.com/help/matlab/creating_guis/about-the-simple-guide-gui-example.html
- Página web oficial del software Office 365.***Microsoft.*** https://microsoft.com.
- XIX apartado del BOE. ***Convenio colectivo nacional de empresas de ingeniería y oficinas de estudios técnicos***. https://tecniberia.es/publicacion-convenio-colectivo-ingenieria/
- Biblioteca de trabajos de la Universidad Politécnica de Valencia, ***Riunet***. https://riunet.upv.es/

ANEXOS

ANEXO A: GUÍA PARA EL DISEÑO DE LA INTERFAZ DE USUARIO MEDIANTE GUIDE

Para desplegar la ventana de programación gráfica GUIDE, basta con escribir *guide* en la ventana de comandos de Matlab. Seguidamente, el software preguntará qué tipo de interfaz se va a diseñar, donde se ha seleccionado la opción *Black GUI Default* (Figura 50).

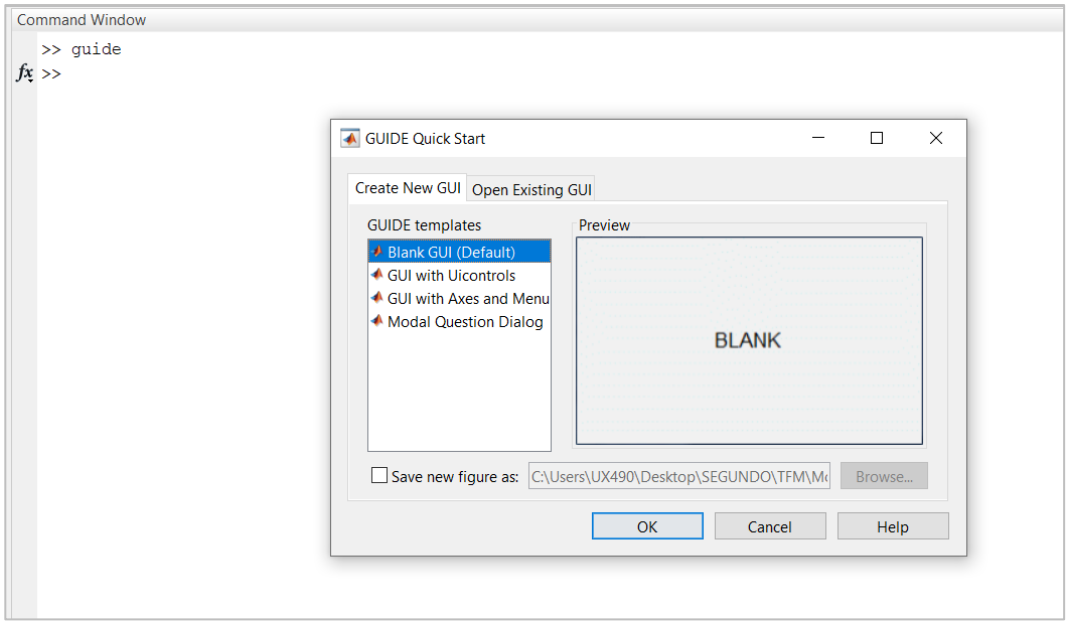


Figura 50. Ventana de selección del tipo de GUIDE a diseñar.

GUIDE genera de forma automática dos tipos de archivos, un archivo *.fig* encargado del diseño gráfico de la interfaz, y un archivo *.m* donde se desarrollan las funciones de programación de los distintos bloques diseñados.

Una vez abierta la ventana de diseño de GUIDE (archivo *.fig*), Figura 51, en la parte izquierda de la ventana se encuentran todos los tipos de bloques de programación que se han utilizado para el diseño de la interfaz, y en la parte central, se ha configurado el tamaño que tendrá la ventana de la aplicación.

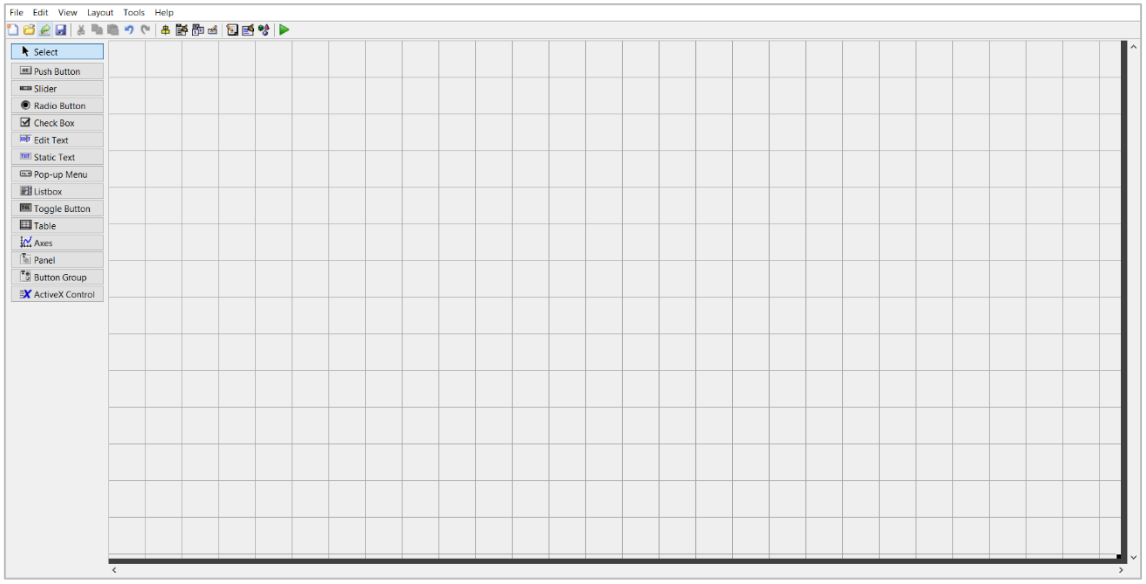


Figura 51. Ventana inicial para el diseño de la interfaz GUI.

También se ha configurado la barra de herramientas de la aplicación accediendo al botón *Toolbar Editor* para añadir los botones de Zoom y Cursor, ya que se va a trabajar con gráficos y facilitará mucho el estudio de los resultados (Figura 52). Estas herramientas, son independientes de las creadas mediante las funciones de los bloques, explicadas en el desarrollo del documento.

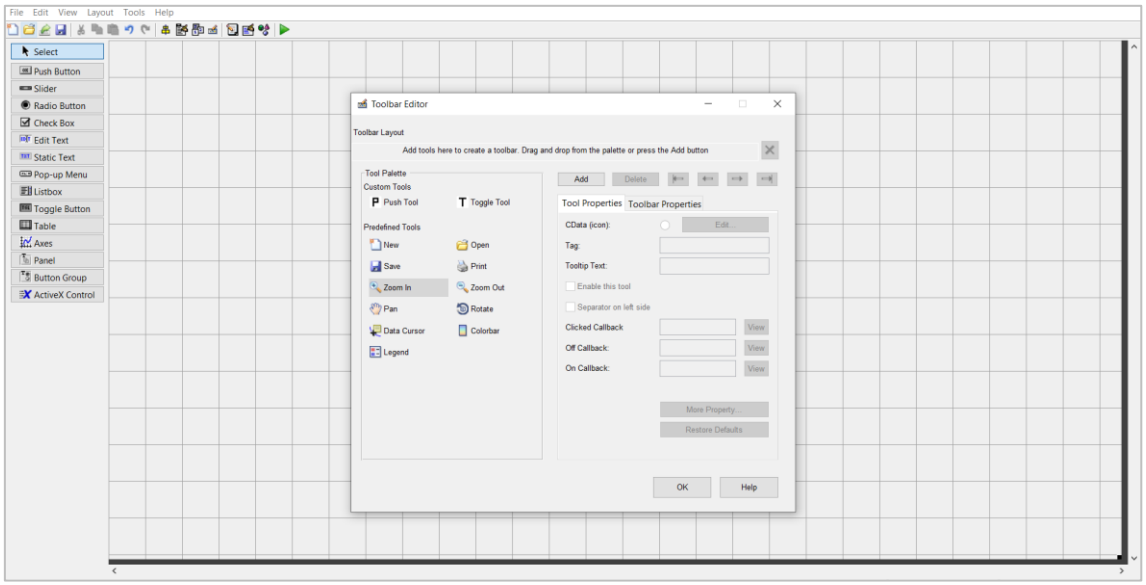


Figura 52. Ventana para la configuración de la barra de herramientas de la interfaz.

Para terminar con la configuración base de la interfaz se ha cambiado el color de fondo de la misma pulsando con el botón derecho sobre el fondo de la aplicación y modificando el parámetro *color*, tal como se muestra en la Figura 53.

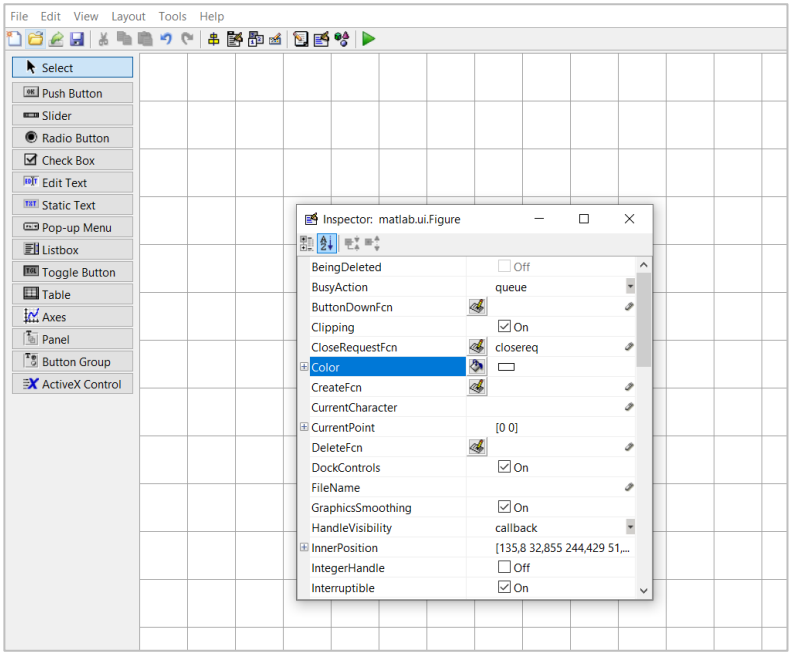


Figura 53. Configuración del color de fondo de la interfaz.

La interfaz se compone principalmente de botones, bloques editables y gráficos, con los cuales se podrán configurar las características de las señales de entrada y salida, el inversor, el filtro de la señal de salida y del motor; así como, analizar los resultados obtenidos e iniciar/parar la simulación.

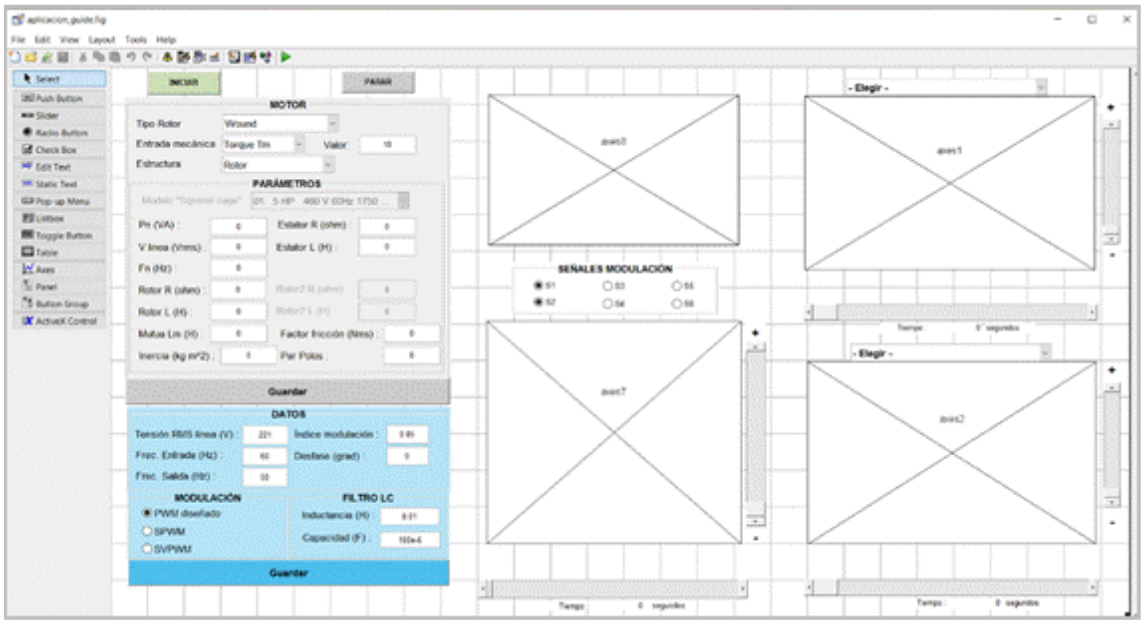


Figura 54. Diseño pantalla de inicio de la interfaz GUI.

En la Figura 54 se puede visualizar el modelo creado para la interfaz GUIDE. A continuación, se explican con mayor precisión cada una de las secciones de la interfaz.

Opciones INICIAR / PARAR

Se han diseñado 2 botones con los que se podrá iniciar y parar la simulación del modelo de variador de frecuencia creado en Simulink. Para el diseño se ha seleccionado el **bloque *Push Button*** y se ha configurado el estilo de cada botón, el nombre interno con el que se van a denominar (parámetro *Tag*) y el nombre que se muestra en el bloque (parámetro *String*). La Figura 55 muestra la ventana de configuración del botón *INICIAR.*

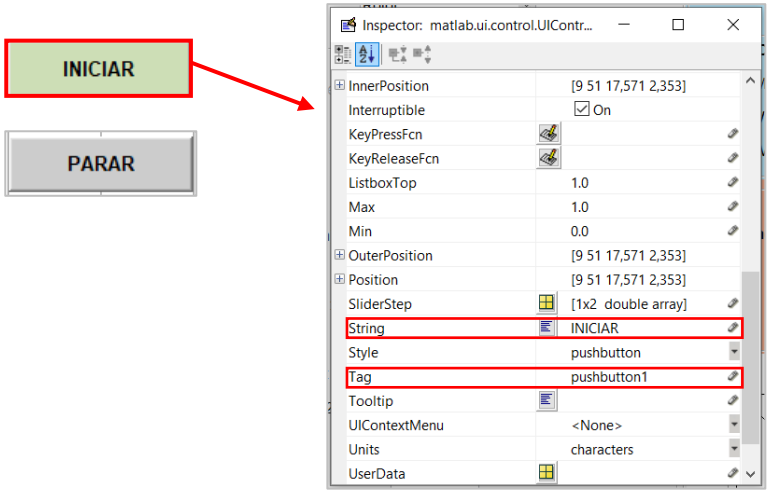


Figura 55. Ventana para la configuración de parámetros del botón INICIAR.

Configuración de parámetros del modelo

Para el diseño de la ventana de parámetros del motor se han estudiado las diferentes pestañas de características que dispone el bloque *Asynchronous Machine SI Units* en Simulink, y de esta forma, poder añadir todos los bloques necesarios para su configuración. Se han tenido en cuenta aquellos parámetros que se habilitan o deshabilitan en función de las características seleccionadas para el motor, de tal forma que se represente de igual manera en la interfaz. En la Figura 56 se muestra el ejemplo de cómo los parámetros de impedancia e inductancia para el segundo rotor están deshabilitados por no estar seleccionado el tipo de rotor *Double Squirrel-Cage*.

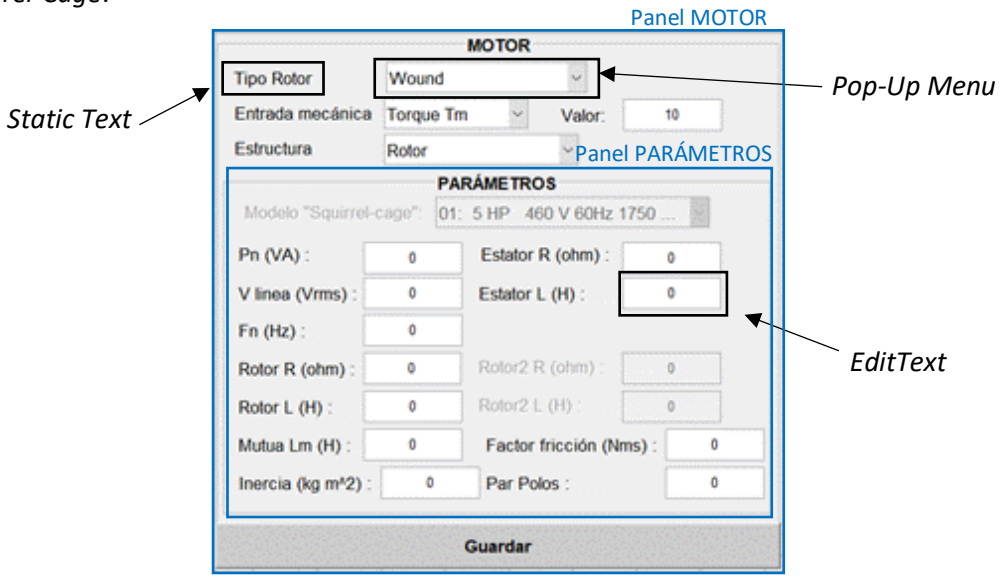


Figura 56. Bloque diseñado en GUI para la configuración de los parámetros del motor asíncrono de Simulink.

El diseño de esta sección se compone, principalmente, de bloques de texto (*Static Text*), bloques editables (*Edit Text*) y bloques de menú desplegable (*Pop-Up Menu*), agrupados en 2 paneles: MOTOR y PARÁMETROS.

Los bloques *Static Text* se han utilizado como etiquetas, por lo que será suficiente con definir el nombre de cada uno. Para los bloques editables *Edit Text*, se ha configurado el valor de inicio del mismo (en este caso, es suficiente con poner el valor como *string*) y el nombre interno del bloque al cual se hará referencia en las funciones de programación, Figura 57.

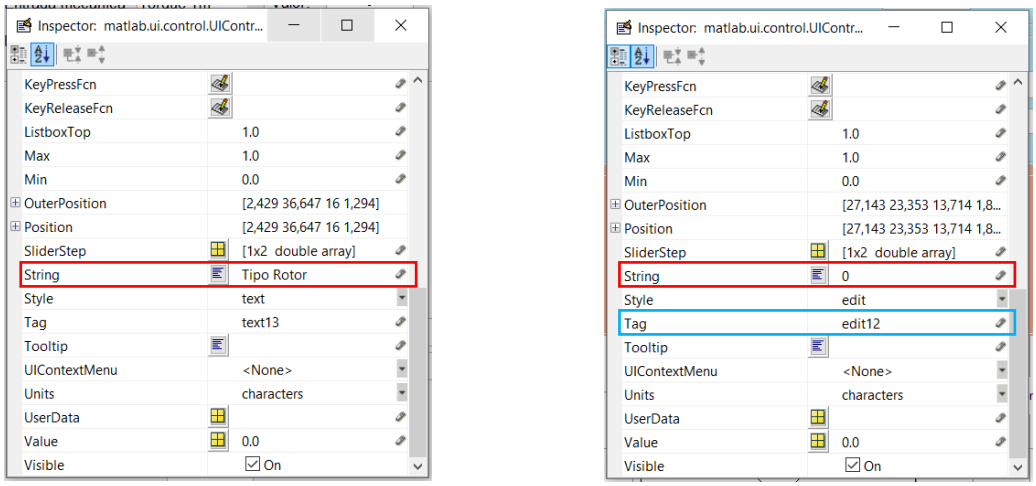


Figura 57. Parámetros a configurar en los bloques *Static Text* (izquierda) y *Edit Text* (derecha).

Por último, en los bloques *Pop-Up Menu* se han introducido todas las opciones que se quieren mostrar al desplegar la flecha del lado derecho, y el nombre interno del bloque para trabajar con él mediante funciones de programación, Figura 58.

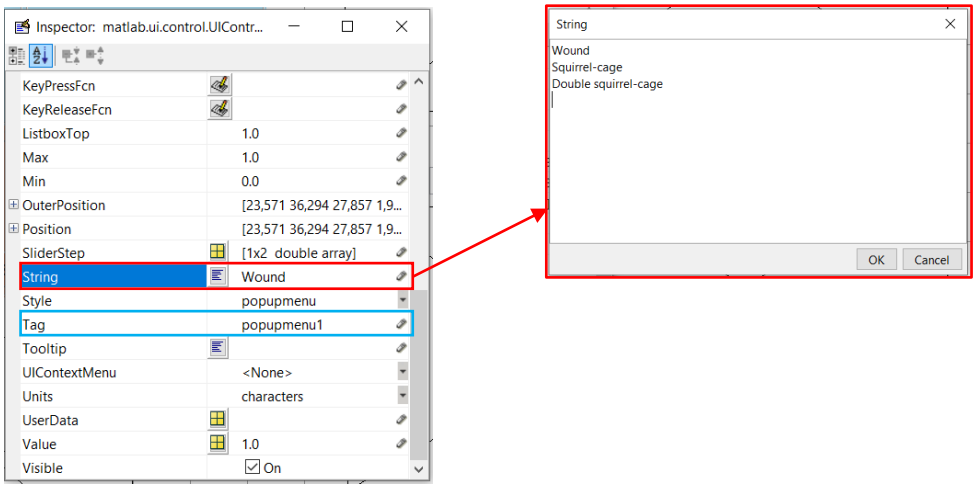


Figura 58. Parámetros a configurar en los bloques *Pop Up Menu*.

Por otro lado, se ha creado una ventana de parámetros para configurar el modelo del variador de frecuencia. Se han diferenciado 3 secciones, donde se pueden configurar los valores de las señales de entrada y salida, así como los criterios de modulación y el filtro LC de la señal se salida, tras la etapa inversora.

Para el diseño de los paneles DATOS y FILTRO LC se han utilizado nuevamente bloques *Static Text* y *Edit Text*; sin embargo, para el panel MODULACIÓN se ha configurado un bloque *Button Group* formado por elementos *Radio Button*, donde se listan las diferentes opciones de modulación pero únicamente una de ellas podrá estar habilitada (Figura 59).

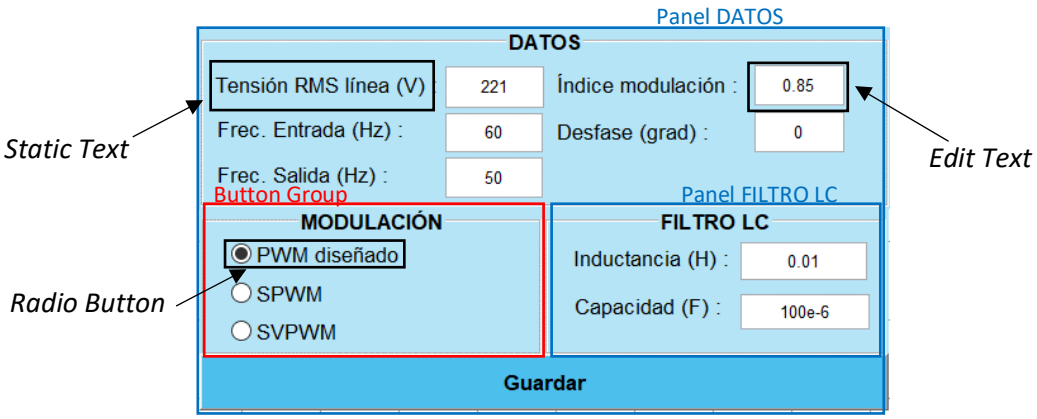


Figura 59. Bloque para la configuración de parámetros del variador de frecuencia diseñado en Simulink.

A la hora de configurar los parámetros de los diferentes bloques, para los *statis text* se ha configurado únicamente la variable *string*, en los *edit text* se han configurado las variables *string* y *tag*, y finalmente, en los bloques *radio button* (Figura 60) se han configurado las variables *string, tag* y *value*. Con esta última variable lo que se ha conseguido es configurar el valor con el que va a iniciar el bloque, de esta forma, únicamente uno de ellos va a comenzar con valor 1 (activado) y el resto a 0 (desactivado). Al estar insertados en un panel *button group* se actualizarán automáticamente de forma que, cuando se seleccione una opción, la anterior se desactive sola. Este tipo de panel asegura que solo una de las opciones pueda estar activada.

En este caso, se ha configurado como opción por defecto, la modulación PWM simple.

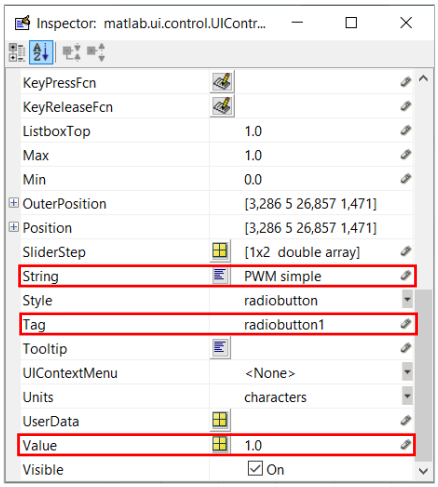


Figura 60. Parámetros a configurar en los bloques *Radio Button*.

Muestreo de resultados

Los resultados del modelo se visualizarán mediante gráficos. Para ello se han creado dos ventanas gráficas donde se podrán visualizar las diferentes señales almacenadas en el *Workspace* desde el modelo de Simulink, como ya se explicó en el desarrollo del documento.

Junto a las ventanas gráficas *axes* (Figura 61) se han añadido diversos bloques que ayudarán a la visualización de los resultados. Estos bloques se componen de un menú desplegable *pop-up menu* con el que seleccionar la señal a graficar, un *slider* para realizar zoom en el eje horizontal de la señal y otro *slider* con el que poder señalar verticalmente un punto de la señal y visualizar el instante de tiempo en el que se encuentra. Para la visualización del tiempo se utilizará un bloque *edit text*, explicados anteriormente. También aparecen bloques *static text*, usados como etiquetas.

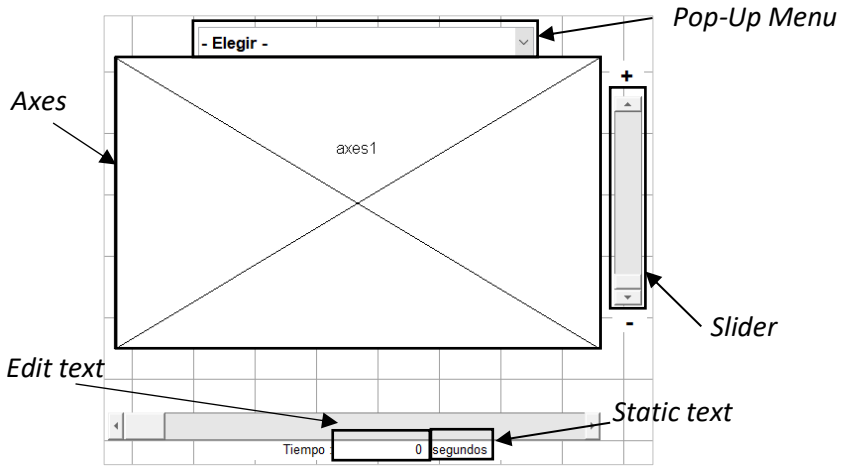


Figura 61. Diseño bloques gráficos y otros elementos para mostrar los resultados obtenidos.

Para el ajuste de parámetros de los diferentes bloques, en los bloques *static text*, bastará con configurar el nombre del bloque en la variable *string*. Tanto en los bloques *axes* como en los *pop up menú*, será necesario configurar las variables *string* y *tag.* Y, por último, en los bloques *slider* (Figura 62) se deberán configurar los parámetros *string, tag, value, slider step, min* y *max.*

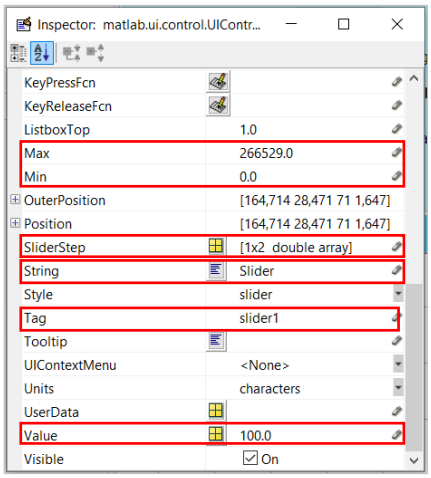


Figura 62. Parámetros a configurar en un bloque *Slider*.

Los valores mínimo y máximo configuran el rango de valores que abarca el *slider,* así como el slider step establece el valor de paso del slider cada vez que se pulsan las flechas de avance/retroceso o de la barra central.

Cabe destacar que todos los parámetros internos de los bloques también se pueden configurar mediante la consola de comandos de Matlab. Esta posibilidad se utilizará en aquellas ocasiones en las cuales alguno de los parámetros de los bloques dependa de cálculos internos o de resultados obtenidos en las funciones internas programadas. Todos estos casos se han comentados en la sección *2.5 Programación de funciones mediante Matlab*.

ANEXO B: CÓDIGO DE PROGRAMACIÓN DE LA INTERFAZ DE USUARIO MEDIANTE LA CONSOLA DE COMANDOS DE MATLAB

```
function varargout = aplicacion_guide(varargin)
% APLICACION_GUIDE MATLAB code for aplicacion_guide.fig
%      APLICACION_GUIDE, by itself, creates a new APLICACION_GUIDE or raises
the existing
%      singleton*.
%
%      H = APLICACION_GUIDE returns the handle to a new APLICACION_GUIDE or
the handle to
%      the existing singleton*.
%
%      APLICACION_GUIDE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in APLICACION_GUIDE.M with the given input
arguments.
%
%      APLICACION_GUIDE('Property','Value',...) creates a new APLICACION_GUIDE
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before aplicacion_guide_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to aplicacion_guide_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help aplicacion_guide

% Last Modified by GUIDE v2.5 14-Nov-2020 10:14:26

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @aplicacion_guide_OpeningFcn, ...
                   'gui_OutputFcn',  @aplicacion_guide_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before aplicacion_guide is made visible.
function aplicacion_guide_OpeningFcn(hObject, eventdata, handles, varargin)
load_system('convertidor_PWM');
find_system('Name','convertidor_PWM');
open_system('convertidor_PWM');
A= imread('Fondo.jpg');
image(handles.axes1,A);
image(handles.axes2,A);
image(handles.axes7,A);
image(handles.axes8,A);
axis(handles.axes8,'off');
```

```
%Variables "globales"
handles.output = hObject;
handles.aux = 0;
handles.aux1 = 0;
handles.aux2 = 0;
guidata(hObject, handles);


% --- Outputs from this function are returned to the command line.
function varargout = aplicacion_guide_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
set_param ('convertidor_PWM','SimulationCommand','start');
A= imread('Fondo.jpg');
image(handles.axes1,A);
image(handles.axes2,A);
image(handles.axes7,A);
image(handles.axes8,A);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
set_param ('convertidor_PWM','SimulationCommand','stop');


function edit1_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% Se usa en otra función

 % --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit3_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
%CONFIGURACION PARAMETROS MOTOR
P_n = get(handles.edit12,'String');
V_n = get(handles.edit13,'String');
F_n = get(handles.edit14,'String');
Nominal_parameters = strcat("[",P_n," ",V_n," ",F_n,"]");
set_param('convertidor_PWM/Asynchronous Machine SI
Units','NominalParameters',Nominal_parameters);
R_e = get(handles.edit15,'String');
L_e = get(handles.edit16,'String');
Estator_parameters = strcat("[",R_e," ",L_e,"]");
set_param('convertidor_PWM/Asynchronous Machine SI
Units','Stator',Estator_parameters);
R_r = get(handles.edit17,'String');
L_r = get(handles.edit18,'String');
Rotor_parameters = strcat("[",R_r," ",L_r,"]");
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
if Rotor == 3
    R_r2 = get(handles.edit23,'String');
    L_r2 = get(handles.edit24,'String');
    Rotor2_parameters = strcat("[",R_r2," ",L_r2,"]");
    set_param('convertidor_PWM/Asynchronous Machine SI
Units','Cage1',Rotor_parameters);
    set_param('convertidor_PWM/Asynchronous Machine SI
Units','Cage2',Rotor2_parameters);
else
    set_param('convertidor_PWM/Asynchronous Machine SI
Units','Rotor',Rotor_parameters);
end
L_m = get(handles.edit19,'String');
set_param('convertidor_PWM/Asynchronous Machine SI Units','Lm',L_m);
I = get(handles.edit20,'String');
F = get(handles.edit21,'String');
P = get(handles.edit22,'String');
mecanical_parameters = strcat("[",I," ",F," ",P,"]");
AUX = {get(handles.popupmenu5,'Value')};
Entrada = cell2mat (AUX);
if Entrada == 2
    set_param('convertidor_PWM/Asynchronous Machine SI Units','PolePairs',P);
else
    set_param('convertidor_PWM/Asynchronous Machine SI
Units','Mechanical',mecanical_parameters);
end
constante = get(handles.edit28,'String');
set_param('convertidor_PWM/Constant','Value',constante);


function edit4_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit5_Callback(hObject, eventdata, handles)
% Se usa en otra función
```

```
% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit6_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
%CONFIGURACION DE ROTOR
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
   if Rotor == 1
       R = 'Wound';
       W = 'on';
       S = 'off';
       D = 'off';
   elseif Rotor == 2
      R = 'Squirrel-cage';
      W = 'off';
      S = 'on';
      D = 'off';
    else
       R = 'Double Squirrel-cage';
       W = 'on';
       S = 'off';
       D = 'on';
    end
set_param('convertidor_PWM/Asynchronous Machine SI Units','RotorType', R);
%Parametros Squirrel-cage
set(handles.text15,'Enable',S);
set(handles.popupmenu4,'Enable',S);
%Parametros Double Squirrel-cage
set(handles.text29,'Enable',D);set(handles.text30,'Enable',D);
set(handles.edit23,'Enable',D);set(handles.edit24,'Enable',D);
%Parametros Wound y Double Squirrel-cage
set(handles.text18,'Enable',W);set(handles.text19,'Enable',W);set(handles.text
20,'Enable',W);
set(handles.text21,'Enable',W);set(handles.text22,'Enable',W);set(handles.text
23,'Enable',W);
set(handles.text24,'Enable',W);set(handles.text25,'Enable',W);set(handles.text
26,'Enable',W);
set(handles.text27,'Enable',W);set(handles.text28,'Enable',W);set(handles.edit
12,'Enable',W);
set(handles.edit13,'Enable',W);set(handles.edit14,'Enable',W);set(handles.edit
15,'Enable',W);
set(handles.edit16,'Enable',W);set(handles.edit17,'Enable',W);set(handles.edit
18,'Enable',W);
set(handles.edit19,'Enable',W);set(handles.edit20,'Enable',W);set(handles.edit
21,'Enable',W);
set(handles.edit22,'Enable',W);
```

```
AUX5 = {get(handles.popupmenu5,'Value')};
speed = cell2mat (AUX5);
if speed == 2
    W='off';
    set(handles.text26,'Enable',W);set(handles.text27,'Enable',W);
    set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
end


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over popupmenu1.
function popupmenu1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
RB1 = {get(handles.radiobutton1,'Value')};
RB_1 = cell2mat (RB1);
if RB_1 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value', '3');
end

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
RB2 = {get(handles.radiobutton2,'Value')};
RB_2 = cell2mat (RB2);
if RB_2 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value','2');
end

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
if RB_3 == 1
    set_param('convertidor_PWM/MODULACIONES/Constant1','Value','1');
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
AUX2 = {get(handles.popupmenu2,'Value')};
Graf = cell2mat (AUX2);
%VALORES GRAFICAS
tiempo=evalin('base','tout');
V_DC=evalin('base','V_DC');
Vabc_entrada=evalin('base','Vabc_entrada');
Vabc_s=evalin('base','Vabc_s');
Vabc_salida=evalin('base','Vabc_salida');
S=evalin('base','S');
%CURSOR
tout=evalin('base','tout');
set(handles.slider1,'Max',length(tout));
tiempo_celda = int32(get (handles.slider1,'Value'));
tiempo_valor = tout(tiempo_celda,1);
set(handles.text47,'String',tiempo_valor);
```

```
%ZOOM
zoom =double(get(handles.slider3,'Value'));
%Modulacion SVPWM
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
Vreferencia=evalin('base','Vref');
vector_angulo=evalin('base','angulo');
angulo=vector_angulo(tiempo_celda,1);

   if Graf == 2
       plot(handles.axes1,tiempo,Vabc_entrada);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[(lim(1)/zoom) (lim(2)/zoom)]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   elseif Graf == 3
       plot(handles.axes1,tiempo,V_DC);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'Continua');
   elseif Graf == 4
       plot(handles.axes1,tiempo,Vabc_s);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   else
       plot(handles.axes1,tiempo,Vabc_salida);
       lim =axis(handles.axes1);
       ylim(handles.axes1,[lim(3)-50 lim(4)+50]);
       xlim(handles.axes1,[lim(1)/zoom lim(2)/zoom]);
       xlabel(handles.axes1,'Tiempo (s)');
       ylabel(handles.axes1,'Voltaje (V)');
       legend (handles.axes1,'a','b','c');
   end

   aux1=handles.aux1;
if tiempo_valor>=(aux1+(lim(2)/zoom))
    aux1=tiempo_valor;
    handles.aux1=aux1;
    guidata(hObject,handles);
end
if tiempo_valor < (aux1+(lim(1)/zoom))
    aux1=tiempo_valor;
    handles.aux1=aux1;
    guidata(hObject,handles);
end
xlim(handles.axes1,[aux1 aux1+(lim(2)/zoom)]);
xline(handles.axes1,tiempo_valor,'--','LineWidth',1.5,'DisplayName','Cursor');



% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
 % --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
AUX3 = {get(handles.popupmenu3,'Value')};
Graf2 = cell2mat (AUX3);
%GRAFICAS
tiempo=evalin('base','tout');
Vel=evalin('base','Velocidad');
Par=evalin('base','Par');
I_estator=evalin('base','Corriente_estator');
%SLIDER
tout=evalin('base','tout');
set(handles.slider2,'Max',length(tout));
tiempo_celda = int32(get (handles.slider2,'Value'));
tiempo_valor = tout(tiempo_celda,1);
set(handles.text42,'String',tiempo_valor);
%ZOOM
zoom =double(get(handles.slider5,'Value'));

   if Graf2 == 2
       plot(handles.axes2,tiempo,Vel);
       lim =axis(handles.axes2);
       xlim(handles.axes2,[(lim(1)/zoom) (lim(2)/zoom)]);
       xlabel(handles.axes2,'Tiempo (s)');
       ylabel(handles.axes2,'Velocidad (rpm)');
       legend (handles.axes2,'Velocidad');
   elseif Graf2 == 3
       plot(handles.axes2,tiempo,Par);
       lim =axis(handles.axes2);
       xlim(handles.axes2,[(lim(1)/zoom) (lim(2)/zoom)]);
       xlabel(handles.axes2,'Tiempo (s)');
       ylabel(handles.axes2,'Par (Nm)');
       legend (handles.axes2,'Par');
   else
       plot(handles.axes2,tiempo,I_estator);
       lim =axis(handles.axes2);
       ylim(handles.axes2,[lim(3)-5 lim(4)+5]);
       xlim(handles.axes2,[(lim(1)/zoom) (lim(2)/zoom)]);
       xlabel(handles.axes2,'Tiempo (s)');
       ylabel(handles.axes2,'Corriente (A)');
       legend (handles.axes2,'a','b','c');
   end

   aux2=handles.aux2;
if tiempo_valor>=(aux2+(lim(2)/zoom))
    aux2=tiempo_valor;
    handles.aux2=aux2;
    guidata(hObject,handles);
end
if tiempo_valor < (aux2+(lim(1)/zoom))
    aux2=tiempo_valor;
    handles.aux2=aux2;
    guidata(hObject,handles);
end
xlim(handles.axes2,[aux2 aux2+(lim(2)/zoom)]);
xline(handles.axes2,tiempo_valor,'--','LineWidth',1.5,'DisplayName','Cursor');


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit7_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu4.
function popupmenu4_Callback(hObject, eventdata, handles)
Rotor_matriz = cellstr(get(handles.popupmenu4,'String'));
Numero={get(handles.popupmenu4,'Value')};
Num = cell2mat (Numero);
Rotor=string(Rotor_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','PresetModel',
Rotor);
if Num == 1
    W = 'on';
    %Parametros Wound y Double Squirrel-cage

set(handles.text18,'Enable',W);set(handles.text19,'Enable',W);set(handles.text
20,'Enable',W);

set(handles.text21,'Enable',W);set(handles.text22,'Enable',W);set(handles.text
23,'Enable',W);

set(handles.text24,'Enable',W);set(handles.text25,'Enable',W);set(handles.text
26,'Enable',W);

set(handles.text27,'Enable',W);set(handles.text28,'Enable',W);set(handles.edit
12,'Enable',W);

set(handles.edit13,'Enable',W);set(handles.edit14,'Enable',W);set(handles.edit
15,'Enable',W);

set(handles.edit16,'Enable',W);set(handles.edit17,'Enable',W);set(handles.edit
18,'Enable',W);

set(handles.edit19,'Enable',W);set(handles.edit20,'Enable',W);set(handles.edit
21,'Enable',W);
    set(handles.edit22,'Enable',W);
    AUX5 = {get(handles.popupmenu5,'Value')};
    speed = cell2mat (AUX5);
    if speed == 2
        W='off';
        set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);
    end
end


% --- Executes during object creation, after setting all properties.
function popupmenu4_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
% --- Executes on selection change in popupmenu5.
function popupmenu5_Callback(hObject, eventdata, handles)
%Almacenar valor Tipo Rotor
AUX = {get(handles.popupmenu1,'Value')};
Rotor = cell2mat (AUX);
%Alamcenar los posibles valores de entrada mecanica en un matriz de string,
%posteriormente almacenar el valor para entrar en la matriz y coger el
%string que corresponde
Mec_matriz = cellstr(get(handles.popupmenu5,'String'));
Numero={get(handles.popupmenu5,'Value')};
Num = cell2mat (Numero);
Mechanical=string(Mec_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','MechanicalLoad',
Mechanical);
if Num == 1 && Rotor ~= 2
    W="on";
else
    W ="off";
end
set(handles.text26,'Enable',W);set(handles.text27,'Enable',W);
set(handles.edit20,'Enable',W);set(handles.edit21,'Enable',W);

% --- Executes during object creation, after setting all properties.
function popupmenu5_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on selection change in popupmenu6.
function popupmenu6_Callback(hObject, eventdata, handles)
Estru_matriz = cellstr(get(handles.popupmenu6,'String'));
Numero={get(handles.popupmenu6,'Value')};
Num = cell2mat (Numero);
Estructura=string(Estru_matriz(Num,1));
set_param('convertidor_PWM/Asynchronous Machine SI Units','ReferenceFrame',
Estructura);

% --- Executes during object creation, after setting all properties.
function popupmenu6_CreateFcn(hObject, eventdata, handles)
% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit12_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit12_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit13_Callback(hObject, eventdata, handles)
% Se usa en otra función
```

```
% --- Executes during object creation, after setting all properties.
function edit13_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit14_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit15_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit16_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit16_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit17_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit17_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edit18_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit19_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit19_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit20_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit21_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit21_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit22_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
 function edit28_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit28_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit23_Callback(hObject, eventdata, handles)
% Se usa en otra función

 % --- Executes during object creation, after setting all properties.
function edit23_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit24_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function edit24_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function text29_CreateFcn(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in checkbox1.
function checkbox1_Callback(hObject, eventdata, handles)
S1 = evalin('base','S1');
S3 = evalin('base','S3');
S5 = evalin('base','S5');

% Hint: get(hObject,'Value') returns toggle state of checkbox1


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% Se usa en otra función


function slider2_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



% --- Executes on button press in pushbutton4.
```

```
function pushbutton5_Callback(hObject, eventdata, handles)
%CONFIGURACION DE VALORES
F_entrada = get(handles.edit1,'String');
F_salida = get(handles.edit2,'String');
V_rms = get(handles.edit3,'String');
ma=get(handles.edit26,'String');
desfase = get(handles.edit28,'String');
L_filtro = get(handles.edit4,'String');
C_filtro = get(handles.edit5,'String');
%Configuración valores señal entrada
set_param('convertidor_PWM/Three-Phase Source','Frequency',F_entrada);
set_param('convertidor_PWM/Three-Phase Source','Voltage',V_rms);
%configuración valores modulación
%Modulación PWM mediante puertas logicas
mad=str2double(ma);
madd= mad*(3^(1/2))/(2^(1/2));
madds=string(madd);
set_param('convertidor_PWM/Three-Phase Source1','Voltage',madds);
set_param('convertidor_PWM/Three-Phase Source1','Frequency',F_salida);
set_param('convertidor_PWM/Sawtooth Generator1','Phase',desfase);
%Modulación PWM bloque
set_param('convertidor_PWM/PWM Generator (2-Level)','Freq',F_salida);
set_param('convertidor_PWM/PWM Generator (2-Level)','m',ma);
set_param('convertidor_PWM/PWM Generator (2-Level)','Phase',desfase);
%Modulación SVPWM bloque
SVPWM_param= strcat("[",ma," ",desfase," ",F_salida,"]");
set_param('convertidor_PWM/SVPWM Generator (2-Level)','ParUref',SVPWM_param);
%Configuración Filtro LC
set_param('convertidor_PWM/Three-Phase Parallel RLC
Branch','Inductance',L_filtro);
set_param('convertidor_PWM/Three-Phase Parallel RLC
Branch1','Capacitance',C_filtro);


function edit26_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit26_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function edit29_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function edit29_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function axes1_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% Se usa en otra función


function slider1_CreateFcn(hObject, eventdata, handles)
% Hint: edit controls usually have a white background on Windows.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider5_Callback(hObject, eventdata, handles)
% Se usa en otra función


% --- Executes during object creation, after setting all properties.
function slider5_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider6_Callback(hObject, eventdata, handles)
%VALORES GRAFICAS
tiempo=evalin('base','tout');
S1=evalin('base','S1');
S2=evalin('base','S2');
S3=evalin('base','S3');
S4=evalin('base','S4');
S5=evalin('base','S5');
S6=evalin('base','S6');
%SELECCIÓN DE SEÑALES
S1button={get(handles.radiobutton4,'Value')};
S1_button=cell2mat (S1button);
S2button={get(handles.radiobutton5,'Value')};
S2_button=cell2mat (S2button);
S3button={get(handles.radiobutton6,'Value')};
S3_button=cell2mat (S3button);
S4button={get(handles.radiobutton7,'Value')};
S4_button=cell2mat (S4button);
S5button={get(handles.radiobutton8,'Value')};
S5_button=cell2mat (S5button);
S6button={get(handles.radiobutton9,'Value')};
S6_button=cell2mat (S6button);
%CURSOR
tout=evalin('base','tout');
set(handles.slider7,'Max',length(tout));
tiempo_celda = int32(get (handles.slider7,'Value'));
tiempo_valor = tout(tiempo_celda,1);
```

```
set(handles.text55,'String',tiempo_valor);
%ZOOM
zoom =double(get(handles.slider6,'Value'));
%Modulacion SVPWM
RB3 = {get(handles.radiobutton3,'Value')};
RB_3 = cell2mat (RB3);
Vreferencia=evalin('base','Vref');
vector_angulo=evalin('base','angulo');
angulo=vector_angulo(tiempo_celda,1);
Vref=Vreferencia(tiempo_celda,1);
if RB_3 ==1
   figure(1)
   x=cos(angulo)*Vref;
   y=sin(angulo)*Vref;
   c=compass(x,y);
   c.LineWidth=2;
end
%Ploteo de graficas
a = plot(handles.axes7,tiempo,S1,'DisplayName','S1');
legend (handles.axes7);
hold (handles.axes7,'on');
if S1_button==0
    delete(a);
end

b = plot(handles.axes7,tiempo,S2,'DisplayName','S2');
if S2_button==0
    delete(b);
end

c = plot(handles.axes7,tiempo,S3,'DisplayName','S3');
if S3_button==0
    delete(c);
end

d = plot(handles.axes7,tiempo,S4,'DisplayName','S4');
if S4_button==0
    delete(d);
end

e = plot(handles.axes7,tiempo,S5,'DisplayName','S5');
if S5_button==0
    delete(e);
end

f = plot(handles.axes7,tiempo,S6,'DisplayName','S6');
if S6_button==0
    delete(f);
end
hold (handles.axes7,'off');

lim =axis(handles.axes7);
ylim(handles.axes7,[lim(3)-0.1 lim(4)+0.1]);
xlim(handles.axes7,[lim(1)/zoom lim(2)/zoom]);
xlabel(handles.axes7,'Tiempo (s)');
ylabel(handles.axes7,'Pulso');

aux=handles.aux;
if tiempo_valor>=(aux+(lim(2)/zoom))
    aux=tiempo_valor;
    handles.aux=aux;
    guidata(hObject,handles);
end
if tiempo_valor < (aux+(lim(1)/zoom))
    aux=tiempo_valor;
    handles.aux=aux;
    guidata(hObject,handles);
end
```

```
xlim(handles.axes7,[aux aux+(lim(2)/zoom)]);
xline(handles.axes7,tiempo_valor,'--','LineWidth',1.5,'DisplayName','Cursor');

%Dibujo de esquemas
S1_on = S1(tiempo_celda,1);
S3_on = S3(tiempo_celda,1);
S5_on = S5(tiempo_celda,1);

if S1_on==1 && S3_on==1 && S5_on==1
    Ima=imread('111.jpg');
    image(handles.axes8,Ima);
elseif S1_on==1 && S3_on==1 && S5_on==0
    Ima=imread('110.jpg');
    image(handles.axes8,Ima);
elseif S1_on==1 && S3_on==0 && S5_on==0
    Ima=imread('100.jpg');
    image(handles.axes8,Ima);
elseif S1_on==1 && S3_on==0 && S5_on==1
    Ima=imread('101.jpg');
    image(handles.axes8,Ima);
elseif S1_on==0 && S3_on==0 && S5_on==1
    Ima=imread('001.jpg');
    image(handles.axes8,Ima);
elseif S1_on==0 && S3_on==1 && S5_on==1
    Ima=imread('011.jpg');
    image(handles.axes8,Ima);
elseif S1_on==0 && S3_on==1 && S5_on==0
    Ima=imread('010.jpg');
    image(handles.axes8,Ima);
elseif S1_on==0 && S3_on==0 && S5_on==0
    Ima=imread('000.jpg');
    image(handles.axes8,Ima);
end
axis(handles.axes8,'off');


% --- Executes during object creation, after setting all properties.
function slider6_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
% Se usa en otra función
```

```
 % --- Executes on slider movement.
function slider7_Callback(hObject, eventdata, handles)
% Se usa en otra función

% --- Executes during object creation, after setting all properties.
function slider7_CreateFcn(hObject, eventdata, handles)
% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```