Universidad Carlos III de Madrid

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

Departamento de Ingeniería Eléctrica

TRABAJO FIN DE GRADO

Grado en Ingeniería en Tecnologías Industriales

Control mediante Arduino del movimiento de una antena directiva con un servomotor

Autor:

Sandra Pérez Prieto

Tutor:

Dr. D. Guillermo Robles Muñoz

Leganés, Octubre 2015

Título: Control mediante Arduino del movimiento de una antena directiva con un servomotor

Autor: Sandra Pérez Prieto

Tutor: Dr. D. Guillermo Robles Muñoz

EL TRIBUNAL

Presidente: Joaquín Eloy-García Carrasco

Vocal: Concepción Alicia Monje Micharet

Secretario: Daniel Serrano Jiménez

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 8 de Octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Resumen

El control y mantenimiento de elementos eléctricos que requieren de un contacto directo, pueden poner en riesgo la vida de las personas dedicadas a ello. Por este motivo, se ha hecho necesario diseñar sistemas que permitan revisar el estado de los equipos y que puedan ser controlados a distancia por el usuario. En este caso, la aplicación se basa en controlar de forma remota la posición de una antena cuya finalidad es localizar fuentes de descargas parciales en los aislantes, y diseñar el sistema de adquisición de datos que recoge la información del fenómeno físico.

Para generar el movimiento automático se empleará una placa *Arduino* que controlará la posición del servomotor, encargado de situar el sistema en la dirección indicada, y para captar las señales emitidas por las descargas parciales se empleará una antena directiva que enviará los datos a una tarjeta de adquisición.

Este trabajo constituye la base del proyecto "*Localización de eventos en radiofrecuencia mediante una antena de alta directividad"*, complementario a este. En él, se analizarán las señales proporcionadas por el sistema de adquisición en cada una de las posiciones recorridas por el sistema, y se realizarán varios estudios que determinarán la localización de la fuente de descargas parciales.

Abstract

The control and maintenance of electrical elements that require direct contact can affect the safety of the staff engaged in it. For this reason, the design of systems that allow to check the conditions of the equipment and that can be controlled remotely by the user has become necessary. In this project, the application is based on the remote control of the position of an antenna whose purpose is to locate sources of partial discharges inside electrical insulators, and to design a data acquisition system that acquire information from the physical phenomenon.

In order to generate the automatic movement, an *Arduino* board will be used to control the position of a servomotor, which is the responsible for locating the system in the right direction; and in order to capture the signals emitted by the partial discharges, a directional antenna will send data to an acquisition card.

This work is the basis of the project "Location of events in radiofrequency using a high directivity antenna", complementary to this. In it, the signals provided by the acquisition system will be analyzed in each of the positions covered by the system, and several studies will be tested to determine the location of the source of the partial discharges.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

1. Introducción

1.1. Motivación

La idea de este proyecto surge de un estudio realizado previamente en el Laboratorio de Alta Tensión de la Universidad Carlos III de Madrid. El estudio consistía en la integración de una tarjeta NI PCI-5152 con una placa *Arduino Mega 2560*, todo ello conectado a LabVIEW, con el fin de localizar descargas parciales a través de la adquisición de datos mediante una antena móvil.

Las descargas parciales se tratan de una avalancha de electrones que tiene lugar en las vacuolas de aire de aislantes sometidos a elevada tensión, y su presencia puede llegar a deteriorar el medio aislante. Durante su aparición se produce la erosión de la superficie del dieléctrico y se generan compuestos químicos que provocan su degradación química [1]. La medida y localización de las descargas parciales proporciona información acerca del estado de los aislantes y en la actualidad, se tiene especial interés en ello para mantener la fiabilidad en los equipos y asegurar la protección de las personas. Para su detección, existen sensores de diversos tipos. Sin embargo, en esta aplicación interesan únicamente aquellos que miden la señal eléctrica, que se dividen en sensores inductivos (ancho de banda comprendido entre 0,1-100 MHz) y sensores de radiofrecuencia (ancho de banda comprendido entre 50 MHz-25 GHz). En este caso se seleccionó como dispositivo de detección una antena monopolo, perteneciente al segundo grupo.

El estudio previo utilizaba la herramienta LabVIEW para la integración de ambas tarjetas bajo el mismo entorno de programación y su aplicación se basaba en una interfaz gráfica de usuario diseñada en dicha herramienta. A través de la placa *Arduino*, se controlaba la posición de dos servomotores que generaban el movimiento en precesión y nutación de la antena, y esta última, era la encargada de enviar la información que recibía a la tarjeta de adquisición. Ambas tarjetas estaban conectadas a una CPU con un procesador de doble núcleo y memoria RAM de 2 GB [2].

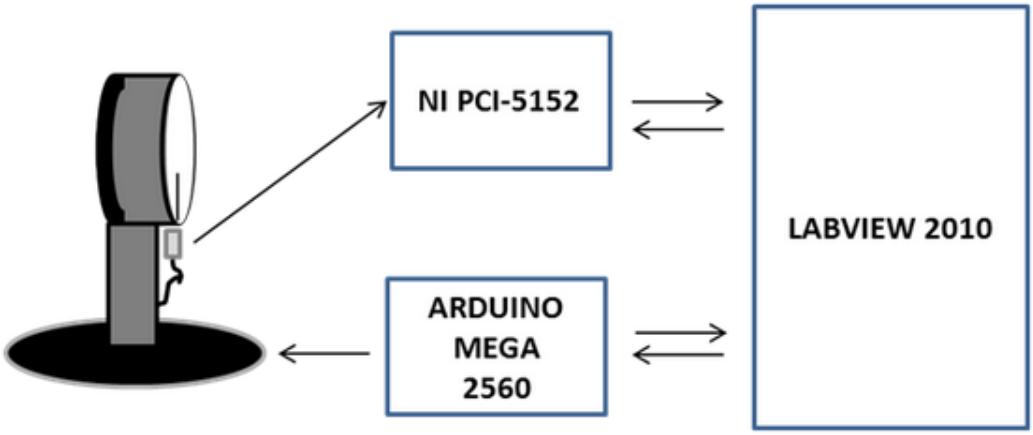


Figura 1. Diagrama esquemático del sistema de adquisición [2]

Al ejecutar el programa, la tarjeta *Arduino* daba la orden a los servomotores de iniciar el barrido en sus correspondientes ejes. A medida que se iba moviendo la antena, la tarjeta NI PCI-5152 iba adquiriendo datos y el programa iba analizando las señales que recogía. En el momento en que detectaba concentraciones de energía superiores a las producidas por el ruido del laboratorio, se almacenaban las ondas percibidas en esa posición y se visualizaban en la interfaz gráfica. Al finalizar el barrido, el sistema se situaba en la posición en la que se había localizado la mayor concentración de energía y se realizaba un segundo barrido más preciso alrededor de la zona de interés, localizando así con mayor exactitud la fuente de descargas parciales.

En la interfaz gráfica (*Figura 2*), se podía visualizar cuál era la posición en cada instante de cada uno de los servomotores, la transformada de Fourier de las señales captadas y la posición en la que se había localizado el fenómeno. También, permitía al usuario posicionar la antena en un punto exacto y elegir la velocidad del barrido.

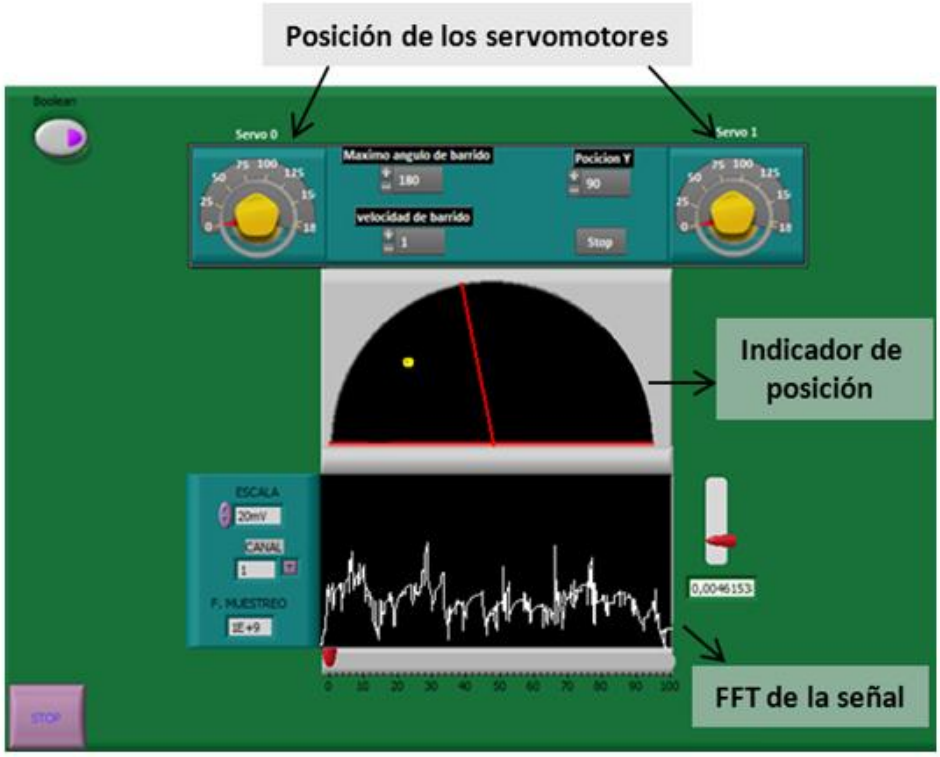


Figura 2. Pantalla gráfica del panel principal del software [2]

Pese a que ya se había realizado este estudio, no se dio por concluida la localización de descargas parciales mediante una antena móvil, y es aquí, donde surge la idea de este proyecto.

Como alternativa, y con el objetivo de mejorar las prestaciones que ofrece el sistema en su conjunto, se propone un nuevo algoritmo de rastreo y localización de descargas parciales basado en la herramienta Matlab, en la placa *Arduino Uno* y en nuevos métodos de estudio de los datos recogidos.

Se plantea la idea de localizar el fenómeno a partir de los resultados de diversos estudios y no sólo en función de la mayor concentración de energía detectada. Teniendo en mente el planteamiento de realizar diversos análisis de las señales detectadas, se ha optado por realizar el programa en el entorno Matlab. Esta herramienta, en comparación con LabVIEW, es más versátil, no presenta limitaciones en cuanto al procesamiento de señales y permite realizar cualquier análisis con los datos adquiridos.

En cuanto al modelo de la tarjeta *Arduino*, en esta ocasión se ha optado por la plataforma *Arduino Uno*, que es la plataforma estándar. Presenta menos pines y aporta menos prestaciones en memoria que la placa *Arduino Mega 2560*, pero es más barata y presenta las características necesarias para nuestra aplicación.

En resumen, con el fin de mejorar las prestaciones del sistema ya probado en el laboratorio, se ha planteado la opción de desarrollar una nueva aplicación que cumple con los mismos objetivos pero que difiere en las herramientas con las que trabaja y en la manera de analizar los datos.

Este proyecto se va a basar en generar el movimiento automático de la antena en una dirección (si se llevase a cabo con éxito se podría ampliar el sistema incorporando otro servomotor que le proporcionase dos grados de libertad) y en el desarrollo de un sistema de adquisición, ambos controlados por una interfaz gráfica a través de la cual el usuario indicará las órdenes que se deben ejecutar.

El análisis de las señales adquiridas y la localización de las descargas parciales forman parte de un proyecto complementario realizado por una compañera del Grado en Ingeniería en Tecnologías Industriales, que parte de este como base para su desarrollo.

1.2. Objetivos del trabajo

El objetivo principal de este proyecto se basa en el control mediante *Arduino* del movimiento de una antena de alta directividad con un servomotor, destinada a la localización de descargas parciales. Para alcanzar dicho propósito, se han planteado los siguientes objetivos:

- Comprender el funcionamiento del microcontrolador *Arduino*.
- Comprender y estudiar la herramienta GUIDE de Matlab.
- Establecer la comunicación entre la tarjeta de adquisición de datos y Matlab.
- Establecer la comunicación entre el servomotor y Matlab.
- Diseñar una interfaz gráfica que integre un sistema de adquisición de datos y controle el movimiento automático del servomotor.
- Construir una estructura en la que esté integrado el servomotor y sobre la cual se posicione la antena.

1.3. Estructura del documento

La memoria de este proyecto se ha estructurado en varios capítulos. A continuación, se expone una breve descripción de los temas tratados en cada uno de ellos.

- **Capítulo 1. Introducción.** En este capítulo se expone la motivación y los objetivos que se han planteado para llevar a cabo el proyecto.
- **Capítulo 2. Estado del arte.** En este capítulo se describen todas las herramientas empleadas. En primer lugar, se define brevemente qué es un sistema electrónico y se introduce el tema de la adquisición de datos. Se explica en qué consiste y los elementos por los que está compuesto. Dentro de este

apartado, también se citan posibles tarjetas de adquisición válidas para nuestra aplicación y se especifican las características de la tarjeta elegida (tarjeta de adquisición NI PCI-5152). A continuación, se habla en términos generales de la placa *Arduino*, se profundiza en el modelo *Arduino Uno* y se indica el paquete de apoyo necesario para establecer la comunicación entre Matlab y la placa *Arduino*. Posteriormente, se explica el funcionamiento de los servomotores y se exponen las características del servo Hitec HS-422. Para finalizar, se realiza una breve descripción de los entornos de programación LabVIEW y Matlab, y se exponen las razones por las que se ha descartado uno de ellos.

- **Capítulo 3. Entorno digital del proyecto.** A lo largo de este capítulo se detallan las ideas básicas que permiten crear una interfaz gráfica de usuario mediante la herramienta GUIDE, y se explican únicamente las funciones implementadas en el código de nuestro programa.
- **Capítulo 4. Comunicación con la tarjeta NI PCI-5152.** En este capítulo se explican detalladamente los pasos a seguir para establecer la comunicación entre la tarjeta de adquisición y el entorno Matlab, y las configuraciones que es necesario llevar a cabo una vez establecida la conexión. También se especifica cómo se realiza la adquisición de datos y cómo se simula el funcionamiento de la tarjeta en el caso de no disponer de ella físicamente.
- **Capítulo 5. Movimiento automático de la antena.** En este capítulo se detalla el proceso de control del movimiento de la antena.
- **Capítulo 6. Montaje experimental** A lo largo de este capítulo se explica el proceso de montaje de todo el sistema.
- **Capítulo 7. Interfaz gráfica.** En este capítulo se explica en qué consiste la interfaz gráfica diseñada para este trabajo y se especifica cuál es la finalidad de cada uno de los elementos que la componen.
- **Capítulo 8. Conclusiones y trabajos futuros.** Este capítulo resume las conclusiones generales que se han extraído en base a los resultados obtenidos, e incluye un apartado de posibles trabajos futuros que serían interesantes desarrollar.
- **Capítulo 9. Presupuesto.** En este capítulo se desglosan todos los costes que intervienen en el desarrollo del proyecto y se da una estimación del presupuesto total.
- **Capítulo 10. Planificación del proyecto.** En este capítulo se indica el tiempo dedicado a cada una de las tareas en las que se ha estructurado la investigación.

Al final de la memoria hay un capítulo de *Anexos* con las hojas de características de la tarjeta de adquisición NI PCI-5152 y del servomotor Hitec HS-422, y el código completo implementado en Matlab, que contiene también las líneas de código del proyecto complementario a este denominado "*Localización de descargas parciales en*

radiofrecuencia mediante una antena de alta directividad", ya que ambos se han elaborado simultáneamente y bajo el mismo archivo *.m*.

Capítulo 2

2. Estado del arte

2.1. Sistema electrónico

El funcionamiento de nuestra aplicación se basa en el concepto de sistema electrónico. Por este motivo, se explica brevemente en qué consiste.

Un sistema electrónico está compuesto por sensores, un circuito encargado del procesamiento y control, actuadores y una fuente de alimentación.

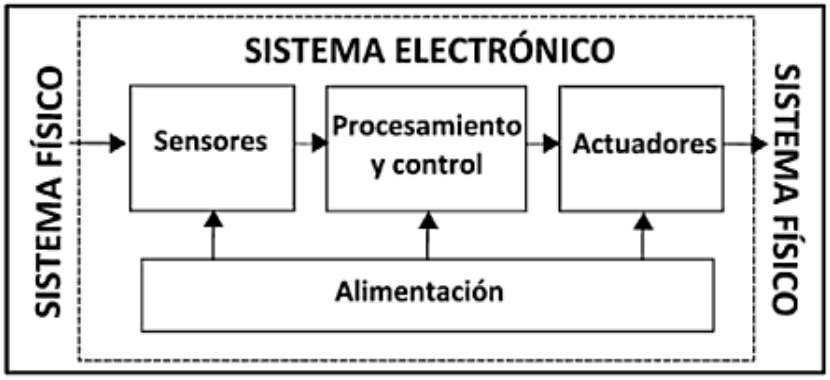


Figura 3. Sistema electrónico [3]

Los sensores obtienen información de un sistema físico, y la convierten en una señal eléctrica que es procesada por los circuitos internos. El procesamiento de esta señal depende del diseño de los componentes hardware del sistema y del programa que se haya definido en dicho hardware. El programa contiene las instrucciones que se ejecutan para el procesamiento de la señal.

Una vez procesada la señal eléctrica, los actuadores la transforman en energía, y esta, actúa directamente sobre el mundo físico.

Todos los componentes que constituyen el sistema electrónico necesitan recibir energía para llevar a cabo sus funciones, y esta energía proviene de una fuente de alimentación.

2.2. Adquisición de datos

2.2.1. Sistemas de adquisición de datos (SAD)

En la actualidad, el avance y continuo desarrollo de la electrónica está integrando la automatización en diversas áreas de nuestra vida cotidiana.

En el ámbito industrial y en estudios ingenieriles, parte del interés de su automatización se centra en el control y supervisión de fenómenos o procesos. Los sistemas de adquisición de datos (SAD) constituyen el sistema electrónico que permite la medición de las variables físicas a analizar.

En la *Figura 4* se muestran los elementos que componen los SAD, y a continuación, se explica el proceso de adquisición de datos.

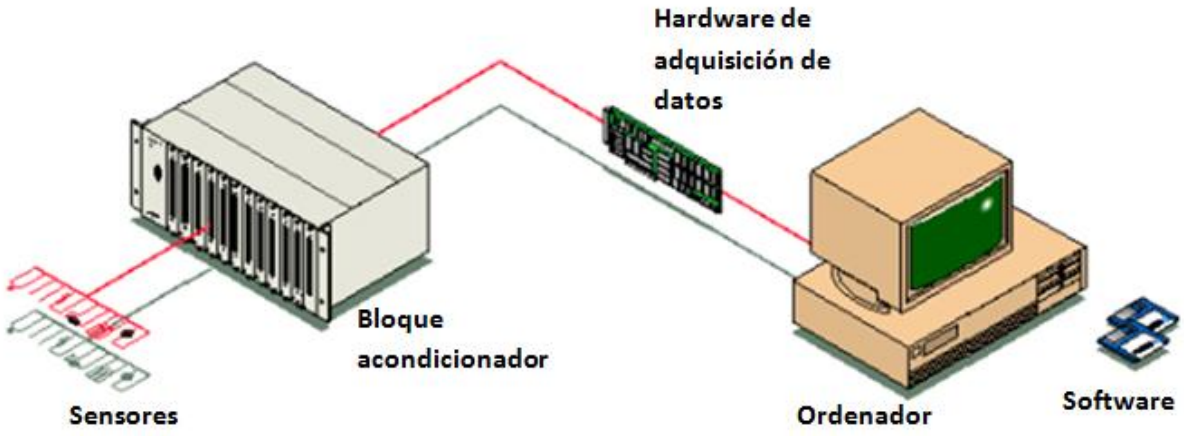


Figura 4. Esquema de los sistemas de adquisición de datos. Adaptado de [4]

El proceso de adquisición comienza con los sensores, que son los encargados de recoger muestras del fenómeno físico y convertirlas en tensiones eléctricas. La información recogida es analógica y no se puede trabajar directamente con ella, para que pueda ser procesada por un ordenador o un dispositivo electrónico, se convierten los datos analógicos a formato digital mediante el hardware de adquisición. La señal recibida por el hardware debe ser compatible con los rangos de medida del mismo. Por lo general, los niveles de tensión a la salida de los sensores no cumplen esta condición, y por ello, entre medias de ambos elementos, se sitúa un bloque acondicionador, aunque hay veces, que el acondicionamiento se lleva a cabo en la tarjeta de adquisición [5]. Las funciones de este bloque son [4]:

- ***Amplificar.*** Se aumenta el nivel de tensión de la señal.
- ***Filtrar.*** Se elimina ruido o parte de la señal que no interesa.
- ***Aislar.*** El aislamiento tiene como fin dos objetivos: evitar que el operario y los equipos resulten dañados por la señal, y proteger la onda de interés de otros fenómenos que le puedan influir y modificar.
- ***Multiplexar***. Un multiplexor consiste en un circuito combinacional que a través de unas entradas de control selecciona una de todas las señales de entrada que le llegan y la sitúa a la salida. Se emplea cuando se tienen muchas señales, cuando el número de señales es superior al número de entradas que posee la tarjeta de adquisición, o simplemente para medir señales en diferentes canales.

En nuestro caso, el acondicionamiento se basaría en una amplificación de la señal, ya que la señal de interés tiene niveles bajos de tensión, y en un filtrado para eliminar el ruido. Sin embargo, como nos interesa analizar exactamente la señal que se recoge y no nos interesa perder información, no se ha realizado ningún acondicionamiento.

Una vez digitalizada la señal, se establece la comunicación de las tarjetas o instrumentos de adquisición con un ordenador mediante drivers, y se transforma la información en datos a través de un software. Por último, con la ayuda del PC, se visualizan los valores y se comunican al usuario.

En la actualidad, existen tres configuraciones de sistemas de adquisición de datos y dependiendo de los requisitos de la recogida de señales, es recomendable emplear una u otra [6] y [7]. En la *Figura 5* se muestran las configuraciones de los SAD.

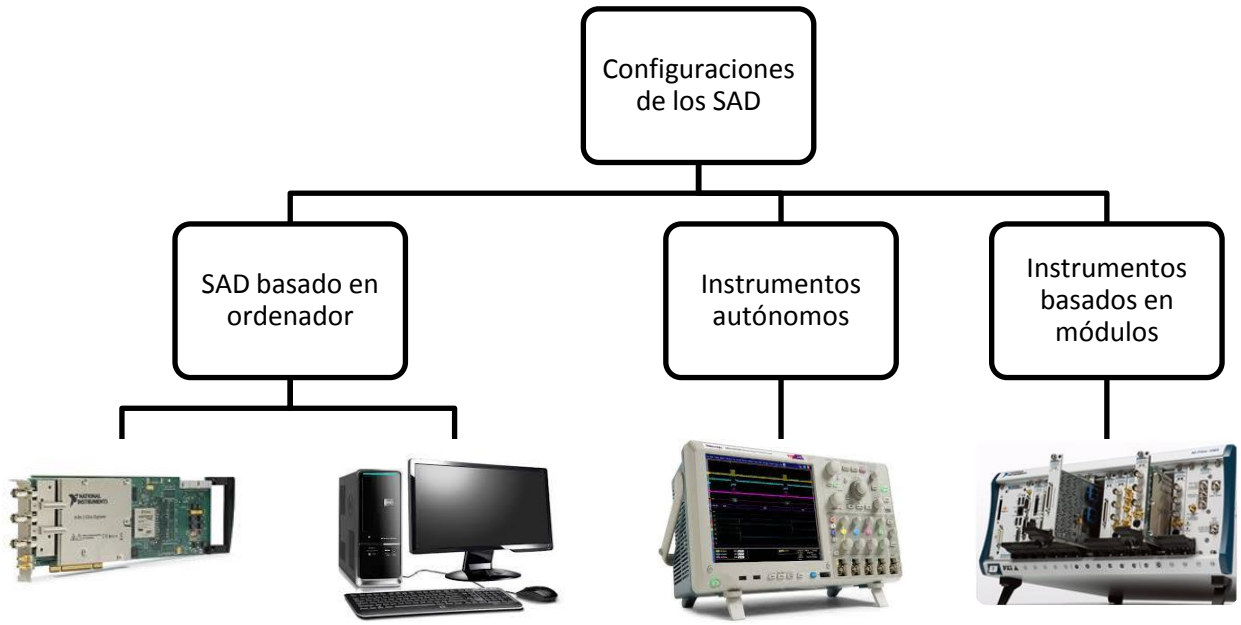


Figura 5. Configuraciones de los sistemas de adquisición de datos (SAD)

Los SAD basados en ordenador están compuestos por una tarjeta de adquisición, que adquiere las señales enviadas por los sensores, y un ordenador, que se encarga del procesamiento, almacenamiento y visualización de los datos. Esta configuración es la más recomendable para aplicaciones con pocas exigencias en las que no se requieren altas velocidades ni un excesivo número de señales de recogida, y es la más barata de las tres. No está preparada para trabajar en entornos industriales exigentes por razones como su baja refrigeración y protección frente a interferencias, o por la limitación de las ranuras de expansión. Pese a ello, presenta una elevada capacidad de visualización y de cálculo, y dispone de potentes herramientas de programación.

La conexión entre tarjeta y ordenador se establece mediante slots internos: bus PCIe (Peripheral Component Interconnect Express) o bus PXI (PCI eXtension for Instrumentation). El primero de ellos está destinado para todo tipo de aplicaciones y el segundo, sólo para instrumentación. La conexión también se puede establecer a través de redes, ya sea por conexión LAN o Wireless.

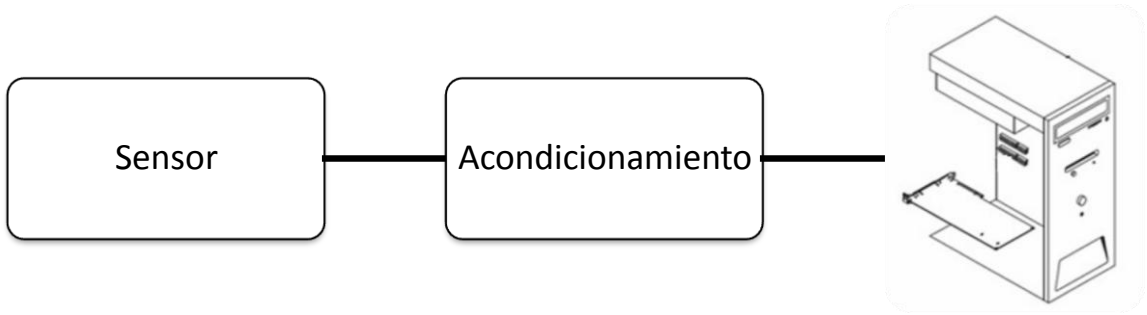


Figura 6. SAD basado en ordenador con bus PCIe

Las configuraciones basadas en instrumentos autónomos presentan un mayor coste, pero a cambio, ofrecen mejores prestaciones: mayor protección frente a ruidos y mayor velocidad. Se emplean en entornos industriales en los que se requiere adquirir una gran cantidad de valores de manera precisa y sin error.

Estos instrumentos están diseñados de tal manera que existen dos posibles formas de enviarles las instrucciones que deben seguir. Por un lado, el usuario puede indicar las órdenes a través del panel frontal del que disponen, y por otro lado, mediante software. Para esta última opción, se establece una conexión a un ordenador a través del bus GPIB (General Purpose Bus Interface), el puerto serie o el bus USB, e incluso, se puede realizar la comunicación a través de la red (comunicación Ethernet). Es en dicho ordenador en el que se escriben las instrucciones que posteriormente se envían al instrumento, y además, se puede emplear como dispositivo de cálculo y visualización de los datos.

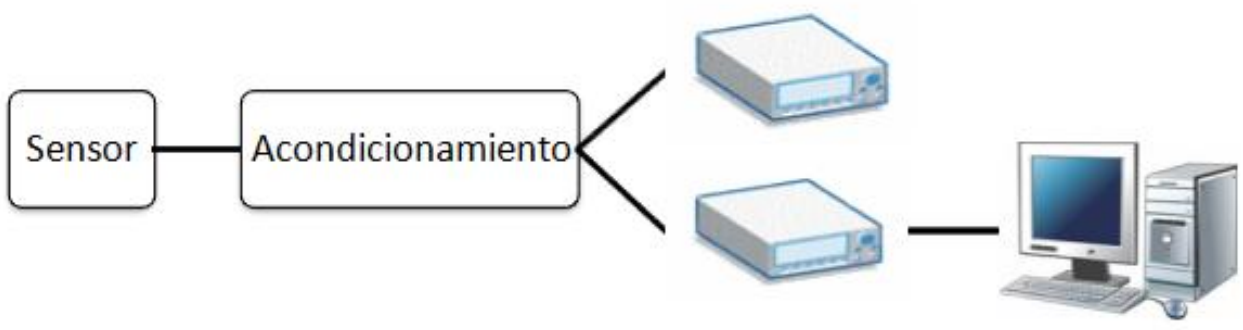


Figura 7. SAD basado en instrumentos autónomos

En cuanto a la última configuración, el sistema de adquisición se construye a partir de la inserción de diversos módulos en buses locales especializados de la estructura del instrumento. Estos módulos se caracterizan por ser estándares, lo que permite trabajar con elementos de distintos fabricantes simultáneamente sin que exista la posibilidad de error por incompatibilidad.

Al igual que los instrumentos autónomos, se tratan de sistemas con elevada potencia, flexibilidad y robustez, pero presentan la ventaja de tener un coste inferior.

Como diferencia se podría destacar que no disponen de un panel frontal, lo que limita las posibles maneras de enviar las instrucciones. Estas se envían desde un ordenador, que puede constituir uno de los módulos de la estructura, o estar conectado al instrumento a través de un bus.

En esta configuración, nos encontramos dos alternativas de estructura del instrumento, denominadas PXI y VXI (VME[1] eXtension for Instrumentation), diferenciadas por el tipo de puertos que emplean: PCI y VME, respectivamente.

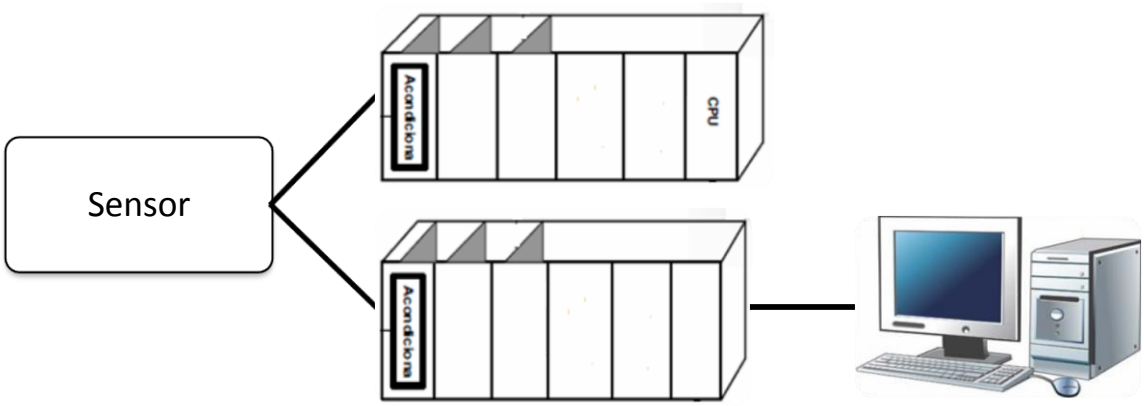


Figura 8. SAD basados en instrumentos modulares

2.2.2. Modelos de tarjetas de adquisición

El sistema de adquisición de datos que se emplea en este proyecto es el conjunto: ordenador-tarjeta de adquisición.

En el mercado existen diversos fabricantes de tarjetas y multitud de modelos de cada uno de ellos. A la hora de seleccionar la tarjeta, hay que tener en cuenta determinados parámetros, siendo algunos de los más importantes los que se especifican a continuación [4], [7] y [8]:

[1] Las siglas VME corresponden a Versa Module Europa.

- ***Número de entradas.*** Es el número de canales de entrada de los que dispone.
- ***Frecuencia de muestreo.*** Es la máxima velocidad a la que la tarjeta es capaz de tomar muestras, y se define como el cociente entre número de muestras por segundo.
- ***Resolución.*** Es el número de bits utilizados para representar una señal analógica. Cuanto mayor sea la resolución, mayor es el número de divisiones que se realizan a la señal, y por tanto, más precisa es la señal discretizada. Si, por ejemplo, la resolución es de 8 bits, se tendrán 2^8 divisiones.
- ***Tipo de conexión:*** PCI, USB, PXI, etc.
- ***Margen de entrada.*** Es el rango de tensión de entrada permitido por la tarjeta. Puede ser unipolar (sólo admite tensiones positivas) o bipolar (admite tanto tensiones positivas como negativas).
- ***Ancho de banda.*** Es el rango de frecuencias en el que la tarjeta trabaja correctamente. La tarjeta detecta aquellas señales cuya frecuencia esté comprendida en este rango.

El parámetro más crítico es la frecuencia de muestreo. Cuanto mayor es este parámetro, mayor es el número de muestras que se recogen por segundo, y por tanto, mejor es la calidad de definición de la señal en el tiempo. Puesto que la tarjeta de la que se dispone en el laboratorio sólo permite trabajar a una velocidad máxima de muestreo de 1 GS/s, a la hora de buscar posibles tarjetas válidas para el estudio, se descartan todas aquellas que tengan un valor inferior.

Analizando tarjetas de dos fabricantes en concreto: National Instruments y GaGe, se comprueba que existen varios modelos compatibles con el criterio anterior. En la *Tabla 1* y *Tabla* 2 se indican los modelos válidos [9] y [10].

Tabla 1. Modelos válidos de tarjetas de adquisición de NI

Tarjetas de adquisición: National Instruments
NI PCI-5152,NI PCI-5153, NI PCI-5154
NI PXIe-5162, NI PXIe-5185, NI PXIe-5186
NI PXI-5152, NI PXI-5153, NI PXI-5154

Las tarjetas del tipo PXIe (PXI Express) se distinguen de las tarjetas PXI en su elevado rendimiento y mayor ancho de banda.

Tabla 2. Modelos válidos de tarjetas de adquisición de GaGe

Tarjetas de adquisición: GaGe		
Modelo	**Conexión**	**Tarjeta**
CobraMax	PCI/PCIe	CSE14G8
		CSE24G8
Cobra	PCI/PCIe	CSE21G8
		CSE22G8

El modelo CSE21G8 sólo es válido si se emplea un único canal, ya que si se emplean los dos canales de los que dispone simultáneamente la frecuencia de muestreo se reduce a 500 MS/s. Como en esta investigación, sólo se hace uso de un canal, se considera un modelo compatible con las especificaciones.

Debido al elevado precio de estas tarjetas, que suele estar entre los 5500-17500 euros, se ha elegido la opción más económica. En el laboratorio se dispone de la tarjeta NI PCI-5152, y por ello, con el fin de evitar una compra innecesaria y ahorrar así una cantidad importante de dinero, es la que se ha utilizado para la adquisición de datos.

Por último, es necesario especificar que, puesto que esta tarjeta tiene una resolución de 8 bits, todos los modelos citados en las tablas anteriores se han buscado con la misma resolución. Esta lista se podría ampliar si se tienen en cuenta tarjetas de mayor resolución.

2.2.3. Tarjetas de adquisición de NI con drivers de Matlab

Una vez que la tarjeta está incrustada en la ranura del ordenador, hay que establecer la comunicación entre la tarjeta y el PC. Para ello, es imprescindible instalar un determinado driver, encargado de realizar esta operación.

En la página oficial de *National Instruments* se puede ver que el driver de Matlab destinado a la adquisición de datos se denomina NI DAQmx, pero este driver no permite la comunicación con todas las tarjetas. Accediendo al enlace de la bibliografía [11] se puede ver que, precisamente para la tarjeta del laboratorio no se puede establecer la comunicación con este driver. Para establecerla, se empleará el driver universal de *National Instruments* denominado NI-Scope, como se detalla en un capítulo posterior.

2.2.4. Tarjeta de adquisición NI PCI-5152

La tarjeta NI PCI-5152 consiste en un osciloscopio/digitalizador de alta velocidad que permite recoger señales en un ancho de banda analógico de hasta 300 MHz, y cuya velocidad de muestreo puede ser de hasta 2 GS/s. Se trata de una tarjeta PCI, por lo que se conecta al ordenador a través de una ranura de la placa base del mismo [12].

Figura 9. Tarjeta de adquisición NI PCI-5152 [12]

Sus especificaciones más importantes se resumen en la *Tabla 3*.

Tabla 3. Especificaciones de la tarjeta de adquisición NI PCI-5152

Especificaciones Tarjeta NI PCI-5152	
General	
Tipo de tarjeta	PCI
Sistema Operativo/Objetivo	Real-Time, Windows
Familia de productos	Digitalizadores/Osciloscopios
Entrada analógica	
Número de canales	2
Resolución de entrada analógica	8 bits
Rango de voltaje máximo	± 5 V
Impedancia de entrada	1 MΩ, 50 Ω
Memoria interna	8 MB-256 MB
Salida Analógica	
Número de canales	0
E/S Digital	
Canales bidireccionales	0
Temporización/Disparo/Sincronización	
Bus de Sincronización (RTSI)	Sí
Capacidad de reloj T	Sí
Disparo	Analógico/Digital
Reloj Externo	Sí

2.3. Placa Arduino Uno

2.3.1. Microcontrolador

Puesto que el *Arduino* se trata de un microcontrolador, antes de explicar en qué consiste esta placa, se expone una breve introducción de este concepto [3].

Un microcontrolador consiste en un circuito integrado programable. Esto implica que es capaz de ejecutar una serie de instrucciones definidas previamente. Los tres componentes básicos por los que está compuesto son:

- ***CPU (Unidad Central de Proceso)***. Se encarga de llevar a cabo las instrucciones definidas por el usuario mediante el uso de los datos de entrada y la generación de unos datos de salida.
- ***Memoria.*** En ella se almacenan tanto las instrucciones como los datos que necesitan, y existen dos tipos de memoria:
    - *Memoria volátil.* La información almacenada en ella se pierde tras desconectar la fuente de alimentación.
    - *Memoria persistente.* La información que contiene permanece incluso cuando se deja de recibir alimentación.
- ***Patillas de E/S.*** Estas patillas permiten la comunicación del microcontrolador con el mundo físico. En las patillas de entrada se conectan los sensores y en las de salida los actuadores. Es importante destacar que, muchas de las patillas, pueden ser empleadas como entrada o salida.

A modo de resumen, decir que un microcontrolador es un circuito integrado encargado de la ejecución de una serie de instrucciones, y para la cual, tiene en cuenta la información recogida y enviada por las patillas E/S.

2.3.2. Introducción a Arduino

Arduino es una herramienta de bajo coste diseñada para interactuar con el mundo exterior de manera directa. Se trata de una plataforma de microcontrolador fácil de usar, y por ello, es la más indicada para personas que desean iniciarse en el mundo de la automatización [3].

A través de interruptores y sensores recibe información del entorno y toma medidas de movimiento o posición, entre otras; y controla una amplia variedad de salidas físicas, como pueden ser luces o motores [13].

2.3.3. Definición de Arduino

La definición de *Arduino* no es simple, ya que se trata de la combinación de una placa hardware, un software y un lenguaje de programación. Por ello, para su mejor entendimiento se explica cada uno de estos elementos [3].

Por un lado, se trata de una placa hardware, es decir, se trata de una placa de circuito impreso que incorpora un microcontrolador reprogramable y una serie de pines-hembra conectadas a las patillas E/S a las que se conectan los sensores y actuadores. Con la placa de circuito impreso se consigue disponer de un circuito electrónico diseñado de la forma más compacta y estable posible.

Por otro lado, el *Arduino* consiste en un software abierto y multiplataforma que se debe instalar en el ordenador. A través de él, se realiza la programación del microcontrolador. Para ello, primero se conecta la placa al ordenador mediante un cable USB, y a continuación, se escriben y guardan en su memoria el conjunto de instrucciones que deberá realizar posteriormente. Existe un entorno de programación o IDE diseñado específicamente para esta plataforma que permite introducir fácilmente los programas en el microcontrolador y depurarlos.

Por último, se trata de un lenguaje de programación. Es un idioma artificial que tiene unas determinadas reglas sintácticas, y que se emplea para expresar las instrucciones. Se caracteriza por tener bloques comunes con otros lenguajes de programación, como pueden ser los bucles o los bloques condicionales.

Los tres componentes de la definición son libres. Esto implica que cualquier persona puede mejorar tanto el diseño del hardware como el software y el lenguaje de programación, realizar copias y distribuirlo. Incluso, tienen acceso a todas las especificaciones de su diseño para elaborar su propia placa partiendo de cero.

La definición en sí de *Arduino* contiene las ventajas que le distinguen del resto de microcontroladores que existen en el mercado, y que se resumen en los siguientes puntos [13]:

- *Barato.*
- *Software multiplataforma.* Los programas se pueden desarrollar en Windows, Macintosh OS y Linux.
- *Entorno de programación simple.*
- *Software de código abierto y con capacidad de ampliación.*
- *Hardware ampliable.* Todos los planos y esquemas de las placas han sido publicados bajo una licencia Creative Commons.

Existen diversos modelos de placas *Arduino*, que se diferencian en forma, rendimiento y prestaciones, y en función de la aplicación deseada, será recomendable una u otra.

Para la realización de este proyecto, se ha seleccionado el modelo estándar, denominado *Arduino Uno*, ya que constituye la placa más asequible para alcanzar nuestros objetivos.

2.3.4. Placa Arduino Uno

Arduino Uno es una placa basada en el microcontrolador ATmega328. Cuenta con 14 pines de entradas y salidas digitales (6 de ellos se pueden utilizar como salidas PWM[2]), 6 entradas analógicas, conexión USB, conector de alimentación y un botón de reinicio [13].

La versión empleada en el proyecto es la revisión tres, y el nombre completo de la placa es *Arduino Uno Rev3*.

Figura 10. Arduino Uno R3 (Parte frontal) [13]

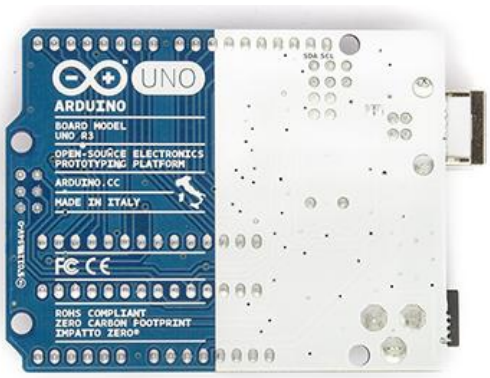


Figura 11. Arduino Uno R3 (Parte de atrás) [13]

[2] Pulse Width Modulation: Técnica para obtener resultados analógicos mediante un control digital. Se explica más detalladamente en la página 31.

En la *Tabla 4* se especifican las características principales:

Tabla 4. Características de la placa Arduino Uno Rev3

Arduino Uno Rev3	
Microcontrolador	ATmega328
Tensión de operación	5 V
Tensión de entrada (recomendado)	7-12 V
Tensión de entrada (límites)	6-20 V
Pins E/S digitales	14 (6 disponibles como salidas PWM)
Pins de entrada analógica	6
Corriente DC por pin E/S	40 mA
Corriente DC por pin de 3.3 V	50 mA
Memoria Flash	32 KB (ATmega328) ,0.5 KB utilizados por el gestor de arranque (bootloader)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del reloj	16 MHz
Longitud	68,6 mm
Anchura	53,4 mm
Peso	25 g

2.3.4.1. Alimentación

Para que funcione la placa *Arduino Uno*, es necesario suministrarla energía. Puede ser alimentada de dos maneras distintas [3].

- ***Mediante una fuente externa.*** Como fuente externa se puede utilizar un adaptador AC/DC o una pila. Si se dispone de un adaptador AC/DC, la conexión se realiza a través del zócalo diseñado para introducir una clavija tipo "jack", y si por el contrario, la alimentación se realiza mediante una pila, sus bornes se conectan a las patillas denominadas "Vin" y "GND".
  En esta alternativa, la tensión de alimentación puede estar comprendida entre los 6-20 V, puesto que la placa está diseñada para soportar dicho voltaje. Sin embargo, no es recomendable trabajar en este rango porque la mayoría del voltaje se pierde en forma de calor, produciéndose un sobrecalentamiento que puede perjudicar al funcionamiento de la placa. El rango recomendado es el de 7-12 V. A pesar de ello, la placa dispone de un circuito regulador de tensión que la reduce a 5 V.
- ***Mediante un ordenador.*** La alimentación se lleva a cabo a través del conector USB hembra de tipo B situado en la placa, y la tensión siempre está regulada en 5 V. Presenta la característica de que no se puede sobrepasar un límite de 500 mA, y para evitar que se supere, está protegido con un polifusible reseteable que

interrumpe la conexión durante el período de tiempo en el que se supera dicho límite de corriente.

Debido a la limitación de corriente del segundo modo de alimentación, los proyectos que requieren un consumo elevado de potencia deben ser alimentados mediante una fuente externa. En nuestra investigación, se aplica el segundo tipo de alimentación.

En la tarjeta, la zona correspondiente a la alimentación viene señalada con la palabra "POWER", y está compuesta por los siguientes pines:

- **GND.** Son las patillas conectadas a tierra, y se encargan de que todos los elementos del circuito tengan una tierra común.
- **Vin.** Esta patilla proporciona a cualquier componente electrónico la tensión definida en la fuente externa de alimentación en el caso de que se trate de un adaptador AC/DC, o una tensión de 5 V si se alimenta mediante cable USB. Incluso, se puede emplear para alimentar la propia placa conectando a ella directamente la fuente externa.
- **5 V.** Independientemente del tipo de alimentación, proporciona una tensión de 5 V a cualquier componente electrónico conectado a ella. También puede alimentar la propia placa si se conecta a dicho pin una fuente externa que necesariamente ha tenido que ser regulada a la tensión de 5 V.
- **3,3 V.** Este pin proporciona una tensión de 3,3 V, regulada por un circuito contenido en la placa.

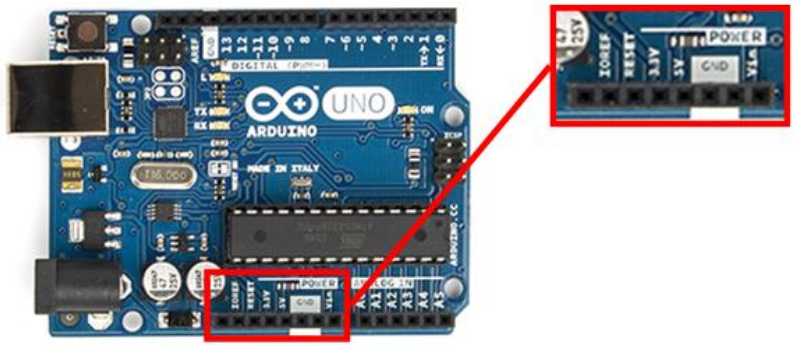


Figura 12. Patillas de alimentación de la placa Arduino Uno. Adaptado de [13]

2.3.4.2. Patillas de E/S

Aunque todas las patillas de E/S comunican el microcontrolador con el mundo físico, cada una de ellas tiene una función específica. La disposición de los pines de la placa *Arduino Uno* es la correspondiente al modelo ATmega328P, y es la que se muestra en la siguiente figura [3].

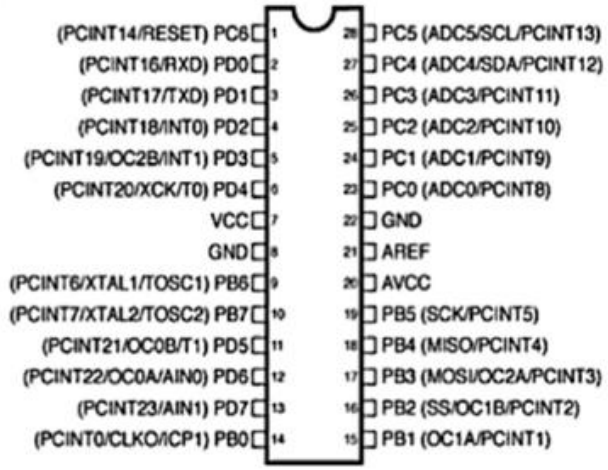


Figura 13. Esquema del microcontrolador ATmega328P [3]

La *Figura 13*, además de mostrar la disposición de los pines, indica el nombre de cada una de las patillas, su función específica y su función genérica de entrada/salida. Estas dos últimas se corresponden con la información expresada entre paréntesis. Las patillas de entrada/salida se identifican porque están señaladas con el nombre PBx, PCx o PDx. En la *Figura 14* se indica cuál es el microcontrolador en la placa *Arduino Uno*.

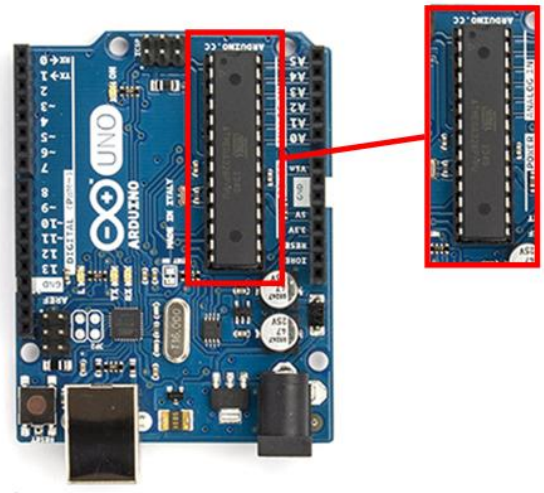


Figura 14. Microcontrolador ATmega328P en la placa Arduino Uno. Adaptado de [13]

2.3.4.2.1. Entradas y salidas digitales

La placa *Arduino Uno* dispone de 14 patillas digitales de entrada/salida, numeradas de 0 a 13. En ellas, se conectan los sensores para recibir información del mundo exterior o los actuadores para enviar órdenes a los sistemas del mundo físico. Todos estos pines-hembra funcionan a 5 V y pueden enviar o recibir como máximo una corriente de 40

mA. Pese a que cada pin es capaz de proporcionar de manera individual una corriente de 40 mA, las patillas están agrupadas de tal manera que un conjunto de ellas no pueden superar el valor de 100 mA. Así, los grupos de patillas que no pueden sobrepasar dicho valor son: pines 0-4 y pines 5-13. Por tanto, como máximo podrían estar trabajando simultáneamente proporcionando una corriente de 20 mA diez de los catorce pines [3].

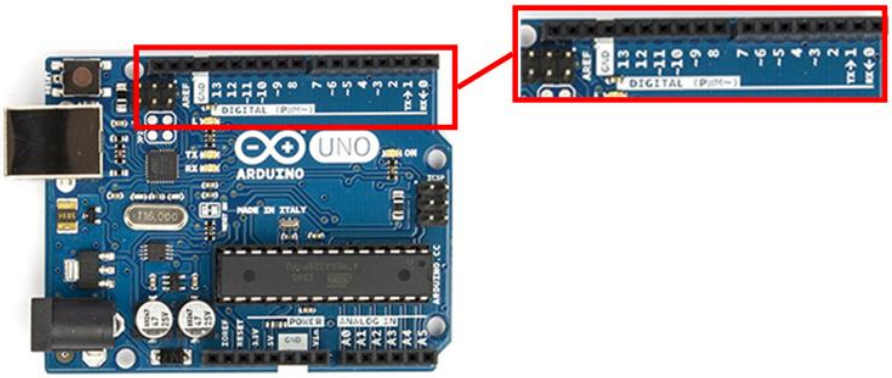


Figura 15. Patillas de entradas/salidas digitales de la placa Arduino Uno. Adaptado de [13]

2.3.4.2.2. Entradas analógicas

La tarjeta dispone de 6 entradas analógicas, numeradas como A0-A5, y funcionan con tensiones comprendidas en un rango de 0-5 V. Puesto que el circuito electrónico trabaja exclusivamente con datos digitales, es imprescindible realizar una conversión de analógico a digital. El encargado de ello, es un circuito conversor analógico/digital que forma parte de la placa [3].

Es importante resaltar que, si en algún momento se necesitasen más pines-hembra digitales, se pueden emplear como tales las entradas analógicas, puesto que tienen la misma funcionalidad.

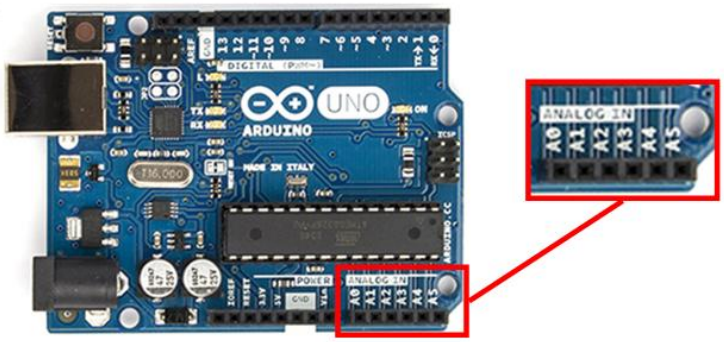


Figura 16. Patillas de entradas analógicas de la placa Arduino Uno. Adaptado de [13]

2.3.4.2.3. Salidas analógicas (PWM)

En ocasiones, es necesario enviar señales que varíen continuamente con el tiempo, es decir, señales analógicas. Sin embargo, la placa no dispone de salidas analógicas, y el envío de este tipo de señales se consigue simulando dicho comportamiento mediante patillas de salida digital. No todas las salidas digitales son capaces de trabajar en este modo de funcionamiento, sólo aquellas que aparecen señaladas con la palabra "PWM" (pines 3, 5, 6, 9, 10 y 11) [3].

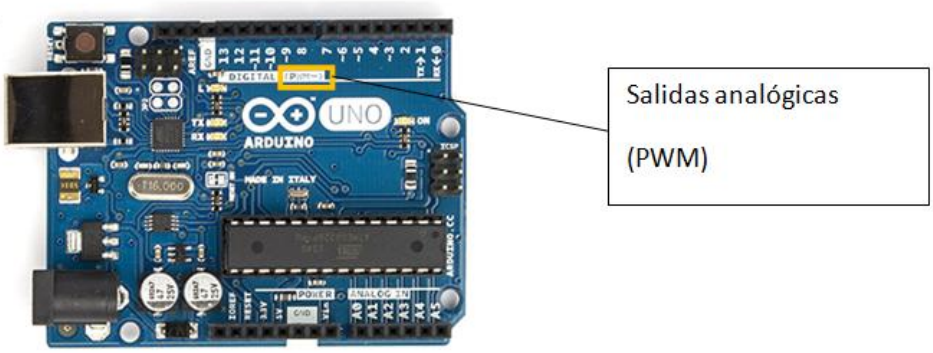


Figura 17. Patillas de salidas analógicas (PWM) de la placa Arduino Uno. Adaptado de [13]

Como explicación general, las patillas PWM (Pulse Width Modulation o Modulación de Ancho de Pulso), envían señales cuadradas constituidas por pulsos de frecuencia constante. Al variar el ancho de los pulsos, se modifica la tensión promedio, y por tanto, la tensión de salida. Esta variación se lleva a cabo mediante un control digital, y se pueden simular voltajes comprendidos entre 0 y 5 V. Cuanto mayor sea la duración del pulso, mayor será la tensión resultante, y cuanto menor sea su duración, menor será la tensión de salida, como puede verse en la *Figura 18*.

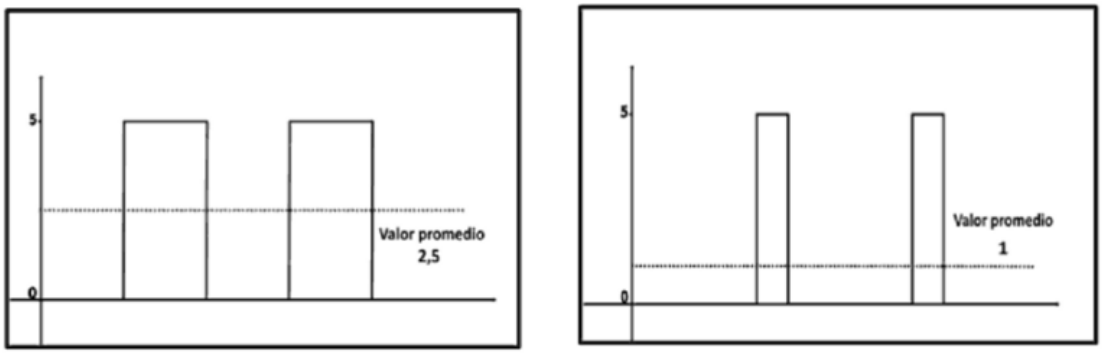


Figura 18. Tensión promedio con distintos anchos de pulso [3]

En la *Figura 19* se muestra un ejemplo de la variación de la tensión al variar el ancho de los pulsos.

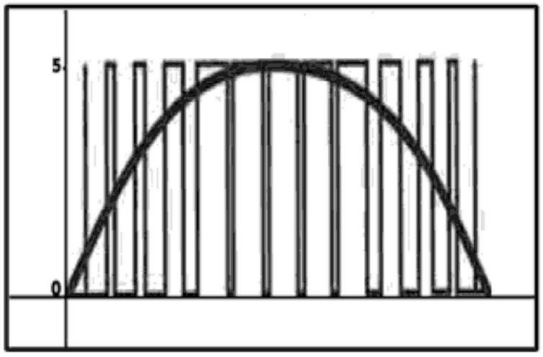


Figura 19. Variación de la tensión con la variación del ancho del pulso [3]

Las patillas PWM sirven principalmente para controlar servomotores, aunque también se pueden utilizar para otro tipo de aplicaciones como puede ser regular la intensidad de luminosidad de leds.

2.3.5. Comunicación con el Arduino

Para que la placa *Arduino* realice una determinada función es imprescindible grabar en su memoria el conjunto de instrucciones que debe ejecutar. Con el fin de que esta grabación sea posible, hay que instalar en el ordenador lo que se denomina IDE o Entorno de Desarrollo Integrado, donde se escribe el programa.

En este proyecto, la comunicación con la placa *Arduino* se realiza de una manera alternativa. Puesto que el programa principal que gobierna todas las operaciones del trabajo está escrito en el entorno Matlab, la comunicación con el dispositivo *Arduino* también se va a realizar en este entorno digital. Por ello, no va a ser necesario instalar el IDE, pero sí será necesaria la instalación de un paquete de apoyo de Matlab.

La instalación de Matlab no incluye todas las funciones que es capaz de realizar. En el caso de la comunicación con la placa *Arduino*, requiere de la instalación adicional de un paquete de apoyo, disponible en su página web oficial: *MathWorks*. Este paquete consiste en un programa servidor que se ejecuta en la placa, lee y realiza las instrucciones, y devuelve, si es necesario, unos resultados. Una vez instalado, la comunicación se basa simplemente en la conexión de la placa con el ordenador mediante un cable USB, y la ejecución de unas líneas de código implementadas en el entorno Matlab [14].

2.4. Servomotores

2.4.1. Introducción

Un servomotor, también denominado servo, consiste en un sistema compuesto por un motor eléctrico, un sistema de regulación que actúa sobre el motor y un sistema de sensor que controla el movimiento del motor. El hecho de que un motor incluya un servo implica que está destinado para el control en bucle cerrado de su par, velocidad o posición del eje [15].

Con él se pueden crear toda clase de movimientos de forma controlada y es la mejor opción para accionamientos rápidos y precisos. Su elevada capacidad de sobrepar, igual a tres veces el par nominal, y la baja inercia del motor, le proporciona la capacidad de realizar elevados esfuerzos y rápidas aceleraciones y deceleraciones con gran precisión [16].

Se trata de un dispositivo económico y fácil de utilizar, muy extendido en las aplicaciones de robótica. En la actualidad, se está incorporando en todas las ramas de la industria.

En este proyecto, se utilizará para controlar el movimiento de una antena de alta directividad.

2.4.2. Control de la posición del servomotor

Como se ha especificado anteriormente, los servomotores se emplean para controlar de manera precisa la posición de distintos dispositivos, y este control es en lazo cerrado. Para comprender cómo se lleva a cabo este control, es necesario conocer cuáles son sus elementos internos. Todo servomotor está compuesto por [17]:

- *Motor eléctrico.* Genera el movimiento.
- *Sistema de control.* Circuito electrónico que controla el movimiento mediante la técnica PWM.
- *Sistema de regulación.* Está formado por engranajes reductores y actúa sobre el motor para regular su velocidad y par. Los engranajes convierten la velocidad de giro en par.
- *Potenciómetro conectado al eje central.* Indica el ángulo en el que se encuentra el eje del motor.

El proceso de control es el siguiente: El sistema de control recibe por el cable correspondiente a la señal de control pulsos eléctricos enviados por la placa *Arduino* que determinan la nueva posición. A continuación, supervisa la posición actual a través

del potenciómetro, que le envía una tensión proporcional al ángulo en el que se encuentra en ese momento el servomotor, y calcula el error existente entre ambos voltajes (tensión proporcionada por los pulsos y tensión de la posición actual). Si la diferencia es nula, el motor está bien colocado y no recibe tensión, permanece en reposo; por el contrario, si la diferencia es distinta de cero, se le aplica al motor una tensión proporcional a la distancia que debe recorrer y este gira hasta posicionarse en el lugar correcto [18].

2.4.3. Movimiento del servomotor

Como se ha mencionado anteriormente, la posición del servo se controla a través de un sistema de control basado en el ancho de pulsos (técnica PWM), de tal forma que [15]:

- Cuando el ancho es igual a 1,5 ms, el servo se encuentra centrado, en la posición cero.
- Cuando es inferior a 1,5 ms, gira hacia la izquierda.
- Cuando es superior a 1,5 ms, gira hacia la derecha.

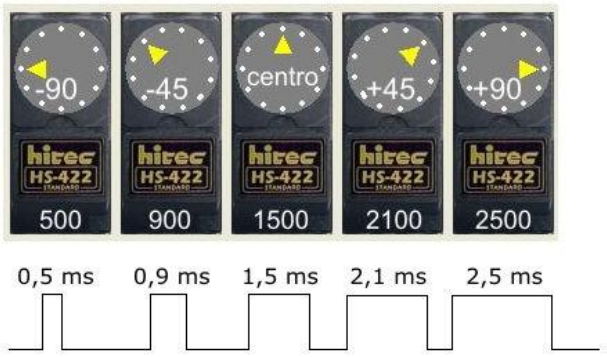


Figura 20. Ancho del pulso según la posición [15]

En este trabajo, la posición del servomotor se controla con la placa *Arduino* a la que está conectado, y el tiempo que tarda en enviarse cada pulso depende de la frecuencia de refresco del pulso. Así, si la frecuencia es de 50 Hz, como es nuestro caso, se enviarán pulsos cada 20 ms.

2.4.4. Limitación de los servomotores

El movimiento de los servos está limitado, y en la mayoría de los modelos, esta limitación es de 180º, sólo son capaces de girar 90º hacia cada lado. Puesto que existe una restricción respecto a posiciones, y estas se alcanzan indicando el ancho del pulso, también existe un rango en el que deben encontrarse los anchos de los pulsos. Este

rango varía en función de los modelos y de los fabricantes, pero suele estar en torno a los siguientes valores [15]:

Tabla 5. Rango del ancho del pulso de un servo

Grados	Ancho del pulso
-90º	500 μsec
90º	2500 μsec

2.4.5. Conexiones de los servomotores

Los servos disponen de tres conexiones eléctricas: negativo, alimentación y señal de control; que se identifican por el color del cable [15]:

Tabla 6. Conexiones eléctricas de un servomotor

Conexiones eléctricas de un servomotor	
Color del cable	**Tipo de conexión**
Negro	Tierra (Negativo)
Rojo	Alimentación (Positivo)
Amarillo	Señal de control

2.4.6. Servo Hitec HS-422

El servomotor empleado es el Servo Hitec HS-422. Tiene unas dimensiones estándar y se diferencia del resto en que es capaz de proporcionar mayor potencia gracias al cojinete de salida metálico que transfiere toda la potencia al eje de salida con precisión y suavidad [19].

Figura 21. Servo Hitec HS-422 [20]

Este modelo forma parte del grupo de servos que presentan una limitación de movimiento igual a 180º, pero en esta ocasión, el rango del ancho del pulso de las señales difiere del especificado en el apartado *Limitación de los servomotores*.

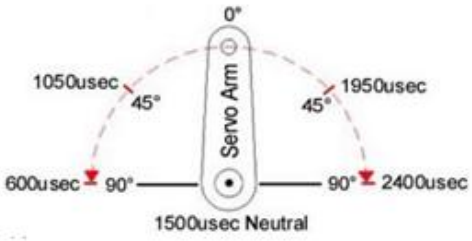


Figura 22. Rango de operación del Servo Hitec HS-422 [21]

El rango del ancho del pulso se resume en la siguiente tabla.

Tabla 7. Rango del ancho del pulso del Servo Hitec HS-422

Grados	Ancho del pulso
-90º	600 μsec
90º	2400 μsec

En nuestra aplicación nos referiremos al ángulo -90º como posición inicial o posición cero, ya que es el punto de partida del barrido del servomotor, y al ángulo 90º, como posición final (180º). Esta nomenclatura será la utilizada en el resto de la memoria.

Todas las especificaciones del servo se muestran en el apartado *Hoja de características: Servomotor Hitec HS-422* del capítulo *Anexos* del presente proyecto.

2.5. Entornos de programación

Existen dos posibles entornos de programación en los que desarrollar los algoritmos necesarios para la adquisición, análisis y procesamiento de datos: LabVIEW y Matlab. A continuación, se expone una breve descripción de ambos entornos y las razones por las que se ha optado por el segundo de ellos para nuestra aplicación.

2.5.1. LabVIEW

LabVIEW es un entorno de programación desarrollado por *National Instruments* destinado a aplicaciones de adquisición, control, análisis y presentación de datos. A diferencia del resto de entornos, es el único que emplea un lenguaje de programación gráfico, y es muy intuitivo y fácil de utilizar [22].

Los programas desarrollados con LabVIEW se denominan "*Instrumentos Virtuales (VI's)"*. El origen de este nombre se debe a que su aspecto y funcionamiento es el mismo que el de un instrumento real. Los *VI's* se diseñan a partir de dos ventanas: un

panel frontal que constituye la interfaz gráfica mediante la cual se interactúa con el usuario, y una ventana con diagramas de bloques que contiene el código.

El proceso de creación de un VI es el siguiente. En primer lugar, se diseña la interfaz gráfica, encargada de recoger las entradas definidas por el usuario y de mostrar las salidas proporcionadas por el programa. En ella, se colocan los pulsadores, gráficas, potenciómetros y demás elementos que requiera la aplicación.

Por último, se implementa el programa en la segunda ventana a partir de funciones y estructuras integradas en las librerías de LabVIEW representadas mediante bloques, y de todos los elementos colocados en el panel frontal, representados por terminales. Todos estos objetos se conectan mediante cables que indican la trayectoria que siguen los datos.

2.5.2. Matlab

Matlab (MATrix LABoratory) es un software muy potente utilizado en ingeniería. Trabaja con vectores y matrices, además de números escalares, ya sean reales o complejos; y su uso se extiende a una gran variedad de aplicaciones [23] y [24]:

- Desarrollo de algoritmos
- Modelado, simulación y prueba de prototipos
- Cálculos numéricos
- Representación de datos
- Análisis de datos, exploración y visualización
- Desarrollo de aplicaciones basadas en la creación de una interfaz gráfica de usuario

Matlab posee un lenguaje de programación propio y dispone de librerías especializadas denominadas *toolboxes* que le permiten trabajar en todos los ámbitos científicos.

Además, incluye dos herramientas basadas en editores gráficos: *Simulink* y *GUIDE*. La herramienta *Simulink* está destinada a modelar sistemas y simular su comportamiento. En cuanto a la herramienta *GUIDE*, se trata de una interfaz gráfica de usuario muy intuitiva.

2.5.3. Matlab frente a LabVIEW

Ambos entornos de programación presentan los mismos objetivos finales: permiten desarrollar aplicaciones basadas en interfaces gráficas de usuario muy intuitivas, y ambos permiten alcanzar el objetivo de este proyecto: la adquisición y análisis de datos.

La diferencia entre ellos se encuentra en la forma de implementar el código a ejecutar y la manera que tiene cada uno de procesar las señales.

LabVIEW tiene como ventaja su sencillez a la hora de programar, puesto que sólo requiere la unión de los bloques adecuados, pero presenta un inconveniente: estos bloques representan funciones y estructuras incluidas en las librerías de este entorno. Habría que desarrollar el programa teniendo en cuenta en todo momento las funciones de las que se dispone o bien desarrollar nuevos bloques con el código de las funciones que se quieren añadir, lo que puede entrañar cierta dificultad.

Matlab, sin embargo, es más versátil, y mediante su lenguaje de programación escrito, se pueden crear funciones que no existen, permitiendo realizar todo tipo de estudios con los datos adquiridos. A continuación, se muestran varios ejemplos de la implementación de algunas funciones en ambos entornos, en los que se demuestra que en el primer entorno es imprescindible que exista el bloque requerido.

Tabla 8. FFT de una señal en LabVIEW y Matlab

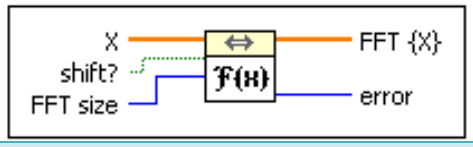
Transformada de Fourier (FFT)	
LabVIEW	**Matlab**
	`>> FFT=fft(X)/L`
X es la señal de entrada. ***Shift*** indica si la componente DC está en el centro de la transformada de Fourier. Por defecto, su valor es *False*. ***FFT size*** es la longitud de la FFT que se desea realizar. ***L*** es la longitud del vector *X*. ***FFT*** es la transformada de Fourier del vector *X*.	

Tabla 9. Transpuesta de una matriz en LabVIEW y MATLAB

Transpuesta de una matriz	
LabVIEW	**Matlab**
	`>> transposed_matrix=matrix'`
matrix es la matriz de entrada. Debe ser de dos dimensiones y puede estar compuesta por números reales o complejos. ***transposed matrix*** es la transpuesta de *matrix*.	

Otro inconveniente de LabVIEW es que procesa los comandos con mayor lentitud y no está preparado para realizar análisis de las señales que se recogen. Para operar con ellas, sería necesario pasar las variables correspondientes a Matlab.

Por estos motivos, además del hecho de que ya se elaboró un proyecto similar con la misma finalidad que este pero con la herramienta LabVIEW, en esta ocasión, se va a implementar el programa en el entorno Matlab, y en concreto, se va a emplear la herramienta *GUIDE*, explicada detalladamente en el capítulo posterior.

Capítulo 3

3. Entorno digital del proyecto

Con el fin de facilitar la comprensión de los conceptos explicados en los siguientes apartados se ha decidido emplear la notación en inglés que aparece en la interfaz de Matlab.

3.1. Introducción a Guide

GUIDE (Graphical User Interfase Development Environment) es un conjunto de herramientas diseñado para crear interfaces gráficas de usuario (GUI, Graphical User Interfaces) de manera sencilla. Mediante estas interfaces se establece una comunicación entre el programa de ordenador y el usuario [23] y [24].

La interfaz gráfica, a través de un conjunto de botones, menús, campos de texto y gráficos, muestra información por pantalla, y el usuario, a través del ratón o el teclado, selecciona, activa e introduce datos en dichos objetos, produciéndose así una interacción entre ambos.

El empleo de GUI's presenta la ventaja de que, puesto que se trata de un programa muy intuitivo, una vez diseñada la interfaz, cualquier persona puede hacer uso de ella, independientemente de que tenga o no conocimientos de Matlab.

3.1.1. Interfaz de una GUI

Existen dos posibilidades de acceder a la creación de una nueva GUI desde Matlab:

- Escribiendo en la ventana de comandos la instrucción
  >>guide
- Seleccionando el icono correspondiente a *GUIDE*, tal y como se muestra en la *Figura 23*.

Figura 23. Acceso a la herramienta GUIDE mediante icono

Tras ejecutar alguno de los comandos anteriores, aparece un cuadro de diálogo en el que se debe seleccionar una de las cuatro opciones que se muestran por pantalla. En la *Figura 24* se presenta dicho cuadro de diálogo, especificando a continuación la aplicación a la que está destinada cada una de las alternativas.

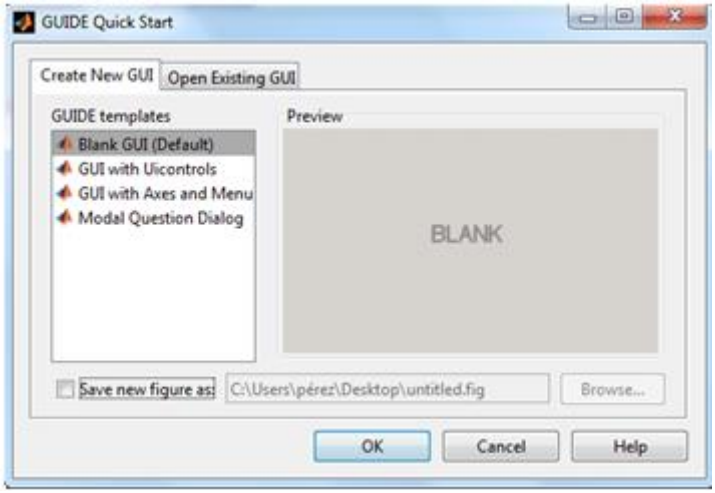


Figura 24. Cuadro de diálogo para la creación de una nueva GUI

- ***Blank GUI (Default).*** Se abre una interfaz gráfica de usuario en blanco en la que se puede crear un nuevo programa.
- ***GUI with Uicontrols.*** Muestra cómo introducir datos y operar con ellos.
- ***GUI with Axes and Menu.*** Muestra cómo se maneja un menú desplegable (diseñado mediante un *Pop-Up-Menu*[3]) y un objeto *Axes*.
- ***Modal Question Dialog.*** Muestra cómo se resuelven preguntas del tipo *sí/no*, *aceptar/cancelar*, etc.

Para la creación de nuestro programa, partimos de una interfaz en blanco. En ella, aparece a la izquierda una paleta de componentes mediante los cuales se diseña la interfaz gráfica, y en la parte superior, existe un conjunto de menús e iconos que facilitan la configuración. En la *Figura 25* se muestra la interfaz gráfica de partida.

[3] Los elementos *Pop-Up-Menu* y *Axes* son dos de los componentes que se pueden incorporar a una interfaz gráfica. Posteriormente se explican en profundidad.

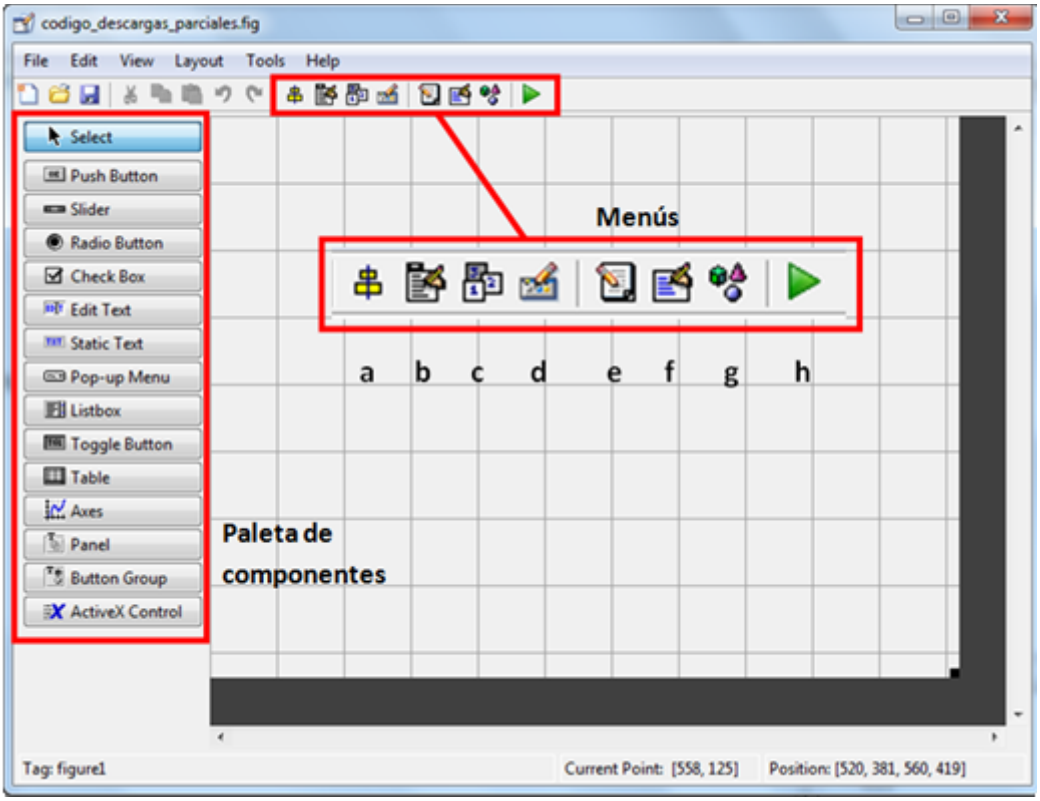


Figura 25 Editor gráfico de una GUI

3.1.2. Paleta de componentes

Select

Este elemento está activado por defecto. Permite seleccionar cualquier elemento de la interfaz con el fin de cambiar su posición o acceder a su *Property Inspector*. Más adelante, se explica en qué consiste esta última ventana.

Push Button

Crea un botón rectangular con una etiqueta de texto que se activa cuando se hace click sobre él. Cuando se selecciona, se ejecutan las acciones definidas en este objeto. Finalizadas las órdenes, vuelve a su estado predeterminado a la espera de volver a ser pulsado. La finalidad de este botón es realizar un conjunto de acciones en un momento determinado.

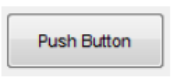


Figura 26. Push Button

Slider

Se trata de una barra deslizante que puede ser desplazada por el usuario. Cada posición de la barra está asociada a un valor, y la posición de inicio y fin se corresponden con el rango de valores que puede tomar la variable asociada a dicha barra.

Figura 27. Slider

Radio Button

Consiste en un elemento que permite activar o desactivar una acción. Se parte de un *Button Group* con varias opciones, y sólo se puede seleccionar una de ellas.

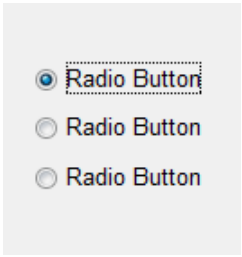


Figura 28. Radio Button

Check Box

Al igual que el *Radio Button*, consiste en un elemento que puede activarse o desactivarse. La diferencia que existe con el objeto anterior es que se pueden seleccionar varias de las acciones que componen el *Button Group.*

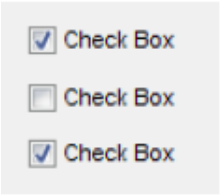


Figura 29. Check Box

Edit Text

Se trata de un elemento en el que el usuario introduce un conjunto de caracteres, ya sean letras o números, que se almacenan en una variable.

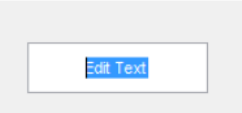


Figura 30. Edit Text

Static Text

Es un elemento que muestra información, tanto texto como números, y no puede ser modificada por el usuario.

Figura 31. Static Text

Pop-Up Menu

Crea un menú desplegable que muestra diferentes tareas entre las que el usuario puede elegir. Según la opción que se seleccione, se ejecutan una serie de instrucciones.

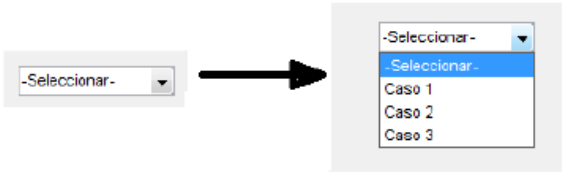


Figura 32. Pop-up Menu

Listbox

Este elemento tiene la misma función que el *Pop-Up Menu*. Se diferencia en que no se trata de un menú desplegable, muestra directamente las diversas posibilidades en una lista.

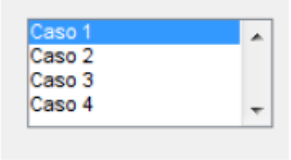


Figura 33 - Listbox

Toggle Button

Se trata de un botón similar al *Push Button*. Tiene dos estados: encendido y apagado, y toma los valores de 1 y 0, respectivamente. Cuando se hace click sobre el botón, se activa (color azul) y ejecuta una serie de instrucciones. Cuando se pulsa de nuevo, se desactiva (color gris) y deja de ejecutar las acciones.

Figura 34. Toggle Button

Table

Se crea una tabla de contenidos en la que se muestran datos en cada uno de los elementos que la componen.

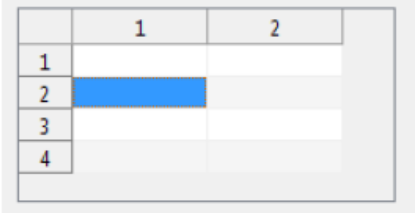


Figura 35. Table

Axes

Se trata de un elemento en el que se pueden mostrar tanto gráficos generados con el comando *plot* como imágenes con formato *.jpeg* o *.png*.

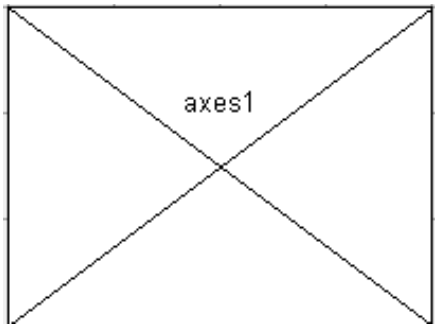


Figura 36. Axes

Panel

Se emplea para englobar varios elementos de la interfaz que están relacionados en un recuadro, de tal manera que se puede desplazar el conjunto entero. Así, se consigue una interfaz con un aspecto más organizado.

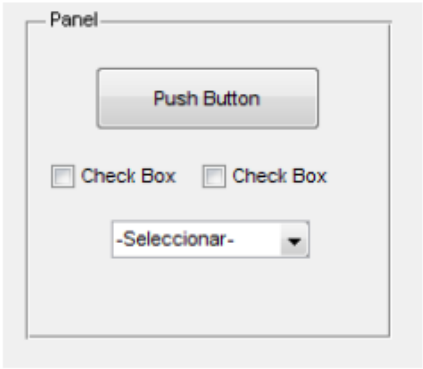


Figura 37. Panel

Button Group

Es un objeto similar al *Panel*, permite agrupar objetos, pero establece exclusividad mutua entre botones, sólo permite seleccionar uno de los elementos.

3.1.3. Menús

a. Align Objects

Este menú permite alinear los elementos incorporados en la interfaz. Se puede elegir entre diversas configuraciones.

Figura 38. Align Objects

b. *Menu Editor*

Crea menús y submenús desplegables, y estos se generan en la parte superior de la interfaz.

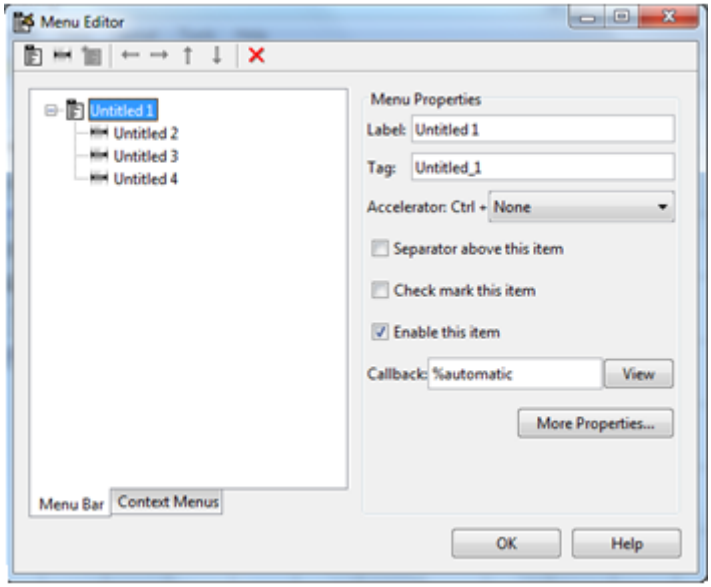


Figura 39. Menu Editor

c. *Tab Order Editor*

Permite establecer por tabulación el orden de acceso a cada objeto de la GUI.

d. *Toolbar Editor*

Permite añadir los iconos de una barra de herramientas y personalizarlos: guardar, imprimir, zoom, rotar, etc.

e. *M-file Editor*

Con este icono se abre el archivo *.m* asociado al GUI creado.

f. *Property Inspector*

Se trata de un menú en el que se pueden modificar las propiedades de los elementos de la GUI. En el apartado *Edición de los objetos de una GUI* de este capítulo se describen algunas de estas propiedades.

g. *Object Browser*

Muestra todos los componentes en un esquema en forma de árbol.

h. Run Figure

Cuando se selecciona el icono  , se ejecuta la interfaz, permitiendo al usuario interactuar con el programa.

3.2. Programación en GUIDE

3.2.1. Creación de una GUI

En este apartado, se explican las ideas más importantes que permiten adquirir un conocimiento básico para poder programar una interfaz gráfica de usuario con *GUIDE*.

Como primer concepto, hay que saber que cuando se crea una GUI se generan dos archivos: un archivo *.fig* y un archivo *.m*, ambos necesarios para la ejecución del programa.

El *archivo .fig* contiene el diseño gráfico de la GUI. Dependiendo de cómo se acceda a él, se abre la ventana final o la ventana de edición.

Seleccionando el archivo desde las pestañas *File->Open* de Matlab se abre directamente la interfaz gráfica final, con la que interactúa el usuario.

Sin embargo, si todavía no está terminada la interfaz, interesa acceder a su ventana de edición con el fin de terminar de configurarla y realizar todos los cambios que sean oportunos. Para ello, hay que abrir el archivo de la siguiente manera:

1. Se accede a la ventana de creación de una nueva GUI, ya sea escribiendo el comando *guide* o haciendo click en el icono correspondiente.
2. Una vez abierta, se selecciona la pestaña *Open Existing Gui* y a continuación, el botón *Browse.*
3. Por último, se abre el archivo *.fig*

Todas las modificaciones que se llevan a cabo en el editor, se reflejan en el archivo *.m* (archivo de Matlab), en el que se definen todas las instrucciones que se ejecutarán en un momento determinado del programa. Cada vez que se incorpora un nuevo objeto a la interfaz, se genera automáticamente un código en este archivo.

Dicho archivo se caracteriza además, en que el código presenta una estructura segmentada y no se ejecuta de forma secuencial, es decir, comenzando por la primera línea y finalizando en la última. Cada objeto de la GUI tiene sus propias líneas de código, y según el usuario o el programa active un objeto u otro, se ejecuta el segmento de código correspondiente.

El conjunto de líneas de código que se ejecutan se llama *Callback* y se puede acceder a él de dos formas. La primera consiste en buscar visualmente el *Callback* que nos interesa en el código completo de Matlab, y la segunda, es a partir del objeto, haciendo click en el botón derecho y seleccionando:

View Callbacks->Callback

De esta última manera, se abre el archivo *.m* directamente en la parte que nos interesa, evitando así perder tiempo en su localización.

Cuando se crea un GUI en blanco, el archivo *.m* que se genera contiene un código inicial formado por tres funciones que, en principio, no es preciso modificar:

- `function varargout = codigo(varargin).`
- `function codigo_OpeningFcn(hObject, eventdata, handles, varargin)`
- `function varargout = codigo_OutputFcn(hObject, eventdata, handles)`

En las funciones anteriores, la palabra "*codigo"* se corresponde con el nombre que se le ha asignado a los archivos *.m* y *.fig* desarrollados en el proyecto.

La primera función contiene la inicialización del código, y no se puede editar.

```
function varargout = codigo(varargin)
% CODIGO M-file for codigo.fig
%      CODIGO, by itself, creates a new CODIGO or raises the existing
%      singleton*.
%
%      H = CODIGO returns the handle to a new CODIGO or the handle to
%      the existing singleton*.
%
%      CODIGO('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in CODIGO.M with the given input
arguments.
%
%      CODIGO('Property','Value',...) creates a new CODIGO or raises
the
%      existing singleton*.  Starting from the left, property value
pairs are
%      applied to the GUI before codigo_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to codigo_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 codigo
```

```
% Last Modified by GUIDE v2.5 02-Jul-2015 20:01:42

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @codigo_OpeningFcn, ...
                   'gui_OutputFcn',  @codigo_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 40. Código inicial de una GUI titulada "codigo": Función 1

La segunda función se ejecuta una única vez antes de que la interfaz gráfica se haga visible. Pese a que este código inicial no se suele modificar, en nuestro trabajo, sí ha sido necesario incluir líneas de código. Antes de ejecutar cualquiera de las instrucciones de los objetos de la GUI, es imprescindible establecer una comunicación con el *Arduino*, una comunicación con la tarjeta de adquisición y configurar sus parámetros con el fin de que se puedan llevar a cabo todas las acciones con éxito. Estas operaciones sólo se tienen que ejecutar una vez al principio del programa, y por ello, se programan dentro de esta función.

```
% --- Executes just before codigo is made visible.
function codigo_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 codigo (see VARARGIN)

% Choose default command line output for codigo
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes codigo wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

Figura 41. Código inicial de una GUI titulada "codigo": Función 2

La función anterior presenta la característica de no tener argumentos de salida, y es la última función, la que se encarga de devolver las salidas a la línea de comandos.

```
% --- Outputs from this function are returned to the command line.
function varargout = codigo_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

Figura 42. Código inicial de una GUI titulada "codigo": Función 3

3.2.2. Edición de los objetos de una GUI

Todos los objetos que componen una GUI tienen un conjunto de propiedades que pueden modificarse. Para ello, se abre el menú *Property Inspector* haciendo doble click en el elemento que se desee editar.

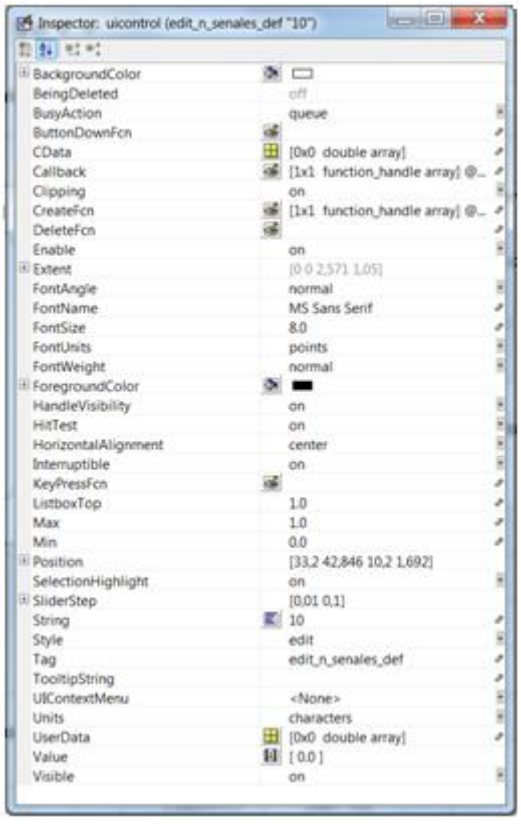


Figura 43. Menú Property Inspector de un Edit Text con etiqueta "edit_n_senales_def"

Las propiedades que se han tenido en cuenta en este proyecto son:

Tag

Es la etiqueta o nombre asociada a cada objeto. Por defecto, cuando se introduce un nuevo objeto se le asigna de forma automática una etiqueta que depende del tipo de elemento. Es recomendable cambiar este nombre por otro que nos ayude a identificarlo rápidamente.

String

Se trata de un cuadro de texto que sólo se encuentra en los menús de propiedades de determinados objetos. Permite visualizar en la pantalla de la interfaz un texto estático, dejar en blanco un cuadro de texto editable que posteriormente será rellenado por el usuario, o mostrar un resultado numérico.

FontSize/FontWeight/FontAngle/FontName

Permite cambiar el formato del texto escrito en el *String*.

3.2.3. Handles

En la creación de una GUI aparece una de las funciones más importantes de nuestro proyecto. Se trata de la función *guidata*, encargada de almacenar datos en una estructura de *handles*. Los *handles* son manejadores asociados a objetos y su contenido permanece mientras se esté ejecutando el programa.

En un principio, en esta estructura sólo se encuentran almacenados los *handles* correspondientes a los objetos de la interfaz, mediante los cuales también se pueden cambiar sus propiedades. Sin embargo, a lo largo de la programación de nuestro código añadiremos a la estructura nuevos componentes.

Las variables que aparecen en cada parte del código no se almacenan en ningún espacio de memoria, y una vez que termina de ejecutarse el *Callback* al que pertenecen, desaparecen. Sin embargo, en nuestro trabajo, todas las funciones comparten datos, por lo que debemos conservar sus valores. Es aquí donde toma importancia la estructura de *handles*. Para que los parámetros queden almacenados, deben guardarse en variables *handles*, y a continuación, actualizar la estructura para que queden grabados sus nuevos valores. Esta actualización se realiza con el comando:

```
guidata(hObject, handles)
```

Los *Callback* en los que aparecen *handles* se cierran con este comando. Así, nos aseguramos de que no se nos olvide actualizar sus valores. Una vez que se cierra la GUI, el valor de todos los *handles* desaparece.

En el código implementado, en vez de trabajar con la estructura *handles*, por simplicidad, al inicio de cada función se crean variables a las que se le asignan el contenido de las variables *handles* con el mismo nombre, y al final de la función, se asigna a los elementos correspondientes de la estructura *handles* los nuevos valores adquiridos por las variables. A modo de ejemplo se muestra el proceso seguido con la variable *Range* del programa:

```
% Inicio de la función
Range=handles.Range;

% Final de la función
handles.Range=Range;

guidata(hObject, handles);
```

3.2.4. Funciones más importantes empleadas en el proyecto

Str2double

La función *Str2double* (String-to-double) convierte un *String* en un valor numérico de doble precisión. Con esta función, los datos introducidos por el usuario en un *Edit Text* se expresan en formato numérico, pudiendo así el resto del código operar con ellos.

Get

El comando *get* devuelve el valor de una propiedad de un objeto de la GUI. Para ello, se especifica el *handle* del objeto y el nombre de la propiedad.

```
handles.num_senales=str2double(get(handles.edit_n_senales_def,'String'
))
```

En la línea de código anterior se indica que el valor guardado en la propiedad *String* del objeto con etiqueta *edit_n_senales_def*, introducido por el usuario, se convierte a formato numérico con la función *Str2double* y se almacena en la variable *handles.num_senales*.

Set

Con el comando *set* se cambia la propiedad de un elemento. Se especifica el objeto a modificar a través de su *handle* y el nombre de la propiedad.

```
set(handles.text_info,'String','Recogida de la señal de calibración');
```

En la línea de código mostrada, se indica que en el *String* del objeto con etiqueta *text.info*, debe aparecer la expresión "*Recogida de la señal de calibración*".

Save

La función *save* permite guardar variables en un archivo con extensión *.mat*. En este proyecto, se emplea para guardar campos de la estructura *handles* como variables individuales. En el comando se especifica el nombre del archivo, se indica que se trata de una estructura y se escribe el nombre de la estructura y de todas las variables que se deseen guardar.

```
save('datos_calibracion','struct','handles','matriz_calibracion');
```

En este ejemplo se indica que se debe crear un archivo con el nombre *"datos_calibracion"* en el que se guardará la variable *matriz_calibración*, perteneciente a la estructura *handles*.

Capítulo 4

4. Comunicación con la tarjeta NI PCI-5152

4.1. Introducción

La tarjeta NI PCI-5152 consiste en un osciloscopio/digitalizador de alta velocidad que permite recoger señales en un ancho de banda analógico de hasta 300 MHz y presenta una velocidad de muestreo de hasta 2 GS/s. A través de ella, se realiza la adquisición de datos de las señales emitidas en el laboratorio, que posteriormente se analizarán con el fin de localizar la fuente de descargas parciales.

Tanto la adquisición como el análisis se realizan en el entorno Matlab y para llevarlo a cabo es necesario establecer una comunicación entre la tarjeta y el software. En el caso de la tarjeta NI PCI-5152 no existe ningún driver específico que establezca una comunicación directa, por lo que hay que recurrir al driver universal de *National Instruments*, denominado NI-Scope.

La comunicación con la tarjeta de adquisición mediante este driver no es sencilla, ya que la documentación que proporciona los conocimientos necesarios para llevarla a cabo no es fácil de localizar. Además, no existe ninguna página que incluya todos los pasos a seguir, sino que hay que acudir a varios enlaces de *Mathworks* para recopilar la información a partir de la cual establecer la secuencia de los mismos. Lo mismo ocurre con el proceso de configuración de la tarjeta, necesario tras la conexión: no es fácil encontrar dónde se explican los parámetros a tener en cuenta y las funciones con las que estos se configuran.

En los siguientes apartados se explican los pasos a seguir para establecer la comunicación entre la tarjeta y el entorno de programación, y las configuraciones que se

requieren. También se describe el proceso de adquisición y cómo se simula el funcionamiento de la tarjeta si no se dispone de ella físicamente.

4.2. Instalación del paquete de apoyo NI-Scope (NI-Scope Support Package)

La comunicación con los osciloscopios NI-Scope se realiza a través de la herramienta Instrument Control Toolbox™, disponible en los paquetes de apoyo de hardware de Matlab (Hardware Support Package). La instalación de esta toolbox incluye la instalación de dos carpetas [25]:

- MATLAB Instrument Driver for NI-Scope support
- National Instruments® driver file: NI-Scope driver version 3.9.7

Para realizar la instalación, hay que abrir el instalador de paquetes de apoyo (Support Package Installer) escribiendo en el *Command Window* de Matlab la instrucción:

>>supportPackageInstaller

En él, se selecciona el paquete denominado NI-Scope.

Ciertas funciones de NI-Scope requieren un compilador C compatible para su ejecución. El compilador empleado por la mayoría de los usuarios es *Microsoft Windows SDK 7.1*. Una vez instalado, hay que elegir el lenguaje de programación que va a emplear (C o Fortran), y esto se configura escribiendo el comando:

>>mex –setup

Pese a que es el compilador más utilizado, y en teoría, debería funcionar correctamente, a la hora de probar el código de este proyecto daba problemas de compatibilidad y no se podía ejecutar el programa. Para evitar esta situación, existe una solución alternativa. A diferencia de la versión de Matlab de 64 bits (primera versión instalada), la de 32 bits incluye en su instalación un compilador C. Por tanto, el error anterior se soluciona instalando directamente esta última versión, independientemente de si el ordenador con el que se trabaja es de 32 o 64 bits.

4.3. Examinar los recursos del hardware

Instalado el paquete de apoyo, se comprueba si se dispone del módulo NI-Scope.

Existe una función llamada *instrwinfo* que permite ver todos los recursos relacionados con el hardware (tarjeta de adquisición) a los que la herramienta Instrument Control Toolbox™ tiene acceso. Esta función proporciona información de las siguientes categorías [26]:

- Información general
- Información de la interfaz
- Información del adaptador
- Información del instrumento
- Información del driver instalado

Para realizar la comprobación de la existencia del módulo NI-Scope, primero se analiza qué drivers están instalados a través del comando *instrwinfo:*

```
>> instrwinfo

                    MATLABVersion: '8.3 (R2014a)'

                    SupportedInterfaces: {1x8 cell}

                    SupportedDrivers: {'matlab'  'ivi'  'vxipnp'}

                    ToolboxName: 'Instrument Control Toolbox'

                    ToolboxVersion: '3.5 (R2014a)'
```

A continuación, se comprueba si NI-Scope forma parte de la lista de módulos del driver 'ivi', driver al que pertenece.

```
>> driversInfo = instrhwinfo ('ivi')

                    LogicalNames: {}

                    ProgramIDs: {}

                    Modules: {1x14 cell}

                    ConfigurationServerVersion: '1.7.0.12115'

                    MasterConfigurationStore: 'C:\ProgramData\IVI
                    Foundation\IVI\IviConfi...'

                    IVIRootPath: 'C:\Program Files\IVI Foundation\IVI\'
```

```
>> Modules= driversInfo.Modules

Modules =   'nisACPwr'   'niScope'   'nisCounter'   'nisDCPwr'   'nisDigitizer'
'nisDmm'   'nisDownconverter'   'nisFGen'   'nisPwrMeter'   'nisRFSigGen'
'nisScope'   'nisSpecAn'   'nisSwtch'   'nisUpconverter'
```

Como se puede observar, se dispone del módulo NI-Scope.

4.4. Comunicación con la tarjeta

Además de la instalación del driver, con el fin de que la comunicación con la tarjeta NI PCI-5152 sea posible, se necesitan crear dos elementos indispensables. Estos elementos son los citados a continuación y deben crearse en el siguiente orden:

- Controlador de instrumentos de Matlab (Matlab Instrument driver)
- Objeto de dispositivo (Device object)

El primero de ellos se genera desde el *Command Window* y su creación sólo se realiza una vez, ya que se guarda como un archivo en la carpeta en la que se está trabajando. El segundo, sin embargo, se debe crear cada vez que se inicie la ejecución del programa, por lo que su instrucción se implementa al principio del *Script* (archivo *.m*) del código del proyecto.

Controlador de instrumentos de Matlab

La herramienta Instrument Control Toolbox™ realiza las comunicaciones de los drivers de instrumentos (*Instrument driver communication*) apoyándose en controladores de instrumentos de Matlab. Estos actúan como una delgada capa que envuelve el driver IVI con el fin de proporcionar una interfaz de controlador compatible en este entorno de programación, y se crean a partir de la información del driver IVI [27].

Puede ocurrir que el driver de Matlab que se requiere venga instalado con el programa. Por ello, es importante comprobar si forma parte de la lista de drivers que vienen instalados por defecto. Para acceder a esta lista basta con escribir las siguientes instrucciones en el *Command Window*:

>>Matlabinfo = instrhwinfo('Matlab')

>>Matlabinfo.InstalledDrivers

En el caso de que no esté incluido en dicha lista, se crea mediante la función *makemid* y debe tener el mismo nombre que el driver IVI. De esta manera, se genera un archivo con

formato *.mdd* que representa el controlador de instrumento de Matlab (*Matlab Instrument driver*).

>>makemid ('niScope')

En la creación de este driver actúa el compilador mencionado anteriormente.

Objeto de dispositivo

A continuación se crea el objeto de dispositivo (*Device Object*). Se trata de una variable de Matlab que, basándose en la información del controlador de instrumentos de Matlab, contiene todas las propiedades y los métodos del instrumento. Dicha información le permite saber cómo tiene que usar el driver IVI para establecer la conexión con la tarjeta.

La creación de este objeto se lleva a cabo con la función *icdevice*, y como argumentos se le debe pasar el nombre del archivo *.mdd* generado y el nombre de la tarjeta de adquisición [28]. El nombre de la tarjeta se consigue a través de la herramienta *Measurement & Automation Explorer*, que se instala de forma automática con el paquete de apoyo NI-Scope. En la lista de programas instalados del ordenador aparece con el nombre *NI MAX*. En la *Figura 44* se muestra la ventana correspondiente a dicha herramienta.

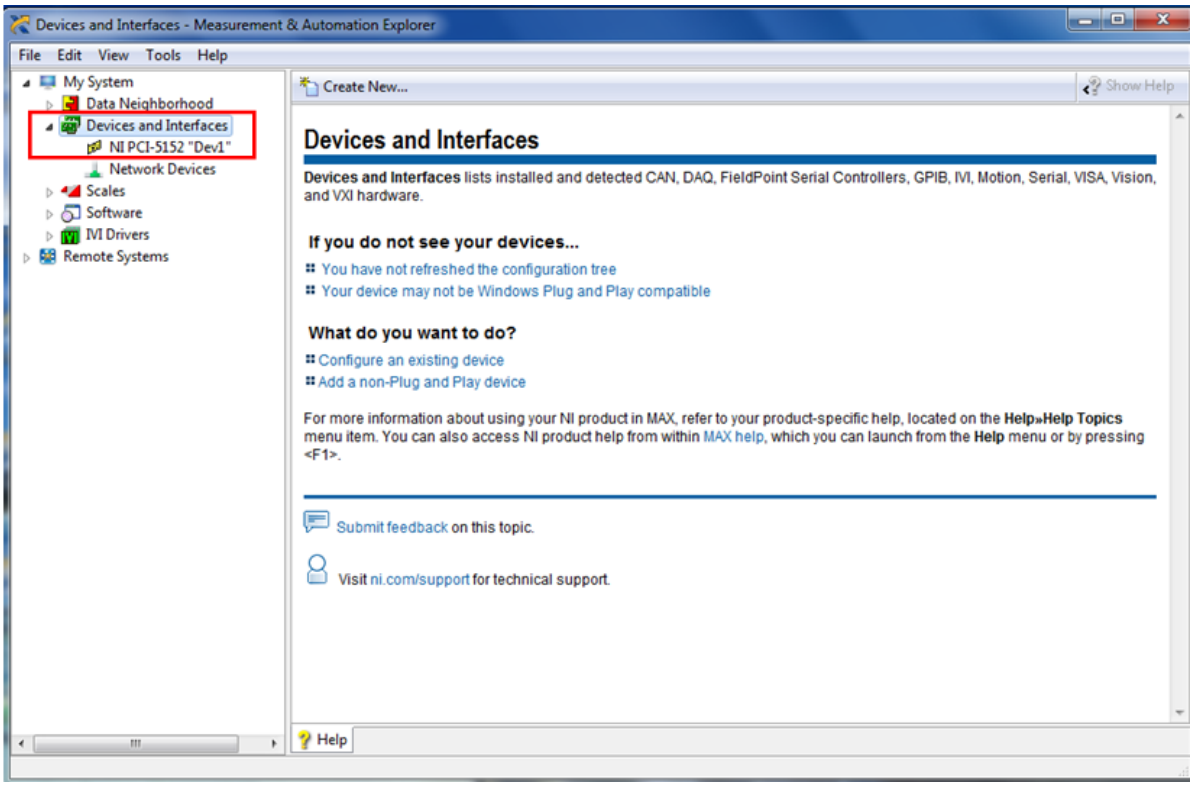


Figura 44. Programa Measurement & Automation Explorer

En la pestaña *"Devices and Interfaces"* se puede apreciar que el nombre correspondiente a la tarjeta de adquisición NI PCI-5152 es "Dev1".

Por tanto, la instrucción con la que se crea el objeto es:

```
ictObj = icdevice('niscope.mdd', 'DAQ::Dev1');
```

Para finalizar, se establece la conexión con la tarjeta:

```
connect(ictObj);
```

Es importante comprobar que la conexión funciona correctamente. Para ello, se visualizan las características del objeto creado y se comprueba el estado de la comunicación. En el momento de la conexión, el estado debe pasar de "*closed*" a "*open*". El objeto se visualiza mediante el comando:

```
disp(ictObj);
```

A continuación se muestran las características del objeto antes y después de la conexión, que se diferencian en el estado de la comunicación:

Antes de la conexión

Instrument Device Object Using Driver: niScope

Instrument Information
- Type: IVIInstrument
- Manufacturer: National Instruments Corp.
- Model: NI Digitizers

Driver Information
- DriverType: MATLAB IVI
- DriverName: niScope
- DriverVersion: 1.0

Communication State
- **Status: closed**

Después de la conexión

Instrument Device Object Using Driver: niScope

Instrument Information
- Type: IVIInstrument
- Manufacturer: National Instruments Corp.
- Model: NI Digitizers

Driver Information
- DriverType: MATLAB IVI
- DriverName: niScope
- DriverVersion: 1.0

Communication State
- **Status: open**

4.5. Test & Measurement Tool

Cuando se genera el objeto de dispositivo (*Device Object*) se crea de forma automática un almacén de configuración IVI, que se explica en detalle más adelante. Este almacén se puede crear también manualmente a través de una herramienta denominada *Test & Measurement Tool*.

Esta herramienta muestra los recursos de los instrumentos (hardware, drivers, interfaces…) y permite configurarlos y comunicarse con ellos. Se emplea para administrar los recursos de la sesión de control del instrumento y se utiliza para [29]:

- Buscar adaptadores instalados
- Examinar hardwares disponibles
- Examinar drivers instalados (controladores)
- Examinar objetos del instrumento

Para acceder a dicha herramienta se escribe la siguiente instrucción en el *Command Window* de Matlab:

>>tmtool

En la *Figura 45* se muestra la ventana correspondiente a dicha herramienta. En ella se puede ver que existen tres tipos de controladores: Matlab Instrument Drivers, VXIplug&play Drivers e IVI. Los dos primeros sólo requieren la presencia de un archivo de controlador para la instalación del driver, mientras que la instalación del último, implica un almacén de configuración IVI [26].

El almacén de configuración IVI tiene la capacidad de adaptarse a distintos modelos de instrumentos, controladores o puertos sin tener que modificar el código diseñado para un modelo específico. Está formado por cuatro componentes que se encargan de identificar las ubicaciones de los instrumentos, los módulos de software a través de los cuales se controlan los instrumentos, y las asociaciones entre los módulos de software y los instrumentos [30].

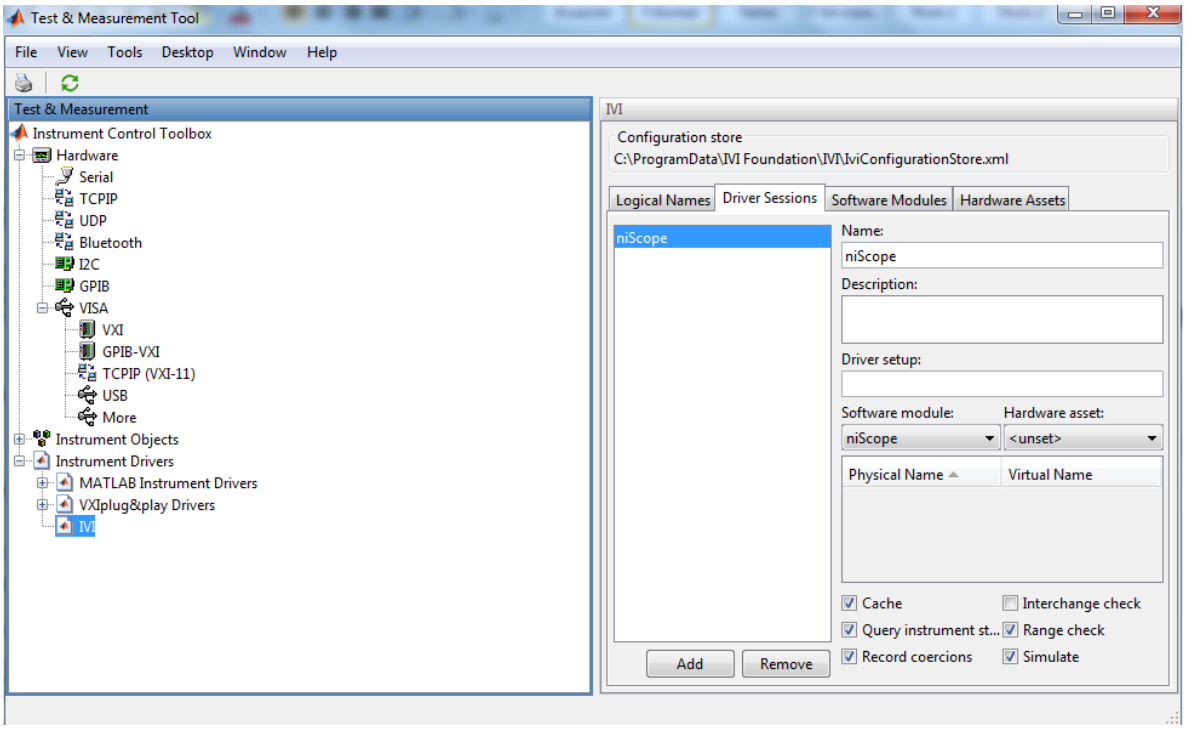


Figura 45. Test & Measurement Tool

En la *Figura 46* se muestra un esquema de un almacén de configuración IVI, seguida de una breve descripción de cada componente.

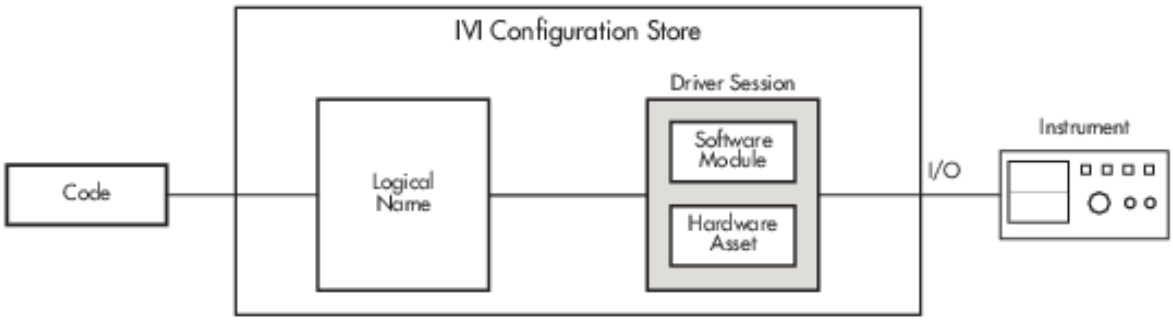


Figura 46. Almacén de configuración IVI [30]

- *Logical name (Nombre lógico).* Proporciona un acceso a la sesión del driver. El código se comunica con el instrumento a través del nombre lógico. Si se cambia de instrumento, basta con modificar este componente.
- *Driver session (Sesión de driver).* Establece una asociación entre el módulo del software y los activos del hardware. Se tiene una sesión de driver diferente para cada instrumento en cada una de sus posibles localizaciones.
- *Software module (Módulos del software).* Es un instrumento específico que contiene los comandos y funciones necesarias para comunicarse con el instrumento.
- *Hardware asset (Activos de hardware).* Identifica el puerto de comunicación conectado al instrumento.

Los pasos a seguir para la creación del almacén de configuración IVI desde la herramienta *tmtool* son:

- **Paso 1.** En el esquema en forma de árbol situado a la izquierda se selecciona:
  Instrument Drivers->IVI
- **Paso 2.** Se pulsa en la pestaña *Driver Session* y se crea una nueva sesión con ayuda del botón *Add*. El nombre de nuestra sesión será "*niScope*".
- **Paso 3.** Se selecciona el módulo del software *"niScope"*.
- **Paso 4.** En la esquina inferior derecha se señalan únicamente las casillas adecuadas para la configuración de la sesión del driver deseada.
- **Paso 5.** Se guardan todos los cambios:
  File->Save IVI Configuration Store

En la *Figura 45* se muestra la configuración de la sesión de driver "*niScope*", que es la requerida para este proyecto.

4.6. Configuración y parámetros

Establecida la conexión con la tarjeta, el siguiente paso es configurar el osciloscopio [28]. En un primer momento, se ajustan de manera automática parámetros como el rango, la frecuencia de muestreo y el nivel de disparo invocando la función de configuración en el modo de "*autosetup"*.

```
configuration = ictObj.Configuration;
 invoke(configuration, 'autosetup');
```

A continuación, se realiza una configuración más detallada en la que se introducen los valores de los parámetros adecuados para nuestro estudio. En este proyecto, es necesario realizar modificaciones en la configuración vertical, horizontal y del trigger.

Las modificaciones se llevan a cabo mediante funciones vinculadas al objeto de dispositivo creado. Primero, se especifica el valor de las variables que componen las funciones, y a continuación, se invocan estas últimas para queden grabados los cambios.

La sintaxis de las funciones se puede ver en la herramienta *Test & Measurement Tool* seleccionando el objeto de dispositivo creado, y accediendo a la ayuda de *National Instruments*, se puede conocer en qué consiste cada una de las variables y los valores que pueden tomar. A ella se puede acceder directamente desde el menú de inicio del ordenador (se instala automáticamente con el paquete de apoyo NI-Scope) seleccionando las siguientes carpetas:

Todos los programas-> National Instruments->NI-SCOPE ->Documentation->

->NI High Speed Digitizers Help

En la ventana que aparece, se seleccionan las siguientes pestañas del esquema de la izquierda:

NI High-Speed Digitizers Help->Programming->Reference->

->NI-SCOPE Function Reference Help->Attributes

En la *Figura 47* se muestra la ventana de la herramienta *Test & Measurement Tool* en la que aparece la sintaxis de las funciones y en la *Figura 48* se muestra la ventana de ayuda de *National Instruments* para NI-Scope.

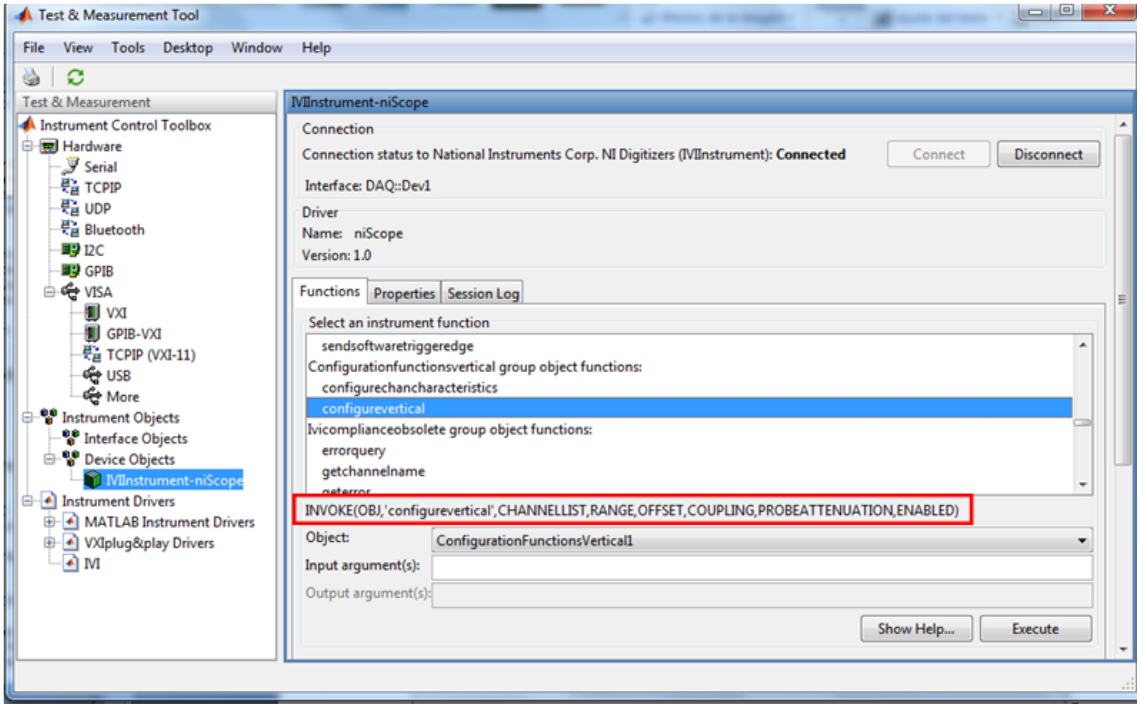


Figura 47. Test & Measurement Tool. Sintaxis de las funciones de las configuraciones de la tarjeta de adquisición

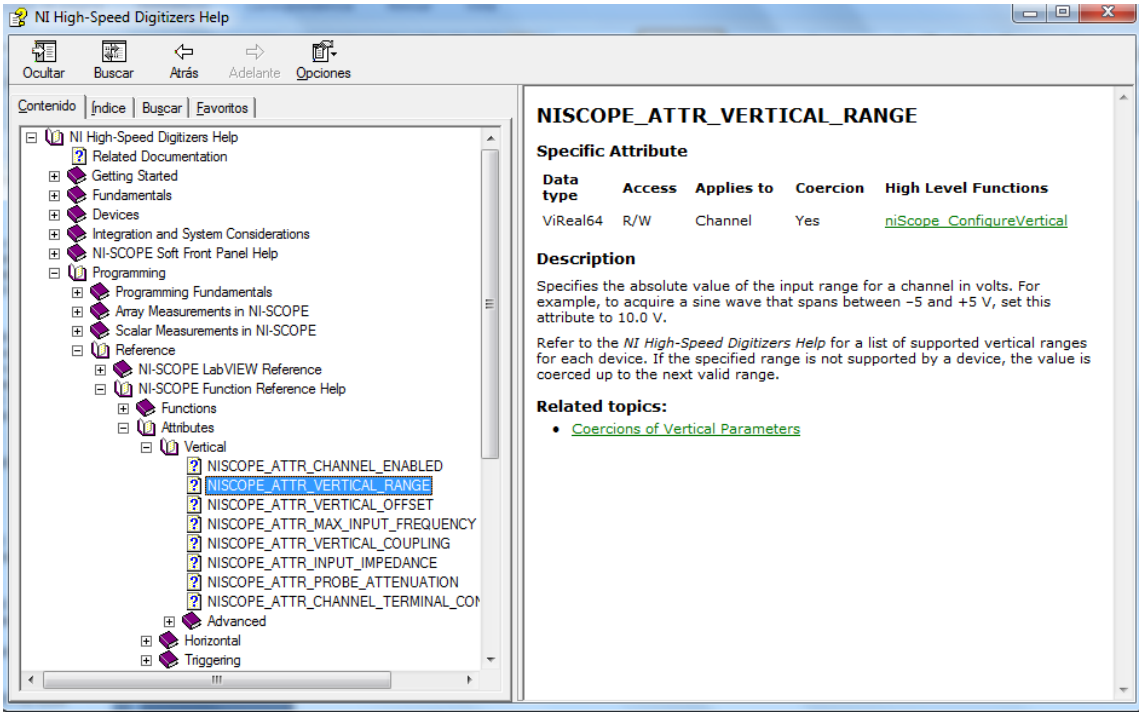


Figura 48. NI High-Speed Digitizer Help. Ayuda de National Instruments para NI-SCOPE

En los siguientes apartados se especifica en primer lugar la sintaxis de cada una de las funciones correspondientes a las configuraciones a realizar, así como los atributos que las componen y todos sus posibles valores. Por último, se justifica el valor seleccionado para nuestro estudio.

4.6.1. Configuración vertical

4.6.1.1. Sintaxis de las funciones y atributos

Para la configuración vertical se requieren dos funciones. La primera contiene las características del canal de entrada de la señal y la segunda, los parámetros verticales del osciloscopio.

Tabla 10. Configuración vertical de la tarjeta de adquisición. Función 1

Configuración vertical	
Función 1	`invoke(ictObj.Configurationfunctionsvertical, 'configurechancharacteristics','0',Inputimpedance,Maxinputfrequency);`
Argumentos de entrada	**Definición y posibles valores**
ChannelList	Es el canal de entrada de la señal. Puede tomar cualquier valor correspondiente a los canales de los que dispone el dispositivo. En nuestro caso la tarjeta dispone de dos canales: canal 0 y canal 1, por tanto, puede tomar el valor 0 ó 1.
Inputimpedance	Indica la impedancia de entrada en Ohmios. Los valores válidos son: 50 Ω, 1000000 Ω (1 MΩ).
Maxinputfrequency	Indica el ancho de banda del canal en Hz. En el caso de que se desee tener el ancho de banda completo su valor es "-1", y si se desea tener el ancho de banda que viene por defecto, su valor es "0". También puede tomar el valor de 20000000 Hz (20 MHz) ó 100000000 Hz (100 MHz).

Tabla 11. Configuración vertical de la tarjeta de adquisición: Función 2

Configuración vertical	
Función 2	`invoke(ictObj.Configurationfunctionsvertical, 'configurevertical', '0', Range, Offset, Coupling, ProbeAttenuation, true);`
Argumentos de entrada	**Definición y posibles valores**
Range	Especifica el valor absoluto del rango de entrada de un canal en Voltios. Mediante él se determina el valor máximo y mínimo de tensión que detecta la tarjeta durante la adquisición El valor máximo de este parámetro depende del rango vertical soportado por el dispositivo de adquisición y puede adquirir cualquier valor comprendido hasta este límite.
Offset	Indica la posición del centro del rango con respecto a tierra en Voltios. Su valor está limitado por el offset soportado por cada dispositivo.
Coupling	Especifica cómo se acopla el digitalizador a la señal de entrada. El acoplamiento puede ser AC, DC o GND, y los valores que toma en cada caso son 0, 1 y 2, respectivamente.
ProbeAttenuation	Indica la atenuación de la señal del canal de entrada. Puede tomar como valor cualquier número real positivo y los valores típicos son 1,10 y 100.
Enabled	Especifica si el digitalizador adquiere o no señales de un canal. Los valores que puede tomar son "*true*" (adquiere datos) o "*false*" (no adquiere datos).

4.6.1.2. Valores de los parámetros de la configuración vertical en nuestra aplicación

- *ChannelList*. La antena se ha conectado al canal 0 de la tarjeta de adquisición, por tanto, a esta variable se le asigna el valor 0.
- *Inputimpedance*. Con el fin de que no se produzcan rebotes de la onda recibida en el cable que comunica la antena con la tarjeta de adquisición (cable coaxial RG-58), la impedancia de entrada del osciloscopio debe ser igual a la impedancia de dicho cable. Puesto que el cable coaxial empleado es de 50 Ω (es el idóneo para altas frecuencias), el valor de la impedancia de entrada también es de 50 Ω.
- *Maxinputfrequency*. El valor seleccionado para este parámetro es "-1". Así, se dispone del ancho de banda del canal completo.
- *Range*. Este parámetro no tiene un valor fijo, se modifica a lo largo de la ejecución del programa. En el código inicial que se ejecuta antes de que se haga visible la interfaz gráfica, cuando se establece la comunicación con la tarjeta, se le asigna un valor igual a 0,01 V. Este valor se trata de un dato empírico. Lo esperable es obtener señales de 0,01 V. Sin embargo, las señales que se desean analizar no son fáciles de recoger, tienen una relación señal-ruido baja, lo que implica que este rango no siempre sea el más adecuado. Pese a ello, se considera que en esta primera adquisición las señales tienen ese rango inicial. En el resto del programa su valor se va modificando en función de las características de las señales recogidas en cada posición y es distinto en cada una de ellas. Su valor se

calcula de forma automática a partir de las señales adquiridas en una recogida previa, así se asegura que se adquiere la señal de interés en su totalidad y se detectan todos los valores de tensión por los que está compuesta. El proceso que se sigue para su cálculo es el siguiente: en cada ángulo se recogen cinco señales previas y se guarda en un vector el mayor pico de tensión de cada una de ellas. A continuación, se analiza cuál es el máximo valor guardado en dicho vector y se calcula el rango final, que es un 40 % superior a este último.

- *Offset*. Se le asigna un valor igual a 0 debido a que las señales que se adquieren no tienen una componente continua.
- *Coupling*. Se establece un acoplamiento de la señal DC porque la señal de interés no tiene componente continua. Por ello, se asigna a esta variable el valor 1.
- *ProbeAttenuation*. A esta variable se le asigna un valor igual a 1. De esta manera, se mantienen los niveles de tensión que se recogen, no se amplifica la señal.
- *Enabled*. Se le asigna el valor "*true*" para que se adquieran señales del canal.

A continuación, se muestran las líneas de código pertenecientes a esta configuración con los valores de los parámetros asignados en nuestro estudio.

```
%  ** Configuración vertical: canal 0 **
Range = 0.01;
Offset = 0;
Coupling = 1;
ProbeAttenuation = 1;
Inputimpedance=50;
Maxinputfrequency=-1;

invoke(ictObj.Configurationfunctionsvertical, 'configurevertical',
'0', Range, Offset, Coupling, ProbeAttenuation, true);
invoke(ictObj.Configurationfunctionsvertical,'configurechancharacteris
tics','0',Inputimpedance,Maxinputfrequency);

% NOTA: El true corresponde a la variable enabled
```

Figura 49. Código de la configuración vertical de la tarjeta de adquisición

4.6.2. Configuración horizontal

4.6.2.1. Sintaxis de la función y atributos

En este apartado se indica la función y los parámetros correspondientes a la configuración horizontal.

Tabla 12. Configuración horizontal de la tarjeta de adquisición

Configuración horizontal	
Función	`invoke(ictObj.Configurationfunctionshorizontal, 'configurehorizontaltiming',MinSampleRate,MinNumPts,RefPosition, NumRecords,EnforceRealTime);`
Argumentos de entrada	**Definición y posibles valores**
MinSampleRate	Especifica la frecuencia de muestreo en $nº\ de\ muestras\ (Samples)/segundo$. Estas unidades se corresponden con Hz. Su valor depende del digitalizador, cada modelo presenta una velocidad máxima de muestreo determinada.
MinNumPts	Especifica el número mínimo de puntos que se recogen en cada señal. Puede adquirir valores comprendidos entre uno y la memoria interna disponible.
RefPosition	Indica la posición en la que empieza a recogerse la señal después del disparo. Este parámetro permite visualizar cómo es la señal antes de que se recojan sus valores. Se expresa como un % de la longitud de la onda y puede tomar valores entre 0 y 100.
NumRecords	Especifica el número de grabaciones que se adquieren. Su valor depende del número de grabaciones que se deseen recoger.
EnforceRealTime	Indica si las medidas son tomadas en tiempo real o en tiempo equivalente. Los valores que puede tomar esta variable son 1 (tiempo real) ó 0 (tiempo equivalente).

El osciloscopio calcula el tiempo de adquisición a partir de la frecuencia de muestreo (*MinSampleRate*), que es un parámetro fijo, y el número de muestras, que es determinado por el usuario, y la expresión que utiliza es:

$$Tiempo\ de\ adquisición = \frac{nº\ muestras}{frecuencia\ de\ muestreo\ [Hz]}\ \ [s] \qquad [\ 1\ ]$$

4.6.2.2. Valores de los parámetros de la configuración horizontal en nuestra aplicación

- *MinSampleRate*. Pese a que la frecuencia de muestreo máxima de la tarjeta NI PCI-5152 es de 2 GS/s, a la hora de realizar los ensayos, se observó que para este modelo el driver no permitía la adquisición en estas condiciones. Por este motivo, la adquisición se ha llevado a cabo con una velocidad de muestreo igual a 1 GS/s.

- *Tiempo de adquisición*. Sabemos que un tiempo igual a 1 μs es suficiente para recoger en su totalidad las señales a analizar en el laboratorio. Por ello, se ha establecido como tiempo de adquisición 1 μs.
- *MinNumPts*. Este parámetro lo definimos a partir de la frecuencia de muestreo y el tiempo de adquisición, que valen 1 GS/s y 1 μs, respectivamente. Despejando de la ecuación [1] se determina que su valor es igual a 1000.
- *RefPosition*. Interesa empezar a recoger la señal en alguna posición posterior a la que se produce el disparo. Así, podemos conocer la señal que existía antes de la onda recogida o ver si antes del impulso registrado en la adquisición se habían producido otros impulsos. Sin embargo, su valor no debe ser elevado, ya que si no, no se registrarían los picos de tensión correspondientes al fenómeno de la descarga parcial. Teniendo en cuenta esta idea, se ha asignado a este parámetro un valor igual al 20 %.
- *NumRecords*. El número de grabaciones indicado en la tarjeta es igual a 1. En realidad, en cada posición se recogen varias señales, pero en vez de darle la orden a la tarjeta de que adquiera un determinado número de grabaciones, estas se adquieren con un bucle *for* en el que en cada ciclo sólo se recoge una señal. En un principio, se optó por la primera opción, pero la tarjeta no era capaz de responder a esta instrucción, por lo que se eligió el segundo método.
- *EnforceRealTime*. Como las medidas se desean tomar a tiempo real, se le asigna el valor de 1.

A continuación, se muestran las líneas de código pertenecientes a esta configuración con los valores de los parámetros asignados en nuestro estudio.

```
% ** Configuración horizontal: canal 0 **

MinSampleRate=1e9;
MinNumPts=1000;    % Mediante este número y la frecuencia de muestreo
                   % se determina el tiempo de adquisición
RefPosition=20;
NumRecords=1;
EnforceRealTime=1;

invoke(ictObj.Configurationfunctionshorizontal,'configurehorizontaltim
ing',MinSampleRate,MinNumPts,RefPosition,NumRecords,EnforceRealTime);
```

Figura 50. Código de la configuración horizontal de la tarjeta de adquisición

4.6.3. Configuración del trigger

4.6.3.1. Sintaxis de la función y atributos

En este apartado se indica la función y los parámetros correspondientes a la configuración del trigger.

Tabla 13. Configuración del trigger de la tarjeta de adquisición

Configuración del trigger	
Función	`invoke(ictObj.Configurationfunctionstrigger,'configuretriggeredge', TriggerSource,Level,Slope,TriggerCoupling,HoldOff,Delay);`
Argumentos de entrada	**Definición y posibles valores**
TriggerSource	Especifica la fuente del disparo. Varía según el modelo del dispositivo de adquisición. Puede tomar cualquier valor correspondiente a los canales de los que dispone el dispositivo, en nuestro caso, 0 ó 1, o una de las siguientes opciones: VAL_EXTERNAL, VAL_IMMEDIATE, VAL_RTSI_0 – VAL_RTSI_6, VAL_PFI_0 – VAL_PFI_2, VAL_PXI_STAR, VAL_SW_TRIG_FUNC.
Level	Indica el nivel de disparo en Voltios. Es el umbral de tensión a partir del cual se inicia la recogida de señales. El valor elegido debe cumplir la siguiente condición: $Trigger\ level \leq (Vertical\ Range/2) + Vertical\ Offset$
Slope	Indica si el disparo se produce en la subida o bajada de tensión. Puede tomar el valor "1" (subida) ó "0" (bajada).
TriggerCoupling	Define cómo se acopla el digitalizador a la fuente de disparo. El rango de valores que puede tomar son 0 (AC), 1 (DC), 2 (HF_REJECT), 3 (LF_REJECT) ó 1001 (AC_PLUS_HF_REJECT). Los más comunes son acoplamiento 0 (AC) y 1 (DC)
HoldOff	Es el tiempo en segundos que el digitalizador espera después de detectar un disparo para activar el subsistema que detecta el siguiente disparo. Este parámetro sólo se tiene en cuenta cuando se adquieren varias grabaciones, y puede tomar cualquier valor comprendido entre 0 y 171,8 segundos.
Delay	Es el tiempo que transcurre desde que se detecta el disparo hasta que se empieza a recoger la señal. Se expresa en segundos y es determinado por el usuario.

4.6.3.2. Valores de los parámetros de la configuración del trigger en nuestra aplicación

- *TriggerSource*. La fuente del disparo en nuestro caso es el canal de la señal de entrada. Por tanto, su valor es 0.
- *Level*. El valor del trigger es igual a 0,005 V y se ha establecido por ensayo y error en el laboratorio. Este nivel de disparo permite ignorar el ruido del laboratorio, ya que se considera que este posee niveles de tensión inferiores a 0,005 V, y por tanto, por encima de él lo que se coge es señal. Pese a ello, puesto que las señales que se analizan tienen una relación baja con el ruido, son difíciles de coger, y es muy probable que se adquieran también ondas de ruido.

- *Slope*. Como las descargas parciales se caracterizan por una elevada subida de tensión, el disparo debe producirse en la subida. Por ello, este parámetro es igual a 1.
- *TriggerCoupling*. Se establece un acoplamiento DC porque la señal de interés no tiene componente continua. Por ello, se asigna a esta variable el valor 1.
- *HoldOff*. Puesto que este parámetro sólo se tiene en cuenta cuando se adquieren varias grabaciones, en nuestro caso toma el valor de 0.
- *Delay*. Con el fin de que se recoja señal una vez que se detecte el disparo se le asigna el valor 0.

A continuación, se muestran las líneas de código pertenecientes a esta configuración con los valores de los parámetros asignados en nuestro estudio.

```
% ** Configuración del trigger: canal 0 **
TriggerSource='0'; % Es el canal de entrada
Level=0.005;
Slope=1;
TriggerCoupling=1;
HoldOff=0;
Delay=0;

invoke(ictObj.Configurationfunctionstrigger,'configuretriggeredge',
TriggerSource,Level,Slope,TriggerCoupling,HoldOff,Delay);
```

Figura 51. Código de la configuración del trigger de la tarjeta de adquisición

4.7. Preparación de la información de la señal

Antes de realizar la primera adquisición de datos (en el código inicial de la interfaz), se configura la información de la señal [28]. En una estructura, se guarda la información relativa a tiempos, número de muestras, ganancia y offset, que definen la adquisición de la onda y se inicializa a cero. Los valores de esta estructura se actualizarán automáticamente en función de las señales recogidas por la tarjeta durante las adquisiciones llevadas a cabo en el resto del programa.

En el caso de que se trabajase con varios canales, habría que definir una estructura para cada canal.

En la *Tabla 14* se muestran las variables que componen la estructura con la información de la señal, y debajo, el código correspondiente a la inicialización de la estructura.

Tabla 14. Estructura con la información de la señal

Estructura con la información de la señal	
Variables	**Definición**
absoluteInitialX	Tiempo en segundos correspondiente a la primera muestra de la señal.
relativeInitialX	Tiempo transcurrido en segundos desde que se produce el disparo hasta que se toma la primera muestra de la señal.
xIncrement	Tiempo en segundos entre dos muestras consecutivas de la señal adquirida.
actualSamples	Número de muestras de la onda adquirida.
offset	Factor de desplazamiento del canal utilizado
gain	Factor de ganancia del canal utilizado
reserved	Variable reservada. No se usa.

```
% Se prepara la información de la señal
numChannels = 1;
channelList = '0';
numSamples = 1000;


    waveformInfo.absoluteInitialX = 0;
    waveformInfo.relativeInitialX = 0;
    waveformInfo.xIncrement = 0;
    waveformInfo.actualSamples = 0;
    waveformInfo.offset = 0;
    waveformInfo.gain = 0;
    waveformInfo.reserved1 = 0;
    waveformInfo.reserved2 = 0;
```

Figura 52. Código de la preparación de la información de la señal a adquirir

4.8. Adquisición de datos con la tarjeta

En este apartado se desarrolla el código que indica a la tarjeta que debe empezar a adquirir señales. Este proceso se va a llevar a cabo en varias de las funciones que componen el programa: código inicial que se ejecuta antes de hacerse visible la interfaz gráfica, calibración horizontal de la tarjeta y recogida de señales en cada una de las posiciones recorridas por el servomotor. Estas dos últimas funciones se explican detalladamente en el capítulo *Interfaz gráfica diseñada*.

En el código inicial, es necesario crear una variable denominada *Timeout*, que especifica el tiempo disponible en segundos para empezar a detectar señales. Este parámetro es muy importante, ya que transcurrido este período de espera, si el programa no ha recibido ninguna señal, interrumpe la ejecución y se lo comunica al usuario mediante un error. En este proyecto, en un principio, se le asignó el valor de 10 s, pero durante los ensayos, se tuvo que ampliar a 30 s porque había posiciones en las que se superaba este tiempo de espera sin haber detectado ninguna señal.

Definida esta variable, se explica el proceso de adquisición. En primer lugar, se inicializa una variable denominada *waveformArray*, que es un vector de dimensiones $1xn$ (n es el número de muestras recogidas de la señal) en el que se almacenan todos los valores de tensión que componen la onda recogida [28]. Como se había definido en la tarjeta un número de grabaciones igual a uno, en él sólo se almacena una única señal. Si el número de grabaciones fuese superior a uno, se almacenarían en el mismo vector todas las señales, una detrás de otra, y su tamaño sería $1xs$, siendo s el número de grabaciones por el número de muestras. Por último, se extrae y guarda la onda en el vector *waveformArray* y se actualizan todos los campos de la estructura correspondiente a la información de la señal, *waveformInfo*.

Con el fin de trabajar con una variable fácil de identificar en el programa, la señal recogida se almacena en un vector denominado *senal*, que tiene las mismas dimensiones que el vector *waveformArray*.

En la siguiente figura se muestran las líneas de código que permiten la adquisición a través de la tarjeta.

```
% Inicialización de las variables

waveformArray = zeros(numChannels * numSamples,NumRecords );
TimeOut = 30; % Expresado en segundos


% Adquisición de la señal

    invoke(ictObj.Acquisition, 'initiateacquisition');
    [waveformArray, waveformInfo] = invoke(ictObj.Acquisition,
'fetch', channelList,...
    TimeOut, numSamples, waveformArray, waveformInfo);
    senal=waveformArray;
```

Figura 53. Código de la adquisición de datos

4.9. Desconexión

Finalizadas todas las operaciones y concluido el estudio realizado con el programa, es imprescindible desconectar y eliminar el objeto de dispositivo creado (*Device Object)* [28]. Este objeto sólo existe durante la ejecución del código, por lo que se crea al inicio de las instrucciones y se elimina al terminar el estudio.

La desconexión se lleva a cabo mediante los siguientes comandos:

```
disconnect(ictObj)
delete(ictObj)
```

Figura 54. Código de la desconexión del objeto de dispositivo (Device Object)

4.10. Simulación de dispositivos NI-DAQmx

En el desarrollo del proyecto, la tarjeta NI PCI-5152 es un elemento fundamental. En todo momento, ha sido necesario disponer de ella para probar el código y comprobar si las instrucciones implementadas realizaban exactamente las operaciones que se pretendían. Sin embargo, se trata de un dispositivo muy caro y sólo se podía hacer uso de él en el Laboratorio de Alta Tensión de la Universidad Carlos III de Madrid, lugar en el que se han efectuado todos los ensayos y mediciones.

Por suerte, el programa *Measurement & Automation Explorer* (*NI MAX*) permite simular la tarjeta y trabajar con ella aunque no se disponga del dispositivo físico, lo que ha facilitado mucho la elaboración de la aplicación en cuanto a tiempo se refiere, ya que se ha podido trabajar fuera del laboratorio sin la limitación de los horarios de acceso al mismo.

Los pasos a seguir para la creación del dispositivo simulado son [31]:

1. Se abre *Measurement & Automation Explorer* (NI MAX).
2. Se selecciona *My system->Devices and Interfaces* y se pulsa en la opción *Create new*.
3. En la siguiente ventana, se selecciona *Simulated NI-DAQmx Device or Modular Instrument*, y a continuación, la tarjeta que se desea simular. En nuestro caso la tarjeta se encuentra dentro de la pestaña *High Speed Digitizers.*

Es importante resaltar que, a la hora de crear el objeto de dispositivo (*Device Object*) en el código del programa, hay que indicar si se va a trabajar en el modo simulación. Para ello, se sustituye el comando que se había especificado en el apartado "*Comunicación con la tarjeta*" por la siguiente instrucción:

```
ictObj=icdevice('niscope.mdd','DAQ::Dev1','OptionString',
                    'simulate=true');
```

En el programa *NI MAX* los dispositivos simulados se diferencian de los reales en que aparecen en color amarillo en vez de en color verde.

Capítulo 5

5. Movimiento automático de la antena mediante un servomotor

La función de la placa *Arduino* en este proyecto consiste en controlar el movimiento del servomotor encargado de enfocar la antena en distintas direcciones. El control del servo se consigue con los pasos explicados a continuación [32]:

Paso 1. Conexiones.

Se establece la conexión eléctrica entre el servo y la placa. En la *Tabla 15* se muestra la conexión correcta.

Tabla 15. Conexiones entre el servo Hitec HS-422 y la placa Arduino Uno

Conexión entre servo y *Arduino Uno*	
Cable del servo Hitec HS-422	**Patilla de la placa *Arduino Uno***
Alimentación (Rojo)	5 V
Tierra (Negro)	GND
Señal de control (Amarillo)	Pin 4

Paso 2. Creación de un "objeto Servo" y calibración del motor.

En primer lugar, se crea un "objeto Arduino" y se incluye la librería del servo en el *Script* (archivo *.m*) del programa. Para ello, en la creación del objeto, se especifica el puerto del ordenador al que está conectado el *Arduino*, el modelo de la placa utilizada y el nombre de la librería del servo.

El nombre de los dos primeros parámetros se consigue accediendo a las siguientes ventanas del ordenador:

Panel de control->Sistema y seguridad->Sistema->Administrador de dispositivos

En esta última (*Figura 55*), la pestaña *Puertos (COM y LPT)* contiene el nombre de los elementos indicados.

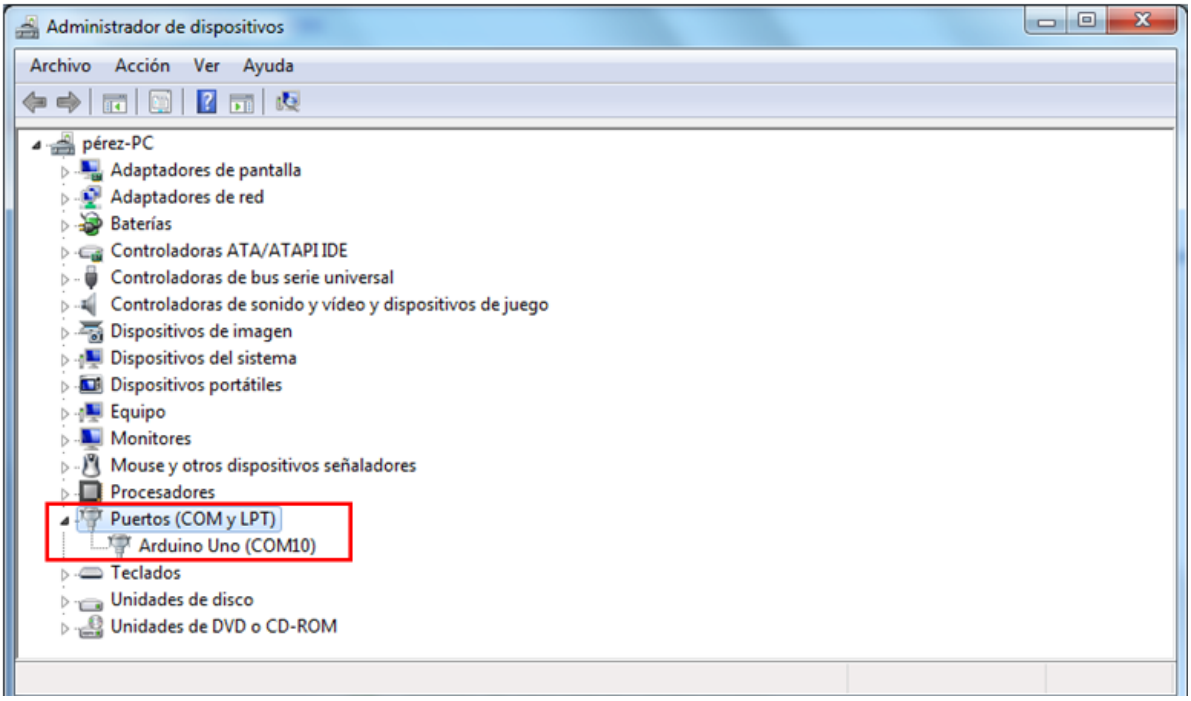


Figura 55. Identificación del puerto correspondiente y nombre de la placa Arduino

A continuación, se crea un "objeto Servo" especificando la variable en la que se ha guardado el "objeto Arduino" y el número de la patilla de la placa en la que está conectada la señal de control del servomotor.

Por último, se calibra el motor para que gire en un rango determinado, en nuestro caso, de 0º a 180º, indicando los anchos de pulso correspondientes. La variable "objeto Servo" creada es muy probable que tenga unos valores de ancho de pulso distintos a los que aparecen en la hoja de características de nuestro servo, aunque puede darse el caso de que coincidan. Por este motivo, se realiza una comprobación de sus valores escribiendo el nombre de dicha variable en el *Command Window* de Matlab, tal y como se muestra a continuación:

```
>> a=arduino ('com10','uno','Libraries','Servo');

>> s=servo (a,4)

s =

        Servo with properties:

                    Pins: 4
  MinPulseDuration: 5.44e-04 (s)
  MaxPulseDuration: 2.40e-03 (s)
```

Si no coinciden, se elimina el "objeto Servo" y se crea uno nuevo especificando, además del número del pin, el ancho de pulso mínimo y máximo para que gire el rango de posiciones requerido. Puesto que nuestro servo presenta un ancho de pulso mínimo y máximo de 600 μsec y 2400 μsec, respectivamente, los valores no coinciden con los proporcionados por Matlab y por tanto, es necesario llevar a cabo este último paso.

El proceso descrito en el paso 2 se muestra en las siguientes líneas de código.

```
% Se crea el "objeto Arduino" y se incluye la librería del Servo

a=arduino('com10','uno','Libraries', 'Servo');

% Se crea un "objeto Servo" inicial y se comprueba su rango de ancho
de pulsos

s=servo(a,4)

% Se elimina el objeto anterior si su rango no se corresponde con el
del servo empleado en el proyecto: Servo Hitec HS-422

clear s;

% Se crea un nuevo "objeto Servo" con el ancho de pulso mínimo y
máximo especificado en la hoja de características del Servo
Hitec HS-422

s = servo(a, 4, 'MinPulseDuration', 0.6e-3, 'MaxPulseDuration', 2.4e-
3)
```

Figura 56. Código en Matlab: Creación de un "objeto Servo" y calibración del motor

Paso 3. Escritura y lectura de la posición del servo.

El movimiento se configura de tal manera que sea automático. Para ello, se emplea un bucle *for* en el que se indica la posición inicial, la posición final y el paso.

Estos tres parámetros en el programa no se indican en grados, se expresan en por unidad tomando como base 180º y la ecuación que se emplea para expresarlos en por unidad es:

$$ángulo\ (p.u.) = \frac{ángulo\ (grados)}{180º} \qquad [2]$$

Así, la posición inicial se corresponde con el valor 0 y la posición final con el valor 1.

El bucle *for* simplemente se encarga de calcular los ángulos en cada momento. Para que el servo se dirija hacia el nuevo ángulo hay que escribir sobre él su nueva posición. Si, además, queremos conocer a qué posición se ha movido exactamente, hay que añadir en el código un comando de lectura. A continuación, se muestran las instrucciones que componen el paso 3.

```
% Movimiento automático: En el bucle for se indica la posición
inicial, el paso y la posición final (expresadas en p.u.) en este
orden.

for angle = 0:10/180:1    % El paso indicado en este caso es de 10°

    % Se escribe en el servo la nueva posición a la que se tiene que
    % mover

       writePosition(s, angle);

    % Se lee la posición (ángulo) en el que se encuentra

       current_pos = readPosition(s);

    % El servo no trabaja en grados. Si queremos saber el ángulo en
    % grados se multiplica por 180°

       current_pos = current_pos*180;

    % Se muestra cada posición por pantalla

       fprintf('La posición actual del servomotor es %d grados\n',
current_pos);

    % Mantenemos el servo parado en cada ángulo durante dos segundos

       pause(2);
end
```

Figura 57. Código en Matlab: Escritura y lectura de la posición del servo

Paso 4. Limpieza de las variables.

Una vez que no se requiera la conexión con el servo, se eliminan los dos objetos creados: “objeto Arduino” y “objeto Servo”. Si no se lleva a cabo esta operación y se desconecta directamente, la próxima vez que se desee repetir el proceso anterior, dará

problemas. Pese a que los objetos anteriores no permanecen guardados en Matlab, el programa no dejará comunicarse con el dispositivo porque detectará que existe otro elemento conectado a él. Por esta razón, este último paso se convierte en el más importante de todos los mostrados, y la instrucción que se encarga de eliminar los objetos es:

```
% Se eliminan los dos objetos creados

clear s a
```

Figura 58. Código en Matlab: Limpieza de las variables

Capítulo 6

6. Montaje experimental

En los capítulos anteriores se ha explicado de forma detallada cómo establecer las conexiones de cada uno de los elementos del sistema de forma separada. En este apartado, se va a describir el montaje final de todo el conjunto. En la *Figura 59* se muestra el esquema de todo el sistema.

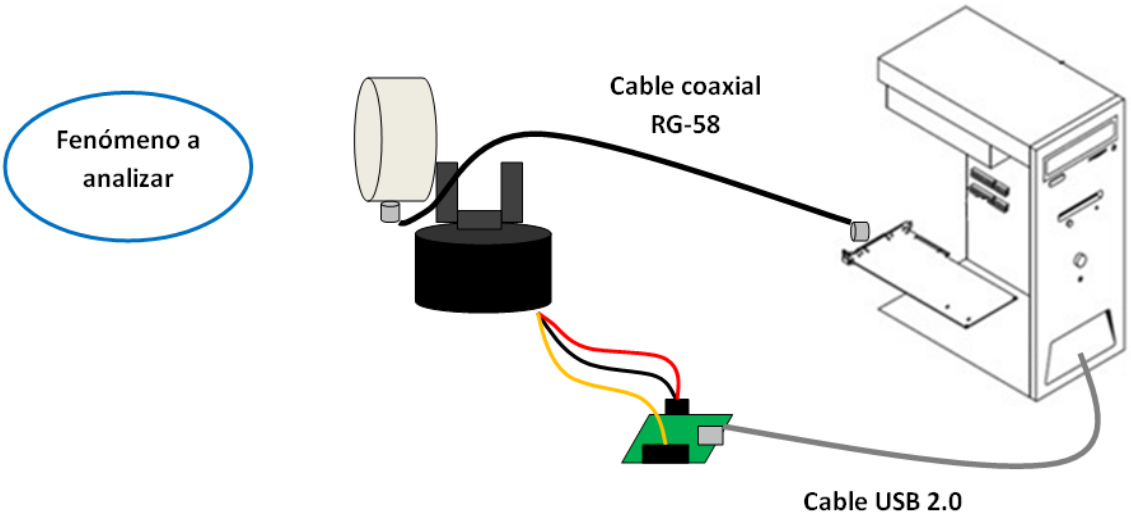


Figura 59. Esquema del montaje de todo el sistema

El montaje de todo el conjunto se puede dividir en dos fases. En primer lugar, se va a desarrollar el montaje de la antena con el servomotor, y por último, la disposición y el cableado del resto de componentes.

Como se ha mencionado con anterioridad, la función del servomotor es mover de forma automática una antena destinada a la detección de descargas parciales. El servomotor en sí no está preparado para soportar directamente la antena, es necesario construir una

estructura sobre la cual se ensamble el dispositivo de recogida de señales. Todos los fabricantes de servomotores ponen a disposición de los clientes accesorios compatibles que permiten diseñar diversas estructuras. Para nuestra aplicación, la estructura a fabricar es muy básica, sólo requiere de dos accesorios: un kit de base giratoria y un brazo de aluminio "C" con rodamientos de bolas.

El servomotor se atornilla a la parte interna de la base giratoria, que sirve de apoyo de la masa a mover. Sobre este soporte se acopla un brazo cuya función es ensamblar la antena a la estructura.

En la *Figura 64* se muestran todos los componentes empleados en este submontaje[4] y en la *Figura 65* , la estructura completa.

Figura 60. Servo Hitec HS-422

Figura 61. Kit de base giratoria

Figura 62.Brazo de aluminio "C" con rodamientos de bolas

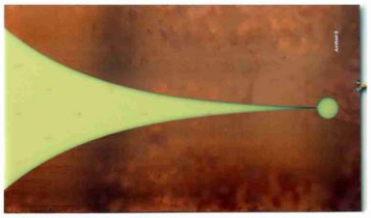


Figura 63. Antena Vivaldi

Figura 64. Componentes del submontaje servomotor-antena

[4] Las fotos de la memoria se han realizado en distintos períodos del desarrollo del proyecto. A lo largo de toda la investigación, se han empleado diversos tipos de antenas. Por ello, en las fotos del montaje no siempre se aprecia la misma antena.

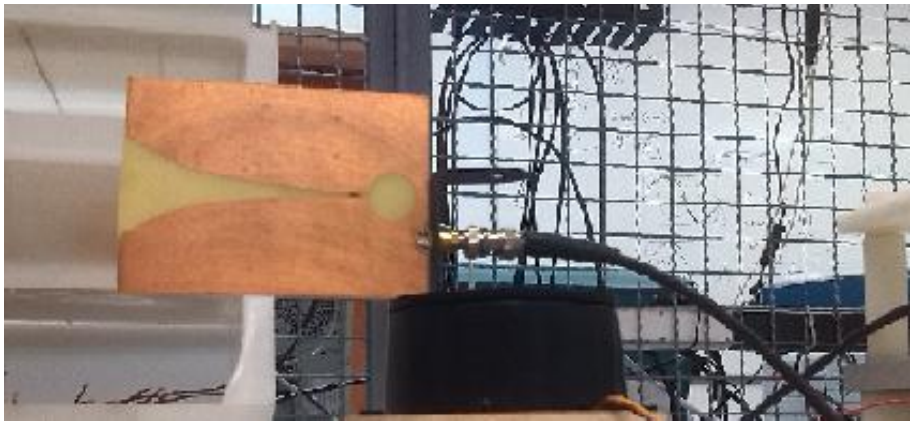


Figura 65. Submontaje servomotor-antena

Terminado el submontaje servomotor-antena, se procede a la conexión de todo el conjunto. Por un lado, se conectan los tres cables del servomotor a los pines correspondientes de la placa *Arduino*, y esta se comunica con el ordenador mediante un cable USB 2.0. Por otro lado, se conecta la antena a la tarjeta de adquisición NI PCI-5152 situada en la placa base del PC mediante un cable coaxial RG-58. Realizadas estas conexiones, ya es posible iniciar el giro automático y la adquisición de datos.

Este sistema se ha instalado en el Laboratorio de Alta Tensión de la Universidad Carlos III de Madrid, donde se ha comprobado su funcionamiento.

Tal y como se ha especificado el montaje, a la hora de ponerlo en marcha presentó ciertos problemas que se fueron solucionando a medida que fueron surgiendo.

En el primer ensayo, se observó que la antena no era lo suficientemente ligera como para que la base giratoria pudiese soportarla sin ningún problema. Conforme iba girando, esta base tambaleaba de un lado a otro. Como solución, se optó por fijarla a otra base con mayor peso. Con el fin de ahorrar gastos, se empleó una tabla de madera de la que se disponía, pero se podría haber empleado cualquier otro material que cumpliese con el objetivo.

Esta medida solucionó el problema pero apareció un nuevo inconveniente. Las dimensiones de la tabla eran superiores a las de la base giratoria, y como consecuencia, el cable coaxial conectado a la antena rozaba con la madera, lo que dificultaba el movimiento. La solución planteada en esta ocasión consistió en desplazar la base giratoria a uno de los extremos de la tabla de madera, pero no tuvo éxito, por lo que se optó por elevar la estructura de la antena. Para ello, se colocó bajo la base giratoria otro soporte de madera con la misma forma que la base del servomotor pero de diámetro algo superior para poder atornillarla. La altura necesaria para que no se produjese el roce se alcanzó con dos tablas de madera circulares.

En la *Figura 66* se muestran las soluciones aplicadas al submontaje servomotor-antena.

Figura 66. Soluciones aplicadas al submontaje servomotor-antena

El montaje final de todo el sistema se muestra en la *Figura 67*.

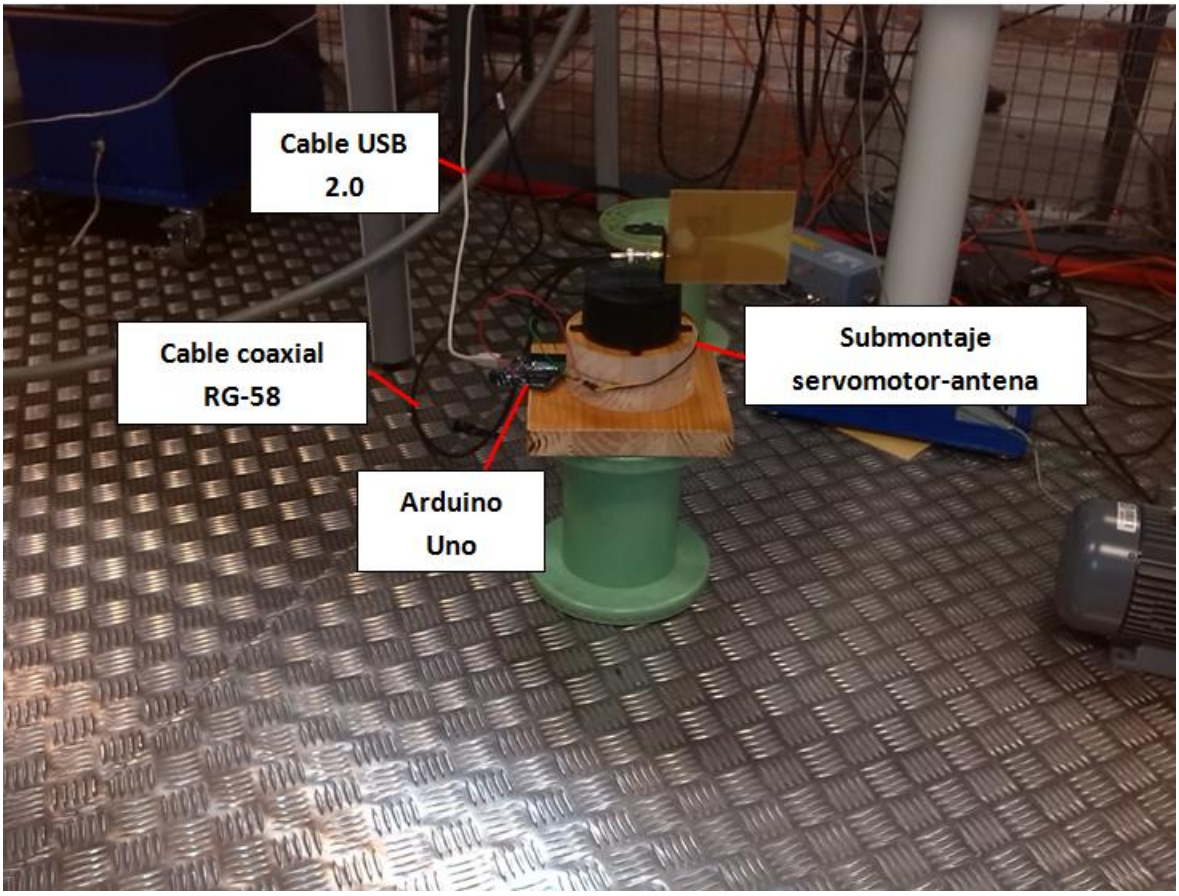


Figura 67. Montaje de todo el sistema

Capítulo 7

7. Interfaz gráfica diseñada

El diseño e implementación de la interfaz gráfica está orientado a la localización de descargas parciales mediante una antena de alta directividad. Esta interfaz proporciona la comodidad de localizar el fenómeno de manera automática, además de realizar el estudio de manera interactiva. El usuario puede modificar el valor de determinados parámetros a través de la pantalla y controla en todo momento qué operación quiere que se esté ejecutando: calibración, inicio o interrupción de la adquisición de datos en todas las posiciones, o desconexión del sistema, seleccionando el botón correspondiente.

El funcionamiento general de la interfaz consiste en:

- Calibración de los parámetros de adquisición de las señales. Se determinan las condiciones del proceso de adquisición.
- Adquisición de las señales emitidas por el fenómeno de descargas parciales. Se recogen y almacenan todas las señales detectadas en cada una de las posiciones en las que se sitúa el servomotor.
- Visualización de los pulsos. Se muestran a tiempo real las señales que se están adquiriendo.
- Cálculo de la energía y análisis estadístico de los picos máximos de tensión de las señales. Se realizan estos dos estudios con el fin de localizar el fenómeno de descargas parciales.
- Representación de la energía en función del ángulo. Se muestran los resultados del estudio de la energía.
- Representación del análisis estadístico en función del ángulo. Se muestran los resultados del estudio de los picos máximos de tensión.
- Desconexión de *Arduino* y de la tarjeta de adquisición. Finalizado todo el proceso se desconecta el sistema.

En la *Figura 68* se muestra un esquema con todos los elementos incorporados en la *GUIDE*, y en la *Figura 69*, *Figura 70* y *Figura 71* se muestra la posición de cada uno de ellos en la interfaz.

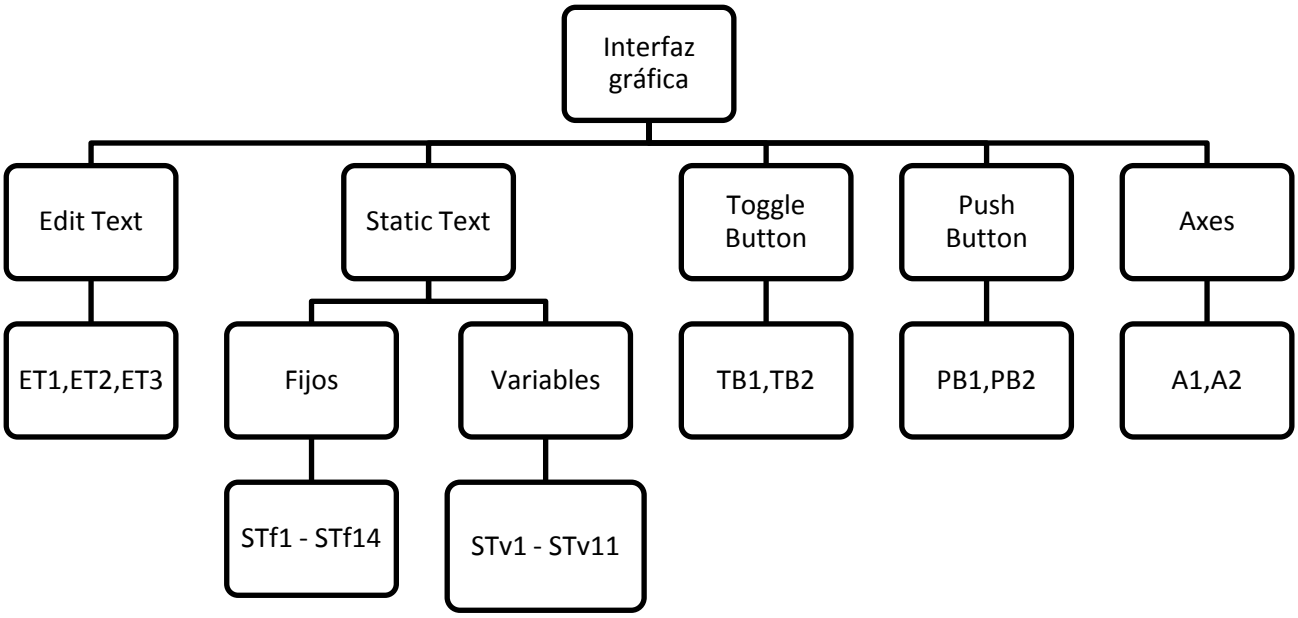


Figura 68. Elementos que componen la interfaz gráfica

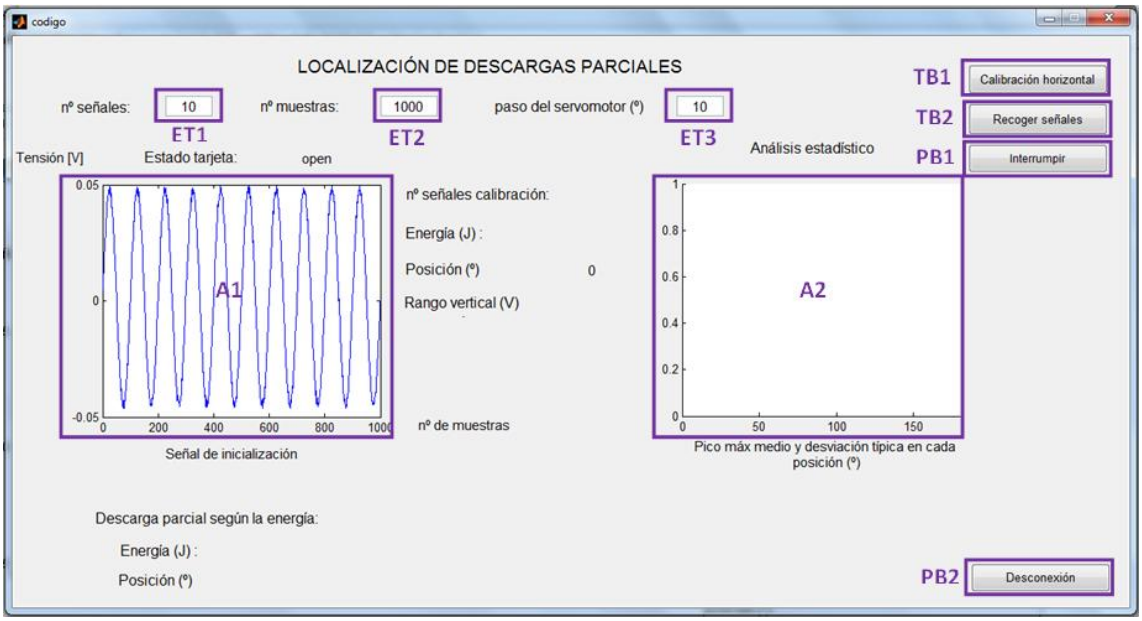


Figura 69. Elementos Edit Text, Toggle Button, Push Button y Axes en la interfaz gráfica

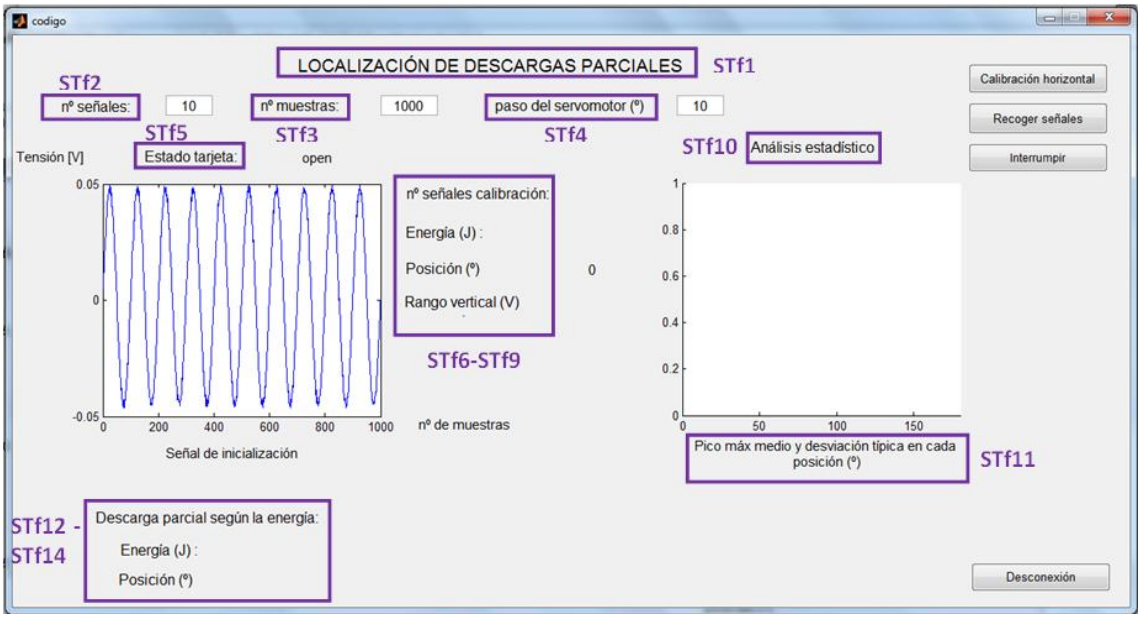


Figura 70. Elementos Static Text fijos en la interfaz gráfica

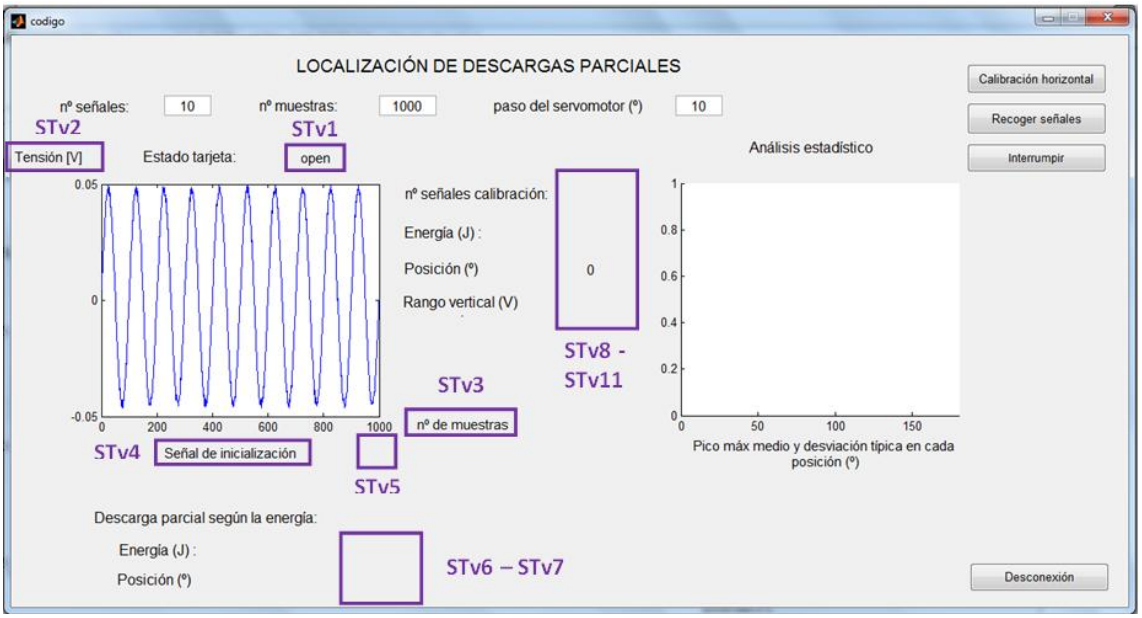


Figura 71. Elementos Static Text variables en la interfaz gráfica

La correlación que existe entre las abreviaturas asignadas a los elementos en las imágenes anteriores y su etiqueta en el programa se muestra en la *Tabla 16.*

Tabla 16. Correlación entre abreviaturas y etiquetas de los elementos de la interfaz gráfica

Correlación entre abreviaturas y etiquetas de los elementos de la interfaz gráfica			
Abreviatura	**Etiqueta**	**Abreviatura**	**Etiqueta**
ET1	edit_n_senales_def	STf9	text_rango_vertical
ET2	edit_n_muestras	STf10	text_titulo_estadistico
ET3	edit_paso_servo	STf11	text_info_estadistico
TB1	togglebutton_calibracion	STf12	text_datos_descarga
TB2	togglebutton_recoger_senales	STf13	text_energia_descargap
PB1	pushbutton_Interrumpir	STf14	text_posicion_descargap
PB2	pushbutton_desconexion	STv1	text_valor_estado_tarjeta
A1	axes1	STv2	text_ejey
A2	axes2	STv3	text_ejex
STf1	text_titulo_interfaz	STv4	text_info
STf2	text_num_senales	STv5	text_n_senales
STf3	text_num_muestras	STv6	text_energia_descarga
STf4	text_paso_servomotor	STv7	text_posicion_descarga
STf5	text_estado_tarjeta	STv8	text_n_senales_calibr
STf6	text_num_senal_cal	STv9	text_energia
STf7	text_energia_titulo	STv10	text_posicion
STf8	text_posicion_titulo	STv11	text_range

A continuación, se explica detalladamente con qué finalidad se ha creado cada elemento y las instrucciones que se llevan a cabo al activarse. El código completo se adjunta en el apartado *Código final del programa* del capítulo *Anexos.*

7.1. Edit Text

El programa diseñado utiliza determinados parámetros que no se pueden fijar desde un principio, su valor depende de las condiciones que quiera imponer el usuario a su estudio. En cada *Edit Text*, que aparecen como casillas en blanco, el usuario introduce el valor del parámetro al que está asociado. Dicho valor es leído y guardado en una variable en la parte del código correspondiente a la función de este componente de la *GUIDE*.

Las únicas variables que puede establecer el usuario son:

- Número de señales (*edit_n_senales_def*). Es el número de señales que se recogen en cada posición y todas ellas se tienen en cuenta en los cálculos correspondientes a los estudios llevados a cabo en el programa. Es recomendable adquirir varias señales para que los resultados proporcionados en cada ángulo sean más precisos. Durante las pruebas del programa se consideró un número de

señales igual a 10, pero en el análisis final, se estableció un número de señales igual a 50.

- Número de muestras (*edit_n_muestras*). Es el número de puntos que se recogen de cada onda y determina el número de columnas de los vectores y matrices en los que se almacenan las señales. A partir de esta variable también se establece el tiempo de adquisición. Partiendo de la ecuación [1] y sabiendo que la frecuencia de muestreo es 1 GS/s, si el número de muestras es igual a 1000, se obtiene un tiempo de adquisición de 1 μs. Si se aumentase, por ejemplo, el número de muestras a 10000, el tiempo de adquisición aumentaría a 10 μs. En nuestro caso, como sabemos que una ventana de adquisición de 1 μs es suficiente para recoger en su totalidad las señales de interés, se ha establecido un número de muestras igual a 1000.
- Paso del servomotor (*edit_paso_servo*). Son los grados que gira el servomotor cada vez que se mueve. Interesa que su valor sea pequeño para analizar más posiciones y encontrar de manera más precisa la posición de la fuente de descargas parciales. Teniendo en cuenta que el movimiento del servomotor está limitado a 180º, si el paso fuese, por ejemplo, de 90º, sólo se analizarían tres posiciones (0º, 90º, 180º). Sin embargo, si fuese igual a 20º, se analizarían 10 posiciones, por lo que el estudio sería más preciso y la dirección que indicaría el estudio estaría más cerca de la correspondiente a la fuente. En nuestro caso, se ha considerado que con un paso de 10º se alcanza una precisión buena en los resultados, ya que se analiza un número elevado de posiciones, igual a 19.

7.2. Static Text

Como se ha visto en el apartado anterior, el usuario se comunica con el programa a través de los *Edit Text*, pero la comunicación debe ser en ambas direcciones. El medio por el que el programa transmite información al operario son los cuadros de texto *Static Text*. En ellos, escribe los datos que permiten conocer la situación de la antena, las operaciones que se están ejecutando en cada momento y los resultados que obtiene al aplicar distintos métodos de análisis a las señales.

En esta interfaz se distinguen dos tipos de *Static Text*. Por un lado, están los cuadros de texto que permanecen fijos, y por otro, los cuadros de texto con contenido variable, que, en función de las líneas de código que se estén ejecutando, muestran una información u otra.

Fijos

Este tipo engloba todos los cuadros de texto que muestran títulos o nombres de variables. Son empleados para identificar los resultados que aparecen en los cuadros de

texto situados a su derecha. En la *Tabla 17* se muestran las etiquetas de todos los elementos de este tipo y se indica la información que muestran.

Tabla 17. Etiquetas de los Static Text fijos

Etiquetas de los *Static Text* fijos	**Información**
text_titulo_interfaz	Muestra el título de la interfaz *"Localización de descargas parciales"*.
text_num_senales	Se corresponde con el cuadro de texto *"nº señales"*.
text_num_muestras	Está asociada al cuadro de texto *"nº muestras"*.
text_paso_servomotor	Está asociada al cuadro de texto *"paso del servomotor"*.
text_estado_tarjeta	Se corresponde con el cuadro de texto *"Estado tarjeta"*
text_num_senal_cal	Está asociada al cuadro de texto *"nº señales calibración"*.
text_energia_titulo	Se corresponde con el cuadro de texto *"Energía (J)"*, asociado a la información que se muestra por posición.
text_posicion_titulo	Se corresponde con el cuadro de texto *"Posición (º)"*, asociado al ángulo en el que se encuentra el servomotor.
text_rango_vertical	Se corresponde con el cuadro de texto *"Rango vertical (V)"*, asociado al valor del rango que se obtiene por posición.
text_titulo_estadistico	Muestra el título de la gráfica con los resultados del estudio estadístico *"Análisis estadístico"*.
text_info_estadistico	Indica qué se muestra exactamente en la gráfica del análisis estadístico.
text_datos_descarga	Es el título que introduce los resultados del estudio de la energía *"Descarga parcial según la energía".*
text_energia_descargap	Se corresponde con el cuadro de texto *"Energía (J)"* situado en los resultados del estudio de la energía.
text_posicion_descargap	Se corresponde con el cuadro de texto *"Posición (º)"* situado en los resultados del estudio de la energía.

Variables

Informan al usuario de lo que está ocurriendo en cada momento y muestran los valores de los parámetros que cambian según el ángulo en el que se encuentre la antena. También muestran los resultados de los estudios llevados a cabo por el programa. En la *Tabla 18* se muestran las etiquetas de todos los elementos de este tipo y se indica la información que muestran.

Tabla 18. Etiquetas de los Static Text variables

Etiquetas de los *Static Text* variables	**Información**
text_valor_estado_tarjeta	Indica si el estado de comunicación de la tarjeta es *"closed"* o *"open"*. Durante la ejecución del programa, debe aparecer este último. Si no ocurre, significaría que la conexión con la tarjeta no se ha establecido correctamente.
text_ejey	Indica qué se está representando en el *eje y* de la gráfica situada a la izquierda.
text_ejex	Indica qué se está representando en el *eje x* de la gráfica situada a la izquierda.
text_info	Este elemento mantiene informado al usuario de lo que está ocurriendo en el programa. Especifica qué se está representando en la gráfica de la izquierda y comunica cuándo se inicia o finaliza la calibración y la recogida de señales, cuándo cambia de posición el servomotor y cuándo se interrumpe o desconecta el sistema.
text_n_senales	Durante la adquisición de señales muestra el número de la señal que se está recogiendo. Cuando finaliza la recogida en una posición, indica el siguiente ángulo en el que se va a situar el servomotor.
text_energia_descarga	Muestra la energía correspondiente a la posición de la descarga según el estudio de la energía.
text_posicion_descarga	Muestra el ángulo en el que se ha localizado la fuente de descargas parciales según el estudio de la energía.
text_n_senales_calibr	Indica cuántas señales se han recogido durante la calibración.
text_energia	Muestra la energía correspondiente a cada posición.
text_posicion	Indica el ángulo en el que se encuentra el servomotor.
text_range	Muestra el rango vertical calculado en cada posición.

7.3. Toggle Button

7.3.1. Calibración horizontal

Al ejecutar el programa, se parte de una configuración inicial de los parámetros de la tarjeta de adquisición, pero no tiene por qué ser la más idónea. Según el fenómeno o señales a analizar, es conveniente tener unos determinados valores. Las descargas parciales se caracterizan por presentar un elevado pico de tensión en los primeros instantes, y es este pico el que interesa analizar. El resto de muestras tomadas después de esta subida son innecesarias y podrían estropear el análisis si componen una gran parte de la señal recogida. Uno de los estudios que se realiza en el proyecto complementario consiste en el cálculo de la energía media de la señal, y si existen muchos puntos con bajo nivel de tensión, la media no sería muy elevada, haciendo inapreciable el pico de tensión, y por tanto, haciendo pasar desapercibida la descarga parcial. Si por el contrario, se recogen pocas muestras con esas características, al

calcular la energía media, el pico correspondiente a la descarga elevaría su valor, permitiendo identificar su posición al destacar frente a la energía de cualquier otro ángulo. Por este motivo, es necesario acotar la longitud de la señal.

Con el fin de recoger las señales correctas y adecuar el programa al caso de estudio, el usuario debe realizar antes de adquirir datos una previa calibración con la que determine la longitud de las señales adecuada.

En este apartado se va a explicar por un lado el proceso que debe realizar el operario, y por otro, las instrucciones que se ejecutan en el programa durante la calibración.

Acciones realizadas por el usuario

En primer lugar, el usuario introduce manualmente la longitud de la señal en la casilla *"ET2"* de la *Figura 69*, que conlleva la modificación de la variable *numSamples* (nº de muestras).

A continuación, activa el botón de calibración, iniciándose una adquisición de datos. En la interfaz se visualizan a tiempo real las ondas que se van adquiriendo, de manera que, el usuario, puede apreciar si las ondas recogidas presentan las características adecuadas para su estudio. Analizada la validez de los parámetros introducidos mediante la visualización de las señales, desactiva el botón de calibración. Si son correctos, da por finalizado este proceso, en caso contrario, inicia una nueva calibración.

Código en Matlab

Al seleccionar por primera vez el botón *Calibrar*, este se activa y el código se sitúa en la parte correspondiente a la función *togglebutton_calibracion_Callback*. En ella, se inicializa una variable *m* y una matriz denominada *matriz_calibracion* con valor uno y cero, respectivamente. La variable *m* se trata de un contador que registra el número total de señales adquiridas, y *matriz_calibracion* es una matriz de dimensiones pxn (p es el número de señales de calibración y n es igual al número de muestras) en la que se almacenan todas las señales recogidas durante la calibración. El número de filas que debería tener realmente esta matriz no se conoce, depende de cuántas señales se hayan recogido durante el tiempo que dure la calibración. Con el objetivo de que Matlab ejecute el código con mayor rapidez, se ha reservado para ella un espacio de memoria con un número de filas elevado e igual a 500. En teoría, no se recogerán más de 500 señales durante la calibración, ya que basta con un número pequeño para realizarla. Sin embargo, en el caso de que se adquiriese un número mayor, se aumentaría la dimensión de la matriz de forma automática debido a que Matlab tiene la capacidad de ampliar el tamaño de vectores y matrices en el momento en que se requiera. En la siguiente tabla se muestra un esquema de la variable *matriz_calibracion*.

Tabla 19. Matriz calibración

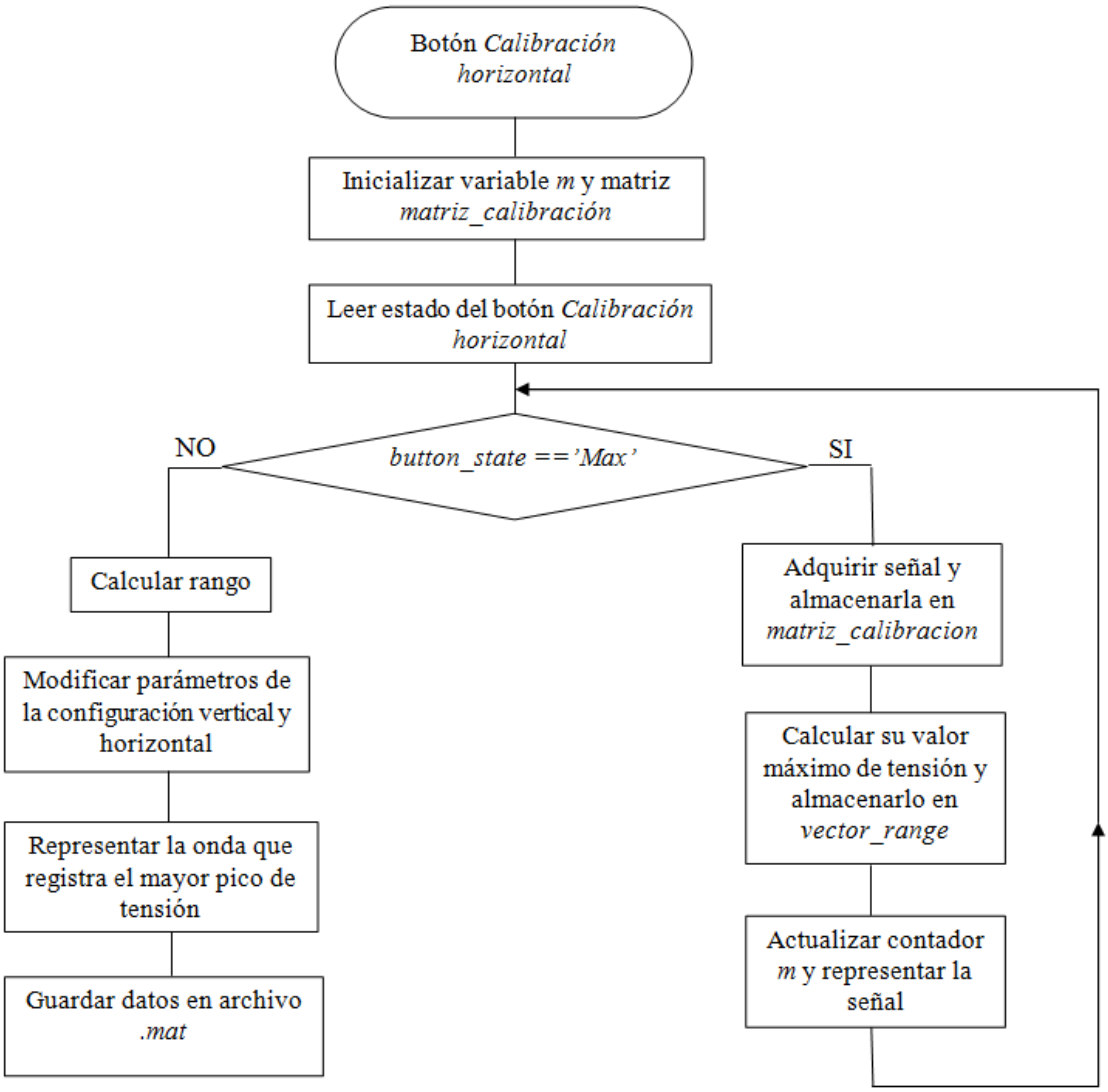
		1	***2***	***3***	***…***	***Nº muestras***
Nº señales calibración	***1***					
	2					
	3					
	…					
	500					

Inicializadas las variables anteriores, se inicia un bucle *while* que finaliza cuando se desactiva el botón, y las operaciones que se repiten en él son:

- Se adquiere una señal y se guarda en la matriz *matriz_calibracion*.
- Se calcula el valor máximo de la onda y se guarda en un vector llamado *vector_range* en la posición que indica el contador *m*.
- Se actualiza el contador.
- Se representa la señal en el gráfico situado a la izquierda de la interfaz (*axes1*). De esta manera, se visualiza a tiempo real la señal que se adquiere. Para asegurar que la señal no aparece cortada, se establecen como límites el número de muestras en el eje horizontal y el rango establecido para las señales recogidas durante la ejecución del código inicial en el eje vertical.
- Se comprueba el estado del botón.
- Si el botón continúa activado, se repiten las instrucciones anteriores.

Si el botón se ha desactivado, se calcula el rango final, que será un 40% superior del mayor valor guardado en el vector *vector_range*. Posteriormente, se invoca a la función de la configuración vertical y horizontal de la tarjeta para que queden grabados los nuevos valores del rango y número de muestras, respectivamente. Puesto que todas las señales adquiridas presentan la misma forma, una vez finalizada la calibración, sólo se representa una de ellas en la gráfica *axes1*, siendo la elegida la que registra el mayor pico de tensión (se podría graficar otra cualquiera). Así, se puede apreciar detenidamente las ondas adquiridas una vez finalizada la calibración, y comprobar si es necesario repetir el proceso. Por último, se guardan los datos almacenados en la matriz *matriz_calibracion* en un archivo denominado "*datos_calibracion.mat*".

Para una mejor comprensión, se muestra a continuación un diagrama de flujo de todo el proceso:



7.3.2. Recogida de señales

En este apartado, se explica cómo se inicia el movimiento y la recogida automática de señales a través de la antena acoplada al servomotor.

Acciones realizadas por el usuario

Antes de poner en marcha la recogida automática, el usuario debe especificar el número de señales que desea adquirir en cada posición y el paso del servomotor en grados. Para ello, introduce su valor en las casillas "*ET1*" y "*ET3*" de la *Figura 69*, respectivamente. A continuación, activa el sistema, es decir, la adquisición y movimiento automático pulsando el botón *Recoger señales*. Puesto que se trata de un botón *togglebutton*, es

necesario desactivarlo en algún momento. El programa está diseñado para que se desactive automáticamente al llegar a la última posición. Puede ocurrir que se interrumpa la recogida y no se dé esta situación. En este caso, se desactivaría al pulsar el botón *Interrumpir*, y para volver a iniciar la adquisición habría que activar de nuevo el botón *Recoger señales*.

Código en Matlab

En esta función, la variable *handles.movimiento* tiene una gran importancia. Esta variable es la que determina si el servo debe continuar con el movimiento establecido o permanecer quieto, y los valores que se le asignan son "1" y "0", respectivamente.

También aparece una variable denominada *cont_angulo* que registra el número de posiciones en las que se ha situado el servomotor y se inicializa al principio del programa, cuando se establece la comunicación entre Matlab y *Arduino*.

Las señales de una misma posición se guardan primero en una matriz denominada *senal* que como máximo tendrá una dimensión de zxn, siendo z el número de señales definido por el usuario y n, el número de muestras (en cada fila se guarda una señal). El número de filas depende del número de señales que se hayan recogido, de manera que, si no se ha interrumpido la adquisición se almacenará un número total de señales igual al definido. En caso contrario, el número de filas coincidirá con el número de ondas que se hayan almacenado hasta ese momento. Esta matriz se actualiza cada vez que se cambia de ángulo, y todos sus valores se almacenan en una matriz denominada *matriz_todas_senales*.

La matriz *matriz_todas_senales* contiene todas las señales adquiridas en todas las posiciones y es de dimensiones lxn, siendo l el número total de señales y n, el número de muestras definido. El número total de señales se determina a partir del número de señales indicado por el usuario para cada posición, la posición final del servomotor (180º) y el paso definido para su movimiento.

$$\begin{aligned} nº\ total\ señales &= nº\ total\ posiciones \cdot num\ señales\ (1\ posición) = \\ &= \left[\frac{posición\ final\ (grados)}{paso(grados)} + 1\right] \cdot num\ señales\ (1\ posición) = \\ &= \left[\frac{180º}{paso\ (p.u.) \cdot 180º} + 1\right] \cdot num\ señales\ (1\ posición) = \\ &= \left[\frac{1}{paso\ (p.u.)} + 1\right] \cdot num\ señales\ (1\ posición) \end{aligned} \qquad [3]$$

La ecuación final es la expresión que se emplea en el programa. En la siguiente tabla se muestra un esquema de la variable *matriz_todas_senales*.

Tabla 20. Matriz de todas las señales

		1	***2***	***3***	***…***	***nº muestras***
Posición 1	***1***					
	2					
	…					
	nº señales					
Posiciones intermedias	***…***					
Última posición	***1***					
	2					
	…					
	nº señales					

Definida la existencia de estas variables, se procede a explicar las líneas de código que componen la función *togglebutton_recoger_senales*.

Cuando se selecciona el botón *Recoger señales*, este se activa, el servomotor se sitúa en el ángulo cero y se le asigna a la variable *handles.movimiento* el valor de uno, permitiendo así el movimiento del servo. También se inicializa a cero la matriz *matriz_todas_senales*.

A continuación, se inicia un bucle *while* que se ejecuta mientras se cumpla la condición *handles.movimiento=1*. Su finalidad es realizar el movimiento automático de la antena mientras este esté permitido y las instrucciones que se ejecutan en él son:

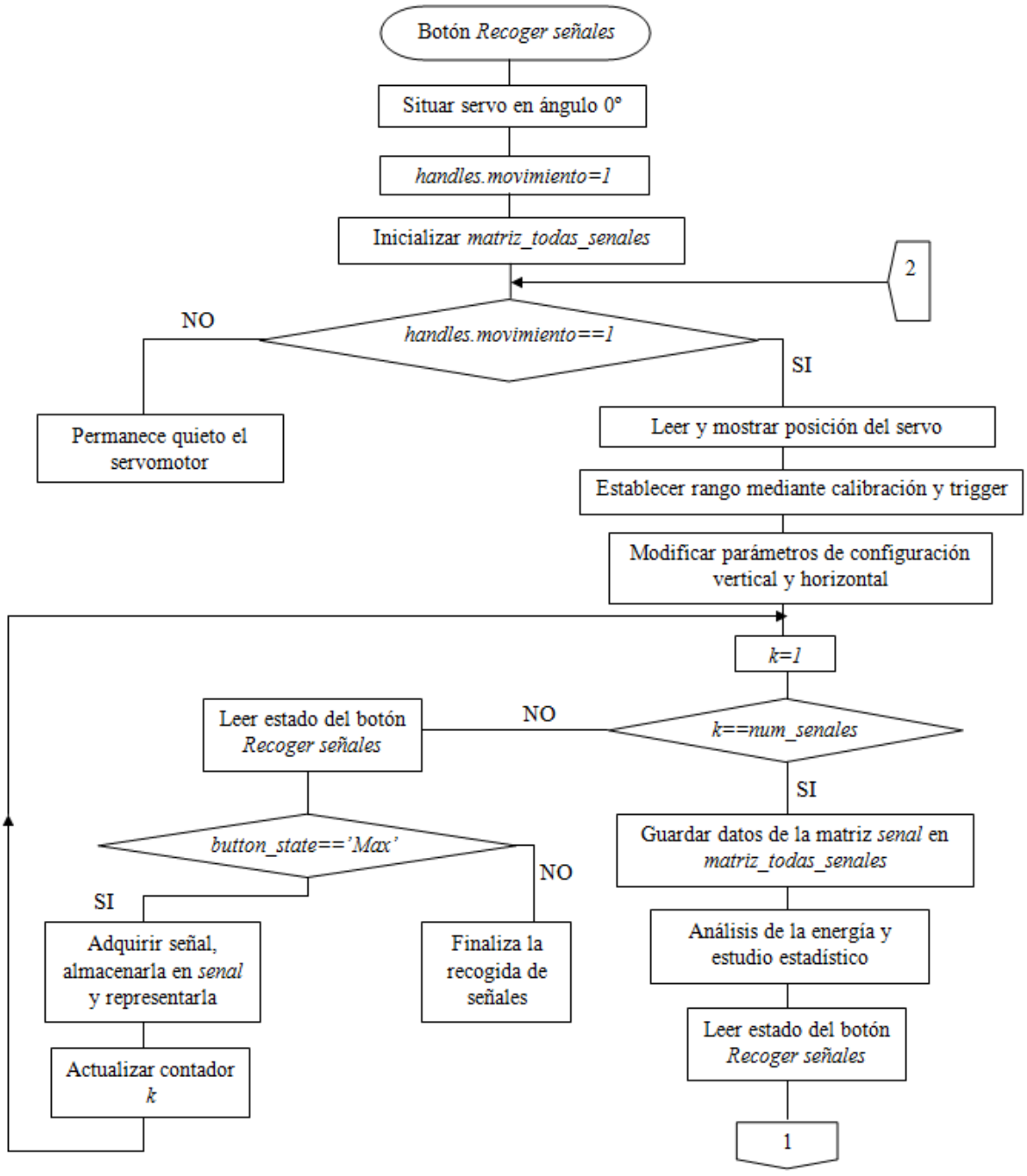
- Se lee la posición del servo y se muestra por pantalla.
- Se realiza una previa calibración vertical para determinar el rango tal y como se ha explicado en el apartado *"Configuración y parámetros*" del *Capítulo 4* y se establece el nivel de disparo adecuado para el estudio, definido también en ese apartado. Ambos valores se actualizan con las funciones de configuración vertical y de trigger de la tarjeta de adquisición.
- Se inicia la adquisición de las señales que se van a estudiar. Para ello, se ejecuta un bucle *for* cuyo contador se denomina *k* y cuyo límite es el número de señales definido por el usuario. Cada vez que se inicia un ciclo se comprueba el estado del botón *Recoger señales*, y sólo si sigue activo, se recoge la siguiente señal, se guarda en la matriz *senal* y se representa en la gráfica con etiqueta *axes1* para visualizarla a tiempo real. Si se encuentra desactivado, se detiene la adquisición de señales.

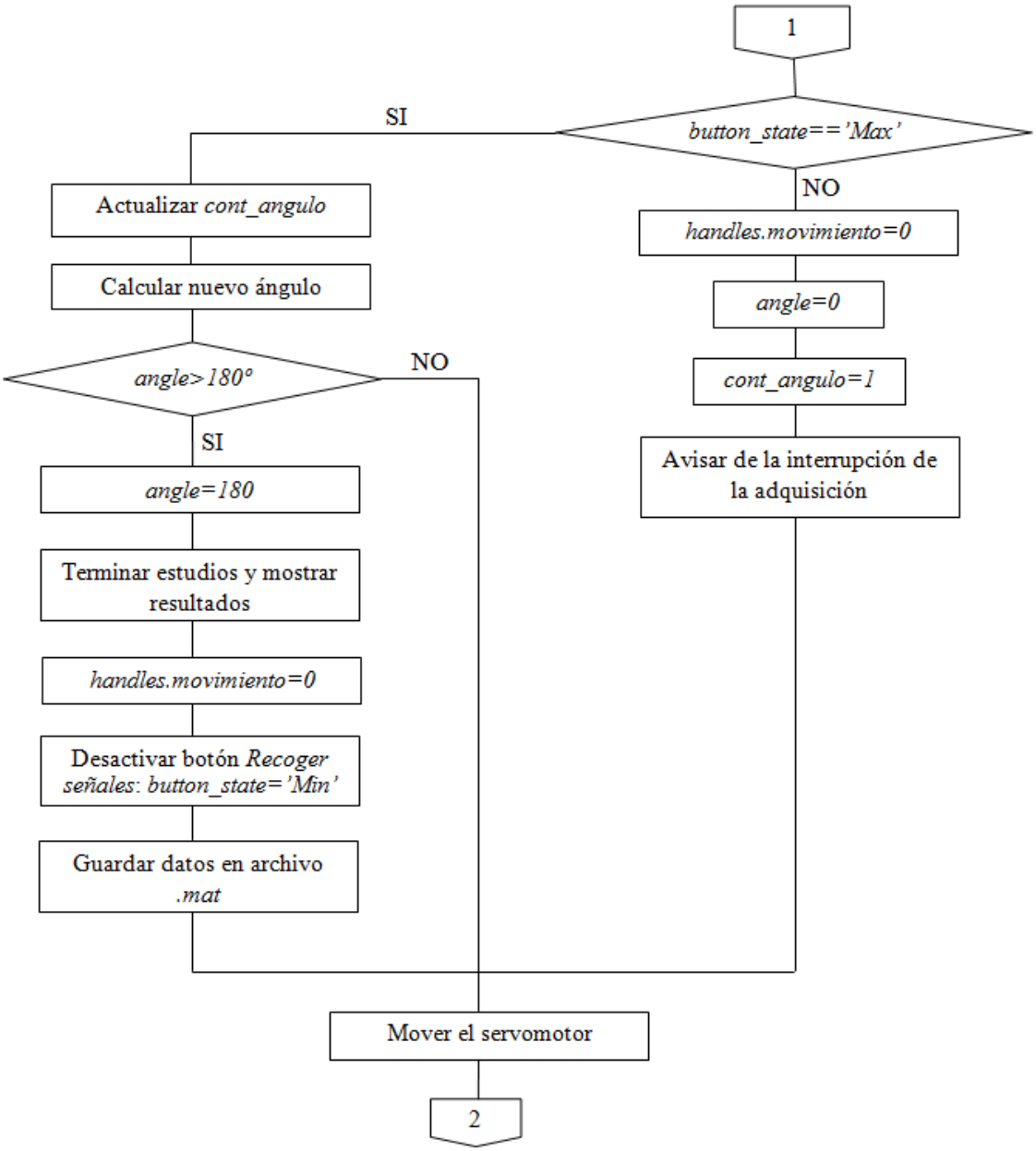
- Se guardan todas las ondas almacenadas en *senal* en la matriz *matriz_todas_senales* (en esta matriz se van incorporando las señales de cada ángulo).
- Se hace un análisis de la energía y un análisis estadístico de las señales. Esta parte del código no es objeto de estudio de este proyecto, se explica en el proyecto *"Localización de eventos en radiofrecuencia mediante una antena de alta directividad"*.
- Se comprueba de nuevo el estado del botón *Recoger señales*:

- Si continúa activo, se actualiza el contador *cont_angulo* (aumenta de uno en uno), se calcula la nueva posición sumando al ángulo actual el paso del servomotor y se analiza su nuevo valor.
    - Si el nuevo ángulo es inferior o igual al máximo permitido por el servo (180º), se guarda este valor en la variable *angle*, que indica la posición.
    - Si el ángulo es superior significa que se han analizado todas las posiciones, y como el servo está limitado se le asigna a la variable *angle* el valor de 180º. Se termina el estudio de la energía, se representan los resultados en la gráfica *axes1* y se muestra por pantalla la conclusión del análisis (posición de la descarga parcial). También se muestran los resultados del estudio estadístico en la gráfica *axes2* .A continuación, se desactiva el botón forzando el valor de su estado a cero y se indica al servo que debe permanecer en reposo en la última posición a la espera de recibir una nueva orden (*handles.movimiento=0*). Por último, se guardan los datos en un archivo denominado "*datos.mat*".
- Si se encuentra desactivado, significa que el botón *Interrumpir* ha sido pulsado. En este caso, se impide el movimiento del servomotor asignando a la variable *handles.movimiento* el valor cero y se le indica que su nueva posición es la posición inicial (posición 0º). Después se inicializa a uno el contador *cont_angulo* y se avisa al usuario por pantalla que se ha interrumpido la adquisición.

- Se mueve el servomotor al ángulo que indique la variable *angle* y se inicia de nuevo el bucle *while*.

Para una mejor comprensión, se muestra a continuación un diagrama de flujo de todo el proceso:


Botón Recoger señales
Situar servo en ángulo 0°
handles.movimiento=1
Inicializar matriz_todas_senales
2
NO
handles.movimiento==1
SI
Permanece quieto el servomotor
Leer y mostrar posición del servo
Establecer rango mediante calibración y trigger
Modificar parámetros de configuración vertical y horizontal
k=1
Leer estado del botón Recoger señales
NO
k==num_senales
SI
Guardar datos de la matriz senal en matriz_todas_senales
button_state=='Max'
NO
SI
Adquirir señal, almacenarla en senal y representarla
Finaliza la recogida de señales
Análisis de la energía y estudio estadístico
Actualizar contador k
Leer estado del botón Recoger señales
1


1
button_state=='Max'
SI
NO
Actualizar cont_angulo
Calcular nuevo ángulo
angle>180º
NO
SI
angle=180
Terminar estudios y mostrar resultados
handles.movimiento=0
Desactivar botón Recoger señales: button_state='Min'
Guardar datos en archivo .mat
handles.movimiento=0
angle=0
cont_angulo=1
Avisar de la interrupción de la adquisición
Mover el servomotor
2

7.4. Push Button

7.4.1. Interrumpir

Acciones realizadas por el usuario

Si por algún motivo, el usuario desea parar la adquisición automática de datos debe seleccionar el botón *Interrumpir*.

Código en Matlab

Al pulsar este botón, el programa desactiva el botón *Recoger señales*, forzando el valor de su estado a cero.

7.4.2. Desconectar

Acciones realizadas por el usuario

Una vez terminado el estudio, no se puede cerrar la interfaz gráfica del programa directamente. Previamente, hay que desconectar la tarjeta de adquisición, la placa *Arduino* y el servomotor. Para ello, se pulsa el botón *Desconexión*.

Código en Matlab

Cuando se pulsa este botón, el programa manda la orden al servomotor de situarse en la posición cero. A continuación, se elimina el "objeto Arduino" y el "objeto Servo", y por último, se desconecta y borra el "objeto dispositivo" creado al establecer la comunicación con la tarjeta de adquisición. También se avisa al usuario por pantalla que se ha seleccionado el botón *Desconexión*.

7.5. Axes

La interfaz gráfica está diseñada de tal manera que el usuario pueda analizar visualmente los resultados proporcionados por el programa y establecer a partir de ellos sus propias conclusiones. Para ello, dispone de dos gráficas, cada una con una función específica.

Gráfica: Axes1

Es la gráfica situada a la izquierda y tiene como objetivos:

- Mostrar las señales que se adquieren a tiempo real durante la calibración y el proceso de recogida en todas las posiciones.
- Mostrar los resultados del estudio de la energía al finalizar todo el proceso.

Gráfica: Axes2

Esta gráfica está situada a la derecha y en ella, se muestran los resultados del análisis estadístico. Estos resultados se calculan y muestran por pantalla a tiempo real. Cada vez que se termina la adquisición en una posición, se realiza su análisis estadístico y se representan los resultados de todos los ángulos estudiados hasta ese momento. El hecho de no esperar a que finalice todo el proceso para mostrarlos permite al usuario identificar si los resultados que se están obteniendo son los esperados. Si las gráficas no son coherentes interesaría interrumpir el proceso para analizar qué está ocurriendo y solucionar el problema.

Capítulo 8

8. Conclusiones y trabajos futuros

8.1. Conclusiones

El presente documento tiene como finalidad el control mediante la plataforma *Arduino* del movimiento de una antena directiva con un servomotor y la adquisición de las ondas electromagnéticas para las que está destinada la antena, en este caso, descargas parciales. Con objeto de alcanzar estos dos objetivos, el proyecto se ha dividido en tres partes independientes que después se han unificado.

En primer lugar, se ha diseñado el sistema de adquisición de datos. Para ello, ha sido necesario conocer las características de la tarjeta de adquisición disponible en el laboratorio (tarjeta NI PCI-5152) e investigar si existía algún driver específico que permitiese conectar dicha tarjeta con el entorno de programación en el que se ha desarrollado el programa (Matlab). A diferencia de otros modelos de tarjetas, la tarjeta empleada no es compatible con ningún driver de dicho entorno, por lo que ha sido necesario acudir al driver universal de *National Instruments* denominado NI Scope y realizar las correspondientes configuraciones. Para comprobar esta parte del proyecto, se realizaron pruebas en el laboratorio del código implementado. La comunicación se llevó a cabo con éxito: Matlab era capaz de conectarse a la tarjeta, y a través de ella, adquirir las señales detectadas por la antena.

Finalizada la primera parte, el proyecto se centró en el movimiento automático de la antena mediante un servomotor. El control de la posición del servomotor se realiza a través de la placa *Arduino*, que recibe las órdenes escritas en un archivo *.m* de Matlab. Para realizar dicho control es necesario establecer una comunicación entre la placa y el software. En esta ocasión, la conexión es muy simple, basta con instalar un paquete de

apoyo de Matlab destinado específicamente a la comunicación con esta plataforma. Establecida la conexión, se configuró el movimiento automático y se definió el paso del servomotor en un *Script* de Matlab. Para concluir con esta parte, se realizaron pruebas en el laboratorio, y se comprobó si el movimiento era automático y si el servo se desplazaba los ángulos indicados mediante el paso al moverse a la siguiente posición. Para esta última comprobación, se mostraba por pantalla las distintas posiciones en las que se había situado.

Tras realizar pruebas con distintos valores de paso, se llegó a la conclusión de que el servomotor no siempre gira el ángulo correcto. Con un paso de 45º sí se posiciona en las direcciones adecuadas; sin embargo, con un paso de 10º, se aprecian unas pequeñas desviaciones, el servo no es preciso: va intercalando un paso de 10,8º y 9º. El motivo puede estar en el cálculo del paso en por unidad en el código. Con el primer valor, se obtiene un paso en por unidad exacto, mientras que con el segundo, se obtiene un número periódico. El programa debe redondear este valor, y por eso, los anchos de pulsos que se envían al servomotor, y en consecuencia, las nuevas posiciones que se le indican, no se corresponden con las que se obtendrían si todos los cálculos se realizasen con los ángulos expresados en grados. Esto, además, impide que se puedan adquirir datos en la última posición (180º). A la hora de calcularla, el programa devuelve como resultado un ángulo superior, que difiere del correcto en apenas unos decimales, y al compararlo con la limitación del servo, detecta que la posición calculada supera los 180º, dando por finalizado todo el proceso, hecho que no se corresponde con la realidad.

Pese a que no se ha alcanzado precisión en el movimiento del servo, como las variaciones son pequeñas y el análisis de la última posición no afectaría a la localización de las descargas parciales, ya que la fuente no se sitúa en esa dirección, se puede concluir que esta segunda parte también ha proporcionado resultados satisfactorios.

Por último, se diseña un programa basado en una interfaz gráfica de usuario que unifica las dos partes anteriores.

La interfaz gráfica se ha enfocado a la localización de descargas parciales, y en ella, se han incorporado los elementos necesarios para alcanzar este fin. El usuario puede introducir manualmente el valor de ciertos parámetros y determinar mediante una previa calibración si son los adecuados para su estudio. También puede indicar el instante en que se inicia el movimiento de la antena y por tanto, la adquisición de señales en todas las posiciones, pudiendo interrumpir el barrido si lo considera oportuno. La interfaz muestra a tiempo real las señales y los resultados de uno de los estudios, facilitando así la decisión de interrumpir o seguir con el proceso. Además muestra los resultados de todos los estudios una vez finalizado el barrido.

Como conclusión, la aplicación desarrollada cumple con el propósito planteado en este trabajo, es capaz de generar el movimiento automático de la antena y llevar a cabo la adquisición de las señales.

8.2. Trabajos futuros

En este apartado se plantean posibles trabajos futuros que se podrían desarrollar como mejora o complemento a este proyecto.

- Realizar las medidas en una cámara anecoica de alta frecuencia con el fin de evitar que haya reflexiones de las ondas y garantizar así que la antena recoge bien los datos.
- Mover la antena en dos direcciones a partir de una estructura basada en dos servomotores. Uno de ellos, realizaría el giro en acimut, y el otro, en altura. Así, se determinaría de manera más precisa la posición de la fuente de descargas parciales.
- Diseñar una interfaz humano-máquina que controle el movimiento de la antena mediante diversas opciones (todas ellas conectadas a *Arduino*):
    - Giróscopo integrado en un guante.
    - Nunchuk.
    - Sensor de gestos que reconozca por ejemplo el movimiento de la cabeza o de las manos.
    - Gafas que detecten la dirección de la mirada.
- Emplear un sistema de adquisición de datos con comunicación inalámbrica para eliminar la limitación que el conexionado de cables conlleva.

Capítulo 9

9. Presupuesto

En este capítulo se presenta una estimación de los costes económicos que conlleva la realización del proyecto "*Control mediante Arduino del movimiento de una antena directiva con un servomotor*".

El presupuesto total se trata de una aproximación, ya que los costes de todos los elementos a tener en cuenta no son exactos, sino que son valores orientativos.

9.1. Costes directos

9.1.1. Coste de personal

Tabla 21. Coste de personal

Coste de personal				
Apellidos y nombre	**Categoría**	**Dedicación (meses)**	**Coste hombre mes (€)**	**Coste (€)**
Pérez Prieto, Sandra	Ingeniero	11	3000	33000
			Total	33000 €

9.1.2. Coste de equipos

Este apartado tiene en cuenta los costes de todos los equipos, dispositivos y licencias de software necesarias para llevar a cabo la investigación.

Pese a que en el momento de realizar los ensayos sólo se requiere un ordenador, en el presupuesto se incluyen dos PCs. El portátil se ha empleado para trabajar fuera del laboratorio, y en él, se ha desarrollado el código y realizado las pruebas y modificaciones necesarias hasta conseguir el funcionamiento correcto. El de sobremesa es el disponible en el laboratorio y con él se han tomado las medidas reales.

Tabla 22. Coste de equipos

Coste de equipos						
Descripción	**Precio unitario (€)**	**Número de unidades**	**% Uso dedicado al proyecto**	**Dedicación (meses)**	**Período de depreciación (meses)**	**Coste* (€)**
RECURSOS HARDWARE						
PC de sobremesa	700	1	100	11	36	213,889
PC portátil	400	1	100	7	36	77,778
Tarjeta de adquisición NI PCI-5152	8270	1	100	7	24	2412,083
Arduino *UNO*	20	1	100	7	24	5,833
RECURSOS SOFTWARE						
Matlab R2014. Licencia	2000	2	100	7	12	2333,333
RECURSOS TANGIBLES						
Servomotor Hitec HS-422	9,20	1	100	7	24	2,683
Accesorios del servomotor	41,40	1	100	7	24	12,075
Cable coaxial RG-58	6,19	1	100	7	24	1,805
Cable USB 2.0 (Tipo A/M-B/M)	3,99	1	100	7	24	1,164
Antena parabólica	0,90	1	100	7	24	0,263
Plato metálico circular	4,50	1	100	7	24	1,313
Antena monopolo 10 cm	10	2	100	7	12	11,667
Antena Vivaldi	10	1	100	7	24	2,917
					Total	5076,803 €

*Para el cálculo de los costes se realiza un cálculo de la amortización:

$$Amortización = \frac{A}{B} \cdot C \cdot D \qquad [4]$$

A = Tiempo en meses en que el equipo es utilizado desde la fecha de facturación.

B = Tiempo de depreciación en meses. Se trata de una estimación.

C = Coste del equipo (sin I.V.A)

D = % de uso dedicado al proyecto. En la fórmula se pone el valor dividido entre 100.

9.1.3. Subcontratación de tareas

Tabla 23. Subcontratación de tareas

Subcontratación de tareas		
Descripción	**Empresa**	**Coste (€)**
	Total	0 €

9.1.4. Otros costes directos

Tabla 24. Otros costes directos

Otros costes directos (Fungibles, viajes y dietas…)		
Descripción	**Empresa**	**Coste (€)**
	Total	0 €

9.2. Costes indirectos

Tabla 25. Costes indirectos

Costes indirectos			
Descripción	**Coste (€/mes)**	**Dedicación (meses)**	**Coste total (€)**
Internet y teléfono	47	11	517
Luz, agua y gas	100	11	1100
Material de oficina	30	11	330
		Total	1947 €

9.3. Importe presupuestado

Todos los costes anteriores son costes sin I.V.A. Para obtener el importe presupuestado se aplica un I.V.A. del 21%.

$$Importe\ presupuestado = importe\ total + 0{,}21 \cdot importe\ total \qquad [5]$$

siendo el importe total el coste total sin I.V.A.

Tabla 26. Resumen de costes

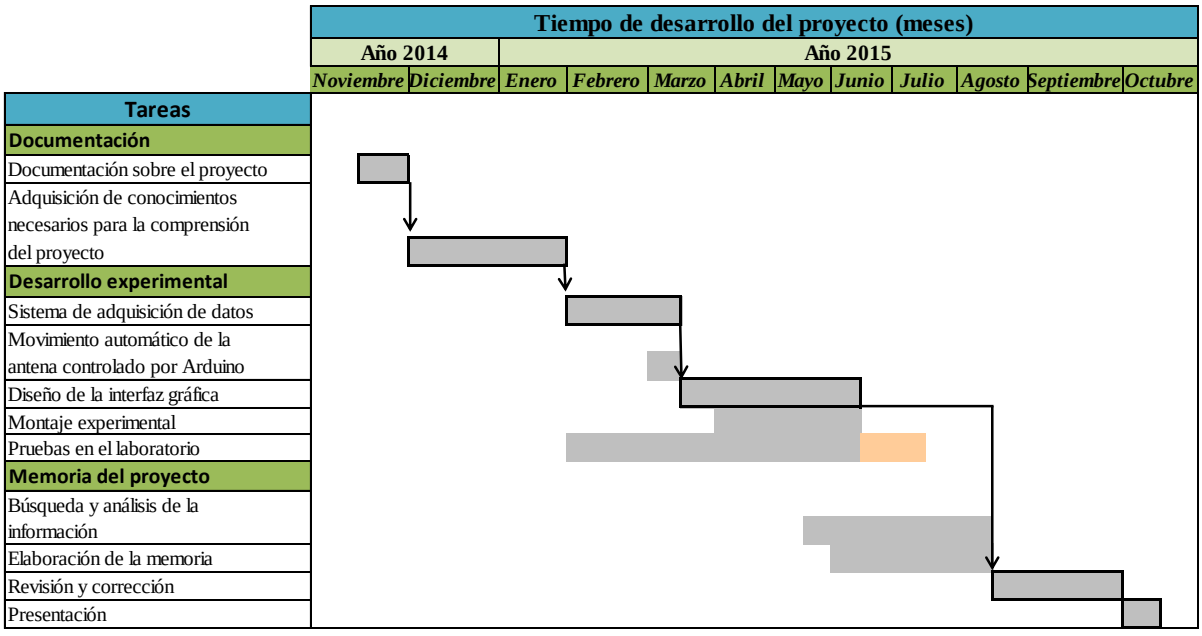
Resumen de costes	
Concepto	**Coste (€)**
Personal	33000
Equipos	5076,803
Subcontratación de tareas	0
Otros costes directos	0
Costes indirectos	1947
Importe total	40023,803
I.V.A (21%)	8404,999
Importe presupuestado	48428,802

Capítulo 10

10. Planificación del proyecto

En este capítulo se indica el tiempo dedicado a cada una de las tareas en las que se ha estructurado la investigación

Tabla 27. Planificación del proyecto. Diagrama de Gantt



En la tarea "*Pruebas en el laboratorio*" se distinguen dos colores. El color gris se corresponde con los ensayos de este proyecto, y el color naranja, a los ensayos que pertenecen únicamente al proyecto complementario a este. Ambos se incluyen en la planificación de este trabajo debido a que todas las pruebas se han realizado conjuntamente.

Bibliografía

[1] J. A. M. Velasco, Coordinación de aislamiento en redes eléctricas de alta tensión, McGraw-Hill, 2008.

[2] «National Instruments,» [En línea]. Available: http://sine.ni.com/cs/app/doc/p/id/cs-15865. [Último acceso: 11 Agosto 2015].

[3] Ó. T. Artero, Arduino: curso práctico de formación, RC Libros, 2013.

[4] [En línea]. Available: http://www.sc.ehu.es/acwamurc/Transparencias/(4)TAD.pdf. [Último acceso: 18 Julio 2015].

[5] A. R. Nava, Adaptación y control de plataformas hardware docentes mediante tarjeta de adquisición de datos.Proyecto Fin de Carrera en Ingeniería Técnica Industrial. Universidad Carlos III de Madrid, Octubre de 2011.

[6] M. G. M. y. E. M. Bolado, Instrumentación electrónica: transductores y acondicionadores de señal, Universidad de Cantabria, 2015.

[7] J. P. González, Instrumentación, adquisición de datos para laboratorio multidisciplinar y motorización de bomba manual de mecate. Proyecto Fin de Carrera en Ingeniería Industrial. Universidad Carlos III de Madrid, Enero de 2012.

[8] [En línea]. Available: https://prezi.com/gbnwmlmmd3dk/parametros-analogos-de-entradasalida-de-una-daq-hardware-y-software/. [Último acceso: 18 Julio 2015].

[9] «National Instruments,» [En línea]. Available: http://sine.ni.com/np/app/main/p/bot/no/ap/mi/lang/es/pg/1/ps/1000/sn/n25:device,n17:mi,n10:10988/sb/default/. [Último acceso: 20 Julio 2015].

[10] «GaGe,» [En línea]. Available: http://www.gage-applied.com/. [Último acceso: 20 Julio 2015].

[11] «National Instruments,» [En línea]. Available: http://www.ni.com/download/ni-

daqmx-14.5.1/5271/en/. [Último acceso: 20 Julio 2015].

[12] «National Instruments,» [En línea]. Available: http://sine.ni.com/nips/cds/view/p/lang/es/nid/203361. [Último acceso: 20 Julio 2015].

[13] «Arduino,» [En línea]. Available: http://www.arduino.cc/. [Último acceso: 12 Junio 2015].

[14] «MathWorks,» [En línea]. Available: http://es.mathworks.com/hardware-support/arduino-matlab.html. [Último acceso: 7 Marzo 2015].

[15] «SuperRobotica,» [En línea]. Available: http://www.superrobotica.com/Servosrc.htm. [Último acceso: 14 Junio 2015].

[16] A. H. y. B. Drury, Electric motors and drives: fundamentals, types and applications, Newnes, 2006.

[17] «Tecnología,» [En línea]. Available: http://www.areatecnologia.com/electricidad/servomotor.html. [Último acceso: 22 Julio 2015].

[18] P. E. Sandin, Robot Mechanisms and Mechanical Devices Illustrated, McGraw-Hill, 2003.

[19] «SuperRobotica,» [En línea]. Available: http://www.superrobotica.com/S330165.htm. [Último acceso: 14 Junio 2015].

[20] «Sparkfun,» [En línea]. Available: https://www.sparkfun.com/products/11884. [Último acceso: 14 Junio 2015].

[21] «servocity,» [En línea]. Available: https://www.servocity.com/html/hs-422_super_sport_.html#.Vc5S9Pntmko. [Último acceso: 14 Junio 2015].

[22] [En línea]. Available: http://www.esi2.us.es/~asun/LCPC06/TutorialLabview.pdf. [Último acceso: 10 Agosto 2015].

[23] J. B. Pérez, Desarrollo de una aplicación en interfaz gráfica de MatLab para la determinación del comportamiento dinámico de un vehículo automóvil. Proyecto Fin de Carrera en Ingeniería Mecánica. Universidad Carlos III de Madrid, Mayo de 2011.

[24] M. S. Gervaso, Diseño de una banda transportadora mediante GUIDE de MatLab.

Proyecto Fin de Carrera en Ingeniería Industrial. Universidad Carlos III de Madrid, Octubre de 2013.

[25] «MathWorks,» [En línea]. Available: http://es.mathworks.com/help/instrument/install-the-ni-scope-support-package.html. [Último acceso: 3 Febrero 2015].

[26] «MathWorks,» [En línea]. Available: http://es.mathworks.com/help/instrument/examining-your-hardware-resources.html. [Último acceso: 5 Febrero 2015].

[27] «MathWorks,» [En línea]. Available: http://es.mathworks.com/products/instrument/code-examples.html?file=/products/demos/instrument/using_vxipnp.html. [Último acceso: 5 Febrero 2015].

[28] «MathWorks,» [En línea]. Available: http://es.mathworks.com/help/instrument/examples/fetch-waveforms-through-niscope-matlab-instrument-driver-in-simulation-mode.html#zmw57dd0e2188. [Último acceso: 5 Febrero 2015].

[29] «MathWorks,» [En línea]. Available: http://cn.mathworks.com/help/pdf_doc/instrument/instrument.pdf. [Último acceso: 5 Febrero 2015].

[30] «MathWorks,» [En línea]. Available: http://es.mathworks.com/help/instrument/ivi-configuration-store.html. [Último acceso: 5 Febrero 2015].

[31] «National Instruments,» [En línea]. Available: http://www.ni.com/tutorial/3698/es/. [Último acceso: 5 Febrero 2015].

[32] «MathWorks,» [En línea]. Available: http://es.mathworks.com/help/supportpkg/arduinoio/examples/control-servo-motors.html?prodcode=ML. [Último acceso: 7 Marzo 2015].

[33] A. C. Hernández, Manual PDS100: Dispositivo portátil de inspección de descargas parciales mediante la medida de energía electromagnética en el espectro de RF. Trabajo Fin de Grado en Ingeniería Eléctrica. Universidad Carlos III de Madrid, Julio de 2013.

Anexos

Hoja de características: Tarjeta NI PCI-5152

NATIONAL INSTRUMENTS

Technical Sales

(866) 531-6285
orders@ni.com

Ordering Information | Detailed Specifications

For user manuals and dimensional drawings, visit the product page resources tab on ni.com.

Last Revised: 2014-11-06 07:14:06.0

2 GS/s High-Speed Digitizers: Optimized for Automated Test

NI PXI-5152, NI PCI-5152

- 2 GS/s maximum real-time sample rate
- 300 MHz, 500 MHz, and 1 GHz bandwidths
- Up to 20 GS/s equivalent.time sampling
- 2 channels simultaneously sampled
- Edge, window. hysteresis. Digital, immediate, and software triggering

Overview

NI PXl-5152 and PCI-5152 high-speed digitizersTPC-based oscilloscopes provide the industry's first gigahertz solutions optimized for automated test. A digtizer optimized for automated test takes advantage of a high-throughput bus 10 lower lesl times, provides picosecond-level synchronization between modules, and integrates with the entire suite of NI hardware — including arbitrary waveform generators, high-speed digital I/O, and other digitzers — so you can build and customize a complete mixed-signal or high-channel-count test system.

Application and Technology

NI High-Speed Digitizers: Optimized for Automated Test

Prior to these products. high-bandwidth digitizers and oscilloscopes incorporated features and functionality best suited for benchtop use. An unaddressed area in this high-bandwidth space has been the automated test use model, where measurement throughput and test system footprint can dramatically affect overall cost of test. NI high-speed digitizers are the first digltizers on the market to share three characteristics making them uniquely optimized for automated test: high data throughput. tight synchronization between channels, and ease of integration v.'ith other instruments.

High Data Throughput

Bus bandwidth and latency, two common considerations for an automated test system, dictate the overall speed of your meas..rement system. Latency describes the amount of time it lakes for an instrument to respond to a remote command, like a measurement query. Bus bandwidth refers primarily to the data throughput capacity of the data bus that connects the measurement instrument with the host PC or controller.

The PXI platform — upon which NI high-speed digitizers are built — provides high speed due 10 the high-bandwidth and low-latency PCI and PCI Express buses. Both PXI and PXI Express data throughput rates are significantb' faster than that of GPIB, USB, or LAN — other popular buses for automating test instrumentation. This translates to lower test times.

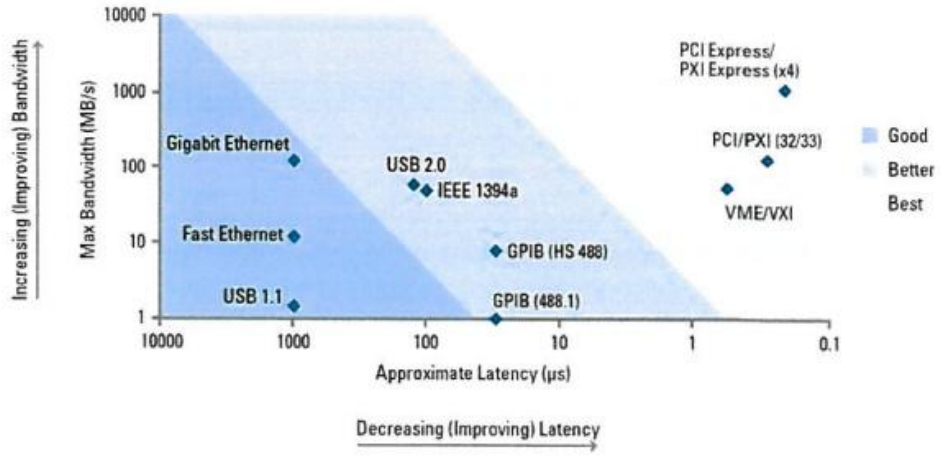


Figure 1. The PXI platform provides the best combination of high-bandwidth and low-latency measurement throughput.

Tight Synchronization between Channels

The PXI backplane offers a built-in common reference clock for synchronization of multiple digitizers 'n a measurement or control system. Each slot has a 10 MHz TTL clock. transmitted on equal-length traces, that provides picosecond-level synchronization between digitizer modules 'cr high-channel-count systerns_ For example, it is possible to have 34 phase-synchronous 1 GS/s channels in a single PXI chassis. and scale to even higher-channel counts.

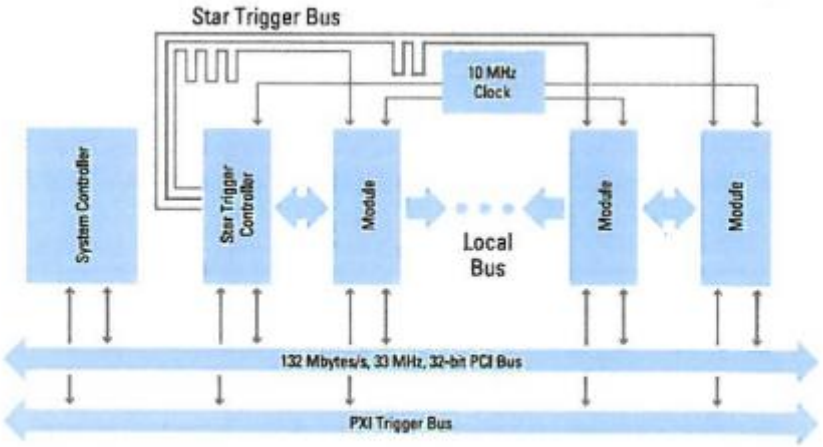


Figure 2. The PXI platform delivers picosecond-level synchronization.

Ease Of Integration with Other Instrumentation

Test systems typically contain many instrument types, including signal sources, measurement devices. and switches. The PX! platform has unparalleled breadth, WIth modules for analog and digital I/O. high-speed instrumentation, vision, motion, and numerous bus interfaces. More than 1.500 PXI modules are available from the more than 70 members of the PX' Systems Alliance (PXISA). So you can not only build a comprehensive test system in a single chassis but also synchronize modules in thal chassis to picosecond-level accuracy when using NI modular instruments.

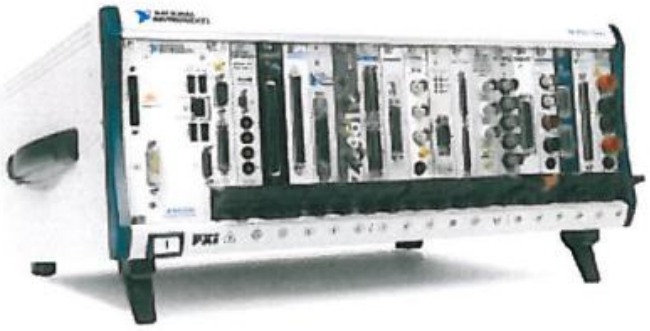


Figure 3. The PXI platform supports more than 1,500 instrument modules.

Achieve Flexible Performance Using M Software-Defined Instrumentation

NI high-speed digitizers Offer several advantages over traditional stand-alone oscilloscopes by delivering an open architecture and fkxible software, Wilh an NI digitizer. you can not only perform standard oscilloscope measurements but also easily buiEd other instruments such as spectrum analyzers, transient recorders, and ultrasonic receivers. And National Instrurnents offers a comprehensive library in the NI LabVlEW graphical development environrr,ent of prebuilt functions and example programs geared at getting you up and running quickly.

Open Architecture: NISCOPE Driver and the Application Programming Interface (API)

Using the full power of a pc-based measurement device requires the ability to programmatically define and control its behavior. You can programmaticalW control all NI digitizers using the Nt-SCOPE instrument driver, which provides the following:

- High-level functions for getting started quickly as well as low-level control for accessing all the digitizer features
- More than 50 prewritten example programs that illustrate howto access the full functionality of any NI digitizer
- Programming examples available for LabVlEW, , and Visual Basic

Flexible Software: Define Your Instrument

In an automated test environment. there are times when the ability to quickly troubleshoot an issue is crucial. For those occasions. the NESCOPE driver offers the measurelnent features and responsiveness of a traditional benchtcp oscilloscope through the NISCOPC Soft Front Panel user interface. Take advantage of the more than 50 prebuilt measurement and analysis functions included with the software.

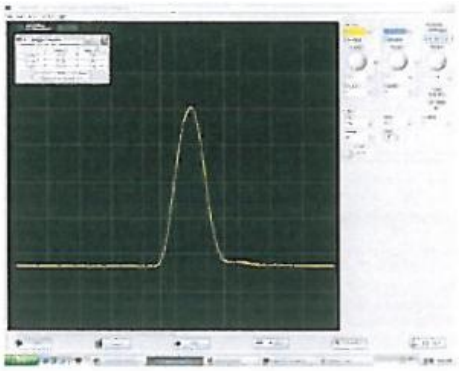


Figure 4. The NI-SCOPE Soft Front Panel provides measurement features and responsiveness comparable to traditional benchtop oscilloscopes.

For rapid initiation of an automated test sequence, use preconfigured Express VIS 10 quickly set up your digitizer to immediately acquire data. With the LabVlEW SignalExpress interactive environment, you can acquire, analyze, and log your dala Wilh no programrWng required.

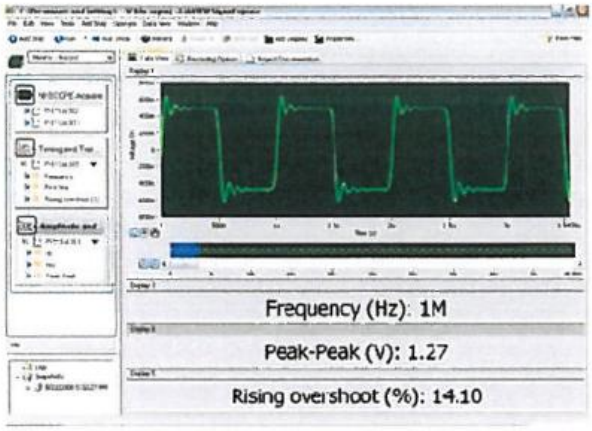


Figure 5. With LabVIEW SignalExpress, you can quickly set up your digitizer to immediately acquire data.

While a quick signal check is valuable at tirnes. other circumstances may call for custom measurements. Stand-alone instruments such as dedicated oscilloscopes and spectrum analyzers deliver common functions that appeal to the needs of many engineers. As you can imagine. these standard functions do not meet every application need. particularly in automated test applications. But 'Mth LabVlEW and the NESCOPE API, the digitizer that you use as a general-purpose oscilloscope for one application can be used as a custom instrument ror more specialized measurements.

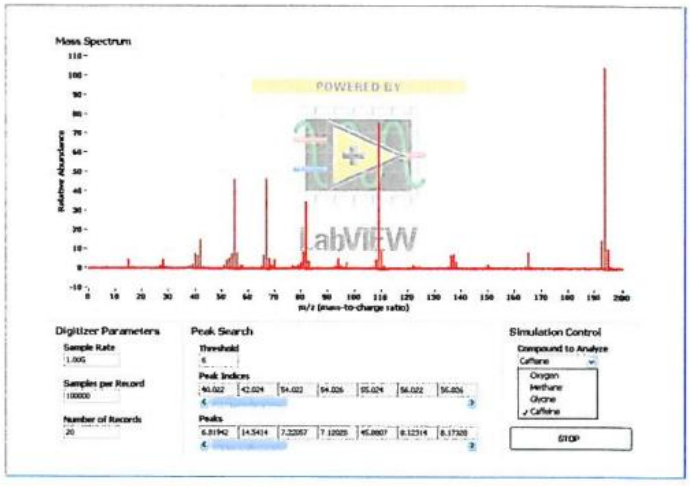


Figure 6. You can achieve custom measurements, such as those required for mass spectrometry, using the combination of LabVIEW and NI modular instruments.

Ordering Information

For a complete list of accessories, visit the product page on ni.com.

Products	Part Number	Recommended Accessories	Part Number
NI PXI-5152/64MB			
NI PXI-5152/64MB Requires: 1 Cables ;	779772-02	**Cables:** Unshielded - SMB112, Double Shielded SMB to BNC Male Coax Cable, 50 Ohm, 1m	778827-01
NI PCI-5152_64			
NI PCI-5152 64MB/ch Requires: 1 Cables ;	779945-02	**Cables:** Unshielded - SMB112, Double Shielded SMB to BNC Male Coax Cable, 50 Ohm, 1m	778827-01

Back to Top

Support and Services

System Assurance Programs

NI system assurance programs are designed to make it even easier for you to own an NI system. These programs include configuration and deployment services for your NI PXI, CompactRIO, or Compact FieldPoint system. The NI Basic System Assurance Program provides a simple integration test and ensures that your system is delivered completely assembled in one box. When you configure your system with the NI Standard System Assurance Program, you can select from available NI system driver sets and application development environments to create customized, reorderable software configurations. Your system arrives fully assembled and tested in one box with your software preinstalled. When you order your system with the standard program, you also receive system-specific documentation including a bill of materials, an integration test report, a recommended maintenance plan, and frequently asked question documents. Finally, the standard program reduces the total cost of owning an NI system by providing three years of warranty coverage and calibration service. Use the online product advisors at ni.com/advisor to find a system assurance program to meet your needs.

Calibration

NI measurement hardware is calibrated to ensure measurement accuracy and verify that the device meets its published specifications. To ensure the ongoing accuracy of your measurement hardware, NI offers basic or detailed recalibration service that provides ongoing ISO 9001 audit compliance and confidence in your measurements. To learn more about NI calibration services or to locate a qualified service center near you, contact your local sales office or visit ni.com/calibration.

Technical Support

Get answers to your technical questions using the following National Instruments resources.

- **Support.** Visit ni.com/support to access the NI KnowledgeBase, example programs, and tutorials or to contact our applications engineers who are located in NI sales offices around the world and speak the local language.
- **Discussion Forums.** Visit forums.ni.com for a diverse set of discussion boards on topics you care about.
- **Online Community.** Visit community.ni.com to find, contribute, or collaborate on customer-contributed technical content with users like you.

Repair

While you may never need your hardware repaired, NI understands that unexpected events may lead to necessary repairs. NI offers repair services performed by highly trained technicians who quickly return your device with the guarantee that it will perform to factory specifications. For more information, visit ni.com/repair.

Training and Certifications

The NI training and certification program delivers the fastest, most certain route to increased proficiency and productivity using NI software and hardware. Training builds the skills to more efficiently develop robust, maintainable applications, while certification validates your knowledge and ability.

- **Classroom training in cities worldwide** - the most comprehensive hands-on training taught by engineers.
- **On-site training at your facility** - an excellent option to train multiple employees at the same time.
- **Online instructor-led training** - lower-cost, remote training if classroom or on-site courses are not possible.
- **Course kits** - lowest-cost, self-paced training that you can use as reference guides.
- **Training memberships** and training credits - to buy now and schedule training later.

Visit ni.com/training for more information.

Extended Warranty

NI offers options for extending the standard product warranty to meet the life-cycle requirements of your project. In addition, because NI understands that your requirements may change, the extended warranty is flexible in length and easily renewed. For more information, visit ni.com/warranty.

OEM

NI offers design-in consulting and product integration assistance if you need NI products for OEM applications. For information about special pricing and services for OEM customers, visit ni.com/oem.

Alliance

Our Professional Services Team is comprised of NI applications engineers, NI Consulting Services, and a worldwide National Instruments Alliance Partner program of more than 700 independent consultants and integrators. Services range from start-up assistance to turnkey system integration. Visit ni.com/alliance.

Detailed Specifications

8-Bit 2 GS/s Digitizer

This topic lists the specifications for the NI PXI/PCI-5152 (NI 5152) high-speed digitizer. Unless otherwise noted, the following conditions were used for each specification:

- All filter settings
- All impedance selections
- Sample clock set to 1GS/s

Real-Time Interleaved Sampling (TIS) mode provides a 2 GS/s real-time sample rate for a single channel.

Typical values are representative of an average unit operating at room temperature. Specifications are subject to change without notice. For the most recent NI 5152 specifications, visit ni.com/manuals.

To access the NI 5152 documentation, including the *NI High-Speed Digitizers Getting Started Guide*, go to **Start»All Programs»National Instruments»NI-SCOPE»Documentation.**

Hot Surface If the NI 5152 has been in use, it may exceed safe handling temperatures and cause burns. Allow the NI 5152 to cool before removing it from the PXI chassis or PC. Refer to the *Environment* section for operating temperatures of this device.

Caution Refer to the Read Me First: Safety and Electromagnetic Compatibility document for important safety and electromagnetic compatibility information. To obtain a copy of this document online, visit ni.com/manuals.

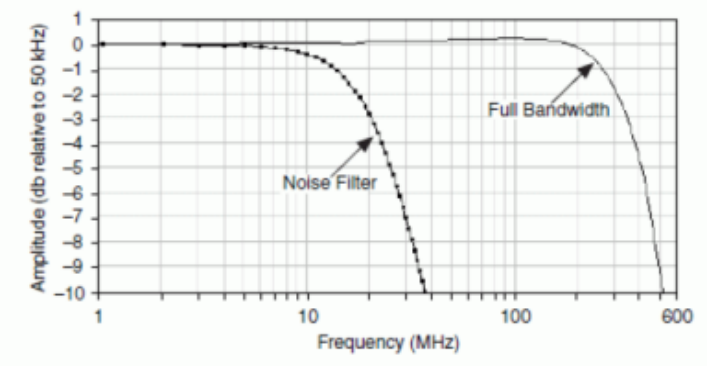
Vertical

Analog Input (Channel 0 and Channel 1)

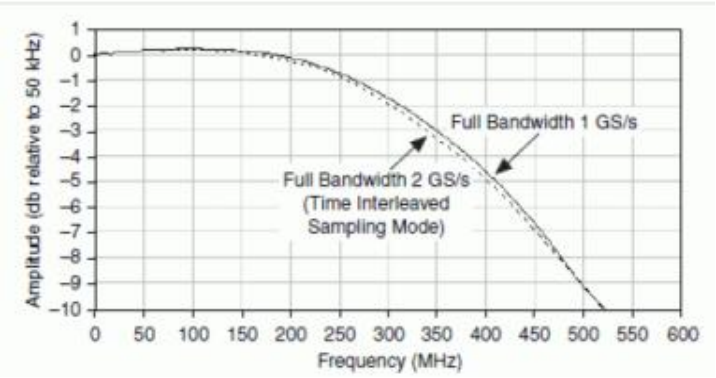
Specification	Value				Comments
Number of Channels	Two (simultaneously sampled)				—
Connectors	BNC				—
Impedance and Coupling					
Input Impedance	50 Ω ±1.5% 1 MΩ ±0.75% in parallel with a typical capacitance of 22 pF				Software selectable.
Input Coupling	AC, DC, GND				—
Voltage Levels					
Full Scale (FS) Input Range and Programmable Vertical Offset	50 Ω		1 MΩ		—
	Range (V_{pk-pk})	Offset (V)	Range (V_{pk-pk})	Offset (V)	
	0.1	±1	0.1	±1	
	0.2	±1	0.2	±1	
	0.4	±1	0.4	±1	
	1	±1	1	±1	
	2	±6	2	±10	
	4	±5	4	±10	
	10	±2	10	±10	
Maximum Input Overload	50 Ω		1 MΩ		—
	7 V_{rms} with |Peaks| ≤ 10 V		|Peaks| ≤ 42 V		
Accuracy					
Resolution	8 bits				—
DC Accuracy (Programmable Vertical Offset = 0 Volts)	Range (V_{pk-pk})		50 Ω and 1 MΩ		Within ±5 °C of self-calibration temperature.
	0.1 to 1		± (1.26% of Input + 1.0% of FS + 500 µV)		
	2 to 10		± (1.26% of Input + 1.0% of FS + 5 mV)		
Programmable Vertical Offset Accuracy	±0.9% of offset setting				Within ±5 °C of self-calibration temperature.
DC Drift	Range (V_{pk-pk})		50 Ω and 1 MΩ		Use DC drift to calculate errors when temperature changes more than ±5 °C since the last self-calibration.
	0.1 to 1		± (0.052% of Input + 100 µV) per °C		
	2 to 10		± (0.052% of Input + 1.0 mV) per °C		
Crosstalk, Typical	CH 0 to/from CH 1*		Ext Trig to CH 0 or CH 1†		* Measured on one channel with test signal applied to another channel, with same range setting on both channels. † 10 V_{pk-pk} signal applied to external trigger channel. Applies to all ranges on CH 0 and CH1.
	<–80 dB at 10 MHz <–60 dB at 100 MHz		<–80 dB at 10 MHz <–80 dB at 100 MHz		

Bandwidth and Transient Response				
Bandwidth (–3 dB)	Range (V_{pk-pk})	50 Ω	1 MΩ	Filter off. Bandwidth for 0 to 30 °C. Reduce by 0.25% per °C above 30 °C.
	All ranges except 0.1	340 MHz typical, 300 MHz min	300 MHz typical, 260 MHz min	
	0.1	165 MHz typical 135 MHz min	135 MHz typical 110 MHz min	
Rise/Fall Time, Typical	Range (V_{pk-pk})	50 Ω	1 MΩ	Filter off. ‡ 50 Ω terminator connected to front panel BNC connector.
	All ranges except 0.1	1.2 ns	1.4 ns‡	
	0.1	2.4 ns	2.8 ns‡	
Bandwidth Limit Filter	20 MHz Noise Filter			—
AC Coupling Cutoff (–3 dB), Typical	50 Ω	1 MΩ		50 Ω source assumed.
	106 kHz	12 Hz		

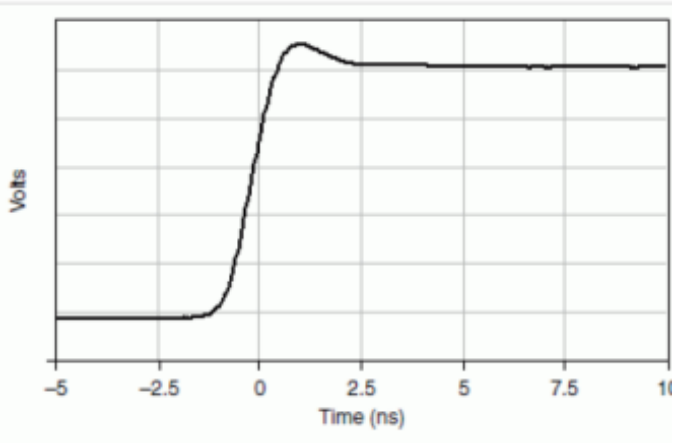
NI 5152 Frequency Response, 50 Ω, 1 V (Typical)



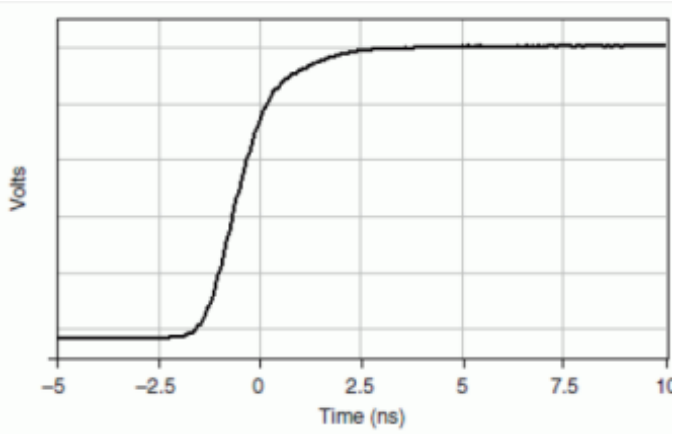
NI 5152 Frequency Response, 50 Ω, 1 V (Typical)



NI 5152 Step Response, 50 Ω, 10 Vpk-pk through 0.2 Vpk-pk Range (Typical)

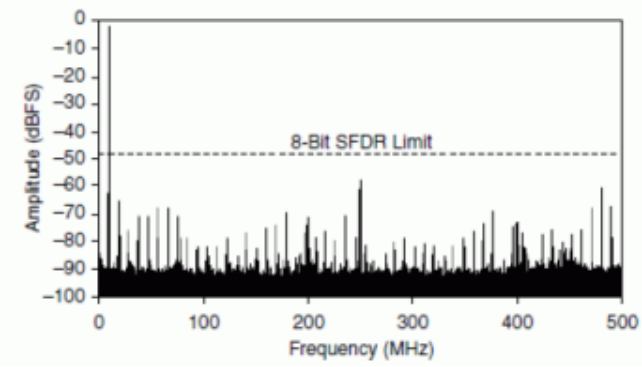


NI 5152 Step Response, 50 Ω, 0.1 Vpk-pk Range (Typical)

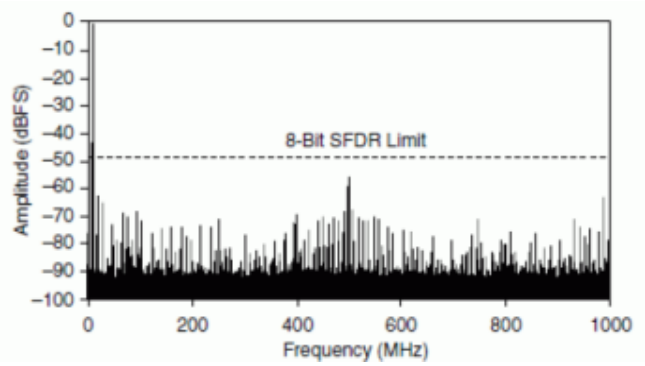


Specification	Value		Comments
Spectral Characteristics			
ENOB	Noise Filter On	Noise Filter Off	1 V_{pk-pk} range, 10 MHz, –1 dBFS input signal. Includes the 2nd through the 5th harmonics.
	7.3	7.1	
Signal to Noise and Distortion (SINAD), Typical	45 dB	43 dB	

NI 5152 Typical Dynamic Performance, 50 Ω, 1 Vpk-pk Range, 9.425 MHz,-1 dBFS Input Signal



NI 5152 TIS Typical Dynamic Performance, 50 Ω, 1 Vpk-pk Range, 9.425 MHz,-1 dBFS Input Signal



Specification	Value			Comments
RMS Noise	Range (V_{pk-pk})	Noise Filter On	Noise Filter Off	50 Ω terminator connected to input.
	0.1	240 μV_{rms} (0.24% FS)	320 μV_{rms} (0.32% FS)	
	0.2	480 μV_{rms} (0.24% FS)	600 μV_{rms} (0.30% FS)	
	0.4	960 μV_{rms} (0.24% FS)	1.12 mV_{rms} (0.28% FS)	
	1	2.4 mV_{rms} (0.24% FS)	2.6 mV_{rms} (0.26% FS)	
	2	4.8 mV_{rms} (0.24% FS)	6.0 mV_{rms} (0.30% FS)	
	4	9.6 mV_{rms} (0.24% FS)	11.2 mV_{rms} (0.28% FS)	
	10	24 mV_{rms} (0.24% FS)	26 mV_{rms} (0.26% FS)	
Skew				
Channel to Channel Skew, Typical	<100 ps			—

Horizontal

Sample Clock

Specification	Value			Comments
Sources	Internal: Onboard Clock (internal VCSO)* External: PFI 0 (front panel SMB connector)			* Internal Sample Clock is locked to the Reference Clock or derived from the onboard VCSO.
Onboard Clock (Internal VCSO)				
Sample Rate Range	Real-Time Sampling, Single-Shot	TIS† Mode, Single-Shot	Random Interleaved Sampling (RIS)‡ Mode	* Divide by *n* decimation used for all rates less than 1 GS/s. † TIS is a type of real-time sampling that is sometimes called ping-pong. ‡ RIS is a type of equivalent-time sampling.
	15.26 kS/s to 1 GS/s*	2 GS/s (Single channel only)	2 GS/s to 20 GS/s in increments of 1 GS/s (Repetitive waveforms only)	
Timebase Accuracy	Not Phase-Locked to Reference Clock	Phase-Locked to Reference Clock		Reference Clock accuracy is typically ±25 ppm across all temperatures in most PXI chassis. When using the NI PXI-6652 module for the reference clock, the accuracy is ±1 ppm. When using the NI PXI-6653 module, the accuracy is 0.05 ppm. ppm = parts per million (1×10^{-6})
	1 GHz ±30 ppm within ±3 °C of external calibration temperature	Equal to the reference clock accuracy		
Timebase Drift	Not Phase-Locked to Reference Clock	Phase-Locked to Reference Clock		—
	±7 ppm per °C	Equal to reference clock drift		
Sample Clock Delay Range	±1 Sample Clock period			—
Sample Clock Delay/Adjustment Resolution	≤5 ps			—
External Sample Clock				
Sources	PFI 0 (front panel SMB connector)			—
Frequency Range	350 MHz to 1 GHz			Divide by *n* decimation available where $1 \le n \le 65{,}535$. For more information about Sample Clock and decimation, refer to the *NI High-Speed Digitizers Help*.
Duty Cycle Tolerance	45% to 55%			—

Phase-Locked Loop (PLL) Reference Clock

Specification	Value	
Sources	NI PXI-5152	NI PCI-5152
	PXI_CLK10 (PXI backplane connector) PFI 0 (front panel SMB connector)	RTSI 7 PFI 0 (front panel SMB connector)
Frequency Range	1 MHz to 20 MHz in 1 MHz increments. Default of 10 MHz. The PLL Reference Clock frequency must be accurate to ±50 ppm.	
Duty Cycle Tolerance	45% to 55%	
Exported Reference Clock Destinations	NI PXI-5152	NI PCI-5152
	PXI_Trig <0..7> (backplane connector) PFI 1 (front panel SMB connector)	RTSI <0..7> PFI 1 (front panel SMB connector)

PFI 0 (Sample Clock and Reference Clock Input, Front Panel Connector)

Specification	Value
Input Voltage Range	Sine wave: 0.65 V_{pk-pk} to 2.8 V_{pk-pk} (0 dBm to 13 dBm)
Maximum Input Overload	7 V_{rms} with |Peaks| ≤ 10 V
Impedance	50 Ω
Coupling	AC

PFI 1 (Reference Clock Output, Front Panel Connector)

Specification	Value
Output Impedance	50 Ω
Logic Type	3.3 V CMOS
Maximum Drive Current	±24 mA

Trigger

Reference (Stop) Trigger

Specification	Value			Comments
Trigger Types	Edge, Window, Hysteresis, Digital, Immediate, and Software			Refer to the following sections and to the *NI High-Speed Digitizers Help* for more information about what sources are available for each trigger type.
Trigger Sources	NI PXI-5152		NI PCI-5152	
	CH 0, CH 1, TRIG, PFI <0..1> PXI_Trig <0..6>, PXI Star Trigger, and Software		CH 0, CH 1, TRIG, PFI <0..1>, RTSI <0..6>, and Software	
Time Resolution	TDC	Onboard Clock	External Clock	TDC = Time to Digital Conversion Circuit
	On	5 ps	N/A	
	Off	1 ns	External Clock Period	
Minimum Rearm Time	TDC		Rearm Time	Holdoff set to 0. Onboard sample clock at maximum rate.
	On		8 µs	
	Off		1 µs	
Holdoff	From Rearm time up to [(2^{32} – 1) × Sample Clock Period]			—
Trigger Delay	From 0 up to [(2^{35} – 1) – posttrigger samples] × (1/sample rate), in seconds			—
Analog Trigger (Edge, Window, and Hysteresis Trigger Types)				
Sources	CH 0, CH 1, TRIG (front panel BNC connectors)			—
Trigger Level Range	CH 0, CH 1		TRIG (External Trigger)	DC to 300 MHz
	100% FS		±5 V	
Voltage Resolution	8 bits (1 in 256)			—
Edge Trigger Sensitivity	CH 0, CH 1		TRIG (External Trigger)	DC to 300 MHz

Specification	Value		Comments
	10% FS	0.5 V_{pk-pk}	
Trigger Level Accuracy, Typical	CH 0, CH 1	TRIG (External Trigger)	—
	±5% FS up to 10 MHz	±1 V (±10% FS) up to 10 MHz)	
Trigger Jitter	≤10 ps rms typical, ≤20 ps rms maximum		Within ±5 °C of self-calibration temperature.
Trigger Filters	Low Frequency (LF) Reject	High Frequency (HF) Reject	—
	50 kHz	50 kHz	
Digital Trigger (Digital Trigger Type)			
Sources	NI PXI-5152	NI PCI-5152	—
	PXI_Trig <0..6> (backplane connector) PFI <0..1> (front panel SMB connectors) PXI Star Trigger (backplane connector)	RTSI <0..6> PFI <0..1> (front panel SMB connector)	
External Trigger Input (Front Panel Connector)			
Connector	BNC		—
Impedance	1 MΩ in parallel with a typical capacitance of 22 pF		—
Coupling	AC, DC		—
AC Coupling Cutoff (–3 dB)	12 Hz		—
Input Voltage Range	±5 V		—
Maximum Input Overload	|Peaks| ≤42 V		—
PFI 0 and PFI 1 (Programmable Function Interface, Front Panel Connectors)			
Connector	SMB jack		—
Direction	Bidirectional		—
As an Input (Trigger)			
Destination	Start Trigger (Acquisition Arm) Reference (Stop) Trigger Arm Reference Trigger Advance Trigger		—
Input Impedance	150 kΩ		—
V_{IH}	2.0 V		—
V_{IL}	0.8 V		—
Maximum Input Overload	–0.5 V to 5.5 V		—
Maximum Frequency	25 MHz		—
As an Output (Event)			
Sources	Start Trigger (Acquisition Arm) Reference (Stop) Trigger End of Record Done (End of Acquisition) Probe Compensation (1 kHz, 50% duty cycle square wave, PFI 1 only)		—
Output Impedance	50 Ω		—
Logic Type	3.3 V CMOS		—
Maximum Drive Current	±24 mA		—
Maximum Frequency	25 MHz		—

TClk Specifications

National Instruments TClk synchronization method and the NI-TClk driver are used to align the sample clocks on any number of SMC-based modules in a chassis. For more information about TClk synchronization, refer to the *NI-TClk Synchronization Help*, which is located within the *NI High-Speed Digitizers Help*.

- Specifications are valid for any number of PXI modules installed in one NI PXI-1042 chassis.
- All parameters set to identical values for each SMC-based module.
- Sample Clock set to 1 GS/s and all filters are disabled.
- For other configurations, including multichassis systems, contact NI Technical Support at ni.com/support.

Note Although you can use NI-TClk to synchronize nonidentical modules, these specifications apply only to synchronizing identical modules.

Specification	Value	Comments
Intermodule SMC Synchronization Using NI-TClk for Identical Modules (Typical)		
Skew	500 ps	Caused by clock and analog path delay differences. No manual adjustment performed.
Average Skew After Manual Adjustment	≤5 ps	For information about manual adjustment, refer to the *Synchronization Repeatability Optimization* topic in the *NI-TClk Synchronization Help*. For additional help with the adjustment process, contact NI Technical Support at ni.com/support.
Sample Clock Delay/Adjustment Resolution	≤5 ps	—

Waveform Specifications

Specification	Value		Comments
Onboard Memory Size	Real-Time and RIS Modes	Real-Time TIS Mode	* NI PXI-5152 only
	8 MB Standard (8 MS per channel)	8 MB Standard (8 MS)	
	64 MB Option (64 MS per channel)	64 MB Option (64 MS)	
	256 MB Option (256 MS per channel)	256 MB Option (256 MS)	
	512 MB Option* (512 MS per channel)	512 MB Option* (512 MS)	
Minimum Record Length	1 Sample		—
Number of Pretrigger Samples	Zero up to full record length		Single-record mode and multiple-record mode.
Number of Posttrigger Samples	Zero up to full record length		Single-record mode and multiple-record mode.
Maximum Number of Records in Onboard Memory	Memory Option	Real-Time Sampling Mode	† It is possible to exceed these numbers if you fetch records while acquiring data. For more information, refer to the *NI High-Speed Digitizers Help*.
	8 MB per channel	32,768†	
	64 MB per channel	100,000†	
	256 MB per channel	100,000†	
Allocated Onboard Memory per Record	[(Record length × 1 byte/sample) + 400 bytes] rounded up to next multiple of 128 bytes		—

Calibration

Specification	Value
Self-Calibration	Self-calibration is done on software command. The calibration corrects for gain, offset, triggering, and timing errors for all input ranges.
External Calibration (Factory Calibration)	The external calibration calibrates the VCSO and the voltage reference. Appropriate constants are stored in nonvolatile memory.
Interval for External Calibration	2 years
Warm-Up Time	15 minutes

Power

Specification	Typical Value	
+3.3 VDC	NI PXI-5152	NI PCI-5152
	1.1 A	2.5 A
+5 VDC	1.9 A	2.4 A
+12 VDC	500 mA	200 mA
–12 VDC	210 mA	0 A
Total Power	21.65 W	22.65 W

Software

Specification	Value
Driver Software	NI-SCOPE 3.2 or later for NI PXI-5152 NI-SCOPE 3.3 or later for NI PCI-5152

Specification	Value
	NI-SCOPE is an IVI-compliant driver that allows you to configure, control, and calibrate the NI 5152. NI-SCOPE provides application programming interfaces for many development environments.
Application Software	NI-SCOPE provides programming interfaces, documentation, and examples for the following application development environments: • LabVIEW • LabWindows™/CVI™ • Measurement Studio • Microsoft Visual C/C++ • Microsoft Visual Basic
Interactive Soft Front Panel and Configuration	The NI-SCOPE Soft Front Panel version 2.5 or later supports interactive control of the NI 5152. The NI-SCOPE Soft Front Panel is included on the NI-SCOPE CD. National Instruments Measurement & Automation Explorer (MAX) also provides interactive configuration and test tools for the NI 5152. MAX is included on the NI-SCOPE CD.

Environment

NI PXI-5152

Note To ensure that the NI PXI-5152 cools effectively, follow the guidelines in the *Maintain Forced-Air Cooling Note to Users* included in the hardware kit. The NI PXI-5152 is intended for indoor use only.

Specification	Value
Operating Temperature	0 ºC to +55 ºC in all NI PXI chassis except the following: 0 ºC to +45 ºC when installed in an NI PXI-1000/B or PXI-101*x* chassis. Meets IEC 60068-2-1 and IEC 60068-2-2.
Storage Temperature	–40 ºC to +71 ºC. Meets IEC 60068-2-1 and IEC 60068-2-2.
Operating Relative Humidity	10% to 90%, noncondensing. Meets IEC 60068-2-56.
Storage Relative Humidity	5% to 95%, noncondensing. Meets IEC 60068-2-56.
Operating Shock	30 g, half-sine, 11 ms pulse. Meets IEC 60068-2-27. Test profile developed in accordance with MIL-PRF-28800F.
Storage Shock	50 g, half-sine, 11 ms pulse. Meets IEC 60068-2-27. Test profile developed in accordance with MIL-PRF-28800F.
Operating Vibration	5 Hz to 500 Hz, 0.31 g_{rms}. Meets IEC 60068-2-64.
Storage Vibration	5 Hz to 500 Hz, 2.46 g_{rms}. Meets IEC 60068-2-64. Test profile exceeds requirements of MIL-PRF-28800F, Class 3.
Altitude	2,000 m maximum (at 25 °C ambient temperature)
Pollution Degree	2

NI PCI-5152

Note To ensure that the NI PCI-5152 cools effectively, make sure that the chassis in which it is used has active cooling that provides at least some airflow across the PCI card cage. To maximize airflow and extend the life of the device, leave any adjacent PCI slots empty. Refer to the *Maintain Forced-Air Cooling Note to Users* included in the NI PCI-5152 kit for important cooling information. The NI PCI-5152 is intended for indoor use only.

Specification	Value
Operating Temperature	0 ºC to +45 ºC. Meets IEC 60068-2-1 and IEC 60068-2-2.
Storage Temperature	–40 ºC to +70 ºC. Meets IEC 60068-2-1 and IEC 60068-2-2.
Operating Relative Humidity	10% to 90%, noncondensing. Meets IEC 60068-2-56.
Storage Relative Humidity	5% to 95%, noncondensing. Meets IEC 60068-2-56.
Storage Shock	50 g, half-sine, 11 ms pulse. Meets IEC 60068-2-27. Test profile developed in accordance with MIL-PRF-28800F.
Storage Vibration	5 Hz to 500 Hz, 2.46 g_{rms}. Meets IEC 60068-2-64. Test profile exceeds requirements of MIL-PRF-28800F, Class 3.
Altitude	2,000 m maximum (at 25 °C ambient temperature)
Pollution Degree	2

Safety, Electromagnetic Compatibility, and CE Compliance

Safety Standards

This product is designed to meet the requirements of the following standards of safety for electrical equipment for measurement, control, and laboratory use:

- IEC 61010-1, EN 61010-1
- UL 61010-1, CSA 61010-1

Note For UL and other safety certifications, refer to the product label or the *Online Product Certification* section.

Electromagnetic Compatibility

This product meets the requirements of the following EMC standards for electrical equipment for measurement, control, and laboratory use:

- EN 61326 (IEC 61326): Class A emissions; Basic immunity

- EN 55011 (CISPR 11): Group 1, Class A emissions
- AS/NZS CISPR 11: Group 1, Class A emissions
- FCC 47 CFR Part 15B: Class A emissions
- ICES-001: Class A emissions

Note For the standards applied to assess the EMC of this product, refer to the *Online Product Certification* section.
Note For EMC compliance, operate this device with RG223/U or equivalent shielded cable. Operate according to product documentation.

CE Compliance

This product meets the essential requirements of applicable European Directives, as amended for CE marking, as follows:

- 2006/95/EC; Low-Voltage Directive (safety)
- 2004/108/EC; Electromagnetic Compatibility Directive (EMC)

Online Product Certification

Refer to the product Declaration of Conformity (DoC) for additional regulatory compliance information. To obtain product certifications and the DoC for this product, visit ni.com/certification, search by module number or product line, and click the appropriate link in the Certification column.

Environmental Management

National Instruments is committed to designing and manufacturing products in an environmentally responsible manner. NI recognizes that eliminating certain hazardous substances from our products is beneficial not only to the environment but also to NI customers.

For additional environmental information, refer to the *NI and the Environment* Web page at ni.com/environment. This page contains the environmental regulations and directives with which NI complies, as well as other environmental information not included in this document.

Waste Electrical and Electronic Equipment (WEEE)

EU Customers At the end of the product life cycle, all products *must* be sent to a WEEE recycling center. For more information about WEEE recycling centers, National Instruments WEEE initiatives, and compliance with WEEE Directive 2002/96/EC on Waste Electrical and Electronic Equipment, visit ni.com/environment/weee.htm.

电子信息产品污染控制管理办法 （中国 RoHS）

中国客户 National Instruments 符合中国电子信息产品中限制使用某些有害物质指令 (RoHS)。
关于 National Instruments 中国 RoHS 合规性信息，请登录 ni.com/environment/rohs_china。
(For information about China RoHS compliance, go to ni.com/environment/rohs_china.)

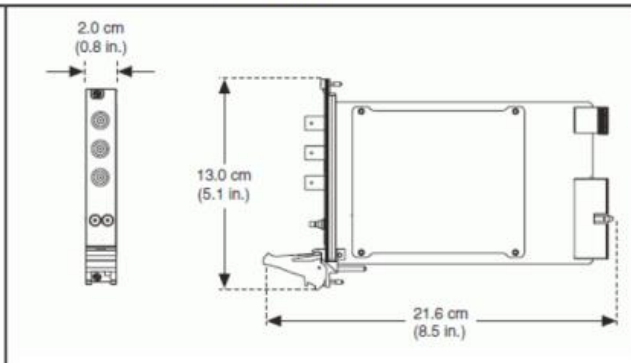
Physical

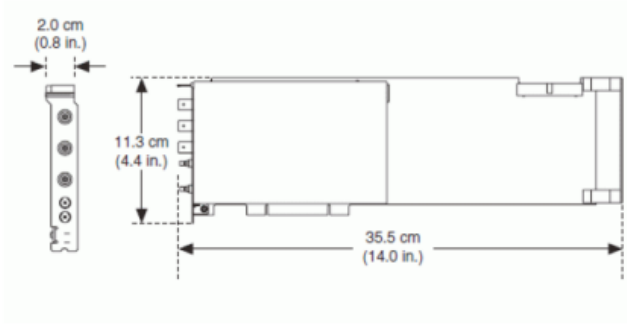
Front Panel Connectors

Label	Function	Connector Type
CH 0	Analog input	BNC female
CH 1	Analog input	BNC female
TRIG	External Trigger	BNC female
PFI 0	Sample Clock Input, Reference Clock Input Digital Trigger Input/Output	SMB jack
PFI 1	Reference Clock Output, Digital Trigger Input/Output	SMB jack

Dimensions and Weight

NI PXI-5152	
Dimensions	3U, One slot, PXI/cPCI Module 21.6 × 2.0 × 13.0 cm (8.5 × 0.8 × 5.1 in.)

NI PXI-5152	
	
Weight	462 g (16.3 oz)

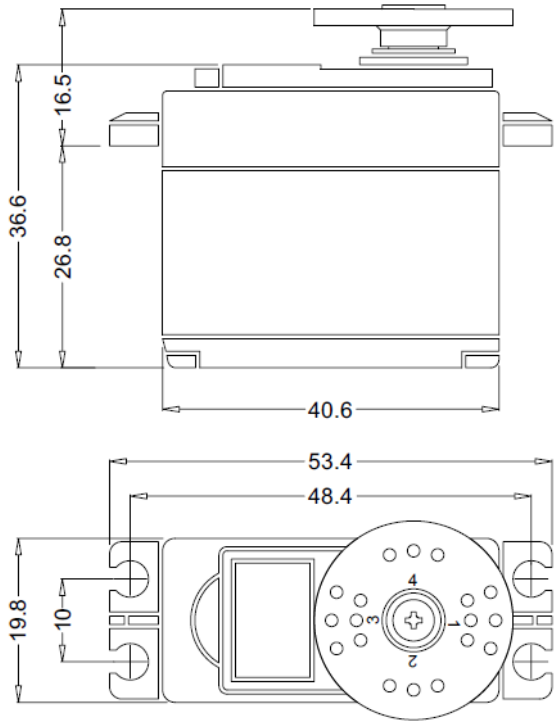
NI PCI-5152	
Dimensions	35.5 × 2.0 × 11.3 cm (14.0 × 0.8 × 4.4 in.) 
Weight	445 g (15.7 oz)

Hoja de características: Servomotor Hitec HS-422

ANNOUNCED SPECIFICATION OF HS-422 STANDARD DELUXE SERVO

1 .TECHNICAL VALUES

CONTROL SYSTEM:	:+ PULSE WIDTH CONTROL ISOOusec NEUTRAL	
OPERATING VOLTAGE RANGE:	:4.8V TO +60ºC	
OPERATING TEMPERATURE RANGE:	:- 20 T0 +60 ºC	
TEST VOLTAGE:	:AT 4.8V	AT 6.0V
OPERATING SPEED:	:0.21sec/600 AT NO LOAD	0.16sec/60C AT NO LOAD
STALL TORQUE:	:3.3kg.cm(45.820z.in)	4.1kg.cm(56.930z.in)
OPERATING ANGLE:	:45ºONE SIDE PULSE TRAVELING 400usec	
DIRECTION:	:CLOCK WISE!PULSE TRAVELING 1500 TO 19COusec	
CURRENT DRAIN:	:8mA/lDLE AND 150mA/NO LOAD RUNNING	
DEAD BAND WIDTH:	:8usec	
CONNECTOR WIRE LENGTH:	:300mm(11.81in)	
DIMENSIONS:	:40.6x19.8x36.6mm(1.59xo.77x1.44in)	
WEIGHT:	:45.5g(1.60z)	



2.FEATURES

3-POLE FERRITE MOTOR
LONG LIFE POTENTIOMETER
DUAL OILITE BUSHING
INDIRECT POTENTIOMETER DRIVE

3.APPLICAT10NS

AIRCRAFT 20-60 SIZE
30 SIZE HELICOPTERS
STEERING AND THROTTLE SERVO FOR CARS
TRUCK AND BOATS

Código final del programa

```
function varargout = codigo(varargin)
% CODIGO MATLAB code for codigo.fig
%      CODIGO, by itself, creates a new CODIGO or raises the existing
%      singleton*.
%
%      H = CODIGO returns the handle to a new CODIGO or the handle to

%      the existing singleton*.
%
%      CODIGO('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CODIGO.M with the given input arguments.
%
%      CODIGO('Property','Value',...) creates a new CODIGO or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before codigo_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to codigo_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 codigo

% Last Modified by GUIDE v2.5 29-Jul-2015 11:00:34

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @codigo_OpeningFcn, ...
                   'gui_OutputFcn',  @codigo_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 codigo is made visible.
function codigo_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 codigo (see VARARGIN)

% Choose default command line output for codigo

handles.output = hObject;

handles.vector_energia_media=zeros(1,360);
```

SERVOMOTOR

```
% Se crea el "Objeto arduino"

a=arduino('COM10','uno','Libraries', 'Servo');
s=servo(a,4)
clear s;
s = servo(a, 4, 'MinPulseDuration', 0.6e-3, 'MaxPulseDuration', 2.4e-3)
ang_inic=0;
ang_fin=1;
angle=ang_inic;
writePosition(s, angle);
cont_angulo=1; % Contador que indica todas las posiciones del servomotor
```

TARJETA DE ADQUISICIÓN

```
% Se crea el "Controlador de instrumentos de Matlab (Matlab Instrument
Object)" y se conecta

% * TARJETA REAL*
%ictObj = icdevice('niscope.mdd', 'DAQ::Dev1');

% *TARJETA EN MODO SIMULACIÓN*
ictObj = icdevice('niscope.mdd', 'DAQ::Dev1','OptionString','simulate=true');

connect(ictObj);

% Estado de la tarjeta

set(handles.text_valor_estado_tarjeta,'String',ictObj.status)
set(handles.text_n_senales,'String',' ');

%  ** Configuración del Scope **
```

```
configuration = ictObj.Configuration;
invoke(configuration, 'autosetup');

%  ** Configuración vertical: canal 0 **

Range = 0.01;
Offset = 0;
Coupling = 1;
ProbeAttenuation = 1;
Inputimpedance=50;
Maxinputfrequency=-1;

invoke(ictObj.Configurationfunctionsvertical, 'configurevertical', '0', Range,
Offset, Coupling, ProbeAttenuation, true);
invoke(ictObj.Configurationfunctionsvertical,'configurechancharacteristics','0
',Inputimpedance,Maxinputfrequency);

% NOTA: El true corresponde a la variable enabled

% ** Configuración horizontal: canal 0 **

MinSampleRate=1e9;
MinNumPts=1000;
RefPosition=20;
NumRecords=1;
EnforceRealTime=1;

invoke(ictObj.Configurationfunctionshorizontal,'configurehorizontaltiming',Min
SampleRate,MinNumPts,RefPosition,NumRecords,EnforceRealTime);

% ** Configuración del trigger: canal 0 **

TriggerSource='0';  % Es el mismo que el canal de entrada
Level=2e-3;
Slope=1;
TriggerCoupling=1;
HoldOff=0;
Delay=0;

invoke(ictObj.Configurationfunctionstrigger,'configuretriggeredge',TriggerSour
ce,Level,Slope,TriggerCoupling,HoldOff,Delay);

% Se prepara la información de la señal

numChannels = 1;
channelList = '0';
numSamples = 1000;
```

```
    waveformInfo.absoluteInitialX = 0;
    waveformInfo.relativeInitialX = 0;
    waveformInfo.xIncrement = 0;
    waveformInfo.actualSamples = 0;
    waveformInfo.offset = 0;
    waveformInfo.gain = 0;
    waveformInfo.reserved1 = 0;
    waveformInfo.reserved2 = 0;


% Obtención de la señal

waveformArray = zeros(numChannels * numSamples,NumRecords );
TimeOut = 30; % (segundos)


% Adquisición de la primera señal (señal de inicialización)

    invoke(ictObj.Acquisition, 'initiateacquisition');
    [waveformArray, waveformInfo] = invoke(ictObj.Acquisition, 'fetch',
channelList,...
    TimeOut, numSamples, waveformArray, waveformInfo);
    senal=waveformArray;

  fig1=handles.axes1;
  plot(fig1,senal');

  set(handles.text_info,'String','Señal de inicialización');
  set(handles.text_n_senales,'String',' ');

  set(handles.edit_n_muestras,'String',numSamples);

  set(handles.text_energia_descarga,'String',' ');
  set(handles.text_posicion_descarga,'String',' ');
  set(handles.text_energia,'String',' ');
  set(handles.text_posicion,'String',0);
  set(handles.text_range,'String',' ');
  set(handles.text_n_senales_calibr,'String',' ');

  fig2=handles.axes2;
  xlim(fig2,[0 180]);

  % ** Variables globales **

  handles.tarjeta=ictObj;
  handles.arduino=a;
  handles.servo=s;
  handles.cont_angulo=cont_angulo;
  handles.ang_inic=ang_inic;
  handles.ang_fin=ang_fin;
```

```
  handles.angle=angle;

  handles.Range=Range;
  handles.Offset=Offset;
  handles.Coupling=Coupling;
  handles.ProbeAttenuation=ProbeAttenuation;

  handles.MinSampleRate=MinSampleRate;
  handles.MinNumPts=MinNumPts;
  handles.RefPosition=RefPosition;
  handles.NumRecords=NumRecords;
  handles.EnforceRealTime=EnforceRealTime;

  handles.TriggerSource=TriggerSource;
  handles.Level=Level;
  handles.Slope=Slope;
  handles.TriggerCoupling=TriggerCoupling;
  handles.HoldOff=HoldOff;
  handles.Delay=Delay;

  handles.numChannels = numChannels;
  handles.channelList = channelList;
  handles.numSamples = numSamples;
  handles.waveformInfo = waveformInfo;
  handles.waveformArray = waveformArray;
  handles.TimeOut = TimeOut;

  handles.num_senales=str2double(get(handles.edit_n_senales_def,'String'));
  handles.paso_servo=(str2double(get(handles.edit_paso_servo,'String')))/180;

  handles.movimiento=1;  % Se va a emplear para que cuando se interrumpa
                         % la recogida de señales el servomotor no pase a
                         % la siguiente posición de giro

% Choose default command line output for p_gui_todojunto
handles.output = hObject;


% Update handles structure
guidata(hObject, handles);

% UIWAIT makes codigo wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
```

```
function varargout = codigo_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
```

```
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

ELECCIÓN DEL NÚMERO DE MUESTRAS

```
function edit_n_muestras_Callback(hObject, eventdata, handles)
% hObject    handle to edit_n_muestras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_n_muestras as text
%        str2double(get(hObject,'String')) returns contents of edit_n_muestras
as a double

handles.numSamples=str2double(get(hObject,'String'));

guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function edit_n_muestras_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_n_muestras (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% 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
```

CALIBRACIÓN HORIZONTAL

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

% Hint: get(hObject,'Value') returns toggle state of Toggle Button_calibracion
```

```
ictObj=handles.tarjeta;
a=handles.arduino;
s=handles.servo;
cont_angulo=handles.cont_angulo;
ang_inic=handles.ang_inic;
ang_fin=handles.ang_fin;
angle=handles.angle;

Range=handles.Range;
Offset=handles.Offset;
Coupling=handles.Coupling;
ProbeAttenuation=handles.ProbeAttenuation;

MinSampleRate=handles.MinSampleRate;
RefPosition=handles.RefPosition;
NumRecords=handles.NumRecords;
EnforceRealTime=handles.EnforceRealTime;

TriggerSource=handles.TriggerSource;
Level=handles.Level;
Slope=handles.Slope;
TriggerCoupling=handles.TriggerCoupling;
HoldOff=handles.HoldOff;
Delay=handles.Delay;

numChannels=handles.numChannels;
channelList=handles.channelList;
numSamples=handles.numSamples;

waveformInfo=handles.waveformInfo;
waveformArray=handles.waveformArray;
TimeOut=handles.TimeOut;

% Posición del servomotor. Se lee la posición para mostrarla en el Guide

current_pos = readPosition(s);
current_pos = current_pos*180;  %Se expresa la posición en grados
set(handles.text_posicion,'String',current_pos);
set(handles.text_n_senales,'String',' ');

set(handles.text_energia,'String',' ');
set(handles.text_range,'String',' ');
set(handles.text_n_senales_calibr,'String',' ');

m=1; % Contador de las señales que se van recogiendo en la calibración

button_state = get(hObject,'Value'); % Se mira el estado del toggle button
                                     % de "Calibración horizontal"
```

```
matriz_calibracion=zeros(500,numSamples); % Hemos considerado inicialmente
```

```
                                        % que como mucho vamos a recoger
                                        % 500 señales de calibración
while button_state == get(hObject,'Max')

        waveformArray = zeros(NumRecords,numChannels * numSamples);
    invoke(ictObj.Acquisition, 'initiateacquisition');
    [waveformArray, waveformInfo] = invoke(ictObj.Acquisition, 'fetch',
channelList,...
    TimeOut, numSamples, waveformArray, waveformInfo);

    senal=waveformArray;
    vector_range(m)=max(senal);
    matriz_calibracion(m,:)=senal;
    m=m+1;

    fig1=handles.axes1;
    plot(fig1,senal)
    axis(fig1,[0 numSamples -Range Range]);
    pause(0.2)

    set(handles.text_info,'String','Recogida de la señal de calibración');

    button_state = get(hObject,'Value');

    if  button_state == get(hObject,'Min')

        Range=(max(vector_range))*1.4;
        indice_rango_max=min(find(vector_range==max(vector_range)));


        invoke(ictObj.Configurationfunctionsvertical, 'configurevertical',
'0', Range, Offset, Coupling, ProbeAttenuation, true);

invoke(ictObj.Configurationfunctionshorizontal,'configurehorizontaltiming',Min
SampleRate,numSamples,RefPosition,NumRecords,EnforceRealTime);

        plot(fig1,matriz_calibracion(indice_rango_max,:));
        axis(fig1,[0 numSamples -Range Range]);

        set(handles.text_info,'String','Calibración terminada');
        set(handles.text_range,'String',Range);
        set(handles.text_n_senales,'String',' ');

        set(handles.text_n_senales_calibr,'String',m-1);

        handles.tarjeta=ictObj;

        handles.ang_inic=ang_inic;

        handles.ang_fin=ang_fin;

        handles.angle=angle;
```

```
        handles.numSamples=numSamples;
        handles.waveformInfo=waveformInfo;
        handles.waveformArray=waveformArray;
        handles.Range=Range;
        handles.matriz_calibracion=matriz_calibracion;

        save('datos_calibracion','-struct','handles','matriz_calibracion');

        guidata(hObject, handles);
    end

end
```

RECOGIDA DE SEÑALES

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

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

ictObj=handles.tarjeta;
a=handles.arduino;
s=handles.servo;
cont_angulo=handles.cont_angulo;
paso_servo=(str2double(get(handles.edit_paso_servo,'String')))/180;
ang_inic=handles.ang_inic;
ang_fin=handles.ang_fin;
angle=handles.angle;

Range=handles.Range;
Offset=handles.Offset;
Coupling=handles.Coupling;
ProbeAttenuation=handles.ProbeAttenuation;

MinSampleRate=handles.MinSampleRate;
RefPosition=handles.RefPosition;
NumRecords=handles.NumRecords;
EnforceRealTime=handles.EnforceRealTime;

TriggerSource=handles.TriggerSource;
Level=handles.Level;
```

```
Slope=handles.Slope;
```

```
TriggerCoupling=handles.TriggerCoupling;
```

```
HoldOff=handles.HoldOff;
Delay=handles.Delay;

numChannels=handles.numChannels;
channelList=handles.channelList;
numSamples=handles.numSamples;
waveformInfo=handles.waveformInfo;
TimeOut=handles.TimeOut;

num_senales=str2double(get(handles.edit_n_senales_def,'String'));

% Se lee el estado del Toggle Button para poder iniciar la recogida

button_state = get(hObject,'Value');

if button_state==get(hObject,'Max')
    handles.movimiento=1;  % Indica al servomotor si debe continuar con
                           % el movimiento o detenerse
    angle=0;
    writePosition(s, angle);
end

matriz_todas_senales=zeros(((1/paso_servo)+1)*num_senales,numSamples);


while(handles.movimiento==1)

        % Posición del servomotor

        current_pos = readPosition(s);
        current_pos = current_pos*180; %Se expresa la posición en grados
        set(handles.text_posicion,'String',current_pos);
        set(handles.text_n_senales,'String',' ');

        % Calibración automática del rango: Se recogen cinco señales

          for k =1:5

                waveformArray = zeros(NumRecords,numChannels * numSamples);
                invoke(ictObj.Acquisition, 'initiateacquisition');
                [waveformArray, waveformInfo] = invoke(ictObj.Acquisition,
'fetch', channelList,...
                TimeOut, numSamples, waveformArray, waveformInfo);
                senal_cal=waveformArray;

                vector_range(k)=max(senal_cal);

          end
```

```
                Range=(max(vector_range))*1.4;
```

```
                %Level=0.6*Range;
                Level=5e-3;

                invoke(ictObj.Configurationfunctionsvertical,
'configurevertical', '0', Range, Offset, Coupling, ProbeAttenuation, true);

invoke(ictObj.Configurationfunctionstrigger,'configuretriggeredge',TriggerSour
ce,Level,Slope,TriggerCoupling,HoldOff,Delay);


        % Recogida del número de señales definido, y cálculo de la energía a
        % tiempo real


        for k =1:num_senales


            fig2=handles.axes2;
            xlim(fig2,[0 180]);

            button_state = get(hObject,'Value');

            if button_state==get(hObject,'Max')

                set(handles.text_info,'String','Número de la señal
recogida:');
                set(handles.text_n_senales,'String',num2str(k));

                waveformArray = zeros(NumRecords,numChannels * numSamples);
                invoke(ictObj.Acquisition, 'initiateacquisition');
                [waveformArray, waveformInfo] = invoke(ictObj.Acquisition,
'fetch', channelList,...
                TimeOut, numSamples, waveformArray, waveformInfo);
                senal(k,:)=waveformArray;

                fig1=handles.axes1;
                plot(fig1,senal(k,:)')
                pause(0.1)
                axis([0 numSamples -Range Range]);

                senal_potencia(k,:)=(senal(k,:)).^2;
            end

        end


        potencia=senal_potencia';  % Se calcula la transpuesta para expresarlo
                                   % como un vector columna
```

```
        energia_cada_senal=trapz(potencia*(1/MinSampleRate)); % Energía de
```

```
                                                                % cada
                                                                % señal en una
                                                                % posición
        energia_media=(sum(energia_cada_senal')/num_senales); % Energía media
                                                                % en una
                                                                % posición
    set(handles.text_energia,'String',energia_media);

        vector_energia_media(cont_angulo)=energia_media;

        vector_angulo(cont_angulo)=angle*180;

        % Se guardan todas las señales de cada posición en una matriz de tal
        % manera que en una matriz estén todas las señales de todas las
        % posiciones

         matriz_todas_senales(1+(num_senales*(cont_angulo-
1)):num_senales*cont_angulo,:)=senal;


         handles.matriz_todas_senales=matriz_todas_senales;
         handles.vector_energia_media=vector_energia_media;
         handles.vector_angulo=vector_angulo;
         handles.MinSampleRate=MinSampleRate;
         handles.num_senales=num_senales;
         handles.paso_servo=paso_servo;
     %
**********************************************************************
**********

      % *** CÁLCULO DE PICOS MÁXIMOS Y ESTUDIO ESTADÍSTICO ***

     % Vector de picos máximos de todas las señales

                for k=1:num_senales*cont_angulo

                    vector_picos_max(k)=max(matriz_todas_senales(k,:));

                end

     % Se crea una matriz en la que las filas se corresponden con las
     % posiciones del servomotor y las columnas con los picos de las
     % señales en esa posición

                j=1;

                for k=1:cont_angulo
                    for i=1:num_senales

                        matriz_picos_max(k,i)=vector_picos_max(1,j);

                        if j<=((1/paso_servo)+1)*num_senales  % Es el nº de
```

```
                                                                % señales
                                                                % totales
                            j=j+1;
                        end
                    end
                end


      % Gráfico con los picos máximos medios y la desviación típica en cada
      % posición

        fig2=handles.axes2;

                    for k=1:cont_angulo
                        pico_max_medio(k)=mean(matriz_picos_max(k,:));
                        desv_tipica(k)=std(matriz_picos_max(k,:),0);
                        % El 0 indica que calcula la desviación corregida

                        valor_max(k)=pico_max_medio(k)+desv_tipica(k);
                        valor_min(k)=pico_max_medio(k)-desv_tipica(k);

                        plot(fig2,[vector_angulo(k)
vector_angulo(k)],[valor_max(k) valor_min(k)]);
                        hold on
                        plot(fig2,[vector_angulo(k)
vector_angulo(k)],[valor_max(k) valor_min(k)],'*');
                        hold on

                        % Se ha puesto dos veces la misma gráfica para que
                        % primero se una con una recta, y después, se añadan
                        % los asteriscos en los extremos


                    end

          plot(fig2,vector_angulo,pico_max_medio,'o');
          axis(fig2,[0 180 -Range Range]);

          set(handles.text_info_estadistico,'String','Pico máx medio y
desviación típica en cada posición (°)');



     %
*********************************************************************************
*********

    button_state = get(hObject,'Value');
```

```
    if button_state==get(hObject,'Max')
```

```
        cont_angulo=cont_angulo+1;
        angle=angle+paso_servo;



            if angle>ang_fin
                posicion_final=length(vector_angulo);

                % *** POSICIÓN DE ENERGÍA MÁXIMA CALCULADA POR EL PROGRAMA
                % TENIENDO EN CUENTA LA ENERGÍA MEDIA DE CADA POSICIÓN.
                % RESULTADO NO EXACTO. ***

                energia_maxima=max(vector_energia_media);
                indice_vector=find(vector_energia_media==energia_maxima);


set(handles.text_energia_descarga,'String',vector_energia_media(indice_vector)
);

set(handles.text_posicion_descarga,'String',vector_angulo(indice_vector));
                set(handles.text_info,'String','Ha finalizado la recogida en
todas la posiciones. Se muestran los datos de la aprox de la descarga
parcial');
                set(handles.text_n_senales,'String',' ');


                % ** GRÁFICA: Energía media en cada posición

                fig1=handles.axes1;
                plot(fig1,vector_angulo,vector_energia_media);
                set(handles.text_info,'String','Energía media en cada posición
(°)');

                set(handles.text_ejey,'String',' ');
                set(handles.text_ejex,'String',' ');

                angle=ang_fin;
                handles.movimiento=0;

                set(handles.Toggle Button_recoger_senales,'Value',0);




               % *** RESULTADO MÁS CORRECTO. ANÁLISIS DE LOS PICOS DE TODAS
               %  LAS SEÑALES DE CADA POSICIÓN ***
```

```
               % Para obtener un resultado más correcto se analizan los picos
```

```
% de cada señal, ya que al calcular la energía media no se sabe
% si el valor obtenido es afectado por fenómenos distintos a
% las descargas parciales.
% De esta manera se puede ver dónde tiende a encontrarse el
% pico máximo y si hay algún pico que se salga fuera de su
% valor normal, y por tanto, se deba a otros fenómenos


% *** CÁLCULO DE PICOS MÁXIMOS Y ESTUDIO ESTADÍSTICO ***

% Vector de picos máximos de todas las señales

 for k=1:num_senales*posicion_final

     vector_picos_max(k)=max(matriz_todas_senales(k,:));

 end

% Se crea una matriz en la que las filas se corresponden con
% las posiciones del servomotor y las columnas con los picos de
% las señales en esa posición

 j=1;

 for k=1:posicion_final
     for i=1:num_senales
         matriz_picos_max(k,i)=vector_picos_max(1,j);
         if j<=num_senales/paso_servo  % Es el nº de señales
                                       % totales
             j=j+1;
         end
     end
 end

% Gráfico con con los picos máximos medios y la desviación típica
% en cada posición


fig2=handles.axes2;

      for k=1:posicion_final
         pico_max_medio(k)=mean(matriz_picos_max(k,:));
         desv_tipica(k)=std(matriz_picos_max(k,:),0);
         % El 0 indica que calcula la desviación corregida

         valor_max(k)=pico_max_medio(k)+desv_tipica(k);
         valor_min(k)=pico_max_medio(k)-desv_tipica(k);
```

```
                    plot(fig2,[vector_angulo(k)
vector_angulo(k)],[valor_max(k) valor_min(k)]);
```

```
                        hold on
                        plot(fig2,[vector_angulo(k)
vector_angulo(k)],[valor_max(k) valor_min(k)],'*');
                        hold on

                        % Se ha puesto dos veces la misma gráfica para que
                        % primero se una con una recta, y después, se añadan
                        % los asteriscos en los extremos

                    end

             handles.matriz_picos_max=matriz_picos_max;
             handles. pico_max_medio= pico_max_medio;
             handles.desv_tipica=desv_tipica;
             handles.valor_max=valor_max;
             handles.valor_min=valor_min;

            plot(fig2,vector_angulo,pico_max_medio,'o');
            axis(fig2,[0 180 -Range Range]);

            set(handles.text_info_estadistico,'String','Pico máx medio y
desviación típica en cada posición (°)');


            save('datos','-
struct','handles','matriz_todas_senales','vector_energia_media','matriz_picos_
max','pico_max_medio','desv_tipica','valor_max','valor_min','MinSampleRate','n
um_senales','vector_angulo','paso_servo','numSamples');

            else

                set(handles.text_info,'String',' Recogida de la señal
terminada. Se cambia a la nueva posición (°):');
                set(handles.text_n_senales,'String',angle*180);

            end
    end


     if button_state==get(hObject,'Min')

          handles.movimiento=0;
          angle=0;
          cont_angulo=1;

          set(handles.text_info,'String','AVISO: Se ha pulsado el botón de
interrumpir');
          set(handles.text_n_senales,'String','');
          set(handles.text_energia,'String','');
          set(handles.text_posicion,'String',angle*180);
```

```
    end

% Se mueve el servomotor

pause(2)

writePosition(s, angle);

end

handles.tarjeta=ictObj;
handles.cont_angulo=cont_angulo;
handles.angle=angle;
handles.ang_inic=ang_inic;
handles.ang_fin=ang_fin;
handles.paso_servo=paso_servo;

handles.num_senales=num_senales;

handles.vector_energia_media=vector_energia_media;
handles.vector_angulo=vector_angulo;


guidata(hObject, handles);
```

ELECCIÓN DEL NÚMERO DE SEÑALES A RECOGER EN CADA POSICIÓN

```
function edit_n_senales_def_Callback(hObject, eventdata, handles)
% hObject    handle to edit_n_senales_def (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_n_senales_def as text
%        str2double(get(hObject,'String')) returns contents of
edit_n_senales_def as a double

handles.num_senales=str2double(get(hObject,'String'));

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit_n_senales_def_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_n_senales_def (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% 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
```

ELECCIÓN DEL PASO DEL SERVOMOTOR

```
function edit_paso_servo_Callback(hObject, eventdata, handles)
% hObject    handle to edit_paso_servo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_paso_servo as text
%        str2double(get(hObject,'String')) returns contents of edit_paso_servo
as a double

paso_servo=str2double(get(hObject,'String'));
paso_servo=paso_servo/180;

handles.paso_servo=paso_servo;

guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function edit_paso_servo_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_paso_servo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% 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
```

INTERRUPCIÓN DE LA RECOGIDA

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

a=handles.arduino;
s=handles.servo;

set(handles.Toggle Button_recoger_senales,'Value',0);

guidata(hObject, handles);
```

DESCONEXIÓN DEL SISTEMA

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

ictObj=handles.tarjeta;
a=handles.arduino;
s=handles.servo;

set(handles.text_info,'String','AVISO: Se ha pulsado el botón de
desconexión');
angle=0;
handles.movimiento=0;
writePosition(s, angle);

handles.servo=s;
clear s a
disconnect(ictObj)
delete(ictObj)
guidata(hObject, handles);
```