Universidad de Cantabria

Facultad de Ciencias

Trabajo de Fin de Máster
Máster en Computación

Desarrollo del software de un sistema para calibración de medidas horizontales

---

Software development for a horizontal measurement calibration system

Autor:
Pablo Gutiérrez Peón
Departamento de Electrónica y Computadores

Director:
Dr. Michael González Harbour
Departamento de Matemáticas, Estadística y Computación

Santander, Octubre 2013

Cuando te dedicas a un trabajo durante varios meses de tu vida, ves que al final hay gente que sin tener ni idea de lo que aquí hay escrito, te ha hecho la vida más fácil e indirectamente te ayudan a conseguir muchas cosas.

Por eso, va mi recuerdo y agradecimiento para mis padres y mi hermana, que hacen que volver a casa apetezca todos los días. La otra mitad de mi casa está en Santander, con mi abuela, Conchi, José Luis y Elena.

Luego están mis amigos. Desde los que hacen pasártelo bien todos los días: Jorge, Arkaitz y Dani, mi hermano de padre y madre diferentes hasta aquellos a los que se ve menos, pero también están ahí: Diego, Sandra, Toni, Guti, David...

Y si mezclas familia y amigos está el Grupo de Computadores y Tiempo Real de la Universidad de Cantabria. Son unos cracks, porque trabajan mucho y te hacen sentir estupendamente en el trabajo. De todos ellos me llevo muy buen recuerdo y de algunos una buena amistad que espero perdure. Agradezco especialmente a mi director Michael González Harbour su inestimable orientación y ayuda en todo momento y el aprecio por mi trabajo.

Por último, pero esenciales en este trabajo, están los amigos de Equipos Nucleares S.A. (ENSA). Lo primero por darme la oportunidad de trabajar en esto junto a ellos, algo que para mi ha sido realmente interesante y enriquecedor. Mención especial a mi supervisor Fernando G. Estefanía del Laboratorio de Robótica y Automática, que desde sus primeras visitas me fue poco a poco despertando el interés por el proyecto. Cada poco tiempo me traía un nuevo trasto con el que jugar y aprender. Su compañero Luis Rincón me ayudó en los momentos más difíciles de la puesta a punto de la máquina en ENSA, lo cual le agradezco enormemente. Y no olvidaré que aunque mi trabajo dependía del Laboratorio de Automática, físicamente lo desarrollé entre los amigos del Laboratorio de Metrología. Me han hecho sentir el empleado del mes cada visita que les hacía. El trato que me han dado ha sido fuera de lo normal. Espero que todos ellos sepan que les estoy también muy agradecido.

Índice general

Resumen

Los sistemas de robótica aplicados a la industria pueden ser una valiosa inversión al automatizar tareas que por tener que hacerse a mano pueden resultar arduas y costosas.

El Laboratorio de Metrología de la empresa Equipos Nucleares, dedicado a labores de medición para calibración de magnitudes, tiene una carga muy elevada de trabajo que conduce a plantearse si operarios cualificados deben dedicar su escaso tiempo a tareas pesadas y repetitivas. Una de esas tareas es la calibración de instrumentos o piezas longitudinales como reglas o barras. Con estos objetos es necesaria la toma de muchas medidas mediante un instrumento de referencia para completar una calibración.

En este contexto surge la *medidora de coordenadas horizontales de 4 metros*, un instrumento para medición de longitudes de barras, cintas métricas, etc. La medidora consiste en un banco de extrema planitud sobre el que se ha instalado un sistema móvil con un sensor de posición. Desplazando ese sistema móvil a distintos puntos se toman las medidas que permiten realizar una calibración.

El proyecto aquí descrito busca crear un software que permita la monitorización y control del movimiento lineal del sistema de medida a través de una interfaz gráfica corriendo sobre un PC. Esta interfaz debe permitir a su vez visualizar la imagen del punto de calibración procedente de una cámara. Todo con el objetivo final de hacer más sencillo y automatizar en la medida de lo posible el proceso de calibración.

Para conseguir este objetivo, se planteó una aproximación que manejase tecnologías como servos digitales para control de motores, protocolos de comunicación industriales como CAN, programación sobre librerías para tarjetas de PC, etc. De esta forma también se consigue probar la funcionalidad de estas tecnologías para futuros desarrollos. El manejo de todos estos componentes hardware ha resultado en que aparte de la aplicación principal, existan varios módulos software que son independientes y por lo tanto pueden ser aprovechados para otros desarrollos.

Summary

Robotic systems can be a valuable invest in industrial applications by automating tasks that can be arduous and hard when done manually.

The Metrology Laboratory in the company Equipos Nucleares is dedicated to measurement and magnitude calibration. Their heavy workload leads to thinking of ways to automate heavy and repetitive tasks. One of these tasks is the calibration of instruments for the measurement of linear pieces such as rules or bars. The calibration of these objects needs many measurements to be taken using a reference instrument.

Within this context, a *4 metres horizontal coordinates meter* is being developed, an instrument for measuring lengths of bars, measurement tapes, etc. The meter consists of an extremely flat workbench on which a mobile system with a position sensor has been installed. The calibra-

tion process consists of moving this mobile system to different points and taking measurements at them.

The project here described aims to create a software application for monitoring and controlling the movement of the measuring system using a graphical interface running on a PC. This interface has to allow viewing the image of each calibration point coming from a digital camera. The final goal is to do the calibration process easier and more automatic.

To reach this goal, we use technologies such as digital servos for motor control, industrial communication protocols like CAN, programming over PC card libraries, etc. The use of these technologies is a test for their functionality for future developments at Equipos Nucleares. The management of all these hardware components has lead to the creation of new software modules that are independent from the main application and can be used for other developments.

Capítulo 1

Introducción

1.1. Metrología

El desarrollo material de la humanidad en los dos últimos siglos ha sido posible gracias a la cantidad de productos que la industria ha proporcionado con una calidad y precios aceptables [SHMS02]. Para ello se abandonaron los sistemas de fabricación propios de la artesanía donde una sola persona o muy pocas realizaban un producto completo en todos sus aspectos, dando paso a los sistemas de fabricación en serie donde cada operario realiza una gran cantidad de unidades de una sola pieza, e incluso normalmente tan sólo alguna de las operaciones necesarias para obtener dicha pieza. Esta forma de producción impuesta por el factor económico ha creado a cambio la necesidad de intercambiabilidad que permita el montaje de un mecanismo complejo a partir del conjunto de sus piezas componentes y que posteriormente puedan sustituirse una o varias de ellas sin fallos en el conjunto. Así surgió la *metrotecnia* como conjunto de técnicas para la realización de medidas.

Además de conseguir la intercambiabilidad, el desarrollo técnico y la calidad de los mecanismos han conducido a un aumento en las exigencias de precisión que traen consigo la aplicación de técnicas muy específicas haciendo que la medición haya tomado carácter de ciencia: *metrología*.

La metrología es por tanto la ciencia de las medidas [SHMS02]. Su ámbito trata del estudio y aplicación de todos los medios propios para la medida de magnitudes como longitudes, masas, tiempos, temperaturas, etc. Esta enumeración hace entrever que la metrología está presente en muchos de los dominios de la ciencia y la ingeniería. Dentro de la metrología existen divisiones que tratan con las distintas magnitudes. Así se tiene la metrología dimensional, que se ocupa de las medidas de las dimensiones, la metrología ponderal, que se ocupa de la medida de pesos, etc.

Un aspecto clave para el correcto desarrollo de la metrología es la *normalización*. La elaboración y aceptación de una norma por parte de los interesados permite establecer una referencia que facilite el intercambio del que se hablaba anteriormente. Existen una serie de entidades que acreditan el cumplimiento de estas normas en los distintos ámbitos en que se aplican.

1.1.1. Metrología dimensional

La metrología dimensional, encargada de la medida de las dimensiones, tiene en el plano de fabricación de la pieza su elemento fundamental ya que éste contiene todas las cotas necesarias para determinar sus *dimensiones* (longitudes, ángulos), *forma* (rectitud, paralelismo, planitud, redondez...) y *acabado superficial* [SHMS02]. La medición de estas tres características permite determinar la geometría de una pieza.

1.1.2. Laboratorio de metrología

El campo de la medición abarca desde los denominados *laboratorios nacionales*, con la misión específica de la conservación, reproducción y diseminación de las unidades de medida, hasta el propio *taller de fabricación*, en donde una serie de instrumentos ayudan a conseguir la forma deseada de las piezas marcadas en los planos.

Entre ambos extremos existen diferentes tipos de centros de medición, con diferentes misiones y características. El más representativo es el *laboratorio de metrología*, capaz de proporcionar medidas de precisión y dar calibración a los equipos en los departamentos de control de calidad de los diferentes talleres de fabricación.

Los laboratorios de metrología cuentan con acreditaciones que dan validez al trabajo de medición que realizan. Para que estas mediciones sean aceptadas y sea posible conseguir estas acreditaciones, se debe reducir la posibilidad de error al mínimo estableciendo un exhaustivo control sobre aspectos como condiciones ambientales, organización del trabajo, adopción de procedimientos y trazabilidad.

El Laboratorio de Metrología de la empresa Equipos Nucleares S.A. (ENSA) surge como respuesta a las necesidades de metrología dentro de la propia empresa, dedicada a la fabricación de componentes nucleares. No obstante, sus acreditaciones le han hecho constituirse en uno de los laboratorios de referencia para empresas de todo el norte de España.

Está equipado con más de 60 instrumentos de calibración en las áreas dimensional, presión, electricidad, masa, temperatura, humedad, fuerza, momentos, revoluciones y ensayos destructivos y mecánicos.

1.2. Calibración sobre medidora de coordenadas horizontales

El instrumento de medida que centra este trabajo pertenece al área dimensional y se ocupa de la medición de longitudes. En el Laboratorio de Metrología de ENSA se dispone de decenas de instrumentos de medición de longitudes. La peculiaridad de este instrumento radica en sus grandes dimensiones, que permiten la medida de longitudes de hasta 4 m en una sola medición. Además, la realización de medidas por tramos permite la calibración de componentes con longitudes aún mayores en caso de que éstos no sean rígidos, como sucede con las cintas de metro.

La medidora de coordenadas horizontales consiste en un bloque de granito de 4 m de longitud sobre el que se ha construido un sistema móvil consistente en un patín que se desplaza sobre cojines de aire. Para conocer de forma precisa la posición de medida que ocupa el patín, se ha instalado una regla de alta resolución. Mediante una cabeza lectora se hace la lectura de esta regla. Un motor adosado al patín permite su desplazamiento automático. El patín incorpora una cámara que señala el punto que se está midiendo. Así, el operario puede desplazar el patín a la posición que desee.

La importancia que por sus características dan en el Laboratorio de Metrología al bloque de granito, que ellos denominan mármol, hace que sea precisamente Mármol el nombre clave con el que se maneja este proyecto.

Aunque la medidora de coordenadas horizontales puede ser utilizada en la medición de piezas, su principal uso está en la calibración de otros instrumentos de medida como metros (rígidos y cintas) o barras patrón.

En el proceso de calibración, la medidora de coordenadas horizontales actúa como *referencia*. Al calibrar un objeto, se establece una comparación entre el instrumento de medida y dicho

objeto. Estas comparaciones se anotan para estudiar las divergencias y comprobar si están o no dentro de lo admisible.

El hecho de que la medidora de coordenadas horizontales actúe como referencia es debido a que a sus características físicas, cuidado en su manejo y localización dentro de un laboratorio garantizan la precisión de sus medidas. No obstante, tanto este instrumento de calibración como cualquier otro del laboratorio no están libres de ser a su vez calibrados por otros instrumentos que reciben el nombre de *patrones*, cuya jerarquía garantiza la validez de las medidas a nivel mundial.

1.3. Objetivos del proyecto

ENSA dispone de un *Laboratorio de Robótica y Automática* encargado del desarrollo de aplicaciones especiales de robotización y automatización de procesos de la propia factoría como soldadura, control, inspección y montaje. El Laboratorio de Metrología planteó la necesidad de construir la medidora de coordenadas horizontales a sus colegas de Robótica. Desde 1987 el Grupo de Computadores y Tiempo Real (CTR) de la Universidad de Cantabria lleva colaborando con el Laboratorio de Robótica y Automática de ENSA en la realización del software para diversos robots.

En este marco de relaciones, se trasladó la responsabilidad del desarrollo del software del sistema a la Universidad, dando lugar al proyecto que se describe en esta memoria.

El desarrollo de la máquina física por parte de ENSA y del software por parte de la Universidad ha experimentado una evolución paralela que ha traído consigo cambios en ambas direcciones, por lo que la colaboración ha sido estrecha. Fruto de esa relación y con las opiniones dadas por los usuarios finales, los objetivos concretos del proyecto se han ido puliendo hasta el punto actual.

El objetivo general de este proyecto ha sido el desarrollo del software del sistema de calibración sobre medidora de coordenadas horizontales. Para conseguirlo, se propuso con el fin de investigar y evaluar ciertas tecnologías que el software cumpliera con las siguientes características:

- Monitorización y control del movimiento del sistema de calibración mediante servo digital con interfaz de comunicación CAN sobre la que funciona el protocolo CANopen.
- Interfaz gráfica basada en GtkAda que permita monitorizar y controlar el servo, además de visualizar la imagen del punto de calibración procedente de la cámara.

El cumplimiento de estos objetivos desencadenó en la necesidad de desarrollar los siguientes módulos software, los cuales constituyen resultados independientes que pueden ser utilizados en otros ámbitos:

- Creación de un controlador de dispositivo (driver) en Windows NT para tarjeta de entradas y salidas digitales PCI-1761 de Advantech.
- Creación de librerías software para manejo de los siguientes componentes:
  - Cámara digital marca DALSA basada en el estándar de transmisión de vídeo y control de dispositivos GigE Vision.
  - Tarjeta de comunicaciones CAN PCI-1680U de Advantech.

- Creación de una librería para lenguaje Ada que permita la utilización de las partes que se necesitan en el proyecto del protocolo CANopen, con un diseño modular para que sea extensible.

Capítulo 2

Descripción del sistema

2.1. Descripción del sistema de calibración

En metrología las condiciones en las cuales se realiza una medida son extremadamente importantes para garantizar la exactitud y por tanto la validez del proceso. En metrología dimensional uno de los factores más condicionantes para una medida es la temperatura, debido a que el efecto dilatador que produce puede variar de forma dramática las medidas realizadas. La temperatura es importante tanto por el efecto que produce en el instrumento medido como en el propio instrumento de medida. Por ello es importante que el instrumento de medida no sufra demasiado de este efecto y lo minimice en lo posible.

Otro de los aspectos que más se cuida en metrología dimensional es que la geometría del instrumento de medida no influya en la propia medida. Si se desea, por ejemplo, realizar mediciones de longitudes lineales, se debe evitar que el instrumento de medida tenga algún tipo de desviación por rugosidad o curvatura.

Es por ello que el elemento que más destaca del sistema que ocupa este proyecto es el denominado *mármol de calibración*. Se trata de un bloque de granito de dimensiones 4000x200x200 mm caracterizado por su bajo coeficiente de dilatación y extrema planitud (5 μ en los 4 m). De esta forma, la incertidumbre de las medidas, aunque está cuantificada y se tiene en cuenta, es mínima.

El mecanismo de medida se compone de un *patín de bajo rozamiento* que se desplaza sobre el mármol con una cámara de aire de por medio (Figura 2.1 y 2.2). Para conocer de forma precisa la posición de medida que ocupa el patín, se ha instalado en un lateral del mármol una *regla de alta resolución*. El patín dispone de una cabeza lectora que le permite leer la regla. Aunque el patín puede desplazarse de forma manual, un *motor* se encarga del desplazamiento automático. Existe un *mecanismo de embrague* que permite pasar de una forma de desplazamiento a otra. Además, el eje de movimiento tiene instalados dos sensores que indican al sistema de control los extremos (*finales de carrera*) y un contacto de *home* que permite referenciar la posición a un punto conocido.

El patín incorpora una *cámara* que señala el punto que se está midiendo. De esta forma, el operario puede desplazar el patín al punto de calibración que desee.

El sistema de control se basa en la utilización de una aplicación software para PC. La aplicación se maneja mediante una interfaz gráfica que cuenta con todos los controles necesarios para gestionar el sistema y obtener información de su estado a través de elementos como un monitor de posición o el vídeo procedente de la cámara. La aplicación permite realizar calibraciones, esto es, hacer correcciones entre la posición consignada y las desviaciones del elemento a medir,

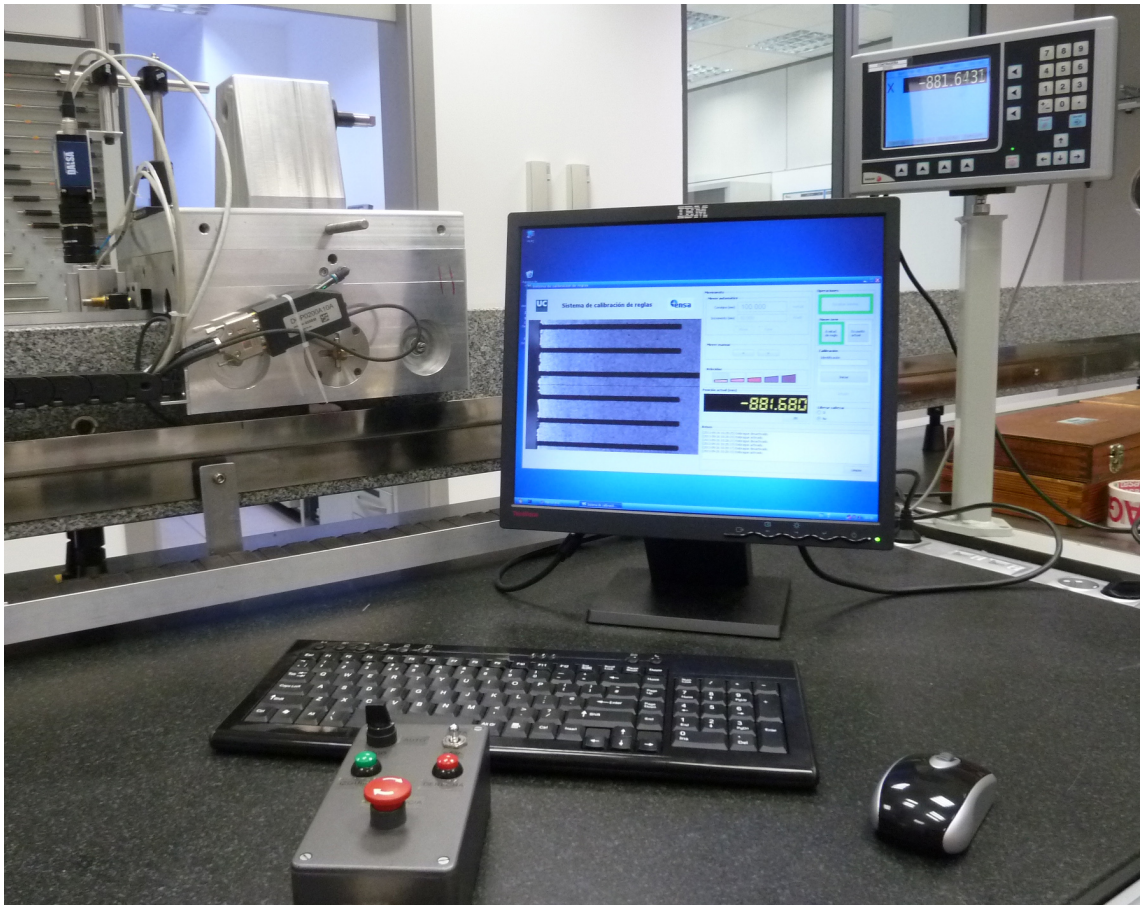

Figura 2.1: Fotografía del sistema de calibración completo.

guardando el resultado en un fichero compatible con hojas de cálculo.

Además, se dispone de una botonera para el manejo de algunos modos de movimiento. Esto otorga al operario de cierta movilidad al no precisar encontrarse en el puesto del PC para todas las operaciones.

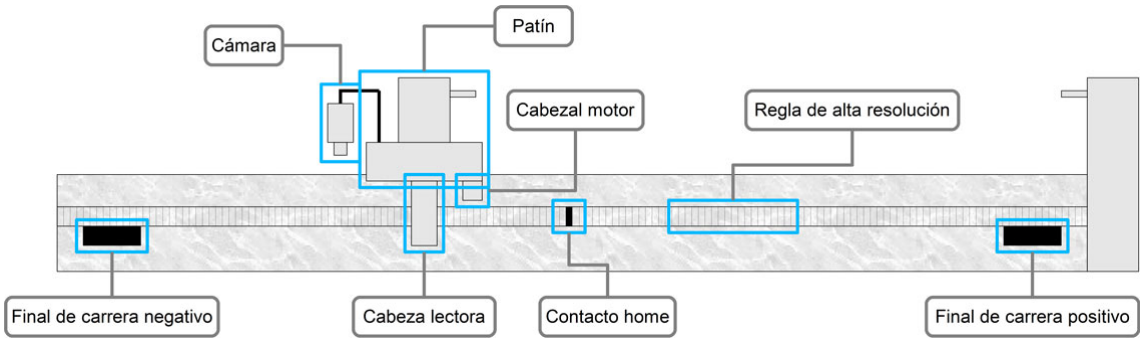

Figura 2.2: Esquema del sistema de calibración

La *aplicación de calibración* desarrollada sobre este sistema físico permite y gestiona la calibración de diversos tipos de objetos. Para ello, controla el movimiento del patín a lo largo del mármol mediante el accionamiento del motor. La imagen procedente de la cámara hace posible al usuario enfocar el punto concreto que se desea calibrar. La regla de alta resolución permite conocer en la aplicación la posición del patín en cada momento. De esta forma, se puede comprobar si la posición apuntada por la cámara es la misma que indica la regla de alta

resolución y en función de las posibles divergencias realizar la calibración del objeto en cuestión.

2.2. Descripción hardware

En la Figura 2.3 se muestra el esquema detallado que recoge los componentes hardware del sistema divididos por grupos y la conexión que existe entre ellos.

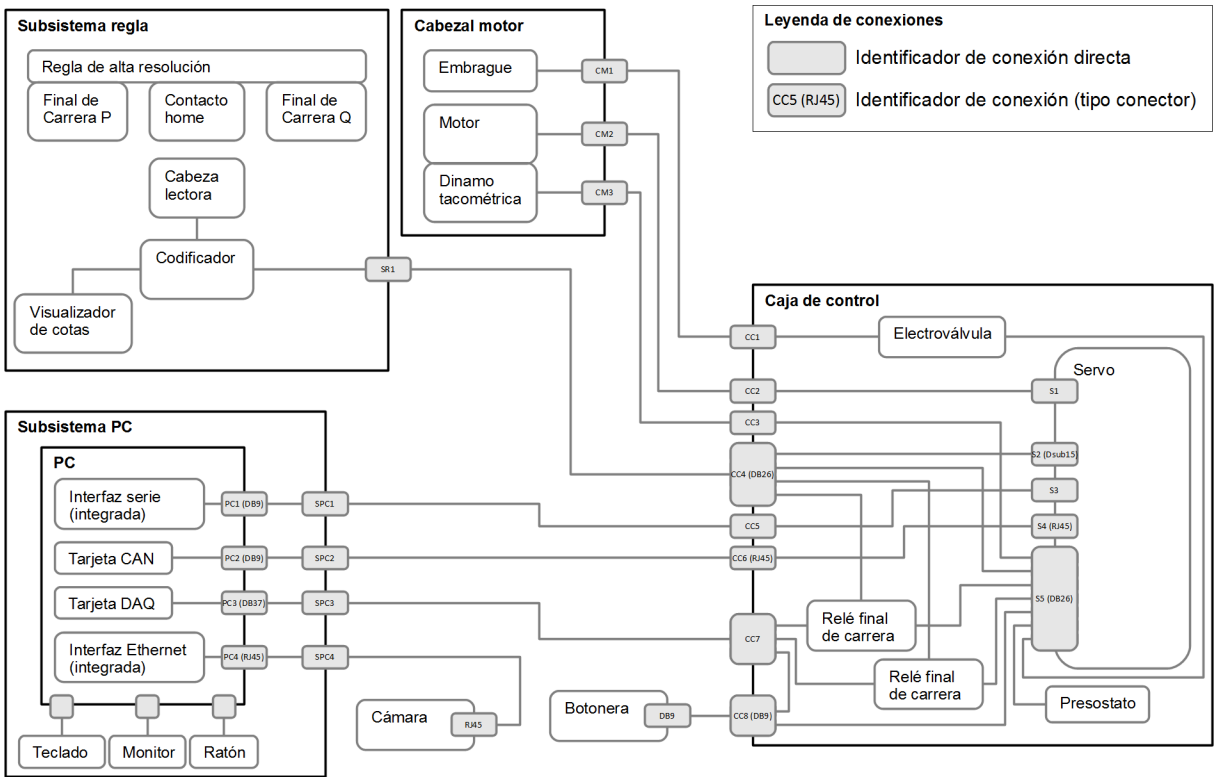

Figura 2.3: Diagrama de elementos hardware del sistema agrupados por subsistemas y sus conexiones.

En las secciones siguientes se describe cada uno de los subsistemas y sus componentes.

2.2.1. Subsistema de la regla

Subsistema encargado de medir la posición que ocupa el patín y detectar los extremos y punto medio del mármol. El sistema completo forma parte de la solución que la marca Renishaw ofrece para codificadores de posición en el campo de la metrología [Ren13a] [Ren13b]. La Figura 2.4 contiene un esquema de sus componentes.

Regla de alta resolución

Regla de acero inoxidable con resolución de 0.1 μm (10.000 marcas por mm). La composición del material garantiza un coeficiente de dilatación similar en cualquier punto. Su precisión garantiza errores de ± 4 μm en 5 m.

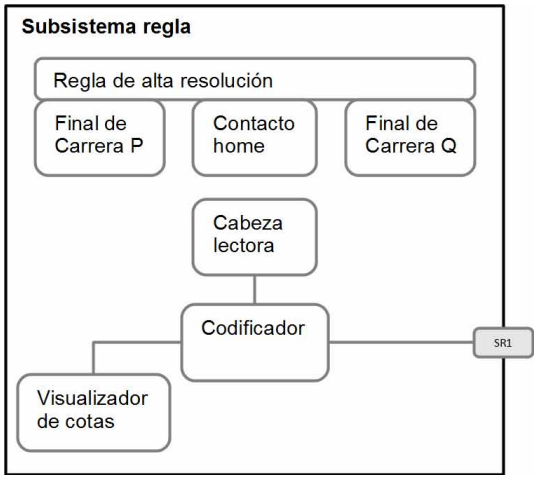

Figura 2.4: Diagrama de elementos hardware del subsistema de la regla y su conexión.

Finales de carrera

La regla permite la instalación de límites magnéticos que indiquen el final de carrera al patín. Se instalan magnéticamente sobre la regla metálica, pudiendo ser cambiados de posición en cualquier momento. Se dispone de dos tipos de límites (nombrados como P y Q) que permiten distinguir entre los dos finales de carrera.

Contacto de home

La regla cuenta con una marca impresa en su punto medio que sirve como referencia de posición. Se detecta ópticamente.

Cabeza lectora

El patín lleva adosado un cabezal situado a mínima distancia de la regla. Equipado con un sensor óptico, detecta las marcas de la regla, así como el contacto de home. Dispone además de un sensor magnético que detecta los finales de carrera. En la Figura 2.5 se muestra un esquema de los elementos que componen una regla Renishaw.

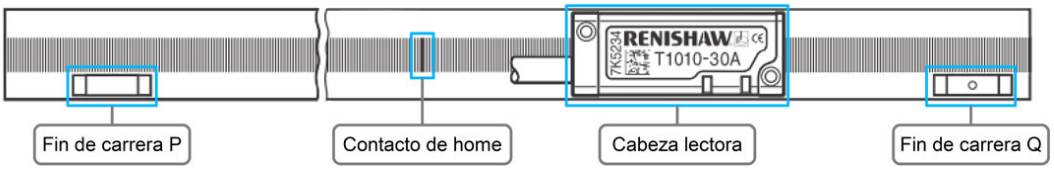

Figura 2.5: Esquema de los elementos de una regla Renishaw.

Codificador

El codificador está conectado al cabezal y realiza el procesado de los pulsos de la regla, dando a la salida una onda cuadrada cada 0.1 μm. Debido a la frecuencia de muestreo, el fabricante nos asegura que la velocidad de desplazamiento máxima del patín será de 0.81 m/s, suficiente para las necesidades de este sistema.

Visualizador de cotas

Debido a que el codificador dispone de salida dual, se pensó en utilizar una de las salidas para mostrar la posición en un visualizador externo marca Fagor. Este visualizador traduce los pulsos dados por el codificador a un valor de posición. La posición puede reiniciarse a cero para establecer un punto de referencia.

Señales disponibles para el controlador

La interfaz de señales externas del subsistema de la regla aparece representada en la Figura 2.4 con el código **SR1**. Las señales disponibles a través de SR1 son:

- Valor de posición dado por el codificador (señal de salida).
- Señal de contacto de home (señal de salida).
- Señal de final de carrera P (señal de salida).
- Señal de final de carrera Q (señal de salida).

2.2.2. Cabezal del motor

El cabezal del motor es un bastidor metálico que alberga en su interior el motor y la dinamo tacométrica. La Figura 2.6 contiene un esquema de sus componentes.

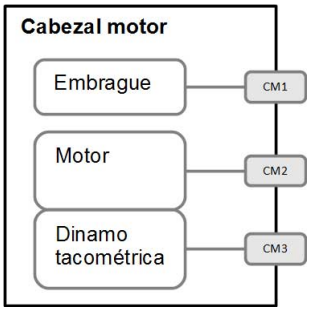

Figura 2.6: Diagrama de elementos hardware del cabezal del motor y sus conexiones.

Motor

Es el encargado del desplazamiento del patín a lo largo del mármol. Dispone en su eje de una rueda con superficie de caucho que es la que toma contacto con el mármol y ejerce la tracción.

La principal característica del motor es lo que se denomina *cero holgura*. La holgura es la pérdida de tracción del motor causada por espacios entre sus ruedas dentadas. En este caso esos espacios se reducen al mínimo, haciendo que cuando el motor está en reposo, mantenga su posición firmemente evitando pérdida de precisión del sistema de calibración.

Dinamo tacométrica

La dinamo tacométrica es un sensor de velocidad para máquinas rotativas. Acoplado al eje del motor, permite conocer la velocidad de rotación de este, lo cual redunda en un mejor control del movimiento.

Embrague

El cabezal que alberga el motor dispone de un embrague activado por un circuito neumático que desacopla la rueda del eje del motor que hace tracción contra el mármol. De esta forma se permite el movimiento manual del patín.

Señales disponibles para el controlador

La interfaz de señales externas del cabezal del motor aparece representada en la Figura 2.6 con los códigos CM1, CM2 y CM3. Las señales disponibles a través de esta interfaz son:

- **CM1**. Entrada neumática para control del embrague (señal de entrada).
- **CM2**. Conexión de alimentación del motor que posibilita el movimiento del mismo (señal de entrada).
- **CM3**. Conexión procedente del sensor de velocidad de la dinamo tacométrica (señal de salida).

2.2.3. Caja de control

La caja de control centraliza el hardware de control del sistema. Además se encarga de dar alimentación al resto de componentes. Su elemento de mayor importancia es el servo. La Figura 2.7 contiene un esquema de sus componentes.

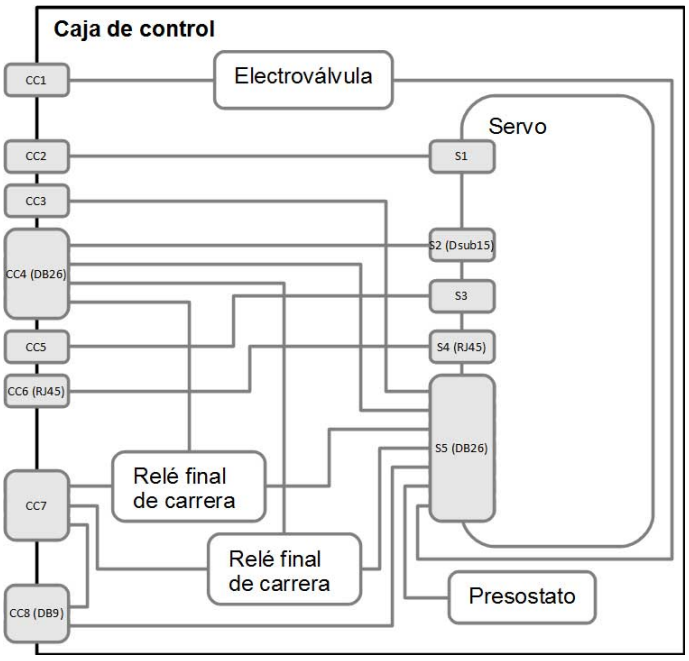

Figura 2.7: Diagrama de elementos hardware de la caja de control y sus conexiones.

Servo

El servo es un amplificador electrónico empleado para dar corriente a los llamados *servomecanismos*. El servo monitoriza la señal de retroalimentación del servomecanismo para ajustar la desviación respecto al comportamiento previsto siguiendo lo que se denominan *lazos de control*.

A grandes rasgos, un sistema regido por lazo de control basa la acción de control en la orden de entrada y en la señal de salida procedente de un sensor (ver Figura 2.8). De esta forma a la hora de emitir una nueva orden se tiene en cuenta la desviación de la señal de salida del motor respecto al comportamiento previsto (error). El cálculo de este error suele basarse en un algoritmo PID. PID son las siglas de Potencial, Integral y Derivativo, los parámetros involucrados en el cálculo.

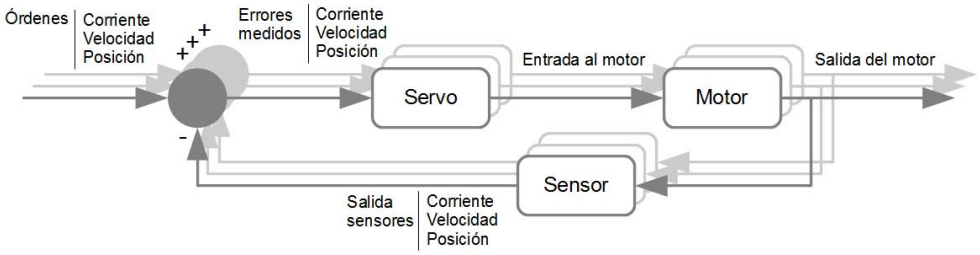

Figura 2.8: Visión esquemática de los lazos de control empleados: corriente, velocidad y posición.

Este tipo de lazos pueden ser empleados sobre distintas magnitudes físicas. Una mayor cantidad de lazos de control en un sistema de control supone un control más fino. En este proyecto han sido empleados tres lazos de control para el movimiento del motor basados en la corriente de alimentación, la velocidad de rotación del motor dada por la dinamo tacométrica y el sensor de posición del cabezal.

Por otro lado, el servo dispone de entradas/salidas programables que permiten asociar eventos internos ante la activación de determinadas señales. Por ejemplo, esto permite que si se activa un final de carrera, se active un evento interno del servo que impida el movimiento en ese sentido.

Las interfaz de conexiones del servo se muestra en la Figura 2.7. Las conexiones se muestran referidas al identificador dado en dicho diagrama:

- **S1**. Conexión del servo al motor para alimentar el movimiento.
- **S2** (Dsub15). Retroalimentación de posición procedente del codificador de la regla.
- **S3**. Conexión mediante interfaz serie RS-232 al PC para configuración.
- **S4** (RJ45). Conexión mediante bus CAN al PC para monitorización y control.
- **S5** (DB26). Puerto de entradas y salidas analógicas y digitales. Conexiones realizadas:
  - Entrada analógica de retroalimentación de velocidad procedente de la dinamo tacométrica acoplada al motor.
  - Entrada digital de señal de contacto de home procedente de la cabeza lectora de la regla.
  - Entrada digital de señal de error procedente del codificador de la regla.
  - Entradas digitales de señales de finales de carrera procedentes de los relés.
  - Entradas digitales de seta de emergencia y conmutador de embrague procedentes de la botonera.
  - Entrada digital de señal de error del presostato.
  - Salida digital de activación de la electroválvula.

  Las señales de entrada pueden ser leídas y las de salida leídas o escritas desde el PC a través del bus CAN.

En la Sección 3.4 se amplía la información referente al servo.

Relés para finales de carrera

Los relés activan la señal de salida cuando se alcanzan los finales de carrera y permanecen activos hasta que se desenclavan mediante las señales digitales procedentes del PC. De esta forma se evita que el sistema no detecte el final de carrera por sobrepasarlo con exceso de velocidad.

Presostato

Los presostatos abren o cierran un circuito eléctrico dependiendo de la lectura de presión de un fluido. En este caso, el presostato se utiliza para detectar si la presión del sistema neumático que crea la cama de aire entre el patín y el mármol no es correcta. El circuito eléctrico está conectado al servo, de forma que cualquier anomalía es detectada.

Electroválvula

La electroválvula abre o cierra el circuito de aire que actúa el embrague del cabezal. El accionamiento eléctrico se realiza desde una salida digital del servo.

2.2.4. Subsistema del PC

En el PC corre el software desde el cual el usuario puede monitorizar y controlar el sistema. Para ello el usuario utiliza los tradicionales periféricos de entrada/salida: teclado, ratón y monitor. Se ha escogido un controlador PC por su bajo coste y abundancia de tarjetas de entrada/salida.

Es necesario que el PC esté conectado con el servo, la cámara y la botonera. Se han empleado para este objetivo las interfaces serie RS-232 y Ethernet ya integradas y ha sido necesario instalar dos tarjetas PCI (para CAN y Data AcQuisition - DAQ) que amplían las interfaces de comunicación del PC. La Figura 2.9 contiene un esquema de los componentes del subsistema PC.

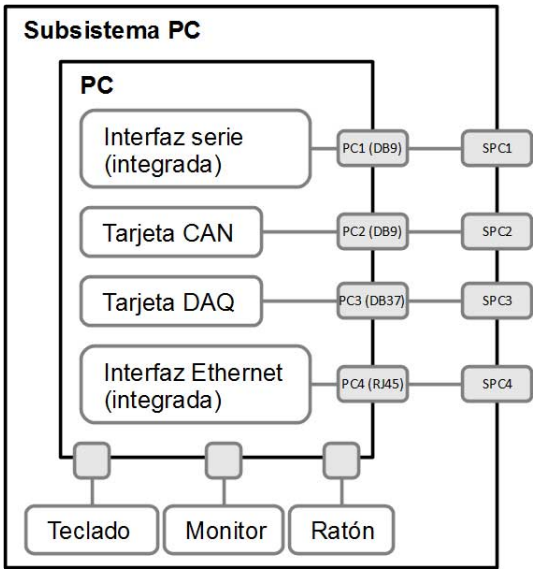

Figura 2.9: Diagrama de elementos hardware del subsistema del PC y sus conexiones.

Interfaz serie

El servo dispone de un software de configuración de parámetros estáticos que emplea la interfaz RS-232 para las comunicaciones. Se amplía la información sobre este software en la Sección 3.4.3.

Tarjeta CAN

Se precisa de una interfaz basada en el bus CAN para la comunicación de monitorización y control con el servo. Por ello se instaló una tarjeta PCI modelo 1680U de Advantech, que permite contar en el PC con dos puertos CAN.

Tarjeta DAQ

La tarjeta DAQ PCI-1761 de Advantech da al PC la capacidad de contar con un total de 8 entradas y 8 salidas digitales. Las señales de entrada se utilizan para conectar algunos controles de la botonera. Por su lado, las señales de salida se emplean para desenclavar los relés de los finales de carrera cuando el software de control considera que ya se han abandonado.

Interfaz Ethernet

La interfaz de conexión Ethernet se emplea para recibir la transmisión de imagen procedente de la cámara de vídeo digital.

Monitor, teclado y ratón

Periféricos para el manejo de la aplicación software por parte del usuario.

2.2.5. Cámara

Figura 2.10: Diagrama de la cámara y sus conexiones.

La cámara de vídeo digital se utiliza para mostrar en la interfaz gráfica del software de calibración la imagen del punto de calibración. De esta forma, si el usuario está calibrando una regla puede ver aumentados los trazos de la misma y desplazar el patín hasta que el trazo deseado coincida con la marca sobreimpresa en la imagen.

La cámara utiliza la interfaz Ethernet en su conexión con el PC. La Figura 2.10 muestra las conexiones de la cámara.

2.2.6. Botonera

La botonera dispone de controles para el desplazamiento del patín en ambos sentidos que se conectan a la tarjeta DAQ del PC. Cuenta también con una seta de emergencia y un actuador del embrague del cabezal conectados al servo. Estos dos últimos están conectados de tal forma que deshabilitan el circuito de alimentación del servo al motor, por lo que el movimiento es imposible en caso de que se activen.

Figura 2.11: Diagrama de la botonera y sus conexiones.

La Figura 2.11 muestra las conexiones de la botonera.

2.3. Especificación software

Esta sección constituye una síntesis del contenido recogido en el documento de especificación software del sistema [EGP13b]. Se plantea la descripción del software a partir de las operaciones que con él se pueden hacer y los modos de operación en que puede funcionar. A continuación se describen los elementos de interacción con el usuario como son la interfaz gráfica y la botonera manual, para finalizar con una pequeña reseña sobre la configuración del software.

2.3.1. Operaciones del sistema

Inicialización

El sistema debe pasar por la inicialización para estar operativo. La inicialización establece conexión con el servo y la tarjeta de entradas/salidas digitales. Por ello, si la inicialización no se produce, ninguna otra operación está disponible.

Hacer cero

Es necesario un mecanismo que permita poner a 0 el valor del visor que informa de la posición actual del patín para que esta sirva como referencia. Se ofrecen dos modos de hacer cero:

- **Cero relativo**. Cero que puede hacerse en cualquier posición que ocupe el patín. Por ejemplo, sobre la marca de cero de una regla que se está calibrando.
- **Cero home**. Cero basado en la posición del contacto de home de la regla. Hacer este cero conlleva la entrada en una rutina de búsqueda de dicho contacto. La configuración de la rutina de home puede ser prefijada mediante el puerto serie del servo para que cuando sea necesario realizar el proceso, únicamente deba activarse la rutina preprogramada.

  Según el procedimiento de home seguido, el contacto de home se busca desplazando el patín siempre en el mismo sentido, por lo que es necesario que el usuario desplace el patín a un lado predeterminado del contacto de home cada vez antes de empezar.

Cambio de velocidad

Se debe contar con una operación que permita el cambio de velocidad de desplazamiento del patín. Se ofrecen 5 velocidades expresadas en 5 controles de la interfaz. En la botonera, a su vez, existe un conmutador que permite seleccionar entre velocidad alta y velocidad baja. Si el conmutador de la botonera está a velocidad alta, los dos botones de la interfaz que indican mayor velocidad permanecen habilitados. En caso de que la velocidad seleccionada en la botonera sea baja, los tres botones de menor velocidad son los que están habilitados. En todo caso, sólo permanece seleccionada una velocidad. La velocidad puede ser modificada tanto si el servo está en reposo como si está en movimiento.

Embrague del cabezal del motor

La interfaz debe contar con un conmutador que permita seleccionar entre acoplar o desacoplar el cabezal del motor del mármol.

Movimiento

Desde la aplicación se pueden realizar dos tipos de movimiento con el patín:

- **Movimiento automático a consigna de posición**. En este tipo de movimiento, el usuario indica una posición absoluta y el servo desplaza el patín hasta ella. Se dispone de un mecanismo que permite añadir de forma sencilla aumentos a la consigna permitiendo así la realización de movimientos por pasos repetitivos. Se cuenta con un botón que permite detener el movimiento. Si se conoce la posición del home, se limita el movimiento a los finales de carrera software. En caso de desconocer la posición de home porque la inicialización fue relativa, los finales de carrera son los contactos hardware de la regla.

- **Movimiento manual en un sentido**. Se activa pulsando los botones de izquierda y derecha de la botonera, el teclado o la interfaz. El movimiento continúa hasta que el usuario suelta el control que inició el movimiento. Si se conoce la posición de home, los finales de carrera son software. En caso contrario, los finales de carrera son los de la regla.

2.3.2. Modos de operación

El sistema puede operar en modo de calibración o fuera de él.

Calibración

La calibración es el proceso por el cual se almacenan en un fichero compatible con hoja de cálculo parejas de valores que el usuario adquiere. Cada pareja de valores está compuesta por la posición a calibrar del objeto de calibración X_{objeto} y la posición en la cual la regla de alta resolución del mármol dice que se encuentra esa posición.

Al realizar la calibración de X_{objeto} se ordena al patín desplazarse hasta X_{marmol} (ambos deben tener la misma referencia de origen). Mediante la cámara es posible ver si X_{objeto} y X_{marmol} coinciden. En caso de que no coincidan, se desplaza el patín en modo manual hasta la posición X_{objeto}. La posición dada por la regla de referencia habrá variado $X_{marmol} \pm error$. Se almacenan los valores X_{objeto} y X_{marmol} de tal forma que se puede conocer el error del objeto calibrado respecto a la regla de referencia del mármol en cada punto de calibración adquirido ($X_{marmol} \pm error = X_{objeto}$).

Para entrar en modo de calibración es necesario indicar un nombre para el fichero de calibración e iniciar el proceso mediante un botón. A partir de entonces y hasta que se decida parar la calibración, se pueden adquirir las parejas de valores siempre que el patín esté en reposo.

En modo de calibración se permiten los dos tipos de movimiento, a consigna de posición o desplazamiento continuo en un sentido.

No calibración

Modo empleado cuando se quiere realizar desplazamientos a lo largo del mármol pero sin adquisición de datos. En este modo no es posible adquirir puntos de calibración ni realizar movimientos a consigna de posición.

2.3.3. Interfaz gráfica de usuario

La aplicación se maneja mediante una interfaz gráfica que cuenta con todos los elementos necesarios para operar con el sistema de la forma descrita en las secciones anteriores (ver Figura 2.12). Dispone además de un visor que indica la posición que ocupa el patín en cada momento y una zona donde mostrar la imagen de la cámara.

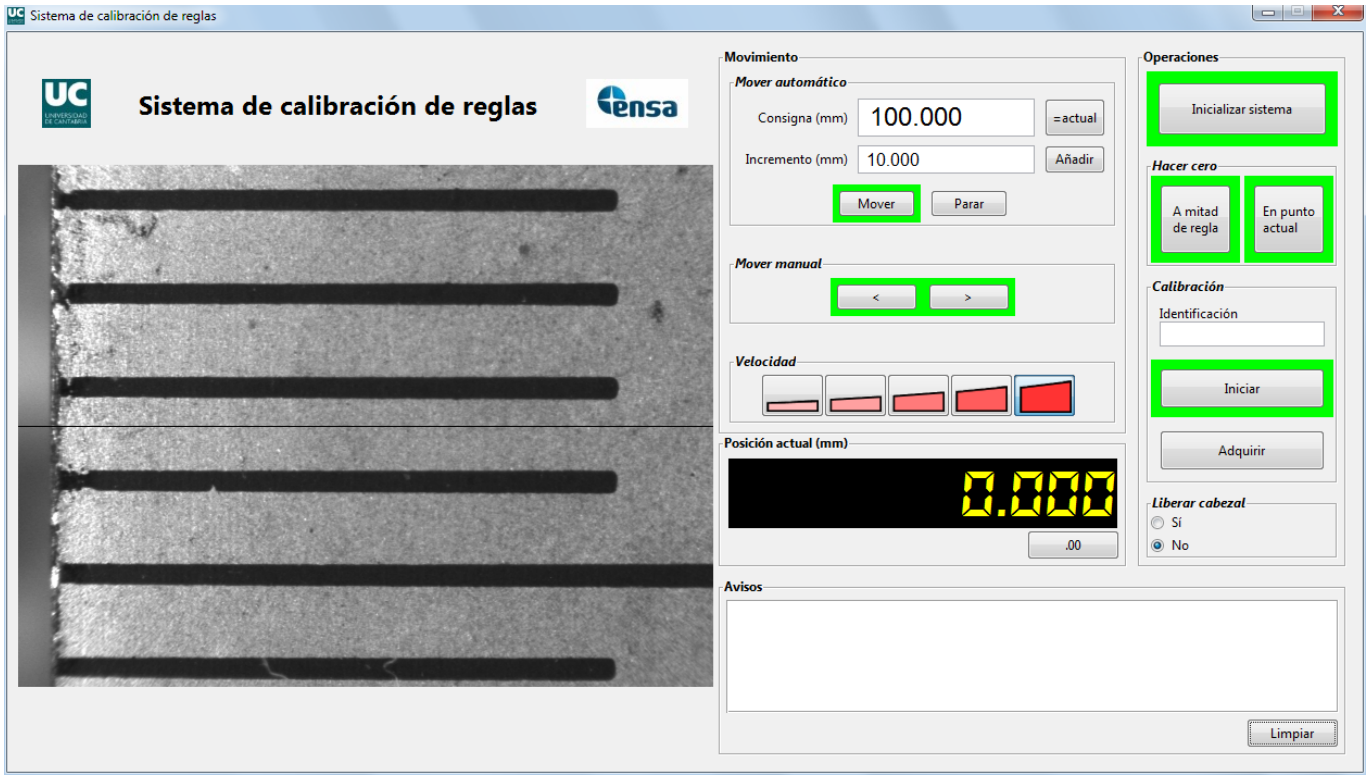

Figura 2.12: Aspecto de la interfaz gráfica de usuario.

Los elementos de la interfaz pueden encontrarse en distintos estados. Todos ellos pueden ser *habilitados* o *deshabilitados*. Los pilotos asociados a algunos botones pueden encontrarse además *parpadeando*, mientras que los propios botones pueden encontrarse *pulsados*. La descripción de los eventos que conducen a unos estados u otros de los controles conlleva un nivel de detalle que no es objeto de esta memoria.

A continuación se enumeran y explican brevemente cada uno de los elementos de la interfaz:

- Operaciones
  - Botón *Inicializar sistema*. Ejecuta la operación de inicializar el sistema.
  - Hacer cero
    - Botón *A mitad de regla / Parar*. Activa o detiene la rutina de home.
    - Botón *En punto actual*. Hace cero en el punto actual que ocupa el patín.
  - Calibración
    - Campo texto *Identificación*. Identifica el elemento a calibrar.
    - Botón *Iniciar / Finalizar*. Inicia o finaliza el proceso de calibración. Activa los controles que permiten el movimiento automático.
    - Botón *Adquirir*. Adquiere un punto de calibración y lo almacena en el fichero de calibración. El punto de calibración se compone de los valores de consigna de posición y la posición actual.

- Conmutador *Liberar cabezal*. Libera el cabezal del motor del mármol para permitir el libre desplazamiento del patín.

- Movimiento
  - Mover automático
    - Campo *Consigna*. Permite introducir la posición a la que se desea mover el patín.
    - Botón *=actual*. Establece el valor del campo *Consigna* al valor del visor de *Posición actual*.
    - Campo *Incremento*. Permite introducir un incremento de posición que añadir al campo *Consigna*.
    - Botón *Añadir*. Suma el valor del campo *Incremento* al del campo *Consigna*.
    - Botón *Mover*. Ordena el movimiento del patín al punto indicado en el campo *Consigna*.
    - Botón *Parar*. Detiene el movimiento ordenado por el botón *Mover*.
  - Mover manual
    - Botón <. Mientras el botón permanece pulsado, el patín se desplaza en movimiento continuo en sentido negativo con tope en los finales de carrera hardware o software. La tecla *Izquierda* del teclado y el botón *Izquierda* de la botonera tienen la misma funcionalidad.
    - Botón >. Mientras el botón permanece pulsado, el patín se desplaza en movimiento continuo en sentido positivo con tope en los finales de carrera hardware o software. La tecla *Derecha* del teclado y el botón *Derecha* de la botonera tienen la misma funcionalidad.
  - Conmutador *Velocidad*. Permite seleccionar un valor de velocidad de entre 5 posibles. La velocidad aquí indicada se utiliza en todos los desplazamientos, excepto en el desplazamiento en busca del contacto de home. El conmutador divide las velocidades en dos grupos excluyentes. *Baja* formado por las 3 velocidades menores y *Alta* formado por las 2 velocidades mayores. La selección del grupo activo en cada momento viene dada por el valor del conmutador *Velocidad* de la botonera.

- Posición actual
  - Visor *Posición actual*. Indica la posición del patín en el mármol.
  - Botón *.00*. Modifica el número de posiciones decimales del valor contenido en el visor de *Posición actual*. Pulsaciones consecutivas permiten cambiar entre 3, 2 o 1 decimales.

- Avisos
  - Zona texto *Avisos*. Zona de la interfaz gráfica donde aparecen avisos relacionados con la operación del controlador. Los avisos pueden informar de eventos producidos, situaciones anómalas, aportar sugerencias para solucionar errores o mostrar errores de programación orientados a la depuración del software.
  - Botón *Limpiar*. Permite eliminar los avisos de la zona de texto de *Avisos*.

- Zona de imagen. Muestra la imagen de vídeo procedente de la cámara.

- Botón *Salir de aplicación*. Botón situado en la esquina superior derecha de la ventana. Permite finalizar la aplicación, deteniendo cualquier operación en curso sobre el servo. Libera además el cabezal mediante la activación de la electroválvula. Después finaliza la conexión con el servo y la cámara. Por último, se cierra la ventana y la aplicación queda finalizada.

2.3.4. Botonera manual

La botonera manual permite controlar ciertos aspectos de la aplicación con la versatilidad que da no tratarse de un mando fijo.

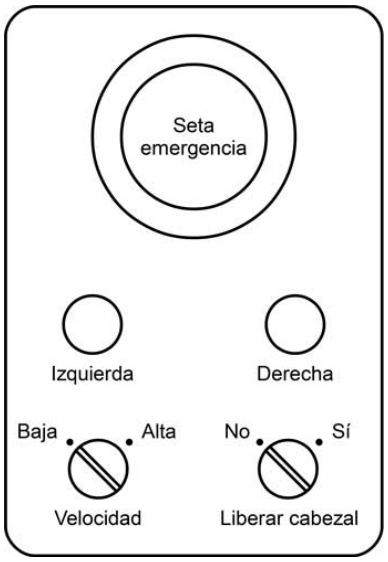

Figura 2.13: Diagrama de la botonera manual.

Los controles de la botonera (ver Figura 2.13) son los que siguen:

- Seta *Emergencia*. La seta de emergencia desactiva la alimentación del servo al motor. Dado que el servo no puede operar hasta que la seta sea liberada, todas las operaciones a realizar sobre él también están deshabilitadas.
- Botón *Izquierda*. Mientras el botón permanece pulsado, el patín se desplaza en movimiento continuo en sentido negativo con tope en los finales de carrera hardware o software. La tecla *Izquierda* del teclado y el botón *Izquierda* de la interfaz tienen la misma funcionalidad.
- Botón *Derecha*. Mientras el botón permanece pulsado, el patín se desplaza en movimiento continuo en sentido positivo con tope en los finales de carrera hardware o software. La tecla *Derecha* del teclado y el botón *Derecha* de la interfaz tienen la misma funcionalidad.
- Conmutador *Velocidad*. Selecciona el grupo de velocidades *Alta* o *Baja* del conmutador *Velocidad* de la interfaz gráfica.
- Conmutador *Liberar cabezal*. Permite actuar sobre la electroválvula que libera el cabezal del motor del mármol. Al igual que la seta de *Emergencia*, desactiva la alimentación del servo al motor, por lo que las operaciones sobre el servo están deshabilitadas.

2.3.5. Parámetros de configuración

Para configurar ciertos parámetros de la aplicación, se ha creado un fichero de texto que incluye parejas de elementos nombre-valor, admitiendo también comentarios. A continuación se enumeran y explican brevemente todos ellos. La existencia de algunos parámetros se debe a razones que se expondrán posteriormente en la sección de Arquitectura software de Mármol (sección 4.6).

- Constantes de posición.

- `CONSIGNA_POSICION_POR_DEFECTO`. Valor al inicio de la aplicación del campo *Consigna*.
- `INCREMENTO_POSICION_POR_DEFECTO`. Valor al inicio de la aplicación del campo *Incremento*.
- `FIN_CARRERA_NEGATIVO_SW`. Valor respecto a la posición de home del final de carrera negativo software.
- `FIN_CARRERA_NEGATIVO_HW`. Valor respecto a la posición de home del final de carrera negativo físico.
- `FIN_CARRERA_POSITIVO_SW`. Valor respecto a la posición de home del final de carrera positivo software.
- `FIN_CARRERA_POSITIVO_HW`. Valor respecto a la posición de home del final de carrera positivo físico.
- `VENTANA_FINALES_DE_CARRERA`. Al alcanzar un final de carrera físico, el patín se detiene. Sin embargo, esa parada puede no ser inmediata y por tanto no se sabe a qué distancia del final de carrera ha quedado el patín. Para sacar al patín del final de carrera, se desplaza el servo en sentido contrario al de entrada. Cuando se cubre un desplazamiento igual o mayor que el de este parámetro, se considera que se ha salido del final de carrera.
- `VENTANA_EN_POSICION`. Margen alrededor de la posición objetivo en el que se considera que el objetivo ha sido alcanzado.
- `TIEMPO_EN_VENTANA_EN_POSICION`. Tiempo que debe mantenerse el patín dentro de la ventana de posición alrededor de la posición objetivo para que se considere que ésta se ha alcanzado.

- Constantes de velocidad.
  - `VELOCIDAD_[1-5]`. Valores de las 5 velocidades de desplazamiento permitidas.
- Valores para el homing.
  - `ACELERACION_DURANTE_HOMING`. Aceleración durante la búsqueda de la posición de home.
  - `VELOCIDAD_BUSQUEDA_HOME`. Velocidad durante la búsqueda de la posición de home.
- Constantes motor-regla.
  - `CUENTAS_POR_MM`. Cuentas (marcas sobre la regla instalada en el mármol) por milímetro.
- Constantes de desplazamiento manual.
  - `DURACION_MAXIMA_TOQUE_PARA_MINIPASO_MANUAL`. Define la duración máxima de la pulsación del control de movimiento para que se considere un minipaso.
  - `DURACION_DESPLAZAMIENTO_MINIPASO_MANUAL`. El desplazamiento de un minipaso se calcula de la siguiente forma: `Desplazamiento=Vactual*DURACION_DESPLAZAMIENTO_MINIPASO_MANUAL`.
  - `INTERVALO_ENTRE_PASOS_MANUAL`. Define el tiempo entre cada paso de desplazamiento.
  - `DURACION_DESPLAZAMIENTO_PASO_MANUAL`. El desplazamiento de un paso se calcula de la siguiente forma: `Desplazamiento=Vactual*DURACION_DESPLAZAMIENTO_PASO_MANUAL`.

- Entradas digitales servo. Parámetros que asocian a las entradas digitales del servo una función.
  - `ENTRADA_DIGITAL_SERVO_BOTONERA_SETA_EMERGENCIA`
  - `ENTRADA_DIGITAL_SERVO_FIN_CARRERA_NEGATIVO`
  - `ENTRADA_DIGITAL_SERVO_FIN_CARRERA_POSITIVO`
  - `ENTRADA_DIGITAL_SERVO_PRESOSTATO`
  - `ENTRADA_DIGITAL_SERVO_BOTONERA_EMBRAGUE`
  - `ENTRADA_DIGITAL_SERVO_ERROR`
- Salidas digitales servo. Parámetros que asocian a las salidas digitales del servo una función.
  - `SALIDA_DIGITAL_SERVO_ELECTROVALVULA`
- Entradas digitales tarjeta DAQ. Parámetros que asocian a las entradas digitales de la tarjeta DAQ una función.
  - `ENTRADA_DIGITAL_DAQ_BOTONERA_DERECHA`
  - `ENTRADA_DIGITAL_DAQ_BOTONERA_IZQUIERDA`
  - `ENTRADA_DIGITAL_DAQ_BOTONERA_VELOCIDAD`
- Salidas digitales tarjeta DAQ. Parámetros que asocian a las salidas digitales de la tarjeta DAQ una función.
  - `SALIDA_DIGITAL_DAQ_DESENCLAVAMIENTO_FIN_CARRERA_NEGATIVO`
  - `SALIDA_DIGITAL_DAQ_DESENCLAVAMIENTO_FIN_CARRERA_POSITIVO`

Capítulo 3

Selección de tecnologías

3.1. Lenguajes de programación y entorno gráfico

La elección del lenguaje de programación sobre el que codificar un proyecto es un paso importante por las implicaciones que la decisión conlleva.

Capacidades como la legibilidad del código y la modularidad son siempre deseables para que el lenguaje de programación se comporte como una herramienta de desarrollo y no como un problema que acabe siendo fuente de los siempre indeseables errores.

Además, en un proyecto industrial como este centrado en la monitorización y control de varios dispositivos, la capacidad de atender varios subsistemas de forma solvente mediante un buen mecanismo de concurrencia es imprescindible.

Desde 1987 el grupo de investigación CTR ha trabajado en la aplicación de sistemas de tiempo real a controladores industriales, robots e instrumentación utilizando Ada como lenguaje de referencia.

Dados los requisitos y por estos motivos se ha escogido Ada como lenguaje de programación para este desarrollo.

Sin embargo, la heterogeneidad del sistema manejado impide que los dominios de Ada se extiendan a todos los componentes. Cuando se desciende a un nivel próximo al hardware es común encontrarse con los lenguajes C/C++. En este proyecto se manejan de forma directa componentes hardware como tarjetas CAN y DAQ y la cámara conectada mediante interfaz Ethernet.

Cada uno de ellos dispone de sus propios controladores y APIs. Dada la importancia de C/C++ en componentes de acceso al hardware, no es de extrañar que las tecnologías que se emplean para manejar estos componentes impliquen su utilización.

El hecho de contar con estos dos lenguajes de programación en un mismo proyecto no supone un problema. Ada dispone de un mecanismo mediante el cual se puede utilizar código de C/C++ desde sus fuentes, con lo que la integración es completa.

3.1.1. GtkAda

Del lenguaje de programación Ada se ha aprovechado también la posibilidad de utilizar para el diseño de la interfaz gráfica de usuario la librería Gtk+ [Ada13]. Gtk+ permite crear modernas interfaces y es la base del escritorio GNOME de Unix, aunque Gtk+ también está disponible para otros sistemas como Windows. Una de las vías de creación de la interfaz consiste en codificarla

en ficheros XML creados mediante editores visuales, lo cual resulta sencillo y manejable. La adaptación a lenguaje Ada de Gtk+ se denomina GtkAda.

3.2. Sistema operativo Windows

Microsoft Windows es una familia de Sistemas Operativos (SO) desarrollados y vendidos por Microsoft. Su historia comienza en 1981 con un sistema operativo muy diferente desarrollado por Microsoft para el primer computador personal de IBM y conocido como MS-DOS [Sta09].

La evolución del hardware de la época hizo que MS-DOS acabase aprovechando el entorno muy por debajo de sus posibilidades. Por su parte Macintosh desarrolló un sistema operativo basado en interfaz gráfica que resultaba insuperable por su facilidad de uso. En 1990 surgió Windows 3.0 como primer sistema de Microsoft que integraba una interfaz gráfica de usuario. Sin embargo, las limitaciones de Windows 3.0 abocaron a Microsoft a la creación de un nuevo sistema operativo desde cero que denominó Windows NT.

Windows NT explotaba las capacidades de los procesadores contemporáneos y proporcionaba multitarea en un entorno mono o multiusuario. La evolución de NT ha seguido con el paso de los años hasta las versiones actuales de Windows.

Estimaciones recientes basadas en el tráfico web calculan que la cuota de mercado de Windows en sistemas de escritorio es del 91.56 % [Net13].

El Laboratorio de Metrología de ENSA utiliza los PCs en labores de ofimática y para el control y monotorización de máquinas de calibración. En ambos casos se utiliza Windows.

En este proyecto se optó por seguir en la línea de utilizar el SO Windows por 3 razones principales:

- Familiaridad. Los potenciales usuarios están habituados al sistema y lo utilizan cada día en su puesto de trabajo.
- Software necesario. Los controladores y librerías para manejo de los dispositivos del sistema están disponibles (a veces de manera única como ha ocurrido con el software de la cámara) para el sistema operativo de Microsoft.
- Restriciones de tiempo real. No se apreciaron limitaciones en el sistema operativo Windows que impidieran realizar el sistema especificado. El uso del servo para cerrar los lazos de control del movimiento descarga a la aplicación de la necesidad de hacer operaciones en *tiempo real estricto*. Por otro lado la respuesta a los eventos de la interfaz gráfica se puede considerar que es de *tiempo real no estricto*. El operador necesita tener una sensación de reactividad casi inmediata. Sin embargo, los tiempos de respuesta habituales de un sistema Windows son suficientes para conseguir esta sensación.

3.2.1. Controladores de dispositivo WDM

Los controladores de dispositivo (en inglés y comúnmente *drivers*) son programas que permiten al sistema operativo manejar el hardware conectado al computador [GP13] [BL01].

Lo común es que los fabricantes del dispositivo proporcionen el controlador junto con el hardware. Así ha ocurrido con los dispositivos que en este proyecto han precisado ser instalados en el PC: cámara (controlador Sapera DALSA), tarjeta CAN (controlador Advantech) y tarjeta DAQ (controlador DAQNavi de Advantech). Sin embargo a este último controlador se le encontró una limitación consistente en una interacción indeseada con el *mecanismo de excepciones* de Ada.

La falta de código fuente del controlador, cuyo fabricante sólo suministra código binario, impidió cualquier labor de depuración que aclarase el origen del problema. Es en este momento cuando surge la opción de crear un controlador de dispositivo propio para la tarjeta DAQ.

El controlador se ha creado siguiendo Windows Driver Model (WDM), el modelo de creación y uso de controladores vigente en Windows desde la versión 2000. La principal aportación que aprecia el usuario con WDM es la tecnología Plug and Play (PnP). PnP permite que un dispositivo recién conectado al computador sea detectado e instalado de forma automática sin necesidad de configuración hardware (jumpers). La asignación de recursos al dispositivo será dinámica, pudiendo incluso desactivar y reactivar el dispositivo sin reiniciar el sistema.

Otra característica destacable de los controladores WDM es que permiten gestionar los dispositivos utilizando la abstracción del sistema de ficheros. De esta forma, para utilizar un dispositivo se crea un manejador y se ejecutan sobre él operaciones de lectura y escritura como si de un fichero se tratara.

En Windows, los controladores se alojan en el núcleo del sistema. Esto les otorga acceso a la memoria y dispositivos externos. Sin embargo también los convierte en una pieza crítica del sistema en la que un fallo puede provocar un indeseable bloqueo general, la conocida pantalla azul.

Estructura del código del controlador

Los controladores Windows se codifican en ficheros de código C/C++. WDM impone una serie de requisitos respecto a las funciones que es necesario incluir en estos ficheros. Las funciones se dividen en tres grupos según su propósito:

- Funciones de inicialización y limpieza. Sirven de punto de entrada y salida al controlador.
- Funciones para tratar peticiones de E/S. Gestionan las peticiones procedentes del usuario. Como se ha comentado, la relación del usuario con el controlador se establece mediante la abstracción de los ficheros. En este grupo se incluyen por tanto funciones para abrir/cerrar el controlador y leer/escribir datos.
- Funciones manejadoras de mensajes Plug and Play. Se encargan de dar soporte a los mensajes que se generan en respuesta a eventos PnP. Algunos de los eventos/funciones más importantes son:
  - Añadir dispositivo (`AddDevice`). Debe localizar el hardware a controlar, reservar recursos como puertos, interrupciones o DMA y dar un nombre al dispositivo para que pueda ser accedido en el sistema.
  - Iniciar dispositivo (`IRP_MN_START_DEVICE`). Se invoca cuando el gestor PnP ha asignado los recursos hardware al dispositivo. Aquí se toma nota de las direcciones de estos recursos para que puedan ser accedidos.
  - Detener dispositivo (`IRP_MN_STOP_DEVICE`). Libera los recursos que se hubieran tomado hasta el momento.

Compilación e instalación

A pesar de tratarse de código escrito en C/C++, la programación de controladores precisa de un compilador suministrado por Microsoft y creado expresamente para esta labor. La compilación además depende de la versión del sistema operativo y de la arquitectura por lo que se debe

generar una versión para cada sistema objetivo. La compilación genera un único fichero `.sys` de controlador.

La instalación automatizada basada en diálogo característica de los controladores PnP se basa en la utilización de un fichero de texto con extensión `INF`.

La instalación del controlador provoca los siguientes cambios en el sistema:

- Se crean nuevas entradas en el Registro de Windows que describen el controlador, el modo de carga y cualquier otro dato de configuración.
- Los ficheros del controlador son copiados a su correspondiente directorio de sistema.

Los ficheros INF cuentan con un identificador que es el mismo que emplea el dispositivo cuando se anuncia en un bus PnP. De esta forma se establece la relación dispositivo-controlador.

3.3. GigE Vision: Estándar para cámaras digitales industriales

En el año 2006 un grupo formado por una docena de las compañías más importantes del sector de visión por computador para aplicaciones industriales publicaron el estándar GigE Vision [AIA10] [GPAR11]. Este estándar define una interfaz de comunicación para aplicaciones de visión en el ámbito de las cámaras de alto rendimiento industriales comunicadas mediante red Ethernet. Con este estándar se pretende unificar los protocolos de comunicación que se estaban empleando en las cámaras industriales. De esta forma se permite que hardware y software de distintos fabricantes pueda interoperar sin problemas.

En este proyecto se ha escogido una cámara que cumpliera con el estándar GigE Vision. Algunos motivos para su elección son:

- Tecnología digital frente a analógica:
  - Mayor frecuencia de captura.
  - Ausencia de ruido por transmisión.
  - Facilidad de procesamiento. Evita conversores analógico-digital.
- Se está convirtiendo en un denominador común dentro de los catálogos de cámaras industriales.
- Emplea Ethernet. Buena capacidad de interconexión, sencillez y velocidad.

La cámara empleada dispone de un software controlador para su manejo. Internamente utiliza el protocolo GigE Vision para comunicarse a través de Ethernet con la cámara. Sin embargo, al usuario del controlador se le enmascaran los detalles internos del estándar con una interfaz más amigable.

3.4. Servos CAN de Advanced Motion Controls

Advanced Motion Controls (AMC) es una de las principales compañías en el sector de fabricación de servos. En ENSA se llevan utilizando servos de este fabricante en decenas de desarrollos como los brazos robóticos Sophia y Caseva desarrollados en colaboración con el grupo CTR de la Universidad.

Sin embargo, hasta este proyecto los servos utilizados eran de tipo analógico. De forma experimental se decidió emplear en este proyecto una tecnología más novedosa basada en servos digitales.

Un servo digital es básicamente lo mismo que un servo analógico, excepto porque cuenta con un microprocesador que analiza la señal de entrada y controla el motor [Fut13].

Este microprocesador recibe la señal de entrada y aplica parámetros predefinidos a la señal antes de enviarla al servomotor. De esta forma, se pueden hacer ajustes en la función de salida y en el rendimiento.

AMC tiene en el mercado varios tipos de servos digitales [AMC13c]. Las diferencias entre unos u otros radican principalmente en el método de interconexión con el PC, es decir, el método para recepción de instrucciones. En este caso se optó por utilizar un servo con puerto para *bus CAN* porque se apoya en un protocolo ampliamente extendido en el ámbito industrial por lo que cuenta con soporte en los sistemas operativos más comunes y amplia oferta de hardware a un coste contenido. Otra alternativa hubiese sido la *interfaz serie RS232* que hubiese resultado más económica pero con una limitación a priori en el ancho de banda que puede restringir las aplicaciones. Por último AMC ha desarrollado servos con *interfaz EtherCAT* que permiten transmisión a gran velocidad usando un estándar basado en Ethernet. Esta última opción resulta interesante por lo novedoso de la tecnología. Sin embargo, se trata de un tipo de bus que aún no está muy extendido.

3.4.1. Bus CAN

CAN (Controller Area Network) es un bus serie empleado para dar soporte a las comunicaciones en sistemas de control distribuidos. Inicialmente desarrollado para su uso en automóviles por Robert Bosch GmbH a finales de los 80, su uso se ha extendido a otros campos hasta convertirse en el más difundido de la industria europea [UPV13].

Nació para resolver el problema de comunicación existente entre los dispositivos existentes en el automóvil. Su creciente número hacía que el cableado punto a punto resultase complejo y costoso. CAN se basa en una topología en bus, aquella en la que hay un único canal de comunicaciones en el que se conectan todos los dispositivos.

Entre sus características más importantes se encuentran la transmisión de tiempo real con tiempos de respuesta deterministas y la posibilidad de establecer prioridades entre los mensajes.

El estándar CAN sigue el modelo de referencia OSI para interconexión de sistemas. En concreto, se ocupa de los niveles físico y de enlace de OSI. Algunas de las características más relevantes que se definen en cada nivel son las siguientes:

- Nivel físico.
  - Especificación del cableado. Consta de 2 líneas. Es necesario instalar resistencias de terminación de bus en cada extremo al construir la red.
  - Conexión multipunto. Posibilidad de cambio dinámico en el número de nodos. Comunicación broadcast. Cada nodo puede filtrar o procesar los mensajes que desee.
  - No existe direccionamiento al nodo, sino al contenido del mensaje.
  - Acceso al bus basado en CSMA/CD con prioridad de mensajes basada en su identificador.
  - Mecanismos avanzados de detección de errores.
- Nivel de enlace.

- Acceso al bus. Cada nodo empieza enviando el identificador del mensaje y monitoriza el bus para ver si lo que hay realmente en el bus es lo que ha querido poner. Si se detecta concurrencia en el acceso, se inhibe el nodo de menor prioridad.
- Trama de datos. La trama de datos CAN puede transportar hasta 8 bytes de datos.

Por encima de estos niveles físico y de enlace se pueden encontrar varios niveles de aplicación posibles. El servo de este proyecto emplea el protocolo CANopen.

3.4.2. Protocolo CANopen

CANopen es una de las implementaciones de la capa de aplicación OSI sobre el bus CAN [CIA13] [AMC13a]. El concepto central en que se basa CANopen son los *objetos*. Cada objeto es equivalente a una localización de memoria en la que se almacena un valor. Cada dispositivo cuenta con un objeto único para cada parámetro que precisa ser almacenado o usado. El conjunto de estos objetos constituye el denominado *diccionario de objetos*. Para utilizar un dispositivo CANopen es necesario que el fabricante proporcione el diccionario de objetos, ya que este describe de forma implícita su funcionalidad. Dependiendo del dispositivo, los objetos CANopen pueden contener una información u otra. Lo habitual es que cuenten con los siguientes campos:

- Índice y subíndice. Dirección del objeto en el diccionario.
- Nombre del objeto y descripción.
- Tipo de dato.
- Atributo de acceso. Lectura y/o escritura.

Estructura del mensaje CANopen

La estructura de un mensaje CANopen lo divide en dos partes (Figura 3.1):

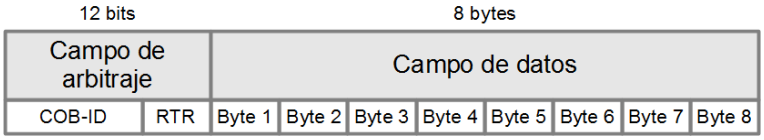

Figura 3.1: Esquema de la trama CANopen.

- Campo de arbitraje. Establece la prioridad del mensaje. A menor valor, mayor prioridad. Este campo, está definido en la especificación de CAN. A su vez, este campo se compone de:
  - COB-ID. Identifica el tipo de mensaje CANopen y el número de nodo. Se calcula mediante la suma de ambos campos:
    - Tipo de mensaje. Se define un rango para cada tipo de mensaje. Los tipos de mensajes se citarán a continuación.
    - Número de nodo. Cada nodo de una red CANopen debe tener un identificador único que se fija a nivel hardware.
  - RTR. Se utiliza en casos concretos de petición de datos desde un nodo a otro.
- Campo de datos. Su contenido depende del tipo de mensaje CANopen.

Tipos de mensajes CANopen

- Mensajes administrativos. Se utilizan para control y gestión de la red CANopen. Al conectar o resetear un nodo se envía un mensaje de *Bootup*. A partir de entonces, se utilizan estos mensajes para transicionar entre los estados *Pre-operacional*, *Operacional* y *Detenido* que permiten identificar el estado de cada nodo en la red.
- Mensajes SDO Y PDO. Son los dos métodos posibles para leer y escribir datos de objetos.
  - SDO. Protocolo con confirmación. En caso de lectura de un nodo, la confirmación incluye los datos leídos.
  - PDO. Intercambio de datos sin confirmación. Implementan un modelo productor-consumidor en que la transmisión por parte del productor puede depender de eventos periódicos, mensajes de activación concretos, etc.
- Otros mensajes. Para sincronización de eventos, trabajo con relojes globales y emergencia.

Protocolo CANopen en los servos CAN de Advanced Motion Controls

Advanced Motion Controls dispone de una serie de servos digitales cuya interfaz de comunicación con el PC es el bus CAN y el protocolo de aplicación sobre el bus es CANopen.

El fabricante pone a disposición del usuario un documento donde se recoge el diccionario de objetos CANopen del servo [AMC13a]. Algunas operaciones como conocer la posición actual se solventan fácilmente con la lectura del objeto CANopen correspondiente. Otras requieren de mayor trabajo sobre el servo para ser llevadas a cabo. Algunos de los aspectos importantes recogidos en el diccionario de objetos del servo son los siguientes:

- Máquina de estados de control. Los servos CANopen operan basándose en una máquina de estados de control en la que cada estado conlleva un comportamiento específico. Por ejemplo, el estado *Operation Enabled* indica que el servo ejecuta los comandos que se le hayan especificado previamente. El estado *Fault* indica error y por tanto el servo deshabilita el motor. *Quick Stop* permite detener el movimiento de forma ordenada. La escritura del objeto `ControlWord` permite transicionar entre estados.
- Homing. Es el proceso de determinar la posición de referencia. Los servos CAN de AMC permiten muchos tipos de homing, basándose en señales de límites, índices y contactos de home. En función de las necesidades de cada caso se puede programar el servo para aceptar los diferentes tipos de rutinas.
- Modos de operación. Los servos CAN de AMC basan el control del motor en los mecanismos de control por realimentación PID. Este control puede ejercerse sobre magnitudes de corriente eléctrica, velocidad y posición. El ajuste sobre todas ellas resulta más certero que sobre sólo una. Los modos de operación permiten indicar sobre qué magnitudes se produce el ajuste, existiendo las posibilidades de corriente, velocidad+corriente y posición+velocidad+corriente.

3.4.3. Configuración del servo mediante herramienta Driveware

Los servos CAN de AMC disponen de una aplicación para PC denominada Driveware [AMC13b]. Permite la configuración y prueba del servo utilizando el puerto serie del PC.

Algunos aspectos de la configuración del servo se almacenan en *memoria no volátil*. Esto permite que estén disponibles cada vez que el servo se inicia. Los parámetros de configuración se almacenan automáticamente de esta forma, por lo que es posible establecer mediante Driveware un perfil de configuración a emplear con el servo de forma permanente.

El diccionario de objetos CANopen del servo recoge todos estos parámetros de configuración. Sin embargo, el ajuste de algunos de ellos, como los parámetros de los lazos PID, supone un verdadero quebradero de cabeza. Es por ello que Driveware constituye una herramienta casi imprescindible para este tipo de cometidos al facilitar la elección de los valores de configuración más apropiados y poder comprobar los resultados de forma inmediata mediante herramientas de monitorización como osciloscopio y multímetro digital.

Algunos de los aspectos más relevantes que se pueden (y deberían) configurar con esta herramienta son:

- Parámetros del motor. Retroalimentación del motor.
- Parámetros de retroalimentación de posición y velocidad.
- Asignación de entradas y salidas del servo.
- Fijación de limitaciones para parámetros de corriente, posición, velocidad, tensión, etc.
- Gestión de eventos.
- Ajuste de los lazos PID.

Capítulo 4

Diseño de la arquitectura software e implementación

4.1. Ciclo de vida del software

El ciclo de vida del software define los procesos involucrados en las fases de desarrollo, explotación y mantenimiento de un producto software abarcando desde la definición de requisitos hasta la finalización de su uso [RG13].

Las *fases finales de explotación y mantenimiento* de este software se centran en dar soporte técnico a los usuarios y aceptar cuantas sugerencias de mejora crean oportunas para estudiarlas y adoptarlas si se consideran útiles en versiones posteriores del software.

Pero es la *fase de desarrollo* del software la que centra el interés en estas páginas. En este proyecto se ha optado por adoptar un modelo de desarrollo consistente en cuatro procesos, que se consideraron suficientes para la dimensión del proyecto y la consecución de forma estructurada y ordenada del objetivo planteado. A saber:

1. Especificación funcional. Describe cómo funcionará el producto desde el punto de vista del usuario. No tiene en cuenta (hasta cierto punto) la forma en que esos requisitos son implementados.

2. Diseño arquitectónico. Describe la estructura del programa, las relaciones entre sus partes y el comportamiento de las mismas.

3. Implementación. Traslada el diseño a la realidad software mediante la codificación.

4. Pruebas. Verifica el funcionamiento del software. Se depuran fallos y comprueban las divergencias con la especificación funcional.

Los procesos recién enumerados pueden abordarse de formas variadas [GH13]. Los denominados modelos de ciclo de vida proponen distintas alternativas de transición entre unas fases y otras. En ocasiones es difícil encajar un desarrollo en un modelo concreto pues se utilizan características de varios de ellos. En este caso se ha optado por un *modelo incremental* ya que responde a un modelo basado en la evolución iterativa del software.

Así, el software se desarrolla en una serie de versiones incrementales donde las primeras fases están más cerca de *modelos y prototipos* y las últimas fases consisten en una aproximación cada vez mayor al *sistema final*. Cada una de estas versiones sigue a su vez secuencias lineales

escalonadas como las del *modelo en cascada*, donde se van sucediendo las fases en orden y el paso de una fase a otra sólo se produce si la anterior ha finalizado.

El desarrollo de este software comienza con el acuerdo de los Laboratorios de Metrología y Automática de ENSA para el desarrollo de la medidora de coordenadas horizontales de 4 m. Con este acuerdo, Metrología busca cubrir las necesidades de una máquina nueva y Automática utilizar el proyecto como un test que permita evaluar varias tecnologías nuevas para el desarrollo de futuros controladores (servos digitales, cámaras GigE Vision, etc.) que les puedan ser de utilidad.

El acuerdo se plasma mediante una *especificación funcional* del software basada en las pretensiones de los usuarios finales. En este momento se crea además un *prototipo hardware* pensado para probar la tecnología de los servos digitales. Este prototipo lo constituyen el propio servo, un motor y una dinamo tacométrica.

Una vez dados estos primeros pasos, comienzan las iteraciones incrementales de desarrollo del software:

- Iteración 1. El objetivo es la consecución de una *prueba que demuestre el correcto funcionamiento de la tecnología del servo* (Figura 4.1). La prueba se basa en la monitorización de datos como la posición y el envío de comandos de movimiento a consigna de posición. Para ello fue necesario construir toda la jerarquía de software que permitiera utilizar la tarjeta CAN y el protocolo CANopen.

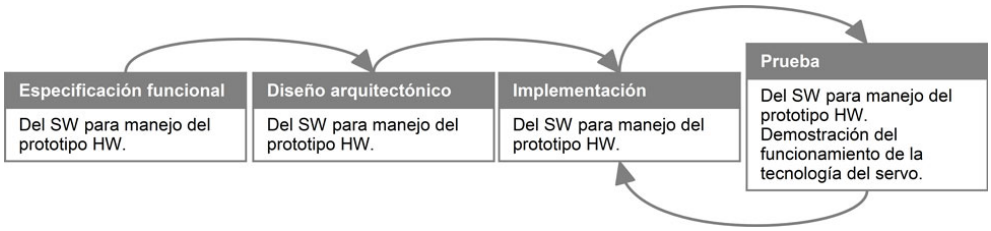

Figura 4.1: Esquema de las fases de la primera iteración del ciclo de vida del software.

- Iteración 2. Retomando la especificación software acordada en ENSA, se crea un *prototipo rápido* de la GUI con el objetivo de recibir sugerencias de los usuarios de Metrología (Figura 4.2). Al no tratarse de una GUI cuya utilización vaya a ser excesivamente compleja, el hecho de que sus controles no funcionen no supone un obstáculo para su comprensión por parte del usuario final.

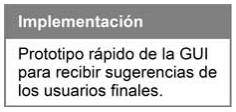

Figura 4.2: Esquema de la única fase de la segunda iteración del ciclo de vida del software.

- Iteración 3. Esta iteración recoge las sugerencias realizadas sobre el prototipo de la fase anterior para añadirlas a la última especificación funcional existente (Figura 4.3). El objetivo final es la *creación de la aplicación completa sobre el prototipo hardware*. La primera versión continúa con el trabajo de la Iteración 1. Versiones posteriores van añadiendo funcionalidades al sistema hasta alcanzar el sistema completo.

- Iteración 4. De forma paralela, en ENSA habían trabajado en la construcción de la máquina final en el Laboratorio de Metrología (Figura 4.4). En esta fase, se adapta el software de

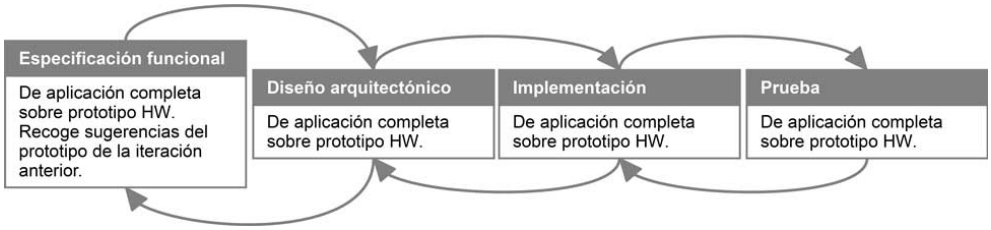

Figura 4.3: Esquema de las fases de la tercera iteración del ciclo de vida del software.

la Iteración 3 para su funcionamiento en el entorno final. Las pruebas permiten detectar además de fallos software, limitaciones y fallos en el hardware que son también modificados.

Figura 4.4: Esquema de las fases de la cuarta iteración del ciclo de vida del software.

4.2. Visión general de la arquitectura

La arquitectura software del proyecto se guía por los principios del *diseño modular* y la *metodología orientada a objetos*. Esto facilita la comprensión del problema y su descomposición, haciendo más fácil la implementación y posteriores modificaciones y extensiones.

El software que ocupa esta memoria abarca varios ámbitos: desde el tratamiento directo con el hardware hasta la gestión de la interfaz que interacciona con el usuario. En el denominado *diseño descendente* se opta por abordar estos ámbitos descomponiendo el software en capas empezando por el nivel superior hasta aproximarse a los niveles inferiores. En este caso el nivel superior corresponde a la aplicación en su conjunto. La descomposición continúa hasta alcanzar los módulos que interaccionan de forma directa con el hardware.

La Figura 4.5 muestra la jerarquía de módulos software del proyecto. En el nivel superior se encuentra la aplicación principal de Mármol. Si se continúa descendiendo, se encuentra en primer lugar la librería para el manejo del protocolo CANopen, a continuación las interfaces de manejo del hardware, los controladores y por último el hardware en sí. En sombreado gris aparecen los módulos cuyo diseño y desarrollo ha sido realizado dentro de este proyecto.

Para el diseño de la arquitectura se ha recurrido al Lenguaje Unificado de Modelado (Unified Modeling Language - UML). UML permite describir el sistema mediante diagramas que muestran el sistema desde puntos de vista estructurales, de comportamiento, etc.

Las secciones posteriores se ocupan de explicar el diseño arquitectónico de cada uno de los módulos de esta jerarquía.

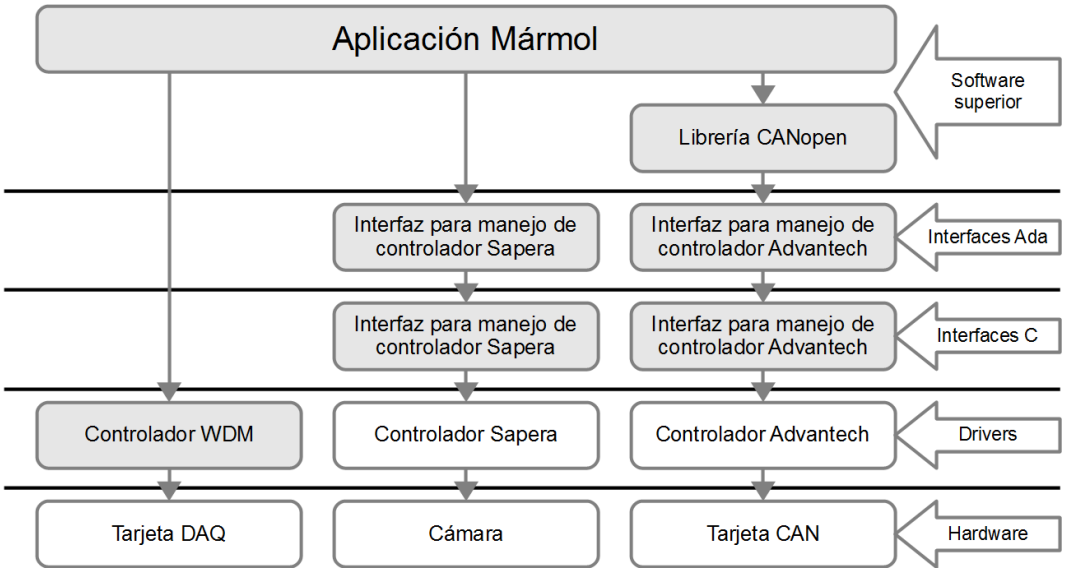

Figura 4.5: Esquema del software. En gris sólido aparecen los módulos software de desarrollo propio de este proyecto.

4.3. Arquitectura de las interfaces sobre controladores de dispositivo

Las interfaces sobre controladores de dispositivo han sido creadas para, apoyándose sobre controladores de dispositivo (drivers) ya existentes, ofrecer una interfaz de uso básico y sencillo del dispositivo subyacente por parte de otras entidades software.

Muchos de estos dispositivos pueden manejarse mediante un esquema similar consistente en la *inicialización o apertura* del dispositivo, su consecuente *finalización o cierre* y operaciones de *lectura y/o escritura de datos*. Sin embargo, estas operaciones básicas pueden complicarse según los dispositivos se vuelven más sofisticados. Además está el hecho de que el fabricante no puede ceñirse a dar un controlador sencillo que desaproveche las capacidades del dispositivo subyacente. Por ello, los controladores acaban complicándose hasta resultar engorrosos de utilizar también en usos básicos.

En consecuencia se decidió abordar la utilización de estos dispositivos creando interfaces adaptadas al uso que se le fuera a dar, enmascarando bajo funciones sencillas el conjunto de llamadas al controlador de dispositivo dado por el fabricante.

Como ya se expuso en la Sección 3.1, el lenguaje C/C++ es muy común a nivel de trabajo directo con el hardware y en este caso ambos controladores de dispositivo manejados contaban con acceso a la librería mediante lenguaje C/C++. Por ello, el primer nivel de estas interfaces también está codificado en C/C++.

El segundo nivel es una adaptación al lenguaje Ada de la interfaz C/C++, pues Ada es el lenguaje de programación en el que está escrita la aplicación principal que utiliza en última instancia el dispositivo. La Figura 4.6 corresponde al diseño estructural "tipo" de estas interfaces empleando los diagramas de clases UML.

Un aspecto importante a tener en cuenta es que las interfaces debían disponer de mecanismos que informen y ayuden a depurar errores originados en el uso de estos dispositivos. Para ello, todas las funciones devuelven un resultado numérico que informa del estado de la operación en cuestión.

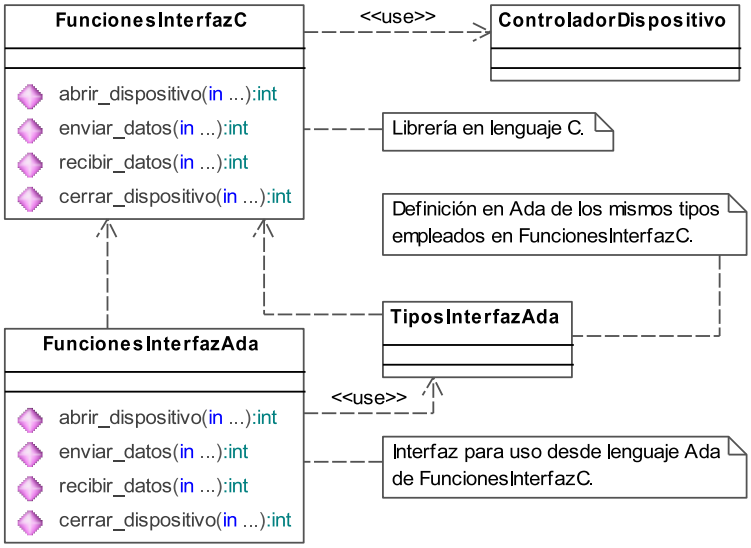

Figura 4.6: Diagrama de clases UML de la arquitectura "tipo" para las interfaces sobre controladores de dispositivo.

A destacar también el hecho de que estas interfaces pueden compilarse separadamente y ser empleadas con independencia del software principal.

4.3.1. Interfaz sobre tarjetas CAN de Advantech

El objetivo de esta interfaz es dotar al código escrito en lenguaje Ada de la capacidad de establecer una comunicación CAN mediante las tarjetas CAN de Advantech de forma sencilla.

Las operaciones disponibles permiten abrir y cerrar una interfaz CAN y enviar y recibir mensajes CAN a través de ella.

4.3.2. Interfaz sobre cámaras digitales GigE Vision de DALSA

El objetivo de esta interfaz es dotar al código escrito en lenguaje Ada de la capacidad de establecer comunicación con una cámara GigE Vision de DALSA y obtener imágenes procedentes de ella de forma sencilla.

Las operaciones disponibles permiten inicializar y finalizar una conexión con cámara DALSA y capturar un frame con o sin una línea de referencia sobreimpresa.

4.3.3. Implementación y lecciones aprendidas

La mayor dificultad de trasladar este diseño a código viene de las particularidades que cada fabricante da a cada librería de manejo de dispositivo. A veces el fabricante aborda el manejo del dispositivo desde un punto de vista conceptual que resulta sencillo. Otras veces el abanico de posibilidades se amplía hasta convertirse en algo más complejo. Lo normal es que el fabricante ponga a disposición del usuario un documento explicativo de la API en la misma web de donde se obtiene el controlador o en los directorios del sistema donde éste se ha instalado. Además es común que en estos directorios se incluyan ejemplos útiles que permitan realizar pruebas sobre el dispositivo en cuestión. Es por ello que si la utilización que se va a hacer del dispositivo no es excesivamente compleja, a veces sea suficiente con adaptar alguno de estos ejemplos.

Al probar la librería dada por el fabricante es aconsejable realizar programas de prueba que sean lo más próximos al uso que se va a hacer desde Ada. Así es posible evaluar si tanto el dispositivo como el uso que se hace de él son satisfactorios y valorar posibles alternativas en su caso. También pueden detectarse posibles limitaciones, como ocurrió con la API de manejo de la cámara DALSA, cuya compilación solo era posible para Windows de 32 bits. No es menos importante comprobar de forma exhaustiva en este nivel que la librería funciona correctamente para poder proceder de forma segura a realizar la adaptación a Ada.

Un aspecto común en el manejo de la mayoría de estas APIs es que las llamadas devuelven un código de error. Es vital hacer un tratamiento de estos códigos de error, bien optando por alguna alternativa que lo solucione o informando al usuario de la situación. Es importante dar al usuario un mecanismo mediante el cual pueda obtener la información concreta del error para que pueda identificar de forma directa el problema. Por ejemplo se pueden crear macros C/C++ que comprueben estos códigos de error para evitar escribir el código de comprobaciones demasiadas veces.

Las cabeceras de las funciones a exportar al usuario junto con las constantes globales es aconsejable que estén localizadas en un fichero de cabecera `.h`. De esta forma el usuario conoce los aspectos de la librería a los que tiene acceso sin tener conocimiento de detalles internos. Las constantes globales citadas no van a poder ser modificadas desde el lenguaje Ada. Es por ello que es importante decidir si merecen formar parte de los parámetros de llamada de las funciones en caso de que interese cambiarlas.

A la hora de trasladar la librería C/C++ a Ada lo más complicado es establecer la correspondencia entre los tipos de datos C/C++ y Ada. En lenguaje de programación Ada dispone de un paquete `Interfaces.C` con tipos de datos Ada correspondientes a tipos de datos C. Los punteros y dobles punteros se consiguen mediante direcciones de memoria Ada `System.Address` y `access System.Address` respectivamente.

Una vez creadas las cabeceras de las funciones C/C++ en Ada, su conexión se realiza mediante la directiva `pragma Import([C|CPP], <Nombre función Ada>, "<Nombre función C/C++>")`. Después se precisa indicar al enlazador del compilador Ada el fichero `.o` resultante de la compilación de la librería C/C++. A veces la compilación desde Ada requiere enlazar otros objetos o librerías.

4.4. Arquitectura del controlador WDM para tarjeta de E/S digitales PCI-1761

El controlador para tarjeta de entradas y salidas digitales DAQ PCI-1761 fue diseñado para seguir el modelo Windows Driver Model (WDM). Tal y como se explicó en la Sección 3.2.1, para que un controlador de dispositivo cumpla con WDM, debe implementar un conjunto de funciones manejadoras.

Una de las características también citadas de este tipo de controladores es que son utilizados mediante una interfaz muy similar a la de manejo de ficheros, donde el dispositivo se puede abrir/cerrar y se permite la lectura/escritura de datos.

La Figura 4.7 muestra la única clase que compone el controlador junto con la representación esquemática de la API de manejo de ficheros en Windows a través de la cual se utiliza el controlador.

Figura 4.7: Diagrama de clases UML de la arquitectura del controlador WDM para tarjeta de E/S digitales PCI-1761.

4.4.1. Implementación y lecciones aprendidas

Crear un controlador de dispositivo para Windows es un proceso complicado, sobre todo si se realiza por primera vez. El hecho de ser una pieza de software alojada en el núcleo del sistema lo convierte en una peligrosa forma de desestabilización del mismo. Por eso una de las mejores primeras decisiones es realizar una *compilación cruzada*, esto es, programar el controlador en un PC e instalarlo en otro. De esta forma, los obligados reinicios que provocan los fallos en el controlador no trastocan la actividad sobre el PC de desarrollo. Para automatizar el proceso de compilación e instalación es útil crear una carpeta de red de tal forma que el PC donde se instala el controlador tenga acceso a la carpeta de desarrollo.

Para depurar el código implementado, la mejor opción es utilizar las llamadas que imprimen en la salida de depuración. De esta forma es posible conocer qué puntos va alcanzando el flujo del código y el valor de las variables. La salida de depuración puede visualizarse mediante herramientas como DebugView [Win13b].

Como en otros casos, es muy importante comprobar el valor de retorno de las llamadas a la API de desarrollo de controladores (`NTSTATUS`). Su valor da cuenta de los motivos por los cuales una llamada puede ser fallida. Estos códigos pueden ser también empleados al programar el controlador para dar información de retorno al usuario.

Una de las características del controlador desarrollado es que sólo puede ser instalado en versiones de Windows de 32 bits. El motivo es el denominado *firmado digital de controladores*. Este mecanismo permite verificar la autoría del controlador con el objetivo de no instalar software fraudulento o que pueda comprometer la estabilidad del sistema [Win13a]. En la práctica supone una dificultad administrativa añadida, pues hay que hacer pasar al controlador por varios procesos que incluyen el registro en las instancias competentes de Microsoft hasta conseguir el

firmado. En los sistemas operativos de 64 bits, el firmado es obligatorio.

4.5. Arquitectura de la librería para implementar el protocolo CANopen

La librería Ada de utilización del protocolo CANopen procede de la implementación realizada en un proyecto anterior desarrollado en el año 2000 por Alberto Pigazo López en el propio grupo CTR de la Universidad de Cantabria [PL00].

En aquel proyecto, el objetivo fue conseguir que las aplicaciones desarrolladas en Ada pudieran utilizar el protocolo CANopen con independencia de la implementación concreta de la red de comunicaciones desde el punto de vista hardware (tipo de tarjeta) y software (sistema operativo, driver, etc.). Esta aproximación permite que las aplicaciones siempre utilicen la librería de la misma forma e independientemente de la arquitectura subyacente. Eso sí, es necesario modificar la interfaz de llamadas al hardware en caso de que este cambie, tal y como ha ocurrido aquí.

El software de Pigazo López fue pensado siguiendo una arquitectura orientada a objetos (ver Figura 4.8). Empezando desde el nivel más próximo al hardware, se encuentra la clase `CanBus` que con operaciones para abrir/cerrar y enviar/recibir mensajes utiliza directamente la interfaz sobre tarjetas CAN de Advantech explicada anteriormente en la Sección 4.3.1.

Otra clase denominada `CanopenDevices` se utiliza para almacenar en una estructura de datos todos los nodos de la red con los que se desea establecer comunicación. De cada uno de ellos se guarda su nombre y dirección. Al enviar o recibir un mensaje solo es necesario referenciar la entrada correspondiente de esta estructura de datos para indicar el destino o procedencia del mensaje.

Los cambios realizados sobre estas dos clases son los necesarios para haber adaptado el software anterior a la nueva capa hardware subyacente. Sin embargo, se decidió escribir de nuevo la librería para comprender mejor su funcionamiento, eliminar partes no necesarias en este proyecto y adaptarla al estilo de codificación, nombrado de variables y comentarios de documentación seguidos a lo largo del proyecto.

El resto de la librería se encuentra en la clase `Canopen`. Esta clase recoge las representaciones de los distintos mensajes/tramas que se pueden enviar en el protocolo CANopen y operaciones para enviarlos y recibirlos. Por cuestión de tiempo y comodidad, a la hora de reescribir la librería solo se han conservado los tipos de mensajes relevantes para la comunicación con el servo: mensajes NMT y SDO. Los mensajes PDO fueron descartados porque con SDO se conseguía acceder al diccionario de objetos de forma más directa (PDO requiere configuración previa).

Para representar las tramas se utilizan clases contenedoras de datos estructuradas de acuerdo a una jerarquía que hace uso de la herencia. De esta forma, del *frame principal* cuelgan a su vez los *frames SDO y NMT*. SDO tiene a su vez otros dos frames que heredan de él y lo extienden: *SDO de lectura y escritura*.

Similar ha sido el tratamiento de las funciones que manejan estos frames. Para cada uno de ellos existe una función que permite tomar un *frame CANopen* y transformarlo en un *mensaje CAN* y una función que realiza el proceso contrario. Partiendo del primer par de funciones genéricas se van derivando las implementaciones correspondientes a cada tipo de mensaje.

Para finalizar, el diseño arquitectónico recoge funciones para enviar y recibir frames, además de otras funciones para inicializar o finalizar la red CANopen.

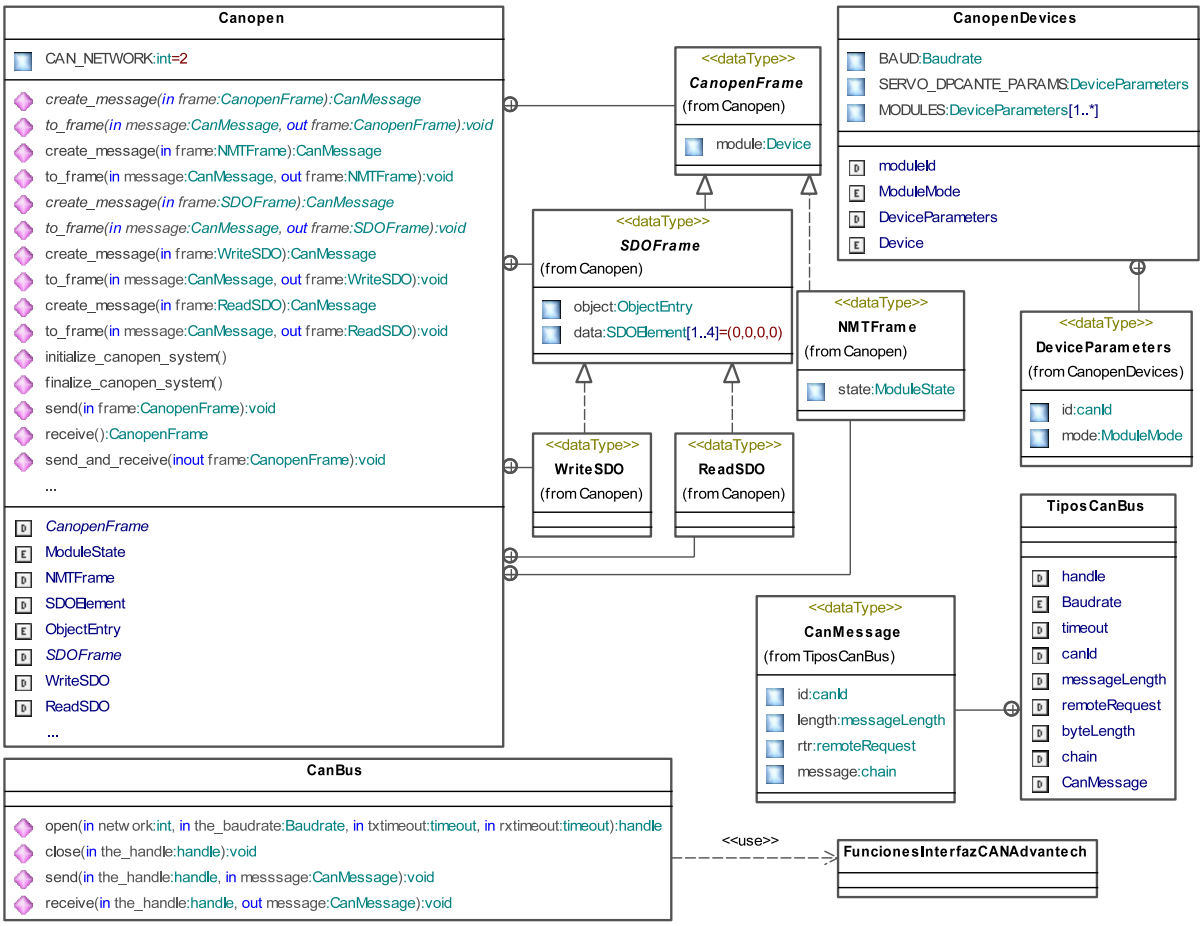

Figura 4.8: Diagrama de clases UML de la arquitectura de la librería para implementar el protocolo CANopen.

4.5.1. Implementación y lecciones aprendidas

La implementación de la librería de Pigazo López en Ada se basa en el paradigma de programación orientada a objetos aprovechando las características de *herencia* y *polimorfismo*. Al tratarse CANopen de un protocolo formado por mensajes distintos pero con campos y operaciones comunes, es posible definir objetos con las características comunes y extenderlos para particularizarlos en las diferencias.

Por su parte el polimorfismo permite invocar las mismas operaciones sobre estos objetos distintos, eligiendo en momento de ejecución la operación adecuada al objeto manejado.

Para definir los atributos o parámetros comunes a todos los mensajes existen en Ada los denominados *tipos etiquetados*. Inmediantamente a continuación del tipo etiquetado se definen las operaciones primitivas, aquellas comunes a todos los mensajes CANopen. Uno de los parámetros de estas operaciones primitivas es el propio tipo etiquetado definido anteriormente (ver Figura 4.9).

Una vez definidos tanto tipos como operaciones, se puede pasar a la fase de herencia para crear los distintos tipos de mensaje. Se debe crear en primer lugar una instancia del tipo etiquetado que bien puede contener los mismos datos que el tipo etiquetado primitivo o extenderlos con otros nuevos. Al extender el tipo etiquetado, se heredan todas las operaciones primitivas del padre y también se pueden añadir operaciones nuevas (ver Figura 4.10). Las operaciones primitivas heredadas pueden a su vez ser conservadas tal cual se definieron o redefinirlas.

```
type Canopen_Frame is abstract tagged record
   Module : Canopen_Devices.Device;
end record;

function Create_Message (Frame : in Canopen_Frame)
   return Tipos_Can_Bus.Can_Message is abstract;

procedure To_Frame (
   Message : in Tipos_Can_Bus.Can_Message;
   Frame : out Canopen_Frame) is abstract;
```

Figura 4.9: Código Ada para definir el tipo etiquetado que representa a la trama CANopen y sus operaciones primitivas que permiten convertir de trama CANopen a mensaje CAN y viceversa.

```
type NMT_Frame is new Canopen_Frame with record
   State : Module_State;
end record;

function Create_Message (Frame : in NMT_Frame)
   return Tipos_Can_Bus.Can_Message;

procedure To_Frame (
   Message : in Tipos_Can_Bus.Can_Message;
   Frame : out NMT_Frame);
```

Figura 4.10: Código Ada para definir el tipo de mensaje NMT. Se extiende el tipo etiquetado `Canopen_Frame` con un campo más. Las operaciones primitivas de `CANopen_Frame` se redefinen para aceptar el tipo de trama NMT.

Tal y como se ha explicado hasta el momento, sería posible definir objetos del tipo primitivo. Sin embargo, este tipo primitivo no representa a ningún mensaje sino que únicamente reúne las características comunes de los mensajes que heredan de él. Definiéndolo como abstracto se impide crear objetos de este tipo y las operaciones quedan sin definir, obligando a los hijos a escribirlas al realizar la herencia.

4.6. Arquitectura del software de Mármol

La arquitectura del software principal está diseñada para gestionar de forma modular cada uno de los componentes involucrados en el sistema. La aplicación hace uso de los otros módulos cuya arquitectura se ha expuesto hasta este punto del capítulo, integrándolos hasta conseguir la aplicación final. Las siguientes secciones suponen un resumen del contenido del documento de diseño de la arquitectura software [EGP13a].

4.6.1. Diseño UML

La aplicación principal también ha sido diseñada empleando diagramas UML. En este caso, aparte de los diagramas de clases ha sido necesario incluir otros que aportasen información no sólo desde el punto de vista estructural sino de comportamiento.

Figura 4.11: Diagrama de clases UML de los módulos relacionados con la interfaz gráfica de usuario (GUI) en el software de Mármol. Algunos detalles como operaciones o clases privadas han sido omitidos por razones de espacio.

Debido a la envergadura de algunos de ellos, no se ha considerado oportuno incluirlos explícitamente en esta memoria y solo se abordan sus explicaciones. La versión completa se encuentra en el documento de diseño de la arquitectura software [EGP13a].

Diagramas de clases

Compartimentan la aplicación en módulos con un propósito definido que cuentan con operaciones que guardan relación entre sí. Son útiles para ordenar responsabilidades y facilitar la posterior programación, ya que ocultan detalles innecesarios para el usuario externo. Un buen diseño de clases evita la ambigüedad respecto a quién es responsable de cada operación y facilita el mantenimiento gracias al diseño modular.

Se ha propuesto un diseño de clases que divide la aplicación en tres partes:

- Por un lado las clases relacionadas con la interfaz gráfica **GUI** (Figura 4.11) que representan entidades que se ocupan de la interacción con el usuario mediante la interfaz gráfica (gestión de eventos, modificación y obtención de estado y valores de elementos gráficos, gestión de la imagen de vídeo, gestión de avisos, operaciones de actualización periódicas, etc.).

- Un segundo grupo recoge los módulos relacionados con el **servo** (Figura 4.12) que se

Figura 4.12: Diagrama de clases UML de los módulos relacionados con el servo en el software de Mármol. Algunos detalles como operaciones o clases privadas han sido omitidos por razones de espacio.

encargan de gestionar la interacción con este elemento del sistema. En la interacción con el servo, algunos elementos son consultados de forma periódica mientras que otras operaciones se realizan bajo demanda. Un aspecto muy a tener en cuenta es gestionar que la interacción con el servo no se haga de forma concurrente desde distintos puntos de la aplicación, ya que esto acarrearía graves problemas.

- Un tercer diagrama recoge los módulos de GUI y servo que exponen sus operaciones al exterior para relacionarse entre sí y con otros módulos no incluidos en los grupos anteriores (**global**) (Figura 4.13). Se encuentran en este apartado los paquetes de interacción con los ficheros de configuración y calibración, el gestor de la cámara y el control de la tarjeta DAQ.

Diagramas de comunicación

Los diagramas de comunicación han sido empleados porque muestran la interacción entre las instancias de las clases (los objetos). Permiten conocer qué operaciones solicitan unas clases a

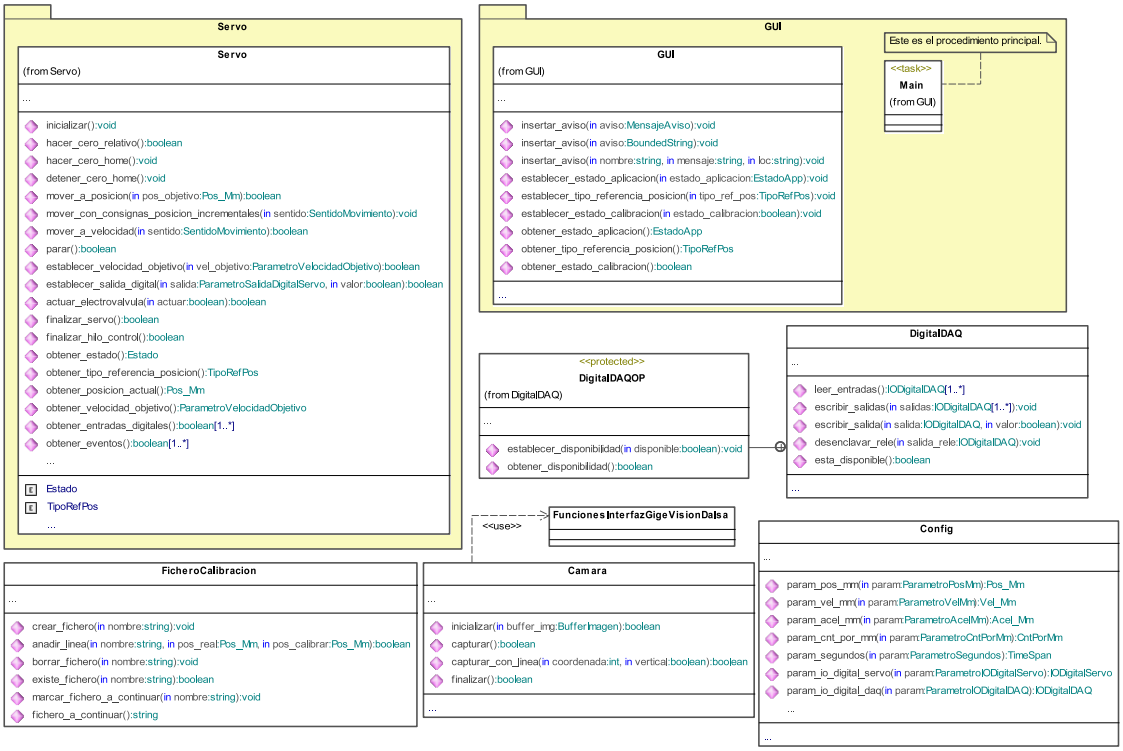

Figura 4.13: Diagrama de clases UML de los módulos de GUI y servo que exponen sus operaciones al exterior y de los módulos no incluidos en los grupos anteriores en el software de Mármol. Algunos detalles como operaciones o clases privadas han sido omitidos por razones de espacio.

otras, lo cual hace que su principal utilidad sea verificar la validez del diseño de clases realizado. Nuevamente se establece la división del diagrama de clases: GUI (Figura 4.14), servo (Figura 4.15) y resto de elementos (Figura 4.16).

Diagramas de estados

Los diagramas de estados se han empleado para modelar las máquinas de estados presentes en la aplicación y clarificar las transiciones posibles entre dichos estados. Esto permite, conociendo el estado del elemento en cuestión, saber qué ocurre en cada momento, los posibles antecedentes que han conducido a esa situación (transiciones hacia ese estado) y las posibles opciones que se despliegan (transiciones desde ese estado). Por ello, además de ser diagramas útiles en fase de diseño e implementación, resultan ser de gran importancia para comprobar el comportamiento de ciertos aspectos de la aplicación en ejecución.

Existen un total de tres máquinas de estados en el sistema. La **primera** de ellas no pertenece al dominio de la aplicación sino que describe el funcionamiento interno del servo (Figura 4.17). Sin embargo, su importancia es tal que debe recogerse en el diseño general. Las transiciones entre estados de esta máquina de estados permiten habilitar las operaciones del servo o conocer si el servo se encuentra en un estado de error, entre otras interesantes operaciones.

Una **segunda** máquina de estados, ya en el dominio de la aplicación rige sobre el estado del módulo servo desde el punto de vista de la aplicación (Figura 4.18). Los estados considerados son los siguientes:

- Sin inicializar. El servo no está inicializado.

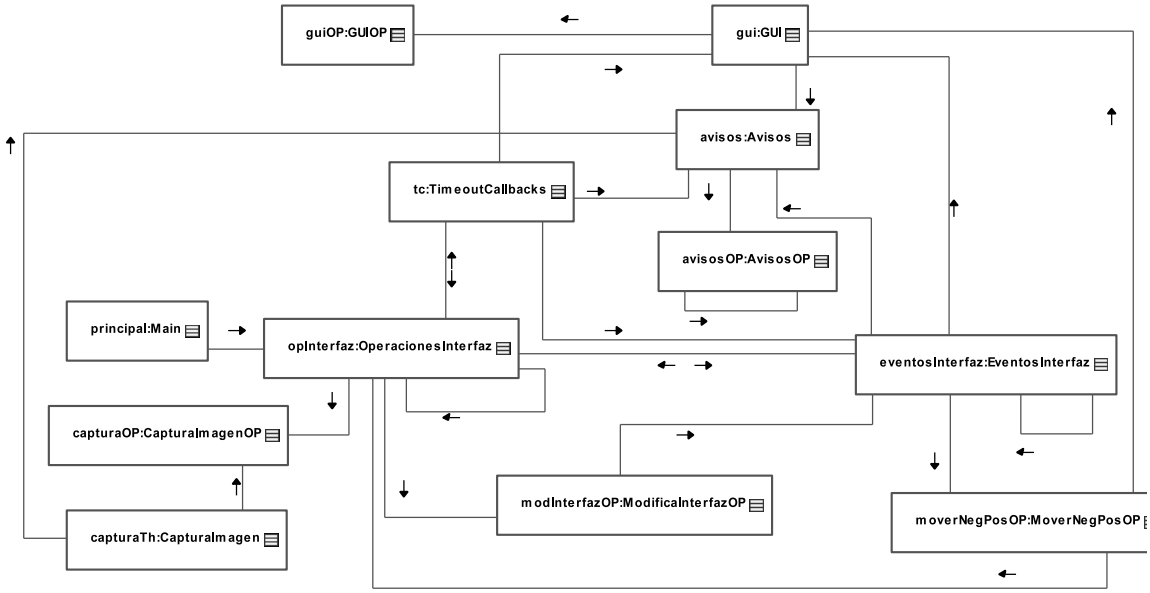

Figura 4.14: Diagrama de comunicación UML de los módulos relacionados con la interfaz gráfica de usuario (GUI) en el software de Mármol. La lista concreta de llamadas entre clases ha sido omitida por razones de espacio.

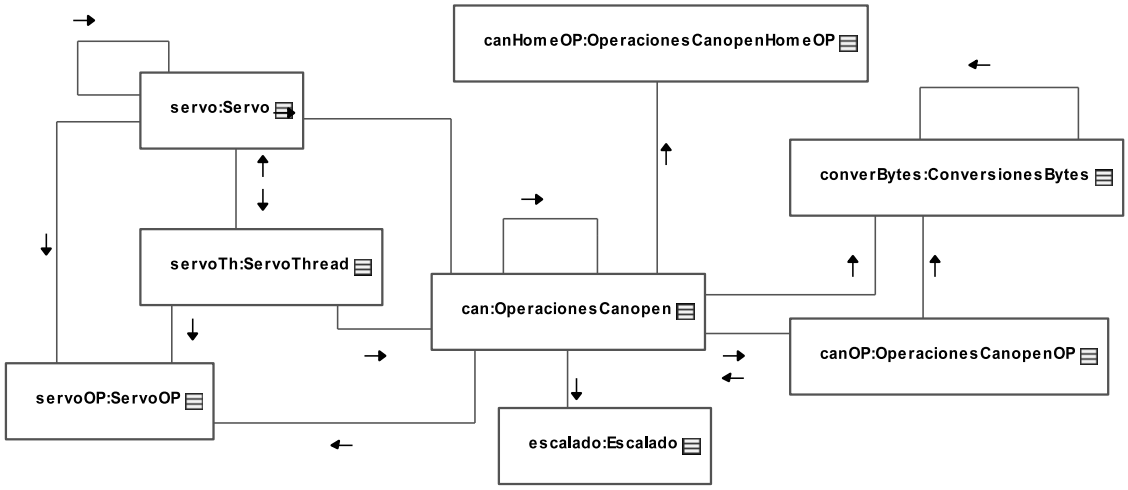

Figura 4.15: Diagrama de comunicación UML de los módulos relacionados con el servo en el software de Mármol. La lista concreta de llamadas entre clases ha sido omitida por razones de espacio.

- Inicializando. El servo está en proceso de inicialización. En la inicialización se establece la comunicación con el servo y se da valor a variables, salidas digitales, etc.
- En posición. El servo se encuentra en reposo.
- En rutina home. El servo se encuentra en fase de búsqueda de la posición de home.
- Posicionando a consigna. El servo se dirige a una consigna de posición definida.
- Posicionando a velocidad. El servo se desplaza en un sentido a la velocidad predeterminada.
- Posicionando a incrementos. El servo se encuentra en un modo de movimiento consistente en consignas de posición incrementales, esto es, se le van ordenando consignas pequeñas frecuentemente, dando la sensación de un movimiento continuo similar al posicionamiento a consigna.
- Parando posicionamiento a incrementos. El movimiento basado en el posicionamiento a incrementos está siendo detenido.
- Cabezal liberado. El cabezal del motor ha sido desacoplado del mármol, por lo que el motor no tiene tracción.
- Deshabilitado. Se ha producido un evento que ha conducido a la deshabilitación del servo.

Como se puede observar, hay cuatro estados de posicionamiento. Su justificación viene de los distintos modos de movimiento indicados en la especificación funcional y su comportamiento respecto a los finales de carrera SW (ver Cuadro 4.1).

El tipo de movimiento en un sentido se detiene al alcanzar los finales de carrera o cuando el usuario libera el control que ordenó dicho movimiento. Para este último caso, se envía un comando de parada sobre el servo. El comando de parada se comporta de dos formas atendiendo al tipo de movimiento:

- En movimientos a consigna de posición aplica una deceleración hasta que la velocidad es cero. El servo es consciente del punto en que se le ordenó parar, por lo que a continuación retrocede hasta ese punto.

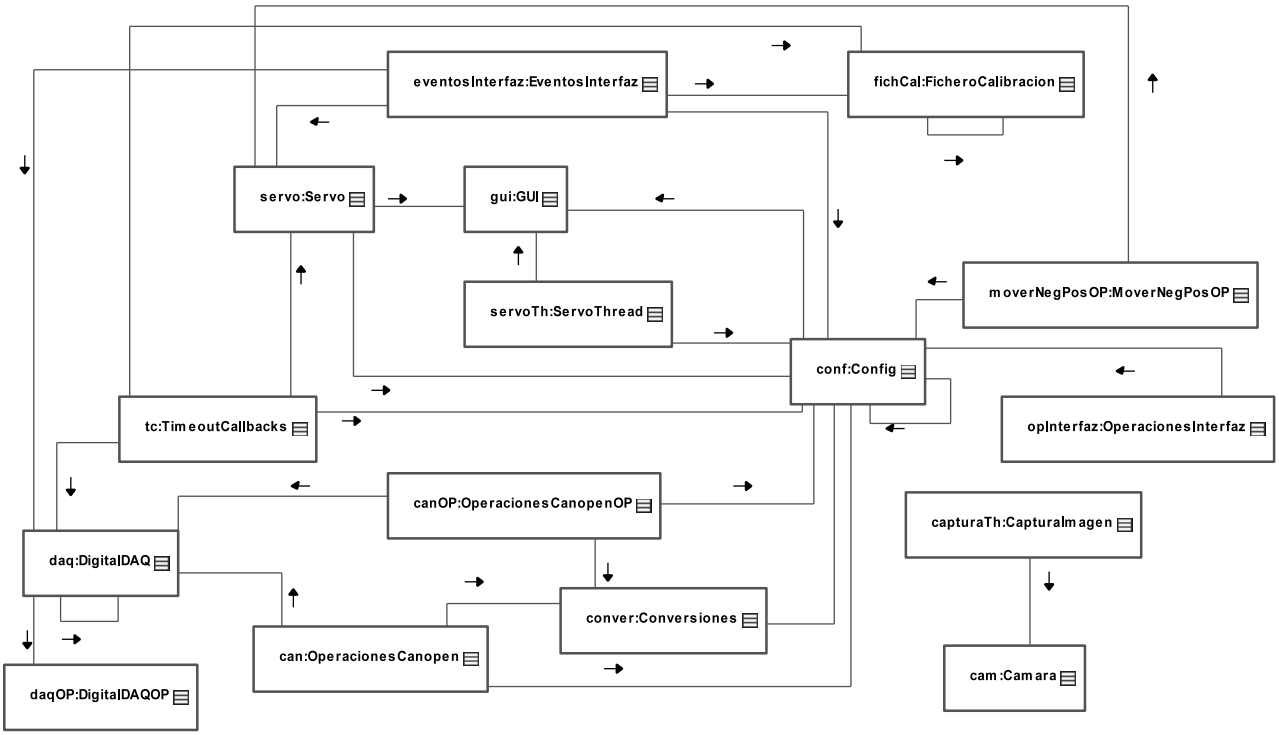

Figura 4.16: Diagrama de comunicación UML de los módulos de GUI y servo que exponen sus operaciones al exterior y de los módulos no incluidos en los grupos anteriores en el software de Mármol. La lista concreta de llamadas entre clases ha sido omitida por razones de espacio.

Cuadro 4.1: Estado de la máquina de estados del servo vigente en función del tipo de movimiento (especificación funcional) y la existencia o no de una posición de home conocida que implique conocer o no los finales de carrera SW.

Tipo de movimiento	**¿Hay home?**	**Estado del servo**
Movimiento automático a consigna de posición	Si	Posicionando a consigna
	No	
Movimiento manual en un sentido	Si	Posicionando a consigna (el final de carrera)
	No	Posicionando a velocidad

- En movimientos con consigna de velocidad, se aplica una consigna de velocidad cero que es rápidamente alcanzada.

Lamentablemente el funcionamiento de *la parada en movimientos a consigna de posición* no es del todo satisfactorio. La deceleración aplicada es muy pequeña, haciendo que la posición de parada sea superada con creces y el retorno a ella sea también largo. Este efecto se acrecienta a velocidades pequeñas (las de ajuste fino de posición), haciendo que este modo de movimiento sea poco práctico de utilizar. Por ello este mecanismo fue sustituido por otro que daba una solución al problema (ver Cuadro 4.2).

Como se puede ver, se ha sustituido el *movimiento a consigna en el final de carrera* por el de *posicionamiento a incrementos*. Este movimiento consiste en ordenar al servo incrementos de posición constantes y proporcionales al tiempo que se ha mantenido pulsado el control del movimiento desde la anterior consigna enviada. De esta forma, solo es necesario liberar el control que ordena el movimiento para que se dejen de enviar consignas de posición nuevas, lo cual a

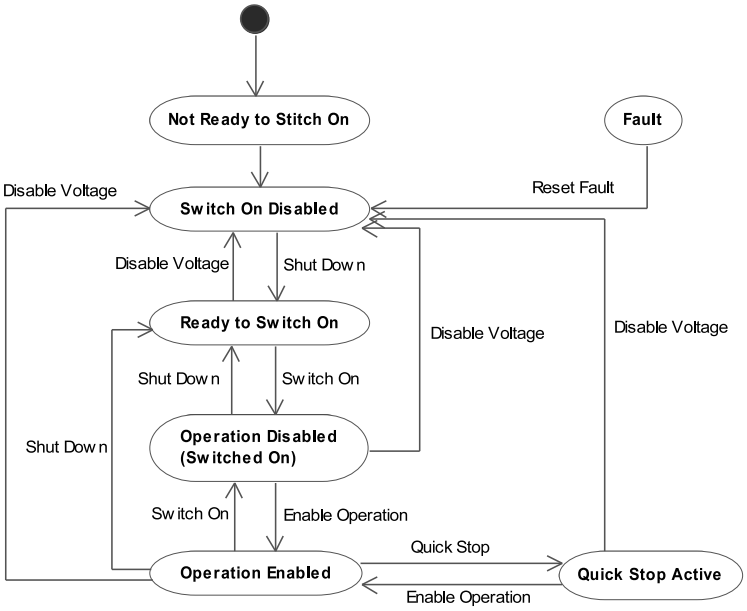

Figura 4.17: Diagrama de estados UML de la máquina que regula el funcionamiento interno del servo.

Cuadro 4.2: Estado de la máquina de estados del servo vigente en función del tipo de movimiento (especificación funcional) y la existencia o no de una posición de home conocida que implique conocer o no los finales de carrera SW. En este cuadro se puede apreciar el nuevo estado del servo denominado *Posicionando a incrementos* que soluciona el problema de la implementación anterior.

Tipo de movimiento	**¿Hay home?**	**Estado del servo**
Movimiento automático a consigna de posición	Si	Posicionando a consigna
	No	
Movimiento manual en un sentido	Si	Posicionando a incrementos
	No	Posicionando a velocidad

todos los efectos supone una parada. Se podría pensar que por qué no se utiliza para esto un movimiento a velocidad constante. La respuesta es que trabajando con consignas de posición se puede controlar que el servo no sobrepase los finales de carrera SW, algo necesario según la especificación funcional para este caso en que se conoce el contacto de home.

Aún queda pendiente un aspecto más: la existencia del estado *Parando posicionamiento a incrementos*. El motivo de su existencia radica en el diseño del posicionamiento a incrementos.

Este tipo de desplazamiento se basa en la utilización de botones cuya pulsación puede tener dos usos (ver Figura 4.19):

- Uno de ellos consiste en dejar el botón pulsado un intervalo de tiempo para hacer un desplazamiento largo.
- El otro uso consiste en la pulsación rápida para acometer un desplazamiento mínimo.

Para distinguir entre ambos movimientos se establece un intervalo de tiempo bajo el cual el desplazamiento a realizar es *mínimo* y sobre el cual el desplazamiento es *proporcional al tiempo pulsado*. No se utiliza esta segunda forma para ambos casos porque se quiere que ese

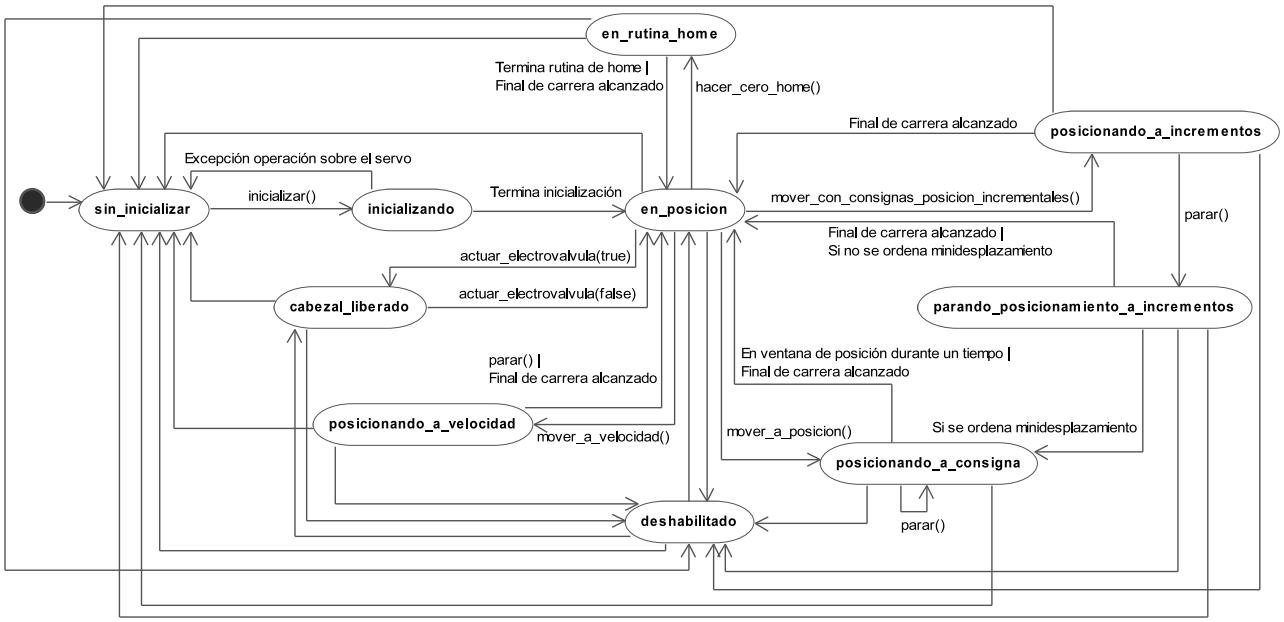

Figura 4.18: Diagrama de estados UML del módulo servo. Por motivos de espacio, se ha omitido el texto en las transiciones a `sin_inicializar`, a/desde `deshabilitado` y algunas a/desde `cabezal_liberado`. A modo de resumen, las transiciones a `sin_inicializar` se producen cuando se invoca la finalización del servo o se ha producido un error/excepción. El paso al estado `deshabilitado` se produce cuando el servo activa los eventos `user_disable` o `motor_overspeed`. Una vez se desactivan estos eventos, se vuelve a un estado estable (sin movimiento ni operaciones pendientes).

desplazamiento mínimo sea siempre proporcional solo a la velocidad y no a la duración de la pulsación. Es por ello que el estado *Parando posicionamiento a incrementos* existe para comprobar si al liberar el control que inició el movimiento, el tiempo entre pulsación y liberación implica desplazamiento mínimo o no.

La **tercera** máquina de estados de la aplicación trata del estado de la aplicación en su conjunto (Figura 4.20). Está orientada a su utilización con la interfaz gráfica, cambiando el estado de los controles de la interfaz y por tanto permitiendo o no ciertas interacciones por parte del usuario. Es una máquina de estados muy similar a la del servo, pero obviando la diferenciación de tipos de movimiento y teniendo en cuenta también el estado de la tarjeta DAQ.

Diagramas de actividad

Los diagramas de actividad permiten representar de forma gráfica algoritmos. Debido a su importancia y relativa complejidad, se ha utilizado un diagrama de actividad para describir el algoritmo del hilo de control que interacciona con el servo. En la siguiente Sección 4.6.2 se explica más sobre este hilo de control. El diagrama no ha sido incluido por su gran tamaño. Puede encontrarse en el documento de diseño arquitectónico [EGP13a].

4.6.2. Clases activas

Las clases activas son aquellas cuyos objetos tienen uno o más hilos de ejecución y por tanto pueden dar lugar a actividades de control. La introducción de clases activas implica concurrencia,

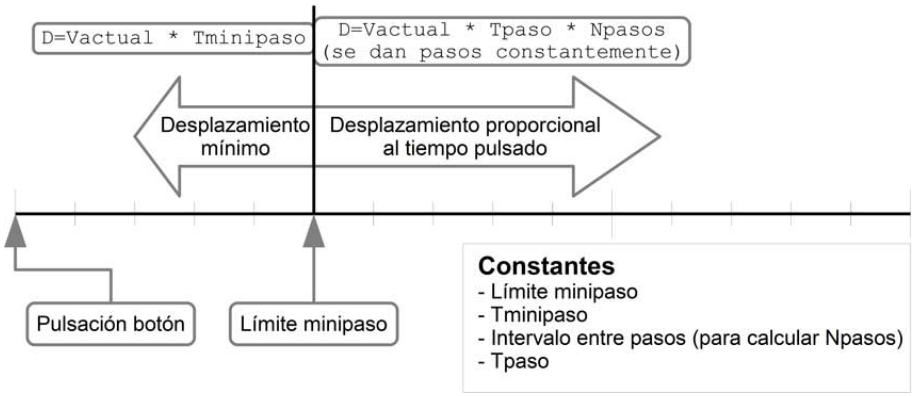

Figura 4.19: Esquema de funcionamiento del movimiento por *posicionamiento a incrementos*. En este movimiento, una vez se pulsa el botón todo depende del momento en que éste sea liberado. Si se libera antes del *Límite minipaso*, el desplazamiento es mínimo mientras que si se libera después de ese instante, el desplazamiento es proporcional al tiempo que se mantiene pulsado el botón. En esta segunda modalidad se van mandando consignas de posición de forma constante dando pasos incrementales.

por lo que es necesario cuidar el acceso exclusivo a aquellas operaciones que así lo requieran.

El software se ha planteado para contar con 3 clases activas que corren cada una un hilo de ejecución.

Hilo `CapturaImagen`

Se encarga de inicializar la cámara y realizar la captura periódica de la imagen enviada por el dispositivo.

Hilo `ServoThread`

Se ocupa de dar soporte a las operaciones que se ejecutan sobre el servo. Los motivos de la existencia de este hilo radican por un lado en que algunas de las órdenes bien tienen una duración considerable o bien requieren de un control que se extiende en el tiempo y es necesario dejarlas a un hilo distinto al principal para evitar bloqueos.

Por otro lado, algunas de las operaciones sobre el servo deben ejecutarse cada poco tiempo, por lo que este hilo puede encargarse de esta ejecución en cada iteración de su bucle interno. Estas operaciones suelen ocuparse de consultar el estado de distintos aspectos del servo (ej: la posición que ocupa). En el caso de obtener las entradas digitales del servo y los eventos activos, esta consulta además de ser informativa puede desencadenar consecuencias. Ejemplo es la detección del evento de final de carrera, cuya activación provoca el enclavamiento del relé de final de carrera que es necesario desenclavar cuando se considera que el servo ha salido del final de carrera.

Por último hubo que contemplar la posibilidad de que el usuario quisiera ejecutar una operación sobre el servo (ej: mover a una consigna de posición) mientras el hilo del servo está ocupado con el servo. Esta situación provoca una concurrencia en el acceso a la interfaz de comunicaciones con el servo a todas luces inadmisible que obligó a implementar un *mecanismo de ejecución de operaciones pendientes*. Este mecanismo funciona de tal forma que si una operación detecta que el servo está ocupado, se anota como pendiente y es el hilo `ServoThread` el que en cada iteración repasa las operaciones que quedaron pendientes y las ejecuta.

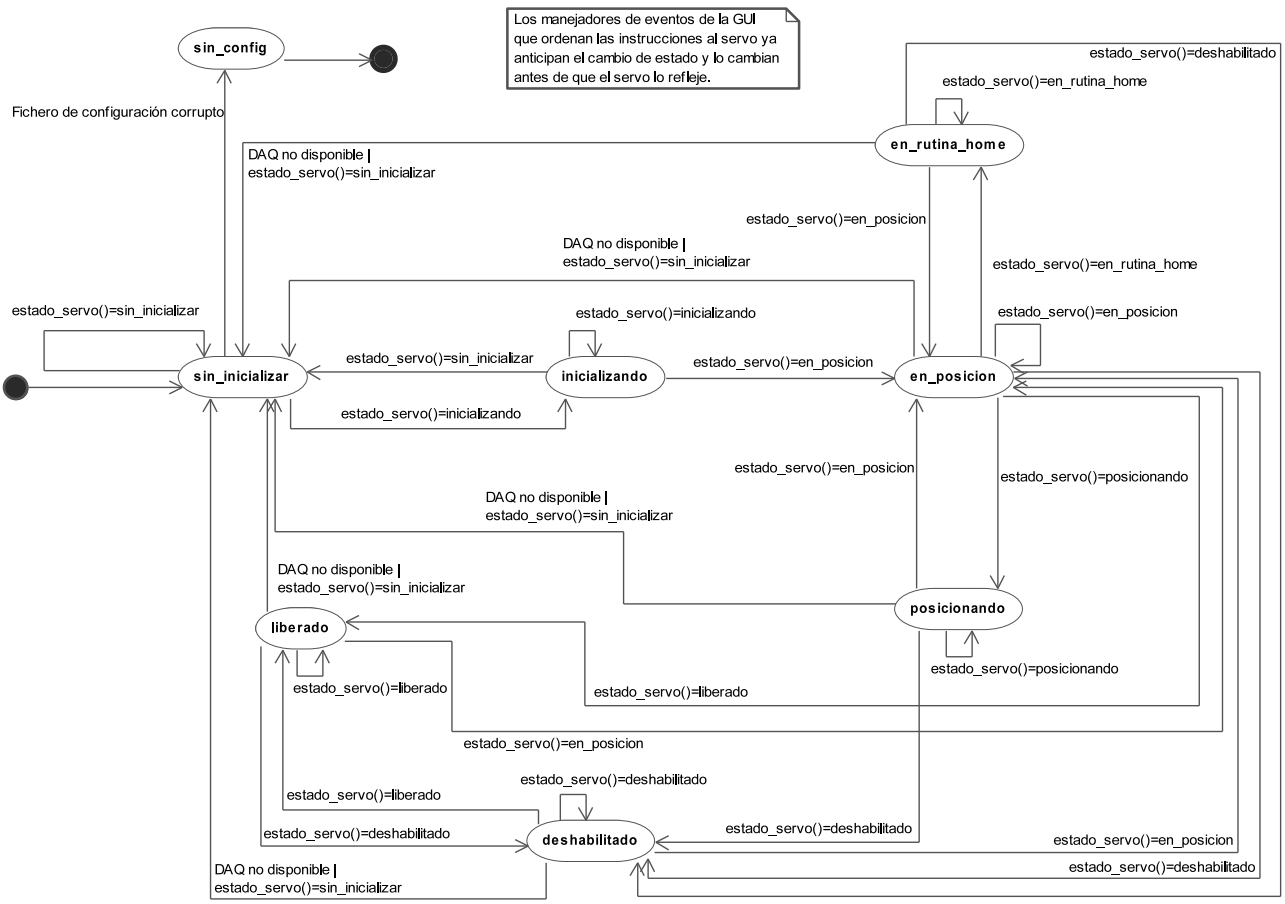

Figura 4.20: Diagrama de estados UML de la aplicación Mármol.

Hilo `Main`

Hilo principal de la aplicación, primero en activarse y último en terminar. Se encarga de inicializar la interfaz gráfica y de lanzar el bucle Main de GktAda que gestiona los eventos de la interfaz gráfica. Este bucle permite la posibilidad de introducir en él llamadas periódicas. Este mecanismo puede ser aprovechado para actualizar aspectos relacionados con la interfaz de forma periódica. En concreto ha sido aprovechado para las siguientes tareas:

- Actualizar la posición actual.
- Comprobar el estado del servo y la tarjeta DAQ.
- Comprobar si se han producido avisos.
- Comprobar las entradas digitales del servo y la tarjeta DAQ.
- Comprobar los eventos del servo.
- Refrescar la imagen procedente de la cámara.
- Refrescar los pilotos que parpadean.

4.6.3. Secciones protegidas

La existencia de varios hilos de ejecución exige una forma de proceder especial para aquellas funciones cuyo acceso no puede hacerse de forma concurrente para evitar errores o incoherencias en los datos. Además, los hilos suelen necesitar comunicarse datos entre ellos por lo que se deben programar zonas de memoria compartidas cuyo acceso esté regulado para evitar nuevamente incoherencias en los datos escritos/leídos.

Estos requisitos son cubiertos mediante los *objetos protegidos*. A continuación se exponen los objetos protegidos planteados en la aplicación.

Objeto protegido `Servo_OP`

Gestiona el acceso concurrente a las variables que permiten controlar y conocer el estado del servo y gestionar su funcionamiento. Se utiliza para compartir información entre `Servo_Thread` y otros paquetes.

Objeto protegido `Operaciones_Canopen_OP`

Este objeto protegido no está orientado a la compartición de datos sino a impedir el acceso concurrente de los hilos de ejecución a sus procedimientos. Estos procedimientos se ocupan de llevar a cabo operaciones sobre el servo que emplean la interfaz de comunicaciones CAN. Al utilizar el objeto protegido se evita la utilización concurrente de la interfaz de comunicación, lo cual conduciría a fallos.

Objeto protegido `Operaciones_Canopen_Home_OP`

El procedimiento que realiza el home puede estar ejecutándose durante varios segundos, ya que se mantiene activo durante todo el proceso incluyendo la búsqueda de home por el patín del sistema. Los valores de eventos y entradas digitales del servo son consultados por las tareas periódicas que ejecutan en el hilo `Main`. Sin embargo, `Servo_Thread`, el encargado de consultar estos valores en el servo en cada iteración de su bucle interno para dejarlos en la zona protegida de `Servo_OP` se encuentra ejecutando el procedimiento de home, por lo que estos valores están desactualizados. Para evitarlo, el procedimiento de home escribe constantemente el valor que consulta de eventos y entradas digitales en el objeto protegido `Operaciones_Canopen_Home_OP`. Así, para conocer estos valores, el procedimiento encargado consulta si el servo está en el procedimiento de home para acceder a este objeto protegido en vez de a `Servo_OP` (que tiene un valor antiguo).

Otra de las funciones de este objeto protegido es contar con la variable que permite finalizar con el procedimiento de home en caso de que el usuario desee abortarlo.

Objeto protegido `GUI_OP`

Objeto protegido que gestiona el acceso concurrente a las variables donde se almacena el estado de la aplicación.

Objeto protegido `Modifica_Intefaz_OP`

Objeto protegido que impide modificar de forma concurrente el estado y aspecto de los elementos de la interfaz.

Objeto protegido `Captura_Imagen_OP`

Gestiona el arranque y finalización de la tarea que se ocupa de capturar la imagen procedente de la cámara.

Objeto protegido `Mover_Neg_Pos_OP`

Los controles de movimiento en sentido negativo o positivo permiten mientras se mantienen pulsados desplazar el patín en un sentido u otro. Por cada sentido existen tres controles que ponen en marcha dicha acción. Éstos son el botón de la botonera, una tecla del teclado y el botón de la interfaz gráfica. Este objeto protegido se encarga de llevar a cabo el movimiento impidiendo que si un control ha iniciado un movimiento, la pulsación o liberación de otro control tenga algún efecto.

Objeto protegido `Avisos_OP`

Objeto protegido cuya función es gestionar la inserción y consulta de avisos en una cola de avisos.

Objeto protegido `Digital_DAQ_OP`

La disponibilidad de la tarjeta DAQ se almacena en este objeto protegido.

4.6.4. Implementación y lecciones aprendidas

Al tratarse Ada de un lenguaje orientado a objetos, el diseño realizado puede trasladarse al código de forma casi directa.

Cada clase del modelo orientado a objetos crea en Ada un paquete (`package`). Los paquetes se crean en base a una especificación y un cuerpo (`body`) con los detalles internos y la implementación.

Los atributos/campos del diseño son variables globales Ada y las operaciones/métodos pasan a ser procedimientos (`procedure`) y funciones (`function`) Ada. El acceso tanto a atributos como operaciones puede restringirse a un acceso interno de la clase (privados). Esto se consigue mediante las partes visible y privada de los paquetes Ada.

Los datos en Ada siempre tienen un tipo asociado. Ese tipo puede ser *predefinido* (`Integer`, `Float`, `Character`, `String`, `Boolean`, `Duration`, etc.) o *definido por el usuario*. La definición de tipos de datos se realiza a partir de otros tipos de datos y permite ajustar el tipo de dato a los valores que en él se van a manejar, restringiendo rangos o definiendo mediante ellos nuevas propiedades.

Las tareas expuestas en la Sección 4.6.2 tienen traducción directa a Ada mediante las `task`. El cuerpo de las tareas cuenta con un bucle que ejecuta continuamente el trabajo asignado. Mediante las instrucciones `delay` y `delay until` es posible suspender la ejecución del hilo durante

un tiempo o hasta un instante de tiempo determinado respectivamente. Cambiar la condición de permanencia en el bucle permite salir de él. Una vez alcanzado el final del hilo, este se destruye. Es importante controlar que todos los hilos se destruyan al salir de la aplicación para evitar que el proceso que inició la aplicación deba ser abortado.

El mecanismo de ejecución de operaciones pendientes citado para el hilo `ServoThread` se implementa mediante un punto de entrada `entry` o *rendezvous*. Esto permite sincronizar tareas de tal forma que hay una tarea que llama y otra que acepta el encuentro. La tarea que llama se queda esperando hasta que la tarea que acepta la llamada termina de ejecutarla.

Otro de los aspectos fundamentales en la aplicación es el uso de objetos protegidos (`protected`). Los objetos protegidos tienen una parte visible que consta de funciones, procedimientos y puntos de entrada y una parte privada que contiene los datos protegidos. Las operaciones de un objeto protegido se ejecutan en exclusión mutua respecto a sí mismas y a las demás operaciones protegidas. Esta característica puede ser explotada no sólo para compartir datos sino para agrupar funciones y procedimientos que no pueden ser ejecutados de forma concurrente.

Los objetos protegidos, al igual que las tareas, cuentan con puntos de entrada (`entry`). Los puntos de entrada proporcionan la misma protección que los procedimientos protegidos pero además permiten a la tarea que lo invoca esperar hasta que se cumpla una determinada condición.

El punto de entrada ha sido aprovechado en el objeto protegido `Operaciones_Canopen_OP`. Estos puntos de entrada actúan sobre variables booleanas que indican si la conexión con el servo existe, si el servo está ocupado o si existe una referencia de posición válida. De esta forma, si se desea invocar una operación del objeto protegido que exija conexión con el servo, esta operación quedará bloqueada a la espera de que la condición se cumpla.

Una de las limitaciones con que cuentan los procedimientos y funciones Ada incluidos en los objetos protegidos es la imposibilidad de ejecutar *esperas temporizadas*, pues se considera un mecanismo bloqueante. Debido a que algunas de las operaciones a ejecutar sobre el servo deben incluir esperas de este tipo, no es posible incluirlas en este objeto protegido. En su lugar, se optó por utilizar una variable de barrera para todas las operaciones que se comuniquen con el servo denominada `Servo_Ocupado`. Al ejecutar una operación sobre el servo desde fuera del objeto protegido (porque incluye esperas temporizadas), esta variable es modificada, haciendo que si se busca ejecutar una operación de acceso al servo en el objeto protegido, ésta quede bloqueada. El acceso para modificación y lectura de esta variable se realiza mediante sendos procedimientos en el objeto protegido.

Por último citar el uso de Gtk+ para Ada (GtkAda). El lenguaje Ada comparado con otros más populares como Java o C dispone de una cantidad mucho menor de documentación y recursos siempre útiles como foros o tutoriales donde otras personas exponen sus soluciones a problemas de programación. El caso de GtkAda no ha sido una excepción, pues se trata de una librería cuya adaptación a Ada, al menos por información en la web, es escasa en recursos. Sin embargo el tratarse de una librería multilenguaje hace que los problemas que surgen en la adaptación a un lenguaje puedan ser los mismos que en otro lenguaje, y también sus soluciones. De esta forma, aunque apenas hay información para Ada, es posible buscarla para C o Phyton y encontrar respuestas que es fácil trasladar a Ada.

Otro comentario respecto a Gtk+ centrado en el modo de construcción de la interfaz es que al igual que ocurre en otro tipo de lenguajes y sistemas, por ejemplo las plataformas móviles Android o iOS, la interfaz puede ser generada bien programando el código o bien mediante editores gráficos. Esta última opción es sin duda la más interesante por la sencillez y control que aporta sobre el diseño. Estos editores gráficos pueden crear el código del programa directamente o en su defecto un fichero intermedio con la definición de la interfaz. GtkAda admite esta opción

de diseño de la interfaz usando la herramienta Glade, que genera un fichero con extensión `.xml` que, al ser pasado como parámetro de una función de GtkAda permite que con una sola línea de código la interfaz sea incorporada al programa.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El objetivo principal del proyecto se ha visto satisfecho:

- El Laboratorio de Metrología de ENSA dispone de un software de calibración para la medidora de coordenadas horizontales útil, que cumple sus expectativas y del que pueden y hacen uso.

Además, este software ha sido desarrollado utilizando las tecnologías concretas que se plantearon con el objetivo de investigar y evaluar su funcionalidad. Estas eran:

- Utilización del protocolo CANopen sobre interfaz de comunicación CAN para la monitorización y control del sistema de calibración mediante un servo digital.
- Creación de una interfaz gráfica basada en GtkAda que permitiera monitorizar y controlar el servo, además de recibir la imagen de vídeo procedente de una cámara que señala el punto de calibración actual.

En ambos casos, ha sido posible utilizar la tecnología planteada para conseguir el objetivo. Por un lado los servos digitales constituyen un avance respecto a los analógicos al pasar la responsabilidad de gestionar el lazo de control PID del controlador al servo. Ahora, en vez de ser necesarios complejos cálculos para dar valor a las distintas señales conectadas al servo, sólo es necesario configurar el servo de forma previa mediante una herramienta de configuración y hacer uso de una única interfaz de comunicación para ordenar sencillos comandos como *ir a consigna de posición*. De hecho, en el Laboratorio de Automática de Equipos Nucleares se han planteado la sustitución de los servos analógicos por digitales en varios de sus desarrollos y la utilización de servos digitales en los futuros utilizando el software contenido en este proyecto.

Por otro lado, GtkAda ha sido probado como una buena tecnología para la construcción de interfaces gráficas para lenguaje Ada. La utilización de Glade como editor de la interfaz gráfica supone un gran avance, pues simplifica la creación de interfaces gráficas y permite conseguir unos resultados profesionales.

Adicionalmente se han desarrollado los siguientes módulos software que han permitido el cumplimiento de estos objetivos y constituyen resultados independientes utilizables en otros ámbitos:

- Nuevo controlador de dispositivo en Windows NT para tarjeta de entradas y salidas digitales PCI-1761 de Advantech.
- Librerías software para el manejo de las cámaras digitales DALSA que soportan el protocolo GigE Vision y para la tarjeta de comunicaciones CAN PCI-1680U de Advantech.
- Creación de una librería para lenguaje Ada que implemente las partes necesarias en el proyecto del protocolo CANopen, pero de diseño modular que facilita su extensión en un futuro.

5.2. Trabajo futuro

Los objetivos establecidos en un principio han sido plenamente cubiertos. Sin embargo, siempre queda espacio para mejoras. Más aún al tratarse de una máquina que partiendo del sencillo principio de monitorizar la posición en un eje lineal, tiene muchas posibilidades. Estas posibilidades salen a la luz muchas veces cuando el sistema ya se está probando en el entorno final. Los usuarios ven en la máquina nuevas posibilidades o mejoras. Algunas de ellas, al ser sencillas se pueden incluir sobre la marcha y de hecho están ya incluidas (por ejemplo, los ceros relativos). Otras se dejan para posteriores versiones:

- Sistema de máximos y mínimos. Es una nueva aplicación cuyo objetivo es mejorar el sistema de calibración de barras. Al hacer la calibración de barras se aprisiona la barra entre un poste al final del mármol y otro poste situado en el patín. La posición del patín es la medida de la barra. Sin embargo, la medida obtenida puede ser equívoca si no está perfectamente perpendicular a los postes que la sostienen. Por ello, la barra se apoya en unos soportes que permiten regular los tres ejes espaciales hasta que la barra se encuentra perfectamente colocada. Esta situación puede visualizarse en el visor de posición cuando al mover la barra se alcanza un valor máximo o mínimo, que indica que la posición de perpendicularidad ha sido alcanzada y que si se sigue avanzando el plano de la barra se estará inclinando. Para visualizar este efecto, otros instrumentos similares que tienen en el Laboratorio disponen de una aguja que marca este cambio de tendencia. Desde Metrología están muy interesados en que este mecanismo esté disponible en revisiones futuras.
- Movimientos mediante un jostick. Se ha planteado la posibilidad de que el sentido del movimiento y la velocidad vengan controladas mediante un jostick.
- Línea de referencia de la imagen de vídeo. La imagen procedente de la cámara cuenta con una línea de referencia sobreimpresa que permite desplazarse en la calibración de reglas al trazo deseado. Se ha sugerido que para facilitar este trabajo, el grosor y color de la línea de referencia sea configurable, de tal forma que el grosor coincida con el del trazo medido y el color sea fácilmente identificable, ya que debido a que las reglas tienen diferentes colores, a veces se hace difícil de distinguir.
- Sistema de doble coordenada. En Metrología creen interesante que por cada coordenada X del eje lineal, se mida otra coordenada Y procedente de alguna otra fuente como por ejemplo un láser. Esto facilitaría la calibración de piezas como tubos con distintos diámetros en su recorrido.
- Programador de movimientos automáticos. Consistiría en automatizar más aún el proceso de calibración. Para ello, en un fichero de texto se definen una serie de posiciones. Al hacer cero, se va a la primera posición de forma automática. Una vez en esa posición, se pulsa adquirir y a continuación el patín se mueve automáticamente al siguiente punto dado por

el fichero. Este proceso continuaría hasta que se adquiriese el último punto de calibración. Mediante este funcionamiento, se evita tener que introducir a mano una nueva consigna tras cada adquisición. Además, se pueden crear ficheros para todo tipo de calibraciones que pueden ser reutilizados posteriormente.

Bibliografía

[Ada13] AdaCore. GtkAda: a complete Ada graphical toolkit. `http://libre.adacore.com/tools/gtkada/`, August 2013.

[AIA10] AIA Automated Imaging Association. *GigE Vision Specification version 1.2.* AIA Automated Imaging Association, 900 Victors Way, Suite 140, Ann Arbor, MI 48108, 2010.

[AMC13a] AMC Advanced Motion Controls. CANopen Communication Reference Manual. `http://www.a-m-c.com/download/manual/AMC_CANopenCommunicationManual.pdf`, August 2013.

[AMC13b] AMC Advanced Motion Controls. DriveWare 7 User Guide. `http://www.a-m-c.com/download/manual/AMC_DriveWareSoftwareManual.pdf`, August 2013.

[AMC13c] AMC Advanced Motion Controls. Network Options for Digiflex Servo Drives. `www.a-m-c.com/download/support/an-001.pdf`, August 2013.

[BL01] Art Baker and Jerry Lozano. *The Windows 2000 Device Driver Book: A Guide For Programmers.* Prentice Hall, Inc., Upper Saddle River, New Jersey, 2nd edition, 2001.

[CIA13] CIA Can in Automation. CANopen Protocol. `http://www.can-cia.org/index.php?id=systemdesign-canopen`, August 2013.

[EGP13a] ENSA and Pablo Gutiérrez Peón. Diseño de la arquitectura software del sistema de control de posición lineal para medidora de coordenadas horizontales (Mármol) versión 1.0, August 2013.

[EGP13b] ENSA and Pablo Gutiérrez Peón. Especificación funcional del sistema de control de posición lineal para medidora de coordenadas horizontales (Mármol) versión 1.0, August 2013.

[Fut13] Futaba Corporation. Digital FET Servos. `http://www.futaba-rc.com/servos/digitalservos.pdf`, August 2013.

[GH13] Michael González Harbour. Fundamentos de Computadores y Lenguajes (Introducción al análisis y diseño de programas. Licenciado en Física. Universidad de Cantabria). `http://www.ctr.unican.es/asignaturas/fundamentos/cap3-2en1.pdf`, August 2013.

[GP13] Pablo Gutiérrez Peón. Programación de controladores de dispositivos en Windows NT, October 2013.

[GPAR11] Pablo Gutiérrez Peón and Mario Aldea Rivas. Desarrollo de un controlador para cámaras digitales basadas en el estándar GigE Vision. Master's thesis, Universidad de Cantabria, 2011.

[Net13] Net Applications. Desktop Operating System Market Share. http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=0, August 2013.

[PL00] Alberto Pigazo López. Plataforma distribuida basada en bus CAN para el desarrollo de aplicaciones en Ada 95. Master's thesis, Universidad de Cantabria, 2000.

[Ren13a] Renishaw plc. RSLM high accuracy stainless steel scale. Data sheet: L-9517-9305-05-B. http://resources.renishaw.com/download.aspx?data=25409, August 2013.

[Ren13b] Renishaw plc. TONIC DOP (dual output) encoder system. Data sheet: L-9517-9411-02-E. http://resources.renishaw.com/download.aspx?data=34654, August 2013.

[RG13] Francisco Ruiz and Michael González. Procesos de Ingeniería del Software (Ingeniería del Software I. Ingeniero en Informática. Universidad de Cantabria). http://www.ctr.unican.es/asignaturas/is1/is1-t02-trans.pdf, August 2013.

[SHMS02] Lorenzo Sevilla Hurtado and María Jesús Martín Sánchez. *Metrología dimensional.* Universidad de Málaga, Málaga, 2002.

[Sta09] William Stallings. *Operating systems: internals and design principles.* Pearson Education, cop., Upper Saddle River, New Jersey, 6th edition, 2009.

[UPV13] UPV Universidad Politécnica de Valencia. Periféricos en Interfaces Industriales. Comunicaciones con CAN. https://poliformat.upv.es/access/content/group/OCW_1432_2008/Tema%206.%20Comunicaciones%20con%20el%20bus%20CAN%20_Controller%20Area%20Network_/Tema%207%20OCW%20-%20Comunicaciones%20con%20CAN-1.pdf, August 2013.

[Win13a] Windows Dev Center Hardware. Digital Signatures. http://msdn.microsoft.com/en-us/library/windows/hardware/ff543743%28v=vs.85%29.aspx, August 2013.

[Win13b] Windows Sysinternals. DebugView v4.81. http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx, August 2013.

Glosario

AMC. *Advanced Motion Controls*. Empresa dedicada al diseño y fabricación de servos.

API. *Application Programming Interface*. Interfaz de Programación de Aplicaciones ofrecida por un componente software para su uso por parte de un tercero.

CAN. *Controller Area Network*. Protocolo de comunicaciones basado en una topología bus para transmisión de mensajes en entornos distribuidos.

CANopen. Protocolo de comunicaciones de alto nivel para uso industrial basado en el bus CAN.

Coeficiente de dilatación. Cociente que mide el cambio relativo de longitud o volumen que se produce cuando un cuerpo sólido o un fluido dentro de un recipiente experimenta un cambio de temperatura que lleva consigo una dilatación térmica.

CSMA/CD. *Carrier Sense Multiple Access with Collision Detection*. Acceso Múltiple con Escucha de Portadora y Detección de Colisiones es un protocolo de acceso a un medio compartido de comunicaciones. Su uso está especialmente extendido en redes Ethernet. En CSMA/CD, los dispositivos de red escuchan el medio antes de transmitir para determinar si el canal y sus recursos están disponibles para realizar una transmisión. Si se detecta una colisión, se finaliza el envío.

CTR. *Computadores y Tiempo Real*. Grupo de investigación de la Universidad de Cantabria donde se ha desarrollado este proyecto. Sus líneas de investigación están centradas en los sistemas de tiempo real. En sus colaboraciones con ENSA desde 1987 ha dedicado parte de sus esfuerzos al desarrollo de sistemas robóticos.

DAQ. *Data AcQuisition*. Adquisición de Datos. Consiste en la toma de muestras del mundo real (analógicas o digitales) para generar datos que puedan ser manipulados por un sistema electrónico (sistema digital).

ENSA. *Equipos Nucleares S.A.*. Empresa de componentes nucleares para la cual ha sido realizado este proyecto.

GNOME. Entorno de escritorio e infraestructura de desarrollo para sistemas operativos GNU/Linux y Unix.

Gtk+. Conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario (GUI), principalmente para los entornos gráficos GNOME y XFCE, aunque también se puede usar en el escritorio de Windows y MAC OS entre otros.

GtkAda. Adaptación de las bibliotecas Gtk+ a lenguaje Ada.

GUI. *Graphical User Interface*. Interfaz Gráfica de Usuario.

Jumper. Elemento que permite interconectar dos terminales de manera no definitiva sin tener que efectuar una operación que requiera una herramienta adicional (ej: soldadura). Dicha unión de terminales cierra el circuito eléctrico del que forma parte.

OSI. *Open System Interconnection*. Modelo de interconexión de sistemas abiertos, es la propuesta de la Organización Interfacional para la Estandarización (OSI) para estandarizar la interconexión de sistemas de comunicaciones, estableciendo un marco de referencia para ellas.

PCI. *Peripheral Component Interconnect*. Interconexión de Componentes Periféricos. Bus de computador estándar para conectar dispositivos periféricos directamente a la placa base.

PID. *Proporcional Integral Derivativo*. Mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener, para aplicar una acción correctora que ajuste el proceso.

Registro de Windows. Base de datos del hardware y software instalado en un sistema operativo Windows.

Relé. Dispositivo electromecánico que funciona como un interruptor pero activado por un circuito eléctrico.

Servomecanismo. Sistema formado por partes mecánicas y electrónicas que suele emplearse en robots como parte móvil o fija.

SO. *Sistema Operativo*.

Sistema de tiempo real. Combinación de computadoras, dispositivos hardware de E/S y software en el cual hay una fuerte interacción con un entorno cambiante con el tiempo y en el cual el sistema debe de forma simultánea controlar y reaccionar a distintos aspectos de dicho entorno, cumpliendo los requisitos temporales establecidos.

Tiempo real estricto. Sistema de tiempo real en el que el incumplimiento de los plazos resulta en un fallo del sistema.

Tiempo real no estricto. Sistema de tiempo real en el que el incumplimiento de los plazos resulta en una degradación en el rendimiento del sistema, pero no en un fallo del mismo.

Trazabilidad. Conjunto de aquellos procedimientos preestablecidos y autosuficientes que permiten conocer el histórico, la ubicación y la trayectoria de un objeto o producto a lo largo de la cadena de suministros o manipulaciones en un momento dado.

UML. *Unified Modeling Language*. Lenguaje Unificado de Modelado. Lenguaje de modelado de sistemas software basado en una representación gráfica que permite visualizar, especificar, construir y documentar un sistema.

XML. *eXtensible Markup Languaje*. Lenguaje extensible basado en etiquetas, diseñado para transmitir información de manera fácilmente procesable tanto por humanos como por máquinas.