ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

SISTEMA DE ADQUISICIÓN Y PROCESADO DE IMÁGENES DE SOLDADURAS TOMADAS MEDIANTE UN ROBOT IRB2400

(Acquisition and processing system for weld images taken by an IRB2400 robot)

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: Carlos Fernando Velilla Zancada

Septiembre – 2011

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Carlos Fernando Velilla Zancada
Director del PFC: Michael González Harbour

Título: "Sistema de adquisición y procesado de imágenes de soldaduras tomadas mediante un robot IRB2400"
Title: "Acquisition and processing system for weld images taken by an IRB2400 robot"

Presentado a examen el día: 30 de Septiembre de 2011

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Pérez Vega, Constantino
Secretario (Apellidos, Nombre): Drake Moyano, José María
Vocal (Apellidos, Nombre): Cobo García, Adolfo

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

ProyectoFin de Carrera Nº
(a asignar por Secretaría)

Gracias a todos, a los que estáis y a los que no, a los que me habéis ayudado a lo largo de este camino tan largo que he recorrido hasta llegar aquí. Sin todos y cada uno de vosotros esto no sería posible. Gracias por escucharme y ayudarme en mis momentos malos, por celebrar y alegraos por mi en los buenos, por tener un hombro sobre el que llorar y una sonrisa a la que mirar.

Gracias a mis padres, a mi hermana y a Ani, que siempre habéis estado a mi lado, con esa comprensión, cariño y paciencia que solo una familia sabe dar, mostrándome todo vuestro apoyo para que no me quedase en el camino y haciendo que, en cada caída, hubiese una mano con la que levantarme. Gracias a mis tíos y familiares, especialmente a mi tío Miguel, que siempre me has dado fuerza para seguir adelante, no me olvido de ese viaje que tenemos pendiente y que tanto ansío en hacer a tu lado.

Qué decir de vosotros, amigos, os quiero un montón, gracias por los buenos momentos que hemos pasado juntos en estos últimos años y por los que aún quedan por venir. A Laura, compañera pero sobre todo amiga, a Juan, Rober, Diego, Toño, Lucía, Paloma, Rubén, Borja y Clara. También a ti Ana, por tu apoyo incondicional de estos últimos meses. Sin todos vosotros se me habría hecho muy dura la carrera.

Todo esto no habría salido adelante sin Equipos Nucleares S.A. y sin Michael. Agradeceos la oportunidad que me habéis brindado al haber confiado y apostado por mí para desarrollar este proyecto. Ha sido un logro tanto personal como profesional.

Gracias a Raúl Beci y a Fidel Gómez, que me habéis tanto ayudado, como enseñado a relacionar los conocimientos adquiridos a los largo de la carrera con el mundo del automóvil que tanto me apasiona.

A mis compañeros de trabajo de Santander Teleport, gracias. Gracias por el apoyo y la ayuda que me habéis dado a lo largo de estos meses, por todas las facilidades otorgadas para compaginar la realización de este proyecto, que cierra mi etapa de estudiante, y abre mi inicio en el mundo laboral.

Hay una persona más que me gustaría nombrar: Juan Fernando, mi director del colegio. Siempre te estaré agradecido, fuiste la única persona que, cuando nadie apostaba por mí, tu veías en aquel chico travieso un ingeniero en potencia, y al final tenías razón, solo era un chico travieso.

Gracias a toda persona que de una forma u otra ha hecho que este día sea posible.

ÍNDICE:

CAPÍTULO 1: Introducción 1

1. Objetivos generales 2
2. Centrales nucleares 3
   2.1. Introducción 3
   2.2. Reactor nuclear 3
      2.2.1. Partes del reactor 3
      2.2.2. Tipos de reactores 4
3. Robótica industrial 5
   3.1. Introducción 5
   3.2. Importancia de la Robótica en la Industria 5
   3.3. Aplicaciones 5
4. Procesado de imagen, visión artificial 6
   4.1. Introducción 6
   4.2. Elementos de un Sistema de Visión Artificial 6
   4.3. Aplicaciones 7
5. Metodología de trabajo 7
   5.1. Requisitos del sistema de adquisición y procesado de imágenes planteados por *Equipos Nuclerares S.A* 7
      5.1.1. Descripción general 7
      5.1.2. Estructura del sistema 7
      5.1.3. Protocolo de comunicaciones 8
      5.1.4. Adquisición y procesado 9
      5.1.5. Parámetros del sistema 10
      5.1.6. Comentarios generales 10
   5.2. Objetivos secundarios 10
      5.2.1. Simulación de robot ABB 10
      5.2.2. Dispositivo VA31. Programación en Sherlock 10
      5.2.3. Soldaduras de los tubos de refrigeración 11
6. Estructura de la memoria 11

CAPÍTULO 2: Elementos y bases del sistema de adquisición y procesado de imágenes 13

1. Introducción 14
2. Robot ABB IRB2400 14
3. Cámara Jai® 15
4. Unidad de procesado VA31 17
5. Programa de procesado Sherlock 18

CAPÍTULO 3: Simulación del robot IRB2400 21

1. Introducción 22
2. Hardware 22
   2.1. Equipo 22
   2.2. Puerto serie 23
   2.3. Puerto paralelo 25

3. Software....28
3.1. Sistema Operativo....28
3.2. Puerto serie....28
3.3. Puerto paralelo....31
3.4. Protocolo de comunicaciones....32
3.5. Simulación del robot ABB....33
3.5.1. Librerías....33
3.5.2. Constantes y variables....33
3.5.3. Preliminares....34
3.5.4. Apertura de puertos e inicialización de variables....35
3.5.5. Bucle principal....35
3.5.6. Espera al sistema de adquisición....35
3.5.7. Comprobaciones iniciales....36
3.5.8. Generación del telegrama....37
3.5.9. Envío del telegrama....39
3.5.10. Espera de confirmación....39
3.5.11. Acciones finales....40

CAPÍTULO 4: Desarrollo del sistema de adquisición....42

1. Introducción....43
2. Introducción al programa Sherlock....43
3. Interconexión con el robot ABB....46
3.1. Líneas digitales....46
3.1.1. Lectura....46
3.1.2. Escritura....47
3.2. Puerto serie....49
3.3. Esquema general....51
4. Guardado de imágenes en disco....51
5. Funcionalidades del programa de adquisición....54
6. Estructura del programa de adquisición....56
7. Programa de adquisición....58
7.1. Subrutina *Genera nombre1*....58
7.2. Subrutina *Genera nombre2*....60
7.3. Subrutina *Control errores*....62
7.4. Programa principal....64
7.4.1. Inicialización....66
7.4.2. Adquisición....68

CAPÍTULO 5: Desarrollo del programa de procesado....72

1. Introducción....73
2. Procesado de imágenes....73
2.1. Introducción a las soldaduras....73
2.2. Algoritmo de procesado de imágenes....76
2.2.1. Explicación del algoritmo....76
2.2.2. Aplicación del algoritmo a soldaduras....79
3. Funcionalidades del programa de procesado....85
4. Estructura del programa de procesado....87
5. Programa de procesado....90

5.1. Subrutina *Lee registro de adquisición* 90
5.2. Subrutina *Procesado* 96
5.3. Subrutina *Control fallos* 102
5.4. Subrutina *Comienza* 104
5.5. Subrutina *Rango* 109
5.6. Subrutina *Continúa* 116
5.7. Subrutina *Completa* 119
5.8. Programa principal 124

CAPÍTULO 6: Desarrollo del programa de visión de resultados 130

1. Introducción 131
2. Funcionalidades del programa de visión de resultados 131
3. Estructura del programa de visión de resultados 133
4. Programa de visión de resultados 135
   4.1. Subrutina *Visualiza imagen* 135
   4.2. Subrutina *Comenzar* 138
   4.3. Subrutina *Rango* 139
   4.4. Subrutina *Imagen* 142
   4.5. Subrutina *Defectos* 144
   4.6. Programa principal 146

CAPÍTULO 7: Conclusiones y línea de futuros trabajos 149

Referencias 152

Acquisition and processing system for weld images taken by an IRB2400 robot

Resumen:

Se pretende en este proyecto el desarrollo de un sistema completo de visión artificial compuesto por un robot dotado de dos cámaras que muestrean soldaduras en una placa de un generador de vapor de una central nuclear.

Los objetivos buscados son dos. Por un lado se pretende realizar un registro en disco de una imagen de todas las soldaduras, y por otro, realizar un procesado automático para determinar aquellas soldaduras que deben ser revisadas por un operario.

Más concretamente se va a realizar la ejecución del proyecto en dos partes. Debido a que no se dispone de un brazo robótico en el laboratorio para poder probar el funcionamiento de este proyecto se va a proceder a su simulación en un computador bajo un entorno de Linux, resultado esta una de las dos partes ya mencionadas. La otra parte del proyecto consiste en programar en un computador específico con sistema operativo Windows XP para obtener un registro en disco de todas las imágenes de soldaduras y su posterior procesado.

Abstract:

This Project develops an artificial vision system that is composed by robot with two cameras that checks the quality of welds in the tube plate of a steam generator of a nuclear power station.

There are two main goals. Firstly, the system will record in a disk one image for every weld. On the other hand, it will automatically process the images to determine which of them should be reviewed by an operator.

Specifically, the project is carried out in two parts. Due to the unavailability of a robotic arm in the laboratory which could test the operation of this project, the robot is going to be simulated in a computer under Linux, being that simulation one of the two parts mentioned before. The other part consist of programming a specific computer under a Windows XP operating system in order to record on disk of all the weld images and their subsequent processing.

CAPÍTULO 1

Introducción

INTRODUCCIÓN

1. Objetivos generales.

El presente proyecto tiene como meta, tal y como su propio título describe, el diseño y el análisis de parte de un sistema automatizado para la adquisición y procesado de imágenes de soldaduras en la placa de tubos de un generador de vapor de centrales nucleares de agua a presión (PWR). Esta placa está taladrada por miles de tubos por los que circula agua caliente a presión. Los tubos se hallan soldados a la placa, y es preciso comprobar la calidad de todas las soldaduras.

En las piezas empleadas para la construcción de una central nuclear es de vital importancia asegurar un factor de calidad muy alto, ya que eso la hace más segura y hace más eficiente el proceso de fabricación, al disminuir la posibilidad de tener que hacer reparaciones o rectificaciones en fases tardías del proceso. Con este proyecto se persigue ese objetivo de aseguramiento de la calidad, consiguiendo comprobar que todas las soldaduras sean correctas y además pudiendo mostrárselo al cliente siempre que lo solicite.

Se pretende fotografiar todas y cada una de las soldaduras. Seguidamente se ordenarán en un directorio, en el cual todas las imágenes correspondientes a una misma fila se encuentran en la misma carpeta. Con esto se consigue tener un registro en disco de todas las soldaduras, obteniendo una prueba física de cómo son las soldaduras por si hiciese falta su consulta futura. Si alguna fotografía tomada ya existía se descartará sin ser ni procesada ni guardada en disco.

Por otro lado y opcionalmente, una vez fotografiadas las soldaduras se procesarán para comprobar que, a priori, son correctas. Esta información se suministrará al operador encargado de realizar el proceso de control de calidad, para su inspección visual y posterior reparación en caso necesario. Con este sistema se asegura un tratamiento completo de todas las soldaduras, que no se puede garantizar con el simple tratamiento manual. De modo que, el proceso posterior de control de calidad con un robot que comprueba la estanqueidad por medios físicos no pasará por soldaduras erróneas, ahorrando así tiempo.

Para los procesos de análisis de soldadura se utilizarán varios dispositivos, que son: un robot, dos cámaras digitales de alta definición y una unidad de procesado de imágenes. El primero es un brazo robótico comercial de la marca ABB modelo IRB2400, el cual es el encargado de posicionar las cámaras, que son el segundo elemento, en las soldaduras. El tercer elemento es un computador preparado específicamente para el procesado de imagen. Para dicho procesado se utilizará un equipo comercial de la marca Dalsa modelo VA31 con Windows XP embebido.

2. Centrales nucleares.

2.1 Introducción.

Una central nuclear es una instalación industrial empleada para la generación de energía eléctrica a partir de energía nuclear. Se caracteriza por el empleo de combustible nuclear compuesto básicamente de material fisionable que mediante reacciones nucleares proporciona calor, que a su vez, es empleado a través de un ciclo termodinámico convencional para producir el movimiento de alternadores que transforman el trabajo mecánico en energía eléctrica. Las centrales nucleares constan de uno o más reactores [1].

2.2 Reactor nuclear.

Un reactor nuclear es una instalación capaz de iniciar, mantener y controlar las reacciones de fisión en cadena con los medios adecuados para extraer el calor generado [2].

2.2.1 Partes del reactor.

Una vez explicada cuál es la misión de un reactor nuclear se procederá a ver los diferentes elementos que los constituyen, teniendo cada uno de estos un papel importante en la generación del calor. Dichos elementos son los siguientes:

- **El combustible:** formado por un material fisionable, generalmente un compuesto de uranio, en el que tienen lugar las reacciones de fisión, y por tanto, es la fuente de generación del calor.

- **El moderador:** se trata de un mecanismo que controla la velocidad con que los neutrones impactan en nuevos núcleos de uranio. Este elemento no existe en los reactores denominados rápidos. Se emplean como materiales moderadores el agua, el grafito y el agua pesada.

- **El refrigerante**: extrae el calor generado por el combustible del reactor. Generalmente se usan refrigerantes líquidos, como el agua ligera y el agua pesada, o gases como el anhídrido carbónico o el helio.

- **El reflector:** permite reducir el escape de neutrones de la zona del combustible, y por tanto, disponer de más neutrones para la reacción en cadena. Los materiales usados como reflectores son el agua, el grafito y el agua pesada.

- **Los elementos de control:** actúan como absorbentes de neutrones, permitiendo controlar en todo momento la población de neutrones, y por tanto, la reactividad del reactor, haciendo que sea crítico durante su funcionamiento, y subcrítico durante las paradas. Los elementos de control tienen formas de barras, aunque también pueden encontrarse diluido en el refrigerante.

- **El blindaje**: evita el escape de radiación gamma, que es la mas dañina, y de neutrones del reactor. Los materiales usados como blindaje son el hormigón, el agua y el plomo.

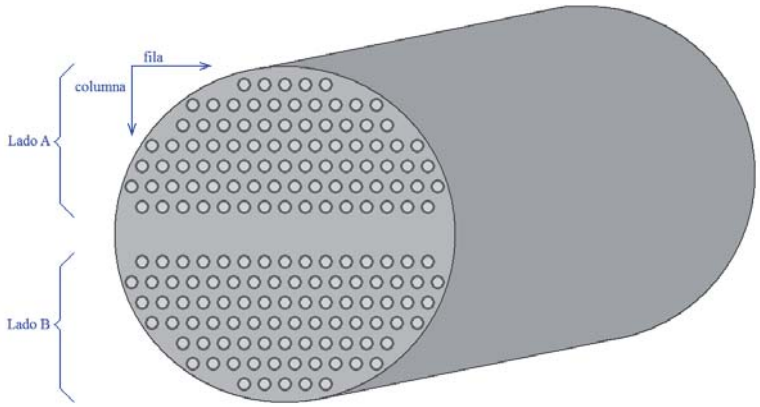
2.2.2 Tipos de reactores.

Los reactores nucleares se clasifican, de acuerdo con la velocidad de los neutrones que producen las reacciones de fisión, en reactores rápidos y reactores térmicos [2]. Los reactores más empleados en las centrales nucleares eléctricas son:

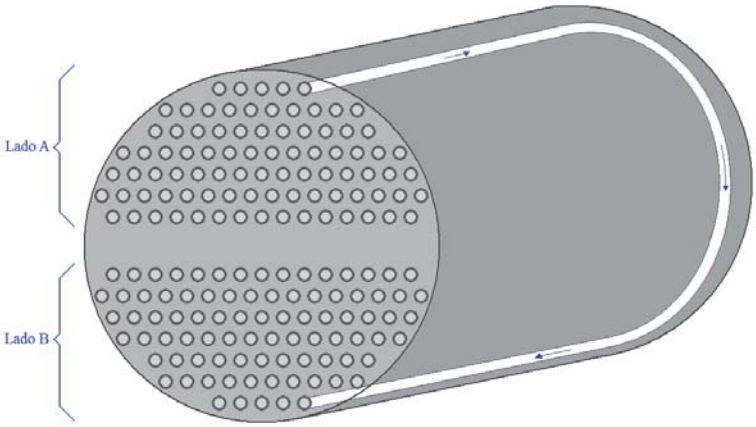
- Reactor de agua a presión (PWR).
- Reactor de agua en ebullición (BWR).
- Reactor de agua pesada (HWR).
- Reactor de grafito-gas.
- Reactor de agua en ebullición (RBMK).

Más concretamente, el fin de este proyecto gira en torno a los reactores nucleares del tipo PWR, en los que son necesarios generadores de vapor, que son el objeto de estudio en este proyecto.

Entrando más en profundidad en los generadores de vapor, se va a analizar la placa de tubos, que consta de miles de soldaduras por donde circulará líquido a refrigerar. La placa de tubos se divide en dos caras, y en cada lado la soldadura se la hace referencia con su posición de fila y columna, como se puede observar en la siguiente ilustración:



Estudiando con más detalle la figura anterior, del lado *A* de la placa se suelda un tubo, el cual recorre varios metros en el interior del generador de vapor, hace un giro de 180 grados y vuelve otra vez a la placa inicial, pero esta vez termina con una soldadura en el lado *B*, como se muestra a continuación:



3. Robótica industrial.

3.1 Introducción.

La historia de la robótica ha estado unida a la construcción de "artefactos", que trataban de materializar el deseo humano de crear seres semejantes a nosotros los cuales nos descargasen de trabajo.

Más concretamente se puede definir como robot industrial a un dispositivo de maniobra destinado a ser utilizado en la industria y dotado de uno o varios brazos, fácilmente programable para cumplir operaciones diversas con varios grados de libertad y destinado a sustituir la actividad física del hombre en las tareas repetitivas, monótonas, desagradables o peligrosas [3].

3.2 Importancia de la Robótica en la Industria.

El diseño e implementación de procesos cada vez más automatizados, es lo que ha permitido a la industria en general fabricar cantidades inimaginables de un producto en concreto a precios competitivos, sin descuidar su calidad y seguridad.

Hablar de automatización de una industria hace pensar en sumas considerables de dinero para invertir en su implementación, pero considerando la disminución del scrap (rechazo interno de producción) y los altos volúmenes de producción, el precio que tiene la automatización en los procesos es realmente bajo [3].

La tendencia a conseguir una completa automatización de la producción abarca todas las etapas de ésta, inclusive el control de la calidad. El robot industrial puede participar en esta tarea usando su capacidad de posicionamiento y manipulación, es decir, a partir del conocimiento que en todo instante tiene la unidad de control del robot de la posición y orientación de su extremo, se puede obtener información, ayudados de algún tipo de sensor si fuese necesario, para registrar posibles desviaciones del elemento bajo test sobre los valores deseados [3].

3.3 Aplicaciones.

Los robots son utilizados en una diversidad de aplicaciones, desde robots tortugas en los salones de clases, robots soldadores en la industria automotriz, hasta brazos teleoperados en el transbordador espacial [3].

La implantación de un robot industrial en un determinado proceso exige un detallado estudio previo del proceso en cuestión, ya que el abanico de posibilidades que ofrece un robot industrial es muy amplio. Sus principales aplicaciones son las siguientes: trabajos en fundición, soldadura, aplicación de materiales, aplicación de sellantes y adhesivos, alimentación de máquinas, procesado, corte, montaje, paletización, control de calidad y manipulación en salas blancas [4].

4. Procesado de imagen, visión artificial.

4.1 Introducción.

Con la mejora de los sistemas de iluminación y captación de imágenes, y el incremento en la capacidad de procesado de los ordenadores personales, la visión artificial ha ido ganando terreno en la industria.

La aplicación de la visión artificial en la industria comprende la adquisición y análisis automáticos de imágenes con el fin de extraer información que permita controlar un proceso o una actividad. Se trata, por tanto, de una tecnología que combina los computadores con las cámaras de vídeo para adquirir, analizar e interpretar imágenes de una forma equivalente a la inspección visual humana. En función del problema que se quiera resolver, los sensores operan el rango de luz visible, son sensibles al infrarrojo no visible o a los rayos X, pudiendo también operar con haces de láser [5].

4.2 Elementos de un Sistema de Visión Artificial.

Un sistema de visión artificial se compone básicamente de los siguientes elementos:

- Fuente de Luz:

Es un aspecto de vital importancia ya que debe proporcionar unas condiciones de iluminación uniformes e independientes del entorno, facilitando además, si es posible, la extracción de los rasgos de interés para una determinada aplicación [5].
La fuente de luz es un factor de vital importancia en los sistemas de visión artificial y afectan de forma crucial a los algoritmos de visión que se vayan a utilizar bajo esas condiciones.

- Sensor de Imagen:

Es el encargado de recoger las características del objeto bajo estudio. Los sensores de imagen, son componentes sensibles a la luz que modifican su señal eléctrica en función de la intensidad luminosa que perciben. La tecnología más habitual en este tipo de sensores es el CCD (Charge Coupled Devices o Dispositivos de Acoplamiento de Carga) donde se integra en un mismo chip los elementos fotosensibles y el conjunto de puertas lógicas y circuitería de control asociada, quedando fuera del mismo el procesado de la señal, que se realiza en un chip externo. La principal ventaja que presentan es menor ruido en la imagen y una mayor uniformidad, es decir, que un píxel sometido al mismo nivel de excitación de luz que sus vecinos no presente cambios apreciables respecto de ellos [6].

También existe otro tipo de tecnología llamada CMOS (Complementary Metal Oxide Semiconductor o Semiconductor de Óxido de Metal Complementario), que presenta la ventaja de realizar todo el procesado de la señal dentro del mismo sensor, obteniendo como ventajas una mayor velocidad en la captura de la imagen y menor coste de fabricación [6].

4.3 Aplicaciones.

La utilización de la visión artificial en la industria abarca un amplio espectro de aplicaciones que, de manera simplificada, pueden agruparse en cinco áreas principales:

- Control de calidad.
- Clasificación por calidades.
- Manipulación de materiales.
- Test y calibración de aparatos.
- Monitorización de procesos.

Prácticamente cualquier sector industrial puede beneficiarse de este tecnología para acelerar sus procesos, para reducir sus costes de producción o para mejorar la calidad de sus productos finales.

5. Metodología de trabajo.

5.1 Requisitos del sistema de adquisición y procesado de imágenes planteados por *Equipos Nucleares S.A.*

5.1.1. Descripción general.

Se pretende en este proyecto el desarrollo de un sistema compuesto por un robot dotado de dos cámaras que muestrean soldaduras en una placa para obtener dos resultados, que son:

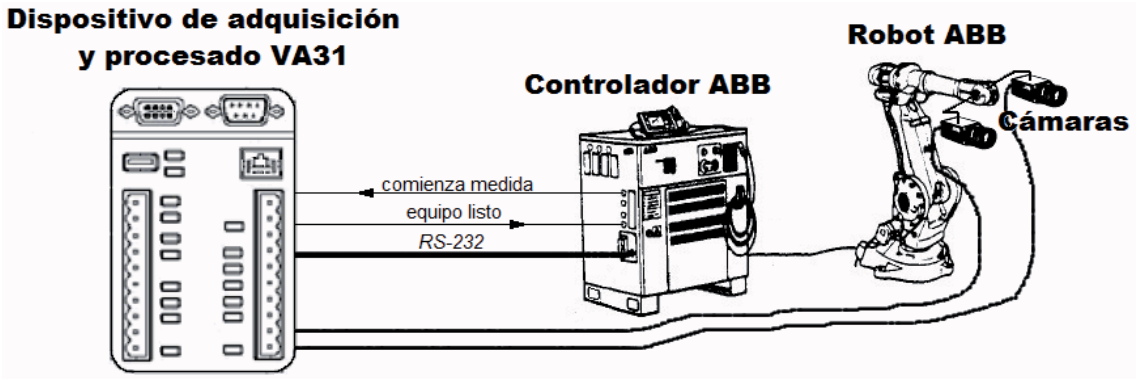
- ✓ Un registro en disco de una imagen de todas las soldaduras
- ✓ Opcionalmente realizar un procesado automático para determinar aquellas soldaduras que deban ser revisadas por un operador.

5.1.2 Estructura del sistema.

Según se puede ver en el diagrama adjunto el sistema sobre el que se va a hacer el desarrollo está compuesto por:

- ✓ Un robot marca ABB modelo IRB 2400.
- ✓ Un sistema de procesado de video marca Dalsa modelo VA31.
- ✓ Dos cámaras monocromo de alta definición.
- ✓ El robot ABB y el VA31 están conectados por una conexión RS-232 y dos líneas digitales, que son:

    - Una salida digital del robot ABB, llamada *comienza medida*.
    - Una salida digital del VA31, llamada *equipo listo*.



5.1.3. Protocolo de comunicaciones.

La comunicación entre el robot ABB IRB2400 y el equipo VA31 se realiza según un protocolo, el cual se detalla a continuación.

En primer lugar como condición inicial el equipo VA31 está operativo y tiene a uno lógico la señal "equipo listo".

A continuación el robot ABB se coloca en una posición de medida y si se cumple la condición anterior éste envía un telegrama de identificación. El telegrama es siempre de longitud constante y es de la siguiente forma:

STXMED_L_R□□□C□□□_R□□□C□□□ETX

Donde:

- Los guiones bajos corresponden a espacios en blanco.
- STX corresponde al ASCII 2.
- ETX corresponde al ASCII 3.
- MED se añade para cubrir posibles aumentos de funcionalidad enviando otras etiquetas.
- L indica el lado que se está procesando, pudiendo ser "A" o "B".
- La primera referencia de Fila y Columna corresponde a la cámara con canal o conexión más bajo.
- R□□□C□□□ corresponde a la soldadura concreta, donde □□□ que está a continuación de *R* corresponde al número de columna, y □□□ que está a continuación de *C* corresponde al número de fila.

Por ejemplo, si la imagen de la primera cámara corresponde a la Fila 5 Columna 9 y la segunda corresponde a la Fila 5 Columna 10 la identificación será:

STXMED_L_R005C009_R005C010ETX

Una vez enviado dicho telegrama, el equipo VA31 contestará con un "ACK" si el mensaje fue leído correctamente o un "NACK" si la recepción fue errónea. Si la recepción fue errónea el proceso automáticamente se detiene. Si el mensaje fue leído correctamente el robot ABB pone a uno lógico la señal "comienza medida" durante el tiempo de un segundo.

Finalmente el equipo VA31 pone a cero la línea “equipo listo” y comienza a realizar en la adquisición y procesado. Una vez acabado pone un uno lógico la señal “equipo listo”, volviendo a estar en la condición inicial.

5.1.4. Adquisición y procesado.

El VA31 adquiere una imagen con cada cámara y las almacena en disco. Todas las imágenes de una fila se almacenan en un directorio de nombre:

LR□□□

Donde:

- L indica el lado qué se esta midiendo, pudiendo valer “A” o “B”.
- R□□□ indica la fila a la que pertenece el tubo.

Por ejemplo, si se está procesando el tubo Fila 5 Columna 8 del lado A, la imagen se grabará en el directorio *AR005*.

El nombre de cada fichero será:

LR□□□C□□□

Donde:

- L indica el lado que se está midiendo, pudiendo valer “A” o “B”.
- R□□□ indica la fila a la que pertenece el tubo.
- C□□□ indica la columna a la que pertenece el tubo.

Por ejemplo, si se procesa el tubo Fila 5 Columna 8 del lado A, la imagen se grabará en el fichero AR005C008.

Es inevitable que alguna soldadura sea adquirida más de una vez. En caso que el fichero ya exista no se reescribirá y se despreciará la adquisición.

El VA31 realiza el procesado de la imagen. En este momento no tenemos experiencia suficiente como para especificar el procesado. Para empezar podemos buscar una mancha de mayor superficie a una dada. Si se encuentra alguna indicación reportable:

- Se añade la referencia de la soldadura al fichero de soldaduras con indicación. El formato de la referencia será igual al nombre del fichero donde se guarda la imagen.
- Se guarda la imagen procesada en el directorio de imágenes con indicaciones con el mismo nombre que la imagen original.
- Se enciende un piloto un tiempo, para indicar al operario que se ha encontrado una soldadura con indicación.

El VA31 llevará una cuenta del número de imágenes adquiridas y el número de imágenes con indicación. Si una variable lo indica, el sistema se detendrá cuando se alcance un número máximo de fallos o cuando el porcentaje de errores sea mayor que uno dado. El piloto de fallo se quedará encendido.

5.1.5. Parámetros del sistema.

- ✓ Situación del directorio raíz donde se guardan las imágenes.
- ✓ Variable que indica que el sistema se detendrá por un número de fallos o por porcentaje de fallos.
- ✓ Número de fallos que provocan la parada del sistema.
- ✓ Porcentaje de fallos que provoca la parada del sistema.

5.1.6. Comentarios generales.

- ✓ En todas las posiciones del robot, ambas cámaras tomarán imagen de una soldadura.
- ✓ La estructura de directorios donde se graban las imágenes, para facilitar el trabajo, será dada por *Equipos Nucleares S.A.*

5.2 Objetivos secundarios.

5.2.1 Simulación de robot ABB.

Debido a que durante la fase de desarrollo del proyecto no se dispone de un brazo robótico comercial de la marca ABB modelo IRB2400, surge como objetivo secundario la necesidad de su simulación software mediante un computador.

Esta simulación se ha llevado a cabo mediante un computador de sobremesa, cuyo sistema operativo, bajo el que se va a desarrollar un programa en C, es Linux. Ha sido necesario el estudio detallado de cómo funciona y se gestionan el puerto serie y el puerto paralelo del que se obtendrán las dos líneas digitales necesarias. Por ello se han estudiado las funciones de Linux que permiten esta interactuación de I/O, así como la forma de diseñar y escribir un programa que permita la simulación de dicho robot.

En el tercer capítulo se explica más en detalle del desarrollo de este objetivo secundario, pero tan necesario como un objetivo principal.

5.2.2 Dispositivo VA31. Programación en Sherlock.

El desarrollo de este proyecto se ha llevado a cabo mediante un computador específico de la marca Dalsa, modelo VA31, el cual tiene Windows XP embebido, siendo necesario el estudio del hardware, dispositivo VA31, y software, programa Sherlock bajo el cual se han desarrollado los programas oportunos.

Desde el punto de vista hardware, ha sido necesario leer y comprender bien cómo funciona este dispositivo. Para ello se ha realizado una lectura exhaustiva del manual de usuario, donde se explica detalladamente su manejo. Por otro lado se han tenido que analizar cómo funcionan las interfaces I/O, así como las limitaciones físicas que presentan.

Desde el punto de vista software, el programa de visión artificial, bajo el cual se han desarrollado los programas necesarios para este proyecto, es el Sherlock. Ha sido

necesario aprender y comprender el manejo de dicho programa. Por un lado ha sido necesario adquirir los conocimientos necesarios para poder programar en este lenguaje. Dicho lenguaje es bastante sencillo y visual, ya que este programa ofrece un amplio abanico de funciones a utilizar. Por otro lado, ha sido necesario aprender cómo manejar desde el propio programa los dispositivos de I/O, así como los parámetros de configuración de dichos puertos y los de configuración de las cámaras de alta definición.

5.2.3 Soldaduras de los tubos de refrigeración.

Es necesario para el desarrollo de este proyecto tener unas nociones básicas sobre soldaduras en los tubos de la placa, que son el sujeto a estudio. Para ello, y en colaboración con *Equipos Nucleares S.A.*, se ha realizado el estudio de cómo son las soldaduras correctas y las anomalías típicas que presentan las mismas.

Mas concretamente, se ha hecho un estudio de las soldadura defectuosas con las causas de fallo más típicas, que son, manchas, burbujas, deformaciones y soldaduras incompletas.

Desde el punto de vista de visión artificial, el programa Sherlock tiene numerosos algoritmos de procesado de imágenes. Para determinar cuál de ellos se ajusta mejor a las exigencias del proyecto, se ha realizado un profundo análisis de cada uno de ellos, viendo así los puntos fuertes de que presentan y cuál realiza mejor esta labor de detección de anomalías.

6. Estructura de la memoria.

En el primer capítulo se han definido los objetivos de este proyecto haciendo referencia a la idea de la que nace. Además, se ha hecho una introducción de los pilares en los que se sostiene el proyecto, que son, los reactores nucleares, robots industriales y los sistemas de visión artificial. Por otro lado, se explicarán los requisitos del sistema de adquisición y procesado de imágenes planteados por *Equipos Nucleares S.A.* El presente capitulo finaliza con la exposición de los objetivos secundarios.

En el segundo capítulo se hace una descripción de los sistemas que integran este proyecto, obteniendo así una visión de sus principales características a modo de introducción de los equipos empleados para el desarrollo del mismo. Este capítulo se concluye con la relación que existe entre estos elementos anteriormente explicados con el proyecto desarrollado.

El tercer capítulo se centra en la simulación del robot que se va a realizar debido a que no se dispone un brazo robótico ABB IRB2400. Se aborda la simulación del robot tanto desde el punto de vista hardware como software. También se explica con detalle el protocolo de comunicación entre el robot y el computador.

El capítulo cuarto se centra en la programación software destinada a la adquisición de imágenes. Se explica cómo funciona dicho programa y qué reglas se siguen para el almacenamiento de imágenes.

El capítulo quinto se centra en el procesado de imágenes, necesario para poder detectar anomalías en las soldaduras. Más concretamente se ha desarrollado un programa, el cual da la posibilidad, bajo un mismo criterio de procesado, realizar el tratamiento de imágenes de cuatro formas diferentes, buscando así la máxima comodidad en su utilización. En este capítulo se muestra la necesidad de este programa, así como la estructura seguida y explicación del programa desarrollado.

El capítulo sexto se centra en la interpretación de resultados de las imágenes anteriormente adquiridas y procesadas. Para ello se ha realizado un programa que muestra dichos resultados, pudiendo escoger el criterio según el cual se muestran las imágenes. En esencia, en este capítulo, se muestra la necesidad de este programa, así como la estructura seguida y explicación del programa desarrollado.

Por último, en el capítulo séptimo se plantean las conclusiones a las que se ha llegado, estableciéndose las posibles mejoras futuras, algunas de las cuales serán detalladas.

CAPÍTULO 2

Elementos y bases del sistema de adquisición y procesado de imágenes

ELEMENTOS Y BASES DEL SISTEMA DE ADQUISICIÓN Y PROCESADO DE IMÁGENES

1. Introducción.

Este presente capítulo se va a centrar en explicar cada unos de los cuatro elementos que forman los pilares del desarrollo del proyecto.

Estos cuatro apartados que vienen a continuación, y que forman el segundo capítulo, se centra, en explicar cada uno de los cuatro dispositivos empleados, que son, el robot ABB, la cámara de video, el dispositivo VA31, que es el computador de uso dedicado a la visión artificial, y el programa Sherlock, que es programa bajo el cual se han desarrollado los programas de adquisición, procesado y visualización de resultados.

Se pretende mostrar y ver las potencialidades de cada uno de estos cuatro elementos, explicando además las características que posee cada uno de ellos.

2. Robot ABB IRB2400.

El IRB2400 es el robot industrial más popular de ABB, muy utilizado para labores de soldadura. Dicho robot, en sus distintas versiones y con su mejor precisión, proporciona unas prestaciones excelentes para manipulación de materiales, asistencia a máquinas y aplicaciones de procesos. Dicho robot ofrece altos volúmenes de producción, bajos tiempos de producción y en consecuencia, de entrega rápida para los productos fabricados [7].

El diseño compacto de este robot facilita su instalación, ofreciendo la posibilidad de montaje invertido. Su robusta construcción y un menor número de partes de que se compone contribuyen a su fiabilidad y mayores intervalos de tiempo entre intervenciones de mantenimiento [7].

Por otro lado, gracias al sistema de control del movimiento de ABB, el robot optimiza la aceleración y desaceleración, lo que implica un bajo tiempo de ciclo, presenta buena precisión de la trayectoria y de repetibilidad de la posición. (RP= 0.06 mm) [7].

Este brazo robótico es muy potente, ya que combina una capacidad de carga de 7 - 20 kg y un alcance de 1810 mm., pudiendo incluso trabajar en ambientes de trabajo exigentes, ya que cuenta con clasificado IP67, Steam Washable, Foundry Plus y Clean Room ISO class 100 [7].

Por todo esto, le hace como se expuso al principio, ser un robot referente en la industria. A continuación se muestra una imagen de este robot:

Hasta ahora solo se ha mentado lo que es en sí el brazo robótico, pero éste por sí solo no tiene sentido si no se habla de su controlador, ya que éste es el que hace las veces de cerebro humano que emite órdenes a los músculos del brazo. El controlador es un computador que se encarga de gestionar las órdenes que se le dan al robot, descompuestas en movimientos de los motores que tiene que mover para alcanzar una determinada posición.

A continuación se muestra una imagen ilustrativa de un controlador típico de robot ABB.

3. Cámara Jai®.

La cámara escogida para la realización de este proyecto es la Jai® CV-A11 de la casa Dalsa, de la cual se muestra una imagen a continuación.

Esta cámara está especialmente diseñada para aplicaciones donde se precise la máxima calidad de imagen, consiguiendo además una excelente focalización [2].

Con respecto al sensor que utiliza es del tipo CCD de alta resolución monocromo con una resolución de 648x492 y una velocidad estándar de captura de 30 cuadros por segundo, existiendo la posibilidad de trabajar en barrido variable, llegando de este modo a obtener imágenes a 124 imágenes por segundo [2].

En cuanto a la interfaz que utiliza esta cámara, hay que destacar que puede trabajar tanto en formato RS-170, obteniendo así, como se comentó antes, 30 fotogramas por segundo [2].

Por otro lado, es importante hablar del objetivo que monta esta cámara. Dicho objetivo es un Computar TEC-55, de la casa Telecentric lens, de 2/3"-1/3", 55.0mm, F2.8, o lo que es lo mismo, se puede montar sobre sensores de 2/3" y 1/3", posee distancia focal de 55mm, y que su apertura es de 2.8. El ajuste del iris y del enfoque se realiza manualmente [3]. A continuación de muestra una foto de dicho objetivo:

Finalmente, se van a describir las principales características de esta cámara [2]:

- Sensor de Alta resolución 1/3" IT CCD.
- Extrema sensitividad: - 0.05 Lux en el sensor.
- Rango dinámico mejorado y relación señal/ ruido superior a 56dB.
- Funciona en formato entrelazado y progresivo.
- Shutter Electrónico de (1:60 a 1:100.000) y posibilidad de Reset Asíncrono.

- Posibilidad de trabajar con largo tiempo de exposición, para aplicaciones con baja iluminación.
- Posibilidad de trabajar con sincronismo HD/VD de entrada o salida.
- Dispone de señal de píxel clock para aplicaciones donde se necesite precisión sub-píxel.
- Configuración RS-232, se acompaña programa en Windows 9X/NT
- Montura C
- Dimensiones: 29 x 44 x 66mm.
- Peso: 150g.

4. Unidad de procesado VA31.

La unidad de procesado VA 31 es un sistema compacto de visión artificial, resultando ser una plataforma integrada que incluye el procesamiento, display, captura de imágenes, networking y comunicaciones industriales I/O. Estos componentes de hardware estándar, alojados en un chasis de aluminio, proporcionan la base para un sistema de visión industrial de gran versatilidad [2].

Este dispositivo es ideal para resolver los problemas de control de procesos de fabricación y para la ayuda en la producción en los procesos de automatización industrial, además de aplicaciones como posicionamiento de robots y verificación del ensamblaje final, por no olvidar el propósito de este proyecto, la calidad [2].

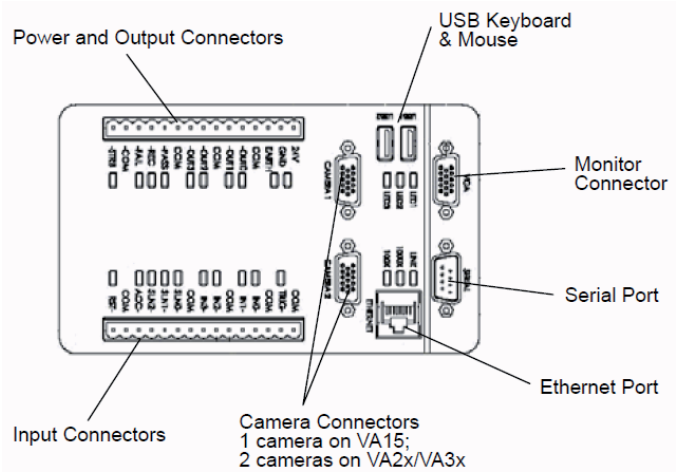
El VA31 es un sistema independiente que no requiere una interfaz con un PC para la configuración, ya que todo el software necesario, las interfaces de usuario y controles de comunicación, están integrados en este dispositivo. Mediante las conexiones remotas se puede controlar y monitorizar este computador [2].

Dicho dispositivo soporta cámaras de varias resoluciones, pudiendo llegar a procesar imágenes de 1600 x 1200 píxeles, tanto en monocromo como en color, y en formato síncrono o asíncrono [2].

El VA31 puede incluir dos tipos de software dependiendo de las necesidades de los usuarios (iNspect y Sherlock). Ambos software soportan los protocolos estándar tales

como Modbus y Ethernet I/P, para poder ser conectados a otros equipos del entorno industrial [2].

Con respecto a las interfaces que posee es importante mostrar la siguiente figura ilustrativa que representa perfectamente las I/O con las que se puede interactuar:



Finalmente se van a describir sus principales características [2]:

- Memoria de Programa: 512MB.
- Memoria de almacenamiento 4GB Flash.
- Adquisición: Hasta DOS cámaras simultaneas.
- Resolución de cámaras: 640x480, 1300x1024, 1600x1300 píxeles.
- Comunicación: 2 USB, Ethernet 10/100/1000Mbps, 1RS232, 26 indicadores de estado.
- Visualización: Posibilidad de conectar un monitor local XGA.
- Conexión para teclado, ratón y/o disco externo.
- Input: 10 Opto-ajustables.
- Output: 8 Opto-ajustables.
- Software: iNspect o Sherlock.
- Alimentación: 24VDC.
- Dimensiones: 9.5x16x5cm.
- Peso: 0.75Kg.

5. Programa de procesado Sherlock

Sherlock es un software de visión artificial avanzado, que puede configurarse fácilmente, para resolver una gran variedad de aplicaciones de inspección automatizada.

Este entorno de diseño gráfico proporciona un conjunto de potentes herramientas y capacidades, que mediante un entorno gráfico hace muy sencilla su programación, no necesitando de este modo ningún conocimiento de lenguajes de programación, y facilita, a los integradores de sistemas y a los clientes finales de sistemas de visión industrial, un entorno que les permite reducir el tiempo de desarrollo y la puesta en funcionamiento de los sistemas, así como de un programa muy visual y fácilmente comprensible [2].

Sherlock proporciona un conjunto completo de herramientas y potencialidades de visión, que pueden utilizarse en aplicaciones en todo tipo de industria, pudiendo así generar una solución utilizando la extensa biblioteca de preprocesadores y algoritmos avanzados de Sherlock. Si fuese necesario, existe la posibilidad de escribir desarrollos propios de programación, importar herramientas de terceros y desarrollar interfaces de operador personalizadas [2]. Las herramientas más importantes que incorpora son las siguientes:

- Herramientas Especiales: Incluye un extenso número de herramientas especiales, que han sido desarrolladas específicamente para simplificar las tareas de inspección más complejas [2].

- Herramienta Cordón: El algoritmo de la herramienta cordón inspecciona un cordón (línea fina) de material. Una aplicación típica es inspeccionar cordones de pegamento, que sujetan las juntas en los ensamblajes de automóviles [2].

- Herramienta de Búsqueda de Esquinas o a la perspectiva: La herramienta de búsqueda de esquinas genera una matriz de "puntos esquina", que pueden manipularse con las fórmulas de Sherlock para medir el espacio entre "picos y valles" de objetos manufacturados, como por ejemplo tornillos [2].

- Herramienta Color: Sherlock proporciona herramientas de corrección, clasificación y presencia de color. Soporta además mapeado de color, una técnica que permite segmentar la imagen por color, para posteriormente aplicar herramientas en monocromo a la inspección [2].

- Herramienta Línea Láser: Las herramientas láser se utilizan para medir el perfil de las partes o para detectar irregularidades, como por ejemplo detectar la envoltura de protección en una tubería de alta presión [2].

- Herramienta de Calibración: Sherlock ofrece numerosos métodos para trasladar los píxeles a coordenadas del mundo real. Las herramientas de calibración, también corrigen las distorsiones debidas a la óptica [2].

- Comunicación: Sherlock proporciona interfaces con una gran variedad de medios de comunicación y soporta protocolos estándar de fabricación, como son Modbus y Ethernet/IP, entre otros [2].

- Personalización: Las herramientas Java Script de Sherlock, conjuntamente con las instrucciones de edición de dibujar y arrastrar, permiten desarrollar fórmulas personalizadas, para realizar operaciones en línea y de fondo [2].

CAPÍTULO 3

Simulación del robot IRB2400

SIMULACIÓN DEL ROBOT IRB2400

1. Introducción.

Este capítulo nace de la carencia de disponibilidad de un brazo robótico IRB2400 de la casa ABB, siendo su objetivo principal la simulación del mismo.

En este capítulo se va a simular el comportamiento de dicho robot. Para ello, a partir de un computador de sobremesa, se va a manejar su puerto serie y paralelo, de donde se obtendrán las dos líneas digitales necesarias, del mismo modo que lo haría el robot ABB. Más concretamente, se ha desarrollado un programa en C bajo Linux que replica el comportamiento del ABB según el protocolo de comunicaciones, que se explicó en el capítulo segundo y que se volverá a recordar en éste.

Con lo que respecta a la parte de comunicación con el otro sistema, el computador VA31, cabe comentar a modo de esquema las conexiones físicas necesarias para una comunicación entre ambos dispositivos, que son las siguientes:

- Dos líneas digitales, las cuales se obtienen del puerto paralelo, como se explicará mas adelante.
- Un conexión RS-232, la cual se obtiene del puerto serie, como también se desarrollará en este capítulo.

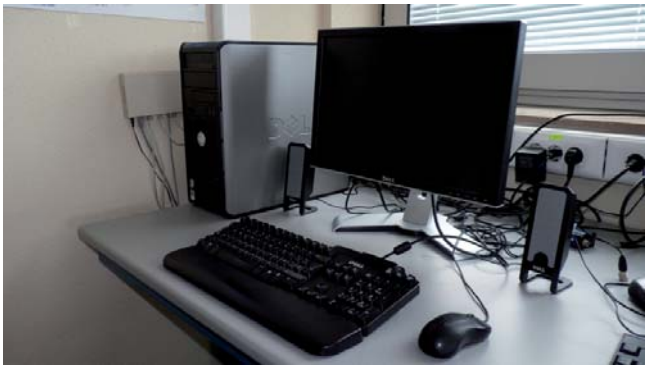
Este capítulo se va a enfocar desde un punto de vista tanto hardware como software, incluyendo en cada uno de estos el cable utilizado para llegar al objetivo de este capítulo.

2. Hardware.

2.1. Equipo:

El computador utilizado para esta simulación es un DELL modelo Optiplex 755, con tecnología Core vPro. La elección de este PC se basa en el requisito de que dispone tanto de un puerto serie (DB9) como de un puerto paralelo (DB25). Es muy importante que posea estos dos mencionados puertos, ya que es el interfaz físico por el cual se comunicará con el dispositivo VA31.

A continuación se muestra una imagen de dicho equipo:

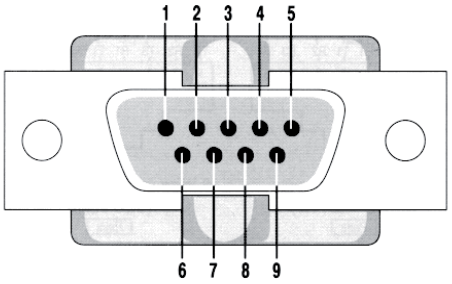


2.2. Puerto serie.

El puerto serie sirve para conectar un dispositivo a una computadora. La información se transmite por un solo conducto, o lo que es lo mismo, bit a bit de forma secuencial [10].

La comunicación serie se va a realizar mediante una conexión RS232, que es una interfaz que designa una norma para el intercambio serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Data Communication Equipment, Equipo de Comunicación de datos) o dos DTE's entre sí [10].

Una vez hecha esta breve introducción a la interfaz RS232, se va a proceder al estudio de cada pin en particular, viendo cuál es la misión de cada uno [10].



Pin	Señal	
1	Carrier Detect	DCD
2	Received Data	RD
3	Transmitted Data	TD
4	Data Terminal Ready	DTR
5	Common Ground	G
6	Data Set Ready	DSR
7	Request To Send	RTS
8	Clear To Send	CTS
9	Ring Indicator	RI

Como la comunicación a implementar es muy sencilla, sólo se utilizarán tres pines, que son, el pin 2 (recived data), el pin 3 (transmitted data) y el pin 5 (common ground).

Por otro lado, cabe destacar que el puerto serie posee el equipo para poder establecer la comunicación es un conector DB9 macho integrado en la placa base.

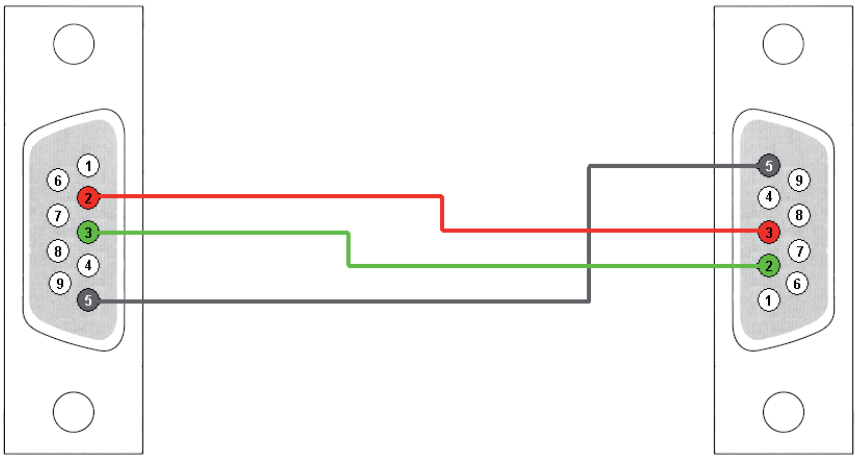
Cable:

Esta parte del capítulo, se va a centrar en el diseño del cable necesario para la comunicación de los dos sistemas.

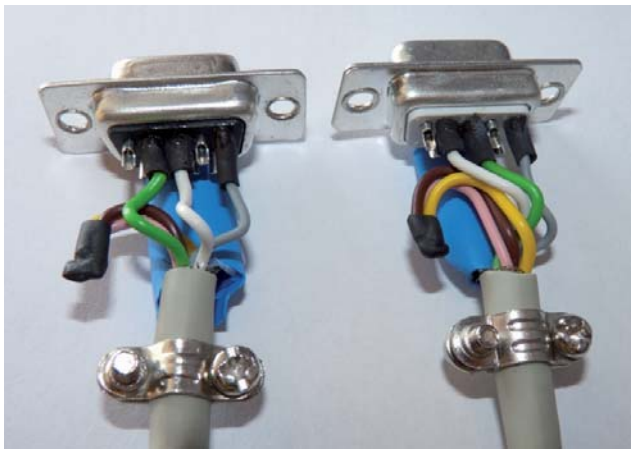
Lo que se persigue, es comunicar dos equipos (DTE's) entre sí, los dos con un puerto serie DB9 macho, leyéndose de aquí los dos requisitos cruciales para poder implementarlo.

Por un lado necesitamos que los dos terminales DB9 del cable sean hembras, ya que los dos computadores poseen un terminal macho integrado a su placa base.

Por otro lado se tiene que unir dos DTE's, por lo que la transmisión y recepción tienen que ser cruzadas, ya que si no los dos equipos colisionarán al transmitir y no oirán nada al recibir. La implementación física de esto que se acaba de comentar es tan fácil como cruzar los pines 2 y 3 en la conexión, como se muestra a continuación:



Para implementar esta conexión se compraron dos terminales DB9 hembras con sus correspondientes carcasas plásticas y un cable de 6 hilos apantallado. De esos 6 hilos se utilizaron tres, dejando los otros tres restantes sin conectar. A continuación, se muestra una imagen de cómo quedó dicho cable después de la conexión de los hilos, donde se aprecia perfectamente el cruce de los pines 2 y 3:

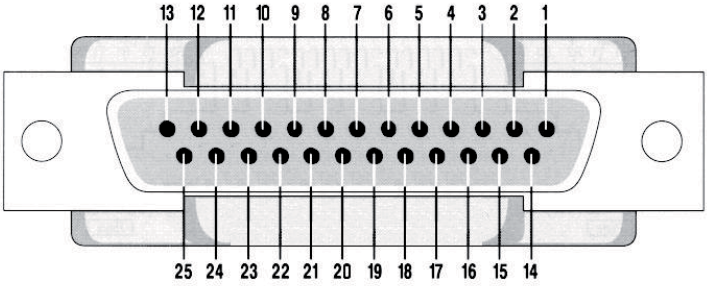


Finalmente se muestra una figura de cómo quedó construido el cable

2.3. Puerto paralelo.

Un puerto paralelo es una interfaz entre un computador y un periférico, cuya principal característica es que los bits de datos viajan juntos, enviando un paquete de un byte a la vez. Es decir, se implementa un cable o una vía física para cada bit de datos formando un bus [11]. Mediante el puerto paralelo se puede controlar también periféricos como focos, motores, sensores y un amplio abanico de dispositivos.

El terminal que posee el computador para implementar dicho puerto paralelo es un DB25 hembra, cuyo esquema de conexiones es el que se muestra a continuación [11]:



Pin	Señal	
1	Control	CTR
2	Data out	DOUT
3	Data out	DOUT
4	Data out	DOUT
5	Data out	DOUT
6	Data out	DOUT
7	Data out	DOUT
8	Data out	DOUT
9	Data out	DOUT
10	Data in	DIN
11	Data in	DIN
12	Data in	DIN
13	Data in	DIN
14	Control	CTR

15	Data in	DIN
16	Control	CTR
17	Control	CTR
18	Ground	GND
19	Ground	GND
20	Ground	GND
21	Ground	GND
22	Ground	GND
23	Ground	GND
24	Ground	GND
25	Ground	GND

Se aprecia que la gran ventaja que tiene utilizar el puerto paralelo es que se tiene en él un control bit a bit, pudiendo manejar un pin concreto de los 25 que posee.

Con respecto a los niveles de tensión y corriente presentes en el puerto paralelo del PC es importante destacar que responden a los estándares de la familia lógica TTL, siendo un estado alto representado por 5V de corriente continua y un estado bajo indicado por la ausencia de tensión (0V). Hablando de corriente, cada pin del puerto paralelo puede proporcionar hasta 10mA en forma segura y 20mA como límite por breves períodos de tiempo [11].

Para la comunicación entre el computador que hace las veces de brazo robótico y el sistema VA31 se necesitan dos líneas digitales, que se implementan gracias a este puerto paralelo. Las dos líneas necesarias son las siguientes:

- "Comienza medida": señal que envía el robot ABB para indicarle al equipo VA31 puede comenzar la adquisición de imágenes.
- "Equipo listo": señal que envía el VA31 para indicarle al robot ABB que está preparado para que el envía el telegrama.

Es decir, desde el punto de vista del este PC hay una línea de escritura, *comienza medida*, y otra de lectura, *equipo listo*.

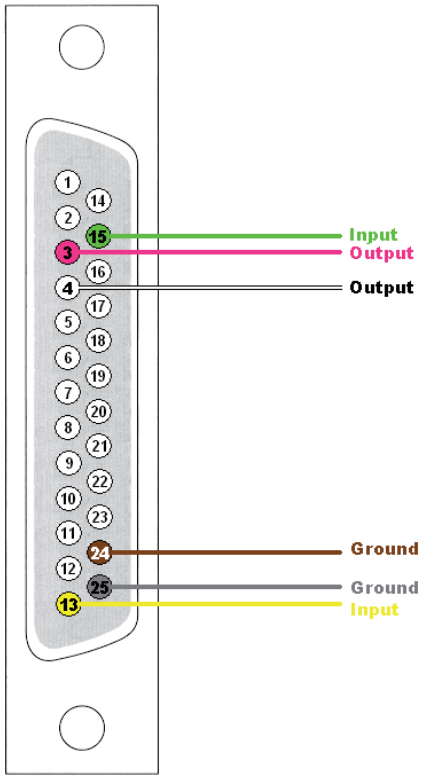
Cable:

Una vez vistas las necesidades, lo siguiente es implementar físicamente el cable que permita tener esas líneas digitales necesarias.

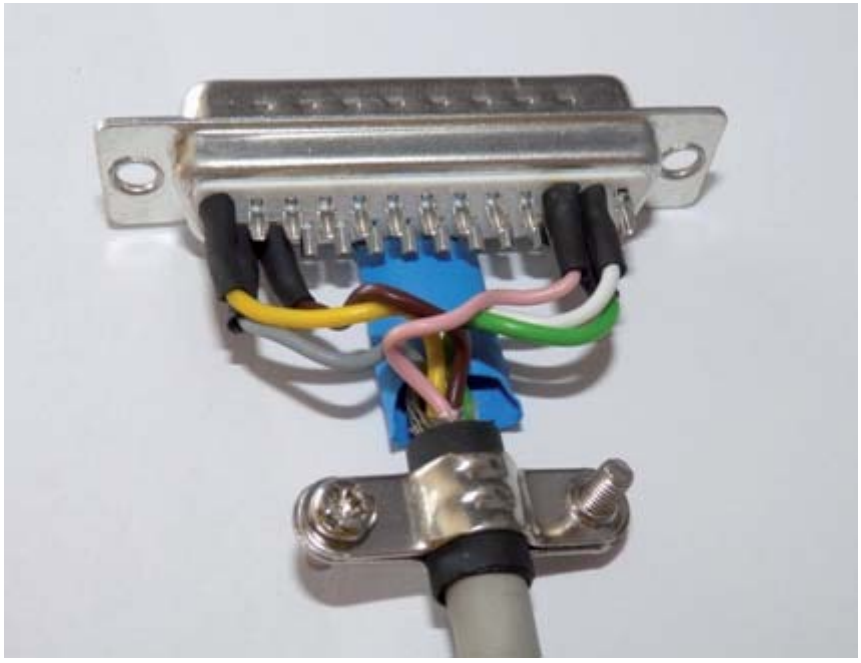
Para implementar esta conexión se compraron dos terminales DB25 machos con sus correspondientes carcasas plásticas y un cable de 6 hilos apantallado. De esos 6 hilos se utilizaron uno para transmisión, uno para recepción y otro de tierra. Los otros seis hilos se repartieron de forma equitativa por si fuese necesario en un futuro aumentar las señales, por lo que además de lo anteriormente descrito se tiene un hilo más para transmisión, recepción y tierra.

Concretando más, para las líneas de transmisión se utilizó el pin2 (hilo rosa) y pin3 (hilo blanco), para las líneas de recepción el pin13 (hilo amarillo) y pin15 (hilo verde) y

finalmente para tierra los pines 24 (hilo marrón) y 25 (hilo gris). A continuación se muestra una imagen ilustrativa de los pines utilizados con el color de hilo asociado:



Una foto que ilustra cómo quedaron los hilos soldados a la carcasa DB25



Finalmente el cable quedó consolidado de la siguiente manera:

Se aprecia que el extremo que no tiene la carcasa DB25 tiene los hilos sueltos, esto es debido a que es necesario dejarlos así para la interconexión con el computador VA31, como se verá en el Capítulo IV.

3. Software.

3.1. Sistema operativo:

El sistema operativo bajo el que se va a desarrollar y ejecutar el software que simula el robot es Linux, ya que ofrece un grado de libertad total en cuanto a la programación y manejo de I/O. Más concretamente se va a realizar sobre distribución Ubuntu.

Para el manejo tanto del puerto serie como del paralelo es imprescindible programarlos y ejecutarlos desde Linux, ya que ofrece la posibilidad del control total sobre dichos puertos, siendo posible la lectura bit a bit del puerto paralelo.

3.2. Puerto serie.

Para conseguir el control de puerto serie se han implementado cuatro funciones, que se corresponden a dos funciones para apertura y clausura y otras dos funciones para enviar y recibir datos, para así facilitar el manejo de este puerto [12][13][14]. A continuación se detallan un poco más las cuatro funciones empleadas:

- Función ***serial_open()***:

Esta función abre el puerto serie para lectura/escritura y en modo crudo (raw). En este modo no se procesan ciertos caracteres especiales de control. Se usa el puerto serie sólo para enviar/recibir bytes sin interpretarlos.

Se configura el modo de funcionamiento serie a 8N1 (8 bits de datos, 1 bit de stop y sin paridad). La velocidad es la indicada por el usuario en el parámetro *baud*, pudiendo tomar como valor este parámetro: B9600, B19200, etc.

Esta función devuelve el descriptor del puerto serie (o -1 si ha ocurrido un error), que será necesario para realizar las lecturas y escrituras.

```
int serial_open(char *serial_name, speed_t baud)
  {
    struct termios newtermios;
    int fd;
    //Abrimos el puerto serie
    fd = open(serial_name,O_RDWR | O_NOCTTY);
    //Configuramos los atributos del puerto serie:
    // -No paridad
    // -8 bits de datos
    // -Otras cosas
    newtermios.c_cflag= CBAUD | CS8 | CLOCAL | CREAD;
    newtermios.c_iflag=IGNPAR;
    newtermios.c_oflag=0;
    newtermios.c_lflag=0;
    newtermios.c_cc[VMIN]=1;
    newtermios.c_cc[VTIME]=0;
    //Fijamos las velocidad
    cfsetospeed(&newtermios,baud);
    cfsetispeed(&newtermios,baud);
    //Limpiamos el buffer de entrada
    if (tcflush(fd,TCIFLUSH)==-1)
       {
          return -1;
       }
    //Limpiamos el buffer de salida
    If (tcflush(fd,TCOFLUSH)==-1)
       {
          return -1;
       }
    //Configuramos el puerto serie
    if (tcsetattr(fd,TCSANOW,&newtermios)==-1)
       {
          return -1;
       }
    //Devolvemos el descriptor del fichero
    retreturn fd;
  }
```

- Función ***serial_send()*:**

Esta función envía una cadena de bytes por el puerto serie. Lo único que hace es invocar la llamada al sistema **write()**. Como parámetros se pasa el descriptor serie del puerto serie (*serial_fd*), devuelto por la función *serial_open()*, el array con los datos a enviar (*data*) y su tamaño (*size*).

```
void serial_send(int serial_fd, char *data, int size)
  {
     write(serial_fd, data, size);
  }
```

- Función **serial_read():**

Esta función se usa para leer datos del puerto serie. Se le pasan como parámetros el descriptor del puerto serie (*serial_fd*), el array donde almacenar los datos recibidos

(*data*), el tamaño máximo de bytes a recibir (*size*, para no desbordar el array) y el tiempo máximo para recibir los datos (*timeout_usec*). Si transcurre un tiempo igual a *TIMEOUT* y no se han recibido datos, la función retornará y devolverá el control a la que la invocó.

Para realizar las lecturas no bloqueantes, se usa la llamada al sistema select(). En este caso devuelve 1 si hay datos esperando a ser leídos y 0 si ha ocurrido un timeout. A continuación se invoca a read() para leer estos datos. El proceso se repite para garantizar que es posible recibir datos de tamaño *size*.

Se devuelve el número de bytes leídos (ó 0 si ha ocurrido un timeout).

```
int serial_read(int serial_fd, char *data, int size, int timeout_usec)
  {
     fd_set fds;
     struct timeval timeout;
     int count=0;
     int ret;
     int n;
     //Esperamos los datos. Un bloque de tamaño byte se espera que llegue antes del tiempo
     //"timeout_usec".
     do
        {
           //Fijamos la variable fds para esperar al descriptor serie
           FD_ZERO(&fds);
           FD_SET (serial_fd, &fds);
           //Fijamos el time-out (en microsegundos)
           timeout.tv_sec = 0;
           timeout.tv_usec = timeout_usec;
           //Esperamos los datos
           ret=select (FD_SETSIZE,&fds, NULL, NULL,&timeout);
           //Si se están escribiendo datos los leemos
           if (ret==1)
              {
                 //Leemos los datos (n bytes)
                 n=read (serial_fd, &data[count], size-count);
                 //El número de bytes recibidos se incrementa en n
                 count+=n;
              }
           //Repetimos bucle hasta que recibamos el tamaño del bloque en bytes o se exceda
           // el tiempo de espera "timeout_usec"
        } while (count<size && ret==1);
     //Devolvemos el numero de bytes leídos o “0” si el time-out si ha ocurrido
     return count;
  }
```

- Función ***serial_close()***:

Esta es la última función, que simplemente invoca a close() para cerrar el puerto serie.

```
void serial_close(int fd)
  {
     close(fd);
  }
```

3.3. Puerto parelelo.

Para la interactuación con el puerto paralelo se han utilizados dos funciones de Linux, incluidas en la librería *sys/io.h*, que permiten leer y escribir en dicho puerto, además de otra función necesaria para la apertura previa y clausura posterior a su uso [12][13][14]. A continuación se explican más detalladamente las funciones utilizadas:

- Función ***ioperm ()***:

Con esta función lo que se consigue es abrir y cerrar el puerto paralelo, estableciendo los bits de permiso de acceso al puertos, y requiriendo además privilegios de root para su ejecución.

Esta función necesita tres variables de entrada, las cuales se explican a continuación. El primer parámetro es la dirección del puerto, que en este caso es la dirección *0x379* correspondiente al puerto *lp0*. El segundo parámetro es el número de puertos a los que se accede después del primero. Por ejemplo, tomando como puerto inicial el *0x379*, si en este parámetro se introduce el valor de tres, lo que se esta haciendo es abrir los puertos *0x379*, *0x380* y el *0x381*. Finalmente, el tercer dato indica si se está abriendo el puerto, en cuyo caso tendrá valor "1", o si se esta cerrando, en cuyo otro caso valdrá "0".

Una vez ejecutada esta función, en caso de éxito devuelve cero y en caso de error devuelve -1, poniendo en errno un valor apropiado.

- Función ***outb()***:

Esta función es la que permite escribir un valor en el puerto serie. Sus parámetros de entrada son, un por lado el valor se que desea sacar por el puerto paralelo, y por otro la dirección del mismo.

Para el caso concreto de este proyecto se utilizará el valor en hexadecimal *0x01* para escribir un "1" en el pin 4 del puerto paralelo, y el valor *0x00* para escribir un "0" en el mismo.

- Función ***inb()***:

Esta función es la que permite leer un valor del puerto serie. Sólo posee un valor de entrada, y es la dirección de las líneas de *Status*, que en este caso es la dirección hexadecimal *0x379*.

Una vez realizada la lectura, lo que se obtiene es una lectura de todas las líneas de *Status*, por lo que es necesario aplicar una máscara para solo leer el bit que interese, es decir, si se hace una operación lógica AND entre el valor leído y la máscara obtendremos únicamente el valor, sea "0" o "1", del pin del cual es necesario conocer su estado.

Para este caso concreto la máscara es 1000, que es la que permite leer el valor del 4º bit, correspondiente a la a la línea de Status S3 (pin15).

3.4. Protocolo de comunicaciones.

El protocolo de comunicación se detalla en el segundo capítulo, pero recuérdese su funcionamiento. En primer lugar, como condición inicial el equipo VA31 está operativo y tiene a uno lógico la señal “equipo listo”.

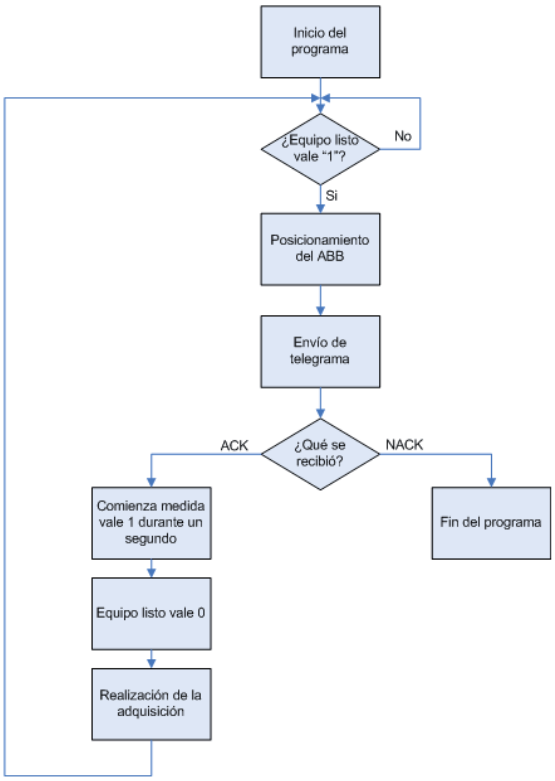
A continuación, el robot ABB se coloca en una posición de medida y si se cumple la condición anterior este envía un telegrama de identificación. El telegrama es siempre de longitud constante y es de la siguiente forma:

STXMED_L_R□□□C□□□_R□□□C□□□ETX

Una vez enviado dicho telegrama, el equipo VA31 contestará con un “ACK” si el mensaje fue leído correctamente o un “NACK” si la recepción fue errónea. Si la recepción fue errónea el proceso automáticamente se detiene. Si el mensaje fue leído correctamente el robot ABB pone a uno lógico la señal “comienza medida” durante el tiempo de un segundo.

Finalmente el equipo VA31 pone a cero la línea “equipo listo” y comienza a realizar la adquisición y procesado. Una vez acabado pone un uno lógico la señal “equipo listo”, volviendo a estar en la condición inicial.

A continuación se muestra un diagrama ilustrativo de este protocolo desde el punto de vista del robot



3.5. Simulación del robot ABB.

Esta parte del capítulo se centra en explicar cómo se ha realizado el programa que simula el robot. En concreto se ha escrito un código en c compilado bajo Linux que reproduce fielmente el comportamiento que tendría este brazo robótico. En adelante se explicarán las partes de dicho código y el papel que desempeña cada trozo de código para llegar a su simulación. Para ello, se muestra a continuación el siguiente pseudo código:

- Preguntar si se desea ejecutar el programa o cancelar el mismo.
- Si ejecutar:
  {
  - Apertura de puertos e inicialización de variables.
  - Bucle principal (sin condición):
    {
    - Esperar al sistema de adquisición.
    - Comprobaciones: Si se han recorrido ya todas soldaduras se finaliza el programa. En otro caso se continúa dentro del bucle principal.
    - Generar el telegrama.
    - Enviar el telegrama.
    - Esperar confirmación
    - Acciones finales.

    }

  }
- Si cancelar: se termina la ejecución del programa

3.5.1 Librerías.

En un primer lugar, se mostrarán las librerías utilizadas para poder utilizar todas las funciones de Linux requeridas, así como las necesarias para el manejo tanto del puerto serie como paralelo. Dichas librerías son las siguientes:

```
#include <termios.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/io.h>
#include <fcntl.h>
```

3.5.2 Constantes y variables.

En segundo lugar, se mostrarán las constantes que se han utilizado para que el programa resulte de más fácil manejo:

```
#define puerto_paralelo 0x378 //definimos el puerto a utilizar (lp0)
#define est_puerto 0x379 //dirección de las líneas de Status

#define TIMEOUT 1000 //time-out (1 milisegundos puestos en microsegundos)
```

```
#define mybaud B9600 //velocidad de la comunicación (en baudios)
#define puerto_serie "/dev/ttyS0" //descripción del puerto serie que vamos a utilizar

#define num_lados 2  //numero de lados que tiene la placa (siempre dos)
#define num_filas 10 // numero de filas que tiene la placa
#define num_columnas 10 // numero de lcolumnas que tiene la placa
```

A continuación y antes de comenzar con la explicación del código en sí, se muestran todas las variables utilizadas:

```
//Variables del puerto paralelo
unsigned int comienza_medida; //variable que contiene el valor de comienza mediada.
short int lectura_paralelo; //variable que contiene lo leído del puerto paralelo (0 es un "0" y 8 es un "1")
int iter_para; //numero de iteraciones que estamos haciendo preguntando al puerto paralelo sobre si hay
cambios

//Variables del puerto serie
int serial_fd; //descriptor del puerto serie
int n; //variable que contiene el numero de bytes recibidos
int iter_serie; //numero de iteraciones que estamos haciendo para leer el puerto serie y ver si hay cambios

//Variables para generar el telegrama
int i,j,k;
int num_ceros_F,num_ceros_C;
int num_char_aux1,num_char_aux2;
int lado_actual,fila_actual,columna_actual;
char cabecera1[10]="STXMED_";
char cabecera2[10]="STXEND_";

char lado_A[5]="A_";
char lado_B[5]="B_";
char fila1[6]="R";
char columna1[6]="C";
char fila2[5]="_R";
char columna2[4]="C";
char cola[5]="ETX";
char telegrama[40]="";
char aux1[5]="";
char ceros[5]="000";

//Variables generales
char listo='Z'; //variable para comenzar o no la ejecución del programa
char data[10]; //Variable donde se guardarán los datos recibidos
int equipo_listo; //variable que me indica si el equipo VA61
int compara_recv; //variable para guardar la comparación de lo recibido con lo esperado
```

3.5.3. Preliminares.

Una vez realiza esta introducción, se va a explicar el desarrollo del programa. En primer lugar, una vez ejecutado el programa, se pregunta si desea empezar a que “corra”, o por el contrario desea cancelar su ejecución.

```
printf("Listo para ejecutar el programa. Pulse 's' para empezar o 'n' para finalizar.\n");
scanf("%c",&listo);
while(getchar()!='\n');
if(listo=='s' || listo=='S')
   {
```

if 3 1

"Se ha encontrado el siguiente error: " //si no somos ROOT mostrará este error
1 //terminará el programa mostrando el mensaje de error

if 1 //comprobamos que se abre correctamente

"Error al abrir el dispositivo serie: %s\n"
"OPEN"
0

0
0
0

while 1

"Esperando a que el equipo VA61 este listo...\n"

0
do

```
    //Controlo para las columnas el número de ceros y la longitud de "aux1" para que el telegrama quede
    //siempre de longitud constante
    if(columna_actual+1<10)
        {
           num_ceros_C=2;
           num_char_aux1=1;
        }
    if(columna_actual+1>9 && columna_actual+1<100)
        {
           num_ceros_C=1;
           num_char_aux1=2;
        }
    if(columna_actual+1>99)
        {
           num_ceros_C=0;
           num_char_aux1=3;
        }
     strncat(telegrama,columna2,1);
    strncat(telegrama,ceros,num_ceros_C);
    sprintf(aux1, "%d",columna_actual+1);
    strncat(telegrama,aux1,num_char_aux1);
}
```

Finalmente y en quinto lugar lo que se hace es añadir la cola.

```
strncat(telegrama,cola,3);
```

3.5.9. Envío del telegrama.

Una vez generado el telegrama lo que se va a hacer es enviarlo por el puerto serie para que así pueda recepcionarlo el sistema VA31 y seguidamente se muestra en pantalla el telegrama que se ha enviado.

```
serial_send(serial_fd, telegrama,30);
printf("Enviado telegrama:'%s'\n",telegrama);
```

A continuación se incrementa en dos el valor actual de columna (avanzamos dos posiciones) y se borran las variables auxiliares.

```
//Incremento la columna
columna_actual=columna_actual+2; //avanzo dos posiciones ya que tengo dos cámaras
//Borro las variables que tengo que volver a utilizar
strcpy(telegrama,"\0");
strcpy(aux1,"\0"); //puede que sobre
```

3.5.10. Espera de confirmación.

Lo siguiente que hay que hacer es leer la confirmación del equipo, pare ver si es un ACK o un NACK. Aquí surge un pequeño problema, y es el siguiente la longitud del ACK es distinta de la del NACK. Dicho de otro modo, si se le ordena que cuando reciba 4 byte pare de leer el puerto serie, cuando mande un ACK se esperaría siempre el timeout del puerto, ya que este mensaje no llega a 4 bytes. Este problema se solucionó realizando varias iteraciones con un timeout muy corto, por lo que si se recibe un ACK se esperará a que se agote el timeout, pero como es muy breve el usuario no lo percibe.

Al igual que antes, cuando se llega a un número máximo de iteraciones dado (tarda en llegar a ese número 60 segundos) es porque no se ha leído nada, por lo que se finaliza el programa mediante un error en pantalla que indica que el sistema VA31 no ha dado

confirmación. Si antes de llegar a ese número máximo de iteraciones se ha leído algo del puerto serie se sale del bucle, ya que eso indica el equipo VA31 ha contestado algo.

```
//Espero 60segundos a que me mande una señal el otro equipo
iter_serie=0;
n=0;
do
   {
      fflush(stdout); //limpiamos el buffer de salida
      n=serial_read(serial_fd,data,4,TIMEOUT);
      //Si llegamos al numero máximo de iteraciones permitidas terminamos la ejecución del programa
      if(iter_serie==5000)
         {
            printf("Tiempo máximo agotado. No se recibió ninguna respuesta del equipo VA61.\n");
            exit(0);
         }
      iter_serie++;
}while(n==0);
```

3.5.11. Acciones finales.

A continuación, se comprueba si es un ACK o un NACK. Si es un ACK se continúa con la ejecución del programa, poniendo a "1" a la salida en la línea digital valor de *comienza_medida*. Seguidamente se espera un tiempo razonable, por ejemplo de un segundo, y se vuelve a poner un "0" a la salida de la línea *comienza_medida*. Posteriormente se espera medio segundo para dar tiempo al sistema VA31 a que baje la línea *equipo_listo* y se vuelve al principio del *while* sin condición, para así generar y enviar el siguiente telegrama.

Si por el contrario, es un NACK se finaliza inmediatamente la ejecución del mismo y se le incida al usuario que ha habido un error de confirmación. Si se contestase con otra baliza que no se un ACK o un NACK, el programa lo reconoce y finaliza la ejecución del mismo mediante un código de error distinto a los anteriormente mencionado.

```
//Comprobamos que nos contestó el VA61
compara_recv=strcmp(data,"ACK"); //comprobamos si es un "ACK"
if(compara_recv==0) //se recibió un ACK
   {
      //Levantamos la línea "Comienza medida" a "1" durante un segundo
      comienza_medida=0x01; //asignamos a "Comienza medida" un "1"
      outb(comienza_medida,puerto_paralelo); //sacamos por el puerto paralelo el valor de "Comienza
      //medida"
      sleep(1); //esperamos un segundo
      comienza_medida=0x00; //asignamos a "Comienza medida" un "0"
      outb(comienza_medida,puerto_paralelo);
      //Ahora volvemos a comenzar y esperamos que el equipo VA61 halla terminado de procesar la
      //información
      usleep(5000); //espero medio segundo para que le tiempo al VA61 a bajar "Equipo listo" y así leer
      //correctamente la línea
   }
else
   {
      compara_recv=strcmp(data,"NACK"); //comprobamos si es un NACK
      if(compara_recv==0) //se recibió un NACK
         {
            printf("Comunicación terminada. Revise VA 61 y verifique que es lo que produjo el error\n");
            exit(0);
         }
      else //se recibió información inválida
         {
            printf("Error de confirmación\n");
```

```
                exit(0);
            }
    }
  } //fin del while sin condición
 } //fin del if
} //fin del main
```

CAPÍTULO 4

Desarrollo del sistema de adquisición

DESARROLLO DEL SISTEMA DE ADQUISICIÓN

1. Introducción.

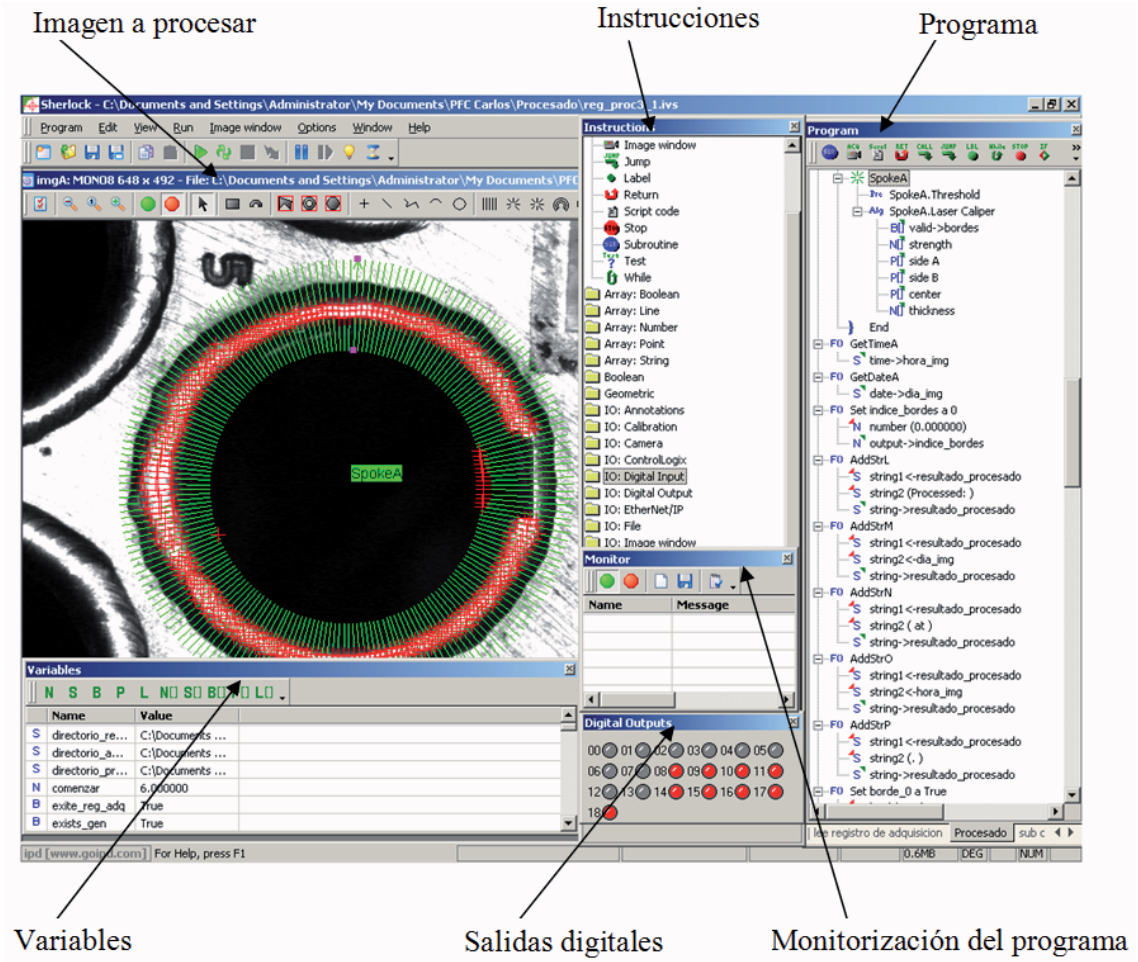
Una vez simulado el robot ABB necesario para la interactuación con el dispositivo VA31, se puede proceder a la adquisición de imágenes. Este capítulo se centra en esta parte de obtención de imágenes.

El objetivo de la adquisición es crear un registro en memoria, ordenado según una estructura, en la cual se almacenen en soporte digital todas las soldaduras que forman la placa para poder ser consultadas en cualquier momento.

Por otro lado se generará un registro, a modo de histórico, donde se irá reportando toda la información relacionada con la adquisición de imágenes, ya sea de las imágenes obtenidas o de los fallos ocurridos durante este proceso de adquisición.

2. Introducción al programa Sherlock.

Sherlock es un software de visión artificial avanzado y que puede configurarse fácilmente, debido al entorno gráfico bajo el cual está desarrollado este programa. El aspecto que presenta el programa es el siguiente:



En la ilustración anterior se aprecian las diferentes partes del programa, que son: ventana con la imagen a procesar, variables del programa desarrollado, instrucciones Sherlock, programa Sherlock y monitorización de las salidas digitales. A continuación se va a explicar cada una de ellas.

- ✓ Ventana con la imagen a procesar:

Esta ventana muestra la imagen sobre la que se está desarrollando un procesado concreto, así como el algoritmo que se está utilizando. Además, durante la ejecución del programa muestra en esta ventana en tiempo real el estado de la imagen. Dicha imagen se puede obtener mediante una cámara digital, o se puede cargar de memoria una imagen ya existente.

- ✓ Variables:

En esta ventana se aprecian las variables utilizadas en el desarrollo de un determinado programa. Concretando se muestran dos campos, que son, el nombre de la variable y su valor.

En la parte superior se aprecian las letras (en verde) de los tipos de variables que se pueden utilizar, siendo estos los siguientes:

- **N**: variable numérica.
- **S**: variable del tipo string.
- **B**: variable del tipo boleano, pudiendo valor solo *TRUE* o *FALSE*.
- **P**: variable del tipo gráfico. Representa un punto en la imagen sobre la que se está trabajando.
- **L**: variable del tipo string, en la cual sólo se permite grabar una línea de texto.
- **N[]**: array unidimensional de variables del tipo numérico.
- **S[]**: array unidimensional de variables del tipo string.
- **B[]**:array unidimensional de variables del tipo boleano.
- **P[]**:array unidimensional de variables que representan puntos en la imagen sobre la que se está trabajando.
- **L[]**:array unidimensional de variables del tipo línea de string.

Los tipos de variables que permite utilizar Sherlock sólo son los explicados anteriormente, no permitiendo utilizar ni crear ninguna variable de un tipo que no tenga predefinido este programa. Esto es un problema ya que, por ejemplo, no se pueden utilizar arrays de dos dimensiones. Además, es un lenguaje fuertemente tipado, por lo cual no permite bajo ningún concepto definir un tipo de variable y utilizarlo como otro sin haber una conversión de por medio.

- ✓ Instrucciones:

El abanico de instrucciones ofrecido por Sherlock es muy amplio. Este menú que, puede apreciar en la imagen anteriormente mostrada, proporciona un acceso rápido a las instrucciones de uso más común.

En este menú, las instrucciones se encuentran organizadas por categorías, las cuales se estructuran según el tipo, como se muestra resumidamente a continuación:

- Condicionales: If, If-else, jump, label, stop y while.
- Manejo de subrutinas: crear subrutina, return, stop, call subrutina.
- Manejo de arrays: boleenaos, líneas, numéricos, strings y points.
- IO: cámara, entradas y salidas digitales, calibración, anotación, manejo de ficheros, comunicaciones serie, comunicaciones TCP/IP, funciones del sistema, manejo de imágenes, Region of Interest (ROI), modbus Master y Slave, control a nivel de registros, Plot statistical data (SPC) y de teclado.
- Numéricas: suma, resta, valor absoluto, multiplica…
- Strings: concatena, añade a un string, longitud, extrae caracteres…
- Trigonométricas: seno, coseno, tangente, conversión de grados a radianes…
- Estadísticas: suma de los elementos de un array numérico, cálculo de medias, búsqueda de máximos…

Las funciones que ofrece Sherlock son numerosas, pero al estar predefinidas puede ocurrir que no exista la función deseada para un uso concreto. Esto se puede solucionar haciendo un programa específico, ya sea bajo javascript o desarrollando un nuevo entorno gráfico con nuevas funciones bajo el lenguaje Visual Basic.

✓ Programa desarrollado:

Esta ventana de programa muestra la secuencia de instrucciones o acciones que comprende un programa.

La programación es muy sencilla, como se detalla a continuación. Primero, se pincha en la barra de instrucciones la función que se desea utilizar, y seguidamente se arrastra hasta la ventana de programa, soldándola en el punto del código donde se quiere insertar. A continuación se introduce, ya sea manualmente o mediante una o varias variables alojadas en memoria, datos de entrada. Finalmente, y en el caso de que la función concreta devuelva algún parámetro, se le señala en qué variable se quiere guardar ese dato de salida.

El programa Sherlock permite el uso y la creación de subrutinas. Los datos de entrada necesarios para estas subrutinas se le pasan mediante una o varias variables globales que son comunes tanto al programa principal como a todas las subrutinas entre sí. Con los resultados devueltos por las subrutinas pasa los mismo, se devolverán por una o diversas variables globales. Como se puede observar, no existen parámetros de entrada ni de salida a la hora de interactuar con dichas subrutinas.

✓ Monitorización de las salidas digitales:

En esta ventana se aprecian las salidas digitales que están activas en cada momento. Las salidas digitales que están a "1" se muestran con una iluminación en color rojo, mientras que las que están a "0" se muestran en un color gris.

Esta ventana también permite activar o desactivar las salidas digitales sin necesidad de hacerlo mediante una instrucción, siendo tan sencillo como hacer doble clik sobre la

salida concreta, y si ésta estaba apagada se encenderá, o por si el contrario se encontraba encendida se apagará.

Ventana monitor:

Esta ventana muestra la información de la ejecución del programa. En detalle, si no ocurren errores va mostrando información de las instrucciones ejecutadas, y una vez llegado al final del programa, muestra el tiempo que ha durado la ejecución del programa. Si por el contrario, ocurren errores reportará en esta ventana qué error concreto ocurrió y en qué instrucción se produjo.

3. Interconexión con el robot ABB

Para realizar la adquisición es necesario conectar el dispositivo VA31 con el robot ABB, o en este caso con el computador que hace las veces de robot. Se necesita por una lado una conexión RS232 y por otro dos líneas digitales, una de lectura y otra de escritura.

3.1. Líneas digitales:

3.1.1 Lectura.

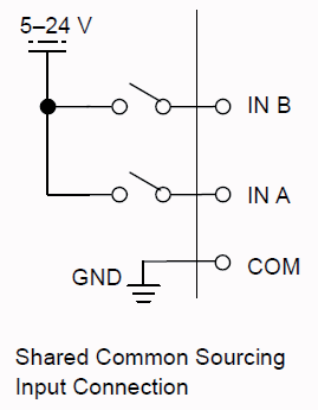
Se necesita una línea digital de lectura, la cual pueda leer del robot ABB la línea *comienza medida*. Para ello es necesario leer el manual del equipo VA31, donde se detalla toda la información de lectura de las líneas digitales.

En el manual de este dispositivo viene el conexionado con el que puede interactuar el usuario, habiendo sombreado las que se utilizarán para el desarrollo concreto [18].

Input Connector Pinout

Pin #	*Label*	*Description*
1	COM	Common for Trigger inputs
2	TRIG	Sensor Trigger input
3	COM	Common for Inputs 0 and 1
4	IN0	Input 0, general purpose input or decision trigger input
5	IN1	Input 1, general purpose input
6	COM	Common for Inputs 2 and 3
7	IN2	Input 2, general purpose input
8	IN3	Input 3, general purpose input
9	COM	Common for Solution inputs
10	SLN0	Solution select input 1, BCD value for current solution
11	SLN1	Solution select input 2, BCD value for current solution
12	SLN2	Solution select input 3, BCD value for current solution
13	ACC	Accept, Latch for current solution select
14	COM	Common for Reset
15	RST	Input reset, software configurable as part/soft/hard reset.

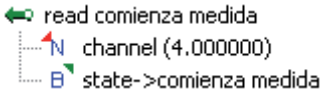
Una vez visto dónde se puede leer una línea digital, el siguiente paso es saber cómo. Para ello se consulta de nuevo el manual [18], donde se lee el siguiente esquema de conexión, ajustado al uso que se le va a dar en este proyecto:



Shared Common Sourcing
Input Connection

Como se observa, la salida digital del equipo está directamente conectada a la entrada del VA31, sin necesidad de ningún elemento entre medio.

Desde un punto de vista software su manejo es muy sencillo. Basta con incorporar esta función al programa y configurar dos parámetros, que son, el canal del que se va a leer y la variable booleana donde se va a guardar el valor leído, tal y como se muestra a continuación:



3.1.2 Escritura.

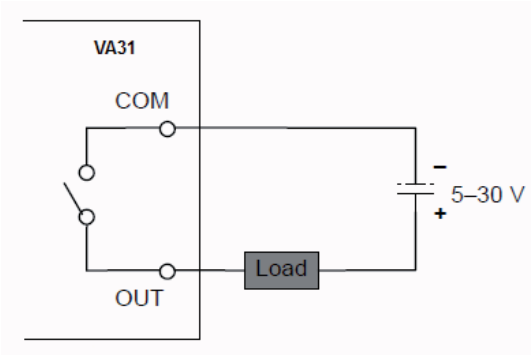
Por otro lado también es necesaria una línea digital de escritura, en la cual se le pueda escribir al robot ABB la línea *equipo listo*. Para ello, al igual que en el caso de lectura, es necesario leer el manual del equipo VA31, donde se detalla toda la información de lectura de las líneas digitales.

En concreto, se lee el siguiente conexionado, en el cual se han sombreado los pines utilizados en este diseño [18].

Output Connector Pinout

Pin #	*Label*	*Description*
1	24V	Power input supply voltage 24 V
2	GND	Power input ground 0V
3	ERTH	EARTH connection for system
4	COM	Common for Output 0 and 1
5	OUT0	Output 0, general purpose output
6	OUT1	Output 1, general purpose output
7	COM	Common for Output 2 and 3
8	OUT2	Output 2, general purpose output
9	OUT3	Output 3, general purpose output / Running status
10	COM	Common for PASS/FAIL/RECYCLE outputs
11	PASS	Dedicated PASS output
12	REC	Dedicated RECYCLEoutput
13	FAIL	Dedicated FAIL output
14	COM	Common for Strobe
15	STRB	Strobe output

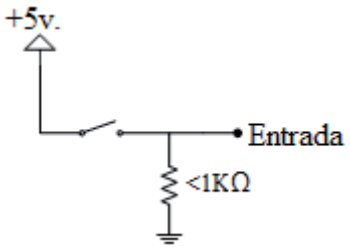
Una vez visto en qué pin se puede escribir, el siguiente paso es saber cómo se hace. Para ello se consulta de nuevo el manual [18], donde se lee el siguiente esquema de conexión:



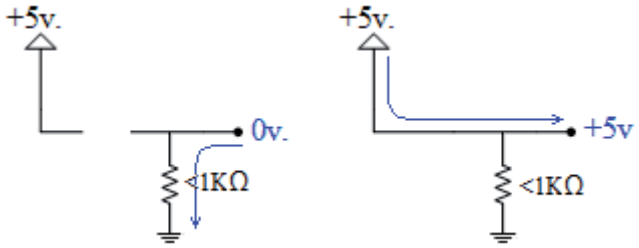
Obsérvese que el dispositivo VA31 cuando escribe en la línea digital no da tensión a la salida, sino que se limita a hacer un contacto entre los pines *COM* y *OUT*, no pudiéndose realizar una conexión directa entre el robot y el VA31. Es tarea del usuario, a partir de este contacto, adaptar esta salida a su diseño en concreto. Es importante destacar que la máxima intensidad que permite conmutar es de *300mA*, como indica el manual del dispositivo.

Es necesario que al robot ABB le legue un "1" o un "0", es decir, que cuando el dispositivo VA31 active su salida le llegue al robot un "1", en vez de un cortocircuito, y cuando el VA31 desactive salida digital le llegue al robot un "0", en vez de un circuito

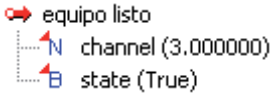
abierto. Por ello se ha diseñado un pequeño circuito con resistencias en configuración *pull-down*, como se muestra a continuación:



Es válido cualquier valor de resistencia mayor a 1KΩ. En cuanto al funcionamiento es muy sencillo, si el interruptor está abierto le llegará un "0" al robot, y si está cerrado le llegará un "1", como se aprecia en la siguiente imagen ilustrativa:



Desde un punto de vista software su manejo es también sencillo. Es suficiente con incorporar esta función al programa y configurar dos parámetros, que son, el canal en el que se va a escribir y la variable boleana que se desea escribir, tal y como se muestra a continuación:



3.2. Puerto serie:

El puerto serie que posee el dispositivo VA31 es un puerto estándar DB9 con terminación en macho. Para interconectar el robot con el VA31 basta con conectar en cada extremo el cable fabricado y que se explica en el capítulo tercero.

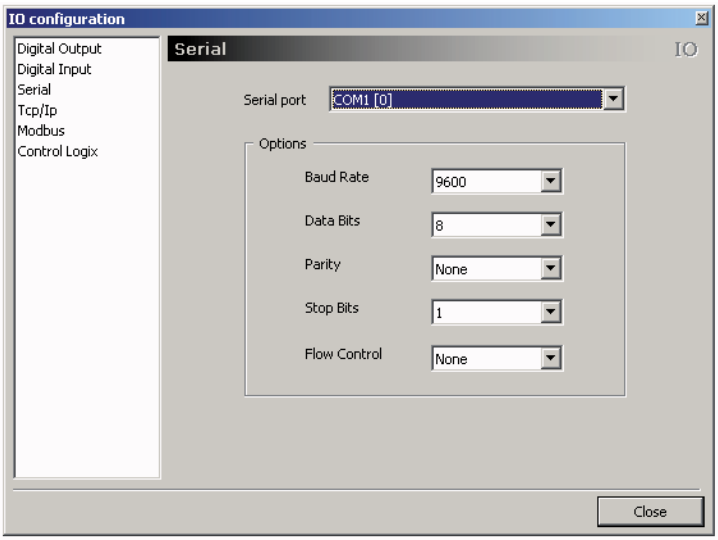
No hay que olvidar que el protocolo de comunicación que está por encima es el RS232, y la gestión de dicho protocolo se gestiona desde Sherlock.

Antes de nada, hay que configurar el puerto serie a utilizar con la misma configuración, que lo tiene el puerto serie del computador que realiza las veces de robot. Para ello, en el menú de opciones se halla la configuración de dicho puerto, donde se pueden modificar sus propiedades.

Todos los parámetros de configuración posibles se encuentran ya predefinidos, haciendo así muy sencilla su configuración, ya que basta con coger de la lista de valores que ofrece el programa el que más se ajusta a la necesidad en concreto.

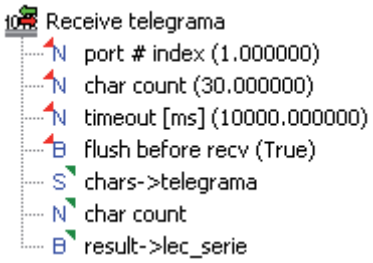
El primer parámetro es el puerto serie, pudiendo escoger entre el *COM1[0]* o el *COM4[1]*. El segundo es la velocidad, que puede variar entre 110 baudios y 256000 baudios. El tercer parámetro es el número de bits de datos, cuyo valor puede ser 5, 6, 7 u 8. El Cuarto es la paridad, que puede ser par, impar o señal. El quinto son los bits de stop, que pueden ser 1, 1.5 o 2. El último parámetro es el control de flujo, que puede ser hardware o Xon/Xoff.

Los parámetros con los que se ha configurado este puerto serie son los que se muestran a continuación:



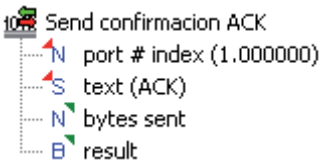
Una vez configurado se van a ver las dos instrucciones que permiten leer y escribir del puerto serie.

La lectura serie se realiza con la siguiente función, que tiene como parámetros de entrada el número de puerto del que se va a leer, el número de caracteres que espera recibir, el time out el cual una vez transcurrido detendrá la lectura de este puerto, y hará un borrado del buffer de lectura del mismo. En cuanto a los parámetros de salida se tiene la línea recibida, el número de caracteres recibidos y el resultado de lectura correcta, pudiendo valer *TRUE* o *FALSE*. A continuación se muestra una imagen de dicha instrucción:



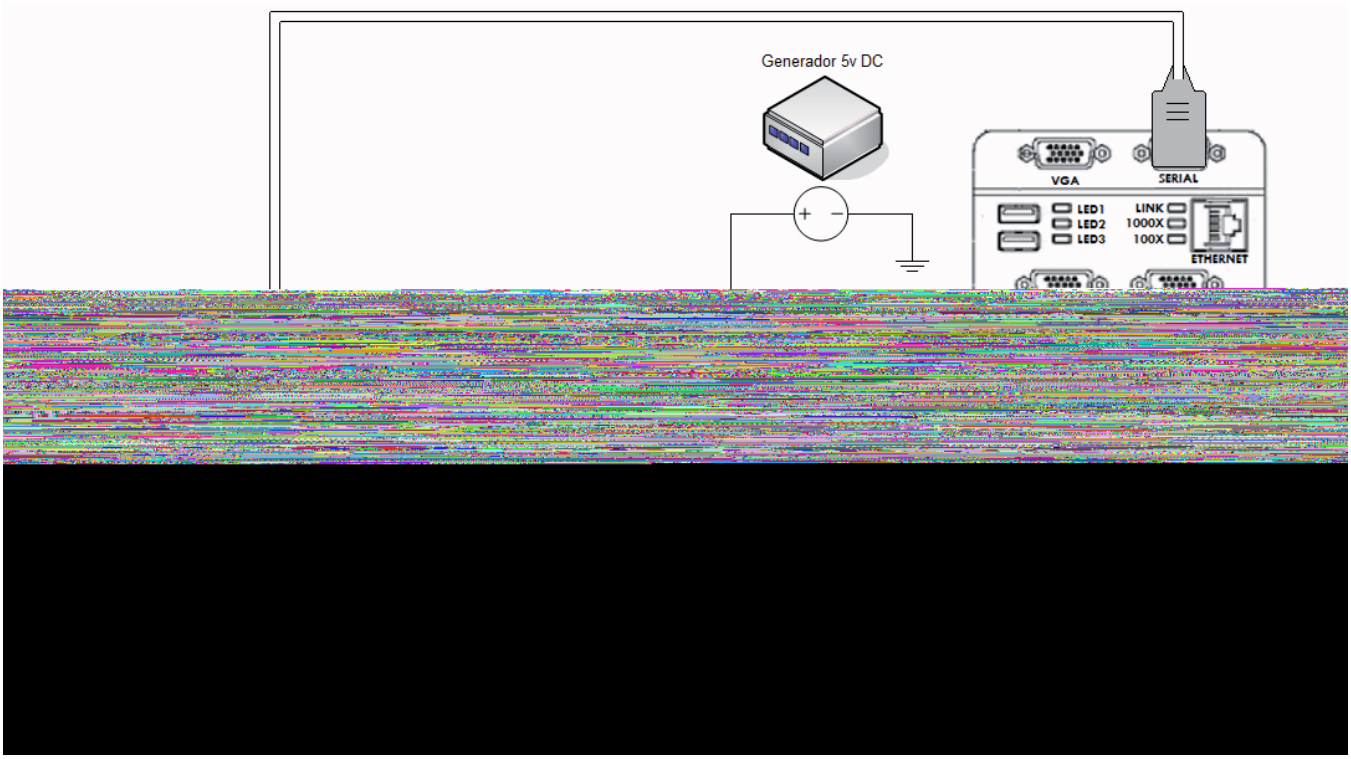
En cuanto a la escritura se utiliza otra instrucción, la cual tiene como parámetros de entrada el índice de puerto a escribir y los datos a enviar, y como salida el número de

bytes enviados y el resultado de escritura correcta, pudiendo valer *TRUE* o *FALSE*. A continuación se muestra una imagen de dicha instrucción:



3.3. Esquema general:

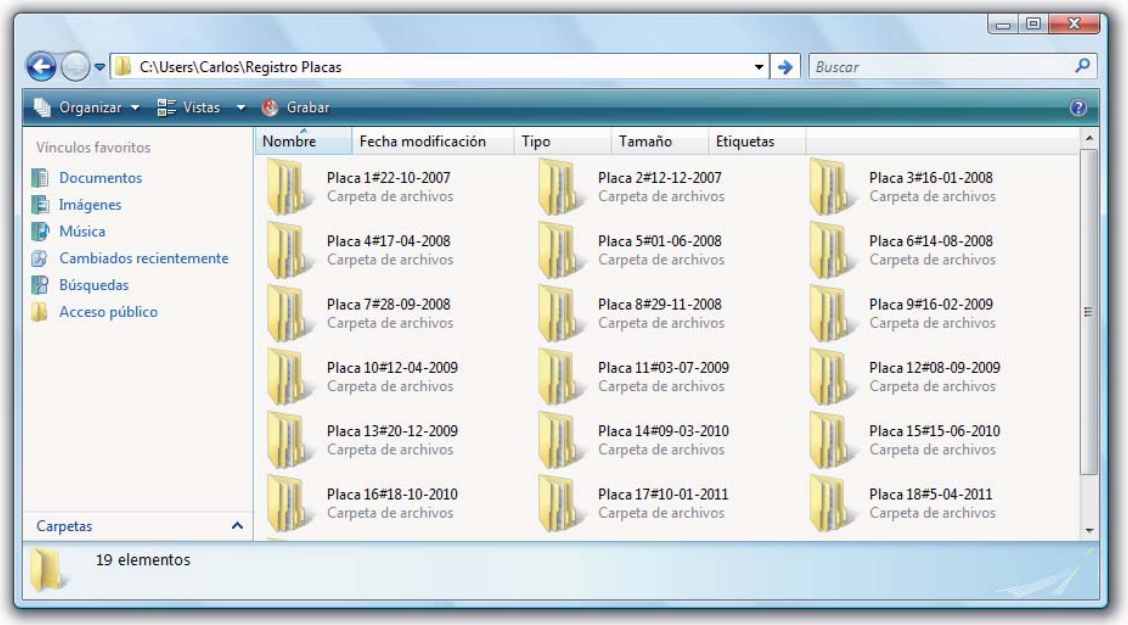
A continuación, se muestra un esquema general de cómo ha quedado la interconexión, tanto de los puertos serie como de las líneas digitales. Como comprobación de que por las líneas digitales hay señal, se han añadido dos diodos LED, uno de color verde que indica lectura de la línea digital por parte del dispositivo VA31 y otro de color rojo que indica la escritura del mismo, cada uno con una resistencia de 150Ω para evitar dañarlos. También se observa la fuente de alimentación necesaria para la configuración de resistencias *pull-down*.



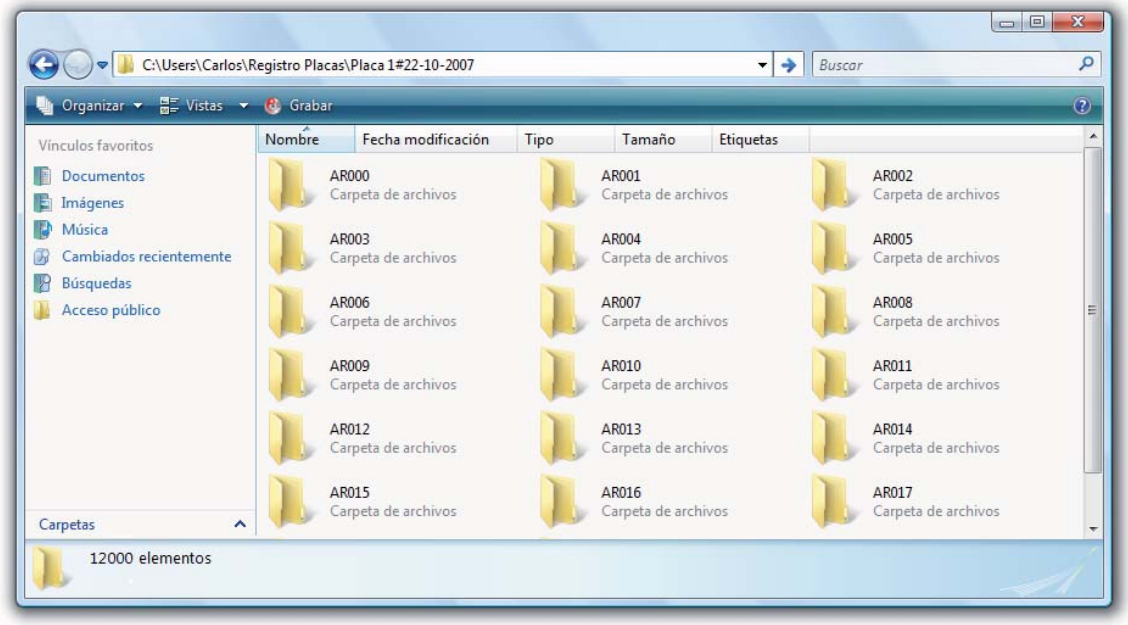
4. Guardado de imágenes en disco.

El equipo VA31 adquiere una imagen con cada una de las dos cámaras que posee y las almacena en disco. El directorio necesario para la adquisición de las imágenes de un generador de vapor determinado se llama *directorio generador*, y es el que posee todos los subdirectorios necesarios para la adquisición ordenada de las imágenes de ese generador. El directorio raíz es el que contiene todos los directorios generadores.

Por ejemplo, si el directorio raíz es *C:\Users\Carlos\Registro Placas*, lo que se tendría dentro de él serían todos los directorios generadores, representando cada unos de ellos a una placa concreta, la cual se ha podido adquirir ya o se adquirirán en un futuro.



Dentro del mencionado directorio generador se encuentran todas las carpetas correspondientes a cada una de las filas que posee la placa a fotografiar. De acuerdo con el ejemplo anterior, dentro de la carpeta del primer generador, llamado *Placa 1#22-10-2007*, se verían las siguientes carpetas correspondientes a cada fila, como se explica más adelante:



Todos estos directorios tienen que existir de antemano para el correcto funcionamiento del programa desarrollado, siendo responsabilidad del usuario su creación. En caso de que no exista alguno en concreto, el programa de adquisición lo indicará.

En cuanto a la adquisición, todas las imágenes se almacenan en el directorio anteriormente mencionado. El nombre de los directorios que posee el generador se genera de la siguiente manera:

LR□□□

Donde:

- L indica el lado que se esta procesando, pudiendo ser "A" o "B".
- R□□□ indica la fila a la que pertenece cada soldadura.

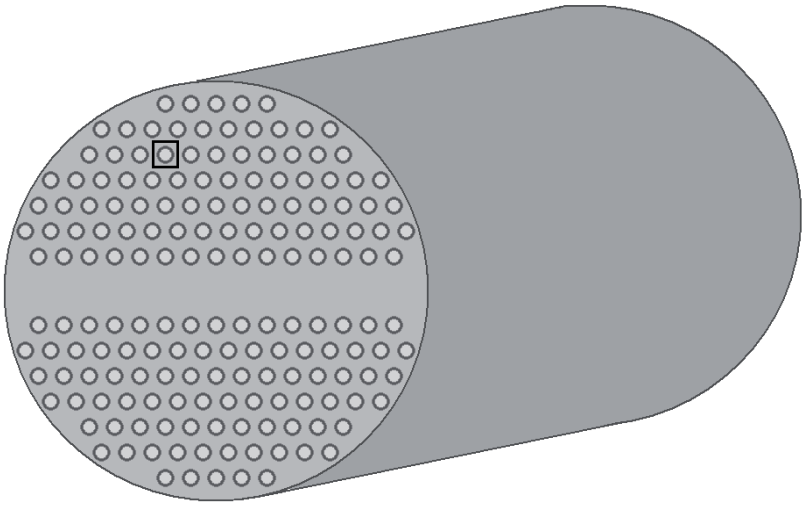
En lo que corresponde al nombre de la imagen, se genera de la siguiente manera:

LR□□□C □□□

Donde:

- L indica el lado que se esta procesando, pudiendo ser "A" o "B".
- R□□□ indica la fila a la que pertenece cada soldadura.
- C□□□ indica la columna a la que pertenece cada soldadura.

A continuación, se va a ver en ejemplo para ilustrar cómo se genera el nombre de la carpeta y el de la imagen. Para ello se tiene la siguiente soldadura enmarcada con un recuadro negro, correspondiente a la fila tres columna cuatro, como muestra a continuación:



De acuerdo con la explicación anterior, el nombre de la carpeta donde se guardará la imagen será el *LR003*, y el nombre de la imagen en concreto el *AR003C004*.

Es inevitable que alguna soldadura sea adquirida más de una vez, así que en caso de ocurrir esto el fichero que ya exista no se sobrescribirá y se despreciará la adquisición.

Si el operador desea adquirir una imagen nueva, deberá borrar el fichero antiguo manualmente, y luego realizar una nueva adquisición.

5. Funcionalidades del programa de adquisición.

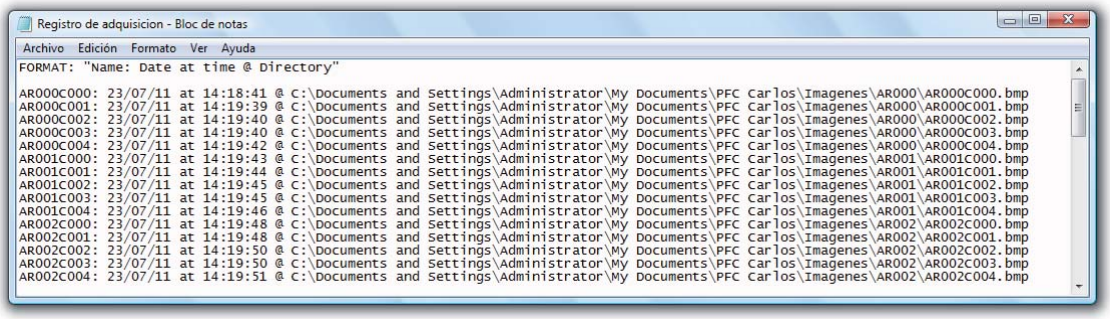
Durante la ejecución del programa de adquisición se va generando un registro, llamado *Registro de adquisición.txt*, en el que se reporta la información básica de las fotos adquiridas y los fallos ocurridos. En dicho registro se deja todo lo ocurrido durante la adquisición reflejado en el orden que acaeció, de forma tal que se queda constancia escrita del histórico de adquisición, con fechas, directorios y errores para su posterior consulta.

Ahora se van a explicar los dos tipos de entradas posibles en el registro de adquisición más a fondo, estudiando los diferentes casos que se pueden dar en cada uno de ellos:

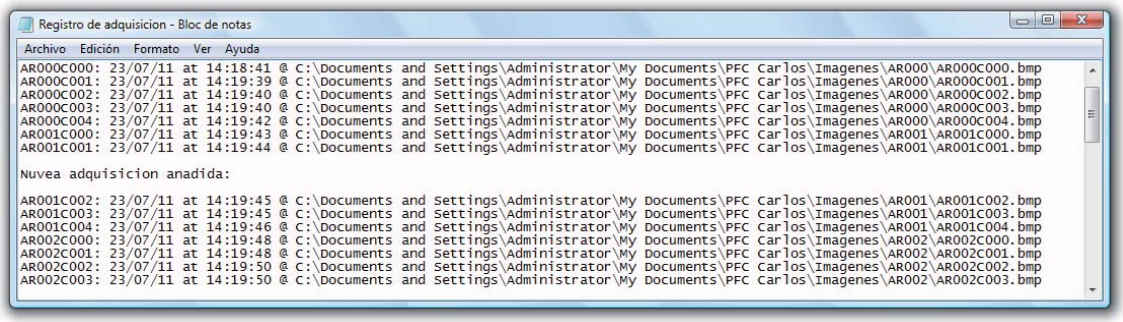
- Adquisición ocurrida sin errores:

Una vez adquirida una foto con éxito se añade al registro una línea de texto, la cual contiene el nombre de la foto, día y hora de adquisición y directorio donde se guardó.

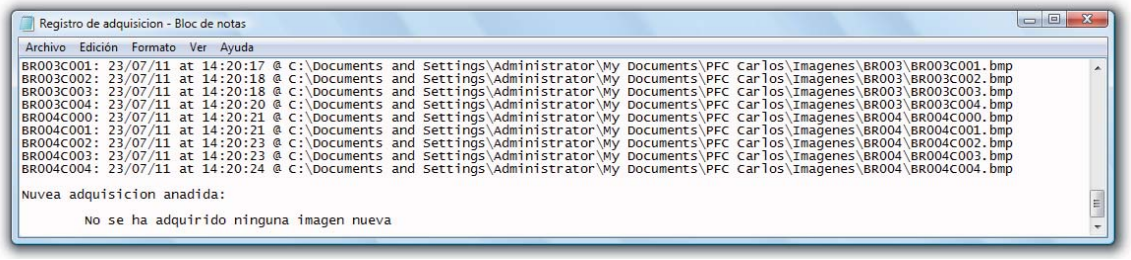
Si no hay ninguna foto guardada se inserta al principio del registro una cabecera que indica el formato que tiene el registro y se procede a la adquisición, dejando en dicho registro la información de la foto adquirida, tal y como se muestra a continuación:



Si la adquisición ya está comenzada se notificará en el registro y se continuará con la adquisición, como se muestra a continuación:



Si la adquisición ya está empezada pero no se adquiere ninguna imagen nueva por encontrarse ya todas guardadas en disco, se notificará en el registro de la siguiente manera:



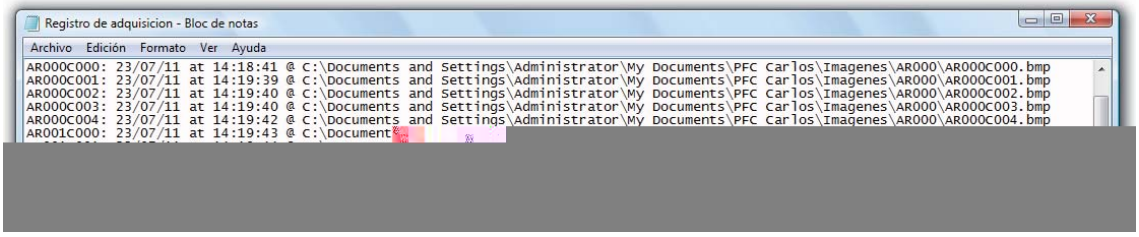
- Aparición de errores durante la adquisición:

Los errores pueden ocurrir por diversas causas. En el momento de ocurrir un error se añaden dos líneas de texto al registro. La primera indica que la adquisición ha fallado, junto con el día y la hora a la que se produjo el error. La segunda línea es específica de cada fallo, y es la que indica el tipo de error ocurrido.

A continuación se va a explicar cada uno de ellos, así como causa que lo genera:

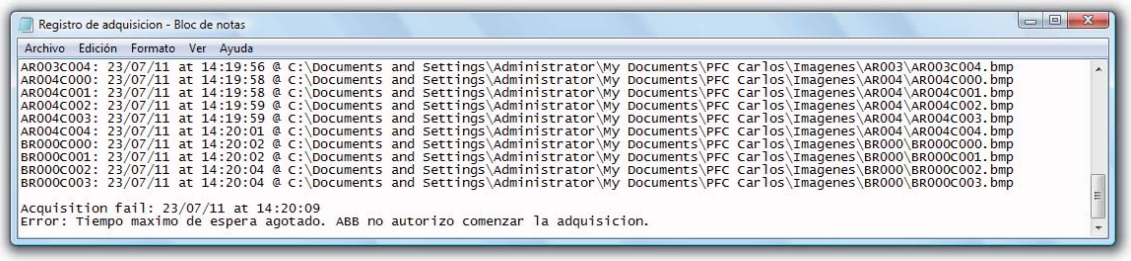
- Error en el puerto serie:

Ocurre cuando transcurrido un tiempo *time out*, determinado por el usuario, no se lee nada del puerto serie. Este error detiene la ejecución del programa, y una vez insertado tiene el siguiente aspecto:



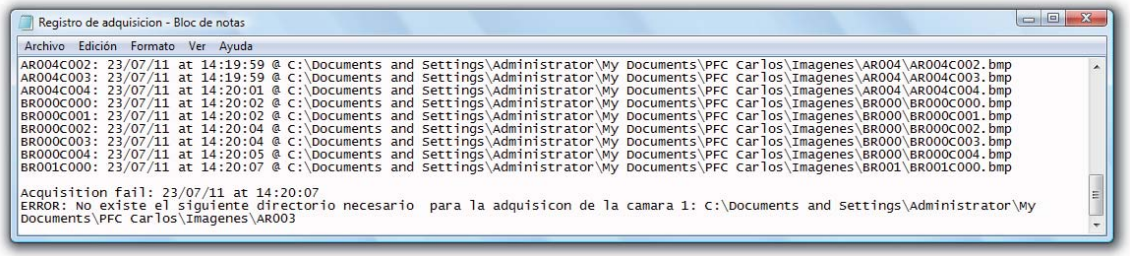
- Error en la línea digital:

Ocurre cuando el ABB no da el "OK" para comenzar la adquisición. El equipo VA31 espera durante un tiempo *time out*, determinado por el usuario, a que le dé luz verde el ABB para comenzar la adquisición, y si una vez transcurrido ese tiempo no ha dado su confirmación se detiene la adquisición y se inserta el siguiente error:



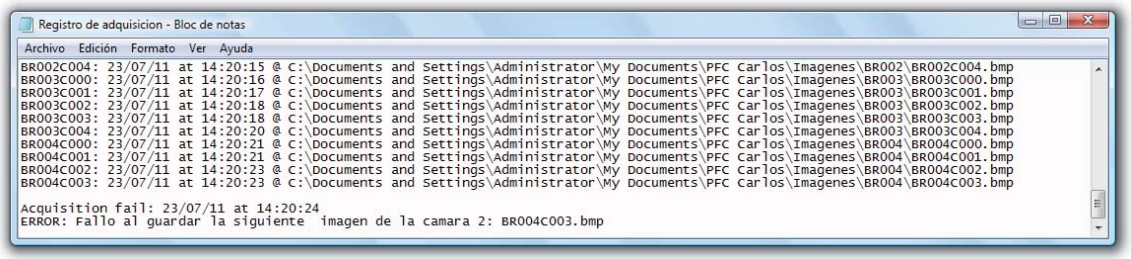
- Error de directorio:

Ocurre cuando no se encuentra el directorio de alguna de las dos cámaras. Si se intenta acceder a una carpeta que no existe, ya sea con la cámara 1 o la cámara 2, se añadirá el error mostrado a continuación al registro. Cuando ocurre este tipo de fallos no se detiene la adquisición, sino que se le notifica al usuario y al registro y se pasa a la siguiente imagen a adquirir.



- Error al guardar:

Ocurre cuando no se pude guardar una foto. Este error no es producido por no existir un directorio concreto, ya que para eso está el error anteriormente explicado. El principal motivo por el cual se puede dar este error es porque durante el proceso de grabación Sherlock detecte algún problema. Una vez ocurrido este fallo se notifica al usuario y al registro y se pasa a la siguiente imagen a adquirir, no deteniendo en ningún caso la ejecución del programa. El mensaje de error es el siguiente:



6. Estructura del programa de adquisición.

El programa de adquisición desarrollado está compuesto por un programa principal y tres subrutinas, creando así una jerarquía de dos niveles.

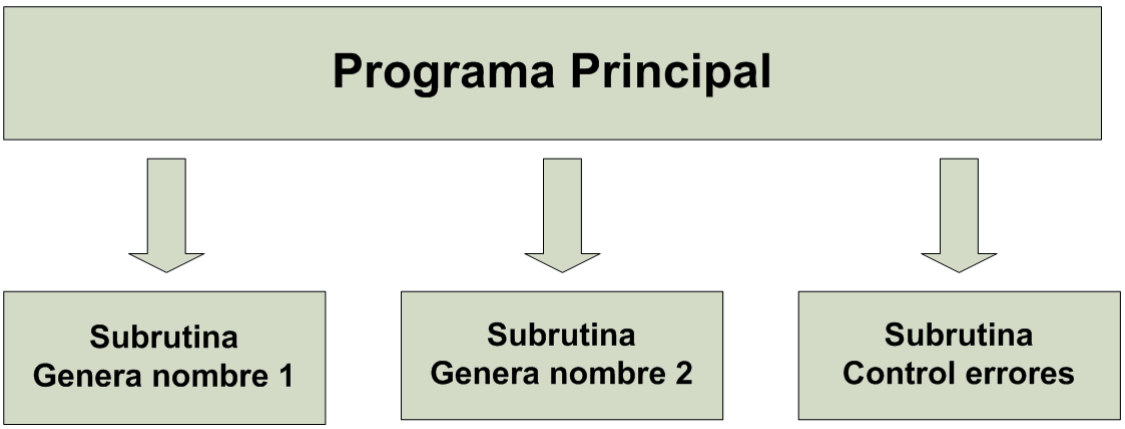
En un primer nivel se encuentra el programa principal, que es el que coordina e invoca a las subrutinas de un segundo nivel. Este programa principal se encarga de realizar la

adquisición, así como el manejo de E/S y, a partir de un error ocurrido, detener el programa de adquisición o no.

Por debajo, en un segundo nivel hay tres subrutinas, dos de ellas encargadas de generar el directorio donde se guardarán las imágenes y reportar al registro de adquisición la información de la captura de imágenes, y una tercera que añade al registro de adquisición los errores ocurridos.

Estas subrutinas de segundo nivel no tienen la autoridad para cancelar la ejecución del programa, se limitan a recibir los parámetros de entrada, procesarlos y retornar el o los resultados oportunos.

A continuación se muestra una imagen ilustrativa que representa la estructura del programa desarrollado:



Otro aspecto importante a comentar son las variables compartidas. Por ello, en esta última parte de este presente capítulo se van a explicar las variables que comparte el programa principal con cada una de las tres subrutinas existentes, así como de su nombre, tipo y descripción.

- Programa principal y Subrutina *Genera nombre 1*:

  - ✓ *directorio*: variable del tipo string que contiene el directorio del generador introducido por el usuario.
  - ✓ *telegrama*: variable del tipo string que contiene el telegrama que se recibió del robot ABB.
  - ✓ *nom_img1*: variable del tipo string que contiene el nombre completo del directorio donde se debe guardar el nombre de la imagen correspondiente a la primera cámara.
  - ✓ *dir_entrada_img1*: variable del tipo string que contiene la información correspondiente a la adquisición de la imagen tomada con la primera cámara.

- Programa principal y Subrutina *Genera nombre 2*:

  - ✓ *directorio*: variable del tipo string que contiene el directorio del generador introducido por el usuario.

- ✓ *telegrama*: variable del tipo string que contiene el telegrama que se recibió del robot ABB.
- ✓ *nom_img2*: variable del tipo string que contiene el nombre completo del directorio donde se debe guardar el nombre de la imagen correspondiente a la segunda cámara.
- ✓ *dir_entrada_img2*: variable del tipo string que contiene la información correspondiente a la adquisición de la imagen tomada con la segunda cámara.

- Programa principal y Subrutina *Control errores*:

  - ✓ *ctrl_errores*: variable del tipo numérico que contiene el código de error que se produjo.
  - ✓ *reg_errores*: array de strings que contiene el texto de los diferentes errores que puede producir.

7. Programa de adquisición.

Este último apartado del cuarto capítulo se va a centrar en explicar cada una de las subrutinas desarrolladas así como del programa principal necesarios para la adquisición.

Para facilitar la comprensión del programa se van a empezar a explicar las tres subrutinas, y a continuación el programa principal.

7.1. Subrutina Genera nombre 1:

Esta subrutina es la que se encarga de generar para la imagen adquirida con la primera cámara, por un lado el nombre completo del directorio a guardar, y por otro, de generar la línea de información asociada a la adquisición que se añadirá al *Registro de adquisición.txt*.

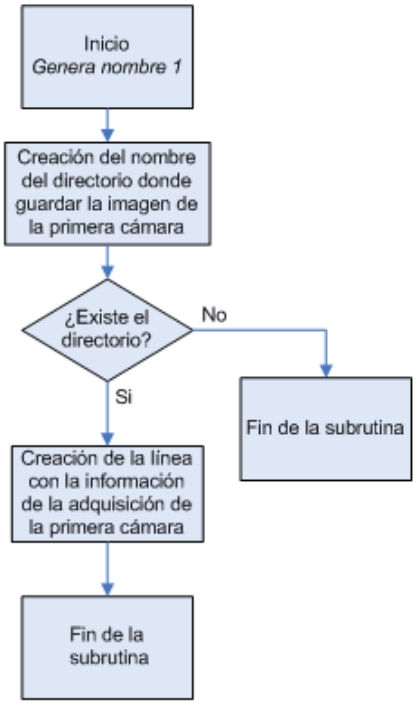
Las variables que recibe son las que se muestran a continuación:

- ✓ *directorio*: variable del tipo string que contiene el directorio del generador introducido por el usuario.
- ✓ *telegrama*: variable del tipo string que contiene el telegrama que se recibió del robot ABB.

Las variables que devuelve una vez ha terminado su ejecución son las siguientes:

- ✓ *nom_img1*: variable del tipo string que contiene el nombre completo del directorio donde se debe guardar el nombre de la imagen correspondiente a la primera cámara.
- ✓ *dir_entrada_img1*: variable del tipo string que contiene la información correspondiente a la adquisición de la imagen tomada con la primera cámara.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada la subrutina,

- Lo primero que se hace es inicializar una variable, llamada *nom_img1* y que es la que contendrá el nombre del directorio de la imagen adquirida por la primera cámara, con el directorio de generador introducido, seguido por una contrabarra.

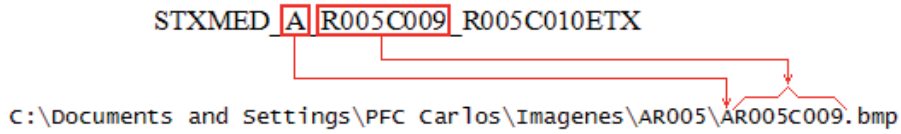
- Se extrae del telegrama recibido el carácter que está en la séptima posición del mismo, que se corresponde con el lado que se está adquiriendo, así como los que están en las posiciones nueve, diez, once y doce, y que son los correspondientes a la fila.

STXMED_A_R005C009_R005C010ETX STXMED_A_R005C009_R005C010ETX

- Se añaden todos ellos a *nom_img1*, representando así el nombre de la carpeta donde se almacenan todas las imágenes que comprende dicha fila.

```
C:\Documents and Settings\PFC Carlos\Imagenes\AR005
```

- Se comprueba que el directorio donde se debe salvar la imagen existe. En caso de no existir, se volverá al programa principal, indicándole en una variable que este directorio no existe. En el supuesto contrario, se sigue generando el nombre del directorio. Para ello se añade otra contrabarra y se extrae el séptimo carácter seguido por el noveno, décimo, decimoprimero, decimosegundo, decimotercero, decimocuarto, decimoquinto y decimosexto, añadiendo al final *.bmp* y agregando todo ello al contenido de *nom_img1*, como se muestra a continuación:



- Una vez terminado el directorio de donde se va a guardar la imagen de la primera cámara, el siguiente paso es generar la línea de información de esta adquisición. Para ello se inicializa en blanco una variable llamada *entrada_img1*, y que es la que contendrá dicha información asociada. Seguidamente se le añade la referencia a la foto que representa seguida por dos puntos, como se muestra a continuación:

```
AR005C009:
```

- Se realiza una llamada a dos funciones que tiene Sherlock implementadas. Una que permite conocer la fecha actual y la otra la hora. Se añaden estos datos a la línea de información, separando la fecha de la hora por un "at", como se muestra a continuación:

```
AR005C009: 23/07/11 at 14:19:38
```

- Se incorpora el símbolo "@" seguido por el directorio donde se guardo dicha imagen, dando así por finalizada la construcción de esta línea de registro.

```
AR005C009: 23/07/11 at 14:19:38 @ C:\Documents and Settings\PFC Carlos\Imagenes\AR005\AR005C009.bmp
```

- Antes de finalizar, es necesario reemplazar el carácter "\" por "\\", ya que el programa cuando la guarda en el archivo sustituye la doble contrabarra por la contrabarra simple, debido a caracteres reservados como el *\n* o *\t*. Una vez realizado esto, se añade al final de la línea un retorno de carro y se vuelve a programa principal.

```
AR005C009: 23/07/11 at 14:19:38 @ C:\\Documents and Settings\\PFC Carlos\\Imagenes\\AR005\\AR005C009.bmp\n
```

7.2. Subrutina Genera nombre 2:

Esta subrutina es análoga a la anterior, y es la encargada de generar, para la imagen adquirida con la segunda cámara, el directorio a guardar esta imagen y de generar la línea de información asociada a la adquisición que se añadirá al *Registro de adquisición.txt*.

No es válido utilizar la subrutina Genera nombre 1 ya que los índices para acceder el telegrama y leer el nombre de la imagen a adquirir son distintos.

Las variables que recibe son las que se muestran a continuación:

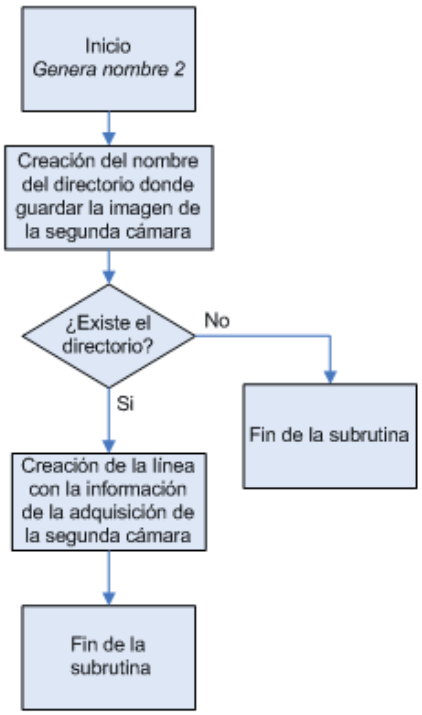
- ✓ *directorio*: variable del tipo string que contiene el directorio del generador introducido por el usuario.

- ✓ *telegrama*: variable del tipo string que contiene el telegrama que se recibió del robot ABB.

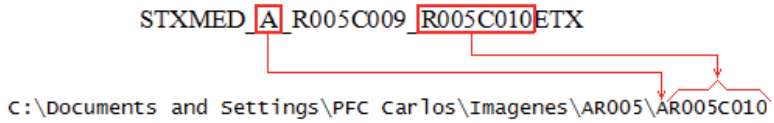
Las variables que devuelve una vez ha terminado su ejecución son las siguientes:

- ✓ *nom_img2*: variable del tipo string que contiene el nombre completo del directorio donde se debe guardar el nombre de la imagen correspondiente a la segunda cámara.
- ✓ *dir_entrada_img2*: variable del tipo string que contiene la información correspondiente a la adquisición de la imagen tomada con la segunda cámara.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:

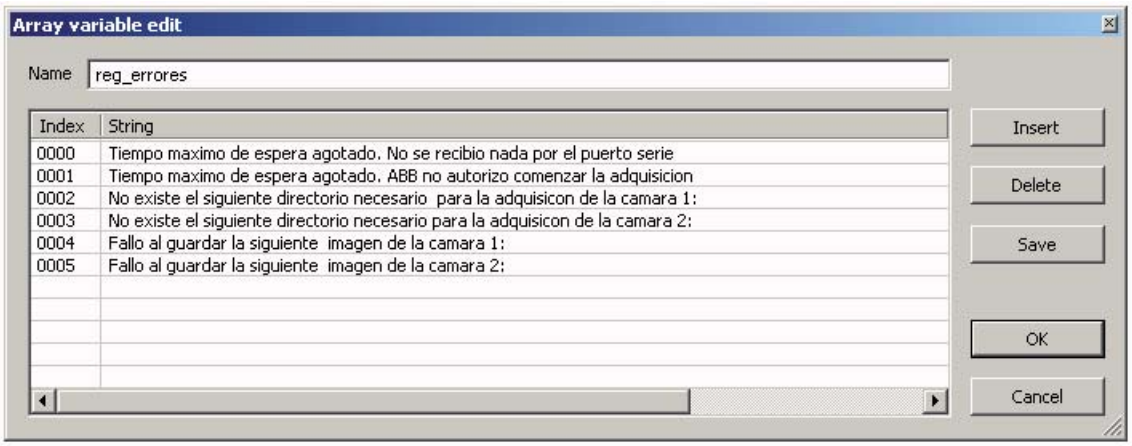


El comportamiento de esta subrutina es idéntico al anterior. Únicamente cambia la información de la cámara que se lee del telegrama, que en este caso es la posición de la segunda, generándose así el directorio de forma análoga al de la anterior subrutina, tal y como se muestra a continuación:

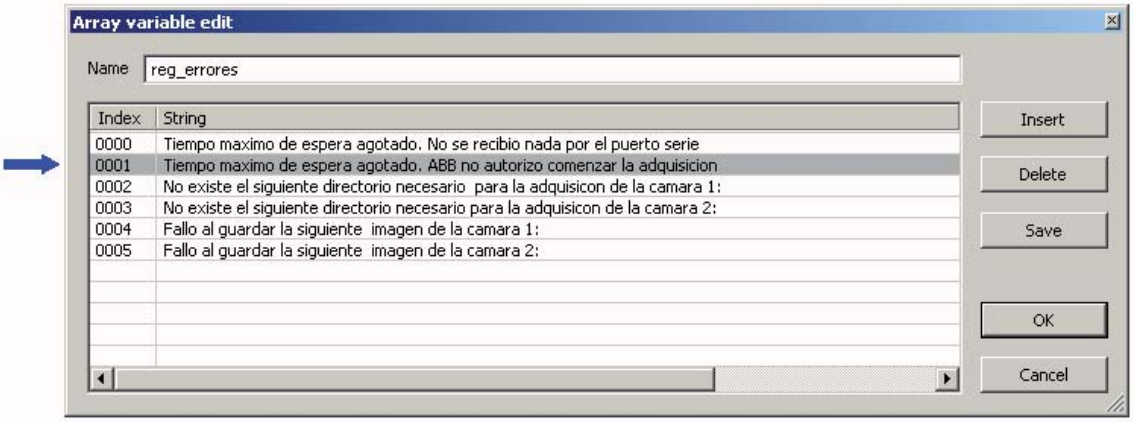


7.3. Subrutina Control errores

Esta subrutina es la encargada de gestionar todos los errores que puedan ocurrir durante la adquisición. Para ello hay un registro cargado en memoria con los errores soportados. Dicho registro tiene seis entradas, como se muestra a continuación:



Si ocurre un error, el programa principal le pasará a esta subrutina un código en una variable, representado este código el tipo de error surgido. Si por ejemplo no se leyó nada de la línea digital, el programa principal llamará a esta subrutina con el código de invocación valiendo 1, indicándole así que el fallo se produjo al leer de la línea digital.



Esta subrutina, una vez ocurrido un fallo, genera dos líneas de error en el registro de adquisición. La primera indica que ha ocurrido un fallo, seguido por la fecha y la hora. La segunda indica el fallo en concreto, que es el que obtiene del registro de fallos a partir del código de error.

Es importante destacar que esta subrutina, ante un fallo, no para la ejecución del programa, sino que se limita a reportarlo en el registro y dejar así constancia del mismo. Es tarea del programa principal, ante un fallo, detener o no la ejecución del programa de adquisición.

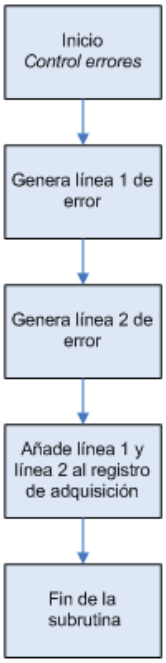
Las variables que recibe son las que se muestran a continuación:

- ✓ *ctrl_errores*: variable del tipo numérico que contiene el código de error que se produjo.

- ✓ *reg_errores*: array de strings que contiene el texto de los diferentes errores que puede producir.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada esta función,

- Se genera la primera línea de error. Para ello se inicializa la variable *error_linea_1* con el texto "Acquisition fail". A continuación, mediante una función de Sherlock, se obtiene el día actual, y se añade a esta línea, seguido por un "at" y finalizada por la hora en la que se produjo el error, obteniéndose así la primera línea:

```
Acquisition fail: 23/07/11 at 14:18:42
```

- Se genera e inicializa la segunda línea de error con el texto "Error: ". y se carga del registro de errores la entrada de error que indica el código con el que invocó el programa principal a esta subrutina.

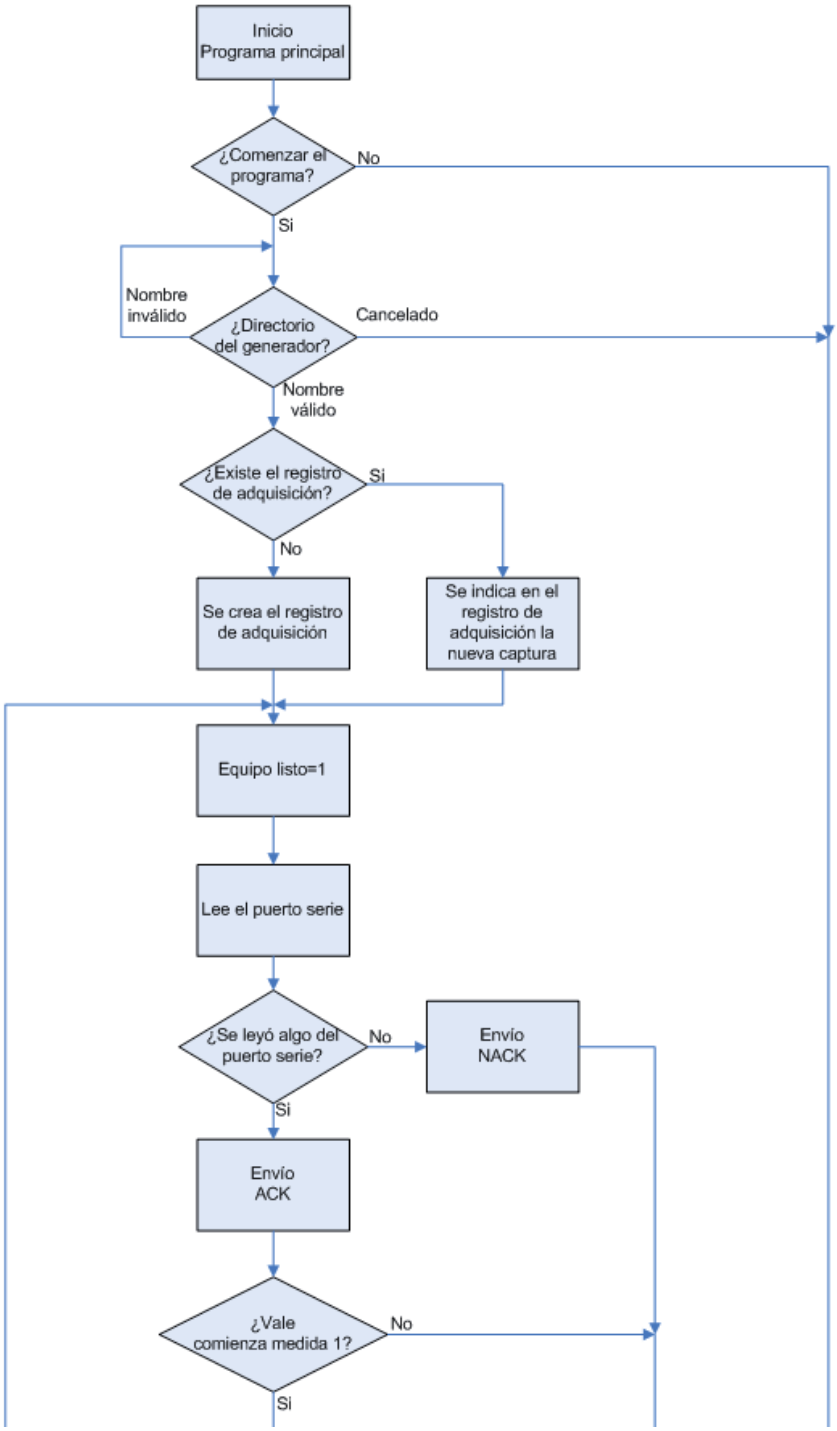
- Finalmente se añaden estas dos líneas al registro de adquisición, como se muestra a continuación, y se termina la ejecución de esta subrutina.

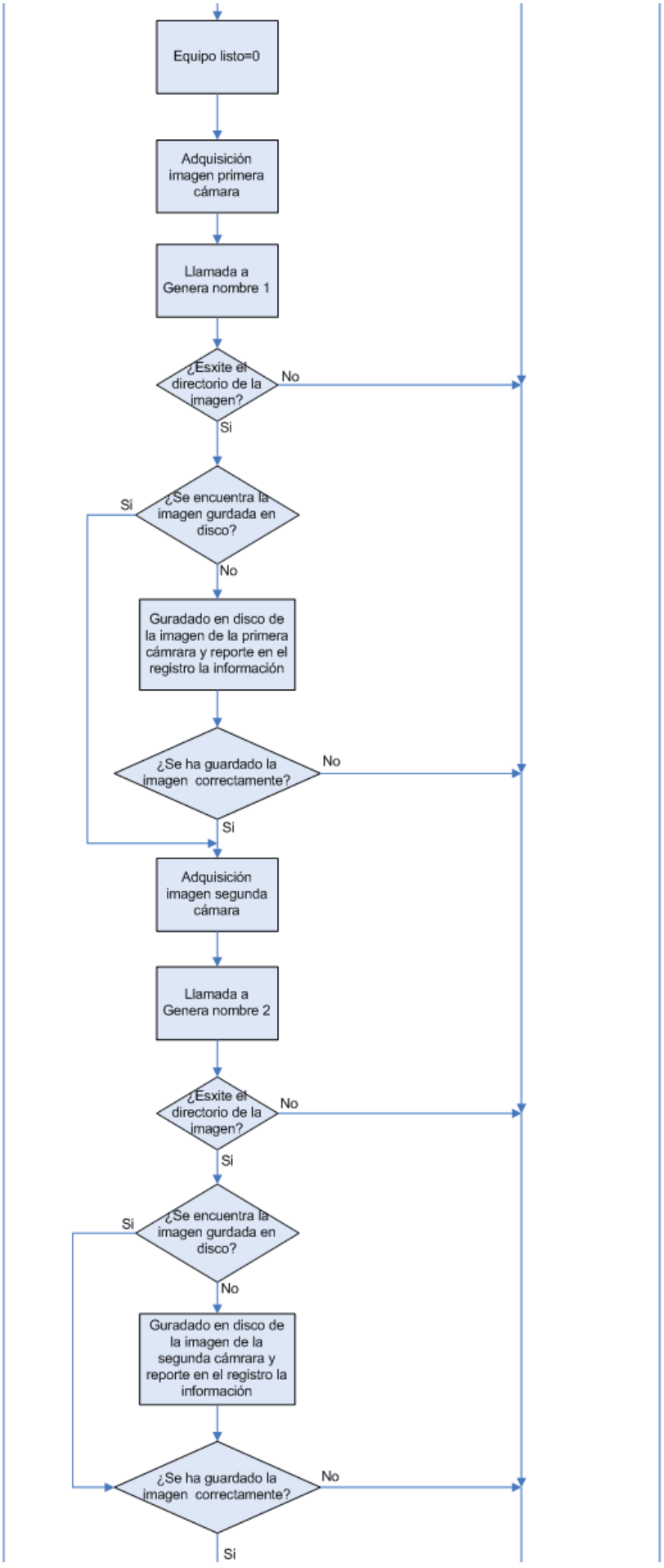
```
Acquisition fail: 23/07/11 at 14:18:42
Error: Tiempo maximo de espera agotado. ABB no autorizo comenzar la adquisicion.
```

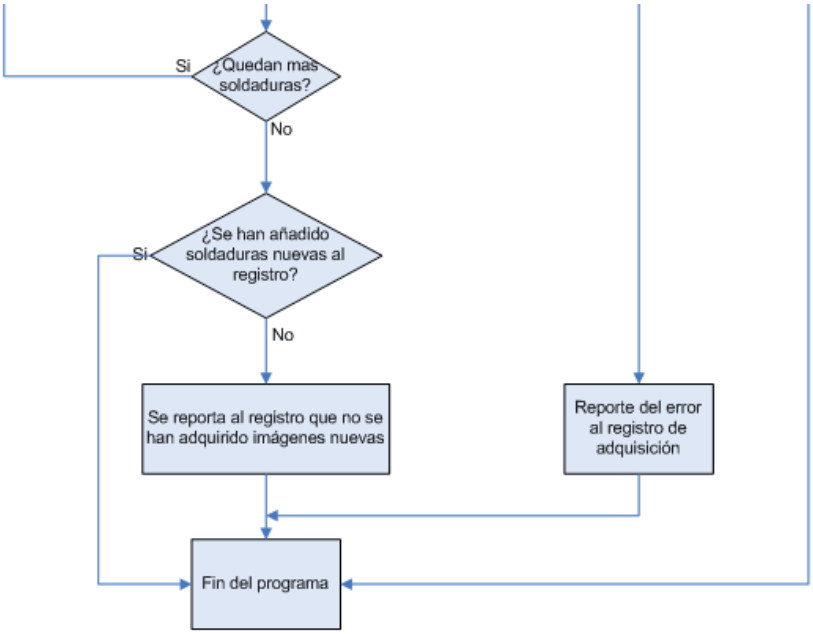
7.4. Programa principal:

Este programa principal es el que se que encarga de realizar la adquisición en sí y de gestionar y coordinar a las otras tres subrutinas

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:




Equipo listo=0
Adquisición imagen primera cámara
Llamada a Genera nombre 1
¿Esxite el directorio de la imagen?
No
Si
¿Se encuentra la imagen gurdada en disco?
Si
No
Guradado en disco de la imagen de la primera cámrara y reporte en el registro la información
¿Se ha guardado la imagen correctamente?
No
Si
Adquisición imagen segunda cámara
Llamada a Genera nombre 2
¿Esxite el directorio de la imagen?
No
Si
¿Se encuentra la imagen gurdada en disco?
Si
No
Guradado en disco de la imagen de la segunda cámrara y reporte en el registro la información
¿Se ha guardado la imagen correctamente?
No
Si

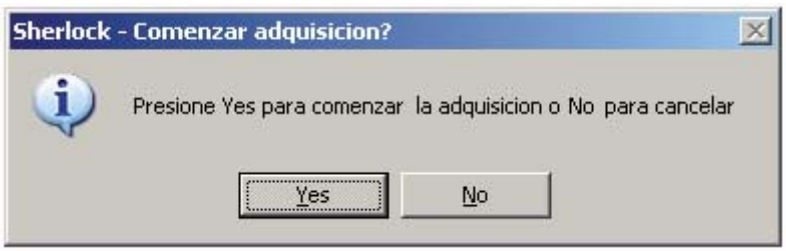


Una vez ejecutado el programa,

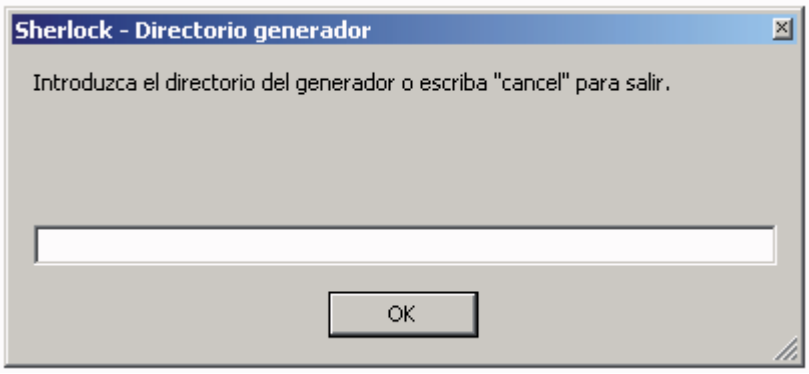
7.4.1. Inicialización.

- Se inicializan a *FALSE* todas las salidas digitales, apagando así cualquier piloto o indicador que se pudiese quedar encendido de un procesado anterior.

- Se muestra en pantalla un mensaje preguntando si se desea empezar el procesado o por el contrario desea cancelarlo. Si se pulsa "Yes" se continuará ejecutando el resto de código, pero si por el contrario se pulsó "No" se para la ejecución del mismo, como se muestra a continuación:

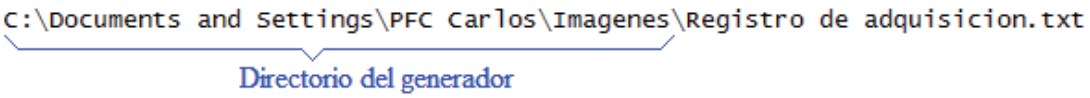


- Se entra en un bucle *while*, del cual solo se saldrá si se introduce un directorio de generador correcto o se escribe la palabra "cancel". Una vez dentro, el programa pregunta por la dirección del generador, ya que es donde se encuentran las imágenes guardadas y el registro de adquisición, dando también opción a cancelar la ejecución escribiendo "cancel":



- Se comprueba la existencia del directorio generador introducido. De no existir se mostrará el siguiente mensaje por pantalla, y se volverá al principio del bucle *while*, volviendo a preguntar por el directorio del generador y permaneciendo así dentro de este bucle hasta que se cumpla una de las condiciones anteriormente explicadas.

- Se rellena en una variable el nombre completo del directorio del registro de adquisición. Para ello se coge el directorio del generador y se le añade una contrabarra seguido por el nombre del fichero, que es *Registro de adquisición.txt*.



- A partir del nombre del directorio del registro de adquisición se comprueba la existencia del mismo. De este modo, si existe, implica que la adquisición estaba ya comenzada, indicándoselo así al usuario mediante la siguiente ventana:

Además, se añadirán en el registro de adquisición dos retornos de carro seguidos por el texto "Nueva adquisición" y finalizado por otro retorno de carro. Seguidamente se salta a una etiqueta llamada *comienzo*, que indica el principio de la adquisición.

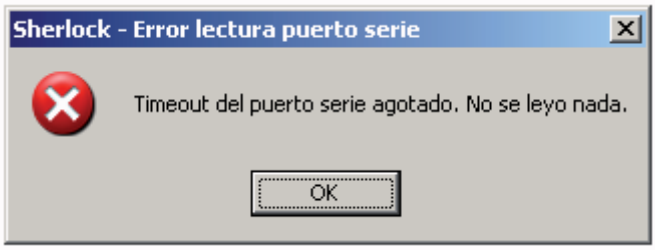
- Si el archivo del registro de adquisición no existe se crea, añadiendo al principio del mismo el texto "FORMAT: "Name: Date at time @ Directory"".

7.4.2. Adquisición.

- A continuación se encuentra la etiqueta *comienza*, que es la que indica que comienza en sí la parte adquisición.

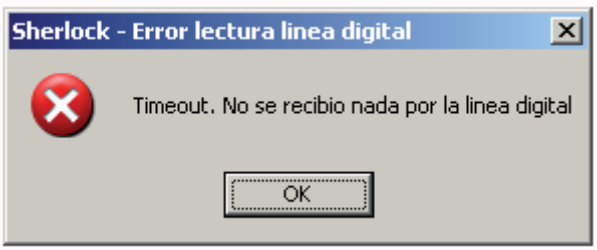
- Se levanta la línea digital *equipo listo* y se lee del puerto serie el telegrama recibido, que es el que envió el robot ABB.

- Si no se recibió nada se muestra el siguiente mensaje en pantalla y se le envía al robot un *NACK*. Además se llama a la subrutina *Control errores* con el código de invocación "0", finalizando así la ejecución de este programa.



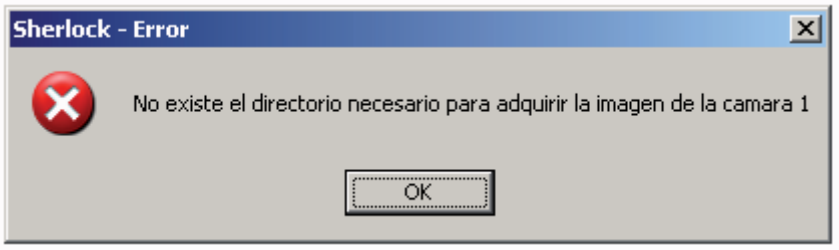
- Si por el contrario se recibió el telegrama del ABB, se le notifica su correcta recepción mediante un *ACK*.

- Se espera a que el robot levante la línea digital *comienza medida*, necesaria para adquirir la imagen. Para lograr esto se entra en un bucle *while*, del cual se saldrá una vez que se haya levantado la línea o por el contrario se alcance un time out. Una vez dentro del bucle se lee la entrada digital, y seguidamente se cuenta el número de lecturas que se han realizado. Si esta lectura es igual a un valor dado, constante definida en el programa, indica que el time out se superó, en cuyo caso se mostrará el siguiente error en pantalla y se llamará a la subrutina *Control errores*, con el código de invocación 1.



- Si en la lectura se leyó levantada la línea digital *comienza medida*, se saldrá del bucle *while* y se bajará la línea digital *equipo listo*. Seguidamente se adquirirá la imagen correspondiente a la primera cámara y invocará a la subrutina *Genera nombre 1*.

- Se comprobará si existe el directorio necesario para guardar la imagen, como indica la subrutina *Genera nombre 1* después de su invocación. En caso de no existir, me reportará el siguiente mensaje por pantalla:



Además, se generará la línea de texto que indicará que error ocurrió. Esta línea consiste en el tipo de error seguido por el directorio fallido, como se muestra a continuación:

```
No existe el siguiente directorio necesario  para la adquisicon de la camara 1: C:\Documents and Settings\PFC Carlos\Imagenes\AR003
```

- Se añade esta línea al registro de errores, con índice 2, y se llama a la subrutina *Control errores* con el código de invocación 2.

- Si el directorio existe, lo siguiente que se realiza es comprobar si la imagen adquirida se encuentra ya guardada en disco. Si existe se pasa a adquirir la imagen a la que apunta la segunda cámara.

- Si por el contrario esta imagen no existiese, se procede a su guardado en disco. Una vez guardada la imagen en disco, se comprueba que no se produjo ningún fallo durante el salvado. De ocurrir una anomalía, se indicará por pantalla mediante este mensaje:

- Se genera la siguiente línea de texto que indicará qué error ocurrió. Esta línea consiste en el tipo de error seguido por el nombre de la imagen fallida, como se muestra a continuación:

```
Fallo al guardar la siguiente  imagen de la camara 1: BR004C003.bmp
```

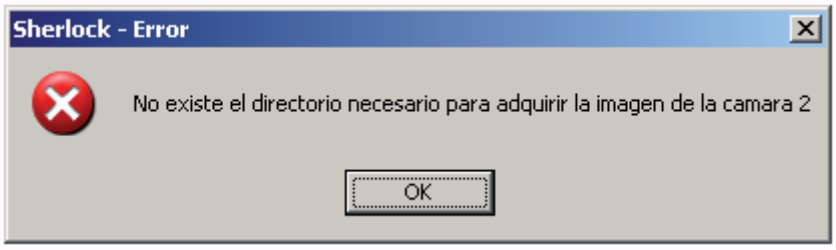
- Se añade esta línea al registro de errores, con índice 4, y se llama a la subrutina *Control errores* con el código de invocación 4.

- Una vez comprobado que se realizó con éxito el salvado en disco, se añade al registro de adquisición la línea de información correspondiente a esta imagen y que generó la subrutina *Genera nombre 1*. Además se modifica una variable, llamada *ctrl_img_adq*, que es la encargada de indicar si una vez ejecutado y

finalizado el programa no se adquirió ninguna imagen por estar ya todas guardadas en disco.

- Se realiza la adquisición de la segunda cámara, siendo este proceso idéntico al que se realizó con la primera cámara. Para ello se adquiere la imagen correspondiente a esta segunda cámara e invoca a la subrutina *Genera nombre 2*.

- Se comprobará si existe el directorio necesario para guardar esta imagen como indica la subrutina *Genera nombre 2* después de su invocación. En caso de no existir, me reportará el siguiente mensaje por pantalla:



Además se genera la línea de texto que indicará qué error ocurrió, añadiéndola al registro de errores, con índice 3, y seguidamente se llama a la subrutina *Control errores* con el código de invocación 3. Esta línea consiste en el tipo de error seguido por el directorio fallido, como se muestra a continuación:

```
No existe el siguiente directorio necesario  para la adquisicon de la camara 2: C:\Documents and Settings\PFC Carlos\Imagenes\AR001
```

- Si el directorio existe, lo siguiente que se realiza es comprobar si la imagen adquirida se encuentra ya guardada en disco. Si existe se pasa a adquirir la imagen a la que apunta la segunda cámara.

- Si por el contrario esta imagen no existiese, se procede a su guardado en disco. Una vez guardada la imagen en disco, se comprueba que no se produjo ningún fallo durante el salvado. De ocurrir una anomalía, se indicará por pantalla mediante este mensaje:

Además se genera la línea de texto que indicará qué error ocurrió, añadiéndola al registro de errores, con índice 5, y se seguidamente se llama a la subrutina *Control errores* con el código de invocación 5. Esta línea consiste en el tipo de error seguido por el directorio fallido, como se muestra a continuación:

```
Fallo al guardar la siguiente  imagen de la camara 1: BR004C003.bmp
```

- Una vez comprobado que se realizó con éxito el salvado en disco, se añade al registro de adquisición la línea de información correspondiente a esta imagen y que generó la subrutina *Genera nombre 2*. Además se modifica una variable, llamada *ctrl_img_adq*, que es la encargada de indicar si una vez ejecutado y finalizado el programa no se adquirió ninguna imagen por estar ya todas guardadas en disco.

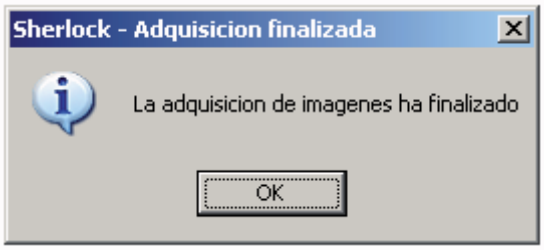
- Cuando finaliza el proceso de adquisición de la segunda imagen se comprueba si quedan más soldaduras. Para ello se lee del telegrama el cuarto, quinto y sexto carácter, como se muestra a continuación:

STXMED_A_R005C009_R005C010ETX

- Si forman la palabra "MED", indica que hay más soldaduras a procesar, por lo cual se saltará a la etiqueta *comienzo*, que indica el principio del proceso de adquisición.

- Si por el contrario forma un "END", señala que esta era la última soldadura a procesar. En este caso se comprueba si se han adquirido nuevas imágenes en esta ejecución, como indica la variable *ctrl_img_adq*, y, si no ha habido nuevas adquisiciones, se añade al registro de adquisición la indicación "No se ha adquirido ninguna imagen nueva".

A continuación, tanto si se hubiesen añadido imágenes a la adquisición como si no, se le notifica al usuario el fin de este proceso de adquisición mediante el siguiente mensaje:



- Finalmente se baja la línea *equipo listo* y se termina con la ejecución de este programa.

CAPÍTULO 5

Desarrollo del programa de procesado

DESARROLLO DEL PROGRAMA DE PROCESADO

1. Introducción.

Una vez adquirida la imagen, el siguiente paso a realizar es procesarla, y así verificar si se trata de una soldadura correcta o por el contrario presenta anomalías y debe ser revisada por un operario. Este capítulo se centra en la parte correspondiente al procesado de imágenes, cuyo objetivo se centra en localizar defectos en las soldaduras.

El objetivo del procesado es encontrar anomalías en las soldaduras, como pueden ser, deformaciones, manchas o soldaduras incompletas, además de crear un registro en disco con la principal información de cada soldadura procesada.

En caso de encontrar una soldadura sospechosa, se añadirá una referencia en un fichero de texto que registra todas las soldaduras procesadas. Además, esta imagen se guardará dentro de una capeta específica para soldaduras con defectos en el directorio del generador, manteniendo la imagen su nombre original. Finalmente se encenderá un piloto en el dispositivo VA31 para indicar al operario que se ha encontrado una soldadura con indicación.

El programa de procesado lleva la cuenta del número de fotos adquiridas y el número de fallos, imágenes con indicación. Dependiendo de lo deseado por el usuario, se le preguntará al comenzar dicho programa de procesado, el sistema se detendrá cuando se alcance un número máximo de fallos o cuando el porcentaje de errores sea mayor que uno dado. Una vez superado dicho valor, en el dispositivo VA31 se quedará encendido un piloto que indica al operario que se superó ese umbral de fallos.

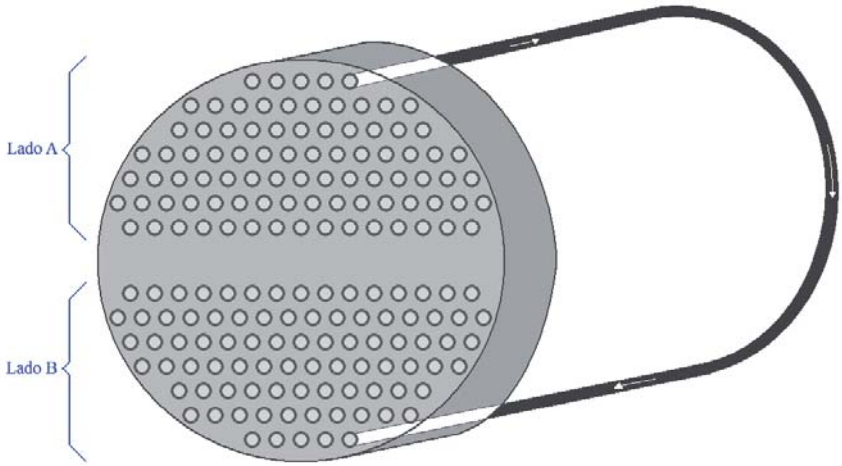
Para lograr esto, se ha desarrollado un programa principal con siete subrutinas, las cuales se explicarán a lo largo del presente capítulo.

2. Procesado de imágenes.

2.1. Introducción a las soldaduras

En esta introducción se van a explicar cómo son y que anomalías típicas presentan estas soldaduras que se van a procesar.

Como se comentó en el primer capítulo, los generadores de vapor constan de miles de soldaduras por donde circulará líquido a refrigerar. La placa de tubos donde se encuentran estas soldaduras se divide en dos caras. Del lado *A* de la placa se suelda un tubo, el cual recorre varios metros en el interior del generador de vapor, hace un giro de 180 grados y vuelve otra vez a la placa inicial, pero esta vez termina con una soldadura en el lado B.



La placa es una pieza circular de acero con medio metro de espero. Dicha placa se encuentra agujereada. En estos agujeros irán insertados los tubos por donde circulará el líquido. A continuación se muestra una imagen de un trozo de placa hecha a escala con sus tubos.

El tipo de soldadura empleado es la denominada TIG (Tungsten Inert Gas), que se caracteriza por el empleo de un electrodo permanente de tungsteno, aleado a veces con torio o zirconio en porcentajes no superiores a un 2%. Dada la elevada resistencia a la temperatura del tungsteno (funde a 3410 °C), acompañada de la protección del gas, la punta del electrodo apenas se desgasta tras un uso prolongado. Además este tipo de soldaduras es un sistema de soldadura al arco con protección gaseosa, que utiliza el intenso calor de un arco eléctrico generado entre un electrodo de tungsteno no consumible y la pieza a soldar, donde puede o no utilizarse metal de aporte. Los gases más utilizados para la protección del arco en esta soldadura son el argón y el helio, o mezclas de ambos [16.].

La gran ventaja de este método de soldadura es, básicamente, la obtención de cordones más resistentes, más dúctiles y menos sensibles a la corrosión que en el resto de procedimientos, ya que el gas protector impide el contacto entre el oxigeno de la atmósfera y el baño de fusión. Además, dicho gas simplifica notablemente la soldadura

de metales ferrosos y no ferrosos, por no requerir el empleo de desoxidantes, con las deformaciones o inclusiones de escoria que pueden implicar. Otra ventaja de la soldadura por arco en atmósfera inerte es que permite obtener soldaduras limpias y uniformes debido a la escasez de humos y proyecciones. La movilidad del gas que rodea al arco transparente permite al soldador ver claramente lo que está haciendo en todo momento, lo que repercute favorablemente en la calidad de la soldadura. El cordón obtenido es por tanto de un buen acabado superficial, que puede mejorarse con sencillas operaciones de acabado, lo que incide favorablemente en los costes de producción. Además, la deformación que se produce en las inmediaciones del cordón de soldadura es menor [16].

Los defectos más típicos en las soldaduras son, deformaciones, manchas, burbujas y soldaduras incompletas. Aunque se analizarán en el siguiente apartado, donde se mostrarán imágenes capturadas por las cámaras, se procederá a explicar a modo de introducción estos principales defectos.

* Soldadura con deformación:

Dentro de esta categoría de anomalías, los síntomas que puede presentar este tipo de fallo son, poca penetración, poros y surcos a los lados de las soldaduras. Este defecto suele ser producido por un arco demasiado largo, o lo que es lo mismo, que el arco de la soldadura TIG está muy separado del elemento a soldar [16][17].

* Soldadura con mancha:

En esta anomalía lo que se observan son manchas en la soldadura, las cuales son debidas a inclusiones de tungsteno. Este defecto se produce básicamente por dos motivos. El primero ocurre cuando el arco está extremadamente cerca el arco de la soldadura, llegando a haber contacto entre el metal y la punta caliente del electrodo, La segunda causa ocurre por un arranque brusco en la intensidad que fluye por el arco de soldadura, ya que para evitar esto es necesario en arranque gradual del arco [16][17].

* Burbujas en las soldaduras.

En este tipo de defecto se observan en la superficie de la soldadura pequeñas pompas, pudiéndose encontrarse también en el interior de la misma, y no pudiendo de este modo ser detectadas mediante una inspección visual. Este fallo suele ser producido por una excesiva inclinación de la pistola de calor que realiza la soldadura TIG [16][17].

* Soldadura incompleta:

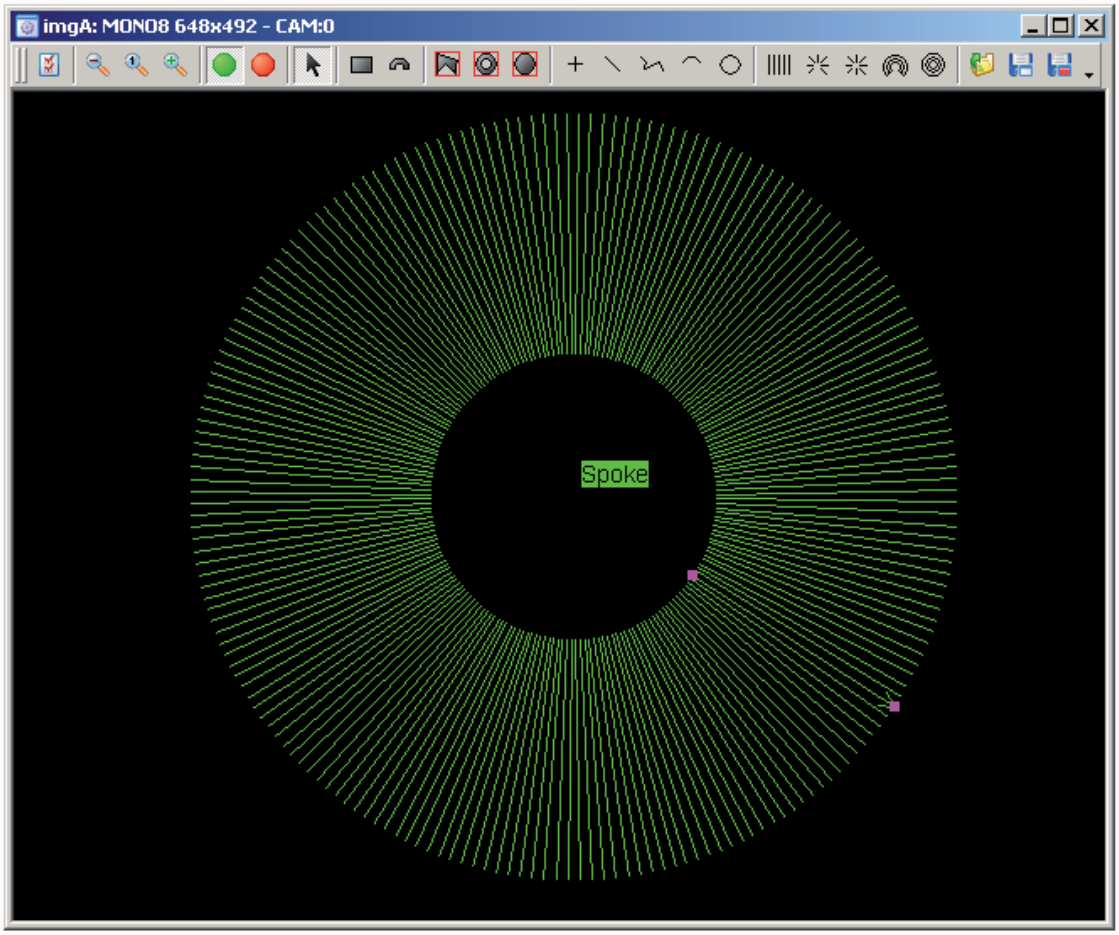
En este tipo de defectos falta por soldar una parte del tubo. Es debido a que por alguna causa el proceso de soldadura no se ha podido completar satisfactoriamente, y además, al finalizar esta parte del proceso no se ha reportado el fallo del mismo al operario [16][17].

2.2. Algoritmo de procesado de imágenes.

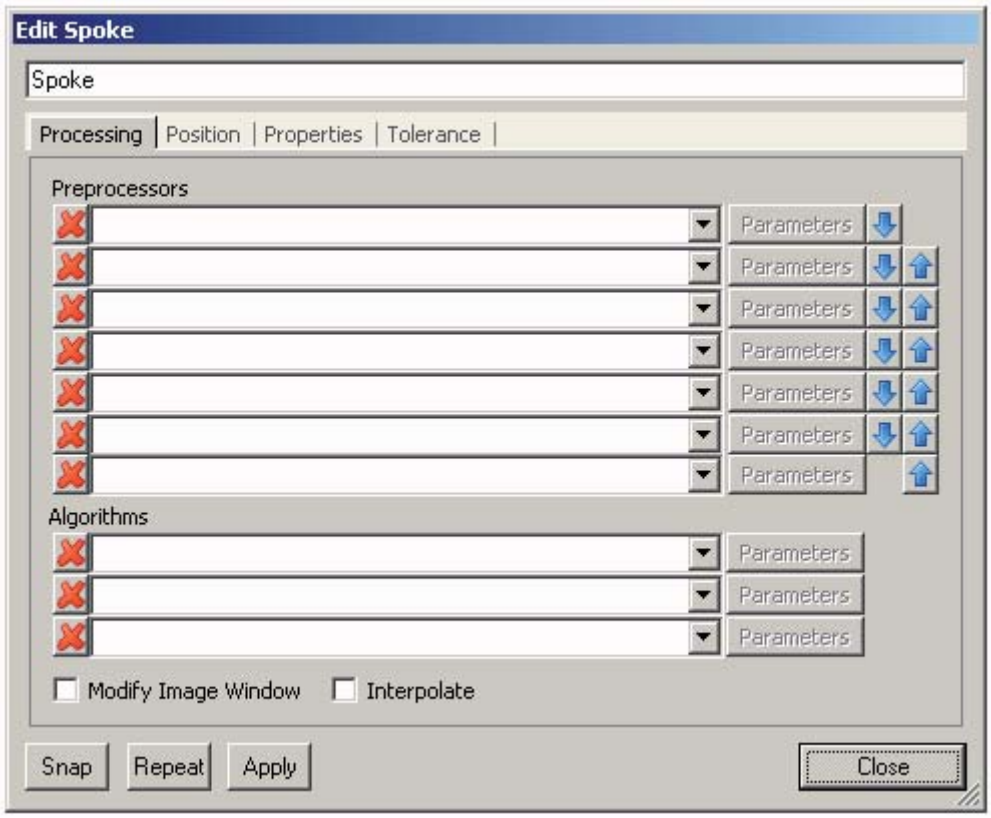
2.2.1. Explicación del algoritmo.

Este apartado del presente capítulo quinto se va a centrar en explicar todo lo relacionado con el algoritmo de procesado de imágenes.

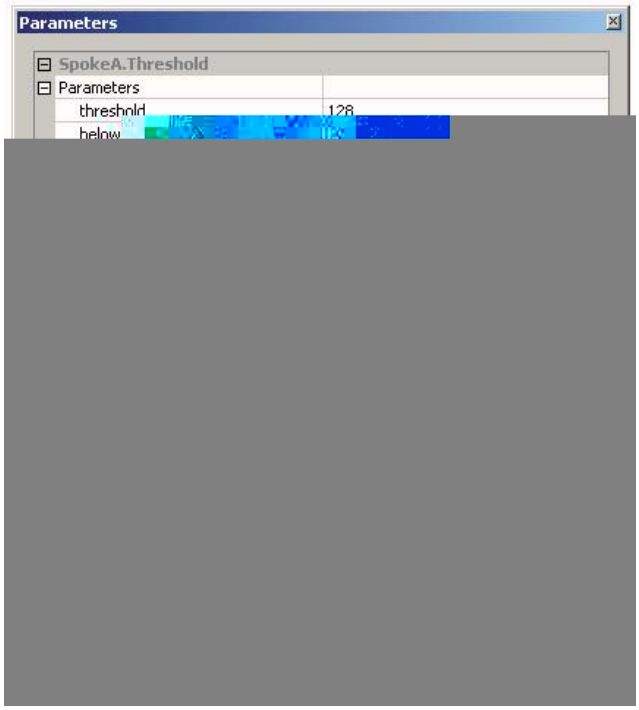
Para ello, es necesario digitalizar la zona de la imagen de interés mediante un ROI, que es el que define la zona de interés (*Region Of Interest*) donde se va a trabajar. Para el desarrollo del procesado de este proyecto se ha utilizado el ROI denominado como *Spoke*, que consiste en segmentos de radio de dos circunferencias concéntricas. El motivo de esta elección es porque al ser las soldaduras circulares, el ROI que mejor se adapta es uno circular, cuyas líneas de muestreo sean perpendiculares a la soldadura. Concretamente, se han utilizado 200 radios, o lo que es lo mismo, se analizan 200 puntos de la soldadura para determinar si es correcta o no, como se muestra a continuación:



En el sistema de procesado de imágenes Sherlock, cada ROI admite hasta 7 preprocesados y 3 algoritmos de procesado. Haciendo doble clic en la ROI aparece una ventana como la siguiente, que es la que permite elegir los preprocesadores y los algoritmos de procesado:



El preprocesado utilizado es el *Threshold*, con un valor de128, de manera que todo píxel dentro de la ROI que tenga un valor inferior a 128 en la escala de grises pasará a tener el valor 0 y todo aquel que tenga un valor superior pasará a tener un valor 255, como se indica en los campos inferiores al campo *threshold*. Se ha escogido este preprocesador con estos vales de *threshold* porque son los que consiguen el mejor rendimiento del algoritmo utilizado.



El algoritmo utilizado es el *Laser Caliper* (*Calibrador Láser)*, el cual mide la anchura de puntos sobre una línea brillante en una imagen. Dicha línea brillante debe ser clara frente a un fondo oscuro, es decir, tiene que haber un claro contraste. La elección de este algoritmo es debido a que es el que mejor detecta las anomalías bajo las condiciones en las cuales se han capturado las imágenes. Cuando se realiza la adquisición se ilumina la soldadura que se está obteniendo, y es ese tipo de iluminación la que hace aparecer ese destacado contraste entre la soldadura y el metal de fondo, que es el parámetro indispensable para que este algoritmo trabaje correctamente.

Este algoritmo busca un oscuro a brillante y a continuación un brillante a oscuro a lo largo de la línea ROI, explorado en la dirección de la flecha ROI. Para un procesado exacto, la línea ROI debería ser perpendicular a la línea brillante sujeta a estudio, ya que si no, no se podría analizar la anchura de esta línea brillante. Por ejemplo, si la soldadura presenta una deformación, al iluminarla durante la adquisición se va a perder la continuidad de esa línea brillante, y en consecuencia el algoritmo detectará que hay algo extraño en ese punto, ya que no verá este cambio de oscuro a brillante y brillante a oscuro. Aquí también entra en juego el preprocesador *threshold*, haciendo que un píxel valga un "0" o un "1" dependiendo de si está por encima o debajo del umbral establecido.

Los parámetros de salida que devuelve este algoritmo una vez ejecutado son los que se muestran a continuación:

- *Valid*: Verdadero (TRUE) si la medida del calibrador es correcta, es decir, si había una línea brillante a través del calibrador y además ambos bordes de la línea estaban por encima del umbral fijado en *min. strength.* Esta variable es un array de *booleanos*, cuya longitud coincide con el número de radios del ROI *Spoke*.

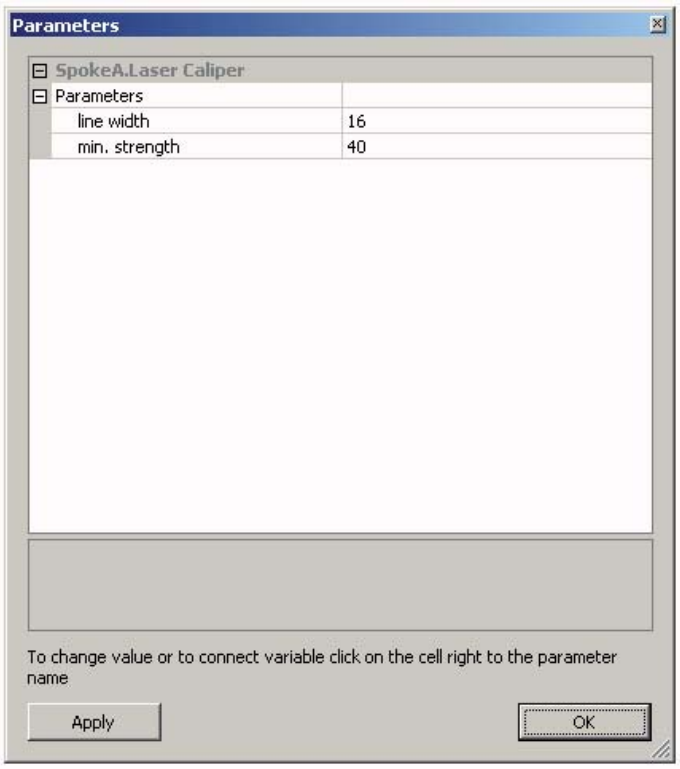
- *Strength*: intensidad de borde media en cada unos de los bordes usados en la medida de calibrador.
- *Side A*: posición X,Y del primer borde encontrado en la dirección de la línea ROI.
- *Side B*: posición X,Y del segundo borde encontrado en la dirección de la línea ROI.
- *Center:* posición X,Y del centro entre los lados A y B del calibrador
- *Thickness*: distancia entre la cara A y B del calibrador. A and B side calipers

Para poder ejecutar el algoritmo es necesario establecer dos parámetros. El parámetro *line width* es una condición previa al procesado y representa la anchura de línea brillante, debiendo ser ajustado a la anchura real de la línea brillante. Un valor alto para la anchura de la línea brillante ayuda a reducir el ruido, pero también puede reducir la exactitud de la medida. Para el análisis de soldaduras desarrollado en este proyecto se ha fijado un valor de *16*, que es el que se ajusta a la anchura de la línea brillante producida en una soldadura válida al iluminarla.

El otro parámetro a fijar es *Min. strength*, que establece la intensidad mínima en el borde para descubrir los puntos de línea brillantes. Este umbral es usado para suprimir líneas de intensidad bajas debido a ruido o a reflexiones de la luz. El valor que se le ha

dado a este campo para poder realizar un correcto procesado es de *40*, siendo este valor el que mejor se ajusta según las condiciones de luz y reflexiones producidos en la adquisición.

A continuación se muestra una ilustración que muestra los valores fijados para estos dos parámetros que se acaban de explicar:



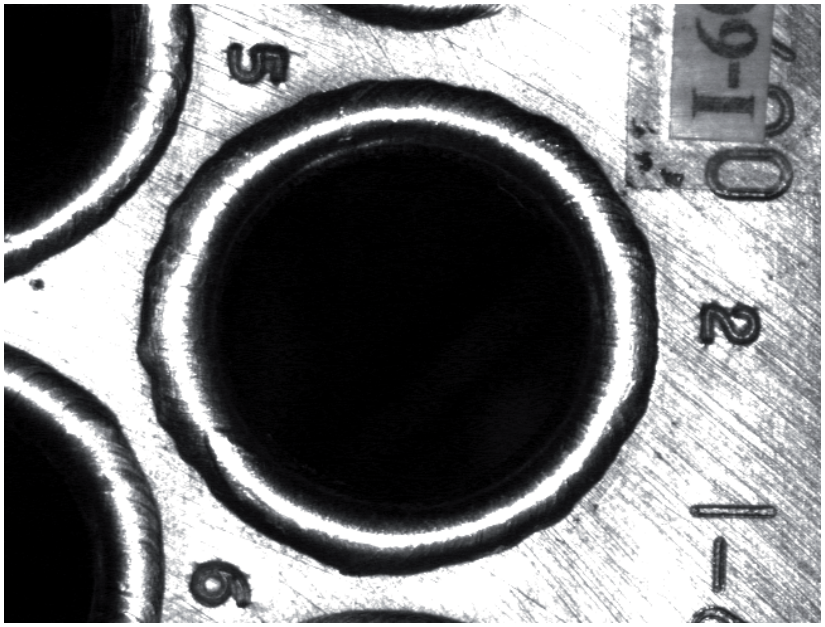
Para determinar si una soldadura es correcta o no, hay que analizar el array de valores del parámetro de salida *valid*¸ el cual es verdadero (TRUE) si la medida del calibrador fue correcta o falsa (FALSE) si esta medida no lo fue. Más concretamente se considerará **no apta** la soldadura si hay tres o mas *FALSE* seguidos en este registro *valid*.

2.2.2. Aplicación del algoritmo a soldaduras reales.

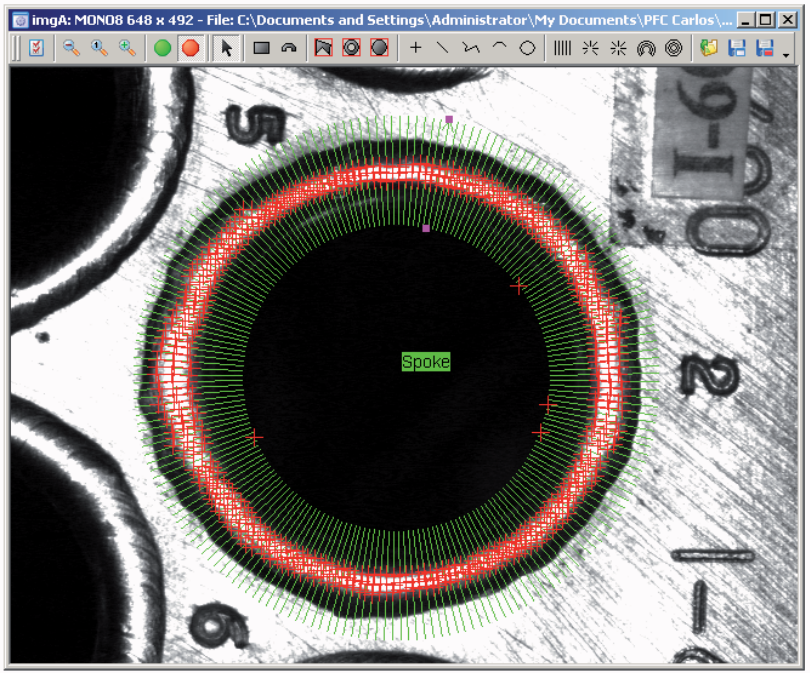
En esta sección se va a aplicar el algoritmo anteriormente explicado a varias soldaduras, observando así su comportamiento ante una soldadura correcta y varias con anomalías las anteriormente descritas.

- Soldadura correcta:

La primera imagen a procesar corresponde a una soldadura correcta que no presenta ningún tipo de fallo, como se muestra a continuación:



En ella se puede apreciar el aro uniforme y brillante alrededor de la soldadura, el cual será procesado y analizado a continuación. Si se le aplica el algoritmo anteriormente explicado obtendremos el siguiente resultado:



Se pueden observar las líneas ROI (líneas de color verde) perpendicular en todo momento a la soldadura. Según sea la muestra de la soldadura correcta o no se pueden dar dos casos. Si la muestra concreta de soldadura que está analizando esa línea ROI no es correcta, se verá una cruz (de color rojo) al principio de la misma, indicando que ese punto no superó la medida del calibrador, y devolviendo así un FALSE la posición del registro *valid* correspondiente a esta muestra. Que se dé una muestra inválida aislada no implica que la soldadura sea inválida, ya que esto se puede deber a una imperfección de la imagen capturada o a una pequeña sombra que haga la propia soldadura. Este es el

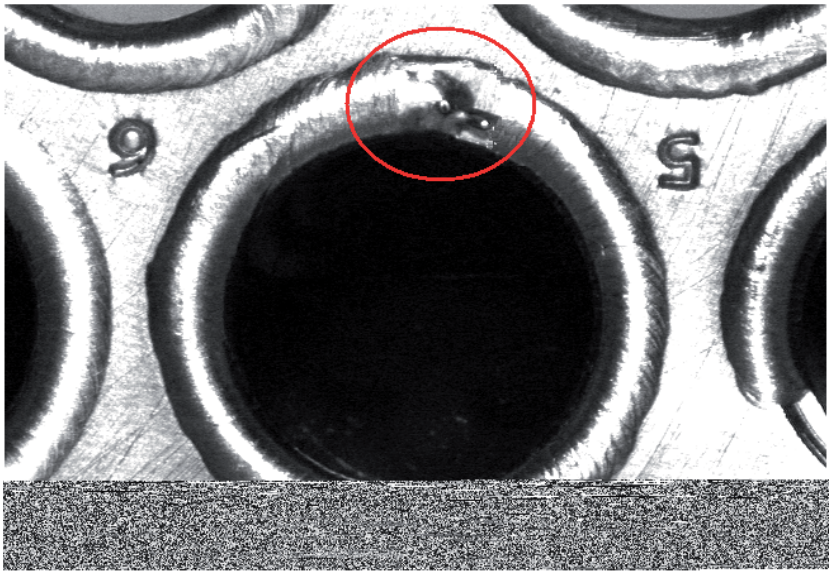
motivo por el cual, como se comentó antes, para afirmar que una soldadura presenta un defecto y deba ser revisada por un operario debe tener tres o más de estas cruces seguidas.

Si por el contrario la muestra concreta de soldadura que está analizando esa línea ROI es correcta, se observará la línea ROI y justo cuando pasa por la zona brillante de la soldadura aparecen las cruces (de color rojo), pero ahora posicionadas sobre la línea brillante, encontrándose situadas en el principio, punto medio y fin de la misma. Además se reportará un TRUE en la posición del registro *valid* correspondiente a esta muestra.

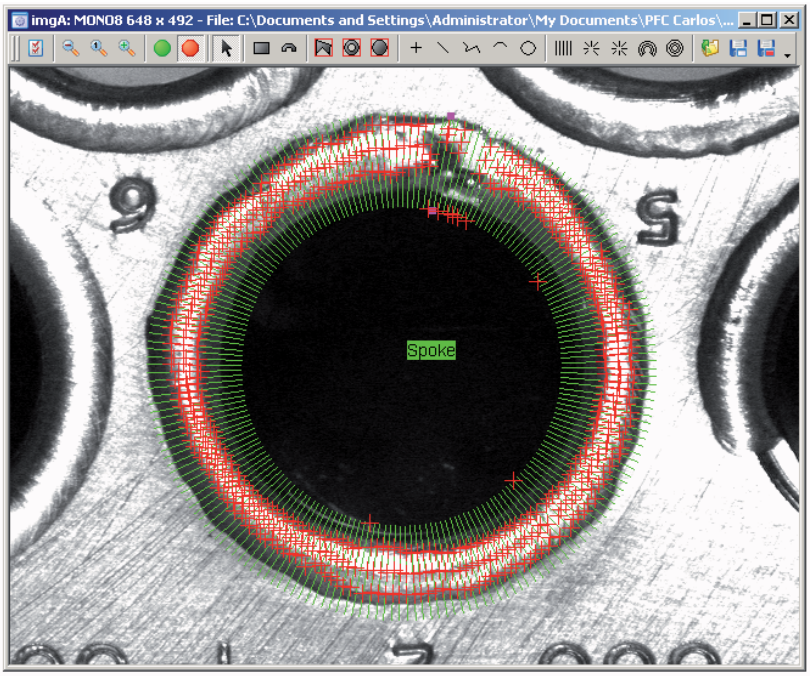
De acuerdo a lo anteriormente expuesto, se observa que en esta soldadura sólo hay 4 muestras de 200 que no son válidas, y como éstas no están seguidas implicaría que la soldadura es correcta.

- Soldadura con deformación:

La segunda foto que se va a procesar corresponde a una soldadura que presenta un defecto en la soldadura. En la parte superior de la misma, estando rodeado, se puede apreciar un surco junto con un par de imperfecciones más en forma de burbuja.



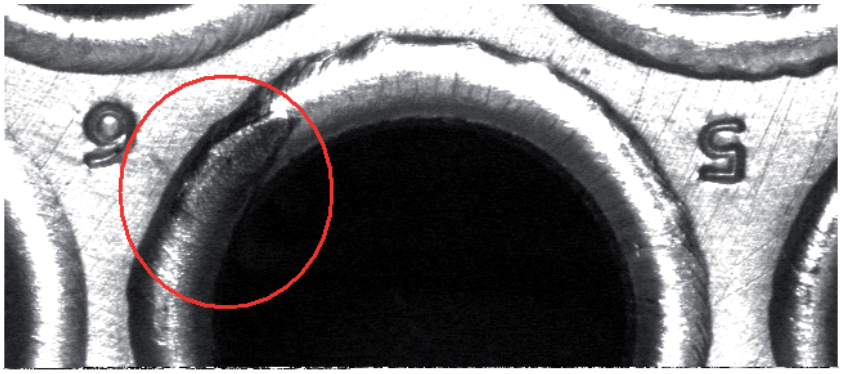
En ella se puede apreciar que el aro ya no es uniforme y brillante a lo largo de la soldadura, ya que este defecto rompe con esa continuidad. Si se le aplica el algoritmo anteriormente explicado obtendremos el siguiente resultado:



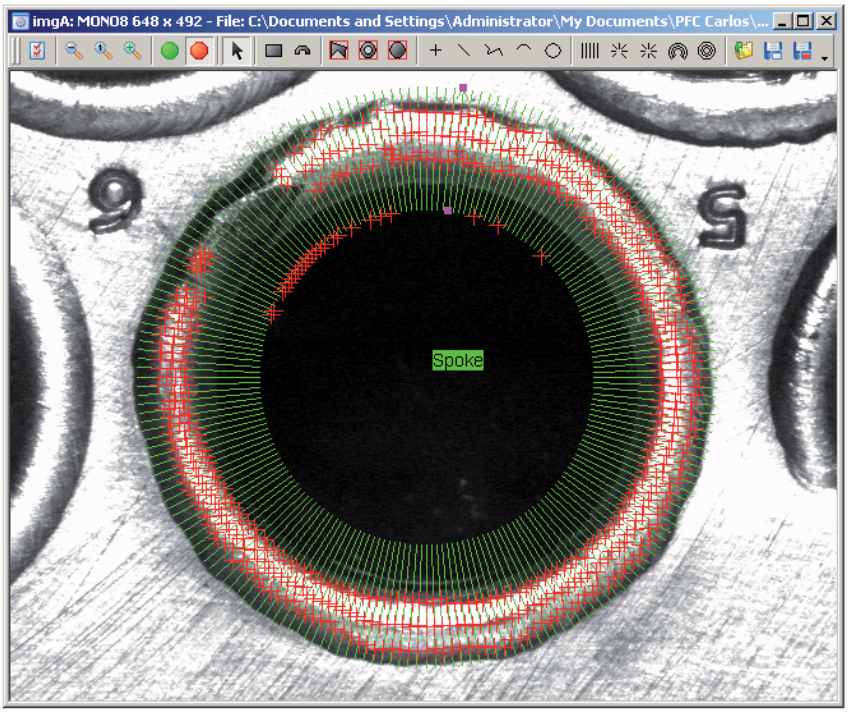
Como se observa en el resultado, quitando tres puntos aislados, se concentran todas las muestras inválidas del procesado en la zona de la soldadura que presenta la anomalía. Esto es debido a que en esa zona el algoritmo no encuentra el cambio oscuro a brillante y brillante a oscuro necesario para dar por válida la muestra, y agrupándose de ese modo las muestras inválidas en esa zona de la soldadura. Por ello, al haber más de tres muestras inválidas seguidas se consideraría esa soldadura defectuosa, debiendo ser revisada por un operario.

- Soldadura con mancha:

La tercera imagen se corresponde a una soldadura que presenta una mancha. Como se puede observar en la parte superior izquierda de la soldadura, estando rodeada, se aprecia una mancha extendida sobre una parte de la misma.



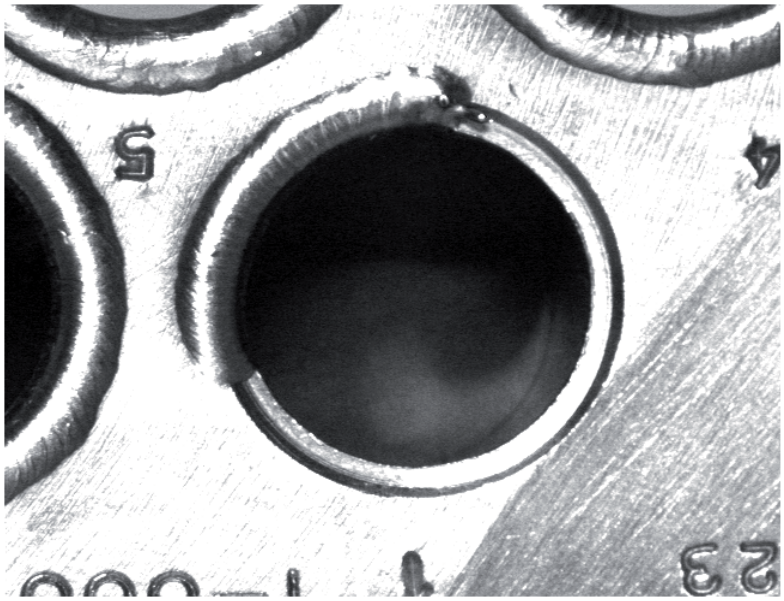
En ella se puede apreciar la discontinuidad del aro, el cual ya no es brillante en esta zona de la soldadura donde presenta la anomalía. Si se le aplica el algoritmo anteriormente explicado obtendremos el siguiente resultado:



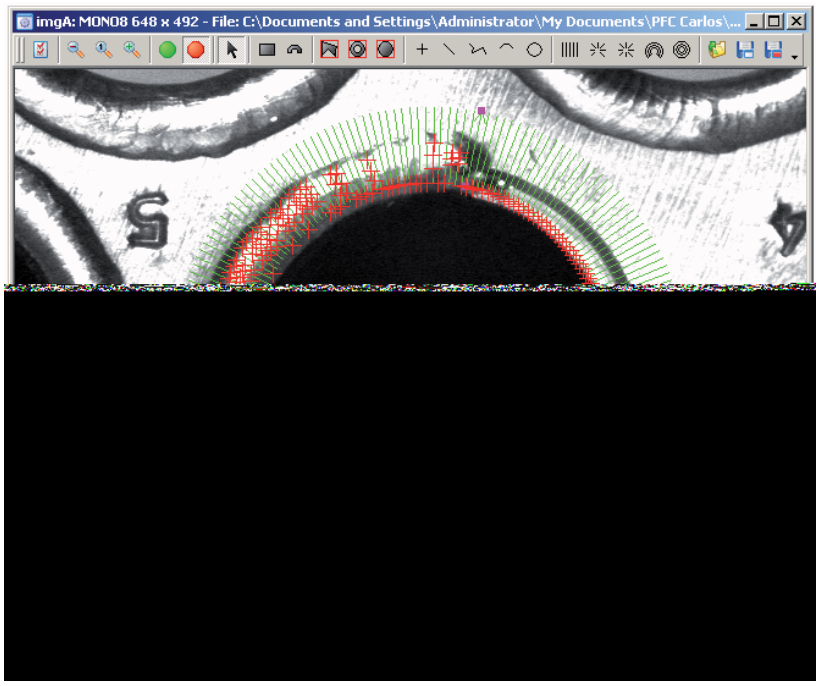
Como se observa en el resultado, quitando tres puntos aislados al igual que ocurría en el defecto anterior, se concentran todas las muestras inválidas del procesado en la zona de la soldadura que presenta la mancha. Esto es debido a que se ha perdido el contraste entre brillante y oscuro, y por tanto el algoritmo no puede encontrar un cambio de oscuro a brillante y brillante a oscuro, resultando de esta forma que la muestra es no válida. Por ello, al no producirse este contraste, se agrupan todas estas muestras inválidas a lo largo de la mancha, y en consecuencia, al haber más de tres muestras inválidas seguidas se consideraría esa soldadura defectuosa, debiendo ser revisada por un operario.

- Soldadura incompleta:

La cuarta y última imagen a procesar se corresponde con una soldadura incompleta. Como se puede observa, la parte derecha de la soldadura por algún motivo no se llegó a completar, estando de este modo la soldadura incompleta.



En ella se puede apreciar la discontinuidad del aro, el cual no es brillante en los extremos de la soldadura realizada. Por otro lado, también se observa que el aro brillante en la parte superior de la soldadura es muy ancho, llegándose a perder el contraste. Si se le aplica el algoritmo anteriormente explicado obtendremos el siguiente resultado:



Como se observa en el resultado, se concentran todas las muestras inválidas del procesado en la zona de la soldadura incompleta. Esto es debido a que, aún produciéndose el contraste oscuro a claro y claro a oscuro, en esta zona sin soldadura, el punto donde se inicia la línea brillante no posee la suficiente intensidad marcada según el parámetro *min. strength*, y por tanto no considerándose válida esa muestra de la línea brillante

Si se observa la anchura del aro brillante que produce en la parte de la soldadura incompleta, es prácticamente igual a la que presenta el aro brillante de la soldadura, por lo que en este caso el parámetro *line width* no detectaría la anomalía.

Además, también se puede apreciar en la parte superior de la soldadura, que al ser muy ancho el aro brillante, excediendo así el parámetro *line width*, y perder el contraste necesario para el algoritmo, en esa zona devolverá muestras inválidas, indicando que además en esa zona la soldadura presenta algún tipo de anomalía.

Por todo ello, al producirse numerosas muestras inválidas seguidas, tanto en la zona de la soldadura en la que es demasiado ancho el aro brillante, como en la zona donde no se ha realizado dicha soldadura, se consideraría a esta defectuosa, debiendo ser revisada por un operario.

3. Funcionalidades del programa de procesado.

Al igual que con la parte correspondiente a la adquisición de imágenes, durante el procesado se crea un registro llamado *Registro de procesado.txt*, el cual contiene información de la imagen procesada.

En este registro donde se salva la información del procesado de imágenes, no se guardan por orden de procesado como en el registro de adquisición, sino que se ordenan alfabéticamente mediante le método de ordenación bin sorting, es decir, que la primera siempre sería la imagen AR000C000 y la última sería, por ejemplo, la BR999C999.

El motivo de ordenarlo por orden alfabético es para facilitar al usuario la lectura del mismo, ya que si es necesario consultar el resultado de una imagen concreta que haya sido procesada con anterioridad, basta con ir leyendo el registro de arriba a abajo hasta encontrar la misma.

La información asociada a cada imagen que se añade al registro depende de si el procesado se pudo realizar o no, así como del resultado del procesado.

Por ejemplo, si no se encontró el directorio de la imagen a procesar se añadiría el siguiente error:

```
AR000C017: ERROR: Directorio no encontrado.
```

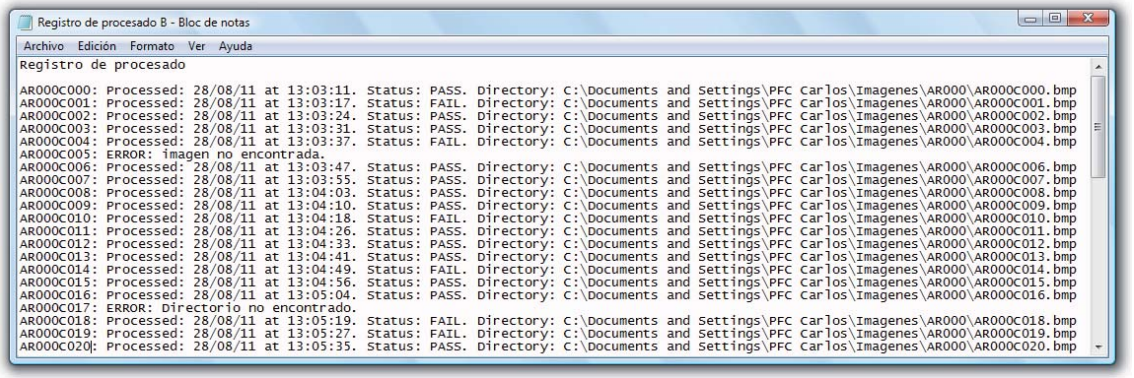
Si lo que no se encontró fue la imagen se reportaría el siguiente error:

```
AR000C005: ERROR: imagen no encontrada.
```

Si la soldadura se pudo procesar correctamente se añade la siguiente información al registro, en el que dependiendo de si es correcta o no se pondrá un *PASS* o un *FAIL* a continuación de *STATUS*:

```
AR000C009: Processed: 28/08/11 at 13:04:10. Status: PASS. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C009.bmp
AR000C010: Processed: 28/08/11 at 13:04:18. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C010.bmp
```

A continuación, se muestra una imagen de dicho registro, en el que hay errores de acceso a las imágenes, procesados con resultado a favor y en contra:

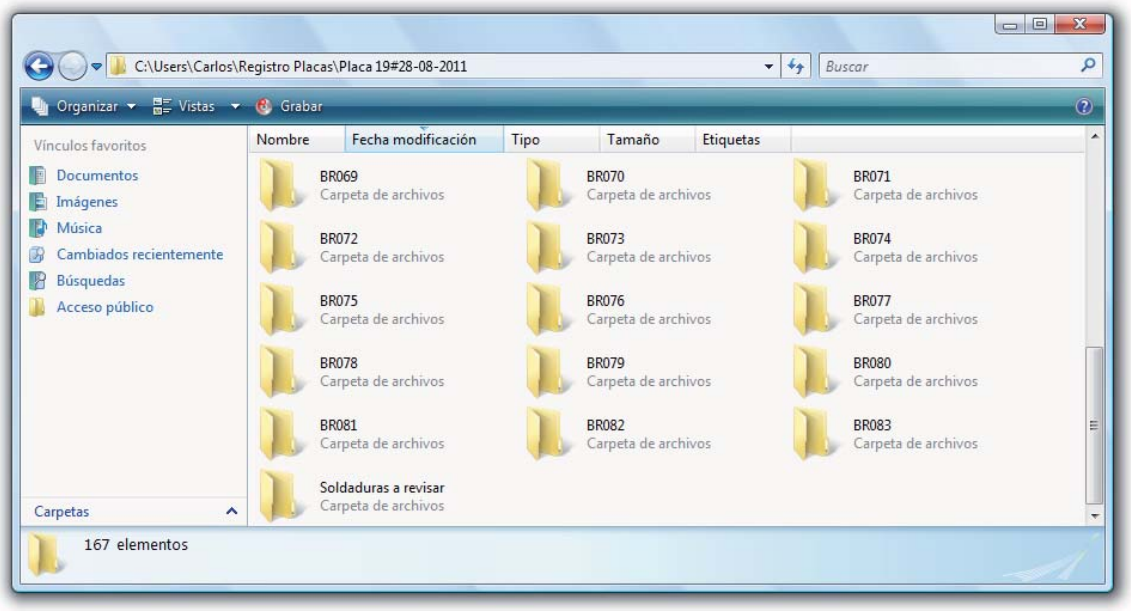


Al utilizar solo un algoritmo de procesado no se pude distinguir el tipo de fallo, es decir, el algoritmo desarrollado distingue las soldaduras inválidas, pero no es capaz de decir el motivo por el cual no es válida. Por este motivo no se ha añadido al registro de procesado el motivo por el cual la soldadura no es correcta, ya que tras un *FAIL* la información reportada del fallo sería siempre la misma.

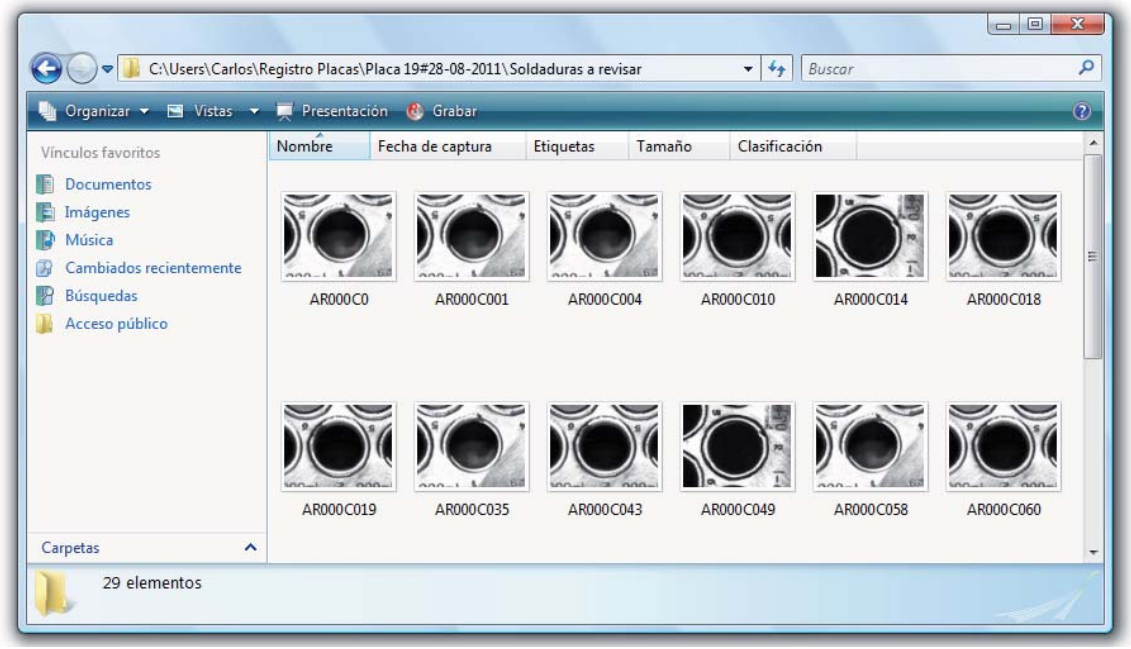
Es importante destacar que el programa, aunque en la actualidad no lo realice, está preparado para que se le incorporen más algoritmos de procesado, y así, de esta manera, poder reportar en el registro de procesado el tipo de fallo por el cual la soldadura es inválida, pudiendo destacar si fue por una deformación, una mancha, porque está incompleta o cualquier otro fallo.

Por otro lado, además de este registro de procesado anteriormente explicado, también se guardan todas las fotos sospechosas de fallo en una carpeta aparte, facilitando así la consulta al usuario.

Dentro de la carpeta del generador, aparte de las carpetas correspondientes a cada fila de la placa, como se explicó en el capítulo cuarto, se encuentra una carpeta, llamada *Soldaduras a revisar*, que contiene una copia de todas las imágenes de soldaduras defectuosas y que deben ser revisadas por un operario. La carpeta es la que se muestra a continuación:



Si se entra dentro de la carpeta se observará, como se explicó anteriormente, una copia de las imágenes a revisar, como se muestra a continuación:



4. Estructura del programa de procesado.

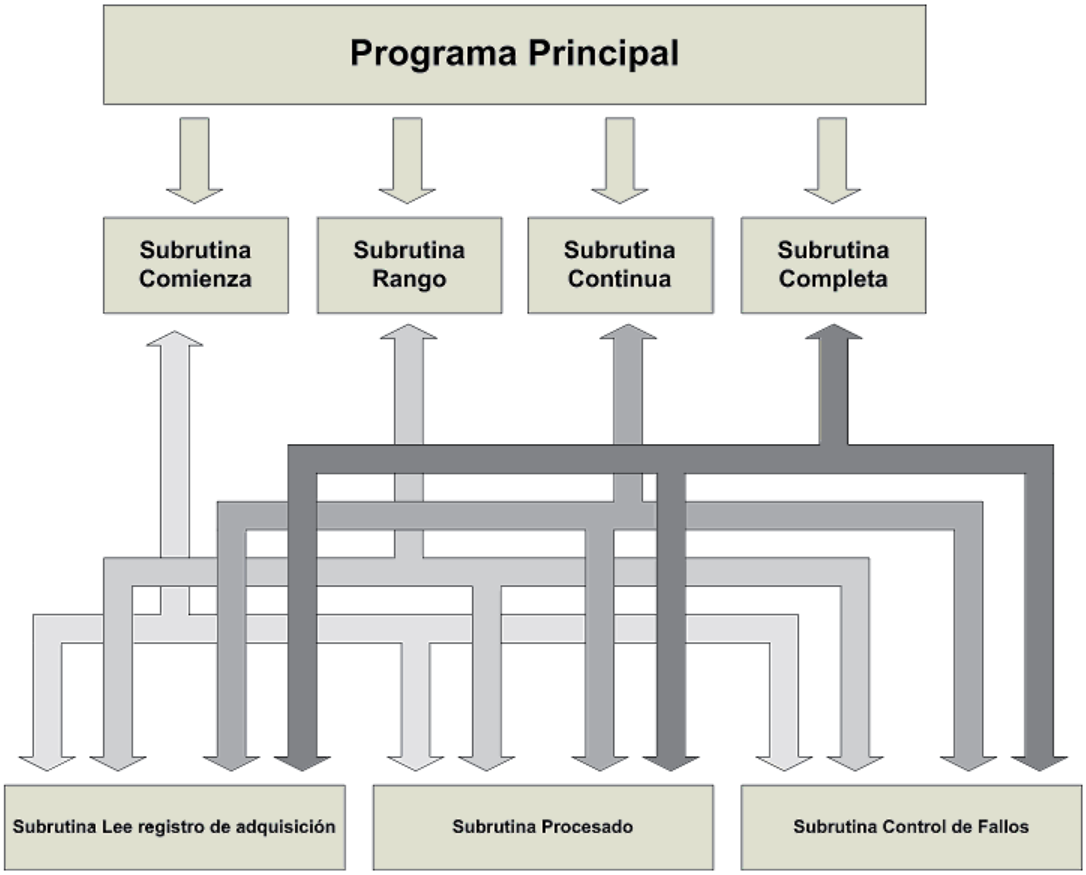
El programa de procesado desarrollado está compuesto por un programa principal y siete subrutinas, las cuales están jerarquizadas en diferentes niveles.

En un primer nivel se encuentra el programa principal, que es el que coordina e invoca a las subrutinas de un segundo nivel. Este programa principal realiza una labor de gestión a alto nivel. No se encarga de nada de lo que es el procesado de imágenes en sí.

Por debajo, en un segundo nivel hay cuatro subrutinas, encargadas de, por un lado, ejecutar la opción de procesado que le corresponda, y por otro coordinar e invocar a las subrutinas de un tercer nivel. Estas subrutinas son las encargadas de decir qué imágenes se procesan y cuáles no.

En un tercer y último nivel se encuentran tres subrutinas, que son las que realmente hacen todo el trabajo del procesado. Estas subrutinas no realizan ninguna labor de gestión o coordinación, se limitan a recibir los parámetros de entrada, procesarlos y retornar el o los resultados oportunos.

A continuación, se muestra una imagen ilustrativa que representa la estructura del programa desarrollado. Las flechas indican cómo se relaciona el programa principal y las subrutinas entre sí, así como del sentido de comunicación, que puede ser bidireccional o unidireccional.



Otro aspecto importante a comentar son las variables compartidas. Por ello, en esta última parte de este presente capítulo se van a explicar las variables que comparte el programa principal con cada una de las cuatro subrutinas de segundo nivel, así como las que comparten las subrutinas de segundo nivel con las de tercer nivel, detallando su nombre, tipo y descripción. Las variables que comparte el programa principal con cada una de las subrutinas de segundo nivel son las mismas para todas estas. Esto ocurre igual con las subrutinas de segundo y tercer nivel.

- Programa principal y subrutina *Comienza*, subrutina *Rango*, subrutina *Continúa* y subrutina *Completa*:

- ✓ *directorio_procesado*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de procesado.
- ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de adquisición.
- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"número"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.
- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

- Subrutina *Comienza*, subrutina *Rango*, subrutina *Continua* y subrutina *Completa* y la subrutina *Lee registro de adquisición*:

  - ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de adquisición.
  - ✓ *registro_procesado*: variable del tipo array que contiene strings, los cuales representan las imágenes a procesar.
  - ✓ *numero_soldaduras*: variable del tipo numérico que contiene el número de soldaduras que se han adquirido y que así contempla el registro de adquisición.

- Subrutina *Comienza*, subrutina *Rango*, subrutina *Continua* y subrutina *Completa* y la subrutina *Procesado*:

  - ✓ *líena_procesado*: variable del tipo string que contiene el nombre completo del directorio de la imagen a procesar.
  - ✓ *resultado_procesado*: variable del tipo string que contiene la información correspondiente al procesado de la imagen.
  - ✓ *estado_soldadura*: variable del tipo string que contiene el estado de la soldadura, pudiendo valer *"PASS"* o *"FAIL"*.
  - ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

- Subrutina *Comienza*, subrutina *Rango*, subrutina *Continua* y subrutina *Completa* y la subrutina *Control fallos*:

  - ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"numero"*.
  - ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
  - ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.

- ✓ *numero_soldaduras*: variable del tipo numérico que contiene el número de soldaduras que se han adquirido y que así contempla el registro de adquisición.
- ✓ *estado_soldadura*: variable del tipo string que contiene el estado de la soldadura, pudiendo valer *"PASS"* o *"FAIL"*.
- ✓ *resultado_ctrl_fallos*: variable del tipo string que contiene la información de si se superó el umbral de soldaduras defectuosas o no. Puede valer *"GO"* o *"STOP"*.

Además de estas variables, hay dos de ámbito general para todo el programa que son las que limitan el número máximo de soldaduras a procesar que admite el programa. Estas variables son *max_filas* y *max_columnas*.

5. Programa de procesado

Este último apartado del quinto capítulo se va a centrar en explicar cada una de las subrutinas desarrolladas así como el programa principal.

Antes de proceder a su explicación hay que resaltar la importancia de que para poder realizar el procesado es necesario que esté el registro de adquisición, ya que sin él no se puede realizar el procesado.

Para facilitar la comprensión del programa, se van a empezar a explicar las subrutinas de tercer nivel, continuadas por las de segundo y finalizada por el programa principal.

5.1. Subrutina Lee registro de adquisición.

Esta subrutina se encarga de leer el registro de adquisición (*Registro de adquisición.txt*) y crear un índice en memoria en el cual se encuentren todas las imágenes adquiridas ordenadas alfabéticamente por nombre y así poder ser procesadas en orden.

Debido a limitaciones del programa Sherlock, no se permite crear un array bidimensional de strings, como se muestra a continuación:

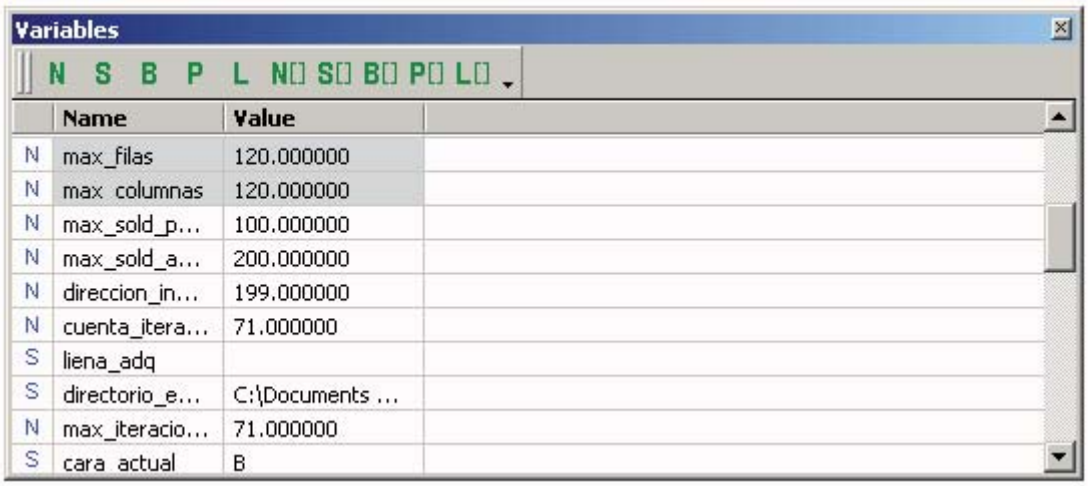
String de AR000C000	String de AR000C001	String de AR000C002	String de AR000C003	String de AR000C004	⋯	String de AR000C999
String de AR001C000	String de AR001C001	String de AR001C002	String de AR001C003	String de AR001C004	⋯	String de AR001C999
String de AR002C000	String de AR002C001	String de AR002C002	String de AR002C003	String de AR002C004	⋯	String de AR002C999
String de AR003C000	String de AR003C001	String de AR003C002	String de AR003C003	String de AR003C004	⋯	String de AR003C999
⋮	⋮	⋮	⋮	⋮	⋱	⋮
String de BR998C000	String de BR998C001	String de BR998C002	String de BR998C003	String de BR998C004	⋯	String de AR003C999
String de BR999C000	String de BR999C001	String de BR999C002	String de BR999C003	String de BR999C004	⋯	String de BR999C999

La solución adoptada es crear un array de strings de una sola dimensión, pero en el que se pueda saber los strings que corresponden a una misma fila, como el que se muestra a continuación

Para poder realizar este tipo de agrupación hay que reservar en memoria toda esa longitud necesaria, para ello hay dos variables que se encargan de ello, que son:

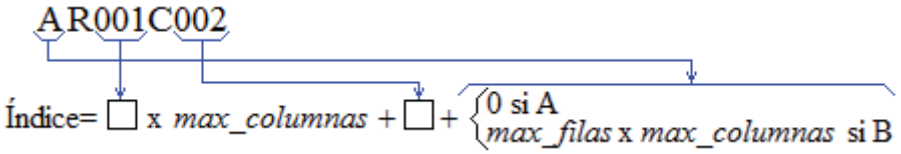
- ✓ max_filas: indica el número máximo de filas que puede tener la placa a procesar.
- ✓ max_columnas: indica el número máximo de columnas que puede tener la placa a procesar.

Estas variables tienen siempre el mismo valor por defecto, y para cambiarlo hay que hacerlo manualmente sobre la variable, en el cuadro que contiene todas las variables y que se muestra a continuación.



El único problema que presenta este tipo de agrupación es la cantidad de memoria innecesaria que ocupa cuando se trabaja con una placa pequeña, pero se optó por este diseño ya que es mas cómodo de cara al usuario, debido a que de esta forma no tiene que preocuparse en saber el número de filas y columnas que tiene en concreto la placa a procesar.

Una vez solucionado este problema, lo siguiente que se debe hacer es que a partir del nombre de una imagen se cree un índice único que haga referencia a una única entrada en el registro anteriormente mostrado. Esto se logró de la siguiente manera, y es multiplicando la fila de la imagen cargada por *max_columnas*, sumarle el número de columnas de la imagen cargada y dependiendo de si está en el lado A o B sumarle un offset o no, el cual es fruto de multiplicar *max_filas* por *max_columnas*. A modo de ejemplo, si *max_filas=100* y *max_columnas=100*, el offset sería de *100x100,* o lo que es lo mismo de *10000.* A continuación se muestra una imagen que muestra lo anteriormente explicado:



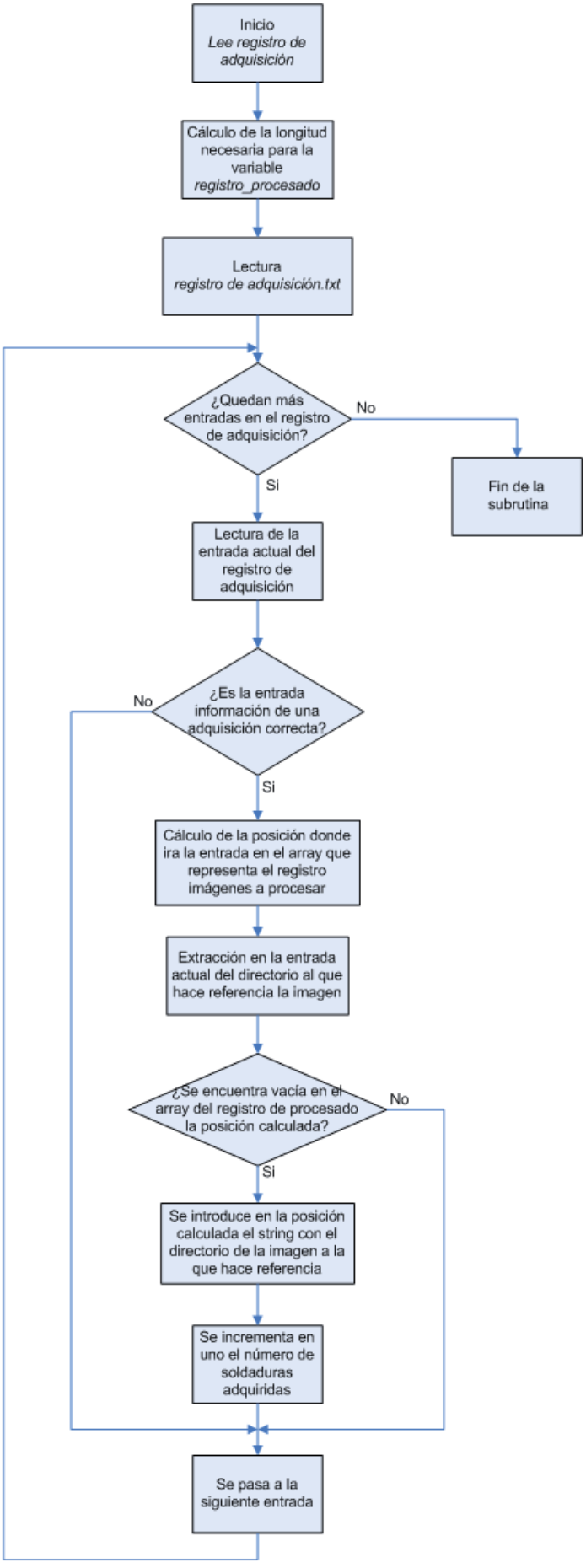
La variable que recibe es la que se muestra a continuación:

- ✓ *directorio_adquisición*: variable del tipo string que contiene el directorio completo de dónde se haya el registro de adquisición.

Las variables que devuelve una vez ha terminado su ejecución son las siguientes:

- ✓ *registro_procesado*: variable del tipo array de string, que contiene las imágenes a procesar.
- ✓ *numero_soldaduras*: variable del tipo numérico que contiene el número de soldaduras que se han adquirido y que así contempla el registro de adquisición.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:


Inicio
Lee registro de adquisición
Cálculo de la longitud necesaria para la variable registro_procesado
Lectura registro de adquisición.txt
¿Quedan más entradas en el registro de adquisición?
No
Si
Fin de la subrutina
Lectura de la entrada actual del registro de adquisición
¿Es la entrada información de una adquisición correcta?
No
Si
Cálculo de la posición donde ira la entrada en el array que representa el registro imágenes a procesar
Extracción en la entrada actual del directorio al que hace referencia la imagen
¿Se encuentra vacía en el array del registro de procesado la posición calculada?
No
Si
Se introduce en la posición calculada el string con el directorio de la imagen a la que hace referencia
Se incrementa en uno el número de soldaduras adquiridas
Se pasa a la siguiente entrada

Una vez invocada la subrutina,

- Se calcula, a partir de *max_filas* y *max_columnas* el número de soldaduras que hay por lado, y eso se hace multiplicando *max_filar* por *max_columnas*. Este valor es el que se utilizará como offset para calcular parte de la dirección del índice anteriormente explicado. Seguidamente se multiplica este número de soldaduras por lado por dos, y así tenemos el número total de soldaduras en la placa.

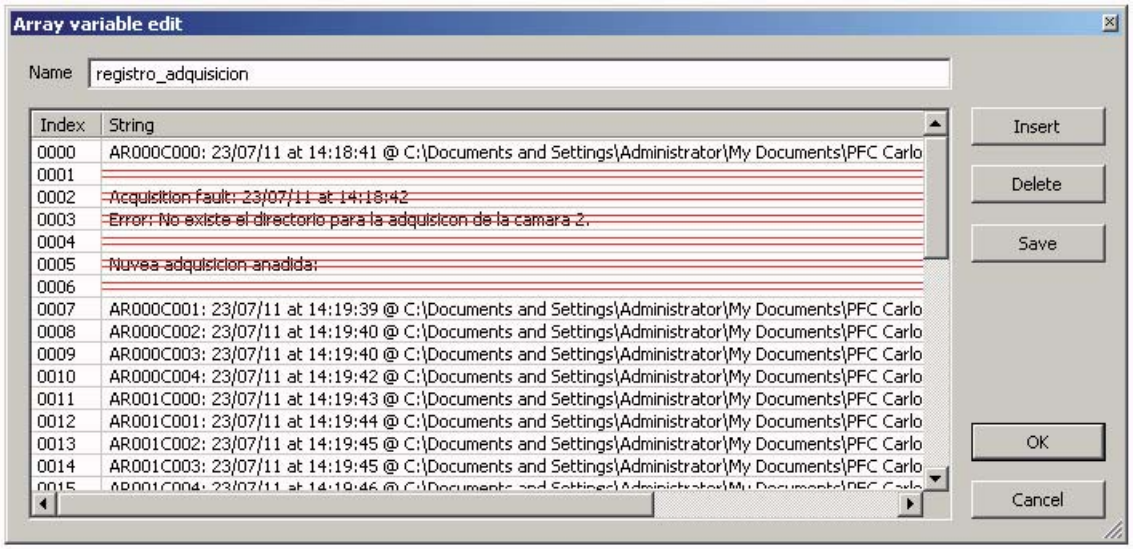
- Se crea en memoria un registro, llamado registro de imágenes a procesar, al cual se le introducirán por orden todos los nombres de los directorios de las imágenes a procesar. En primer lugar, se inicializa el array con dimensión de cero, y a continuación se redimensiona con una longitud igual al número máximo de soldaduras que puede tener la placa (anteriormente ya calculado, es el fruto de hacer *max_filas* x *max_columnas x 2*).

- Se realiza la subrutina es cargar en un array de strings en memoria el registro de adquisición (Registro de adquisición.txt) y seguidamente se borran las dos primeras líneas de dicho registro, ya que carecen de información relevante.

- Se inicializa a cero la variable que va a ser la encargada de contar el número de soldaduras que se tienen en el registro de adquisición.

- Se entra en un bucle *while*, del cual sólo se puede salir una vez recorridas todas las posiciones del este registro de adquisición anteriormente cargado.

- Dentro de este bucle lo primero que se hace es cargar la primera línea del registro de adquisición leído, y a continuación comprobar que la línea contiene el nombre de una imagen, siendo este del tipo AR□□□C□□□. Esto de hace para desechar la línea en el caso de que se trate de una entrada que indica un fallo, es decir, no tiene información sobre la imagen adquirida. A continuación, se muestra una imagen en la que aparecen tachadas en rojo todas las líneas que no se procesarían en función de lo anteriormente expuesto:



- Si la línea se corresponde con una imagen adquirida, se calcula el índice. Para ello lo primero que se hace es leer el lado. A continuación se lee el número de fila y se convierte de string a valor numérico. Seguidamente se lee el número de columna y se convierte de string a valor numérico.

- Se calcula el índice como se explicó anteriormente, que consiste en multiplicar esta fila cargada por *max_columnas*, sumarle el número de columna calculado.

- Se comprueba si se está en el lado A o B, y dependiendo de si se está en uno u otro se le suma al índice un offset o no. Recuérdese que este offset era el fruto de multiplicar *max_filar* por *max_columnas*.

- Se extrae de la línea cargada del registro de adquisición la información del directorio donde se encuentra dicha imagen almacenada. Para ello se eliminan los 34 primeros caracteres:

```
AR000C001; 23/07/11 at 14:19:39 @ C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

Quedándose así solo la información necesaria para el registro:

```
C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

Esta línea es el directorio de la foto que se debe procesar, y que se llama línea de procesado.

- Se comprueba que en la posición del array donde tiene que guardar el nombre del directorio de la foto correspondiente, no hay ninguna entrada. De haberla, no se escribe encima, ya que indicaría que ese nombre de directorio ha sido guardado anteriormente. Si por el contrario no hay ninguna entrada lo que se hace es guardar este nombre del directorio en su correspondiente lugar. Además de esto, siempre que se escriba algo en el registro de imágenes a procesar se incrementa en uno el número de soldaduras, para así poder tener controlado el número de las mismas.

- Una vez llegado a este punto se vuelve al principio del bucle *while*, procesando la siguiente imagen a la actual y así sucesivamente hasta haber analizado todas las entradas del registro de adquisición.

5.2. Subrutina Procesado.

Esta subrutina es la encargada de procesar una imagen, estando de esta forma incluido en esta subrutina el algoritmo de procesado desarrollado y explicado al comienzo del presente capítulo.

Las variables que recibe son las que se muestran a continuación:

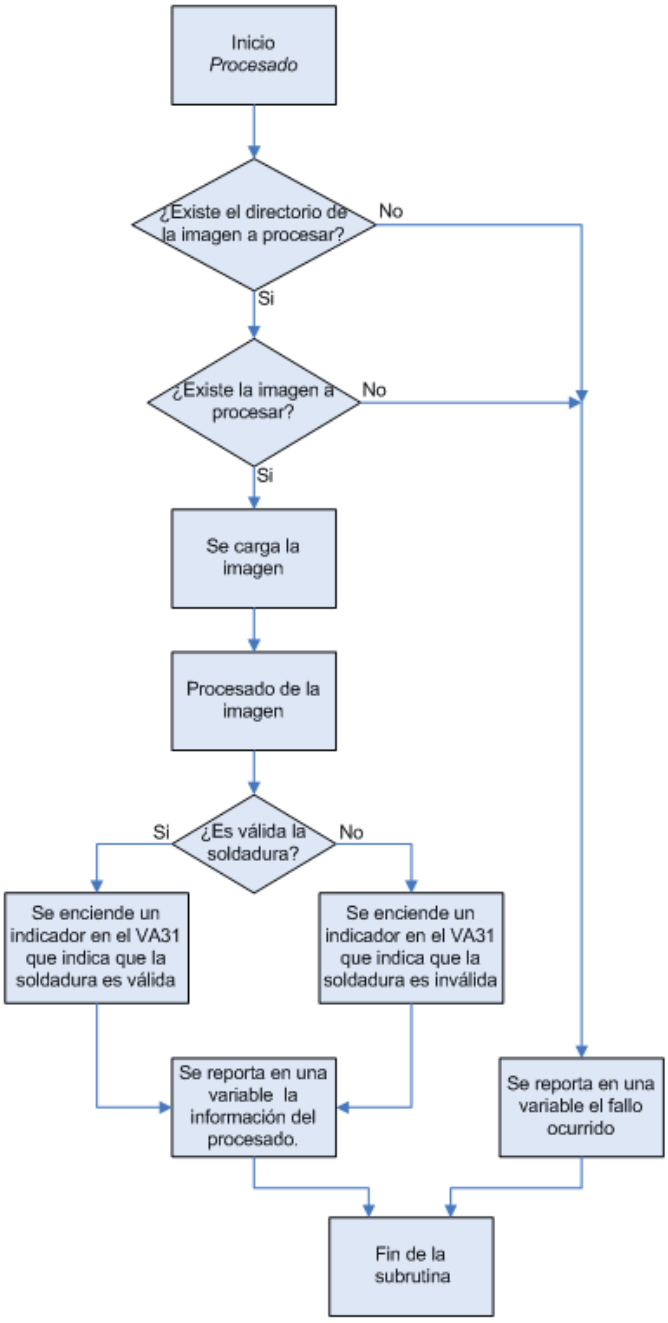
- ✓ *líena_procesado*: variable del tipo string que contiene el nombre completo del directorio de la imagen a procesar.

- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

Las variables que devuelve una vez ha terminado su ejecución son las siguientes:

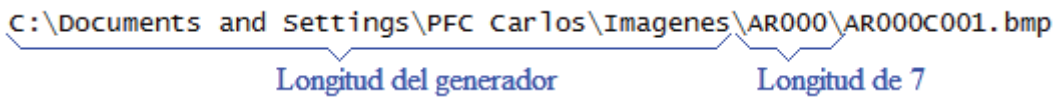
- ✓ *resultado_procesado*: variable del tipo string que contiene la información correspondiente al procesado de la imagen.
- ✓ *estado_soldadura*: variable del tipo string que contiene el estado de la soldadura, pudiendo valer *"PASS"* o *"FAIL"*.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:

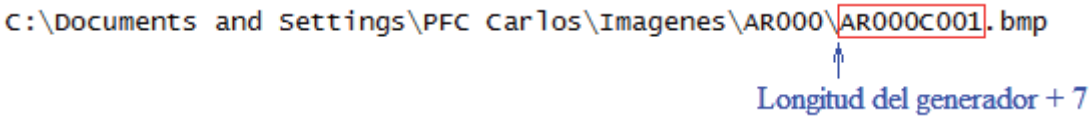


Una vez invocada la subrutina,

- Se crea la línea de texto que contendrá el resultado del procesado. Para ello es necesario extraer del directorio de adquisición el nombre de la imagen. Esto se consigue cogiendo la variable que contiene la longitud del generador y se le suma 7:



Sabiendo cuál es la posición del primer carácter que deseamos extraer y que después de ése es necesario sacar los nueve siguientes es posible obtener el nombre de la imagen, como se muestra a continuación:



- Se comienza a rellenar esa línea que representa el resultado del procesado, empezando por el nombre de la imagen y continuado por dos puntos:

```
AR000C001:
```

- Se comprueba que existe la carpeta que contiene la imagen a procesar. Esto se consigue extrayendo del directorio de la imagen a procesar solo la parte que hace referencia a la carpeta, como se muestra a continuación:

```
C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

Obteniendo así:

```
C:\Documents and Settings\PFC Carlos\Imagenes\AR000\
```

- Si no existe el directorio en concreto, se reporta al usuario el siguiente error y se finaliza la subrutina:

Además, se añade esta información en la línea que contiene la información del procesado, como se muestra a continuación:

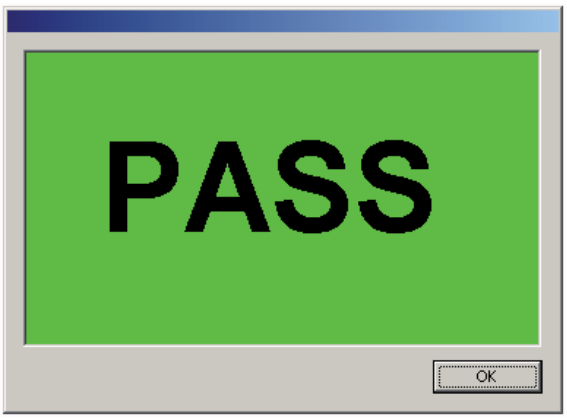
```
AR000C001: ERROR: Directorio no encontrado.
```

- Se apaga el piloto del dispositivo VA31 que indica que la soldadura no es correcta, se añade a la línea de información el directorio donde se encuentra la imagen procesada, como se muestra a continuación, y se finaliza la subrutina:

```
AR000C001: Processed: 15/08/11 at 18:47:45. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

- Si por el contrario no hay mas de tres FALSE seguidos, indica que la soldadura es correcta. Es ese caso lo primero que se realiza es encender en el dispositivo VA31 una piloto que indica que la soldadura es válida.

- Se muestra el siguiente mensaje por pantalla, el cual aparecerá durante un segundo y para posteriormente desaparecer, al igual que sucedía en el caso de una soldadura inválida:



- Se pone en estado PASS la variable *estado_soldadura*. Además, se añade a la línea que contiene la información de la imagen procesada un *Status: PASS.*, como se muestra a continuación:

```
AR000C001: Processed: 15/08/11 at 19:47:45. Status: PASS.
```

- Se apaga el piloto del dispositivo VA31 que indica que la soldadura es correcta, se añade a la línea de información el directorio donde se encuentra la imagen procesada, como se muestra a continuación, y se finaliza la subrutina:

```
AR000C001: Processed: 15/08/11 at 19:47:45. Status: PASS. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

5.3. Subrutina Control de fallos.

Esta subrutina es la encargada de llevar el control de las soldaduras inválidas para así poder detener el proceso de procesado cuando se supere un número de soldaduras defectuosas.

Los datos de entrada son el número de soldaduras adquiridas, el tipo de detención, sea por porcentaje de fallos o por número de fallos, el valor numérico del porcentaje de fallos o número el cual una vez superado detendrá el procesado, y el estado de la soldadura, sea válida, *PASS*, o inválida, *FAIL*.

Como salida devuelve en una variable llamada *resultado_ctrl_fallos* el estado *GO* si no se ha superado ese número máximo de soldaduras fallidas o *STOP* si se superó.

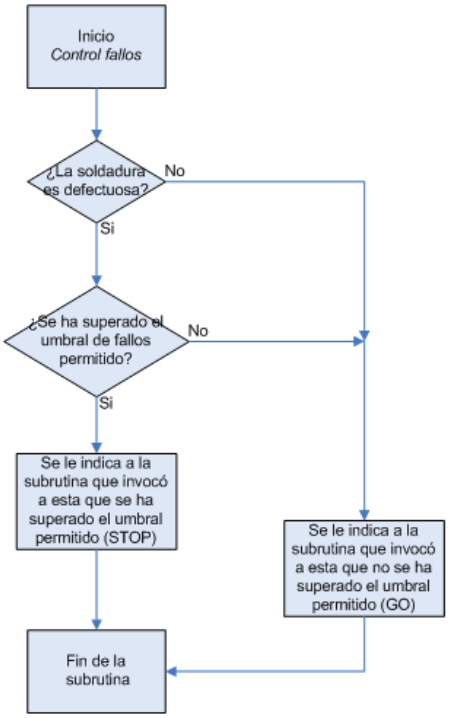
Las variables que recibe son las que se muestran a continuación:

- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"número"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.
- ✓ *numero_soldaduras*: variable del tipo numérico que contiene el número de soldaduras que se han adquirido y que así contempla el registro de adquisición.
- ✓ *estado_soldadura*: variable del tipo string que contiene si estado de la soldadura, pudiendo valer *"PASS"* o *"FAIL"*.

La variable que devuelve una vez ha terminado su ejecución es la siguiente:

- ✓ *resultado_ctrl_fallos*: variable del tipo string que contiene la información de si se superó el umbral de soldaduras defectuosas o no. Puede valer *"GO"* o *"STOP"*.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada esta subrutina,

- Se discierne si se está en el caso de parada por porcentaje de fallos o por el contrario está por número de fallos.

En el caso de que se haya seleccionado que se detenga por porcentaje de fallos, lo primero que se comprueba es si la soldadura fue válida, *PASS*, o no, *FAIL*.

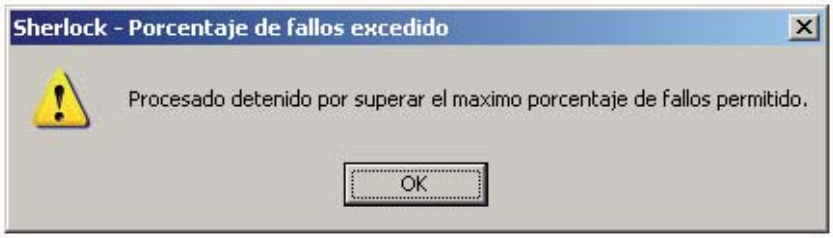
- Si es válida no se realiza ninguna operación, se termina la ejecución de la subrutina y se le pasa un *GO* a quien la invocó.

- Si por el contrario no es válida, *FAIL*, lo primero que se hace es incrementar en uno la variable que contiene el número de soldaduras fallidas y que se inicializó a cero cuando se ejecutó el programa principal.

- Se realiza una proporción matemática. Como se sabe el número de soldaduras totales, la subrutina *Lee registro de adquisición* lo grabó en una variable, y el número de soldaduras fallidas se puede calcular el porcentaje de fallos actual:

$$\frac{total\ \ soldaduras}{100} = \frac{soldaduras\ fallidas}{x}$$

- Se compara con el porcentaje umbral. Si se superó, se muestra el siguiente mensaje por pantalla y se devuelve en la variable *resultado_ctrl_fallos* un *STOP*, dejando además un piloto encendido en el dispositivo VA31. Seguidamente se finaliza la ejecución de la subrutina.



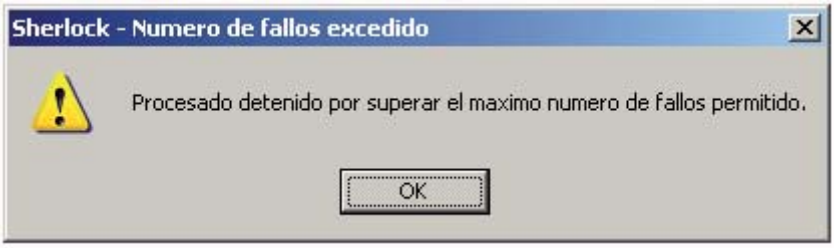
- Si por el contrario no se supera dicho umbral, se devuelve en *resultado_ctrl_fallos* un *GO* y se finaliza la ejecución de la subrutina.

En el caso de que se haya seleccionado que se detenga por número de fallos, lo primero que se comprueba es si la soldadura fue válida, *PASS*, o no, *FAIL*.

- Si la soldadura resultó ser válida, no se realiza ninguna operación, se termina la ejecución de la subrutina y se le pasa un *GO* a quien la invocó.

- Si por el contrario no es válida, *FAIL*, lo primero que se hace es incrementar en uno la variable que contiene el número de soldaduras, y seguidamente se compara con el máximo número de fallos permitidos.

- Si se supera dicho umbral, se muestra el siguiente mensaje de advertencia por pantalla y se devuelve en la variable *resultado_ctrl_fallos* un *STOP*, dejando además un piloto encendido en el dispositivo VA31. Seguidamente se finaliza la ejecución de la subrutina.



- Si por el contrario no se supera dicho umbral, se devuelve en *resultado_ctrl_fallos* un *GO* y se finaliza la ejecución de la subrutina.

5.4. Subrutina Comienza.

Esta subrutina se encarga de empezar un nuevo procesado desde el principio y registrar el resultado de las soldaduras procesadas en un archivo de texto. Si el procesado está ya comenzado y se invoca a esta función se vuelve a comenzar de cero y a reemplazar los resultados existentes por los nuevos resultados del procesado.

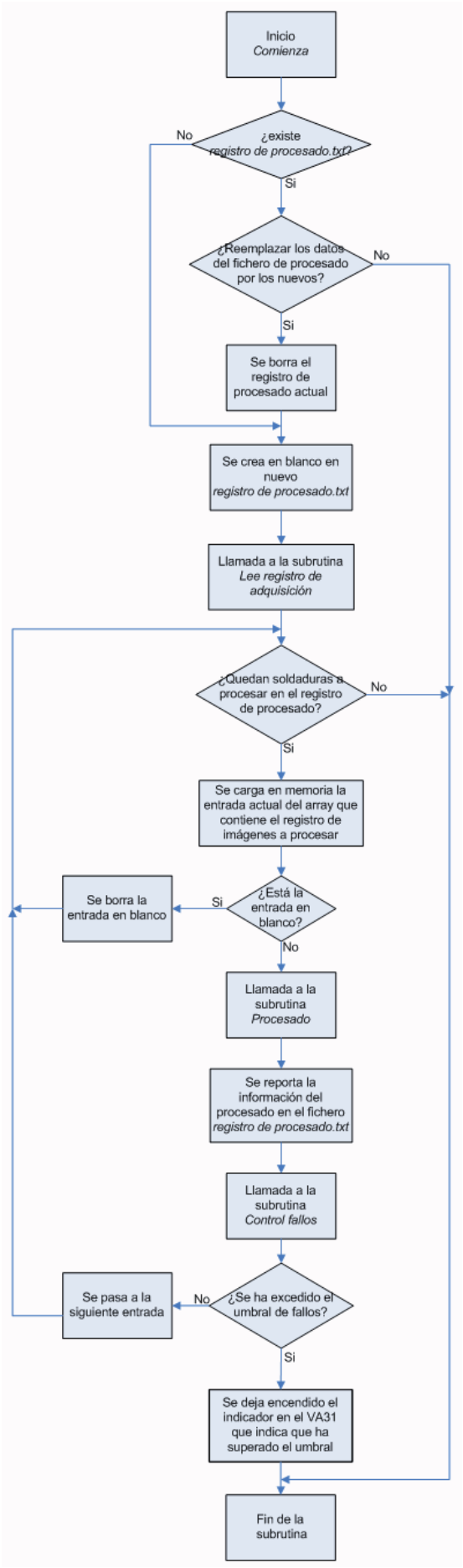
Las variables que recibe son las que se muestran a continuación:

- ✓ *directorio_procesado*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de procesado.

- ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de adquisición.
- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"numero"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.
- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

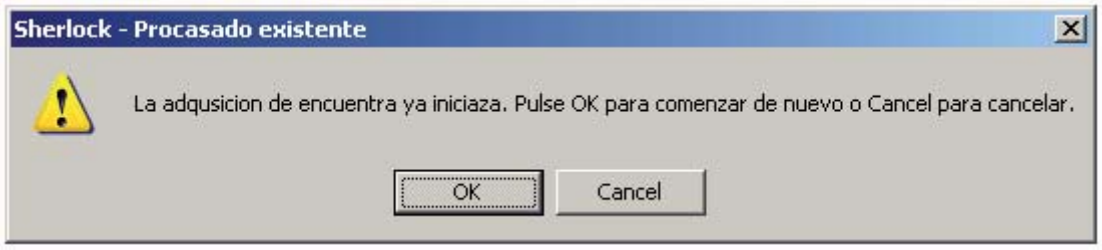
El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:


Inicio
Comienza
¿existe registro de procesado.txt?
No
Si
¿Reemplazar los datos del fichero de procesado por los nuevos?
No
Si
Se borra el registro de procesado actual
Se crea en blanco en nuevo registro de procesado.txt
Llamada a la subrutina Lee registro de adquisición
¿Quedan soldaduras a procesar en el registro de procesado?
No
Si
Se carga en memoria la entrada actual del array que contiene el registro de imágenes a procesar
¿Está la entrada en blanco?
Si
Se borra la entrada en blanco
No
Llamada a la subrutina Procesado
Se reporta la información del procesado en el fichero registro de procesado.txt
Llamada a la subrutina Control fallos
¿Se ha excedido el umbral de fallos?
No
Se pasa a la siguiente entrada
Si
Se deja encendido el indicador en el VA31 que indica que ha superado el umbral
Fin de la subrutina

Una vez ejecutada esta subrutina,

- Se comprueba si existe el archivo *Registro de procesado.txt*.

- Si existe dicho registro, se le informa al usuario, indicándole que si se vuelve a comenzar se reemplazará el actual registro de procesado por el nuevo, y se le pregunta si desea continuar:

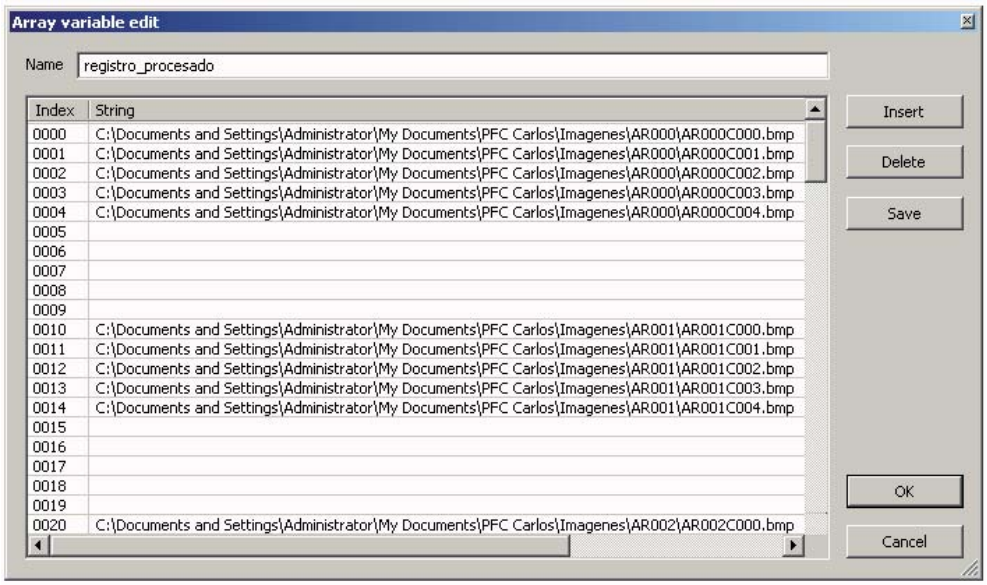


- Si se desea comenzar y reemplazar los resultados del registro, lo que hace la subrutina es borrar el contenido de registro de procesado, y una vez en blanco, escribe en él la cabecera *Registro de procesado* seguido por un retorno de carro,*\n*.

- Por el contrario, si no se desea continuar, se finaliza la subrutina sin modificar así dicho registro de procesado.

- Si el registro de procesado no existe, la subrutina no realiza ninguna operación, simplemente omite las acciones anteriormente descritas y comienza su ejecución a partir de aquí.

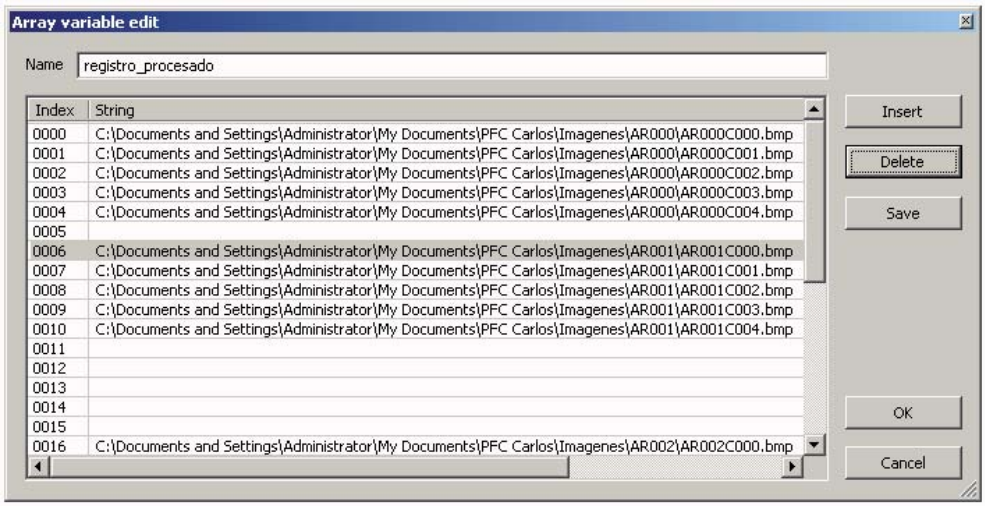
- Se invoca a la subrutina *Lee registro de adquisición*, para así tener ya en memoria el registro con todos los directorios ordenados de las fotos a procesar, al cual se le hará referencia a lo largo del capítulo como registro de imágenes a procesar, y tiene el siguiente aspecto:

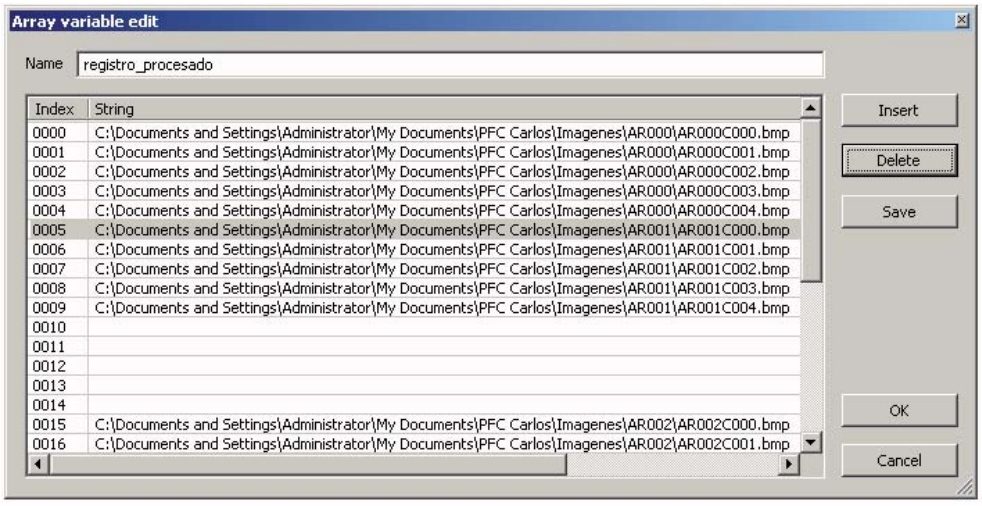


- Si entra en un bucle *while*, del cual se saldrá una vez que se hayan recorrido todas las posiciones del registro de imágenes de procesar. Para ello existe un índice a modo de puntero, el cual indica en el registro de imágenes a procesar cuál es la posición de la imagen que va a ser analizada. Este índice se inicializa a valor cero antes de entrar en el bucle y, cuando supera la longitud del registro de imágenes a procesar, se sale de este bucle *while*.

- Se extrae de este array la línea de texto que indica el índice y se comprueba si está en blanco.

- Si esta línea está en blanco se borra y se vuelve al principio del *while*. Por otro lado se recalcula la nueva longitud de esta variable y, además, se deja el índice sin modificar, ya que al suprimir la línea retroceden un puesto las demás:





- Si la línea cargada no está en blanco, se invoca a la subrutina *Procesado* y se le pasa como argumento este parámetro, ya que es el que se necesita para poder analizar la soldadura. Seguidamente, esta subrutina devolverá el resultado de la soldadura en forma de línea de texto, a la cual se le reemplaza el carácter "\" por "\\", ya que el programa cuando la guarda en el archivo sustituye la doble

contrabarra por la contrabarra simple, debido a caracteres reservados como el *\n* o *\t*, y se salva la información en el registro de procesado.

- Se invoca a la subrutina *Control de fallos*, a la cual se la pasa una variable en la que la subrutina *Procesado* grabó un *PASS* si la soldadura fue buena o un *FAIL* si la soldadura no era correcta.

- La subrutina *Control de fallos* devuelve un *GO* si no se superó el umbral de fallos que detiene este proceso, o un *STOP* si lo superó. Por ello, lo siguiente que realiza la subrutina *Comienza* es leer este parámetro, y si su valor es de *STOP* se enciende en el dispositivo VA31 un piloto que advierte de este fallo y se detiene la ejecución del programa.

- Si por el contrario fue un *GO*, se incrementa en uno el índice que indica la imagen a procesar y se vuelve al comienzo del *while*.

- Una vez procesadas todas las imágenes se termina la ejecución de esta subrutina.

5.5. Subrutina Rango.

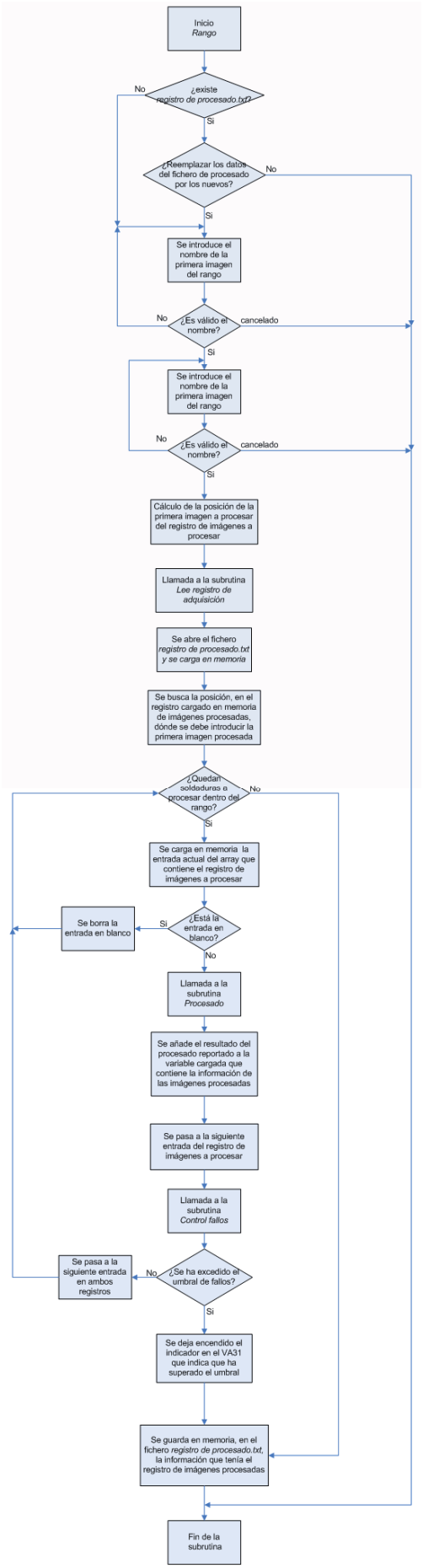
Esta subrutina procesa las imágenes comprendidas dentro de un rango que establece el usuario, registrando además el resultado del procesado en un archivo de texto. Las imágenes comprendidas dentro de ese rango se procesarán, y si ya se había realizado en el pasado los resultados viejos se reemplazarán por los nuevos.

Las variables que recibe son las que se muestran a continuación:

- ✓ *directorio_procesado*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de procesado.
- ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se encuentra el registro de adquisición.
- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"numero"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.
- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:


Inicio
Rango
¿existe registro de procesado.txt?
No
Si
¿Reemplazar los datos del fichero de procesado por los nuevos?
No
Si
Se introduce el nombre de la primera imagen del rango
¿Es válido el nombre?
No
cancelado
Si
Se introduce el nombre de la primera imagen del rango
¿Es válido el nombre?
No
cancelado
Si
Cálculo de la posición de la primera imagen a procesar del registro de imágenes a procesar
Llamada a la subrutina Lee registro de adquisición
Se abre el fichero registro de procesado.txt y se carga en memoria
Se busca la posición, en el registro cargado en memoria de imágenes procesadas, dónde se debe introducir la primera imagen procesada
¿Quedan soldaduras a procesar dentro del rango?
No
Si
Se carga en memoria la entrada actual del array que contiene el registro de imágenes a procesar
¿Está la entrada en blanco?
Si
Se borra la entrada en blanco
No
Llamada a la subrutina Procesado
Se añade el resultado del procesado reportado a la variable cargada que contiene la información de las imágenes procesadas
Se pasa a la siguiente entrada del registro de imágenes a procesar
Llamada a la subrutina Control fallos
¿Se ha excedido el umbral de fallos?
No
Se pasa a la siguiente entrada en ambos registros
Si
Se deja encendido el indicador en el VA31 que indica que ha superado el umbral
Se guarda en memoria, en el fichero registro de procesado.txt, la información que tenía el registro de imágenes procesadas
Fin de la subrutina

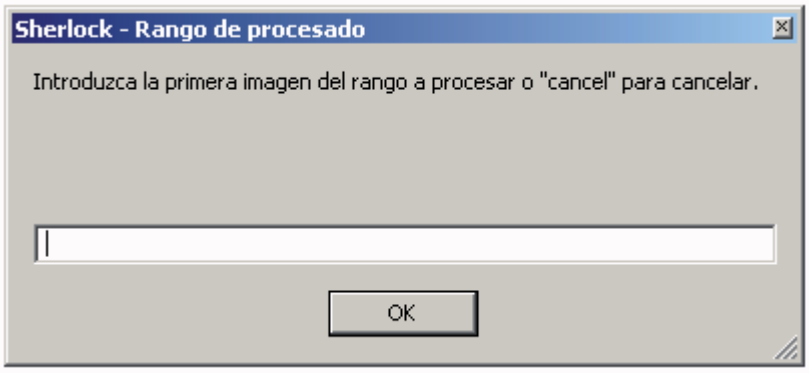
Una vez invocada esta subrutina,

Se comprueba si existe el registro de adquisición. Si existe, se le notifica al usuario que todos los resultados de las imágenes que se procesen sustituirán a los presentes, pudiendo elegir si se desea continuar o no:

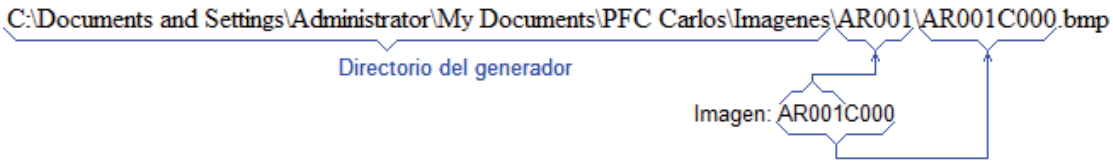
- Si el usuario desea seguir adelante se continuará ejecutando el programa. Si por el contrario, no desea continuar con el procesado se detendrá el programa.

- Si no existiese el registro de procesado, una vez invocada esta subrutina, continuaría a realizar lo descrito a partir de aquí.

- A continuación, hay una etiqueta llamada *inicio img1*, de la cual se verá su importancia más adelante. Seguidamente aparece un cuadro de diálogo donde se debe introducir el nombre de la primera imagen del rango o escribir la palabra "cancel" si se desea cancelar:

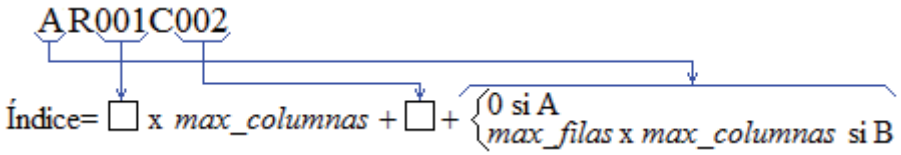


- Se construye el directorio completo de la imagen a partir del nombre que ha introducido el usuario por pantalla. Para ellos se coge la dirección del generador, se le añade una contrabarra y, como todas las fotos que están en la misma fila se almacenan en la misma carpeta, del nombre de la foto se extrae el nombre de la fila. A continuación, se añade otra contrabarra seguido por el nombre de la imagen introducido por el usuario. Finalmente se le añade la extensión *.bmp*, resultando así el directorio completo de la imagen, como se muestra a continuación:

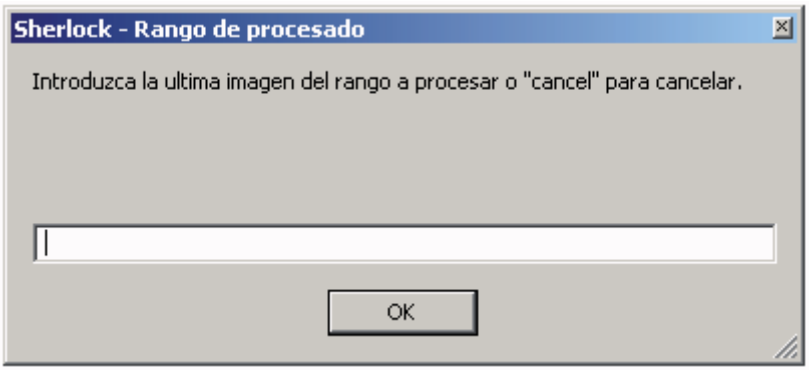


- Se comprueba, a partir del directorio anteriormente creado, la existencia de la imagen. Si la imagen no existe, aparecerá el siguiente mensaje de error y volveremos a la etiqueta anteriormente mencionada llamada *inicio img1*, para así volver a repetir este proceso hasta que el usuario introduzca un nombre correcto o cancele su ejecución escribiendo "cancel".

- Si la imagen existe, se crea un índice a modo de puntero, indicando en el registro de imágenes a procesar cuál es la primera imagen del rango a procesar. La generación del índice a partir de este nombre de imagen es idéntico al que se explicó en la subrutina *Lee registro de adquisición*, como se muestra a continuación:



- A continuación, hay una etiqueta llamada *inicio img2*, la cual indica que va a empezar la solicitud de la última imagen del rango a procesar. Para ello aparece el siguiente cuadro en pantalla que indica el nombre de dicha imagen:



- Se comprueba que la última imagen del rango sea igual o posterior a la inicial. Si la última imagen del rango es anterior a la inicial se mostrará el siguiente cuadro de error y se volverá a la etiqueta *inicio img2*, solicitando así su nombre de nuevo.

- Se comprueba que existe el nombre de la imagen, generando de la misma manera el directorio necesario para su comprobación.

- A partir del directorio anteriormente creado, se comprueba la existencia de la imagen. Si la imagen no existe aparecerá el siguiente mensaje de error y volveremos a la etiqueta llamada *inicio img2*, para así volver a repetir este proceso hasta que el usuario introduzca un nombre correcto o cancele su ejecución escribiendo "cancel".

- Si la imagen existe, se crea otro índice a modo de puntero, indicando en el registro de imágenes a procesar cuál es la última imagen del rango a procesar. La generación del índice a partir de este nombre de imagen es idéntico al explicado anteriormente para generar el de comienzo.

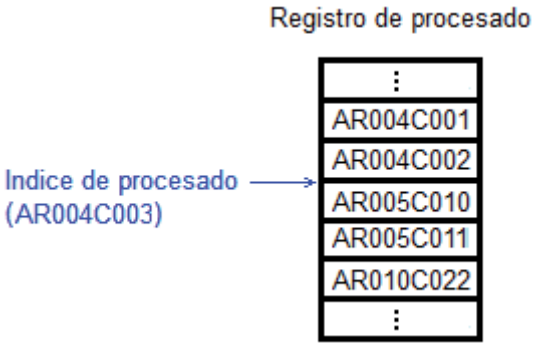
- Se invoca a la subrutina Lee registro de adquisición, para que genere el registro de imágenes a procesar. Seguidamente se abre el fichero *Registro de procesado.txt*, se lee y se cierra. El sentido de esta operación es para tener cargado en la memoria dicho registro e ir añadiendo en él los resultados de las nuevas imágenes procesadas. De dicho registro se eliminan las dos primeras filas, por carecer de información.

- Se crea e inicializa a cero otro índice, llamado índice de procesado, que indique en el registro de procesado dónde se debe guardar la imagen procesada, consiguiendo así que se almacenen en orden alfabético en correlación al nombre de la imagen

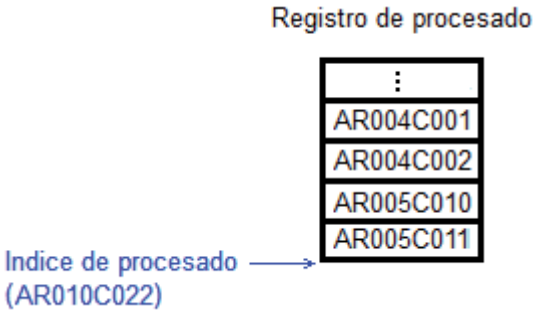
- Se entra en un bucle *while,* del cual se saldrá una vez hayan sido recorridas todas las posiciones del registro de procesado. Durante cada iteración del bucle carga la línea de registro de procesado que muestra el índice, por lo que si el índice vale cuatro se cargará la quinta línea del registro, y de ahí se extraen los 9

primero caracteres, que representan el nombre de la imagen a la que hace referencia esa línea de procesado.

Este bucle tiene por objetivo el buscar en qué posición del registro de procesado se debe introducir la siguiente imagen a procesar para que quede así ordenado alfabéticamente. Para ello, se compara el nombre de la foto introducida como imagen inicial del rango con el nombre extraído de la línea cargada del registro de procesado. Por ejemplo, si se introdujo como imagen inicial del rango AR004C003, se tendría lo siguiente:



De este bucle se sale si ha encontrado dicha posición o si se han recorrido todas las filas del registro de procesado cargado, en cuyo caso se saldrá tomando como valor del índice la última posición que tiene el registro, como se muestra a continuación:

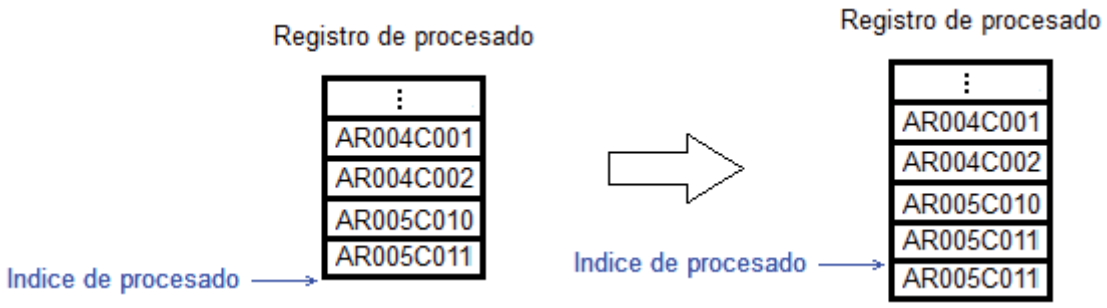


- Se entra en otro bucle *while*, del cual se saldrá cuando se hayan recorrido todas las imágenes comprendidas en este rango a procesar, o lo que es lo mismo, cuando el índice de la foto actual tome el mismo valor que el índice final.

- Permaneciendo dentro del bucle anterior, se carga la línea del registro de imágenes a procesar que indica el índice inicial.

- Se comprueba si esta línea cargada está en blanco. De estarlo, dicha línea se borra y se vuelve al principio del *while*. Por otro lado, se recalcula la nueva longitud de esta variable y, además, se deja el índice sin modificar, ya que al suprimir la línea retroceden un puesto las demás. Si por el contrario, contiene información, se invoca a la subrutina *Procesado*.

- Se comprueba que el índice de procesado donde se tienen que guardar las imágenes no señala fuera del array. En caso de apuntar fuera del array, lo que se hace es crear en el registro de procesado cargado una copia de la última imagen procesada, incrementando así en uno la longitud de dicho registro, como se muestra a continuación:



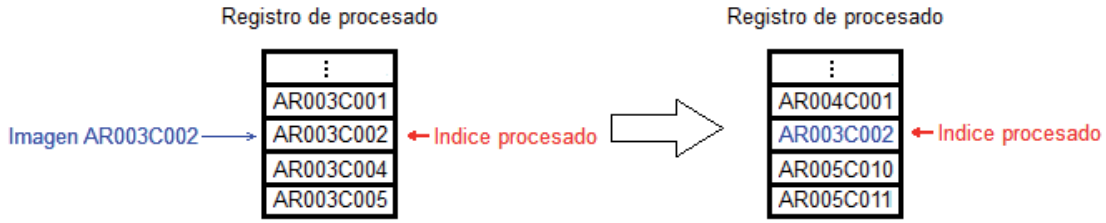
No se opta por añadir una nueva fila en blanco ya que esta línea después se comparará con la proveniente del registro de imágenes a procesar, y si está en blanco daría problemas al comparar.

- Se extrae del registro de imágenes procesadas la línea que indica el índice de procesado, y a continuación el nombre de la imagen. Esto se hace para saber si la nueva imagen a procesar reemplazará a una ya procesada anteriormente, actualizando así su resultado, o si por el contrario es nueva, añadiendo una nueva entrada a este registro.

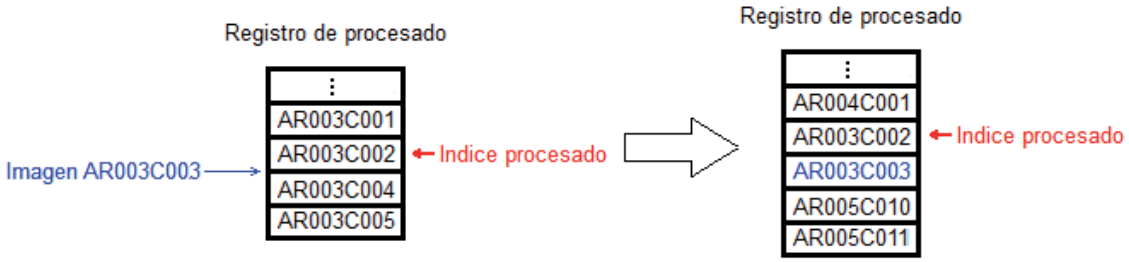
```
AR000C000: Processed: 15/08/11 at 18:47:45. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C000.bmp
```

-

Se compara la imagen a procesar con la que se encuentra en la posición que indica el índice de procesado. Si la imagen es igual se sustituye el viejo valor de procesado por el nuevo, como se muestra a continuación:



Si la imagen no es la misma, se inserta a continuación de la última el nuevo procesado, como se aprecia en el siguiente ejemplo:



- Se incrementan en uno tanto el valor del índice inicial como el del registro de procesado, señalando así la siguiente imagen a procesar y la posición a guardar. Seguidamente se invoca a la subrutina *Control de fallo*, y si se excedió el umbral de fallos se encenderá el piloto que indica esto y se saltará a una etiqueta llamada *guarda fichero (rango)*. Si no se superó el umbral, se vuelve al principio del bucle.

- Después de todas estas condiciones viene la anteriormente mencionada etiqueta *guarda fichero (rango)*. Seguidamente se abre el fichero *Registro de procesado.txt* en modo *appened=FALSE*, por lo que al abrirlo borrará su contenido. Se añade la línea introductoria *Registro de procesado* continuada por un retorno de carro.

- Se entra en un bucle *while*, del cual se saldrá una vez recorridas todas las posiciones de este registro cargado en memoria, y al que se la han ido añadiendo las nuevas imágenes procesadas. Dentro de este bucle se cargan por orden todas las filas que tiene este registro y se guardan una a una en el archivo que almacena el registro de procesado.

- Una vez recorridas todas las posiciones se cierra el fichero y se termina con la ejecución de la subrutina.

5.6. Subrutina Continua.

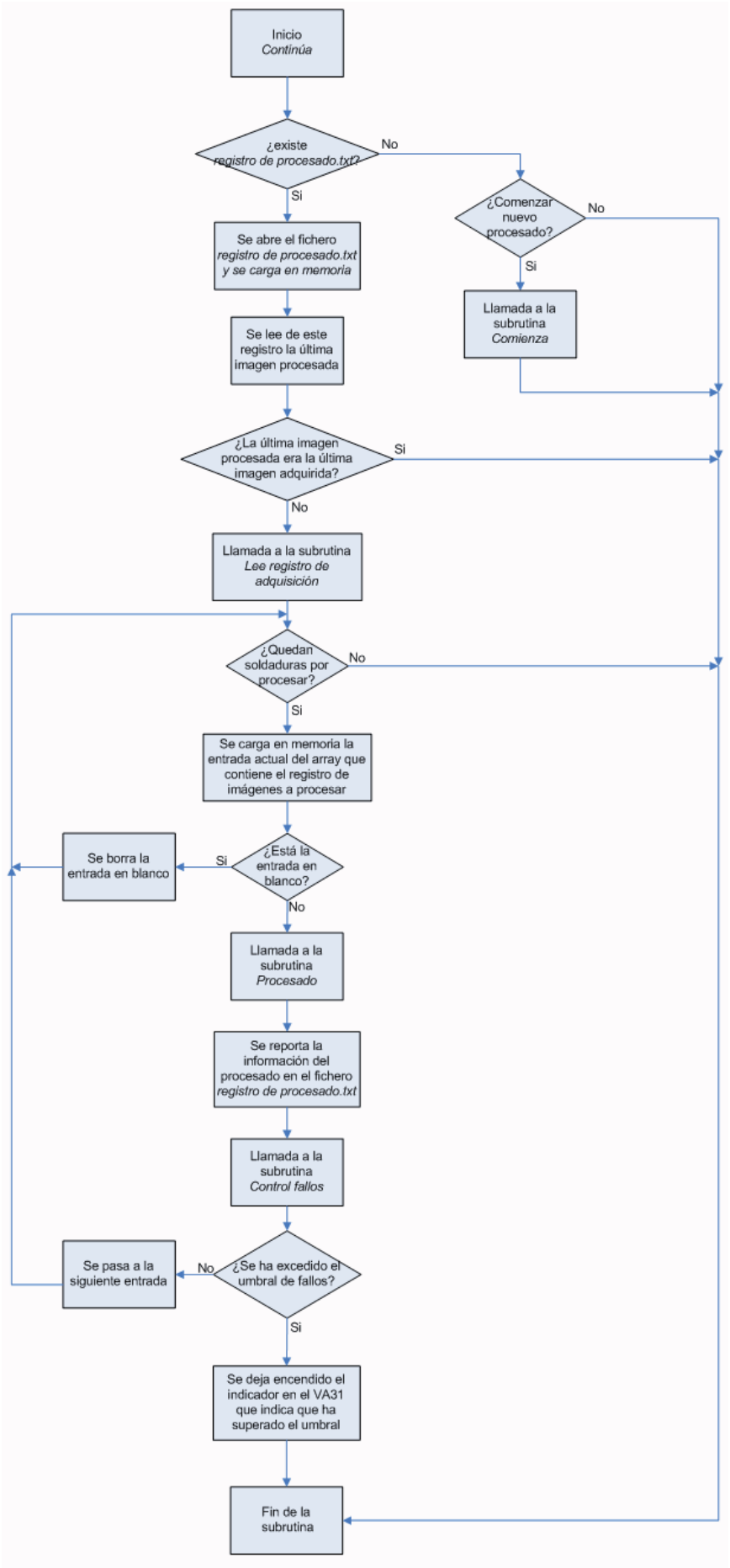
Esta subrutina se encarga de continuar un procesado ya empezado. Para ello carga la última soldadura procesada y a partir de ella continúa analizando el resto.

Las variables que recibe son las que se muestran a continuación:

- ✓ *directorio_procesado*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de procesado.
- ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de adquisición.
- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"número"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.
- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

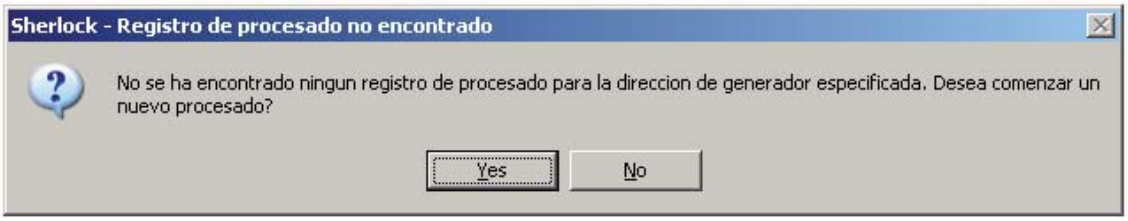
Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:


Inicio
Continúa
¿existe registro de procesado.txt?
No
Si
¿Comenzar nuevo procesado?
No
Si
Llamada a la subrutina Comienza
Se abre el fichero registro de procesado.txt y se carga en memoria
Se lee de este registro la última imagen procesada
¿La última imagen procesada era la última imagen adquirida?
Si
No
Llamada a la subrutina Lee registro de adquisición
¿Quedan soldaduras por procesar?
No
Si
Se carga en memoria la entrada actual del array que contiene el registro de imágenes a procesar
¿Está la entrada en blanco?
Si
Se borra la entrada en blanco
No
Llamada a la subrutina Procesado
Se reporta la información del procesado en el fichero registro de procesado.txt
Llamada a la subrutina Control fallos
¿Se ha excedido el umbral de fallos?
No
Se pasa a la siguiente entrada
Si
Se deja encendido el indicador en el VA31 que indica que ha superado el umbral
Fin de la subrutina

Una vez ejecutada la subrutina,

- Se comprueba la existencia del registro de procesado. De no existir implicaría que es un procesado nuevo, por lo que, mediante el siguiente cuadro, se le preguntaría al usuario si desea empezar un nuevo procesado.



- Si se pulsa "YES" se invoca a la subrutina *Comienza* y se termina la ejecución de esta subrutina. Si se pulsa "NO", se finaliza la presente subrutina.

- Si por el contrario, existiese el registro de procesado, se continuaría leyendo este registro y cargándolo en memoria. Seguidamente se lee la longitud de este registro cargado, para así poder acceder a la última posición del mismo, y obtener cuál fue la última imagen procesada. Más concretamente se le resta uno a la longitud, ya que el array se comienza a numerar desde cero.

- Se calcula la posición que ocupará, índice, en el registro de imágenes a procesar, para así poder continuar procesando imágenes. La forma de calcular este índice es la misma que la explicada en otras ocasiones.

- Se comprueba que la última foto procesada no sea la última imagen que había que procesar, en cuyo caso ya estaría completado el procesado y se le notificaría de la siguiente forma al usuario, finalizando así la ejecución de esta subrutina.

- De haber más imágenes a procesar, el siguiente paso es invocar a la subrutina *Lee registro de procesado*, creando así el array de imágenes a procesar.

- Se entra en un bucle *while*, del cual se saldrá una vez llegada a la última imagen a procesar. Lo primero que se hace dentro de este bucle es cargar la línea del registro de imágenes a procesar que nos indica el índice anteriormente calculado, el cual irá recorriendo desde la última imagen procesada hasta el final este registro de imágenes a procesar.

- Se comprueba si esta línea cargada está en blanco. De estarlo, dicha línea se borra y se vuelve al principio del *while*. Por otro lado se recalcula la nueva

longitud de esta variable y, además, se deja el índice sin modificar, ya que al suprimir la línea retroceden un puesto las demás.

- Si por el contrario la línea cargada no está en blanco, se invoca a la subrutina *Procesado* y se le pasa esta información para que proceda el procesado. Al resultado que nos devuelve esta subrutina a la que se la ha llamado, se le cambia el carácter "\" por "\\", recuérdese que el programa cuando guarda en el fichero de texto sustituye "\\" por "\", y se añade esta entrada al registro de procesado.

- Se llama a la subrutina *Control de fallos*, para así verificar si se superó el umbral máximo de fallos. Si se superó este umbral, se deja encendido el piloto del VA31 que indica que la soldadura no fue válida y se finaliza con la ejecución de este programa.

- En caso de no superar este umbral, se incrementa en uno el índice que indica cuál es la siguiente imagen a procesar y se vuelve al principio del *while*.

- Una vez que se han procesado todas las imágenes que faltaban, y no siendo superado ese umbral de fallos, se finaliza esta subrutina, y en consecuencia, el programa.

5.7. Subrutina completa.

Esta subrutina de encarga de completar un procesado existente. Debido a la posibilidad de procesar rangos de imágenes y dejar incompletos procesados, existe la posibilidad de que dentro de un generador existan imágenes procesadas y otras sin procesar estando todas mezcladas entre sí, pudiendo resultar molesto comprobar que imágenes no se procesaron para hacerlo manualmente.

De este problema expuesto surge el diseño de esta subrutina. Más concretamente se encarga de "coger" de memoria el registro de imágenes procesadas y mediante el registro de imágenes a procesar ver cuales le faltan para así completar este procesado.

Por otro lado, si en el registro de procesado lee que una imagen no se ha podido procesar porque no se encontró la imagen o el directorio, lo vuelve a intentar, por si se diera el caso de que el operario solucionó ese problema.

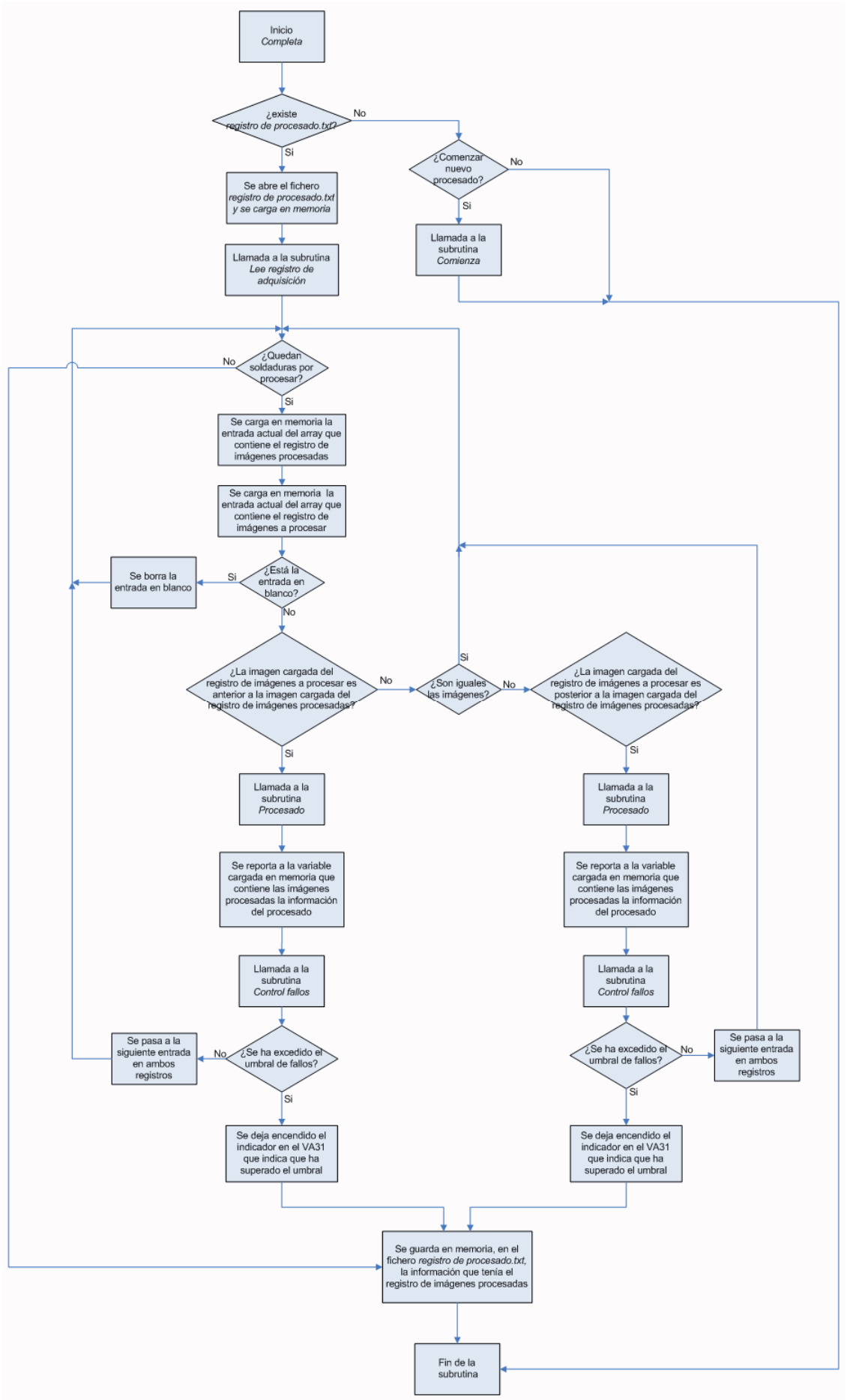
Las variables que recibe son las que se muestran a continuación:

- ✓ *directorio_procesado*: variable del tipo string que contiene el nombre completo del directorio donde se halla el registro de procesado.
- ✓ *directorio_adquisición*: variable del tipo string que contiene el nombre completo del directorio donde se encuentra el registro de adquisición.
- ✓ *tipo_detección*: variable del tipo string que contiene el tipo de detección, pudiendo valer *"porcentaje"* o *"número"*.
- ✓ *porcentaje_fallos*: variable del tipo numérico que contiene el valor de porcentaje de fallos que detendrá el programa.
- ✓ *Numero_fallos*: variable del tipo numérico que contiene el valor del número de fallos que detendrá el programa.

- ✓ *length_gen*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

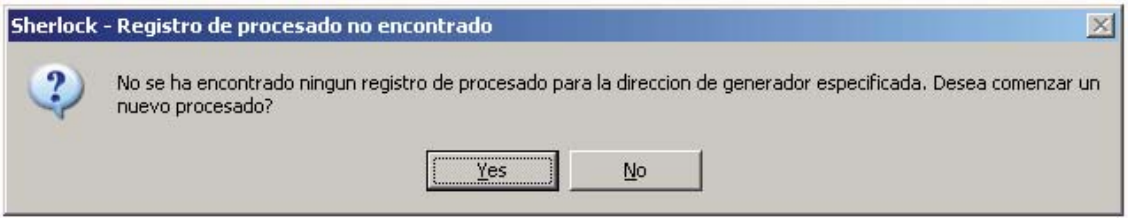
Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:


Inicio
Completa
¿existe registro de procesado.txt?
No
Si
¿Comenzar nuevo procesado?
No
Si
Se abre el fichero registro de procesado.txt y se carga en memoria
Llamada a la subrutina Comienza
Llamada a la subrutina Lee registro de adquisición
¿Quedan soldaduras por procesar?
No
Si
Se carga en memoria la entrada actual del array que contiene el registro de imágenes procesadas
Se carga en memoria la entrada actual del array que contiene el registro de imágenes a procesar
¿Está la entrada en blanco?
Si
Se borra la entrada en blanco
No
¿La imagen cargada del registro de imágenes a procesar es anterior a la imagen cargada del registro de imágenes procesadas?
No
¿Son iguales las imágenes?
Si
No
¿La imagen cargada del registro de imágenes a procesar es posterior a la imagen cargada del registro de imágenes procesadas?
Si
Si
Llamada a la subrutina Procesado
Llamada a la subrutina Procesado
Se reporta a la variable cargada en memoria que contiene las imágenes procesadas la información del procesado
Se reporta a la variable cargada en memoria que contiene las imágenes procesadas la información del procesado
Llamada a la subrutina Control fallos
Llamada a la subrutina Control fallos
¿Se ha excedido el umbral de fallos?
No
Se pasa a la siguiente entrada en ambos registros
Si
¿Se ha excedido el umbral de fallos?
No
Se pasa a la siguiente entrada en ambos registros
Si
Se deja encendido el indicador en el VA31 que indica que ha superado el umbral
Se deja encendido el indicador en el VA31 que indica que ha superado el umbral
Se guarda en memoria, en el fichero registro de procesado.txt, la información que tenía el registro de imágenes procesadas
Fin de la subrutina

Una vez ejecutada la subrutina,

- Se comprueba la existencia del registro de procesado. De no existir implicaría que es un procesado nuevo, por lo que, mediante el siguiente cuadro, se le preguntaría al usuario si desea empezar un nuevo procesado.



- Si se pulsa "YES" se invoca a la subrutina *Comienza* y se termina la ejecución de esta subrutina. Si se pulsa "NO", se finaliza la presente subrutina.

- Si por el contrario, existiese el registro de procesado, se continuaría leyendo este registro y cargándolo en memoria. Como las dos primeras líneas de este registro no contienen información relevante se suprimen, para así sólo disponer en este registro de información relacionada con el procesado.

- Se crean y se inicializan dos índices a modo de punteros. Uno de ellos indicará la posición actual en el registro de imágenes a procesar, y el otro mostrará la posición actual en el registro de procesado.

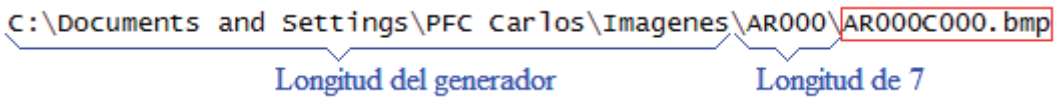
- Se invoca a la subrutina *Lee registro de adquisición*, generando así el array que contiene las imágenes a procesar, de las cuales una parte están ya procesadas.

- Se entra en un bucle *while*, del cual se saldrá una vez llegada a la última imagen a procesar.

- Dentro de este bucle, lo primero que se realiza es obtener la primera línea del registro de imágenes a procesar, recuérdese que el índice se inicializó a cero, y de ahí obtener el nombre de la imagen correspondiente.

- Para ello se comprueba si esta línea cargada está en blanco. De estarlo, dicha línea se borra y se vuelve al principio del *while*. Por otro lado, se recalcula la nueva longitud de esta variable y, además, se deja el índice sin modificar, ya que al suprimir la línea retroceden un puesto las demás.

- De contener información, se procede a sumarle a la longitud del directorio generador siete, y de ahí extraer los primeros nueve caracteres, como se muestra a continuación:

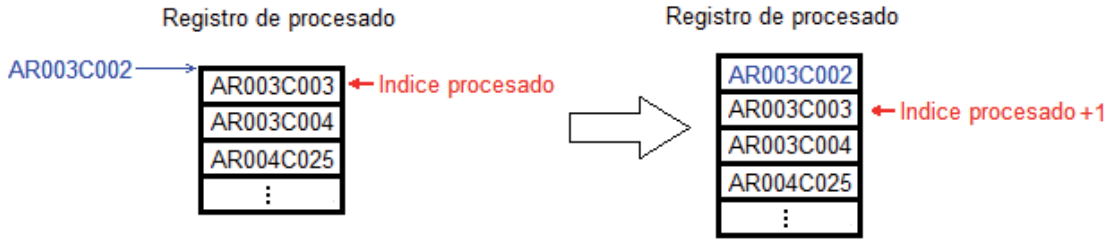


- De acuerdo al índice que indica la posición a cargar del registro de procesado, se carga la línea de este registro de procesado que indica el índice, para posteriormente extraer el nombre de la imagen a la que corresponde dicha entrada en el registro. Como el nombre de la imagen es lo primero que contiene esta línea, basta con extraer los primeros nueve caracteres:

```
AR000C000: Processed: 15/08/11 at 18:47:45. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C000.bmp
```

- Una vez que se tienen los dos nombres se comparan, para así poder determinar el orden alfabético. Dependiendo del resultado de la comparación de dan varios casos.

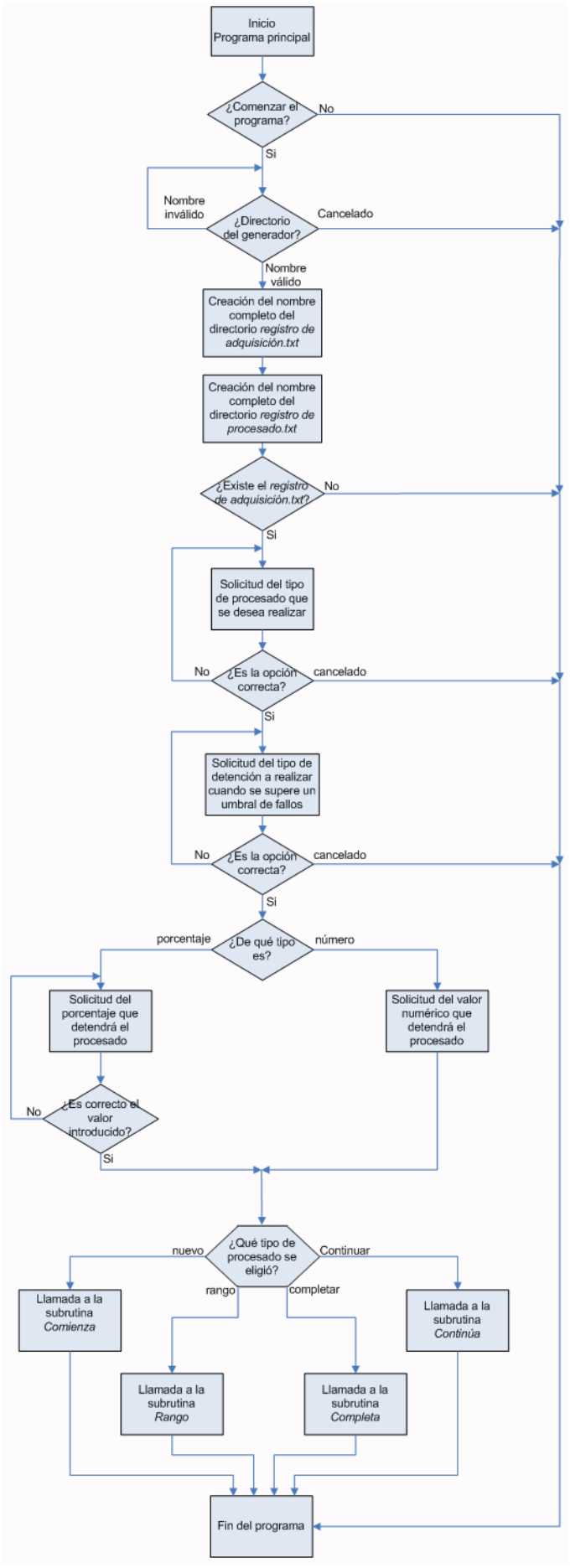
- Si la imagen de adquisición es anterior a la de procesado se carga la línea a procesar, actualmente señalada por el índice en el registro de imágenes a procesar, y se invoca a la subrutina *Procesado*. El resultado devuelto por esta subrutina se inserta en el registro de procesado cargado en memoria, de la siguiente manera:



- Se incrementa en uno cada uno de los dos índices y se recalcula la longitud del registro de procesado cargado en memoria. Seguidamente se invoca a la subrutina *Control de fallo*, y si se excedió el umbral de fallos se encenderá el piloto que indica esto y se saltará a una etiqueta llamada *guarda fichero (completa)*, de la cual se verá mas adelante su función. Si no se superó el umbral se vuelve al principio del bucle.

- Si la imagen de adquisición es igual a la de procesado y no hay error se incrementan en uno el puntero del registro de imágenes a procesar y, si no se está apuntando a la última posición del índice de procesado, también se incrementa en uno el índice del registro de procesado cargado. Si se está apuntando ya a esta última posición mantiene el índice su valor sin ser modificado. Además, no se realiza ninguna operación, ya que esto indica que la imagen estaba ya procesada.

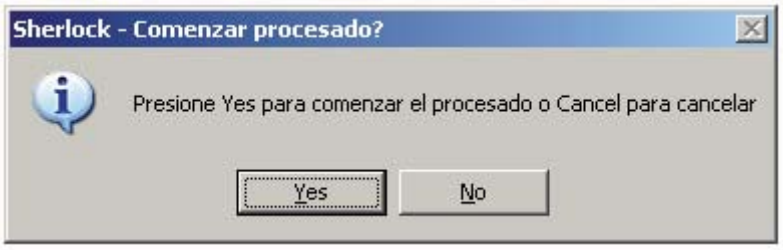
- Si la imagen de adquisición es igual a la de procesado y hay errores se repite el procesado. Para ello se carga la imagen a procesar e invoca a la subrutina *Procesado*. A continuación, se comprueba si se superó el umbral de fallos permitidos llamando a la subrutina *Control de fallo*, y si se excedió el umbral de fallos se encenderá el piloto que indica esto y se saltará a una etiqueta llamada *guarda fichero (completa)*. Si no se superó el umbral se incrementa en uno cada índice y se vuelve al principio del bucle.


Inicio
Programa principal
¿Comenzar el programa?
No
Si
Nombre inválido
¿Directorio del generador?
Cancelado
Nombre válido
Creación del nombre completo del directorio registro de adquisición.txt
Creación del nombre completo del directorio registro de procesado.txt
¿Existe el registro de adquisición.txt?
No
Si
Solicitud del tipo de procesado que se desea realizar
No
¿Es la opción correcta?
cancelado
Si
Solicitud del tipo de detención a realizar cuando se supere un umbral de fallos
No
¿Es la opción correcta?
cancelado
Si
porcentaje
¿De qué tipo es?
número
Solicitud del porcentaje que detendrá el procesado
Solicitud del valor numérico que detendrá el procesado
No
¿Es correcto el valor introducido?
Si
nuevo
¿Qué tipo de procesado se eligió?
Continuar
rango
completar
Llamada a la subrutina Comienza
Llamada a la subrutina Continúa
Llamada a la subrutina Rango
Llamada a la subrutina Completa
Fin del programa

Una vez ejecutado el programa,

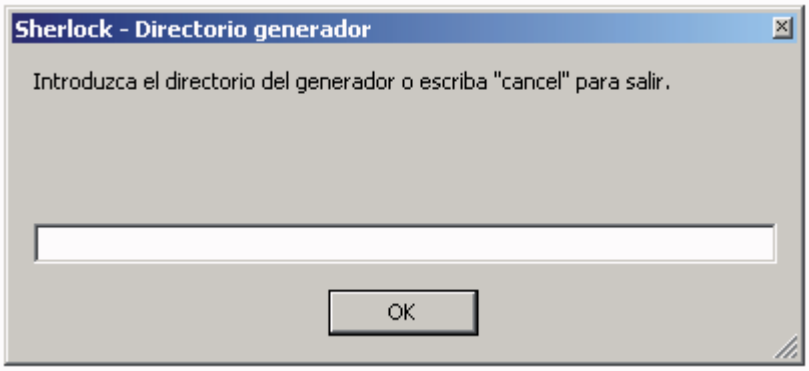
- Se inicializan a *FALSE* todas las salidas digitales, apagando así cualquier piloto o indicador que se pudiese quedar encendido de un procesado anterior.

- Se muestra en pantalla un mensaje preguntando si se desea empezar el procesado o por el contrario desea cancelarlo, como se muestra a continuación:

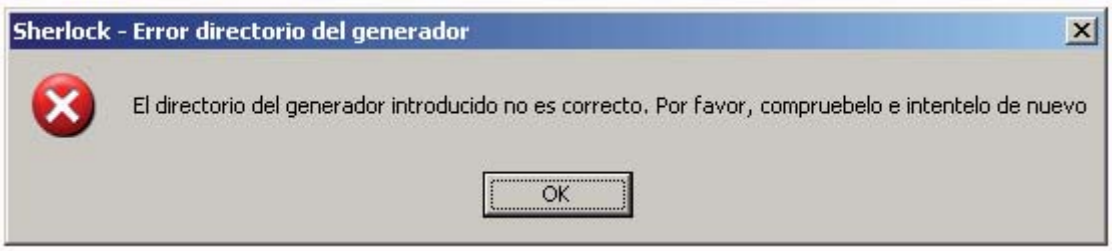


- Si se pulsa “Yes” se continuará ejecutando el resto de código, pero si por el contrario se pulsó “No” se para la ejecución del mismo.

- Se entra en un bucle *while*, del cual sólo se saldrá si se introduce un directorio de generador correcto o se escribe la palabra “cancel”. Una vez dentro, el programa pregunta por la dirección del generador, ya que es donde se encuentran las imágenes guardadas y el registro de adquisición, dando también opción a cancelar la ejecución escribiendo “cancel”:



- Se comprueba su existencia. De no existe el generador, se mostrará el siguiente mensaje por pantalla, y se volverá al principio del bucle *while*, volviendo a preguntar por el directorio del generador y permaneciendo así dentro de este bucle hasta que se cumpla una de las condiciones anteriormente explicadas.



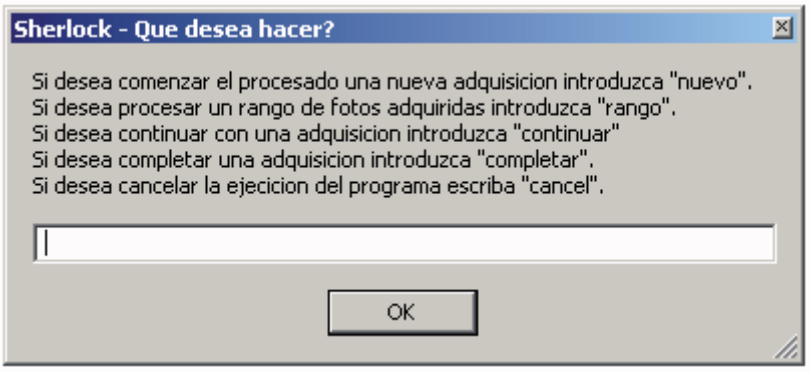
- Se crea el directorio de adquisición, necesario para posteriormente cargarlo y saber qué fotos han sido procesadas, y por otro lado se crea la dirección del registro de procesado que nacerá más adelante. Además se introduce en una variable la longitud (número de caracteres) que tiene este generador, ya que será necesario más adelante para varias subrutinas.

- Se comprueba que existe el registro de adquisición. De no existir este se mostrará el siguiente mensaje por pantalla y se finalizará su ejecución.

- Si la comprobación fue correcta y existe el registro de adquisición, lo siguiente que se realiza es preguntar qué tipo de procesado se desea realizar, existiendo cuatro posibilidades:

  - ✓ Comenzar un nuevo procesado.
  - ✓ Procesar un rango de fotos determinado por el usuario.
  - ✓ Continuar un procesado que estaba incompleto.
  - ✓ Completar un procesado, procesando todas aquellas soldaduras que faltan.

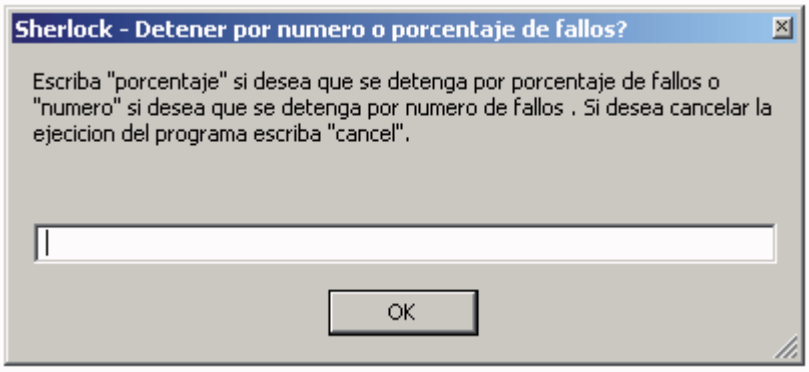
Si no se desea realizar ninguna opción también existe la posibilidad de cancelar, como se aprecia en el cuadro de diálogo:



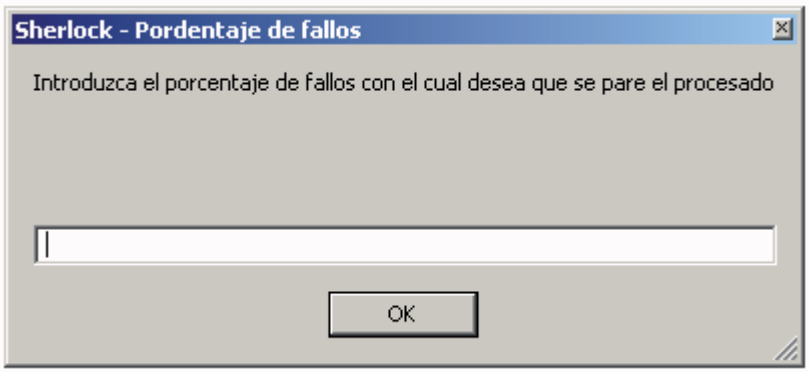
- Si se introduce alguna otra opción que no está incluida, este cuadro de diálogo aparecerá continuamente hasta que se le introduzca de forma correcta una de las opciones existentes. Si por el contrario se desea abortar la ejecución basta con escribir el comando “cancel” en el diálogo.

- Si no se cancela la ejecución del programa, aparece otro cuadro de diálogo, en el cual se pregunta por el tipo de detección a la hora de parar la ejecución del programa cuando se supere cierta cantidad de fallos, pudiendo elegir entre que

se detenga por número de fallos o por porcentaje de fallos. Al igual que antes, si se introduce “cancel” se detiene la ejecución del programa,

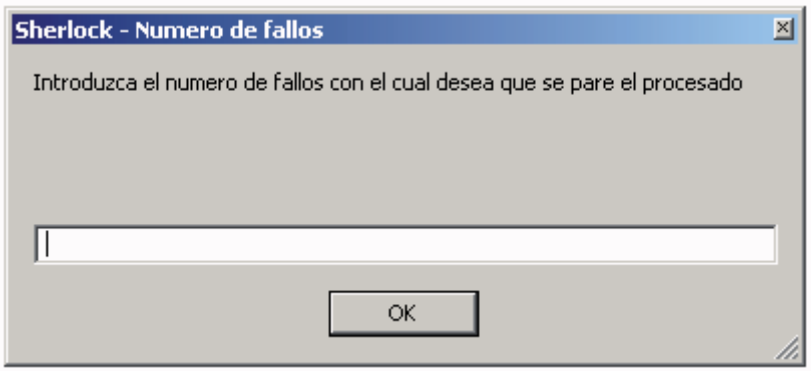


- Se pregunta por el número de fallos o el porcentaje de fallos concretos que hacen que el programa se detenga. Si se ha elegido que se detenga por porcentaje de fallos se obtendrá en siguiente diálogo:



Este cuadro de diálogo aparecerá mientras no se introduzca un número mayor o igual que cero y menor o igual que cien, apareciendo cada vez que se introduzca un valor fuera de este rango el siguiente mensaje:

- Si por el contrario se escogió por número de fallos se obtendrá este otro diálogo:



Se inicializa a cero la variable que indica el número de fallos y dependiendo de la opción escogida se carga una de las cuatro subrutinas existentes.

CAPÍTULO 6

Desarrollo del programa de visión de resultados

DESARROLLO DEL PROGRAMA DE VISIÓN DE RESULTADOS

1. Introducción.

Una vez adquirida y procesada la imagen, el siguiente paso a realizar es mostrar por pantalla las imágenes junto con los resultados del procesado, pudiendo así consultar los resultados del mismo contrastándolos con la imagen adquirida.

El objetivo de este programa es facilitar al operario la consulta de imágenes adquiridas, consiguiendo su rápida y cómoda inspección. Para ello, en este capítulo se ha desarrollado un programa y un conjunto de subrutinas que permiten visualizar estos resultados, pudiendo elegir el tipo de imágenes que se quieren mostrar.

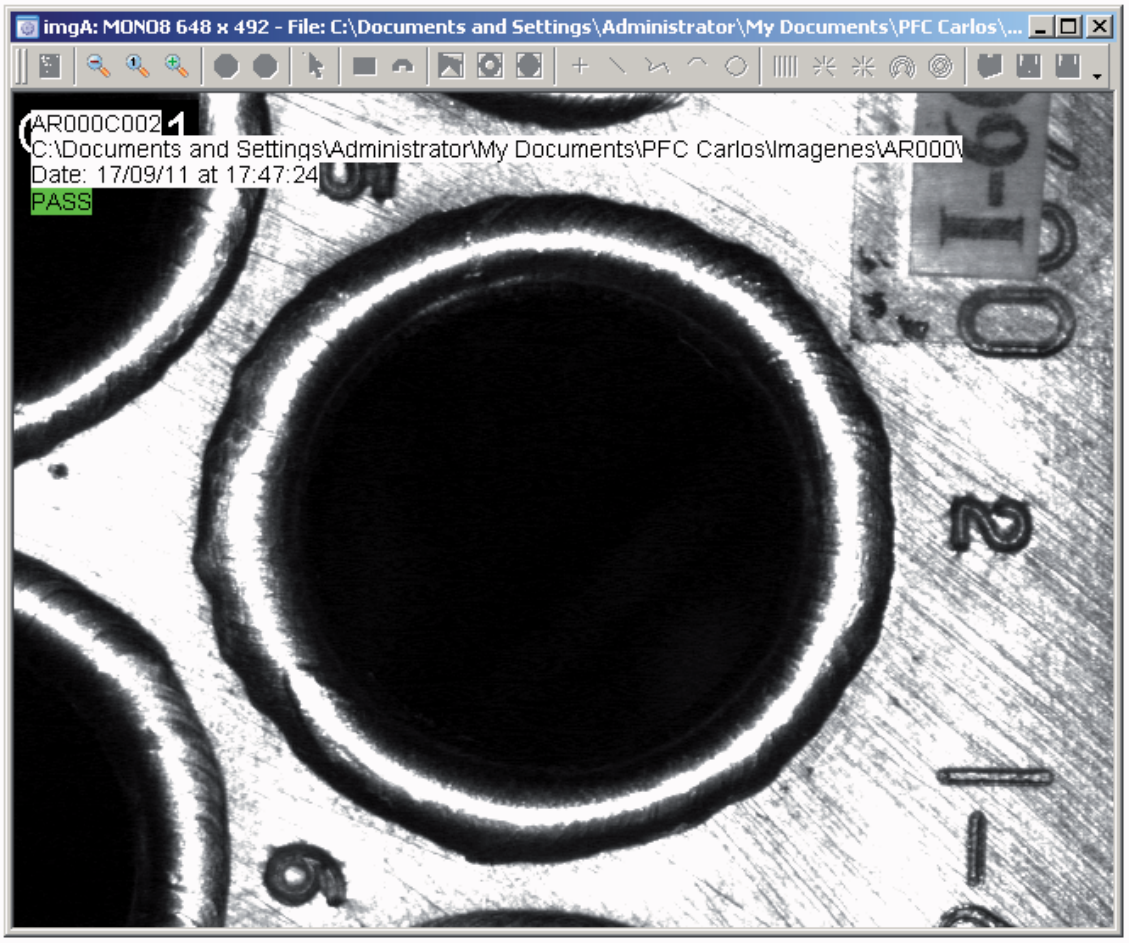
2. Funcionalidades del programa de visión de resultados

Este programa muestra por pantalla una imagen adquirida junto con la información correspondiente a su procesado.

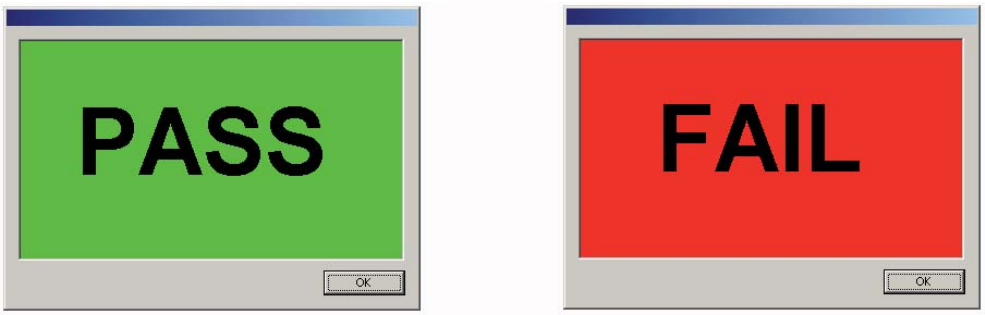
El funcionamiento consiste en ir leyendo el registro de procesado, y para cada imagen en particular, cargar la imagen de disco y mostrar sobre ella los principales parámetros del procesado. Más concretamente se van a mostrar los siguientes datos:

- ✓ Nombre de la imagen.
- ✓ Directorio donde se halla guardada.
- ✓ Fecha y hora de cuándo se realizó el procesado.
- ✓ Información de si es válida o no.

Toda esta información se va a mostrar en la parte superior izquierda de la imagen, en un sitio que no cubra parte de la soldadura expuesta en pantalla. Además, para que se pueda leer de una forma legible, se van a mostrar el nombre de la imagen, el directorio y la fecha y hora del procesado en letra de color negro sobre fondo oscuro, y el estado de la soldadura se indicará en letra negra sobre fondo verde o rojo, dependiendo de si la soldadura es válida o no. La imagen descrita tendrá el siguiente aspecto:



Además de lo anteriormente expuesto, junto a la ventana de la imagen procesada, se va a mostrar otra ventana que indica si la soldadura es válida o no. Para pasar de visualizar un resultado de procesado concreto al siguiente, basta con pulsar el botón "OK" que aparece en este cuadro, y automáticamente se pasará a visualizar la siguiente imagen. Recuérdese que estos cuadros mencionados tenían el siguiente aspecto:

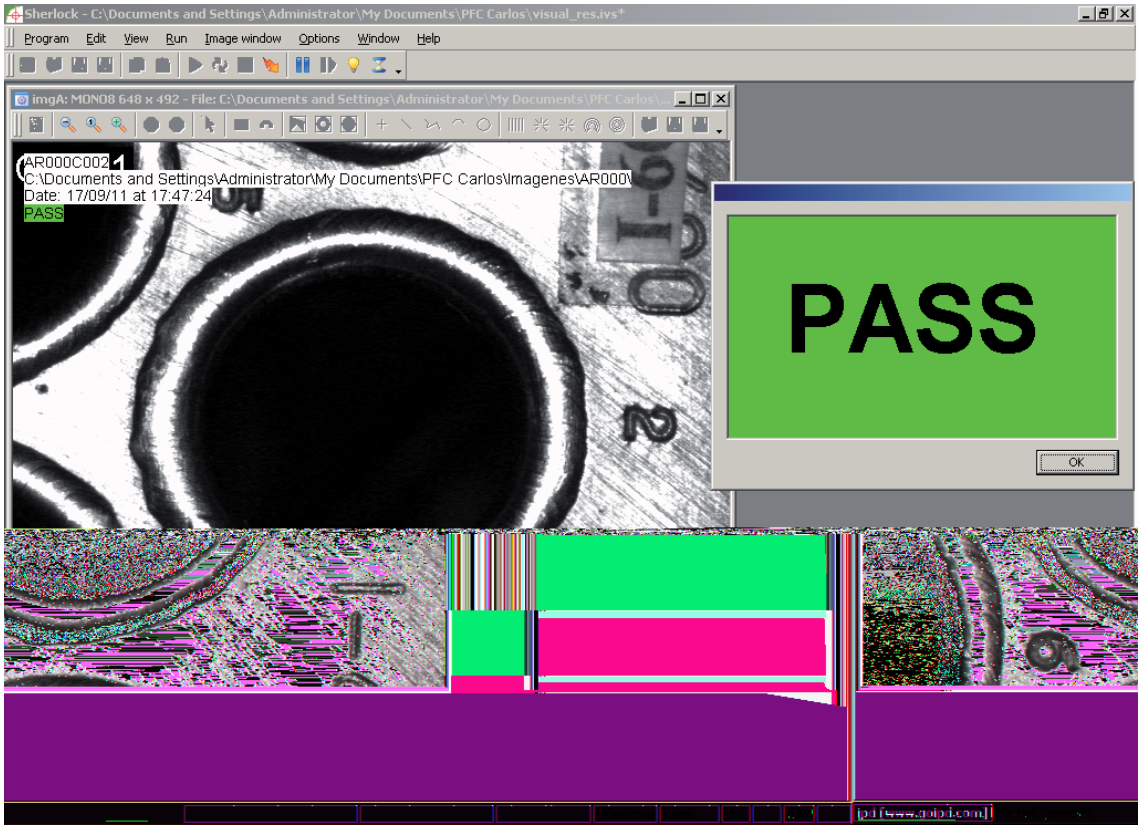


Puede ocurrir que visualizando los resultados de un procesado concreto aparezca un error en el registro de procesado. Recuérdese que se podían dar estos dos errores:

```
AR000C017: ERROR: Directorio no encontrado.
AR000C005: ERROR: imagen no encontrada.
```

De ocurrir esto, se le notificará al usuario el error mediante una ventana de error, en la cual se muestra el nombre de la imagen que posee el fallo, como se indica a continuación:

Finalmente, lo que se obtendrá en la pantalla del programa es lo mostrado a continuación:



3. Estructura del programa de visión de resultados.

El programa de visión de resultados está compuesto por un programa principal y cinco subrutinas, las cuales están jerarquizadas en diferentes niveles.

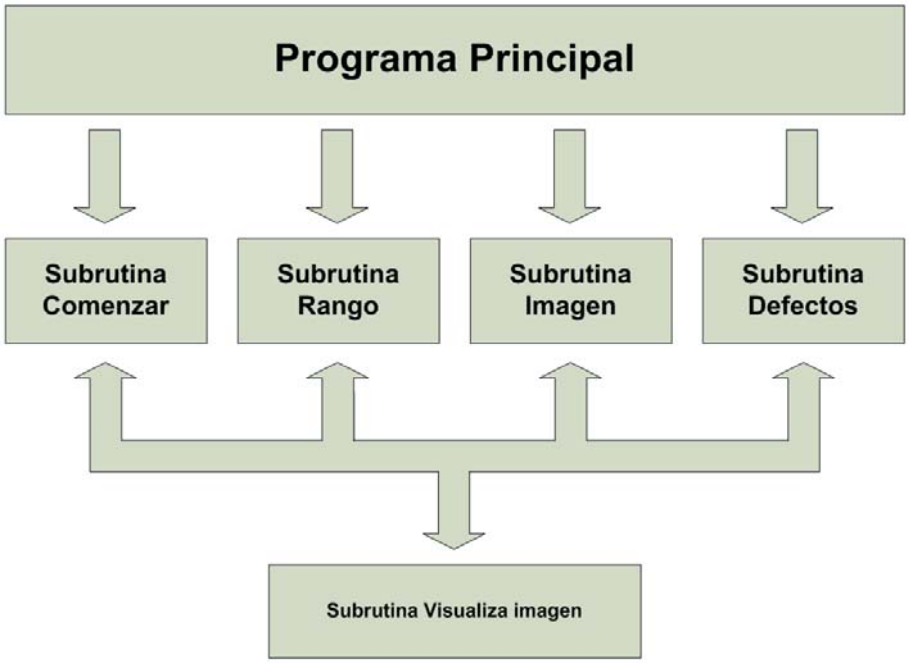
En un primer nivel se encuentra el programa principal, que es el que coordina e invoca a las subrutinas de un segundo nivel. Este programa principal realiza una labor de gestión a alto nivel. No se encarga de nada de lo que es la visualización de los resultados en sí.

Por debajo, en un segundo nivel hay cuatro subrutinas, encargadas de, por un lado, ejecutar el tipo de visualización haya escogido el usuario, y por otro coordinar e invocar

a la subrutina de tercer nivel. Estas subrutinas son las encargadas de decir qué imágenes se van a mostrar por pantalla y cuáles no.

En un tercer y último nivel se encuentran una subrutina, que se encarga de realizar todo lo relacionado con la visualización de resultados, como es cargar la imagen, comprobar que los directorios existen y demás gestiones de esta índole. Esta subrutina no realiza ninguna labor de gestión o coordinación; se limita a recibir los parámetros de entrada, procesarlos y retornar el o los resultados oportunos.

A continuación se muestra una imagen ilustrativa que representa la estructura del programa desarrollado. Las flechas indican cómo se relaciona el programa principal y las subrutinas entre sí, así como el sentido de comunicación, que puede ser bidireccional o unidireccional.



Otro aspecto importante a comentar son las variables compartidas. Por ello, en esta última parte de este presente capítulo se van a explicar las variables que comparte el programa principal con cada una de las cuatro subrutinas de segundo nivel, así como las que comparten las subrutinas de segundo nivel con la de tercer nivel, detallando su nombre, tipo y descripción. Las variables que comparte el programa principal con cada una de las subrutinas de segundo nivel son las mismas para todas ellas.

- Programa principal y subrutina *Comenzar*, subrutina *Rango*, subrutina *Imagen* y subrutina *Defectos*:

  - ✓ *registro_procesado*: variable del tipo array de string, que contiene la información de las imágenes procesadas.
  - ✓ *long_reg_pro*: variable del tipo numérico contiene el número de entradas de la variable *registro_procesado*.
  - ✓ *long_dir_reg_pro*: variable del tipo numérico que contiene el número de caracteres del directorio del generador.

- Subrutina *Comenzar*, subrutina *Rango*, subrutina *Imagen* y subrutina *Defectos* y la subrutina *Visualiza Imagen*:

  - ✓ *resultado_procesado*: variable del tipo string que contiene la línea de información resultante del procesado de una imagen concreta.
  - ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

4. Programa de visión de resultados.

Este último apartado del sexto capítulo se va a centrar en explicar cada una de las subrutinas desarrolladas así como el programa principal.

Antes de proceder a su explicación hay que resaltar la importancia de que para poder realizar una correcta visualización es necesario que esté el registro de procesado, ya que sin él no se puede realizar el visionado de resultados.

Para facilitar la comprensión del programa, se va a empezar a explicar la subrutina de tercer nivel, continuadas por las de segundo y finalizada por el programa principal.

4.1. Subrutina Visualiza imagen.

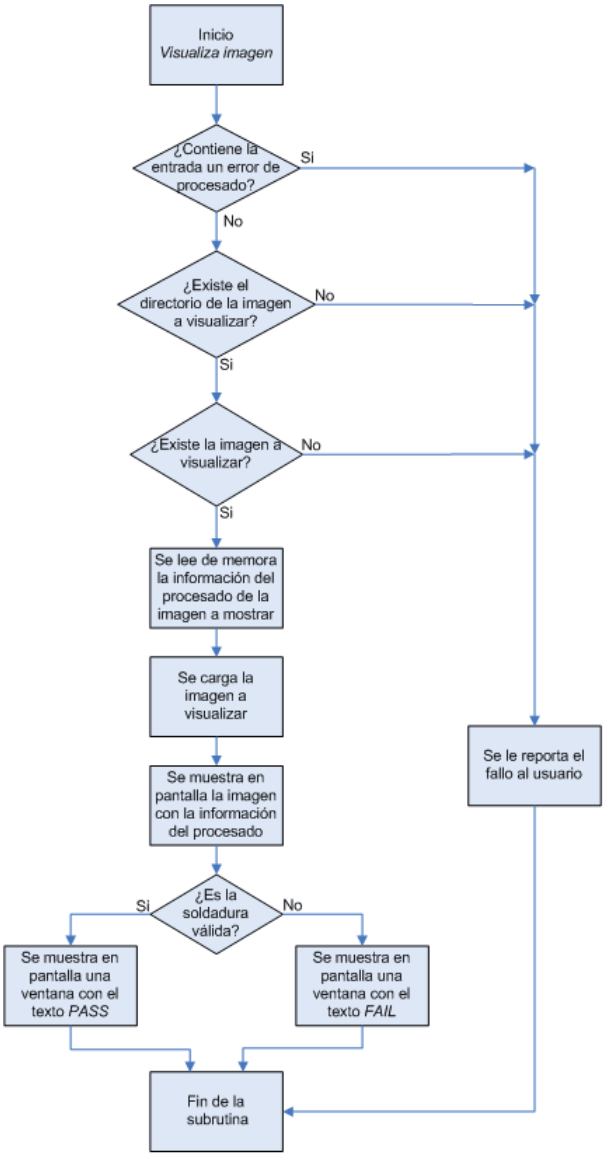
Esta subrutina es la encargada de mostrar en pantalla la información de una imagen que ha sido ya procesada. La forma y manera de pintar en pantalla estos resultados se ha explicado en el punto dos de este capítulo, pero cabe comentar que esta subrutina recibe como argumento de entrada la línea de información del procesado de una imagen concreta, y a partir de ella, extrae la información y la muestra en pantalla junto con la imagen a la que pertenece esa información.

Las variables que recibe son las que se muestran a continuación:

- ✓ *resultado_procesado*: variable del tipo string que contiene la línea de información resultante del procesado de una imagen concreta.
- ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, únicamente muestra por pantalla el resultado obtenido, sin retornar ningún dato a la subrutina de segundo nivel que la invocó.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada la subrutina,

- Se lee de la línea que contiene la información de procesado el nombre de la imagen que se va a mostrar por pantalla.

- Se comprueba que la línea no contiene ningún error de procesado. Para ello, se leen los caracteres doce, trece, catorce, quince y dieciséis, como se muestra a continuación:

- Si forman la palabra “ERROR”, se genera, junto con el nombre de la imagen que se debería mostrar, un mensaje de error y se le notifica al usuario, finalizando además la ejecución de la subrutina:

- Si la entrada no presenta ningún error, se extrae de ella el directorio de la imagen a la que hace referencia, como se muestra a continuación, y se comprueba su existencia.

```
AR000C001: Processed: 15/08/11 at 18:47:45. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

- Si este directorio no existe se le notificará al usuario mediante el error que se muestra a continuación y se finaliza la ejecución de la subrutina:

- Si existe el directorio, se sustituyen en él las contrabarras por dobles contrabarras y se guardan en una variable. Esto se hace, como se comentó en otros capítulos, para evitar la confusión con caracteres reservados como "\n" o "\t".

- Se carga el nombre del directorio completo, como se muestra a continuación, y se comprueba la existencia de la imagen en disco.

```
AR000C001: Processed: 15/08/11 at 18:47:45. Status: FAIL. Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

- Si no existe la imagen, se le notificará al usuario mediante el siguiente mensaje de error y se finaliza la ejecución de la subrutina:

- Se lee de la línea que contiene la información de procesado la fecha y hora del mismo, así como su resultado, pudiendo valer *PASS* o *FAIL*.

```
AR000C001: Processed: 15/08/11 at 18:47:45 Status: FAIL Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

- Se carga de disco la imagen y se muestra por pantalla. A continuación, y en otra instrucción diferente, se añade a esta imagen la información del procesado, que como se comentó al principio de este capítulo consta del nombre de la imagen, el directorio donde se halla, la fecha y hora y si la soldadura es válida o no, indicándolo mediante un PASS si es correcta o un FAIL si presenta anomalías.

- Finalmente, y dependiendo de si la soldadura es válida o no, se mostrará mediante una nueva ventana un mensaje de PASS o FAIL, dando así por finalizada esta subrutina.

4.2. Subrutina Comenzar.

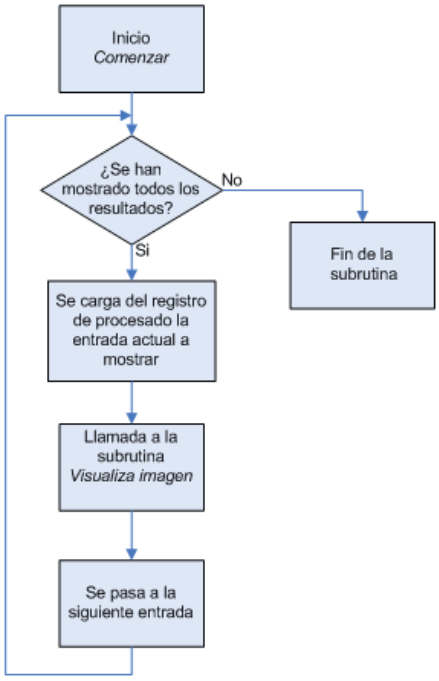
Esta subrutina se encarga de mostrar de principio a fin, siempre que no se cancele antes su ejecución, los resultados de todas las imágenes que han sido procesadas y que así lo contempla el registro de procesado.

Las variables que recibe son las que se muestran a continuación:

- ✓ *registro_procesado*: variable del tipo array de string, que contiene la información de las imágenes procesadas.
- ✓ *long_reg_pro*: variable del tipo numérico contiene el número de entradas de la variable *registro_procesado*.
- ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada la subrutina,

- Se inicializa a cero una variable, que a modo de índice, indicará en todo momento la posición que ocupa en el registro de procesado cargado en memoria la imagen que se está visualizando. Esta variable se llama *indice_com*.

- Se entra dentro de un bucle *while*, del cual se saldrá una vez que se hayan mostrado todos los resultados de las imágenes procesadas. Más concretamente, se saldrá del bucle una vez que la variable *indice_com* indique que ya se ha llegado a la última posición del registro de procesado, y, una vez ocurra esto, se finaliza la ejecución de esta subrutina.

- Dentro del bucle, se lee del registro de procesado la línea de información que indica el índice *indice_com*.

- Se llama a la subrutina *Visualiza imagen*, y una vez terminada ésta su ejecución, se incrementa en uno el índice *indice_com* y se vuelve al principio del bucle.

4.3. Subrutina Rango.

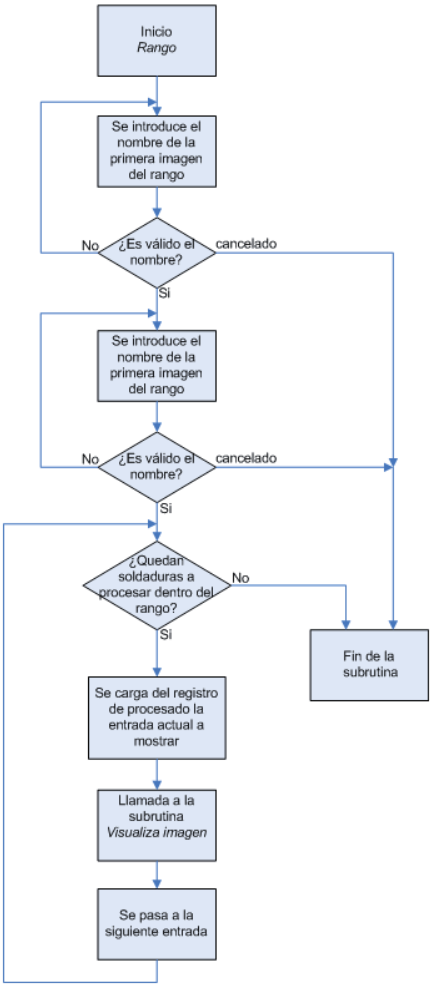
Esta subrutina se encarga de mostrar por pantalla el resultado de un rango de imágenes que han sido ya procesadas, siendo este rango definido por el usuario.

Las variables que recibe son las que se muestran a continuación:

- ✓ *registro_procesado*: variable del tipo array de string, que contiene la información de las imágenes procesadas.
- ✓ *long_reg_pro*: variable del tipo numérico contiene el número de entradas de la variable *registro_procesado*.
- ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

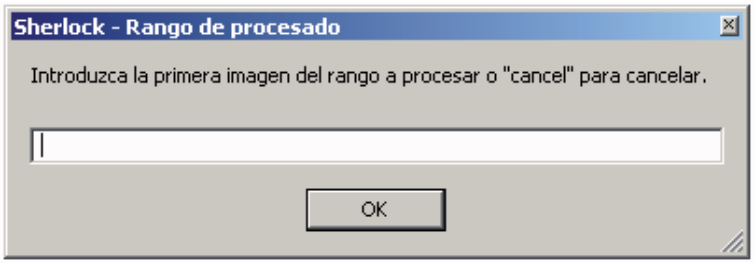
El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada esta subrutina,

- Nada más comenzar hay una etiqueta llamada *imagen inicial* que indica el principio de solicitud de la imagen inicial del rango de imágenes a mostrar.

- Se solicita la imagen inicial del rango a mostrar mediante la siguiente ventana:

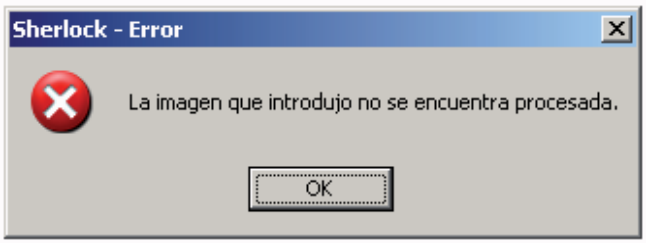


- Se comprueba si se escribió “cancel”, en cuyo caso finaliza la ejecución de la subrutina. Si se introdujo otra palabra se continúa con el siguiente punto.

- Se busca la posición de la imagen inicial del rango dentro del registro de procesado cargado en memoria. Para ello, se entra dentro de un bucle *while*, del cual se saldrá cuando se encuentre la imagen inicial del rango dentro de dicho

registro de procesado o se hayan recorrido todas las posiciones del mismo sin haber encontrado la imagen.

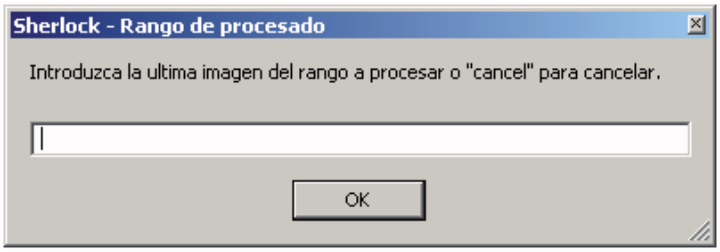
- Si no se encuentra la imagen solicitada, se mostrará el siguiente error y saltará a la etiqueta *imagen inicial*.



- Si se encuentra la imagen inicial del rango solicitado, se guarda en una variable llamada *indice_ini_rango* la posición que ocupa en el registro de procesado esta imagen inicial y se sale del bucle.

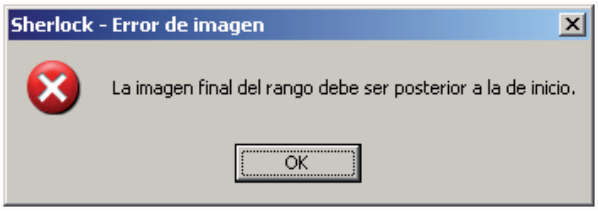
- Una vez se ha salido fuera del bucle, se encuentra una etiqueta llamada *imagen final*, y que indica el principio de solicitud de la imagen final del rango de imágenes a mostrar.

- Se solicita la imagen final del rango a mostrar mediante la siguiente ventana:



- Se comprueba si se escribió “cancel”, en cuyo caso finaliza la ejecución de la subrutina. Si se introdujo otra palabra se continúa con el siguiente punto.

- Se comprueba que el nombre de la imagen introducido como imagen final de rango es posterior al introducido anteriormente como imagen inicial del mismo. Si es posterior se continúa con el siguiente punto, pero si no lo es, se muestra el siguiente error por pantalla y se salta a la etiqueta *imagen final*.



- Se busca la posición de la imagen final del rango dentro del registro de procesado cargado en memoria. Para ello, se entra dentro de otro bucle *while*, del cual se saldrá cuando se encuentre la imagen final del rango dentro de dicho

registro de procesado o se hayan recorrido todas las posiciones del mismo sin haber encontrado la imagen.

- Si no se encuentra la imagen solicitada, se mostrará el siguiente error y saltará a la etiqueta *imagen final*.

- Si se encuentra la imagen final del rango solicitado, se guarda en una variable llamada *indice_fin_rango* la posición que ocupa en el registro de procesado esta imagen final y se sale del bucle.

- Una vez abandonado este bucle, se entra dentro de otro bucle *while*, el cual se abandonará una vez mostrados todos los resultados de rango de imágenes solicitado. Más concretamente, se saldrá de este bucle cuando el índice *indice_ini_rang* sea mayor al otro índice *indice_fin_rango*, lo que indicará que se mostraron todas las imágenes del rango, y llegado a esta situación, se termina la ejecución de la subrutina.

- Dentro del bucle se lee del registro de procesado la línea de información que indica el índice *indice_ini_rang*.

- Se llama a la subrutina *Visualiza imagen*, y una vez terminada ésta su ejecución, se incrementa en uno el índice *indice_ini_rang* y se vuelve al principio del bucle.

4.4. Subrutina Imagen.

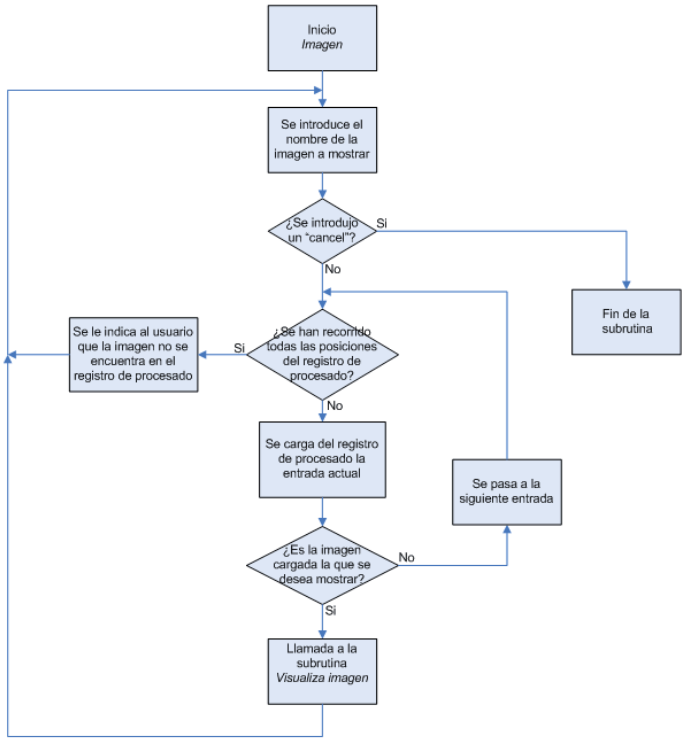
Esta subrutina se encarga de mostrar por pantalla resultados de imágenes sueltas que hayan sido ya procesadas. Esta subrutina nace de la necesidad de poder visualizar imágenes sueltas y no consecutivas de una forma sencilla para el usuario.

Las variables que recibe son las que se muestran a continuación:

- ✓ *registro_procesado*: variable del tipo array de string, que contiene la información de las imágenes procesadas.
- ✓ *long_reg_pro*: variable del tipo numérico contiene el número de entradas de la variable *registro_procesado*.
- ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada esta subrutina,

- Nada más comenzar hay una etiqueta llamada *comienza bucle*, que indica el principio del proceso de visionado de imágenes sueltas.

- Se solicita el nombre de la imagen que se desea mostrar mediante la siguiente ventana:

- Se comprueba si se escribió “cancel”, en cuyo caso se finaliza la ejecución de la subrutina. Si se introdujo otra palabra se continúa con el siguiente punto.

- Se busca la posición de la imagen solicitada dentro del registro de procesado cargado en memoria. Para ello, se entra dentro de un bucle *while*, del cual se saldrá cuando se encuentre la imagen inicial del rango dentro de dicho registro de procesado o se hayan recorrido todas las posiciones del mismo sin haber encontrado la imagen.

- Si no se encuentra la imagen solicitada, se mostrará el siguiente error y saltará a la etiqueta *comienza bucle*.

- Si se encuentra la imagen solicitada, se llama a la subrutina *Visualiza imagen*, y una vez terminada ésta su ejecución, se salta a la etiqueta *comienza bucle*, volviendo a solicitar de nuevo otro nombre de imagen a mostrar.

4.5. Subrutina Defectos.

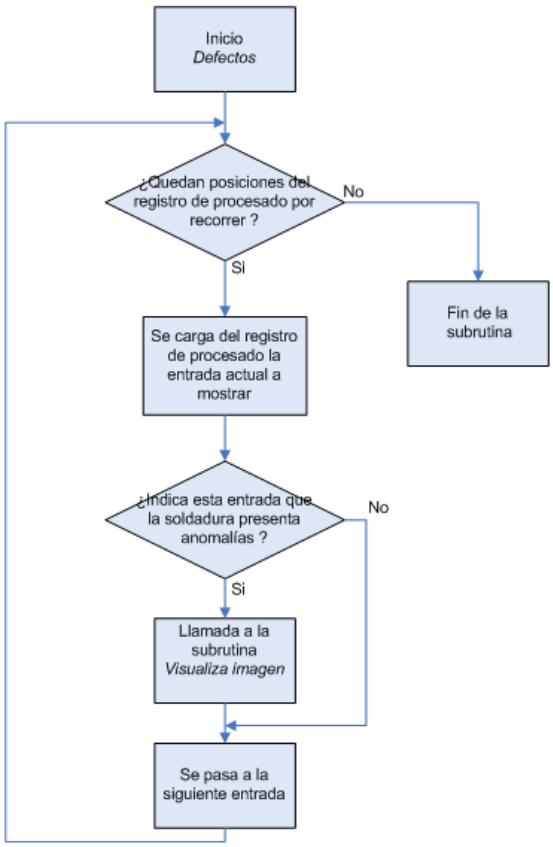
Esta subrutina se encarga de mostrar únicamente por pantalla las imágenes que presentan anomalías, es decir, todas aquellas imágenes que aparecen en el registro de procesado con el estado a *FAIL*. Esta subrutina se ha creado para poder ofrecer al operario una forma fácil, sencilla y rápida de consultar todas las imágenes que son sospechosas de presentar alguna anomalía.

Las variables que recibe son las que se muestran a continuación:

- ✓ *registro_procesado*: variable del tipo array de string, que contiene la información de las imágenes procesadas.
- ✓ *long_reg_pro*: variable del tipo numérico contiene el número de entradas de la variable *registro_procesado*.
- ✓ *long_dir_reg_pro*: variable del tipo numérico contiene el número de caracteres del directorio del generador.

Esta subrutina no devuelve ningún resultado, por lo que no hay variables de salida.

El diagrama de flujo que representa el comportamiento de esta subrutina es el que se muestra a continuación:



Una vez invocada esta subrutina,

- Se inicializa a cero una variable, que a modo de índice, mostrará en todo momento la posición que ocupa en el registro de procesado cargado en memoria la entrada sobre la que se está trabajando. Esta variable se llama *indice_def*.

- Se entra en un bucle *while*, el cual sólo será abandonado cuando se recorran todas las posiciones del registro de procesado.

- Se carga del registro de procesado la entrada que indica la variable *indice_def*.

- Se extraen de la línea anteriormente cargada, y que contiene la información de procesado, los caracteres cincuenta y dos, cincuenta y tres, cincuenta y cuatro y cincuenta y cinco, como se muestra a continuación, e indicando de este modo si la soldadura es válida o no.

```
AR000C001: Processed: 15/08/11 at 18:47:45. Status: FAIL  Directory: C:\Documents and Settings\PFC Carlos\Imagenes\AR000\AR000C001.bmp
```

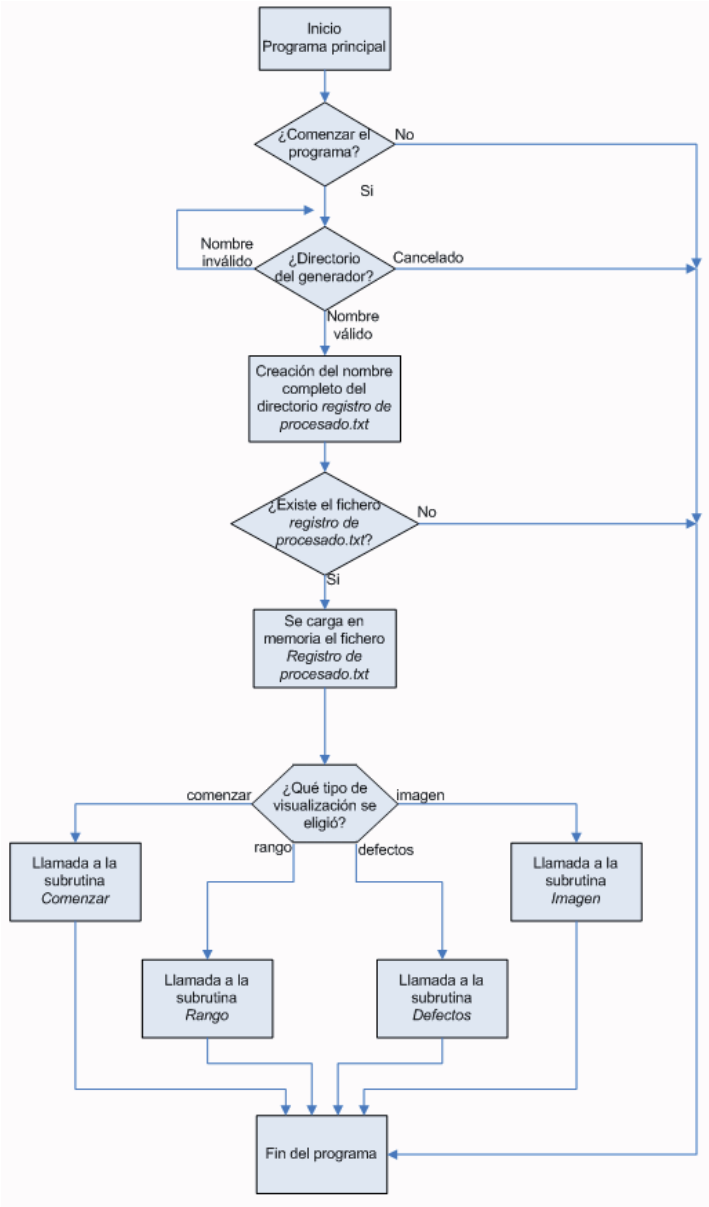
- Si la soldadura es correcta se incrementa en uno la variable *indice_def* y se vuelve al principio del bucle.

- Si la soldadura es inválida se llama a la subrutina *Visualiza imagen*, y una vez terminada esta su ejecución, ése incrementa en uno la variable *indice_def* y se vuelve al principio del bucle.

4.6. Programa principal.

Este programa principal es el que se que encarga de gestionar qué se desea hacer, y así coordinar a las subrutinas de segundo nivel.

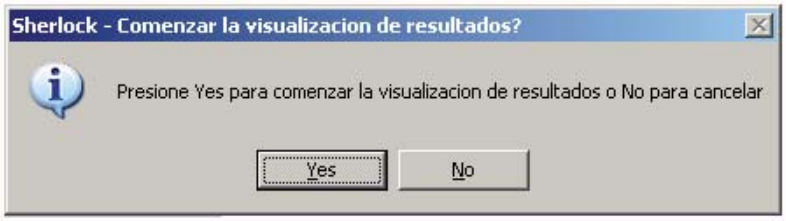
El diagrama de flujo que representa el comportamiento del programa principal es el que se muestra a continuación:



Una vez ejecutado el programa,

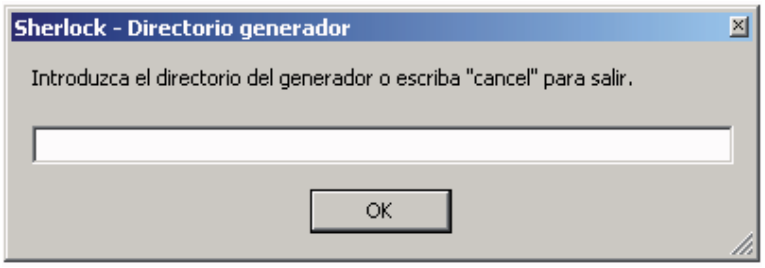
- Se inicializan a *FALSE* todas las salidas digitales, apagando así cualquier piloto o indicador que se pudiese quedar encendido de un procesado anterior.

- Se muestra en pantalla un mensaje preguntando si se desea empezar el programa de visión de resultados o si por el contrario se desea cancelarlo, como se muestra a continuación:

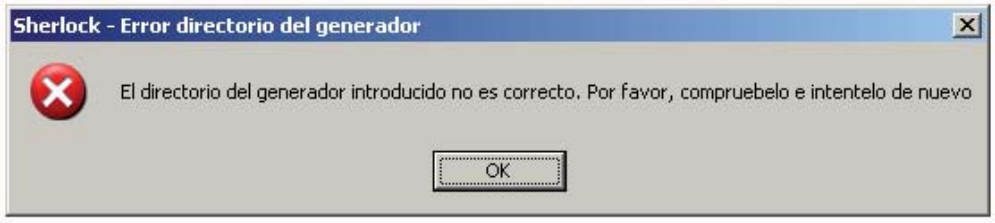


- Si se pulsa “Yes” se continuará ejecutando el resto de código, pero si por el contrario se pulsó “No” se para la ejecución del mismo.

- Se entra en un bucle *while*, del cual solo se saldrá si se introduce un directorio de generador correcto o se escribe la palabra “cancel”. Una vez dentro, el programa pregunta por la dirección del generador, ya que es donde se encuentran las imágenes guardadas y el registro de procesado, dando también opción a cancelar la ejecución escribiendo “cancel”:



- Se comprueba su existencia. De no existir el generador, se mostrará el siguiente mensaje por pantalla, y se volverá al principio del bucle *while*, volviendo a preguntar por el directorio del generador y permaneciendo así dentro de este bucle hasta que se cumpla una de las condiciones anteriormente explicadas.



- Se crea el directorio de procesado, necesario para posteriormente cargarlo y saber qué fotos han sido procesadas. Además se introduce en una variable la longitud (número de caracteres) que tiene este generador, ya que será necesario mas adelante para varias subrutinas.

- Se comprueba que existe el registro de procesado. De no existir éste se mostrará el siguiente mensaje por pantalla y se finalizará su ejecución.

- Si la comprobación fue correcta, se carga en memoria el fichero *Registro de procesado.txt*, que contiene toda la información de las imágenes procesadas.

- Se pregunta qué tipo de criterio se quiere seguir para visualizar los resultados obtenidos en el procesado, existiendo cuatro posibilidades:

  - ✓ Visualizar un procesado desde el principio.
  - ✓ Visualizar un rango de imágenes.
  - ✓ Visualizar imágenes sueltas y no consecutivas.
  - ✓ Visualizar solo imágenes que presentan anomalías

Si no se desea realizar ninguna opción también existe la posibilidad de cancelar, como se aprecia en el cuadro de diálogo:

- Si se introduce alguna otra opción que no está incluida, este cuadro de diálogo aparecerá continuamente hasta que se le introduzca de forma correcta una de las opciones existentes. Si por el contrario se desea abortar la ejecución basta con escribir el comando “cancel” en el diálogo.

- Finalmente, y dependiendo de la opción escogida, se carga una de las cuatro subrutinas existentes.

CAPÍTULO 7

Conclusiones y líneas futuras de trabajo

CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

Para poner el punto final al presente proyecto se extraerán algunas conclusiones y se aportarán algunas ideas para una posible línea de futuros trabajos.

Los objetivos iniciales planteados por Equipos Nucleares S.A., de los cuales se partió, se han visto superados durante el desarrollo, haciendo el presente proyecto de una envergadura considerable. Además de los especificado por Equipos Nucleares S.A., se han reportado errores y logs en registro a modo de histórico, se han creados subrutinas de control de errores, se han creado subrutinas para poder elegir el tipo de procesado y se ha diseñado un programa que permite visualizar los resultados de las imágenes analizadas según varios criterios de presentación.

Se han adquirido los conocimientos suficientes sobre la construcción de los generadores de vapor, las propiedades y principales anomalías de las soldaduras que se realizan, el comportamiento del robot, manejo de las cámaras y programación del dispositivo VA31 como para poder comprender y desarrollar un sistema completo de visión artificial.

Gracias a este proyecto propuesto en colaboración con Equipos Nucleares S.A., se ha desarrollado un conjunto de herramientas software capaces de capturar, procesar y representar gráficamente las soldaduras de los tubos de un generador de vapor, y que, por medio de una serie de programas, se permite trabajar de una manera sencilla con los diferentes programas desarrollados, sin tener conocimiento del software diseñado. De esta manera, el usuario de la aplicación obtiene una visión de todo lo que ocurre durante la ejecución de los programas desarrollados.

Con las herramientas desarrolladas en este proyecto se permite capturar y guardar imágenes en disco, procesarlas gracias a la visión artificial y poder visualizar los resultados de las mismas, todo ello con la finalidad de poder ofrecer una mejor calidad final de los generadores de vapor, reduciendo los costes y facilitando al usuario la detección y consulta de anomalías.

La posibilidad de implementar este proyecto es totalmente factible. Se dispone de todos los componentes necesarios, con los cuales, salvo el robot ABB, se ha trabajado, viendo que cumplen con los requisitos del proyecto.

En cuando a líneas futuras, hay varios aspectos en los que se podría trabajar en un futuro para ofrecer a este sistema de visión artificial desarrollado ciertas mejoras.

Aunque no se ha contemplado en este proyecto, uno de los grandes problemas que tiene este sistema la visión artificial es la iluminación a la hora de realizar la captura de imágenes. La placa que se fotografía es de acero inoxidable pulido, lo que la hace especialmente brillante. Esto se traduce en reflejos y sombras de la iluminación. Este es un aspecto muy crítico, ya que debido a una sombra o un brillo se puede marcar una soldadura correcta como defectuosa. Por ello, se podrían estudiar otras alternativas de iluminación, tanto en el tipo de fuente lumínica como en la forma de iluminar las soldaduras a capturar en el momento de ser capturadas.

El programa Sherlock permite cargar y desarrollar todas sus funciones mediante Visual Basic, además de las posibilidades que ofrece el Visual Basic en sí. Por ese motivo, se podría estudiar el desarrollo bajo este lenguaje de un entorno gráfico más amigable para conseguir hacer mucho más sencillo el manejo de los programas de visión artificial al operario que lo utiliza. Además, si se realiza un desarrollo bajo VB se podrían crear los directorios automáticamente cuando se realice la adquisición, sin tener que ser responsabilidad y labor del usuario generarlos y comprobar su existencia.

Finalmente, con respecto al procesado, estaría bien desarrollar un algoritmo algo más complejo, que ya no solo detecte que la soldadura presenta una anomalía, si no que la reconozca de entre una lista de fallos comunes y señale claramente en la imagen cargada la zona que presenta la anomalías y además reporte en el registro de procesado el defecto concreto que se encontró.

Referencias.

[1] Antonio Ferrer Soria, *Física nuclear y de partículas*, PUV, (2006)

[2] Samuel Glasstone,Alexander Sesonsk, *Nuclear Reactor Engineering: Reactor systems Engineering*, Chapman & Hall, (1994)

[3] Arantxa Rentería, *Robótica industrial: Fundamentos y aplicaciones*, Mc Graw Hill

[4] Ana María González, Mauricio Espina, *Robótica y aplicaciones*, monografía

[5] Jesús Cáceres Tello, *La visión artificial y las operaciones morfológicas en imágenes binarias*, Dpto. Ciencias de la Computación Escuela Técnica Superior de Informática, Universidad de Alcalá

[6] José Luis Esteban Sánchez-Marín, *Visión por computador*, (2003)

[7] Página del fabricante ABB: http://www.abb.es

[8] Página del distibuidor INFAIMON de productos DALSA: http://www.infaimon.com

[9] Página del distribuidor de objetivos: http://www.ccddirect.com

[10] Christopher E. Strangio, *The RS-232 Standard*, (2006)

[11] Dhananjay V. Gadre, *Programming the Parallel Port*, R&D Books, (1998)

[12] Manual Ubuntu http://manpages.ubuntu.com

[13] Fred Zlotnick, *The POSIX standard: a programmer's guide*, Benjamin/Cummings Pub. Co, (1991)

[14] Tutorial *Programación puerto serie en Linux*, http://www.iearobotics.com

[15] Tutorial *Puerto paralelo*, http://www.obsoletos.org

[16] Abello Linde, *Soldadura TIG*

[17] Samuel Rosario Francia, *Imperfecciones en soldaduras TIG*, Universidad Nacional Mayor de San Marcos.

[18] Manual VA31, *User's Reference Manual*, DALSA, (2010)

[19] Ayuda en Sherlock, *Sherlock help*