Escuela Politécnica Superior de Jaén

Grado en Ingeniería Electrónica Industrial

**UNIVERSIDAD DE JAÉN**

Escuela Politécnica Superior de Jaén

# SISTEMA BASADO EN VISIÓN POR COMPUTADOR PARA LA INSPECCIÓN DE PCB

**Alumno: Lucía Baeza Moreno**

Tutor: Prof. D. Diego Manuel Martínez Gila

Prof. D. Silvia María Satorres Martínez

Dpto: Electrónica y Automática

# RESUMEN

Una etapa fundamental en cualquier proceso de fabricación es el control de calidad. El prototipo desarrollado en en este proyecto tiene el objetivo de realizar el análisis del control de calidad de un tipo específico de PCB mediante dos métodos, uno basado en visión por computador, donde detectaremos la ausencia de material conductor de las pistas, y otro basado en continuidad, en el cual mediremos valores de resistencia eléctrica. En esta memoria trataremos el desarrollo del Software de la máquina.

# ABSTRACT

Quality control is a fundamental step in any manufacturing process. The goal of the developed prototype for this project is performing the quality control analysis of a specific type of PCB through two methods, one based on computer vision, where we will detect the absence of conductive material of the tracks, and another based on continuity, in which we will measure values of electrical resistance. In this report the development of the software of the machine is discussed.

A mis padres, a mis amigos y a mis profesores, por acompañarme, ser mi apoyo y guía durante esta etapa académica que termina.

# Sistema basado en visión por computador para la inspección de PCB


# Índice

# Lista de tablas



# Lista de figuras

# Lista de ilustraciones

## Definiciones, Acrónimos y Abreviaturas

| | |
|---|---|
| **PCB** | Printed Circuit Board |
| **IME** | In Mould Electronics |
| **AOI** | Automated Optical Inspection |
| **GRAV** | Grupo de Robótica, Automática y Visión por Computador |
| **ISR** | Integración Sensorial y Robótica |
| **LED** | Light-Emitting Diode |
| **OLED** | Organic Light-Emitting Diode |
| **OTFT** | Organic Thin-Film Transistor |
| **RFID** | Radio Frequency Identification |
| **IMD** | In-Mould Decoration |
| **FIM** | Film Insert Moulding |
| **PVC** | Policloruro de Vinilo |
| **spin-off** | Empresa nacida como extensión de otra |

# 1.-INTRODUCCIÓN.

En este capítulo haremos una introducción del proyecto que se ha desarrollado para este Trabajo de Fin de Grado. Para una mejor comprensión, dividiremos el capítulo en cuatro apartados. En el primer apartado, la motivación, hablaremos sobre el móvil del proyecto haciendo hincapié en la definición de la rama del mismo, la Plastrónica. En el apartado del contexto, mencionaremos los organismos implicados. Las soluciones tecnológicas que existen en el mercado actual para nuestro problema las veremos en el apartado del estado de la tecnología. Por último lugar, en los objetivos, enumeraremos las finalidades que se pretenden conseguir con el desarrollo de este proyecto

## 1.1.-Motivación.

El desarrollo de este Trabajo de Fin de Grado viene impulsado ante la necesidad de desarrollar una máquina que realice el control de calidad de un tipo específico de PCB (Printed Circuit Board), en concreto, una implementación de la tecnología IME (In Mould Electronics). Junto con otras dos tecnologías más, por un lado el empleo de polímeros y plásticos conductores y por otro parte, la electrónica flexible, conforman las tres ramas que tiene la Plastrónica. Para hacer un análisis completo y exhaustivo de la PCB debemos distinguir dos análisis:

- Visión: En esta etapa nos aseguraremos que las pistas no tienen ningún defecto visual grave, siendo este cuando existe ausencia de material conductor sobre la pista.

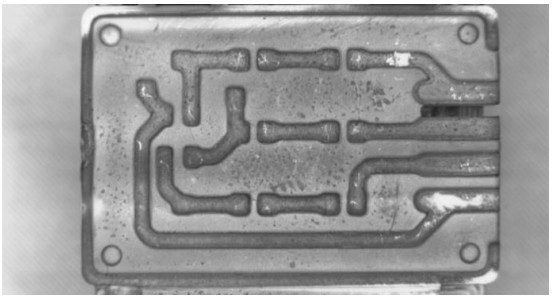


Figura 1. 1.PCB con defecto visual grave identificado por las regiones de color blanco sobre las pistas.

- Continuidad: En ciertas ocasiones no existen defectos visuales pero las pistas pueden tener problemas de continuidad. Para la comprobación eléctrica mediremos valores de resistencia eléctrica y si la medida que realizamos es mayor que el umbral fijado indica que no existe continuidad.

Por otro lado, para entender mejor el contenido de este TFG explicaremos a continuación en qué consiste el concepto de Plastrónica.

**¿Qué es la Plastrónica?**

Esta tecnología es una nueva línea de investigación y desarrollo de la electrónica que estudia la integración de componentes electrónicos en materiales plásticos, el desarrollo de electrónica flexible y el uso de materiales conductores basados en polímeros a partir de materiales orgánicos e híbridos.

Una de las grandes ventajas de la plastrónica es que permite obtener dispositivos flexibles y ligeros que puedan integrarse en determinados productos con geometrías complejas. Otra gran ventaja es que se trata de una tecnología económica debido a su capacidad de procesado a gran escala, a diferencia de la electrónica de silicio.

Esta línea de investigación tiene un gran potencial de crecimiento debido a sus numerosas aplicaciones en diferentes sectores industriales como en automoción, aeronáutica, aeroespacial y electrónica de consumo ente otros.

La plastrónica se divide según los materiales y tecnologías de procesado y distinguimos tres campos:

- **Polímeros y plásticos conductores**.

Los polímeros conductores son aquellos materiales que son capaces de conducir la corriente eléctrica. Encontramos dos tipos:

- Polímeros intrínsecamente conductores: Son aquellos polímeros que poseen conductividad eléctrica, la cual se aumenta debido al uso de diferentes agentes dopantes como por ejemplo: iones pequeños (Cl,Br,..); moléculas($Cl_2$, $I_2$,…) y moléculas más grandes como polímeros, ácido hialurónico.
- Polímeros extrínsecamente conductores**:** Se trata de polímeros convencionales aislantes donde se introducen cargas conductoras como metales, grafito, etc que proporcionan conductividad. Estos polímeros combinan la conductividad eléctrica de los metales y de cargas carbonosas con las propiedades de los polímeros haciéndolos ligeros, fácilmente procesables y resistentes al ataque químico y a la corrosión. Estas propiedades, han hecho que encontremos estos materiales en múltiples aplicaciones.

- **Electrónica flexible.**

La electrónica flexible hace referencia a los dispositivos electrónicos que pueden doblarse, estirarse independientemente de su composición material sin perder la funcionalidad. Esta tecnología combina el uso de nuevos materiales (tintas conductoras, semiconductoras y dieléctricas), la relación coste-eficiencia y procesos de producción de gran superficie y a gran escala que abren nuevos campos de aplicación.

Las ventajas y propiedades que proporciona son: delgadez, ligereza y flexibilidad. Aparte permite desarrollar una amplia gama de componentes electrónicos y aplicaciones (células solares de polímero, LEDs (Light-Emiting Diode), orgánicos de emisión de luz OLED (Organic Light-Emiting Diode), transistores orgánicos OTFT (Organic Thin-Film Transistor), antenas RFID (Radio Frequency Identification), pantallas táctiles flexibles, baterías impresas,..).


Figura 1.2. Ejemplo de electrónica flexible.

- **In-Mould Electronics**.

Nuestro proyecto está basado en esta rama. In-Mould Electronics hace alusión a Circuitos electrónicos impresos, en película, que se han sometido a procesos de termoformado y moldeo por inyección. El circuito sigue siendo funcional ya que las pistas conductoras contornean la forma 3D.

Esta tecnología se puede considerar una extensión a IMD (In-Mould Decoration) / FIM (Film Insert Moulding) tecnología de base de la década de 1990. Combina esencialmente films con componentes electrónicos impresos, gráficos y electrónica para formar un dispositivo electrónico totalmente funciona con componentes 3D. Las principales características y ventajas:

- Componentes y piezas más ligeras y de volumen reducido.
- Menores costos de fabricación debido a su simplificación y mejora de eficiencia.
- Reducción de los tiempos de ensamblaje entorno a un 40%. El ensamblaje es un proceso de conexión única, de “conexión instantánea”, que reduce significativamente el tiempo de montaje a la vez que aumenta la confiabilidad.
- Adaptación a geometrías complejas y piezas 3D.


Figura 1.3. Ejemplo de IME.

## 1.2.-Contexto.

Este trabajo fin de grado se ha realizado en colaboración con el Grupo de Investigación de Robótica, Automática y Visión por Computador (GRAV) de la Universidad de Jaén. Son muchos los proyectos los que el GRAV realiza relacionados con el empleo de visión por computador aplicada a la inspección de la calidad y a la automatización industrial. Estas líneas de investigación no son las únicas que se desarrollan en el grupo. Otros ejemplos serían:

- Control de la interacción física de brazos robóticos.
- Fusión sensorial.
- Aplicación del control automático a la elaiotecnica y olivicultura.

Además de las actividades de investigación propias del GRAV se realizan actividades de transferencia en colaboración con empresas privadas, y es una de estas colaboraciones la que ha servido como oportunidad y experiencia para el desarrollo de este trabajo fin de grado.

## 1.3.-Estado de la tecnología.

No existe una placa de circuito impreso universal que nos sirva para múltiples usos, si no que la mayoría de las veces debemos diseñar un modelo específico para cada aplicación. Este hecho hace que exista una enorme cantidad de modelos de PCB y en consecuencia, de procesos de fabricación. A pesar que cada proceso de fabricación debe adecuarse a las características de la PCB que se desea obtener, todos comparten un patrón similar, ya que en la totalidad de procesos se distinguen dos bloques principales, fabricación y control de calidad.

De los dos bloques mencionados, nos centraremos en aquel que nos concierne, el control de calidad. En concreto analizaremos las soluciones tecnológicas que existen para el análisis eléctrico y la inspección visual.

### 1.3.1.-Visión.

Esta etapa puede realizarse de forma automatizada (Automated Optical Inspection, AOI), o por otro lado puede hacerse de forma manual. Si comparamos un sistema de inspección visual automática frente a un sistema de inspección manual observamos las siguientes ventajas:

- Son capaces de adaptarse, rápida y fácilmente, a diferentes productos y superficie.
- Se pueden programar y monitorizar de forma remota.
- Son capaces de trabajar las 24 horas del día.
- Tienen una mayor velocidad de inspección que los inspectores humanos.
- Reducción del error humano. Moganti, M., & Ercal, F. (1995). Automatic PCB inspection systems:”To ensure quality, human operators simply inspect the work visually against prescribed standards.The decisions made by this labor intensive, and therefore costly, procedure often also involve subjetive judgements. Automatic inspection systems however, remove the subjetive aspects and porvide fast, quantitative dimensional

assessments.”[Para garantizar la calidad, los operarios simplemente inspeccionan el trabajo visualmente en función de los estándares prescritos. Las decisiones tomadas por este procedimiento de mano de obra intensiva y, por lo tanto, costosa, también suelen implicar juicios subjetivos. Sin embargo, los sistemas de inspección automáticos eliminan los aspectos subjetivos y permiten realizar evaluaciones dimensionales cuantitativas rápidas]. IEEE Potentials, 14(3), 6–10.doi:10.1109/45.464686

- Mayor precisión: cuando un sistema automatizado de visión artificial está programado para realizar una tarea una y otra vez, la precisión y la repetitividad en comparación con un empleado es mucho mayor.
- Los sistemas de inspección automatizados aprovechan las cámaras de visión artificial para la recopilación de datos y los programas informáticos para el análisis de datos. La cámara captura imágenes y las envía al programa el cual se encarga de la toma decisiones sobre si la imagen pasó o no pasó la inspección. Por lo tanto la decisión es completamente objetiva y está sujeta al mismo umbral en todos los casos.

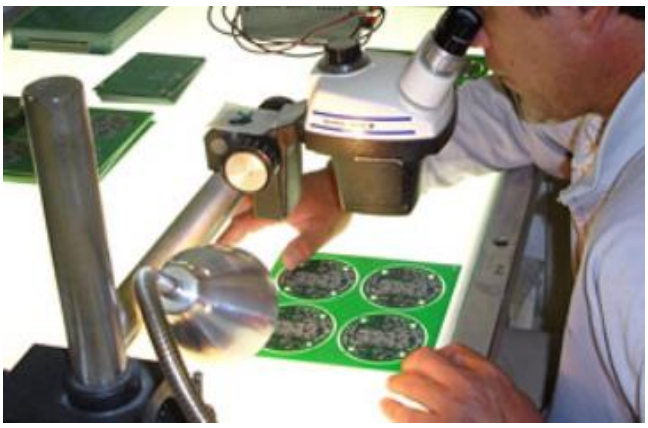


**Figura 1.4. Inspección visual llevada a cabo por un operario.**

A pesar de que todavía existen procesos donde la inspección la realiza un operario, la mayoría de soluciones que existen en el mercado son aplicaciones de AOI por todas las ventajas mencionadas anteriormente. Un ejemplo de máquina de inspección visual que podemos encontrar en la industria sería el siguiente.


Figura 1.5. Ejemplo de AOI.

Se trata de una de las muchas soluciones que existen en el mercado. Esta máquina realiza .la inspección del circuito frente a imágenes digitales para verificar que coincide con el diseño y que está libre de defectos. Se consigue a través de que algoritmos inspectores entrenados verificarán por si se detecta alguna anomalía.

Cabe mencionar que una parte fundamental de las máquinas de AOI de PCB es la parte software. Los algoritmos de inspección los podemos clasificar en tres categorías: referencial; no referencial y métodos híbridos. Dentro de estas clases encontramos distintos métodos. Podemos ver la clasificación completa en el siguiente diagrama de árbol.

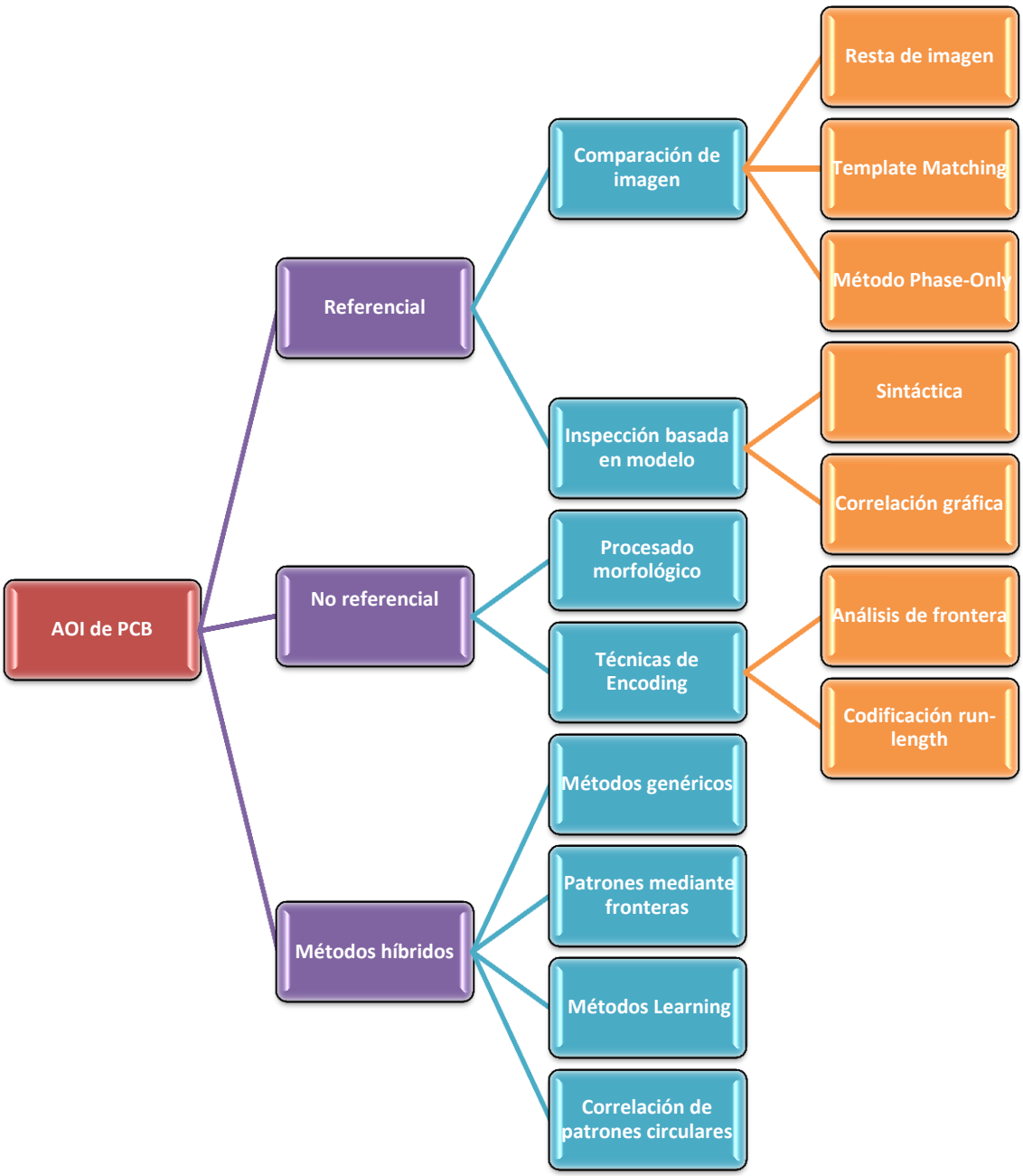



Ilustración 1 Clasificación algoritmos de inspección de PCB.

A continuación analizaremos cada una de estas técnicas en detalle.

- **Resta de la imagen**: La resta de la imagen es la técnica más simple y de las más efectivas. Fue de las primeras técnicas de inspección. La imagen que obtenemos de la resta, es la imagen de los defectos la cual puede ser posteriormente analizada.

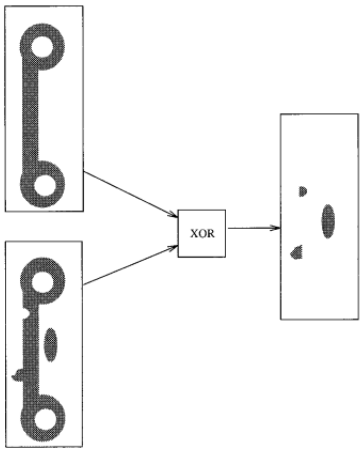
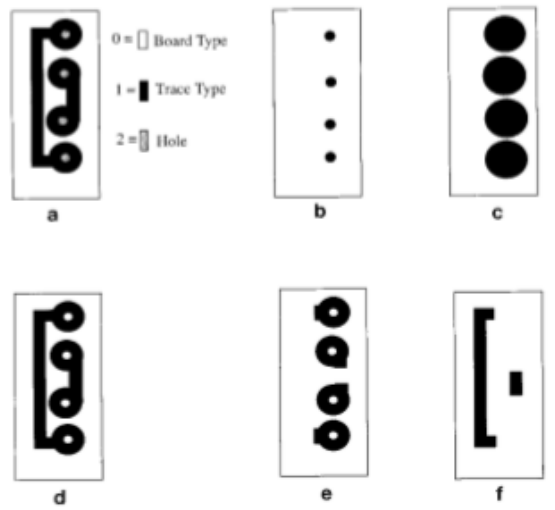



Figura 1.6. Ejemplo de substracción de imagen.

- '**Template matching'**: Es una forma mejorada de la substracción de imágenes en la cual obtenemos en primer lugar características del objeto y posteriormente se comparan con las propiedades definidas del modelo. Una de las mayores limitaciones de esta técnica es que la mayoría de las veces hay que hacer una gran cantidad de correlaciones, hecho que hace que sea de difícil procesado computacional.
- '**Phase-only'**: Por otro lado este método está basada en la correlación de fase única. Una imagen phase-only es una imagen la cual tiene una unidad de amplitud espectral, haciendo que toda la información sea almacenada en la fase. Esta técnica emplea la transformada de Fourier.
- **Sintáctico:** También llamado técnica de emparejamiento de cadenas. En el enfoque sintáctico, una imagen de PCB se modela como un conjunto finito de letras / símbolos. El método implica trazar el límite para producir una lista ordenada de puntos de límite y analizar la forma para producir una descripción sintáctica de esta, utilizando las formas primitivas que mejor describen el patrón de PCB.
- **Correlación gráfica**: Está basado en las propiedades estructurales, topológicas y geométricas de la imagen. Se lleva a cabo una comparación estructural con una imagen obtenida de una PCB de referencia.
- **Procesado morfológico**: Es una de las técnicas más empleadas en inspección de PCB. Consiste en aplicar operaciones de dilatación y erosión, lo que no implica el uso de ningún modelo como referencia.




Separating conductor surrounding holes from other conductor traces. (a) Original PCB pattern, (b) after (1 → 0) and (2 → 0) operation, (c) after dialation operation, (d) after (2 → 0) operation on (1), (e) after AND operation on (c, d), and (f) after XOR operation on (d, e).

Figura 1.7. Ejemplo de procesado morfológico.

- **Análisis de fronteras**: Estudian la representación de las fronteras seguida por comprobación. Usando el método de codificación Freeman las fronteras se traducen a una aproximación poligonal donde podemos detectar varios tipos de defecto en las PCB como muescas y golpes.
- **Codificación 'run-lenght'**: Se basa en el análisis del histograma horizontal y vertical. El método cuenta las pistas continuas a lo largo de las filas y de las columnas. Podemos verificar que la anchura de la pista es la correcta si el valor en el histograma de los píxeles es menor que el umbral establecido.
- **Métodos genéricos**: Consisten en una combinación de algoritmos referenciales y no referenciales. No compara la imagen que queremos analizar con una imagen de referencia píxel por píxel sino que elimina la información innecesaria comprobando solo los puntos más críticos.
- **Patrones mediante fronteras:** Utilizan detección de patrones y técnicas de análisis de fronteras. Los algoritmos de fronteras localizan las áreas que son más probables a tener algún defecto las cuales son marcadas y posteriormente mediante detección de patrones se mide la anchura de las pistas.

- **Métodos Learning**: El “Deep Learning” es un conjunto de algoritmos de aprendizaje automático (Machine Learning) y el cual usa estructuras lógicas que se asemejan a la organización de sistema nervioso de los humanos, donde existen varias capas de unidades (neuronas artificiales) que se especializan en detectar determinadas características existentes en los objetos que se analizan. El aprendizaje profundo es parte de un conjunto más amplio de métodos de aprendizaje automático basados en asimilar representaciones de datos. La visión artificial es una de las áreas donde el Deep Learning proporciona mejores resultados en comparación con algoritmos más tradicionales. El esquema que representa los pasos que lleva a cabo un algoritmo de visión aplicando Deep Learning podría ser el siguiente:

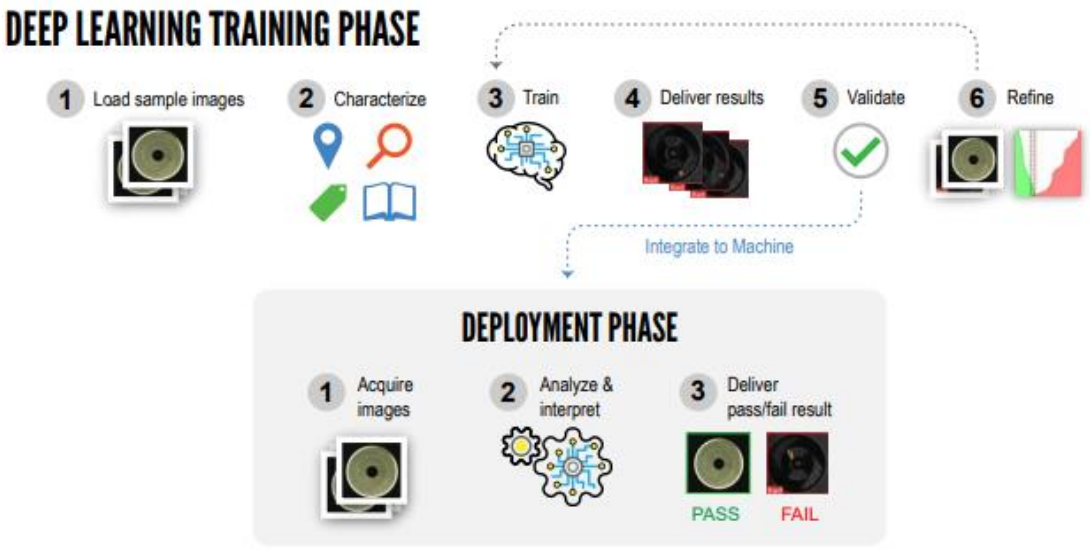



Figura 1.8. Deep Learning aplicado a visión.

Un ejemplo de ‘Deep Learning ’ aplicado a PCB sería el algoritmo de correlación de radios .El sistema utiliza un método de autoaprendizaje de código radial en el cual la red neuronal se entrena con patrones de PCB buenas, y el sistema convierte estos patrones en características conocidas como códigos radiales.

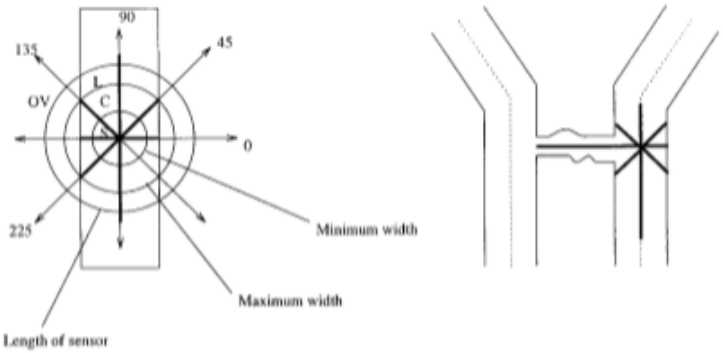



Figura 1.9. Correlación de radios.

- **Correlación de patrones circulares:** Se basa en el principio de que una ventana de tamaño 32x32 recorre la PCB de izquierda a derecha y de arriba abajo mientras que se genera una codificación. Posteriormente e analiza este código mediante una comparación con una plantilla para ver si hemos detectado algún defecto.

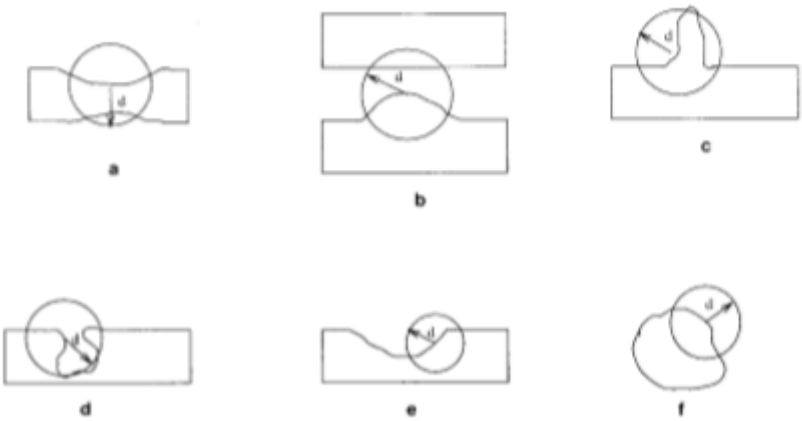



Figura 1.10. Correlación de patrones circulares.

Tras analizar las opciones para la inspección visual de PCBs que podemos encontrar en la industria, llegamos a la conclusión de que existe un amplio abanico de soluciones para circuitos impresos estándares. Pero la solución para el tipo de PCB que estamos analizando nosotros, IME, no está implementada ya que se trata de una línea de investigación tan novedosa que todavía no se ha implementado ninguna máquina de AOI para su inspección. Esto nos indica el motivo de la necesidad del prototipo que hemos construido.

### 1.3.2.-Continuidad.

Si analizamos la comprobación de continuidad nos encontramos con dos tipos de soluciones al igual que ocurre en la etapa de visión, siendo una de ellas comprobación automatizada y la otra de forma manual. Las ventajas que presenta un análisis de continuidad automatizado frente a uno manual son muy parecidas a las ventajas que tenían los sistemas automatizados de inspección visual. Si analizamos las ventajas trasladándolo a nuestra inspección estas serían:

- Reducción de errores. El tamaño reducido de las pistas que tenemos que medir haría que la probabilidad de cometer un fallo en la medida manual fuera muy alta. Sin embargo con un sistema calibrado y diseñado específicamente para nuestra PCB el error se reduciría.
- Mayor rapidez. El número de medidas que debemos realizar para completar el análisis en este caso son: 10 medidas intrapista (misma pista) y 45 medidas interpista (distinta pista). El tiempo que supondría a una persona hacer todas las medidas y analizarlos sería bastante superior al tiempo que le supondría a la máquina.
- Puede trabajar durante largos períodos de tiempo. Una misma persona no podría estar 24 horas al día haciendo estas medidas y aunque el tiempo fuera menor sería una tarea muy repetitiva y tediosa para el operario.

Por todas estas ventajas, la mayoría de procesos para la comprobación eléctrica que encontramos se realizan de forma automatizada. Existen dos métodos que implementan la inspección automática que podemos encontrar en la industria, los cuales son los siguientes

### Test sonda volante

En el ejemplo siguiente se comprueban: la integridad de las pistas y la inexistencia de circuitos abiertos o cortocircuitos en la placa terminada. Hay dos métodos de comprobación, sonda volante para pequeños volúmenes y para grandes volúmenes. Comprobamos eléctricamente cada PCB multicapa con los datos originales de la placa. Utilizando la sonda volante se comprueba cada red para asegurase de que está completa (sin circuitos abiertos) ni en corto con otras redes.

Figura 1.11. Implementación real del test de sonda volante.

En este procedimiento la tarjeta es colocada en un marco en posición vertical y las puntas de prueba se mueven en el sentido XY sobre ambas caras del circuito para ubicarse exactamente sobre los pads cuyas características eléctricas se van a probar. Luego las puntas de prueba se desplazan en el eje Z hasta hacer contacto con los pistas como se aprecia en la siguiente figura, permitiendo que el software efectúe la medidas requeridas y presente los resultados que determinan si la tarjeta es eléctricamente correcta (PASS) o rechazada (FAIL).

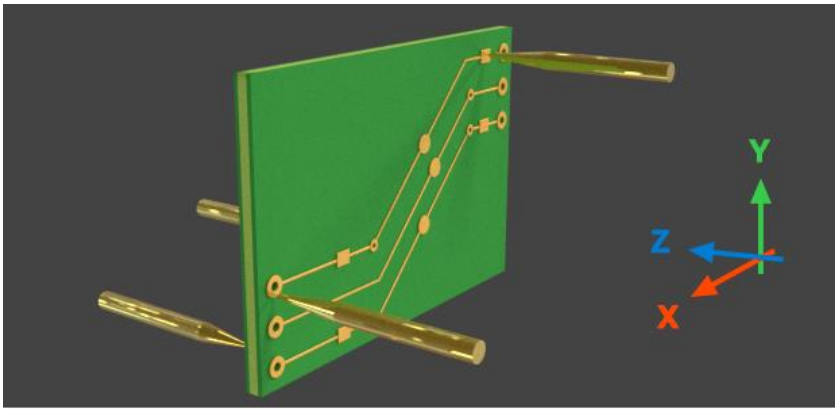



Figura 1.12. Explicación test sonda volante en el plano tridimensional.

### Test cama de agujas

Por otro lado, existe otro tipo de prueba usualmente llamado “Test en cama de agujas”. Este se implementa básicamente colocando la tarjeta sobre un marco que tiene un número de agujas retráctiles que hacen contacto con igual cantidad de pads de nuestra tarjeta y que van conectadas a una interfase que a través de un programa efectúa las diferentes medidas de continuidad, capacitancia y aislamiento requeridas. Este proceso está orientado a grandes producciones donde el tiempo de puesta en el mercado de un producto, exige la producción rápida de los circuitos impresos para su ensamble. En la siguiente figura se observan dos tarjetas montadas sobre un marco de aluminio, antes de bajar sobre la cama de agujas de prueba en la parte inferior de la máquina para efectuar el proceso descrito.

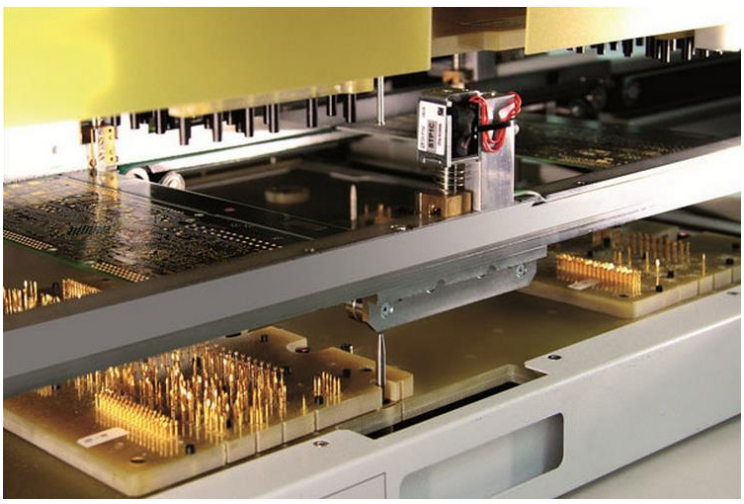


Figura 1.13. Implementación test cama de agujas.

De igual forma que ocurría con la etapa de inspección visual, no hay soluciones en el mercado para realizar el test eléctrico a nuestro modelo. De ahí otro motivo para implementar este prototipo.

Como conclusión destacar que ningún sistema de inspección automatizado es totalmente infalible pero todas las ventajas mencionadas anteriormente suponen una justificación sólida de por qué un sistema de inspección automatizado en los dos análisis son las mejores opciones en la solución de nuestro problema.

## 1.4.-Objetivos.

### 1.4.1.-Objetivo general.

El objetivo general de este Trabajo de Fin de Grado es el diseño del sistema de visión de la máquina. Los procesos que debe llevar a cabo esta máquina son los siguientes:

- **Proceso de Visión**: Durante esta etapa se pretende discriminar las PCBs que tengan defectos visuales graves en sus pistas. Este análisis tiene el objetivo de identificar que PCBs van a ser posibles candidatas a ser consideradas como defectuosas al final de la inspección.

- **Proceso de Continuidad:** Esta es la etapa crítica donde identificaremos si la PCB es correcta y tiene continuidad en cada una de sus pistas o si por el contrario es defectuosa y tiene falta de continuidad en alguna de sus pistas. También analizaremos si las pistas están aisladas eléctricamente y no existe continuidad entre ellas.

### 1.4.2.-Objetivos específicos.

- Estudio de la adquisición de imágenes, donde se seleccionará tanto la cámara a usar como las condiciones óptimas para adquisición.
- Procesado de la imagen para identificar las PCB con defectos visuales.
- Desarrollo del software de control en Arduino, el cual se encargará de activar y desactivar los relés que accionará el sistema de iluminación y los actuadores.
- Desarrollo del software de Java el cual será el programa principal que gobernará la máquina y el cual deberá comunicarse con Arduino.
- Implementar una interfaz gráfica que permita al operario interactuar en el proceso y que muestren los resultados del procesado de visión y del análisis de continuidad.
- Realización de pruebas para la validación de la máquina.

## 2.-METODOLOGÍA.

Una vez vista la introducción de esta memoria, pasamos a explicar el contenido principal de esta, la Metodología. Se explicará cada una de las partes que han sido elaboradas, detallando tanto el procedimiento llevado a cabo como aspectos técnicos importantes.

Las partes en las que se va estructurar este capítulo son las siguientes:

- Diseño mecánico
- Desarrollo software

### 2.1.-Diseño mecánico.

Antes de explicar la parte desarrollada en este TFG, es necesario hacer un breve inciso para explicar el diseño mecánico del que se partió, y de esta forma tener una visión más global del proyecto. Este diseño consta de las siguientes partes:

- Estructura metálica elaborada con perfiles de aluminio.
- Plataforma metálica para soporte de utillajes.
- Dos platos giratorios para ajustar el ángulo de rotación de los útiles respecto a la cámara y al sistema de testeo mediante puntas de cobre. Incluye pieza de bloqueo de ambos. En la Figura 2.4, en color verde, podemos ver plato correspondiente del sistema de testeo.
- Mesa X Y para centrar la placa a inspeccionar con respecto al sistema de testeo, la cual se puede observar en la Figura 2.4.
- Útiles para el soporte de los moldes de las placas de inspección.
- Sistema de testeo, el cual podemos ver en la Figura 2.1. Este está formado por un actuador neumático y una cama de puntas.
- Sistema de visión completo, formado por cámara y placa de iluminación como podemos ver en la Figura 2.2.
- Cuadro eléctrico, cuyo diseño mecánico podemos ver en la Figura 2.3.
- Cerramientos de PVC extruido.

A continuación se muestran cada uno de los componentes por separado:

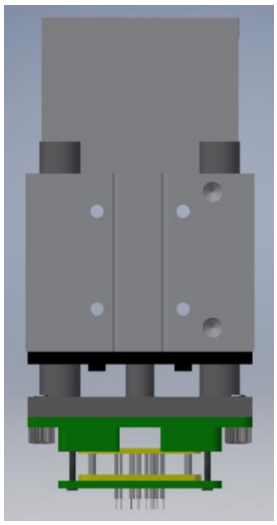


Figura 2.1. Vista frontal del sistema de testeo.

Figura 2.2: Vista frontal del sistema de visión.

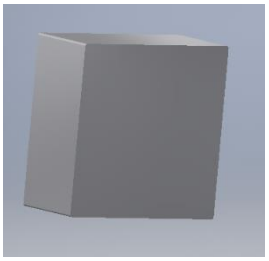


Figura 2.3. Vista del cuadro eléctrico

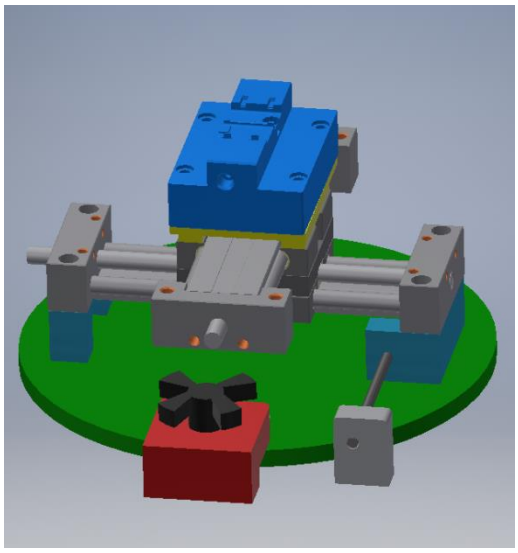


Figura 2.4. Vista del centrador del sistema de testeo

.

.El diseño de la máquina completa se ve en la siguiente Figura:

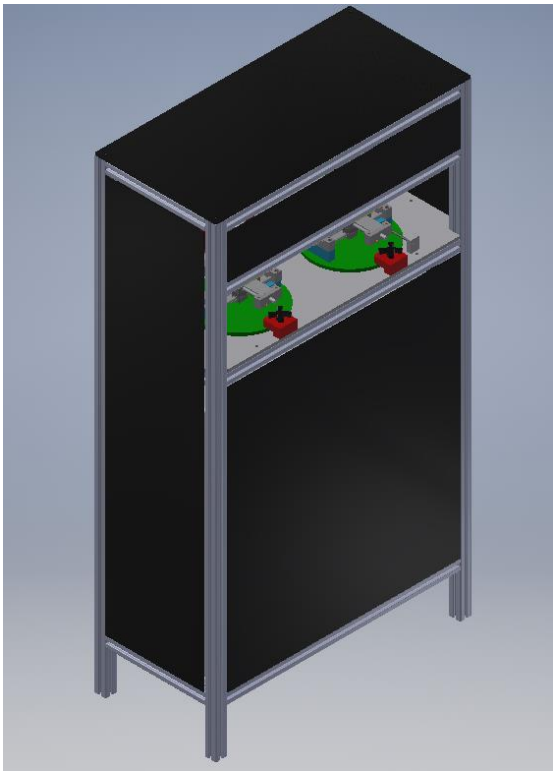


Figura 2.5 . Perspectiva isométrica de la máquina.

## 2.2.-Desarrollo del software.

El desarrollo de nuestro programa consta de dos etapas: visión y continuidad. Teniendo en cuenta esta distinción podemos estructurar nuestro código en función del siguiente flujograma.

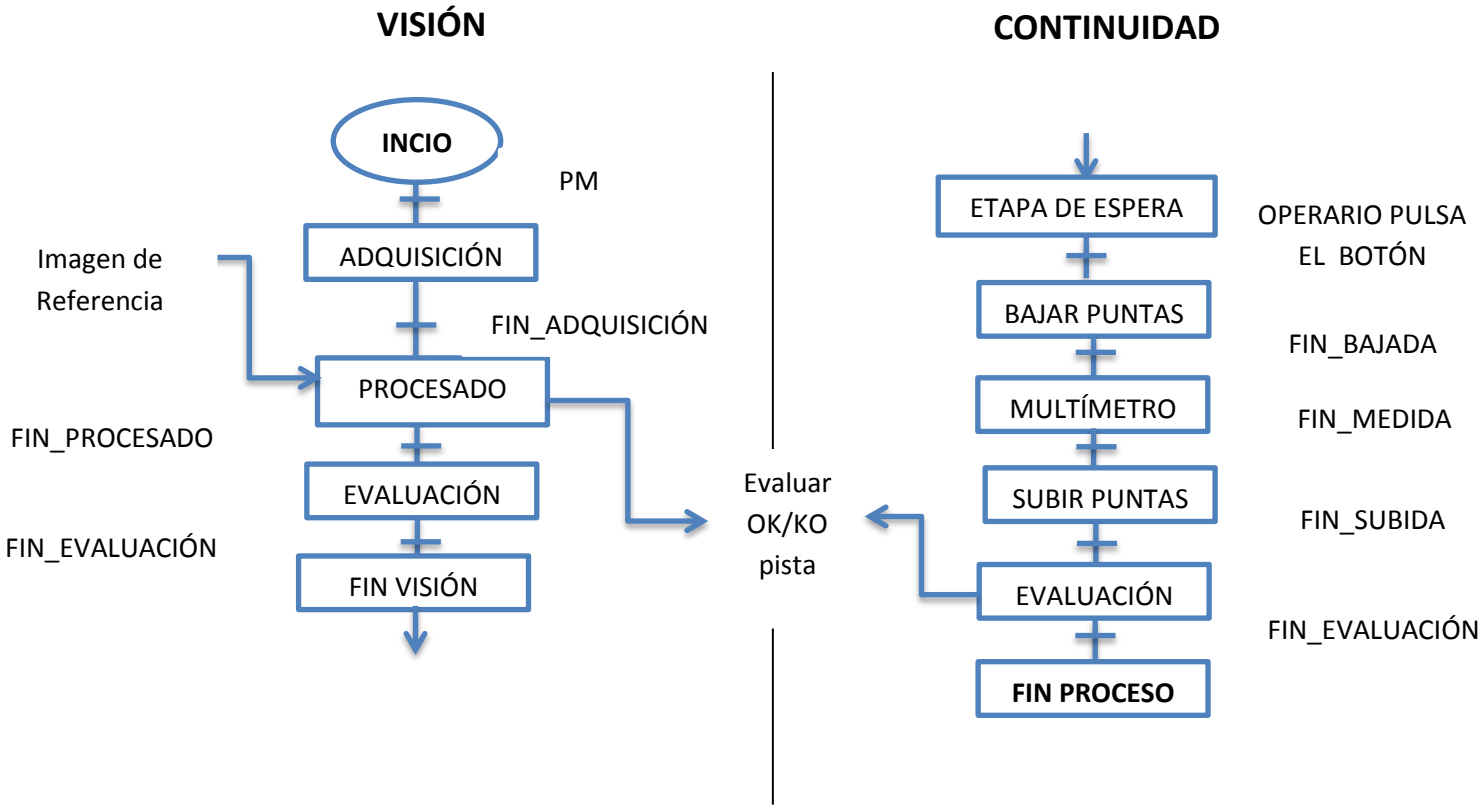



Ilustración 2. Flujograma de la parte software.

En el esquema anterior podemos ver que en primer lugar se lleva a cabo la fase de análisis de visión, la cual comienza cuando el operario le da al botón de inicio. Seguidamente se procede a hacer la adquisición de la imagen que posteriormente compararemos con una imagen de referencia con el fin de evaluar los defectos visuales. Cuando la etapa de visión termine, el operario deberá trasladar la PCB al set-up donde se realizará el análisis de continuidad. Una vez que la PCB está correctamente colocada, el operario dará el inicio del proceso de continuidad, expandiendo el cilindro y haciendo que las puntas bajen y toquen las pistas correspondientes. El multímetro realizará las medidas que serán almacenadas y evaluadas posteriormente. En el último lugar, el cilindro volverá a la posición de reposo concluyendo con el proceso. A

continuación pasaremos a estudiar las distintas etapas del análisis por separado y de forma más detallada.

### 2.2.1.-Desarrollo del programa de visión.

La finalidad de la parte de visión es la elaboración de unos algoritmos que detecten defectos visuales en las pistas de las PCBs que estamos analizando, ya que se tratan de la parte fundamental de estas y que son críticas en el correcto funcionamiento de estas.

Si analizamos la adquisición de las imágenes, podemos observar que las imágenes que se adquieren tienen las mismas condiciones, ya que en primer lugar se ha llevado a cabo una calibración del set-up de la cámara para adquirir correctamente nuestra PCB y una posterior fijación del mismo, de esta forma conseguimos unificar la adquisición. El procedimiento que hemos llevado a cabo para hacer la adquisición lo explicaremos más adelante.

Una vez fijadas las condiciones de adquisición, nos decantamos por una solución, y esta es la substracción de imagen .Esta técnica que hemos aplicado en el estado de la tecnología, parte de una imagen de una PCB 'buena', a partir de la cual obtendremos la máscara. Mediante una operación XOR entre la imagen que queremos analizar y la máscara, nos quedamos con la parte de la imagen que buscamos, los defectos de las pistas.

Definido el método a seguir para resolver el problema, nos dispusimos a elaborar el programa propiamente dicho. En primera instancia se resolvió en Matlab, por comodidad, ya que se trata de una plataforma con gran potencial y de uso intuitivo. Por último lugar, se trasladó la solución a Java, haciendo uso de la librería OpenCV.

#### 2.2.1.1.-Implementación en Matlab.

Esta plataforma es un sistema de cálculo numérico que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Entre sus prestaciones básicas se encuentran: la manipulación de matrices; la

representación de datos y funciones; la implementación de algoritmos; la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete Matlab dispone de dos herramientas adicionales que expanden sus prestaciones: Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario. Una gran ventaja que tiene esta plataforma es que sus capacidades se ven ampliadas con las Toolboxes, las cuales ofrecen una amplia gama de funcionalidades. Muchas de estas toolboxes tienen aplicación directa en el campo de la visión artificial.

Conocidas a grandes rasgos las funcionalidades que tiene la plataforma que vamos a usar podemos empezar con el desarrollo del programa. Como hemos mencionado anteriormente la estrategia que seguiremos se basa en la resta de imágenes, de modo que se dividirá el problema en dos partes:

- Obtención de máscara.
- Aplicación de la correlación.

## Obtención de la máscara

Para aplicar la técnica de XOR, necesitamos la imagen que vamos a usar como plantilla. Mediante la comparación con esta imagen, hallaremos las áreas en las que difieren las dos imágenes que estamos analizando. Para elaborar la máscara, escogeremos una PCB que no tenga ningún defecto visual.

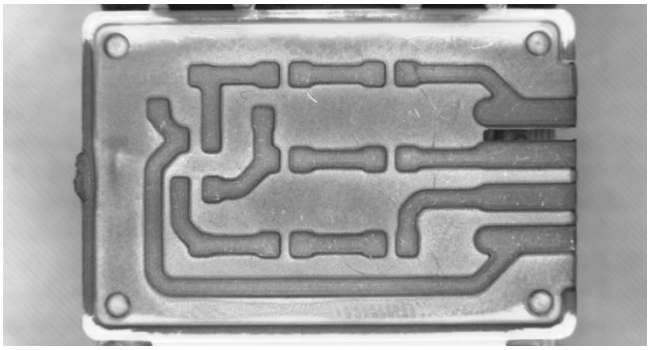


Figura 2.6. PCB que usaremos para elaboración de la máscara.

Cuando se inició este proyecto, los conocimientos que se tenían sobre visión artificial eran insuficientes para elaborar una máscara desde cero por lo que se optó por usar una de las Toolboxes que incorporaba Matlab, en concreto: Image Segmenter, la cual como su nombre indica se usa para segmentar.

Esta toolbox aplica segmentación basada en regiones por crecimiento simple. Dicha técnica consiste en que a partir de unos píxeles semilla buscamos píxeles similares los cuales se añaden a la semilla creando una región con atributos parecidos y de esta forma haciendo la segmentación. Las semillas son seleccionadas de forma manual, pudiendo seleccionar uno o varios píxeles por región.

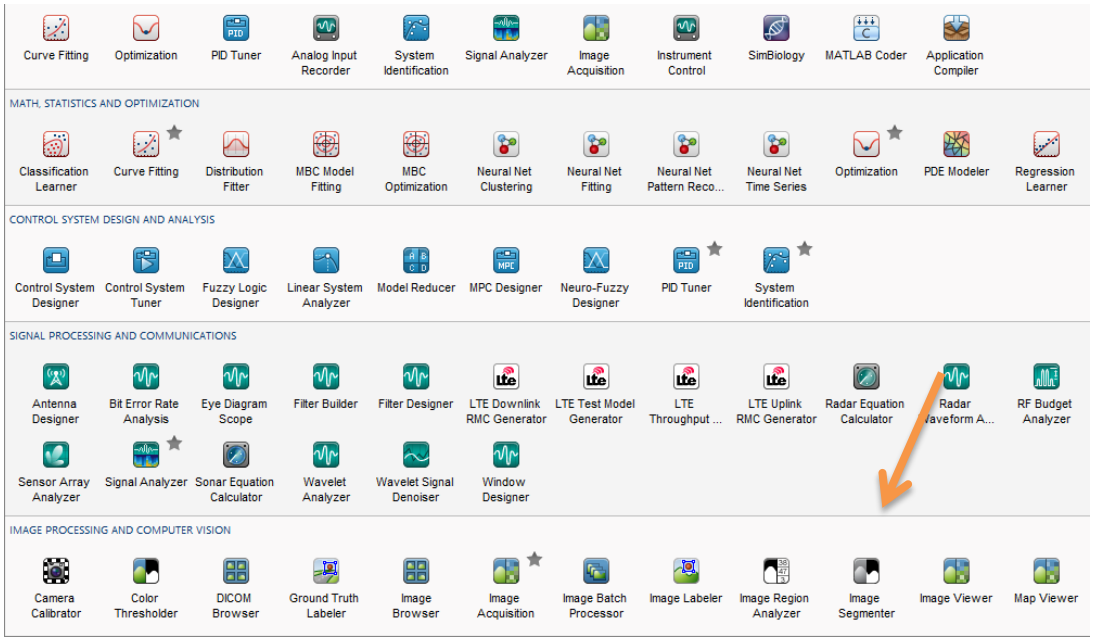



Figura 2.7.Toolboxes que incorpora Matlab.

El primer paso sería seleccionar de forma manual que elementos queremos segmentar y cuáles no. Para ello usaremos el comando Graph cut.

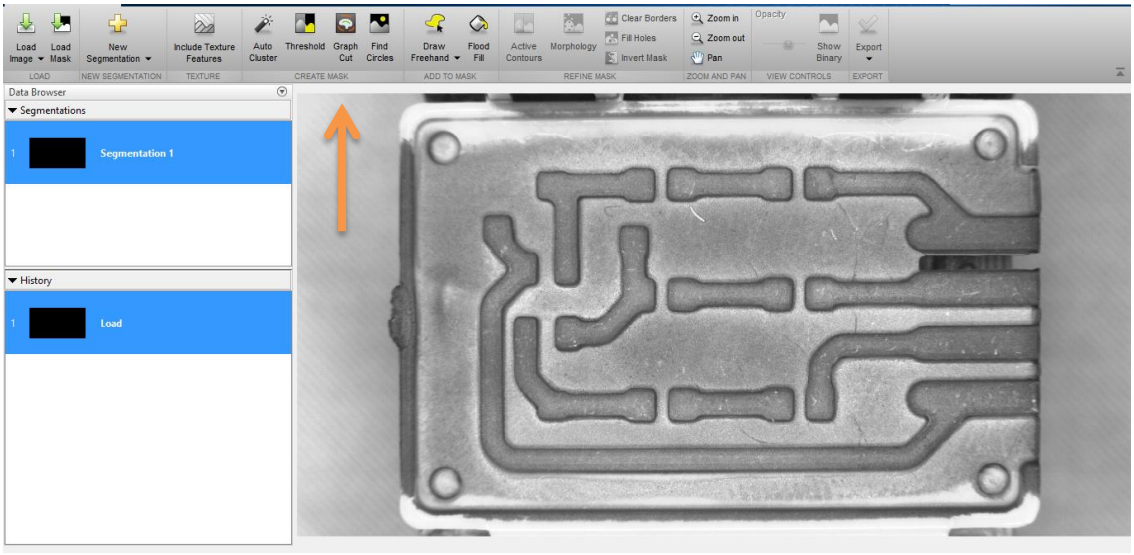



Figura 2.8. Interfaz Image Segmenter.

A continuación marcamos con ayuda del comando Mark Foreground los elementos que queremos segmentar, en nuestro caso las pistas de la PCB.

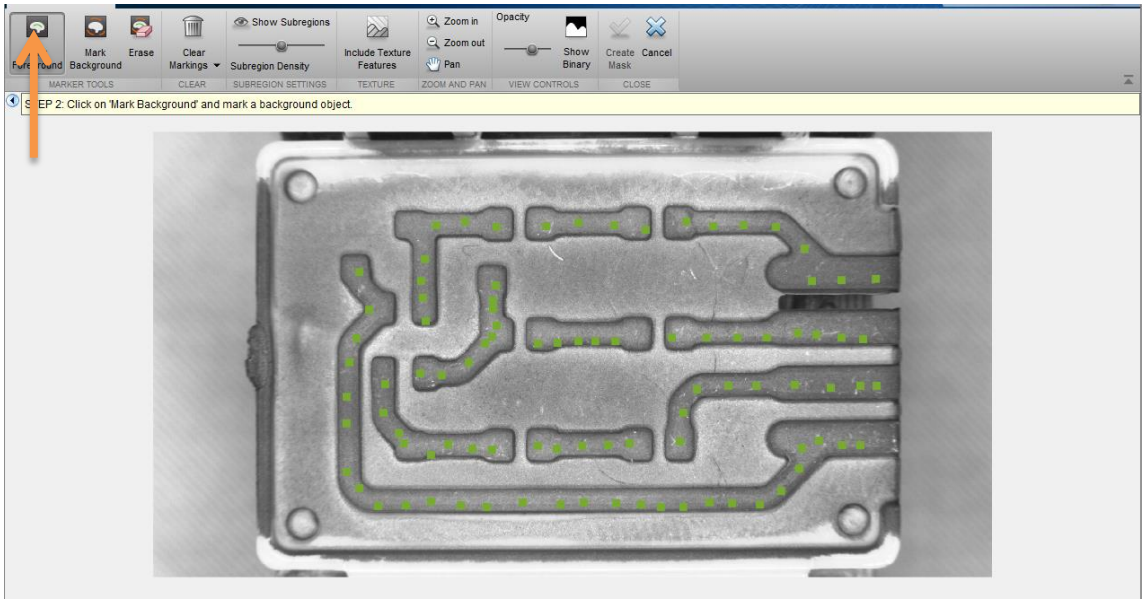



Figura 2.9. Interfaz Graph Cut: Mark Foreground.

El siguiente paso sería marcar los elementos que no queremos segmentar y que por lo tanto van a formar parte del fondo para ello usaremos el comando Mark Background.

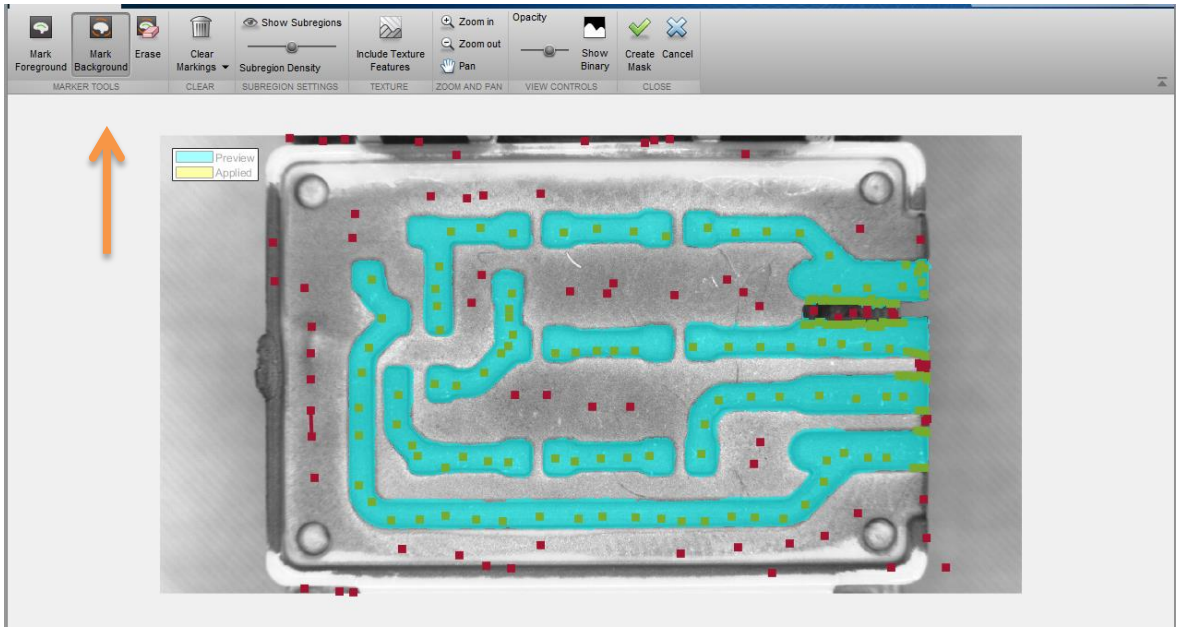



Figura 2.10. Interfaz Graph Cut: Mark Background.

Una vez que ya hemos marcado los elementos necesarios para obtener la máscara, con el comando Create Mask ya obtenemos la máscara.

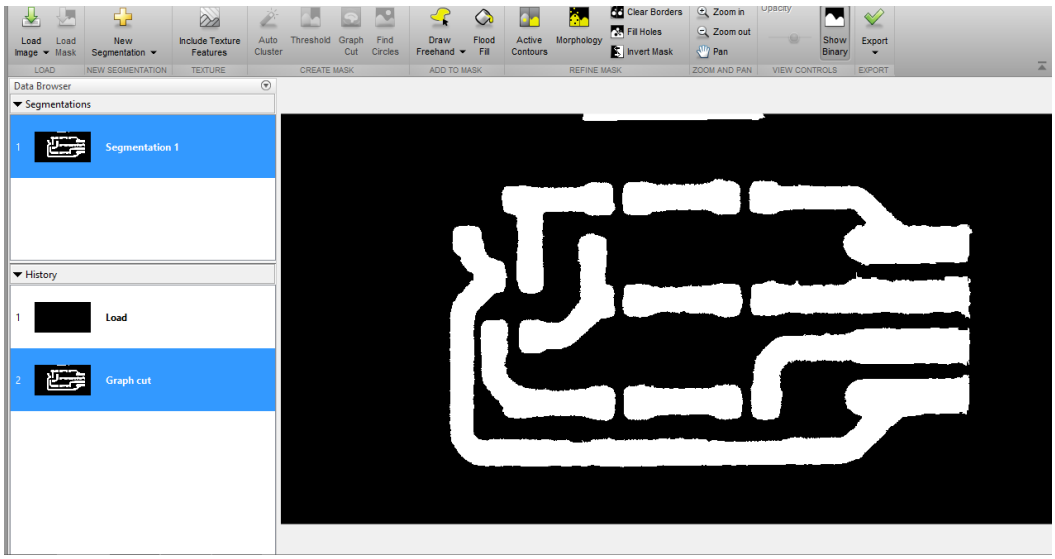



Figura 2.11.Máscara final creada con Image Segmenter.

Aunque hayamos obtenido la máscara, que era la finalidad que perseguíamos, esta no es la forma más óptima ya que debemos seleccionar manualmente los píxeles de las regiones que queremos segmentar. Es por este motivo que una vez adquiridos los conocimientos básicos en la asignatura de

Sistemas de percepción decidimos hacer un programa que construyera la máscara automáticamente.

Las distintas etapas que podemos identificar en el algoritmo de obtención de la máscara son las siguientes:


Ilustración 3. Etapas en el procesado de la imagen para la obtención de la máscara.

A continuación, estudiaremos cada una de las etapas mencionadas anteriormente:

- **Adquisición.**

Guo, F., & Guan, S. (2011). Research of the Machine Vision Based PCB Defect Inspection System:"In the AOI system, the quality of acquiring images is one of the key technologies,which depends on the camera devices and the environment".[ En el sistema AOI, la calidad de la adquisición de imágenes es una de las tecnologías clave, que depende de los dispositivos de la cámara y del entorno.] International Conference on Intelligence Science and Information Engineering. doi:10.1109/isie.2011.47

Como se dice en la cita anterior la adquisición es una etapa fundamental en el procesado .En primer lugar debemos elegir el tipo de iluminación más idónea con el fin de que las características que queremos ver, en este caso los defectos, resalten más y faciliten la etapa de procesado. Tras un estudio nos decantamos por un sistema de iluminación radial de tipo directa. En este sistema, la luz proviene del perímetro del eje de la cámara y esto hace que se

reduzcan sombras, se suavicen texturas y se minimicen la influencia de imperfecciones.

Figura 2.12. Modelo de la iluminación HPR-150SW.

Para la adquisición usaremos un modelo de cámara: Mako G223C PoE. Con la ayuda de un anillo extensor para modificar la distancia focal de nuestra cámara, conseguimos obtener imágenes donde se apreciaban los defectos de las pistas correctamente.


Figura 2.13. Modelo de la cámara Mako G223C PoE.

Para calcular la distancia de trabajo para que nuestra imagen este enfocada aplicaremos el modelo de pin-hole partiendo de los siguientes datos.

| Características generales | |
|---|---|
| Resolución de la cámara | 2048x1088 |
| Tamaño píxeles | 5,5 x 5,5 µm |
| Distancia focal objetivo | 35 mm |
| Anillo extensor | 5mm |
| Campo de visión eje horizontal | 30 mm |
| Campo de visión eje vertical | 20 mm |

Tabla 2.1 Datos para aplicar el modelo pin-hole.

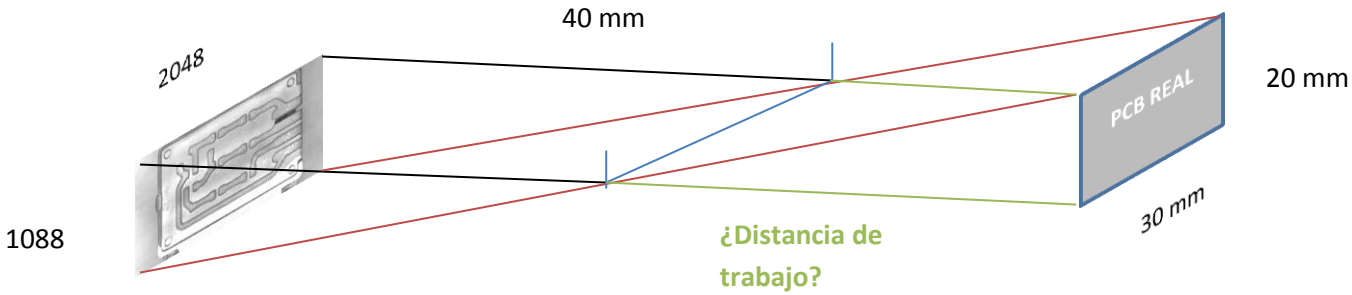



Ilustración 4. Modelo pin-hole.

Teniendo en cuenta los datos podemos calcular la distancia de trabajo aplicando semejanza de triángulos.

$$\frac{2048 * \ 5{,}5 * 10^{-3} mm}{40\ mm} = \frac{30\ mm}{dt} \rightarrow \boldsymbol{dt = 106.5\ mm}$$

Una vez que elegimos el material adecuado para la adquisición, nos dispusimos a adquirir imágenes. Para realizar pruebas de adquisición usamos una toolbox de Matlab: 'Image Acquisition'.

- **Preprocesado.**

El preprocesado de imágenes es el conjunto de técnicas cuyo objetivo fundamental es obtener, a partir de una imagen original, una imagen resultado en la cual se hallan mejorado ciertas características que posteriormente facilite el procesado sobre ella. Los principales objetivos que se persiguen con esta etapa son:

- Suavizar la imagen: reduciendo la cantidad de variaciones de intensidad entres pixeles vecinos.
- Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imagen.

Nosotros nos centraremos en las dos últimas. En primer lugar llevaremos a cabo el realce y posteriormente una reducción de ruido. Si lo hiciéramos a la inversa, en el caso de que el ruido no se eliminara completamente sería realzado en la siguiente etapa.

***Realzar bordes***

Con esto pretendemos aumentar el contraste de la imagen haciendo que las pistas destaquen más sobre el resto de la imagen. El contraste señala la dispersión en el nivel de gris de una imagen. Una imagen poco contrastada indica que hay poca variabilidad en el nivel de gris de la imagen. Este efecto mencionado, se muestra en un histograma muy concentrado.

La fórmula del contraste sería la siguiente, donde LxA es la dimensión, o(x,y) la imagen original y μ la media de intensidad de los todos los píxeles.

$$Contraste = \sigma^2 = \frac{1}{L * A} \sum_{(i,j)\epsilon\, S} (o(x,y) - \mu)^2$$

Se escogieron cuatro funciones para llegar al objetivo final y estas son:

- **Función imsharpen.**

  Devuelve una versión mejorada de la imagen de entrada de escala de grises o en RGB. A, donde las características de la imagen, como los bordes, se han afilado mediante el método enmascaramiento nítido . Esta técnica de enmascaramiento consiste en que la imagen se afila restando una versión sin nitidez de la imagen de sí misma.

- **Función imadjust.**

  Asigna los valores de intensidad de la imagen en escala de grises a nuevos valores. Para ello satura el 1% inferior y el 1% superior de todos los valores de píxel. Esta operación aumenta el contraste de la imagen de salida.

- **Función localcontrast.**

  Esta función mejora el contraste local de la escala de gris o de la imagen en RGB, todo ello con conciencia de bordes.

- **Función locallapfilt.**

  Filtra la escala de grises o la imagen RGB con un filtro laplaciano local, donde sigma caracteriza la amplitud de bordes y alpha controla el suavizado de los detalles.

Los resultados que obtuvimos aplicando las funciones mencionadas anteriormente fueron los siguientes.

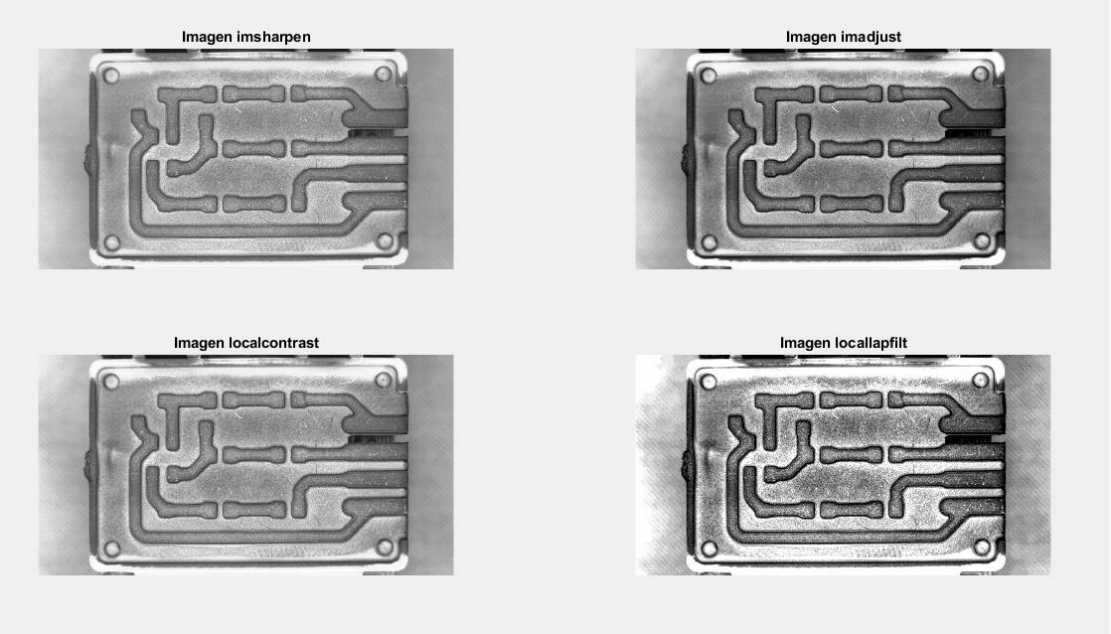



Figura 2.14. Resultados del aumento de contraste.

Como podemos observar obtenemos buenos resultados con la función 'imadjust' y con 'locallapfilt'. Pero descartamos esta última porque a pesar de que realza bastante los bordes de las pistas, también aumenta las zonas de brillo de la imagen.

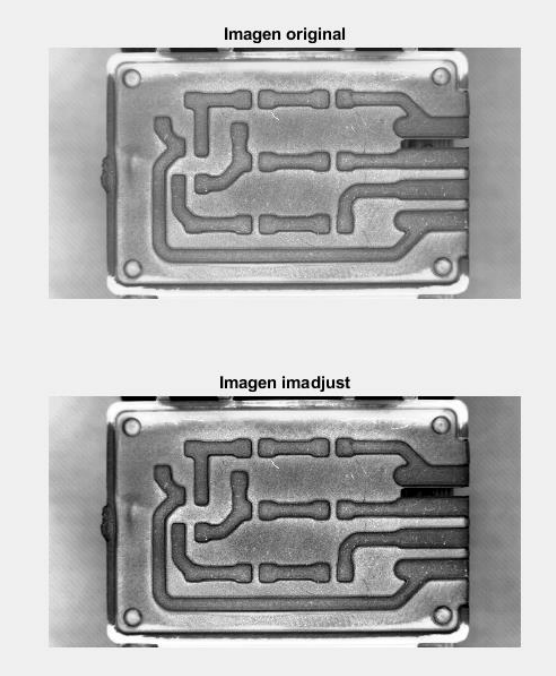



Figura 2.15. Justificación de la elección de imadjust.

## ***Eliminar ruido***

El ruido es la información indeseada que contamina la imagen. Consiste en la aparición de señales ajenas a la imagen original, especialmente en las zonas de sombra de la imagen. Se materializa en la imagen como alteraciones de brillo y color que provocan la existencia del conocido 'grano'. Estos 'granos' no son más que píxeles que no se corresponden con luminancia y tonalidad real de la fotografía. Existen tres posibles causas del ruido en una imagen digital las cuales son: Ruido de fotones (se corresponde con el ruido aleatorio correspondiente a los fotones); ruido Front end (este tipo de ruido está relacionado con la construcción del sensor) y ruido back end (este ruido tiene lugar cuando el procesador de nuestra cámara convierte la señal en un archivo digital).Para eliminar el ruido hemos escogido tres filtros de la gama de filtros existente.

- **Filtro average.**

Este filtro es de tipo lineal y por tanto se basa en la convolución de la imagen o(*x,y)* con una máscara obteniendo como resultado una imagen r(x,y)

de dimensión LxA, cuyos valores de intensidad para cada punto se han calculado con la media de los valores de intensidad de los píxeles en el entorno de vecindad.

$$r(x,y) = \frac{1}{L * A} * \sum_{(i,j) \in S} o\ (i,j)$$

- **Filtro gaussiano**.

Filtro lineal en el que la operación de convolución sigue una discretización gaussiana de media cero y varianza sigma.

$$r(x,y) = \frac{1}{\sqrt{2\pi * \sigma}} * e^{\frac{x^2+y^2}{2*\sigma^2}}$$

- **Filtro de la mediana.**

Filtro no lineal en el cual los píxeles de la nueva imagen se generan calculando la mediana de los píxeles que pertenecen al entorno de vecindad del píxel.

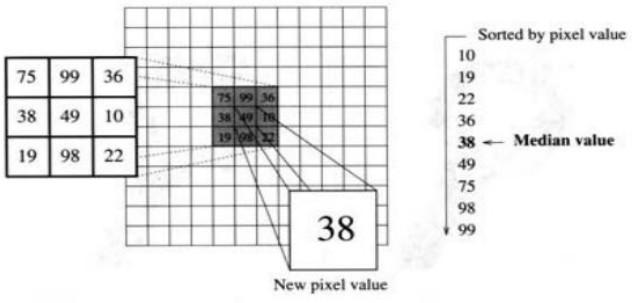



Figura 2.16. Explicación filtro de la mediana.

El resultado de aplicar los distintos filtros fue el siguiente:

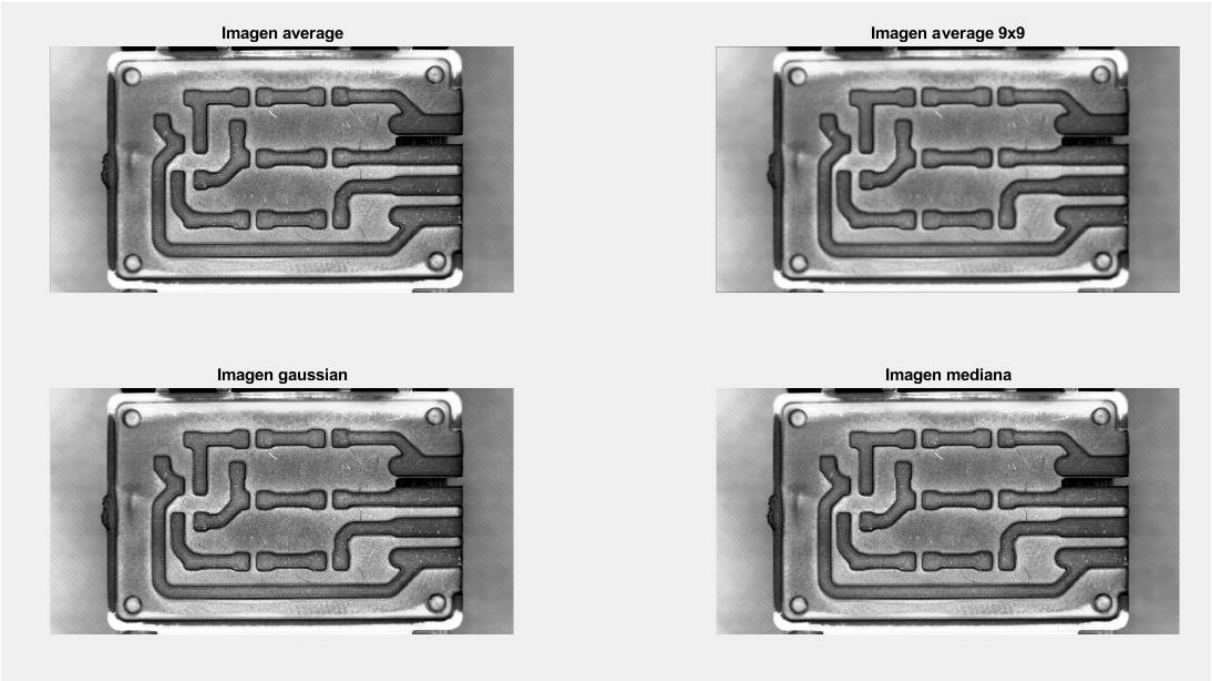



Figura 2.17.Reducción de ruido.

Los resultados que obtenemos no son tan evidentes como en la etapa de aumento de contraste. Sin embargo tras analizar en detalle cada uno de los resultados, nos decantamos por el resultado de la aplicación del filtro de la media 9x9.

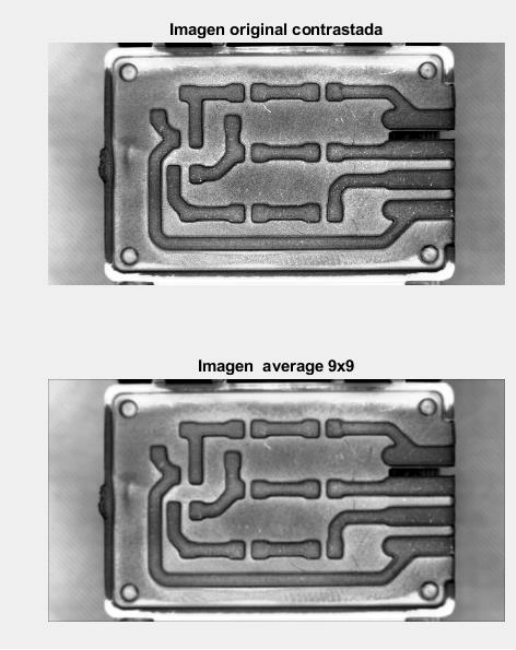



Figura 2.18. Justificación de la elección del filtro average 9x9.

- **Detección de bordes.**

El objetivo principal que se pretende con el uso de esta técnica es resaltar las fronteras del objeto que queremos identificar.

Los bordes de una imagen digital se pueden definir como la transición entre dos regiones de niveles de gris significativamente diferentes. Suministran una valiosa información sobre las fronteras de los objetos y puede ser utilizada para segmentar la imagen, reconocer objetos, etc. La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la primera y segunda derivada de los niveles de grises de la imagen. De la variedad de operadores que existen, nos centraremos en dos: 'Sobel' y 'Prewitt'. Estudiaremos ambos casos basándolos en la primera y la segunda derivada.

***Operador primera derivada***

La derivada de una señal obtiene las variaciones locales respecto a una variable, de este modo a mayor valor de derivada más rápida son las variaciones. Si la función tiene dos dimensiones, la derivada es el vector que apunta en la dirección de la máxima variación. Este vector recoge el nombre de gradiente y se define:

$$\nabla f(x,y) = \begin{bmatrix} \frac{df(x,y)}{dx} \\ \frac{df(x,y)}{dy} \end{bmatrix} \qquad Mag[\nabla f(x,y)] = \sqrt{\left(\frac{df(x,y)}{dx}\right)^2 + \left(\frac{df(x,y)}{dy}\right)^2}$$

$$\theta = arctag \frac{\frac{df(x,y)}{dx}}{\frac{df(x,y)}{dy}}$$

Siendo:

$$gradiente\ fila\ (G_F) \rightarrow \frac{df(x,y)}{dx} = \nabla f(x,y) = f(x,y) - f(x-1,y)$$

$$gradiente\ columna\ (G_C) \rightarrow \frac{df(x,y)}{dy} = \nabla f(x,y) = f(x,y) - f(x,y-1)$$

Si trasladamos el significado de la primera derivada al campo de detección de bordes, esta produce un cambio de pendiente en las zonas que la intensidad no es uniforme.

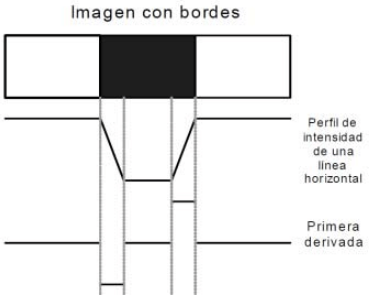



Figura 2.19. Efecto primera derivada.

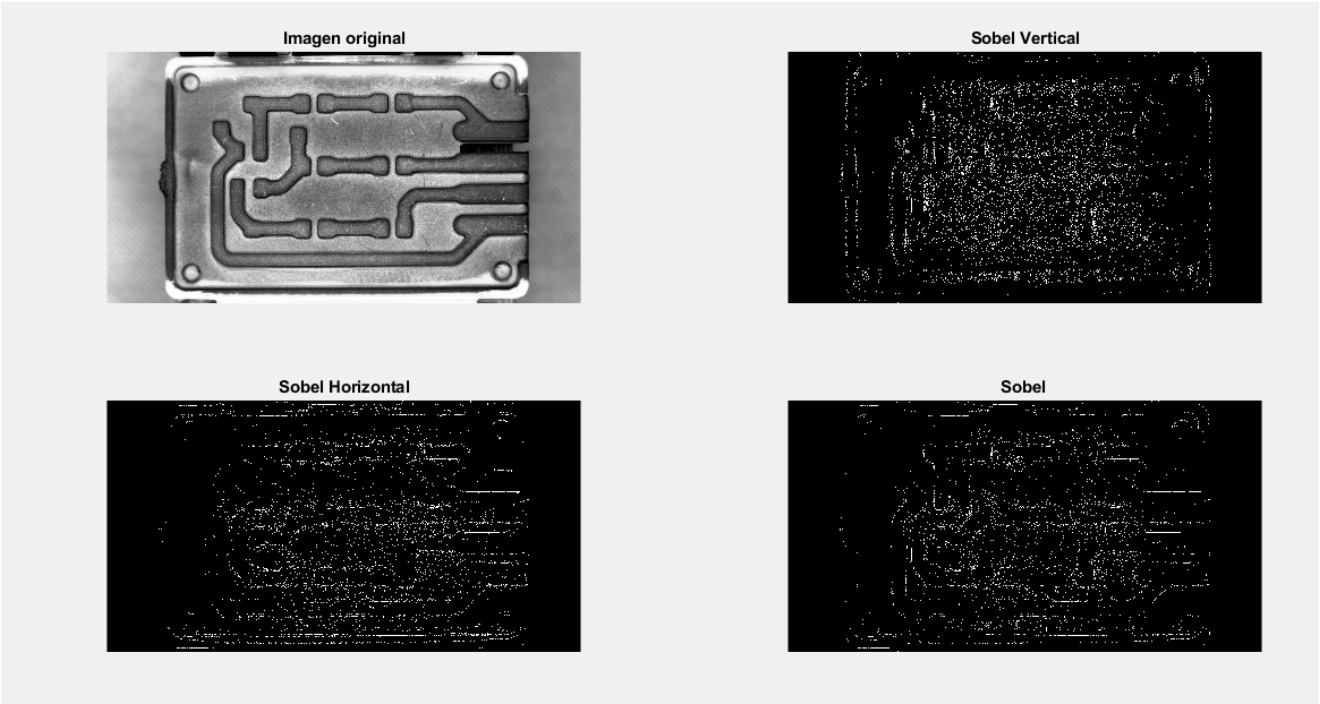



Figura 2.20. Operador Sobel basado en la primera derivada.

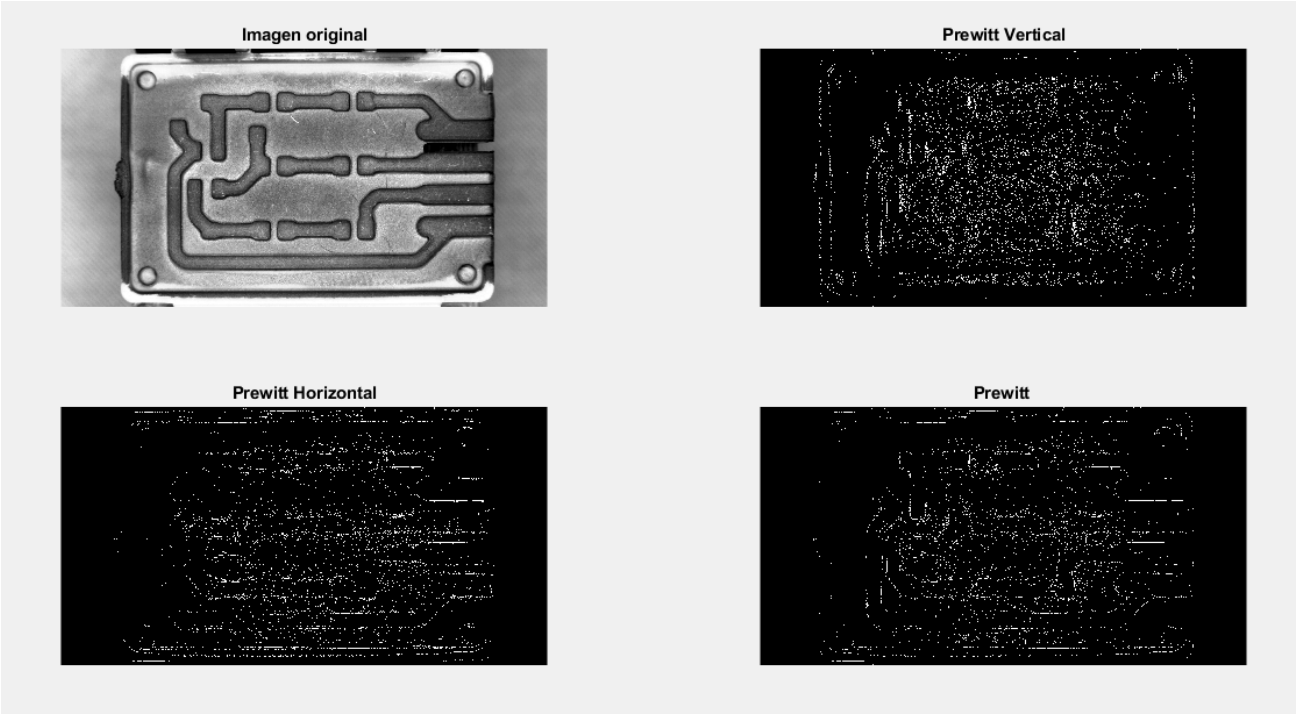



Figura 2.21. Operador Prewitt basado en la primera derivada.

Los operadores basados en la primera derivada no proporcionan buenos resultados en la detección de bordes, ya que aparece mucho ruido y muchos puntos de los bordes no los detecta.

***Operador segunda derivada***

Si la imagen presenta un cambio en la intensidad en una dirección, habrá segunda derivada, es decir un máximo de la primera derivada. La ventaja que tiene esta derivada frente a la primera es que proporciona la posición exacta del borde. La segunda derivada se define como:

$$\nabla^2 f(x,y) = \frac{d^2 f(x,y)}{dx^2} + \frac{d^2 f(x,y)}{dy^2}$$

Siendo:

$$Derivada\ G_F \rightarrow \frac{d^2 f(x,y)}{dx^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$Derivada\ G_C \rightarrow \frac{d^2 f(x,y)}{dy^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

De modo que quedaría:

$$\nabla^2 f(x,y) = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)$$

El efecto de la segunda derivada en el campo de detección de bordes es que provoca un cambio de signo en la localización del borde.

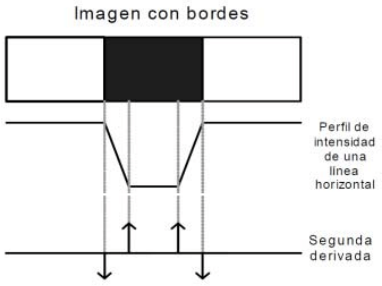



Figura 2.22. Efecto segunda derivada.

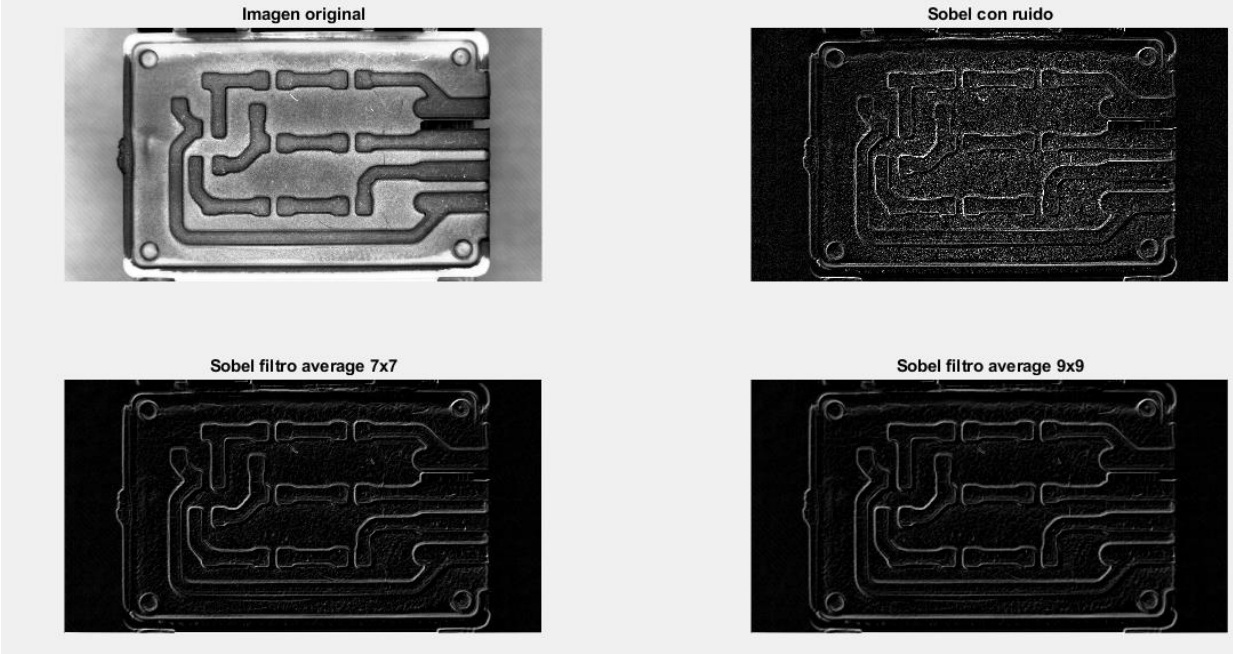



Figura 2.23. Operador Sobel basado en la segunda derivada.

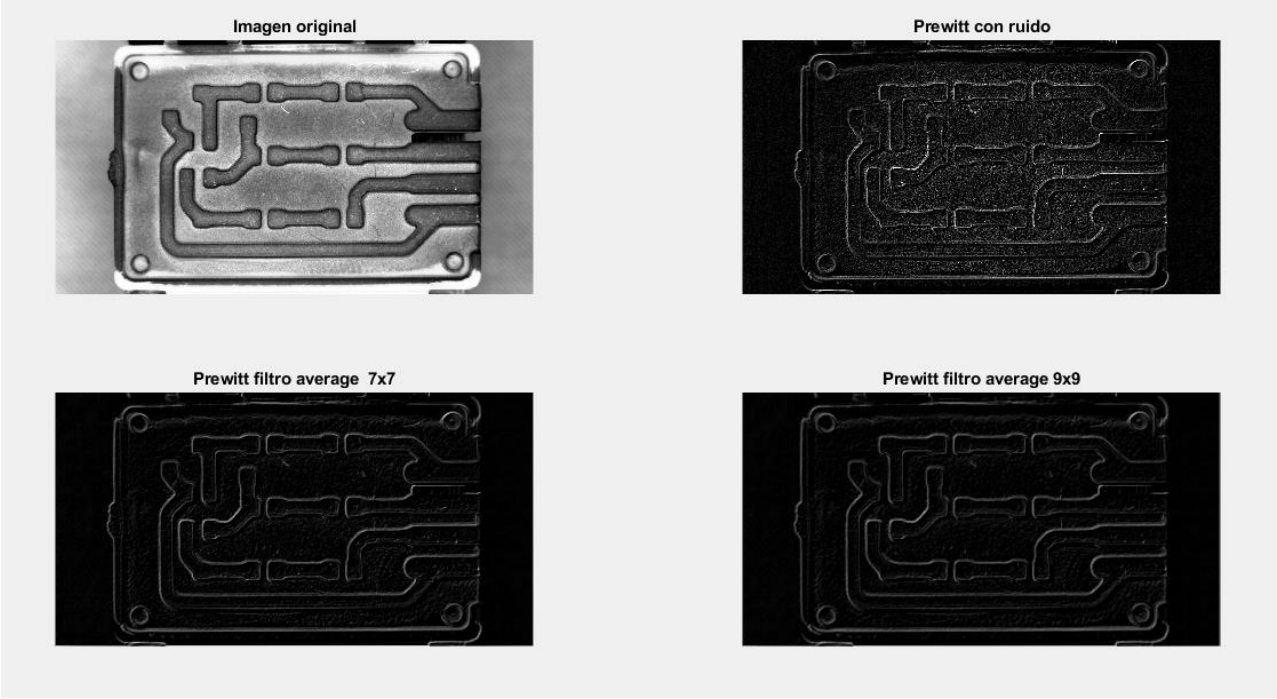



Figura 2.24. Operador Prewitt basado en la segunda derivada.

Aplicando la segunda derivada obtenemos mejores resultados que los obtenidos con la primera. Una vez que hemos obtenido la imagen gradiente, el siguiente paso sería pasar esta imagen a un detector de frontera para extraer los bordes. A pesar de que en un principio íbamos a segmentar mediante técnicas basadas en frontera decidimos elegir otro método de segmentación que resultará menos compleja y de solución más sencilla para nuestro caso de estudio. Por lo que no utilizaremos los resultados que hemos obtenido en esta etapa.

- **Segmentación.**

La segmentación consiste en el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. En concreto esta se trata de asignar una etiqueta a cada píxel dela imagen de forma que los píxeles que tengan la misma etiqueta compartirán también características visuales. Con esto se pretende simplificar y/o transformar la imagen en otra más significativa y más fácil de analizar. La principal aplicación de la segmentación es la identificación de objetos. Existen distintas formas de segmentar, estas son:

- Técnicas basadas en la frontera: Método que usa los bordes para hacer la segmentación.
- Técnicas basadas en regiones: Fórmula en la que se basa Image Segmenter, toolbox de Matlab vista al comienzo de este subapartado.
- Técnicas de umbralización: Procedimiento que hemos elegido para hallar la máscara.

***Segmentación mediante umbralización***

Usando este método se pretende diferenciar objetos de distintos niveles de gris. Se implementa realizando una binarización con un determinado umbral. Para la selección umbral existen distintas funciones dependiendo del tipo de umbral que queramos aplicar. Existen varios modos de obtener un umbral, nosotros nos centraremos en dos: Umbral global (se utiliza cuando existe una clara diferencia entre los objetos a segmentar y el fondo) y umbral adaptativo (indicado cuando el fondo no es constante y el contraste de los objetos varía en la imagen). Para obtener el umbral global usaremos la función 'graythresh' y para obtener el adaptativo usaremos 'adapthresh'. Aplicando estas funciones obtenemos las siguientes imágenes:

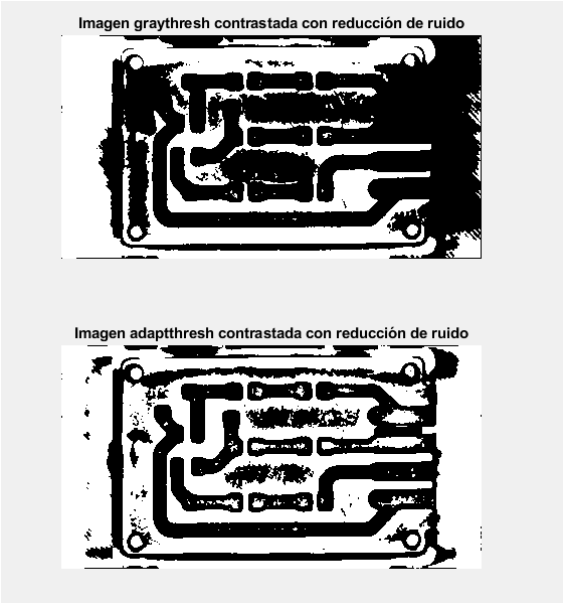



Figura 2.25.Resultado de distintas técnicas de umbralización.

Después de ver los resultados que hemos obtenido con las funciones. Vemos que con umbralización adaptativa la binarización conseguimos una mejor segmentación. Ya detectada la mejor técnica, hacemos pruebas con distintos umbrales adaptativos.

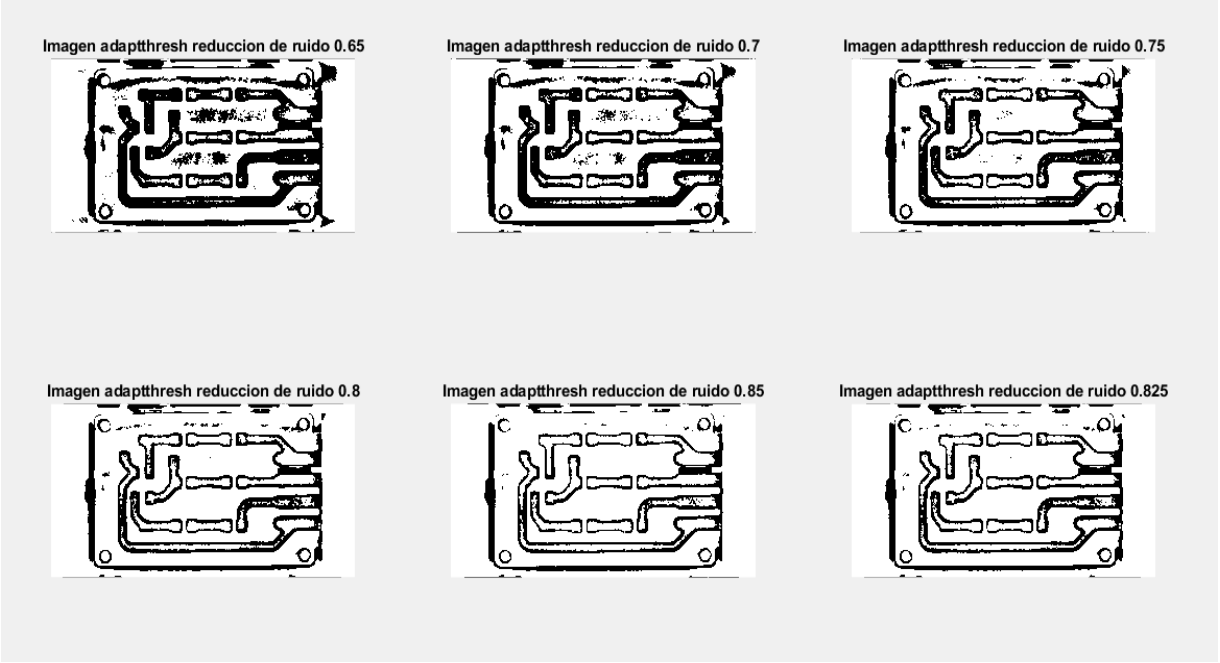



Figura 2.26.Pruebas con distintos umbrales adaptativos.

Obtenemos buenos resultados con valores altos de umbrales, nos quedaremos con el umbral: 0.825. A continuación hacemos el complemento de la imagen de esta forma los pixeles de la pista pasarán a ser los píxeles activos.

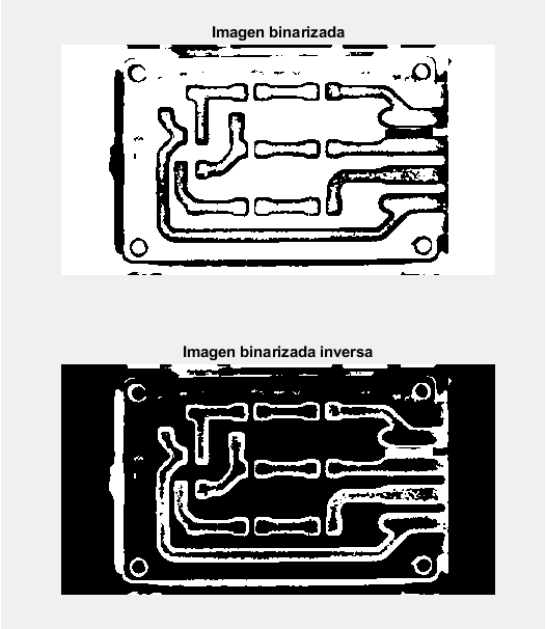



Figura 2.27.Imagen inversa binarizada.

Una vez que tenemos los píxeles de las pistas activos podemos rellenar los huecos mediante la función ‘imfill’.

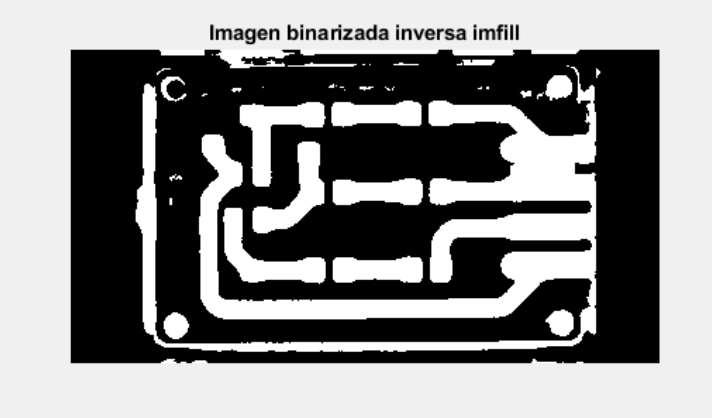



Figura 2.28.Imagen binarizada inversa imfill.

Estamos más cerca de la solución final, pero podemos observar que existen regiones que detectamos y que no nos interesan. Con la función ‘bwareaopen’ eliminamos las regiones de un área menor que el umbral que le pasemos, en este caso menor que 20000 y de esta forma obtenemos el siguiente resultado.

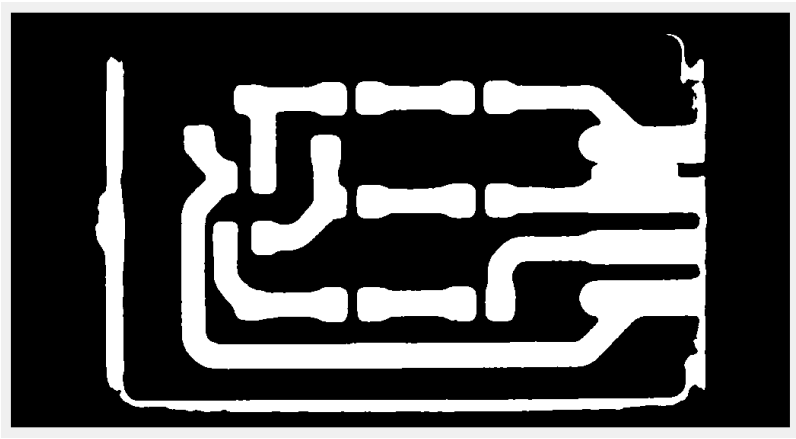


Figura 2.29.Imagen binarizada tras bwareaopen.

La región que queda por eliminar no podemos quitarla con 'bwareaopen' ya que posee un área muy similar a las pistas. Una solución es seleccionar con la función 'imcrop' la región con la que nos queremos quedar.

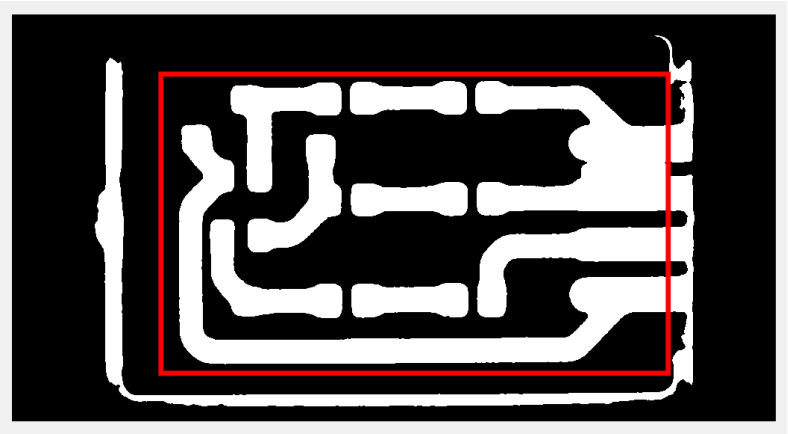


Figura 2.30. Selección de las pistas con imcrop.

Una vez que tenemos las coordenadas y las dimensiones del rectángulo copiamos el rectángulo en otra imagen que hemos definido previamente con las mismas dimensiones que el resto de imágenes y en negro.

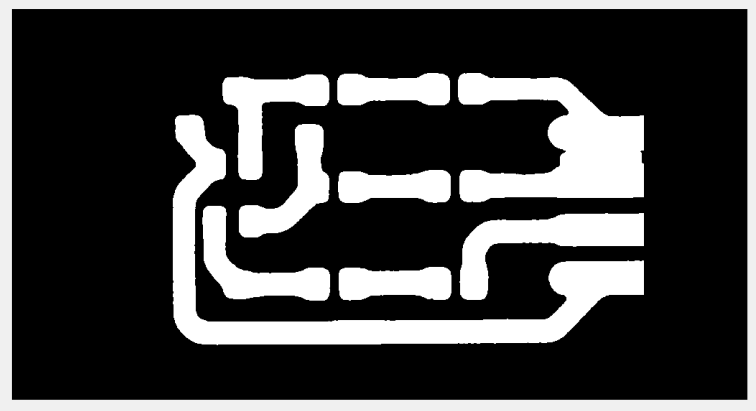


Figura 2.31.Máscara final.

## Aplicación de la correlación

Tras obtener la máscara que vamos a usar como plantilla podemos aplicar ya la correlación. Una posible solución sería aplicar alguna función de correlación (como corr2) que nos devolviera un coeficiente que indicara la similitud entre la máscara y la imagen que queremos analizar. Sin embargo nosotros compararemos ambas imágenes mediante operaciones matemáticas que nos permitan obtener una imagen en la que se vean exclusivamente los defectos, para poder contabilizar dichos defectos y en función de un umbral establecido, decidir si la PCB es defectuosa o no.

En primer lugar escogemos una PCB que tenga defectos visuales en sus pistas.

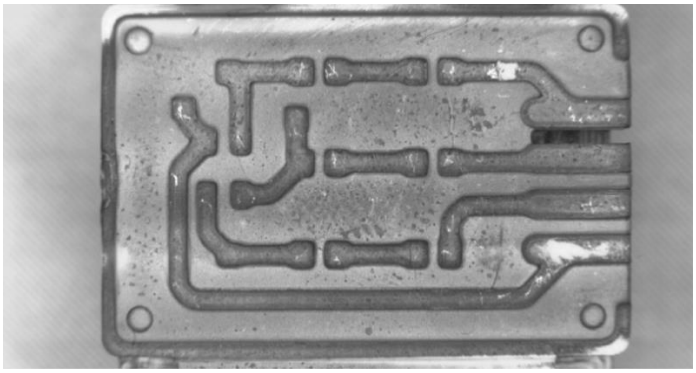


Figura 2.32. PCB analizada.

El siguiente paso sería binarizar esta imagen. Tras varias pruebas con distintos tipos de binarizado y distintos umbrales, obtuvimos buenos resultados con un binarizado adaptativo y con un umbral alto, en concreto de 0.9.

Figura 2.33. Imagen binarizada umbral de 0.9.

En el resultado que obtuvimos al binarizar, los defectos de las pistas se destacaban, pero también lo hacían muchas otras regiones que no nos interesaban. Para eliminar las partes indeseadas usaremos la máscara que obtuvimos anteriormente. Como los defectos aparecen en blanco (regiones activas), si queremos hacer una comparación con la máscara debemos hacer la inversa de la imagen de los defectos, para que estos pasen a ser negros, y al hacer una comparación obtengamos las diferencias. Si no de otra forma al ser las pistas blancas y los defectos blancos no podríamos hacer ninguna comparación.

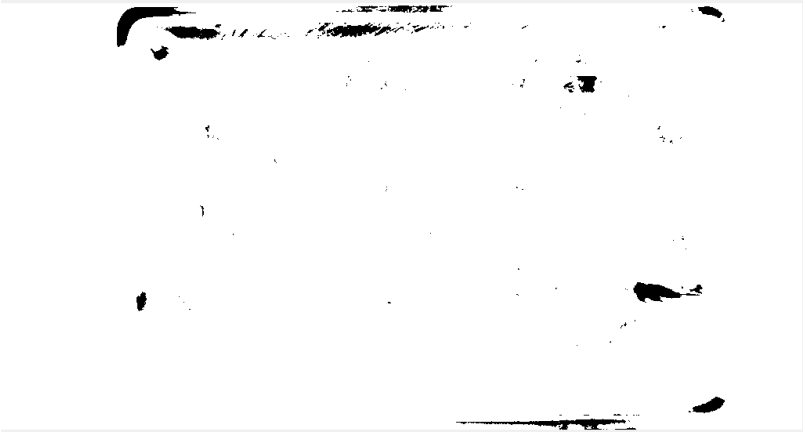


Figura 2.34. Imagen inversa de los defectos.

Con la función 'imshowpair' podemos mostrar la imagen que queremos analizar y la imagen que usamos como plantilla. Las regiones que de las pistas que difieran en ambas imágenes aparecerán en rosa. A través del cálculo del 'coeficiente de jaccard', obtenemos un parámetro que nos indica la similitud de estas dos imágenes, siendo el cero que no coinciden en nada y el uno que coinciden completamente.

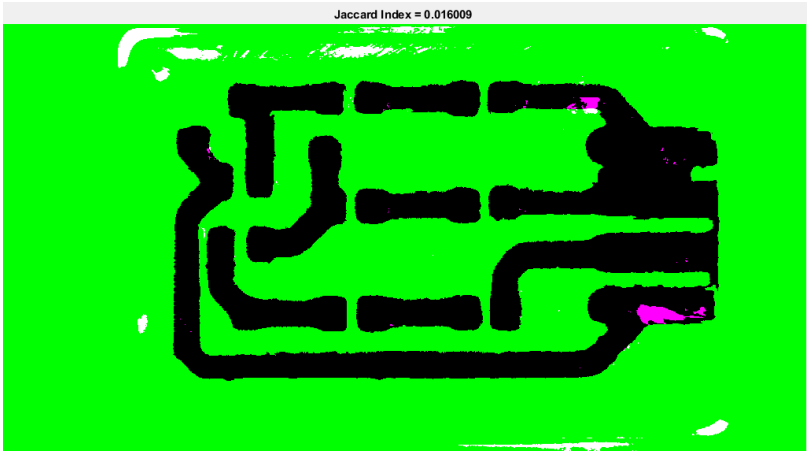



Figura 2.35. Comparación imagen original y plantilla.

Con la operación anterior podemos identificar las regiones que queremos resaltar a través del uso de la máscara. Si multiplicamos las imágenes obtenemos el siguiente resultado.

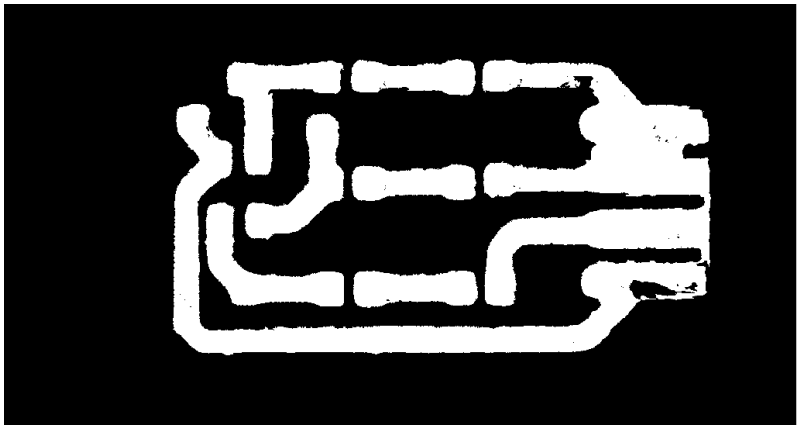


Figura 2.36. Resultado de la multiplicación.

Si comparamos esta imagen con la máscara, llegamos a nuestro objetivo que era destacar únicamente los defectos de las pistas.

Figura 2.37. Defectos visuales en las pistas.

Si queremos estudiar estos defectos debemos hacer la inversa para que pasen a ser píxeles activos (con valor 1). Con la función 'nnz' contabilizamos los elementos que no son cero, de modo que contabilizamos los píxeles de defecto y ya tenemos un valor para hacer una discriminación mediante un umbral que establezcamos.


Figura 2.38. Resultado final.

### 2.2.1.2.-Implementación en OpenCV.

Tras elaborar el programa en Matlab, el siguiente paso fue trasladarlo a Java (lenguaje que analizaremos en apartados posteriores) con la ayuda de la librería OpenCV.

OpenCV (Open Source Computer Vision) es una biblioteca de código libre de visión artificial creada por Intel. Fue diseñada teniendo en cuenta la eficiencia en cuanto a los gastos de recursos computacionales. Desde su creación se ha utilizado en una enorme cantidad de aplicaciones entre las cuales podemos encontrar desde sistemas de seguridad hasta aplicaciones de control de procesos. Esto se debe a que dispone de una gran gama de funciones que abarcan áreas como: reconocimiento de objetos; calibración de cámaras; visión estérea y visión robótica. Esta librería es muy usada para propósitos comerciales y de investigación debido a que está publicada bajo la licencia BSD que permite el uso de OpenCV para estos fines.


Figura 2.39. Logo OpenCV.

Empleando esta librería hemos creado una clase que se encarga de realizar todo el procesado. Esta clase es Processing la cual está asociada al agente AProcessing. Para entender cómo funciona la analizaremos en detalle.

**Atributos:** Sólo posee un único atributo, log, de tipo protected. La función de este atributo es guardar el identificador de la clase.

**Métodos:** Tiene una serie de métodos cuyo objetivo final es llevar a cabo el procesado. Para optimizar el código se ha construido un método por cada función a realizar, en lugar de hacer un único método. Para llegar a nuestro fin,

los métodos serán llamados dentro de otros métodos y de esta forma conseguiremos una mayor encapsulación del código. A continuación explicaremos cada método empezando por la base hasta llegar al método final.

- **muestraMascaraConDefectos.**

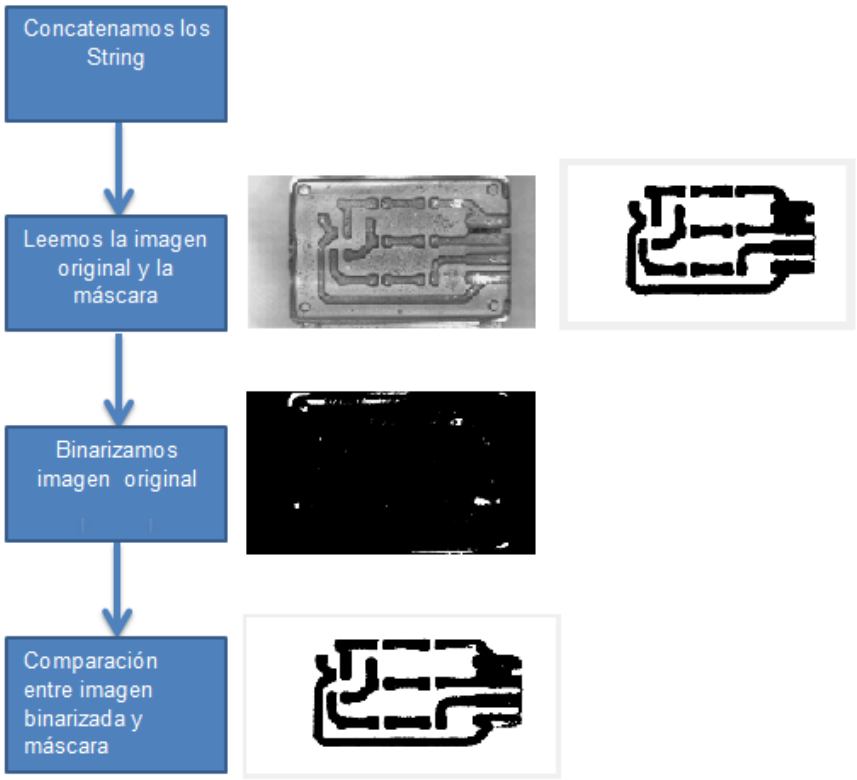



Ilustración 5. Esquema método muestraMascaraConDefectos.

Este método tiene cuatro parámetros de entrada de tipo 'String'. Estos son: la ruta donde guardamos la máscara sin incluir el nombre de esta; el nombre de la máscara; la ruta de la imagen que queremos analizar sin incluir el nombre y el nombre de la imagen.

En primer lugar concatenamos los 'String' de la ruta con su nombre correspondiente para poder leer la imagen original y la máscara respectivamente. Leemos las imágenes en escala de gris con la ayuda de la función 'imread'.

A continuación creamos un objeto tipo Mat donde almacenaremos el resultado de la binarización de la imagen original. Obtenemos como resultado los defectos de las PCB en blanco. Para distinguir los defectos de las pistas del resto de defectos que no nos interesa haremos de uso de la función 'bitwise_or'. Esta hace una operación de convolución de las dos imágenes si cualquiera de los píxeles es blanco (con valor uno) da uno y si no da cero (negro) de esta forma veremos solamente los defectos de la pista de la PCB en blanco. El resultado de esta función lo guardamos en otro objeto de tipo Mat el cual será a su vez el parámetro de salida del método.

- **ContabilizadorAreasDefectos.**

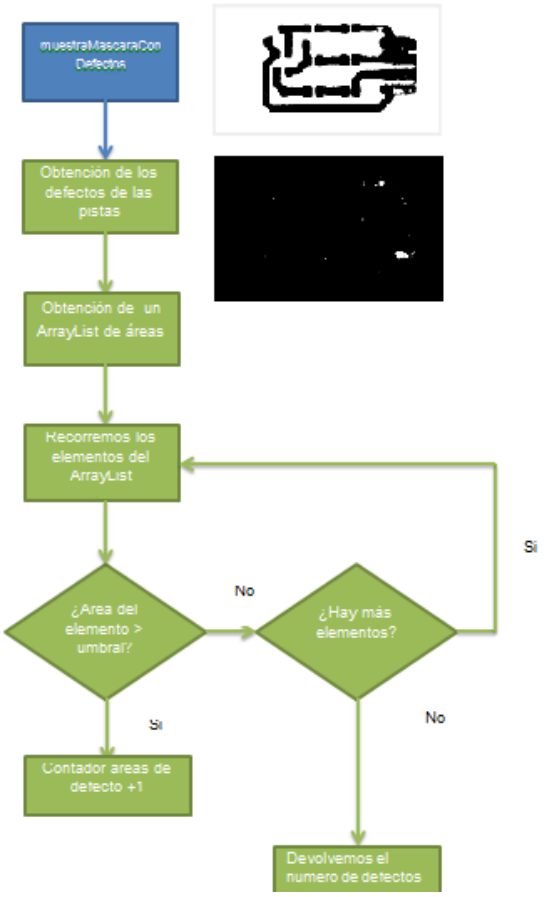



Ilustración 6.Esquema método ContabilizadorAreasDefecto.

Este método tiene los parámetros de entrada que tenía el método anterior junto con otro de tipo entero y que será el umbral que establezcamos nosotros para identificar si se trata de un defecto.

El primer paso sería llamar al método anterior para obtener el resultado donde las pistas se ven en negro y los defectos de estas en blanco.

Una vez hecho esto, debemos obtener una imagen donde se vean los defectos únicamente. Para ello usaremos la función bitwise_xor donde hacemos una comparación de la imagen que hemos obtenido de la operación anterior con la máscara binarizada. Al hacer la operación xor hacemos que las diferencias entre ambas imágenes sean regiones activas (blanco) y el resto regiones no activas (negro), de esta forma solo nos quedamos con los defectos de las pistas.

A continuación encontramos estas áreas de defecto con la función findcountours (como el regionprops en Matlab). Esta función nos devuelve un ArrayList donde almacenamos las distintas áreas. Recorriendo cada elemento del ArrayList accediendo al área de cada uno y comparándolo con el umbral podemos contabilizar el número de defectos que posee la PCB el cual será el parámetro de salida del método.

- **Procesado.**

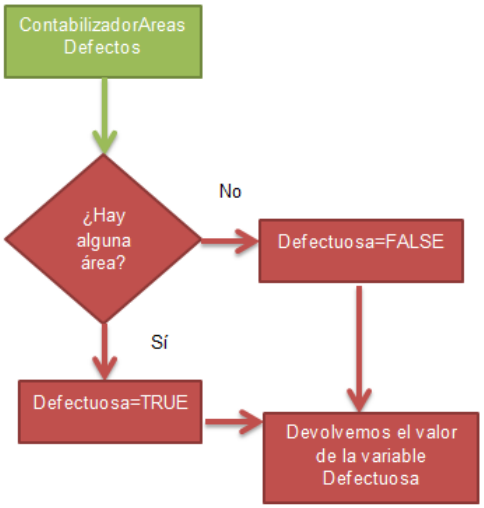



Ilustración 7.Esquema método Procesado.

Tiene los mismos parámetros de entrada que el método 'ContabilizadorAreasDefectos', el cual llamaremos y si contabilizamos alguna área, indica que existe un defecto y por lo tanto devolveremos como parámetro de salida True y si no False.

### 2.2.2.-Programa de continuidad.

El desarrollo software de esta parte lo podemos dividir en función de los programas usados. En consecuencia distinguiremos entre:

- Programa de Java.
- Programa de Arduino.

#### 2.2.2.1.-Programa de Java.

Java es un lenguaje de programación de propósito general y orientado a objetos creado en 1991.Fue desarrollado por Sun Microsystem, (la cual fue adquirida por Oracle Corporation en 2010) y lanzado en 1996. La sintaxis de Java es una extensión de C y se basa en parte en C++, ya que Java surgió para corregir algunos de los inconvenientes de este y de C#. Las ventajas que presenta este lenguaje son:

- Usa la POO: La programación orientada a objetos se define como un métodos de implementación en el los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencia. Las ventajas que posee este tipo de programación frente a la metodología tradicional: fácil mantenimiento; permite la reutilización de partes de un programa para otro programa y facilita el diseño de programas de gran envergadura.

- Independencia de la plataforma: Permite la ejecución de un mismo programa en distintos sistemas operativos: Es posible ya que se trata de un lenguaje interpretado, esto significa que los programas que

usan este lenguaje se traducen a Byte codes los cuales son interpretados por una máquina virtual de java (JVM).

- Posee un recolector de basura automático: Esto hace que se eviten problemas de fugas de memoria. El programador determina cuándo se crean los objetos, y por otro lado el entorno es el encargado de gestionar los ciclos de vida de los objetos. Tanto el programa como otros objetos pueden tener localizado un objeto mediante una referencia. Si no existe ninguna, el recolector de basura de Java borra el objeto liberando de esta forma la memoria que ocupaba y previniendo posibles fugas, proporcionando seguridad al proceso de creación y eliminación de objetos.

Todas las ventajas que hemos mencionado hacen que Java sea uno de los lenguajes de programación más usados.

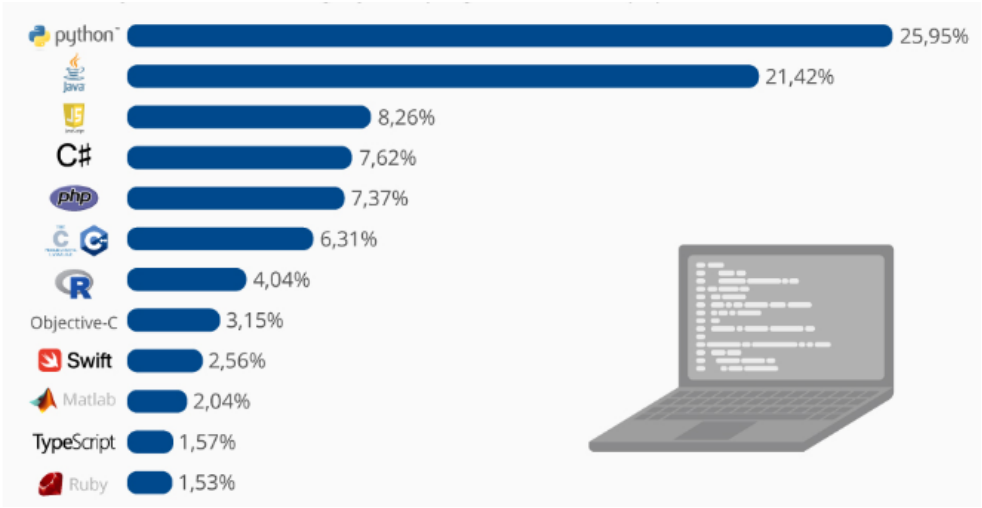



Figura 2.40. Ranking de los lenguajes más usados.

Por todas estas razones elegimos este lenguaje para programar la máquina. Otra razón de peso fue que se disponía conocimientos básicos sobre este lenguaje y esto hacía que no se partiera desde cero.

Una vez justificado el motivo de la elección del lenguaje y tras una breve explicación sobre este, procederemos a explicar el desarrollo del código para ello dividiremos el problema en dos partes:

- Máquina de Estados.
- Interfaz de usuario.

***Máquina de Estados***

Una máquina de estados consiste en un modelo de comportamiento de un sistema con entradas y salidas, en el cual las salidas dependen no sólo de las señales de entradas actuales sino de las anteriores. Las máquinas de estado son un conjunto de estados, cada estado sirve como intermediario en la relación entre una entrada y una salida. Los estados estarán separados entre ellos mediante transiciones.

Para implementar la máquina de estados usaremos una programación basada en agentes, esta consiste en una ejecución múltiple de hilos en paralelo. Un agente puede verse como es una entidad inteligente capaz de percibir su entorno, procesar tales percepciones y responder a su entorno de manera racional. Cada agente ejecuta al mismo tiempo un hilo con el código que tiene programado. En términos de programación pueden definirse como un proceso que existe dentro de un ambiente y que puede comunicarse mediante un protocolo de comunicación. Por lo tanto los agentes pueden comunicarse entre sí para enviarse mensajes, el envío de un mensaje será una transición entre dos estados de la máquina. Cuando un agente se comunica con un segundo agente enviando un mensaje solicitando un dato, en ese instante se para la ejecución del primer agente y el segundo agente contesta con otro mensaje enviando el dato.

Los agentes presentan dos métodos:

- **toInit:** Este método lo usaremos para inicializar el agente.
- **toProcess:** Este consiste en un ‘switch’ en el que cada ‘case’ se describe lo que se ejecuta tras la recepción de un mensaje, esto

normalmente es enviar un mensaje a otro agente. Cada 'case' corresponde a un mensaje y cada uno de ellos son atributos del propio agente. Para que podamos acceder a estos mensajes o atributos fuera del agente, estos deben inicializarse como variables estáticas.

En nuestro programa identificamos seis agentes distintos, a cada uno de ellos le asociaremos un color para distinguirlos en el esquema de la máquina de estados. Antes de analizar las máquinas de estado, hablaremos en primer lugar de cada uno de los agentes.

### AInterfaz

Este agente es el que gobierna la Interfaz de usuario. Para ello deberemos crear un objeto de la clase 'Interface' el cual nosotros hemos llamado 'miInterfaz'. Este agente se comunicará con los agentes de manejan las máquinas de estado con el fin de ir mostrando los resultados de los análisis. Para entender mejor el funcionamiento del agente mostraremos el siguiente diagrama.

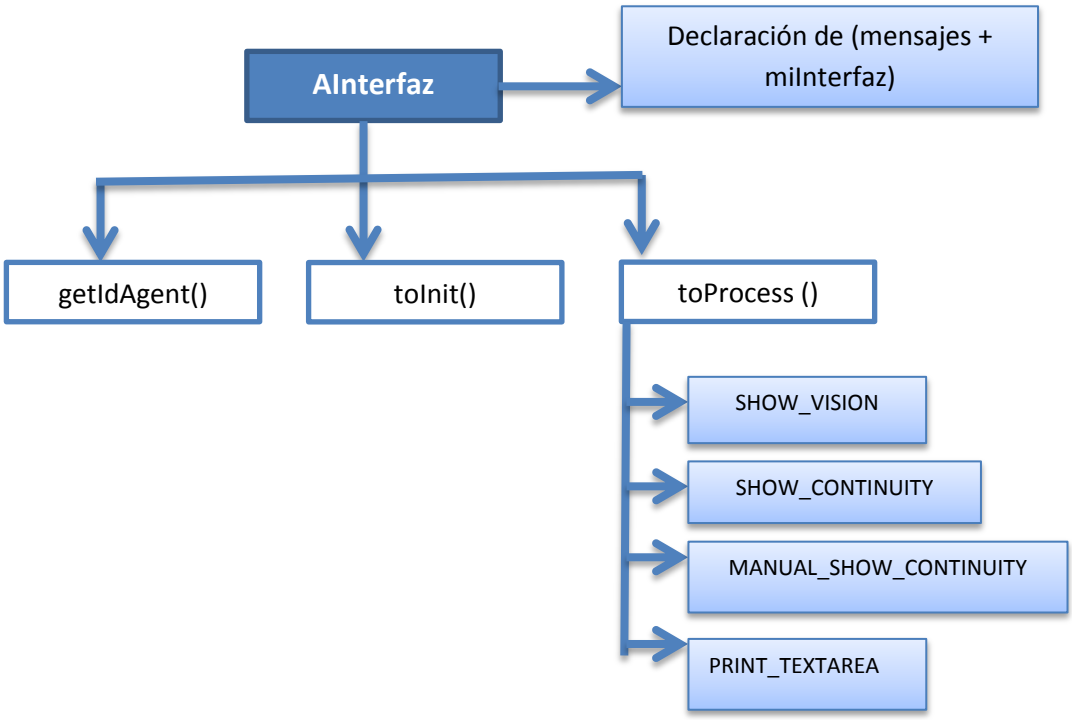



Ilustración 8. Agente AInterfaz.

**ASerial**

Es el agente encargado de hacer la comunicación serie con Arduino (para lo cual crearemos un objeto de la clase 'SerialArduino' llamado 'interfaceSerialArduino') y con el multímetro (para lo cual crearemos un objeto de la clase 'SerialMultimeter' llamado 'interfaceSerialMultimeter'). Para entender el funcionamiento del agente es necesario explicar antes las clases 'SerialArduino' y 'SerialMultimeter'.

La clase 'SerialArduino' consta de:

**Atributos:** Posee tres atributos. Uno es 'arduino' de tipo 'SerialPort' donde guardaremos el puerto del Arduino. Para usar la clase 'SerialPort' hemos importado la librería 'jSerialComm'. Otro de los atributos es 'ack' es de tipo booleano y lo usaremos como 'flag'. El último atributo es la cadena 'bufferString' donde almacenaremos los datos que recibamos por el puerto serie.

**Métodos:** La clase 'SerialArduino' posee ocho métodos que analizaremos individualmente.

- **openSerialConection**: Función que nos devuelve un booleano, el cual nos informa si conseguimos abrir el puerto de Arduino (openPort()) durante un segundo si al segundo el puerto no ha sido abierto lanzamos un mensaje de error a través de una excepción.
- **closeSerialConection**: Función que nos devuelve un booleano, el cual nos informa si conseguimos cerrar el puerto de Arduino (closePort()).
- **sendString**: Método para mandar una cadena por el puerto serie de Arduino. Para ello concatenamos a la cadena un retorno de carro y un salto de línea. A continuación lo convertimos a bytes los cuales los escribiremos por el puerto del Arduino.
- **isAck:** Devuelve el valor de la variable 'ack'.
- **setAck:** Asigna a la variable 'ack' el valor que le pasamos como parámetro a la función.

- **getBufferString():** Devuelve el valor del 'bufferString'.
- **setBufferString():** Asigna a la variable 'bufferString'el valor que le pasamos como parámetro a la función.
- **respondeEventos:** Método que se encarga de la gestión de eventos de la comunicación.

```
public SerialPortDataListener respondeEventos = new SerialPortDataListener() {


    @Override
    public int getListeningEvents() {
        return SerialPort.LISTENING_EVENT_DATA_AVAILABLE + SerialPort.LISTENING_EVENT_DATA_WRITTEN;
    }

    @Override
    public void serialEvent(SerialPortEvent evento) {
        if (evento.getEventType() == SerialPort.LISTENING_EVENT_DATA_AVAILABLE) {
            byte[] nuevosBytes = new byte[evento.getSerialPort().bytesAvailable()];
            evento.getSerialPort().readBytes(nuevosBytes, nuevosBytes.length);
            bufferString += new String(nuevosBytes);
            if (bufferString.endsWith("\n")) {
                log.info(bufferString);
                if (bufferString.contains("ok")){
                    setAck(true);
                }
                bufferString = "";
            }
        }
        if (evento.getEventType() == SerialPort.LISTENING_EVENT_DATA_WRITTEN) {

        }
    }
};
```

Figura 2.41. Gestión de eventos en el puerto serie del Arduino.

Inicialmente, hemos creado el objeto 'respondeEventos', el cual implementa los métodos de escucha y respuesta de eventos. El método 'getListeningEvents()' nos permite configurar la escucha de los eventos. En nuestro caso, estos eventos son los que producen cuando existen datos disponibles en el buffer de entrada y cuando un dato se ha enviado con éxito. Para ello, se utilizan las constante 'LISTENING_EVENT_DATA_AVAILABLE' y' LISTENING_EVENT_DATA_WRITTEN' respectivamente. El método "serialEvent()" es disparado cada vez que se produzca un evento. Una vez disparado, se identifica el evento y se actúa en consecuencia. Si el evento corresponde a "datos disponibles" ("DATA_AVAILABLE"), se crea un array de bytes ("nuevosBytes") del tamaño del mensaje de entrada. Posteriormente, se escribe en el array mencionado el mensaje recibido. Por

último, se crea un String con la información del array y lo acumulamos en "bufferString". En el momento que se complete el mensaje (se detecta mediante los caracteres finales salto de línea \n") hacemos un log de la información contenida en 'bufferString', comprobamos si el 'String' contiene el mensaje de 'ok' que manda Arduino cuando recibe un mensaje de Java. Si es así llamamos al método 'setAck' el cual ya hemos explicado. Finalmente vaciamos el 'bufferString'.

- **SerialArduino:** Método constructor por defecto en el que inicializamos las variables de 'arduino' y 'ack'. También llamamos a la función 'respondeEventos'.

La clase 'SerialMultimeter' tiene los mismos métodos que posee 'SerialArduino' a diferencia que posee un atributo más de tipo 'float' el cual llamamos 'datoMedido' (el cual tendrá sus correspondientes métodos de 'set' y 'get'). El nuevo atributo contiene el valor de la medida que el multímetro manda a Java, el cual gestiona la llegada de este dato al buffer con la función 'respondeEventos'. Esta función no es lo mismo que en 'SerialArduino', ya que en esta ocasión debemos convertir el dato 'String' a 'float' y almacenarlo en 'datoMedido'.

```
public SerialPortDataListener respondeEventos = new SerialPortDataListener() {


    @Override
    public int getListeningEvents() {
        return SerialPort.LISTENING_EVENT_DATA_AVAILABLE + SerialPort.LISTENING_EVENT_DATA_WRITTEN;
    }

    @Override
    public void serialEvent(SerialPortEvent evento) {
        if (evento.getEventType() == SerialPort.LISTENING_EVENT_DATA_AVAILABLE) {
            byte[] nuevosBytes = new byte[evento.getSerialPort().bytesAvailable()];
            evento.getSerialPort().readBytes(nuevosBytes, nuevosBytes.length);
            bufferString += new String(nuevosBytes);
            if (bufferString.endsWith("\n")) {
                setDatoMedido(Float.parseFloat(bufferString));
                log.info("Dato recibido por puerto serie: "+getDatoMedido());
                setMedida(true);
                bufferString = "";
            }
        }
        if (evento.getEventType() == SerialPort.LISTENING_EVENT_DATA_WRITTEN) {

        }
    }
};
```

Figura 2.42. Gestión de eventos en el puerto serie del multímetro.

Tras la explicación de las clases 'SerialArduino' y 'SerialMultimeter' para una mayor comprensión del agente 'ASerial' veremos el siguiente flujograma.

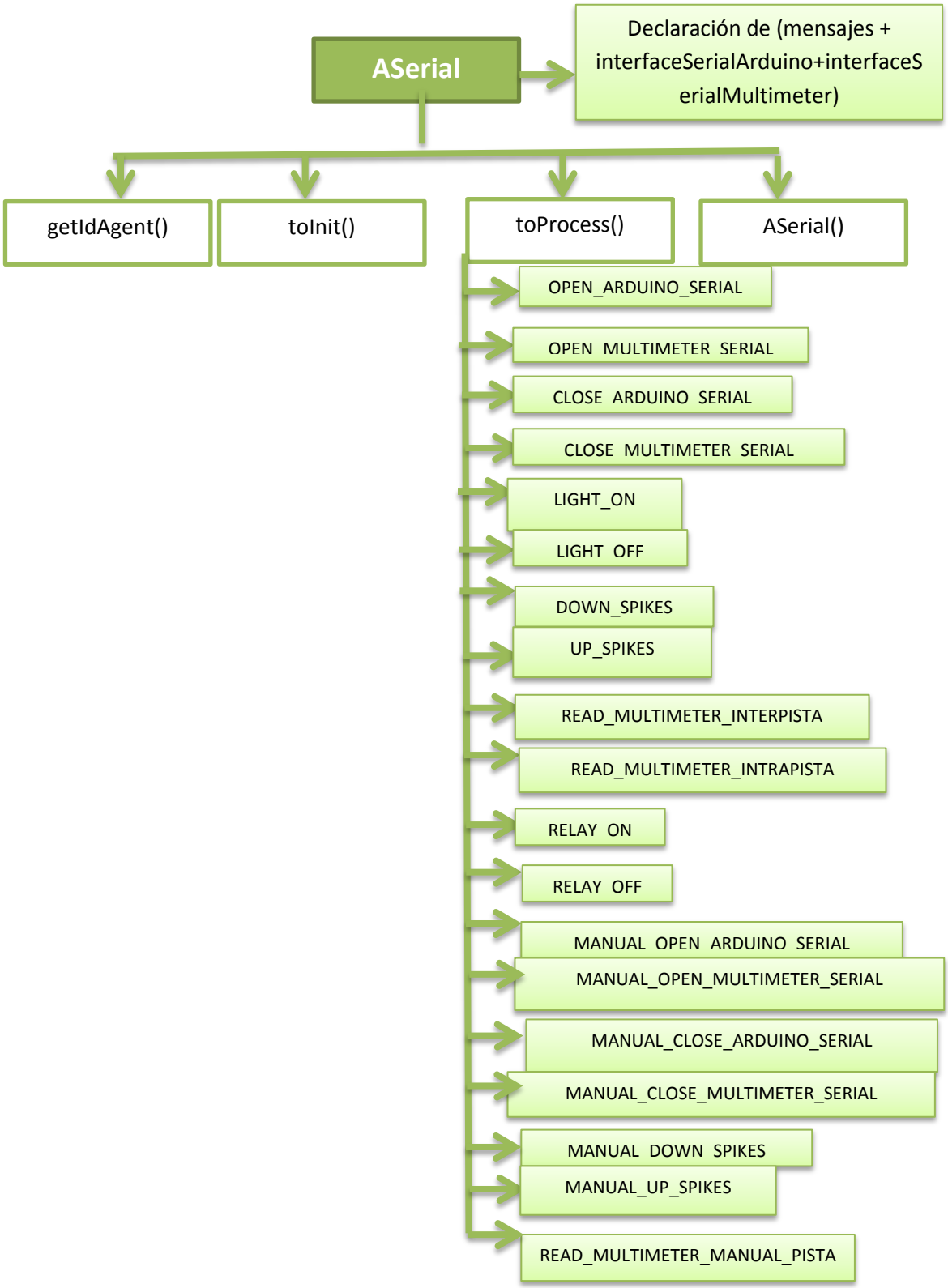



Ilustración 9. Agente ASerial.

### ACamera

Agente encargado de hacer la adquisición para ello le asociamos un objeto de la clase 'Camera' llamado 'interfaceCamara' la cual tiene una serie de métodos que permiten esta acción. El diagrama del agente sería.

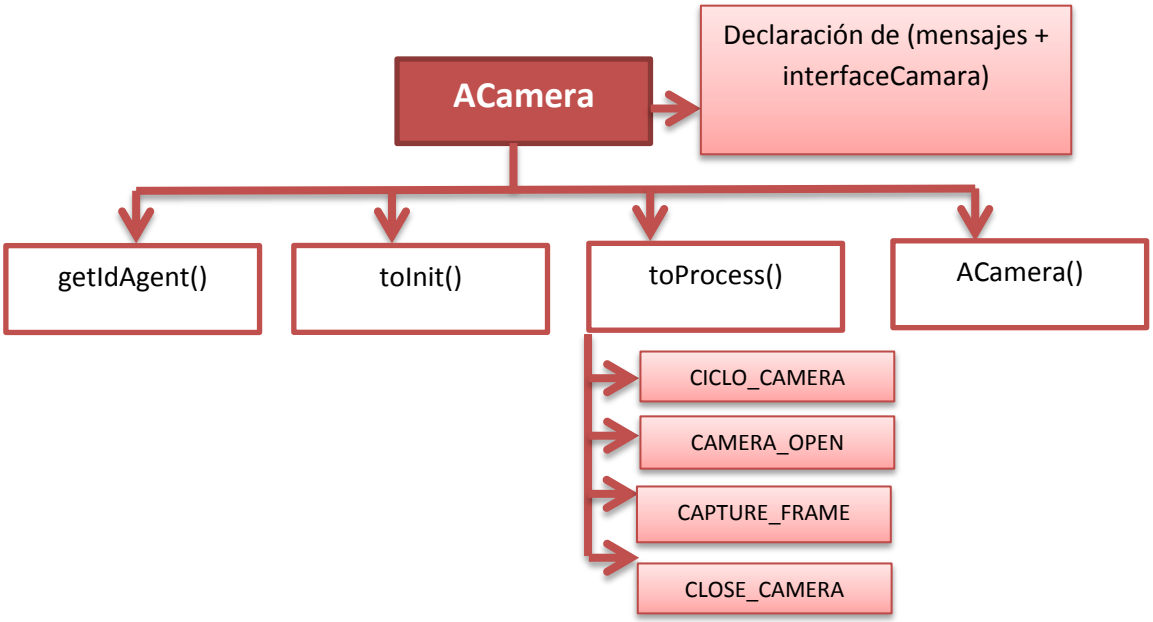



Ilustración 10. Agente ACamera.

### AProcessing

Este agente se encarga de llamar a los métodos de la clase 'Processing' explicada en el apartado de visión para realizar el procesado de la imagen que previamente hemos adquirido con el agente 'ACamera'. El funcionamiento del agente se ve en el siguiente esquema:

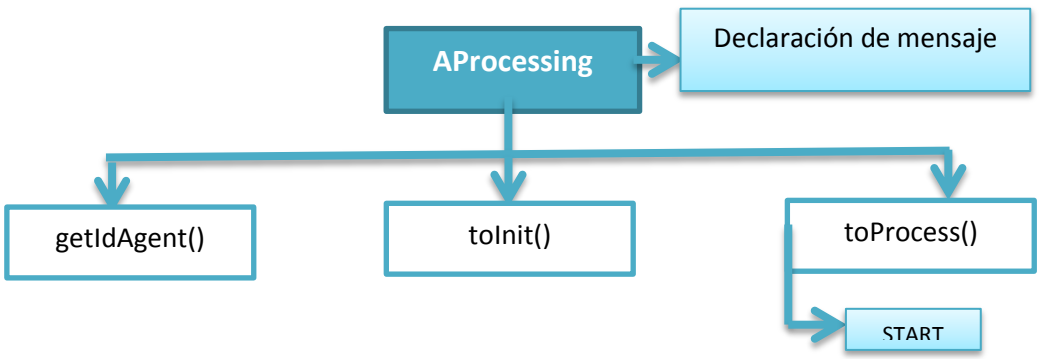



Ilustración 11. Agente AProcessing.

**AStateManager**

Es el agente que gobierna la máquina de estados del proceso automático. Va asociado a tres 'Map<Integer,Boolean>'. Uno de ellos es 'portMap' el cual usamos para almacenar el número de puertos que debemos abrir y los 'flags' que nos dicen si están activos. También podemos encontrar a 'readingMapIntraPista' y 'readingMapInterPista' los cuales almacenan el número de intrapistas e interpistas respectivamente y los 'flags' correspondientes. Para una mayor compresión de las medidas y lograr diferenciar las distintas medidas a realizar, 'intrapista' e 'interpista', se han creado tres clases: 'ViasModel'; 'PCBModel' e 'InterPistaTestModel'.La clase 'ViasModel' está destinada para representar el modelo 'intrapista'. Diferenciamos las siguientes partes:

**Atributos:** Posee tres atributos de tipo 'String', los cuales son: 'maskImage';'positive_digital_pin' y 'negative_digital_pin'. También cuenta con dos atributos booleanos, 'hasVisualDefect' y 'hasConductivity'. El último de los atributos es de tipo 'float' que recibe el nombre de 'measure'.

**Métodos:** Cada atributo tiene un método 'get' y 'set' por lo que esta clase dispone de doce métodos. El método 'get' devuelve el valor del atributo mientras que 'set' le asigna el valor que le hemos pasado por parámetros a la función.

Por otro lado, la clase 'InterPistaTestModel' representa el modelo 'interpista' y tiene las siguientes características:

**Atributos:** Posee dos atributos de tipo 'String', los cuales son: 'positive_digital_pin' y 'negative_digital_pin'. También cuenta con dos atributos más, uno de tipo booleano, 'hasConductivity' y otro de tipo 'float' que recibe el nombre de 'measure'.

**Métodos:** Cada atributo tiene un método 'get' y 'set' por lo que esta clase dispone de ocho métodos. El método 'get' devuelve el valor del atributo

mientras que 'set' le asigna el valor que le hemos pasado por parámetros a la función.

Por último, la clase 'PCBModel' con el fin de crear un modelo de PCB la cual se caracteriza por el nombre de la imagen de la PCB y el número de pistas.

**Atributos:** Posee un atributo de tipo 'String' llamado 'imageName'. Y un 'ArrayList' de elementos de tipo 'ViasModel' donde están las pistas de la PCB.

**Métodos:** Cada atributo dispone de un método 'get' y de un método 'set'. Para un mayor entendimiento del agente mostraremos el flujograma de este:

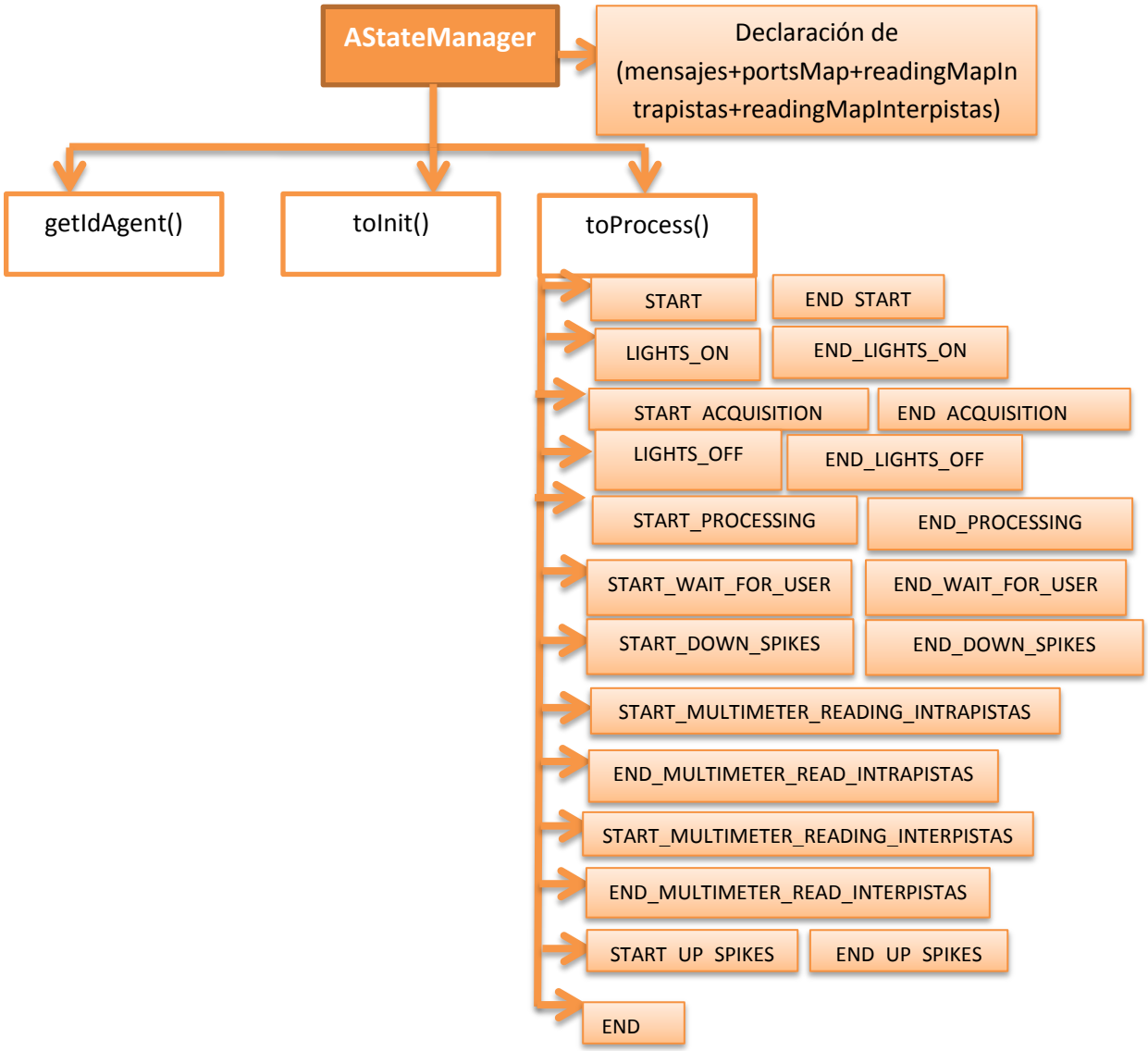



Ilustración 12. Agente AStateManager.

### AManualStateManager

Agente que gobierna la máquina de estados del proceso manual. Este proceso está indicado para cuando el operario quiere hacer una comprobación de continuidad de las pistas de forma manual, para ello puede seleccionar los dos puntos entre los que se medirá la continuidad. Sigue un proceso similar al 'AStateManager'.

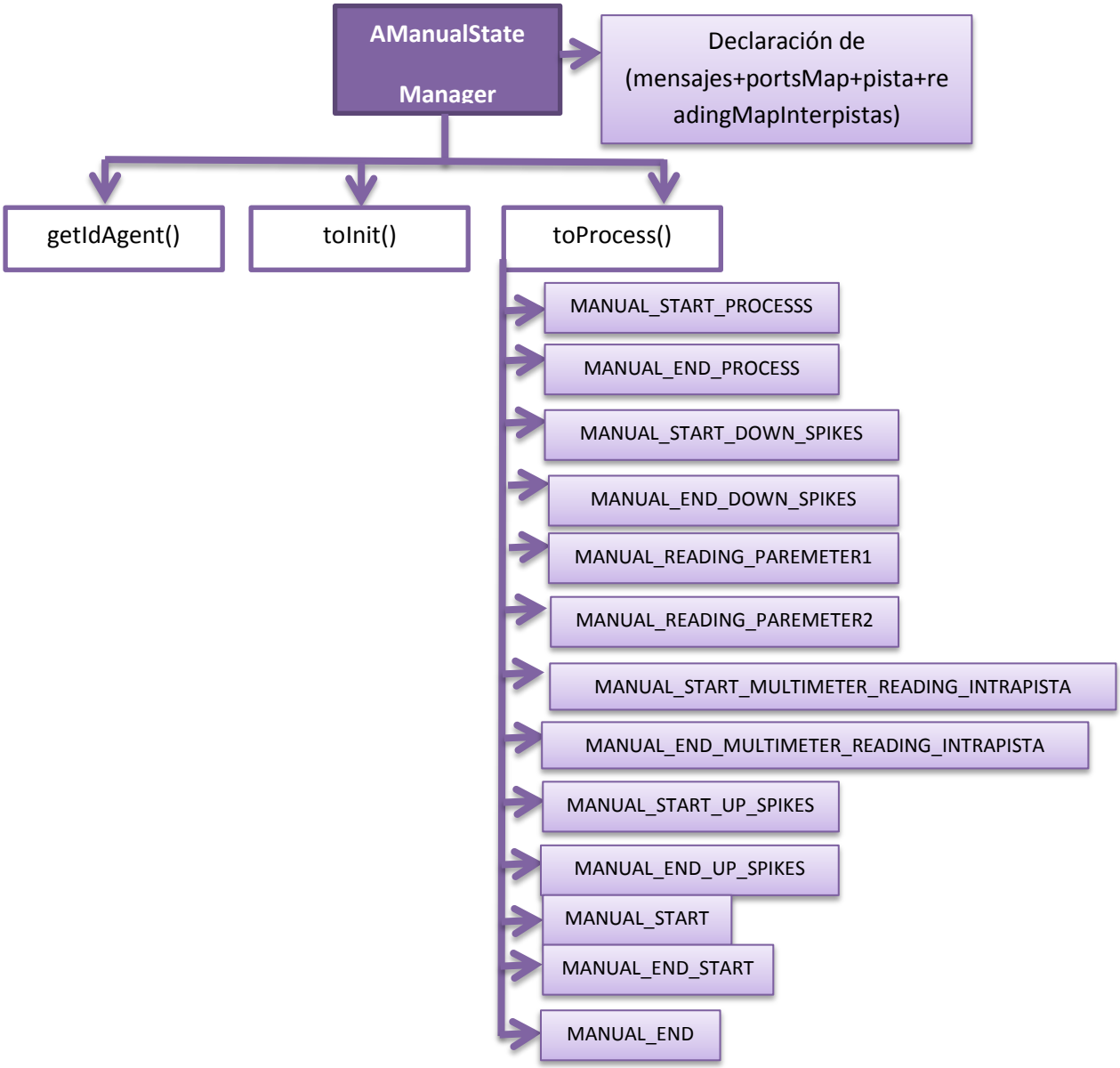
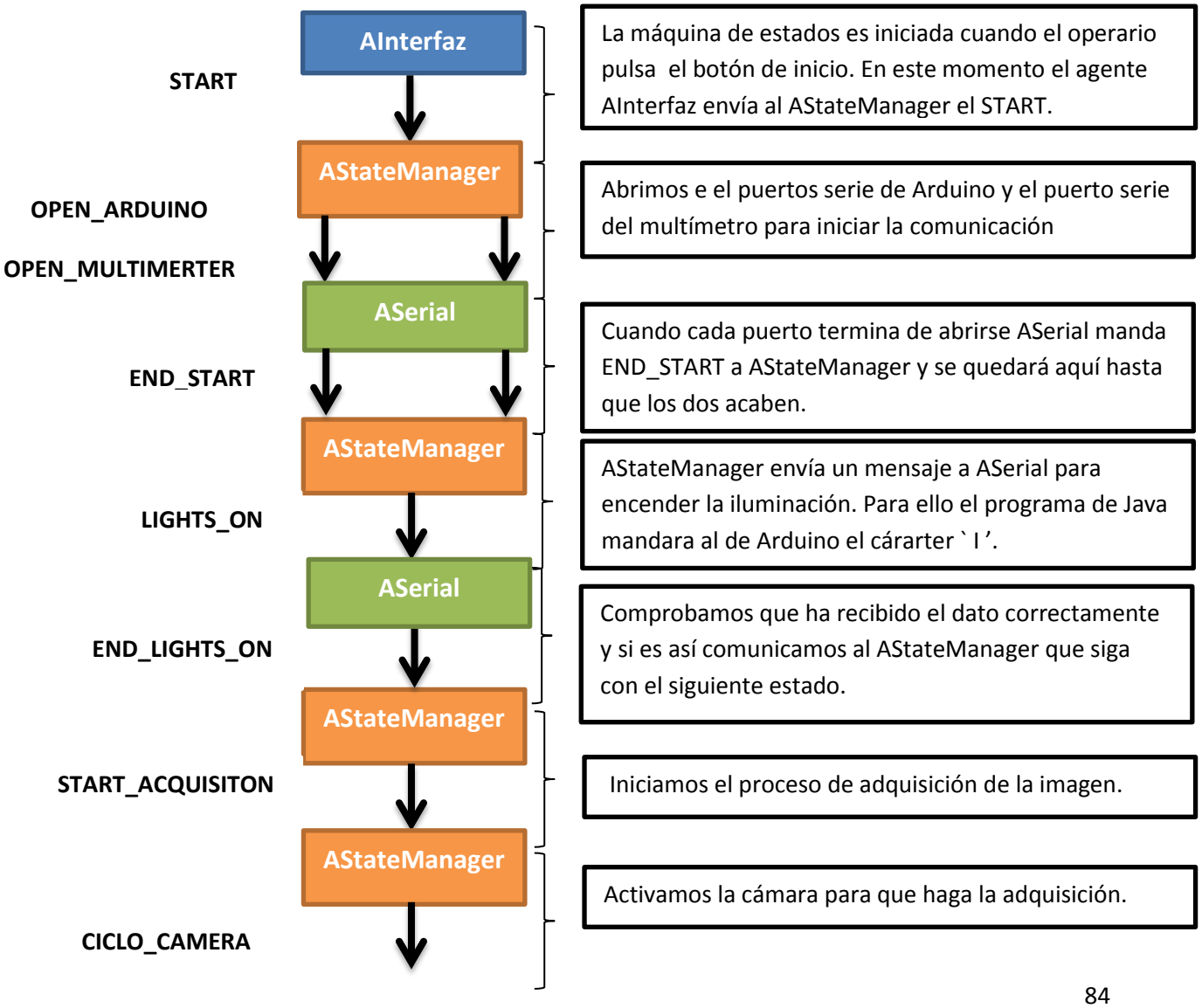


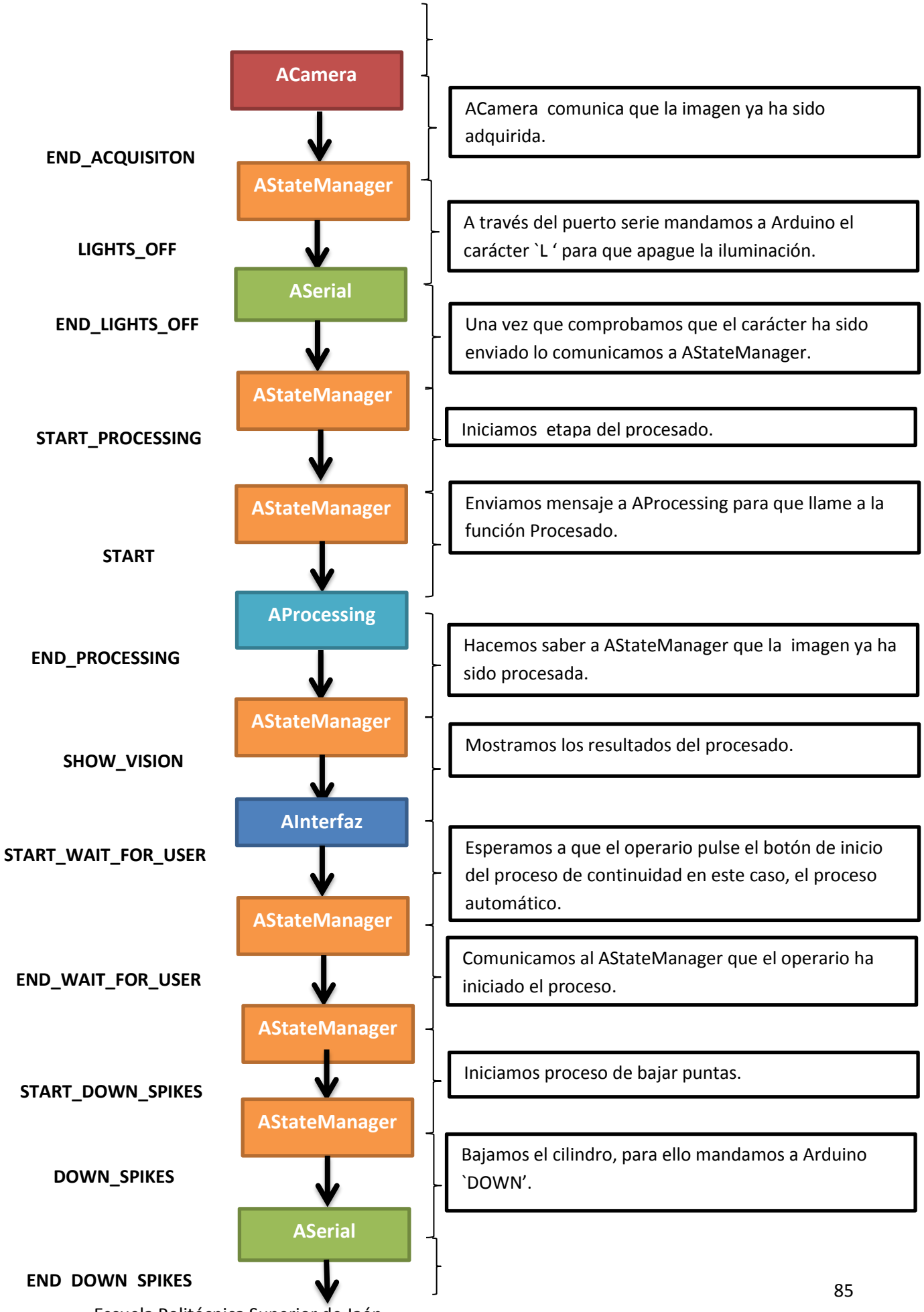

Ilustración 13. Agente AManualStateManager.

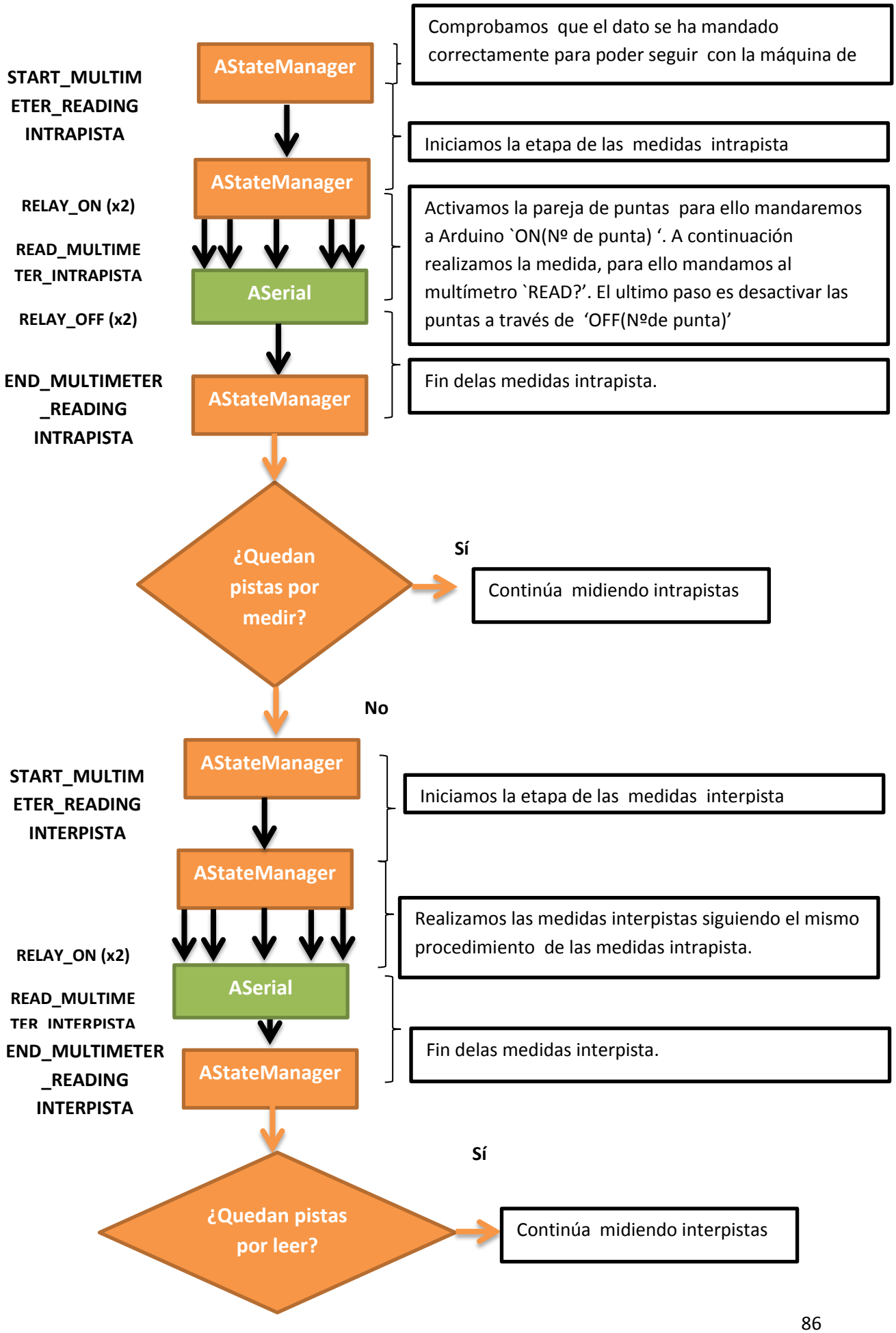
Una vez que hemos visto los agentes pasaremos a ver los esquemas de las máquinas de estados. Vamos a distinguir dos procesos o máquinas de estados en función de cómo vamos a realizar el test de continuidad:

- Automático: En este proceso como su nombre indica el test de continuidad se hace de forma automática. Se hacen todos los análisis 'interpista' e 'intrapista' necesarios para analizar completamente la PCB. Este proceso está controlado por el agente 'AStateManager'.
- Manual: El operario desea comprobar la continuidad de dos puntos, para lo cual debe seleccionar dichos puntos en la interfaz. Está controlada por el agente 'AManualStateManager'.

**MÁQUINA DE ESTADOS AUTOMÁTICA (AStateManager)**




ACamera
ACamera comunica que la imagen ya ha sido adquirida.
END_ACQUISITON
AStateManager
A través del puerto serie mandamos a Arduino el carácter `L ‘ para que apague la iluminación.
LIGHTS_OFF
ASerial
END_LIGHTS_OFF
Una vez que comprobamos que el carácter ha sido enviado lo comunicamos a AStateManager.
AStateManager
START_PROCESSING
Iniciamos etapa del procesado.
AStateManager
Enviamos mensaje a AProcessing para que llame a la función Procesado.
START
AProcessing
END_PROCESSING
Hacemos saber a AStateManager que la imagen ya ha sido procesada.
AStateManager
SHOW_VISION
Mostramos los resultados del procesado.
AInterfaz
START_WAIT_FOR_USER
Esperamos a que el operario pulse el botón de inicio del proceso de continuidad en este caso, el proceso automático.
AStateManager
END_WAIT_FOR_USER
Comunicamos al AStateManager que el operario ha iniciado el proceso.
AStateManager
START_DOWN_SPIKES
Iniciamos proceso de bajar puntas.
AStateManager
DOWN_SPIKES
Bajamos el cilindro, para ello mandamos a Arduino `DOWN’.
ASerial
END DOWN SPIKES


Comprobamos que el dato se ha mandado correctamente para poder seguir con la máquina de
START_MULTIMETER_READING INTRAPISTA
AStateManager
Iniciamos la etapa de las medidas intrapista
AStateManager
RELAY_ON (x2)
READ_MULTIMETER_INTRAPISTA
AStateManager
Activamos la pareja de puntas para ello mandaremos a Arduino `ON(Nº de punta) '. A continuación realizamos la medida, para ello mandamos al multímetro `READ?'. El ultimo paso es desactivar las puntas a través de 'OFF(Nºde punta)'
RELAY_OFF (x2)
END_MULTIMETER_READING INTRAPISTA
AStateManager
Fin delas medidas intrapista.
¿Quedan pistas por medir?
Sí
Continúa midiendo intrapistas
No
START_MULTIMETER_READING INTERPISTA
AStateManager
Iniciamos la etapa de las medidas interpista
AStateManager
Realizamos las medidas interpistas siguiendo el mismo procedimiento de las medidas intrapista.
RELAY_ON (x2)
READ_MULTIMETER_INTERPISTA
AStateManager
Fin delas medidas interpista.
END_MULTIMETER_READING INTERPISTA
AStateManager
Sí
¿Quedan pistas por leer?
Continúa midiendo interpistas

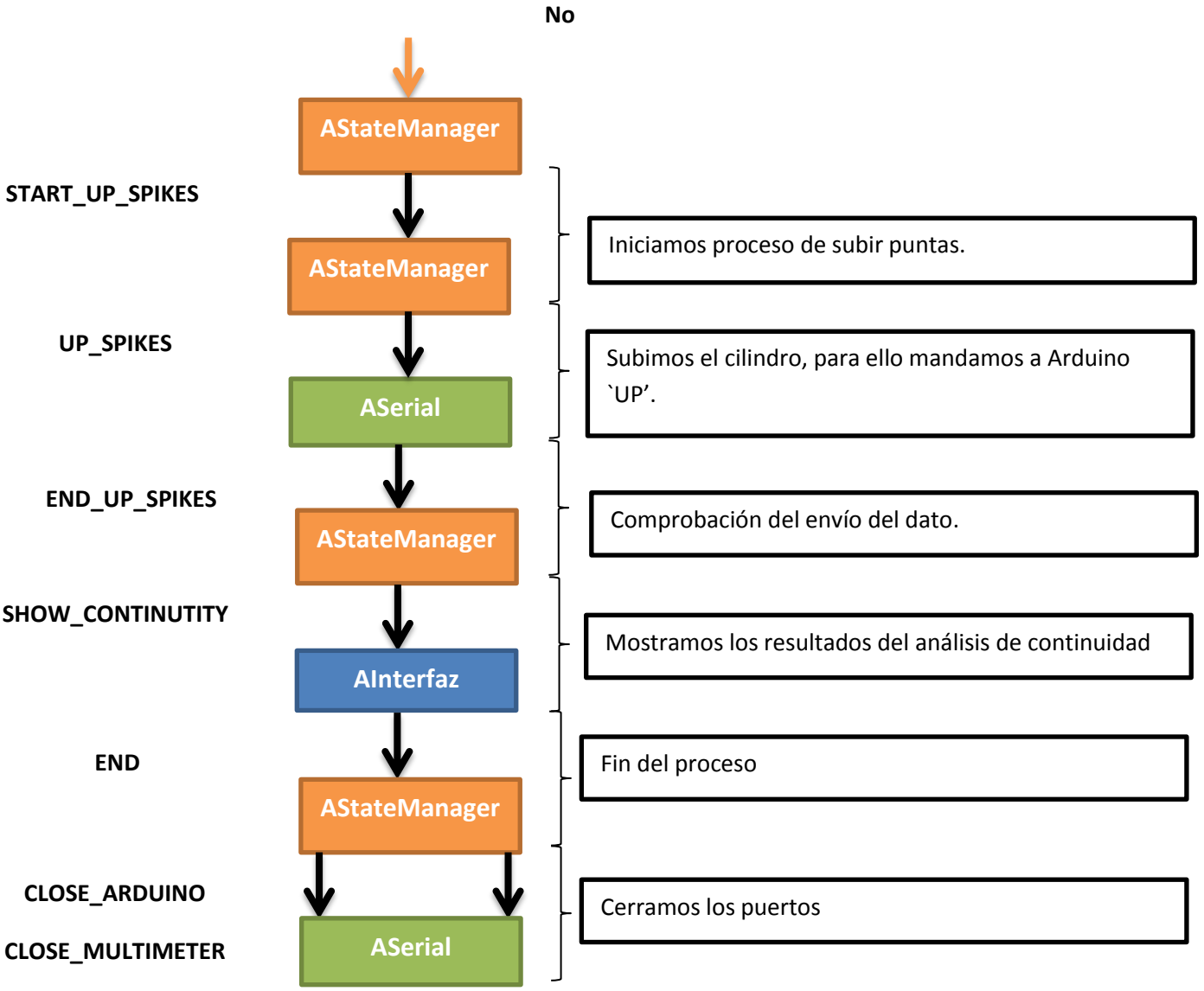
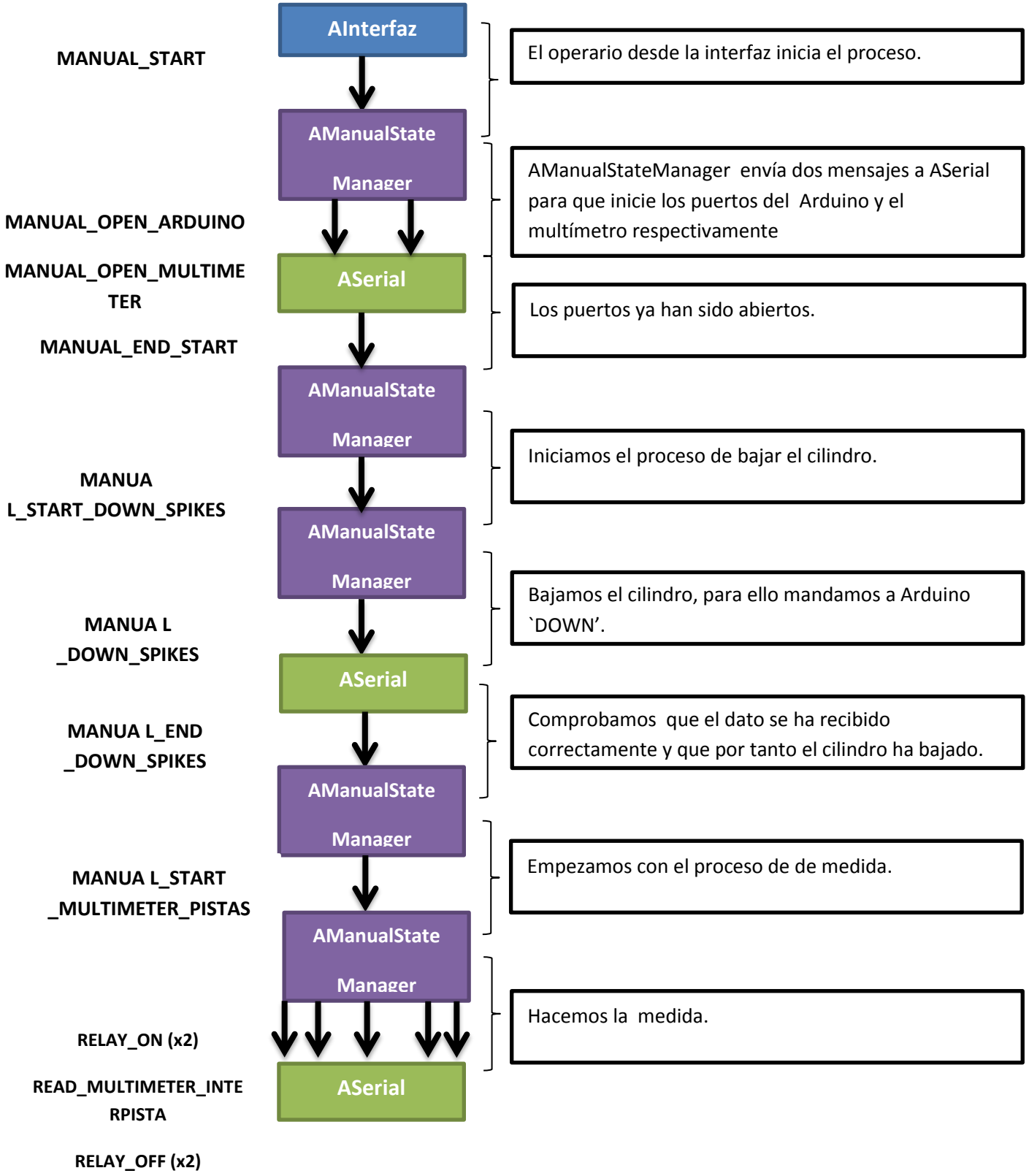



Ilustración 14. Máquina de estados de proceso automático AStateManager.

## MÁQUINA DE ESTADOS MANUAL (AManualStateManager)



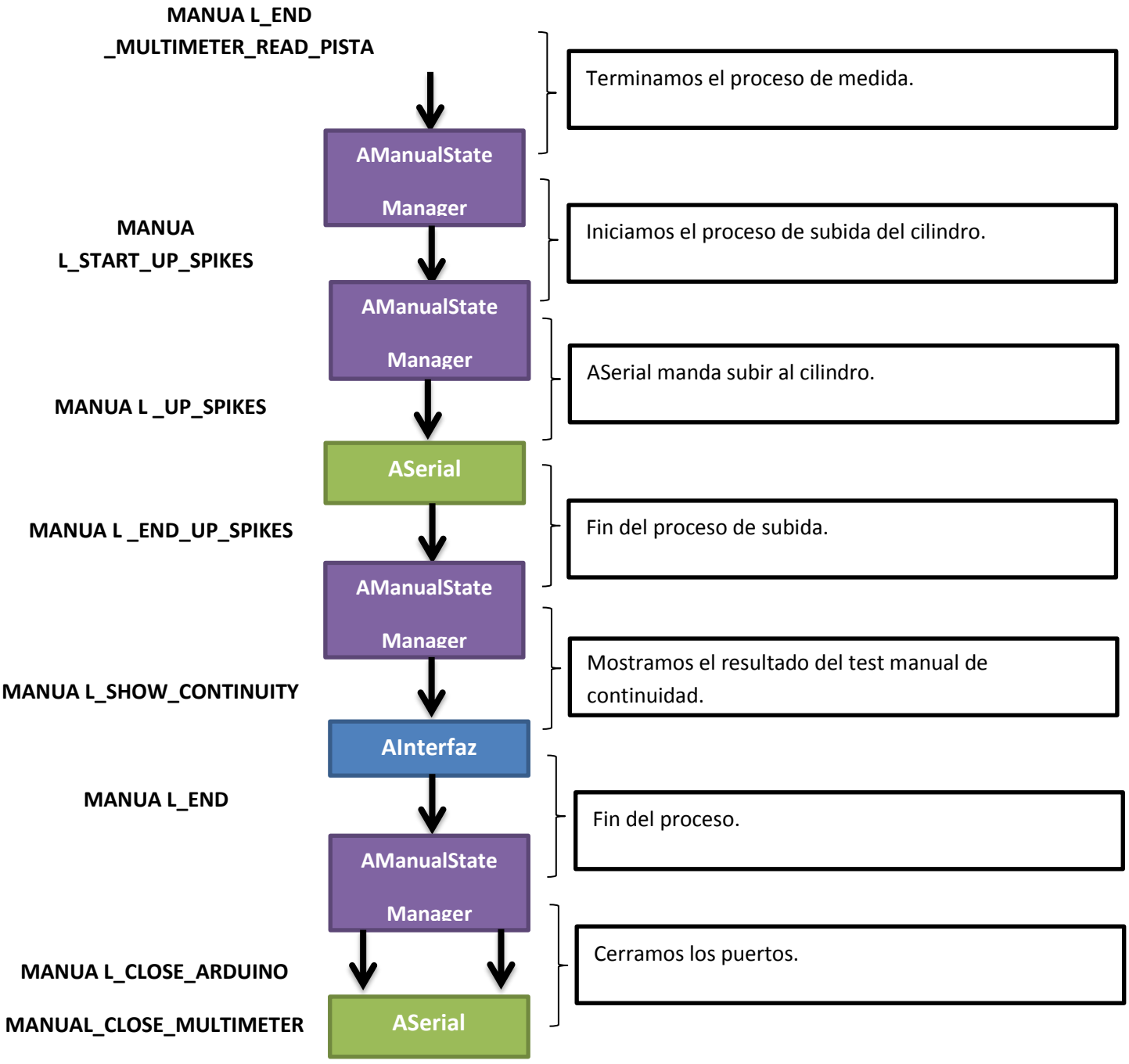



Ilustración 15. Máquina de estados de proceso manual AManualStateManager.

## ***Interfaz***

Una de los objetivos de este Trabajo de Fin de Grado era crear una interfaz gráfica donde se mostraran los resultados de ambos análisis. En un principio se pensó en hacer dos interfaces o pantallas, una para cada proceso. Sin embargo se optó finalmente por agrupar los datos en una única interfaz.

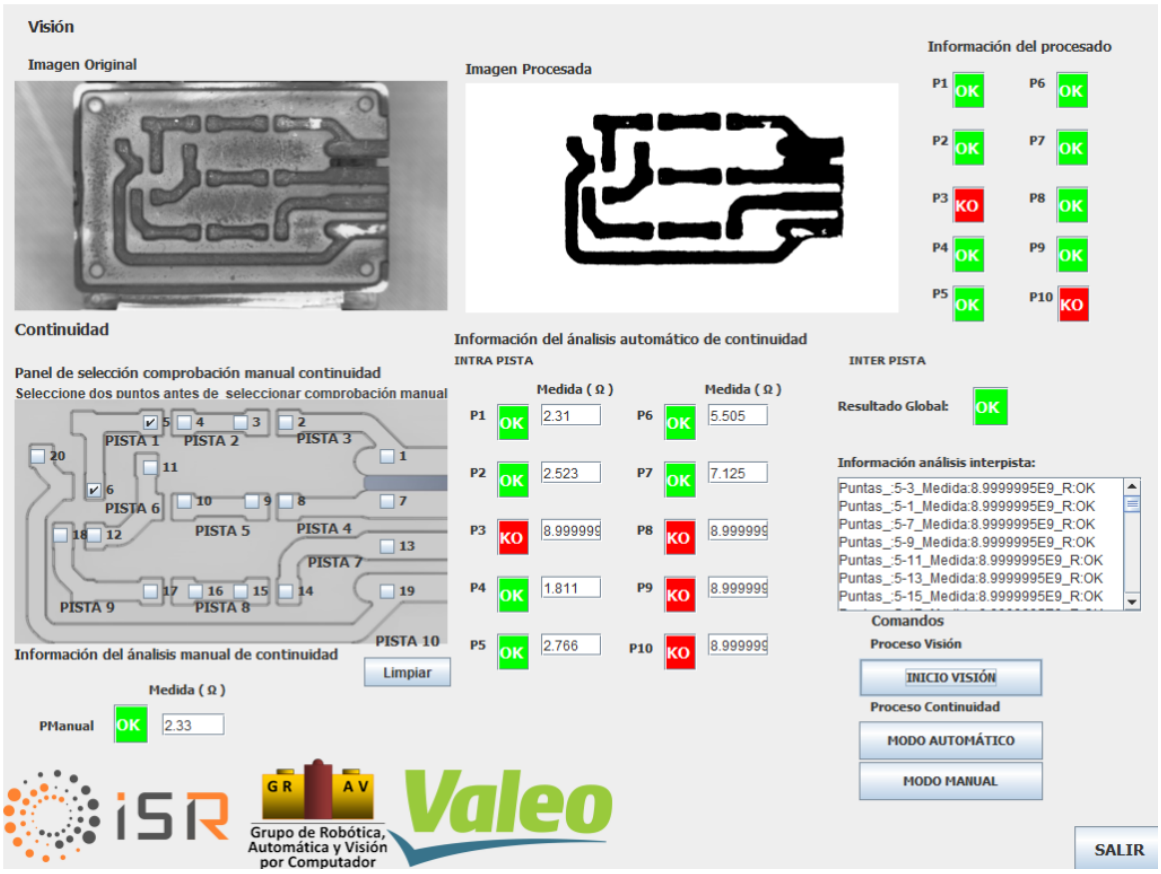



Figura 2.43. Interfaz de usuario.

En la imagen anterior podemos ver una captura de nuestra interfaz. Se distinguen tres partes principales:

- Visión: Esta corresponde a la parte superior de la interfaz. Consta de dos ‘jPanel’, uno para la imagen original y otro para la imagen procesada. Cada ‘jPanel’ lleva asociado un ‘jLabel’ con el título correspondiente. Además se puede ver que en la esquina superior derecha hay un ‘jPanel’ donde se muestran el resultado del procesado.

Este a su vez está formado por 10 ‘jTextField’, uno para cada pista donde se muestra un ‘OK’ si la pista no presenta ningún defecto visual y por el contrario si sí lo tiene muestra un ‘KO’.

- Continuidad: Se sitúa en la parte inferior de la interfaz. A la derecha se encuentra el espacio donde se imprimen los resultados del análisis eléctrico. Dentro de esta zona, se encuentra un ‘jPanel’ donde se muestran 10 ’jTextField’ que corresponden cada uno a la medida de resistencia de una pista individual (intrapista). Cada medida va a asociado a otro ’jTextField’ que indica si la pista es defectuosa (‘KO’) o no tiene defecto de continuidad (‘OK’) y a un ‘jLabel’ con el nombre de la pista. En la parte situada más a la derecha se ha colocado un ‘jTextArea’ al cual le hemos asociado ‘jScrollPane’ para poder desplazar el área de texto verticalmente. En este ‘jTextArea’ mostraremos los resultados de la comprobación de continuidad entre pistas distintas (interpista). Encima del área de texto mostramos un ‘OK’ si no habido ningún fallo de continuidad o de lo contrario mostrar un ‘KO’. Cabe destacar que el fallo de continuidad en el análisis interpista no corresponde a un fallo en el análisis intrapista. Ya que en este último comprabamos que las pistas conduzcan, es decir que exista continuidad y por otro lado en el interpisa verificamos que las pistas estén aisladas eléctricamente, en otras palabras, que no exista continuidad. En la parte que se encuentra más a la izquierda, se localiza un panel interactivo que está formado por 20 ‘jCheckBox’, que corresponden a cada una de las puntas con las que hacemos el test eléctrico, y por un ‘jPanel’ que muestra la PCB. Este panel le permite al operario hacer una medida manual. Para ello deberá seleccionar los dos puntos entre los cuales se realizará la comprobación. Debajo de este panel mostramos el valor de la medida junto a un campo de texto donde se mostrará si el resultado de la medida ha sido favorable teniendo en cuenta el tipo de análisis, es decir, si es medida intrapista o interpista. En la esquina inferior derecha del panel manual encontramos el botón de limpiar que se usa para borrar los datos almacenados del análisis manual anterior.

- Comandos: Encontramos tres ‘jButton’ distintos. Uno es para el inicio del proceso de visión; otro para el modo automático de continuidad y el último para el modo manual de continuidad.

El primer paso para obtener nuestra interfaz fue crear la clase Interface la cual está asociada al agente AInterfaz explicado anteriormente.

**2.2.2.2.-Programa de Arduino**.

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso.

Para poder entender este concepto, primero aclararemos los términos del hardware libre y el software libre. El hardware libre son los dispositivos cuyas especificaciones y diagramas son de acceso público, de manera que cualquiera puede replicarlos. Esto quiere decir que Arduino ofrece las bases para que cualquier otra persona o empresa pueda crear sus propias placas, pudiendo ser diferentes entre ellas pero igualmente funcionales al partir de la misma base. Por otro lado, software libre son los programas informáticos cuyo código es accesible por cualquiera para que quien quiera pueda utilizarlo y modificarlo. Para ello Arduino ofrece la plataforma Arduino IDE (Entorno de Desarrollo Integrado), que es un entorno de programación con el que cualquiera puede crear aplicaciones para las placas Arduino, de manera que se les puede dar todo tipo de utilidades.

Los diseños de las placas Arduino usan diversos microcontroladores y microprocesadores. Generalmente el hardware consiste de un microcontrolador Atmel AVR, conectado bajo la configuración de "sistema mínimo" sobre una placa de circuito impreso a la que se le pueden conectar placas de expansión

(shields) a través de la disposición de los puertos de entrada y salida presentes en la placa seleccionada. Las shields complementan la funcionalidad del modelo de placa empleada, agregando circuiteria, sensores y módulos de comunicación externos a la placa original. La mayoría de las placas Arduino pueden ser energizadas por un puerto USB o un puerto barrel Jack de 2.5mm.

El proyecto nació en 2003, cuando varios estudiantes del Instituto de Diseño Interactivo de Ivrea, Italia, con el fin de facilitar el acceso y uso de la electrónico y programación. Lo hicieron para que los estudiantes de electrónica tuviesen una alternativa más económica a las populares BASIC Stamp, unas placas que por aquel entonces valían más de cien dólares, y que no todos se podían permitir. El resultado fue Arduino, una placa con todos los elementos necesarios para conectar periféricos a las entradas y salidas de un microcontrolador, y que puede ser programada tanto en Windows como macOS y GNU/Linux. Un proyecto que promueve la filosofía 'learning by doing', que viene a querer decir que la mejor manera de aprender es cacharreando.

La primera placa Arduino comercial fue introducida en el año 2005, ofreciendo un bajo costo económico y facilidad de uso para novatos y profesionales. A partir de octubre del año 2012, se incorporaron nuevos modelos de placas de desarrollo que empleaban microcontroladores Cortex M3, ARM de 32 bits,3 dichos modelos coexisten con los iniciales, que integran microcontroladores AVR de 8 bits. Cabe resaltar que las arquitecturas ARM y AVR no son iguales, por lo cual tampoco lo es su set de instrucciones a nivel ensamblador y como consecuencia, algunas librerías realizadas para operar en una arquitectura presenten complicaciones al ser empleadas en la otra. A pesar de lo anterior, todas los modelos de placa Arduino se pueden programar y compilar bajo el IDE predeterminado de Arduino sin ningún cambio, esto gracias a que el IDE compila el código original a la versión de la placa seleccionada.

Figura 2.44. Logo de Arduino.

De la amplia variedad de placas de las que dispone Arduino seleccionamos Arduino Mega 2560. Esta es una placa que está especialmente diseñada para los proyectos que requieren circuitos complejos y más espacio de memoria. Está basada en el microcontrolador ATmega2560.Tiene 54 entradas/salidas digitales (de las cuales 15 pueden ser usadas como salidas PWM), 16 entradas analógicas, 4 UARTs que producen una velocidad máxima para configurar la comunicación, un cristal de 16Mhz, conexión USB, jack para alimentación DC, conector ICSP, y un botón de reseteo. La placa Mega 2560 es compatible con la mayoría de shields compatibles para Arduino UNO. Esta placa viene con dos reguladores de voltaje, es decir, 5V y 3.3V, que proporciona la flexibilidad para regular el voltaje según los requisitos. Los pines de esta placa siguen la siguiente disposición:

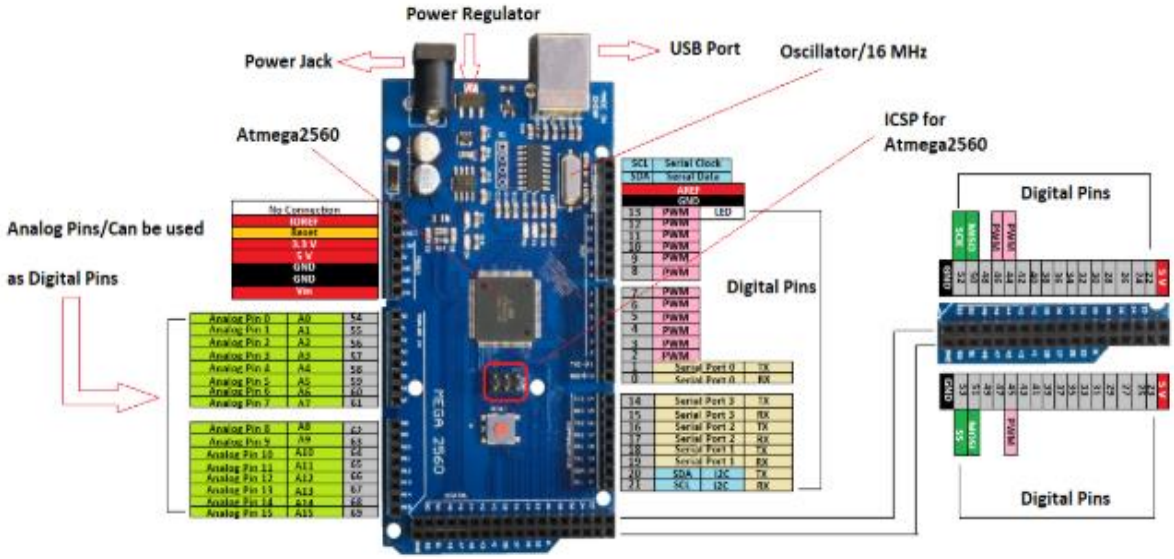



Figura 2.45. Pin-out de Arduino Mega 2560.

Una vez que hemos explicado brevemente que es Arduino y que placa vamos a usar, pasaremos a explicar el sketch (nombre que reciben los programas en Arduino). El código se divide en cuatro partes:

- Declaración de variables: Esta sería la siguiente donde: ‘x0 ‘ es la etapa inicial en la que el cilindro está en la posición de reposo; ‘ x1‘es la etapa donde el cilindro ha bajado hasta llegar al final de carrera; ‘stringComplete’ es el flag que usamos para saber que hemos recibido el string; ‘c0’ es la medida analógica que da el final de carrera que está en la posición de reposo del cilindro; ‘c1’ es el valor analógico que da el final de carrera de cuando el cilindro está extendido y ‘v’ es la variable que almacena el ‘String’ que manda el programa de Java a Arduino.

```
bool x0 = 0,x1 = 0,stringComplete = false;
int c0,c1;
String v = "";
```

Figura 2.46. Declaración de variables.

- void setup(): Esta función se ejecuta cuando alimentamos nuestra placa Arduino o cuando pulsamos el botón de reinicio. Aquí se escriben los criterios que solo necesitan ejecutarse una única vez. En nuestro caso establecemos el comando Serial.begin para indicarle al programa que vamos a iniciar la comunicación serie a 9600 baudios. Con la función ‘reserve()’ asignamos un espacio en la memoria para manipular cadenas. También declaramos los pines digitales que vamos a usar como salidas para ello usaremos el parámetro ‘OUTPUT’ y le indicaremos el número del pin. Si usáramos estos pines como entradas no sería necesario indicarlo ya que vienen configuradas de este modo

por defecto. Necesitaremos 23 salidas digitales: 20 para activar las puntas; 1 para la iluminación y 2 para la válvula. A diferencia de los pines digitales, los pines analógicos no necesitan ser declarados como INPUT u OUTPUT ya que siempre son de tipo INPUT.

```
void setup()
{
  //Configuración Puerto Serie
  Serial.begin(9600);

  v.reserve(200);
  //Configuración I/O Relé Iluminación
  pinMode(31, OUTPUT);

  //Configuración I/O Relés Válvula
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);
  //Configuración I/O Relés Puntas

  pinMode(34, OUTPUT);
  pinMode(35, OUTPUT);
  pinMode(36, OUTPUT);
  pinMode(37, OUTPUT);
  pinMode(38, OUTPUT);
  pinMode(39, OUTPUT);
  pinMode(40, OUTPUT);
  pinMode(41, OUTPUT);
  pinMode(42, OUTPUT);
  pinMode(43, OUTPUT);

pinMode(44, OUTPUT);
pinMode(45, OUTPUT);
pinMode(46, OUTPUT);
pinMode(47, OUTPUT);
pinMode(48, OUTPUT);
pinMode(49, OUTPUT);
pinMode(50, OUTPUT);
pinMode(51, OUTPUT);
pinMode(52, OUTPUT);
pinMode(53, OUTPUT);
```

**Figura 2.47. Declaración de pines y de velocidad.**

Dentro de esta función también debemos asignar el valor inicial de cada pin digital. En nuestro caso será 'HIGH' el valor predeterminado porque los relés se activan a nivel bajo.

```
digitalWrite(31,HIGH);
digitalWrite(32,HIGH);
digitalWrite(33,HIGH);
digitalWrite(34,HIGH);
digitalWrite(35,HIGH);
digitalWrite(36,HIGH);
digitalWrite(37,HIGH);
digitalWrite(38,HIGH);
digitalWrite(39,HIGH);
digitalWrite(40,HIGH);
digitalWrite(41,HIGH);
digitalWrite(42,HIGH);
digitalWrite(43,HIGH);
digitalWrite(44,HIGH);
digitalWrite(45,HIGH);
digitalWrite(46,HIGH);
digitalWrite(47,HIGH);
digitalWrite(48,HIGH);
digitalWrite(49,HIGH);
digitalWrite(50,HIGH);
digitalWrite(51,HIGH);
digitalWrite(52,HIGH);
digitalWrite(53,HIGH);
```

Figura 2.48. Inicialización de los pines.

- void loop(): Loop en inglés significa lazo o bucle. La función loop en Arduino es la que se ejecuta un número infinito de veces. Este bucle albergará en su interior el comportamiento del Arduino. Debe declararse después del setup. Al encenderse el Arduino se ejecuta el código del setup y luego se entra al loop, el cual se repite de forma indefinida hasta que se apague o se reinicie el microcontrolador. A continuación pasaremos a analizar nuestra función del loop.

  En primer lugar leemos los valores analógicos de los finales de carrera (A0 para el cilindro retraído y A1 para el cilindro expandido). Una vez que sabemos la posición del cilindro, comprobamos si hemos recibido algún dato, para ello utilizamos un flag,'stringComplete', que se activa cuando hay datos en el buffer, aunque esto lo explicaremos de manera más detallada en la siguiente función. Si hemos recibido algún dato (en nuestro caso todos serán de tipo 'String') pasamos a analizarlo. Si hemos recibido la cadena de caracteres 'UP' y el cilindro no está retraído, en otras palabras el flag de 'x0' no está activado, subimos el cilindro (dejando a nivel alto el pin 32 y a nivel bajo el 33), imprimimos por pantalla 'ok' para indicar que hemos recibido el 'String'

correctamente y en último lugar vaciamos el buffer con la función 'Serial.flush()'. Repetimos un proceso similar cuando recibimos 'DOWN' con la diferencia que bajamos el cilindro.

```
  c0=analogRead(0);   //Final de carrera cilindro retraído
  c1=analogRead(1);   //Final de carrera cilindro expandido
if(stringComplete){
  //Proceso de bajada de Cilindro
  if(v=="UP"&&!x0)
  {
    x1=1;
    v="";
    stringComplete = false;
    digitalWrite(32,HIGH);
    digitalWrite(33,LOW);
    Serial.println("ok");
    Serial.flush();
  }
  //Proceso de subida de Cilindro
  if(v=="DOWN" && !x1)
  {
    x0=1;
    v="";
    digitalWrite(32,LOW);
    digitalWrite(33,HIGH);
    stringComplete = false;
    Serial.println("ok");
    Serial.flush();
  }
```

Figura 2.49. Principio del procedimiento del loop.

A continuación seguimos el mismo procedimiento para analizar los caracteres 'I' el cual enciende la iluminación y 'L', que la apaga.

```
else if(v=="I"){
  digitalWrite(31,LOW);
  v="";
  stringComplete = false;
  Serial.println("ok");
  Serial.flush();
  }

else if(v=="L"){
  digitalWrite(31,HIGH);
  v="";
  stringComplete = false;
  Serial.println("ok");
  Serial.flush();
  }
```

Figura 2.50. Encendido y apagado de la iluminación.

Para activar y desactivar cada una de las puntas seguimos el método análogo.

```
else if(v=="ON1" ){
  digitalWrite(53,LOW);
  v="";
  stringComplete = false;
  Serial.println("ok");
  Serial.flush();
  }
else if(v=="OFF1" ){
  digitalWrite(53,HIGH);
  v="";
  stringComplete = false;
  Serial.println("ok");
   Serial.flush();
}
      :
      :
      :
else if(v=="ON20"){
  digitalWrite(34,LOW);
  v="";
  stringComplete = false;
  Serial.println("ok");
  Serial.flush();
}
else if(v=="OFF20"){
  digitalWrite(34,HIGH);
  v="";
  stringComplete = false;
  Serial.println("ok");
  Serial.flush();
}
```

Figura 2.51. Activación y desactivación de las puntas.

Si la cadena que ha recibido no es ninguna de la que hemos analizado anteriormente, vaciamos la variable ‘v’ y desactivamos el ‘flag ‘ que nos informa sobre la llegada de ‘String’.

```
  else{
    v="";
    stringComplete = false;
    }
}
```

Figura 2.52. Procedimiento en el caso de que la cadena sea incorrecta.

Por último para acabar con la función del ‘loop’, analizamos los valores analógicos de los finales de carrera, si y el final de carrera del cilindro extendido ’c1’ nos da un valor mayor que 900 esto nos indica que el cilindro está extendido y que podemos desactivar ‘x0’, ‘flag’ que nos indica que estamos en la etapa inicial donde el cilindro retraído. Hacemos la comprobación análoga para la retracción del cilindro.

```
if(x0&&c1>900)
{


  digitalWrite(32,HIGH);
  digitalWrite(33,HIGH);


  x0=0;
  stringComplete = false;

}

if(x1&&c0>900)
{

  digitalWrite(32,HIGH);
  digitalWrite(33,HIGH);

  x1=0;
  stringComplete = false;
}

}
```

Figura 2.53. Comprobación posición del cilindro.

- void serialEvent(): Esta rutina ocurre cada vez que un nuevo dato viene en el RX serial del hardware. Se ejecuta entre cada tiempo que se ejecuta el 'loop'. Consiste en ir leyendo los caracteres y formar un 'String' hasta que el carácter sea un salto de línea, momento que el 'String' estará completo.

```
void serialEvent() {
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // add it to the inputString:
    v += inChar;
    // if the incoming character is a newline, set a flag so the main loop can
    // do something about it:
    if (inChar == '\n') {
      stringComplete = true;
      v = v.substring(0,v.length()-2);
    }
  }
}
```

Figura 2.54. Procedimiento serialEvent().

Para una mayor compresión y tener una visión más global del programa mostraremos el siguiente organigrama.

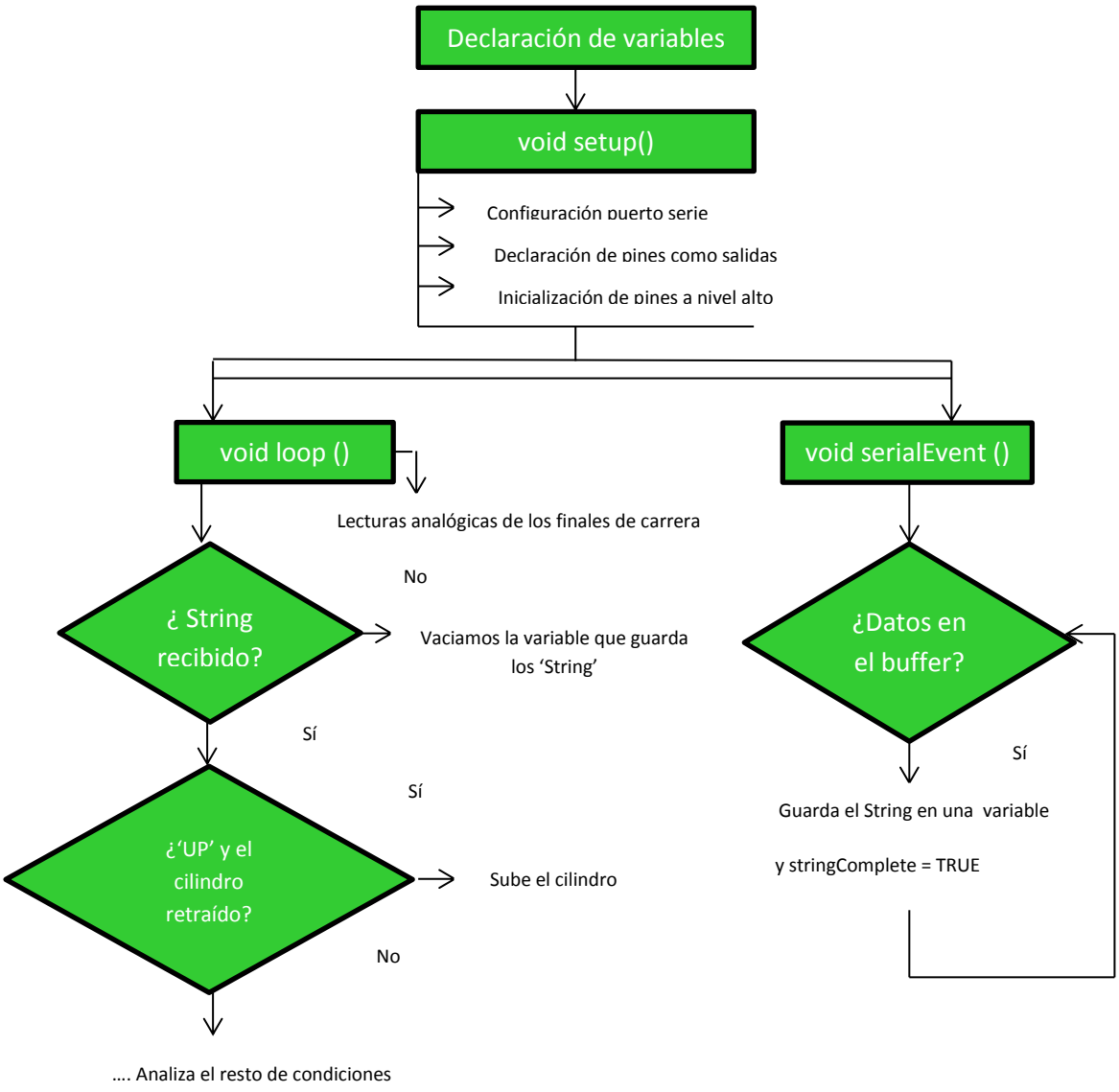



**Ilustración 16. Esquema de Arduino**

En último lugar veremos el esquema del conexionado de nuestra placa de 'Arduino' con las placas de cuatro relés. En el esquema podemos ver etiquetados los cables. Los cables que vienen etiquetados con un número corresponden con el número de la punta que previamente hemos asociado

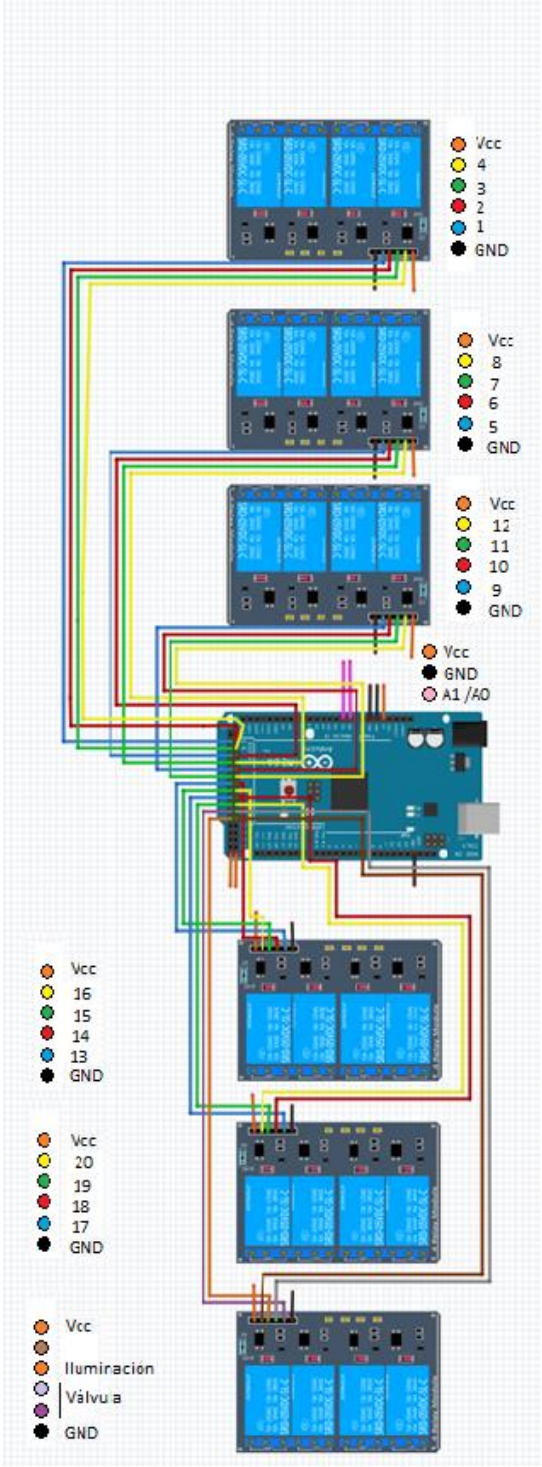



Figura 2.55. Conexionado entre Arduino y los relés.

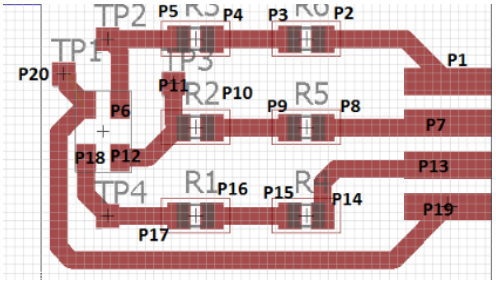



Figura 2.56. Distribución de las puntas.

## 3.-RESULTADOS.

En este capítulo de la memoria pondremos a prueba el prototipo final que hemos obtenido. Con esta finalidad lo subdividiremos en dos apartados:

- Pruebas realizadas. Para ello realizaremos pruebas a ambas etapas, Visión y Continuidad.
- Prototipo final: Veremos el resultado del proceso de construcción a través de imágenes de la máquina.

### 3.1.-Pruebas realizadas.

En primer lugar pondremos a prueba la etapa de visión. Para esta fase del proyecto debemos variar las condiciones del procesado. A la hora de realizar las comprobaciones después de procesar la imagen, poseemos un valor de umbral que influye directamente en la toma de decisión el área del defecto. Con este umbral fijamos a partir de que dimensión del defecto consideraremos que se trata de un defecto crítico y que por tanto la pista es defectuosa. Otro factor con el que podemos jugar y que influye a la hora de dictaminar si una PCB es defectuosa es el umbral de binarización. Este valor influye en el resultado del procesado de la imagen, y como consecuencia en el resultado del análisis. A continuación mostraremos distintas imágenes donde se visualizan distintas umbralizaciones.

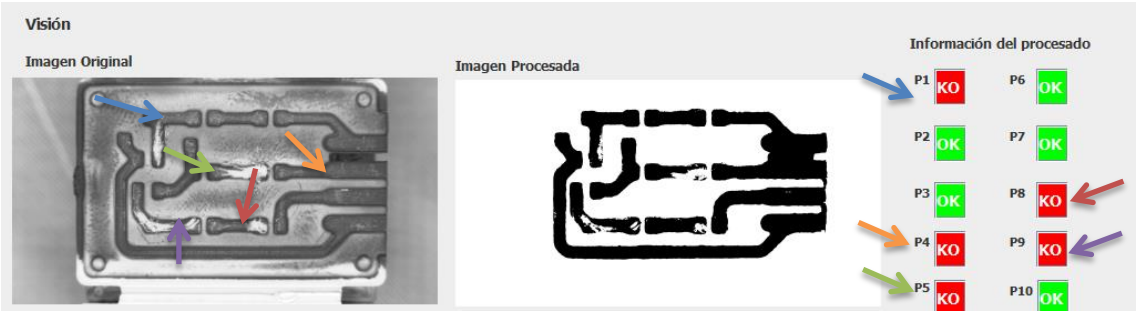



Figura 3.1. Imagen binarizada con umbral 215.

En la Figura 3.1 podemos ver como detectamos un falso defecto en la pista 4, que aparece señalada con una flecha en color naranja. Para intentar resolver el problema cambiamos el valor del umbral.

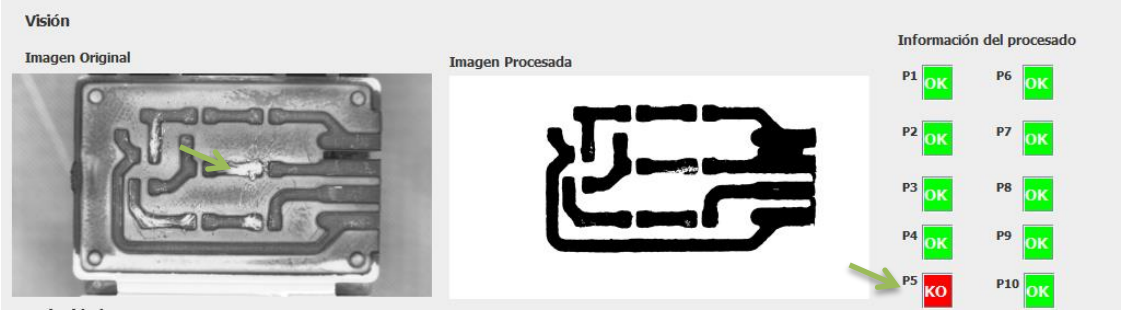



Figura 3.2. Imagen binarizada con umbral 235.

Al subir el umbral eliminamos el error que cometía en la binarización pero cometemos otros, ya que tenemos falsos positivos en las pistas 1,7 y 8. Como consecuencia dedujimos que el umbral óptimo estaría entre el rango de 215-235.

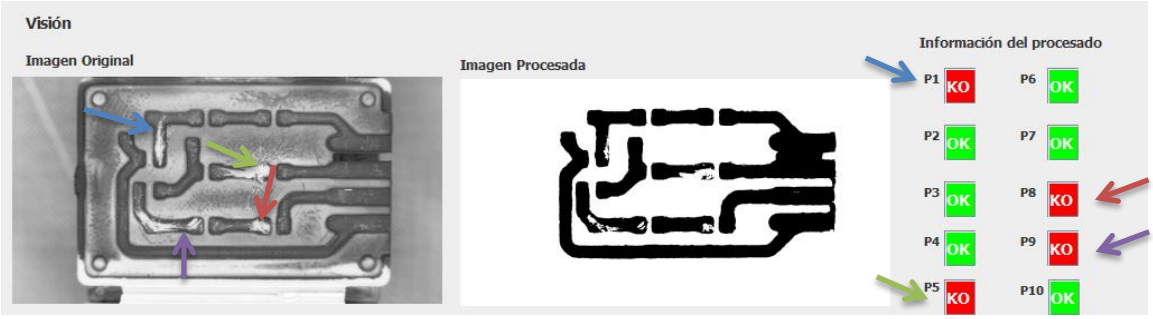



Figura 3.3. Imagen binarizada con umbral 225.

Al hacer pruebas con un valor del rango establecido anteriormente, en concreto con la mediana (225), obtuvimos un resultado positivo suprimiendo los errores que se cometían anteriormente. Para validar la conclusión a la habíamos llegado debíamos probar este umbral con más PCBs.

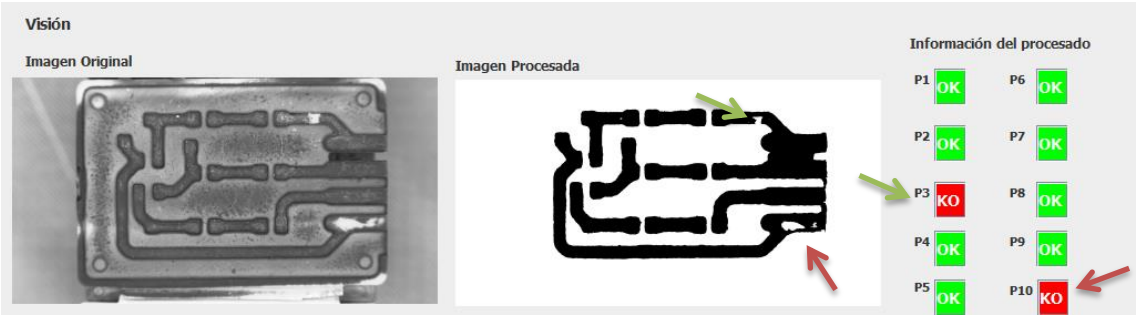



Figura 3.4. Prueba 1 con el umbral elegido.

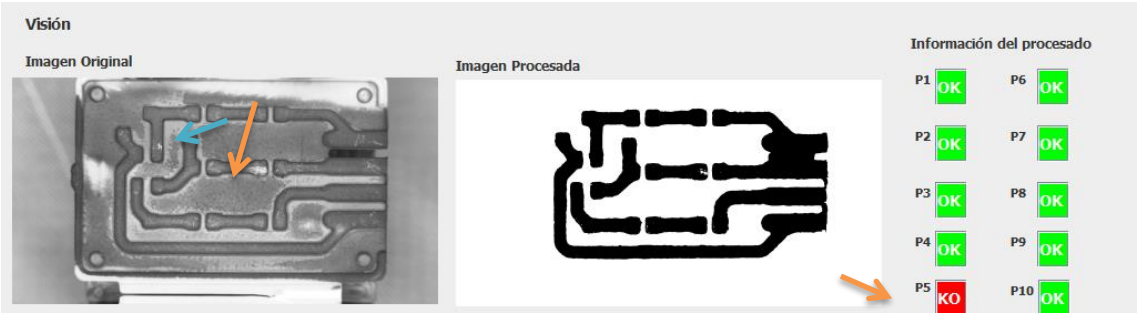



Figura 3.5. Prueba 2 con el umbral elegido.

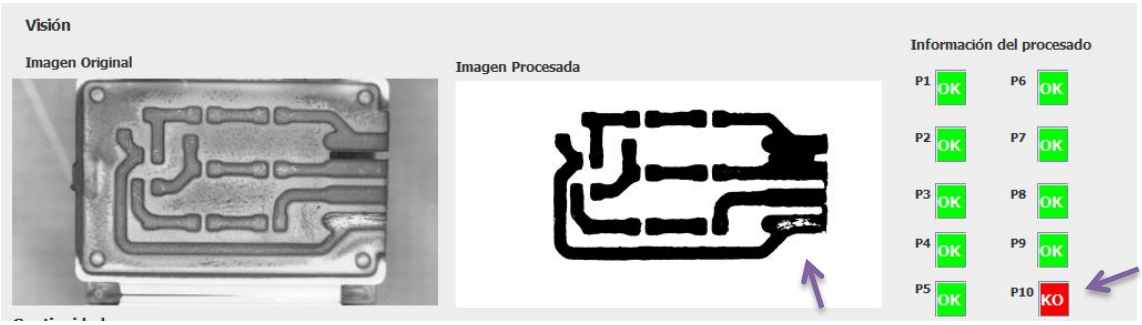



Figura 3.6. Prueba 3 con el umbral elegido.

Como vemos los resultados son óptimos, identificamos las pistas que son defectuosas. Podemos observar en la imagen procesada que el área de defecto sufre una reducción. Este hecho es algo que nos podría perjudicar si nuestro objetivo fuera contabilizar el área de defecto, sin embargo este no es el caso. Además nosotros solo buscamos identificar si una pista es defectuosa o no. En consecuencia nuestro objetivo es identificar defectos visuales graves, siendo estos de una dimensión similar a la anchura de la pista. Queremos identificar estos defectos ya que son estos los que influyen en el correcto

funcionamiento de la PCB. En otras palabras si el defecto de la PCB no es suficientemente grande teniendo como referencia el área de la pista, la PCB puede funcionar correctamente ya que tendría área suficiente para conducir. Este hecho lo podemos observar en la PCB de la figura 3.5. La PCB mencionada posee un defecto visual grave (que ocupa el gran parte del ancho de la pista) en la pista 5 y un defecto visual leve (que tiene una dimensión mucho menor que el ancho de la pista) en la pista 1. Aunque a pesar de que no detectamos el defecto de la pista 1, esto no nos influye ya que como podemos comprobar en el test eléctrico de la figura 3.7, la pista 1 tiene continuidad a diferencia de la pista 5. En consecuencia, nuestro sistema es válido, ya que detectamos los defectos visuales que influyen directamente en el correcto funcionamiento de la PCB.

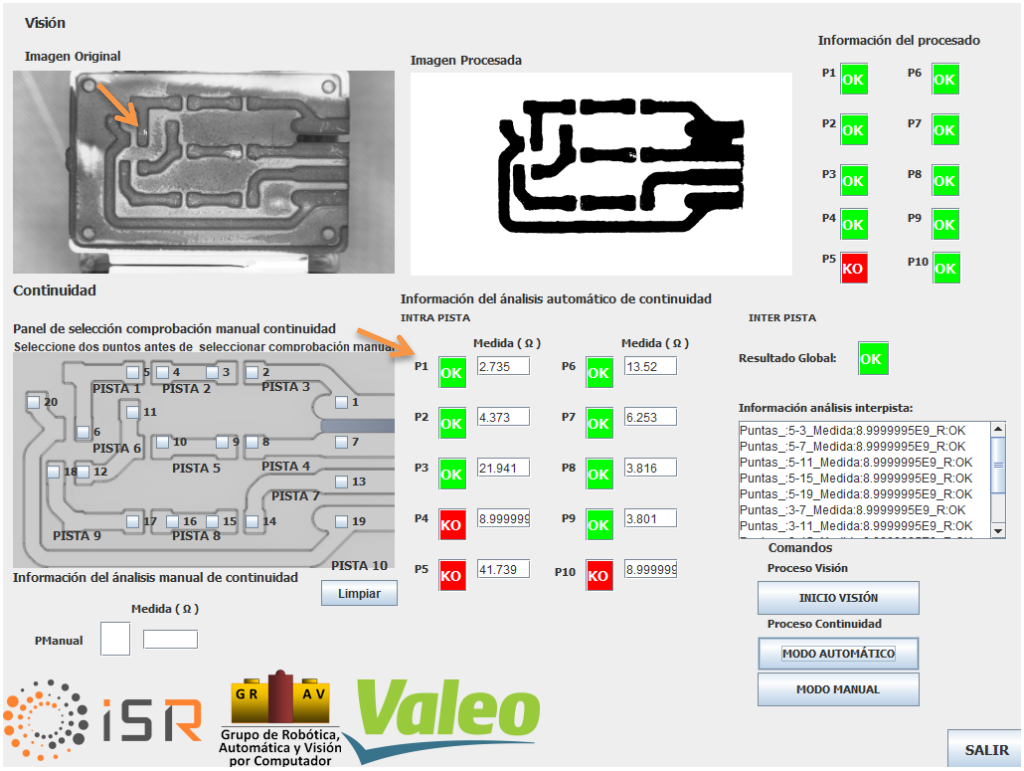



Figura 3.7. Comprobación de la influencia de los defectos visuales leves en continuidad.

Por los motivos mencionados en el párrafo anterior podemos llegar a la conclusión que se ha conseguido el objetivo que se perseguía. Ya analizada la

etapa de visión pasaremos a poner a prueba la fase de continuidad. En primer lugar tuvimos que ajustar las puntas, hecho que no fue sencillo de realizar, ya que la dimensión de las pistas es bastante pequeña y por cuestión de milímetros, la punta podría caer en una posición inadecuada, invalidando el test de continuidad. Tras pensar varios métodos, al final decidimos usar plastilina, esta metodología puede resultar simple en un principio, pero con ella conseguimos nuestro objetivo, ajustar las puntas.

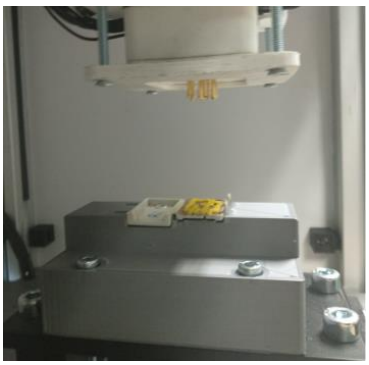


Figura 3.8. Ajuste de las puntas.

Para hacer las pruebas, llevábamos a cabo un estudio previo del umbral de continuidad con el que íbamos hacer las comprobaciones. Tras realizar una gran cantidad de medidas decidimos que usaríamos como umbral un valor de 40 Ω. Analizaremos 11 PCBs distintas para validar el umbral que hemos fijado previamente. Cabe destacar que para comprobar que la medida la estaba realizando correctamente, hicimos la medida con otro multímetro de forma manual para comparar los resultados de las medidas. Las medidas no son exactamente iguales ya que al medir con el óhmetro de nuestro prototipo introducimos los valores de resistencia de los cables de las puntas. A pesar de que no se tratan de los mismos valores de resistencia, si coinciden en los valores de continuidad, ya que estos están en un orden superior de resistencia (normalmente en torno a los MΩ). Sin embargo la resistencia que introducen los cables en considerablemente inferior (con frecuencia en un rango de 2-5 Ω). De ahí que comparar ambas medidas sea una prueba bastante fiable.

## *PCB 1*

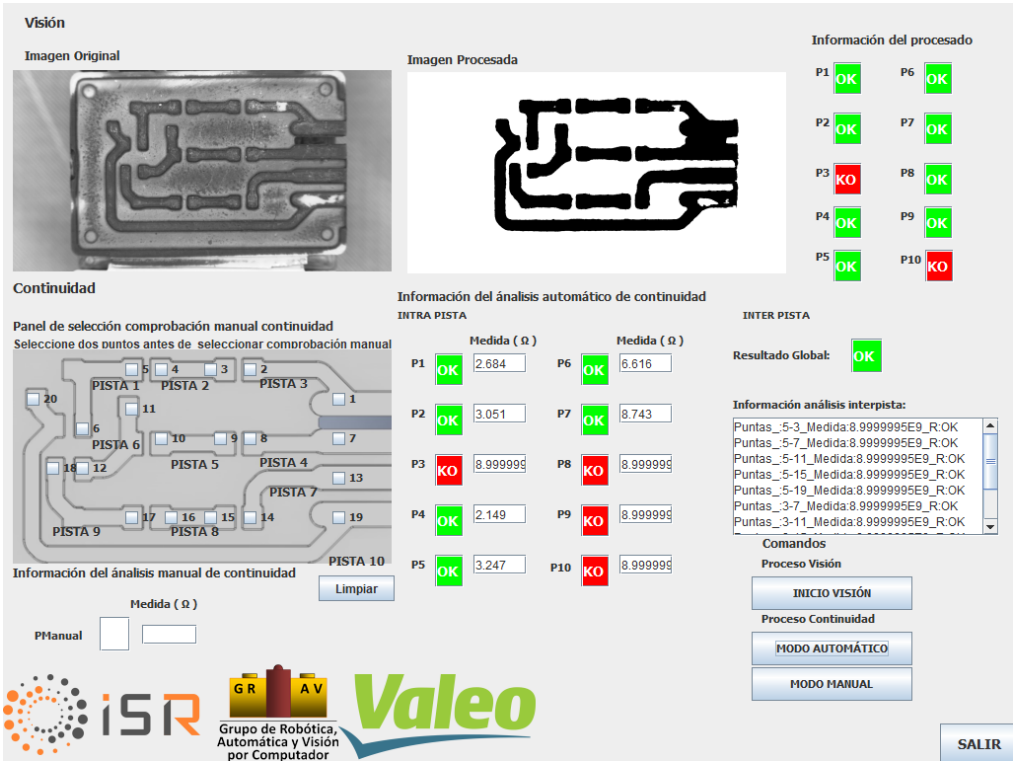



Figura 3.9. PCB 1.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.648 | OK | 0.9 | OK | |
| **2** | 3.051 | OK | 0.2 | OK | |
| **3** | INF | KO | 72.2 | KO | |
| **4** | 2.149 | OK | 0.6 | OK | |
| **5** | 3.247 | OK | 0.5 | OK | |
| **6** | 6.616 | OK | 3.9 | OK | |
| **7** | 8.743 | OK | 5.5 | OK | |
| **8** | INF | KO | INF | KO | |
| **9** | INF | KO | INF | KO | |
| **10** | INF | KO | INF | KO | |

Tabla 3.2. Datos intrapista PCB 1.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | 1.048K | OK | |
| **5-7** | INF | OK | 74.68 K | OK | |
| **5-11** | INF | OK | 276.3 | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | 75.44 K | OK | |
| **3-11** | INF | OK | 903 | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | 2.82M | OK | |

Tabla 3.2. Datos interpista PCB 1.

## *PCB 2*

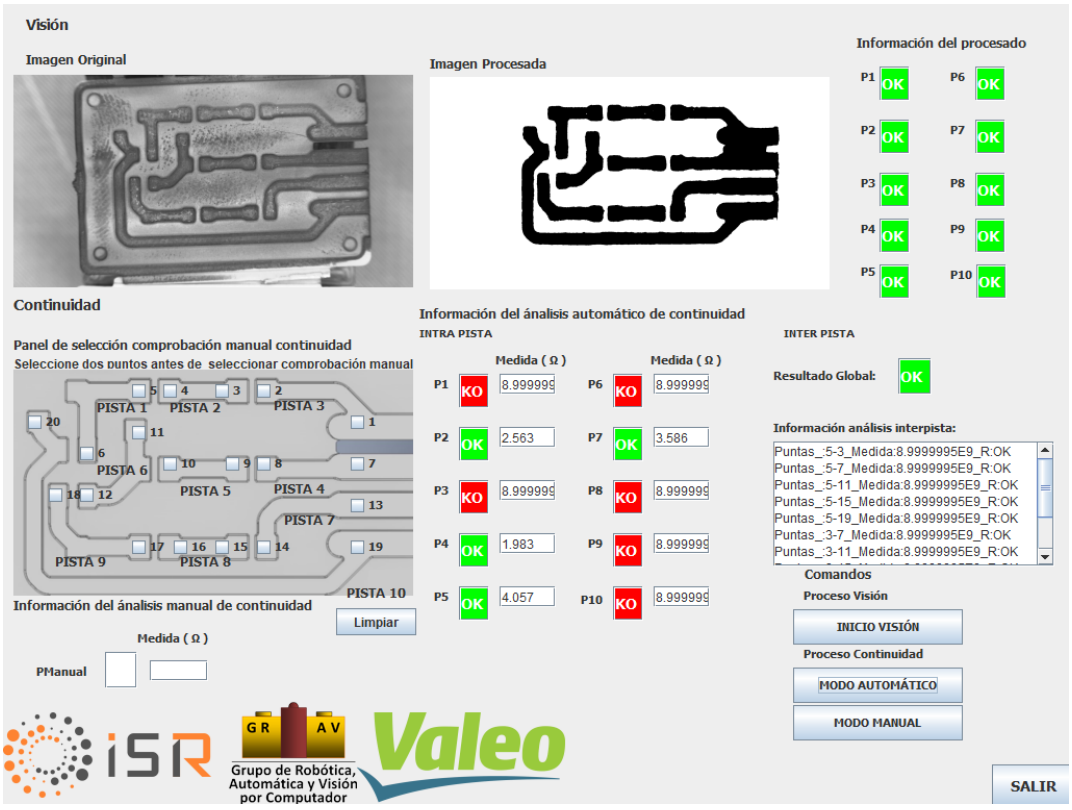



Figura 3.10. PCB 2.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | INF | KO | 11.38K | KO | |
| **2** | 2.563 | OK | 0.3 | OK | |
| **3** | INF | KO | 13.5K | KO | |
| **4** | 1.983 | OK | 0.1 | OK | |
| **5** | 4.057 | OK | 2.1 | OK | |
| **6** | INF | KO | INF | KO | |
| **7** | 3.586 | OK | 1.5 | OK | |
| **8** | INF | KO | 523K | KO | |
| **9** | INF | KO | 241 | KO | |
| **10** | INF | KO | INF | KO | |

Tabla 3.3. Datos intrapista PCB 2.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | 112.9K | OK | |
| **3-15** | INF | OK | 113 | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | 12.73K | OK | |

Tabla 3.4. Datos interpista PCB 2.

## *PCB 3*

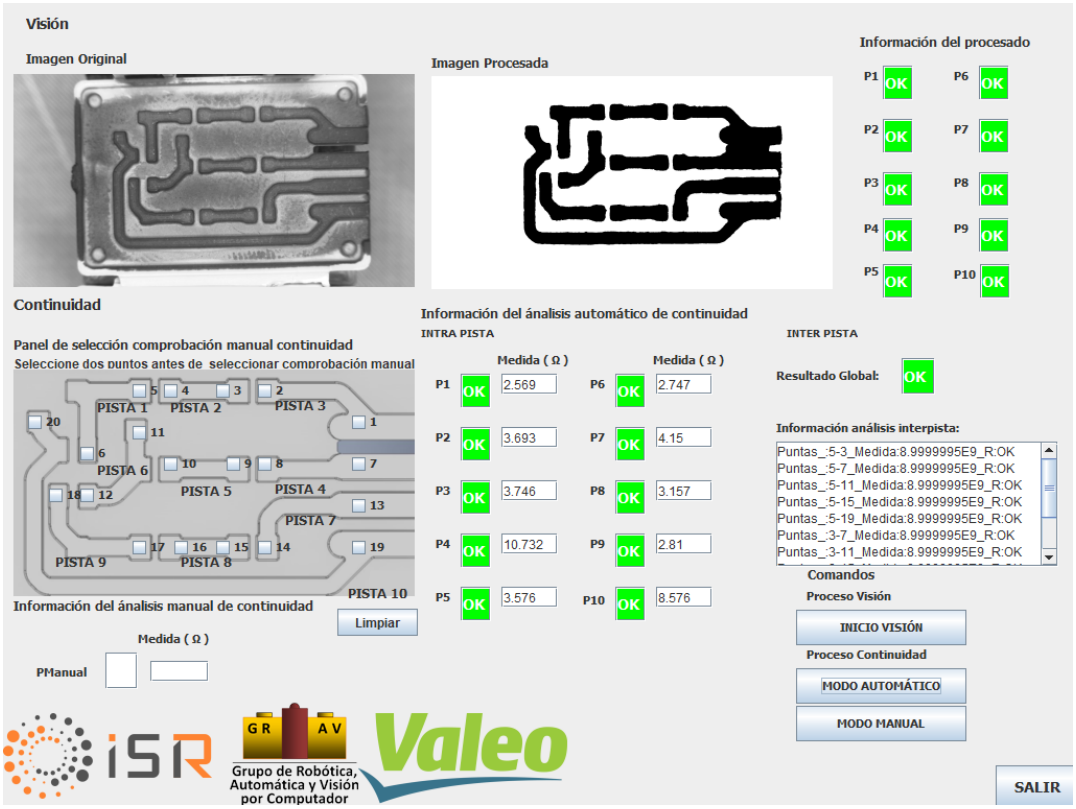



Figura 3.11. PCB 3.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.569 | OK | 0.6 | OK | |
| **2** | 3.693 | OK | 0.8 | OK | |
| **3** | 3.746 | OK | 1.8 | OK | |
| **4** | 10.732 | OK | 11.4 | OK | |
| **5** | 33.576 | OK | 1.7 | OK | |
| **6** | 2.747 | OK | 0.6 | OK | |
| **7** | 4.15 | OK | 1.1 | OK | |
| **8** | 3.157 | OK | 0.2 | OK | |
| **9** | 2.81 | OK | 0.8 | OK | |
| **10** | 8.576 | OK | 6.5 | OK | |

Tabla 3.5. Datos intrapista PCB 3.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | 17.06M | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.6. Datos interpista PCB 3.

## *PCB 4*

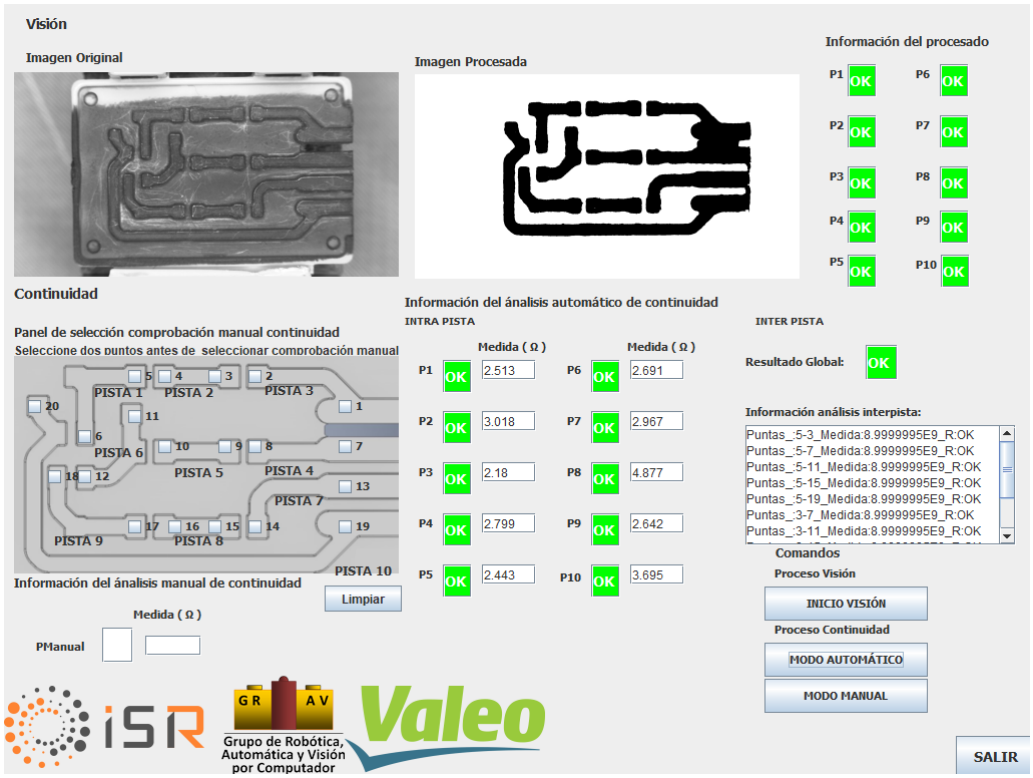



Figura 3.12. PCB 4.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.513 | OK | 0.2 | OK | |
| **2** | 3.018 | OK | 0.2 | OK | |
| **3** | 2.18 | OK | 0.1 | OK | |
| **4** | 2.799 | OK | 0.7 | OK | |
| **5** | 2.443 | OK | 0.4 | OK | |
| **6** | 2.691 | OK | 0.6 | OK | |
| **7** | 2.967 | OK | 0.3 | OK | |
| **8** | 4.887 | OK | 2.4 | OK | |
| **9** | 2.642 | OK | 1.1 | OK | |
| **10** | 3.695 | OK | 1.5 | OK | |

Tabla 3.7. Datos intrapista PCB 4.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.8. Datos interpista PCB 4.

## *PCB 5*

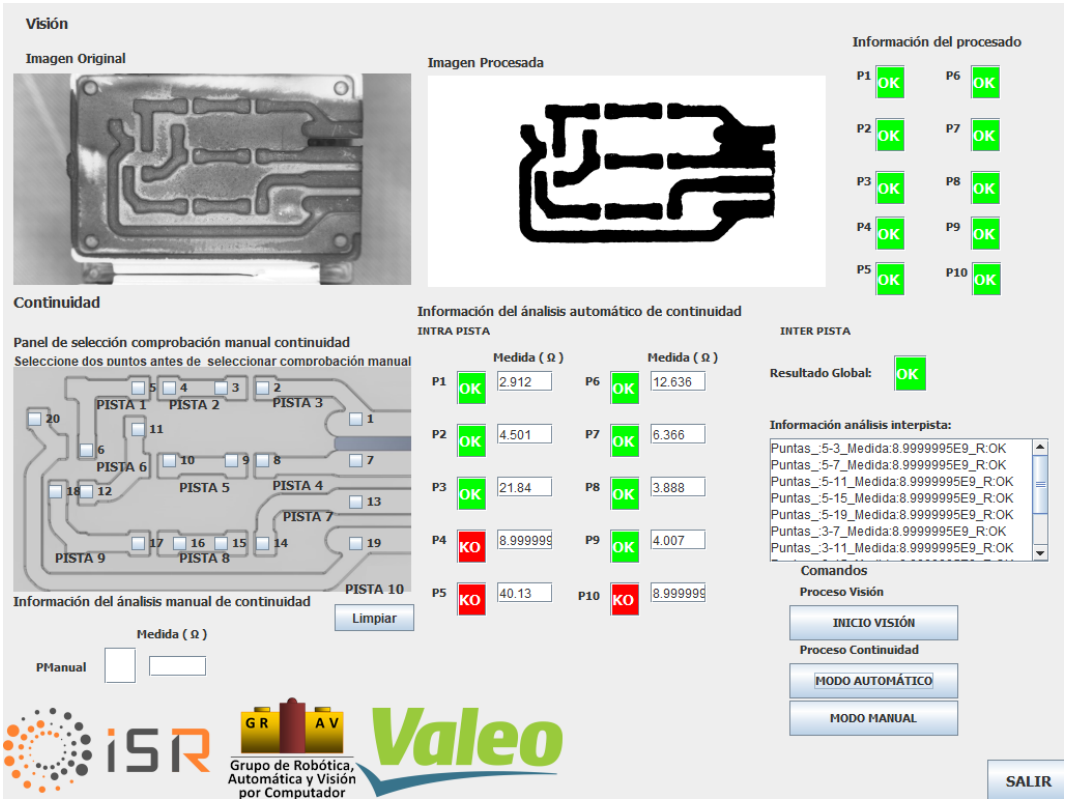



Figura 3.13. PCB 5.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.912 | OK | 1.1 | OK | |
| **2** | 4.501 | OK | 1.6 | OK | |
| **3** | 21.84 | OK | 20.6 | OK | |
| **4** | INF | KO | 117.9 | KO | |
| **5** | 40.13 | KO | 44.9 | KO | |
| **6** | 12.636 | OK | 10.6 | OK | |
| **7** | 6.366 | OK | 3.7 | OK | |
| **8** | 3.888 | OK | 1.4 | OK | |
| **9** | 4.007 | OK | 2.1 | OK | |
| **10** | INF | KO | 84.5 | KO | |

Tabla 3.9. Datos intrapista PCB 5.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.10. Datos interpista PCB 5.

## *PCB 6*

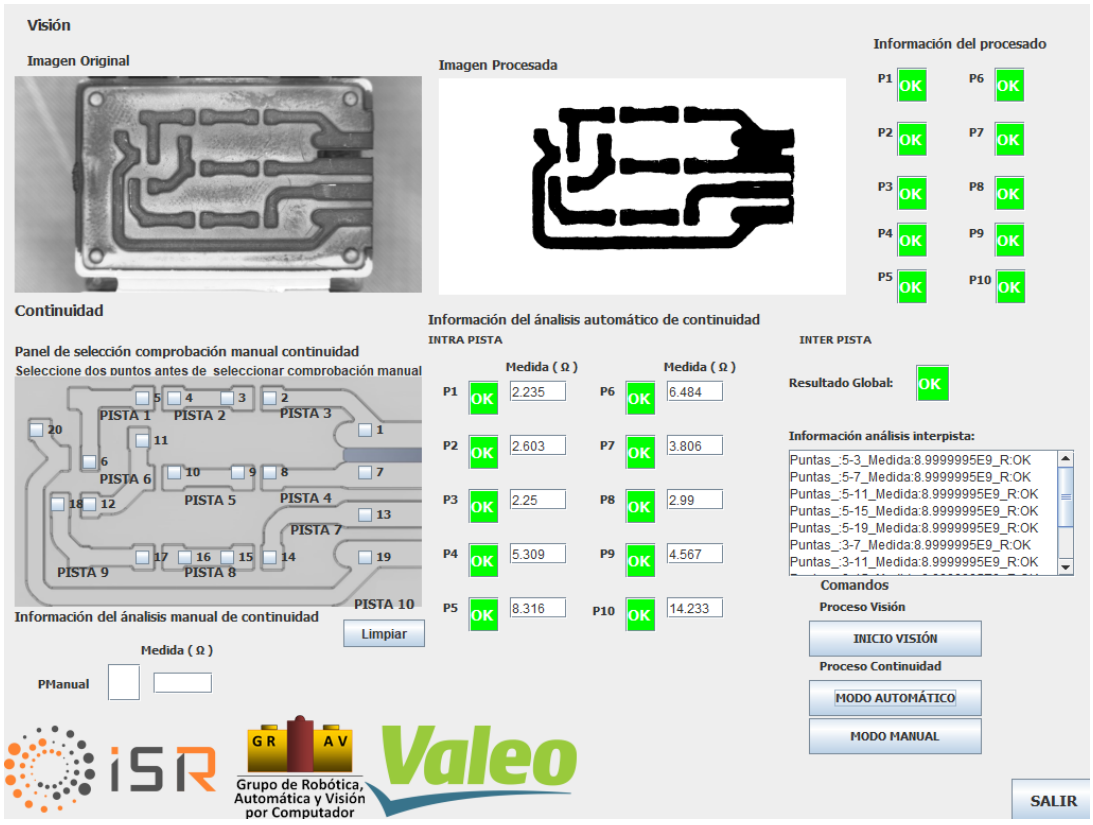



Figura 3.14. PCB 6.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.235 | OK | 0.4 | OK | |
| **2** | 2.603 | OK | 0.2 | OK | |
| **3** | 2.25 | OK | 0.5 | OK | |
| **4** | 5.309 | OK | 2.7 | OK | |
| **5** | 8.316 | OK | 4.1 | OK | |
| **6** | 6.484 | OK | 4.5 | OK | |
| **7** | 3.806 | OK | 1.3 | OK | |
| **8** | 2.99 | OK | 0.4 | OK | |
| **9** | 4.567 | OK | 2.8 | OK | |
| **10** | 14.233 | OK | 10.5 | OK | |

Tabla 3.11. Datos intrapista PCB 6.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.12. Datos interpista PCB 6.

## *PCB 7*

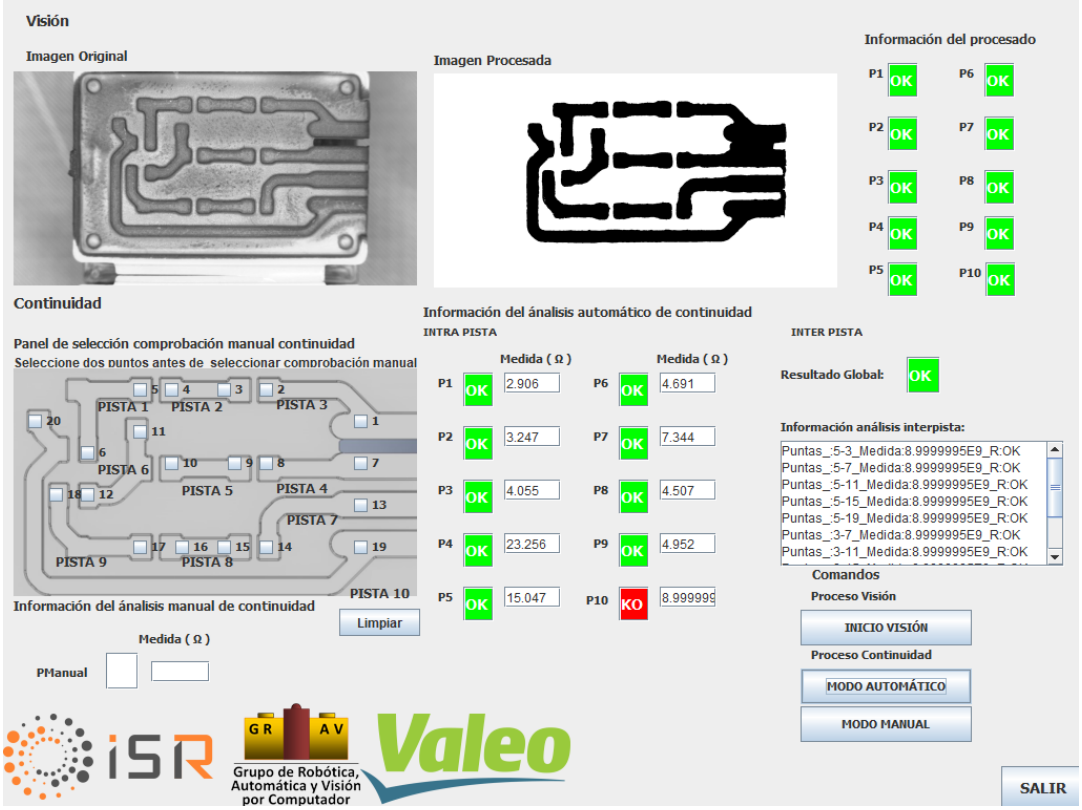



Figura 3.15. PCB 7.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.906 | OK | 1.1 | OK | |
| **2** | 3.247 | OK | 0.8 | OK | |
| **3** | 4.055 | OK | 2.3 | OK | |
| **4** | 23.256 | OK | 27.4 | OK | |
| **5** | 15.047 | OK | 13.1 | OK | |
| **6** | 4.691 | OK | 2.9 | OK | |
| **7** | 7.344 | OK | 4.6 | OK | |
| **8** | 4.507 | OK | 2.1 | OK | |
| **9** | 4.952 | OK | 3.5 | OK | |
| **10** | INF | KO | INF | KO | |

Tabla 3.13. Datos intrapista PCB 7.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.14. Datos interpista PCB 7.

## *PCB 8*

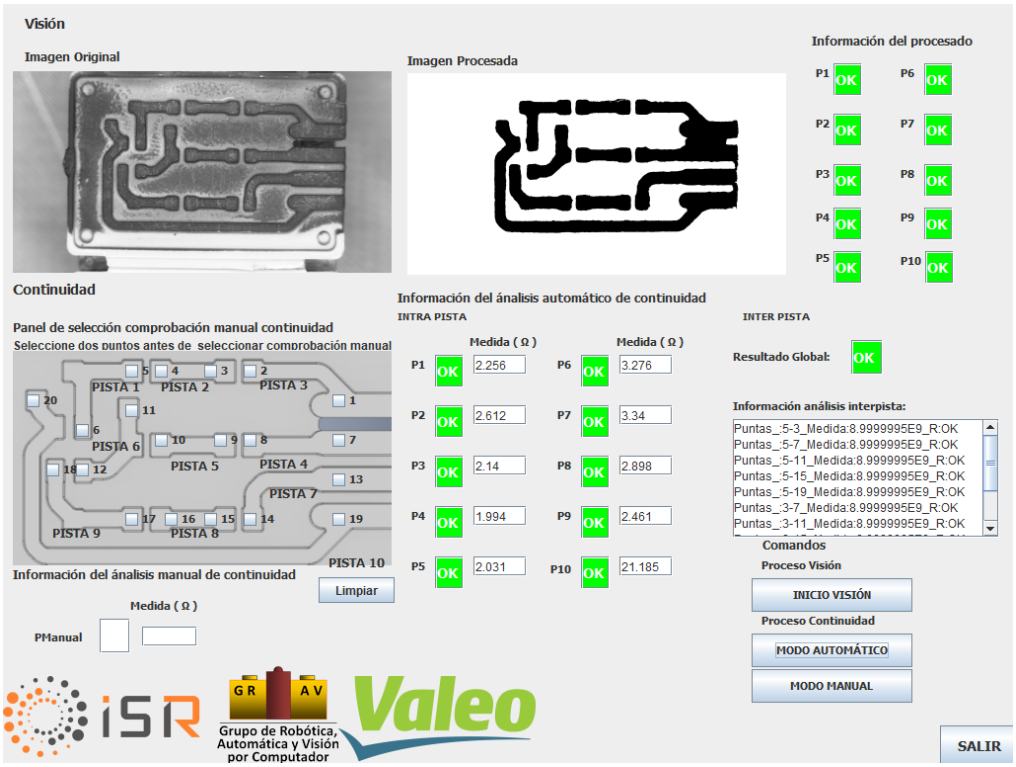



Figura 3.16. PCB 8.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.256 | OK | 0.6 | OK | |
| **2** | 2.612 | OK | 0.1 | OK | |
| **3** | 2.14 | OK | 0.4 | OK | |
| **4** | 1.994 | OK | 0.4 | OK | |
| **5** | 2.031 | OK | 0.1 | OK | |
| **6** | 3.276 | OK | 1.2 | OK | |
| **7** | 3.34 | OK | 0.7 | OK | |
| **8** | 2.898 | OK | 0.2 | OK | |
| **9** | 2.461 | OK | 0.6 | OK | |
| **10** | 21.185 | OK | 19.6 | OK | |

Tabla 3.15. Datos intrapista PCB 8.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.16. Datos interpista PCB 8.

## *PCB 9*

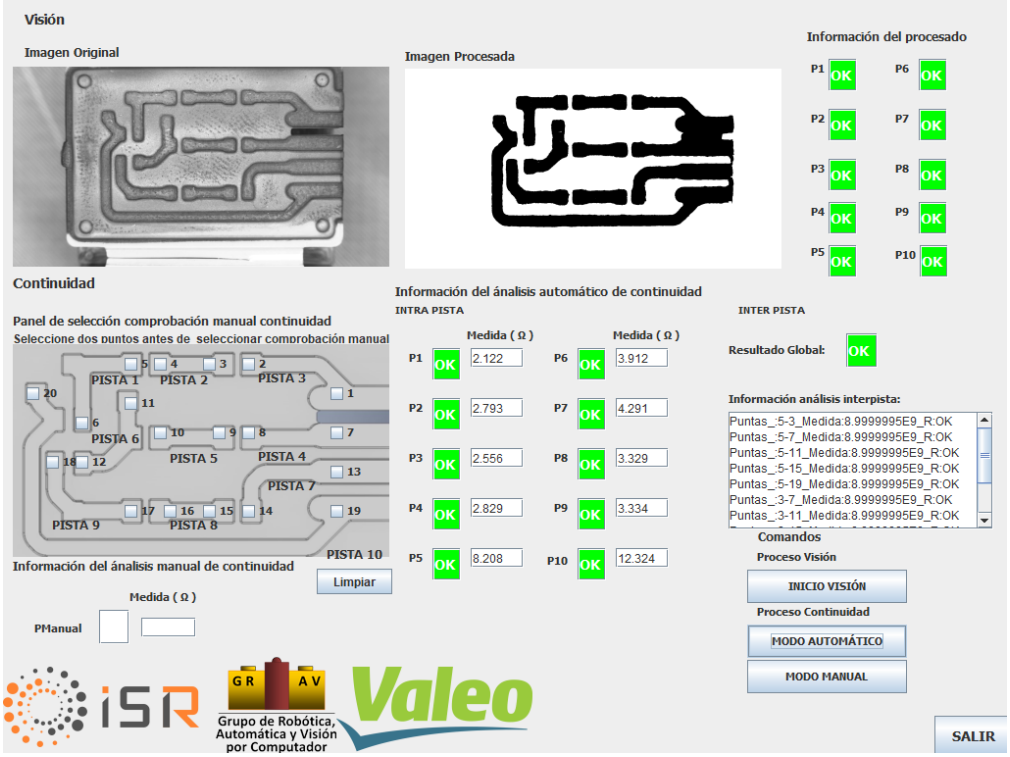



Figura 3.17. PCB 9.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.122 | OK | 0.2 | OK | |
| **2** | 2.793 | OK | 0.2 | OK | |
| **3** | 2.556 | OK | 0.8 | OK | |
| **4** | 2.829 | OK | 1.2 | OK | |
| **5** | 8.208 | OK | 6.8 | OK | |
| **6** | 3.912 | OK | 1.9 | OK | |
| **7** | 4.291 | OK | 1.6 | OK | |
| **8** | 3.329 | OK | 0.6 | OK | |
| **9** | 3.334 | OK | 1.6 | OK | |
| **10** | 12.324 | OK | 10.7 | OK | |

Tabla 3.17. Datos intrapista PCB 9.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.18. Datos interpista PCB 9.

## *PCB 10*

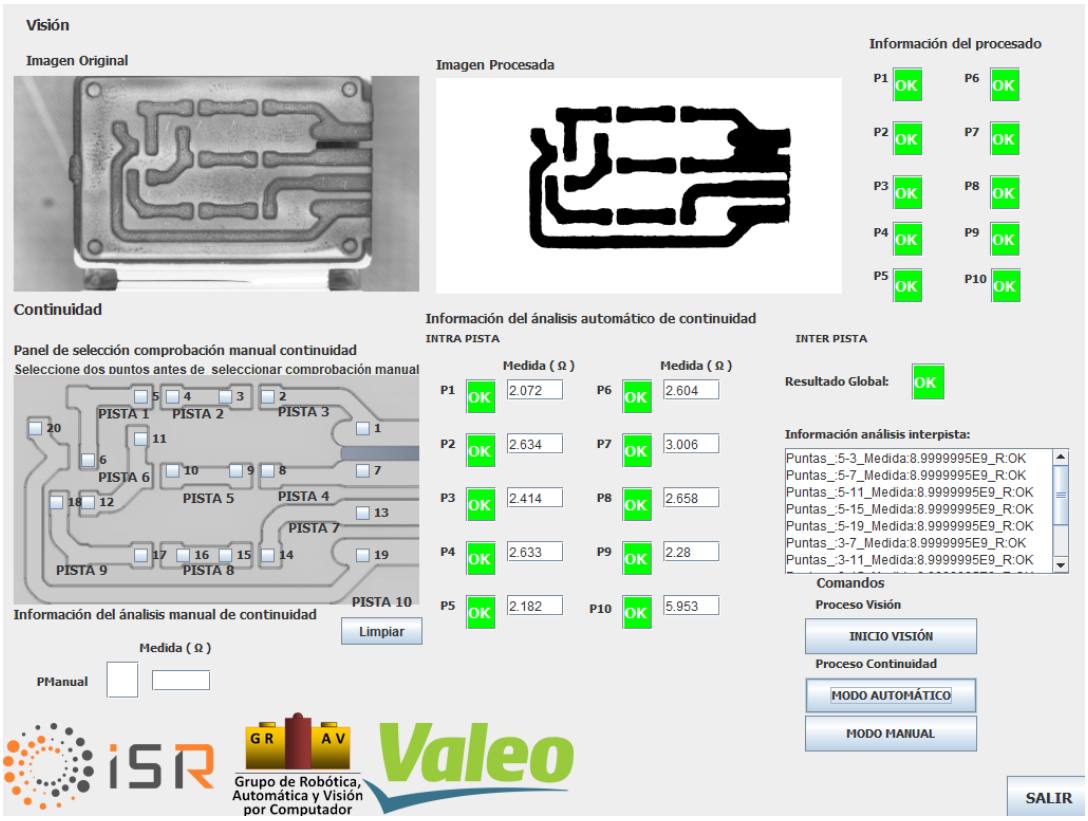



Figura 3.18. PCB 10.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | 2.072 | OK | 0.3 | OK | |
| **2** | 2.634 | OK | 0.1 | OK | |
| **3** | 2.414 | OK | 0.6 | OK | |
| **4** | 6.633 | OK | 1 | OK | |
| **5** | 2.182 | OK | 0.3 | OK | |
| **6** | 2.604 | OK | 0.6 | OK | |
| **7** | 3.066 | OK | 0.3 | OK | |
| **8** | 2.658 | OK | 0.2 | OK | |
| **9** | 2.28 | OK | 0.4 | OK | |
| **10** | 5.953 | OK | 3.7 | OK | |

Tabla 3.19. Datos intrapista PCB 10.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.20. Datos interpista PCB 10.

## *PCB 11*

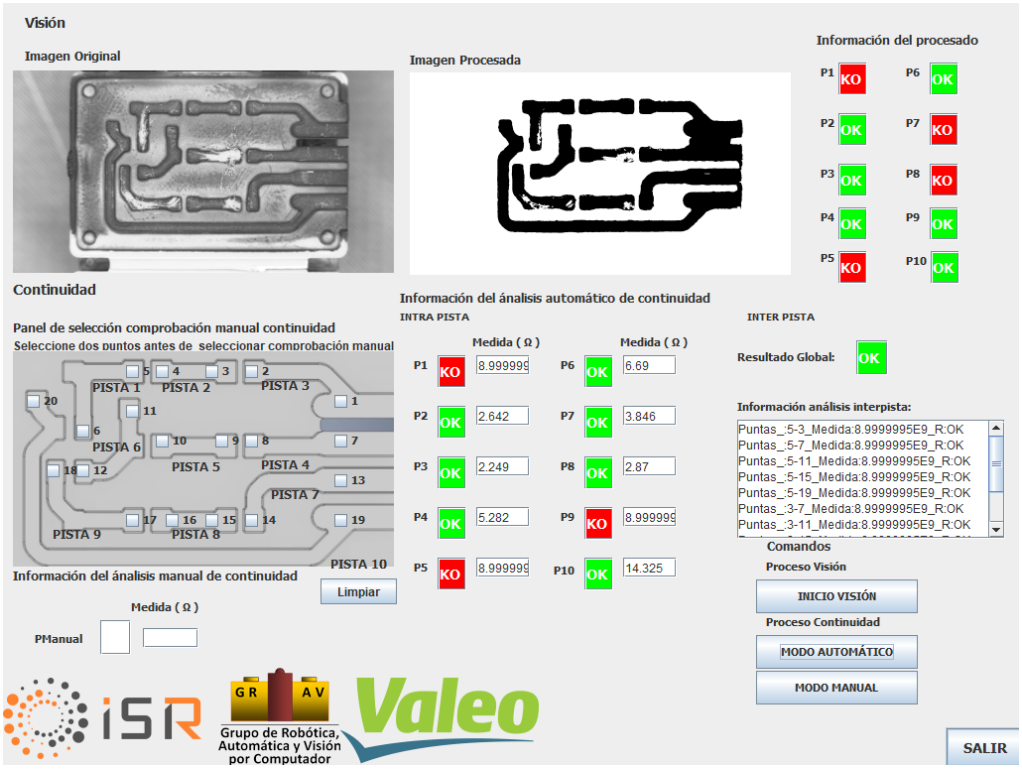



Figura 3.19. PCB 11.

| ANÁLISIS INTRAPISTA | | | | | |
|---|---|---|---|---|---|
| **Pistas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **1** | INF | KO | INF | KO | |
| **2** | 2.642 | OK | 0.1 | OK | |
| **3** | 2.249 | OK | 0.5 | OK | |
| **4** | 5.282 | OK | 4 | OK | |
| **5** | INF | KO | INF | KO | |
| **6** | 6.69 | OK | 4.6 | OK | |
| **7** | 3.846 | OK | 1.1 | OK | |
| **8** | 2.87 | OK | INF | KO | |
| **9** | INF | KO | INF | KO | |
| **10** | 14.325 | OK | 12.6 | OK | |

Tabla 3.21. Datos intrapista PCB 11.

| ANÁLISIS INTERPISTA | | | | | |
|---|---|---|---|---|---|
| **Puntas** | **Medida Máquina (Ω)** | **Resultado** | **Medida Manual (Ω)** | **Resultado** | **Diferencia** |
| **5-3** | INF | OK | INF | OK | |
| **5-7** | INF | OK | INF | OK | |
| **5-11** | INF | OK | INF | OK | |
| **5-15** | INF | OK | INF | OK | |
| **5-19** | INF | OK | INF | OK | |
| **3-7** | INF | OK | INF | OK | |
| **3-11** | INF | OK | INF | OK | |
| **3-15** | INF | OK | INF | OK | |
| **3-19** | INF | OK | INF | OK | |
| **1-9** | INF | OK | INF | OK | |

Tabla 3.22. Datos interpista PCB 12.

Si analizamos las pruebas realizadas a las distintas fases de análisis podremos obtener el porcentaje de acierto de nuestro prototipo

| | **Visión** | **Continuidad** |
|---|---|---|
| **Nº de pruebas** | **8** | **220** |
| **Nº de aciertos** | **8** | **219** |
| **Nº de fallos** | **0** | **1** |
| **Tasa de acierto** | **100 %** | **99.54 %** |

Tabla 3.23. Tasa de acierto.

En el cálculo del porcentaje de acierto de la etapa de visión hemos tenido en cuenta los defectos visuales graves (8 en total) y no hemos tenido en cuenta los defectos visuales leves ya que como hemos comprado anteriormente estos nos son críticos en el funcionamiento. Si tuviéramos en cuenta los defectos leves el porcentaje de acierto sería del 88.88%.

Por otro lado si estudiamos el único error que cometemos en la etapa de continuidad podemos entender cómo se ha originado. El error se comete en la pista 8, como podemos ver en la figura 3.20 el defecto visual está en el extremos de la pista, esto justifica por qué el resultado del análisis de continuidad ha sido positivo. Las puntas caen justamente en el tramo de pista que queda entero.

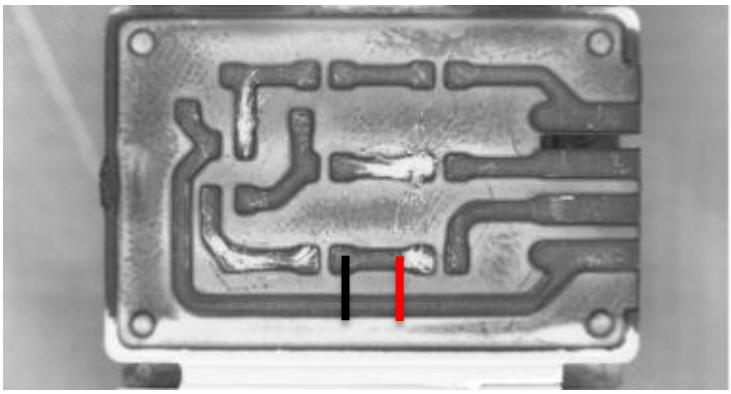


Figura 3.20. Causa del error en continuidad.

Este error tiene su causa en el diseño de la cama de puntas, debido a que a la hora de diseñar esta se eligieron las posiciones fijas que ocuparían las puntas. Al fijar las puntas se siguió el criterio del que la punta cayera cerca de los extremos. A priori podemos pensar que las posiciones óptimas están justo en los extremos de la pista, sin embargo aunque solucionaríamos el error que hemos mencionado anteriormente, el número de fallos que evitaríamos sería mayor, debido a que si las posiciones de las puntas se sitúan cerca del borde de la pista, existiría una mayor probabilidad de que la punta resbalase y cayese en el fondo de la PCB.

Una vez que hemos visto las pruebas que hemos llevado a cabo pasaremos a ver las imágenes del prototipo final.

## 3.2.-Prototipo final.

Figura 3.21. Vista frontal de la máquina.

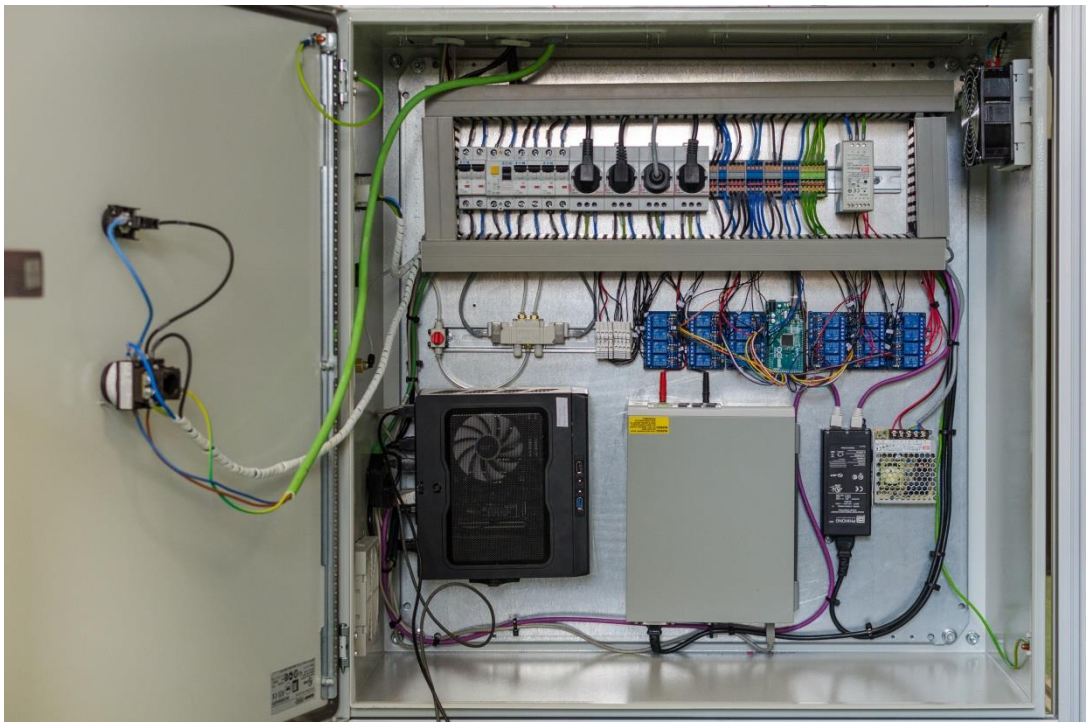


Figura 3.22. Cuadro eléctrico.

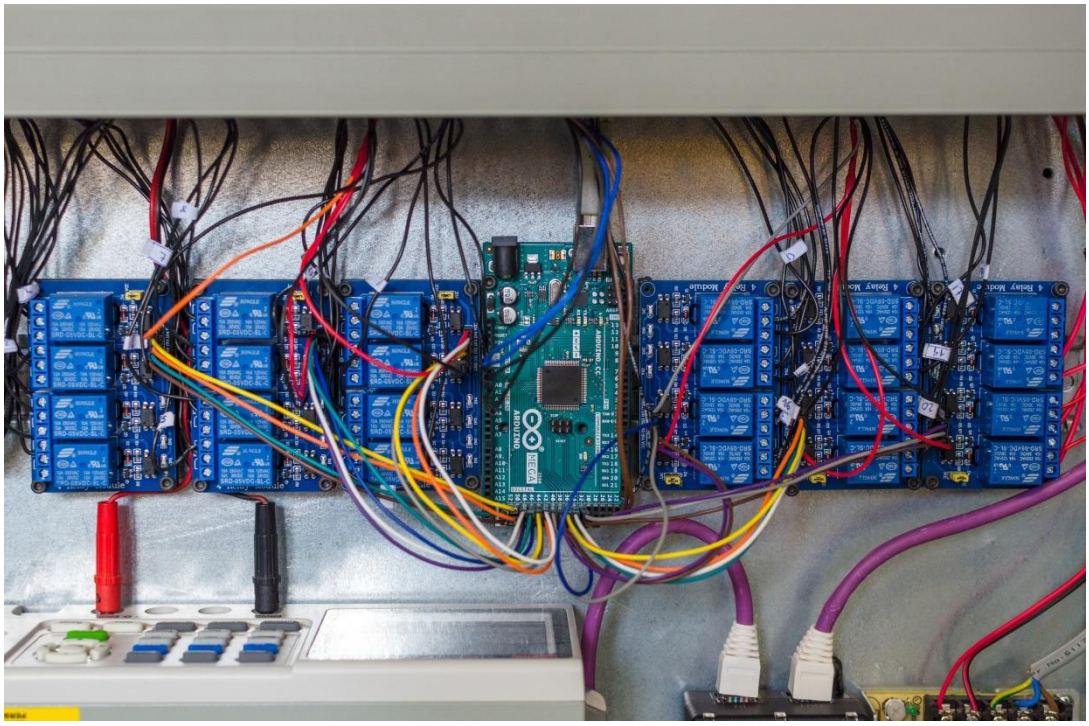


Figura 3.23. Conexionado de Arduino con los relés.

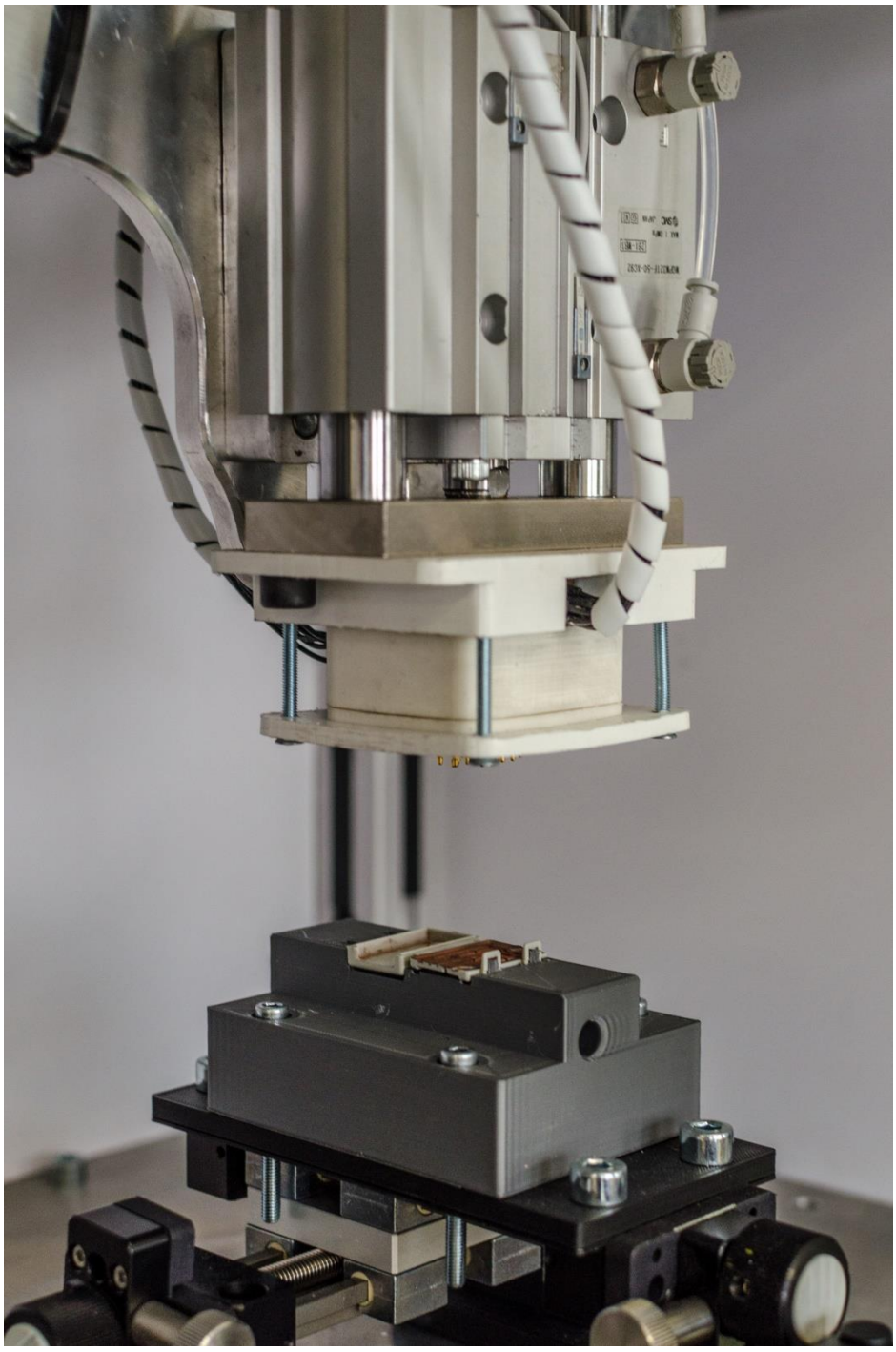


Figura 3.24. Sistema de testeo.

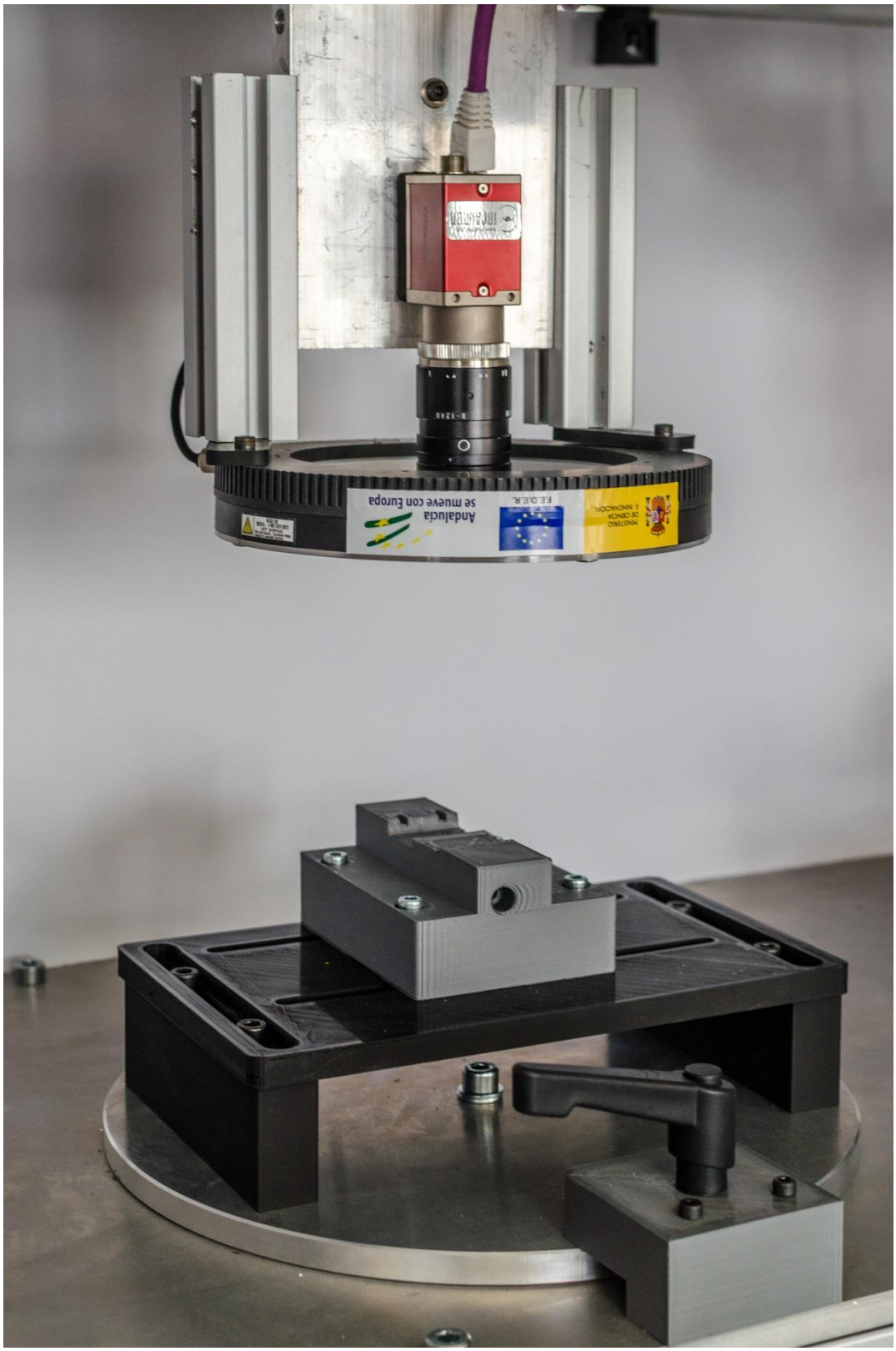



Figura 3.25. Sistema de visión.

# 4.-CONCLUSIONES Y TRABAJOS FUTUROS.

En este capítulo de la memoria discutiremos si se han logrado alcanzar los objetivos que se marcaron en el apartado 1.4 de la introducción. Además expondremos posibles mejoras del prototipo para implementar en un futuro.

Si analizamos los objetivos generales (los cuales eran dos, uno para cada fase de comprobación, Visión y Continuidad), concluimos que hemos logrado cumplir las metas que nos propusimos al comienzo de la memoria. Estas son:

- **Proceso de Visión**: Hemos conseguido diferenciar las PCBs que tienen defectos visuales graves, ya que como hemos explicado anteriormente en el capítulo de resultados, estos son los defectos que contribuyen a provocar un fallo de continuidad.
- **Proceso de Continuidad**: Por otro lado alcanzamos discriminar los defectos de continuidad, es decir, distinguimos cuando una pista no conduce y cuando las pistas no existe un cortocircuito entre ellas, es decir, cuando están aisladas eléctricamente.

En cuanto a los objetivos específicos, estudiaremos cada caso de forma individual.

- Hemos seleccionado tanto los elementos y condiciones más adecuadas para la adquisición.
- Conseguimos obtener una imagen procesada de la PCB, la cual nos permita identificar los fallos visuales de las pistas.
- Elaboramos un programa en Arduino que se comunica con Java. Java le manda distintos ‘Strings’ y Arduino los interpreta y en función de que cadena haya recibido activará o desactivará el relé correspondiente.
- Construimos el programa de Java, cuya función principal es gobernar las máquinas de estados que controlan el funcionamiento del prototipo.

- Desarrollamos la interfaz gráfica que permite al operario interactuar en el proceso y que muestra los resultados del procesado de visión y del análisis de continuidad.
- Hemos sometido a pruebas a nuestro prototipo para ver la robustez del mismo.

Aunque hemos conseguido todos los objetivos que nos marcamos, siempre es posible mejorar y nuestro prototipo no es ninguna excepción. Si hablamos de las mejoras de la etapa que podemos considerar que está más relacionada con este Trabajo de Fin de Grado, es decir con las que se refieren a la etapa de visión, podemos pensar en varias.

Por ejemplo, una posible mejora sería realizar una máscara "dinámica" ante la máscara "estática" actual. Al obtener un modelo dinámico solucionaríamos los posibles problemas que pudieran aparecer por desplazamientos indeseados cuando situemos nuestra PCB en nuestro set-up de adquisición. Estos problemas no tienen por qué ocurrir si tenemos nuestro set- up de adquisición está fijado, pero es posible que la PCB sufra algún defecto que no sea crítico en el funcionamiento pero que provoque que la imagen adquirida y la imagen de la máscara no estén alineadas. Un posible procedimiento para obtener una máscara "dinámica" sería buscar puntos de referencia para colocar nuestra máscara sobre la imagen original. Estas referencias podrían ser los círculos que vemos en la figura 4.1.

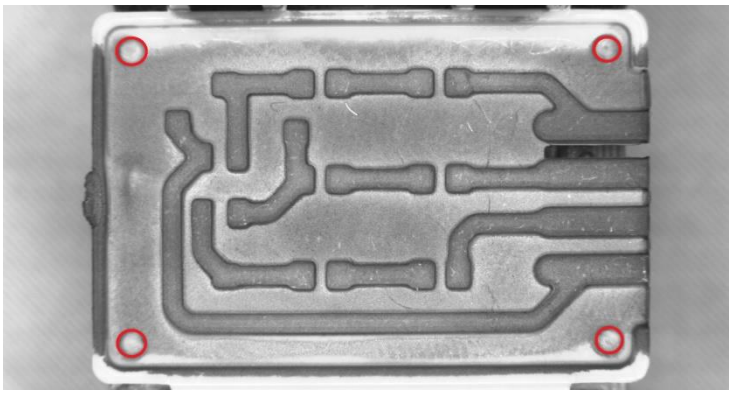


Figura 4.1. Posible referencia para máscara dinámica.

Otra campo de mejora del programa de visión sería la elaboración de un programa que automáticamente nos diga si la PCB es defectuosa. Para esto usaríamos una red neuronal, la cual entrenaríamos con imágenes de defectos con el objetivo de que posteriormente sea capaz de identificar de forma autónoma sin la necesidad de interpretar cada vez las características que extraemos (en nuestro caso el área) para decidir si se trata de un defecto. Este tipo de software de visión está basado en "Template matching" como explicamos en el apartado 1.3.1 del estado de la tecnología.

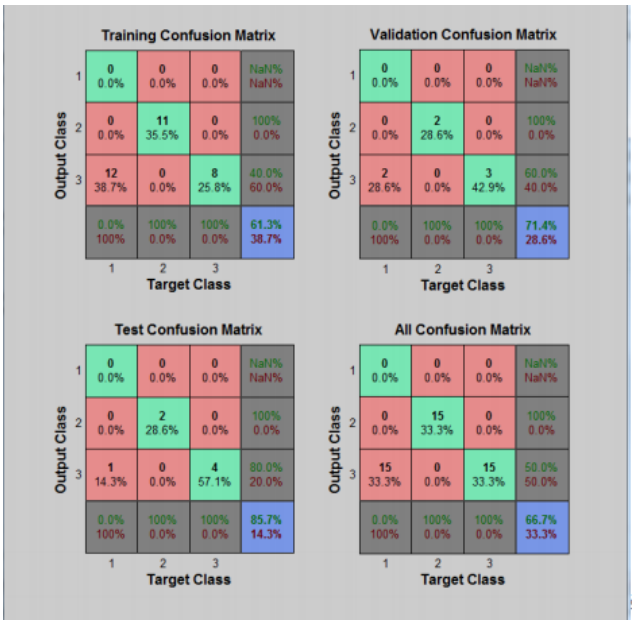



Figura 4.2.Resultado de un entrenamiento de una red neuronal.

Por otra parte, otra posible mejora del prototipo, ya en un ámbito general sería colocar una cinta que moviera la PCB de la etapa de visión a la fase del test de continuidad. También sería interesante la incorporación de un brazo robótico que colocara la PCB en el soporte al inicio del análisis y que la quitara al final del proceso, descartándola en caso de que se tratara de una PCB defectuosa. En la Figura 4.3 podemos ver un caso de colaboración entre un brazo robótico y una cinta.

Figura 4.3.Brazo robótico interactuando en una cinta.

## 5.-BIBLIOGRAFÍA Y REFERENCIAS.

AIMPLAS Instituto Tecnológico del Plástico (https://www.aimplas.es)

Proceso de producción de PCB https://www.pcbway.es/pcb-service.html

Moganti, M., & Ercal, F. (1995). Automatic PCB inspection systems IEEE Potentials, 14(3), 6–10.doi:10.1109/45.464686

Guo, F., & Guan, S. (2011). Research of the Machine Vision Based PCB Defect Inspection System International Conference on Intelligence Science and Information Engineering. doi:10.1109/isie.2011.47

Fundamentos de Arduino. https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

Adquisición https://dv.ujaen.es/goto_docencia_file_486151_download.html

Preprocesadohttps://dv.ujaen.es/goto_docencia_file_486160_download.html

Bordes https://dv.ujaen.es/goto_docencia_file_486161_download.html

Segmentaciónhttps://dv.ujaen.es/goto_docencia_file_486162_download.html

OpenCV https://opencv.org/

Arduino https://www.arduino.cc/

Matlab https://es.mathworks.com

# 6.-ANEXOS.

En este capítulo hablaremos sobre el material complementario necesario para una mayor compresión de este Trabajo de Fin de Grado y como consecuencia para un mayor entendimiento de nuestro prototipo. Encontraremos en esta parte de la memoria los siguientes cuatro sub-apartados:

- Presupuesto.
- Manual de usuario.
- Esquemas electrónicos

## ANEXO 1.-Presupuesto.

Para elaborar el presupuesto, lo dividiremos en tres capítulos, los cuales son:

- Capítulo 1: Cuadro eléctrico.
- Capítulo 2: Hardware Parte Electrónica.
- Capítulo 3: Hardware Parte Mecánica.

# PRESUPUESTO, DESCOMPUESTOS Y MEDICIONES

## Capítulo 1: Cuadro eléctrico

| CÓDIGO | UD. | RESUMEN | CANTIDAD | PRECIO | IMPORTE |
|---|---|---|---|---|---|
| *247649* | *u* | Interruptor Magnetotérmico de 25 A. | *1.00* | *20.040* | *20.040* |
| *247647* | *u* | Interruptor Magnetotérmico de 16 A. | *1.00* | *20.040* | *20.040* |
| *247645* | *u* | Interruptor Magnetotérmico de 10 A. | *1.00* | *20.040* | *20.040* |
| *235753* | *u* | Interruptor Diferencial de 25 A a 2 polos a 30mA | 1.00 | 89.180 | 89.190 |
| *091078* | *u* | Seccionador de 20 A Montaje Empotrado | *1.00* | *46.870* | *46.870* |
| *VE3085* | *u* | *Ventilador 120x120 230 V* | *1.00* | *31.680* | *31.680* |
| *49337* | *u* | *Rejilla filtro Ventilador 150x150 mm* | *1.00* | *13.320* | *13.320* |
| *RJV0004* | *u* | *Rejilla para Ventiladores de 120 mm* | *1.00* | *3.000* | *3.000* |
| *216771* | *u* | *Indicador luminoso blanco* | *1.00* | *5.190* | *5.190* |
| *216563* | *u* | *LED230-W Blanco 85/265 V AC* | *1.00* | *14.890* | *14.890* |
| *107413* | *u* | *Conector hembra RJ45* | *1.00* | *20.570* | *20.570* |
| *00102* | *u* | *100 A Schuko 2P+T220 16 A Azul* | *1.00* | *3.630* | *3.630* |
| *3094704* | *u* | *Legrand 2P+T220 16 A* | *4.00* | *15.920* | *63.680* |
| *3209510* | *u* | *Borna de conexión PIT 2,5* | *10.00* | *0.341* | *3.410* |
| *3209510* | *u* | *Borna de conexión PIT 2,5 BU* | *10.00* | *0.341* | *3.410* |

# PRESUPUESTO, DESCOMPUESTOS Y MEDICIONES

## Capítulo 1: Cuadro eléctrico

| CÓDIGO | UD. | RESUMEN | CANTIDAD | PRECIO | IMPORTE |
|---|---|---|---|---|---|
| *3209536* | *u* | Borna de conexión PIT 2,5 PE | *10.00* | *1.330* | *13.300* |
| *3030417* | *u* | Tapa final D-ST 2,5 | *5.00* | *0.484* | *2.420* |
| *216563* | u | Portaetiquetas | *3.00* | *14.890* | *14.890* |
| *POE29U-1AF* | *u* | F.A Inyector PoE | 1.00 | 17.140 | 17.140 |
| *2815984* | *u* | F.A Sistema de Iluminación | *1.00* | *11.580* | *11.580* |
| *MDR-60-24* | *u* | *F.A Electroválvula (IN 230 V/ XA OUT 24V /2.5 A )* | *1.00* | *23.270* | *23.270* |
| *3038930* | *u* | *Puente enchufable* | *1.00* | *16.229* | *16.229* |
| *161-4032* | *u* | *Puntera crimpado Diámetro 2,5 mm* | *66.00* | *0.0945* | *6.237* |
| *152325008* | *ml* | *Cable Fase, Neutro y Tierra 2,5 mm* | *15.00* | *0.529* | *7.935* |
| *444-2947* | *u* | *Prensaestopa* | *3.00* | *1.084* | *3.252* |
| *Total cantidades alzadas* | | | *1.00* | | |
| | | | | | *475.213* |

# PRESUPUESTO, DESCOMPUESTOS Y MEDICIONES

## Capítulo 2: Hardware Parte Electrónica

| CÓDIGO | UD. | RESUMEN | CANTIDAD | PRECIO | IMPORTE |
|---|---|---|---|---|---|
| *ARD-0033* | *u* | Arduino Mega 2560 rev 3 | *1.00* | *34.500* | *34.500* |
| *CPM-0067* | *u* | Módulo 4 Relés 5 V | *6.00* | *4.950* | *29.700* |
| | u | PC | *1.00* | *227.999* | *227.999* |
| *122-5156* | *u* | Ohmetro | 1.00 | 921.000 | 921.000 |
| *TPM-150-AS-U* | *u* | Monitor | *1.00* | *760.000* | *760.000* |
| *7587494* | *u* | *Cable USB tipo B* | *2.00* | *3.390* | *6.780* |
| *PRO-0019* | *u* | *Cable Jumper Macho-Hembra* | *30* | *0.115* | *3.450* |
| *Total cantidades alzadas* | | | *1.00* | | |
| | | | | | *1,983.429* |

# PRESUPUESTO, DESCOMPUESTOS Y MEDICIONES

## Capítulo 3: Hardware Parte Mecánica

| CÓDIGO | UD. | RESUMEN | CANTIDAD | PRECIO | IMPORTE |
|---|---|---|---|---|---|
| *5009* | *ml* | Perfil básico 40x40 aluminio anodizado natural 5 | *15.00* | *3.613* | *54.195* |
| | *u* | *Plataforma metálica para soporte de utillajes* | *1.00* | *75.000* | *75.000* |
| | u | Plato giratorio | *2.00* | *35.000* | *70.000* |
| *19-023292-* | *u* | Mesa X-Y | 1.00 | 402.870 | 402.870 |
| | *u* | Útil para el soporte de los moldes de las placas | *2.00* | *15.250* | *30.500* |
| | *u* | *Cama de puntas* | *1.00* | *21.750* | *21.750* |
| *GKS-100-0160* | *u* | *Puntas* | *20.00* | *1.710* | *34.200* |
| *KS-10030G12* | *u* | *Soporte puntas* | *20.00* | *0.410* | *8.200* |
| *G223C* | *u* | *Cámara Mako G223C PoE* | *1.00* | *1441.700* | *1441.700* |
| *HPR-150SW* | *u* | *Sistema de iluminación radial* | *1.00* | *983.840* | *983.840* |
| | *m2* | *Cerramiento PVC extruido color blanco 5 mm* | *3.00* | *15,000* | *45.000* |
| *0204565225216* | *u* | *Electroválvula SY5220-5LOU-01F-Q* | *1.00* | *41.280* | *41.820* |
| *0000201704011* | *u* | *Racor KQ2H04-01AS* | *3.00* | *0.890* | *2.670* |
| *0201041001555* | *u* | *Silencioso AN10-01* | *2.00* | *1.720* | *3.440* |
| *0204301051017* | *u* | *Conector SY100-65-A-10* | *2.00* | *3.470* | *6.940* |

# PRESUPUESTO, DESCOMPUESTOS Y MEDICIONES

## Capítulo 3: Hardware Parte Mecánica

| CÓDIGO | UD. | RESUMEN | CANTIDAD | PRECIO | IMPORTE |
|---|---|---|---|---|---|
| *0204055001610* | *u* | Soporte SX50000-16-1 A | *1.00* | *1.520* | *1.520* |
| *0501220151409* | *u* | Regulador de caudal AS2201F-01-04SA | *6.00* | *5.510* | *33.060* |
| | u | Mecanizado soporte actuador neumático | *1.00* | *45.000* | *45.000* |
| | *u* | Mecanizado soporte iluminación | 1.00 | 22.000 | 22.000 |
| | *u* | Soporte monitor táctil | *1.00* | *25.000* | *25.000* |
| | *u* | *Pata ajustable para perfil de 40 mm* | *4.00* | *7.200* | *28.800* |
| | *u* | *Tornillería variada* | *50* | *0.500* | *25.000* |
| | *u* | *Válvula de corte* | *1.00* | *4.750* | *4.750* |
| | *ml* | *Tubo neumático diámetro 4 mm* | *8.00* | *0.950* | *7.600* |
| | *u* | *Final de carrera* | *2.00* | *5.900* | *11.800* |
| | *ml* | *Canaletas ocultas para perfil de 40 mm* | *6.00* | *2.350* | *14.100* |
| *MGPM32TF-50Z* | *u* | *Cilindro neumático* | *1.00* | *282.180* | *282.180* |
| *Total alzadas* | | | *1.00* | | |

*3,722.135*

## RESUMEN DE PRESUPUESTO MATERIAL

| CAPÍTULO | RESUMEN | IMPORTE | % |
|---|---|---|---|
| 1 | Cuadro eléctrico…… | 475.213 | 7.69 |
| 2 | Hardware Parte Electrónica…….. | 1,983.429 | 32.10 |
| 3 | Hardware Parte Mecánica…….. | 3,722.135 | 60.21 |
| | **PRESUPUESTO DE EJECUCIÓN MATERIAL** | | **6,180.777** |
| | **21%IVA** | | 1,297.963 |
| | **PRESUPUESTO BASE DE LICITACIÓN** | | **7,478.740** |

Asciende el presupuesto a la expresada cantidad de SIETE MIL CUATROCIENTOS SETENTA Y OCHO EUROS con SETENTA Y CUATRO CÉNTIMOS

7 de junio 2018.

## ANEXO 2.-Manual de usuario.

En este apartado veremos la forma correcta de usar nuestro prototipo y algunas señalizaciones importantes a tenar en cuenta para evitar posibles fallos.

En primer lugar debemos tener en cuenta que para que nuestra aplicación se lance correctamente, es decir que nuestro archivo '.jar' se ejecute, este debe estar contenido en una carpeta junto con el resto de archivos que se aprecian en la Figura 6.1.

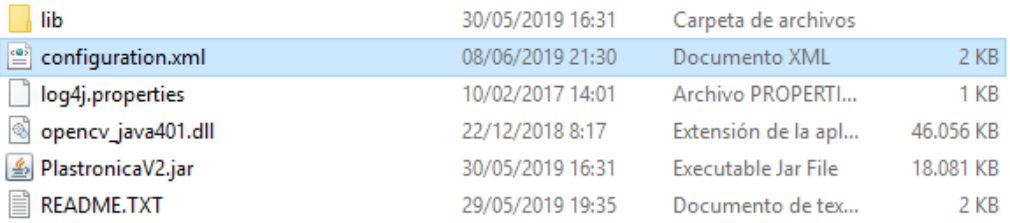



Figura 6.1 Contenido de la carpeta que contiene el .jar.

En la figura anterior podemos observar que hay una carpeta de archivos llamada 'lib' donde están todas las librerías que el programa emplea y las cuales podemos ver en la Figura 6.2. Las librerías que usamos en nuestro programa tienen finalidades distintas. Por ejemplo desde ser soporte para crear nuestra interfaz, como la librería 'AbsoluteLayaout', hasta para permitir la comunicación serie a través de la librería 'jSerialComm'.

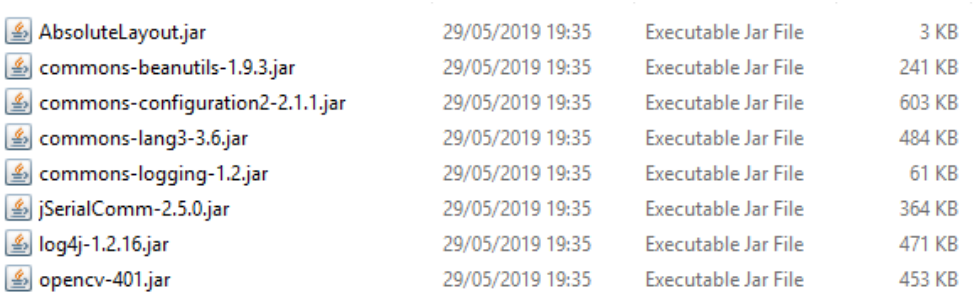



Figura 6.2. Librerías empleadas.

El archivo 'opencv_401.dll' debe aparecer en la carpeta con el fin de que podamos cargar la librería nativa, y poder usar la librería 'opencv_401.jar'. Un archivo fundamental para el funcionamiento de nuestro programa es 'configuration.xml', el cual podemos ver en la Figura 6.4.

```
<?xml version="1.0" encoding="UTF-8"?>
- <configuration>
    <com_arduino>COM3</com_arduino>
    <string_bajar_puntas>DOWN</string_bajar_puntas>
    <string_subir_puntas>UP</string_subir_puntas>
    <string_luz_on>I</string_luz_on>
    <string_luz_off>L</string_luz_off>
    <relay_on>ON</relay_on>
    <relay_off>OFF</relay_off>
    <com_multimeter>COM4</com_multimeter>
    <string_leer_multimetro>READ?</string_leer_multimetro>
    <path_to_images>D:\plastronikaTest\</path_to_images>
    <path_to_mask>D:\plastronikaTest\masks\</path_to_mask>
  - <pcb_vias>
      <via dn="2" dp="1">001.png</via>
      <via dn="4" dp="3">002.png</via>
      <via dn="6" dp="5">003.png</via>
      <via dn="8" dp="7">004.png</via>
      <via dn="10" dp="9">005.png</via>
      <via dn="12" dp="11">006.png</via>
      <via dn="14" dp="13">007.png</via>
      <via dn="16" dp="15">008.png</via>
      <via dn="18" dp="17">009.png</via>
      <via dn="20" dp="19">010.png</via>
    </pcb_vias>
  - <interpista_test>
      <pista dn="3" dp="5">1</pista>
      <pista dn="7" dp="5">2</pista>
      <pista dn="11" dp="5">3</pista>
      <pista dn="15" dp="5">4</pista>
      <pista dn="19" dp="5">5</pista>
      <pista dn="7" dp="3">6</pista>
      <pista dn="11" dp="3">7</pista>
      <pista dn="15" dp="3">8</pista>
      <pista dn="19" dp="3">9</pista>
      <pista dn="9" dp="1">10</pista>
    </interpista_test>
    <threshold_value_for_binarization>225</threshold_value_for_binarization>
    <threshold_for_continuity>40</threshold_for_continuity>
    <image_gray_level_max_value>255</image_gray_level_max_value>
    <threshold_for_defect>128</threshold_for_defect>
    <camera_ip>10.20.50.54</camera_ip>
    <exposure_time>12000</exposure_time>
    <gain>0</gain>
</configuration>
```

Figura 6.3. Archivo de configuración.

En el archivo de configuración está almacenada la información de la que partimos para nuestro programa. Los datos que almacenamos son los siguientes:

- **Puertos de Arduino y del multímetro**: Debemos comprobar los puertos a los que estén conectados estos dispositivos (para ello nos iremos a Administrador de dispositivos) y modificar el archivo poniendo los puertos correspondientes. En nuestro archivo de configuración el puerto del Arduino es el 3, mientras que el del multímetro es el 4, ya que hemos comprobado previamente que puerto corresponde a cada dispositivo, como vemos en la Figura 6.4.

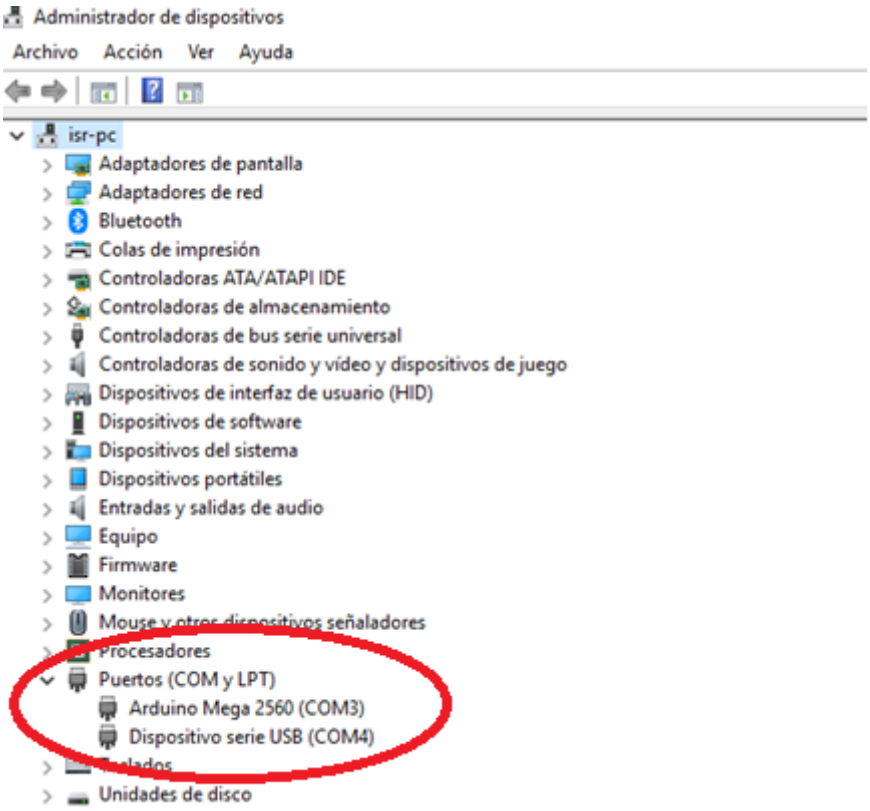



Figura 6.4. Puertos de Arduino y el multímetro.

- **Cadenas que texto que mandamos a Arduino**: Estos 'String' no deben ser modificados a no ser que queramos modificar el funcionamiento de nuestra máquina. Mediante el envío de estos 'String' a Arduino, activamos el relé que baja el cilindro (con el 'String' de 'DOWN'), el que sube el cilindro ('UP'), el que enciende la iluminación ('I'), el que apaga la iluminación ('L'), el que activa una punta ('ON'+'Nº de punta') y el que desactiva una punta (''OFF'+'Nº de punta'')
- **Cadena que texto que mandamos al multímetro:** En este caso solo mandaremos al multímetro 'READ?', 'String' que recibe el multímetro y hace que este devuelva el valor de la medida.
- **Dirección donde guardaremos imágenes:** En nuestro caso tenemos como 'path' 'D:\plastronikaTest\'. Aquí almacenaremos las imágenes adquiridas, como la que podemos ver en la Figura 6.5, y 10 imágenes de resultado del procesado de la última imagen adquirida, una para cada pista. Un ejemplo de estas imágenes resultado serían la que podemos

ver en la Figura 6.6. Esta imagen es el resultado de la operación XOR de la máscara de la pista 9 con la imagen original.

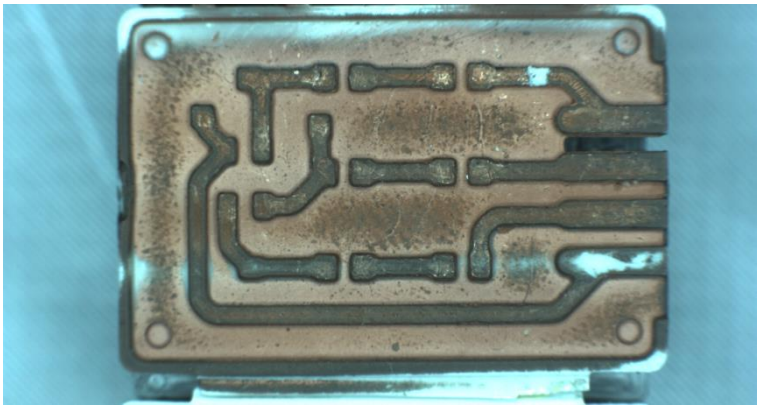


Figura 6.5. Imagen original.

Figura 6.6. Imagen resultado 'res009.png'.

- **Dirección donde guardamos las máscaras:** Son 11 las máscaras que necesitamos, una para cada pista, y una máscara global, estas las podemos ver en la siguiente Figura 6.7.

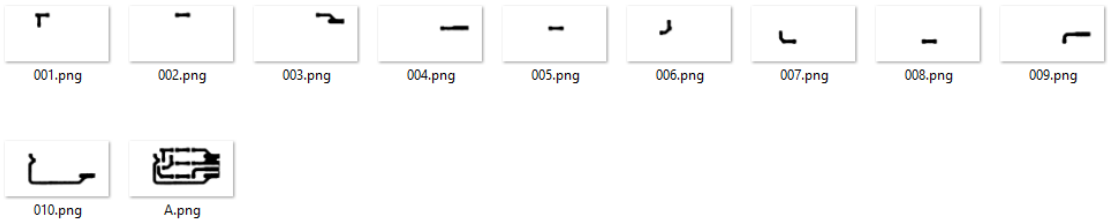



Figura 6.7. Máscaras usadas.

- **Vías para el análisis intrapista (‘pcb_vias’):** Almacenamos para cada pista, la pareja de puntas que debemos activar para medir la continuidad en la pista, junto con el nombre de la máscara para evaluar los defectos visuales de esta. Por ejemplo si observamos el primer valor de ‘pcb_vias’, para medir continuidad de la pista 10 debemos activar las puntas 19 y 20, como podemos ver en la Figura 6.8. Para el análisis de visión de la pista usaremos la máscara ’010.png’.

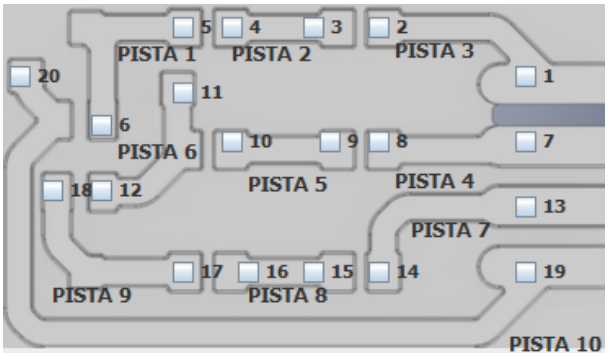



Figura 6.8. Distribución de las puntas.

- **Vías para el análisis interpista (‘interpista_test’):** Al igual que para el análisis intrapista, almacenamos las puntas que debemos activar en cada medida. En este caso no haremos inspección visual, ya que en este test medimos la continuidad entre pistas. Hemos seleccionado 10 de las 45 posibles medidas interpista que podemos hacer.

- **Umbral de binarización:** Valor que usaremos en el procesado para hacer obtener la imagen binarizada.
- **Umbral de continuidad:** Es el valor de resistencia que tomaremos como referencia para decidir si existe o no continuidad en la pista.
- **Máximo valor en nivel de gris:** Establecemos 255 como nivel máximo.
- **Umbral del defecto:** Este es valor representa el área a partir de la cual consideraremos que se trata de un defecto visual.
- **Dirección IP de la cámara:** Nuestra cámara tiene una dirección IP de '10.20.50.54'. Para que la cámara consiga conectarse correctamente debemos comprobar que la dirección de la interfaz Ethernet de nuestro PC. Para ello nos iremos a Panel de control, Redes e Internet y Conexiones de red. Una vez aquí elegiremos la interfaz a la que está conectada nuestra cámara.

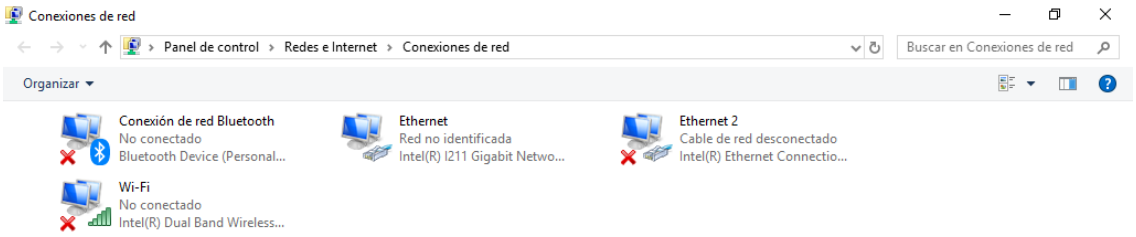



Figura 6.9. Comprobación Interfaz Ethernet.

Elegida la interfaz, comprobaremos en las propiedades de esta que la dirección IP empieza por '10.20.55', como es nuestro caso y como se puede observar en la Figura 6.10.

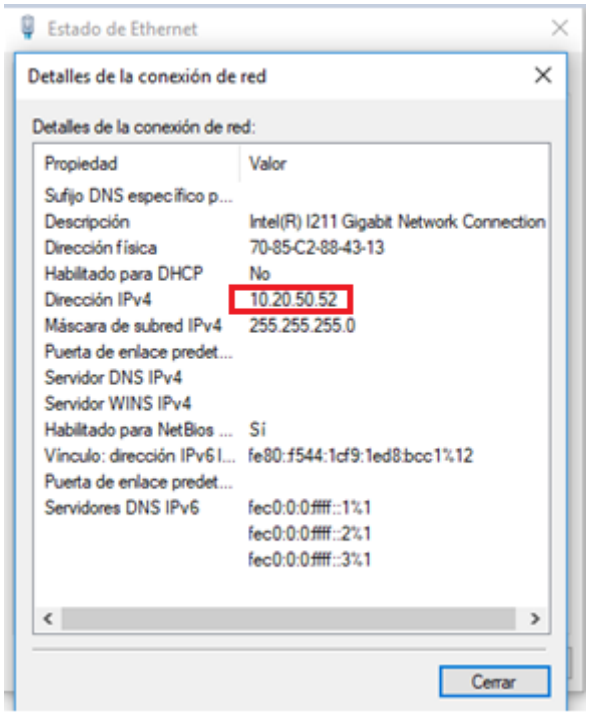



Figura 6.10. Propiedades interfaz Ethernet.

- **Tiempo de exposición de la cámara:** Este es el tiempo que la luz está alcanzando el sensor digital y determina de forma indirecta si nuestra imagen está enfocada. En nuestro conseguimos buenos resultados con un valor de 12000.

Una vez que hemos visto las consideraciones a tener a cuenta para que nuestra aplicación funcione correctamente, pasaremos el modo de empleo de nuestra aplicación.

En primer lugar colocamos la PCB en el set-up del test de visión. A continuación damos al botón de 'INICIO' (que aparece señalado en la Figura 6.11) para que comience el proceso.

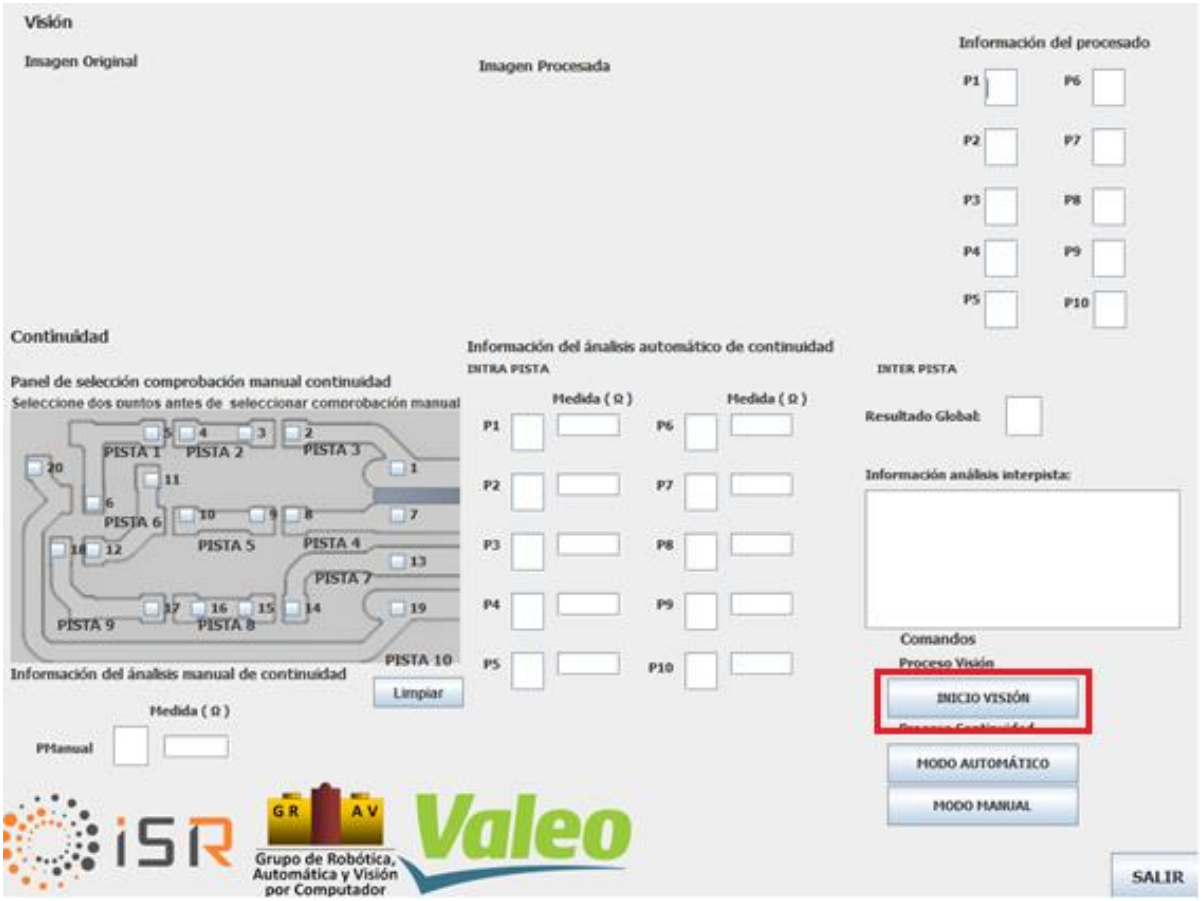



Figura 6.11. Inicio del proceso.

Una vez que la cámara adquiere la imagen, y esta es procesada, los resultados se muestran en la interfaz, los cuales podemos ver en la Figura 6.12.

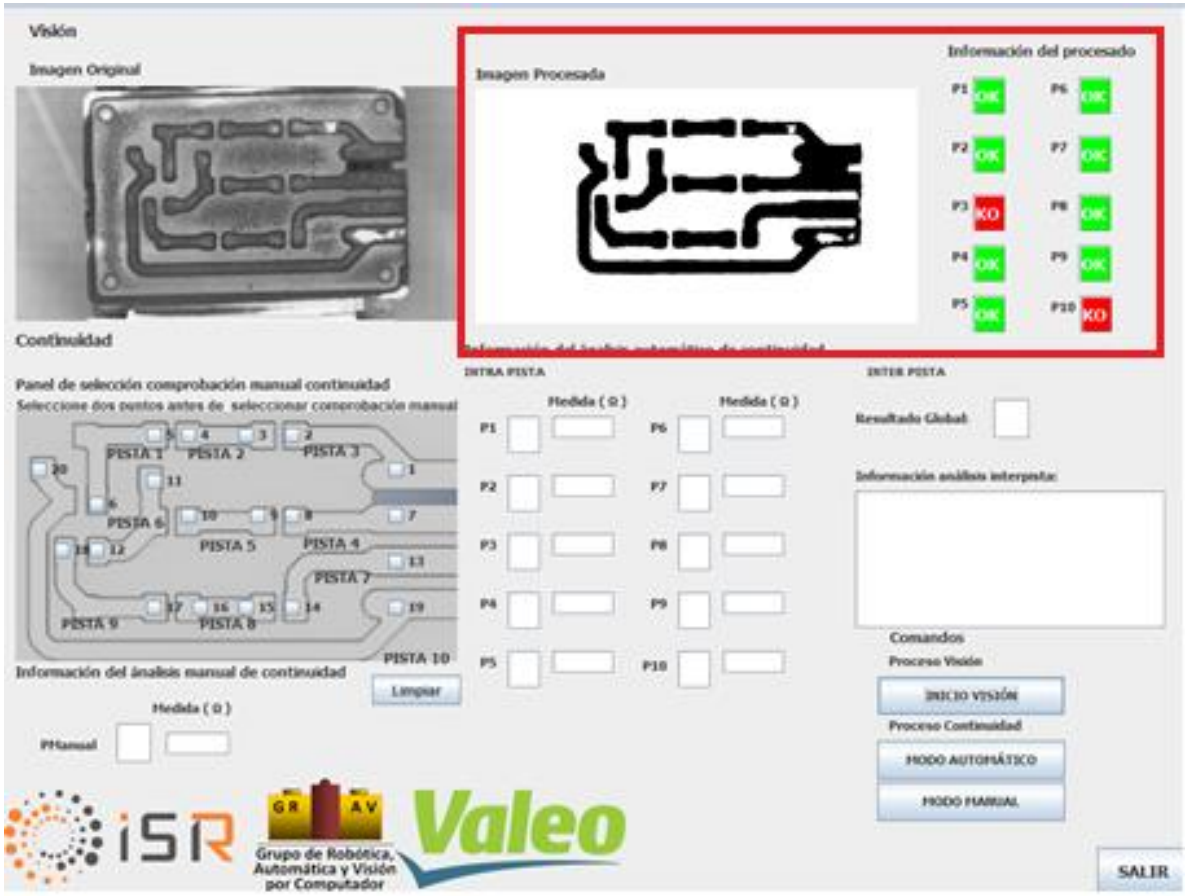



Figura 6.12. Resultado de procesado.

El operario debe trasladar la PCB al set-up de análisis de continuidad. Una vez que la placa está en la posición correcta el operario debe pulsar a 'MODO AUTOMÁTICO' (botón que aparece señalado en la Figura 6.13) para que dé comienzo el test eléctrico.

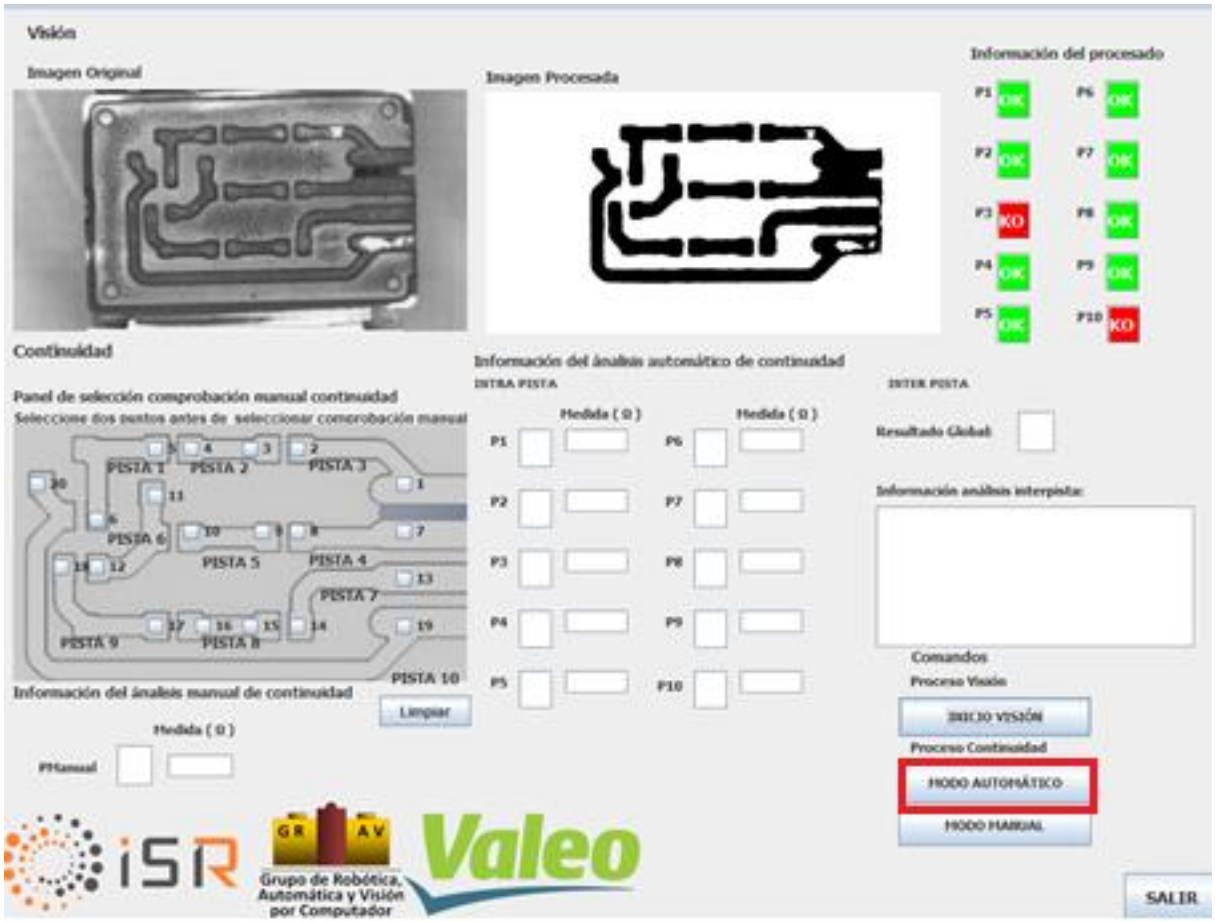



Figura 6.13. Inicio de análisis de continuidad.

.

Cuando se han realizado todos los análisis de las pistas que tenemos en el archivo de configuración, los resultados del análisis de continuidad se muestran en nuestra interfaz. Un ejemplo de estos resultados lo podemos observar en la Figura 6.14

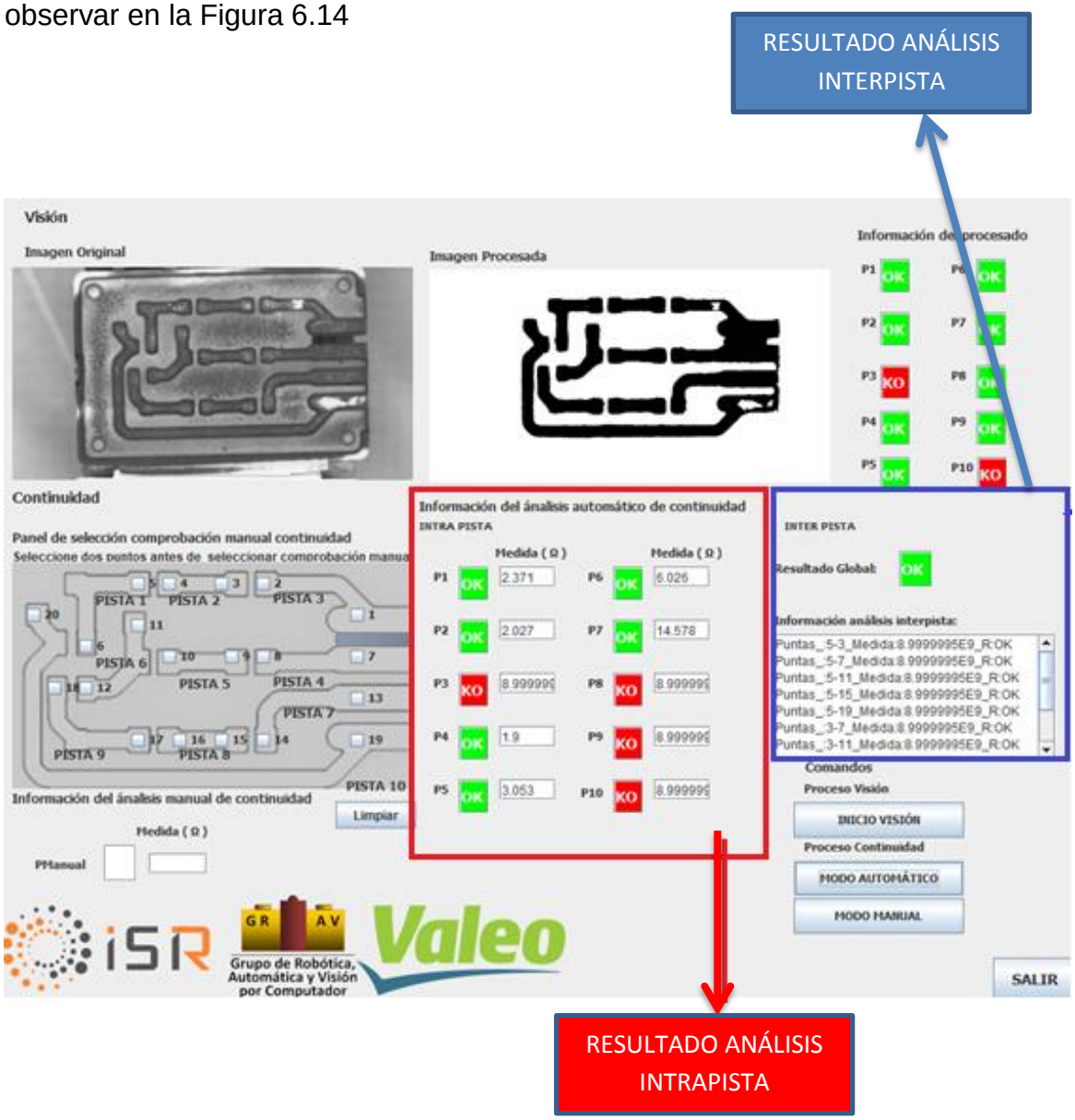



Figura 6.14. Resultado test de continuidad.

En nuestra aplicación tenemos la opción de hacer una medida manual, para ello el operario deberá seleccionar los puntos entre los cuales queremos hacer el test eléctrico. Sólo se podrá seleccionar dos puntos.

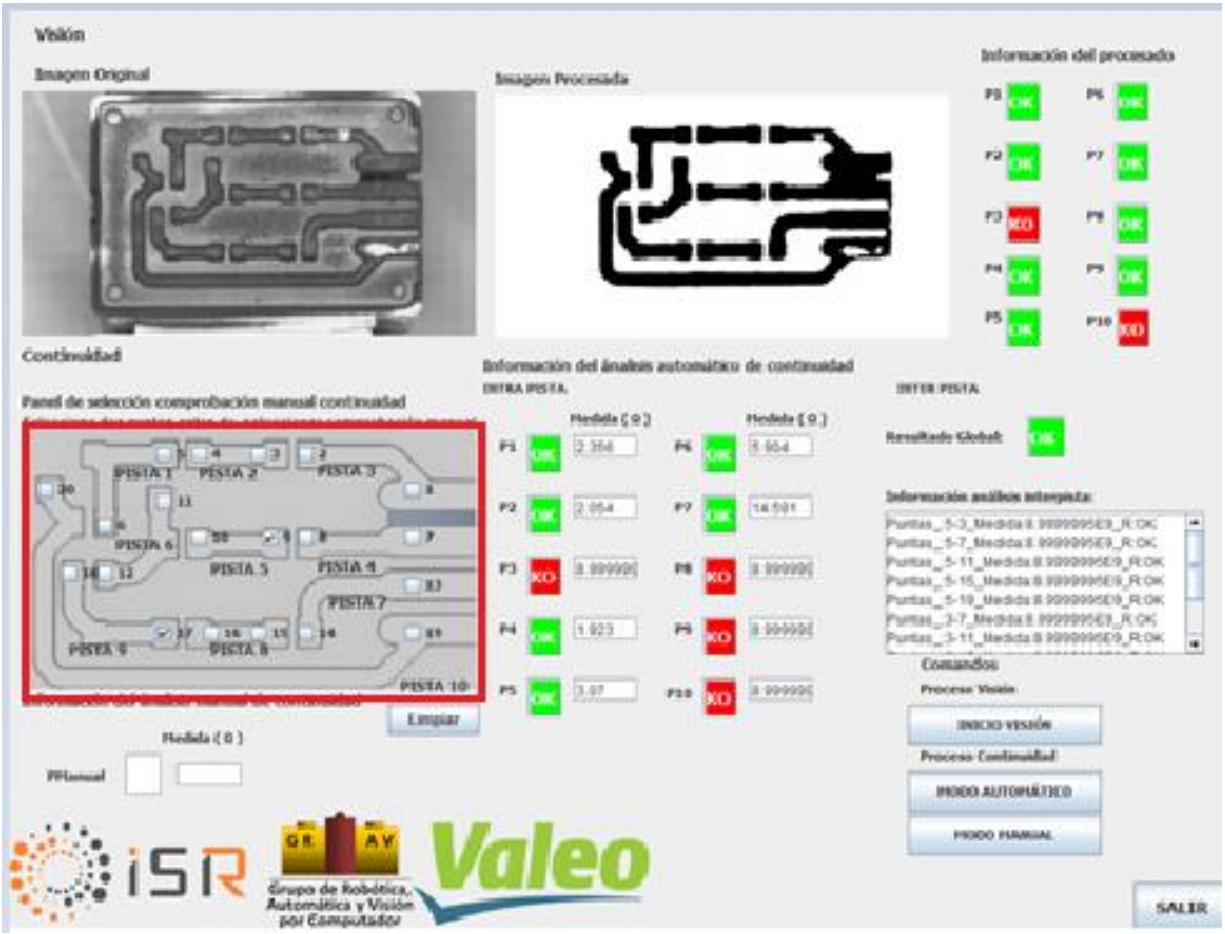



Figura 6.15. Selección manual de las puntas.

Una vez que el operario ha escogido los puntos, pulsará a 'MODO MANUAL', botón que aparece señalado en la Figura 6.16.

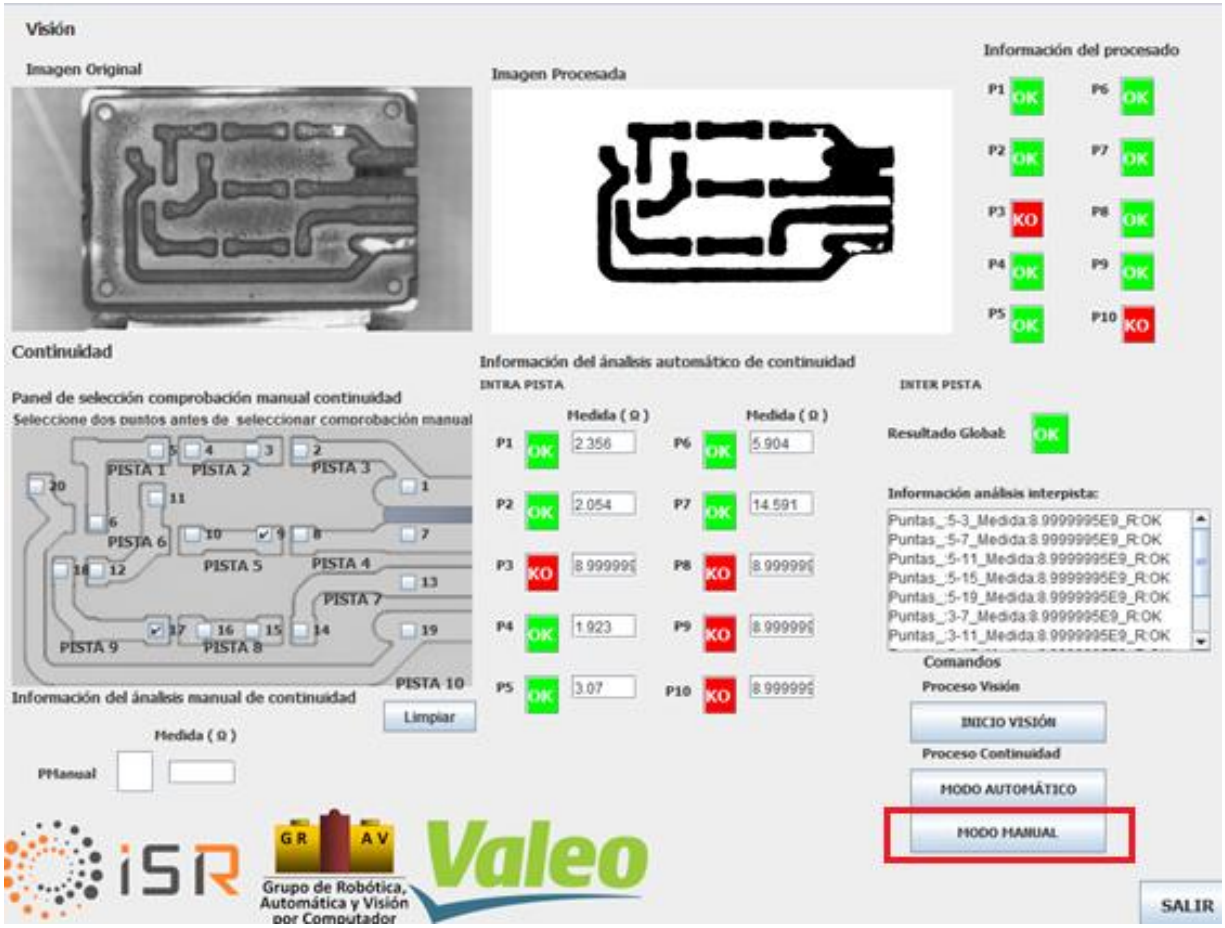



Figura 6.16. Inicio análisis manual de continuidad.

Cuando se haya hecho la medida se mostrará un resultado como vemos en la Figura 6.17.

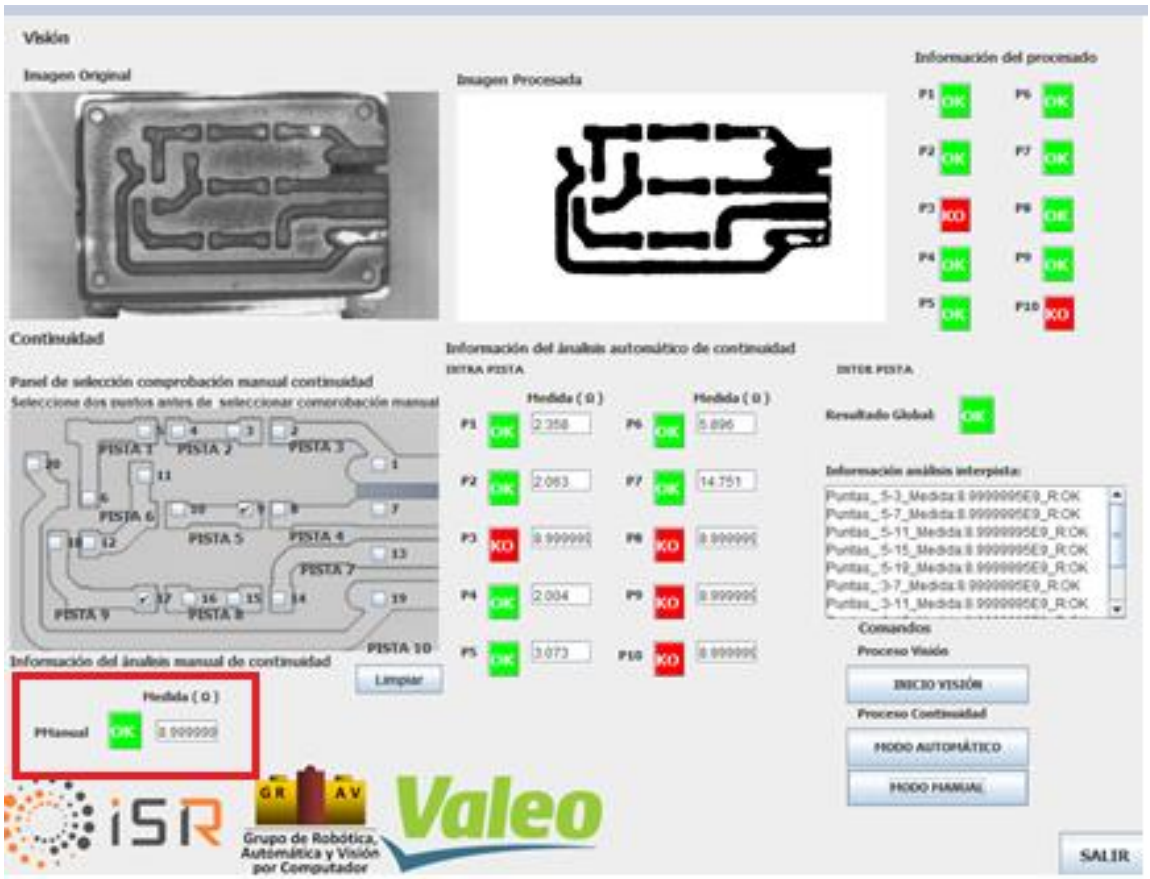



Figura 6.17. Resultado del análisis manual de continuidad.

Si el operario desea hacer una nueva medida manual, deberá quitar los puntos marcados anteriormente, posteriormente deberá pulsar el botón 'Limpiar' y por último seleccionar la nueva pareja de puntos que desea comprobar.

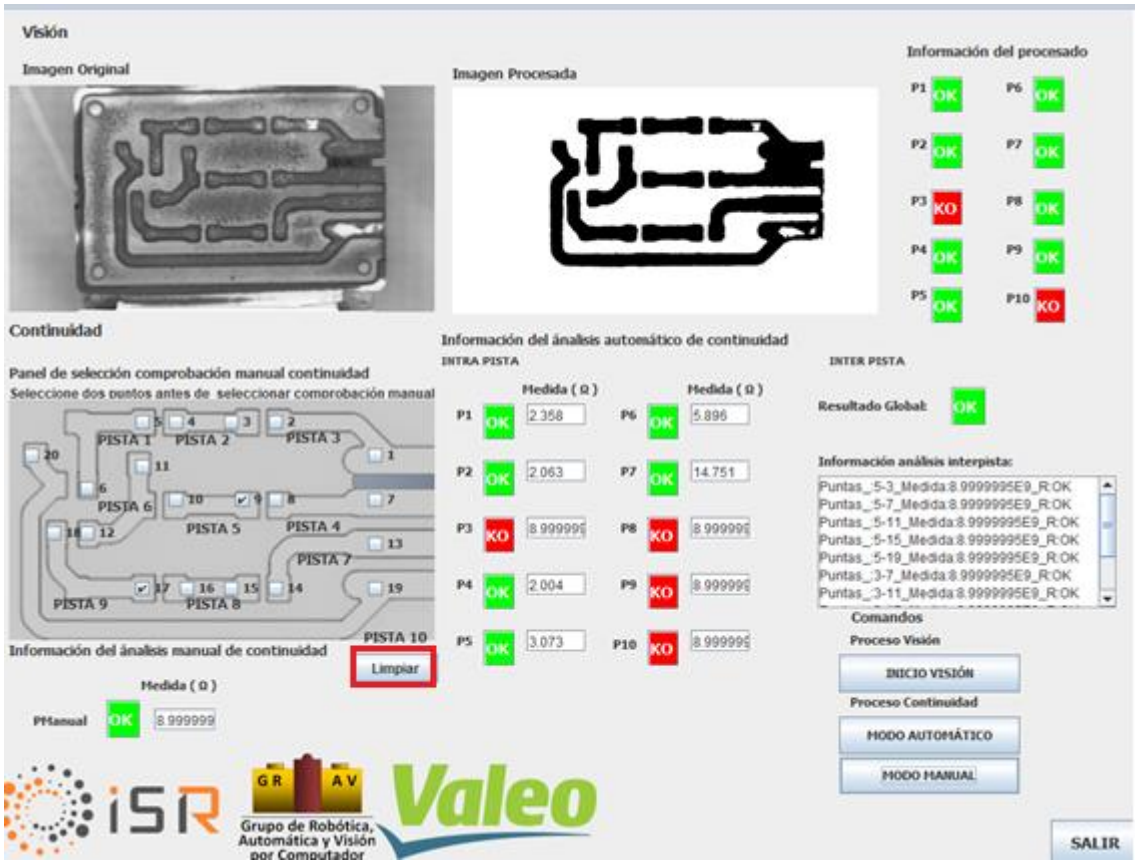



Figura 6.18. Botón de limpiar.

## ANEXO 3.-Esquemas electrónicos.