industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Sistema integrado de control de calidad.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Emilio Molina Cava**
Director: Jorge Juan Feliu Batlle
Codirector: Pablo Alejandro Martínez Ruiz

Cartagena, mayo de 2020

Autor	Emilio Molina Cava.
E-mail del Autor	emiliomoca@hotmail.es
Director	Jorge Juan Feliu Batlle.
E-mail del Director	jorge.feliu@upct.es
Codirector	Pablo Alejandro Martínez Ruiz.
E-mail del Codirector	pablo.martinez@upct.es
Título del TFG	Sistema integrado de control de calidad.

Resumen

El presente TFG trata del desarrollo de una célula de fabricación flexible capaz de realizar el control de calidad y clasificación de piezas fabricadas.

El sistema integra una cámara de visión artificial encargada de detectar tanto la incorporación de una pieza como su geometría superior y color, un brazo robot semi-industrial cuya función es trasladar la pieza por los diversos puntos de control y clasificación, un autómata programable encargado principal de la automatización del proceso y una maqueta con diversa instrumentación electrónica.

Titulación	Grado en Ingeniería en Tecnologías Industriales.
Departamento	Automática, Ingeniería eléctrica y tecnología electrónica.
Fecha de Presentación	8 de Junio de 2020.

AGRADECIMIENTOS.

Quiero dar las gracias a mis directores Jorge Feliu y Pablo Martínez por darme la oportunidad de hacer un trabajo que englobara todos aquellos campos de tecnología que despiertan mi curiosidad y que este trabajo no se tratara de uno más. Así como por su constante comprensión y disponibilidad ante las circunstancias que nos hemos encontrado durante la realización de este trabajo.

Quiero agradecer a Mari Francis su compañía y apoyo durante esas tardes de resiliencia en nuestros trabajos final de estudios y a Gonzalo y Adrián por los constantes ánimos y confianza que siempre me brindan.

Dar las gracias a mis hermanas, las cuales también han realizado sus particulares sacrificios personales en pro de que pudiera finalizar mis estudios.

Por último y sobre todo, agradecer a mis padres por el gran esfuerzo y las continuas fuerzas que me han brindado día a día durante el camino que con este trabajo finaliza y sin los cuales no habría sido posible realizar.

INDICE

FIGURAS

Capítulo 1. INTRODUCCIÓN.

1.1. Descripción.

La tecnología actual respecto al control de procesos industriales esta tan evolucionada que ha hecho posible que las líneas de producción caminen hacía una total automatización. En virtud de las inquietudes que este campo de la tecnología provoca en mí surge la motivación de solicitar este trabajo, que se basa en realizar una aproximación de la tecnología presente en las líneas de producción automatizadas modernas que existen en la actualidad. Esta aproximación abarca el caso de una sección de control de calidad de las piezas producidas por la línea, que consistirá en la detección y recepción por el sistema, control de calidad y clasificación de las piezas.

El proceso de control de calidad consistirá en la recepción de la pieza producida, la cual será detectada por una cámara de visión artificial a modo de sensor óptico y una vez detectada se realizará un análisis visual del color y geometría superficial de la pieza. Posteriormente, un brazo robótico desplazará la pieza hasta una nueva posición en una maqueta provista por el instrumental necesario para realizarle un análisis de la altura de la pieza, completando así sus datos geométrico. Finalizada esta última parte del control de calidad, la pieza será desplazada y clasificada en base a cierta consigna, recurriendo nuevamente al brazo robot. Mientras este ciclo se está realizando, se tiene en cuenta la posibilidad de que otra pieza se aproxime al sistema, de tal forma que una vez el brazo desplaza la primera pieza de la zona de recepción, la cámara vuelve a entrar en acción para poder detectar esta segunda pieza. Si esto sucede antes de la finalización del ciclo en proceso la cámara dejará de realizar su función de detección.

El sistema puede entrar en parada en cualquier momento, aunque se seguirá teniendo en consideración el estado del sistema previo a la parada, que será retomado, y si se había detectado la posible presencia de una segunda a expensas de ser confirmada.

Por tanto, este trabajo combinará varios campos de estudio como son la visión artificial, la robótica y la automatización.

1.2. Equipos que conforman el Sistema.

Como comentamos anteriormente, los equipos que conforman nuestro sistema y que nos proveerán de las funcionalidades necesarias para la realización del control de calidad de la pieza serán:

1. Cámara IDS uEye modelo UI-1540xSE-C con conexión USB y un sistema de iluminación y soporte para la cámara, que nos proporcionará una imagen apropiada a la cual aplicar tratamientos de visión artificial. Para el procesamiento de la imagen usaremos la biblioteca de imagen EmguCV.
2. Dobot Magician, un brazo robot semi-industrial con conexión vía USB y wifi, al cual le daremos instrucciones para realizar desplazamiento de nuestra pieza de estudio.
3. Una maqueta a pequeña escala, provista fundamentalmente por:
    a. Panel de mando.
    b. Placa de Expansión.
    c. Sistema de medición:
        i. Sensor Capacitivo.
        ii. Sensor Inductivo.
        iii. Dos Sensor Magnéticos.
        iv. Pistón y válvula de accionamiento 5/2 monoestable.
        v. Potenciómetro.
4. Un PLC s7-1200 el cual estará conexionado con la instrumentación de la maqueta y que será el principal responsable de la automatización del sistema.

Figura 1.1 Sistema Integrado de Control de Calidad.

1.3. Objetivos.

Los objetivos que lograr en este trabajo, para proveer a nuestro sistema de las capacidades necesarias para adquirir la información dimensional y de pigmentación de la pieza para su posterior clasificación son:

- Puesta a punto y conexionado del PLC.
- Comprobación del correcto funcionamiento de la maqueta y calibración del sistema de medición compuesto por pistón y potenciómetro solidario que nos aportará la medida de altura de la pieza.
- Programación del PLC para la automatización del proceso de control de calidad del sistema, abarcando desde la llamada a las funciones de la cámara para el análisis visual, el envío de ordenes al brazo para su interacción con el sistema y el control de los componentes de la maqueta.
- Estudio de las capacidades del brazo y sus posibilidades de comunicación.
- Estudio de las capacidades de la cámara y de la biblioteca de imagen EmguCV con la finalidad de detectar tanto geometría como el color de las piezas.
- Encontrar un modo de comunicación de todos los componentes del sistema con el PLC.

Capítulo 2. ESTADO DEL ARTE.

La tecnología de hoy en día está tan evolucionada que ha hecho posible que las líneas de producción caminen hacía una total automatización. Hace unas décadas estas líneas de fabricación estaban intervenidas en gran nivel por interacción humana. Tras el avance en tecnología, la producción introdujo una mayor automatización del proceso, recluyendo el uso de mano de obra para inspección o tareas más complejas que no son fáciles de automatizar. Hoy en día, con las mejoras en estas tecnologías y el auge de la inteligencia artificial, ya existen muchas instalaciones industriales de producción que tan solo requieren trabajadores de alta cualificación en planta con la finalidad de corroborar el correcto funcionamiento de la línea de producción, su mantenimiento o introducción de mejoras en el proceso automatizado. Estas instalaciones tan avanzadas alcanzan un grado de automatización tal que solo requieren la aportación de la materia prima a procesar y no requieren más intervención hasta el punto de transporte del producto, pues la automatización logra alcanzar tanto la comunicación como el propio sistema de almacenamiento.

A continuación, con la finalidad de adquirir un enfoque global de los campos de la tecnología que abarca este trabajo, vamos a definir los campos de estudio y describir su estado. Comenzaremos con los autómatas programables, la robótica (centrándonos en los brazos robóticos) y la visión artificial, para acabar con un enfoque más concreto con ejemplos en la industria de sistemas automatizados usando visión artificial en líneas de producción.

2.1. PLC o Autómata Programable.

Un controlador lógico programable (PLC) es un sistema de control industrial por computadora que monitoriza de forma continua el estado de los dispositivos de entrada para en base a un programa configurado controlar el estado de los dispositivos de salida.

Los controladores lógicos programables son una solución de control flexible, robusta y de gran adaptabilidad.

Casi cualquier línea de producción o proceso puede ser mejorado utilizando este tipo de sistema de control. El mayor beneficio de usar un PLC es la capacidad de reprogramación y de replicar el proceso mientras se registra y transmite información.

Los PLCs se caracterizan respecto de los PCs industriales, microcontroladores y otras soluciones de control industrial por:

- I/O. Módulos de entrada y salida que conectan el PLC con el resto de la máquina y proporcionan información a la CPU y que activan resultados específicos.
- Comunicaciones. Los PLCs ofrecen una gama de puertos y protocolos de comunicación para garantizar la comunicación del PLC con otros equipos.
- HMI. Poseen una interfaz que le permita al operador interactuar con él y revisar e introducir información en tiempo real.

Figura 2.1 Autómatas Programables en la industria.

Los PLCs se programan utilizando un software de aplicación en ordenadores, que representan la lógica en forma gráfica. Aunque diagrama de contactos es el lenguaje de programación de PLC más utilizado, encontramos otros como:

- Diagrama de contactos. La lógica de escalera tradicional es un lenguaje de programación gráfico.
- Diagrama de bloques de función (FBD). Un lenguaje gráfico para representar los flujos de señales y datos mediante bloques de función reutilizables.
- Texto estructurado (ST). Un lenguaje de texto de alto nivel que fomenta la programación estructurada parecida a PASCAL.
- Secuencial Function Chart (SFC). Un método que coordina tareas de programación grandes y complicadas en tareas más pequeñas y manejables de forma estructurada.
- Lista de instrucciones (IL): Un lenguaje de bajo nivel tipo ensamblador.

2.2. Robótica (Brazo-Robot).

La robótica es una ciencia que combina varias ramas tecnológicas con el objetivo de diseñar maquinas robotizadas que sean capaces de realizar tareas automatizadas o imitando el comportamiento humano o animal. Mientras que un robot lo podemos definir como una entidad autómata compuesta por mecánica artificial y un sistema electromecánico.

Para el diseño de un robot se emplean varias ciencias como son: mecánica, electrónica de control, informática y computación. Todas estas especialidades conforman la Ingeniería robótica junto otras ramas a considerar como son el álgebra, la automática o las máquinas de estados.

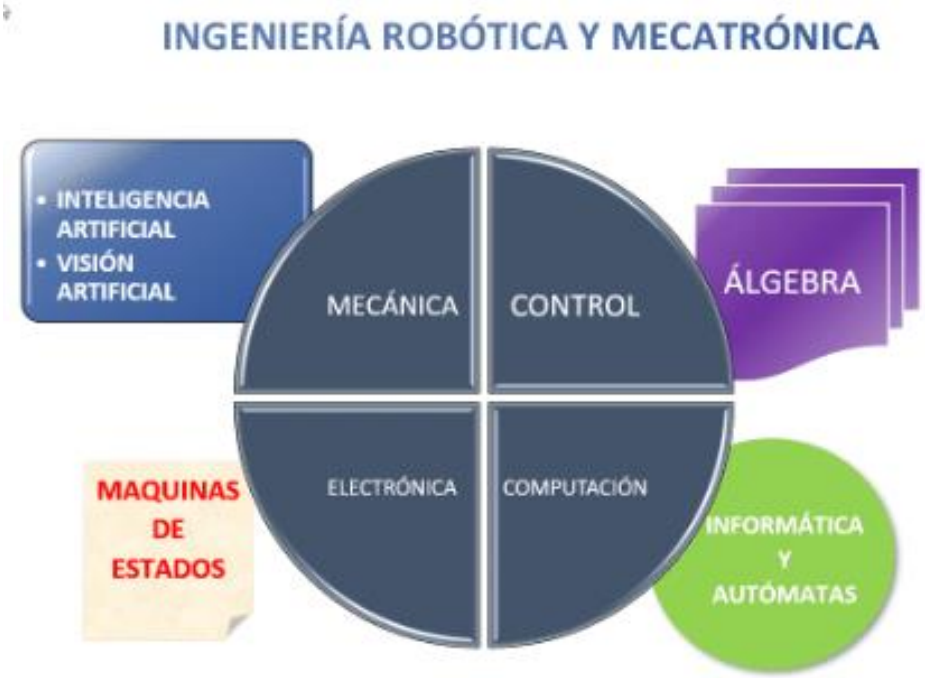


Figura 2.2 Ciencias que conforman la mecatrónica.

Estos equipos están implantados en las cadenas de montaje industriales durante más de cuarenta año. Pero ha sido en estos últimos cuando han experimentado una gran desarrollo gracias a la innovación que se ha producido en tecnologías asociadas a la robótica.

Estos robots destacan ser muy versátiles gracias a su capacidad de giro y desplazamiento. Son capaces de realizar múltiples operaciones típicas de líneas de producción como son: soldaduras, montajes, ensamblajes, ajustes…

Un brazo robótico consiste en un brazo mecánico que se puede programar y cuyas funciones principales intenta replicar las de un brazo humano. Hay varios tipos dependiendo de su utilidad industrial:

- Robot cartesiano: este robot se caracteriza por que sus ejes coinciden con los tres ejes cartesianos. Se emplea soldadura, ensamblado y manipulación de objetos.
- Robot esférico o polar: los ejes de este robot forman un sistema polar de coordenadas. El primero de ellos fue el robot Unimate instalado en una de las fábrica coches de General Motors.
- Robot articulado: presenta una mayor capacidad de giro y por consiguiente mayor precisión, por lo que es destinado a tareas más complejas.
- Robot cilíndrico: sus ejes forman un sistema de coordenadas de círculos concéntricos que le permiten efectuar tareas como la manipulación de máquinas.
- Robot SCARA: Es un robot con dos articulaciones rotatorias paralelas, que permiten que pueda hacer trabajos de “pick and place” (coger y dejar) con materiales pesados.
- Robot paralelo: su uso principal está en plataformas móviles para la simulaciones de vuelo debido a su alto nivel de rotación y movilidad.

Gracias a esta gran variedad de configuraciones que le otorgan una gran flexibilidad para adaptarse es que el brazo robótico es una innovación tecnológica que está presente en multitud de industrias en la actualidad.

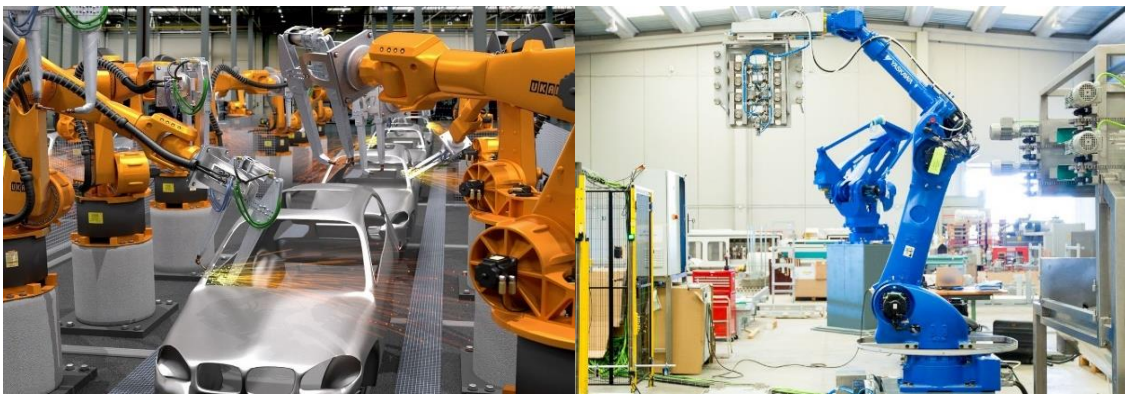


Figura 2.3 Línea fabricación General Motors y Brazo Robótico de carga.

2.3. Visión Artificial.

La visión artificial es una ciencia moderna para el procesamiento y análisis de imágenes. Esta tecnología se puede implementar en multitud de aplicaciones industriales y no industriales para brindar a estas de una mayor automatización.

La aplicación de esta tecnología se traduce en sistemas de visión artificial que se componen por cámaras industriales con sensores digitales y ópticas especializadas para la adquisición de imagen. Estos sistemas aplicados a la robótica facilitan tareas industriales tan importantes como realizar procesos de inspección, supervisión y controles de calidad con gran exactitud analítica.

Aunque los límites entre las tipologías de productos de visión artificial estén muy poco definidos, podemos hablar de diversas categorías:

- Sensores de visión: Se trata de sensores más sofisticados que los tradicionales sensores fotoeléctricos, aunque cuentan limitaciones para la toma de decisiones.
- Cámaras inteligentes y sistemas de visión integrados: Son la tecnología más avanzada y destacan por su capacidad de procesamiento.
- Sistemas de visión avanzados: Son muy similares a los sistemas de visión integrados pero cuentan con un hardware más sofisticado y completo. Su aplicación está pensada para tecnologías y máquinas de mayor complejidad.

Figura 2.4 Aplicaciones de Visión Artificial.

2.4. Aplicaciones de Sistemas de Control de Calidad basados en Visión Artificial.

La implementación más habitual de un sistema de visión artificial se trata de la realización de un control de calidad del producto fabricado, que puede abarcar desde la comprobación de una serie de especificaciones, la detección de posibles defectos, etc.

"El control de calidad en entornos industriales es el seguimiento de los procesos de una organización con el objetivo de mejorar la calidad de un determinado producto".

A continuación se muestran algunos ejemplos de la implementación de la tecnología comentada en los apartados anteriores para el control de calidad en la industria.

- Detección de nivel de llenado de botellas de cerveza.

Algunas empresas de producción de cerveza utilizan sistemas para la inspección de nivel de llenado en las botellas. Cada botella de cerveza pasa a través de un sensor de inspección, que a su vez activa un sistema de visión que utiliza una luz estroboscópica para tomar una fotografía de la botella. Esta imagen es procesada por el software de visión y emite una respuesta de aprobación o fallo basada en el nivel de llenado de la botella. Si el sistema detecta un nivel de llenado incorrecto indica a un desviador que rechace la botella.

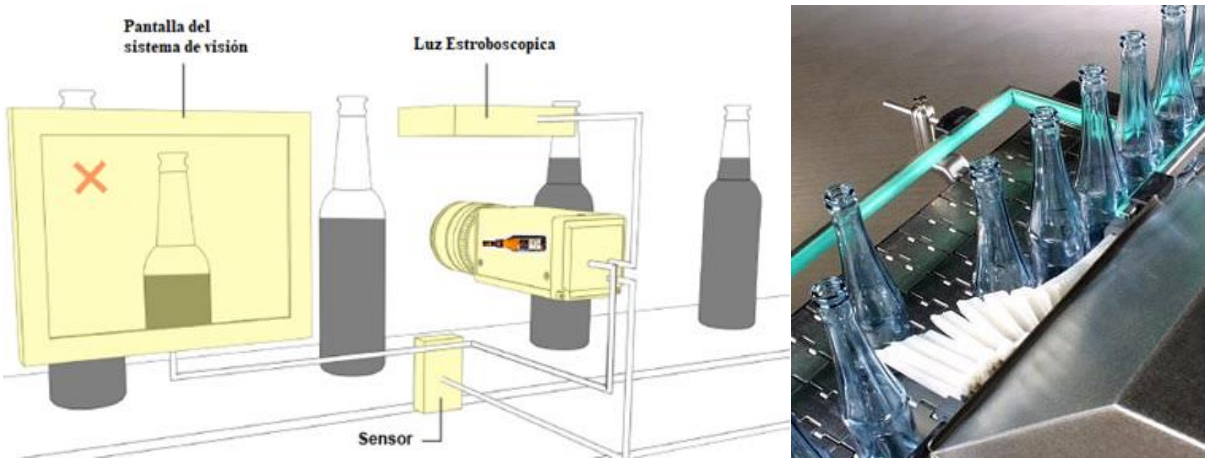


Figura 2.5 Sistema de visión para control de nivel de llenado.

- Sistema de inspección de tapones de sellado de botellas.

Otra tipo de aplicación de los sistemas de visión artificial en el mismo campo anterior de embotellado sería el del sistema de inspección del correcto cierre de botellas mediante tapones. Las botellas en cuestión pasarían por delante de una cámara de visión artificial para capturar una imagen de forma similar al caso de llenado de botellas. En este caso, la potencia de procesamiento necesaria es superior pues requiere de un análisis más complejo de la imagen para detectar la presencia del tapón y la posición en que se encuentra. El proceso que seguir tras un fallo podría ser el mismo que el anterior, rechazando la botella mal sellada.

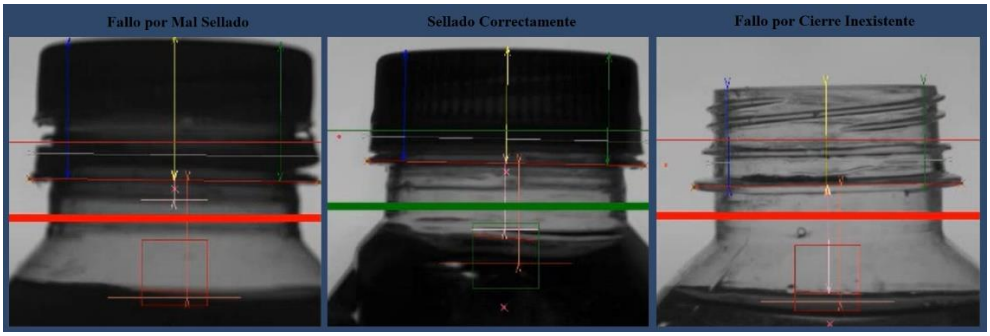


Figura 2.6 Sistema de Inspección de Sellado de Botellas.

- Sistema de túnel de inspección de defectos en carrocería (Fabrica de Ford).

Para terminar este capítulo, vamos a describir con mayor detalle un sistema de visión artificial más complejo como es el siguiente sistema de inspección automatizada de defectos en carrocerías y monitorización por visión artificial.

Este sistema se compone por un sistema de visión conformado por varias cámaras, una estructura mecánica de robot cartesiano que sustenta al sistema anterior y que desplaza el sistema de iluminación. Y por último, un subsistema de pantallas donde los defectos son resaltados.

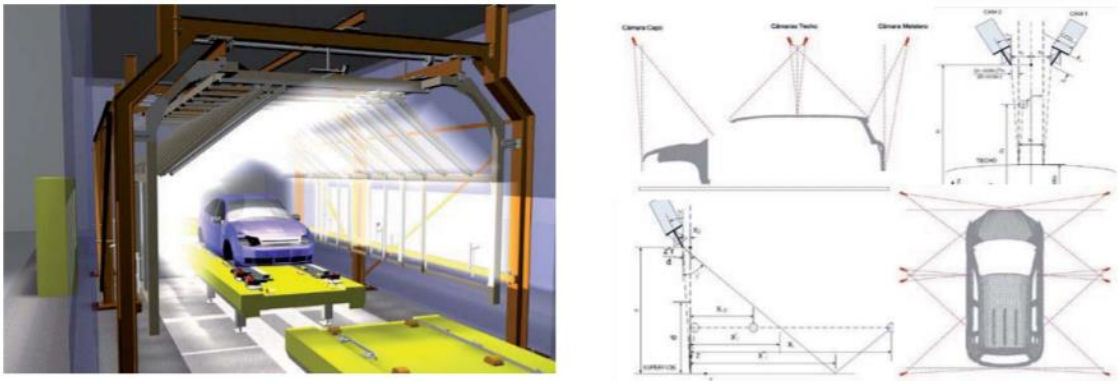


Figura 2.7 Equipos del Sistema de inspección de carrocería.

Cada cámara inspecciona una parte de la carrocería y la información tomada es transmitida a un ordenador. El subsistema de monitorización proporciona información sobre los defectos a los operarios del área de pulido.

Mediante este sistema en cuestión de 10 segundos es capaz de recabar toda la información y de detectar más del 90% de los defectos no detectados con respecto a una inspección manual.

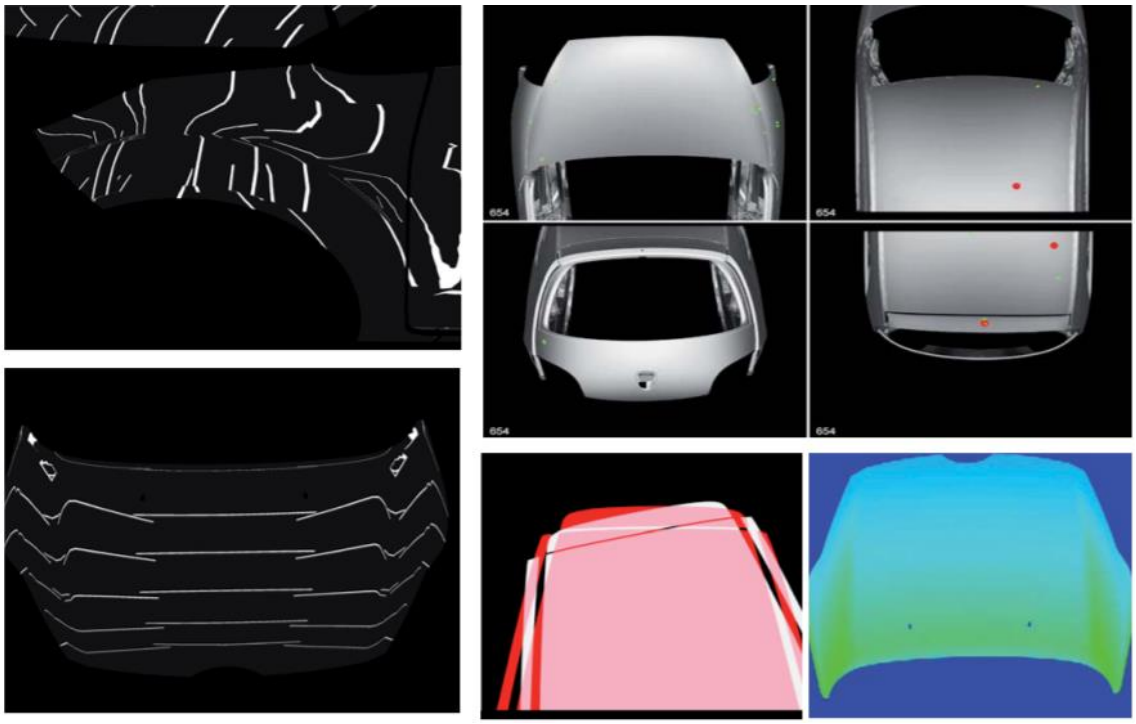


Figura 2.8 Visualización de resultados tras la inspección de la carrocería.

Al final del proceso, los defectos son mostrados en pantalla para su reparación, almacenándose al mismo tiempo los defectos y su tipología.

En conclusión, como hemos podido comprobar los sistemas de control de calidad basados en visión artificial pueden suponer soluciones muy ventajosas y flexibles para tareas de inspección visual en procesos industriales de fabricación. Además, aportan a nuestros sistemas de fabricación una gran velocidad de procesamiento en línea y un nivel elevado de fiabilidad y rendimiento logrando alcanzar un gran nivel de automatización.

Capítulo 3. COMPONENTES DEL SISTEMA.

En este capítulo pasaremos a describir con mayor detalle cada uno de los sistemas con los que cuenta este sistema de control de calidad.

3.1. PLC Siemens s7-1200.

El PLC con el que trabajamos es un S7-1200, montado sobre una carcasa previamente cableada para facilitar la conexión de E/S digitales, así como la conexión a alimentación de 0 y 24 V en DC.

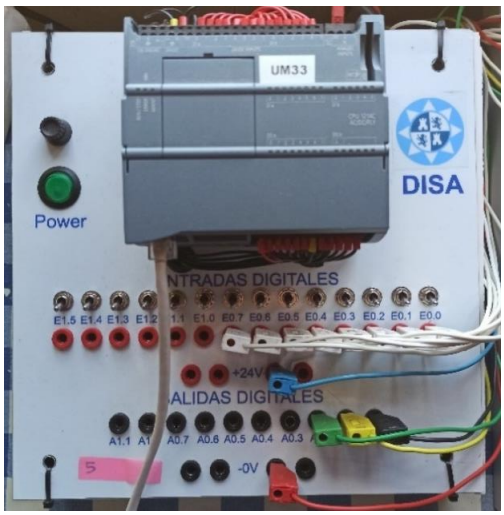


Figura 3.1 Montaje del PLC.

La CPU incorpora un puerto PROFINET y cuenta con otros módulos de comunicación que están disponibles para la comunicación en redes RS485 o RS232.

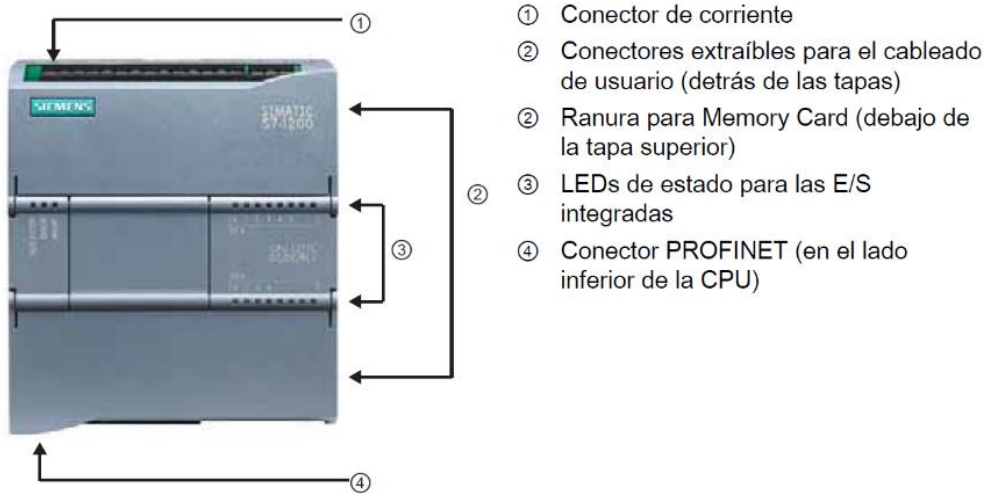


Figura 3.2 PLC S7-1200 y conexiones.

La gama S7-1200 cuenta con una gran variedad de módulos de señales y Signal Boards que permiten ampliar las prestaciones de la CPU.

Módulo		Sólo entradas	Sólo salidas	Entradas y salidas
Módulo de señales (SM)	Digital	8 entradas DC	8 salidas DC 8 salidas de relé	8 entradas DC/8 salidas DC 8 entradas DC/8 salidas de relé
		16 entradas DC	16 salidas DC 16 salidas de relé	16 entradas DC/16 salidas DC 16 entradas DC/16 salidas de relé
	Analógico	4 entradas analógicas 8 entradas analógicas	2 salidas analógicas 4 salidas analógicas	4 entradas analógicas/2 salidas analógicas
Signal Board (SB)	Digital	-	-	2 entradas DC/2 salidas DC
	Analógico	-	1 salida analógica	-
Módulo de comunicación (CM) • RS485 • RS232				

Figura 3.3 Módulos de señales y Signal Boards.

Siemens cuenta para sus PLCs con el software TIA PORTAL, que ofrece un entorno de programación para desarrollar, editar y observar la lógica del programa necesaria para controlar la aplicación.

3.1.1. Configuración del PLC.

Contamos con el software TIA PORTAL v.13 en el cual configuraremos su CPU 1214C AC/DC/Rly, dicha CPU se alojará en el slot 1, cuyo rack será 0 y la cual quedará configurado con una IP 192.168.0.133. Para la comunicación entre los sistemas integrantes de este proyecto usaremos la subred 192.168.0.xxx.

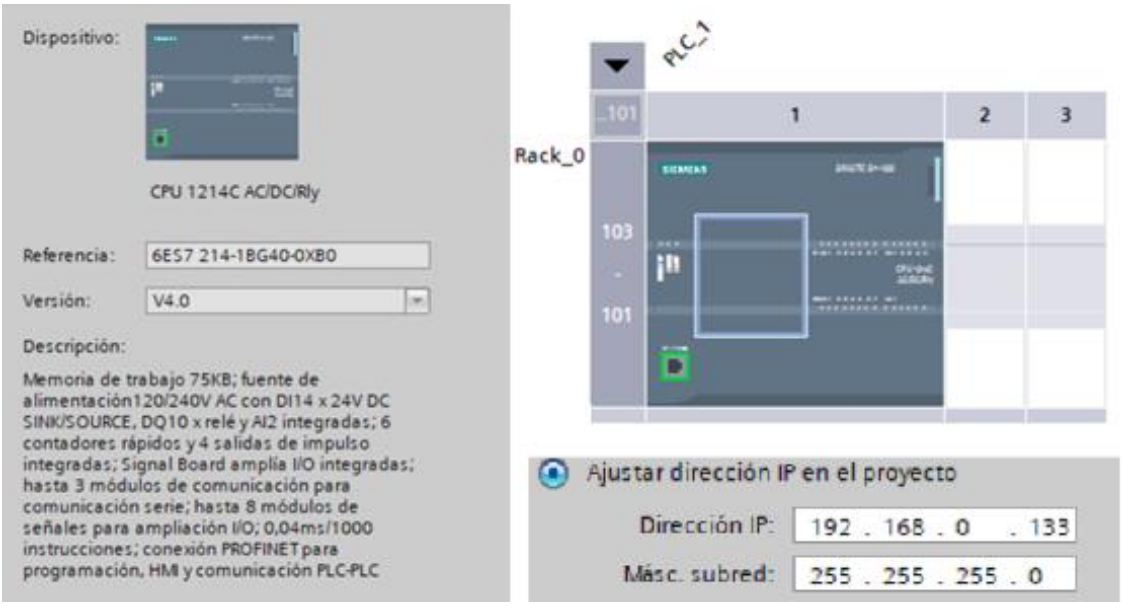


Figura 3.4 Configuración del PLC.

3.2. Maqueta del proceso productivo.

La maqueta a pequeña escala con la que contamos es la unidad funcional "Estación de Reconocimiento y Medición", la cual es un componente del sistema didáctico modular de fabricación flexible de GRV.

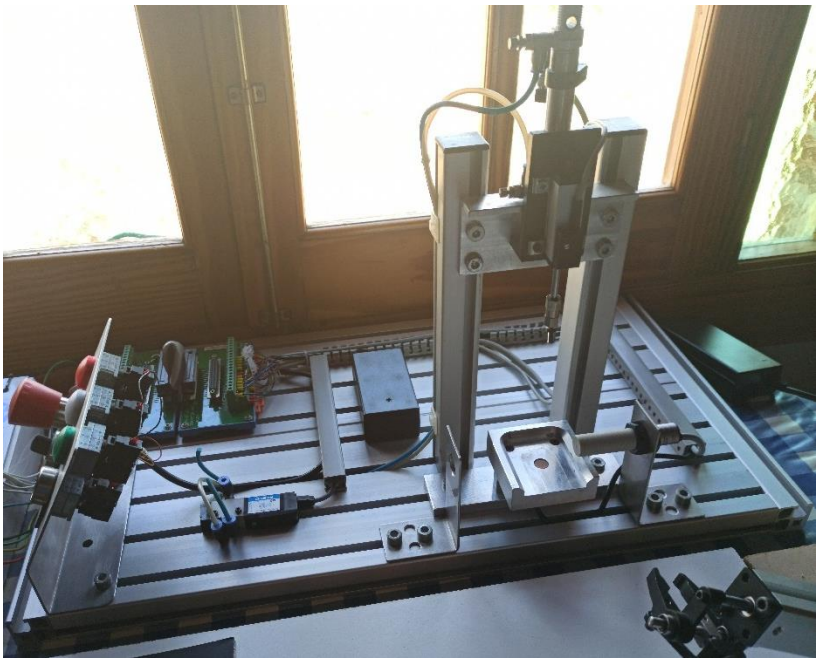


Figura 3.5 Maqueta.

Esta dispone de una celda de reconocimiento con un porta-piezas en el que se coloca la pieza a medir. Cuenta con hasta 3 sensores, un sensor inductivo, un sensor capacitivo, y un potenciómetro lineal solidario a un cilindro neumático que combinan sus informaciones para proporcionar la altura de las piezas. El potenciómetro lineal de 25 mm de carrera es solidario a un cilindro neumático accionado por una válvula 5/2 monoestable. El resto de los sensores permanecen estáticos, si bien su campo de acción puede ser regulado sobre el mismo sensor. Además, sobre el cilindro neumático se disponen dos sensores magnéticos Reed final de carrera.

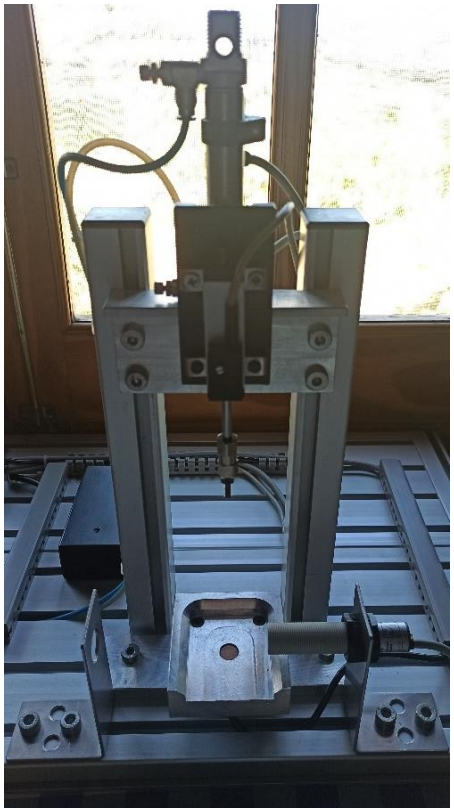


Figura 3.6 Conjunto de Medición de Altura.

Todas las entradas neumáticas están equipadas con válvulas reductoras de presión, con las cuales se puede regular las velocidades de avance y retroceso de los actuadores. La válvula FESTO que controla el cilindro neumático se alimenta a una tensión de 24 VDC.

Dispone también de un panel de control con un interruptor, un pulsador, una pulsador de emergencia con luz y dos lámparas de iluminación (leds) de diferentes colores. Además consta de una placa de expansión de 37 pines en las cuales todas las señales del módulo se encuentran replicadas para ser accesibles tanto vía conector DB-37 (para conectar a una tarjeta I/O) como mediante la bornera eléctrica de tornillos (para ser utilizada con el PLC).

Figura 3.7 Mandos de la maqueta.

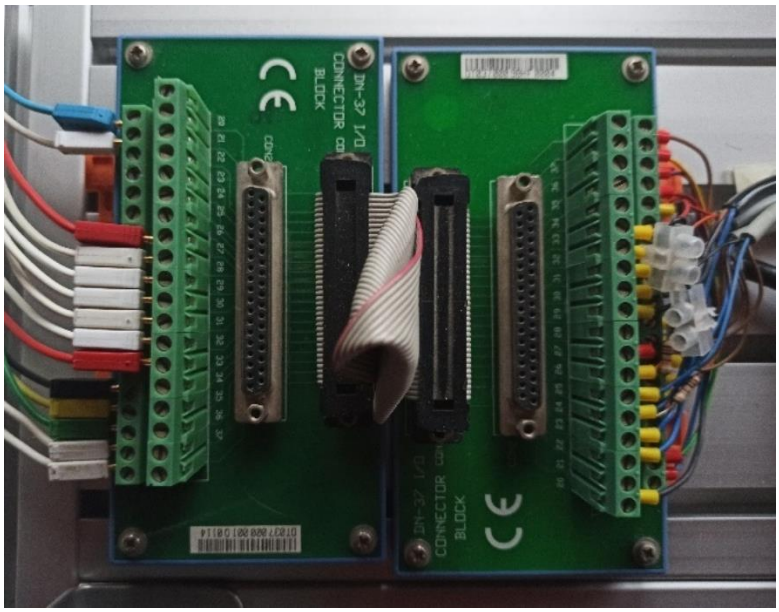


Figura 3.8 Placa de expansión.

Mostramos a continuación tanto el conexionado del cableado desde la distinta instrumentación de la maqueta a la placa de expansión como su correspondiente conexión a las entradas y salidas del PLC.

Pin	Descripción	Color del cable	Variable PLC
1	Alimentación 24 VDC General	Azul	24 V
2	24 VDC L++ (Señal emergencia)	Verde	I0.7
5	24 VDC sensor capacitivo	Marrón	
6	24 VDC sensor inductivo	Marrón	
7	24 VDC sensor magnético abajo	Marrón	
8	24 VDC sensor magnético arriba	Marrón	
9	24 VDC alimentación conversor DC/DC	Marrón	
15	Válvula neumática	Rojo/Negro	O0.0
16	Lámpara blanca	Rojo	O0.1
17	Lámpara verde	Morado	O0.2
18	Interruptor manu/auto	Amarillo	O0.5
19	Pulsador marcha	Marrón	O0.6
20	0 V para válvula neumática (general)	Negro	0 V
22	0 V para sensor capacitivo	Azul	
23	0 V para sensor inductivo	Azul	
24	0 V para sensor magnético abajo	Azul	
25	0 V para sensor magnético arriba	Azul	
26	0 V para conversor DC/DC	Naranja	
27	0 V para lámparas	Negro	
29	Sensor capacitivo	Negro	I0.1
30	Sensor inductivo	Negro	I0.2
31	Sensor magnético abajo	Negro	I0.3
32	Sensor magnético arriba	Negro	I0.4
33	Señal potenciómetro lineal	Rojo	IA0

Figura 3.9 Conexionado PLC - Placas de extensión - Instrumentos Maqueta.

Las dos conexiones entre pines en rojo son las correspondientes a los sensores magnéticos que por protección cuentan con dos resistencias de 1KΩ.

3.2.1. Puesta a punto y adecuación.

Como se indica en la introducción esta maqueta pertenecía a otro trabajo del departamento, por lo que ya estaba montada como se ha descrito. Por tanto, lo primero que se realizo fue su puesta a punto. Desde una intensiva limpieza desmontándola por completo hasta la renovación de todo el cableado del panel de control y placa de expansión, encauzamiento y revisión de todo el cableado de los distinto sensores, sustitución de los tubos de la válvula neumática que se encontraban defectuosos y reparación de la estructura a la que iba fijado pistón y potenciómetro lineal.

Por otro lado, esta maqueta estaba preparada para otro proceso y otros componentes con los que interactuaba de forma diferente a los que ahora se destina. Para ello, modificaremos la orientación del porta-piezas. También, tendremos en cuenta el rango de medición del potenciómetro lineal, que pese a que sea de 25mm podemos calibrar una cota de referencia de medida, para poder medir piezas de un tamaño superior, siempre que las variaciones de altura de las piezas se encuentren dentro del rango del instrumento.

Por último, la maqueta es conectada al PLC haciendo uso de las placas de expansión y de la interfaz sobre la que viene montado el PLC, conectando cada pin con su E/S correspondiente, así como la señal analógica del potenciómetro lineal a la entrada analógica correspondiente del PLC. Cada conexión viene referenciada en la figura 3.9.

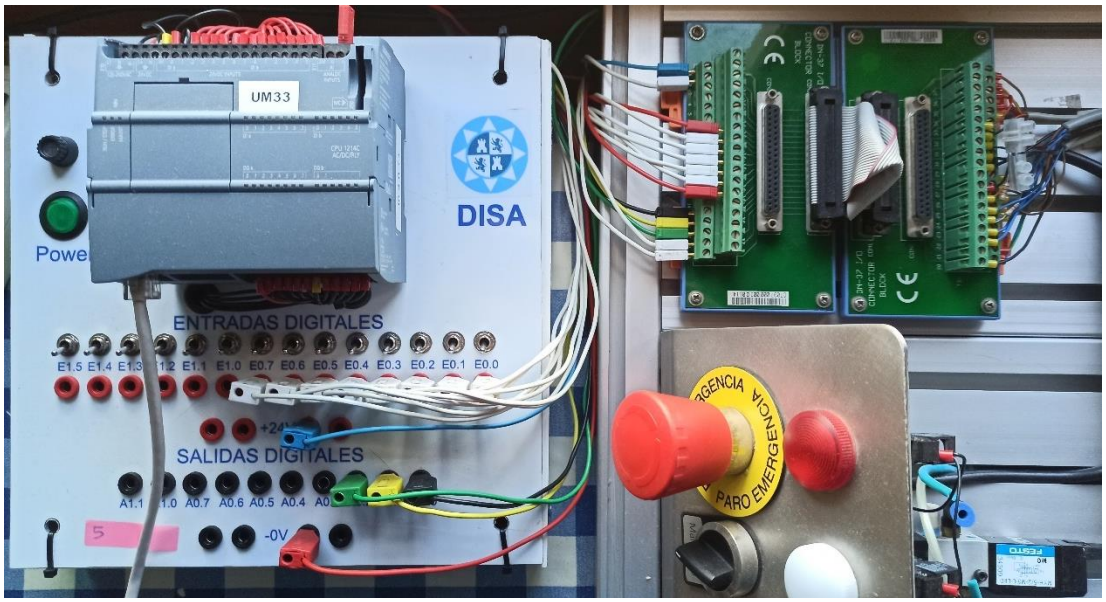


Figura 3.10 Conexionado PLC-Placa-Instrumentación.

3.3. Dobot Magician.

Dobot Magician es un brazo robótico multifuncional que posee diferentes herramientas con las cuales es capaz de realizar funciones como impresión 3D, grabado láser, escritura y dibujo. Admite el desarrollo secundario mediante varias interfaces extensibles y más de 20 lenguajes de programación.

Figura 3.11 Funciones Dobot Magician.

Este brazo robótico cuenta con una base, un brazo trasero, un antebrazo, como se muestra en la figura 3.12 y una serie de elementos finales. Su espacio de trabajo se delimita en la figura 3.13.

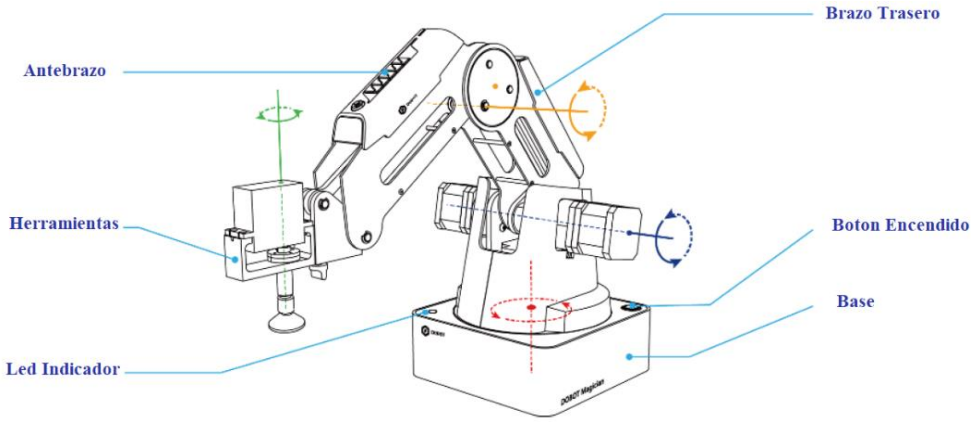


Figura 3.12 Partes de Dobot Magician.

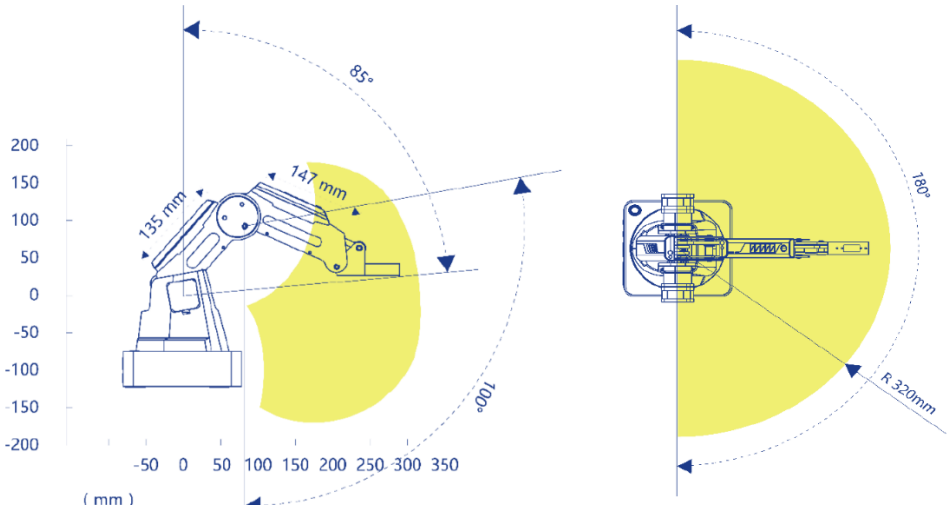


Figura 3.13 Espacio de Trabajo.

Cuenta con dos tipos de sistemas de coordenadas, uno relacionado con el giro de los ejes y otro el de coordenadas cartesianas (Figura 3.14).

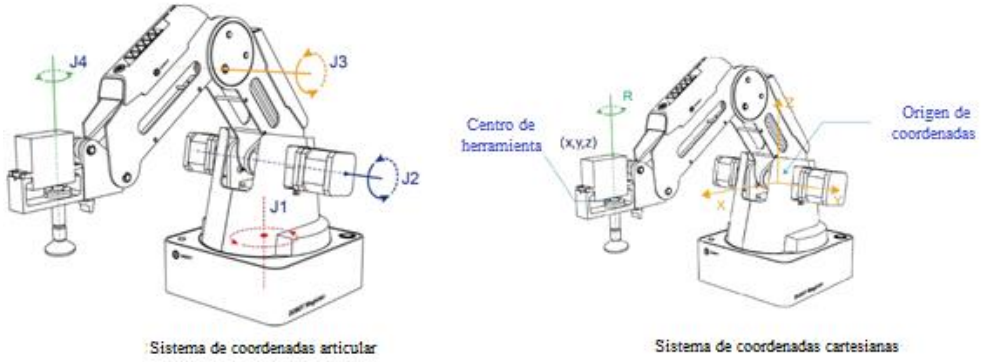


Figura 3.14 Sistemas de coordenadas DM.

- Sistema de coordenadas de articulación.

Las coordenadas están determinadas por el movimiento de la articulación. El brazo cuenta con 3 articulaciones más una extra en caso de contar con un elemento final con servo. La dirección positiva de todas las articulaciones giratorias es el antihorario.

- Sistema de coordenadas Cartesianas.

Las coordenadas están determinadas por la base. El origen es el centro de los tres motores. La dirección del eje X es la perpendicular a la base hacia adelante, la dirección del eje Y es perpendicular a la base hacia la izquierda y la dirección del eje Z es la vertical hacia arriba, que está basado en la regla de la mano derecha. Y el eje R es la actitud del servo con respecto al origen del brazo robótico con sentido positivo el antihorario.

Cuenta con 3 modos de movimiento “Jogging”, punto a punto (PTP) y en arco (ARC, determinado por 3 puntos). El modo PTP admite 3 submodalidades: MOVJ, el movimiento se ejecuta desde el punto A hasta el punto B desde el ángulo inicial hasta el ángulo objetivo, independientemente de la trayectoria; MOVL, movimiento rectilíneo independiente de la trayectoria; y JUMP el movimiento será semejante al de una puerta desde el punto A y el punto B.

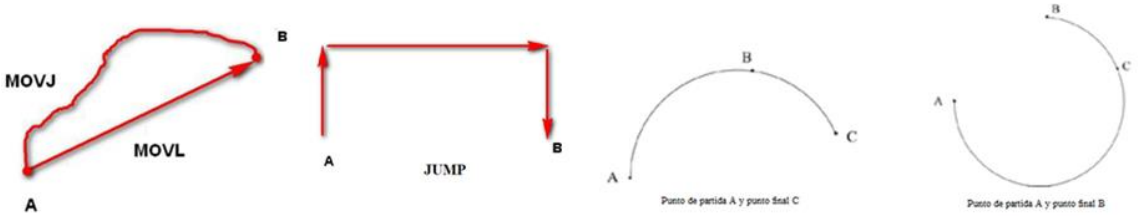


Figura 3.15 Modos de Movimiento.

- Descripción de la Interfaz

El área de interfaces de conexionado de Dobot Magician se encuentra en la parte posterior de la base y el antebrazo. En este último encontramos las interfaces para conectar los accesorios finales con los que cuenta el brazo-robot.

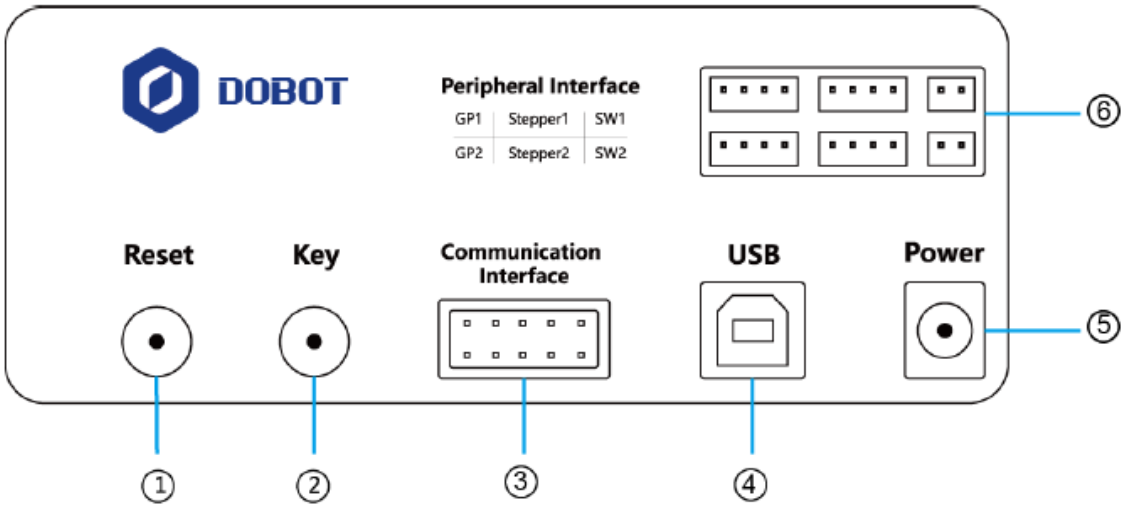


Figura 3.16 Área de Interfaces.

N.º.	Descripción.
1	Botón de reinicio: Reinicia el programa MCU.
2	Botón funcional: - Pulsación corta: comienza a ejecutar el programa sin conexión. - Pulsación 2 segundos: inicio de procedimiento de referencia.
3	Interfaz de comunicación / UART interfaz: conexión con Bluetooth, WIFI y se adopta el protocolo Dobot.
4	Interfaz USB: Conectar con PC.
5	Interfaz de alimentación: conecte con el adaptador de alimentación.
6	Interfaz periférico: Conexiones con bomba de aire, extrusora, sensor y otros equipos periféricos.

Figura 3.17 Descripción de la Interfaz.

- Herramientas finales.

Un kit de ventosa succionadora, provista de su bomba de aire.

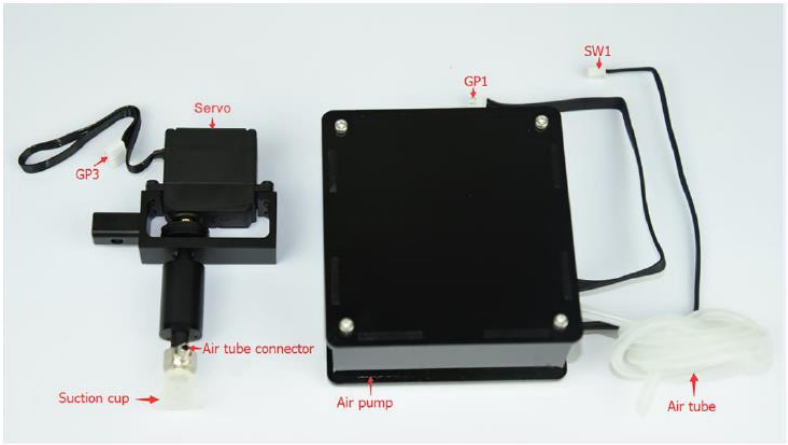


Figura 3.18 Ventosa Succionadora.

Un kit de pinzas, la cual también necesita la bomba succionadora para la apertura y cierre de la pinza.

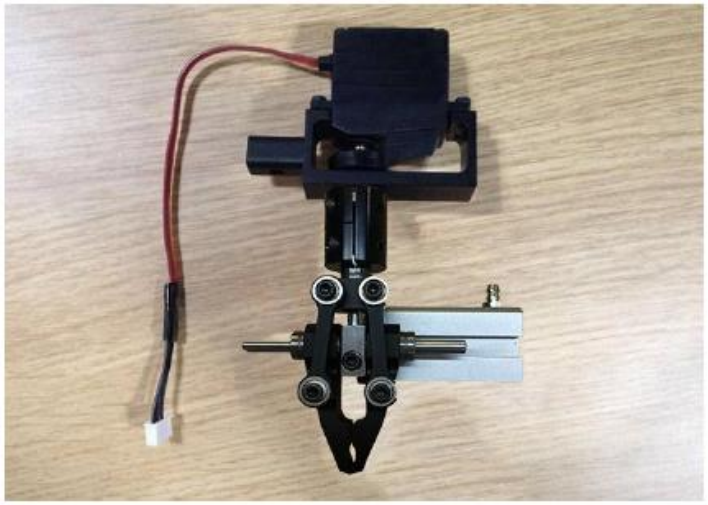


Figura 3.19 Pinza.

Un kit de escritura, que consta de un bolígrafo y un portalápiz.

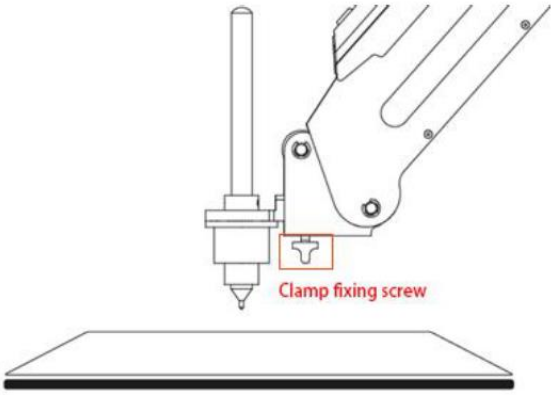


Figura 3.20 Kit de escritura.

Un kit laser, con el cual se pueden hacer grabados de imágenes en escala de grises.

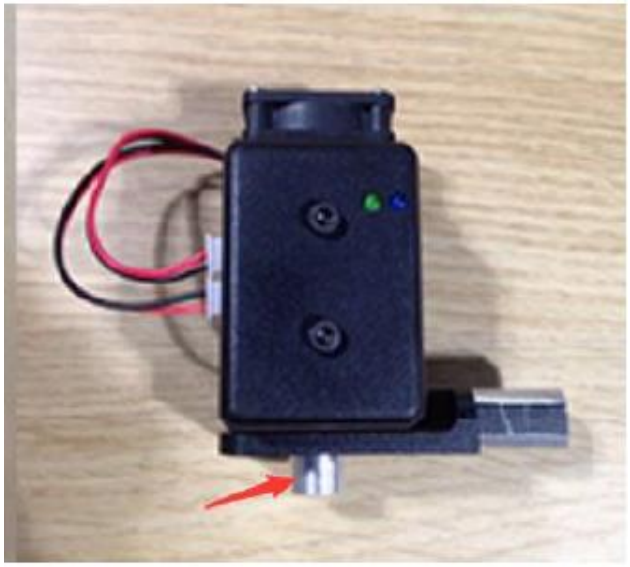


Figura 3.21 Kit Laser.

El kit de impresión 3D que contiene una extrusora con extremo caliente, cable de motor, filamento y soporte de filamento.

Figura 3.22 Kit de Impresión 3D.

3.3.1. Uso y adaptación al sistema.

Para la aplicación de este brazo-robot utilizará el modo de desplazamiento punto a punto usando el sistema de coordenadas cartesiano y procurará tener un control completo sobre estos desplazamientos. Además se usará su kit de pinza para el agarre de las piezas. Es en este punto en el que encontramos un problema de incompatibilidad con el acceso al porta-piezas del conjunto de medición de altura, pues en la disposición que se encuentra esta herramienta no es posible el acceso a depositar la pieza bajo el potenciómetro sin golpearlo por las dimensiones de la herramienta.

Por tanto, como solución a este inconveniente, se ha optado por diseñar una pieza (Figura 3.23), usando las dimensiones del aplicador de la herramienta aportados en el manual de nuestro Dobot Magician para reorientar el efector-final del kit de pinza haciendo posible el acceso al porta-piezas. Esta pieza ha sido impresa mediante la tecnología de impresión 3D.

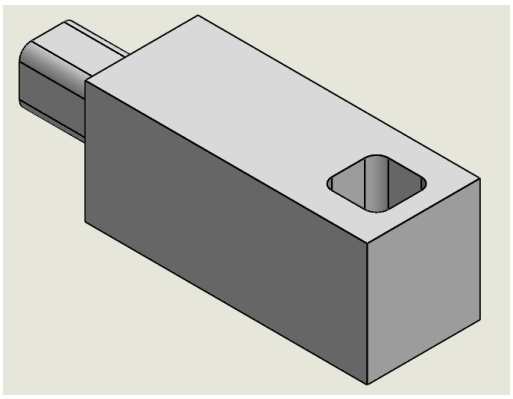


Figura 3.23 Pieza Impresa 3D.

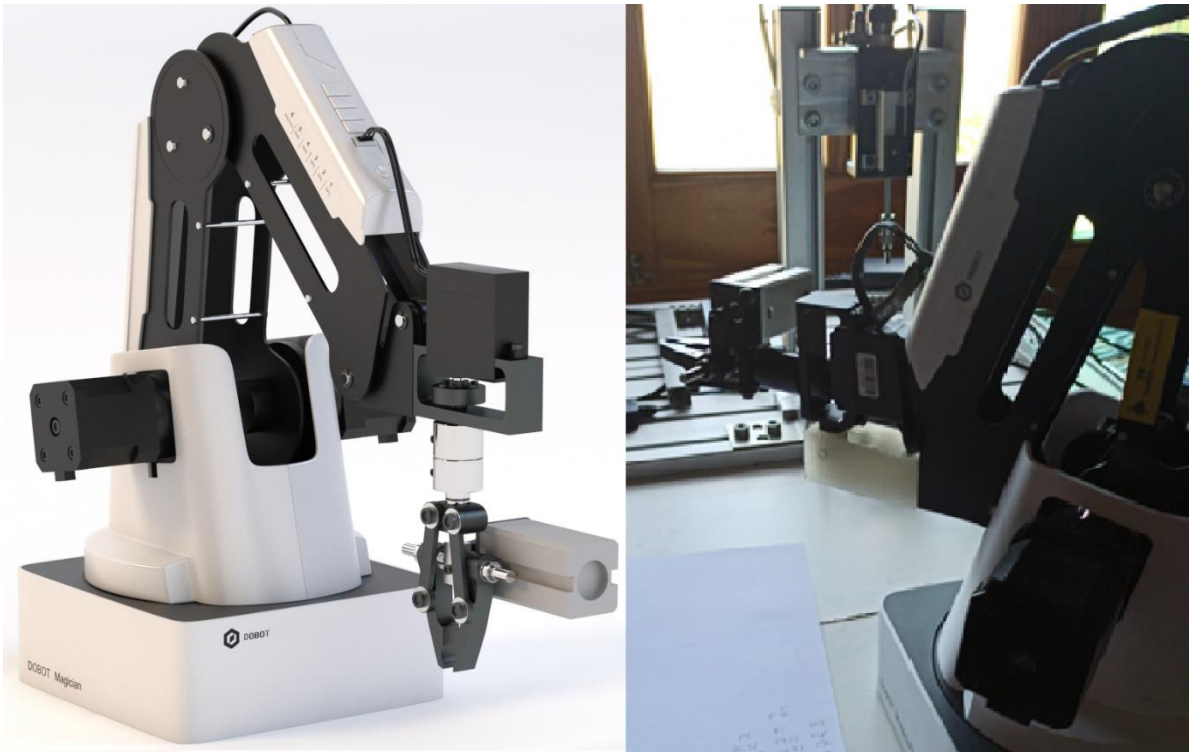


Figura 3.24 Reorientación de la herramienta.

3.4. Cámara de Visión Artificial.

El equipo de visión artificial está compuesto por una cámara industrial IDS modelo “UI-1460-C” con tecnología USB 2.0, una óptica de montura C y un trípode de sujeción donde acoplaremos la cámara en dirección vertical hacía la posición de recepción de la pieza.

No contamos con un sistema de iluminación profesional por las limitaciones impuestas por el confinamiento debido al Covid19, aunque recurrirá a soluciones caseras, como es el uso de un flexo y trabajar bajo un fondo oscuro para reducir la complejidad motivada por las posibles sombras que se produzcan.

Y por último, para poder acceder a la cámara usaremos las librerías del fabricante obtenidas a partir del kit de desarrollo de software suministrado (uEye).

La función correspondiente a la cámara es la de detectar la geometría superficial e identificar el color de la pieza. Para poder realizar estas funciones la cámara requiere de un software que le permita realizar el procesamiento de la imagen, por lo que requerirá de cierta programación y del uso de una biblioteca de imagen. La biblioteca de imagen que hemos elegido para este trabajo ha sido EmguCV.

Figura 3.25 Sistema de Visión Artificial.

Capítulo 4. SOLUCIÓN ADOPTADA.

En este capítulo se va a describir la solución adoptada para alcanzar los objetivos fijados en este Trabajo Fin de Grado.

Se comienza describiendo qué piezas serán objeto de estudio, se analizará cuál es el funcionamiento individual de cada componente en el sistema en el orden en el que intervienen en el proceso, posteriormente se explica a grandes rasgos el funcionamiento del sistema apoyándonos en la descripción de la programación del PLC, responsable principal de la automatización de este proceso, y finalmente, se expondrá cual ha sido la solución para solventar la comunicación de todos los elementos del sistema.

Aunque cabe adelantar, para una mejor comprensión de la solución, que se ha optado por desarrollar una aplicación software para la comunicación entre todos los elementos del sistema. Esto se debe a que de partida era necesario un soporte software y de procesamiento de imagen para la cámara. Tras el estudio de las distintas posibilidades del resto de componentes encontramos esta opción de comunicación, ya que todos disponen de distintas librerías de comunicación basadas en el mismo lenguaje (C#) y el brazo-robot no disponía de un método de comunicación con el PLC.

4.1. Aplicación final del Sistema de Control de Calidad.

Tras el estudio de las capacidades de la cámara y teniendo en consideración el rango de medida que disponía el potenciómetro, se fijó el alcance que tendría el análisis visual y las alturas de nuestras piezas. Este alcance consiste en analizar geometrías básicas, como secciones superficiales circulares, triangulares y rectangulares; el color, que será predominante en la pieza. Mientras que la altura de las piezas se han determinado a partir del espacio de trabajo del que dispone el brazo robot.

Por consiguiente, dispondremos de las siguientes configuraciones de piezas:

Pieza	1	2	3	4	5	6	7	8	9
Geometría / Color									
Altura (mm)	80	75	70	80	70	85	70	85	80

Figura 4.1 Tabla sobre configuración de piezas de estudio.

Como se puede comprobar, se dispone de una gran variedad de configuraciones en base a las 3 características de las piezas. Cabe destacar que la pieza 2 dispone de una altura anómala por defecto en su fabricación y que no se rectificó para poder disponer de una pieza defectuosa que el sistema descartase al configurarlo con la consigna de clasificación por altura.

El motivo particular por el que se han desarrollado estas piezas ha sido la situación excepcional provocada por el Covid-19 que limitaba las opciones a nuestro alcance, por lo que se recurrió a madera para su confección y a los colores y pinturas con las que se contaba en dicho momento. De estas piezas cabe destacar la entalladura que se les realizo para hacer posible el agarre mediante la pinza, ya que esta disponía de una amplitud de apertura reducida con respecto a las piezas, que debían tener ciertas dimensiones para ser manejables.

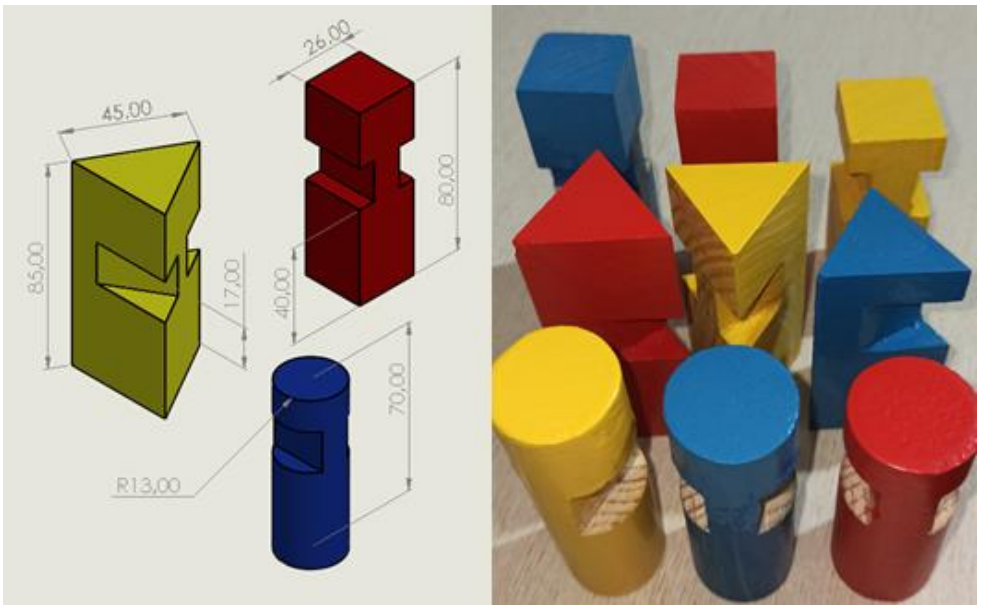


Figura 4.2 Piezas confeccionadas.

4.2. Funcionamiento individual de los componentes del sistema.

4.2.1. Cámara de Visión Artificial.

Las necesidades mencionadas que requiere la cámara de visión artificial son solventadas a partir del SDK del fabricante que cuenta con su propia librería de comunicación y configuración de imagen, mientras que para el aporte de procesamiento de la imagen utilizaremos la librería EmguCV.

La librería EmguCV (usamos la versión 3.1.0.1) es un contenedor .Net multiplataforma para la biblioteca de procesamiento de imágenes OpenCV. Permite la invocación de las funciones de OpenCV desde lenguajes compatibles con .Net como es el caso del lenguaje en el que se basa nuestra aplicación, C#.

Dejando a un lado la solución software, la cámara es el primer componente del sistema en intervenir en el proceso de control de calidad, pues será la encargada de detectar la llegada de una pieza al sistema y posteriormente, realizar un análisis visual de las características de la pieza. Al finalizar cada una de estas funciones, la cámara comunicará la información obtenida y la finalización del proceso que representa.

- **Detección de Movimiento.**

La detección de la aproximación de una pieza al sistema es una funcionalidad extra que se ha aportado al sistema para dotar a este de un mayor grado de automatización y para aprovechar aún más las capacidades de la cámara de visión, convirtiéndola de esta forma también en un sensor óptico. Esta detección está basada en la medición de la variación del gradiente de movimiento de la imagen. Este procesamiento de la imagen consiste en la diferenciación entre el fondo estático de la imagen y sus pixeles móviles, apoyándose en un registro de historial de movimiento.

Validaremos esta detección de movimiento en el momento que se registre un umbral superior de movimiento y se reduzca hasta cierto nivel, en el que garanticemos que no existen interferencias para determinar la presencia de una pieza, ya que la incorporación de estas será de forma manual.

- **Análisis Visual.**

El análisis visual consiste en el procesamiento de la imagen capturada por la cámara para ante la presencia de una pieza, determinar su color y geometría superficial. La determinación de la geometría de la pieza se consigue mediante el uso de métodos de detección de contornos y su aproximación a geometrías básicas. Mientras que la determinación su color, se logra mediante la filtración del color de la imagen. Por último, y como parece evidente, si tras filtrar el color se detecta una geometría en la imagen, además de haber realizado la primera etapa del control de calidad, habremos determinado la existencia de una pieza en la zona de incorporación al sistema.

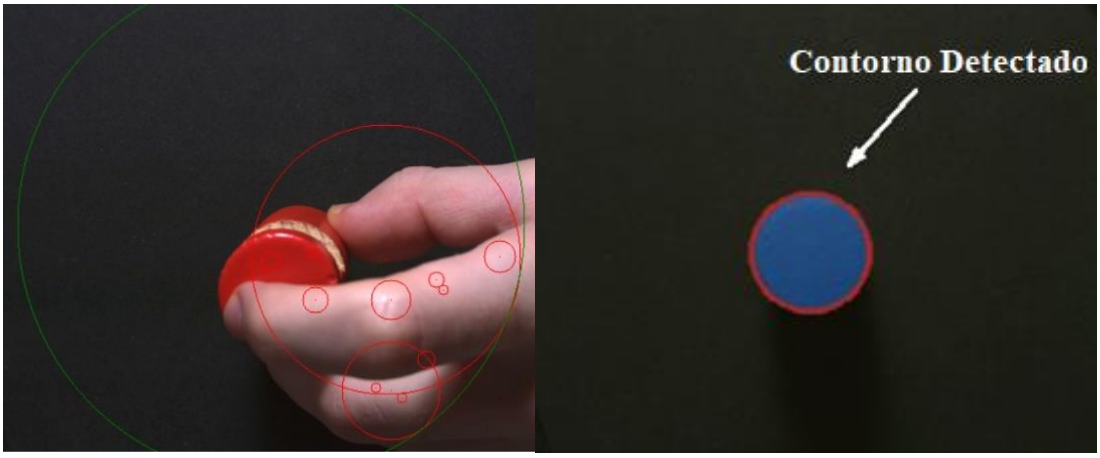


Figura 4.3 Detección de Movimiento y Análisis Visual.

4.2.2. Dobot Magician.

En este apartado vamos a describir la disposición, los procedimientos de desplazamiento (basado en movimiento punto a punto de coordenadas cartesianas) el agarra y deposición, y las distintas localizaciones del brazo en el sistema.

El brazo-robot quedará centrado y enfrentado al porta-piezas donde se realiza la medición de altura por parte de la instrumentación de la maqueta, con la finalidad de poder contar con todo el rango de movimiento de la base del robot y así disponer de 3 zonas del sistema. Una primera zona de recepción de la pieza (donde encontramos la cámara de visión artificial) a la izquierda del brazo, la segunda zona ya mencionada de medición de altura (en la maqueta) y una tercera zona donde realizaremos la clasificación de las piezas ubicada a la derecha del brazo.

En los distintos desplazamientos entre las zonas del sistema, el brazo pasa por determinadas posiciones de referencia. Las zonas de recepción de la pieza y de medición de altura cuentan con dos posiciones principales referencia y una intermedia, que son: la posición de recogida-deposición de la pieza de estudio, la posición final de desplazamiento o de reposo del brazo y una posición intermedia para hacer posible el correcto agarre y depósito de la pieza. En el caso de la zona de clasificación, tendremos una posición final de desplazamiento o reposo del brazo, pero contará con varias ubicaciones de clasificación e intermedias, cuya cantidad dependerá de cuantos grupos de clasificación dispongamos.

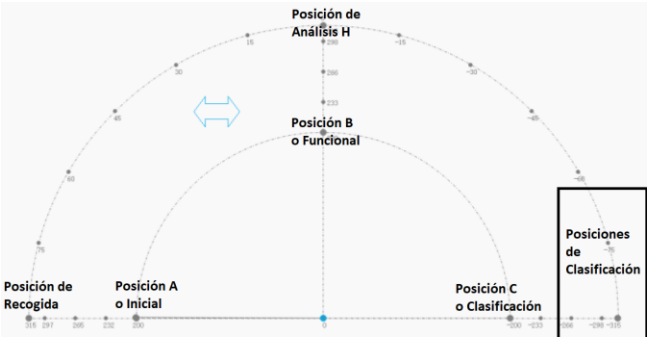


Figura 4.4 Esquema de posicionamiento del brazo.

Respecto a los desplazamientos del brazo, tiene 3 designaciones: Pieza a B o desplazamiento a la zona de medición de altura, Pieza a C o desplazamiento a la zona de clasificación y Brazo a A o desplazamiento a zona de recepción. Estos desplazamientos resultan del conjunto de subfunciones del brazo mencionadas al comienzo. Este conjunto de acciones ha sido definido exhaustivamente con la finalidad de dotar al PLC de un control completo sobre el brazo-robot, mediante el apoyo de la aplicación software desarrollada a partir de la generación de unos finales de carrera “ficticios”. Estos finales de carrera hacen posible el control de la extensión, la recogida y el giro del brazo, el agarre y la deposición de la pieza. Estas dos últimas funciones son más complejas, pues contemplan una aproximación o alejamiento horizontal de la pieza, para la correcta interacción con la pieza y la activación o desactivación de la pinza.

Todos los desplazamientos, salvo el que reposiciona el brazo al estado inicial con un único giro, cuentan con la misma estructura de acciones secuenciadas. Esta estructura consiste en la extensión del brazo hacia la posición intermedia en la que se encuentra la pieza, agarre de la pieza, retracción del brazo, giro del brazo a la siguiente zona que corresponda, extensión del brazo a la posición donde dejar la pieza, deposición de la pieza y finalmente, retracción del brazo a la posición final del brazo o de reposo. Estas posiciones finales del brazo están perfectamente definidas y consideradas en el funcionamiento del sistema para su correcto funcionamiento.

4.2.3. Equipo de medición de altura (Maqueta).

Además de los mandos que utilizamos para poner en marcha el sistema o para detenerlo, contamos con un equipo de medición de altura que utilizamos para completar la información dimensional de la pieza.

El proceso de medición de altura, una vez llega la pieza al porta-piezas y es detectado por los sensores capacitivo y/o inductivo (dependiendo del material del que este compuesta la pieza), consiste en el accionamiento de un pistón en cuyo vástago de forma solidaria se encuentra un potenciómetro lineal. Este desplazamiento de carácter vertical hacia abajo pone en contacto al potenciómetro con la pieza modificando así la resistencia variable interna que se transformará en una señal eléctrica que a su vez traduciremos en unidades de altura (mm). Esta altura corresponderá a un rango de medida acotado a cierta altura (65 mm) que dependerá de dicha dimensión de las piezas a medir. Una vez obtenida esta medida, el sistema volverá a su estado inicial recogiendo el pistón por completo y dando por finalizado, solo entonces, este proceso.

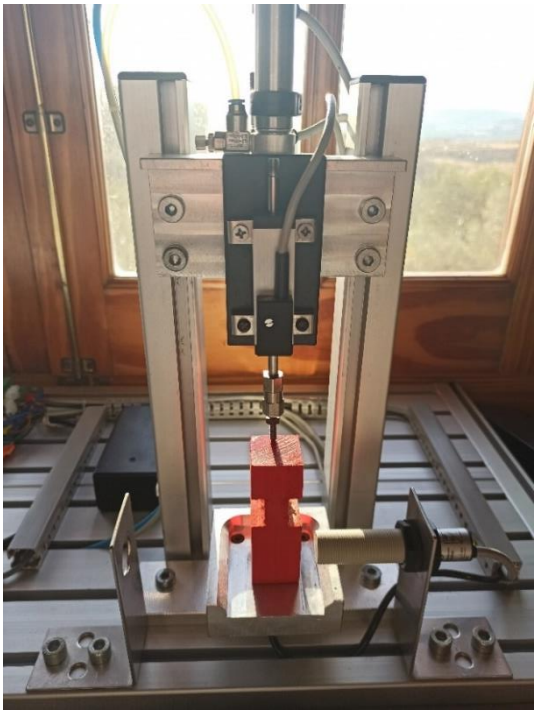


Figura 4.5 Medición de altura.

4.3. Funcionamiento Global del Sistema. Programación PLC.

La programación del PLC estará basada desarrollo de Redes Petri. Estas redes son una representación gráfica de un sistema de eventos discretos. Están formadas por una serie de estados, que se van activando secuencialmente mediante el cumplimiento de ciertas condiciones de transición.

4.3.1. Variables del PLC.

- **Entradas.**

Nombre	Tipo de datos	Dirección	Comentario
sens_Capacitativo	Bool	%I0.1	Sensor Capacitativo.
sensM_Abajo	Bool	%I0.3	Sensor magnético inferior del Pistón.
sensM_Arriba	Bool	%I0.4	Sensor magnético superior del Pistón.
pul_Marcha	Bool	%I0.6	Pulsador de Marcha.
pul_Emergencia	Bool	%I0.7	Pulsador de Emergencia.
m_Fin_Iniciando	Bool	%I2.0	Marca Fin de estado de Iniciando.
m_Mov_Detec	Bool	%I3.0	Marca de Movimiento detectado.
m_Pieza_Detec	Bool	%I3.1	Marca de Pieza detectada.
m_Fin_Detec	Bool	%I3.2	Marca Fin de detección.
m_Fin_AV	Bool	%I4.0	Marca Fin de Análisis Visual.
m_Fin_AH	Bool	%I4.1	Marca Fin de Análisis de Altura.
m_fc_BRetr	Bool	%I6.0	Marca fin de carrera, Retroceder brazo.
m_fc_BExt	Bool	%I6.1	Marca fin de carrera, Extender brazo.
m_fc_A	Bool	%I6.2	Marca fin de carrera, Posición Inicial.
m_fc_B	Bool	%I6.3	Marca fin de carrera, Posición frente AH.
m_fc_C	Bool	%I6.4	Marca fin de carrera, Posición frente Clasifi..
m_Herr	Bool	%I6.5	Marca de Herramienta.
Potenciometro	Word	%IW64	Lectura Potenciometro.

Figura 4.6 Entradas del PLC.

- **Salidas.**

valvula	Bool	%Q0.0	Válvula.
lamp_Blanca	Bool	%Q0.1	Lámpara Blanca.
lamp_Verde	Bool	%Q0.2	Lámpara Verde.
Q_Funcionamiento	Bool	%Q1.0	Salida, Funcionamiento activa.
Q_Detec_Mov	Bool	%Q2.0	Salida, Detección de Movimiento.
Q_AV	Bool	%Q3.0	Salida, Análisis Visual.
Q_PaB	Bool	%Q3.1	Salida, Pieza a B.
Q_LectH	Bool	%Q3.2	Salida, Lectura de Altura.
Q_PaC	Bool	%Q3.3	Salida, Pieza a C.
Q_BaA	Bool	%Q3.4	Salida, Brazo a A.
Q_ExtB	Bool	%Q4.0	Salida, Extensión de Brazo.
Q_RetrB	Bool	%Q4.1	Salida, Retracción de Brazo.
Q_GirarB	Bool	%Q4.2	Salida, Girar Brazo.
Q_Herr_OFF	Bool	%Q4.3	Salida, Desactivar Herramienta.
Q_Herr_ON	Bool	%Q4.4	Salida, Activar Herramienta.

Figura 4.7 Salidas del PLC.

- **Marcas del Programa.**

Nombre	Tipo de datos	Dirección	Comentario
System_Byte	Byte	%MB0	
Primer Ciclo	Bool	%M0.0	Marca que se ejecuta solo al inicio.
DiagStatusUpdate	Bool	%M0.1	
AlwaysTRUE	Bool	%M0.2	
AlwaysFALSE	Bool	%M0.3	
Est_PARO	Bool	%M1.0	Estado PARO.
Est_Iniciando	Bool	%M1.1	Estado Iniciando.
Est_ESPERA	Bool	%M1.2	Estado ESPERA.
Est_FUNCIONAMIENTO	Bool	%M1.3	Estado FUNCIONAMIENTO.
t01	Bool	%M2.0	
t10	Bool	%M2.1	
t12	Bool	%M2.2	
t13	Bool	%M2.3	
t20	Bool	%M2.4	
t23	Bool	%M2.5	
t30	Bool	%M2.6	
t32	Bool	%M2.7	
Est_Detec_Pausa	Bool	%M3.0	Estado Detección Pausada.
Est_Detec_Mov	Bool	%M3.1	Estado Detección de Movimiento.
Est_Detec_Pieza	Bool	%M3.2	Estado Detección de Pieza.
t45	Bool	%M3.3	
t54	Bool	%M3.4	
t56	Bool	%M3.5	
t64	Bool	%M3.6	
t65	Bool	%M3.7	
Est_Inicio_Ciclo	Bool	%M4.0	Estado Inicio de Ciclo principal.
Est_AV	Bool	%M4.1	Estado de Análisis Visual.
Est_PaB	Bool	%M4.2	Estado de Pieza a B.
Est_AH	Bool	%M4.3	Estado de Análisis de Altura.
Est_PaC	Bool	%M4.4	Estado de Pieza a C.
Est_BaA	Bool	%M4.5	Estado de Brazo a A.
t78	Bool	%M5.0	
t89	Bool	%M5.1	
t910	Bool	%M5.2	
t1011	Bool	%M5.3	
t1112	Bool	%M5.4	
t127	Bool	%M5.5	
t87	Bool	%M5.6	

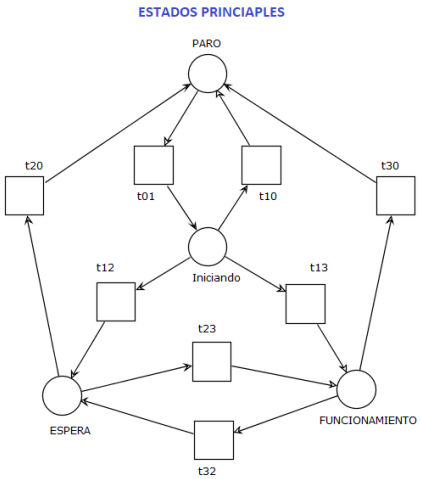
Figura 4.8 Marcas del PLC 1.

Nombre	Tipo de datos	Dirección	Comentario
Est_PB_0	Bool	%M6.0	Estado Posición Brazo Indefinida Inicial.
Est_PB_A	Bool	%M6.1	Estado Posición Brazo Inicial en reposo.
Est_PB_B	Bool	%M6.2	Estado Posición Brazo frente a AH en repo...
Est_PB_C	Bool	%M6.3	Estado Posición Brazo frente a Clasificació..
Est_B_Ext	Bool	%M6.4	Estado Extender Brazo.
Est_B_Retr	Bool	%M6.5	Estado Retroceder Brazo.
Est_B_Gir	Bool	%M6.6	Estado Girar Brazo.
Est_Herr_ON	Bool	%M6.7	Estado Activación Brazo.
Est_Herr_OFF	Bool	%M7.0	Estado Desactivación Brazo.
t1314	Bool	%M7.1	
t1415	Bool	%M7.2	
t1516	Bool	%M7.3	
t1517	Bool	%M7.4	
t1618	Bool	%M7.5	
t1718	Bool	%M7.6	
t1819	Bool	%M7.7	
t1820	Bool	%M8.0	
t1821	Bool	%M8.1	
t1914	Bool	%M8.2	
t1915	Bool	%M8.3	
t2015	Bool	%M8.4	
t2119	Bool	%M8.5	
Est_LectH_OFF	Bool	%M9.0	Estado de Lectura de Altura Desactivada.
Est_Piston_Abajo	Bool	%M9.1	Estado de Bajar Pistón.
Est_Lectura_H	Bool	%M9.2	Estado de Lectura de Altura.
Est_Piston_Arriba	Bool	%M9.3	Estado de Subir Pistón.
t2223	Bool	%M9.4	
t2324	Bool	%M9.5	
t2425	Bool	%M9.6	
t2522	Bool	%M9.7	
Est_V_OFF	Bool	%M10.0	Estado de Válvula apagada.
Est_V_ON	Bool	%M10.1	Estado de Válvula activada.
t2627	Bool	%M10.2	
t2726	Bool	%M10.3	
Est_Luz_OFF	Bool	%M11.0	Estado de luces apagadas.
Est_Luz_Verde_Inter	Bool	%M11.1	Estado de luz verde intermitente.
Est_Luz_Verde_Fija	Bool	%M11.2	Estado de luz verde fija.
Est_Luz_Blanca	Bool	%M11.3	Estado de luz blanca fija.
t2829	Bool	%M11.4	
t2830	Bool	%M11.5	
t2928	Bool	%M11.6	
t2930	Bool	%M11.7	
t2931	Bool	%M12.0	
t3028	Bool	%M12.1	
t3031	Bool	%M12.2	
t3128	Bool	%M12.3	
t3130	Bool	%M12.4	
Est_Inter_OFF	Bool	%M13.0	Estado de Intermitencia apagada.
Est_Inter_ON	Bool	%M13.1	Estado de Intermitencia activada.
t3233	Bool	%M13.2	
t3332	Bool	%M13.3	
Poten_Norm	DWord	%MD100	Lectura de Potenciometro Normalizado.
Poten_Escal	Real	%MD104	Lectura de Potenciometro Escalado.

Figura 4.9 Marcas del PLC 2.

4.3.2. Programación PLC en TIA PORTAL.

Las bases de funcionamiento partirán de un arranque manual mediante la pulsación del botón marcha de la maqueta, con el cual se iniciará una serie de procesos de conexión a los diversos componentes mediante la aplicación software. Una vez corroboradas estas conexiones, el sistema entrará en una fase donde se moverá entre dos estados hasta que se mande a parada. Estos dos estados comprenden, uno de espera mientras se aproxima una pieza para entrar en el sistema y ser detectada y otro de funcionamiento, donde se inicia al ciclo principal de funcionamiento, compuesto por las "macro acciones" del sistema (Red de Petri: *ESTADOS PRINCIPALES*)



t_{01}: pul_Marcha

t_{10}: pul_Parada

t_{12}: m_Fin_Iniciando x ~~Pieza_Detec~~ x PB_A

t_{13}: m_Fin_Iniciando x (Pieza_Detec x Inicio_Ciclo + ~~Inicio_Ciclo)~~

t_{20}: pul_Parada

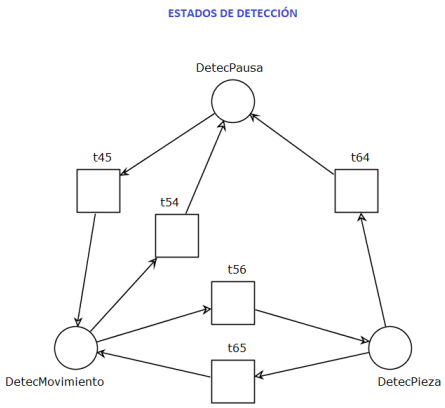
t_{23}: Pieza_Detec

t_{30}: pul_Parada

t_{32}: Inicio_Ciclo x ~~Pieza_Detec~~

Figura 4.10 Red Petri: Estados Principales y transiciones.

Para definir el proceso de detección desarrollamos otra red de Petri. Esta red estará compuesta de un estado de Pausa donde el ciclo de detección se detiene, un estado de detección de movimiento y un último estado de detección de pieza, que lanzará el análisis visual, para tanto detectar la pieza como sus características visuales. Este ciclo entra en pausa tras dar paso al ciclo principal del sistema y se vuelve iniciar tras ser retirada la pieza a la espera de detectar la aproximación de una segunda pieza que pueda aguardar a que el ciclo principal termine, para corroborar su presencia (Red de Petri: *ESTADOS DE DETECCIÓN*).



t_{45}: (ESPERA x + FUNC x fcA) x ~~Pieza_Detec~~ x ~~PARO~~

t_{54}: PARO

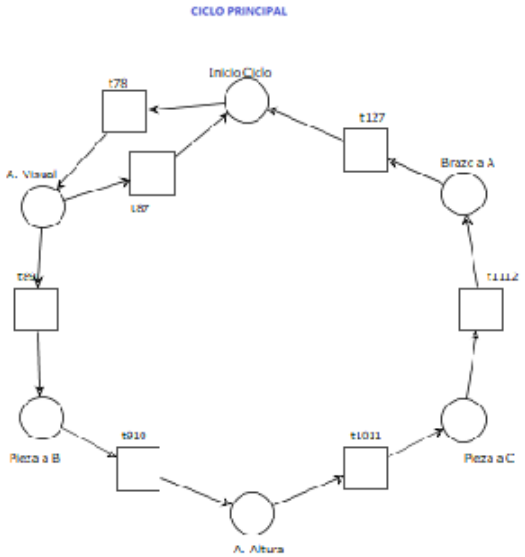
t_{56}: Mov_Detec x ~~PARO~~

t_{64}: PARO + Pieza_Detec x m_Fin_Detec

t_{65}: ~~Pieza_Detec~~ x m_Fin_AV

Figura 4.11 Red Petri: Estados de Detección y transiciones.

El ciclo principal de funcionamiento consta de un estado inicial que, tras detectarse una posible aproximación de una pieza, da paso al análisis visual. En caso de no corroborarse la existencia de la pieza, retorna al estado inicial del ciclo y se reanuda el ciclo de detección. En caso contrario, se activa una serie de acciones secuenciales sobre el brazo para el desplazamiento de la pieza hasta el porta-piezas de la maqueta. Una vez finalizada dicha secuencia, se activa el estado de análisis de altura de la pieza y se vuelve a recurrir al brazo robot para desplazar y clasificar las piezas en base a los criterios requeridos. Y finalmente, tras la clasificación y vuelta a una posición del brazo de reposo, se activa el último estado para reposicionar el brazo en su estado inicial de funcionamiento. (Red de Petri: *CICLO PRINCIPAL*)



t78: Est_Detec_Pieza

t87: MF_AV x ~~Pieza_Detec~~

t89: MF_AV x Pieza_Detec

t910: PB_B

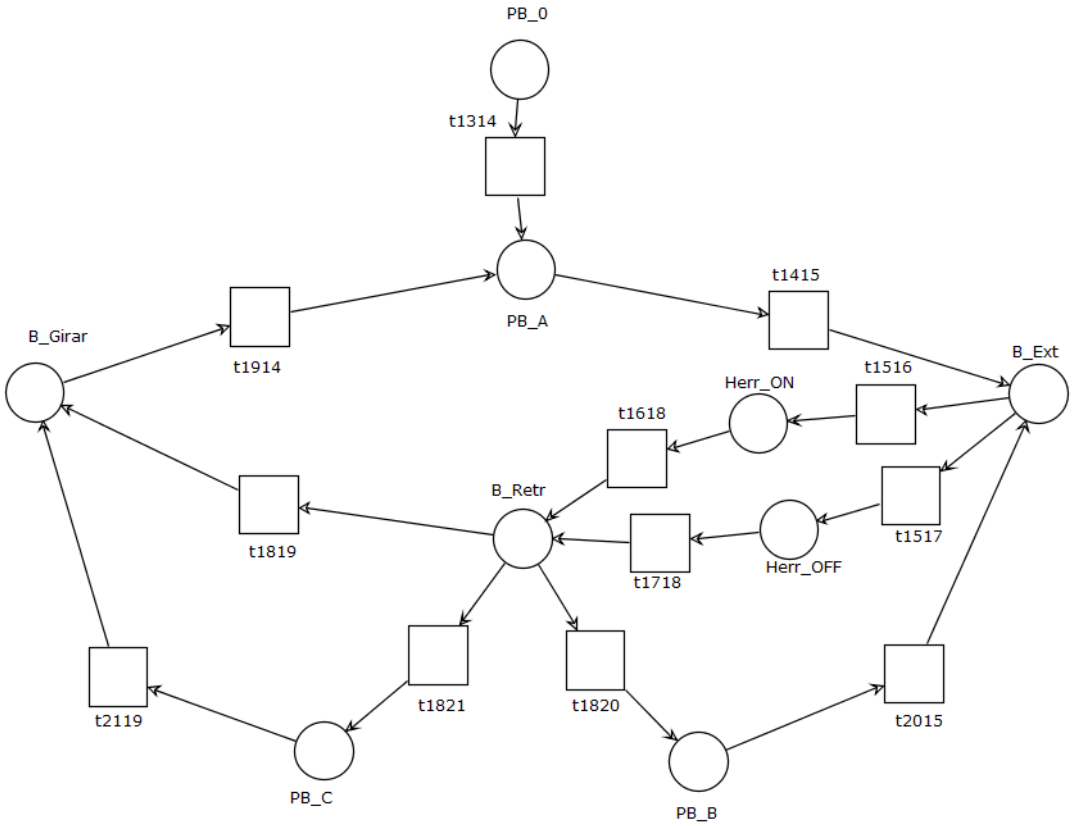
t1011: m_Fin_AH x LectH_OFF

t1112: PB_C

t127: PB_A

Figura 4.12 Red Petri: Ciclo Principal y transiciones.

La red que contempla los estados y acciones del brazo es la más compleja pues se busca un mayor control de este. En primer lugar, partimos de un estado que solo estará activo al principio (PB_0), pues está pensado para contemplar que el brazo se pone en funcionamiento en una posición arbitraría, y es reposicionado al finalizar el estado de inicialización del sistema, llegando así al estado de posicionamiento inicial del brazo (PB_A). Para alcanzar los diversos posicionamientos del brazo, existen otros estados que consisten en las acciones de agarre y deposición de la pieza extensión, retracción, y giro del brazo, necesarias para conseguir las distintas posiciones a las que debe llegar el brazo, quedando por mencionar, las de posicionamiento final frente al análisis de altura (PB_B) y la posición final frente a la clasificación de las piezas (PB_C). (Red de Petri: ESTADOS Y FUNCIONES DEL BRAZO)



t1314: Iniciando x fcBRetr x fcA x ~~Herr~~

t1415: PaB

t1516: (fcA + fcB) x ~~Herr~~ x fcExt

t1517: (fcC + fcB) x Herr x fcExt

t1618: (fcA + fcB) x Herr x fcExt

t1718: (fcC + fcB) x ~~Herr~~ x fcExt

t1819: (fcA + PaC x fcB) x Herr x fcRetr

t1820: fcB x ~~Herr~~ x fcBRetr

t1821: fcB x ~~Herr~~ x fcBRetr

t1914: fcA x Herr x fcBRetr

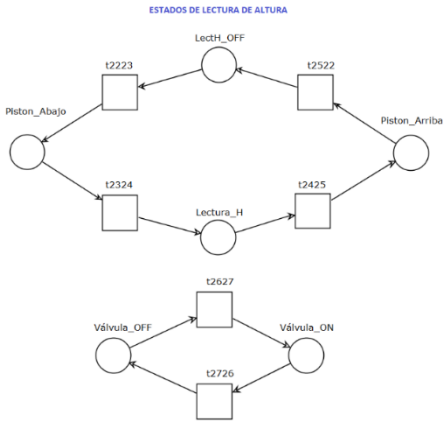
t1915: (fcC + fcB x PaB) x Herr x fcBRetr

t2015: PaC

t2119: BaA

Figura 4.13 Red Petri: Estados y Funciones del Brazo y transiciones.

Contamos con otra red secundaria cíclica para la realización de la lectura de la altura de la pieza, que consta con un estado de inactividad, otro de desplazamiento del pistón ligado al potenciómetro mediante la activación de una válvula, un estado de medida de la lectura donde se toma dicho valor y finalmente otro estado de retracción del pistón, para recuperar el estado inicial de inactividad. (Red de Petri: Lectura de altura). Esta red cuenta con una subred biestable para el accionamiento de la válvula que controla el pistón.



t2223: AH x s_Capac x ~~m_Fin_AH~~

t2324: sM_Abajo

t2425: m_Fin_AH

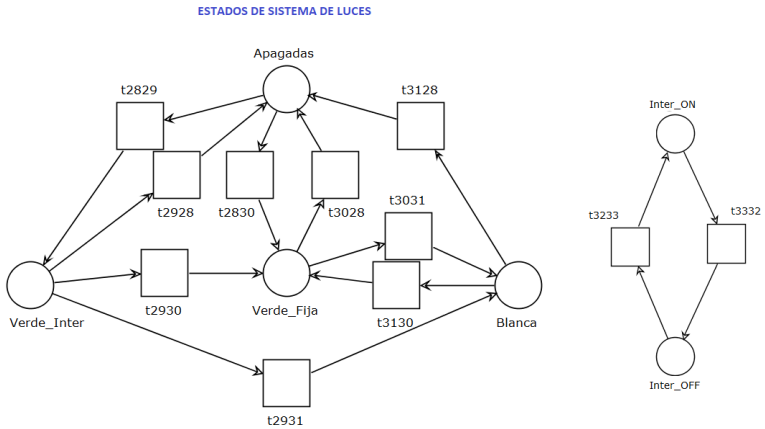
t2522: sM_Arriba

t2627: Est_Piston_Abajo

t2726: Est_Piston_Arriba

Figura 4.14 Red Petri: Estados de Lectura de Altura y transiciones.

también contamos con la red que se ocupa de la señalización luminosa y que se corresponde con los estados principales del sistema. Parte de un estado inicial, con todas las luces apagadas, continua con un estado de la luz verde intermitente que indica que estamos en fase de configuración de la aplicación que establece las comunicaciones, y finalmente dos estados de luz verde fija, para el de funcionamiento y uno de luz blanca fija para el estado de espera. (Red de Petri: *Estados de Luces*) Cuenta con una subred biestable para conseguir la intermitencia mediante dos temporizadores.



t2830: FUNC

t2829: Iniciando

t2928: PARO

t2930: FUNC

t2931: ESPERA

t3028: PARO

t3031: ESPERA

t3128: PARO

t3130: FUNC

t3233: Inter_OFF x TON(1s) x Est_Verde_Inter

t3332: Verde_Fija + Inter_ON x TON(1s) x Est_Verde_Inter

Figura 4.15 Red Petri: Estados de Sistema de Luces y transiciones.

Por último, existe una sección en la programación del PLC que corresponde a la lectura de la entrada analógica del potenciómetro de la maqueta con la que interpolamos la variación de altura medida. En primer lugar normalizamos la señal del potenciómetro de entera a real como viene indicada en el manual de entradas analógicas del PLC, y finalmente la escalamos a la variación de altura que se mide (25mm). Considerando esta variación y tras acotar debidamente el 0 de estas variaciones, obtendríamos las medidas reales de las piezas.

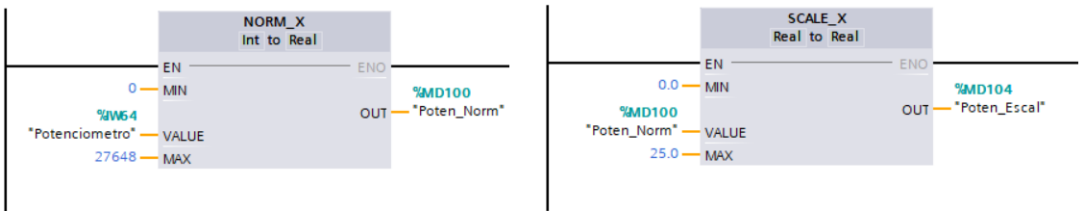


Figura 4.16 Interpolación de la medida del potenciómetro.

4.4. Comunicaciones mediante la aplicación Software.

Como se ha especificado al principio de este capítulo, debido a que los equipos han sido los provistos para desarrollar el sistema y no ha habido un estudio y selección previa de los mismo, la solución adoptada finalmente para la comunicación de todos los componentes ha sido la creación de una aplicación software en C# desarrollada en programación orientada a objetos (POO) haciendo uso de las bibliotecas que se mencionan en los apartados siguientes.

Por tanto, esta aplicación tendrá las funciones de recibir la información del PLC y de la cámara, que realiza el análisis visual de la pieza como la detección de esta, como de transmitir las ordenes correspondientes y aportar control, sobre el brazo robot. Finalizadas las operaciones indicadas por el PLC, comunicaría la finalización de las ordenes dichas ordenes mediante la activación de las entradas correspondientes para que continuará su programación.

Figura 4.17 Entornos de programación.

4.4.1. Comunicación con PLC S7-1200.

La comunicación software con el PLC estará basada en la biblioteca S7.net que funciona tan solo con PLCs Siemens y con conexión Ethernet. Esto significa que el PLC debe tener una CPU o una tarjeta externa Profinet. La biblioteca está escrita en C# por lo que puede ser depurada fácilmente. La página que aparece en la bibliografía donde se puede encontrar la biblioteca, cuenta con toda la documentación necesaria para la configuración del PLC así como la implementación de esta en nuestro código.

Se usará esta biblioteca para conectar y desconectar del PLC, leer y escribir tanto variables E/S como la analógica del potenciómetro lineal.

La lectura de la variable analógica será utilizada para el conocimiento de la altura de las piezas analizadas, pues las piezas podrán ser agrupadas en función de esta variable. En cuanto a la lectura y escritura de las variables E/S, esto servirá para comunicar al software qué procesos deben ejecutarse, en función de la lectura de las salidas activas (no tiene sentido que las salidas puedan ser escritas), mientras que solo tiene sentido la escritura de las entradas, puesto que servirán para comunicarle al PLC el estado del resto de componentes.

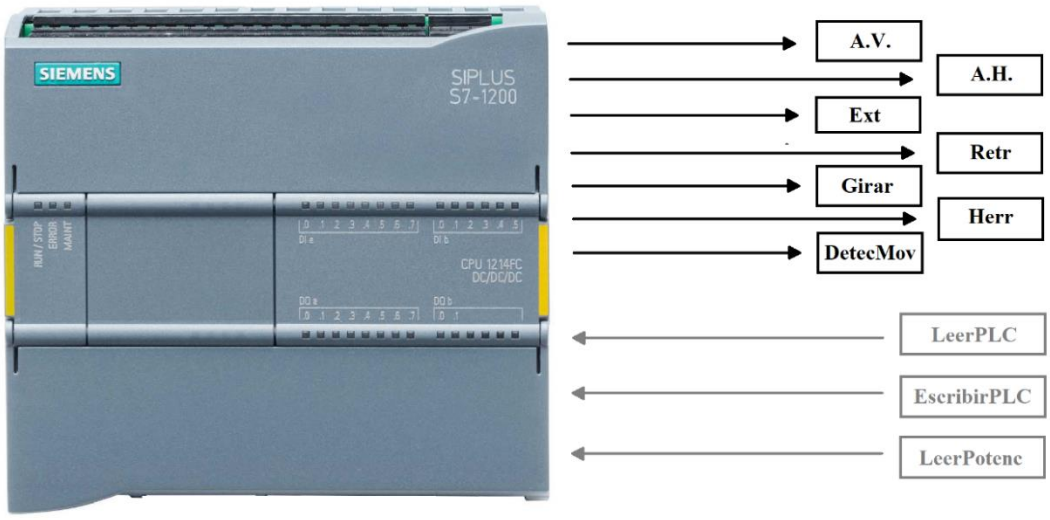


Figura 4.18 Comunicaciones PLC-Aplicación.

4.4.2. Comunicación con Dobot Magician.

La comunicación software con Dobot Magician la estableceremos a partir del aprendizaje a través de una demo en C# de la que nos provee el fabricante (“DobotDemoForC#”) y que cuenta con una serie de bibliotecas de comunicación, de funciones y tipos con los que trabaja el brazo robot. Estas bibliotecas tendrán que ser incorporadas a nuestra aplicación software.

Utilizaremos esta biblioteca para conectarnos y desconectarnos de Dobot Magician, establecer los parámetros con los que trabajara el brazo, tener un control en todo momento del posicionamiento del brazo, disponer de una herramienta para grabar posiciones de interés, activar y desactivar la herramienta, generar funciones que ejecuten el agarre y la deposición de la pieza y para garantizar que un determinado movimiento se ha completado.

Esta parte del software generará y trabajará con numerosas variables (finales de carrera “ficticios”) con el objeto de que el PLC contemple cada acción realizada por el brazo robot, pero será la aplicación software la que se encargará de hacer de puente de comunicación entre brazo-robot y PLC, trasmitiendo las ordenes que debe realizar el brazo-robot y devolviéndole la información de estas al PLC que se traduce en la activación y desactivación de las distintas variables mencionadas.

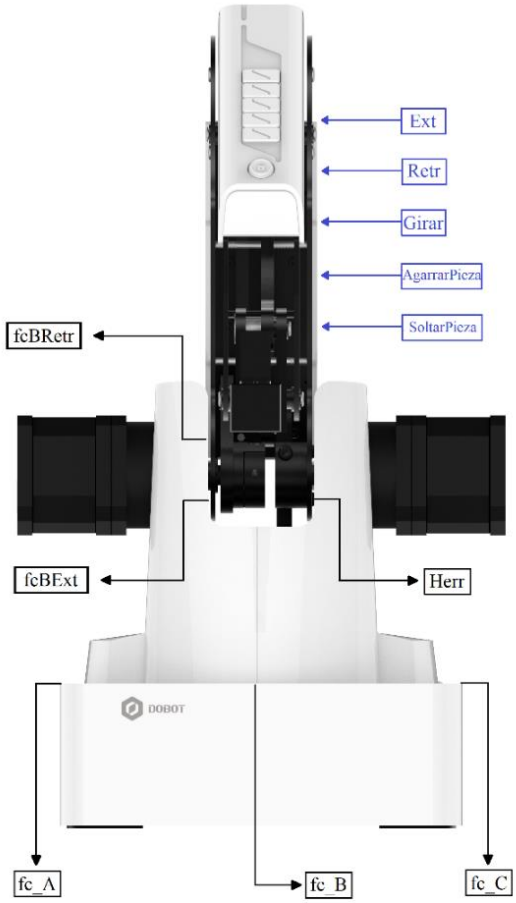


Figura 4.19 Comunicaciones Aplicación-DobotMagician

4.4.3. Comunicación con Cámara de Visión Artificial.

Dentro del SDK del fabricante podemos encontrar tres ejemplo en C# que serán a partir de los que obtendremos todas las funciones necesarias para la comunicación con la cámara, así como dos aplicaciones con las que podemos probar la cámara y guardar la configuración de los parámetros que utilizaremos para su uso. Los ejemplos de los que hablamos son:

- uEye_DotNet_C#_Cockpit
- uEye_DotNet_C#_DirectRenderer
- uEye_DotNet_C# SimpleLive.

Del ejemplo "uEye_DotNet_C#_SimpleLive" obtendremos como realizar un test inicial de la cámara, como iniciarla, cargar parámetros, obtener una imagen y como parar la cámara y salir.

Y del ejemplo "uEye_DotNet_C#_Cockpit" es donde se halla la forma de obtener la imagen de la cámara.

Por tanto, utilizaremos este software del fabricante, para iniciar y detener la cámara, cargar una serie de parámetros que calibrarían la cámara y para obtener imagen en formato captura o video.

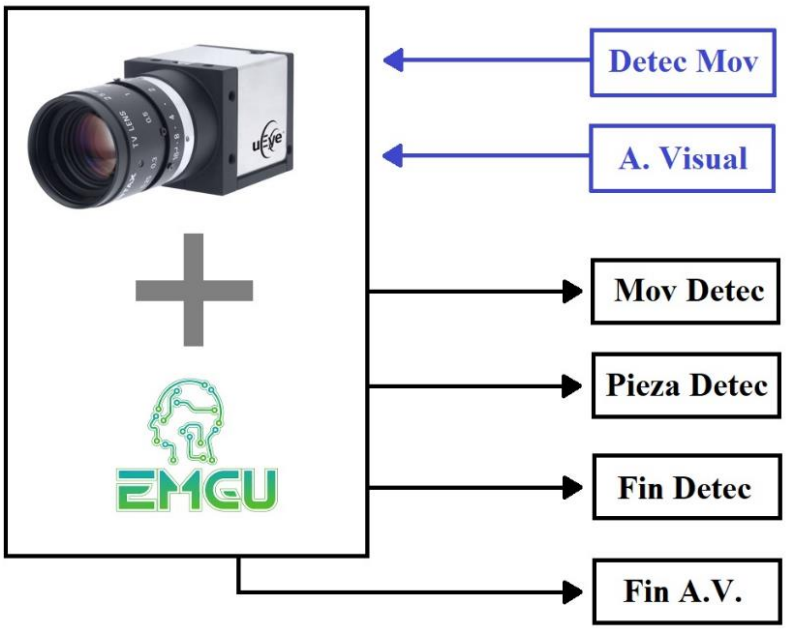


Figura 4.20 Comunicaciones Cámara-Aplicación.

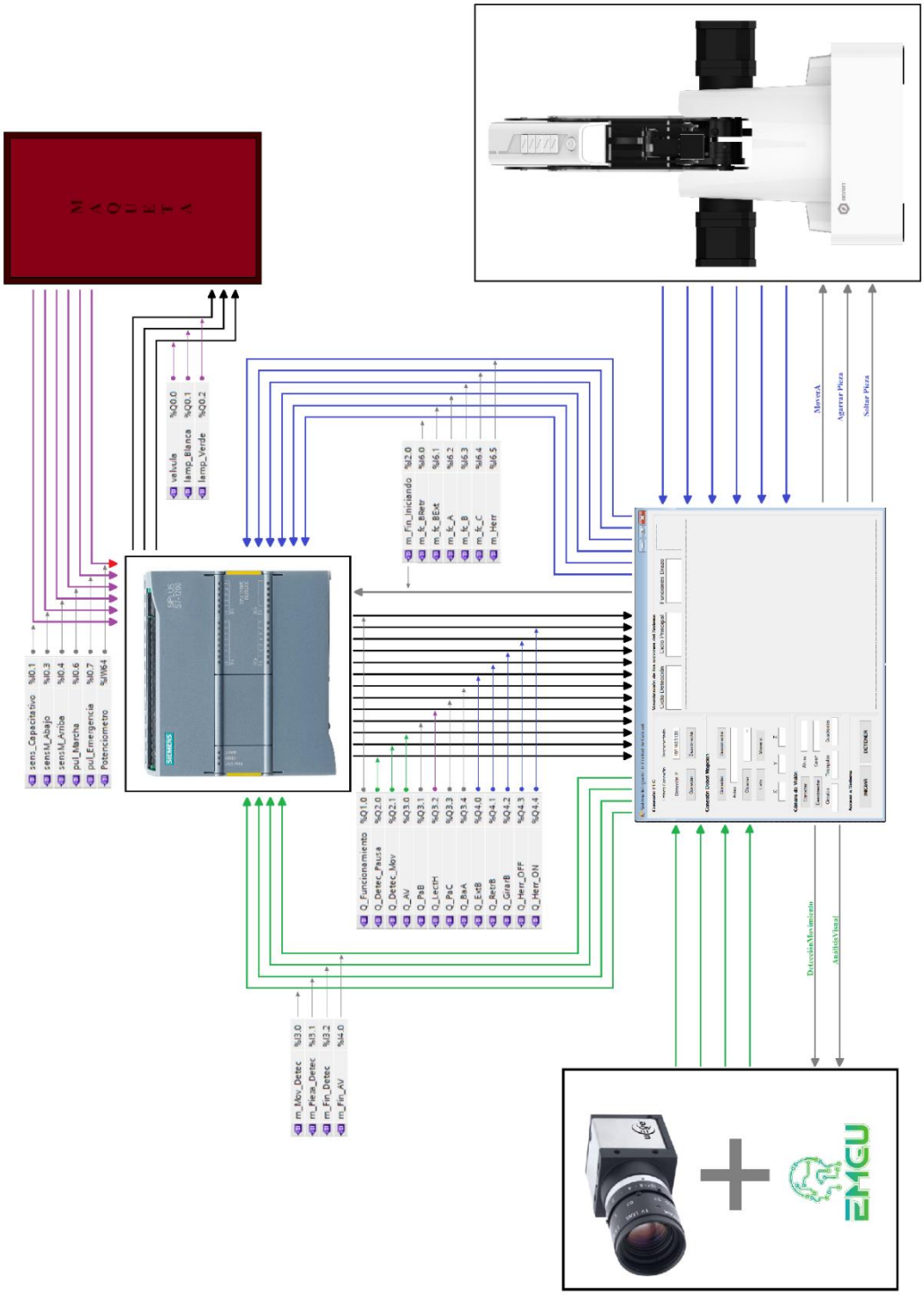


Figura 4.21 Conjunto de sentencias comunicativas del Sistema.

Capítulo 5. POSIBLES MEJORAS.

Cabe recordar que el objetivo de este trabajo es el de crear la aproximación de una parte de una línea de producción, como es un sistema de control de calidad, mediante el equipo disponible con el que se cuenta en el departamento. Por tanto, una mejora sustancial sería la de contar con equipos más profesionales, como contar con brazos industriales y una estructura que contara con cintas mecánicas que sustituyera la maqueta a escala. Pero debido a que este no es el objeto del trabajo, vamos a mencionar posibles mejoras aplicables o futuros trabajos que sobre este trabajo se pudieran realizar.

- Conexión directa de Dobot Magician con PLC S7-1200 y programación interna del propio brazo-robot.

Unas cuantas semanas antes de la finalización de este trabajo, el fabricante de Dobot Magician publico una nueva demo que describía la forma que habían desarrollado para poder conectar y comunicar el brazo-robot con los PLCs de Siemens. Por tanto, una posible mejora que se podría incorporar a este trabajo sería el estudio de esta demo del fabricante para ver las ventajas que supondría implementar esta modificación.

- Uso de la cámara para comunicar al brazo-robot del posicionamiento espacial de la pieza recibida.

Otra mejora que no ha podido ser desarrollada, ha sido la de implementar una coordinación cámara - brazo robot, con la que relacionar los sistemas de referencia espacial de ambos componentes con la finalidad de poder transmitir información de la ubicación de las piezas que llegan al sistema, para que el brazo pudiera actualizar la posición a la que desplazarse para recoger dicha pieza. Esta mejora también supondría incorporar otra mejora de calibración sobre la cámara para evitar el conocido efecto “ojo de pez” que provocan las lentes de las cámaras, distorsionando las dimensiones en la imagen cuanto más se acercan a los extremos de esta.

- Estudio más amplio de las capacidades de análisis visual que nos provee la cámara y biblioteca de visión artificial.

En este trabajo nos hemos limitado a realizar un análisis visual básico de las características de las piezas, detectando geometrías simples y piezas con un solo color. Pero somos conscientes del gran potencial con el que cuentan las bibliotecas de imagen, pudiendo detectar patrones de imagen más complejos. Por ejemplo, se podría estudiar implementar la detección de códigos QR.

- Mejora del motor de comunicaciones mediante el uso de varios hilos.

Los conocimientos sobre programación al abordar a este trabajo eran limitados, pero se han incrementado considerablemente gracias a él. Una avance para este trabajo podría ser el abordar la programación con multihilos para optimizar el motor de comunicaciones de la aplicación.

BIBLIOGRAFÍA

Autómata Programable.

[1] SIEMENS. Controlador programable S7-1200. [En Línea] [Consulta: 22 Mayo 2020]. https://media.automation24.com/manual/es/91696622_s71200_system_manual_es-ES_es-ES.pdf

[2] SIEMENS. Procesamiento de valores analógicos. [En Línea] [Consulta: 22 Mayo 2020]. https://github.com/ELECTROALL/PLC/blob/master/s71500_analog_value_processing_manual_es-ES_es-ES%20(1).pdf

[3] GitHub. S7Net+. [En Línea] [Consulta: 22 Mayo 2020]. https://github.com/S7NetPlus/s7netplus

[4] Educación en la Era Digital. Que es un PLC. [En Línea] [Consulta: 22 Mayo 2020]. https://ayto-torrijos.com/herramientas/que-es-plc/

Maqueta.

[5] Manual "Unidad Funcional Estación de Reconocimiento y Medición".

Dobot Magician

[6] DOBOT MAGICIAN. Download Center. [En Línea] [Consulta: 22 Mayo 2020]. https://www.dobot.cc/downloadcenter.html

[7] RevistadeRobots. ¿Qué es la Robótica? [En Línea] [Consulta: 22 Mayo 2020]. https://revistaderobots.com/robots-y-robotica/que-es-la-robotica/

[8] Escena.com, Qué es el brazo robótico y en qué industrias se emplea. [En Línea] [Consulta: 22 Mayo 2020]. https://www.esneca.com/blog/brazo-robotico-industrias/

Visión Artificial

[9] IDS. SDK del fabricante de la cámara. [En Línea] [Consulta: 22 Mayo 2020]. https://es.ids-imaging.com/download-details/AB.0010.1.09510.24.html

[10] IDS. UI-1460SE. [En Línea] [Consulta: 22 Mayo 2020]. https://es.ids-imaging.com/store/ui-1460se.html

[11] Wikipedia. Visión Artificial. [En Línea] [Consulta: 22 Mayo 2020]. https://es.wikipedia.org/wiki/Visión_artificial

[12] Infaimon. Sistemas de visión artificial: tipos y aplicaciones. [En Línea] [Consulta: 22 Mayo 2020]. https://blog.infaimon.com/sistemas-de-vision-artificial-tipos-aplicaciones/

[13] Cognex. Que es la visión artificial. [En Línea] [Consulta: 22 Mayo 2020]. https://www.cognex.com/es-es/what-is/machine-vision/what-is-machine-vision

[14] Tatomatech. Que sistemas de visión artificial existen. [En Línea] [Consulta: 22 Mayo 2020]. https://www.tatomatech.com/que-sistemas-de-vision-artificial-existen/

[15] YouTube. Inspección de tapones mediante visión artificial usando varias cámaras y diferentes algoritmos. [En Línea] [Consulta: 22 Mayo 2020]. https://www.youtube.com/watch?v=cC6msaRL03M

[16] Instituto de Diseño y Fabricación. Control de calidad mediante Visión Artificial. [En Línea] [Consulta: 22 Mayo 2020]. http://controlcalidad.institutoidf.com/descargas/folleto_control_calidad.pdf

[17] EmguCV. Página Oficial. [En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/index.php/Main_Page

[18] EmguCV. HoughCircles. [En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.0.0/document/html/6e055c09-5a1b-9f70-0a21-4f1a5c46c508.htm

[19] Wikipedia. Transformada de Houg. [En Línea] [Consulta: 22 Mayo 2020]. https://es.wikipedia.org/wiki/Transformada_de_Hough

[20] EmguCV. CvInvoke.Canny. [En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.2.0/document/html/fc9004a0-ce7f-2b0b-a090-ad67a7f97fea.htm

[21] EmguCV. CvInvoke. HoughLinesP. [En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.2.0/document/html/4630f25d-f238-f59a-a4e7-8c8aa3b6fa0c.htm

[22] EmguCV. CvInvoke.ApproxPolyDP.[En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.4.1/document/html/8ab7a295-6be8-7cfa-f345-23cc118f1469.htm

[23] EmguCV. CvInvoke.InRange.[En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.0.0/document/html/6577bad5-d270-efc5-a54f-0861f85c44fd.htm

[24] EmguCV. CvInvoke.ApproxPolyDP.[En Línea] [Consulta: 22 Mayo 2020]. http://www.emgu.com/wiki/files/3.4.1/document/html/8ab7a295-6be8-7cfa-f345-23cc118f1469.htm

[25] Wikipedia. Modelo de color HSV. [En Línea] [Consulta: 22 Mayo 2020]. https://es.wikipedia.org/wiki/Modelo_de_color_HSV

ANEXO. Aplicación software y Código.

1. Descripción e Interfaz.

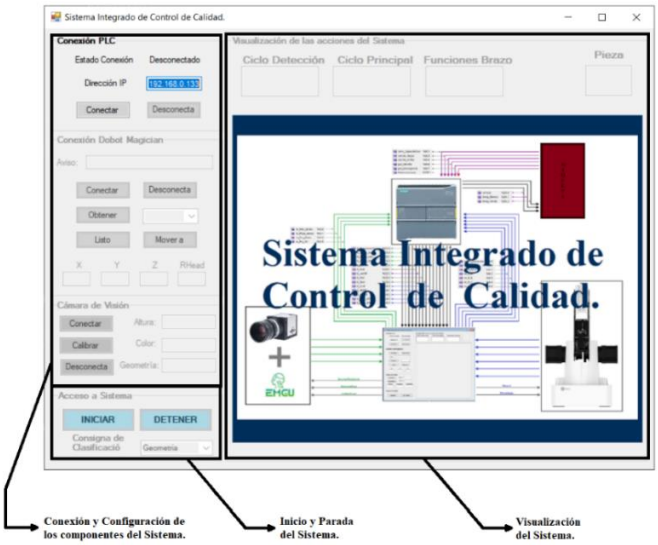


Figura A. 1 Descripción de la interfaz.

Contamos con 3 secciones de la interfaz. La primera se trata del acceso y configuración de los 3 equipos a los que la aplicación nos permite comunicarnos y que, por tanto, está formado por 3 subsecciones. Estas subsecciones tienen en común que nos permiten conectar y desconectar los distintos equipos. En el caso del PLC, nos permite introducir la dirección IP de este antes de conectarnos a él. Para el caso de Dobot Magician, contamos con un menú desplegable con las posiciones de referencia por las que se mueve el brazo robot. Seleccionando una de estas posiciones podemos desplazar hasta ella al brazo robot o grabar manualmente dichas posiciones. Además, cuenta con una serie de cuadros de texto, que nos mostrará avisos sobre las acciones realizadas como la posición espacial en la que se encuentra el robot. Cabe destacar que una vez configurado Dobot Magician es requerido pulsar el botón de listo, para que junto con el resto de las conexiones se reconozca que el sistema está perfectamente conectado y configurado. En el apartado de la cámara de visión artificial, nos muestra las propiedades de la pieza también recurriendo a cuadros de texto.

La segunda sección de nuestra interfaz está formada por dos botones para iniciar y detener el control de comunicaciones del sistema y un menú desplegable para definir la consigna de clasificación.

La tercera sección nos mostrará textualmente los estados del sistema, desde el punto de vista del ciclo de detección, desde el punto de vista del ciclo principal, desde las órdenes que se le dan al brazo, además de indicar el grupo al que pertenece la pieza y quedando la activación y desactivación de la herramienta señalizada mediante imágenes. Y por último, un gran recuadro de imagen al cual le pasaremos intercaladamente en el correspondiente momento la imagen obtenida por la cámara y la imagen procesada por la aplicación.

2. Clase PLCSiemens.

La clase *PLCSiemens*, se encarga de la comunicación entre la aplicación software y el PLC S7-1200. Cuenta con las propiedades y funciones siguientes.

- Inicializador.

```
PLCSiemens comunPLC = new PLCSiemens(CpuType.S71200, "192.168.0.133", 0, 1);
```

Figura A. 2 Inicializador Clase PLCSiemens.

Las cuatro propiedades del inicializador necesarias para establecer la comunicación ya las mencionamos en el apartado de configuración del PLC. Estas propiedades son el tipo de CPU, la dirección del PLC, el rack y el slot en el que se encuentra.

- Propiedades.

Contamos con una serie de variables propias de la comunicación del propio PLC presentes en el inicializador ya comentadas, una variable para conocer el estado de conexión y otra para evitar problemas de comunicación con el PLC, además de todas las variables E/S que intercambian PLC-Software.

Cabe explicar que estas E/S cuentan con variantes de variable del PLC, que son nombre o identificador de variable (ver_), la dirección de la variable en el PLC (direc_) y el estado de la variable (est_). Estos primeros estados de E/S sirven para tener dicha información a nivel local de la aplicación, aunque las salidas son leídas y las entradas son actualizadas localmente y en el PLC al mismo tiempo, de forma continuada. figura

```
varIn = new string[12] {    "MF_Iniciando", "MF_AV", "MF_AH",
                            "fcBRetr", "fcBExt", "fcA", "fcB", "fcC", "Herr",
                            "M_Mov_Detec", "M_Pieza_Detec", "MF_Detec"};
direcIn = new string[12] {  "I2.0", "I4.0", "I4.1",
                            "I6.0", "I6.1", "I6.2", "I6.3", "I6.4", "I6.5",
                            "I3.0", "I3.1", "I3.2"}; // Deben de >I1.7
estIn = new bool[12] {      false, false, false,
                            false, false, false, false, false, false,
                            false, false, false };

varOut = new string[13] {   "Funcionamiento", "AV", "PaB", "AH", "PaC", "BaA",
                            "Ext", "Retr", "Girar", "HerrOFF", "HerrON",
                            "DetecPausa", "DetecMov" };
direcOut = new string[13] { "O1.0", "O3.0", "O3.1", "O3.2", "O3.3", "O3.4",
                            "O4.0", "O4.1", "O4.2", "O4.3", "O4.4",
                            "O2.0", "O2.1" };
estOut = new bool[13] {     false, false, false, false, false, false,
                            false, false, false, false, false,
                            false, false };
```

Figura A. 3 Variables E/S PLC.

- Funciones.

- *ConectarPLC* [void]

Función encargada de establecer la conexión con el PLC, que se ha establecido correctamente y abrir la comunicación con este. Esta función obtiene los valores necesarios para establecer la conexión y que coinciden con los aportados y establecidos con el inicializador. figura

```
public void ConectarPLC()
{
    plcsiemens = new Plc(Modelocpu, DirIP, Rack, Slot);
    plcsiemens.Open();

    if (plcsiemens.IsConnected)
    {
        Conexion = true;
        MessageBox.Show("PLC conectado");
    }
    else
    {
        MessageBox.Show("Error al conectar PLC");
    }
}
```

Figura A. 4 Función ConectarPLC.

- *DesconectarPLC* [void]

Función que desconecta la comunicación con el PLC.

```
public void DesconectarPLC()
{
    plcsiemens.Close();
    Conexion = false;
    MessageBox.Show("PLC desconectado");
}
```

Figura A. 5 Función DesconectarPLC.

- *LeerPLC* (string) [bool]

Esta función provee de seguridad a la lectura de variables simples en el PLC, evitando problemas de acceso a la comunicación con este. Esto lo conseguimos bloqueando la ejecución de la lectura en el caso de que otra parte del código este accediendo al canal de comunicación.

```
private bool LeerPLC(string variable)
{
    bool value;

    while (Buffer) ;
    Buffer = true;
    value = Convert.ToBoolean(plcsiemens.Read(variable));
    Buffer = false;

    return value;
}
```

Figura A. 6 Función LeerPLC.

- *EscribirPLC* (string, bool) [void]

Dicha función es equivalente a *LeerPLC*, realizando la misma medida de protección respecto al abuso del canal de comunicación, pero para la escritura de estados en el PLC.

- *EscribirEntrada* (string, bool) [void]

Esta función escribe un estado a partir del nombre de la entrada que le pasamos como primer argumento, el cual corresponde con el segundo argumento aportado. Esta escritura se realiza tanto localmente como en el PLC hallando el índice del nombre y la dirección correspondiente a dicha entrada como mostramos a continuación.

```
public void EscribirEntrada(string vrbl, bool std)
{
    for (int i = 0; i < varIn.Length; i++)
    {
        if (VarIn(i) == vrbl)
        {
            EstIn(i, std);
            EscribirPLC(DirecIn(i), std);
        }
    }
}
```

Figura A. 7 Función EscribirEntrada.

- *LeerEntrada* (string) [bool]

Esta función es la encargada de, a partir del nombre de la entrada, hallar su índice correspondiente y devolvernos el estado local del índice correspondiente.

```
public bool LeerEntrada(string entrada)
{
    int y = 0;

    for (int i = 0; i < varIn.Length; i++)
    {
        if (VarIn(i) == entrada)
        {
            y = i;
        }
    }
    return EstIn(y);
}
```

Figura A. 8 Función LeerEntrada.

- *LeerSalida* (string) [bool]

Esta función es equivalente a *LeerEntrada*, pero en su versión para obtener el estado de las salidas localmente.

- *ActualizarSalida* [void]

Esta función consiste en una completa lectura del estado de las salidas del PLC para actualizarlas localmente.

```
public void ActualizarSalidas()
{
    for (int i = 0; i < estOut.Length; i++)
    {
        EstOut(i, LeerPLC(DirecOut(i)));
    }
}
```

Figura A. 9 Función ActualizarSalida.

- *LeerPotenciometro* [float]

Esta función recurre a una función más compleja de la biblioteca de comunicación del PLC para la lectura, con el fin de acceder al valor de la entrada analógica del potenciómetro. En este caso, la lectura realizada de la dirección y tipo de dato correspondiente habrá que transformarla en una variable de punto flotante para C# mediante las funciones de conversión de tipos de la biblioteca.

```
public float LeerPotenciometro()
{
    float lectura;
    DataType DataType = DataType.Memory;
    int DB = 1;
    int StartByteAdr = 104;
    int count = 20 * 4;

    while (Buffer) ;
    Buffer = true;
    byte[] byteArray = plcsiemens.ReadBytes(DataType, DB, StartByteAdr, count);
    Buffer = false;

    double[] result = S7.Net.Types.Double.ToArray(byteArray);

    lectura = Convert.ToSingle(Math.Round(result[0],2));
    return lectura;
}
```

Figura A. 10 Función LeerPotenciometro.

3. Clase DobotMagician.

La clase *DobotMagician*, encargada de la comunicación entre la aplicación software y el brazo-robot del cual recibe su propio nombre. Cuenta con las propiedades y funciones que se muestran a continuación.

- Inicializador.

```
DobotMagician comunDobotM = new DobotMagician();
```

Figura A. 11 Inicializado Clase DobotMagician.

- Propiedades.

Respecto a la imagen siguiente, contamos con las siguientes propiedades:

- conexion: variable que alberga el estado de la conexión con el brazo-robot.
- configuración y movON: variables de funcionamiento para optimizar el acceso del código de actualización del posicionamiento del brazo-robot.
- posicion: variable propia de la biblioteca de comunicación del brazo-robot para su desplazamiento.
- posTimer: evento temporal generado para la actualización del posicionamiento.
- posVariables (string): cadena de caracteres correspondientes a los nombres de las posiciones: las de reposo del brazo, las de interacción con la pieza y las intermedias para el proceso de agarre o deposición de la pieza.
- posiciones (matriz float): parámetros x, y, z y r que definen las posiciones.

```
public DobotMagician()
{
    conexion = false;
    configuracion = false;
    movON = false;

    posicion = new Pose();
    posTimer = new System.Timers.Timer();

    posVariables = new string[13] { "P_Inicial",        "PF_AH",        "PF_Clasificación",
                                    "P_Recogida",       "P_AH",         "P_C1", "P_C2", "P_C3",
                                    "P_RecogidaInter",  "P_AHInter",    "P_C1Inter", "P_C2Inter", "P_C3Inter" };

    Posiciones para Clasificación Vertical

    // Para clasificación horizontal
    posiciones = new float[13, 4] { {  50.0f,   81.0f,     0.0f,   59.0f },
                                    {  90.0f,    0.0f,     0.0f,    0.0f },
                                    {  61.0f,  -68.0f,     0.0f,  -48.0f },
                                    { 118.6f,  193.0f,   -87.0f,   57.0f },
                                    { 220.0f,    0.0f,   -38.0f,    0.0f },
                                    { 195.0f, -154.0f,   -89.0f,  -45.0f },
                                    { 166.0f, -177.0f,   -89.0f,  -45.0f },
                                    { 134.0f, -200.0f,   -89.0f,  -54.0f },
                                    {   0.0f,    0.0f,     0.0f,    0.0f },
                                    {   0.0f,    0.0f,     0.0f,    0.0f },
                                    {   0.0f,    0.0f,     0.0f,    0.0f },
                                    {   0.0f,    0.0f,     0.0f,    0.0f },
                                    {   0.0f,    0.0f,     0.0f,    0.0f } };
}
```

Figura A. 12 Propiedades de la Clase DobotMagician.

- Funciones.

- *IniciarDobot* [void]

Esta función establece y comprueba que la conexión al brazo-robot ha sido satisfactoria y llama a otras dos funciones. La primera, *EstablecerParametros*, que fija los parámetros de velocidad y aceleración de los movimientos que hacen posible el desplazamiento del brazo en función del modo de desplazamiento elegido y la segunda, *ObtenerPosicion*, que a partir de *posTimer* adquiere la posición del brazo cada cierto intervalo de tiempo (500 ms), a la vez que pasa los valores de dicha posición a la interfaz para visualizarlos.

La sentencia más relevante de esta función sería *ConnectDobot* cuyos parámetros definen al nombre del puerto, la velocidad de transmisión de este y dos cadenas de caracteres que requiere la conexión.

```
DobotDll.ConnectDobot("", 115200, fwType, version);
```

Figura A. 13 Función ConnectDobot.

Los parámetros que a nosotros nos interesan inicializar son las que conciernen al modo PTP por coordenadas cartesianas y de la herramienta.

```
private void EstablecerParametros()
{
    UInt64 cmdIndex = 0;

    PTPCoordinateParams cpbsParam;
    cpbsParam.xyzVelocity = 100;
    cpbsParam.xyzAcceleration = 100;
    cpbsParam.rVelocity = 100;
    cpbsParam.rAcceleration = 100;
    DobotDll.SetPTPCoordinateParams(ref cpbsParam, false, ref cmdIndex);
}
```

Figura A. 14 Parámetros PTP.

Y en último lugar, de la función ObtenerPosicion destaca la función *GetPose* que nos devuelve los parámetros que definen la posición del brazo (x, y, z, rHead).

```
DobotDll.GetPose(ref posicion);
```

Figura A. 15 Función GetPose.

- *DetenerDobot* [void]

Esta función detiene a su vez a *posTimer* y desconecta el brazo robot. La sentencia más importante de esta función es la siguiente.

```
DobotDll.DisconnectDobot();
```

Figura A. 16 Función DisconnectDobot.

- *MoverA* (string) [bool]

La función en cuestión es la encargada de ejecutar el movimiento al brazo-robot a partir del del nombre de la posición al que debe desplazarse. Para ello, se recorre el vector *posVariables* para identificar el índice de la posición a la que desplazarse, que corresponderá con el índice de la variable posiciones de la que obtendremos los parámetros que definen dicha posición y que se transmitirá a la función *PTPCmd*, propia de la SDK del fabricante y que se encarga de ordenar el desplazamiento al brazo robot.

```
private UInt64 PTPCmd(byte style, float x, float y, float z, float r)...
```

Figura A. 17 Función PTPCmd.

Además, cuenta con una sección de código que utilizando dichas posiciones, garantiza que se logra dicha posición comparándola continuamente con la ubicación instantánea del brazo y una vez culminada, devuelve una confirmación de esto, para que, como veremos en el funcionamiento de nuestro código principal, evitar que el código posterior se ejecute sin finalizar el desplazamiento del brazo-robot.

- *Herramienta* (bool, bool) [void]

Con esta función abrimos, cerramos y desactivamos el efector final que tiene instalado el brazo-robot con las indicaciones de energía (activo o desactivo) y estado (apertura o cierre) a partir de dos variables “bool”. La función característica de la biblioteca para la pinza es la que se muestra a continuación.

```
DobotDll.SetEndEffectorGripper(energy, estado, false, ref cmdIndex);
```

Figura A. 18 Función SetEndEffectorGripper.

- *GrabarPosicion* (string) [void]

Esta función es la inversa a MoverA, pues consiste en la obtención de los parámetros de una posición definida en nuestras propiedades de clase (x, y, z rHead) a través del mismo acceso a dichos parámetros para su reescritura a partir del nombre de esta posición (posVariables).

- *CalcularInterPos* [void]

La función en cuestión, ha sido desarrollada con la finalidad de reducir el número de posiciones a grabar durante la configuración del brazo para a partir de la posición de agarre o deposición de la pieza, calcular la interposición sobre la misma proyección radial y a una distancia prudencial, de tal manera, que la aproximación a la pieza sea horizontal y lineal para el correcto agarre de la pieza. Esto se traduce en recurrir a la ecuación de la pendiente y de la longitud en el espacio plano (x, y), traduciéndose en un algoritmo recogido en la Figura A.19.

```
private void AlgoritmoInterPos(int i)
{
    double radio;
    float pendiente;

    // r^2 = x^2 + y^2
    radio = Math.Sqrt(Math.Pow(posiciones[i, 0], 2) + Math.Pow(posiciones[i, 1], 2));
    radio -= 25.0;
    // m = y / x
    pendiente = posiciones[i, 1] / posiciones[i, 0];

    // x = r / raiz(1 + m^2)
    posiciones[i + 5, 0] = Convert.ToSingle(Math.Round(radio / (Math.Sqrt(1 + Math.Pow(pendiente, 2))), 1));
    // y = m * x
    posiciones[i + 5, 1] = Convert.ToSingle(Math.Round(pendiente * posiciones[i + 5, 0], 1));
    // z1 = z2
    posiciones[i + 5, 2] = posiciones[i, 2];
    posiciones[i + 5, 3] = posiciones[i, 3];
}
```

Figura A. 19 Algoritmo del cálculo de las Interposiciones.

- *AgarrarPieza* (string) [void]

Esta función recurre a las funciones Herramienta y MoverA, para el proceso de agarre de la pieza. Este proceso parte de la posición intermedia tras la acción de extensión del brazo, activa y abre la pinza, introduce la pinza en la posición donde se encuentra la pieza de forma horizontal y cierra y desactiva la pinza. Esta función recibe una variable "string" que corresponde con la zona donde agarrará la pieza para adaptar las posiciones de cada agarre.

- *SoltarPieza* (string, string) [void]

Esta función también recurre a las funciones Herramienta y MoverA pero para el proceso de deposición de la pieza. En este caso, el proceso parte directamente de la posición donde se deposita la pieza tras la acción de extensión del brazo, abre la pinza y retrocede hasta la posición intermedia de forma horizontal y lineal, para finalmente cerrar la pinza y desactivar la energía. Esta función recibe dos variables "string" que corresponden con la zona donde soltará la pieza y una segunda para el caso de clasificación que corresponde al tipo de pieza en cuestión.

```
public void AgarrarPieza(string zona)
{
    if (zona == "ZA")
    {
        Herramienta(true, false);
        while (MoverA("P_Recogida")) ;
        Herramienta(true, true);
    }
    if (zona == "ZB")
    {
        Herramienta(true, false);
        while (MoverA("P_AH")) ;
        Herramienta(true, true);
    }
}
```

```
public void SoltarPieza(string zona, string pieza)
{
    if (zona == "ZB")
    {
        Herramienta(true, false);
        while (MoverA("P_AHInter")) ;
        Herramienta(true, true);
        Herramienta(false, false);
    }
    if (zona == "ZC")
    {
        Herramienta(true, false);
        if (pieza == "C1")...
        if (pieza == "C2")...
        if (pieza == "C3")...
        Herramienta(true, true);
        Herramienta(false, false);
    }
}
```

Figura A. 20 Funciones AgarrarPieza y SoltarPieza.

4. Clase CamarauEye.

La clase *CamarauEye* es la encargada de establecer la comunicación entre la cámara de visión artificial y la aplicación software, así como la adquisición de imagen. Cuenta con una serie de propiedades y funciones para conseguir estos objetivos, pero antes de pasar a comentarlas, cabe aclarar que esta clase tan solo será llamada por la siguiente *VisionArtificial*, pues será la encargada de proveer a esta última clase de las imágenes y video necesario que procesar para que el sistema aplique sus capacidades de análisis de visión artificial requeridas.

- Inicializador y propiedades.

El siguiente inicializador de clase con parte de las propiedades con las que cuenta la clase: la cámara (que es inicializada), estado de la adquisición de video de la cámara y la dirección de memoria. Pero además existe una propiedad *ventana* con la que fijaremos las dimensiones de la imagen que obtenemos.

```
CamarauEye comunCamara = new CamarauEye(new uEye.Camera(), false, 0);
```

Figura A. 21 Inicializador Clase CamarauEye.

- Funciones.

- *IniciarCamara* [void]

Esta función es la encargada de abrir la cámara, asignar la dirección de memoria y la adquisición de video, lo cual consiste en la iniciación de la cámara.

Esta adquisición de imagen por parte de la cámara se consigue a partir del lanzamiento de un evento propio de la biblioteca de la cámara.

Usaremos este proceso para nuestra adquisición de video para el proceso de detección de movimiento.

```
public void IniciarCamara()
{
    camera = new uEye.Camera();

    statusRet = 0;

    // Abrir Cámara
    statusRet = camera.Init();
    if (statusRet != uEye.Defines.Status.Success)
    {
        MessageBox.Show("Error al abrir Camara");
        Environment.Exit(-1);
    }

    // Asignar Memoria
    statusRet = camera.Memory.Allocate();
    if (statusRet != uEye.Defines.Status.Success)
    {
        MessageBox.Show("Error al asignar memoria");
        Environment.Exit(-1);
    }

    // Iniciar Video
    statusRet = camera.Acquisition.Capture();
    if (statusRet != uEye.Defines.Status.Success)
    {
        MessageBox.Show("Error al iniciar video");
    }
    else
    {
        bLive = true;
        camera.EventFrame += onFrameEventInitCamera;
    }
```

Figura A. 22 Función IniciarCamara.

- *PararCamara* [void]

Esta función detiene el evento de adquisición de video de la cámara y cierra la cámara.

```
public void PararCamara()
{
    if (camera.Acquisition.Stop() == uEye.Defines.Status.Success)
    {
        camera.EventFrame -= onFrameEventInitCamera;
        camera.Exit();
        bLive = false;
    }
}
```

Figura A. 23 Función PararCamara.

- *CargarParametros* [void]

La función en cuestión ha sido adquirida a partir de uno de los ejemplos del SDK del fabricante de la cámara ("uEye_DotNet_C#_SimpleLive") y consiste en cargar una serie de parámetros de configuración de la imagen obtenida por la cámara, que pueden almacenarse en un archivo. Este archivo se puede generar fácilmente a partir del propio software aportado por el fabricante "Cockpit".

- *CapturarImagen* (ImageBox) [Mat]

Esta función ha sido desarrollada tras el estudio de los ejemplos del SDK para hallar la forma de obtener una imagen y como convertir el formato de salida en uno con el que pueda trabajar nuestra biblioteca de imagen.

```
public Mat CapturarImagen(ImageBox ventana)
{
    Bitmap bitmap;
    Camera.Memory.ToBitmap(s32MemID, out bitmap);
    Image<Bgr, Byte> image = new Image<Bgr, Byte>(bitmap)
        .Resize(ventana.Width, ventana.Height, Emgu.CV.CvEnum.Inter.Linear, true);
    Frame = image.Mat;
    return Frame;
}
```

Figura A. 24 Función CapturarImagen.

La segunda sentencia de la Figura A.24 es la que nos provee una imagen en formato *Bitmap*, la cual convertiremos en formato *Image<Bgr, Byte>* y finalmente pasaremos a formato *Mat*, con la cual trabajaremos para el campo de visión artificial.

5. Clase VisionArtificial.

La clase *VisionArtificial* es la encargada de albergar todas las funciones requeridas para nuestro control visual de la pieza así como de aportar la nueva funcionalidad a la cámara como sensor de detección de movimiento. Esta clase requiere la llamada a la clase *CamarauEye*.

- Inicializador.

```
VisionArtificial VisionA = new VisionArtificial();
```

Figura A. 25 Inicializador Clase VisionArtificial.

Con este inicializador, establecemos una serie de propiedades que contemplan tanto el proceso de detección de movimiento, detección de geometrías y las variables principales de clasificación. Además de inicializar la variable historial de movimiento.

```
historialMov = new MotionHistory(
       1.0,  // Duración del historial de movimiento que mantener (en segundos).
       0.05, // Mínima variación de tiempo considerado para calcular el gradiente de movimietno
       0.5); // Máxima variación de tiempo considerado para calcular el gradiente de movimiento
```

Figura A. 26 Objeto MotionHistory.

- Propiedades.

```
private bool conexion;
private ImageBox ventana;
private bool inicioDetec;
private bool firstLapDetec;
private bool umbralDetec;
private bool mostrar;
private int nMov;
private TextBox txtbNM;
private Mat segMascara;
private Mat fondoMascara;
private MotionHistory historialMov;
private BackgroundSubtractor fondoDetec;
private string geometria;
private double dimension;
private double escala;
private string color;
```

Figura A. 27 Propiedades de VisionArtificial.

Las propiedades de esta clase las podemos dividir en función de los procesos a los que pertenecen. En primer lugar, *conexión* que define el estado de la cámara y *ventana* que utilizamos para actualizar la imagen de la interfaz que muestra los resultados de esta clase. En segundo lugar, las que intervienen para la detección de movimiento, como son: *inicioDetec*, *firstLapDetec*, *umbralDetec*, *mostrar*, variables utilizadas para el correcto funcionamiento de la función de detección de movimiento y *nMov* que alberga el valor cuantitativo del movimiento y su acceso a la interfaz para mostrarlo (*txtbNM*).

Y otras variables para el procesamiento de la imagen en la detección de movimiento como son: *segMascara* segmenta los movimientos detectados, *fondoMascara* obtiene el fondo inmóvil de la imagen, *historialMov* registra el movimiento y *fondoDetec* que substrae el fondo. Y por último, *geometria* y *color*, que definen las propiedades de la pieza y *dimensión* y *escala*, que son utilizadas para la correcta medida de las dimensiones de la pieza y que será utilizada para la clasificación de estas.

- Funciones.

- *IniciarVA* [void]

Esta función básicamente llama las funciones necesarias de la clase CamarauEye para iniciar y cargar los parámetros de calibración de la imagen. Además de definir el tamaño de la imagen con la que trabajaremos.

- *DetenerVA* [void]

Esta función es la encargada de detener la adquisición de imagen y/o video de la cámara mediante el acceso a la clase de esta.

- *DeteccionMovimiento* [bool]

```
public bool DeteccionMovimiento()
{
    if (inicioDetec == true)
    {
        comunCamara.Camera.EventFrame += Event_DetecMov;
    }
    if (firstLapDetec == true)
    {
        mostrar = true;
    }
    if (umbralDetec == false)
    {
        if (nMov > 100)
        {
            umbralDetec = true;
        }
    }
    if (umbralDetec == true)
    {
        if (nMov < 50)
        {
            mostrar = false;
            umbralDetec = false;
            firstLapDetec = true;
            return true;
        }
    }
    return false;
}
```

Figura A. 28 Función DeteccionMovimiento.

La función *DeteccionMovimiento* es la encargada de añadir al evento de adquisición de la cámara la función de detección de movimiento para conseguir un valor cuantitativo del movimiento en la imagen. Este incorporación de código solo debe realizarse la primera vez, por ello la existencia de la propiedad *inicioDetec*. También al inicio del ciclo para la posible comunicación de la imagen entre clase e interfaz utilizamos la variable *mostrar*.

Establecido este procesamiento de la imagen fijamos un umbral máximo para considerar haber comenzado a detectar movimiento evitando cualquier vibración o interferencia en la imagen y otro umbral inferior con el que considerar la finalización de la detección de movimiento. Este procedimiento se realiza de este modo para evitar cualquier interferencia que pueda afectar a *AnalisisVisual*.

Cabe destacar que esta función será recorrida continuamente mientras la programación del PLC lo indique y cuya llamada será realizada por el motor de comunicaciones con hilo propio del código principal.

- *Event_DetecMov* (object, EventArgs) [void]

Este código es el incorporado al de adquisición de imagen y es el encargado de procesar la imagen que obtiene de la cámara para determinar la variación de movimiento. Esta variación de movimiento se comprende como la variación espacial de los pixeles dinámicos de la imagen.

Este procesamiento consiste en la adquisición de la imagen y determinación del fondo de la imagen, diferenciando entre pixeles estáticos y dinámicos que, en base a los parámetros del historial de movimiento, determinará el gradiente de movimiento. Una vez obtenido todo el campo de movimiento de la imagen, se rechazarán las áreas pequeñas de movimiento, y aquellas áreas que contengan poco movimiento. Para finalizar representa los movimientos individuales (círculos rojos) y un movimiento general (círculo verde).

Para determinar el rango de movimiento a considerar, obtendremos la cantidad de componentes de movimiento detectados (nMov). Y además, añadimos unas líneas de código para que en caso de ser el primer ciclo del evento (*firstLapDetec*), introducir un retraso que evite que el inicio de adquisición de imagen provoque perturbaciones en la detección de movimiento y provoque un falso positivo.

- *DibujarMovimiento* (Mat, Rectangle, double, Bgr) [void]

Esta función es la encargada de dibujar los movimientos mencionados anteriormente. Para ello, recurre a la localización de los movimientos, a su tamaño y el ángulo de dirección del gradiente de movimiento, para determinar el radio y la posición del circulo, y el origen y fin de la línea radial, que identifican el movimiento.

- *DetectarPieza* (Mat) [bool]

Esta función es la encargada de, tras la detección de movimiento, garantizar si existe una pieza y de qué color en la posición de recepción o se trata de un falso positivo.

Esto consiste en someterla a una serie de funciones de EmguCV que detecta los bordes en ella y a partir de estos si existen circulo o la detección de bordes y posibles líneas en la imagen, para comprobar si se aproxima a algún tipo de polígono sencillo (triángulo o rectángulo). Toda esta información es almacenada en las propiedades de la clase y en el caso de detectar cualquier caso de los anteriores devuelve una confirmación. La información de la geometría detectada será transferida a la interfaz.

- *AnalisisVisual* [void]

La función AnalisisVisual es la correspondiente al ciclo principal de la programación del PLC, y es la encargada de detectar la presencia de la pieza analizada y sus características. Esto se logra convirtiendo la imagen obtenida de la cámara a formato HSV y filtrando el color en base a esta codificación. En el caso de pasarle el filtro correspondiente al color de la pieza, la función *DetectarPieza* nos devolvería una confirmación de detectar la geometría, por tanto las características de la pieza quedarían definidas, ya que a esta función tan solo le quedaría pasar la información de color a la interfaz.

```
public bool AnalisisVisual()
{
    Mat imgHSV = new Mat();
    imgThresColor
    string[] Threscolor = new string[3] { "Amarillo", "Azul", "Rojo" };
    Mat[] imgThreshColors = new Mat[3] { imgThreshGreen, imgThreshBlue, imgThreshRed };

    Mat image = comunCamara.CapturarImagen();
    Mat imgAV = new Mat();
    image.CopyTo(imgAV);

    CvInvoke.CvtColor(imgAV, imgHSV, ColorConversion.Bgr2Hsv);
    CvInvoke.InRange(imgHSV, new ScalarArray(new MCvScalar(10, 0, 80)),
        new ScalarArray(new MCvScalar(50, 255, 255)), imgThreshColors[0]);
    CvInvoke.InRange(imgHSV, new ScalarArray(new MCvScalar(51, 0, 80)),
        new ScalarArray(new MCvScalar(149, 255, 255)), imgThreshColors[1]);
    CvInvoke.InRange(imgHSV, new ScalarArray(new MCvScalar(0, 0, 80)),
        new ScalarArray(new MCvScalar(4, 255, 255)), imgThreshLowRed);
    CvInvoke.InRange(imgHSV, new ScalarArray(new MCvScalar(150, 0, 80)),
        new ScalarArray(new MCvScalar(180, 255, 255)), imgThreshHighRed);
    CvInvoke.Add(imgThreshLowRed, imgThreshHighRed, imgThreshColors[2]);

    for (int i = 0; i < Threscolor.Length; i++)
    {
        if (DeteccionPieza(imgThreshColors[i], imgAV) == true)
        {
            color = Threscolor[i];
            return true;
        }
    }
    return false;
}
```

Figura A. 29 Función AnalisisVisual.

- *Calibracion* [void]

```
public void Calibracion()
{
    double patron = 26.0;
    escala = 1.0;

    AnalisisVisual();

    escala = dimension / patron;
}
```

Figura A. 30 Función de Calibración.

Esta función será utilizada para calibrar la medida de la dimensión característica de las geometrías detectadas por la imagen, pues las medidas que obtenemos no son las equivalentes a la realidad sino a la correspondiente a la imagen. Como patrón utilizaremos las piezas de sección circular.

6. Código principal y motor de la aplicación.

El desarrollo de todas las anteriores clases para las comunicaciones y acciones de los distintos elementos del sistema son las responsables de que el código principal sea mucho más breve, conciso y comprensible, pues el código que ejecutan todos los componentes de la interfaz con los que interactuamos, nuestra función fundamental "Control de Comunicaciones" y nuestra función de clasificación, se reduce básicamente a la llamada de las funciones o propiedades que pertenecen a estas clases, salvo mínimas excepciones.

```
// Definición de llamada de clases.
PLCSiemens comunPLC = new PLCSiemens(CpuType.S71200, "192.168.0.133", 0, 1);

DobotMagician comunDobotM = new DobotMagician();

VisionArtificial VisionA = new VisionArtificial();


// Estado del sistema
bool estSistema = false;

// Definición de variables Control de Eventos
Thread control;

//variables para clasificación
string pieza;
string consigna;

// Variables para configurar la transferencia de la altura al brazo-robot
private bool[] primerapieza = new bool[3] { true, true, true };
```

Figura A. 31 Propiedades del Código Principal.

En primer lugar, las variables locales del código principal se reducen a las llamadas de las clases que hemos descrito con sus correspondientes parámetros, una variable que define el estado del sistema (*estSistema*), dos variables *string* que albergaran el tipo de pieza y la consigna de clasificación (*pieza* y *consigna*), un vector para configurar la transferencia de la altura o la dimensión de la pieza al brazo en función de la forma de depositar las piezas al clasificarlas y de forma más relevante, la definición de un subproceso fundamental para el control de comunicaciones y actuaciones del sistema (*control*).

En segundo lugar, como es típico en una ampliación Windows Form, se inicializan los componentes de la interfaz, donde un grupo de estos componentes son pasados a las clases. Para evitar excepciones por violación de acceso a controles de interfaz desde otro subproceso, en cada clase desde la que se accede a dichos controles se generan funciones para ser llamadas a partir de delegados.

En tercer lugar, tendríamos el grupo de eventos asociados a las pulsaciones de los botones de la interfaz. De los cuales, a continuación, solo vamos a destacar aquellos cuyo código no consista solamente en la llamada de las funciones lógicamente correspondientes (ya explicadas en los apartados correspondientes a su clase) y la definición del comportamiento de la interfaz al interactuar con estos botones.

Respecto al acceso a Dobot Magician, el botón Listo representa haber terminado de configurar el propio robot, calcula las interposiciones, ejecuta el movimiento del brazo a la posición inicial y activa los finales de carrera correspondientes a esa posición, necesario para el correcto funcionamiento de la programación del PLC y el sistema.

Respecto al de Sistema, contamos con dos botones y un menú desplegable: en el caso de Iniciar, activa la marca final de Iniciación del sistema, registra la consigna de clasificación y lanza el control de comunicaciones del código principal (Figura A.32), para el caso de detener, detendríamos el evento de control y desactivaríamos la entrada anterior; y por último, generamos un evento asociado al cierre de la aplicación para detener todas la comunicaciones para evitar problemas de ejecución.

```
private void btnIniciar_Click(object sender, EventArgs e)
{
    consigna = cmbxClasif.Text;
    comunPLC.EscribirEntrada("MF_Iniciando", true);

    control = new Thread(Control_Comunicaciones);
    control.Start();

    estSistema = true;

    FormInitSist
}
```

Figura A. 32 Acciones botón Iniciar Sistema.

```
private void Control_Comunicaciones()
{
    while (true)
    {
        comunPLC.ActualizarSalidas();

        #region CONTROL DE CALIDAD

        // ANALISIS VISUAL
        if (comunPLC.LeerSalida("AV") == true)...

        // ANÁLISIS DE ALTURA
        if (comunPLC.LeerSalida("AH") == true)...

        #endregion

        #region ACCIONES DEL BRAZO

        // EXTENDER BRAZO
        if (comunPLC.LeerSalida("Ext") == true)...

        // RETROCEDER BRAZO
        if (comunPLC.LeerSalida("Retr") == true)...

        // GIRAR BRAZO
        if (comunPLC.LeerSalida("Girar") == true)...

        // HERRAMIENTA OFF
        if (comunPLC.LeerSalida("HerrOFF") == true)...

        // HERRAMIENTA ON
        if (comunPLC.LeerSalida("HerrON") == true)...

        #endregion

        #region CICLO DETECCIÓN

        // DETECCIÓN DE MOVIMIENTO
        if (comunPLC.LeerSalida("DetecMov") == true)...

        #endregion
    }
}
```

Figura A. 33 Función Control_Comunicaciones.

En cuarto lugar, encontramos las funciones del código principal.

La denominada función de "*Control_Comunicaciones*" es la más importante de esta aplicación, pues es la encargada de coordinar las acciones del sistema en base a la lectura y escritura de entradas y salidas del PLC para el correcto funcionamiento de la programación del PLC y en consecuencia del sistema. Esta será lanzada por el subproceso *control*.

Como podemos ver en la figura de la izquierda, esta función está basada en un bucle infinito que inicialmente realiza una lectura completa de todas las salidas del PLC para poder conocer la acción que ordena ejecutar. Hemos dividido estas órdenes en 3 grupos en virtud de la sección funcional a la que pertenecen.

En cuanto a la sección de “Control de Calidad”, encontramos las dos acciones de análisis tanto visual como de la medida de altura. En el caso del análisis visual, desactivaríamos la marca de movimiento detectado, lanzaremos la función correspondiente y activaríamos la marca fin del análisis y de detección, y si se hubiera detectado una pieza, se activaría correspondiente entrada y se pasarían a la interfaz sus características analizadas. Y en el caso de la medición de la altura, el valor medido, correspondería a una medida relativa, pues la cota 0 del potenciómetro la fijamos a 65mm respecto al porta-piezas. Por tanto, una vez recopiladas todas las características de la pieza, clasificamos la pieza en función de la consigna indicada, y en virtud de esto, trasladamos dicha información al brazo y finalmente se activa la marca de fin del análisis de altura (Figura A.34).

```
// ANÁLISIS DE ALTURA
if (comunPLC.LeerSalida("AH") == true)
{
    this.Invoke((Action)delegate () { txtbCP.Text = "Análisis Altura"; });

    float z = 65.0f;
    z += comunPLC.LeerPotenciometro();
    this.Invoke((Action)delegate () { txtbH.Text = z.ToString(); });

    Clasificacion(z);

    // Modo Clasificación Horizontal
    ActualizarClasifH(pieza);

    comunPLC.EscribirEntrada("MF_AH", true);

    this.Invoke((Action)delegate () { txtbCP.Text = ""; });
}
```

Figura A. 34 Sección Análisis de Altura de la función Control_Comunicaciones.

En cuanto a la sección de “Acciones del Brazo” encontramos las 5 acciones correspondientes, extensión, retracción, giro, activación y desactivación de la herramienta. El funcionamiento básico de estas acciones es: desactivar el final de carrera ficticio del estado en el que se encontraba inicialmente el brazo, ejecuta las funciones correspondientes a la acción y activa el final de carrera correspondiente al nuevo estado del brazo. Más allá del funcionamiento básico, cada acción recurre a los parámetros específicos de sus funciones para particularizar cada acción. Por ejemplo, para el caso de extensión, el brazo realiza esta acción en varias ocasiones, pero quien determina cual es la posición a la que extender el brazo son: el estado del brazo, el de la herramienta y el de la pieza en cuestión, como podemos ver en la Figura 7.35.

```
if (comunPLC.LeerSalida("Ext") == true)
{
    this.Invoke((Action)delegate () { txtbFB.Text = "Extender"; });

    // Desactivamos fcBRetr
    comunPLC.EscribirEntrada("fcBRetr", false);

    if (comunPLC.LeerSalida("PaB"))
    {
        this.Invoke((Action)delegate () { txtbCP.Text = "Pieza a AH"; });
        if (comunPLC.LeerEntrada("Herr") == false)
        {
            // Desactivar Marca Fin AV.
            comunPLC.EscribirEntrada("MF_AV", false);

            while (comunDobotM.MoverA("P_Recogida")) ;
        }
        if (comunPLC.LeerEntrada("Herr") == true)
        {
            // Desactivar Marca Fin AH
            comunPLC.EscribirEntrada("MF_AH", false);

            while (comunDobotM.MoverA("P_AH")) ;
        }
    }
    if (comunPLC.LeerSalida("PaC"))
    {
        this.Invoke((Action)delegate () { txtbCP.Text = "Pieza a Clasificación"; });
        if (comunPLC.LeerEntrada("Herr") == false)
        {
            while (comunDobotM.MoverA("P_AH")) ;
        }
        if (comunPLC.LeerEntrada("Herr") == true)
        {
            if (VisionA.Pieza == "C1")
            {
                while (comunDobotM.MoverA("PC1")) ;
            }
            if (VisionA.Pieza == "C2")
            {
                while (comunDobotM.MoverA("PC2")) ;
            }
            if (VisionA.Pieza == "C3")
            {
                while (comunDobotM.MoverA("PC3")) ;
            }
        }
    }

    // Activamos fcBExt
    comunPLC.EscribirEntrada("fcBExt", true);
}
```

Figura A. 35 Sección Extender de la función Control_Comunicaciones.

Y por último, en la sección “Ciclo Detección” encontramos una únicamente la acción detección del movimiento que como el resto de las acciones, desactiva y activa las entradas correspondientes para el correcto funcionamiento de la programación del PLC. Pero reside en este lugar y se ha elegido una cadena de *if* en puesto de un *switch* para que sea ejecutado siempre que se liberen las acciones que le preceden del ciclo principal.

Al final del código principal, encontramos las funciones que conciernen a la clasificación. Aquella que lleva su nombre es la encargada de en función de la consigna configurada en la interfaz determina a que grupo de clasificación se corresponde la pieza. Esta función es llamada tras la función de análisis de altura, pues es el momento en que contamos con todas las características de la pieza.

```
private string Clasificacion(float zh)
{
    if (cmbxClasif.Text == "Geometria")...

    if (cmbxClasif.Text == "Color")...

    if (cmbxClasif.Text == "Altura")...

    return "C4";
}
```

Figura A. 36 Función Clasificación.

Pero para el proceso de clasificación además de determinar el grupo al que pertenece cada pieza, requiere de funciones que actualicen la posición donde el brazo debe depositarlas en base a cada grupo para lo que contamos con dos modos cuyo acceso debe de ser establecido por código. Un primer modo vertical donde la altura de las posiciones de clasificación son actualizadas para que el brazo deposite las piezas una encima de otra mediante el uso de la altura (*ActualizarClasifV*) y un segundo modo horizontal donde estas posiciones son actualizadas para que el brazo deposite las piezas de forma radial usando la dimensión característica de la pieza (*ActualizarClasifH*). Este último modo recurre a un algoritmo similar que la clase *DobotMagician* para calcular las nuevas posiciones de clasificación como esta hacía para las interposiciones.

7. Biblioteca de Imagen (EmguCV).

Esta biblioteca será, como ya se ha explicado en el trabajo, la encargada de proveernos de las funciones necesarias para el procesamiento de imágenes. En nuestro caso, nos aportará la capacidad de detectar contornos, filtrar colores y detectar movimiento.

A continuación, procederemos a explicar las funciones clave utilizadas de esta biblioteca y sus parámetros, que aparecen en las funciones de la clase VisionArtificial.

Respecto a la parte de detección de movimiento:

- *BackgroundSubtractor.Apply*(InputArray, OutputArray)

Esta función actualiza el modelo de fondo a partir de la imagen de la cámara.

- *MotionHistory.Update(Mat)*

La función en cuestión actualiza el historial de movimiento a partir de la máscara de fondo captada en ese instante, determinando el movimiento de los pixeles dinámicos considerando los pixeles estáticos (mascara de fondo).

- *MotionHistory.GetMotionComponents(OutputArray, VectorOfRect)*

Función con la que obtenemos todos los componentes registrados por el historial de movimientos que define nuestro umbral de movimiento para considerar detección.

Respecto a la parte de detección de pieza, las funciones más relevantes son:

- *CvInvoke.HoughCircles(image, circles, method, dp, minD, CT, CAT, minR, maxR)*

Esta función es la encargada de detectar los círculos presentes en una imagen en escala de grises y requiere los siguientes argumentos que pasamos a definir:

- Image: Imagen que analizar.
- Circles: vector con la información de los círculos detectados.
- Method: representa el método utilizado para la detección y consiste en la transformada de Hough.
- Dp: Resolución del acumulador utilizado para detectar centros de los círculos.
- minD: mínima distancia posible entre centros de círculos.
- Cannythreshold: umbral superior para la detección de bordes.
- CircleAccumulatorThreshold: cuanto mayor sea este parámetro más falso positivos podrán ser detectados (segundo parámetro del método).
- MinR: radio mínimo a detectar.
- MaxR: radio máximo a detectar.

- *CvInvoke.Canny(image, edges, cannyThreshold, cannyThresholdLinking)*

Función que detecta los contornos presentes en la imagen en base a los umbrales que le pasamos en forma de parámetros de forma similar en el caso anterior.

- *CvInvoke.FindContours(image, contours, hierarchy, mode, method)*

Esta función recupera los contornos y su cantidad. Estos serán utilizados para el análisis de formas y el reconocimiento de objetos.

- Image: imagen procesada por la función *Canny*.
- Contours: contornos detectados en forma de vectores con información.
- Hierarchy: vector opcional con la información sobre la topología de la imagen.
- Mode: modo de recuperación.
- Method: método de aproximación.

- *ApproxPolyDP(curve, approxCurve, epsilon, closed)*

Función que aproxima unas curvas poligonales con cierta precisión. Con esta función, determinaremos la cantidad de contornos del polígono y la información que nos devuelve será los vértices de este polígono.

- Curve: vector de entrada de puntos 2D.
- ApproxCurve: resultado de aproximación.
- Epsilon: parámetro que especifica la precisión basada en la diferencia entre la curva original y su aproximación.
- Closed: especifica si la curva es cerrada o abierta.

Y por último, respecto a la detección de color de la función *AnalisisVisual* tenemos la siguiente función para filtrar color en formato HSV. HSV es un modelo de color basado en una transformación no lineal del espacio RGB que se determina por los valores Hue, Saturation y Value, que corresponden al color de la matriz, la saturación y el brillo.

- *CvInvoke.InRange(src, lower, upper, dst)*

Esta función convierte los pixeles que se encuentran en el rango de color los pone a 1 (blanco) y los pixeles que no en 0 (negro).

- Src: imagen a procesar.
- Lower: valor del rango inferior.
- Upper: valor del rango superior.
- Dst: imagen binaria de salida.