industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Sistema de visión coordinado con un robot para el control de calidad.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Javier Manjón Prado**

Director: Jorge Juan Feliu Battle

Codirector: José Luis Muñoz Lozano

Cartagena, julio de 2017

Universidad Politécnica de Cartagena

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

1.1 Justificación

La motivación principal de este Trabajo de Fin de Grado es la aplicación y profundización en la materia relacionada con la robótica y la visión artificial, ramas de conocimiento integran muchas de las disciplinas presentes en la ingeniería.

Al intentar aplicar estos conocimientos a un caso práctico, se pretende entrar en contacto con las distintas dificultades e imprevistos que puedan surgir y que no suelen ser tratados en la bibliografía de estas áreas.

1.2 Objetivos

El fin de este proyecto es el montaje, programación y puesta en marcha de un sistema de visión artificial coordinado con un brazo robótico, para la realización de manipulación tipo *pick & place*, con piezas sobre una cinta transportadora. Este tipo de aplicaciones son capaces de reemplazar el trabajo humano en tareas muy repetitivas, por lo que son muy demandadas por la industria.

Como material inicial, se dispone de:

- Una maqueta para prácticas, que contiene sensores y actuadores, además de una cinta trasportadora. Esta maqueta se encuentra deteriorada, y es necesaria su reparación.
- Un autómata programable modelo s7-200, desarrollado por la empresa Siemens.
- Una cámara web.
- Un brazo robótico SCORBOT-ER III, reparado por Rubén Jiménez Andreu durante la realización de su TFG Adaptación de un robot SCORBOT-ER III para su control usando Arduino.

Se distinguen varios objetivos claros que es necesario cumplir:

- Reparar la maqueta de prácticas.
- Programar y cablear el PLC correctamente para el control de la cinta transportadora.
- Programar el brazo robótico, ampliando y añadiendo características al software de control diseñado por Rubén Jiménez Andreu, en su TFG arriba mencionado.
- Realizar el montaje y programar los algoritmos necesarios para el correcto funcionamiento del sistema de visión.
- Diseñar las líneas de comunicación necesarias para que estos sistemas actúen de manera coordinada.

1.3 Estructura de la obra

La memoria de este proyecto se organiza en siete partes o capítulos:

- En el primer capítulo, en el que nos encontramos, se realiza una breve introducción al problema propuesto para su resolución en este proyecto, así como a los objetivos, metodología, etc.

- En el capítulo dos, se detalla la estructura física de la solución implementada.

- En el tercer capítulo se muestra la implementación de los algoritmos del sistema de visión artificial, así como de la parte de comunicaciones.

- El capítulo número cuatro trata sobre el brazo robótico SCORBOT-ER III. Se introduce el marco teórico con el que se trabajará para la programación de este, y se explica el software creado.

- En el quinto capítulo se encuentra toda la información concerniente a la puesta en marcha de la maqueta y la programación del PLC.

- El sexto capítulo contiene los resultados obtenidos a lo largo de la duración de este proyecto, se comparan distintos métodos utilizados para la programación del brazo robótico, índice de aciertos, etc.

- En el séptimo y último capítulo se relatan las conclusiones obtenidas y se proponen mejoras y otros posibles futuros trabajos a realizar como continuación de este proyecto.

1.4 Convenciones

Con el fin de facilitar la lectura, se ha evitado la inclusión de código de programación en el cuerpo principal de este trabajo de fin de grado. En vez de ello, este se puede hallar en los anexos que aparecen al final de este trabajo. Se usará la fuente Courier New para hacer referencia a nombres de funciones.

2. ESTRUCTURA FÍSICA

2.1 Estructura general

Dado que en este proyecto se pretende que varios sistemas actúen coordinadamente entre sí en un plano físico, es de gran importancia la situación de cada uno en el espacio de trabajo, ya que una mala disposición de los elementos puede dificultar o incluso hacer imposible la realización de la tarea para alguno de ellos.

El objetivo de este capitulo entonces, es presentar el material utilizado, así como sus características físicas, y posteriormente encontrar una situación para el robot y la cámara web respecto a la cinta transportadora que facilite sus funciones.

2.2 Maqueta de prácticas

Para la realización de este Trabajo de Fin de Grado, como ya se ha mencionado, se va a disponer de una maqueta de prácticas propiedad del Departamento de Ingeniería de Sistemas y Automática y usada para la realización de practicas en este Departamento. Esta contiene, entre otras cosas, una cinta transportadora, consistente en una cinta transfer movida por un motor DC de 24V y 1,8W de potencia, capaz de obtener un par de 4Nm. Este motor dispone a su vez de un sistema reductor para recudir velocidad y aumentar el par, y de un encoder magnético de efecto Hall que devuelve un pulso de 24V por cada revolución del motor. El motor cambia de dirección gracias a unos relees que invierten la polaridad de la alimentación, y la variación de la velocidad de obtiene variando dicha tensión de alimentación entre 0 y 24 V mediante un mando situado en la propia maqueta. Al final de la cinta se sitúa un sensor óptico BERO de la marca Siemens, capaz de detectar la presencia de objetos sobre la cinta.

Para el control de la cinta, en la misma maqueta está presente un panel de mando, con un interruptor, un pulsador, dos luces led y una seta de emergencia. La conexión con el PLC se hace mediante una placa de expansión de treinta y siete pines que replica la señales del modulo para ser accesibles mediante una bornera eléctrica de tornillos (DN37).

Figura 1: Maqueta de prácticas con cinta transportadora (arriba), panel de control (centro), motor DC (izquierda) y bornera eléctrica (derecha).

2.3 Ajustes en la maqueta de prácticas

Debido al continuo uso de esta maqueta, varios de sus componentes se hallaban dañados o incluso desaparecidos, por lo tanto, antes de continuar con el TFG, era necesaria su reparación y puesta en marcha.

El primer paso ha sido sustituir el cableado de la maqueta, que se encontraba desgastado y roto en determinadas partes, y era una de las causas de los fallos de funcionamiento en la maqueta. Aunque el cableado mas deteriorado era el del panel de control de la maqueta y el de la bornera eléctrica, se ha decidido sustituir todo para prevenir fallos futuros. Se ha colocado además un nuevo encoder óptico, cableándolo a la bornera, cuyas conexiones se pueden ver en el anexo A.

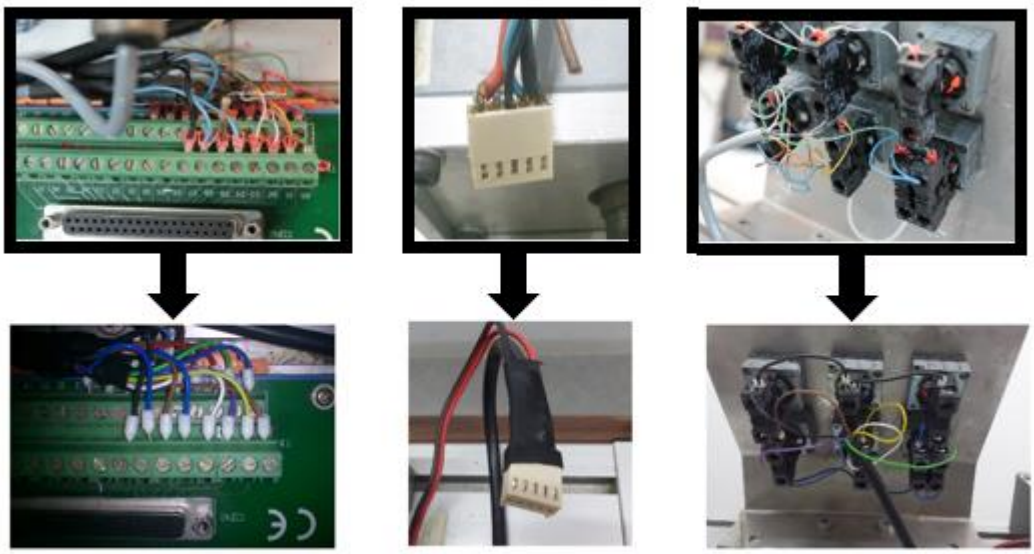


Figura 2: Ejemplo de los cambios realizados en el cableado de la maqueta.

Dado que la placa que contenía el encoder óptico se había desprendido, ha sido necesario volver a fabricar otra y soldarla a las conexiones del motor. La placa se ha creado a partir de los esquemáticos proporcionados por Pablo A. Martínez Ruiz, técnico del departamento de Ingeniería de Sistemas y Automática.

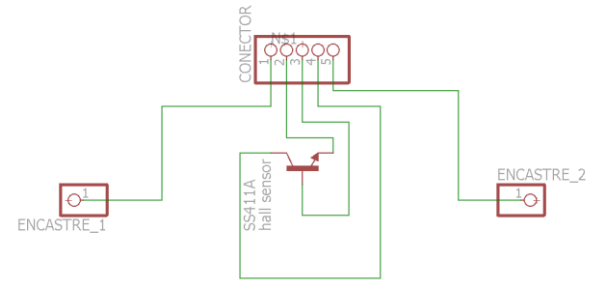


Figura 3: Esquemático de la placa del encoder.

2.4 Situación del robot

A continuación, se muestra un esquema de la cinta transportadora, con sus medidas expresadas en centímetros:

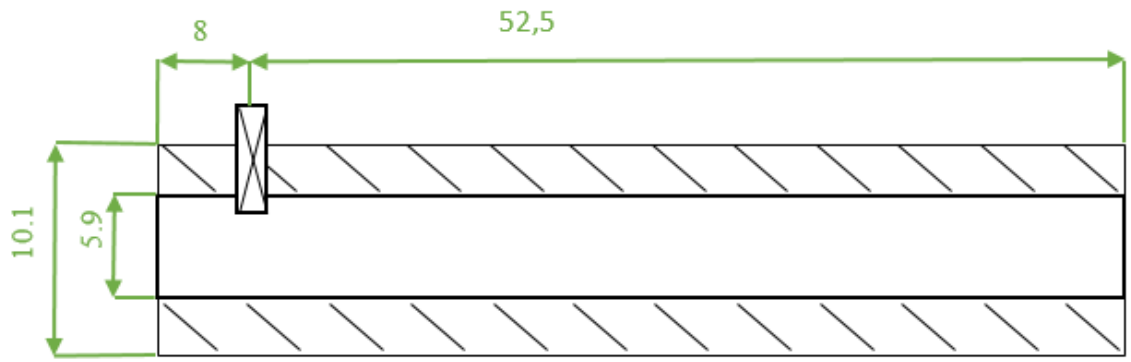
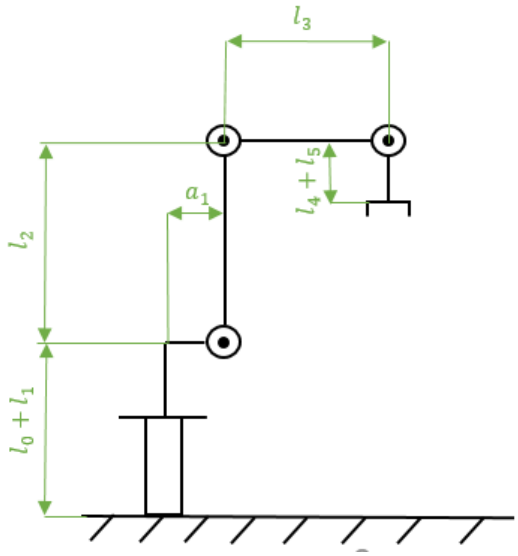


Figura 4: Esquema de cinta transportadora con medidas en cm.

La dirección en la cual se moverán los objetos es de derecha a izquierda. Como se puede observar, el sensor óptico se encuentra al final de la cinta transportadora, y marca el punto donde el SCORBOT-ER III debe recoger el objeto. Por lo tanto, este debe de estar en una posición que le permita acercarse y agarrar el objeto sin necesidad de forzar los motores situándose en una posición demasiado horizontal. También debe ser capaz de alcanzar las posiciones donde se desee situar los objetos una vez agarrados.

Otro aspecto importante del posicionamiento de robot, es que se debe evitar en la medida de lo posible que este se sitúe, durante la ejecución de su tarea, entre el objeto y la cámara web, ya que ello podría llevar a cometer errores en la resolución del problema de visión artificial, dando falsos positivos o negativos.

Por todo ello, es necesario conocer tanto las medidas del robot como su espacio de trabajo. Estas se han obtenido mediante la medición de todos los eslabones del robot y se exponen a continuación.



Nombre	Medida(cm)
l_0	21
l_1	14
l_2	22.5
l_3	22.3
l_4	8.5
l_5	6.5
a_1	2.5

Figura 5: Esquema del robot y tabla con sus medidas.

Además, hay que tener en cuenta también el giro angular máximo permitidos por las articulaciones del robot:

Articulación	Rango (º)
1	-80/140
2	-30/120
3	-120/120
4	0/180
5	Ilimitado

Figura 6: Tabla con límites de giro de las articulaciones.

De estas medidas se puede extraer que el alcance máximo del robot es la suma de las medidas de las articulaciones 1-5 y de a_1.

Teniendo estas medidas y limites angulares y sabiendo que el robot debe recoger un objeto en la cinta transportadora a la altura del sensor óptico, se pueden dibujar todas las posibles posiciones del robot. La imagen que describe estas es la siguiente, una vez eliminadas las zonas donde es imposible colocar el robot:

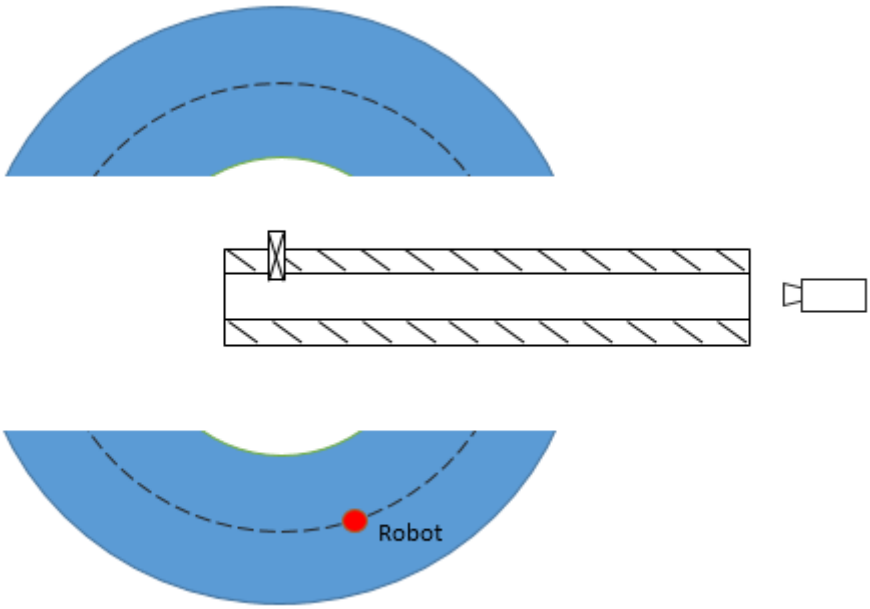


Figura 7: Esquema físico del sistema con posibles situaciones del brazo robótico (azul).

En la figura arriba mostrada, se ha dibujado una circunferencia cuyo radio es el alcance máximo del brazo robótico, centrada en el punto de recogida de los objetos. A esta se le ha restado otra circunferencia que representa la base del robot, dado que la base no puede quedar sobre el objeto. De las posibles posiciones restantes para el robot, se han eliminado también aquellas que sitúan al SCORBOT-ER III en la línea de visión objeto-cámara y aquellas posiciones en las cuales el robot puede reducir la eficacia de los algoritmos de visión artificial. Se ha dibujado también una línea de trazos señalando aquellas posiciones que menos forzarían las articulaciones del motor para recoger el objeto.

El resultado de este procedimiento es la imagen anterior. Para elegir el emplazamiento del robot de las posibilidades restantes, se ha decidido eliminar toda área por encima de la cinta transportadora, para que el sensor óptico no pueda obstaculizar la recogida de los objetos. La posición elegida se muestra con un punto rojo en la imagen.

2.5 Situación de la cámara web

La situación de la cámara web debe estar sobre todo condicionada a reducir la complejidad de los algoritmos de visión artificial que vayan a ser usados después. Es decir, se deben evitar en la medida de lo posible los problemas derivados de:

- **Punto de vista**, es decir, como esta orientado el objeto respecto al observador. Se debe usar un punto de vista tal que permita apreciar la orientación del objeto respecto al robot, para que este pueda agarrarlo.
- **Oclusión**, es decir, que el objeto de interés quede en un segundo plano, con algún otro objeto tapando totalmente o parcialmente características suyas al observador.
- **Escala**. Factores como la distancia o el ángulo pueden hacer que un objeto aparente ser mayor de lo que realmente es.
- **Fondo desordenado**. Puede ser mas complicado distinguir un objeto en un fondo caótico que en uno uniforme.

Aunque existen métodos capaces de obtener buenos resultados a pesar de estos problemas, suelen ser computacionalmente costosos, o difíciles de implementar. Además, suele dar mejores resultados evitar estos problemas que intentar resolverlos.

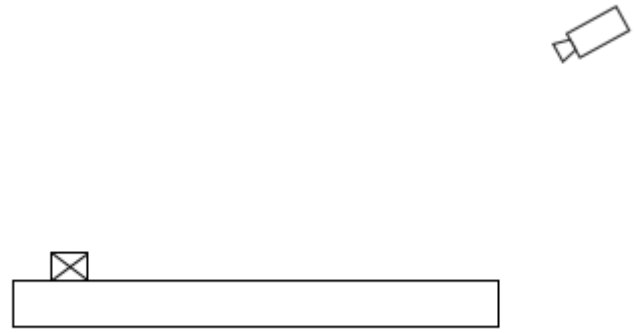


Figura 8: Esquema físico del sistema con situación de la webcam.

En la imagen arriba mostrada, se puede observar que se ha elegido posicionar la cámara a cierta altura sobre la cinta trasportadora, y alejada de esta hacia la izquierda. Al alejar la cámara de la cinta y elevarla por encima de esta, se reduce la variación del ángulo entre la cámara y el objeto, reduciendo con ello la variación de la imagen del objeto por el punto de vista y la escala, mientras las características deseables del objeto permanecen constantes y visibles. Aunque la posición ideal de la cámara para cumplir las funciones exigidas sería directamente sobre el objeto, la disponibilidad de espacio de trabajo ha hecho que se optase por la primera.

3. VISIÓN ARTIFICIAL

Se concibe la visión artificial como el "proceso de extracción de información del mundo físico a partir de imágenes usando para ello un computador". La entrada a un sistema de visión artificial suele ser una imagen o conjunto de imágenes de la escena que se pretende describir, y su salida es la descripción de esa escena, que contiene la información requerida por el usuario.

Un sistema de visión artificial, por lo tanto, actúa sobre una representación bidimensional del mundo, interpretándola y analizándola, lo que le proporciona información sobre brillo, colores, formas, etc. Esta representación suele estar en forma de imágenes estáticas, dinámicas o escenas tridimensionales.

3.1 Introducción histórica a la Visión Artificial

Se puede empezar a hablar de visión artificial a partir de 1961, cuando Larry Roberts, el creador de ARPA net, crea un programa llamado "el mundo de microbloques", con el cual un robot puede extraer las características necesarias de una estructura de bloques situada sobre una mesa para reproducirla desde otras perspectivas, con lo cual se demuestra que la información proveniente de la cámara ha sido mandada y procesada adecuadamente por el ordenador.

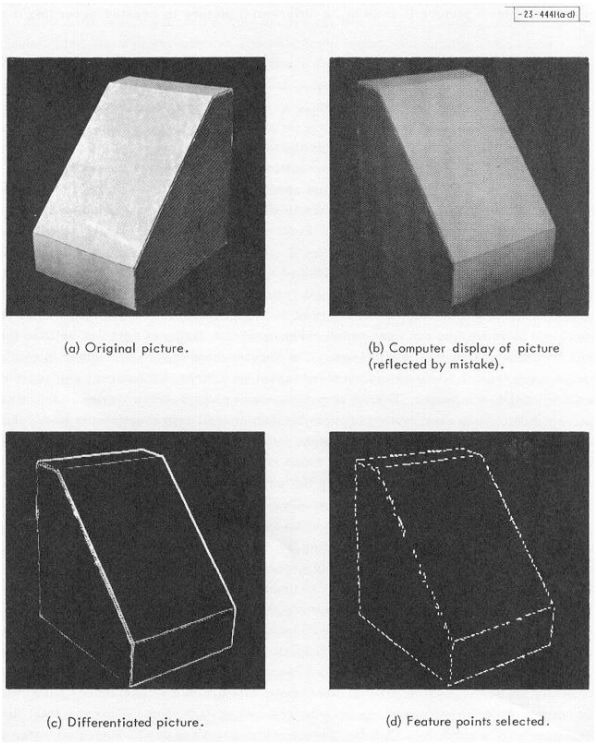


Figura 9: Figuras utilizadas en el programa de "el mundo de microbloques". *www.packet.cc.*

En los primeros años hubo un gran optimismo respecto a la visión artificial, en parte debido al corto tiempo de desarrollo, a la gran confianza en la potencia de computación de los ordenadores y en que la visión era una tarea sencilla para el ser humano. A pesar de todo, se consiguieron muy pocos avances. Factores como el desconocimiento del proceso de visión en el propio ser humano, o la infinidad de posibles soluciones al obtener la información de una proyección bidimensional de objetos tridimensionales llevaron a reconocer la dificultad de la tarea.

En la década de los setenta, se cambia el nombre a "Visión por Computador", y se proponen metas mas realistas, centrándose la investigación entonces en obtener la estructura tridimensional del mundo a partir de imágenes como paso previo hacia el reconocimiento de objetos en escenas, ya fuese mediante la extracción de bordes, el análisis de sombras y variaciones de intensidad, las imágenes en estéreo, etc.

Es en 1990 cuando la visión por computador empieza a aparecer en la industria, llegando a aparecer la visión artificial industrial. Empiezan a aparecer empresas especializadas en tanto la venta de hardware como de software de visión artificial.

La aplicación e la visión artificial a la robótica fue un elemento esencial que permitió a los robots obtener una mayor adaptabilidad, eficacia y flexibilidad, ya que permite una retroalimentación sensorial mayor que la que permiten otros sensores.

A partir del cambio de siglo, la investigación se ha centrado sobre todo en las técnicas basadas en la extracción de características para el reconocimiento de objetos, características basadas en puntos de interés y regiones o contornos. Últimamente, además, se han empezado a aplicar técnicas de inteligencia artificial a la visión por computador, favorecido este hecho por la enorme de cantidad de imágenes etiquetadas disponibles gracias a internet.

Figura 10: Ejemplo de algoritmo SURF, que detecta y extrae puntos de interés de una imagen.

Actualmente las aplicaciones de la visión por computador son enormes, apareciendo en áreas tales como la agricultura, la robótica, el control de calidad, la seguridad, biomedicina…

3.2 Marco teórico

3.2.1 Definiciones básicas

Imagen

"Una imagen es la representación bidimensional de una escena del mundo tridimensional. Una imagen es el resultado de la adquisición de una señal proporcionada por un sensor que convierte la información del espectro electromagnético en codificaciones numéricas." (Conceptos y Métodos en Visión por Computador. Enrique Alegre, Gonzalo Pajares y Arturo de la Escalera. 2016).

Una imagen digital es una matriz de M x N dimensiones cuyos valores representan los niveles de oscuridad o claridad de una escena. A los elementos de esta matriz se les denomina como *píxeles.*

Histograma de la imagen

El histograma de una imagen es un diagrama de barras en el que cada barra tiene una altura proporcional al numero de pixeles que hay para un determinado nivel de intensidad. En el eje de abscisas se suelen poner los diferentes niveles de intensidad que pueden tener los pixeles, mientras que el eje de coordenadas refleja el numero de pixeles que tienen ese nivel en concreto.

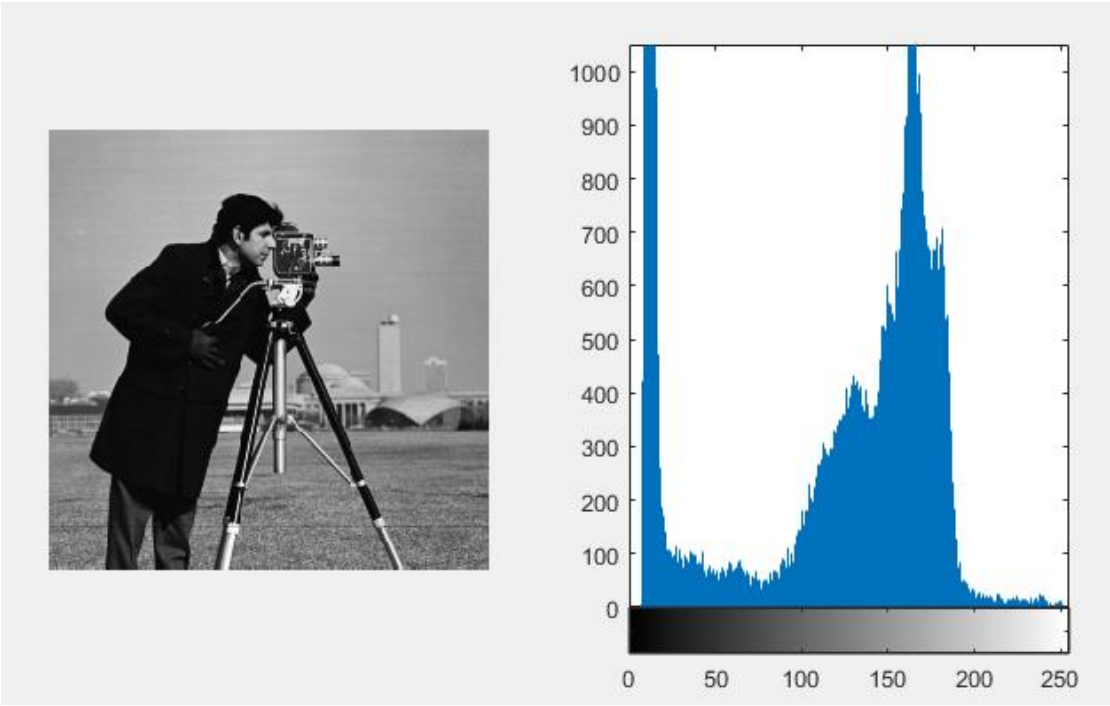


Figura 11: Imagen y su histograma.

3.2.2 Etapas de un sistema de visión artificial

La visión artificial, define cuatro fases principales, resumen de las cuales se escribe a continuación:

1. Adquisición o captura de imágenes
   Esta primera etapa es puramente sensorial y consiste en la captura y transmisión de imágenes desde un dispositivo sensor como una cámara al ordenador. Factores importantes en esta etapa son:
   a. Iluminación
      Para el diseño de un correcto sistema de iluminación, este debe permitir resaltar los rasgos de interés del objeto y reducir la complejidad de la imagen para su posterior análisis, lo que reduce el tiempo requerido para procesar la imagen.
   b. Cámara
   c. Interfaz ordenador-sistema sensor
2. Pre-procesamiento
   Suele ser necesario un tratamiento previo al procesado de la imagen en si para corregir en la medida de lo posible los errores derivados de la adquisición de la imagen por el dispositivo sensor y facilitar las etapas posteriores. Se busca con ello reducir el ruido presente en la imagen, realzar los detalles o características de interés, rotaciones, conversiones a escala de gris, mejora del contraste, etc.
3. Procesamiento o segmentación
   El objetivo final de esta etapa es aislar los elementos característicos del objeto o escena. Es decir, los elementos cuyas características son requeridas por el usuario.
4. Reconocimiento o clasificación
   En esta ultima etapa, se extraen las características de los objetos, procediendo después a su clasificación.

Figura 12: Etapas de un sistema de visión artificial.

3.2.3 Preprocesado. Filtrado y realzado de imagen

A continuación, se ven algunas de las operaciones y transformación que se suelen aplicar sobre la imagen en la etapa de preprocesado, haciendo especial hincapié en los métodos que serán usados en este proyecto:

- **Operaciones aritmético lógicas**, como la conjunción, disyunción, suma, resta, negación, multiplicación y división. De estas, la resta suele ser muy utilizada, a pesar de su sencillez, para realizar operaciones como la substracción de fondo en la detección de objetos en una secuencia de video.

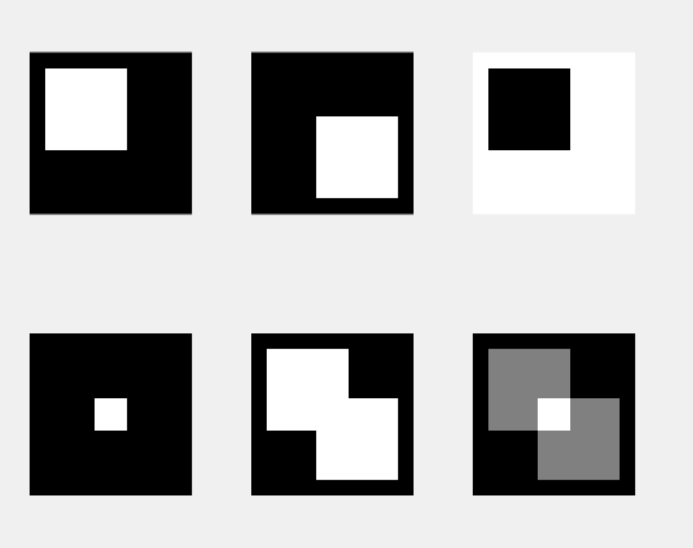


Figura 13: Operaciones aritmético lógicas en imágenes binarias. Arriba, de izquierda a derecha: A, B, -A. Abajo: A&B, A|B, (A+B)/2.

- **Operaciones geométricas**, como la traslación, el escalado o la rotación.

- **Operaciones sobre el histograma** como el aumento o la reducción del contraste, el ecualizado del histograma, etc.

- **Operaciones en el dominio de la frecuencia.**

- **Filtros espaciales**. Se realizan directamente sobre la imagen, los mas usados son los filtros de convolución y correlación. También existen los filtros de suavizado, de obtención de contornos, de la laplaciana, etc.

- **Operaciones morfológicas**, como la dilatación, la erosión, la apertura y el cierre, la coincidencia estructural…

3.2.3.1 Filtros de convolución y correlación. Filtros de suavizado.

La convolución es una operación matemática aplicada entre dos funciones usada para caracterizar el grado de solapamiento entre ellas. A continuación, se muestra la ecuación tanto de la convolución como de la correlación.

Correlación:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)F(i+u, j+v)$$

Convolución:

$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H(u,v)F(i-u, j-v)$$

Como se puede ver, la convolución y la correlación están muy relacionadas, con una única diferencia significativa, el signo negativo de las variables u y v. El efecto de esa variación puede observarse con un sencillo ejemplo:

Si se supone un *kernel* H de tamaño 3x3 aplicado sobre una imagen F del mismo tamaño,

$$H = \begin{bmatrix} A_1 & B_1 & C_1 \\ D_1 & E_1 & F_1 \\ G_1 & H_1 & I_1 \end{bmatrix}, \qquad F = \begin{bmatrix} A_2 & B_2 & C_2 \\ D_2 & E_2 & F_2 \\ G_2 & H_2 & I_2 \end{bmatrix}$$

al realizar una convolución, se obtiene lo siguiente.

$$H \otimes F = A_1I_2 + B_1H_2 + C_1G_2 + D_1F_2 + E_1E_2 + F_1D_2 + G_1C_2 + H_1B_2 + I_1A_2$$

En cambio, al realizar una correlación el resultado en cambio es,

$$G = A_1A_2 + B_1B_2 + C_1C_2 + D_1D_2 + E_1E_2 + F_1F_2 + G_1G_2 + H_1H_2 + I_1I_2$$

En este caso, los elementos cada elemento de la imagen esta multiplicado por el elemento del *kernel* situado en la misma posición. Se puede observar también, que esta misma solución puede ser obtenida aplicando una convolución con un *kernel* invertido.

Un filtro muy utilizado que aplica estos principios es el de suavizado, cuyo objetivo es reducir las variaciones de intensidad en pixeles debido a errores en la cámara, etc. Para ello, este filtro aplica un *kernel* que realiza una operación de promedio por toda la imagen, de manera que cada píxel obtiene un nuevo valor que depende tanto de su anterior valor como de los valores de los pixeles adyacentes.

Aunque existen varios tipos de filtros usados para el suavizado de una imagen, uno de los mas utilizados es el filtro de la gaussiana. Este filtro da mas importancia al píxel central a la hora de calcular el nuevo valor del pixel, obteniendo con ello mejores resultados.

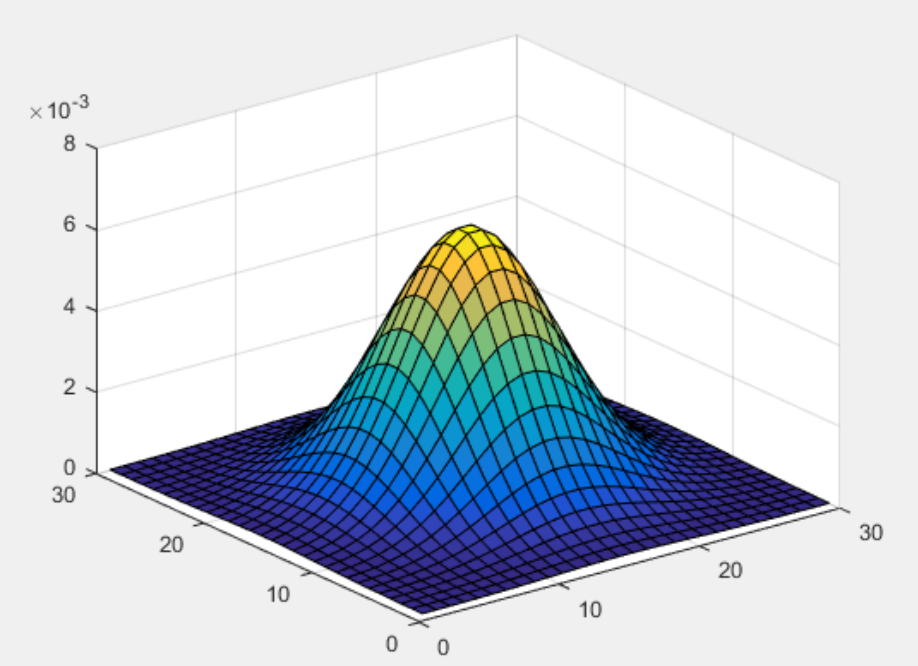


Figura 14: Representación gráfica de un kernel gaussiano, con la altura proporcional al valor del píxel.

Figura 15: Ejemplo de la aplicación de un filtro gaussiano a una imagen para su suavizado.

3.2.3.2 Operaciones morfológicas.

Existen varias operaciones utilizadas durante este proyecto que corresponden al marco de operaciones morfológicas. Estas son:

- Erosión y dilatación.
- Apertura y cierre.

La dilatación consiste en la aplicación de un elemento estructurante a cada pixel de una imagen, si la intersección del elemento estructurante con el conjunto de pixeles de la imagen es no vacía, entonces el pixel sobre el que se realiza la operación pertenece a la imagen dilatada.

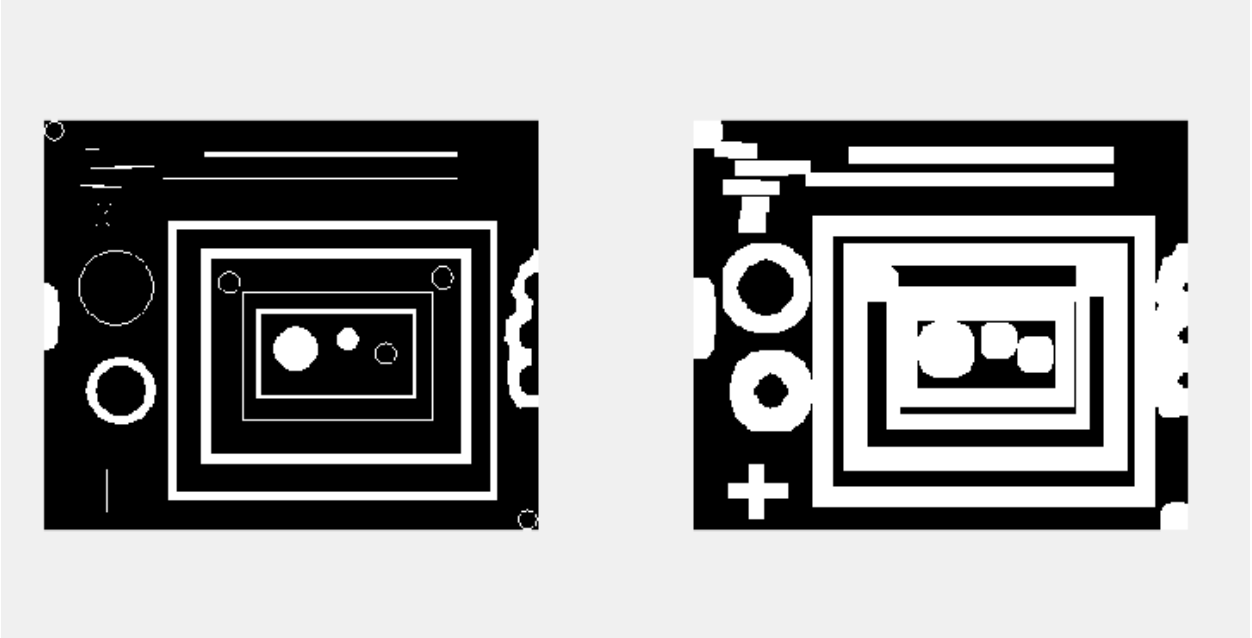


Figura 16: Ejemplo de dilatación (derecha) e imagen original (izquierda).

Como se puede ver en la imagen, la dilatación es una operación con la que se aumenta el tamaño de la imagen y de los objetos que la componen, pero a costa de perder detalles en ella. Se rellenan huecos en la imagen, además de fusionar objetos cercanos entre si, etc.

La erosión es la inversa de la dilatación, reduciendo el tamaño de los objetos en vez de aumentarlos. Al hacerlo, se eliminan objetos pequeños, se separan objetos grandes, se tiende a agrandar los huecos en las imágenes, etc.

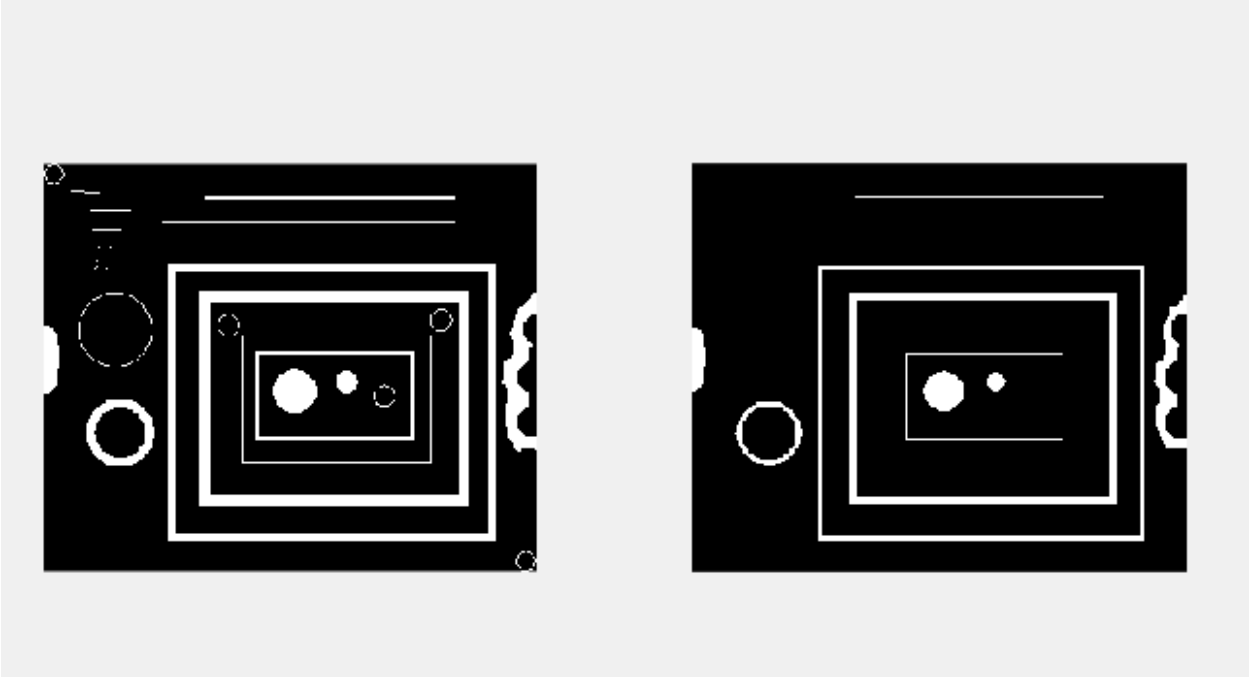


Figura 17: Ejemplo de erosión (derecha) e imagen original (izquierda).

A la aplicación de una erosión seguida de una dilatación utilizando el mismo elemento estructurante se le denomina **apertura**. Esta operación tiene la propiedad de eliminar objetos pequeños gracias al proceso de erosión, pero recuperando después el tamaño del resto de objetos debido a la posterior dilatación.

En el proceso de **cierre**, en cambio, se aplica primero una dilatación y después una erosión. Gracias al proceso de cierre se eliminan huecos en imágenes, pero evitando que la imagen en general crezca, gracias a la erosión.

3.2.4 Segmentación

El proceso de segmentación consiste en la división de una imagen en regiones homogéneas con respecto a una o mas características, con el fin de facilitar un posterior análisis. Puede ser, por ejemplo, la localización y extracción de las caras de las personas de una multitud para su posterior identificación, o la extracción de las líneas de texto de una pagina para su reconocimiento y traducción. Existen diversas técnicas para ello que serán mencionadas a continuación.

- Segmentación basada en umbralización.
- Segmentación basada en la detección de contornos.
- Basada en el crecimiento de regiones.
- Otros enfoques, como la umbralización basada en color, en textura, etc.

La umbralización es un proceso que permite convertir una imagen en escala de grises o en color en una imagen binaria, de tal forma que solo los pixeles cuya intensidad sea mayor que un cierto umbral, también llamado *threshold*, tengan valor uno en la imagen binaria. De esta forma, los pixeles pertenecientes a los objetos de interés tendrán valor distinto de los pixeles del fondo.

Los procesos basados en umbralización como único medio de segmentación suelen resultar ineficaces en muchos problemas reales, ya que muy a menudo la intensidad de los pixeles de fondo no varía lo suficiente con respecto a la intensidad de los pixeles de interés. Aun así, este proceso tiene buenos resultados si es combinado con otras técnicas de segmentación.

El algoritmo de etiquetado es una técnica de segmentación basada en la detección de contornos. Este usa la información proveniente de las fronteras de los objetos para etiquetar los distintos objetos de una imagen de manera que se pueda segmentar esta quedando solo uno de ellos a elección del usuario. Aunque muy útil, el algoritmo de etiquetado es sencillo, y consiste en los siguientes pasos:

1. Barrido de la imagen asignando una etiqueta distinta a cada pixel con valor distinto de cero encontrado. Si en la vecindad del pixel existe algún pixel que ya haya sido etiquetado, este usara la misma etiqueta. Si en la vecindad del pixel existen dos pixeles ya etiquetados con etiquetas distintas, se apunta la igualdad.
2. Se resuelven las igualdades encontradas en el paso anterior.

3.2.5 Análisis de las características de los objetos.

Una vez segmentada la imagen y obtenidos los distintos objetos de interés que la componen, es posible calcular características de estos, como pueden ser el área, perímetro, circularidad, centro de gravedad, orientación, etc. En este proyecto, se han utilizado técnicas para extraer tanto la posición, como el área y la orientación de los objetos.

Se suele entender el área como el numero de pixeles que componen cierto objeto, y cuyo contaje es muy simple. Este método permite posteriormente filtrar la imagen, eliminando todos los objetos cuya área sea menor o mayor que la deseada.

Para el cálculo tanto de la posición como de la orientación de un objeto en una imagen binaria se recurre a la teoría de los momentos generales. Los momentos generales discretos de un objeto son:

$$M_{ij} = \sum_{x} \sum_{y} x^i y^j I(x, y)$$

A partir de esta ecuación, se podría calcular el momento general cuando $i = 0, j = 0$, es decir, M_{00}. Se puede comprobar que este momento es el equivalente al área del objeto.

Los momentos de orden uno, M_{10} y M_{01}, junto con M_{00} determinan en cambio el centro de gravedad del objeto.

$$x_c = \frac{M_{10}}{M_{00}} = \frac{\sum_x \sum_y x \ I(x,y)}{\sum_x \sum_y I(x,y)}$$

$$y_c = \frac{M_{01}}{M_{00}} = \frac{\sum_x \sum_y y \ I(x,y)}{\sum_x \sum_y I(x,y)}$$

Para el calculo de la orientación es necesario obtener la matriz de covarianza de los momentos centrales del objeto, cuyos eigenvectores son los ejes de una elipse. El eigenvector con el mayor eigenvalor asociado será el eje mayor, y ese eje es el usado para calcular la orientación del objeto, que se puede realizar mediante la siguiente ecuación:

$$\theta = \frac{1}{2}\arctan\left(\frac{2(M_{11} - x_C M_{01})}{(M_{20} - x_C M_{10}) - (M_{02} - y_C M_{01})}\right)$$

3.3 Resolución del problema de Visión Artificial

Este trabajo presenta una solución al problema de visión artificial utilizando algoritmos y funciones establecidas por Matlab.

Matlab (abreviatura de *Matrix Laboratory*) es una herramienta de software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Destaca especialmente en la sencillez en la manipulación de matrices y representación de datos y creación de funciones. Por ello, es un software muy utilizado y conocido en la comunidad científica, especialmente en el campo de la ingeniería.

Una de las ventajas de Matlab es la gran cantidad de *toolboxes* que añaden funcionalidad. Entre otras, para la programación de las rutinas relacionadas con la visión artificial, se ha utilizado la *Image Processing Toolbox*.

Según la propia página web de Matlab, esta "proporciona un conjunto completo de algoritmos, funciones y aplicaciones de referencia estándar para el procesamiento, el análisis y la visualización de imágenes, así como para el desarrollo de algoritmos. Puede llevar a cabo análisis de imágenes, segmentación de imágenes, mejora de imágenes, reducción de ruido, transformaciones geométricas y registro de imágenes".

El objetivo de este apartado es crear una función que pueda ser llamada por la rutina principal encargada de controlar el robot, para monitorizar constantemente el color, la orientación y la posición de los objetos que pueda haber sobre la cinta transportadora. De esa manera, en caso de que uno llegue al final del recorrido de la cinta, podrá ser recogido por el brazo robótico gracias a la información proporcionada por esta función.

Por lo tanto, el funcionamiento general de esta aplicación será:

1. Obtener imagen mediante la cámara web.
2. Preprocesar, segmentar y extraer las características de interés de la imagen.

El primer paso es sencillo, existiendo funciones en Matlab que son capaces de crear un objeto vinculado a la *webcam* y permitiendo al usuario extraer "capturas de pantalla" mediante el comando `snapshot`. Mediante este comando, se obtiene entonces una imagen similar a la de la figura 18.

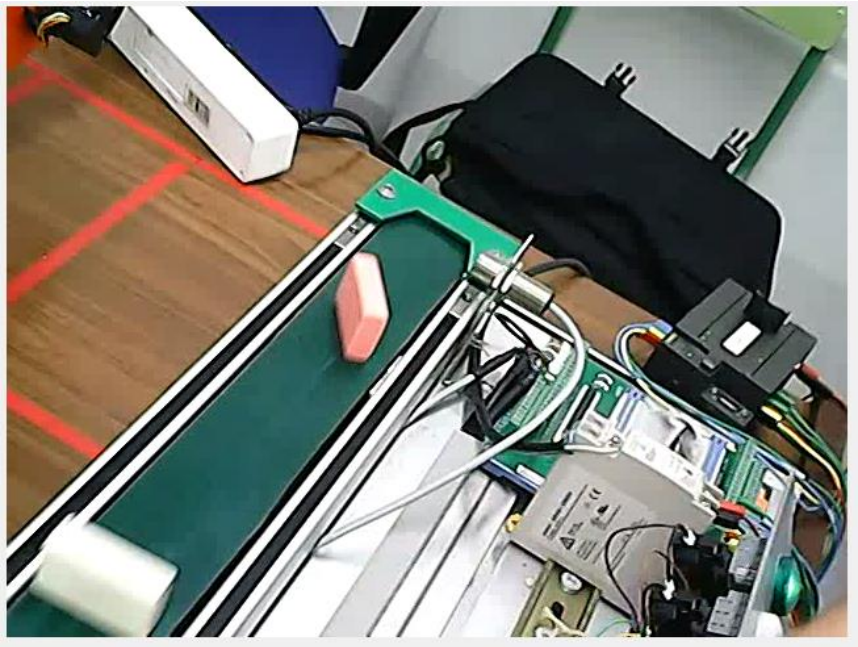


Figura 18: Imagen ejemplo obtenida de la cámara web.

Es sobre esta imagen sobre la cual se aplicarán los algoritmos mencionados en el apartado anterior hasta obtener la información deseada de la imagen. El primero de los cuales es simplemente realizar una rotación y un recorte de la imagen de manera que se elimine en la medida de lo posible todo fondo que no corresponda al área de la cinta transportadora.

Posteriormente se convierte la imagen en escala de grises, se le aplica un filtro gaussiano para reducir el ruido y se convierte entonces a una imagen binaria.

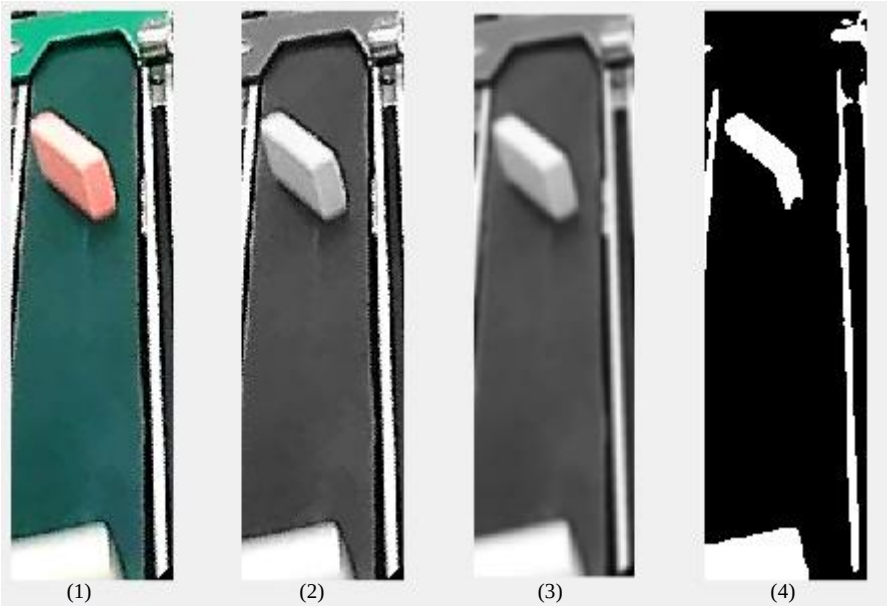


Figura 19: Primera fase del tratamiento de la imagen. En (1) se muestra la imagen una vez recortada. En (2) una vez convertida en escala de grises. En (3) suavizada. Y en (4) umbralizada.

Aunque existen métodos aplicables en esta parte de la solución para reducir las variaciones en la intensidad de la imagen o en el color creadas por la iluminación, como el paso a coordenadas cromáticas o el algoritmo *White Patch*, se ha observado que en este caso tenían poco efecto en general sobre la imagen, decidiendo evitarlas también para reducir el tiempo de computación requerido por la función.

Como se puede ver en la última imagen de la figura 19, aunque se puede apreciar la forma de los objetos claramente sobre la superficie de la cinta transportadora, aun quedan también otras zonas de la imagen que tras la segmentación por umbralización han quedado en blanco. También se puede observar que la principal característica que diferencia a estas regiones de las regiones pertenecientes a los objetos de interés es que las primeras son largas, estrechas y verticales.

Es por ello que se decide usar un elemento estructurante rectangular con una base mayor que su altura para realizar una apertura y cierre de la imagen. De esa manera, los elementos que no puedan contener ese rectángulo, aquellos cuya altura es mayor que su base, quedan eliminados.

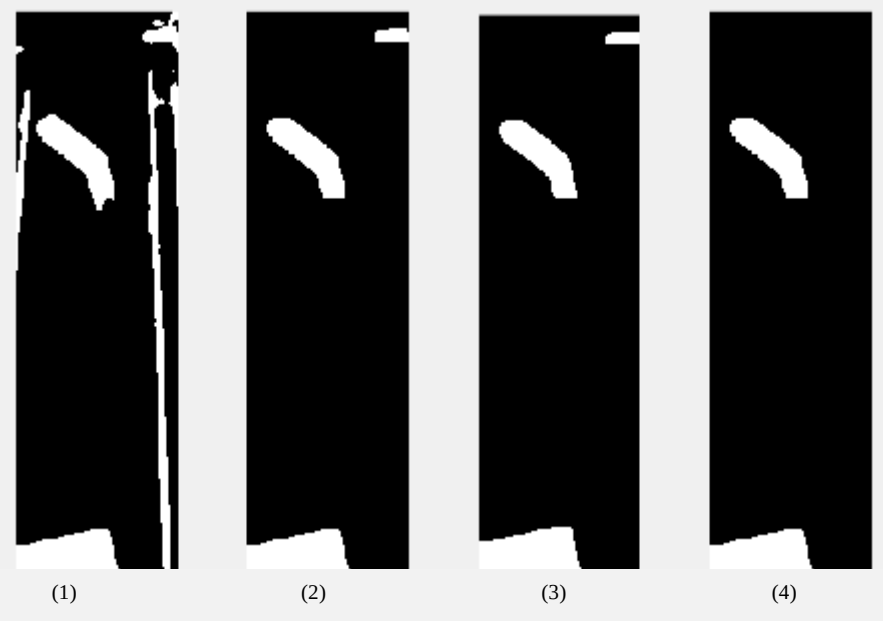


Figura 20: Segunda fase del tratamiento de la imagen. Partiendo de la imagen obtenida anteriormente (1), se realiza una apertura (2) y cierre (3), seguidas de un filtro de tamaño (4).

Como último paso, se añade un filtro de tamaño, que elimina la región situada en la parte superior derecha de la imagen, y que corresponde al reflejo de la luz sobre el sensor óptico de la cinta transportadora. Este elemento no se ha eliminado al realizar la apertura y cierre de la imagen, ya que por su forma es mas similar a las regiones de interés que a las regiones cuya eliminación se buscaba mediante estos procedimientos. Dado que, aunque la forma es similar, la diferencia de tamaño es significativa, es posible eliminar este objeto mediante el filtro antes mencionado.

Una vez obtenida esta última imagen, ya es posible realizar un proceso de etiquetado de regiones, y calcular los centroides de todas las regiones etiquetadas. La región de interés es la superior, ya que esa es la que llegará antes a la zona de recogida marcada y la primera que deberá ser cogida por el brazo robótico. De los objetos se extrae también la característica de orientación. Realizar este paso en Matlab es sencillo, debido a que se dispone de la función `regionprops`, que permite extraer las características que el usuario previamente seleccione de todo el conjunto de regiones de una imagen.

Para reconocer el color del objeto, se extrae de la imagen este objeto usando el *Bounding Box*, término con el cual se referencia al rectángulo de menor tamaño que engloba todo el objeto.

Extrayendo ese rectángulo tanto de la imagen a color como de la imagen en binario es posible multiplicarlos para obtener una imagen que solo muestre el color de la goma, y no el color del fondo. Una vez hecho eso es simple obtener una media del color de toda la región, que será el color de la goma, expresado en coordenadas RGB.

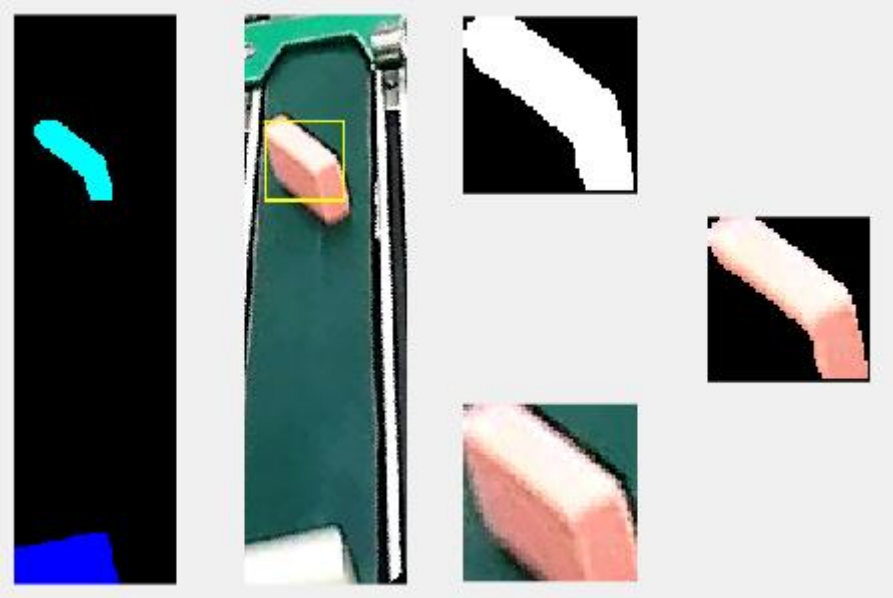


Figura 21: Tercera fase del tratamiento. Se etiquetan las regiones y se localiza la superior. Posteriormente se extrae el color de dicha región (a la derecha).

Para poder clasificar el color de la región, se han calculado las coordenadas de color RGB de varias muestras y se han dibujado los puntos correspondientes en formato *xyz*. El objetivo es encontrar un plano que separe las muestras de un color de las de otro, de tal manera que:

$$\begin{cases} Ax + By + Cz + D < 0, & Color\ A \\ Ax + By + Cz + D = 0, & Indefinido \\ Ax + By + Cz + D > 0, & Color\ B \end{cases}$$

El plano seleccionado, que se puede observar en la figura 22, es de la forma:

$$-2x + y + 262 = 0$$

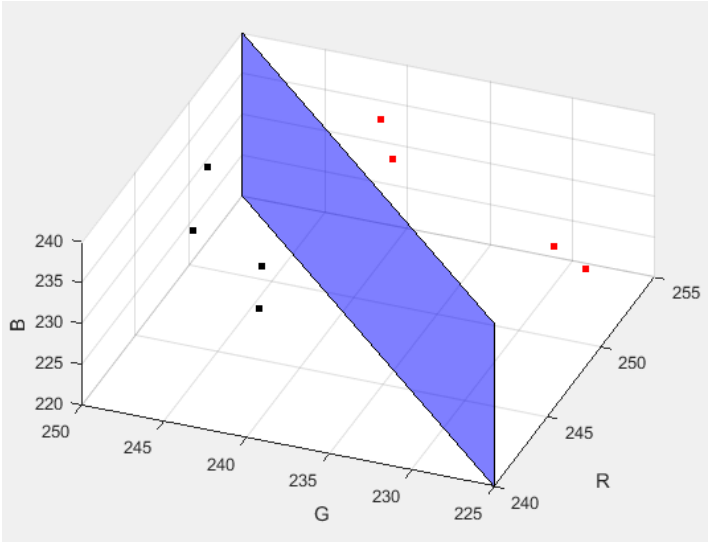


Figura 22: Se muestra el plano (en azul) obtenido que separa las muestras de ejemplo rojas y negras.

Los valores RGB de la media obtenida de la región de la goma serán sustituidos en la anterior ecuación, y según el signo del resultado, la región será clasificada como de un color u otro.

Una vez obtenida toda la información, la función termina su ejecución, devolviendo como parámetros de salida la orientación y posición del objeto superior, además de su color y una imagen mostrando la goma detectada, para fines de visualización.

4. BRAZO ROBÓTICO

4.1 Introducción histórica a la Robótica Industrial

Las bases del primer robot industrial moderno se remontan a 1954, año en que George C. Devol (1912-2011), ingeniero norteamericano, concibe la idea de un dispositivo de "trasferencia de artículos programada", que se patentó en 1961.

En 1956, junto con Joseph F. Engelberger (1925-2015), un ávido lector de Isaac Asimov, empieza a trabajar en la utilización industrial de sus máquinas, fundando la Consolidated Controls Corporation, más tarde llamada Unimation (Universal Automation), la primera empresa de robótica de la historia.

La primera máquina, Unimate, se instaló en 1960 en la fábrica de General Motors de Trenton, en Nueva Jersey. Esta se usaba para transportar piezas fundidas en molde hasta la cadena de montaje y soldar esas partes sobre el chasis del vehículo. Esta tarea era muy peligrosa para los trabajadores, quienes hasta entonces se exponían diariamente a inhalar los gases de combustión de la soldadura o perder un miembro. El hecho de que el primer robot se aplicase a un trabajo peligroso para los humanos no fue casual, Joseph F. Engelberger lo eligió motivado por la primera de las Tres Leyes de la Robótica de Asimov: Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.

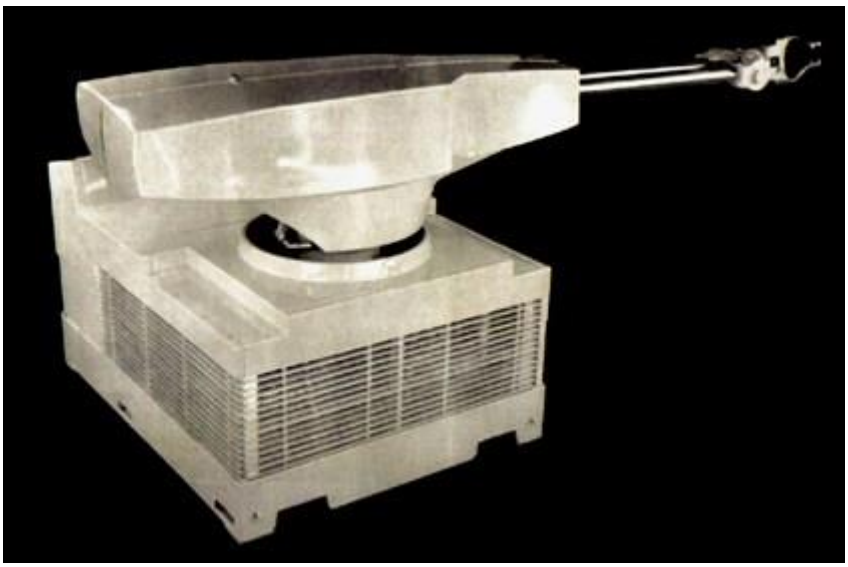


Figura 23: Unimate, primer robot industrial.
http://correll.cs.colorado.edu.

En poco tiempo, el Unimate era producido en masa. En 1973 se construía el primer robot con accionamiento eléctrico, el IRb6 (fabricado por ASEA). El avance de la tecnología robótica era ya vertiginoso, gracias sobre todo a los siguientes beneficios:

- Productividad
- Flexibilidad
- Calidad
- Seguridad

Actualmente, la venta e instalación de sistemas robóticos sigue aumentando, sinónimo de progreso y desarrollo, llegando a ser imprescindible en ciertas industrias, como la automovilística o la electrónica, que suman el 70% del parque de robots del mundo[1].

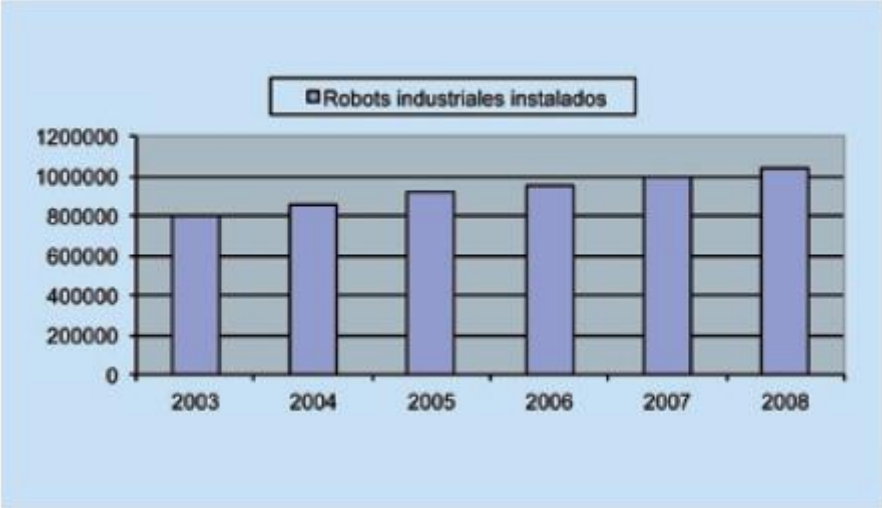


Figura 24[1]: Número de robots industriales instalados según el año.

Los esfuerzos en investigación en cuanto a la robótica industrial intentan conseguir la integración plena en los procesos de fabricación, para lo cual se busca desarrollar nuevos sistemas de manipulación y agarre, métodos de sincronización de robots, métodos para la cooperación con operarios, interfaces avanzadas, tecnología de sensores, métodos de detección de fallos y recuperación, etc.

4.2 Marco teórico

4.2.1 Definición y clasificación de los robots industriales

La definición formal de lo que es un robot industrial proporcionada por la *Organización Internacional de Estándares (ISO)* es la siguiente:

"Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas."

4.2.2 Estructura mecánica de un robot

Un robot está formado por una serie de elementos o eslabones unidos mediante articulaciones que permiten un movimiento relativo entre cada par de eslabones consecutivos. El movimiento de cada articulación puede ser de desplazamiento, giro, o de una combinación de ambos, como se muestra en la figura 25.

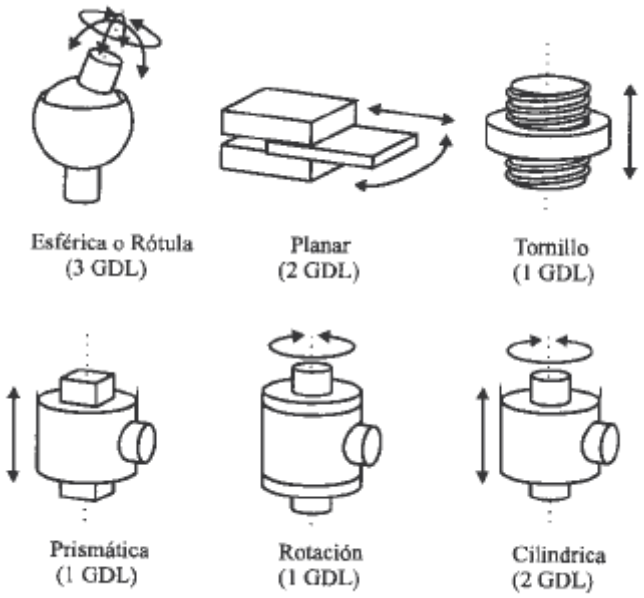


Figura 25[2]: Diferentes tipos de articulación según su movimiento relativo.

Cada movimiento independiente que puede realizar cada articulación con respecto a la anterior se denomina **grado de libertad** (GDL). El número de grados de libertad de un robot vendrá dado por la suma de los grados de libertad de las articulaciones que lo componen.

Nótese que para que el extremo de un robot pueda ser posicionado y orientado de cualquier manera dentro del área de trabajo, son necesarios seis parámetros, tres para posición y tres para orientación, es decir, un mínimo de seis GDL. En caso de haber obstáculos en el área de trabajo, querer

aumentar el volumen del espacio accesible o prever fallos en el actuador de una articulación, serán necesarios más grados de libertad.

4.2.3 Actuadores y sensores

El objetivo de los actuadores de un robot es generar el movimiento relativo entre las articulaciones de este según las órdenes dadas por el sistema de control. Los actuadores utilizados en robótica usan los siguientes tipos de energía:

- Energía neumática
- Energía hidráulica
- Energía eléctrica

De estas, la última es la más usada actualmente, los actuadores más usados son motores de corriente continua, aunque también aparecen en los últimos años motores de corriente alterna o motores paso a paso.

Además de los actuadores, para poder cerrar el lazo de control es necesario añadir sensores. Estos son necesarios para que el robot pueda realizar su tarea con precisión y velocidad. Los sensores internos más fundamentales son los relativos a posición. Para realizar la sensorización de posición en brazos robóticos se suelen usar encoders y resolvers.

Un encoder incremental es básicamente un disco trasparente con una serie de marcas opacas colocadas radialmente y equidistantes entre sí. A ambos lados de este disco se sitúan un elemento fotoemisor y otro fotorreceptor. El eje rotatorio cuya posición se pretende medir va unido a este disco. A metida que el eje rota, la luz generada por el fotoemisor atraviesa las marcas, por lo que el fotorreceptor situado al otro lado la capta y emite pulsos. Llevando una cuenta de los pulsos emitidos se puede conocer la posición angular del eje.

Para poder saber en qué dirección está girando el eje asociado al encoder, se suele incluir una segunda pareja de fotoemisor-fotorreceptor, desplazada con respecto a la primera de manera que los pulsos estén desfasados noventa grados eléctricos.

4.2.4 Representación de la posición y orientación. Matrices de transformación homogénea

El método más usual de representación de la posición son las coordenadas cartesianas, que son muy útiles a la hora de representar la posición de puntos en el espacio. El problema que aparece cuando se intentan representar sólidos, como los eslabones de un brazo robótico, es que se hace necesario no solo representar su posición, sino también su orientación. Esta viene definida por tres grados de libertad, al igual que la posición (x, y, z).

Para poder efectuar esta representación de forma sencilla, se suele asignar un nuevo sistema de referencia cartesiano al solido en cuestión, y describir la serie de translaciones y rotaciones que relacionan los dos sistemas.

Surge, como método para describir esta relación entre sistemas de referencia, la **matriz de trasformación homogénea**.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix}$$

Como se puede ver en la ecuación, esta se compone, entre otras, de una submatriz R, correspondiente a una matriz de rotación, y una submatriz p, que corresponde a una translación.

En robótica, la submatriz f se suele considerar nula, y la w como la unidad, ya que no son útiles en este campo.

Una característica muy útil de estas matrices es que se pueden componer transformaciones muy complejas con varias trasformaciones simples (giros y traslaciones básicas) que se multiplican.

4.2.5 Cinemática directa. Algoritmo de Denavit-Hartenberg

El problema cinemático directo intenta determinar la posición y orientación del extremo del robot en función de los valores de sus coordenadas articulares. Es decir, siendo q_N la variable que representa el valor de la enésima articulación y $[x,\ y,\ z,\ \alpha,\ \beta,\ \gamma\ ]$ las variables usadas para representar posición y orientación, se intentan calcular las siguientes relaciones:

$$x = f_x(q_0, \dots, q_N)$$

$$y = f_y(q_0, \dots, q_N)$$

$$z = f_z(q_0, \dots, q_N)$$

$$\alpha = f_\alpha(q_0, \dots, q_N)$$

$$\beta = f_\beta(q_0, \dots, q_N)$$

$$\gamma = f_\gamma(q_0, \dots, q_N)$$

La resolución de este problema usando matrices de trasformación homogénea no suele ser complicada, basta con calcular las matrices de trasformación homogénea que relacionan cada par de eslabones y multiplicarlas posteriormente en orden, de manera que si ${}^{i-1}A_i$ describe la posición y orientación del eslabón i-ésimo respecto al anterior eslabón, se puede definir T, la matriz resultante de todas las trasformaciones, como:

$$T = {}^0A_1\,{}^1A_2 \cdots {}^{N-1}A_N$$

Aunque para describir la relación existente entre dos eslabones se puede usar cualquier sistema de referencia acoplado a cada elemento, como norma general en robótica se suele usar la representación de Denavit-Hartenberg (D-H).

Denavit y Hartenberg propusieron, en 1955, una manera sistemática de establecer sistemas de coordenadas, denominados como S_i, ligados a cada eslabón y de relacionar estos sistemas de referencia entre sí mediante cuatro transformaciones básicas. Estas trasformaciones son:

1. Rotación alrededor del eje z_{i-1} un ángulo θ_i.
2. Traslación a lo largo de z_{i-1} una distancia d_i.
3. Traslación a lo largo de x_i una distancia a_i.
4. Rotación alrededor del eje x_i un ángulo α_i.

De este modo se tiene que:

$$^{i-1}A_i = T(z,\theta_i)\,T(0,0,d_i)T(a_i,0,0)T(x,\alpha_i)$$

$$^{i-1}A_i = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde $C\theta_i$ y $S\theta_i$ representan $\cos(\theta_i)$ y $\text{sen}(\theta_i)$ respectivamente. Y θ_i, α_i, a_i y d_i son los parámetros D-H del eslabón.

Para que esta matriz relacione dos eslabones consecutivos con sus respectivos sistemas de coordenadas es necesario que estos hayan sido escogidos según unas determinadas normas que permitan luego obtener los parámetros de forma correcta, se presenta así el **algoritmo para la resolución del problema cinemático directo de Denavit-Hartenberg**:

1. Numerar los eslabones comenzando por 1 (primer eslabón móvil de la cadena) y acabando con n (ultimo eslabón móvil). Se numerará como eslabón 0 a la base fija del robot.

2. Numerar cada articulación comenzando por 1(la correspondiente al primer grado de libertad) y acabando por n.

3. Localizar el eje de cada articulación. Si esta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

4. Para i de 0 a n-1, situar el eje z_i sobre el eje de la articulación i+1.

5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier punto del eje z_0. Los ejes x_0 e y_0 se sitúan de forma dextrógira.

6. Para i de 1 a n-1, se sitúa el sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje z_i con la línea normal común a z_i y z_{i-1}. Si ambos ejes se cortasen se situaría $\{S_i\}$ en el punto de corte. Si ambos ejes fuesen paralelos $\{S_i\}$ se situaría en la articulación i+1.

7. Situar x_i en la línea normal común a z_{i-1} y z_i.

8. Situar y_i de modo que forme un sistema dextrógiro con x_i y z_i.

9. Situar el sistema $\{S_n\}$ en el externo del robot de modo que z_0 coincida con la dirección de z_{n-1} y x_n sea normal a z_{n-1} y z_n.

10. Obtener θ_i como el ángulo que hay que girar en torno a z_{i-1} para que x_{i-1} y x_i queden paralelos.

11. Obtener d_i como la distancia, medida a lo largo de z_{i-1}, que habría que desplazar $\{S_{i-1}\}$ para que x_i y x_{i-1} quedasen alineados.

12. Obtener a_i como la distancia medida a lo largo de x_i (que ahora coincidiría con x_{i-1}) que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

13. Obtener α_i como el ángulo que habría de girar en torno a x_i (que ahora coincidiría con x_{i-1}) para que el nuevo $\{S_{i-1}\}$ coincida totalmente con $\{S_i\}$.

14. Obtener las matrices de transformación ${}^{i-1}A_i$.

15. Obtener la matriz de transformación que relaciona la base con el extremo del robot T.

Es importante añadir que existen variantes a este algoritmo en las que el orden de las traslaciones y rotaciones difiere, y por tanto la ecuación de la matriz de trasformación ${}^{i-1}A_i$ cambia.

4.2.6 Cinemática inversa

El objetivo del problema cinemático inverso es encontrar los valores que deben tomar las coordenadas articulares del robot para que su extremo se coloque en una posición y orientación determinada. Es decir, se pretende encontrar funciones tal que, para i=1…n:

$$q_i = f_i(x, y, z, \alpha, \beta, \gamma)$$

Mientras que en el problema cinemático directo se puede hallar una solución de manera sistemática siguiendo una serie de pasos, no ocurre lo mismo en el problema cinemático inverso, existiendo varios métodos cuya efectividad varía en función de la configuración del robot.

Una de las posibilidades existentes es **la resolución del problema cinemático inverso por métodos geométricos.** En este método basa en encontrar tantas relaciones geométricas independientes como grados de libertad tenga el robot, usando las coordenadas del extremo del robot, sus coordenadas articulares y sus dimensiones.

El problema de este método aparece al aumentar la complejidad de la configuración del robot, por ejemplo, al aumentar el número de grados de libertad. Al aumentarla dimensionalidad del espacio articular del robot, la complejidad de los cálculos para encontrar relaciones geométricas aumenta hasta hacer prácticamente imposible su resolución. Por lo tanto, este método solo es válido para robots de tres o menos grados de libertad.

Otro método es la **resolución del problema cinemático inverso a partir de la matriz de trasformación homogénea.** De esta manera, se pretende encontrar la solución al modelo cinemático inverso a partir del conocimiento de su cinemática directa, es decir, del conocimiento de la matriz de trasformación homogénea que relaciona el extremo o herramienta del robot con su base. Se parte entonces de la siguiente igualdad:

$$\begin{bmatrix} f_{11}(q_1 \dots q_n) & f_{12}(q_1 \dots q_n) & f_{13}(q_1 \dots q_n) & f_{14}(q_1 \dots q_n) \\ f_{21}(q_1 \dots q_n) & f_{22}(q_1 \dots q_n) & f_{23}(q_1 \dots q_n) & f_{24}(q_1 \dots q_n) \\ f_{31}(q_1 \dots q_n) & f_{32}(q_1 \dots q_n) & f_{33}(q_1 \dots q_n) & f_{34}(q_1 \dots q_n) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde en el lado izquierdo se tiene la matriz de transformación homogénea obtenida en la resolución del problema cinemático directo. En el lado derecho se encuentra la matriz con las variables para posición y rotación.

Se tiene entonces un sistema de doce ecuaciones que se debe resolver para encontrar seis relaciones entre posición y orientación del efector final y las variables articulares. Como se puede ver, existen más ecuaciones que incógnitas, esto es debido a que las ecuaciones de la matriz de rotación no son independientes entre sí, al haber nueve ecuaciones para tres grados de libertad para la rotación. Es necesario, por tanto, realizar una cuidadosa selección de las ecuaciones que se van a intentar resolver.

Al igual que en el método geométrico, la complejidad de este método escala con la dimensionalidad del espacio articular del robot, y dependiendo de la configuración del robot, puede no existir una solución cerrada. A pesar de ello, este tipo de solución es el más indicado, ya que permite resolver el problema cinemático inverso en tiempo real, además de dar resultados exactos y permitir escoger la solución más conveniente de entre las que existen.

Los dos métodos mencionados anteriormente pretenden obtener una solución explicita al problema cinemático inverso, y esto no siempre es sencillo o incluso posible. Existe también la posibilidad de realizar la **resolución de problema cinemático inverso por métodos numéricos**, que obtienen la solución de forma iterativa.

Estos métodos hacen uso de la **matriz Jacobiana** del robot y su inversa. La matriz Jacobiana de un robot permite relacionar las derivadas de las variables de posición y orientación del extremo del robot con las derivadas de sus variables articulares. Es decir, relaciona las velocidades de las articulaciones con la velocidad (de desplazamiento y giro) del efector final.

La relación que permite obtener las velocidades del extremo del robot a partir de las velocidades articulares es la siguiente:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = J * \begin{bmatrix} \dot{q_0} \\ \vdots \\ \dot{q_n} \end{bmatrix}, donde\ J = \begin{bmatrix} \frac{df_x}{dq_1} & \cdots & \frac{df_x}{dq_n} \\ \vdots & \ddots & \vdots \\ \frac{df_\gamma}{dq_1} & \cdots & \frac{df_\gamma}{dq_n} \end{bmatrix}$$

Existen varios métodos numéricos distintos, el que se usará en este proyecto intenta linealizar la función $r = f(q)$, para ello, se hace una aproximación a la Serie de Taylor de orden uno. De esta manera es posible obtener:

$$r = f(q) = f(q_0) + f'(q_0)(q - q_0), \qquad donde\ f'(q_0) = J(q_0)$$

$$r - f(q_0) = J(q - q_0) \rightarrow \Delta r = J * \Delta q \rightarrow \Delta q = J^{-1} * \Delta r$$

$$q^{i+1} = q^i + \Delta q$$

4.2.7 Trayectorias e interpoladores

A la hora de realizar una determinada tarea, el robot debe moverse de un punto a otro, este movimiento puede ser realizado de diversas maneras, atendiendo a como se desea que se muevan o el extremo del robot o los ejes de este. A continuación, se muestran los diferentes tipos de trayectorias más usuales.

La primera de las mostradas en la figura 26 es la trayectoria eje a eje, en ella, se mueve un eje cada vez, un eje no empezará su movimiento hasta que el anterior lo haya terminado. En la segunda, en cambio, todos los ejes se mueven a la vez, alcanzando sus posiciones finales de manera escalonada. En la tercera, se controla la velocidad de cada eje para que todos lleguen a su destino a la vez. Finalmente, la cuarta es la única que se centra más en el movimiento del extremo del robot que en el de los ejes, se pretende que este tenga un movimiento rectilíneo en el espacio cartesiano. Como puede verse, esto provoca unos movimientos más complejos por parte de cada eje.

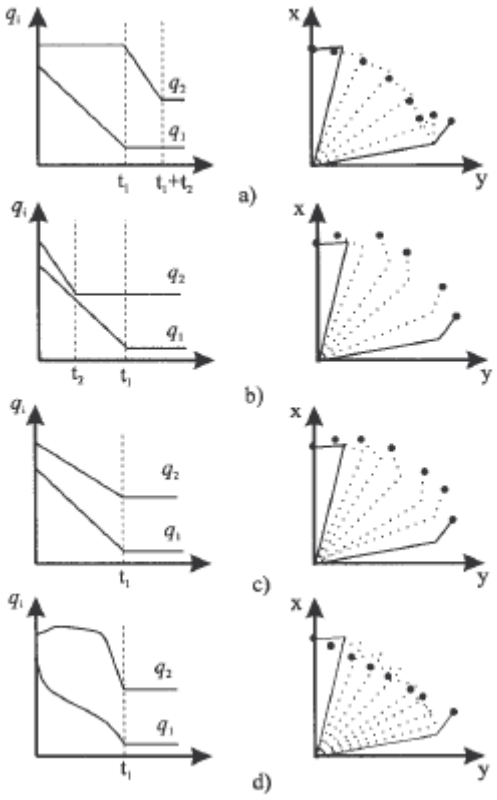


Figura 26[2]: Diferentes tipos de trayectorias en un brazo robótico.

4.3 SCORBOT-ER III

Como ya se ha mencionado, para este proyecto se usará un brazo robótico educativo llamado SCORBOT-ER III, fabricado por Eshed Robotec (1982). Este brazo robótico dispone de cinco grados de libertad, con un motor de corriente continua de 12Vdc para cada uno. Además, para su posicionamiento incorpora encoders ópticos, junto con de finales de carrera.

A modo de herramienta, dispone de una pinza, controlada mediante otro motor de 12Vdc. Este modelo inicialmente disponía también de software propio para su control, aunque no se encuentra disponible ya. En el trabajo de fin de grado anterior se programó un software básico usando MATLAB y la plataforma Arduino para su puesta en marcha.

Una limitación de este software es que solo permitía que el brazo robótico realizase trayectorias eje a eje, es decir, una articulación no iniciaba su movimiento hasta que la anterior había terminado. Este hecho hace que sea muy difícil controlar en cada momento la posición de la pinza, aumentando la probabilidad de colisión entre esta y otros objetos de la zona de trabajo.

Por esta y otras, se decide aumentar la funcionalidad del software de control del robot y adaptarlo al proyecto actual.

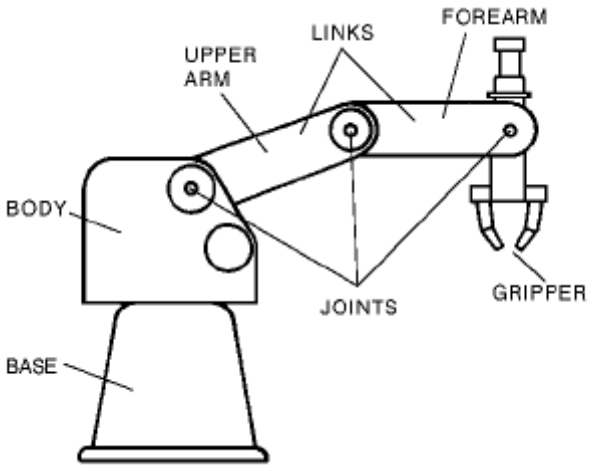


Figura 27[3]: Esquema del SCORBOT-ER III

4.3.1 Características técnicas del SCORBOT-ER III

La siguiente tabla ofrece las especificaciones técnicas del brazo SCORBOT-ER III,

Elemento	**Especificaciones**
Estructura mecánica	5 ejes + pinza
Espacio de trabajo:	
-Eje 1: Rotación de la base	310º
-Eje 2: Rotación del hombro	+130º/-35º
-Eje 3: Rotación del codo	±130º
-Eje 4: Rotación de la muñeca	±130º
-Eje 5: Giro de la muñeca	Ilimitado
Radio máximo de trabajo	610 mm
Transmisiones	Engranajes, correas dentadas y tornillos sin fin

Actuadores	6 motores DC
Realimentación	Sensores Ópticos
Hard Home	Posición de referencia fija en todos los ejes.
Velocidad máxima	330 mm/s
Peso	11 kg

En cuanto a los motores, como ya se ha comentado, los cinco ejes y la apertura y cierre de la pinza están controlados por servomotores de corriente continua cuya dirección es determinada por la polaridad del voltaje aplicado a ellos.

Los encoders ópticos están montados en cada motor, de manera que se puede determinar la posición y velocidad de cada uno individualmente. El número y la frecuencia de los pulsos emitidos por estos son medidos por el controlador.

Los cinco *microswitches* incluidos son usados para situar al robot en una posición llamada *hard home*. Esta posición se usa como punto de referencia desde el que iniciar las operaciones del robot, ya que los encoders al ser incrementales solo ofrecen datos de posición relativa.

A modo de controlador se utiliza un Arduino Mega 2560 junto con el lenguaje de programación Matlab, que ofrece la posibilidad de controlar el Arduino en tiempo real gracias a su paquete de soporte *ArduinoIO*.

Arduino es una plataforma de hardware (y software) libre inicialmente diseñado para prototipado rápido. Su sencillez de uso, flexibilidad y bajo coste han hecho de él un producto mundialmente conocido y utilizado en aplicaciones desde IoT a impresoras 3D.

En este caso, el Arduino es usado para efectuar un control directo sobre motores y para recibir los datos de encoders, que luego serán enviados a Matlab cuando este lo requiera.

También cabe mencionar una característica del SCORBOT-ER III que lo diferencia de otros brazos robóticos: al contrario que otros, en el SCORBOT-ER III, el movimiento de uno de sus ejes provoca movimiento en otros ejes.

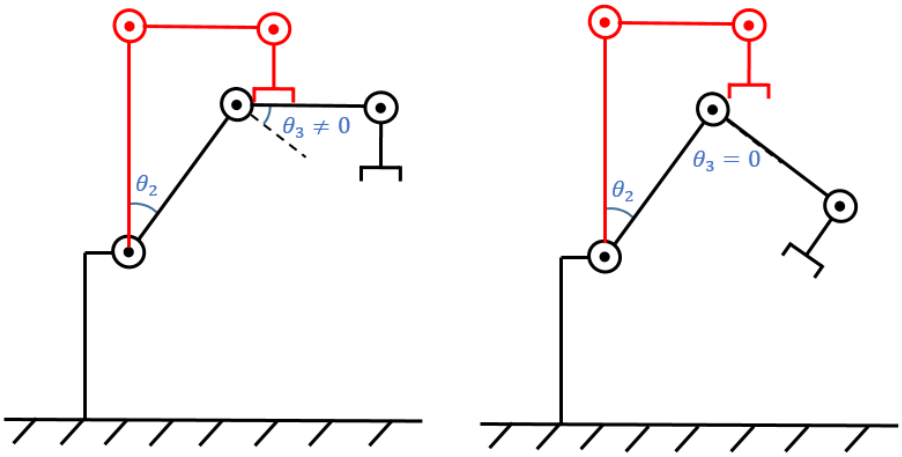


Figura 29: Diferencias en el movimiento de un brazo robótico genérico (derecha) y el SCORBOT-ER III (izquierda).

Como se puede observar en la figura 29, en el brazo robótico genérico, a la derecha, al aumentar θ_2 y θ_3 permanecer constante, el tercer eslabón sufre una rotación y una translación alrededor del eje de la segunda articulación. En cambio, en el SCORBOT-ER III, este eslabón no sufre rotación alguna, permaneciendo en este caso en una orientación horizontal al suelo. Es por ello que θ_3 varía al variar θ_2. Esto también ocurre con θ_4 al variar θ_2 o θ_3.

Es necesario por lo tanto encontrar una relación que permita calcular el ángulo real que deben girar los motores de las articulaciones tres y cuatro para alcanzar la posición angular deseada.

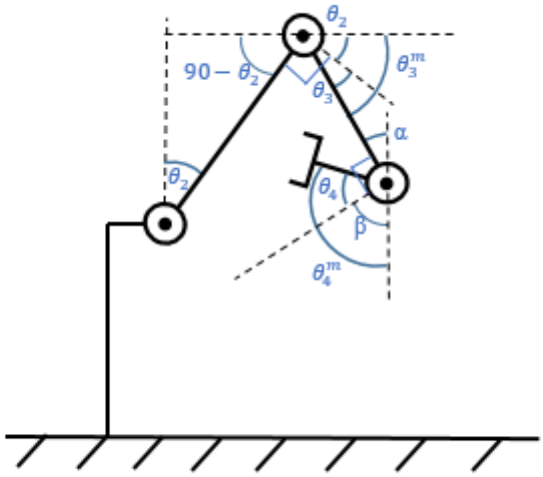


Figura 30: Esquema para el cálculo de los ángulos reales de giro del SCORBOT-ER III

Como se puede observar en la figura 30, esto es posible encontrando relaciones geométricas entre los distintos ángulos importantes para determinar el movimiento del brazo robótico. Llamando θ_i^m

a los ángulos que realmente deben girar las articulaciones del robot, y θ_i a los ángulos teóricos calculados, se pueden establecer relaciones geométricas en cadenas, obteniendo que:

$$\theta_3^m = \theta_2 + \theta_3$$

$$\alpha = 90 - \theta_3^m \rightarrow \beta = 180 - \alpha - 90 = \theta_3^m$$

$$\theta_4^m = \theta_4 + \theta_3^m = \theta_2 + \theta_3 + \theta_4$$

4.4 Resolución del problema cinemático directo

Se resolverá el problema cinemático directo siguiendo el algoritmo de Denavit-Hartenberg, mencionado anteriormente en esta memoria. Dado que el ultimo grado de libertad, de rotación de la muñeca, es necesario solo para orientar la muñeca de tal forma que pueda cerrarse alrededor del objeto sobre la cinta, se va a excluir este tanto del cálculo de la cinemática directa como de la inversa, quedando entonces el sistema con cuatro grados de libertad.

Se utilizará una variante del algoritmo de Denavit-Hartenberg. Esta sigue los mismos pasos que la mencionada anteriormente, excepto que cambia la forma de calcular los parámetros. En este caso:

- a_i: Distancia desde z_i hasta z_{i+1} a lo largo de x_i.
- α_i: Angulo entre z_i y z_{i+1} en torno a x_i.
- d_i: Distancia desde x_{i-1} hasta x_i a lo largo de z_i.
- θ_i: Angulo entre x_{i-1} y x_i en torno a x_{i-1} hasta z_i.

La matriz de transformación homogénea queda entonces de la siguiente manera:

$${}^{i-1}A_i = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & a_{i-1} \\ S\theta_i C\alpha_{i-1} & C\theta_i C\alpha_{i-1} & -S\alpha_{i-1} & -S\alpha_{i-1} d_i \\ S\theta_i S\alpha_{i-1} & C\theta_i S\alpha_{i-1} & C\alpha_{i-1} & C\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Después de numerar los eslabones, situar los sistemas de coordenadas y sus respectivos ejes obtenemos lo siguiente:

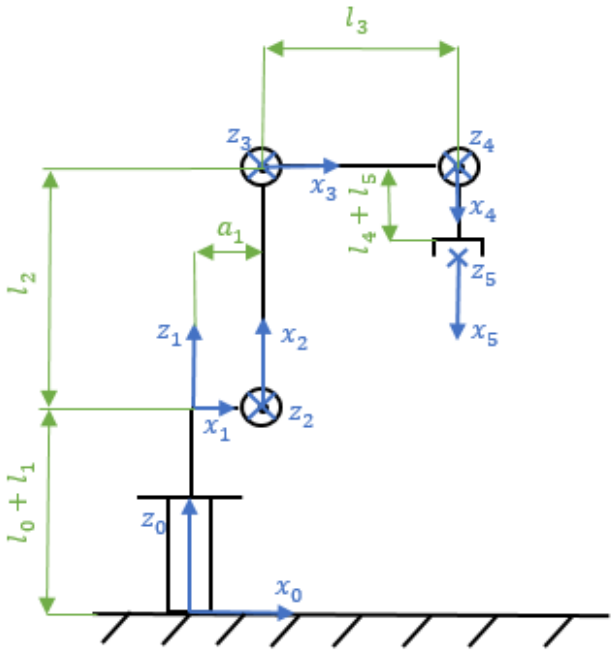


Figura 31: Asignación de ejes de referencia según las disposiciones del algoritmo Denavit-Hartenberg

A continuación, entonces, obtenemos los parámetros del algoritmo:

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	$l_0 + l_1$	θ_1
2	$-\pi/2$	a_1	0	$\theta_2 - \pi/2$
3	0	l_2	0	$\theta_3 + \pi/2$
4	0	l_3	0	$\theta_4 + \pi/2$
5	0	$l_4 + l_5$	0	0

Figura 32: Tabla con parámetros Denavit-Hartenberg

Una vez obtenida esta tabla de parámetros, la obtención de la matriz de trasformación homogénea, solución al problema de la cinemática directa es muy sencillo, basta con sustituir los valores de cada fila de la tabla en su correspondiente matriz, obteniendo entonces cinco matrices, y después multiplicarlas en el orden correcto:

$$ {}^{0}A_1 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & 0 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & (l_0 + l_1) \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}A_2 = \begin{bmatrix} \mathrm{C}(\theta_2 - \pi/2) & -S(\theta_2 - \pi/2) & 0 & a_1 \\ 0 & 0 & 1 & 0 \\ -S(\theta_2 - \pi/2) & -C(\theta_2 - \pi/2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}A_3 = \begin{bmatrix} \mathrm{C}(\theta_3 + \pi/2) & -S(\theta_3 + \pi/2) & 0 & l_2 \\ S(\theta_3 + \pi/2) & C(\theta_3 + \pi/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}A_4 = \begin{bmatrix} \mathrm{C}(\theta_4 + \pi/2) & -S(\theta_4 + \pi/2) & 0 & l_3 \\ S(\theta_4 + \pi/2) & C(\theta_4 + \pi/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}A_5 = \begin{bmatrix} 1 & 0 & 0 & l_4 + l_5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Una vez obtenidas estas matrices, basta multiplicarlas en el orden correcto para obtener la matriz de trasformación homogénea que relaciona la base del brazo robótico con su efector final. Es decir, obtenemos:

$$ T = {}^{0}A_1 {}^{1}A_2 {}^{2}A_3 {}^{3}A_4 {}^{4}A_5 $$

$$ T = \begin{bmatrix} -c_1 s_{234} & -c_1 c_{234} & -s_1 & c_1(a_1 + l_3 c_{23} + l_2 s_2 - (l_4 + l_5) s_{234}) \\ -s_1 s_{234} & -s_1 c_{234} & c_1 & s_1(a_1 + l_3 c_{23} + l_2 s_2 - (l_4 + l_5) s_{234}) \\ -c_{234} & s_{234} & 0 & l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Donde $s = \mathrm{sen}(i)$, $s_{ij} = \mathrm{sen}(i + j)$, $s_{ijk} = \mathrm{sen}(i + j + k)$, $c_i = \cos(i)$, $c_{ij} = \cos(i + j)$, $c_{ijk} = \cos(i + j + k)$.

Con esto se resuelve el problema cinemático directo, ahora basta con saber las coordenadas articulares del robot para saber su posición y orientación exacta.

4.5 Resolución del problema cinemático inverso

Una vez obtenida la función que permite calcular los parámetros de posición y orientación del robot en función de los parámetros de sus coordenadas articulares, se procede a obtener la función inversa, es decir, aquella que, usando como entrada la posición y orientación, permite obtener las coordenadas articulares.

Para ello, se usarán dos de los métodos mencionados anteriormente:

- Resolución del problema cinemático inverso a partir de la matriz de trasformación homogénea.
- Resolución de problema cinemático inverso por métodos numéricos.

No se usarán métodos geométricos para resolver este problema ya que, a causa de la dimensionalidad de este, la dificultad escala por encima de los otros dos métodos.

4.5.1 Resolución del problema cinemático inverso a partir de la matriz de trasformación homogénea.

Como ya se ha visto, en este método se procederá a hallar la solución usando la matriz de trasformación homogénea que relaciona la base del robot con el extremo, es decir:

$$T = {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5$$

$$T = \begin{bmatrix} -c_1 s_{234} & -c_1 c_{234} & -s_1 & c_1(a_1 + l_3 c_{23} + l_2 s_2 - (l_4 + l_5) s_{234}) \\ -s_1 s_{234} & -s_1 c_{234} & c_1 & s_1(a_1 + l_3 c_{23} + l_2 s_2 - (l_4 + l_5) s_{234}) \\ -c_{234} & s_{234} & 0 & l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De esta segunda igualdad se puede obtener θ_1 fácilmente, dividiendo el elemento (2,4) entre el (1,4), de manera que se obtiene:

$$\theta_1 = \arctan(p_y / p_x)$$

Antes de intentar hallar las funciones que devuelvan la segunda coordenada articular y la tercera, y para facilitar su obtención, se ha decidido realizar una simplificación del problema.

El brazo robótico va a ser utilizado en una aplicación tipo *Pick and Place* en la que basta con que la herramienta de este se sitúe en una orientación perpendicular con respecto al suelo y en una posición superior con respecto al objeto. Por lo tanto, es posible calcular el valor que debe tener la cuarta coordenada articular con respecto a la segunda y tercera, ya que debe ser tal que oriente el extremo del brazo robótico como se muestra en la figura 33.

Por geometría, es posible determinar entonces el valor que debe tener θ_4.

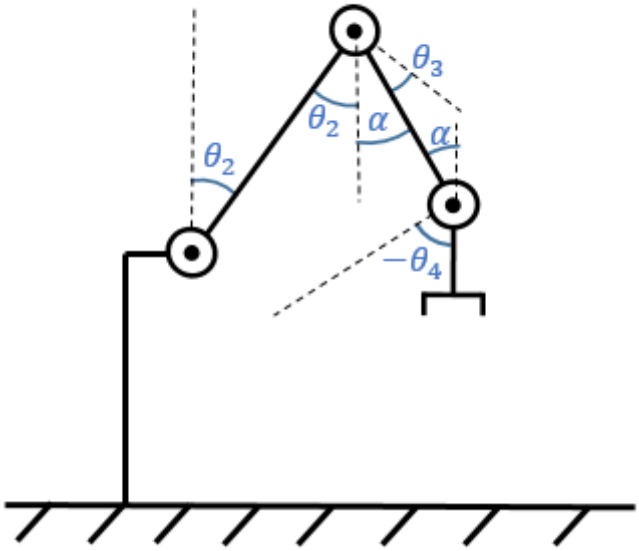


Figura 33: Esquema para el cálculo de θ_4

Definimos un ángulo $\alpha = 90 - \theta_2 - \theta_3$, por lo tanto, $-\theta_4 = 180 - 90 - \alpha \rightarrow \theta_4 = -\theta_2 - \theta_3$

Este cálculo es fácilmente comprobable, ya que, si se sustituye en la matriz de trasformación homogénea obtenida al resolver la cinemática directa θ_4 por $= -\theta_2 - \theta_3$ se obtendrá la siguiente matriz:

$$T = \begin{bmatrix} 0 & -c_1 & -s_1 & c_1(a_1 + l_3c_{23} + l_2s_2) \\ 0 & -s_1 & c_1 & s_1(a_1 + l_3c_{23} + l_2s_2) \\ -1 & 0 & 0 & l_0 + l_1 - l_3s_{23} + l_2c_2 - (l_4 + l_5) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Si se observan las tres primeras filas de la primera columna, que definen la orientación del eje *x* de la herramienta con respecto a la base del robot, se puede ver que:

$x_{herr} = 0 * x_{base} + 0 * y_{base} + -1 * z_{base} = -z_{base}$, es decir, la dirección del eje *x* de la herramienta es la contraria al eje *z* de la base, lo cual concuerda con lo deseado al realizar esta simplificación.

Una vez realizado este paso, se procede a obtener la segunda coordenada articular y la tercera, para lo cual será necesario reordenar la igualdad $T = {}^0A_1\,{}^1A_2\,{}^2A_3\,{}^3A_4\,{}^4A_5$. Realizar esto suele facilitar la tarea de resolver el sistema de ecuaciones. En este caso, es posible obtener θ_3 de la siguiente manera:

$$({}^1A_2^{-1})({}^0A_1^{-1}) \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} ({}^4A_5^{-1}) = {}^2A_3\,{}^3A_4$$

Si al lateral izquierdo de la igualdad se le denomina como X y al derecho como Y:

$$X = \begin{bmatrix} -c_2 & -c_1 s_2 & s_1 s_2 & (p_z + l_4 + l_5 - l_0 - l_1)c_2 + (p_x c_1 + p_y s_1 - a_1)s_2 \\ s_2 & -c_1 c_2 & s_1 c_2 & (p_x c_1 + p_y s_1 - a_1)c_2 - (p_z + l_4 + l_5 - l_0 - l_1)s_2 \\ 0 & s_1 & c_1 & p_y c_1 - p_x s_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} -c_2 & -s_2 & 0 & l_2 - l_3 s_3 \\ s_2 & -c_2 & 0 & l_3 c_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A continuación, se resuelve la siguiente ecuación:

$$X_{14}^2 + X_{24}^2 = Y_{14}^2 + Y_{24}^2$$

Para ello, se usan variables auxiliares $a = (p_z + l_4 + l_5 - l_0 - l_1)$ y $b = (p_x c_1 + p_y s_1 - a_1)$. De tal manera:

$$(ac_2 + bs_2)^2 + (bc_2 - as_2)^2 = (l_2 - l_3 s_3)^2 + (l_3 c_3)^2$$

$$a^2 + b^2 = l_2^2 + l_3^2 - 2l_2 l_3 s_3 \rightarrow s_3 = \frac{l_2^2 + l_3^2 - a^2 - b^2}{2l_2 l_3}$$

Con lo que se obtiene θ_3 como:

$$\theta_3 = \arcsin\left(\frac{l_2^2 + l_3^2 - a^2 - b^2}{2l_2 l_3}\right)$$

Una vez obtenidas las ecuaciones que describen la primera y tercera variable articular en función de la posición y orientación del efector final del robot, el cálculo de la restante es muy sencillo, ya que solo hay que sustituir los valores conocidos en una ecuación, quedando como única incógnita la variable articular restante.

En este caso, la ecuación se obtiene de la tercera fila en la cuarta columna

$$T = \begin{bmatrix} 0 & -c_1 & -s_1 & c_1(a_1 + l_3 c_{23} + l_2 s_2) \\ 0 & -s_1 & c_1 & s_1(a_1 + l_3 c_{23} + l_2 s_2) \\ -1 & 0 & 0 & l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$c_1(a_1 + l_3 c_{23} + l_2 s_2) = p_x$$

$$\frac{p_x}{c_1} - a_1 = l_3(c_2 c_3 - s_2 s_3) + l_2 s_2$$

$$s_2 = \frac{p_x/c_1 - a_1 - l_3 c_2 c_3}{(l_2 - l_3 s_3)}$$

Como se puede observar, esta ecuación depende de c_2. Por lo que se debe sustituir esta ecuacion en otra:

$$l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) = p_z$$

$$l_0 + l_1 - l_3 s_2 c_3 - l_3 s_3 c_2 + l_2 c_2 - (l_4 + l_5) = p_z$$

Usando las siguientes variables auxiliares obtenemos:

$$j = l_0 + l_1 - l_4 - l_5 - p_z, \qquad k = l_2 - l_3 s_3$$

$$c_2 = \frac{kj - \frac{c_3 l_3 p_x}{c_1} + c_3 l_3 a_1}{-{l_3}^2 {c_3}^2 - k^2}$$

Con lo que se obtiene θ_2 como:

$$\theta_2 = arcos\left(\frac{kj - \frac{c_3 l_3 p_x}{c_1} + c_3 l_3 a_1}{-{l_3}^2 {c_3}^2 - k^2}\right)$$

Solo queda un último paso. Para prevenir posibles errores y acotar las coordenadas angulares en el rango [-π, π], se suele usar la función *atan2*. Esta es una función presente en la mayoría de lenguajes de programación, y especialmente en aquellos usados para robótica. Esta función ha sido creada para cubrir ciertas carencias de la función arcotangente *atan*.

Por ejemplo, el ángulo del vector [1, 1] al eje *x* se calcula como $\text{atan}\left(\frac{1}{1}\right) = \pi/4$. Pero el ángulo entre el vector [-1, -1] y el eje x, al calcularlo mediante esta función da exactamente el mismo resultado, cuando este debería ser $-\pi/4$.

En robótica ello daría como resultado en error en el posicionamiento de las articulaciones del robot, por lo que se usa la función *atan2*:

$$atan2(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & si\ x > 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & si\ x < 0\ e\ y \geq 0, \\ \arctan\left(\frac{y}{x}\right) - \pi & si\ x < 0\ e\ y < 0, \\ +\frac{\pi}{2} & si\ x = 0\ e\ y > 0, \\ -\frac{\pi}{2} & si\ x = 0\ e\ y < 0, \\ indefinido & si\ x = 0\ e\ y = 0, \end{cases}$$

Se cambian por tanto las ecuaciones que devuelven las coordenadas articulares para que usen esta función, obteniendo:

$$\theta_1 = \text{atan2}(p_y, p_x)$$

$$s_2 = \sqrt{1 - c_2^2}, \qquad c_2 = \frac{kj - \frac{c_3 l_3 p_x}{c_1} + c_3 l_3 a_1}{-{l_3}^2 {c_3}^2 - k^2}, \qquad \rightarrow \theta_2 = \text{atan2}(s_2, c_2)$$

$$s_3 = \frac{l_2^2 + l_3^2 - a^2 - b^2}{2 l_2 l_3}, \qquad c_3 = \sqrt{1 - s_3^2}, \qquad \rightarrow \theta_3 = \text{atan2}(s_3, c_3)$$

$$\theta_4 = -\theta_2 - \theta_3$$

4.5.2 Resolución del problema cinemático inverso por métodos numéricos

El cálculo de la matriz Jacobiana es un paso imprescindible para la resolución de este problema mediante métodos numéricos. En particular, en este caso se intentará relacionar las variaciones en las coordenadas angulares con las variaciones en la posición del extremo del robot. Relacionar las primeras con la orientación no es necesario, dado que el brazo robótico solo tiene tres coordenadas angulares con las que posicionarse, ya que la cuarta es dependiente de la segunda y tercera. Al solo disponer de tres grados de libertad eficaces, es posible controlar la posición o la orientación, pero no ambas.

Dado que se pretende controlar la posición del extremo del robot, se debe encontrar una manera de relacionar esta con las coordenadas angulares del SCORBOT-ER III. Para ello, se parte de la Matriz de Transformación Homogénea calculada anteriormente y se extrae de ahí el vector de la posición.

$$T = \begin{bmatrix} 0 & -c_1 & -s_1 & c_1(a_1 + l_3 c_{23} + l_2 s_2) \\ 0 & -s_1 & c_1 & s_1(a_1 + l_3 c_{23} + l_2 s_2) \\ -1 & 0 & 0 & l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow$$

$$r = \begin{bmatrix} c_1(a_1 + l_3 c_{23} + l_2 s_2) \\ s_1(a_1 + l_3 c_{23} + l_2 s_2) \\ l_0 + l_1 - l_3 s_{23} + l_2 c_2 - (l_4 + l_5) \end{bmatrix}$$

Una vez conocido el vector r, se usa para calcular la matriz Jacobiana de dicho vector, usando el método antes mencionado:

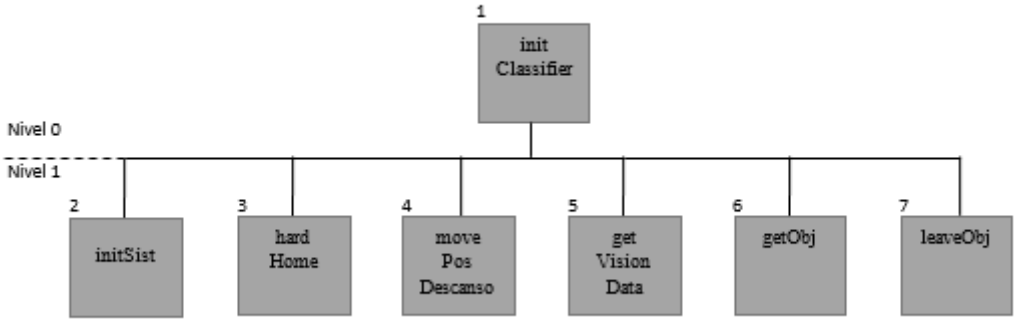
$$J = \begin{bmatrix} \frac{df_x}{dq_1} & \cdots & \frac{df_x}{dq_n} \\ \vdots & \ddots & \vdots \\ \frac{df_\gamma}{dq_1} & \cdots & \frac{df_\gamma}{dq_n} \end{bmatrix}$$

Una vez obtenido esto, solo es necesario un bucle condicionado al número de iteraciones que se desee, el error máximo permitido, etc. Eligiendo un valor iniciar para las tres coordenadas angulares:

$$q_{i+1} = q_i + J(q_i)^{-1}(r_{obj} - r(q_i))$$

4.6 Programación en Matlab

Como ya se ha mencionado, para la programación de este proyecto se ha utilizado el lenguaje MATLAB, usando este, se han creado varias funciones que permiten el control del brazo robótico. A continuación, se muestra un esquema de diseño *top-down* de código creado para el brazo robótico. En este tipo de esquemas, se parte de un nivel inicial en el cual se encuentra la función de más alto nivel en el programa, es decir, la síntesis de la solución al problema. A medida que se añaden niveles, esta función se descompone en sus funciones más básicas, hasta que en el último nivel aparecen solo las funciones de menor nivel creadas.

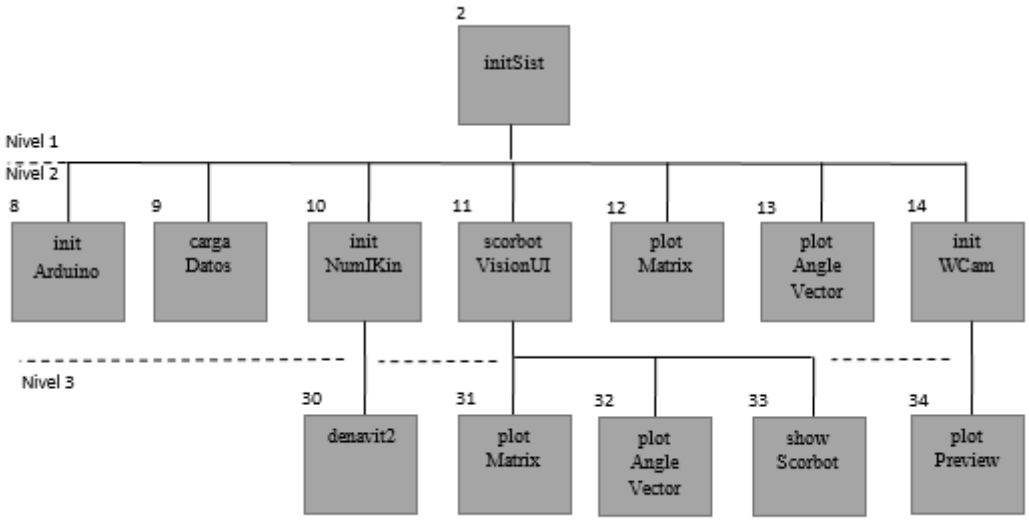


Para iniciar el sistema robot-visión y empezar el proceso de clasificación, se usa la función `initClassifier`. Esta función, como se puede observar, llama a su vez a `initSist`, que se encarga de iniciar la comunicación con el Arduino, cargar datos, iniciar la interfaz gráfica, etc.

Una vez hecho esto, se crea una ventana de aviso recordando a usuario que debe asegurarse de que la fuente de alimentación del robot esté activada. Después sitúa al robot en `hardHome`. Esto se hace para que el robot parta desde una posición conocida, supuesta una desconexión no programada que llevaría a detener el robot sin conocer su posición exacta. Una vez situado el

brazo, se lleva a la posición de descanso, esta es una posición intermedia desde la cual el robot procederá a coger objetos, dejarlos, etc.

Una vez llevado a cabo todo el proceso para iniciar los sistemas, empieza el proceso de clasificación. Para ello, se entra en un bucle *while* cuya condición de salida es que el usuario pulse el botón de detener. El robot entonces se mantiene en la posición de descanso, llamando a la función `getVisionData` cada cierto tiempo. Esta función devuelve la posición, orientación y color del objeto situado en la cinta transportadora mas cercano al final de esta. Una vez se comprueba que el objeto esta en la zona de recogida, se llama a la función `getObj`, pasando como parámetro la orientación de este. Esta función recoge el objeto, volviendo después a la posición de descanso. Se procede entonces a llamar a la función `leaveObj`, usando como parámetro de entrada el color del objeto. Esta función se encarga de depositar el objeto en la zona indicada por el usuario y volver después a la posición de descanso. Una vez acabado el ciclo se vuelve entonces al comienzo del bucle *while*, a no ser que haya sido indicado lo contrario por parte del usuario.



La función `initSist` mencionada anteriormente se compone a su vez de las funciones mostradas en el esquema anterior. `initSist` primero llama a `initArduino`. Esta función crea un objeto para manejar el Arduino Mega, asignándole pines para controlar las salidas PWM y salidas digitales que se encargan de controlar la dirección de los motores. También admite establecer un modo de simulación, con la finalidad de poder probar trayectorias y movimientos del brazo sin mover el robot físico, evitando con ello el riesgo de fallos o posiciones en las cuales el robot pueda resultar dañado. Finalmente, se imprime por pantalla que el Arduino ha sido satisfactoriamente conectado.

A continuación, `cargaDatos` se encarga de crear y actualizar las variables globales que almacenarán los valores de longitud de las articulaciones del robot, la posición angular y cartesiana del robot, las relaciones de transmisión de este, así como las posiciones por defecto de descanso, de recogida del objeto, y aquellas donde se dejará este.

Con la llamada a la función `initNumIKin`, se llevan a cabo los cálculos para obtener la posición y la matriz Jacobiana en función de los valores angulares. Ello se lleva a cabo calculando la matriz de trasformación homogénea que relaciona la base del robot con su extremo gracias a la función `denavit2`, obteniendo a partir de ella el vector de posición, y posteriormente, derivando este, la matriz Jacobiana. Tanto el vector de posición como la matriz Jacobiana son almacenados en variables globales, para ser usados mas tarde en el calculo de la cinemática inversa a partir de métodos numéricos.

Posteriormente, se llama a la función `ScorbotVisionUI`. Esta es la responsable de crear la interfaz gráfica para el usuario. Lleva a cabo esta tarea creando una figura de tamaño fijo y colocando en ella un panel mediante la función de Matlab `uipanel` donde coloca los datos del robot valiéndose de las funciones `plotMatrix` y `plotAngleVector` creadas a tal efecto. Estas son las encargadas de escribir y actualizar estos datos cada vez que son llamadas. También se crean tres ejes para figuras y un botón interruptor.

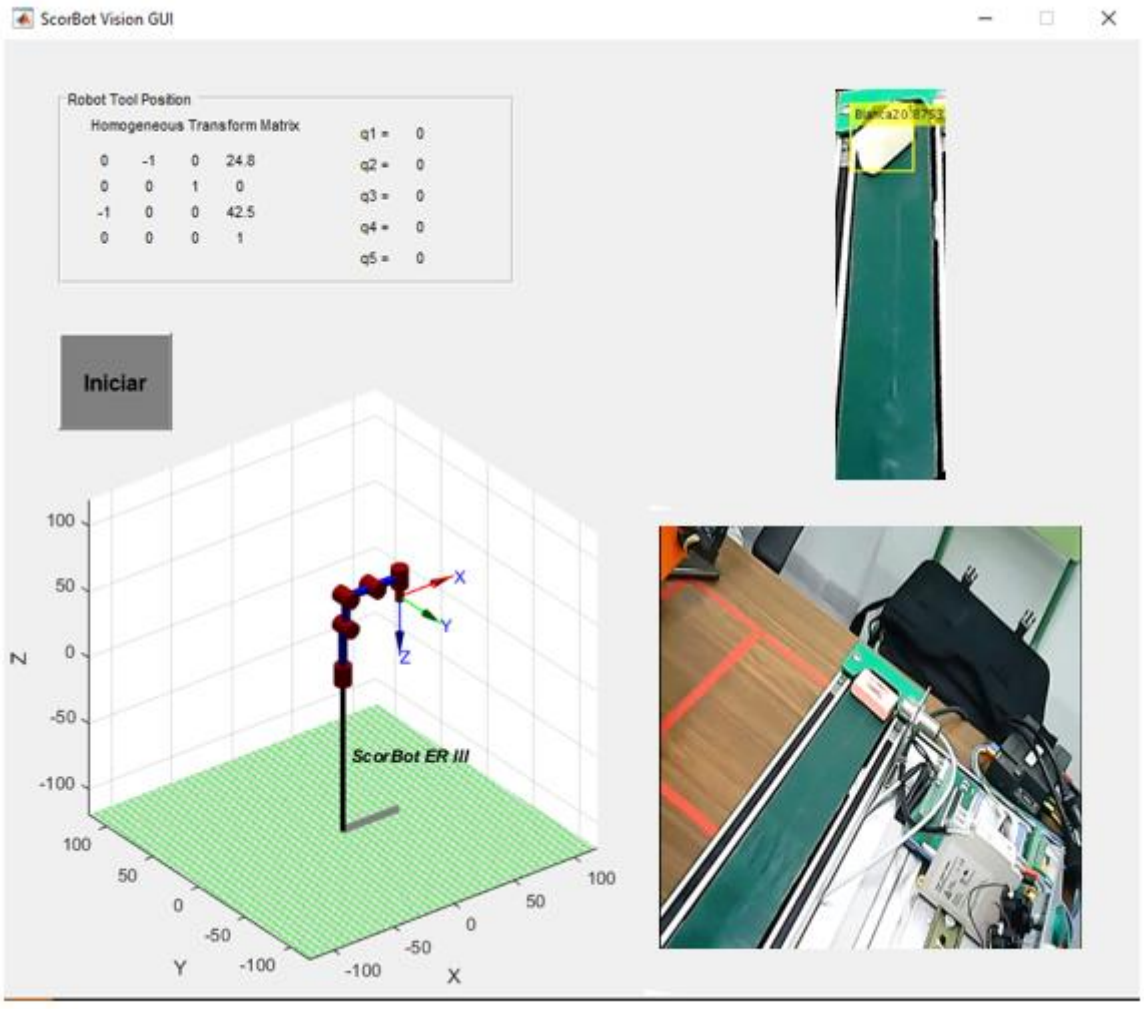


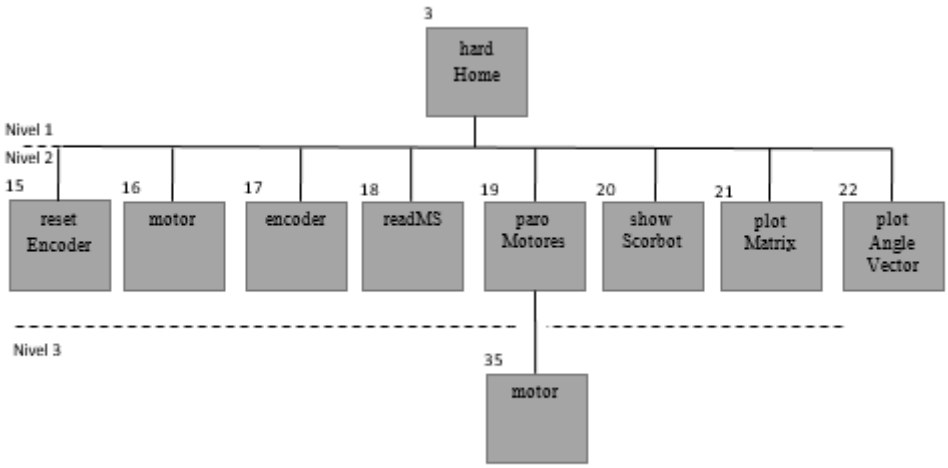
Figura 34: Interfaz gráfica del programa de control del sistema.

Como se puede observar en la figura, la interfaz gráfica se puede dividir en dos secciones, la primera, en la parte izquierda, incorpora los datos e imágenes concernientes al robot. En concreto, se muestra tanto la matriz de trasformación homogénea como el vector de coordenadas angulares del brazo robótico. El botón central es un interruptor usado tanto para iniciar el proceso de clasificación como para detenerlo.

En la imagen inferior se muestra el brazo robótico virtual, que como ya se ha comentado anteriormente, puede seguir los movimientos del Scorbot-ER III, o manejarse individualmente. Para su creación, se ha usado un paquete de funciones creado por **Peter Corke**, profesor de la Queensland University of Technology, llamado *Robotics Toolbox*. Este, entre otras, incorpora funciones tanto para la creación como para la visualización de brazos robóticos.

La sección derecha, en cambio, incorpora lo relativo a la visión artificial. En ella se muestran dos imágenes, la inferior es una retransmisión en directo del proceso visto desde la cámara web. La superior en cambio muestra la imagen una vez tratada con los algoritmos de visión artificial.

Finalmente, con `initWCam` se inicia la *webcam*, asignando el objeto correspondiente a una variable global y llamando a la función `plotPreview` para mostrar la retransmisión a tiempo real por uno de los tres ejes para figuras anteriormente creados.



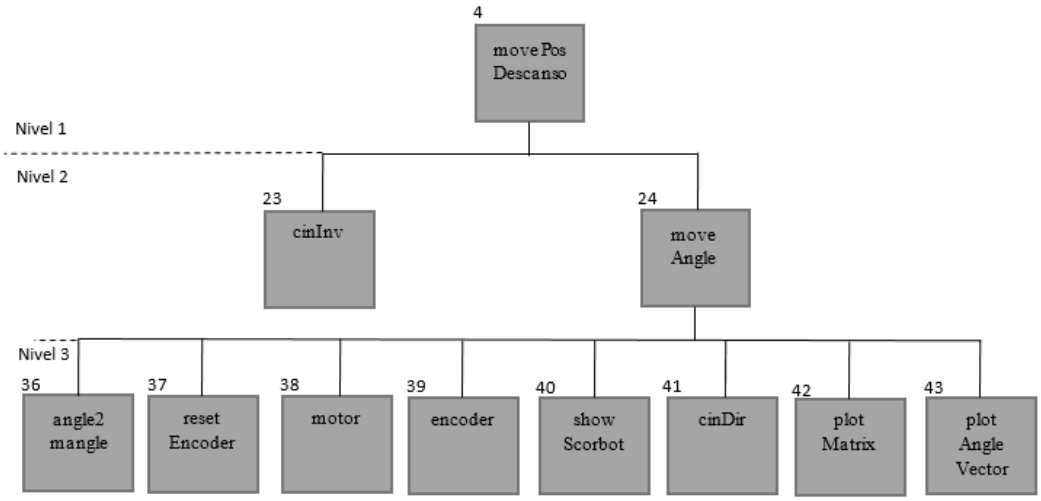
La posición llamada *hard home* es aquella en la cual todas las articulaciones del robot están pulsando sus *microswitches* correspondientes. Esto da al robot la posibilidad de alcanzar una posición exacta y conocida sin importar el estado inicial desde el cual parta.

Para ello, se resetean todos los contadores de los encoders de las articulaciones con `resetEncoder` y se inicia el movimiento de estas. Una vez hecho esto se entra en un bucle *while* cuya condición de salida es que todos los *microswitches* estén pulsados de la manera correcta. Dentro del bucle, existe una condición *if/else* para cada *microswitch* que se encarga de detener el motor si este se pulsa o de invertir su dirección si no se detectan pulsos nuevos desde el encoder. En el caso del segundo, tercer, cuarto y quinto *microswitch*, al tener estos cierta longitud,

es necesario que el *microswitch* sea pulsado desde una dirección en concreto, ya que en caso contrario daría lugar a errores de varios grados en la posición angular.

Para el posicionar la cuarta y quinta actualización, es necesario usar los motores de ambas simultáneamente. Si estos giran en el mismo sentido, variará θ_4. En caso contrario, variará θ_5.

Finalmente, una vez posicionado el robot, se muestra la nueva posición en la interfaz gráfica, además de actualizar la matriz de transformación y los valores de los ángulos.



La función que provoca que el robot se sitúe en la posición de descanso debe determinar primero mediante la llamada a la función `cinInv` las coordenadas articulares que tienen que alcanzar sus ejes. Esta función básicamente aplica los cálculos realizados anteriormente para obtener los valores de $\theta_1, \theta_2, \theta_3$ y θ_4 dada una posición deseada expresada en coordenadas cartesianas.

Una vez conocidos los ángulos, se llama a la función `moveAngle`, que se encarga de mover el brazo robótico a una posición angular introducida como entrada. Consigue esto calculando la diferencia entre la posición actual de los ejes y la posición deseada. Antes de pasar el siguiente paso, es necesario realizar cierto ajuste debido a las características del brazo robótico.

Este ajuste se lleva a cabo mediante la función `angle2mangle`, que se encarga llevar a cabo la corrección en los ángulos calculados aplicando las dos ecuaciones calculadas mas arriba, pasando de ángulos teóricos a los ángulos reales que deben girar las articulaciones.

Los valores obtenidos de esta diferencia son usados para calcular el numero de pulsos de encoder que debe dar cada uno de los motores asociados a los ejes de las articulaciones. Ello se lleva a cabo mediante la siguiente ecuación, donde Rt es la relación de transmisión:

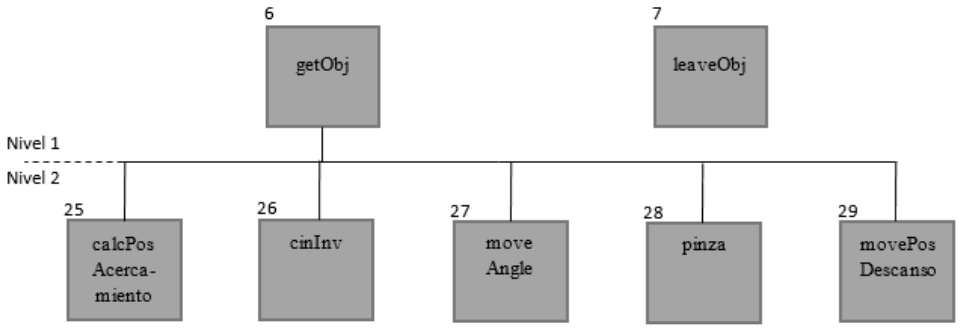
$$Pulsos = \frac{6 * Rt * (\theta_{obj} - \theta_{actual})}{\pi}$$

En el caso de las articulaciones cuatro y cinco, la ecuación para calcular los pulsos necesarios varia, ya que se complementan uno a otro. En caso de la cuarta articulación, es necesario sumar los pulsos que debe dar también la quinta, y en el caso de la quinta, se debe restar los pulsos de la cuarta.

Una vez hecho esto, se resetean los contadores de pulsos de los encoders para iniciar la cuenta, y se ponen en marcha los motores, la dirección dependerá del valor positivo o negativo del numero de pulsos obtenidos en las ecuaciones.

Se entra entonces en un bucle *while*, del cual se saldrá cuando el número de pulsos dados sea igual o mayor al numero de pulsos necesario. Según cada articulación llegue a su destino, el motor correspondiente se detendrá. De esta manera, se obtiene un movimiento tipo eje a eje del robot.

Una vez el robot está situado, se realiza la misma trayectoria en el simulador, creado un vector de matrices de trasformación homogéneas que describa la trayectoria del robot y después mostrándolo con `showScorbot`. Posteriormente se actualiza la variable global de posición angular del robot y se actualizan tanto la matriz de transformación homogénea como el vector angular presentes en la interfaz gráfica.



Las funciones `getObj` y `leaveObj` son funciones de alto nivel usadas para recoger un objeto de la cinta y para dejar un objeto en un área concreta, respectivamente. Ambas hacen uno de las mismas funciones.

La primera usa `calcPosAcercamiento`, función que, dada la posición del objeto en la cinta, devuelve una posición ligeramente superior. Esto se hace para evitar que el robot pueda golpear otros objetos o la cinta trasportadora al desplazarse hacia el objeto seleccionado.

Se calculan entonces mediante la cinemática inversa los ángulos que deben conseguir las articulaciones del robot para llegar tanto a la posición de acercamiento como a la posición donde se encuentra el objeto, añadiendo además el ángulo que debe alcanzar la quinta articulación, que vendrá dado por la orientación del objeto en la cinta, obtenido como parámetro gracias a los algoritmos de visión artificial.

Se mueve entonces el robot a la posición de acercamiento mediante `moveAngle`, y después al objeto, momento en el cual se cierra la pinza. Una vez el objeto esta cogido, se vuelve a la posición de descanso.

En el caso de `leaveObj`, el orden es básicamente el inverso, partiendo de la posición de descanso con la pinza cerrada, se va hacia la posición de clasificación en dos pasos, pasando primero por una posición ligeramente superior. Una vez hecho esto el brazo robótico baja el objeto, y abre la pinza, volviendo posteriormente a la posición de descanso, preparado para el siguiente objeto.

5. PUESTA EN MARCHA Y PROGRAMACIÓN DEL PLC

5.1 Introducción histórica

Hasta la aparición de los autómatas programables, los sistemas de control de maquinaria estaban basados en relés. Estos debían ser conectados en un orden muy especifico para que el proceso o la maquinaria que dependía de ellos funcionase, y situaciones con mucha maquinaria, el montaje de estos sistemas de control era largo y muy costoso. En caso de querer hacer variaciones en el proceso de producción, se debía volver a cablear el sistema de control entero, perdiendo horas y recursos en ello. Además, el consumo de potencia eléctrica de estos sistemas de control era enorme. A esto se le añade que su tiempo de vida era limitado, y dado que el cableado era muy complicado, detectar o corregir los elementos que fallaban era muy difícil, lo cual encarecía el mantenimiento.

A finales de los sesenta, General Motors buscaba reemplazar estos sistemas de control en su división automovilista: Buscando nuevas ideas, hace una solicitud, requiriendo, entre otras cosas:

- Un sistema flexible como un ordenador, con un precio competitivo y una lógica similar a la de los relés.
- De fácil mantenimiento y programación, en línea con el diagrama de contactos usado hasta entonces.
- Vida útil larga y resistente a ambientes difíciles.
- Diseño modular para permitir el intercambio de componentes y la escalabilidad.

De entre todas las propuestas, la ganadora fue la de la empresa Bedford Associates, que propuso un sistema llamado Modular Digital Controller (Modicon). Modicon 084 fue el primer PLC producido comercialmente.

A partir de este momento, los avances y mejoras en la funcionalidad se empezaron a suceder. Mejores microprocesadores aumentaron la capacidad de los PLCs de controlar sistemas de mayor tamaño. Empezaron a aparecer las comunicaciones junto con los estándares por los cuales estas debían regirse, lo que permitió alejar a los PLCs del proceso que controlaban.

En los últimos años, además de las mejoras en la capacidad de computación de los PLCs, se ha intentado unir estándares, para aumentar la compatibilidad de estos.

Como dato de interés, uno de los primeros Modicon 084 se encuentra actualmente en una exposición de Schneider Electric. Este fue devuelto por la empresa automovilística que lo adquirió después de veinte años de servicio ininterrumpido.

5.2 Introducción y características de los autómatas programables

Un autómata programable es un sistema basado en un microprocesador que pertenece a la familia de los computadores, y comparte con estos sus partes fundamentales:

- Unidad Central de Proceso (CPU)
- Memoria
- Sistema de Entradas y Salidas (E/S)
- Dispositivos de programación y comunicaciones
- Alimentación

La CPU realiza el control tanto interno como externo, además de interpretar las instrucciones del programa, generando entonces las señales de salida a partir de las de entrada. En la memoria el PLC guarda programa y datos.

Los dispositivos de programación y comunicaciones permiten al autómata programable comunicarse con los dispositivos periféricos, como otros PLCs o pantallas, y el usuario. Mediante el sistema de Entradas y Salidas se recoge la información del proceso controlado y se envían las acciones de control a este.

Una manera de clasificar los autómatas programables es atendiendo a la estructura por la cual se organizan sus componentes anteriormente mencionados, acorde a esta clasificación, existen:

- PLC Nano
- PLC Compacto
- PLC Modular

El PLC Nano suele ser parecido al compacto, con la diferencia de que este solo puede manejar un conjunto muy reducido de entradas y salidas.

El PLC Compacto tiene incorporaba en el mismo módulo la fuente de alimentación, la CPU, y los módulos de entrada salida. Además, soportan un numero mayor de entradas/salidas y permiten la incorporación de una gran variedad de modelos adicionales.

Finalmente, el PLC Modular se compone de una serie de módulos que contienen los elementos que unidos conforman el controlador final. Estos son el rack, la fuente de alimentación, la CPU y los módulos E/S.

5.3 Siemens S7-200. Características técnicas

La gama s7-200 comprende diversos sistemas de automatización pequeños (PLC Nano) que pueden ser utilizados para numerosas tareas. La CPU s7-200 incorpora en una carcasa compacta un microprocesador, una fuente de alimentación integrada y circuitos de E/S, aunque al ser el PLC de corriente continua, requiere de una fuente de alimentación externa de 24VDC. Este ha sido diseñado de manera que pueda montarse en un panel o rail normalizado DIN, ya sea horizontal o verticalmente.

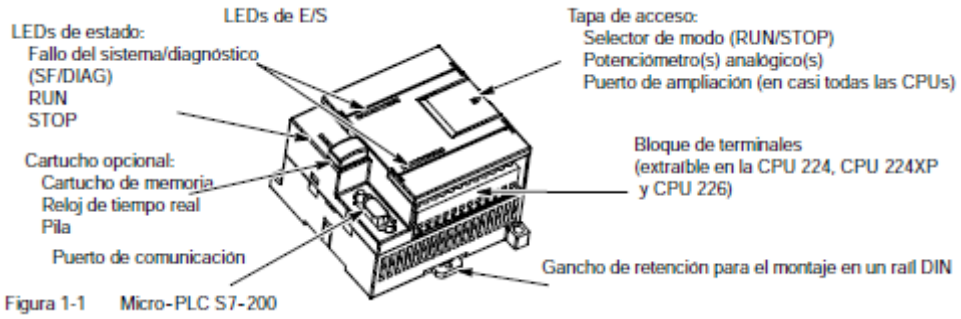


Figura 35: Siemens S7-200 y componentes exteriores.

Siemens ofrece distintos modelos de CPU s7-200 que varían es sus funciones y prestaciones, de manera que puedan abarcar distintas aplicaciones.

Las entradas y salidas se hallan etiquetadas en los bornes y en las luces de indicación. Estas etiquetas identifican las E/S en la programación y relacionan a los dispositivos conectados. Con I se reconocen las entradas digitales, con Q las salidas digitales. Además de las letras, se usa una combinación de dos números separados por un punto, con el primer numero identificando el byte, y el segundo el bit (que ira de cero a siete).

En cuanto a la programación, los tres lenguajes de programación que utilizan los PLCs Siemens de la familia s7-200 son:

- Diagrama de escalera (KOP)
- Lista de instrucciones (AWL)
- Diagrama de bloque de funciones (FUP)

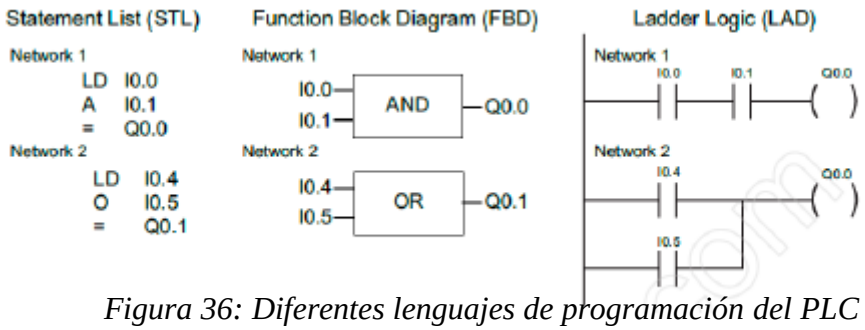


Figura 36: Diferentes lenguajes de programación del PLC

El programa de un PLC se ejecuta repetitivamente, La ejecución empieza con la lectura de las entradas, cuyo estado se almacena en el registro de la entrada. Se procede entonces a ejecutar el programa y obtener el registro de salida. Se ejecutan entonces las operaciones de diagnostico y comunicaciones pertinentes para finalmente actualizar el registro de salidas.

Para la programación se utiliza el paquete de programación STEP 7-Micro/WIN, que constituye un entorno de fácil manejo para desarrollar, editar y observar el programa necesario con objeto de controlar la aplicación.

El s7-200 tiene dos modos de operación, STOP y RUN, indicados por los LEDs de estado. En el modo STOP, el s7-200 no ejecuta el programa, por lo que es posible cargar un nuevo programa o configurar la CPU. En modo RUN, el s7-200 ejecuta el programa. Se utiliza el selector que incorpora el autómata programable para cambiar de modo.

5.4 Programación del PLC

En este proyecto, al autómata programable se le ha asignado la tarea del control de la cinta transportadora mediante el motor acoplado a esta. Como ya se ha mencionado, dispone para ello de las siguientes entradas:

- Sensor óptico.
- Pulsador, que será usado para iniciar el proceso.
- Interruptor, usado para cambiar entre los modos Manual y Automático.
- Seta de emergencia.

El objetivo principal del PLC debe ser llevar los objetos situados en la cinta al área de recogida, para que sea recogido por el brazo robótico y clasificado.

Para ello, se pretende programar el autómata de manera que:

1. El movimiento de la cinta transportadora se inicie al pulsar el botón Marcha.
2. El movimiento continúa hasta que se detecta un objeto con el sensor óptico. En ese momento la cinta se detiene.
3. Si al quitar el objeto de la zona de recogida el interruptor está puesto en modo Manual, la cinta seguirá parada hasta que se vuelva a pulsar Marcha. Si está en modo Automático, se vuelve al punto 2.
4. Pulsar otra vez Marcha detiene el proceso.

5.4.1 GRAFCET

Para facilitar la creación del programa que cumpla con estos requisitos, se usara la herramienta GRAFCET.

El GRAFCET es un modelo de representación gráfica de los sucesivos comportamientos de un sistema lógico, predefinido por sus entradas y salidas Permite hacer un modelo del proceso a automatizar, contemplando entradas, acciones a realizar, y los procesos intermedios que provocan estas acciones. Este se ha universalizado como herramienta de modelado que permite el paso directo a programación.

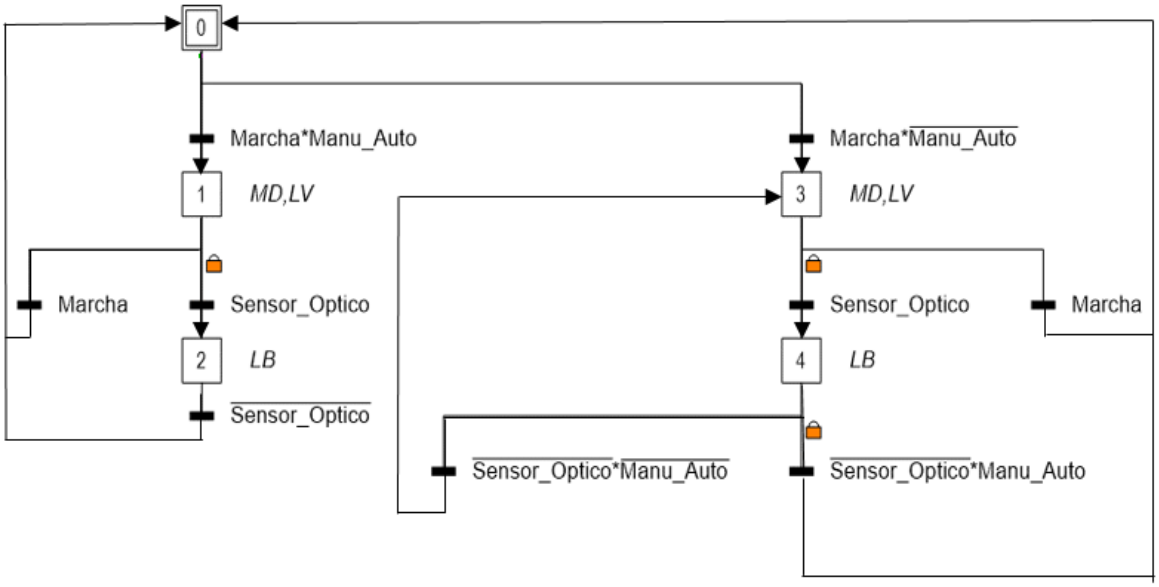


Figura 37: Diagrama GRAFCET para el control de la cinta transportadora.

En la figura 36 se muestra el GRAFCET con el programa realizado cumpliendo con los requisitos mencionados anteriormente. Partiendo del estado inicial, se puede observar que el gráfico se divide en dos ramas, dependiendo de en que posición se encuentre el interruptor de Manual/Automático. La rama izquierda corresponde al interruptor situado en Manual. Como se puede ver, al pulsar el botón de marcha, el motor se pone en funcionamiento, haciendo avanzar la cinta transportadora. Además, se enciende una luz verde indicando el movimiento del motor. Esta situación se mantiene hasta que se vuelva a pulsar Marcha, momento en el cual el motor se detiene y se apaga la luz, o hasta que el sensor óptico detecta un objeto, encendiéndose entonces una luz blanca y deteniéndose el motor. Cuando el sensor óptico deje de detectar una pieza se vuelve al estado inicial.

En caso de que el interruptor este situado en modo Automático y se pulse marcha, se pasa al estado 3, las acciones llevadas a cabo en esta rama son las mismas que en la rama contigua. La excepción se da al dejar de detectar el sensor óptico un objeto. En ese momento, se vuelve al estado 3, en vez de al estado inicial, y se pone en marcha el motor de nuevo. En caso de que durante la ejecución

de todo el proceso el interruptor haya sido cambiado de posición y puesto en modo Manual, se pasaría al estado inicial, quedando a la espera de nuevas ordenes.

5.4.2 Ecuaciones de transición entre estados.

En este punto se van a definir las distintas transiciones que deben cumplirse para conseguir el cambio entre todos los diferentes estados lógicos del sistema.

$$E_0 = E_2 * \overline{SensorOptico} + E_5 * ManuAuto + E_0 * \overline{E_1} * \overline{E_3} + \overline{E_1} * \overline{E_2} * \overline{E_3} * \overline{E_4} * \overline{E_5}$$

$$E_1 = E_0 * ManuAuto * Marcha + E_1 * \overline{E_2}$$

$$E_2 = E_1 * SensorOptico + E_2 * \overline{E_0}$$

$$E_3 = E_0 * \overline{ManuAuto} * Marcha + E_5 * \overline{ManuAuto} + E_3 * \overline{E_4}$$

$$E_4 = E_3 * SensorOptico + E_4 * \overline{E_5}$$

$$E_5 = E_4 * \overline{SensorOptico} + E_5 * \overline{E_0} * \overline{E_3}$$

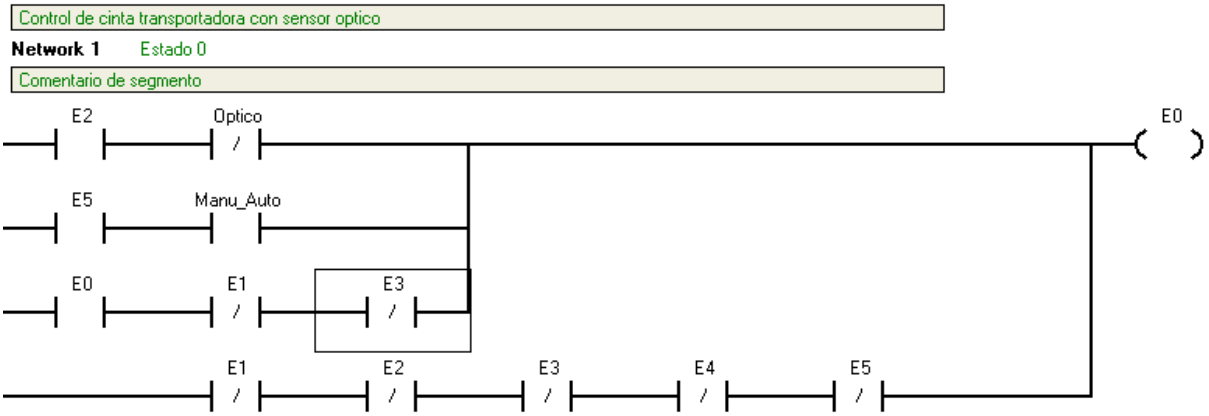
5.4.3 Conexiones del s7-200 y direccionamiento de las variables

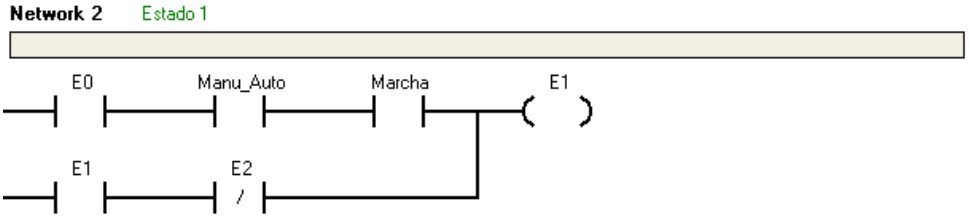
EL primer paso de la programación del PLC con STEP 7-Micro/WIN es necesario definir las variables, tanto entradas como salidas, que se van a utilizar, y sus direcciones físicas. Esto hará mas sencillo para el usuario referirse a ellas, al ponerles nombre acorde a su función. La tabla de direccionamiento de las variables se muestra a continuación.

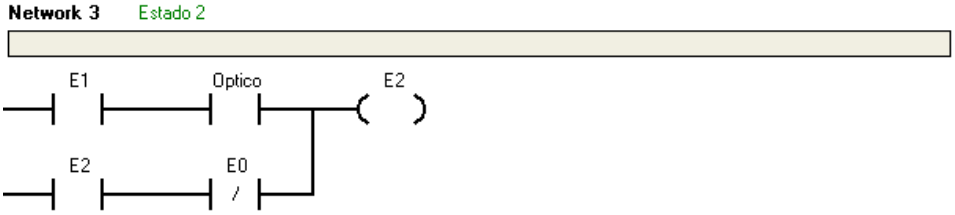
Símbolo	Dirección	Comentario
Marcha	I0.0	Pulsador de marcha
Manu_Auto	I0.1	Interruptor manual/automatico
Optico	I0.3	Sensor Optico
LV	Q0.0	Luz Verde
LB	Q0.1	Luz Blanca
MD	Q0.2	Motor derecha
MI	Q0.3	Motor Izquierda
E0	M0.0	Estado 0
E1	M0.1	Estado 1
E2	M0.2	Estado 2
E3	M0.3	Estado 3
E4	M0.4	Estado 4
E5	M0.5	Estado 5

Figura 38: Tabla de direccionamiento de variables.

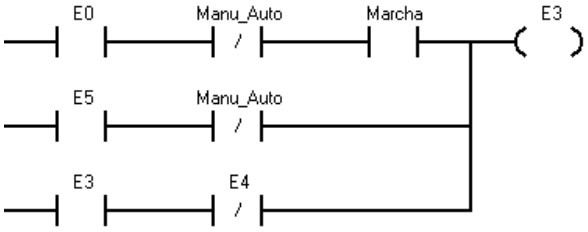
El programa final se muestra a continuación. Es la expresión en lenguaje ladder de las ecuaciones de activación/desactivación obtenidas en el apartado anterior.



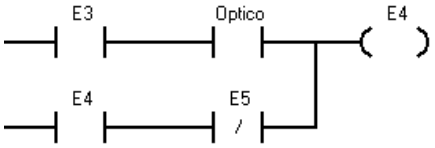




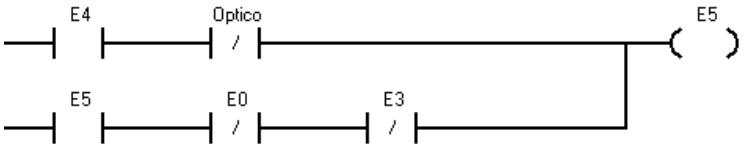
Network 4 Estado 3



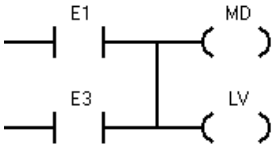
Network 5 Estado 4



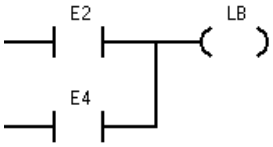
Network 6 Estado 5



Network 7 Combinacional



Network 8



6. RESULTADOS Y ANÁLISIS

6.1 Resultados experimentales

Para medir la eficacia del robot, se ha realizado una serie de pruebas usando gomas y variando el ángulo y color de estas.

Para ello, se han elegido cuatro ángulos, de 0º, 45º, -45º y 90º. Esta configuración en particular se ha elegido porque era sencilla de aplicar, además de que con pequeñas variaciones se recoge todo el espectro de posibilidades que tiene el objeto para orientarse.

Figura 39: Orientación de objetos desde el punto de vista del sistema del brazo robótico.

Se han realizado un total de ochenta pruebas, diez por cada ángulo y color, comprobando en cada una de ellas si el resultado del clasificador de color coincidía con la realidad y comprobando después que el brazo robótico realizase el proceso de recogida satisfactoriamente. Los resultados obtenidos se muestran a continuación.

	Clasificadas con éxito (%)	Recogidas con éxito (%)
0º	100	65
45º	100	70
-45º	100	65
90º	100	45

Se puede ver que la clasificación del color realizada por el algoritmo de visión por computador es correcta en cada una de las pruebas realizadas. La fase de recogida, en cambio, no tiene un nivel

de aciertos tan elevado, aunque, dado que el nivel de error se mantiene constante a pesar de la variación de la orientación, se deduce que no viene afectado por esta excepto en el caso de las gomas orientadas a 90º. La orientación de estas, junto con el sensor óptico, dificultan enormemente la recogida ya que la pinza situada en el extremo del brazo robótico no dispone de espacio para trabajar, tal y como se muestra en la siguiente imagen.

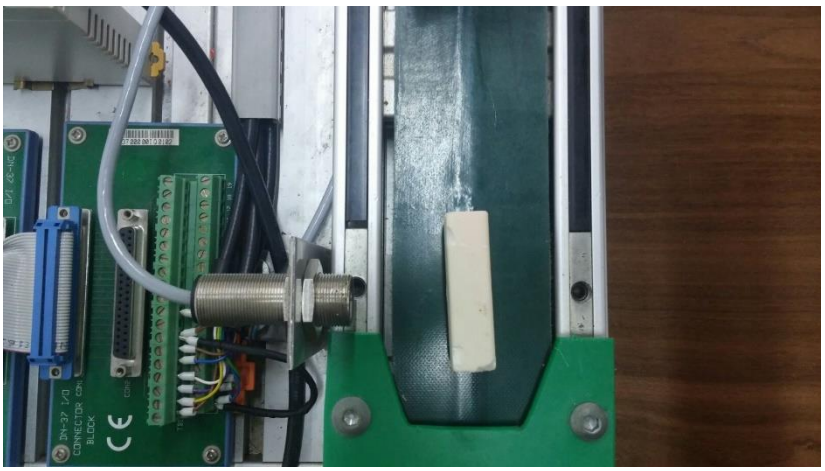


Figura 40: Sensor óptico y objeto situada a 90º.

Además de esta, existen varias causas que explican la aparición de estos errores en la recogida de objetos, las cuales se discuten en el apartado siguiente.

Otra de las pruebas realizadas compara la eficiencia de los dos algoritmos usados para hallar la cinemática inversa del brazo robótico. Se mide tanto la proximidad de las soluciones proporcionadas por estos a la realidad, como el tiempo invertido en hallar dichas soluciones.

Para ello, se han aplicado ambos algoritmos para hallar las coordenadas angulares necesarias para llevar al brazo robótico a una serie de puntos, asegurando que estos sean alcanzables por el robot. Dado que el programa creado para hallar la solución mediante métodos numéricos admite que el usuario seleccione un margen de error para la solución, se ha intentado que este sea un margen aceptable sin ser demasiado estricto.

Los resultados obtenidos tras realizar las pruebas sobre 1700 puntos han sido los siguientes:

	Error medio respecto a la posición (%)	Tiempo medio de cómputo (s)
Métodos Numéricos	1,5	15
Ecuaciones (MTH)	0,5	$0{,}5*10^{-3}$

Como se puede observar, la resolución mediante ecuaciones extraídas de la Matriz de Transformación Homogénea es significativamente más precisa y más rápida. Este último factor

permite resolver la cinemática inversa en tiempo real, por lo que este tipo de solución suele ser el más recomendado. En cambio, los métodos numéricos son utilizados para la resolución de la cinemática inversa en manipuladores robóticos cuyas configuraciones no permiten encontrar una solución mediante la MTH. De hecho, la ventaja de esta última es que se puede ajustar para encontrar mejores resultados a cambio de un mayor tiempo de computación, o viceversa. Además, su rapidez depende de la potencia de computación del sistema encargado de realizar los cálculos, por lo que, con un sistema dedicado con la suficiente potencia, puede alcanzar tiempos mucho menores por cálculo.

6.2 Análisis de las causas de error

En este apartado se comentará brevemente las posibles causas del porcentaje de fallos obtenidos en los resultados experimentales concernientes a la recogida de objetos por parte del robot.

Existen tres razones principales que puede explicarlos:

- Acoplamiento de los motores cuarto y quinto.
- Pérdida de pulsos de los encoder.
- Características del movimiento de las articulaciones.

La primera de ellas fue comentada por Rubén Jiménez Andreu en su TFG "Adaptación de un robot SCORBOT-ER III para su control usando Arduino". Los motores 4 y 5 son usados simultáneamente para mover la cuarta y quinta articulación, de manera que si estas giran en direcciones opuestas θ_4 varía, es decir, el extremo del robot se eleva o desciende. En caso de que giren en la misma dirección, provocan que el extremo rote sobre si mismo, que θ_5 varíe.

Por lo tanto, para que θ_4 y θ_5 giren con precisión, es necesario que ambos motores se muevan exactamente a la misma velocidad durante el mismo tiempo, o que se muevan los mismos grados En caso contrario, un movimiento en θ_4 provocaría a su vez movimiento en θ_5 que no quedaría registrado, llevando a que el robot no estuviese coordinado. Esto no sucede así, por factores como que las reducciones de los motores no son exactas, o que el robot es viejo y lleva mucho tiempo sin usar ni lubricar, por lo que sus sistemas de transmisión no funcionan como deberían.

Es posible que esta última razón explique además la segunda causa de error, la pérdida de pulsos. Circunstancia que se ve agravada además por la conexión Arduino-Ordenador, que no es instantánea. Esto conlleva que muchas veces, la lectura de un encoder llegue al ordenador con cierto retraso, causando que un motor se detenga habiendo avanzado mas de lo que debería. Se han llevado a cabo pruebas para intentar demostrar y cuantificar este error, cuyos resultados se exponen a continuación:

	Error medio (%)
θ_1	1,3
θ_2	1,2
θ_3	1,7
θ_4	4,5
θ_5	1,5

Es importante comentar que este error aumentará en caso de que el ordenador o sistema tenga mas carga computacional, y que, a pesar de no ser muy grande, su acumulación debida a repetidos movimientos del brazo robótico provoca que se deba recalibrar este.

El tercer factor de error es el comentado en el apartado 4.3.1. El movimiento de θ_2 provoca variaciones en θ_3 y θ_4, y el movimiento de θ_3 las provoca en θ_4. Aunque se ha intentado modelar estas variaciones a fin de poder contrarrestarlas en los cálculos realizados, es posible que el modelo definido por dichas ecuaciones no sea exacto, debido entre otras cosas a que ha sido obtenido de forma visual.

$$\theta_3^m = \theta_2 + \theta_3$$

$$\theta_4^m = \theta_2 + \theta_3 + \theta_4$$

7. CONCLUSIONES Y DESARROLLO FUTURO

7.1 Conclusiones

Como conclusión, se puede afirmar que se han cumplido satisfactoriamente los objetivos marcados para el proyecto descritos en la introducción.

La maqueta de prácticas ha sido reparada, y está lista para su uso ya sea en una continuación de este Trabajo de Fin de Grado o en las prácticas de una asignatura del Departamento de Sistemas y Automática.

Se ha programado y cableado un PLC para que realice el control de la cinta transportadora de manera que esta actúe coordinadamente con el brazo robótico.

Se ha instalado una cámara web y se ha diseñado un algoritmo para el sistema de visión artificial capaz de reconocer el objeto mas cercano al final de la cinta trasportadora, así como su color y posición.

Se ha diseñado un sistema de control del brazo robótico específicamente para la tarea de recoger objetos en la cinta transportadora, capaz de recoger la información proveniente del sistema de visión artificial y usarla para realizar un agarre satisfactorio.

Por último, se ha creado una interfaz gráfica para que un usuario sin conocimientos de programación pueda poner en marcha y detener el proceso, así como tener información visual de su estado mediante un robot simulado. Esta interfaz grafica también permite realizar los movimientos solo en el brazo robótico simulado, proporcionando un área de pruebas sin riesgo para el usuario.

7.2 Desarrollo futuro

Este Trabajo de Fin de Grado permite continuar trabajando en él, mejorando factores como la precisión, velocidad, etc. El uso de un microcontrolador intermedio, como una RaspBerry Pi, encargada de realizar el control a bajo nivel del robot, podría suponer un intercambio de datos mas rápido y por lo tanto una mayor velocidad y precisión. Dejando aún el control a alto nivel a un ordenador que disponga de Matlab.

Esta mayor capacidad de computación y velocidad permitiría además el uso de controladores PID para los motores encargados de mover las articulaciones del robot. Estos lazos cerrados de control permitirían corregir errores y un control mayor de los movimientos del robot por parte del usuario, permitiendo el por ejemplo el diseño de trayectorias, etc.

En cuanto a la visión artificial, existe la posibilidad de incluir una segunda cámara calibrada para visión estéreo, lo cual permitiría obtener mas información del sistema y por ejemplo crear lazos cerrados de control del robot usando dicha información como retroalimentación. Una cámara de visión artificial con una mayor resolución permitiría además obtener información mas detallada de la imagen, abriendo posibilidades en la identificación de taras en los objetos, etc.

Estos son solo unos ejemplos de la gran cantidad de posibilidades que ofrece este sistema robot-cinta transportadora.

ANEXO A: CONEXIONES BORNERA DN37

	Pin	Color	Descripción
Cableado Panel de Control	1	Azul	Alimentación
	2	Verde	Señal Emergencia
	20	Marrón	Pulsador Marcha
	21	Amarillo	Interruptor Manu/Auto
	22	Morado	Luz Verde
	23	Blanco	Luz Blanca
	9	Negro	Tierra
Cableado Control Motor	3	Rojo	Alimentación Motor
	24	Azul	Señal Motor Derecha
	25	Marrón	Señal Motor Izquierda
	10	Negro	Tierra
Cableado Encoder	26	Marrón	Pulsos Encoder
	8	Negro	Tierra
	4	Azul	Alimentación
Cableado Sensor Óptico	7	Azul	Tierra
	27	Negro	Señal
	5	Marrón	Alimentación

ANEXO B: CÓDIGO EN MATLAB

Función usada para iniciar el proceso de clasificación-recogida.

```
function  initClassifier()
%CLASSIFIER_INI Inicia todo el proceso.
%   El robot empieza en la posicion de descanso, donde se queda hasta
%   recibir la señal de objeto detectado. Entonces lo coge y lo deja    %
en la zona de clasificacion.
clc;
global init_process;
initSist;
msg_asegurar=msgbox('Por favor, asegurese de que la fuente de alimentacion
está activada.','Advertencia','warn');
msg_asegurar.WindowButtonUpFcn=@uiresume;
uiwait(msg_asegurar);
hardHome;
movePosDescanso;
uiwait(gcf);
while init_process==1%Mientras no se pulse vuelva a pulsar el boton de
%marcha de la UI
    [ outImage, Orientation, posImage, Color]=getVisionData();
    plotImage(outImage);
    if posImage(2)<=80 %Coge el objeto al alcanzar este cierto rango
        getObj(-Orientation*2*pi/360);
        leaveObj(Color);
        hardHome;
    end
    pause(0.5);
end
msg_asegurar=msgbox('Por favor, asegurese de que la fuente de alimentacion
está apagada.','Advertencia','warn');
msg_asegurar.WindowButtonUpFcn=@uiresume;
uiwait(msg_asegurar);
msgbox('Proceso detenido.');
close all force
clear global
end
```

Función usada para cargar los datos, variables, interfaz gráfica…

```
function initSist

global posicion_cartesiana_robot;
global posicion_angular_robot;

newIniciarArduino;
cargaDatos;
initNumIKin;
%%Inicializacion de la UI
ScorBot_Vision_UI;
plotMatrix(posicion_cartesiana_robot);
plotAngleVector(posicion_angular_robot);
initWCam;
end
```

Función usada para situar al brazo robótico en *Hard Home*.

```
function hardHome()
% Función para colocar el brazo en la posción inicial o "hard home"

global USE_ARDUINO
global posicion_angular_robot;
if USE_ARDUINO==1
    global a;
    HH=zeros(1,4); %Variables booleanas donde que seran true(1) cuando el
motor este
    %posicionado.
    digitalWrite(a,52,1); % Se enciende el tercer led rojo
    fprintf('Situando articulaciones en Hard Home\n')
    dir=ones(1,5);%%Vector con las direcciones de los motores.
    dir(3)=0;
    vel=[0.8 0.5 1 0.5 0.5]; %Vector de velocidades
    %%Comienza el movimiento de todos los motores
    for i=1:5
        resetEncoder(i);
        motor(i,dir(i),vel(i))
    end
    enc1=encoder(1);
    enc2=encoder(2);
    enc3=encoder(3);
    enc4=encoder(4);
    enc5=encoder(5);
    nivel_ant2=readMS(2);
    nivel_ant3=readMS(3);
    nivel_ant4=readMS(4);

    while ~(isequal(HH,ones(1,4))) %Mientras no esten todas las
articulaciones posicionadas
        pause(0.01);
        if (~HH(1))
            if (readMS(1))
                fprintf('Se ha pulsado el microswitch 1\n')
                motor(1,0,0)
```

```
                HH(1)=1;
            elseif (enc1==encoder(1))
                dir(1)=~dir(1);
                motor(1,dir(1),vel(1))
            else
                enc1=encoder(1);
            end
        end
        if (~HH(2))
            nivel_ant2=[nivel_ant2 readMS(2)];
            if (dir(2)&&~nivel_ant2(end-1)&&nivel_ant2(end))
                fprintf('Se ha pulsado el microswitch 2\n')
                motor(2,0,0)
                HH(2)=1;
            elseif ((dir(2)&&nivel_ant2(end-
1)&&~nivel_ant2(end))||enc2==encoder(2)||(~dir(2)&&nivel_ant2(end-
1)&&~nivel_ant2(end)))
                dir(2)=~dir(2);
                motor(2,dir(2),vel(2))
            else
                enc2=encoder(2);
            end
        end
        if (~HH(3)&&HH(2))
            nivel_ant3=[nivel_ant3 readMS(3)];
             if (dir(3)&&~nivel_ant3(end-1)&&nivel_ant3(end))
                 fprintf('Se ha pulsado el microswitch 3\n')
                motor(3,0,0)
                HH(3)=1;
            elseif ((dir(3)&&nivel_ant3(end-
1)&&~nivel_ant3(end))||enc3==encoder(3)||(~dir(3)&&nivel_ant3(end-
1)&&~nivel_ant3(end)))
               dir(3)=~dir(3);
                motor(3,dir(3),vel(3))
            else
                enc3=encoder(3);
            end
        end
        if (~HH(4)&&HH(3))
            nivel_ant4=[nivel_ant4 readMS(4)];
            if (~dir(4)&&~nivel_ant4(end-1)&&nivel_ant4(end))
                fprintf('Se ha pulsado el microswitch 4\n')
                motor(4,0,0)
                motor(5,0,0)
                HH(4)=1;
            elseif ((dir(4)&&nivel_ant4(end-
1)&&~nivel_ant4(end))||(enc4==encoder(4))||(enc5==encoder(5))||(~dir(4)&&niv
el_ant4(end-1)&&~nivel_ant4(end)))
                dir(4)=~dir(4);
                dir(5)=~dir(5);
                motor(4,dir(4),vel(4))
                motor(5,dir(5),vel(5))
            else
                enc4=encoder(4);
                enc5=encoder(5);
            end
        end
    end

    %%Posicionamiento por rotacion de la muñeca.
    dir(4)=1;
```

```
    dir(5)=0;
    motor(4,dir(4),vel(4));
    motor(5,dir(5),vel(5));
    while (~readMS(5))
        pause(0.01)
    end
    paroMotores
    fprintf('MS 5 pulsado\n');
    %Desplazamiento para corregir la posicion de la muñeza del robot.
    posAlcanzada=0;
    resetEncoder(5)
    motor(4,0,0.5);
    motor(5,1,0.5);
    while ~posAlcanzada
        if (encoder(5)>=90)
            posAlcanzada=1;
        end
    end
    paroMotores
    posAlcanzada=0;
    motor(4,0,0.5);
    motor(5,0,0.5);
    while ~posAlcanzada
        if (encoder(5)<=55)
            posAlcanzada=1;
        end
    end
    paroMotores
    digitalWrite(a,51,0); % Se apaga el LED
end
posicion_angular_robot=zeros(1,5);
showScorBot([0 0 0 0 0]);
plotMatrix(cinDir(posicion_angular_robot));
plotAngleVector(posicion_angular_robot);
fprintf('\n Todas las articulaciones han sido posicionadas en Hard Home\n')
```

Función para hacer que el brazo robótico se sitúe en la posición de descanso.

```
function  movePosDescanso()
%POSDESCANSO Mueve al robot a una posicion de reposo usada entre recogidas.
%   La posicion en coordenadas cartesianas es (31.85,0,20)
global posDescanso;

Angulos_descanso=cinInv(posDescanso);
Angulos_descanso= [Angulos_descanso 0];
moveAngle(Angulos_descanso);
end
```

Función usada para extraer la información de una imagen.

```
function [ outImage, Orientation, Pos, Color]=getVisionData()
%Funcion que no tiene entradas y devuelve la orientacion del objeto
%superior.
global cam;
imagenGoma=snapshot(cam);

imagenRot=imrotate(imagenGoma,40);
cropRect=[287.5 295.5 102 433];
imagenCrop=imcrop(imagenRot,cropRect);
imagenGris=rgb2gray(imagenCrop);
imagenGris = imgaussfilt(imagenGris,2);
imagenBW=im2bw(imagenGris,0.70);
se=strel('rectangle',[5 12]);
imagenOpen=imopen(imagenBW,se);
imagenClose=imclose(imagenOpen,se);
imagenClose = bwareaopen(imagenClose,700);

imagenLabel=bwlabel(imagenClose,8);
imgStats=regionprops(imagenLabel,'BoundingBox','Centroid','Orientation');
if size(imgStats,1)>0
    CentroideSuperior=size(imagenLabel,1);
    for i=1:size(imgStats,1)
        if (imgStats(i).Centroid(2)<CentroideSuperior)
            CentroideSuperior=imgStats(i).Centroid(2);
            ObjetoSuperior=i;
        end
    end
    %Obtencion color
    binCrop=imcrop(imagenClose,imgStats(ObjetoSuperior).BoundingBox);
    colorCrop=imcrop(imagenCrop,imgStats(ObjetoSuperior).BoundingBox);
    goma=colorCrop;
    goma(:,:,1)=colorCrop(:,:,1).*uint8(binCrop);
    goma(:,:,2)=colorCrop(:,:,2).*uint8(binCrop);
    goma(:,:,3)=colorCrop(:,:,3).*uint8(binCrop);
    r_lvl=sum(sum(goma(:,:,1)));
    g_lvl=sum(sum(goma(:,:,2)));
    b_lvl=sum(sum(goma(:,:,3)));
    color_lvl=[r_lvl,g_lvl,b_lvl]/sum(sum(binCrop));
    coefficients = polyfit([245, 255], [228, 248], 1);
    a = coefficients (1);
    b = coefficients (2);
    if (color_lvl(2)-a*color_lvl(1)-b)>0
        Color='Blanca';
    else
        Color='Roja';
    end
    Pos=imgStats(ObjetoSuperior).Centroid;
    Orientation=imgStats(ObjetoSuperior).Orientation;
    gomaLoc =
insertShape(imagenCrop,'rectangle',imgStats(ObjetoSuperior).BoundingBox,'Lin
eWidth',2);
    gomaLoc = insertText(gomaLoc, imgStats(ObjetoSuperior).BoundingBox(1:2),
Color,'FontSize',14);
    gomaLoc = insertText(gomaLoc,
imgStats(ObjetoSuperior).BoundingBox(1:2)+[0 30],
num2str(Orientation),'FontSize',20);
    outImage=gomaLoc;
else
```

```
    outImage=imagenClose;
    Pos=[NaN NaN];
    Orientation=NaN;
    Color=NaN;
end
```

Función llamada para hacer que el robot recoja un objeto de la cinta transportadora.

```
function getObj(orientacion_objeto,pos_objeto)
%GETOBJ Comienza proceso para recoger objeto.

global posObjeto;
if nargin<2
    pos_objeto=posObjeto;
end
pos_acercamiento=calcPosAcercamiento(pos_objeto);
angulos_objeto=cinInv(pos_objeto);
angulos_acercamiento=cinInv(pos_acercamiento);
angulos_objeto=[angulos_objeto orientacion_objeto+angulos_objeto(1)];
angulos_acercamiento=[angulos_acercamiento
orientacion_objeto+angulos_objeto(1)];
pinza('a')
moveAngle(angulos_acercamiento);
pause(0.5);
moveAngle(angulos_objeto);

pinza('c')%El robot ya se encuentra sobre el objeto
pause(0.5);
moveAngle(angulos_acercamiento);
pause(0.5);
movePosDescanso;
end
```

Función llamada para hacer que el robot deje un objeto en un área asignada.

```
function  leaveObj( posicion )
%LEAVEOBJ Comienza proceso para dejar el objeto.
global pos_clas_Rojo pos_clas_Blanco;
if strcmp(posicion,'Roja')
    pos_clasificacion=pos_clas_Rojo;
elseif strcmp(posicion,'Blanca')
    pos_clasificacion=pos_clas_Blanco;
else
    pos_clasificacion=posicion;
end
pos_acercamiento=calcPosAcercamiento(pos_clasificacion);
angulos_clasificacion=cinInv(pos_clasificacion);
angulos_acercamiento=cinInv(pos_acercamiento);
```

```
angulos_clasificacion=[angulos_clasificacion 0];
angulos_acercamiento=[angulos_acercamiento 0];
moveAngle(angulos_acercamiento);
pause(0.5);
moveAngle(angulos_clasificacion);

pinza('a')
pause(0.5);
movePosDescanso;
end
```

Función usada para establecer la comunicación con el Arduino.

```
function newIniciarArduino

global USE_ARDUINO
USE_ARDUINO=1;
if USE_ARDUINO==1
    global a;
    a=arduino('COM3');
    %Pines de PWM, dirección y los LEDs rojos
    global pinPWM;
    pinPWM=[3 5 6 9 10 11];
    global pinDireccion;
    pinDireccion=[2 4 7 8 12 13];
    global pinLed;
    pinLed=[50 51 52];
    %Establecer los pines como salidas analogicas y digitales
    %Para los motores del 1 al 6
    for i=1:6
        %Establecer los pines como salidas digitales
        pinMode(a,pinDireccion(i),'output');
        %Poner las direcciones y los PWM a 0
        digitalWrite(a,pinDireccion(i),0);
        analogWrite(a,pinPWM(i),0);
    end
    %Secuencia de luces con los LEDs
    for i=1:3
        % Se establecen los pines de los LEDs
        % como salidas digitales
        pinMode(a,pinLed(i),'output');
        %Se encienden secuencialmente durante
        %1/4 de segundo y se apagan
        digitalWrite(a,pinLed(i),1);
        pause(0.25);
        digitalWrite(a,pinLed(i),0);
    end
    % Mantenmos el LED 1 encendido indicando que
    % que el Arduino está conectado.
    digitalWrite(a,pinLed(1),1);
    %Imprimimos por consola que el Arduino está conectado.
    fprintf('Arduino conectado\n');
end
```

Función encargada de la carga de datos y variables globales.

```
function cargaDatos()

%Medidas del robot
global l0 l1 l2 l3 l4 l5 a1;
l0=21;
l1=14;
l2=22.5;
l3=22.3;
l4=8.5;
l5=6.5;
a1=2.5;
global posicion_angular_robot;
posicion_angular_robot=zeros(1,5);
global posicion_cartesiana_robot;
posicion_cartesiana_robot=cinDir(posicion_angular_robot);
global relacion_transmision;
relacion_transmision=[600 523.068 523.068 120 120];
global posDescanso
posDescanso=[31.85 0 36];global posObjeto;
posObjeto=[31 17.5 14];
global pos_clas_Rojo pos_clas_Blanco;
pos_clas_Rojo=[0 25 30];
pos_clas_Blanco=[0 -25 30 ];
end
```

Función encargada de iniciar las variables y las ecuaciones necesarias para la cinemática inversa por métodos numéricos.

```
function initNumIKin()
%%NUMERICAL INVERSE KINEMATICS
syms q1 q2 q3
global l0 l1 l2 l3 l4 l5 a1;
T01=denavit2(q1,l0+l1,0,0);
T12=denavit2(q2-pi/2,0,a1,-pi/2);
T23=denavit2(q3+pi/2,0,l2,0);
T34=denavit2(pi/2,0,l3,0);
H_B3=T01*T12*T23*T34;
global r_B3_inB J_B3_inB
r_B3_inB=H_B3*[0 0 0 1]';
r_B3_inB=r_B3_inB(1:3);
q=[q1 q2 q3];
J_B3_inB=jacobian(r_B3_inB,q);
end
```

Función encargada de cargar la interfaz gráfica.

```
function ScorBot_Vision_UI
global f
global p
global ax_Robot
global ax_Image2
global ax_Image1

%%Creacion de la ventana
f = figure;
f.Name='ScorBot Vision GUI';
f.NumberTitle='off';
f.ToolBar='none';
f.MenuBar='none';
f.Position=[35 150 840 720];
f.Resize='off';

%%Creacion de panel
p = uipanel(f,'Title','Robot Tool Position','Position',[.05 .75 .40 .20]);
text_HMT=uicontrol(p,'Style','text','String','Homogeneous Transform
Matrix','Units','normalized','Position',[0.00 0.80 0.6 0.15]);

%Creacion matriz
M=zeros(4,4);
plotMatrix(M)

%Creacion vector de angulos
V=zeros(1,5);
plotAngleVector(V)

%%Creacion de ejes
ax_Robot = axes('Parent',f,'Position',[.05 .05 .5 .6]);
ax_Image2=axes('Parent',f,'Position',[.60 .05 .35 .40]);
ax_Image1=axes('Parent',f,'Position',[.60 .55 .35 .40]);


%%Creacion del boton de Iniciar
global init_process;
init_process=0;

tb_Iniciar =
uicontrol('Style','togglebutton','Units','normalized','String','Iniciar');
tb_Iniciar.BackgroundColor=[0.5 0.5 0.5];
tb_Iniciar.Position=[0.05 0.6 .1 .1];
tb_Iniciar.FontSize=12;
tb_Iniciar.FontWeight='bold';
tb_Iniciar.Callback=@tb_IniciarCallback;

%%Creacion y display del robot
global l0 l1 l2 l3 l4 l5 a1 Scorbot;
L1=Link('d',l0+l1,'a',a1,'alpha',-pi/2);
L2=Link('offset',-pi/2,'d',0,'a',l2,'alpha',0);
L3=Link('offset',pi/2,'d',0,'a',l3,'alpha',0);
L4=Link('d',0,'a',l3,'alpha',-pi/2);
L5=Link('d',l4+l5,'a',0,'alpha',0);
Scorbot=SerialLink([L1 L2 L3 L4 L5],'name','ScorBot ER III');
showScorBot([0 0 0 0 0]);
movegui('center');
```

Función encargada de escribir los datos de la MTH en la interfaz gráfica.

```
function plotMatrix( M )
%PLOTMATRIX Cambia los valores de la matriz de la UI a los valores de la
%matriz M introducida.
global p;
left=0.05;
bottom=0.60;
n=1;
M=round(M,2);
for i=1:4
    for j=1:4
        text_Matrix(n) =
uicontrol(p,'Style','text','String',num2str(M(i,j)),'Units','normalized','Po
sition',[left bottom 0.10 0.15]);
        left=left+0.10;
        n=n+1;
    end
    left=0.05;
    bottom=bottom-0.15;
end
end
```

Función encargada de escribir los datos del vector de coordenadas angulares en la interfaz gráfica.

```
function plotAngleVector( V )
%PLOTANGLEVECTOR Cambia el vector de pos. angular a los valores
%introducidos en V.
global p;
left=0.75;
bottom=0.75;
V=round(V,2);
for i=1:5
    text_Q(i) =
uicontrol(p,'Style','text','String',strcat(strcat('q',num2str(i)),'
='),'Units','normalized','Position',[left-0.10 bottom 0.10 0.15]);
    text_Vector(i) =
uicontrol(p,'Style','text','String',num2str(V(i)),'Units','normalized','Posi
tion',[left bottom 0.10 0.15]);
    bottom=bottom-0.18;
end
end
```

Función usada para iniciar la conexión con la cámara web.

```
function initWCam

global cam;
cam=webcam(2);
end
```

Función encargada de representar en robot y sus movimientos en la interfaz gráfica.

```
function showScorBot(traj)

global ax_Robot Scorbot;
axes(ax_Robot)
Scorbot.plot(traj);
```

Función usada para mostrar imágenes de la cámara web en la interfaz gráfica.

```
function plotPreview()

global ax_Image2 cam;
tempImg = zeros(640,480,3);
hImage = image(tempImg, 'Parent', ax_Image2);
preview(cam, hImage);
end
```

Función para resetear la cuenta de los encoders.

```
function resetEncoder(num)
    global a;
    fwrite(a.aser,[48+11 num], 'uchar');
end
```

Función usada para poner en marcha o parar un motor concreto.

```
function motor( num, direccion, PWM )
    global a;
    global pinPWM;
    global pinDireccion;
    digitalWrite(a,pinDireccion(num),direccion);
    analogWrite(a,pinPWM(num),round(PWM*255));
end
```

Función para extraer el número de pulsos dados por un encoder.

```
function val=encoder(num)
    global a;
    fwrite(a.aser,[48+10 num], 'uchar');
    val =fscanf(a.aser,'%d');
    if (num==5)
        val=-val;
    end
end
```

Función para extraer el número de pulsos dados por un encoder.

```
function pulsados=readMS(num)
global a;
global pinMS;
pinMS=[14 15 16 17 18];
pulsados=~digitalRead(a,pinMS(num));
```

Función para parar todos los motores.

```
function paroMotores()
for i=1:5
    motor(i,0,0)
end
```

Función para calcular la cinemática inversa por medio de ecuaciones de la MTH.

```
function q=cinInv(T)
%%Acepta una matriz de trasformacion homogenea 4x4 y devuelve los valores
%de las cuaatro variables articulares.
if (isequal(size(T),[4 4]))

nx=T(1,1);ny=T(2,1);nz=T(3,1);ox=T(1,2);oy=T(2,2);oz=T(3,2);ax=T(1,3);ay=T(2
,3);az=T(3,3);px=T(1,4);py=T(2,4);pz=T(3,4);
elseif (isequal(size(T),[1 3]))
    px=T(1,1); py=T(1,2);pz=T(1,3);
else
    error('La variable introducida debe ser una matriz de transformacion
homogénea 5x5 o un vector de posición 1x3.')
end
global l0 l1 l2 l3 l4 l5 a1;
l0=21;
l1=14;
l2=22;
l3=22;
l4=8.2;
l5=6.5;
a1=2.5;

    q1=atan2(py,px);
A=pz+(l4+l5)-(l0+l1);
B=px*cos(q1)+py*sin(q1)-a1;
s3=(l2^2+l3^2-A^2-B^2)/(2*l2*l3);
c3=sqrt(1-s3^2);
    q3=atan2(s3,c3);

J=l0+l1-l4-l5-pz;
K=l2-l3*s3;
c2=(K*J-(c3*l3*px)/cos(q1)+c3*l3*a1)/(-l3^2*c3^2-K^2);
s2=real(sqrt(1-c2^2));
    q2=atan2(s2,c2);

    q4=-q2-q3;
 q=[q1 q2 q3 q4];
```

Función para mover todas las articulaciones del robot a determinados ángulos.

```
function moveAngle(rad)
%Acepta como entrada un vector de 5 angulos expresados en radianes. Se
%encarga de mover al brazo robotico a esos angulos.
%Ademas, tambien provoca que el robot virtual creado en la UI se mueva a la
%posicion deseada.
global USE_ARDUINO;
global posicion_angular_robot;
global relacion_transmision;
if USE_ARDUINO==1
    diferencia_angulos=rad-posicion_angular_robot;
    diferencia_angulos=angle2mangle(diferencia_angulos);
    pulsos=zeros(1,5);
    for i=1:3

pulsos(i)=round((6*relacion_transmision(i)*diferencia_angulos(i))/pi);
    end

pulsos(4)=round((6*relacion_transmision(4)*diferencia_angulos(4))/pi)+round(
(6*relacion_transmision(4)*diferencia_angulos(5))/pi);
    pulsos(5)=-
round((6*relacion_transmision(5)*diferencia_angulos(4))/pi)+round((6*relacio
n_transmision(5)*diferencia_angulos(5))/pi);
    pulsos(1)=-pulsos(1);
    pulsos(2)=-pulsos(2);
    pulsos(3)=+pulsos(3);
    pulsos(5)=-pulsos(5);
    pulsos_dados=zeros(1,5);
    for i=1:5
        resetEncoder(i);
        if(pulsos(i)>0)
            motor(i,1,0.5)
        else
            motor(i,0,0.5)
        end
    end
    while(~isequal(bsxfun(@ge,abs(pulsos_dados),abs(pulsos)),ones(1,5)))
        for i=1:5
            pulsos_dados(i)=encoder(i);
            if (abs(pulsos_dados(i))>=abs(pulsos(i)))
                motor(i,1,0)
            end
        end
    end
end
%%Movemos el robot virtual
traj=zeros(10,5);
for i=1:5
    traj(:,i)=linspace(posicion_angular_robot(i),rad(i),10);
end
showScorBot(traj);
posicion_angular_robot=rad;
%Actualizamos datos de UI
plotMatrix(cinDir(posicion_angular_robot));
plotAngleVector(posicion_angular_robot);
```

Función que convierte los ángulos dados por la cinemática inversa a ángulos reales que deben girar los motores.

```
function mtheta=angle2mangle(theta)
%Funcion que recibe los angulos que queremos que giren las articulaciones y
%los convierte en los angulos que deben girar los motores para ello.

if (size(theta,1)~=1)
    error('La variable introducida debe ser un vector 1x5')
end
if (size(theta,2)~=5)
    error('La variable introducida debe ser un vector 1x5')
end
mtheta(1)=theta(1);
mtheta(2)=theta(2);
mtheta(3)=theta(2)+theta(3);
mtheta(4)=theta(2)+theta(3)+theta(4);
mtheta(5)=theta(5);
```

Función que permite calcular la cinemática directa del brazo robot.

```
function T05=cinDir(q)
%%Acepta tres valores de los tres ejes del robot y devuelve su matriz de
%transformacion homogenea.
global l0 l1 l2 l3 l4 l5 a1;

q1=q(1,1);q2=q(1,2);q3=q(1,3);
q4=-q2-q3;
T01=denavit2(q1,l0+l1,0,0);
T12=denavit2(q2-pi/2,0,a1,-pi/2);
T23=denavit2(q3+pi/2,0,l2,0);
T34=denavit2(q4+pi/2,0,l3,0);
T45=denavit2(0,0,l4+l5,0);
T05=T01*T12*T23*T34*T45;
```

Función que permite calcular la posición previa a la requerida para coger un objeto.

```
function [ pos_acercamiento ] = calcPosAcercamiento( pos )
%CALC_POSACERCAMIENTO Calcula una posicion de acercamiento al objeto.
%Altura sobre la goma a la que se situara la pinza. En cm.
altura=10;
pos_acercamiento=[pos(1),pos(2),pos(3)+altura];
end
```

Función que permite abrir y cerrar la pinza del robot.

```
function pinza(str)
% Función para abrir o cerrar la pinza
global USE_ARDUINO
if USE_ARDUINO==1
    global a;
    % str: 'a' para abrir, 'c' para cerrar
    digitalWrite(a,51,1);
    % Se inicia el encoder 6 de la pinza a 0.
    resetEncoder(6);
    % Según la opción introducida se mueve el
    % motor en una dirección o en la otra.
    if str(1)=='a'
            motor(6,1,1)
    end
    if str(1)=='c'
            motor(6,0,1)
    end
     enc = encoder(6); % Se mide el contador del encoder
     pause(0.1); % Espera de 0.1 segundos
     enc_old = enc; % Se almacena la antigua lectura del encoder en enc_old
     enc = encoder(6); % Se vuelve a medir el encoder
     while enc >enc_old
        enc_old = enc;
        enc = encoder(6);
        pause(0.1);
     end
     motor(6,0,0);
    digitalWrite(a,51,0);
end
if str(1)=='a'
    fprintf('Pinza abierta\n');
end
if str(1)=='c'
    fprintf('Pinza cerrada\n');
end
end
```

Función usada para obtener la cinemática inversa a partir de métodos numéricos.

```
function qGoal=NumIKin(goal_pos,errorchoice)

global r_B3_inB J_B3_inB
syms q1 q2 q3
rGoal=goal_pos'+[0 0 14.2]';
q0=pi/180*[0 10 10]';
emax=1e-4;
max_iter=5e3; %En caso de que la posicion no se pueda conseguir
num_iter=0;
w=0.10; %Variable de peso
if nargin>=2 && strcmp(errorchoice,'errors')
    errors=ones(1,1);
end
while (num_iter<max_iter)&&(norm(rGoal-eval(subs(r_B3_inB,[q1 q2
q3],q0')))>emax)
    num_iter=num_iter+1;
    if nargin>=2 && strcmp(errorchoice,'errors')
```

```
        errors(num_iter)=norm(rGoal-eval(subs(r_B3_inB,[q1 q2 q3],q0')));
    end
    var_q=eval(subs(J_B3_inB,[q1 q2 q3],q0'))\(rGoal-eval(subs(r_B3_inB,[q1
q2 q3],q0')));
    q0=q0+w*var_q;
    %if (num_iter>=max_iter) error('Limite de iteraciones alcanzado.
Compruebe si la posicion requerida esta dentro del rango de alcance del
Scorbot.'); end
end
if nargin>=2 && strcmp(errorchoice,'errors')
    [~, c]=size(errors);
    t=1:c;
    plot(t,errors)
end
qGoal=q0;
qGoal=[qGoal;-qGoal(2)-qGoal(3)]';
end
```

ANEXO C: RESULTADOS

Se muestran a continuación los resultados obtenidos para la prueba de funcionamiento general de sistema *pick & place.*

Pruebas realizadas con gomas de color rojo y una orientación aproximada de 0º:

Color detectado	Orientación detectada	Recogido
R	7	S
R	7	S
R	11	N
R	11	N
R	-10	N
R	2	N
R	8	S
R	9	N
R	10	N
R	8	S

Pruebas realizadas con gomas de color blanco y una orientación aproximada de 0º:

Color detectado	Orientación detectada	Recogido
B	11	N
B	1	S
B	5	S
B	-5	S
B	7	N
B	2	S
B	8	N
B	-10	N
B	5	S
B	8	N

Pruebas realizadas con gomas de color rojo y una orientación aproximada de 45º:

Color detectado	Orientación detectada	Recogido
R	45	S
R	47	S
R	53	S
R	55	N
R	54	S
R	46	S
R	58	N
R	54	S
R	40	S
R	43	S

Pruebas realizadas con gomas de color blanco y una orientación aproximada de 45º:

Color detectado	Orientación detectada	Recogido
B	38	S
B	40	S
B	41	S
B	42	S
B	35	N
B	37	S
B	46	N
B	44	N
B	44	S
B	45	N

Pruebas realizadas con gomas de color rojo y una orientación aproximada de -45º:

Color detectado	Orientación detectada	Recogido
R	-39	S
R	-47	S
R	-38	S
R	-42	N
R	-40	N
R	-46	S
R	-50	S
R	-54	S
R	-35	N
R	-45	S

Pruebas realizadas con gomas de color blanco y una orientación aproximada de -45º:

Color detectado	Orientación detectada	Recogido
B	-41	S
B	-49	S
B	-42	S
B	-55	N
B	-50	S
B	-53	N
B	-43	N
B	-48	N
B	-45	S
B	-46	S

Pruebas realizadas con gomas de color rojo y una orientación aproximada de 90º:

Color detectado	Orientación detectada	Recogido
R	82	S
R	77	S
R	91	S
R	85	N
R	89	N
R	82	S
R	95	S
R	79	S
R	85	N
R	90	S

Pruebas realizadas con gomas de color blanco y una orientación aproximada de 90º:

Color detectado	Orientación detectada	Recogido
B	82	S
B	78	S
B	79	S
B	92	N
B	54	S
B	93	N
B	76	N
B	82	N
B	83	S
B	92	S

BIBLIOGRAFÍA

Bibliografía de referencia

1. Balaguer, C., Barrientos, A., Castellanos, José A., *El libro blanco de la robótica en España*, 2011

2. Barrientos, A., Peñin, L., Balaguer, C., Aracil, R., *Fundamentos de Robótica*, 2007.

3. Jiménez, R., *Adaptación de un robot SCORBOT-ER III para su control usando Arduino*, 2015.

4. Denavit, J., Hartenberg, R. S., *A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices*, 1965

5. P.I. Corke, "Robotics, Vision & Control"**,** Springer 2011, ISBN 978-3-642-20143-1.

Bibliografía de consulta

6. G.Shanmugasundar, R.Sivaramakrishnan, *Software Development for an Inverse Kinematics of Seven-Degrees of Freedom Newly Designed Articulated Inspection Robot,*, 2012.

7. SCORBOT-ER III User´s Manual. 6th Edition, 1999.

8. Vivek A Deshpande, Dr. P M George, *Analytical Solution for Inverse Kinematics of SCORBOT-ER Vplus Robot*, 2012.

9. Szeliski, R., *Computer Vision: Algorithms and Applications*, 2010.

10. de la Escalera, A., Pajares, A., Alegre, E., *Conceptos y Métodos en Visión por Computador*, 2016.

11. González, A., Martínez, F J., Pernía, A V., *Técnicas y algoritmos básicos de visión artificial*, 2006.

12. Siemens, *Manual del sistema de automatización S7-200*, 2008.

13. Echeverría, L., *Trabajando con PLCS: Manejo, Operación y Simulación de PLCS.*