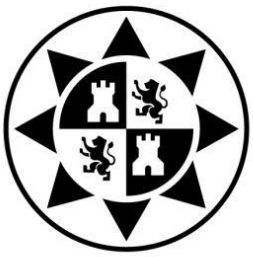


Universidad
Politécnica
de Cartagena

CONTROL VISUAL PARA EL BRAZO ROBOTNIK

Titulación:	I.T.I. esp. Electrónica Industrial.
Intensificación:	Automática.
Alumno:	Alejandro Sánchez Mur
Director:	José Luis Muñoz Lozano
Co-director:	Carlos Alberto Díaz Hernández

Cartagena, 27 de Junio de 2013

Índice general

Índice de figuras y tablas

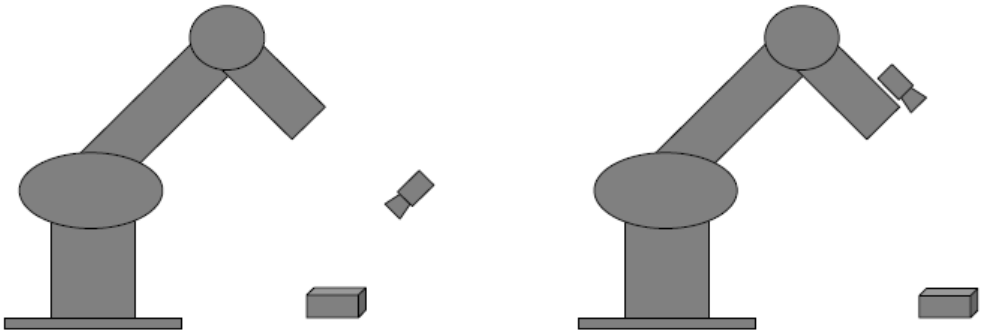
1.- Información acerca del proyecto

1.1-Introducción

La robótica, en la actualidad, ocupa un gran papel en el proceso de modernización e innovación de las industrias, ayudando a mejorar la calidad del producto y la productividad. Los manipuladores robóticos ayudan a sustituir al hombre en aquellos procesos que son repetitivos o que se desarrollan en ambientes peligrosos, o aquellos trabajos en los que el hombre no es capaz de realizar por su complejidad. Por ello, una tarea muy importante es la de diseñar el sistema de control encargado de controlar el robot para que realice la tarea solicitada con eficiencia y exactitud.

Entendemos por visual servoing (o control visual) al uso de visión por computador para el control del movimiento de un robot. Los sistemas de visual servoing se clasifican según tres criterios: la configuración física del sistema de visión, la utilización de las características extraídas de la imagen y la arquitectura del robot.

La configuración física define la relación cinemática del sistema de visión con respecto al robot y al entorno. Dependiendo de dónde se encuentre la cámara podemos distinguir dos disposiciones: cámara en un lugar fijo, independiente del movimiento del robot, (Eye-To-Hand) y perteneciente al cuerpo del robot, y por tanto moverse con él, (Eye-In-Hand).



(a) (b)

Fig. 1.1- *Disposición del sistema de visión: (a) Eye-To-Hand, (b) Eye-In-Hand. [1]*

Dependiendo de las características de la imagen podemos distinguir: control basado en posición (PBVS, control visual 3D), control basado en imagen (IBVS, control visual 2D) y enfoques avanzados como VS híbrido y VS particionado (control visual 2½D), aunque los más utilizados son los dos primeros.

Para el PBVS (Fig.1.2) se realiza una estimación de la posición para compararla con la posición deseada, a partir de las características extraídas de las imágenes.

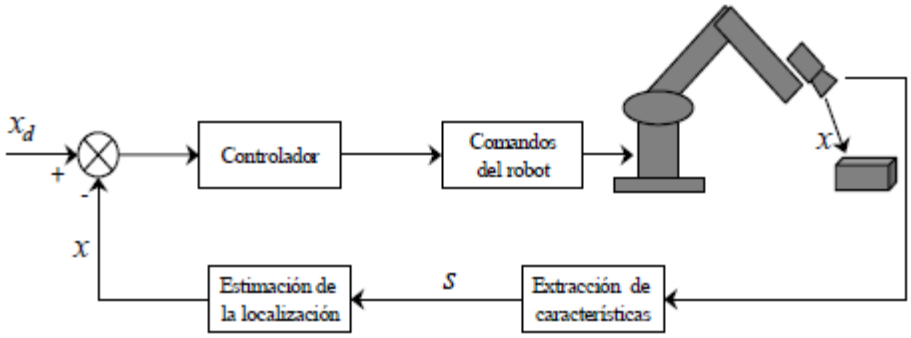


***Fig. 1.2**-Control visual basado en posición. [1]*

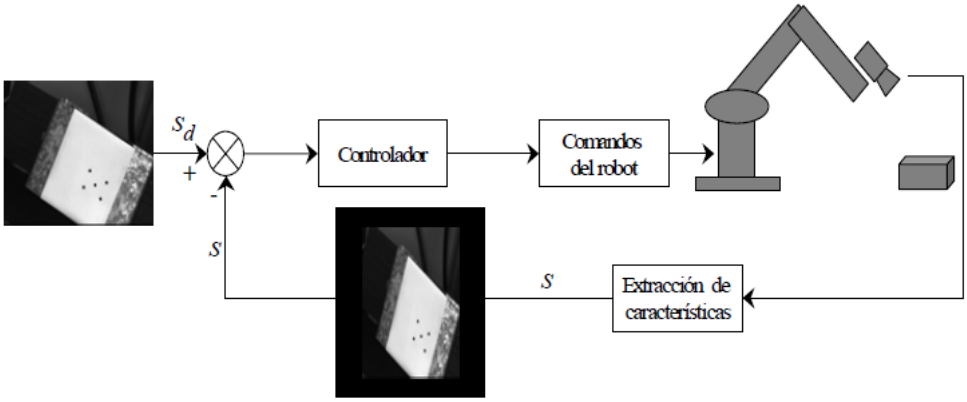


***Fig. 1.3**-Control visual basado en imagen. [1]*

Como se puede observar, el control visual basado en imagen (Fig.1.3) se realiza comparando la imagen obtenida con la imagen deseada (referencia). Para este control es importante el cálculo de la matriz de interacción o matriz jacobiana de la imagen, la cual establece la relación existente entre la imagen 3D real y la imagen 2D captada por la cámara.

A partir de la arquitectura del robot distinguimos dos esquemas de control: control visual indirecto y control visual directo.

Con la realización de este proyecto se pretende desarrollar un algoritmo para controlar un brazo robótico de seis grados de libertad por medio del control visual. El sensor visual se localizará en el extremo final del manipulador robótico (Eye-In-Hand) por lo que la relación entre el efector final del robot y la cámara es constante. Además esta disposición evita que se produzca el efecto conocido como oclusión, el cual se produce cuando el robot se posiciona entre el objeto y la cámara. El control realizado será IBVS, lo que simplifica la problemática al ser desarrollado en 2D.

A lo largo del proyecto se hará uso de técnicas de diferentes campos como procesamiento de imágenes, y teoría de control.

1.2.- Problemática

Se desea realizar un control visual del brazo robótico Robotnik. El control desarrollado tendrá que conseguir que el manipulador sea capaz de seguir la misma trayectoria que un objeto, basándose en la imagen captada por medio de una cámara. El proyecto se centrará en el IBVS (image-based visual servoing o control basado en imagen), por lo que el algoritmo creado deberá comparar las características de la imagen captada con la que sería nuestra imagen de referencia, comparando la posición del objeto con la posición deseada y generando un cambio de posición en el brazo robótico para lograr que el error existente entre ambas posiciones sea nulo.

1.3.-Justificación

En la actualidad, la importancia de los sistemas robotizados va en crecimiento, y, por tanto, el desarrollo de diversos sistemas de control para ellos. Siendo de gran utilidad e importancia en diversos campos: industria, automoción, medicina, militar... En lo que se refiere a control visual, en los últimos años podemos destacar, los avances en el campo de la medicina, en el que la precisión y exactitud es muy importante, el desarrollo de los vehículos no tripulados o en las cadenas de montaje industriales.

Este proyecto nos permitirá ampliar conocimientos en control y visión por computador además de aplicar un enfoque práctica a lo estudiado en asignaturas como “Visión artificial” y “Robótica”. Los avances logrados podrán servir de base para el desarrollo de sistemas para el control de calidad, para el seguimiento de blancos o para la realización de tareas en cadenas de montaje.

La implementación de sistemas de control visual, por ejemplo en una industria, permitiría agilizar el proceso, además de aumentar la productividad y de reducir costes.

Por todo ello es necesaria la realización del estudio, en base de su posible aplicación para la desarrollo de tareas similares.

1.4.-Objetivos

1-Desarrollo e implementación de un algoritmo que permita al brazo posicionarse respecto al objeto en el espacio de trabajo.

Este es el principal objetivo del proyecto. El algoritmo, desarrollado en C++, debe conseguir controlar el movimiento del brazo robótico para que sea capaz de seguir un blanco.

2-Creación de una interfaz gráfica para la supervisión de los parámetros característicos del robot y del sistema de control en tiempo real.

1.5.- Estructura de la memoria.

La memoria de este proyecto se ha decidido dividir en siete apartados claramente diferenciados, además de los apartados referentes a la bibliografía utilizada y a los anexos.

Así, en este primer capítulo se ha realizado una introducción al proyecto, explicándose el por qué la necesidad de su realización, los objetivos a lograr y la estructura seguida en el mismo.

A continuación, se presenta un capítulo dedicado a la robótica, en el que se desarrolla una descripción acerca de su historia.

En el tercer capítulo se da a conocer el hardware utilizado en el sistema. Conteniendo una descripción detallada del brazo robot, su funcionamiento y, además, información sobre la otra parte importante del proyecto, la cámara. También se realiza una breve descripción sobre el modelo pinhole de la cámara.

A lo largo del capítulo cuatro se realiza un estudio detallado de la cinemática del robot, abarcando desde la obtención de los parámetros de Denavit-Hartenberg y cálculo del jacobiano del robot hasta la resolución del problema de la cinemática directa e inversa.

En el siguiente capítulo se presenta el software utilizado, haciéndose una breve descripción acerca del lenguaje C y las librerías utilizadas para programar el robot: m5api y OpenCV. Además, en este capítulo, se detalla el trabajo realizado tanto en Windows XP, con el uso de la herramienta facilitada por Schunk para controlar los motores, como en Linux, con la instalación de las cámaras, OpenCV y el desarrollo y la ejecución de nuestro programa.

En el capítulo seis se comenta, brevemente, los tipos de comunicación brazo-PC posibles.

El capítulo siete es el que condensa la mayor parte del personal a lo largo del proyecto. Es el capítulo más importante debido a que en él se detalla todo el proceso del sistema de control. Abarca desde la adquisición de la imagen, con la explicación de su posterior tratamiento, hasta la explicación de la ley de control utilizada para el movimiento del robot. Además, en este capítulo se exponen dos ejemplos experimentales: control azimuth y control azimuth y elevación.

Por último, se expone una breve conclusión final, acerca del proyecto y el trabajo realizado, y los trabajos futuros que se pueden llevar a cabo.

Finalmente, se presentan como complemento a la memoria 3 anexos. En ellos se encuentran los códigos completos desarrollados en el proyecto, y el código utilizado para la calibración de la cámara.

2.- Estado actual de la robótica.

2.1.- Antecedentes.

La palabra “robot” proviene del vocablo checo “robotnik” que significa “trabajo forzado” o “servidumbre”.

Un robot es un dispositivo electrónico generalmente mecánico, que realiza tareas de forma automática, ya sea con o sin supervisión humana. Estas tareas sustituyen o extienden el trabajo humano en aquellas labores que pueden resultar repetitivas, peligrosas o difícil para el ser humano, como ensamblado en manufactureras, manipulación de objetos peligroso o pesados, etc.

Cabe destacar algunas definiciones de diferentes entidades.

Según la Asociación de Industrias Robóticas (RIA) un robot es un “manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas”.

Según la Organización Internacional de Estándares (ISO) que define al robot como: “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”.

2.2.- Historia de los robots.

Desde siempre, el ser humano ha diseñado máquinas a semejanza de sí mismo u otros seres vivos. Ya en el antiguo Egipto se unieron brazos mecánicos a las estatuas de sus dioses y los griegos construyeron estatuas con sistemas hidráulicos.

Pero no se considera hasta el siglo XVIII el inicio de la robótica actual, siglo en el que se produce un gran avance en la industria textil con la invención, por parte de Joseph Jacquard, de la primera máquina programable mediante tarjetas perforadas. Anteriormente, durante los siglos XVII y XVIII se habían diseñado algunos muñecos mecánicos con ciertas características de robots. Destacando los muñecos músicos diseñados por Jacques de Vaucansos de tamaño humano en el siglo XVIII y la muñeca mecánica capaz de realizar dibujos diseñada por Henri Maillardert en 1805.

Ya en 1954, el inventor Devol desarrolló un brazo primitivo que podía ser programado para realizar tareas específicas. En 1975, el ingeniero Victor Scheinman desarrolló un manipulador polivalente realmente flexible conocido como Brazo Manipulador Universal Programable, más conocido por sus siglas en inglés PUMA. Este

brazo era capaz de mover un objeto y colocarlo en cualquier orientación en un lugar deseado que estuviera a su alcance. La mayoría de los robots actuales se basan en el concepto básico multi-articulado del PUMA.

El desarrollo en la tecnología, donde se incluyen las computadoras electrónicas, actuadores de control retroalimentados, transmisión de potencia a través de engranes, y la tecnología en sensores han contribuido a flexibilizar los mecanismos autómatas para incorporarlos a tareas dentro de la industria.

Poco a poco la robótica va evolucionando hacia los sistemas móviles autónomos que son aquellos capaces de desenvolverse por sí mismos en entornos desconocidos sin necesidad de supervisión humana. Dentro de este tipo de robots podemos destacar varios: ELSIE, SCHACKEY y CART.

En 1953, el ELSIE era capaz de seguir una fuente de luz gracias a un sistema mecánico realimentado.

En 1968, el SCHACKEY del SRI (Standford Research Institute) estaba equipado con diversos sensores y de una cámara con lo que era capaz de desplazarse.

En los años setenta, la NASA en un programa de cooperación con el Jet Propulsion Laboratory comenzó a desarrollar plataformas para la exploración de terrenos desconocidos. De esta cooperación nació el MARS-ROVER, el cual estaba equipado con un brazo robótico STANFORD, un dispositivo telemétrico láser, cámaras estéreo y sensores de proximidad.

En la actualidad, podemos destacar algunos desarrollos como el IT, diseñado para expresar emociones, el CYPHER, un helicóptero militar o los robots mascotas de Sony.

En el campo de la robótica antropomorfa podemos destacar robots como el ASIMO de Honda que mide 1,30 m y pesa 49 kg, con 57° de libertad en los movimientos, posibilidad de andar por suelos irregulares o capaz de diferenciar hasta 3 voces distintas hablando simultáneamente.

2.2.1.- Clasificación.

Podemos clasificar los tipos de robots según diferentes criterios [2], como pueden ser:

1) Según la fuente de energía utilizada para generar su movimiento:

- Robots eléctricos: obtienen la energía de un generador de corriente eléctrica, ya sea continua o alterna.

- Robots neumáticos: generan su movimiento a partir del uso de aire comprimido a través de válvulas.

- Robots hidráulicos: son movidos gracias a la energía de un fluido en estado líquido, generalmente aceites.

2) Según su cronología:

- 1ª Generación:

  Manipuladores. Son sistemas mecánicos multifuncionales con un sistema de control, bien manual, de secuencia fija o de secuencia variable.

- 2ª Generación:

  Robots de aprendizaje. Repiten una secuencia de movimientos que ha sido ejecutada previamente por un humano.

- 3ª Generación:

  Robots con control sensorizado. La computadora ejecuta las órdenes de un programa y las envía al manipulador para que realice los movimientos necesarios.

- 4ª Generación:

  Robots inteligentes. Similares a los anteriores, pero, además, disponen de sensores que envían información a la computadora de control. Esto permite la toma de decisiones y control del proceso en tiempo real.

3) Según su arquitectura:

- Poliarticulados:

  En este grupo encontramos principalmente robots fijos diseñados para realizar una tarea dentro de una industria, como manipuladores.

Fig. 2.1- *Robot poliarticulado de la marca Kuka.*

- Móviles:

  Son robots capaces de moverse por el entorno gracias a un sistema locomotor, además de guiarse por él o bien por telemando o con la información que reciben de sus sensores. Suelen utilizarse para el transporte de piezas dentro de una cadena de fabricación.

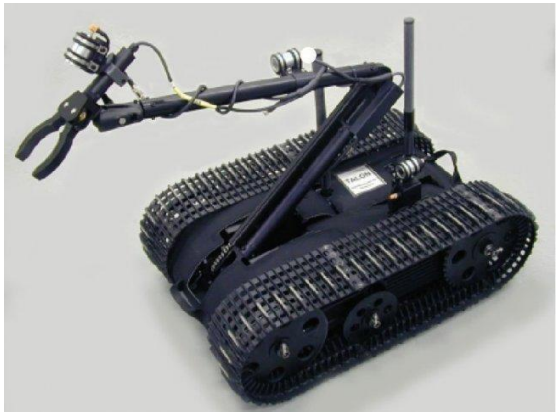


Fig. 2.2- *Robot móvil usado para la desactivación de minas.*

- Androides:

  Son robots con los que se intenta reproducir la forma y el comportamiento del ser humano. Están destinados principalmente al estudio y a la experimentación, aunque también existen algunos desarrollados como robots de servicio.

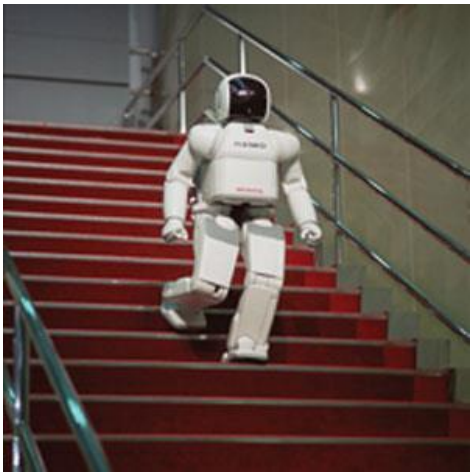


Fig. 2.3*- Robot androide ASIMO de la empresa Honda.*

- Zoomórficos:

  Son robots que pretenden imitar a los seres vivos, excepto a los humanos. Dentro de este grupo podemos diferenciar dos tipos de robots: caminadores y no caminadores. Dentro del grupo de los no caminadores se pueden destacar los diseñados con segmentos cilíndricos que pretenden emular el movimiento de una serpiente o un gusano. Los caminadores, por el contrario, están más desarrollados, ya que permiten, entre otras cosas, moverse por superficies muy accidentadas.

Fig. 2.4*- Robot con la forma y movimientos de un gusano.*

4) Según el modo en que realizan sus movimientos:

- Control en lazo abierto:

  El programa que controla el movimiento de los componentes del robot se realiza en un posicionamiento punto a punto en el espacio.

- Control en lazo cerrado:

  Este tipo de control permite dos formas de trabajo:

  - Gobierno por configuración, donde el control de los movimientos se hace en función de los ejes de sus elementos.
  - Gobierno por sensor: los movimientos se establecen en función de la respuesta que van proporcionando diferentes sensores.

3.- Hardware.

3.1.- Brazo modular.

3.1.1.- Características.

Fig. 3.1*- Brazo robot de Robotnik. [3]*

Se trata de un brazo robot, diseñado por la empresa Robotnik Automation SSL., formado a partir de seis servoaccionamientos modulares PowerCube de Schunk. Estos servos no requieren de armario control externo ya que cada uno de ellos incluye motorreductor, etapa de potencia y controlador. Esto, además, permite que en caso de avería, por ejemplo, del controlador solo sea necesario cambiar el modulo rotatorio y no el armario. La conexión externa consta de sólo 4 hilos, 2 para la alimentación y otros 2 hilos para la comunicación CAN bus.

Entre sus características generales tenemos:

- Hasta 7 grados de libertad (en un mismo bus).

- Alcance de 400 a 1300 mm.

- Capacidad de carga útil en función del diseño (9Kg en la base del efector final con el alcance máximo).

- Peso reducido: aproximadamente 25Kg en una configuración de 6GDL.

- Velocidad máxima de 57 grados/s en la base y 360 grados/s en la muñeca.

- Motores servo de corriente alterna y frenos electromagnéticos.

- Repetitividad posicional de 0.5 mm.

- Electrónica de control mediante tarjeta CAN-PCI desde ordenador personal.

Entre las ventajas más destacadas del brazo robot son:

- Alimentación a 24 VDC.
- Control a través de un bus CAN. No requiere tener un armario de control, aunque sí un ordenador con una tarjeta de comunicaciones CAN.
- Configurable. Su configuración modular hace posible modificar las dimensiones de los enlaces, así como el número de grados de libertad.

Por todo ello este tipo de brazo es el más adecuado para su instalación sobre plataformas móviles.

***Fig. 3.2**- Plataforma móvil con un brazo Robotnik montado.*

3.1.2.- Módulos y enlaces

Los módulos con los que está equipado el brazo robot son módulos rotatorios de la familia PRL diseñados por la empresa Schunk.

***Fig. 3.3**-Módulo giratorio PRL de Schunk.*

Estos módulos trabajan como controladores distribuidos. El controlador maestro, PC, es el encargado de crear y enviar las referencias a cada uno de los ejes del sistema de articulaciones.

Además, el control de corriente, velocidad y posición tiene lugar en cada uno de los módulos, así como las operaciones de supervisión y control de parada.

Entre las características de estos módulos tenemos:

- Cada eje contiene su propio controlador y servo amplificador.

- Motores sin escobillas (brushless).

- Eje pasante que permite el paso interior de los cables (permite un diseño con protección IP).

- Encoder absoluto para posicionamiento y control de velocidad.

- Monitorización de rango, limites, temperatura y corriente.

- Requerimientos de potencia de 20A y 24VDC (en total 480W para todo el brazo).

- Integración inmediata con cualquier tipo de efector final: pinzas, manos robotizadas, taladros, útiles de soldadura, etc.

- Freno magnético integrado en todos los ejes.

- Arquitectura abierta (control de alto nivel sobre el movimiento de cada eje).

Como elementos de unión entre los diferentes rotores tenemos las siguientes piezas:

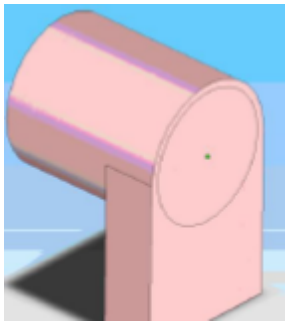


Fig. 3.4- *Elemento 1.*

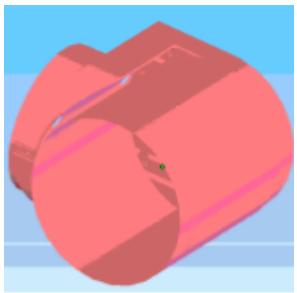


Fig. 3.5- *Elemento 2.*

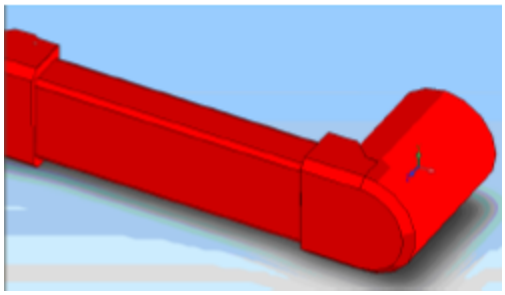


Fig. 3.6- *Elemento 3.*

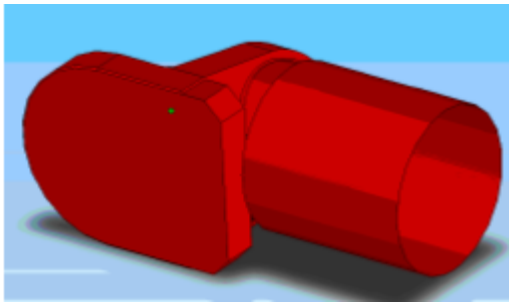


Fig. 3.7- *Elemento 4 y 5.*

En el caso del eslabón para las uniones de los pares de rotores 3-4, 4-5 y 5-6 la única diferencia existente entre ellos es el tamaño de los puntos de unión con los rotores ya que estos son de distintos tamaños.

Además de los eslabones propios del robot, se han añadido 2 piezas más para colocar la cámara.

3.1.3.- Alcances y limitaciones.

Como ya se ha comentado el robot presenta seis grados de libertad rotatorios, lo que nos permite que el campo de trabajo amplio.

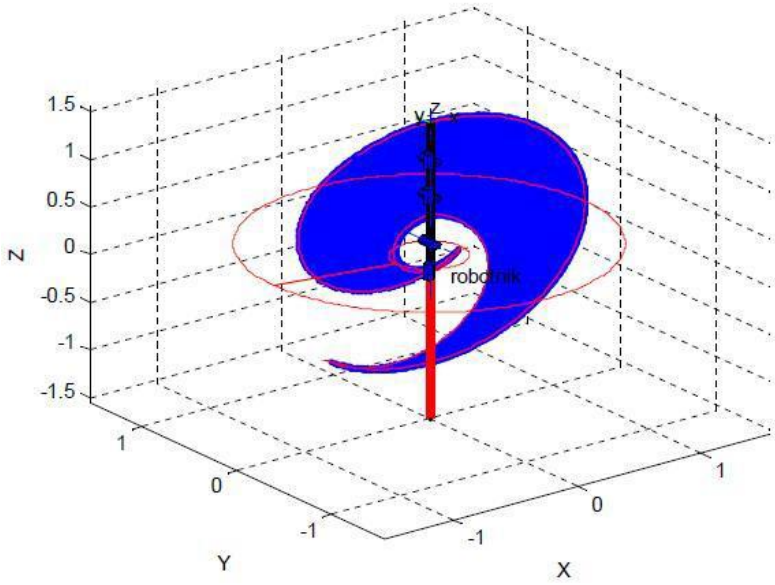


Fig. 3.8.- *Espacio de trabajo del robot.*

A continuación se muestra una tabla con los valores mínimos y máximos de giro para cada eje:

Junta	Mín	Máx
q1	-180º	180º
q2	-126º	126º
q3	-100º	100º
q4	-180º	180º
q5	-90º	90º
q6	-180º	180º

Tabla 3.9.- *Límites establecidos para los eslabones del robot.*

El hecho de limitar el giro de algunos ejes es para evitar posibles colisiones entre sus componentes y para reducir el número de soluciones cinemáticas redundantes.

Antes de realizar cualquier cálculo o prueba en el robot debemos saber que existen diferentes configuraciones que nos pueden influir en los resultados:

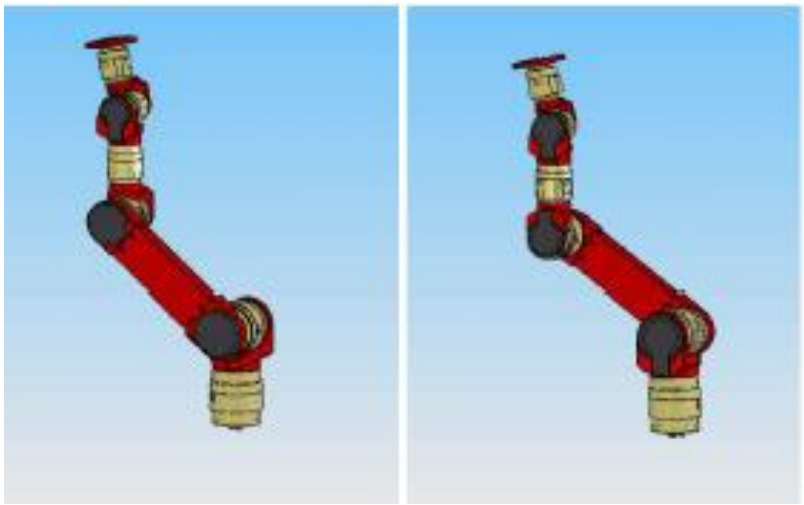


Fig. 3.10.- *Diferencia entre configuración: hombro izquierdo y hombro derecho.*

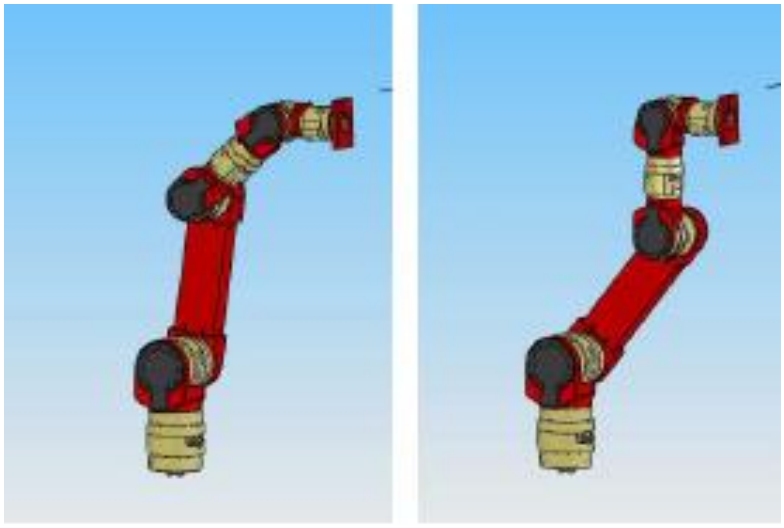


Fig. 3.11.- *Diferencia entre configuración: codo arriba y codo abajo.*

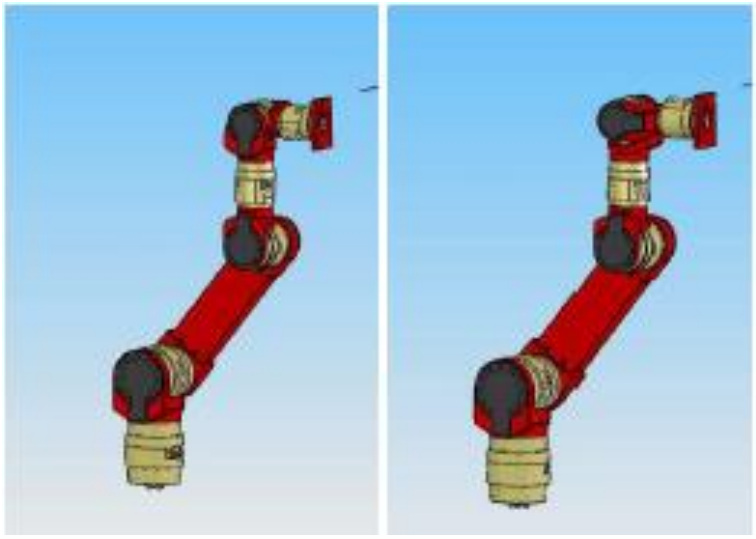


Fig. 3.12.- *Diferencia entre configuración: muñeca girada y muñeca no girada.*

Estas tres diferentes configuraciones nos permiten 8 combinaciones posibles para llegar a la misma posición y orientación final, por lo que será necesario conocer la configuración para seleccionar una única solución.

3.1.4.- Sistema eléctrico.

En la siguiente figura se muestra una visión general del sistema eléctrico del brazo robot.

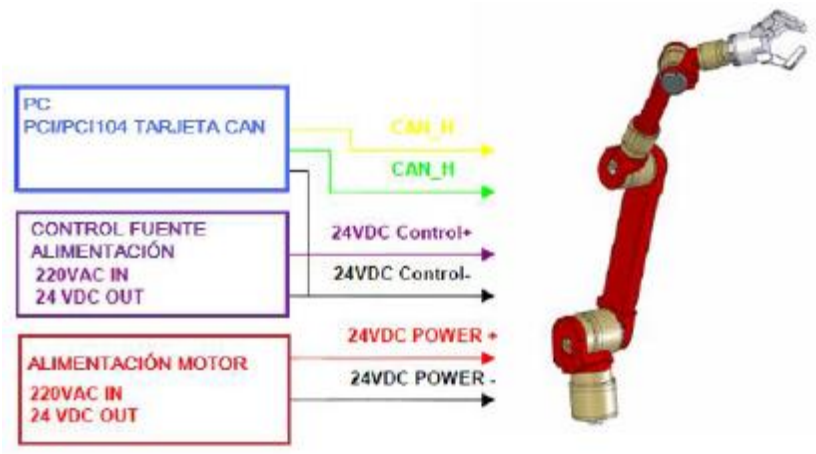


***Fig. 3.13.**- Principales componentes del sistema eléctrico.*

3.1.5.- Fuente de alimentación principal.

El brazo robot necesita una alimentación de 24 Vdc y 20A. Para evitar posibles ruidos en la alimentación, ésta puede estar repartida en dos suministradores independientes, uno para el control y otro para la alimentación del motor. El BUS CAN de masa debe estar conectado junto la masa de control.

El brazo robot puede estar alimentado por baterías. Para cargarlas, es recomendable desconectarlas del robot y después conectarlas al cargador. Nunca conectar el cargador al robot, ya que el equipo electrónico puede sufrir daños.

Si el robot no va a ser usado durante un largo período de tiempo, conviene desconectarlo.

3.1.6.- Secuencia de inicio del robot.

Para iniciar el brazo robot siempre se seguirán los siguientes pasos:

1. Conectar la fuente de alimentación.
2. Asegurarse de que el botón de parada de emergencia no está pulsado.
3. Apretar el botón de reinicio.
4. En este momento, se deberá encender el PC controlador. El PC se pone en marcha y LinuxRT OS se inicia. El sistema está configurado para comenzar el control de los componentes de forma automática.
5. Durante el inicio, el sistema operativo intentará obtener la dirección IP desde el router por DHCP (Protocolo de configuración de host dinámico). Si el router no está conectado, el sistema está configurado para reintentarlo, de modo que el inicio se retrasará más de un minuto.

No existe la posibilidad de conectarse al sistema de forma remota o local y empezar a trabajar con él.

3.1.7.- Botón de paro de emergencia.

El robot modular consta de un botón para paradas de emergencia que corta la alimentación de todos los módulos para evitar daños en los actuadores o en el entorno del robot.

El botón de parada de emergencia también debe ser accionado cuando se va a realizar cualquier tipo de trabajo con el robot, como cambiar el actuador del final del brazo, agregar nuevos componentes, en resumen, cualquier tarea durante la cual un movimiento inesperado del robot pueda resultar peligroso.

Para salir del estado de parada de emergencia, se deberán seguir los siguientes pasos:

1. Comprobar que la situación de peligro que causó la parada ha finalizado.
2. Tirar del botón de parada de emergencia.
3. Pulsar el botón Reinicio del panel de control.

3.2.- Cámara.

Durante el proyecto se ha hecho uso de dos cámaras distintas: una cámara industrial Ueye y una cámara de Logitech Orbit.

Se pretendía utilizar en el proyecto sólo la primera cámara pero debido a un problema, el cual no se pudo resolver, la cámara no podía ser utilizada al mismo tiempo que el brazo robot.

3.2.1.- Características.

Fig. 3.14- *Cámara UEye.*

Fig. 3.15- *Cámara Orbit.*

Cámara UEye:

Esta cámara diseñada y fabricada por la empresa alemana IDS IMAGING DEVELOPMENT SYSTEMS. El modelo de la cámara es UI-1540SE-M, la cual viene en un encapsulado metálico con agujeros de montaje M3 lo cual la hace ideal para su uso en sistemas automáticos e inspección de calidad.

Sus principales características son:

- **Interfaz:** USB 2.0
- **Tecnología del sensor:** CMOS (Aptina)
- **Tipo de imagen:** Monocromática
- **Resolución:** 1280 x 1024
- **Profundidad de resolución:** 8bit (10bit ADC)
- **Categoría de resolución**: 1.3 Megapixel
- **Tamaño del sensor:** 1/2”
- **Obturador:** Rodante
- **Max. Fps in modo freerun:** 25 fps
- **Tiempo de exposición en modo freerun:** 37µs - 983ms
- **Tiempo de exposición en modo trigger:** 37µs - 983ms
- **Modelo del sensor:** MT9M001STM
- **Pixelpitch en µm:** 5.20
- **Tamaño óptico:** 5.656 x 5.325 mm
- **Tipo de protección:** IP30
- **Dimensiones:** Alto: 34.00 mm, Ancho: 32.00 mm, Largo: 27.40 mm
- **Peso:** 65.00 g
- **Alimentación:** USB
- **Característica adicional:** incluye librerías propias para C/C++, C#, Microsoft .NET y Visual Basic.

Cámara Orbit Sphere MP:

Esta cámara está diseñada y fabricada por la empresa Logitech. De esta cámara destaca que tanto su base como el sensor están motorizados, permitiendo movimientos pan-til, aunque durante este proyecto no ha sido usada esta característica de la cámara.

Sus principales características son:

- **Interfaz:** USB 2.0
- **Tecnología del sensor:** CMOS
- **Tipo de imagen:** Color

- **Resolución:** 1280x960
- **Profundidad de resolución:** 32 bit
- **Categoría de resolución**: 1.3 Megapixel
- **Tamaño del sensor:** 1/2”
- **Max. Fps en resolución 640x480:** 30 fps
- **Distancia focal:** 3.85 mm
- **Dimensiones:** Alto: 109 mm, Ancho: 84 mm, Largo: 84 mm
- **Alimentación:** USB
- **Característica adicional:** sensor y base motorizados.

3.3.4.- Modelo de la cámara.

La calibración de la cámara consiste en la estimación de un modelo para una cámara no calibrada. Su objetivo es encontrar los parámetros externos (posición y orientación relativa a un sistema mundial coordinado), y a los parámetros intrínsecos de la cámara (centro de la imagen, longitud focal y coeficientes de distorsión). Para esta calibración una de las técnicas más utilizadas es la propuesta por Tsai en 1986. Su implementación requiere un punto correspondiente 3D en 2D pixeles de la imagen. Se utiliza una técnica de dos etapas: primero, la posición y orientación y, segundo, los parámetros internos de la cámara.

El modelo de Tsai [4] se basa en un modelo de proyección perspectiva de *pinhole* o agujero de alfiler y los siguientes once parámetros son a estimar:

f - Longitud focal de la cámara,

k - coeficiente de distorsión radial de la lente,

C_x, C_y - Coordenadas del centro de la distorsión de la lente radial,

S_x - factor de escala para tener en cuenta cualquier incertidumbre debido a imperfecciones en el tiempo hardware para el escaneo y digitalización,

R_x, R_y, R_z - Ángulos de rotación para la transformación entre el mundo y la cámara de coordenadas,

T_x,T_y, T_z- Componentes de traducción para la transformación entre el mundo y la cámara coordenadas.

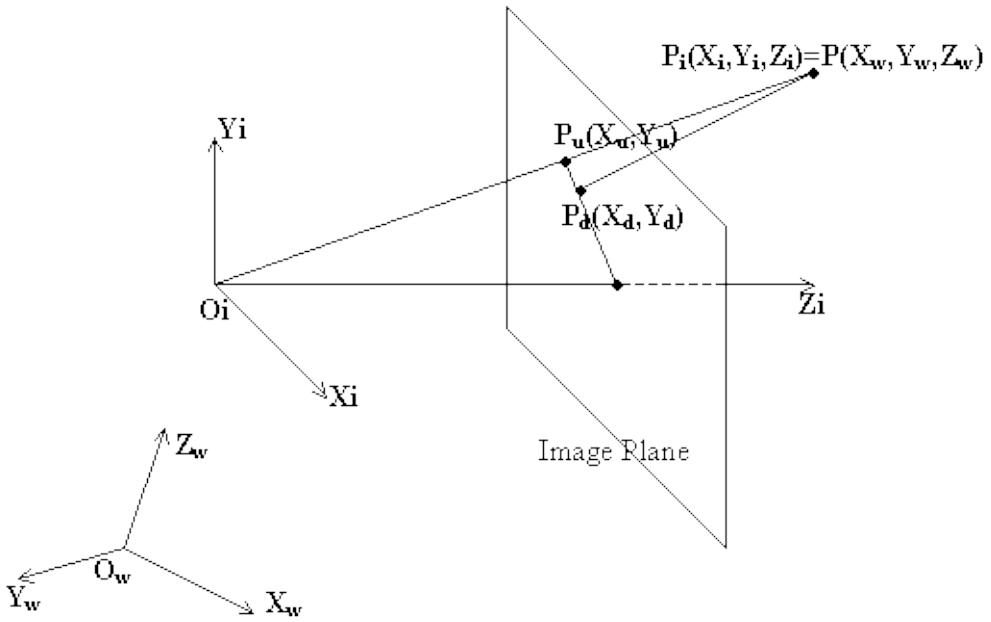


Fig. 3.16.- *Re-proyección del modelo con proyección de perspectiva y distorsión radial.*

La transformación del mundo (X_w,Y_w,Z_w) a la imagen (X_i,Y_i,Z_i) coordenadas considera los parámetros extrínsecos de la cámara (Traslación T y rotación R) con la ecuación:

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \mathrm{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathrm{T}$$

Donde R y T caracterizan la transformación 3D del mundo al sistema de coordenadas de la cámara y son definidos como:

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \quad T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

Con:

$$r_1 = \cos(R_y)\cos(R_z)$$

$$r_2 = \cos(R_z)\sin(R_x) - \cos(R_x)\cos(R_z)$$

$$r_3 = \sin(R_x)\sin(R_z) + \cos(R_x)\sin(R_z)$$

$$r_4 = \cos(R_y)\sin(R_z)$$

$$r_5 = \sin(R_x)\sin(R_y)\sin(R_z) + \cos(R_x)\cos(R_z)$$

$$r_6=\cos(R_x)\sin(R_y)\sin(R_z)-\cos(R_z)\sin(R_x)$$

$$r_7=-\sin(R_y)$$

$$r_8=\cos(R_y)\sin(R_x)$$

$$r_9=\cos(R_x)\cos(R_y)$$

(R_x,R_y,R_z) los ángulos de Euler de rotación alrededor de los tres ejes.

(T_x, T_y, T_z) los parámetros de conversión 3D de mundo a la imagen de coordenadas.

La transformación de la posición 3D (en la imagen de coordenadas) para el plano de la imagen se calcula siguiendo los siguientes pasos:

Transformación del mundo en 3D coordenadas (X_i, Y_i) al plano de la imagen sin distorsiones (X_u, Y_u):

$$X_u=f\frac{X_i}{Z_i}$$

$$Y_u=f\frac{Y_i}{Z_i}$$

Transformación de distorsión (X_u, Y_u) a distorsionada (X_d,Y_d):

$$X_u=X_d(1+kr^2)$$

$$Y_u=Y_d(1+kr^2)$$

Dónde:

$r=\sqrt{X_d^2+Y_d^2}$, y k es el coeficiente de distorsión de la lente.

Transformación de coordenadas del plano de la imagen distorsionada (X_d,Y_d) a la imagen final (X_f, Y_f) son:

$$X_f=\frac{S_x X_d}{d_x}+C_x$$

$$Y_f=\frac{Y_d}{d_y}+C_y$$

Con:

(d_x, d_y): Distancia entre los elementos sensores adyacentes en la dirección X e Y. d_x, d_y son parámetros fijos de la cámara. Ellos dependen del tamaño CCD y la resolución de la imagen (X_f, Y_f) son la posición final del píxel de la cámara.

4.- Estudio cinemático del robot.

4.1.- Grado de libertad.

Como ya se ha comentado el robot está formado por una serie de eslabones unidos mediante unos módulos giratorios que permiten un movimiento relativo entre cada dos eslabones consecutivos. La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la anatomía del brazo humano, por lo que en ocasiones, para hacer referencia a los distintos elementos que componen el robot, se usan términos como cuerpo, brazo, codo y muñeca.

El movimiento de cada articulación en un robot puede ser de desplazamiento, de giro o de una combinación de ambos. Cada uno de estos movimientos independientes que puede realizar cada articulación con respecto a la anterior, se denomina grado de libertad.

El número de grados de libertad del robot viene dado por la suma de los grados de libertad de las articulaciones que lo componen. Para nuestro brazo robot, como ya se ha indicado, disponemos de seis grados de libertad, ya que las articulaciones empleadas son únicamente las de rotación con un solo con grado de libertad cada una, por lo que el número de grados de libertad coincide con el número de articulaciones o enlaces existentes.

4.2.- Parámetros de Denavit-Hartenberg.

El brazo robot utilizado presenta una configuración de 6 GDL, RRR (Rodar-Rodar-Rodar) para posición y una muñeca RPR (Rodar-Paso-Rodar).

Este procedimiento permite describir una estructura cinemática de una cadena articulada constituida por articulaciones de un único grado de libertad.

Para su definición un movimiento rígido necesita seis coordenadas, tres que determinen la posición y tres que establezcan la orientación. Así un movimiento relativo entre n cuerpos requiere 6^ncoordenadas.

Con el uso de los parámetros de Denavit-Hartenberg podemos reducir el número de coordenadas independientes a cuatro. Para ello es necesario que se cumplan las siguientes condiciones para situar los sistemas de referencia locales:

1. El eje X_{i+1} debe de ser perpendicular al eje Z_i.
2. El eje X_{i+1} debe cortar al eje Z_i en un punto.

Debemos tener en cuenta los siguientes comentarios:

- La configuración del robot puede ser cualquiera, aunque es aconsejable utilizar una posición sencilla de analizar.
- Se enumeran los eslabones, asignando el 0 a la base y n-1 para el último eslabón, siendo n el número de grados de libertad del robot.
- El sistema de coordenadas ortonormal dextrógiro de la base (x_0, y_0, z_0) se establece con el eje z_0 localizado a lo largo del eje de movimiento de la articulación 1 y apuntando hacia fuera del hombro del brazo del robot.
- El sistema de referencia de cada eslabón se coloca al final del mismo, en el extremo de la articulación a la cual está conectado el eslabón siguiente.
- El ángulo o desplazamiento de cada eslabón siempre se mide tomando como base el sistema de referencia del eslabón anterior.

Cada sistema de coordenadas se estable según las siguientes reglas:

d_i: es la distancia de X_{i-1} a X_i, medida a lo largo de Z_{i-1} hasta la intersección del eje Z_{i-1} con el eje X_i:

$$d=dist(O_{i-1}, X_i \cap Z_{i-1})$$

a_i: es la distancia de Z_i a Z_{i-1}, medida a lo largo de X_i hasta la intersección con el eje Z_{i-1}:

$$a=dist(Z_i, Z_{i-1} \cap Z_{i-1})$$

θ_i: es el ángulo entre el eje X_{i-1} y el eje X_i, medido respecto al eje Z_{i-1}. Su signo viene determinado por la regla de la mano derecha (rmd).

$\propto_i$: es el ángulo entre el eje Z_i y el eje Z_{i-1}, medido respecto al eje X_i. Su signo lo determina la rmd.

Con la ayuda de la herramienta Robotics Toolbox para MatLab, podemos tomar como posición inicial del robot la que se muestra en la siguiente figura:

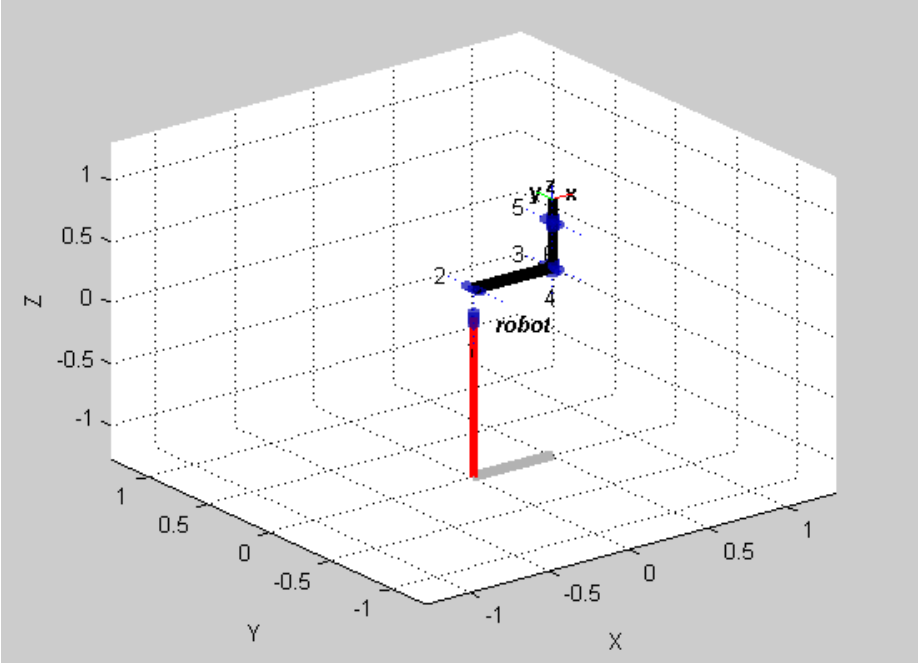


Fig. 4.1.- *Posición 1 vista con la herramienta Robotics Toolbox para MatLab.*

	d	**a**	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$
Eslabón 1	l_1	0	θ_1	$\pi/2$
Eslabón 2	0	l_2	θ_2	0
Eslabón 3	0	0	θ_3	-$\pi/2$
Eslabón 4	l_4	0	θ_4	$\pi/2$
Eslabón 5	0	0	θ_5	-$\pi/2$
Eslabón 6	l_6	0	θ_6	0

Tabla 4.2.- *Parámetros DH para la posición 1.*

Siendo las incógnitas de la tabla los valores correspondientes con las medidas de las articulaciones del robot.

l_1=0.2411;
l_2=0.495;
l_4=0.372;
l_6=0.1832;

Dependiendo de la posición inicial que tomemos del robot los parámetros Denavit-Hartenberg presentarán determinadas diferencias. Así para la posición en la que el robot está en posición completamente vertical:

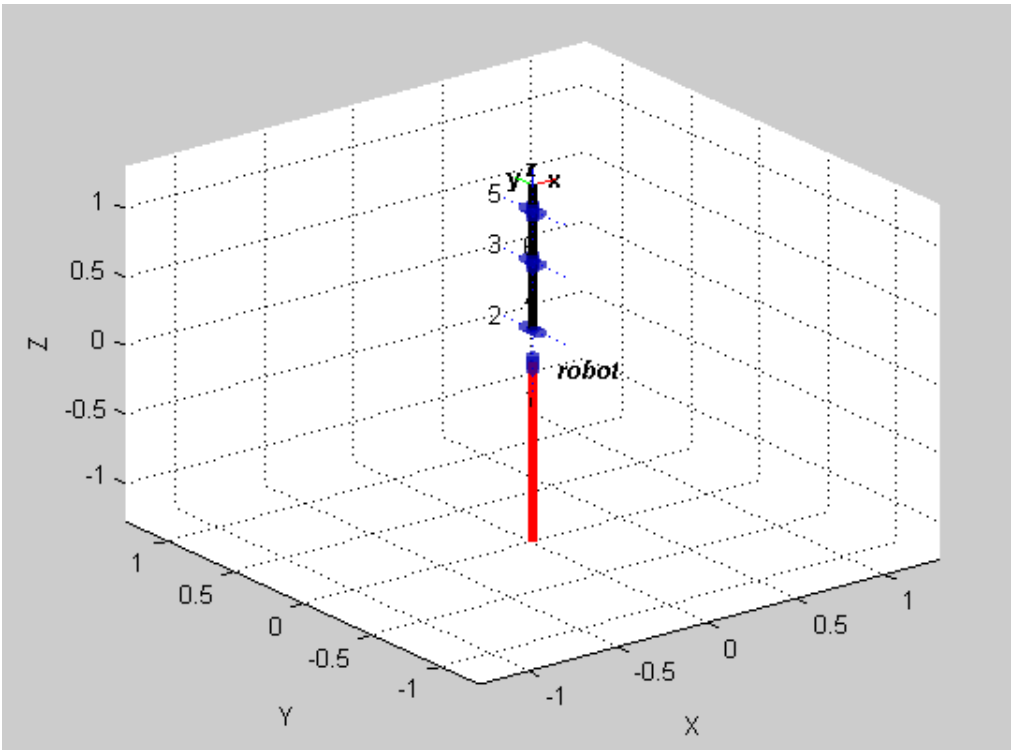


Fig. 4.3.- *Posición 2 vista con la herramienta Robotics Toolbox para MatLab.*

	d	**a**	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$
Eslabón 1	l_1	0	0	$\pi/2$
Eslabón 2	0	l_2	0	0
Eslabón 3	0	0	0	-$\pi/2$
Eslabón 4	l_4	0	0	$\pi/2$
Eslabón 5	0	0	0	-$\pi/2$
Eslabón 6	l_6	0	0	0

Tabla 4.4.- *Parámetros DH para la posición 2.*

4.3.- Estudio de la cinemática del robot.

La cinemática del robot estudia el movimiento del robot respecto a un sistema de referencia. La cinemática describe de forma analítica el movimiento del robot como una función del tiempo y de la posición y orientación del extremo final del brazo con respecto a los valores que toman sus articulaciones en coordenadas.

Podemos distinguir principalmente entre dos problemas a resolver para su estudio:

1. **Cinemática directa:** consiste en determinar la posición y orientación de la herramienta del brazo robot, o extremo final, con respecto al sistema de referencia inicial a partir de conocer los ángulos de cada una de las articulaciones.

2. **Cinemática inversa:** consiste en determinar los ángulos que deben de adoptar las articulaciones para lograr que el robot se coloque en una posición y orientación determinada. Este problema no presenta una solución única.

4.3.1.- Resolución del problema de cinemática directo.

Para resolverlo se utiliza principalmente el álgebra vectorial y matricial para representar y describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo.

En este caso vamos a resolver el problema haciendo uso de dos métodos:

- Método de matrices de transformación homogénea.
- Método mediante algoritmo de Denavit-Hartenberg.

4.3.1.1.- Resolución mediante algoritmo Denavit-Hartenberg.

El método de Denavit-Hartenberg permite definir una matriz de transformación homogénea con 4 grados de libertad.

Esas 4 transformaciones consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i-1 con el elemento i:

1. Rotación alrededor del eje Z_{i-1} un ángulo θ_i.
2. Traslación a lo largo del eje Z_{i-1} una distancia d_i; vector $d_i(0,0,d_i)$.
3. Traslación a lo largo del eje X_i una distancia a_i; vector $a_i\ (a_i,0,0)$.
4. Rotación alrededor del eje X_i un ángulo α_i.

Estas transformaciones se han de realizar en el orden indicado, ya que el producto matricial no es conmutativo, así tenemos que:

$$^{i-1}A_i = Rotz(\theta_i)\cdot T(0,0,d_i)\cdot T(a_i,0,0)\cdot Rotx(\alpha_i)$$

Realizando el producto obtenemos:

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

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

Siendo $\theta_i, \alpha_i, d_i, a_i$ los parámetros D-H del eslabón i y $C\ y\ S$, coseno y seno, respectivamente.

Así podemos determinar de manera sencilla las matrices de transformación para cada eslabón. Para ello es necesario que los sistemas de rotación hayan sido definidos conforme a las convenciones Denavit-Hartenberg.

Para obtener las matrices de transformación, sólo debemos sustituir las incógnitas con los valores de la siguiente tabla:

	d	**A**	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$
Eslabón 1	l_1	0	0	$\pi/2$
Eslabón 2	0	l_2	0	0
Eslabón 3	0	0	0	-$\pi/2$
Eslabón 4	l_4	0	0	$\pi/2$
Eslabón 5	0	0	0	-$\pi/2$
Eslabón 6	l_6	0	0	0

Tabla 4.4.- *Parámetros DH para la posición 2.*

Obteniéndose así las siguientes matrices:

$$^{0}A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{1}A_2 = \begin{bmatrix} 1 & 0 & 0 & l_2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{2}A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{3}A_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^{4}A_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad ^{5}A_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El resultado de la matriz de transformación T que nos relaciona el sistema fijo de la base con el extremo final del robot será:

$$T = {}^{0}A_6 = {}^{0}A_1 \cdot {}^{1}A_2 \cdot {}^{2}A_3 \cdot {}^{3}A_4 \cdot {}^{4}A_5 \cdot {}^{5}A_6$$

4.4.- Cálculo del jacobiano del robot.

El modelo cinemático de un robot busca relacionar las velocidades articulares y su posición con la velocidad y posición del extremo final. Así, el sistema de control del robot debe de calcular las velocidades que hay que imprimir a cada articulación para conseguir una trayectoria temporal. Para ello es de gran utilidad el conocer la relación entre los vectores de velocidad.

La matriz jacobiana directa nos permite conocer las velocidades del extremo del robot a partir de los valores de las velocidades de las articulaciones, mientras que la jacobiana inversa nos permite conocer las velocidades que hay que dar a cada articulación a partir de una velocidad conocida en el extremo.

El cálculo que ahora se expone ha sido realizado, como parte de un trabajo, por los siguientes alumnos de "Robótica Industrial" de 5º de Ingeniería Industrial:

- César Alfaro Sánchez.
- Antonio Miguel García Benítez.
- Álvaro López Parra.

Para el cálculo del jacobiano han partido de la matriz de transformación T, obtenida mediante el algoritmo de Denavit-Hartenberg en el apartado anterior, considerando los ángulos $\theta_i = qi$, sin tenerse en cuenta las medidas del robot (l_i, a_i).

Así, tenemos que para el cálculo de las tres primeras filas de la matriz jacobiana tomamos la última columna de la matriz T, la cual quedaría:

D6*(sin(q5)*(sin(q1)*sin(q4) - cos(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3))) - cos(q5)*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))) - D4*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2)) + a2*cos(q1)*cos(q2)

a2*cos(q2)*sin(q1) - D4*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2)) - D6*(sin(q5)*(cos(q1)*sin(q4) - cos(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1))) + cos(q5)*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2)))

D1 + D4*(cos(q2)*cos(q3) - sin(q2)*sin(q3)) + D6*(cos(q5)*(cos(q2)*cos(q3) - sin(q2)*sin(q3)) - cos(q4)*sin(q5)*(cos(q2)*sin(q3) + cos(q3)*sin(q2))) + a2*sin(q2)

1

Antes de proceder al cálculo de la matriz jacobiana es necesario destacar que esa columna presenta la forma:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Donde X, Y y Z son las las coordenadas del efector final con respecto a la base.

Haciendo las derivadas parciales de cada una de las filas con respecto a cada una de las variables se obtendrían las tres primeras filas de la matriz Jacobiana tal y como se muestra en la siguiente ecuación:

$$J = \begin{pmatrix} \frac{\partial fx}{\partial q1} & \cdots & \frac{\partial fx}{\partial q6} \\ \vdots & \ddots & \vdots \\ \frac{\partial fz}{\partial q1} & \cdots & \frac{\partial fz}{\partial q6} \end{pmatrix}$$

Obteniendose:

Fila 1:

[D6(sin(q5)*(cos(q1)*sin(q4) - cos(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1))) + cos(q5)*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2))) + D4*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2)) - a2*cos(q2)*sin(q1),*

- D6(cos(q5)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)) - cos(q4)*sin(q5)*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))) - D4*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)) - a2*cos(q1)*sin(q2),*

- D6(cos(q5)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)) - cos(q4)*sin(q5)*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))) - D4*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)),*

*D6*sin(q5)*(cos(q4)*sin(q1) + sin(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3))),*

D6(cos(q5)*(sin(q1)*sin(q4) - cos(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3))) + sin(q5)*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))),*

0]

Fila 2:

[D6(sin(q5)*(sin(q1)*sin(q4) - cos(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3))) - cos(q5)*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))) - D4*(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2)) + a2*cos(q1)*cos(q2),*

D6(cos(q5)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)) + cos(q4)*sin(q5)*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2))) + D4*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)) - a2*sin(q1)*sin(q2),*

D6(cos(q5)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)) + cos(q4)*sin(q5)*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2))) + D4*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)),*

*-D6*sin(q5)*(cos(q1)*cos(q4) + sin(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1))),*

-D6(cos(q5)*(cos(q1)*sin(q4) - cos(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1))) - sin(q5)*(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2))),*

0]

Fila 3:

[0,

*a2*cos(q2) - D6*(cos(q5)*(cos(q2)*sin(q3) + cos(q3)*sin(q2)) + cos(q4)*sin(q5)*(cos(q2)*cos(q3) - sin(q2)*sin(q3))) - D4*(cos(q2)*sin(q3) + cos(q3)*sin(q2)),*

- D4(cos(q2)*sin(q3) + cos(q3)*sin(q2)) - D6*(cos(q5)*(cos(q2)*sin(q3) + cos(q3)*sin(q2)) + cos(q4)*sin(q5)*(cos(q2)*cos(q3) - sin(q2)*sin(q3))),*

*D6*sin(q4)*sin(q5)*(cos(q2)*sin(q3) + cos(q3)*sin(q2)),*

-D6(sin(q5)*(cos(q2)*cos(q3) - sin(q2)*sin(q3)) + cos(q4)*cos(q5)*(cos(q2)*sin(q3) + cos(q3)*sin(q2))),*

0]

Para obtener las tres últimas filas de la matriz jacobiana se van a utilizar las diferentes matrices de transformación A obtenidas por el método de Denavit-Hartenberg. Así tendremos:

$$A_0^2 = A_0^1 \times A_1^2$$

$$A_0^3 = A_0^1 \times A_1^2 \times A_2^3$$

A partir de estas ecuaciones se tomarán las tres primeras filas y columnas y se multiplicarán por la matriz

$$b = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

De forma que la primera columna sería únicamente ‘b’, la segunda sería:

$$A_0^1 \times b$$

La tercera sería:

$$A_0^2 \times b$$

Obteniendo las tres últimas filas de la matriz jacobiana tal y como se muestran a continuación.

Fila 4:

[0,
sin(q1),
sin(q1),

*- cos(q1)*cos(q2)*sin(q3) - cos(q1)*cos(q3)*sin(q2),*
*cos(q4)*sin(q1) + sin(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3)),*
sin(q5)(sin(q1)*sin(q4) - cos(q4)*(cos(q1)*cos(q2)*cos(q3) - cos(q1)*sin(q2)*sin(q3))) -*
cos(q5)(cos(q1)*cos(q2)*sin(q3) + cos(q1)*cos(q3)*sin(q2))]*

Fila 5:

[0,
-cos(q1),
-cos(q1),

*-cos(q2)*sin(q1)*sin(q3)-cos(q3)*sin(q1)*sin(q2),*
*- cos(q1)*cos(q4) - sin(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1)),*

-sin(q5)(cos(q1)*sin(q4)- cos(q4)*(sin(q1)*sin(q2)*sin(q3) - cos(q2)*cos(q3)*sin(q1))) -*
cos(q5)(cos(q2)*sin(q1)*sin(q3) + cos(q3)*sin(q1)*sin(q2))]*

Fila 6:

[1,
0,
0,

*cos(q2)*cos(q3)-sin(q2)*sin(q3),*
sin(q4)(cos(q2)*sin(q3)+cos(q3)*sin(q2)),*
cos(q5)(cos(q2)*cos(q3) - sin(q2)*sin(q3)) - cos(q4)*sin(q5)*(cos(q2)*sin(q3) +*
*cos(q3)*sin(q2))]*

5.- Software.

El programa se ha realizado sin el uso de ningún entorno de desarrollo, teniendo como base el lenguaje de programación C y C++ y el compilador de Linux GNU.

Para el tratamiento de las imágenes y el control del brazo robot se han utilizado dos librerías externas para C: OpenCV para el tratamiento de las imágenes y la librería m5api para el control de los módulos del brazo.

5.1.- Lenguaje C.

El lenguaje de programación C fue creado en 1972 en los Laboratorios Bell. Se considera un lenguaje de nivel medio aunque con muchas características de bajo nivel. Algunas de sus características son:

- Un núcleo de lenguaje simple, con funcionalidades añadidas importantes que son proporcionadas por bibliotecas.
- Acceso a memoria de bajo nivel mediante el uso de punteros.
- Un número reducido de palabras clase.

5.2.- OpenCV.

OpenCV (Open Source Computer Vision Library) es una librería o biblioteca abierta de funciones de programación para la visión por computador en tiempo real, desarrollada por Intel.

Desde el lanzamiento de su versión alfa en 1999 ha sido utilizado en infinidad de aplicaciones tanto por su potente motor como por el hecho de ser una librería abierta y de uso gratuito.

OpenCV está escrito en C++, pero a pesar de ello hay interfaces completas para los principales lenguajes de programación como son: Python, Java y MatLab.

En cuanto a los sistemas operativos que lo soportan, OpenCV tiene en su página web oficial instaladores para los principales sistemas operativos: Windows, Linux, iOS, Android y OS X.

Además, OpenCV es usado como el principal paquete de tratamiento de imágenes en la plataforma ROS (Robot Operating System), la cual proporciona librerías y herramientas para los desarrolladores de software para el trabajo con aplicaciones robóticas.

Las principales aplicaciones que podemos desarrollar gracias a OpenCV son:

- Reconocimiento facial y gesticular.

- Interacción hombre-computador (HCI).
- Robótica móvil.
- Identificación y seguimiento de objetos.
- Segmentación y reconocimiento.
- Redes neuronales.
- Machine Learning

5.3.- Librería m5api.

La librería m5api, administrada por la empresa alemana Schunk, es la que nos permite controlar cada uno de los motores por los que está formado el brazo modular. Sólo se encuentra disponible para el desarrollo de programas en C++ y para los siguientes sistemas operativos:

Sistema Operativo	Forma	Compiladores soportados
MS Windows 9x/NT/2000/XP	Librería de enlace dinámico (DLL)	Visual C/C++ Visual Basic National Instrument LabWindows CVI
Suse Linux 6.4	Código abierto	GNU C/C++
QNX 4.25	Código abierto	Watcom C/C++ v.10

5.3.1.- Principales funciones de la librería m5api.

Apertura y cierre de la interfaz de comunicación	Función	Descripción
	PCube_openDevice	Abre la interfaz especificada por InitString. Devuelve un ID válido.
	PCube_closeDevice	Cierra la interfaz especificada.

Funciones administrativas	Función	Descripción
	PCube_getModuleIdMap	Recupera el número de módulos que hay en el bus. Al mismo tiempo visualiza la dirección física de los módulos en ID lógicas, estas son guardadas en un array en orden ascendente.
	PCube_updateModuleIdMap	Actualiza el mapa de módulos
	PCube_getModuleCount	Devuelve el número de módulos conectados.
	PCube_getModuleType	Asocia cada módulo a un tipo de mecanismo. Mecanismo de rotación: TYPEID_MOD_ROTARY = 0x0F Mecanismo lineal: TYPEID_MOD_LINEAR = 0xF0
	PCube_getModuleVersion	Recupera la versión del sistema operativo del módulo, el resultado está expresado en hexadecimal.
	PCube_getModuleSerialNo	Recupera el número de serie de un módulo.
	PCube_getDllVersion	Devuelve la versión del DLL utilizado.

	PCube_configFromFile	Permite configurar el sistema a partir de un archivo Ini.
	PCube_serveWatchdogAll	Refresca el perro guardián de todos los módulos. Sólo es válido para CAN.
	PCube_getDefSetup	Devuelve la configuración por defecto de un módulo. El resultado es una palabra de instalación
	PCube_getDefBaudRate	Recupera el valor por defecto de la velocidad de comunicación o ancho de banda. Sólo valido para RS232 y CAN. El resultado está expresado entre 0 y5.
	PCube_setBaudRateAll	Ajusta la velocidad de comunicación a la especificada. Sólo válido para bus CAN.
	PCube_getDefGearRatio	Devuelev el valor de la relación de transmisión por defecto.
	PCube_getDefLinearRatio	Devuelve el factor de conversión por defecto de un movimiento rotatorio a lineal.
	PCube_getDefCurRatio	Devuelve el factor por defecto para la conversión de corriente en dígitos a Amperios.
	PCube_getDefBrakeTimeOut	Recupera el retraso por defecto entre el final del movimiento y el descanso.
	PCube_getDefIncPerTurn	Devuelve el valor por defecto del número de incrementos por rotación del motor.

Recuperación de posición	**Función**	**Descripción**
	PCube_getPos	Devuelve la posición actual del módulo especificado por las ID del mecanismo y del módulo. El resultado es expresado en radianes.
	PCube_getPosInc	Devuelve la posición actual del módulo especificando por las ID del mecanismo y del módulo. El resultado es expresado en incrementos.

Recuperación de velocidad	**Función**	**Descripción**
	PCube_getVel	Devuelve la velocidad actual asociada al mecanismo y al módulo especificado. El resultado es expresado en radianes/s.
	PCube_getVelInc	Devuelve la velocidad actual del mecanismo asociado a las ID especificadas. El resultado es expresado en incrementos/s.
	PCube_getIPolVel	Devuelve la velocidad interpolada del mecanismo asociado a las ID especificadas. El resultado es expresado en radianes/s.

Recuperación de estado del módulo	**Función**	**Descripción**
	PCube_getModuleState	Devuelve el estado actual del módulo asociado a las ID especificadas. El resultado es una palabra de estado.
	PCube_getStateDioPos	Devuelve una combinación de información acerca del estado del módulo, posición y estado digital.

Recuperación de posición síncrona	**Función**	**Descripción**
	PCube_savePosAll	Fuerza a todos los módulos conectados a guardar la posición actual.
	PCube_getSavePos	Recupera la posición almacenada con la función PCube_savePosAll de un módulo.

Configuración	Función	Descripción
	PCube_getDefConfig	Recuperación la configuración por defecto del módulo asociado a las ID especificadas. El resultado es una palabra de configuración.
	PCube_getConfig	Recupera la configuración actual del módulo asociado a las ID especificadas.
	PCube_setConfig	Ajusta la configuración del módulo a la deseada a través de una nueva palabra de configuración.

Coeficientes del lazo PID	Función	Descripción
	PCube_getDefA0	Devuelve el valor por defecto del parámetro A0.
	PCube_getA0	Devuelve el valor actual del parámetro A0.
	PCube_setA0	Ajusta el valor de A0 al valor deseado (1..12).
	PCube_getDefC0	Devuelve el valor por defecto del parámetro C0.
	PCube_getC0	Devuelve el valor actual del parámetro C0.
	PCube_setC0	Ajusta el valor de C0 al valor deseado (12..64).
	PCube_getDefDamp	Devuelve el valor por defecto del coeficiente de amortiguamiento del sistema.
	PCube_getDamp	Devuelve el valor actual del coeficiente de amortiguamiento del sistema.
	PCube_setDamp	Ajusta el valor del coeficiente de amortiguamiento al valor deseado (1..4).
	PCube_recalcPIDParams	Actualiza el lazo PID y crea nuevos coeficientes válidos.

Rango de posición: posición mínima	Función	Descripción
	PCube_getDefMinPos	Devuelve la posición mínima por defecto del módulo especificado. El resultado está expresado en radianes.
	PCube_getMinPos	Devuelve la posición mínima del módulo especificado. El resultado está expresado en radianes.
	PCube_getMinPosInc	Devuelve la posición mínima del módulo especificado. El resultado está expresado en incrementos.
	PCube_setMinPos	Ajusta el valor actual de posición mínima del módulo especificado. El valor ha de ser en radianes o metros.

Rango de operación: posición mínima	Función	Descripción
	PCube_setMinPosInc	Ajusta el valor actual de posición mínima del módulo especificado. El valor está expresado en incrementos.

Rango de operación: posición máxima	Función	Descripción
	PCube_getDefMaxPos	Devuelve el valor de posición máxima por defecto del módulo especificado. El resultado está expresado en radianes.
	PCube_getMaxPos	Devuelve la posición máxima del módulo especificado. El resultado está expresado en radianes o metros.
	PCube_getMaxPosInc	Devuelve la posición máxima del módulo especificado. El resultado está expresado en incrementos.
	PCube_setMaxPos	Ajusta el valor de posición máxima del módulo especificado. El valor ha de ser expresado en radianes o metros.

	PCube_setMaxPosInc	Ajusta el valor de posición máxima del módulo especificado. El valor ha de ser expresado en incrementos.

Velocidad máxima	Función	Descripción
	PCube_getDefMaxVel	Devuelve la velocidad máxima por defecto del módulo especificado. La velocidad está expresada en rad/s o m/s.
	PCube_getMaxVel	Devuelve la velocidad máxima del módulo especificado. El resultado está expresado en rad/s o m/s.
	PCube_getMaxVelInc	Devuelve la velocidad máxima del módulo especificado. El resultado está expresado en incrementos/s.
	PCube_setMaxVel	Ajusta la velocidad máxima del módulo especificado. El valor ha de ser expresado en rad/s o m/s.
	PCube_setMaxVelInc	Ajusta la velocidad máxima del módulo especificado. El valor ha de ser expresado en incrementos/s.

Aceleración máxima	Función	Descripción
	PCube_getDefMaxAcc	Devuelve la aceleración máxima por defecto del módulo especificado. El resultado está expresado en rad/s² o m/s².
	PCube_getMaxAcc	Devuelve la aceleración máxima del módulo especificado. El resultado está expresado en rad/s² o m/s².
	PCube_getMaxAccInc	Devuelve la aceleración máxima del módulo especificado. El resultado está expresado en Incrementos/s².
	PCube_setMaxAcc	Ajusta la aceleración máxima del módulo especificado. El valor ha de ser expresado en rad/s² o m/s².

Parada rápida	Función	Descripción
	PCube_haltModule	Provoca una parada rápida del módulo especificado.
	PCube_haltAll	Provoca una parada rápida de todos los módulos.

Reinicio del módulo	Función	Descripción
	PCube_resetModule	Reinicia el módulo especificado.
	PCube_resetAll	Reinicia todos los módulos conectados al bus.

Movimiento en modo rampa con posición especificada	Función	Descripción
	PCube_movePos	Comienza un movimiento en modo rampa del módulo especificado a la posición deseada. La posición deseada está expresada en radianes o metros. La velocidad y aceleración deben de ser ajustadas las funciones de ajuste de velocidad de rampa y de aceleración: PCube_setRampVel y PCube_setRampAcc resp. PCube_setRampVelInc y PCube_setRampAccInc.
	PCube_movePosInc	Comienza un movimiento en modo rampa del módulo especificado a la posición deseada. La posición ha de ser expresada en incrementos.
	PCube_setMaxAccInc	Ajusta la aceleración máxima del módulo especificado. El valor ha de ser expresado en Incrementos/s².

Movimiento en modo rampa con posición, velocidad y aceleración especificada	Función	Descripción
	PCube_moveRamp	Comienza un movimiento en modo rampa del módulo especificado. La posición objetivo es dada en radianes o metros. La velocidad ha de ser dada en rad/s o m/s. La aceleración ha de ser dada en rad/s² o m/s².
	PCube_moveRampInc	Comienza un movimiento en modo rampa del módulo especificado. La posición es dada en incrementos, la velocidad en Incrementos/s y la aceleración en Incrementos/s².
	PCube_moveRampExtended	Comienza un movimiento en modo rampa del módulo especificado. La posición es dada en incrementos, la velocidad en incrementos/s y la aceleración en Incrementos/s². Devuelve el estado, posición actual y estado digital.

Movimiento de velocidad constante	Función	Descripción
	PCube_moveVel	Comienza un movimiento de velocidad constante, expresada en rad/s o m/s.
	PCube_moveVelInc	Comienza un movimiento de velocidad constante, expresada en incrementos/s.
	PCube_moveVelExtended	Comienza un movimiento de velocidad constante, expresada en rad/s o m/s. Devuelve el estado del módulo, posición actual y estado digital.

Movimiento con posición y tiempo específicos.	Función	Descripción
	PCube_moveStep	Comienza un movimiento a la posición deseada, especificada en radianes o metros y el tiempo en ms.
	PCube_moveStepInc	Comienza un movimiento a la posición deseada, especificada en incrementos y el tiempo en ms.
	PCube_moveStepExtended	Comienza un movimiento a la posición deseada, especificada en radianes o metros y el tiempo en ms. Devuelve el estado del módulo, posición actual y estado digital.

5.4.- Trabajo sobre Windows XP.

Para Windows disponemos de varios programas proporcionados por Robotnik para la teleoperación del robot: PowerCube, VCollide 201.

5.4.1.- PowerCube Software.

Al arrancar el programa nos aparece una ventana con las siguientes opciones:

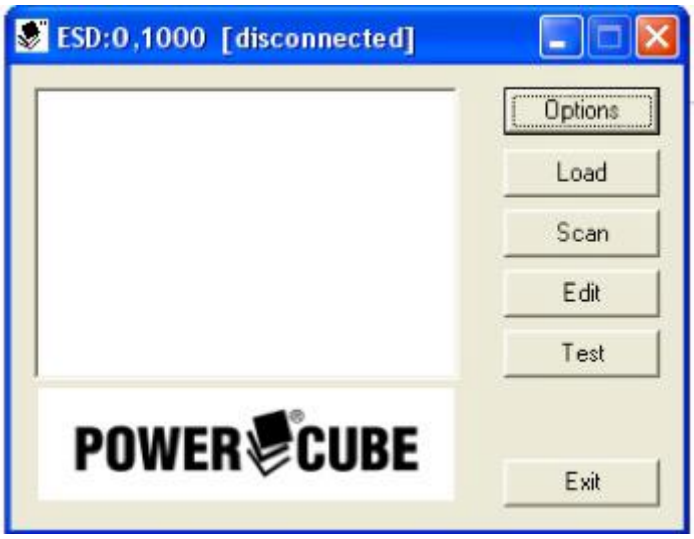


Fig. 5.1.- *Ventana principal del programa PowerCube.*

- Options. En esta opción podemos configurar el puerto de comunicaciones con el robot. En la entrada InitString se define el puerto usado, siendo en nuestro caso "ESD:1,1000". Donde el "0" representa el puerto usado en la tarjeta PCI-CAN y el "1000" es la tasa de baudios por segundos en la comunicación.

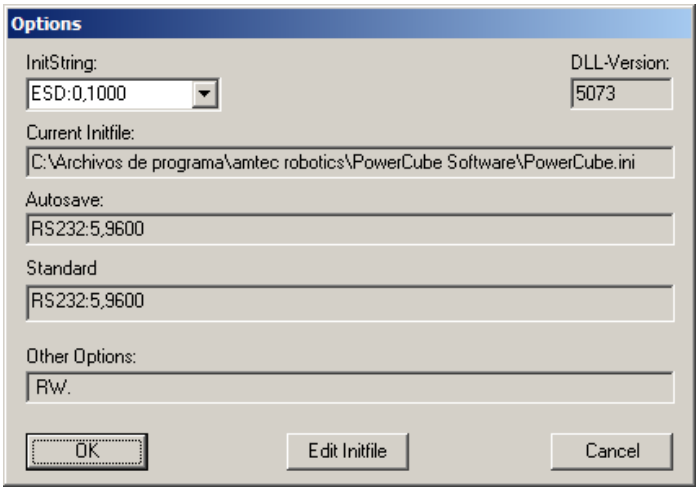


Fig. 5.2.- *Ventana Options en el programa PowerCube.*

- Load. Esta opción nos permite cargar datos en el programa desde un fichero .pcc.
- Scan. El botón de "Scan" permite escanear los dispositivos conectados al puerto CAN de nuestro equipo. Una vez escaneados vemos como aparecen los 6 módulos del brazo en la ventana principal del programa, definidos cada uno por un número de serie "SN".

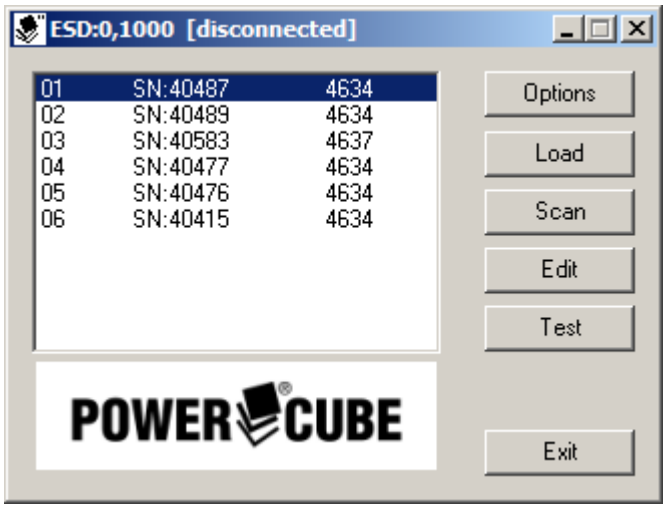


Fig. 5.3.- *Ventana principal tras realización de "Scan".*

- Edit. Esta opción permite modificar los datos cargados con la opción load. Al pulsar el botón nos aparece la siguiente ventana.

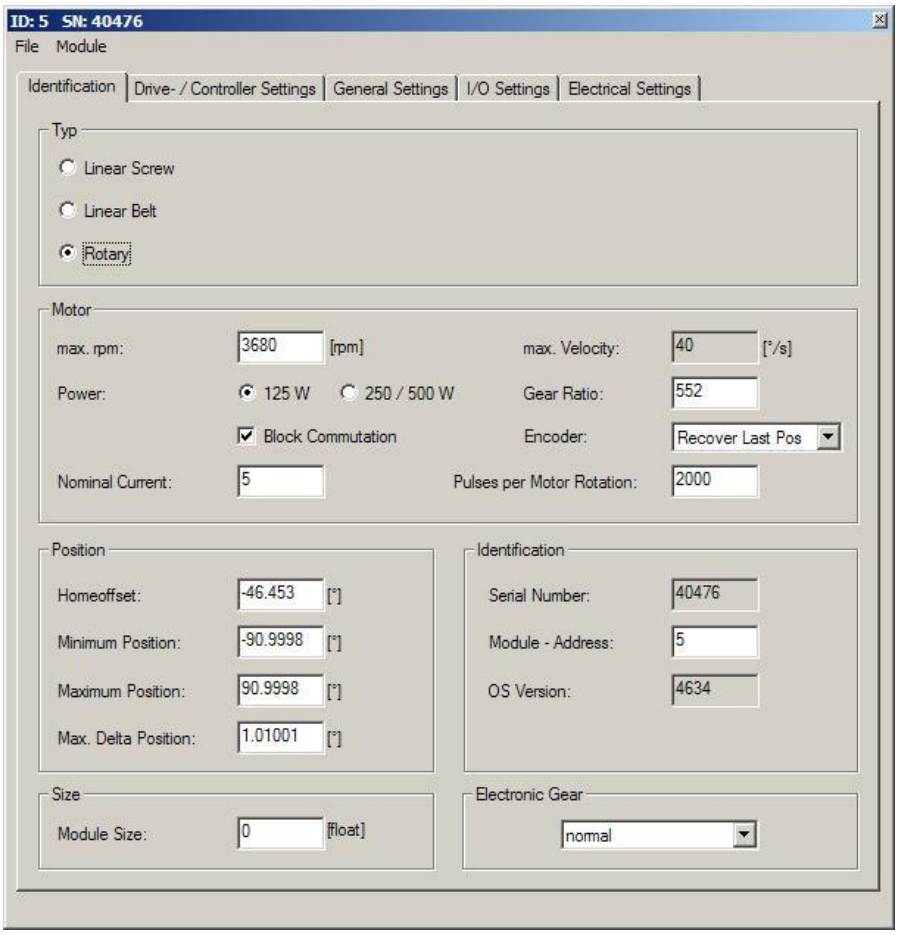


Fig. 5.4.- *Ventana "Edit".*

Como se puede observar en la figura 4 disponemos de 5 pestañas: Identification, Drive-/Controller Settings, General Settings, I/O Settings, Electrical Settings.

- Identification: en esta pestaña podemos modificar diferentes parámetros:
  - Typ: nos permite seleccionar el tipo de motor, en nuestro caso rotatorio.
  - Motor: podemos modificar diferentes parámetros del motor del módulo seleccionado.
    - Max. Rpm: revoluciones máximas por minuto.
    - Power: potencia en W.
    - Nominal Current: corriente nominal del motor.
    - Max. Velocity: viene definida por las max. Rpm.
    - Gear Ratio: relación de transmisión.
    - Encoder: tipo de encoder usado.
    - Pulses per Motor Rotation: nº de pulsos para definir la posición del motor.
  - Position: podemos modificar diferentes parámetros para configurar los límites del motor.
    - Homeoffset: punto inicial.
    - Minimum Position: posición mínima.
    - Maximum Position: posición máxima.
  - Identification: nos da información para la identificación de los módulos.
    - Serial Number: número serie del módulo, este número es único para cada rotor.
    - Module – Address: nombre del módulo.
    - OS Version: versión del rotor.

- Drive-/Controller Settings: en esta pestaña podemos modificar diferentes parámetros para el control del movimiento.

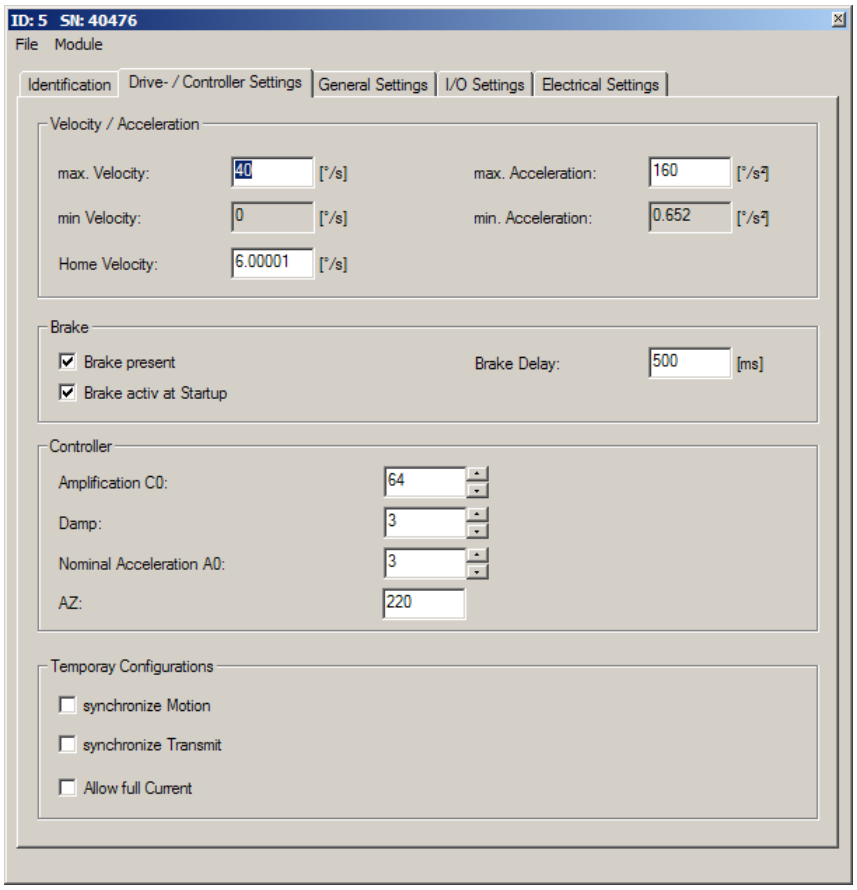


Fig. 5.5.- *Ventana "Drive-/Controller Settings".*

- Velocity / Acceleration: permite modificar parámetros referentes a la velocidad y aceleración del rotor.
    - Max. Velocity: velocidad máxima.
    - Min Velocity: velocidad mínima.
    - Home Velocity: velocidad para el movimiento Home.
    - Max. Acceleration: aceleración máxima.
    - Min Acceleration: aceleración minima.
- Brake: permite modificar el funcionamiento del enclavamiento.
    - Brake present: seleccionar para que el enclavamiento funcione.
    - Brake activ at Startup: seleccionar para que el rotor este enclavado al inicio.
    - Brake Delay: tiempo que tarda en funcionar el freno una vez activado.
- Controller: permite modificar el lazo PID.
    - Amplification C0: permite ajustar el coeficiente C0 del PID.

- Damp: permite ajustar el coeficiente de amortiguamiento del PID.

- General Settings: permite modificar parámetros referidos a la comunicación.

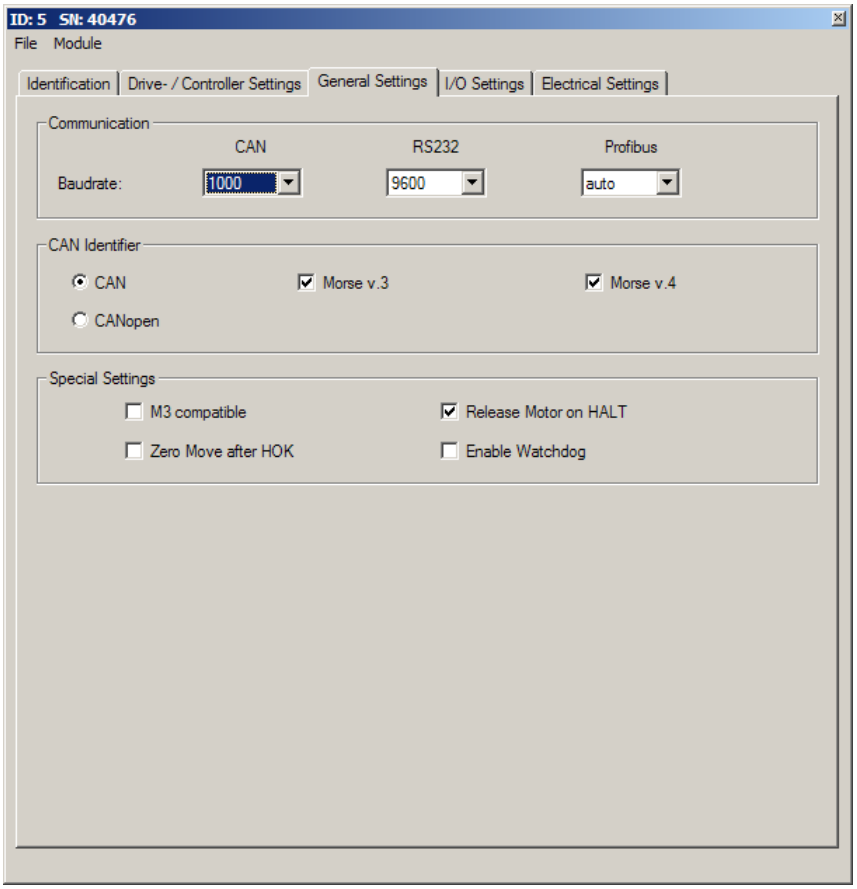


Fig. 5.6.- *Ventana "General Settings".*

- Communication: permite modificar parámetros para la configuración.
  - Baudrate: nos permite seleccionar la tasa de baudios en la comunicación por medio de CAN, RS-232 y Profibus.
- CAN Identifier: selección del tipo de CAN.
- Special Settings: configuraciones especiales.

- I/O Settings: permite configurar las entradas y salidas.

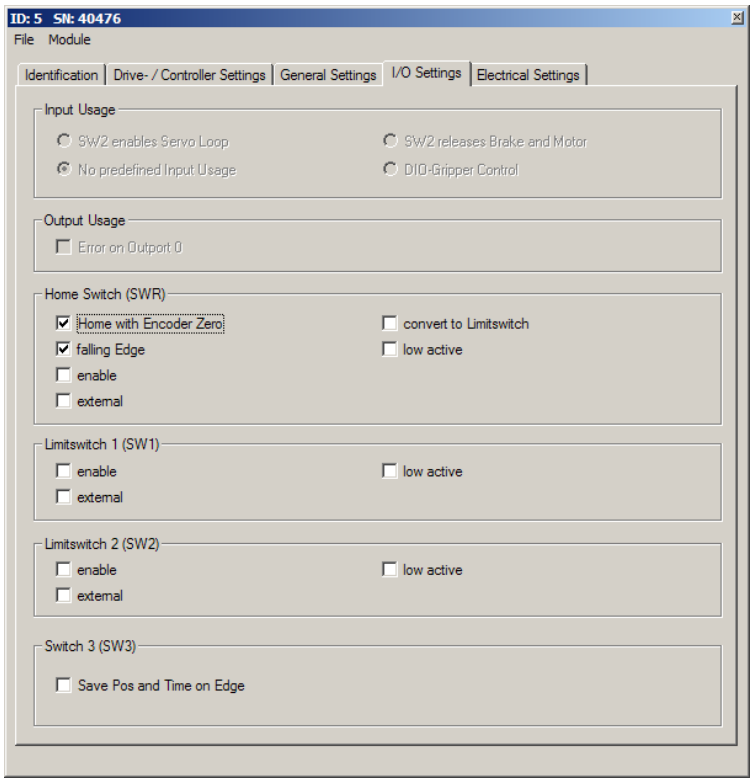


Fig. 5.7.- *Ventana "I/O Settings".*

- Electrical Settings: en esta pestaña podemos observar los valores máximos y mínimos referidos a la parte eléctrica del rotor.

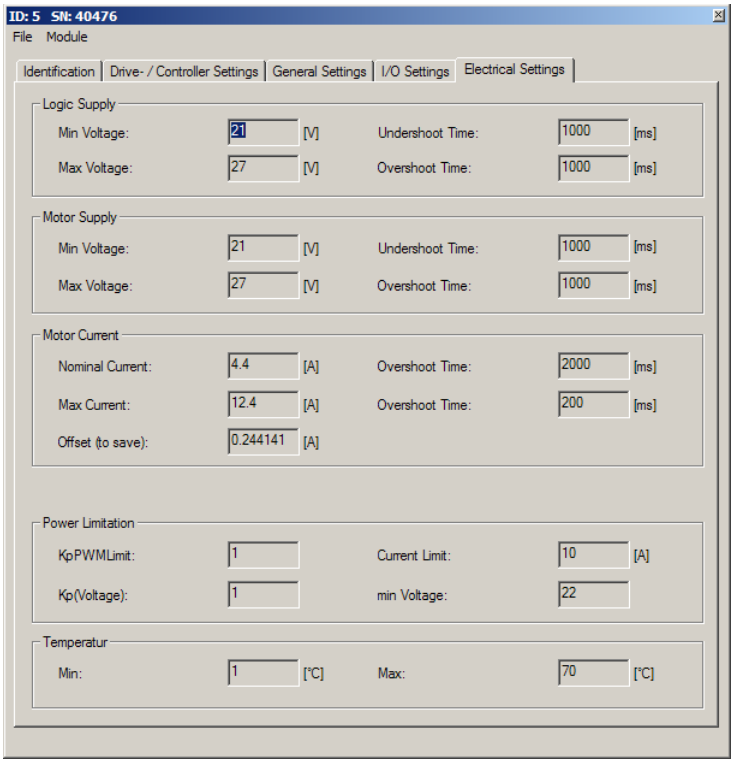


Fig. 5.8.- *Ventana "Electrical Settings".*

- Test. Esta opción nos permite teleoperar cada una de las articulaciones de forma independiente tanto en posición como en velocidad. En la parte superior tenemos 6 pestañas para seleccionar la unión que deseamos mover.

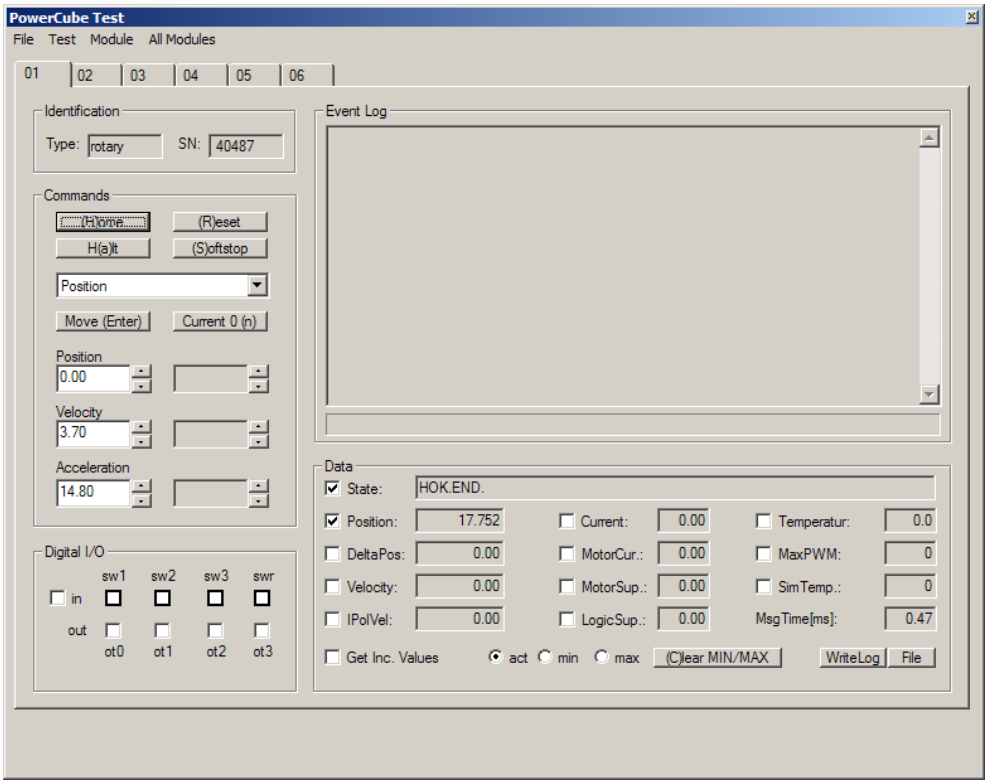


Fig. 5.9.- *Ventana "Test".*

En la ventana podemos diferenciar 5 secciones:

- Identification: muestra el SN y el tipo del rotor.
- Commands: nos permite introducir valores para la posición, velocidad y aceleración.
- Digital I/O: nos permite activar o desactivar las entradas entradas y ver si se activa alguna salida.
- Event Log: esta ventana cuando está activado el log nos muestra un listado con las diferentes acciones que se han producido.
- Data: en este apartado podemos observar en tiempo real diferentes parámetros del rotor como son el estado, velocidad, posición…

5.4.2.- VCollide201.

Este programa implementa la librería Vcollide. Incluye demos sobre la detección de choques entre diferentes objetos.

5.5.- Trabajo sobre Linux.

Bajo este sistema operativo realizaremos la programación y ejecución del robot.

5.5.1.- Creación e inicio de sesión root.

Para la realización de algunas tareas necesitaremos ser administradores del sistema o roots. Esto nos permitirá tener acceso a cualquier tipo de modificación en el sistema.

- Abrimos una ventana del terminal y ponemos:

sudo –u root passwd

- Escribimos la contraseña antigua.
- Nos pedirá una contraseña nueva.
- Pedirá confirmación de contraseña nueva.
- Cerramos el terminal, vamos a System/administration/login window/security
- Y marcamos la opción: *allow local system administrator login.*
- Abrimos un terminal y escribimos:

su

- Nos pedirá la contraseña, ponemos la contraseña nueva y pulsamos intro y ya tendremos acceso root o administrador solo en ese terminal, si abrimos otro será necesario escribir su y la contraseña de nuevo.

En el caso de querer iniciar un programa con privilegios de administrador debemos poner en el terminal "*sudo <programa>"* y escribir la contraseña.

5.5.2.- Instalación de la librería OpenCV.

A continuación, se explica el procedimiento para la instalación de la librería.

-Iniciamos sesión como root.

-Lo primero que tenemos que hacer es instalar las dependencias necesarias y actualizar las existentes, para ello escribimos en el terminal:

 # sudo apt-get install build-essential libgtk2.0-dev libavcodec-dev libavformat-dev libjpeg62-dev libtiff4-dev libswscale-dev libjasper-dev

- Descargamos OpenCV de la página oficial, en nuestro caso seleccionamos la versión 1.0, ya que a partir de la 2.0, requiere la instalación de otros programas, los cuales no pudieron ser instalados: http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/

- Una vez descargado entramos al terminal, accedemos a la carpeta en la que se descargó con el comando cd y escribimos:

tar zxvf opencv-X.X.X.tar.gz; cd opencv-X.X.X

- Ahora necesitamos que se configure y nos de la información sobre si es posible instalarse o no y con qué características, para ello escribimos en el terminal:

./configure

- (FOTO)
- Si todo ha ido correcto nos saldrá un texto similar al anterior y ahora deberemos poner: make. Este proceso dura alrededor de media hora.

make

- Si se ha creado el programa correctamente ya podremos instalarlo en nuestro ordenador, para ello escribimos:

make install

5.5.3.- Instalación del uEye Software para la cámara.

Para la correcta utilización de la cámara en nuestro ordenador necesitamos instalar el paquete de software adecuado para el modelo de cámara del que disponemos.

Para su instalación es necesario seguir los siguientes pasos.

- Conectamos la cámara vía USB al pc.
- Accedemos a la página www.ids-imaging.com, dirigiéndonos a la sección productos/software para descargar los drivers para nuestra cámara.
- Seleccionamos la opción: *Software Package for USB and GigE uEye Cameras Version 4.00 32 bits for Linux.*
- Para descomprimir el archivo descargado escribimos en el terminal:

~$ tar xvf ueyesdk-setup-4.0-usb-i686.tar.gz

- Una vez descomprimido, habremos extraído el archivo: ueyesdk-setup-4.0-usb-i686.gz.run en la carpeta /home/robotnik. El otro archivo incluido en el fichero es para la instalación de cámaras con conexión Ethernet.
- Accedemos a la cuenta en modo root.

- En el terminal ejecutamos el archivo con el siguiente código:

```
# sh ./ueyesdk-setup-4.0-usb-i686.gz.run
```

- Para inicar la cámara escribimos en el terminal:

```
~$ /etc/init.d/ueyeusbdrc start
```

- Para pararla:

```
~$ /etc/init.d/ueyeusbdrc forcé-stop
```

Este paquete de software incluye drivers, interfaces y herramientas para su uso. Para la programación IDS da soporte, con la instalación de diversas librerías, para los siguientes lenguajes: C, C++, C#, Microsoft .NET y Visual Basic.

Dentro de las herramientas proporcionadas están:

- *uEye Camera Manager:* es la herramienta principal para la dirección de todas las cámaras conectadas al sistema.
- *uEye Demo:* es un programa que nos permite iniciar la adquisición de imágenes con la cámara, además de herramientas para la configuración óptima de la cámara para nuestras aplicaciones. Para la utilización de esta herramienta es necesario la instalación de la librería Qt4.

5.5.4.- Instalación de la cámara Orbit Sphere.

A diferencia de la cámara uEye esta cámara no requiere de la instalación de ningún driver o software especial, por lo que con los drivers que vienen incluidos con el S.O es suficiente para su utilización. A pesar de ello, para el control de sus motores sí es necesario la instalación del software QuickCam, disponible en la página oficial de Logitech, de forma gratuita para Windows y para Mac. En el caso de desear usar esta característica en Linux no existe un driver oficial aunque si hay drivers no oficiales desarrollados en proyectos alternativos.

5.5.5.- Desarrollo del programa.

Como ya se ha comentado se hará uso de las librerías m5api y OpenCV programando en el lenguaje C/C++. Para la escritura del código se ha utilizado el programa GEdit de Linux, sin la utilización de ningún entorno de desarrollo.

Para la compilación del programa desarrollado hemos utilizado la misma consola de Linux y el código “make”. Para ello, previamente, se ha generado un archivo conocido como: makefile, el cual es el fichero de texto que es utilizado por make para la realización de la compilación del programa.

5.5.3.- Ejecución del programa.

Para ejecutar el programa tenemos que acceder a la carpeta en la que se encuentra el programa desde el terminal y utilizar el comando:

./<nombre del programa>

6.- Comunicación PC-Brazo.

Para la comunicación del brazo modular con el ordenador de control tenemos dos posibles formas de conexión: Ethernet y CAN bus.

6.1.- Red Ethernet.

El robot modular puede trabajar independiente de un único PC. Para aplicaciones remotas o distribuidas con varios equipos es importante la utilización de la arquitectura JAUS, la cual soporta la comunicación a través de la red Ethernet. Esta red puede ser cableada o inalámbrica.

6.2.- Bus CAN.

El brazo posee una red CAN en la que están conectados todos los módulos del robot con el PC, a través de una tarjeta PCI-CAN que permita la conexión CAN con el ordenador. Los buses son inicialmente asignados según la siguiente tabla:

CAN 1	CAN 0
Brazo derecho – Módulo PowerCube 1	Brazo izquierdo – Módulo PowerCube 1
Brazo derecho – Módulo PowerCube 2	Brazo izquierdo – Módulo PowerCube 2
Brazo derecho – Módulo PowerCube 3	Brazo izquierdo – Módulo PowerCube 3
Brazo derecho – Módulo PowerCube 4	Brazo izquierdo – Módulo PowerCube 4
Brazo derecho – Módulo PowerCube 5	Brazo izquierdo – Módulo PowerCube 5
Brazo derecho – Módulo PowerCube 6	Brazo izquierdo – Módulo PowerCube 6
{Opcional Servo 2 pinza}	{Opcional Servo 2 pinza}

7.- Control visual.

En este proyecto se ha pretendido realizar un control visual para poder controlar el movimiento de un brazo robótico. Como ya se comentó al comienzo de esta memoria, la configuración brazo-cámara que se ha elegido es Eye-To-Hand o cámara en mano, en la que la cámara se encuentra situado en el extremo final del robot.

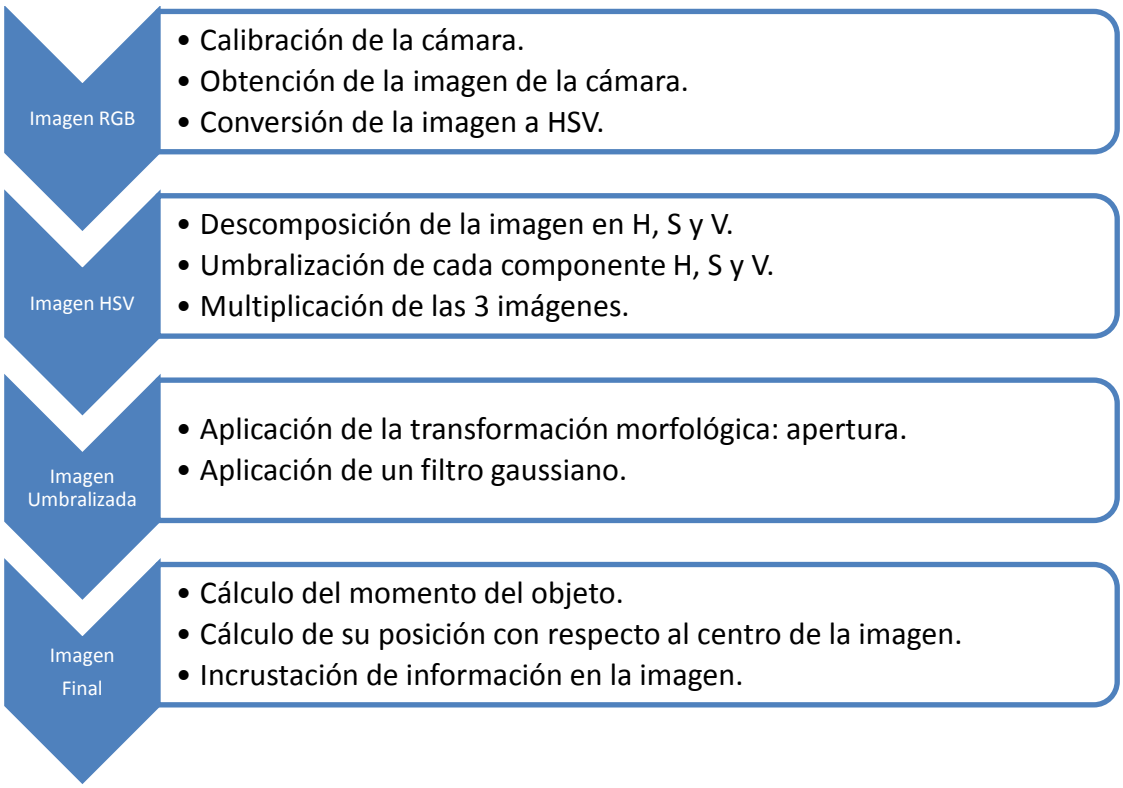
Por otro lado, el tipo de control elegido es IBVS o control visual basado en imagen, en el que se analizan las características de la imagen tomada por la cámara y se comparan con las de la imagen deseada.

En nuestro caso, la característica principal por la que regimos el movimiento del robot es que el centro de gravedad de un objeto se encuentre en el centro de la imagen captada por la cámara.

Podemos dividir en dos grandes partes el proyecto: la parte de visión por computador o tratamiento de la imagen y la parte de control del robot, ambas partes serán desarrolladas en las próximas páginas.

7.1.- Tratamiento de la imagen.

Para el procesamiento de la imagen se han seguido los pasos reflejados en el siguiente esquema:



A continuación se detallarán cada uno de los pasos.

7.1.1.- Calibración.

La calibración de la cámara [5] es un paso muy importante para la corrección, matemáticamente, de las principales desviaciones del modelo de pinhole que el uso de las lentes nos impone. Además esta calibración nos permite poder relacionar las mediciones de la cámara con las mediciones del mundo real, en tres dimensiones, lo cual es importante porque las imágenes que recibimos de la cámara no son en tres dimensiones, sino que son una proyección del mundo 3D a 2D. La relación existente entre la unidades con las que mide la cámara, píxeles, y las unidades físicas del mundo, metros, es una componente crítica en la reconstrucción de cualquier escena tridimensional.

Este proceso de calibración nos da un modelo geométrico de la cámara y un modelo de distorsión de la lente, definiendo ambos modelos los parámetros intrínsecos de la cámara. Con la obtención de ambos modelos se pretende poder corregir la distorsión de la lente y poder interpretar el entorno físico.

Gracias a la biblioteca de OpenCV podemos obtener los parámetros intrínsecos e extrínsecos de nuestra cámara de forma rápida y sencilla con la ayuda de un tablero de ajedrez. Para ello se toman imágenes con el tablero de ajedrez en diferentes posiciones, para que el programa tenga suficiente información para resolver su localización.

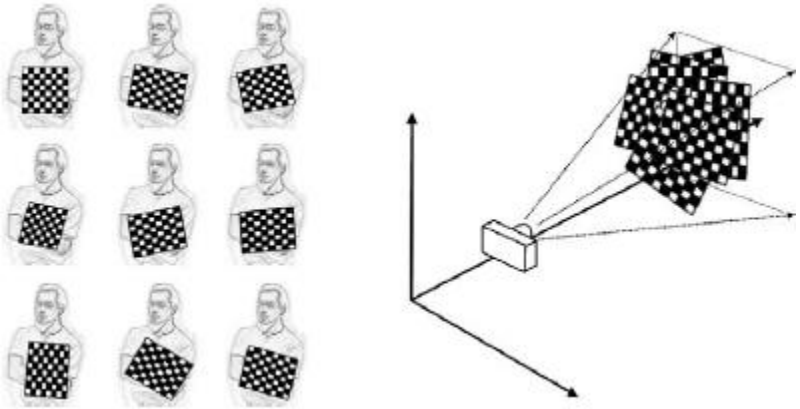


Fig. 7.1.- *Imágenes de un tablero de ajedrez sujetado en diferentes orientaciones para la realización de la calibración.*

El código utilizado para la calibración de la cámara ha sido extraído, con alguna modificación, de la página web http://dsynflo.blogspot.com.es, el cual está basado en el código ejemplo que aparece en el libro “Learning OpenCV” de O’Reilly. Dicho código

se encuentra en el Anexo (). Tras diversas pruebas en los que los resultados han sido muy parecidos hemos tomado los siguientes.

Los parámetros de distorsión obtenidos han sido:

K1= -3.9874e-22 K2=-4.0190e-15 P1=1.1609e-25 P2=3.4696e-25

Los parámetros intrínsecos fueron:

F_x=191.603 C_x=319.500

F_y=326.368 C_y=239.500

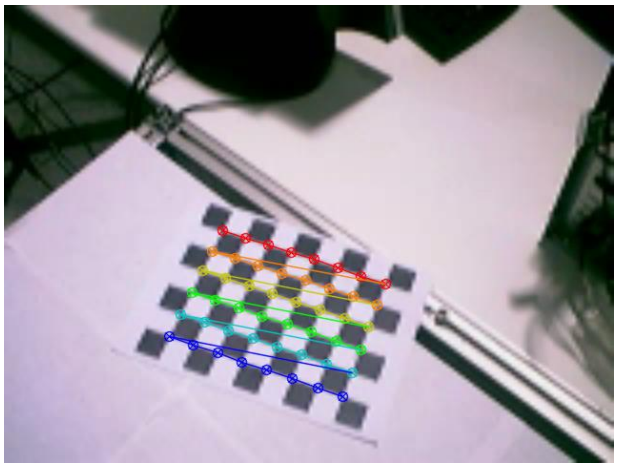


Fig. 7.2.- *Imagen obtenida durante el proceso de calibración.*

7.1.2.- Conversión del color.

Ya que el primer paso para el procesamiento de la imagen ha sido una conversión de RGB a HSV, es necesario explicar la diferencia entre ambos modelos, así el motivo por el cual se realiza dicha transformación.

7.1.2.1.- Modelo de color RGB.

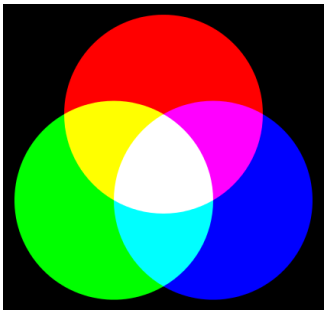


Fig. 7.3.- *Modelo RGB.*

El modelo RGB (Red, Green, Blue; Rojo, Verde, Azul) es el formado por la composición de los colores primarios de la luz. Con este modelo es posible representar un color mediante la mezcla de estos tres colores. Un inconveniente que presenta es que no define lo que es exactamente rojo o azul, por lo que los mismos valores RGB pueden mostrar diferentes colores en diferentes dispositivos, a pesar de usar este mismo modelo, ya que varían sus espacios de color.

7.1.2.2.- Modelo de color HSV.

El modelo HSV (Hue, Saturation, Value; Matiz, Saturación, Valor), también llamado HSB (Hue, Saturation, Brightness; Matiz, Saturación, Brillo) define un modelo de color en términos de sus componentes. Normalmente es presentado en forma de cono o ruleta de color.

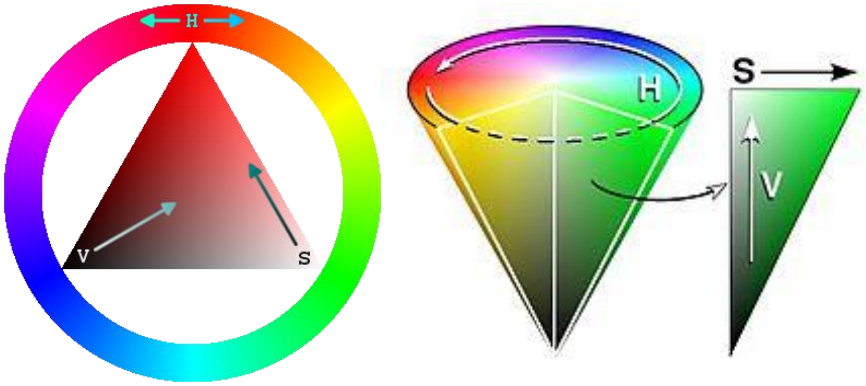


Fig. 7.4.- *Modelo HSV: forma circular y cónica.*

7.1.2.2.1.- Matiz.

Se representa con un valor entre 0° y 360°, aunque en el caso de la librería que utilizamos, OpenCV, se normaliza de 0 a 180°. Correspondiendo cada "grado" a un color, por lo que en cierta medida con la librería OpenCV perdemos 180 matices o colores.

7.1.2.2.2.- Saturación.

Se representa como la distancia al centro o eje de brillo negro-blanco. Se mide en %, siendo sus valores posibles entre 0 y 100. A mayor saturación de un color mayor intensidad presentará su tonalidad.

matiz 100% puro	75% de saturación	saturación media	25% de saturación	0 de saturación

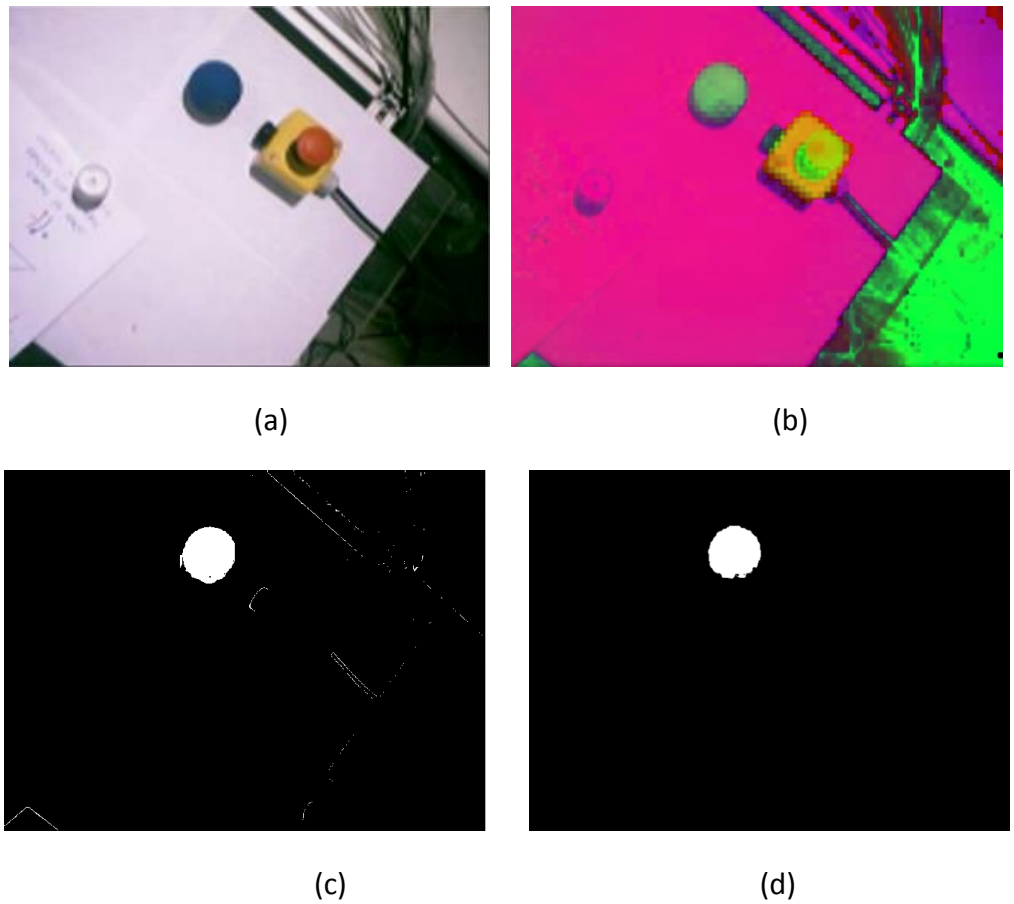
7.1.2.2.3.- Valor.

Representa la altura en el eje blanco-negro. Sus valores posibles varían del 0 al 100%, siendo 0 siempre negro y 100 blanco o un color más o menos saturado.

7.1.2.3.- Modelo HSV vs RGB.

La principal diferencia entre ambos es que el modelo HSV separa el *luma,* o la intensidad de la imagen, del *croma* o información del color. En la visión por computador esto nos permite aumentar la robustez del programa a los cambios de iluminación o eliminación de sombras, entre otros.

A continuación se mostraran diferentes imágenes umbralizadas a partir de la imagen en RGB y en HSV.



(a) (b)

(c) (d)

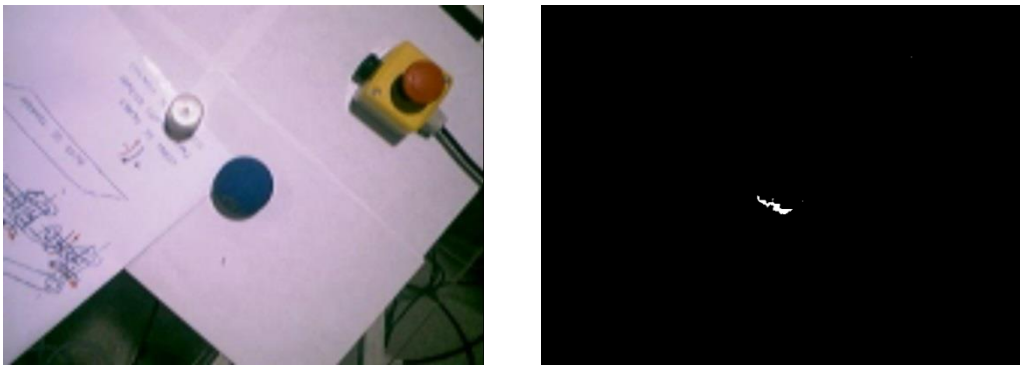
Fig. 7.5.- *Comparación de imágenes RGB y HSV: (a) Imagen en RGB. (b) Imagen HSV. (c) Imagen obtenida tras una umbralización de la imagen RGB usando los siguientes valores: R=0-70,G=0-100,B=70-180. (d) Imagen obtenida tras un filtrado de la imagen HSV usando los siguientes valores: H=90-120,S=110-255,V=50-255.*

(a)

(b) (c)

Fig. 7.6.- *Comparación de imágenes RGB y HSV 2: (a) Imagen en RGB. (b) Imagen obtenida tras una umbralización de la imagen RGB usando los siguientes valores: R=0-30,G=0-60,B=50-70. (c) Imagen obtenida tras una umbralización de la imagen HSV usando los siguientes valores: H=90-120,S=110-255,V=50-255.*

Como se puede apreciar, en este caso, con el modelo HSV no hemos tenido que modificar ningún valor. Mientras que con el modelo RGB es necesario, prácticamente, en cada instante, ya que en el entorno de trabajo se forman sombras, ya sean producidas por el cambio de intensidad de luz o por objetos cercanos. Así, si utilizamos los valores de umbralización de RGB del caso anterior en otro en el que existe más luz podemos ver, claramente, que el objeto es casi indetectable (fig. 7.7).



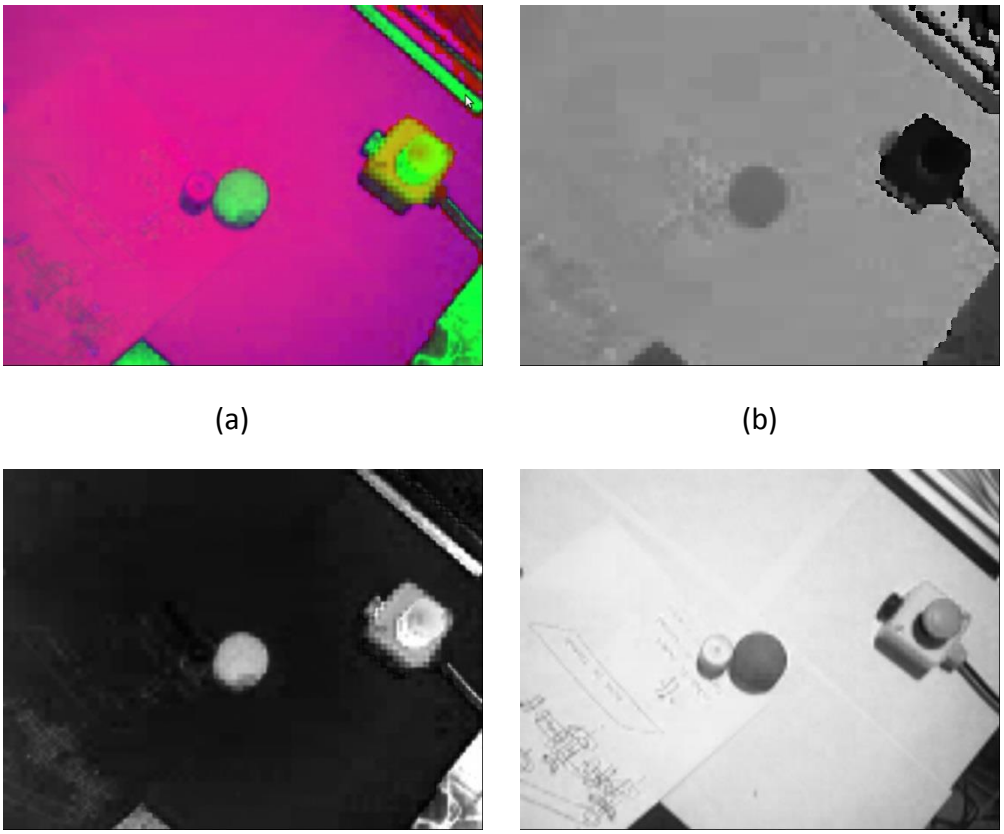
(a) (b)

Fig. 7.7.- *Umbralización de una imagen RGB: (a) Imagen en RGB. (b) Imagen obtenida tras una umbralización de la imagen con los mismos valores del ejemplo anterior.*

7.1.3.- Umbralización de la imagen HSV.

La librería OpenCV nos permite realizar este umbralizado con una única línea de código, pero es importante conocer el proceso que realiza internamente.

Partiendo de la imagen en HSV, se separa en sus 3 componentes Hue, Saturation y Value. En la siguiente imagen podemos la imagen de todas ellas.

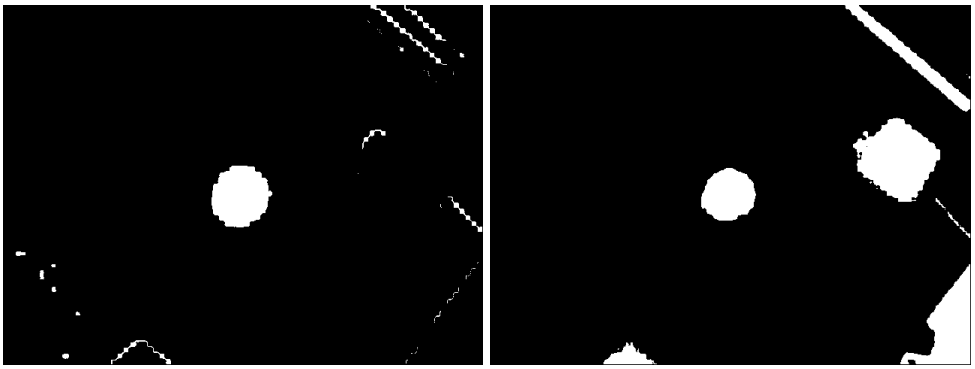


(a) (b)

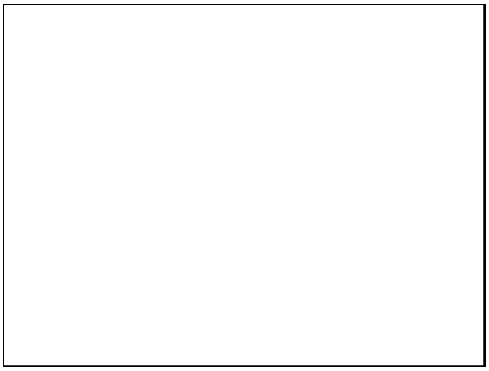
(c) (d)

Fig. 7.8.- *Descomposición de una imagen HSV: (a) Imagen HSV. (b) Componente Hue. (c) Componente Saturation. (d) Componente Value.*

Como se puede observar, las tres imágenes son imágenes en escala de grises por lo que se les realiza un umbralizado inferior y superior a cada una de ellas acorde a los valores que deseamos. En este caso se han tomado los valores expuestos anteriormente: H= 90-120, S= 100-255 y V= 0-255. De esta umbralización obtenemos las siguientes imágenes:



(a) (b)



(c)

Fig. 7.9.- *Umbralización de las componentes HSV: (a) Umbralización Hue. (b) Umbralización Saturation. (c) Umbralización Valor.*

En algunos casos, para trabajar con el modelo HSV será necesario modificar el rango de la magnitud V a 0-255, es decir, aumentar su rango con respecto a la intensidad de luz o luminancia. Esta modificación nos puede generar un pequeño ruido o "falso positivo" en la salida (Fig. 7.10.), el cual se consigue eliminar sin problemas con el procesamiento posterior. Por ello, puede llegar a ser conveniente mantener el rango de V entre 0-255 si el entorno de trabajo no disponible de unas características lumínicas constantes. En estos casos, al umbralizar obtendremos una imagen blanca (Fig. 7.9.).

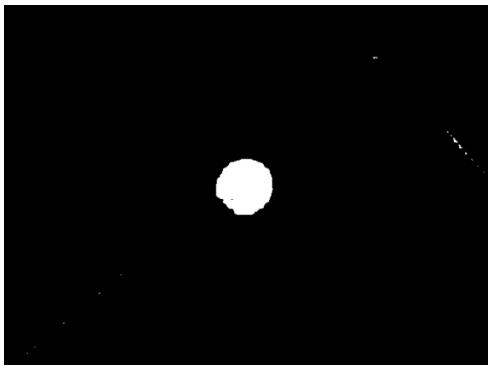


Fig. 7.10.- *Imagen obtenida tras la umbralización de la imagen HSV.*

Una vez realizado el umbralizado de las imágenes se produce su unión, dicha unión se realiza mediante multiplicaciones (recordad que una imagen no es más que una matriz formada por distintos valores). Como nuestras 3 imágenes o matrices estan compuestas por unos y ceros, ya que son binarias, al realizar la multiplicación nuestro resultado será aquella imagen formada por aquellos píxeles que valgan 1 en las tres imágenes.

7.1.4.- Reduccion de ruido.

A pesar del cuidado que se haya podido llevar en la elección de los umbrales durante el proceso de binarización, a menudo es posible encontrar píxeles aislados.

Como se puede apreciar, en la foto anterior existen píxeles a 1 que no pertenecen al objeto que nos interesa, el cual consideramos como ruido y es conveniente eliminar. Para poder eliminar este ruido se utilizarán herramientas de la morfología matemática. Estas herramientas o transformaciones morfológicas se tratan de una erosión, para eliminar el ruido, y una dilatación, para compensar los píxeles del objeto que sean eliminados con la erosión. El uso de estas dos técnicas juntas se conoce como apertura.

7.1.4.1.- Erosión.

$$\varepsilon_C(A) = A \ominus C$$

Es el conjunto de todos los puntos x, tales que C trasladado x, están contenidos en A.

Elimina aquellos grupos de píxeles en los que el elemento estructural no cabe.

Propiedades:

- Es antiextensiva, reduce el tamaño del objeto.
- Elimina los elementos que no caben en el EE.

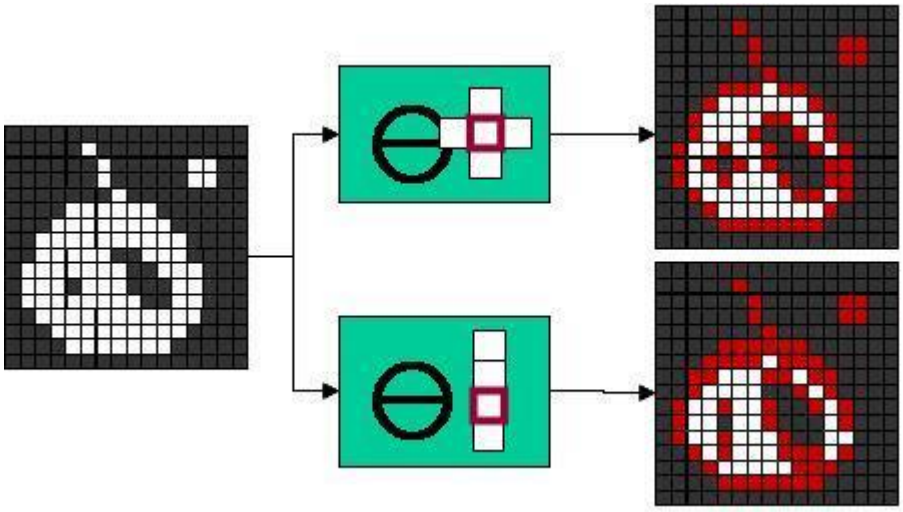


Fig. 7.11.- *Ejemplo de erosión. [7]*

7.1.4.2.- Dilatación.

$$\delta_C(A) = A \oplus C$$

Se obtiene en base a la reflexión de C con respecto a su origen y un desplazamiento x.

Dicho de otro modo convierte a 1 todos aquellos puntos barridos por el centro del elemento estructural mientras que algún punto de C coincida con alguno de A.

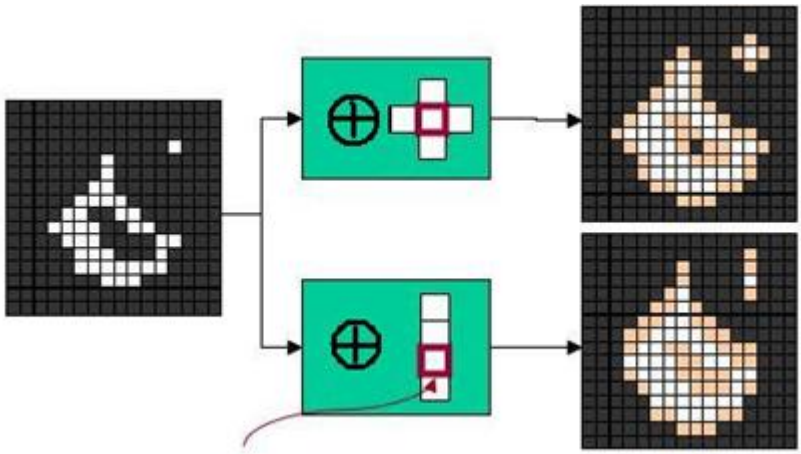


Fig. 7.12.- *Ejemplo de dilatación. [7]*

Propiedades:

- Es extensiva, hace más grande el objeto.
- Rellena entrantes en los que no quepa el elemento estructural.

7.1.4.3.- Apertura: erosión y dilatación.

$$\gamma_C(A) = (A \ominus C) \oplus C = \varepsilon_C(\delta_C(A))$$

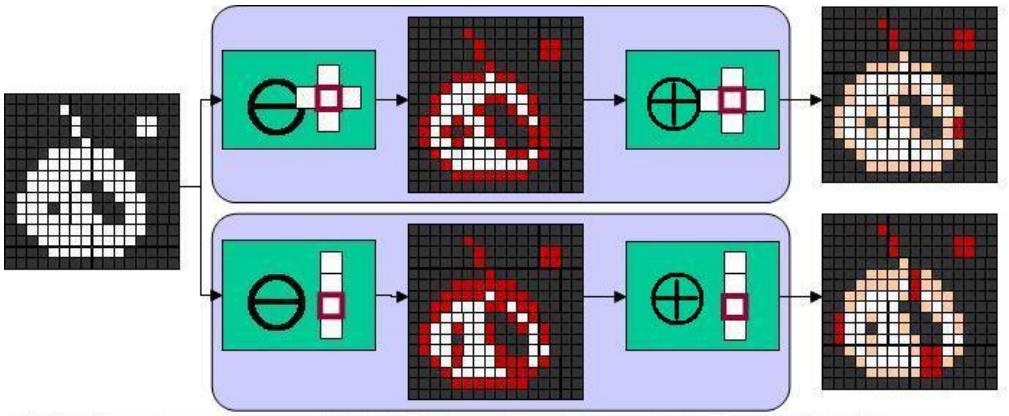


Fig. 7.13.- *Ejemplo de apertura. [7]*

En las siguientes imágenes podemos apreciar como se elimina el error existente en la imagen con la aplicación de la erosión, y la compensación de la pérdida de tamaño con la utilización del proceso de dilatación.

(a) (b)

Fig. 7.14.- *Aplicación de apertura a la imagen: (a) Aplicación del proceso de erosión. (b) Aplicación del proceso de dilatación.*

7.1.5.- Suavizado de contornos.

Probablemente sea el tratamiento de imagen más utilizado para la eliminación de ruido y suavizado. En este proceso se utiliza un filtro para, principalmente, suavizar el contorno del objeto y pueda ser mejor detectado posteriormente, además en el caso de que algún ruido no haya sido eliminado, anteriormente, con la apertura es posible que tras este proceso quede eliminado.

El filtro utilizado es un filtro gaussiano.

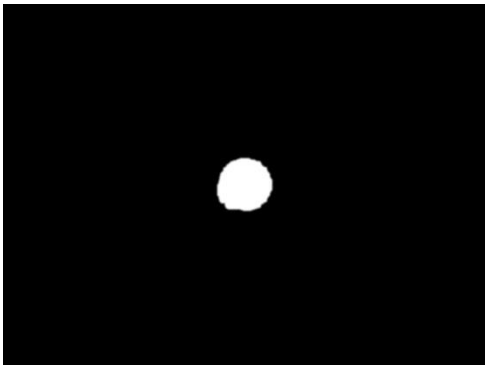


Fig. 7.15.- *Imagen obtenida tras la aplicación del filtro.*

7.1.6.- Identificación del centro del objeto.

En este proceso pretendemos, a partir de la imagen obtenida tras el suavizado, calcular el centro del objeto.

Para este cometido se han probado dos métodos, cada uno con sus ventajas y desventajas sobre el otro:

- Obtención del centro a través del cálculo del momento del objeto [8].
- Obtención del centro a través de la transformada de círculos de Hough [9].

A continuación explicaremos el proceso para cada uno de ellos.

7.1.6.1.- Obtención a través del momento del objeto.

Partiendo de la definición de momentos de una función $f(x,y)$ de un objeto tenemos:

$$m_{p,q} = \iint x^p y^q f(x,y) dxdy$$

Esta integración es calculada a partir del area del objeto. En el caso de usar imágenes binarias, como es nuestro caso, la función $f(x,y)$ se convierte en:

$$f(x,y) = b(x,y) = \begin{cases} 1 & Objeto \\ 0 & Fondo \end{cases}$$

Los momentos pueden ser clasificados según su orden. Este orden depende de los índices p y q del momento $m_{p,q}$, siendo el orden igual a la suma $p+q$. Teniendo en cuenta esto tenemos:

- Momentos de orden 0:

$$m_{0,0} = \iint b(x,y)dxdy = A$$

Este momento de orden cero describe el area A del objeto.

- Momentos de orden 1 ($(p,q) = (1,0)\ o\ (0,1)$)

$$m_{1,0} = \iint xf(x,y)dxdy$$

$$m_{0,1} = \iint yf(x,y)dxdy$$

Los momentos de orden 1 contiene información sobre el centro de gravedad del objeto. Este centro de gravedad puede ser descrito por el momento espacial de primer orden $m_{1,0}$ y $m_{0,1}$ divido por el momento de orden 0 $m_{0,0}$, con lo que nos queda:

$$x_c = \frac{m_{1,0}}{A} = \frac{m_{1,0}}{m_{0,0}}$$
$$y_c = \frac{m_{1,0}}{A} = \frac{m_{0,1}}{m_{0,0}}$$

Implementando estas ecuaciones en nuestro programa, ayudandonos de la librería de openCV, la cual nos pone a nuestra disposición diversas funciones con las que poder realizar estos cálculos, podemos obtener la posición del centro de nuestro objeto.

Este método es muy utilizado e importante para comparar dos objetos en una imagen, pero en este caso el cálculo de la posición del centro de gravedad lo utilizaremos para poder realizar el seguimiento del objeto.

Así en nuestro programa obtenemos los siguientes valores:

$$A = 4409$$
$$m_{1,0} = 1412306$$
$$m_{0,1} = 1061061$$

Obteniendo los siguientes centros de gravedad:

$$X_c = \frac{m_{1,0}}{A} = \frac{1412306}{4409} = 320{,}32 \approx 320$$
$$Y_c = \frac{m_{0,1}}{A} = \frac{1061061}{4409} = 240{,}65 \approx 240$$

En la siguiente imagen podemos ver representado por un punto el centro de gravedad de la pelota usada. Además, como podemos observar, en la esquina inferior izquierda se refleja la posición en la que se encuentra el centro del objeto, la cual tal y como hemos calculado está situada en el centro de la imagen (320,240).

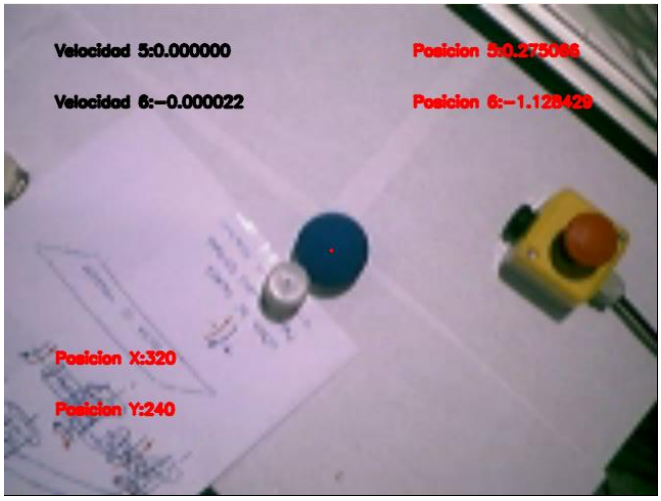


Fig. 7.16.- *Representación del centro de masas del objeto.*

7.1.6.2.- Obtención a través de la transformada de Hough.

La transformada de Hough [9] puede ser usada para determinar los parámetros de un círculo cuando un número de puntos de su perímetro son conocidos. Un círculo con radio R y centro (a, b) puede ser descrito con las siguiente ecuaciones paramétricas:

$$x = a + Rcos(\theta)$$
$$y = b + Rsin(\theta)$$

Cuando el ángulo θ recorre los 360 grados los puntos (x, y) trazan el perímetro de un círculo.

Si una imagen contiene puntos que representan el perímetro de un círculo será necesario obtener los parámetros (a, b, R) que lo describa. Usando como centro cada uno de esos puntos (x, y) conocidos, ya que son el contorno, dibuja diferentes círculos de radio R variable, hasta que con un radio R determinado aparezca un punto (a, b) en el cual coincidan 3 o más círculos. Este punto (a, b) será el centro del círculo representado por su perímetro formado por los puntos de coordenadas (x, y) y su radio será aquel R con el que se haya obtenido el centro del círculo.

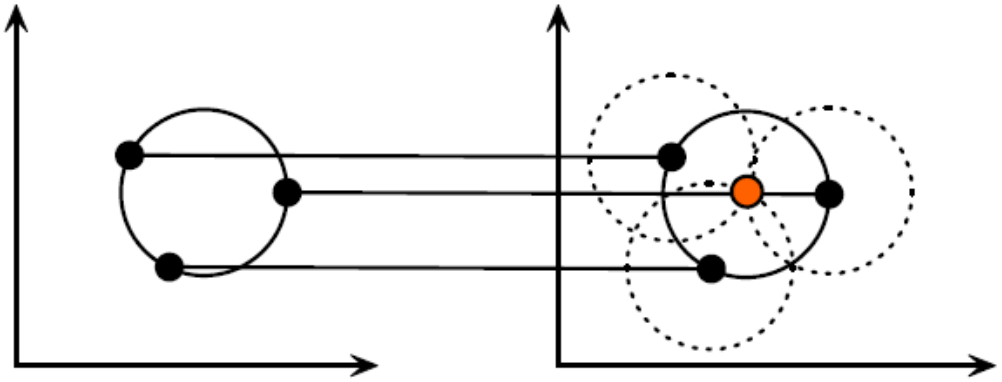


Fig 7.17.- *Obtención del centro de un círculo mediante la transformada de Hough.[9]*

Para poder realizar la transformada de Hough necesitamos previamente obtener el contorno del objeto, por lo que usaremos una función de openCV para ello, obtiendo la siguiente imagen:

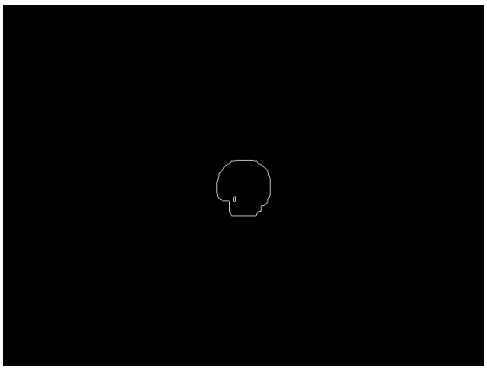
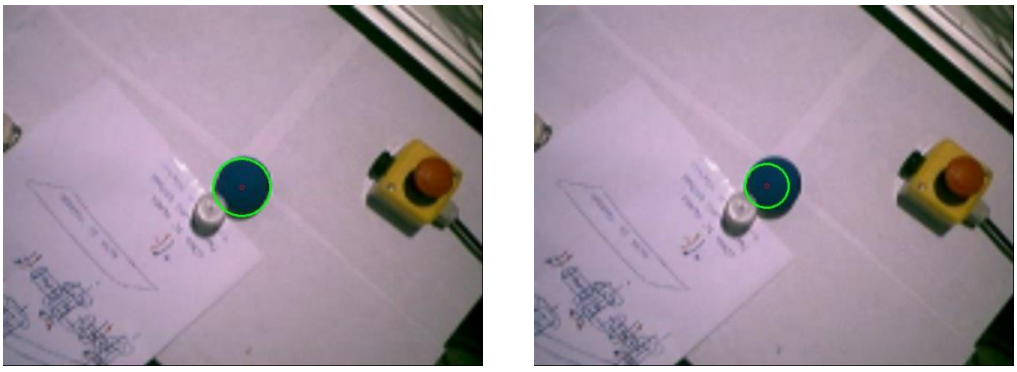


Fig.7.18.- *Representación del contorno del objeto.*

Como se puede observar en la imagen, el contorno obtenido no es del todo correcto, lo que provocará que al realizar la transformada de Hough se obtengan resultados diferentes, aunque no con gran diferencia.

Una vez realizada la transformada de Hough obtenemos a, b y R con lo que podemos dibujar el perímetro y el centro del círculo.



(a) (b)

Fig. 7.19.- *Ejemplos de obtención del centro y radio.*

Como el contorno obtenido no es estable ni "perfecto" dependiendo de los puntos que tome para hallar el centro este puede variar.

7.1.6.3.- Comparación del método de momentos con transformada de Hough.

A continuación se explicarán los motivos por los cuales se ha decidido que el método final usado para localizar el objeto sea a través del cálculo de su momento.

En las siguientes tablas se puede observar los valores que se han obtenido a lo largo de 10 frames, tanto con un método como con otro, con la cámara y el objeto inmóviles y con la intensidad de luz como única posible variante debido a que el foco puede que no proporcione la misma luz.

Valores obtenidos con momentos:

	$m_{1,0}$	$m_{0,1}$	A	X_c	Y_c	X	Y	R
1	1374823	1029730	4292	320,32	239,91	320	239	37
2	1409713	1058477	4400	320,39	240,56	320	240	37
3	1421858	1068641	4436	320,52	240,90	320	240	37
4	1412603	1062013	4410	320,31	240,80	320	240	37
5	1417418	1066136	4424	320,39	240,98	320	240	37
6	1413442	1063030	4413	320,29	240,88	320	240	37
7	1421620	1068385	4437	320,40	240,79	320	240	37
8	1419102	1066808	4430	320,33	240,81	320	240	37
9	1424265	1069567	4443	320,56	240,73	320	240	37
10	1412306	1061061	4409	320,32	240,65	320	240	37
M	1412715	1061384	4409	320,38	240,7	320	240	37

Tabla 7.20.- *Valores obtenidos mediante el cálculo por momentos.*

Posición obtenida con Hough, manteniendo inmóvil el objeto:

	X	Y	Radio
1	320	244	34
2	320	244	33
3	322	246	33
4	320	242	37
5	324	236	30
6	322	244	34
7	322	244	33
8	324	244	31
9	318	234	31
10	320	234	33
MEDIO	321,2	241,2	32,9

Tabla 7.21.- *Valores obtenidos mediante la transformada de Hough.*

Si comparamos ambas tablas podemos determinar que la obtención del centro del objeto, en caso de este ser un círculo, es más fiable con el cálculo del momento de este. Mientras que con la transformada de Hough se obtienen valores muy dispares aunque con poca diferencia con el valor real.

La obtención con el cálculo del momento sirve para cualquier forma del objeto y tiene en cuenta todo "lo que ve", por ejemplo: si pretendemos seguir una pelota azul y colocamos cerca una figura cuadrada y azul, el programa calculará el centro de gravedad de ambos y dará como resultado la posición media de ambos centros, es decir, un resultado malo. En el programa desarrollado para evitar este problema se ha colocado como exigencia que el area del objeto tenga que ser superior a 400 píxeles, por lo que todo aquello que tenga un área inferior a ese valor no será tenido en cuenta. Por el contrario, en ese mismo ejemplo la transformada de Hough sólo tendría en cuenta la pelota, por lo que nos daría un resultado bueno.

En cambio, si gran parte de la pelota esta oculta la transformada de Hough no nos la detectará como un círculo y por tanto el brazo robot no recibirá la orden de moverse. En este caso el método por momentos sí que nos proporcionará un resultado, pero dicho resultado será el centro de lo que ve, es decir, es el centro de gravedad de la parte visible y no el centro del objeto total, devolviendonos un falso resultado pero, posiblemente, cercano al real.

Además, en el caso en el que la pelota se encuentre en movimiento puede ocurrir que si esta se desplaza demasiado rápido, en la imagen captada por la cámara aparece deformada, más semejante a una elipse. En este caso la transformada de Hough no detectaría ningún círculo por lo que no podría seguirla.

Así tenemos que:

a) La transformada de Hough:
   - Ventajas:
     - Sólo sirve para objetos redondos.
   - Inconvenientes:
     - Muy inestable
     - Sólo sirve para objetos redondos.
     - Varía mucho su resultado dependiendo de los puntos que utilice.
     - En caso de oclusión parcial puede dar falsos resultados o no dar ninguno.

b) Momentos:
   - Ventajas:
     - Sirve para objetos de cualquier forma.
     - Resultados fiables.
     - En caso de oclusión parcial si da un resultado "parcialmente bueno".
   - Inconvenientes:
     - Tiene en cuenta todo "lo que ve".

7.1.7.- Incrustación de información en la imagen.

Como último paso en el tratamiento de la imagen se han incrustado unas cadenas de texto en la imagen de la cámara con información relevante. Esto nos permite, además de observar lo que se ve a través de la cámara, obtener o controlar la información que nos interesa del robot y de la imagen en tiempo real.

La información representada es:

- Velocidad a la que se mueve el motor 5, encargado del movimiento de elevación alrededor del eje X, en rad/s.
- Velocidad a la que se mueve el motor 6, encargado del movimiento horizontal alrededor del eje Y, en rad/s.
- Posición del motor 5, expresada en radianes.
- Posición del motor 6, expresada en radianes.
- Posición X en la que se encuentra el centro del objeto, en píxeles.
- Posición Y en la que se encuentra el centro del objeto, en píxeles.

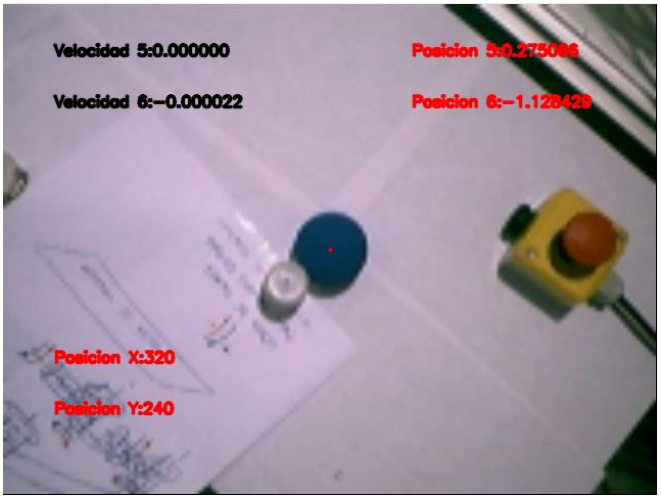


Fig. 7.22.- *Imagen final.*

7.2.- Ley de control del robot.

Como se ha mencionado, la idea principal es que el robot pueda seguir un objeto, para ello hay que lograr minimizar el error ($e(s)$) cometido entre la posición deseada y la posición actual del robot en relación al objeto de interés. Normalmente, este error viene definido a partir de las medidas tomadas directamente del plano de imagen de la cámara.

$$e(s) = s - s*$$

Siendo $s*$ nuestra posición deseada.

Comúnmente el controlador más utilizado es el control de velocidad. Para ello partimos de la velocidad de la cámara $V_c = (v_c, \omega_c)$ donde v_c es la velocidad instantánea lineal del origen de coordenadas de la cámara y ω_c es la velocidad instantánea angular del origen de coordenadas de la cámara. Así podemos definir que:

$$\dot{s} = L_s \cdot V_c$$

Siendo L_s la matriz de interacción o Jacobiana.

Relacionando ambas ecuaciones podemos relacionar la velocidad del error cometido con la velocidad de la cámara:

$$\dot{e} = L_s V_c$$

Como lo que nos interesa es saber la velocidad a la que hay que ajustar la cámara y que, además, se desea que el decrecimiento sea exponencial nos queda:

$$V_c = -\lambda \hat{L}_s^{+} e(s)$$

En la que la matriz $\hat{L}_s^+$ es la aproximación de la pseudo-inversa de Moore-Penrose de L_s.

Como se ha mencionado, esta ley de control es la más utilizada para el control visual siempre que el controlador del robot permita trabajar directamente con velocidades y valores muy cortos de tiempo, alrededor de 25-30ms. Por ello, este tipo de control no ha podido llevarse a cabo, ya que la única solución que se ha encontrado para trabajar con este método fue bloqueando y reiniciando, constantemente, los módulos del robot, además para el cálculo de la matriz de interacción es necesario conocer el parámetro Z, es decir la distancia a la que se encuentra el objeto, cosa que no se podía determinar ya que no se conoce el tamaño real del objeto o no existe un punto de referencia.

Ya que la mayoría de leyes de control que se usan se basan en iteraciones breves de cálculo de velocidad para cada uno de los módulos, en este proyecto se ha intentado hacer algo similar. Como no es posible trabajar con velocidades y tiempo a la vez, ya que la librería m5api no nos da esa posibilidad, se ha optado por realizar iteraciones en las que se calcula una próxima posición, en vez de un tiempo, y una velocidad a alcanzar en un movimiento en modo rampa.

Para el cálculo, tanto de la posición deseada como para la velocidad, se ha utilizado el error $e(s)$, quedando ambas ecuaciones, posición y velocidad, de la siguiente manera:

$$\theta_x = \frac{0.1 \cdot e_x(s)}{450}$$

$$\theta_y = -\frac{0.1 \cdot e_y(s)}{500}$$

$$\omega_x = \frac{\operatorname{atan}(e_x(s))}{f}$$

$$\omega_y = \frac{\operatorname{atan}(e_y(s))}{f}$$

Siendo θ_x, θ_y los ángulos que han de moverse los rotores, ω_x, ω_y las velocidades a alcanzar, y $e_x(s), e_y(s)$ los errores asociados al eje X e Y, es decir $e_x(s)$ es el error en píxeles en el eje horizontal y $e_y(s)$ el error en píxeles en el eje vertical. θ_x para el rotor encargado del giro alrededor del eje Y, en nuestro caso el motor número 6, y θ_y para el rotor encargado del giro alrededor del eje X, motor número 5 del brazo modular.

En el caso de θ_y el incremento está multiplicado por -1 debido a que dicho motor gira en sentido contrario al rotor número 6, además el motor 5 es más grande que el motor 6 y se encuentra distanciado de la cámara por lo que su incremento se ha considerado que debe ser menor.

Así la ley de control se resume en:

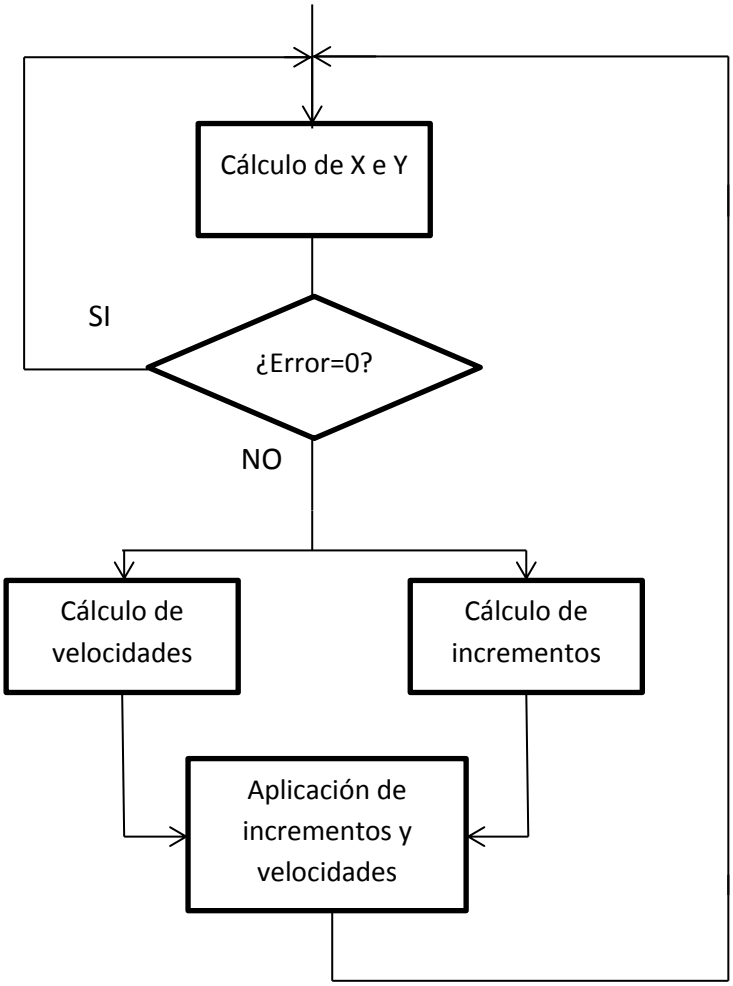


Fig. 7.23.- *Esquema del sistema.*

7.2.1.- Análisis del comportamiento del sistema de IBVS.

Los experimentos han sido realizados tanto con un objeto inmóvil como en movimiento. A continuación se muestran diferentes datos obtenidos para 3 tipos de movimientos: movimiento horizontal, vertical y ambos.

7.2.1.1.- Objeto estático.

En este caso se pretende mantener el objeto en una posición fija diferente a la deseada para poder observar el comportamiento del sistema. Así, y como ya se ha

explicado anteriormente, el objetivo del proyecto es llevar el centro de masas del objeto a la posición deseada, es decir el centro de la imagen.

Este experimento ha sido divido en dos tipos de control: control azimuth, por el cual solo se desplazará un motor y sólo tendremos control sobre la variable x, y control azimuth y elevación, en el que usaremos únicamente dos motores para poder mantener el objeto centrado.

7.2.1.1.1.- Control azimuth.

Para este tipo de control sólo se utilizará el motor número 6 del robot, ya que nos permite generar un movimiento en horizontal en la imagen de la cámara, a lo largo del eje x.

En la siguiente imagen se muestra el centro de masas del objeto, marcado como una cruz roja, en una posición inicial y, con una cruz verde, la posición deseada.

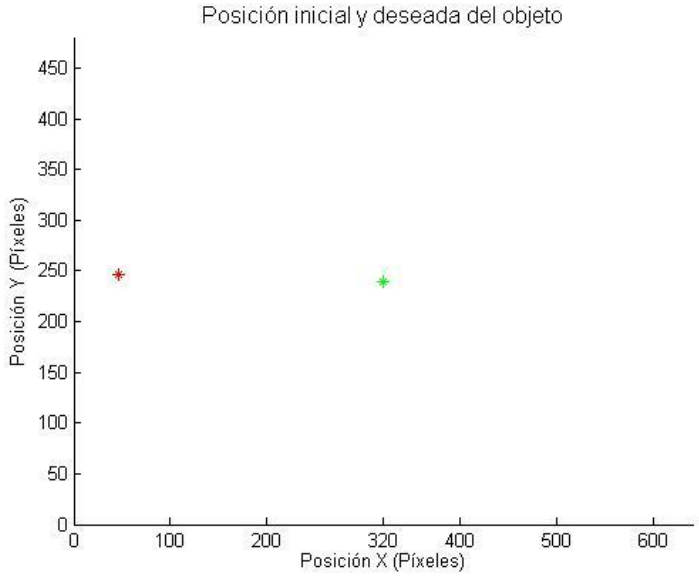


Fig. 7.23.- *Posición inicial y deseada del objeto.*

Con la información de ambas posiciones y la reiteración del cálculo de la posición de la pelota, el programa realiza los cálculos de velocidad e incremento de posición aplicando las fórmulas explicadas en el apartado anterior y provocando el movimiento del robot. De esta forma, hemos obtenido en el experimento la siguiente trayectoria seguida en el plano de la imagen.

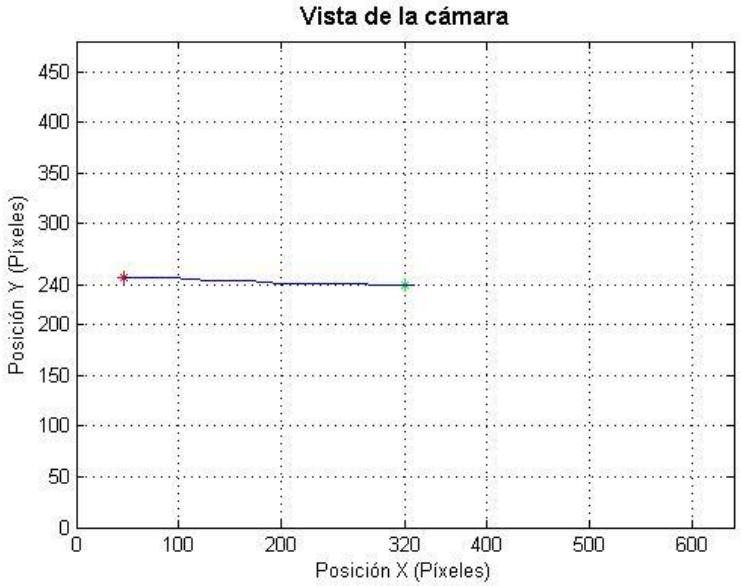


Fig. 7.24.- *Movimiento realizado por cámara.*

A pesar de que el motor encargado del movimiento sobre el eje Y se encontraba desactivado podemos apreciar que se ha producido un desplazamiento sobre este eje. Dicho desplazamiento es debido, principalmente, a dos razones: el plano sobre el que esta el objeto y la cámara no están en paralelo y un posible error en el cálculo del centro.

En la siguiente imagen podemos ver con mayor detalle el movimiento del centro del objeto sobre el eje X y el error, medido en píxeles, que presenta el sistema.

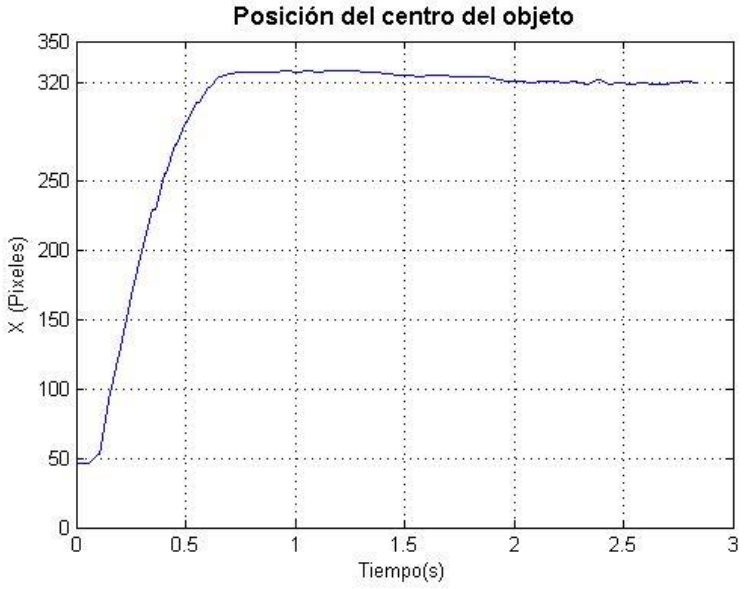


Fig. 7.25.- *Representación de la posición del objeto.*

A continuación podemos observar la variación de la velocidad con respecto al tiempo.

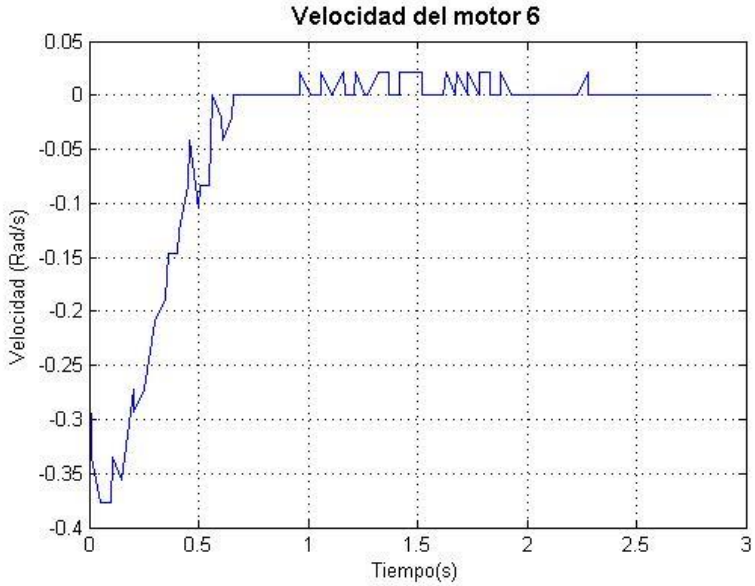


Fig. 7.26.- *Variación de la velocidad durante el movimiento.*

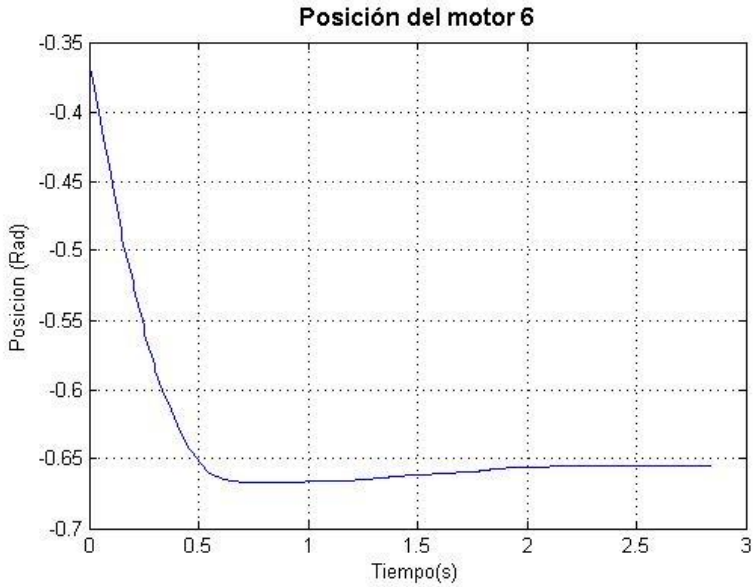


Fig. 7.27.- *Variación de la posición del motor.*

Por último, en la siguiente figura se muestra la variación del error con respecto al tiempo.

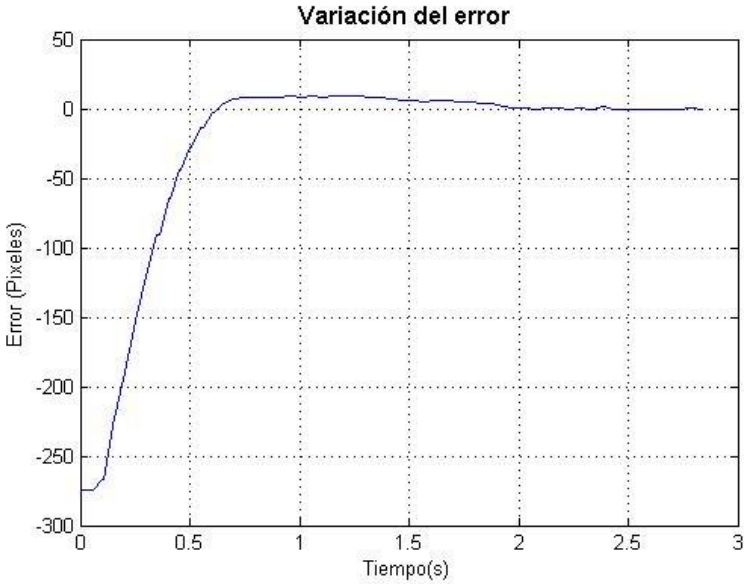


Fig. 4.28.- *Variación del error.*

Como se puede observar la variación del error presenta una pendiente que disminuye según se aproxima al 0. Cabe destacar la velocidad con la que se reduce el error, ya que si analizamos la variación del error tenemos:

$$e_{xMAX} = 274$$

$$iteración\ 12 \begin{Bmatrix} t = 480ms \\ e_x = 28 \end{Bmatrix}$$

Como vemos en apenas medio segundo conseguimos que el error sea reducido en un 89,78%. Mientras que el sistema logra una posición estable, aproximadamente a los 2 segundos del comienzo de la detección.

7.2.1.1.2.- Control azimuth y elevación.

Para este utilizamos dos motores del robot: el motor número 5, encargado del movimiento de elevación, y el motor número 6, para el movimiento azimuth.

En la siguiente imagen se puede ver marcado el centro de masas del objeto con una cruz roja, en una posición inicial, y con un signo positivo verde, la posición deseada.

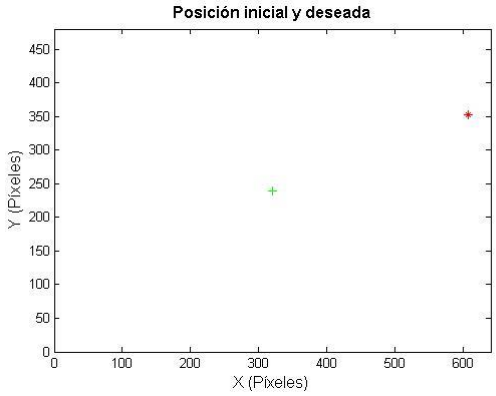


Fig. 7.29.- *Posición inicial y deseada del objeto para el control azimuth y elevación.*

Partiendo de esa posición inicial y ejecutándose nuestro programa, al almacenar los datos de posición y reflejarlos en una gráfica hemos obtenido la siguiente imagen.

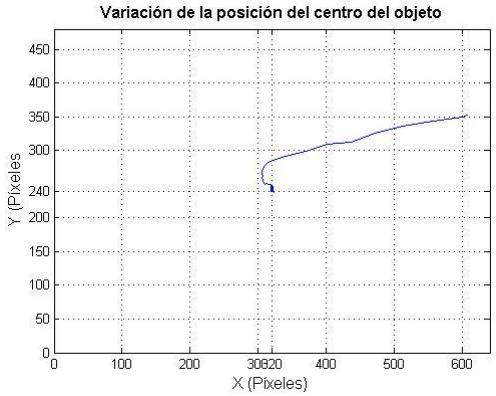


Fig. 7.30.- *Representación del movimiento realizado por la cámara.*

En la imagen podemos observar como no se realiza un movimiento rectilíneo desde el punto inicial y el deseado, lo cual sería lo ideal, sino que se ve como la posición X deseada es alcanzada un tiempo antes que la posición Y, por lo que se produce ese movimiento en curva. Esto es debido a que en el movimiento de un eje no se tiene en cuenta el movimiento y el desplazamiento del otro. Si analizamos por separado la variación de la posición del centro del objeto con respecto al eje X y al eje Y obtenemos las siguientes gráficas:

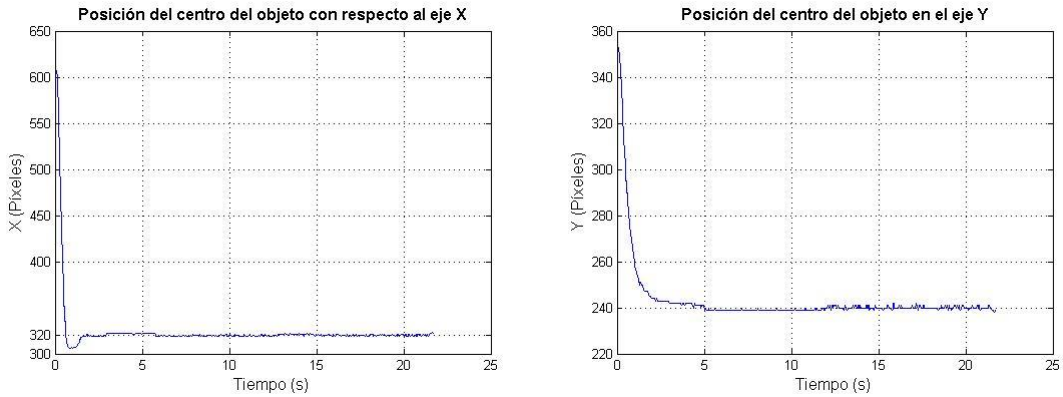


Fig. 7.31.- *Representación de la posición del objeto. Izquierda: respecto al eje X. Derecha: respecto al eje Y.*

Como podemos observar, el movimiento de elevación es más lento que el movimiento en horizontal.

En las siguientes imágenes observamos la variación de la velocidad de los motores y la variación del error en ambos ejes.

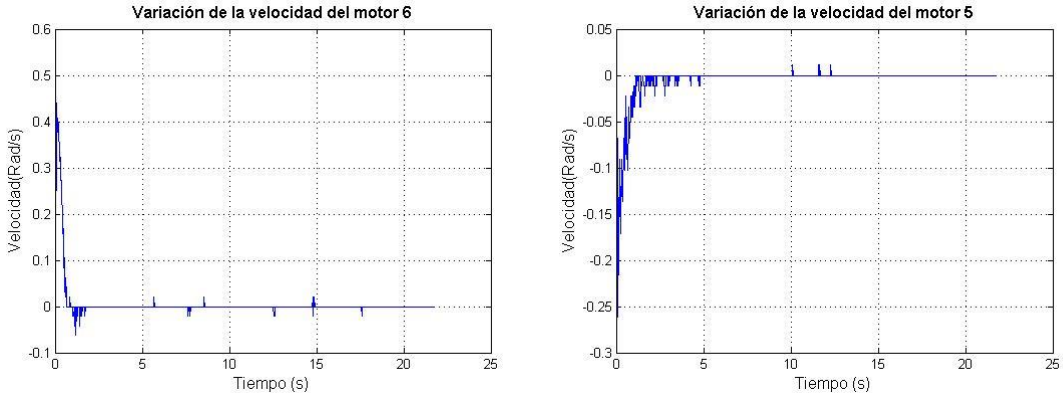


Fig. 7.32.- *Variación de las velocidades de los motores.*

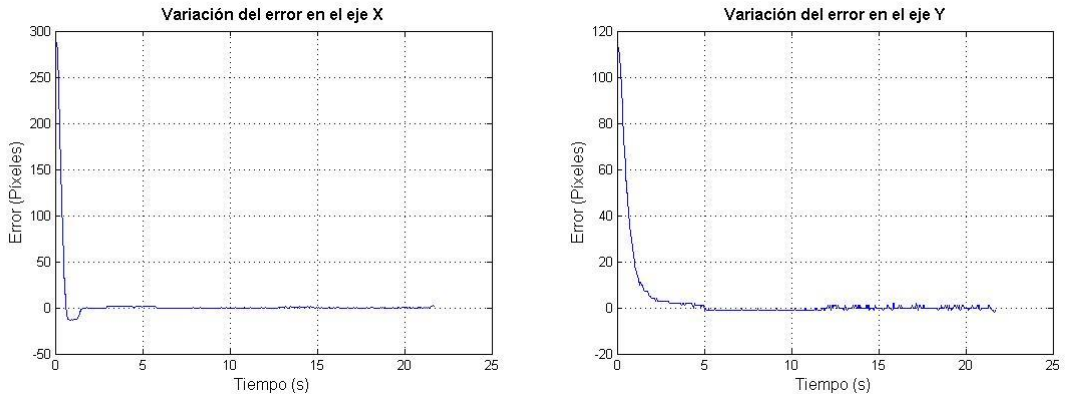


Fig. 7.33.- *Variación del error en el eje X y en el eje Y.*

7.3.- Interfaz gráfica.

Para esta parte del proyecto podría haberse usado algún otro lenguaje de programación como puede ser Qt o GTK para el desarrollo de una interfaz más elaborada y más "visible". Finalmente, para evitar tener que modificar gran parte del código se ha aprovechado la librería HighGUI de OpenCV para desarrollarla.

Dentro de esta "interfaz" en la ventana "cámara" se han incluido una serie de barra de desplazamiento para el control del umbralizado de la imagen HSV con las que se pueden modificar los valores mínimos y máximos de H, S y V, para poder ajustar estos valores a los que se consideren los más óptimos, aunque vienen predefinidos valores para los colores: rojo, azul, verde y amarillo, pudiendo elegir el usuario alguno de estos antes de iniciar el programa.

Además en esta misma ventana, se ha colocado una barra de desplazamiento que hace función de botón, ya que varía entre 0 y 1, y nos permite iniciar o pausar el movimiento del robot sin detener el tratamiento de la imagen. Esta posibilidad de pausarlo nos permite, por ejemplo, que en caso de no estar fijado correctamente el umbralizado podamos modificarlo a lo que buscamos sin que se mueva el robot a una posición no deseada.

Para evitar que este ajuste de umbralizado sea "a ciegas" se han dejado visibles las 4 ventanas que nos interesa para poder ajustarlo correctamente: "matiz", "saturación", "valor" y "filtrada". Por lo que podemos controlar el umbralizado que estamos realizando.

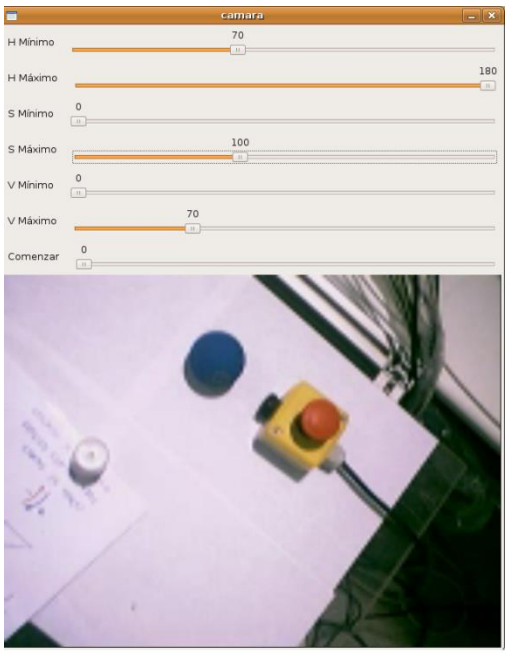


Fig. 7.34.- *Interfaz final*

8.- Conclusiones y trabajos futuros.

En este proyecto se ha realizado un estudio del visual servoing para el seguimiento de objetos. Para este objetivo nos hemos centrado en el Image-Based Visual Servoing o control visual basado en imagen. El hecho de utilizar esta técnica es debido a que es la más utilizada en el ámbito industrial y porque al realizarse el analisis en el espacio de trabajo de la imagen evitamos los problemas derivados de una mala calibración.

Como ya se ha expresado en el capítulo 7 de control visual, el proyecto se puede dividir en dos grandes campos: tratamiento de imágenes y control. Por ello, este proyecto ha permitido el estudio, más a fondo, y la aplicación de diversas técnicas del campo de visión por computador y de la robótica, estudiadas previamente en las asignaturas de “Robótica” y de “Visión Artificial”, de forma experimental. Además del desarrollo del programa de control visual en sí, también se ha trabajado en el análisis de la cinemática del robot.

A lo largo del proyecto nos hemos encontrado con infinidad de problemas: dificultades en la instalación o actualización de librerías necesarias en el sistema operativo, cambio imprevisto de cámara, imposibilidad de aplicación un control de velocidad; lo cual no nos ha impedido llevar a cabo nuestra tarea, intentando encontrar la mejor solución a cada uno de ellos.

Como trabajos futuros, podemos destacar la implementación de la cinemática inversa del robot al programa desarrollado y a la utilización de los seis motores del robot conjuntamente, además del desarrollo de una interfaz más “visible”.

En definitiva, ha sido un trabajo muy importante, permitienme adquirir una gran cantidad de conocimientos y de aptitudes en el campo de la investigación que no se consiguen en las clases teóricas.

ANEXO A - Código del programa desarrollado con el cálculo por momentos.

ANTES DE EJECUTAR EL PROGRAMA HAY QUE EJECUTAR EL SCRIPT
crear_entradas_dev.sh

```
////////////////////////////////////CARGA DE LIBRERÍAS/////////////////////////////////////

#include </usr/local/include/opencv/cv.h>
#include </usr/local/include/opencv/highgui.h>
#include "iostream"
#include </home/robotnik/modular/include/m5apiw32.h>
#include "math.h"
#include "time.h"
//#include </usr/local/include/opencv/cvaux.h>
//#include </usr/local/include/opencv/cxcore.h>
////////////////////////////////////////////////////////////////////////////////////////

FILE *doc;              //Declaramos el tipo de archivo a utilizar (Documento)

int h_min=0;            //Valor mínimo de hue (Matiz)
int s_min=0;            //Valor mínimo de saturatíon (Saturación)
int v_min=0;            //Valor mínimo de value (Valor)

int h_max=180;          //Valor mínimo de hue (Matiz)
int s_max=255;          //Valor mínimo de saturatíon (Saturación)
int v_max=255;          //Valor mínimo de value (Valor)
int on=0;               //Variable utilizada como "Run-Stop" (0=Stop, 1=Run)

//////////////////DECLARACIÓN DE LA FUNCIÓN RGB-HSV-THRESHOLD////////////////////
/* Esta función se encarga de crear una imagen binaria en la cual solo se observen los píxeles
con los valores HSV deseados*/

IplImage* ThresholdedColor(IplImage* hsv_frame)
        {
        //Declaración de la imagen thresholded (Binaria)
        IplImage* thresholded=cvCreateImage(cvGetSize(hsv_frame),IPL_DEPTH_8U, 1);
        cvInRangeS(hsv_frame, cvScalar(h_min,s_min,v_min), cvScalar(h_max,s_max,v_max),
        thresholded);    //HSV Variable según los valores de las barras de desplazamiento, crea
        una imagen binaria con el color deseado

        return thresholded;
        }
////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////DECLARACIÓN DE LA FUNCIÓN AJUSTE HSV//////////////////////////
/*Esta función crea 6 barras para modificar los valores HSV según convenga y una barra para
"Run-Stop"*/

void AjusteHSV()
        {
        cvNamedWindow("camara");

        cvCreateTrackbar("H Mínimo", "camara", &h_min, 180, NULL);
        cvCreateTrackbar("H Máximo", "camara", &h_max, 180, NULL);

        cvCreateTrackbar("S Mínimo", "camara", &s_min, 255, NULL);
        cvCreateTrackbar("S Máximo", "camara", &s_max, 255, NULL);

        cvCreateTrackbar("V Mínimo", "camara", &v_min, 255, NULL);
        cvCreateTrackbar("V Máximo", "camara", &v_max, 255, NULL);

        cvCreateTrackbar("Comenzar", "camara", &on, 1, NULL);
        }

////////////////////////////INICIO DEL PROGRAMA////////////////////////////////////

int main()
{
float inicio,ahora, reloj;  //Declaramos las variables usadas para indicar el tiempo

//Cargamos los valores intrínsecos y de distorsión calculados previamente con el programa
"Calibrar"
CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics_resized.xml");
CvMat *distortion = (CvMat*)cvLoad("Distortion_resized.xml");

doc=fopen("prueba.txt","w");   //Abrimos el documento en el que escribiremos los datos
fprintf(doc,"Reloj \t\tX \tY \tVelocidad 5 \t Velocidad 6 \t Posicion 5 \t Posicion 6 \n");

//Variables robot
int ret=0;          //Variable utilizada para saber si el puerto de comunicaciones está disponible
int dev = 0;                                //Variable para indicar el dispositivo a usar.
float pos5, pos6;                           //Posición deseada para los motores 5 (Eje Y) y 6 (Eje X)
float acc = 2.7925;                         //Aceleración de los motores (Rad/s^2)
float incremento= 0.1;              //Incremento usado para el cálculo del ángulo a moverse (Rad)
char strInitString[] = "ESD:0,1000";        //Puerto CAN a utilizar y baudrate o ancho de banda.
```

```
float pos_act5, pos_act6;          //Variables en la que almacenamos la posición actual
                                   para los motores 5 y 6.
float get_vel5, get_vel6;          //Variables en la que almacenamos la velocidad actual
                                   de los motores 5 y 6.
float f=3.85;                      //Distancia focal
float vel_x;                       //Velocidad para el motor 6
float vel_y;                       //Velocidad para el motor 5
float incremento_x;                //Incremento de posición para el motor 6
float incremento_y;                //Incremento de posición para el motor 5
CvCapture* capture=cvCaptureFromCAM(0);   //Función utilizada para obtener la imagen de
                                          la cámara
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH,640);
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT,480);
IplImage* img;                     //Declaración de la variable img en la que almacenamos los
                                   frames
img=cvQueryFrame(capture);         //Función utilizada para tomar un frame de la imagen y
                                   almacenarlo en img.
//Variables utilizada para almacenar las 2 últimas posiciones del objeto.
//Declaradas como -1 para evitar movimientos indebidos al comienzo
int lastX=-1;
int lastX2=-1;
int lastY2=-1;
int lastY=-1;

int x,y;                           //Posición actual del objeto a seguir
int a=0;                   //Variable utilizada para realizar un movimiento a la última posición
                           del objeto una vez perdido de vista. Si a=0 se ha perdido de vista.
CvFont font;                       //Declaración de la fuente de la letra
cvInitFont( &font, CV_FONT_VECTOR0, 0.5, 0.5, 0, 2.0, CV_AA);   //Función para definir
                                                                 el tipo de letra.
char vel5[255],vel6[255],postxt5[255],postxt6[255],posx[255],posy[255];   //Cadenas de
                                                                          caracteres
                                                                          utilizadas para
                                                                          mostrar
                                                                          información en
                                                                          la imagen
char color;//Cadena de caracteres utilizada para la selección de valores predefinidos de hsv
double area;

//////////CREACIÓN DE LAS VENTANAS PARA LAS IMÁGENES//////////////////////
cvNamedWindow("camara",CV_WINDOW_AUTOSIZE);   //Ventana en la que se muestra
la imagen captada por la cámara
cvNamedWindow("range",CV_WINDOW_AUTOSIZE);  //Ventana en la que se muestra la
imagen binaria tras la apertura y cierre. Esta imagen es la tomada para el cálculo del momento
```

```
////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////DECLARACIÓN DE IMÁGENES////////////////////////////////////
IplImage* hsv_frame = cvCreateImage(cvGetSize(img), 8,3);      //Frame de la imagen en HSV
IplImage* thresholded = cvCreateImage(cvGetSize(img), 8,1);

//Construcción del mapa indistorsionado
IplImage* mapx = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
IplImage* mapy = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
cvInitUndistortMap(intrinsic,distortion,mapx,mapy);


//////////////APERTURA DEL PUERTO DE COMUNICACIONES/////////////////////
ret=PCube_openDevice(&dev,strInitString);
if (ret==-205)
{
printf("Puerto de comunicación cerrado.\n Es necesario ejecutar el archivo
'crear_entradas_dev.sh', localizado en la carpeta Robotnik");
}
else if (ret==0)
{
printf("Puerto de comunicación abierto.\n\n\n");
}
else
{
printf("Error desconocido.");
}
PCube_resetModule( dev, 5 );            //Reinicio del módulo 5 por si se encuentra bloqueado
PCube_resetModule( dev, 6 );            //Reinicio del módulo 6 por si se encuentra bloqueado
////////////////////////////////////////////////////////////////////////////////////////////////


/////////FUNCIÓN UTILIZADA PARA LA ELECCIÓN DE VALORES PREDEFINIDOS//////////////

printf("¿Que color desea?\nPulse 'r' para rojo.\nPulse 'g' para verde.\nPulse 'b' para azul.\n
Pulse 'y' para amarillo.\n");
scanf("%s",&color);

switch(color)
{
case 'r':                           //Caso para el color rojo

        h_min=0;
        s_min=100;
        v_min=0;
```

```
        h_max=10;
        s_max=255;
        v_max=255;
        break;

case 'g':                       //Caso para el color verde

        h_min=0;
        s_min=0;
        v_min=0;

        h_max=180;
        s_max=255;
        v_max=255;
        break;

case 'b':                       //Caso para el color azul

        h_min=90;
        s_min=110;
        v_min=110;

        h_max=120;
        s_max=255;
        v_max=255;
        break;

case 'y':                       //Caso para el color amarillo

        h_min=20;
        s_min=100;
        v_min=100;

        h_max=30;
        s_max=255;
        v_max=255;
        break;
}
////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////COMIENZO DEL VIDEO/////////////////////////////////////
inicio=clock();         //Guardamos los ciclos de reloj
while(1)
{
```

```
//////////////////Almacenamiento de la información en un archivo de texto//////////////////
PCube_getPos(dev, 5,&pos_act5);
PCube_getPos(dev, 6,&pos_act6);
PCube_getVel(dev,5,&get_vel5);
PCube_getVel(dev,6,&get_vel6);
ahora=clock();
reloj=(ahora-inicio)/(double)CLOCKS_PER_SEC;
fprintf(doc,"%f \t",reloj);
if(area>200)     //Utilizamos un condicional para evitar que cuando almacene los datos y no
{                //haya ningún objeto almacene posiciones imposibles.
fprintf(doc,"%d \t",x);
fprintf(doc,"%d \t",y);
}
else
{
fprintf(doc,"000 \t");
fprintf(doc,"000 \t");
}
fprintf(doc,"%f \t",get_vel5);
fprintf(doc,"%f \t",get_vel6);
fprintf(doc,"%f \t",pos_act5);
fprintf(doc,"%f \n\n",pos_act6);
////////////////////////////////////////////////////////////////////////////////////////////
IplImage* img = cvQueryFrame(capture);
cvRemap( img, img, mapx, mapy ); //Aplicación del mapa de distorsión a la imagen
////ELECCION DE COLOR////
AjusteHSV();          //Función creada para el ajuste de HSV, declarada al comienzo del
                      programa .
cvCvtColor(img, hsv_frame, CV_BGR2HSV);                    //Conversión de BGR a HSV
IplImage* thresholded = ThresholdedColor(hsv_frame);       //Binariazación de la imagen
cvShowImage("camara",img);                                 //Muestreo de la imagen RGB

//////////////////DECLARACIÓN DE KERNELS PARA APERTURA ////////////////////////
IplConvKernel *se11 = cvCreateStructuringElementEx(11, 11, 5,  5,  CV_SHAPE_RECT, NULL);
cvMorphologyEx(thresholded, thresholded,0, se11,CV_MOP_OPEN,1); //Aplicación de una
apertura (erosión-dilatación)
cvReleaseStructuringElement(&se11);          //Deasignación del espacio de memoria
cvSmooth(thresholded,thresholded,CV_GAUSSIAN,5,5);
cvShowImage("range",thresholded);    //Muestre de la imagen Binaria tras aplicar apertura
////////////////////////////////////////////////////////////////////////////////////////////


//////////////////////CÁLCULO DEL MOMENTO DEL OBJETO//////////////////////////////
```

```
        /* El método de cálculo aquí utilizado se encuentra explicado en la memoria*/
if(on==1)
{

CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
cvMoments(thresholded, moments, 1);
double moment10 = cvGetSpatialMoment(moments, 1, 0);      //Momento de orden 1
double moment01 = cvGetSpatialMoment(moments, 0, 1);      //Momento de orden 1
area = cvGetCentralMoment(moments, 0, 0);   //Momento de orden 0; cálculo del area del objeto


if(area>200)  //Si el área es mayor a 200 píxeles calcula su momento
        {
        //Cálculo de la posición del objeto. Al dividir la posición del centro de gravedad entre su área obtenemos el centro del objeto.
                x = moment10/area;
                y = moment01/area;

                if(lastX>=0 && lastY>=0 && x>=0 && y>=0)
                {
                //cvLine(resized,cvPoint(x,y),cvPoint(x,y),cvScalar(0,0,255),4);   En caso de solo querer ver un único punto
                cvLine(img,cvPoint(x, y), cvPoint(lastX, lastY), cvScalar(0,0,255), 4);
                //Dibujamos una línea roja que representa la trayectoria del objeto
                }
                 //Almacenamiento de las últimas dos posiciones
                lastX2=lastX;
                lastY2=lastY;
                lastX = x;
                lastY = y;
                a++; //Le sumamos 1 a "a" para saber que hay un objeto que seguir
        }

Int radio;
Radio=sqrt(area/M_PI);          //Calculamos el radio de la pelota
cvCircle( img, cvPoint(x,y) , radio, CV_RGB(0,0,255), 3, 8, 0 );//Dibujamos un círculo alrededor

free(moments);          //Liberamos el espacio
////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////MUESTREO DE IMÁGENES/////////////////////////////////////////
cvShowImage("camara",img);  //Se vuelve a mostrar la imagen para que aparezca la línea dibujada anteriormente
```

```
////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////COMIENZO DEL MOVIMIENTO/////////////////////////////////////
if (area>200)
{

////////////CÁLCULO DE LAS VELOCIDADES Y POSICIÓN PARA LOS MOTORES/////////////////
vel_x=atan((x-320)/f);  //Calculo de la velocidad a alcanzar dependiendo de la posición con respecto a la cámara
vel_y=atan((y-240)/f);
incremento_x=incremento*(x-320)/450; //Cuando el motor 6 deba moverse a la izquierda el incremento será negativo
incremento_y=incremento*(240-y)/500; //Cuando el motor 5 deba moverse hacia arriba el incremento será positivo

PCube_getPos(dev, 6,&pos_act6);          //Obtenemos la posición actual del motor 6
pos6=pos_act6+incremento_x;              //Cálculo de la posición a la que ha de moverse

PCube_getPos(dev, 5,&pos_act5);          //Obtenemos la posición actual del motor 5
pos5=pos_act5+incremento_y;              //Cálculo de la posición a la que ha de moverse
////////////////////////////////////////////////////////////////////////////////////////////

if (vel_x==0)  //Si la velocidad calculada es 0, indica que la posición para el eje X es la correcta
        {}
        else if (vel_x<0) //Si es menor de 0, ha de moverse hacia la izquierda
                {
                PCube_moveRamp( dev, 6, pos6, -vel_x, acc );  //Función para la realización de un movimiento en modo Rampa a la izquierda...
                PCube_getVel(dev,6,&get_vel6);
                }
                else
                        {
                        PCube_moveRamp( dev, 6, pos6, vel_x, acc );  //Movimiento a la derecha
                        PCube_getVel(dev,6,&get_vel6);
                        }
if (vel_y==0)  //Si la velocidad calculada es 0, indica que la posición para el eje Y es la correcta
        {}
        else if (vel_y<0)  //Movimiento hacia arriba
                {
                PCube_moveRamp( dev, 5, pos5, -vel_y, acc );
                PCube_getVel(dev,5,&get_vel5);
                }
                else   //Movimiento hacia abajo
```

```
                        {
                        PCube_moveRamp( dev, 5, pos5, vel_y, acc );
                        PCube_getVel(dev,5,&get_vel5);
                        }

}
//////////////////////FIN DEL MOVIMIENTO////////////////////////////////////


////////////MOVIMIENTO EN CASO DE PÉRDIDA DE OBJETO/////////////////
else if(a!=0) //Si no usamos "a" al iniciar el programa si no existe ningún objeto a seguir, se realizaría un movimiento imposible
{
a=0;

incremento_x=incremento*(lastX2-lastX)/100;  //Como el movimiento ha debido de ser brusco el incremento es  5 veces mayor
incremento_y=incremento*(lastY2-lastY)/100;
vel_x=1.74; //Velocidad máxima de los motores
vel_y=1.74;

PCube_getPos(dev, 6,&pos_act6);
pos6=pos_act6-incremento_x;

PCube_getPos(dev, 5,&pos_act5);
pos5=pos_act5+incremento_y;


if (lastX==0)
        {}
        else if (lastX<320)
                {
                PCube_moveRamp( dev, 6, pos6, vel_x, acc );
                PCube_getVel(dev,6,&get_vel6);
                }
                else
                        {
                        PCube_moveRamp( dev, 6, pos6, vel_x, acc );
                        PCube_getVel(dev,6,&get_vel6);
                        }
```

```
if (lastY==0)
        {}
        else if (lastY>240)
                {
                PCube_moveRamp( dev, 5, pos5, vel_y, acc );
                PCube_getVel(dev,5,&get_vel5);
                }
                else
                        {
                        PCube_moveRamp( dev, 5, pos5, vel_y, acc );
                        PCube_getVel(dev,5,&get_vel5);
                        }


}

////////////////FIN DEL MOVIMIENTO EN CASO DE PÉRDIDA//////////////////////////

//////////////////MUESTREO DE POSICIÓN Y VELOCIDAD EN LA IMAGEN//////////////////
//Creación de cadena de caracteres
sprintf(vel5,"Velocidad 5:%f",get_vel5);
sprintf(postxt5,"Posicion 5:%f",pos_act5);
sprintf(vel6,"Velocidad 6:%f",get_vel6);
sprintf(postxt6,"Posicion 6:%f",pos_act6);
sprintf(posx,"Posicion X:%d",x);
sprintf(posy,"Posicion Y:%d",y);
ahora=clock();
reloj=(ahora-inicio)/(double)CLOCKS_PER_SEC;
//Escribimos en el archivo de texto
fprintf(doc,"%f \t",reloj);
if(area>200)
{
fprintf(doc,"%d \t",x);
fprintf(doc,"%d \t",y);
}
else
{
fprintf(doc,"000 \t");
fprintf(doc,"000 \t");
}
fprintf(doc,"%f \t",get_vel5);
fprintf(doc,"%f \t",get_vel6);
fprintf(doc,"%f \t",pos_act5);
fprintf(doc,"%f \n\n",pos_act6);
```

```
//Escritura sobre la imagen
cvPutText(img, vel5, cvPoint(50,50), &font,CV_RGB(0,0,0) );
cvPutText(img, vel6, cvPoint(50,100), &font,CV_RGB(0,0,0) );
cvPutText(img, postxt5, cvPoint(400,50), &font,CV_RGB(255,0,0) );
cvPutText(img, postxt6, cvPoint(400,100), &font,CV_RGB(255,0,0) );
cvPutText(img, posx, cvPoint(50,350), &font,CV_RGB(255,0,0) );
cvPutText(img, posy, cvPoint(50,400), &font,CV_RGB(255,0,0) );


//Muestra de la imagen modificada con el texto
cvShowImage("camara",img);

//////////////////////FINAL MUESTREO//////////////////////////////
} //Fin (on==1) Si se vuelve a poner a 0 se detendrá el movimiento del robot pero no la visualización del video
char w =cvWaitKey(10);          //Espera 10 ms, en caso de pulsarse Esc, se cierra el programa
if (w==27) //27=Esc en código ASCII
break;
} //Fin while(1)

cvReleaseCapture(&capture);          //Liberación de memoria
cvDestroyAllWindows();               //Cierre de todas las ventanas creadas
fclose(doc);                         //Cierre del documento de texto
return 0;
}                                    //Fin del programa
```

ANEXO B - Código del programa desarrollado con la transformada de Hough.

ANTES DE EJECUTAR EL PROGRAMA HAY QUE EJECUTAR EL SCRIPT
crear_entradas_dev.sh

```
////////////////////////////////////CARGA DE LIBRERÍAS////////////////////////////////////

#include </usr/local/include/opencv/cv.h>
#include </usr/local/include/opencv/highgui.h>
#include "iostream"
#include </home/robotnik/modular/include/m5apiw32.h>
#include "math.h"
#include "time.h"
//#include </usr/local/include/opencv/cvaux.h>
//#include </usr/local/include/opencv/cxcore.h>
////////////////////////////////////////////////////////////////////////////////////////

FILE *doc;              //Declaramos el tipo de archivo a utilizar (Documento)

int h_min=0;            //Valor mínimo de hue (Matiz)
int s_min=0;            //Valor mínimo de saturatíon (Saturación)
int v_min=0;            //Valor mínimo de value (Valor)

int h_max=180;          //Valor mínimo de hue (Matiz)
int s_max=255;          //Valor mínimo de saturatíon (Saturación)
int v_max=255;          //Valor mínimo de value (Valor)
int on=0;               //Variable utilizada como "Run-Stop" (0=Stop, 1=Run)

//////////////////DECLARACIÓN DE LA FUNCIÓN RGB-HSV-THRESHOLD////////////////////
/* Esta función se encarga de crear una imagen binaria en la cual solo se observen los píxeles
con los valores HSV deseados*/

IplImage* ThresholdedColor(IplImage* hsv_frame)
        {
        //Declaración de la imagen thresholded (Binaria)
        IplImage* thresholded=cvCreateImage(cvGetSize(hsv_frame),IPL_DEPTH_8U, 1);
        cvInRangeS(hsv_frame, cvScalar(h_min,s_min,v_min), cvScalar(h_max,s_max,v_max),
        thresholded);    //HSV Variable según los valores de las barras de desplazamiento, crea
        una imagen binaria con el color deseado

        return thresholded;
        }
////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////DECLARACIÓN DE LA FUNCIÓN AJUSTE HSV//////////////////////////
/*Esta función crea 6 barras para modificar los valores HSV según convenga y una barra para
"Run-Stop"*/

void AjusteHSV()
        {
        cvNamedWindow("camara");

        cvCreateTrackbar("H Mínimo", "camara", &h_min, 180, NULL);
        cvCreateTrackbar("H Máximo", "camara", &h_max, 180, NULL);

        cvCreateTrackbar("S Mínimo", "camara", &s_min, 255, NULL);
        cvCreateTrackbar("S Máximo", "camara", &s_max, 255, NULL);

        cvCreateTrackbar("V Mínimo", "camara", &v_min, 255, NULL);
        cvCreateTrackbar("V Máximo", "camara", &v_max, 255, NULL);

        cvCreateTrackbar("Comenzar", "camara", &on, 1, NULL);
        }

////////////////////////////INICIO DEL PROGRAMA////////////////////////////////////

int main()
{
float inicio,ahora, reloj;  //Declaramos las variables usadas para indicar el tiempo

//Cargamos los valores intrínsecos y de distorsión calculados previamente con el programa
"Calibrar"
CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics_resized.xml");
CvMat *distortion = (CvMat*)cvLoad("Distortion_resized.xml");

doc=fopen("prueba.txt","w");   //Abrimos el documento en el que escribiremos los datos
fprintf(doc,"Reloj \t\tX \tY \tVelocidad 5 \t Velocidad 6 \t Posicion 5 \t Posicion 6 \n");

//Variables robot
int ret=0;          //Variable utilizada para saber si el puerto de comunicaciones está disponible
int dev = 0;                            //Variable para indicar el dispositivo a usar.
float pos5, pos6;                       //Posición deseada para los motores 5 (Eje Y) y 6 (Eje X)
float acc = 2.7925;                     //Aceleración de los motores (Rad/s^2)
float incremento= 0.1;              //Incremento usado para el cálculo del ángulo a moverse (Rad)
char strInitString[] = "ESD:0,1000";    //Puerto CAN a utilizar y baudrate o ancho de banda.
float pos_act5, pos_act6;               //Variables en la que almacenamos la posición actual
                                        para los motores 5 y 6.
float get_vel5, get_vel6;               //Variables en la que almacenamos la velocidad actual
                                        de los motores 5 y 6.
```

```
float vel_x;                    //Velocidad para el motor 6
float vel_y;                    //Velocidad para el motor 5
float incremento_x;             //Incremento de posición para el motor 6
float incremento_y;             //Incremento de posición para el motor 5
CvCapture* capture=cvCaptureFromCAM(0);   //Función utilizada para obtener la imagen de
                                          la cámara
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_WIDTH,640);
cvSetCaptureProperty(capture,CV_CAP_PROP_FRAME_HEIGHT,480);
IplImage* img;          //Declaración de la variable img en la que almacenamos los
                        frames
img=cvQueryFrame(capture);      //Función utilizada para tomar un frame de la imagen y
                                almacenarlo en img.
//Variables utilizada para almacenar las 2 últimas posiciones del objeto.
//Declaradas como -1 para evitar movimientos indebidos al comienzo
int lastX=-1;
int lastX2=-1;
int lastY2=-1;
int lastY=-1;

int c;              //Variable utilizada para almacenar el número de círculos encontrados
float f=3.85;               //Distancia focal
int x=0;
int y=0;            //Posición actual del objeto a seguir
int a=0;            //Variable utilizada para realizar un movimiento a la última posición
                    del objeto una vez perdido de vista. Si a=0 se ha perdido de vista.
CvFont font;                    //Declaración de la fuente de la letra
cvInitFont( &font, CV_FONT_VECTOR0, 0.5, 0.5, 0, 2.0, CV_AA);   //Función para definir
                                                                 el tipo de letra.
char vel5[255],vel6[255],postxt5[255],postxt6[255],posx[255],posy[255];   //Cadenas de
                                                                          caracteres
                                                                          utilizadas para
                                                                          mostrar
                                                                          información en
                                                                          la imagen
char color;//Cadena de caracteres utilizada para la selección de valores predefinidos de hsv


//////////CREACIÓN DE LAS VENTANAS PARA LAS IMÁGENES//////////////////////
cvNamedWindow("camara",CV_WINDOW_AUTOSIZE);     //Ventana en la que se muestra
la imagen captada por la cámara
cvNamedWindow("range",CV_WINDOW_AUTOSIZE);  //Ventana en la que se muestra la
imagen binaria tras la apertura y cierre. Esta imagen es la tomada para el cálculo del momento

////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////DECLARACIÓN DE IMÁGENES/////////////////////////////////
IplImage* hsv_frame = cvCreateImage(cvGetSize(img), 8,3);      //Frame de la imagen en HSV
IplImage* thresholded = cvCreateImage(cvGetSize(img), 8,1);

//Construcción del mapa indistorsionado
IplImage* mapx = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
IplImage* mapy = cvCreateImage( cvGetSize(img), IPL_DEPTH_32F, 1 );
cvInitUndistortMap(intrinsic,distortion,mapx,mapy);


//////////////APERTURA DEL PUERTO DE COMUNICACIONES/////////////////////
ret=PCube_openDevice(&dev,strInitString);
if (ret==-205)
{
printf("Puerto de comunicación cerrado.\n Es necesario ejecutar el archivo
'crear_entradas_dev.sh', localizado en la carpeta Robotnik");
}
else if (ret==0)
{
printf("Puerto de comunicación abierto.\n\n\n");
}
else
{
printf("Error desconocido.");
}
PCube_resetModule( dev, 5 );            //Reinicio del módulo 5 por si se encuentra bloqueado
PCube_resetModule( dev, 6 );            //Reinicio del módulo 6 por si se encuentra bloqueado
////////////////////////////////////////////////////////////////////////////////////////////


/////////FUNCIÓN UTILIZADA PARA LA ELECCIÓN DE VALORES PREDEFINIDOS//////////////

printf("¿Que color desea?\nPulse 'r' para rojo.\nPulse 'g' para verde.\nPulse 'b' para azul.\n
Pulse 'y' para amarillo.\n");
scanf("%s",&color);

switch(color)
{
case 'r':                           //Caso para el color rojo

        h_min=0;
        s_min=100;
        v_min=0;

        h_max=10;
```

```
        s_max=255;
        v_max=255;
        break;

case 'g':                       //Caso para el color verde

        h_min=0;
        s_min=0;
        v_min=0;

        h_max=180;
        s_max=255;
        v_max=255;
        break;

case 'b':                       //Caso para el color azul

        h_min=90;
        s_min=110;
        v_min=110;

        h_max=120;
        s_max=255;
        v_max=255;
        break;

case 'y':                       //Caso para el color amarillo

        h_min=20;
        s_min=100;
        v_min=100;

        h_max=30;
        s_max=255;
        v_max=255;
        break;
}
////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////COMIENZO DEL VIDEO/////////////////////////////////////
inicio=clock();         //Guardamos los ciclos de reloj
while(1)
{
```

```
//////////////////Almacenamiento de la información en un archivo de texto//////////////////
PCube_getPos(dev, 5,&pos_act5);
PCube_getPos(dev, 6,&pos_act6);
PCube_getVel(dev,5,&get_vel5);
PCube_getVel(dev,6,&get_vel6);
ahora=clock();
reloj=(ahora-inicio)/(double)CLOCKS_PER_SEC;
fprintf(doc,"%f \t",reloj);
if(x>0)  //Utilizamos un condicional para evitar que cuando almacene los datos y no
{                    haya ningún objeto almacene posiciones imposibles.
fprintf(doc,"%d \t",x);
fprintf(doc,"%d \t",y);
}
else
{
fprintf(doc,"000 \t");
fprintf(doc,"000 \t");
}
fprintf(doc,"%f \t",get_vel5);
fprintf(doc,"%f \t",get_vel6);
fprintf(doc,"%f \t",pos_act5);
fprintf(doc,"%f \n\n",pos_act6);
////////////////////////////////////////////////////////////////////////////////////////
IplImage* img = cvQueryFrame(capture);
cvRemap( img, img, mapx, mapy ); //Aplicación del mapa de distorsión a la imagen
////ELECCION DE COLOR////
AjusteHSV();              //Función creada para el ajuste de HSV, declarada al comienzo del
                          programa .
cvCvtColor(img, hsv_frame, CV_BGR2HSV);                    //Conversión de BGR a HSV
IplImage* thresholded = ThresholdedColor(hsv_frame);       //Binariazación de la imagen
cvShowImage("camara",img);                                 //Muestreo de la imagen RGB
//////////////////DECLARACIÓN DE KERNELS PARA APERTURA ////////////////////////
IplConvKernel *se11 = cvCreateStructuringElementEx(11, 11, 5,  5,  CV_SHAPE_RECT, NULL);
cvMorphologyEx(thresholded, thresholded,0, se11,CV_MOP_OPEN,1); //Aplicación de una
apertura (erosión-dilatación)
cvReleaseStructuringElement(&se11);          //Deasignación del espacio de memoria
cvSmooth(thresholded,thresholded,CV_GAUSSIAN,5,5);
cvShowImage("range",thresholded);    //Muestre de la imagen Binaria tras aplicar apertura
////////////////////////////////////////////////////////////////////////////////////////

//////////////////////CÁLCULO DEL CENTRO DEL OBJETO//////////////////////////////

if(on==1)
{
```

```
CvMemStorage* storage =cvCreateMemStorage(0);
CvMemStorage* storage1 =cvCreateMemStorage(0);
cvSmooth(thresholded,thresholded,CV_GAUSSIAN,5,5);
CvSeq* contours=0;
cvFindContours(thresholded,storage,&contours); //Buscamos contornos
cvZero(thresholded);
if(contours)
cvDrawContours(thresholded,contours,cvScalarAll(255),cvScalarAll(255),50);  //Dibujamos los contornos. Necesario, ya que la transformada de Hough trabaja con ellos
cvSmooth(thresholded,thresholded,CV_GAUSSIAN,9,9);
CvSeq* circles = cvHoughCircles(thresholded, storage1, CV_HOUGH_GRADIENT, 3, thresholded->height/4,100,40,0,0);  //Aplicación de la transformada de Hough
lastX2=lastX;
lastY2=lastY;
lastX=x;
lastY=y;
   for (c = 0; c < circles->total; c++)
          {
          float *p = (float*)cvGetSeqElem( circles, c );
          x= cvRound(p[0]);         //Almacenamos la posición x
           y= cvRound(p[1]);        //Almacenamos la posición y
           int radio = cvRound(p[2]);
          cvCircle(img, cvPoint(x,y), 3, CV_RGB(255,0,0), 0, 8, 0 ); //Dibujamos un círculo rojo para marcar el centro
           cvCircle( img, cvPoint(x,y), radio, CV_RGB(0,255,0), 2, 8, 0 ); //Dibujamos un círculo verde alrededor del objeto
           }

////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////MUESTREO DE IMÁGENES/////////////////////////////////////////
cvShowImage("camara",img);  //Se vuelve a mostrar la imagen para que aparezca la línea dibujada anteriormente

////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////COMIENZO DEL MOVIMIENTO/////////////////////////////////////
if (c!=0)
{

////////////CÁLCULO DE LAS VELOCIDADES Y POSICIÓN PARA LOS MOTORES/////////////////
vel_x=atan((x-320)/f);  //Calculo de la velocidad a alcanzar dependiendo de la posición con respecto a la cámara
vel_y=atan((y-240)/f);
incremento_x=incremento*(x-320)/450; //Cuando el motor 6 deba moverse a la izquierda el incremento será negativo
incremento_y=incremento*(240-y)/500; //Cuando el motor 5 deba moverse hacia arriba el incremento será positivo

PCube_getPos(dev, 6,&pos_act6);          //Obtenemos la posición actual del motor 6
pos6=pos_act6+incremento_x;              //Cálculo de la posición a la que ha de moverse

PCube_getPos(dev, 5,&pos_act5);          //Obtenemos la posición actual del motor 5
pos5=pos_act5+incremento_y;              //Cálculo de la posición a la que ha de moverse
////////////////////////////////////////////////////////////////////////////////////

if (vel_x==0)  //Si la velocidad calculada es 0, indica que la posición para el eje X es la correcta
        {}
        else if (vel_x<0) //Si es menor de 0, ha de moverse hacia la izquierda
                {
                PCube_moveRamp( dev, 6, pos6, -vel_x, acc );  //Función para la realización de un movimiento en modo Rampa a la izquierda...
                PCube_getVel(dev,6,&get_vel6);
                }
                else
                        {
                        PCube_moveRamp( dev, 6, pos6, vel_x, acc );  //Movimiento a la derecha
                        PCube_getVel(dev,6,&get_vel6);
                        }
if (vel_y==0)  //Si la velocidad calculada es 0, indica que la posición para el eje Y es la correcta
        {}
        else if (vel_y<0)  //Movimiento hacia arriba
                {
                PCube_moveRamp( dev, 5, pos5, -vel_y, acc );
                PCube_getVel(dev,5,&get_vel5);
                }
                else  //Movimiento hacia abajo
                        {
                        PCube_moveRamp( dev, 5, pos5, vel_y, acc );
                        PCube_getVel(dev,5,&get_vel5);
```

```
                        }

}
//////////////////////FIN DEL MOVIMIENTO////////////////////////////////////


////////////MOVIMIENTO EN CASO DE PÉRDIDA DE OBJETO/////////////////
else if(a!=0) //Si no usamos "a" al iniciar el programa si no existe ningún objeto a seguir, se realizaría un movimiento imposible
{
a=0;

incremento_x=incremento*(lastX2-lastX)/100;  //Como el movimiento ha debido de ser brusco el incremento es  5 veces mayor
incremento_y=incremento*(lastY2-lastY)/100;
vel_x=1.74; //Velocidad máxima de los motores
vel_y=1.74;

PCube_getPos(dev, 6,&pos_act6);
pos6=pos_act6-incremento_x;

PCube_getPos(dev, 5,&pos_act5);
pos5=pos_act5+incremento_y;


if (lastX==0)
        {}
        else if (lastX<320)
                {
                PCube_moveRamp( dev, 6, pos6, vel_x, acc );
                PCube_getVel(dev,6,&get_vel6);
                }
                else
                        {
                        PCube_moveRamp( dev, 6, pos6, vel_x, acc );
                        PCube_getVel(dev,6,&get_vel6);
                        }



if (lastY==0)
        {}
        else if (lastY>240)
                {
                PCube_moveRamp( dev, 5, pos5, vel_y, acc );
```

```
                PCube_getVel(dev,5,&get_vel5);
                }
                else
                        {
                        PCube_moveRamp( dev, 5, pos5, vel_y, acc );
                        PCube_getVel(dev,5,&get_vel5);
                        }


}

////////////////FIN DEL MOVIMIENTO EN CASO DE PÉRDIDA//////////////////////////

/////////////////MUESTREO DE POSICIÓN Y VELOCIDAD EN LA IMAGEN//////////////////
//Creación de cadena de caracteres
sprintf(vel5,"Velocidad 5:%f",get_vel5);
sprintf(postxt5,"Posicion 5:%f",pos_act5);
sprintf(vel6,"Velocidad 6:%f",get_vel6);
sprintf(postxt6,"Posicion 6:%f",pos_act6);
sprintf(posx,"Posicion X:%d",x);
sprintf(posy,"Posicion Y:%d",y);
ahora=clock();
reloj=(ahora-inicio)/(double)CLOCKS_PER_SEC;
//Escribimos en el archivo de texto
fprintf(doc,"%f \t",reloj);
if(x>0)
{
fprintf(doc,"%d \t",x);
fprintf(doc,"%d \t",y);
}
else
{
fprintf(doc,"000 \t");
fprintf(doc,"000 \t");
}
fprintf(doc,"%f \t",get_vel5);
fprintf(doc,"%f \t",get_vel6);
fprintf(doc,"%f \t",pos_act5);
fprintf(doc,"%f \n\n",pos_act6);

//Escritura sobre la imagen
cvPutText(img, vel5, cvPoint(50,50), &font,CV_RGB(0,0,0) );
cvPutText(img, vel6, cvPoint(50,100), &font,CV_RGB(0,0,0) );
cvPutText(img, postxt5, cvPoint(400,50), &font,CV_RGB(255,0,0) );
cvPutText(img, postxt6, cvPoint(400,100), &font,CV_RGB(255,0,0) );
```

```
cvPutText(img, posx, cvPoint(50,350), &font,CV_RGB(255,0,0) );
cvPutText(img, posy, cvPoint(50,400), &font,CV_RGB(255,0,0) );


//Muestra de la imagen modificada con el texto
cvShowImage("camara",img);

//////////////////////FINAL MUESTREO////////////////////////////
} //Fin (on==1) Si se vuelve a poner a 0 se detendrá el movimiento del robot pero no la visualización del video
char w =cvWaitKey(10);          //Espera 10 ms, en caso de pulsarse Esc, se cierra el programa
if (w==27) //27=Esc en código ASCII
break;
} //Fin while(1)

cvReleaseCapture(&capture);          //Liberación de memoria
cvDestroyAllWindows();               //Cierre de todas las ventanas creadas
fclose(doc);                         //Cierre del documento de texto
return 0;
}                                    //Fin del programa
```

ANEXO C - Código del programa de calibración.

Este código ha sido extraído de http://dsynflo.blogspot.com.es.

Sólo se ha realizado su traducción y una modificación en el tamaño de la imagen, ya que no se mostraba correctamente la imagen.

```
#include </usr/local/include/opencv/cv.h>
#include </usr/local/include/opencv/highgui.h>
#include "iostream"
#include <stdio.h>
#include <stdlib.h>

int imagenes = 0;              //Número de imágenes del tablero de ajedrez
int salto_frame;               //Frames para ser saltados
int casillas_w;                //Esquinas horizontales del tablero
int casillas_h;                //Esquinas verticales del tablero
int main(void)
{
        CvCapture* capture;

        printf("Introduce el número de imágenes = ");
        scanf("%d",&imagenes);

        printf("\rIntroduce el número de imágenes a saltar = ");
        scanf("%d",&salto_frame);

        casillas_w  = 8;
        casillas_h  = 6;

        int board_total  = casillas_w * casillas_h;
        //Número total de esquinas
        CvSize board_sz = cvSize( casillas_w, casillas_h );

        capture = cvCreateCameraCapture( 0 );
        if(!capture)
                {
                printf("\nNo se ha abierto la camara\n");
                return -1;
                }
        cvNamedWindow( "Snapshot" );
        cvNamedWindow( "Video");

        //Modifica el almacenamiento de los parámetros de acuerdo al tamaño del tablero
        utilizado
```

```
CvMat* image_points      = cvCreateMat(imagenes*board_total,2,CV_32FC1);
CvMat* object_points     = cvCreateMat(imagenes*board_total,3,CV_32FC1);
CvMat* point_counts      = cvCreateMat(imagenes,1,CV_32SC1);
CvMat* intrinsic_matrix  = cvCreateMat(3,3,CV_32FC1);
CvMat* distortion_coeffs = cvCreateMat(4,1,CV_32FC1);

//Nota:
//Intrinsic Matrix - 3x3                    Lens Distorstion Matrix - 4x1
//        [fx 0 cx]                                   [k1 k2 p1 p2   k3(optional)]
//        [0 fy cy]
//        [0  0  1]


CvPoint2D32f* corners = new CvPoint2D32f[ board_total ];
int corner_count;
int successes = 0;
int step, frame = 0;

IplImage *image = cvQueryFrame( capture );
IplImage *resized=cvCreateImage(cvSize(640,480), 8,3);
IplImage *gray_image = cvCreateImage(cvGetSize(resized),8,1);
cvResize(image,resized,CV_INTER_LINEAR);

//Las imágenes correctas son aquellas que encuentran tods las esquinas

while(successes < imagenes)
{
        if((frame++ % salto_frame) == 0)
                //Salta el número de frames pedido
        {
                //Busca las esquinas:
                int found = cvFindChessboardCorners(resized, board_sz, corners,
&corner_count,CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS );

                cvCvtColor(resized, gray_image, CV_BGR2GRAY);
                //Precisión de subpíxel en las esquinas
                cvFindCornerSubPix(gray_image, corners, corner_count,
cvSize(11,11),cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));

                cvDrawChessboardCorners(resized, board_sz, corners, corner_count,
found);         //Dibuja el tablero

                // Si el número de esquinas encontradas es correcto, almacena la
                información
                if( corner_count == board_total )
```

```
                        {
                        cvShowImage( "Snapshot", resized );//Muestra la imagen en color si es válida
                        step = successes*board_total;
                        for( int i=step, j=0; j<board_total; ++i,++j ) {
                        CV_MAT_ELEM(*image_points, float,i,0) = corners[j].x;
                        CV_MAT_ELEM(*image_points, float,i,1) = corners[j].y;
                        CV_MAT_ELEM(*object_points,float,i,0) = (float) j/casillas_w;
                        CV_MAT_ELEM(*object_points,float,i,1) =(float) (j%casillas_w);
                        CV_MAT_ELEM(*object_points,float,i,2) = 0.0f;
                        }
                CV_MAT_ELEM(*point_counts, int,successes,0) = board_total;
                successes++;
                printf("\r%d fotos válidas de %d realizadas.",successes,imagenes);
                }
                else
                cvShowImage( "Snapshot", gray_image );//Muestra la imagen en gris si no es válida
        }
        //Permite pausar el programa
        int c = cvWaitKey(15);
        if(c == 'p')
                {
                c = 0;
                while(c != 'p' && c != 27)
                        {
                        c = cvWaitKey(250);
                        }
                }
        if(c == 27)
                return 0;

        image = cvQueryFrame( capture );
        //Coge la siguiente imagen
        cvResize(image,resized,CV_INTER_LINEAR);
        cvShowImage("Raw Video", resized);
}

//Fin del bluce while cuando se alcanza el número de fotos correctas

cvDestroyWindow("Snapshot");

printf("\n\n *** Calibrando la cámara...n");

//Creamos matrices de acuerdo al número de capturas que se van a realizar
```

```
        CvMat* object_points2  = cvCreateMat(successes*board_total,3,CV_32FC1);
        CvMat* image_points2   = cvCreateMat(successes*board_total,2,CV_32FC1);
        CvMat* point_counts2   = cvCreateMat(successes,1,CV_32SC1);

        //Pasamos los puntos a matrices
        for(int i = 0; i<successes*board_total; ++i)
        {
CV_MAT_ELEM( *image_points2, float, i, 0)  =CV_MAT_ELEM( *image_points, float, i, 0);
CV_MAT_ELEM( *image_points2, float,i,1)    =CV_MAT_ELEM( *image_points, float, i, 1);
CV_MAT_ELEM(*object_points2, float, i, 0)  =  CV_MAT_ELEM( *object_points, float, i, 0) ;
CV_MAT_ELEM( *object_points2, float, i, 1) =  CV_MAT_ELEM( *object_points, float, i, 1) ;
CV_MAT_ELEM( *object_points2, float, i, 2) =  CV_MAT_ELEM( *object_points, float, i, 2) ;
        }

        for(int i=0; i<successes; ++i)
                {
                CV_MAT_ELEM( *point_counts2, int, i, 0) =   CV_MAT_ELEM( *point_counts,
                int, i, 0);
                }
        cvReleaseMat(&object_points);
        cvReleaseMat(&image_points);
        cvReleaseMat(&point_counts);

        // Iniciamos la matriz de intrínsecos con ambas longitudes focales en ratio de 1.0

        CV_MAT_ELEM( *intrinsic_matrix, float, 0, 0 ) = 1.0f;
        CV_MAT_ELEM( *intrinsic_matrix, float, 1, 1 ) = 1.0f;

        //Calibramos la cámara
        cvCalibrateCamera2(object_points2,   image_points2,   point_counts2,      cvGetSize(
        resized ), intrinsic_matrix, distortion_coeffs, NULL, NULL,0 );
        //CV_CALIB_FIX_ASPECT_RATIO

        //Guarda los valores en un archivo XML
        printf(" *** Calibración realizada!\n\n");
        printf("Almacenando los archivos Intrinsics.xml and Distortions.xml...\n");
        cvSave("Intrinsics_resized.xml",intrinsic_matrix);
        cvSave("Distortion_resized.xml",distortion_coeffs);
        printf("Archivos guardados.\n\n");

        printf("Comenzando la muestra corregida....");

        //Ejemplo: carga los valores intrínsecos y de distorsión calculados
        CvMat *intrinsic = (CvMat*)cvLoad("Intrinsics_resized.xml");
        CvMat *distortion = (CvMat*)cvLoad("Distortion_resized.xml");
```

```
        // Construye el mapa de undistorsión
        IplImage* mapx = cvCreateImage( cvGetSize(resized), IPL_DEPTH_32F, 1 );
        IplImage* mapy = cvCreateImage( cvGetSize(resized), IPL_DEPTH_32F, 1 );
        cvInitUndistortMap(intrinsic,distortion,mapx,mapy);

        // Muestra la imagen de la cámara distorsionada y sin distorsionar
        cvNamedWindow( "Undistort" );
        while(resized)
                {
                IplImage *t = cvCloneImage(resized);
                cvShowImage( "Raw Video", resized);          // Muestra la imagen
                cvRemap( t, resized, mapx, mapy );           // Undistorsiona la imagen
                cvReleaseImage(&t);
                cvShowImage("Undistort", resized);           // Muestra la imagen correcta

                //Permite pausar o cerrar cerrar el programa
                int c = cvWaitKey(15);
                if(c == 'p')
                        {
                        c = 0;
                        while(c != 'p' && c != 27)
                                {
                                c = cvWaitKey(250);
                                }
                        }
                if(c == 27)
                        break;
                image = cvQueryFrame( capture );
                cvResize(image,resized,CV_INTER_LINEAR);
                }
        return 0;
}
```

Bibliografía

[1] L.A. Silva. “Control visual de robots paralelos. Análisis desarrollo y aplicación a la plataforma robotenis”. Universidad Politécnica de Madrid, 2005.

[2] http://es.wikipedia.org/wiki/robótica

[3] http://www.robotnik.es/es/

[4] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/DIAS1/

[5] G. Bradski, A. Kaehler. “Learning OpenCV”. O’Reilly, 2008.

[6] http://dsp.stackexchange.com/questions/2687/why-do-we--use-the-hsv-colour-space-so-often-in-vision-and-image-processing

[7] http://www.tsc.uc3m.es/imagine/Curso_ProcesadoMorfologico/

[8] J. Kilian. “Simple image analysis by moments”. 2001.

[9] H. Rhody. “Lecture 10: Hough Circle Transform”. Rochester Institute of Technology, 2005.

[10] Manuales del brazo modular de Robotnik.

[11] B. Siciliano, O. Khatib. “Handbook of robotics”. Páginas: 563-582. Springer, 2008.

[12] R. Laganière. “OpenCV 2: Computer visión application programming cookbook”. Packt Publishing, 2011.

[13] E. Dombre, W. Khalil. “Modelling, performance, analysis and control of robot manipulators”. Iste, 2007.

[14] R. Igual, C. Medrano. “Tutorial OpenCV”. Universidad Politécnica de Teruel, 2008.

[15] J. Isern. “Estudio experimental de métodos de calibración y autocalibración de cámaras”. Páginas: 20-26. 2003.

[16] A. Barrientos. “Fundamentos de robótica”.

[17] Schunk. “Programmers guide for PowerCube”. 2004.

[18] J.F. Pertusa. “Técnicas de análisis de imágenes: aplicaciones en Biología”. Universidad de Valencia, 2003.

[19] C. Díaz. “Caracterización y monitorización del comportamiento del brazo robot modular”. Universidad Politécnica de Cartagena, 2012.

[20] R. Stolkin, I. Florescu, M. Baron, C. Harrier, B. Kocherov. “Efficient visual servoing with the ABCshift tracking algorithm”. Páginas: 3219-3224, publicado en IEEE International Conference on Robotics and Automation, 2008.

[21] J.E. Solanes. “Visual servoing multifrecuencia”. Páginas: 74-102. Universidad Politécnica de Valencia, 2010.

[22] http://blogrobotica.linaresdigital.com

[23] http://www.aishack.in/topics/blog

[24] http://petercorke.com/Robotics_Toolbox.html

[25] http://es.wikipedia.org/wiki/visual_servoing