industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Departamento de sistemas de Control y Automática

Control selectivo por visión artificial del brazo Robotnik

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Alberto Marín Guillén**
Director: **Jose Luis Muñoz Lozano**
Codirector: Jorge Feliú Batlle

Cartagena, a 20 de Junio de 2015

Universidad Politécnica de Cartagena

Índice general

Índice de ilustraciones

Índice de tablas

Capítulo 1. Información acerca del proyecto

1. Introducción al capítulo.

Se ha decidido dividir la memoria en seis capítulos que recogen las diferentes etapas de desarrollo del proyecto por separado.

El presente capítulo sirve para justificar la decisión de realizar el proyecto, para introducir los objetivos designados o fases de realización en el proyecto, además de situar el campo de la robótica industrial en el campo de la ingeniería y en la historia y dentro de la robótica el campo que atañe al proyecto, el control visual, cuyas herramientas han sido utilizadas ampliamente a lo largo del mismo.

1.1. Justificación del proyecto.

La intención directa del presente proyecto es la de adquirir conocimientos de manera autónoma sobre materias de robótica y visión artificial, materias que no se imparten como tales en el itinerario de un graduado en G.I.T.I. y que siempre me han llamado la atención e interesado. Esto culminaría con la realización de una aplicación práctica que coordine las diferentes técnicas estudiadas.

Se pretende obtener la solución de un problema real en que el papel de la visión sea tanto selectivo como extractivo y de control. Selectivo porque se desea que el robot elija según el color el objetivo sobre el que realizar alguna acción en lo que se llama procesamiento de imágenes. Extractivo porque la posición del objeto en la imagen permitirá obtener ciertos parámetros con los que el robot realizará debidamente la acción para la cual ha sido programado. La aplicación también incluirá una parte de control de movimiento de los motores.

Tanto el proceso de definición de los objetivos como el de la aplicación a desarrollar han sido extensamente deliberados para conseguir un equilibrio entre practicidad y dificultad. Finalmente, se decidió que la aplicación del brazo robótico consistiera en la extracción de objetos de un color determinado (selectivo) en movimiento en una cinta transportadora. En esta aplicación solo sería necesario el uso de tres de los seis grados de libertad disponibles, aquellos que permiten la disposición del extremo en cualquier posición y orientación en el plano de actuación del robot (plano vertical). Además, la limitación de grados de libertad en el movimiento del objeto que supone el desplazamiento rectilíneo permite el control con una única cámara que extraiga los parámetros necesarios para realizar la captura. La cinta transportadora fue simulada con una guía situada en una mesa métrica para limitar el movimiento a uno rectilíneo.

Se pretendió que ofreciera la menor resistencia al avance para poder simular lo mejor posible velocidad constante.

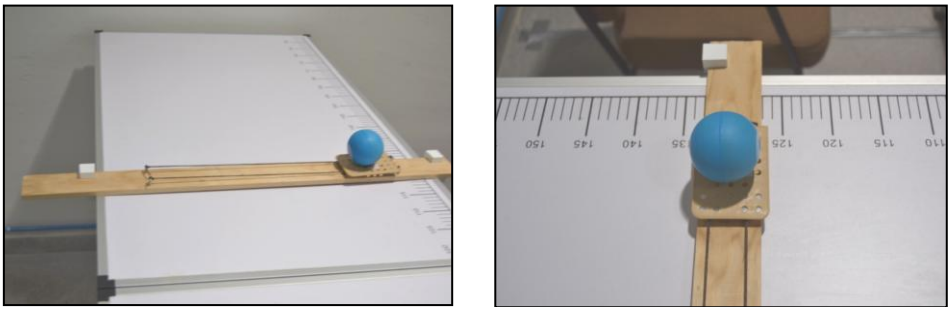


Ilustración 1. Material para la simulación de la aplicación del proyecto

1.2. Antecedentes. Proyectos previos.

En este apartado es imprescindible mencionar el trabajo de fin de carrera de Alejandro Sánchez Mur, de cuyo proyecto se ha aprovechado el programa de umbralización y la obtención del centro de masas del objeto y se han hecho numerosas referencias en el proyecto, dispuestas en la sección referencias. En la parte de control visual para el seguimiento del objeto umbralizado se han realizado ajustes del programa de Alejandro para adaptarlo a la situación del proyecto y al uso de una cámara diferente.

Otro proyecto en el que se ha realizado el control del brazo Robotnik en la UPCT ha sido el de Jose Antonio Torres Sánchez, aunque en este caso no se realizó control visual sino que se utilizó un dispositivo medidor de distancias con un mecanismo de pulso láser y de espejo rotatorio.

1.3. Objetivos del proyecto.

Tal como se ha señalado en la propuesta del proyecto, los objetivos que se han desarrollado a lo largo de la vida del proyecto son:

- La puesta en marcha del robot y el aseguramiento del correcto funcionamiento de todos los motores, mediante el programa de PowerCube. Se detalla en el apartado 2.1.1.

- Selección e instalación del lenguaje de programación más adecuado. Varios han sido los lenguajes que se han probado para la realización del programa del control con la librería OpenCv. Se finalizó realizando el programa en lenguaje C con Microsoft Visual Studio. El proceso de decisión se describe detenidamente en el apartado 2.1.3.

- La selección de la cámara y la búsqueda de información acerca de la calibración. Se comenzó utilizando la cámara Logitech, pero hubo un problema en el modo Eye-in Hand que obligó a situar en su lugar la cámara uEye. La cámara Logitech no dispone de enfoque, la distancia focal es fija. La cámara uEye sí dispone de anillo de enfoque que permite una distancia focal variable. La descripción más detallada sobre el proceso de elección de cámara y calibración se encuentra en el apartado 2.2.2.

- El diseño e implementación del programa de control visual para los modos de control cámara en fija y en mano. Se ha realizado en varias etapas, pues para empezar es necesario el procesamiento de imágenes y la umbralización del color de objeto deseado para después continuar con la adquisición de parámetros y proceder posteriormente con el problema cinemático para la captura y recogida del objeto. La descripción de los pasos seguidos para la definición del programa de control, que se ha finalizado aplicándole un criterio selectivo, ha sido recogida en el capítulo 3.

1.4. La robótica industrial.

En la actualidad, la importancia de los sistemas robotizados va en crecimiento. Su utilidad se ha extendido a diversos campos: industria, servicios domésticos, medicina, militar, entre otros. El proyecto se centra en la rama industrial de la robótica, la que se desarrolló en primer lugar.

Los manipuladores robóticos han contribuido a la modernización de las industrias y a la mejora la productividad, sustituyendo el trabajo realizado por operarios en procesos en los que su salud física y mental está en juego, como los desarrollados en ambientes hostiles o procesos repetitivos y tediosos que pueden llevar a un descenso del rendimiento o incluso a problemas psicológicos. También son útiles en aquellos trabajos que el hombre no es capaz de realizar por su complejidad.

Las definiciones más destacadas de robot industrial han sido dadas por las siguientes entidades [1]:

- 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”.
- La Organización Internacional de Estándares (ISO) define 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”.

1.4.1. Origen de la robótica.

Hasta que Taylor propuso a principios del siglo XX un modelo de producción teórico enfocado especialmente en el control de los tiempos y los movimientos de operarios dentro de una línea de producción no se investigó ninguna alternativa a la operación manual directa o con herramientas. La extrema especialización que proponía el modelo de Taylor, que fue puesto en práctica en primer lugar en la línea de producción de la Ford Company, reservaba a cada operario una función muy concreta que requería escasa cualificación. Dichas funciones eran controladas hasta el más mínimo gesto por la oficina de métodos y tiempo, con el objetivo de que el desempeño de cada operario no influyera significativamente sobre el tiempo de producción. [2]

La necesidad creciente de minimizar los tiempos de operación llevó a la idea de introducir máquinas que realizaran operaciones concretas de manera autónoma.

La palabra robot apareció por primera vez en la novela del escritor checo Carel Kapek "Rossus Universal Robot" de 1921 y fue difundida y desarrollada por el escritor de origen ruso Isaac Asimov. Dicha palabra proviene de la palabra checa robota, que significa trabajo penoso o trabajo duro.

1.4.2. Desarrollo de la robótica.

Si no se considera el más antiguo de los antecesores del robot, el autómata, nos queda que su más directo progenitor fue el telemanipulador. El primero fue desarrollado por Goertz en 1948 para manipular elementos radioactivos sin riesgo para el operador, un dispositivo mecánico maestro-esclavo. El maestro se situaba en la zona segura transmitiendo mecánicamente al esclavo el movimiento. Años más tarde, en 1954, Goertz hizo uso de la tecnología electrónica y del servocontrol sustituyendo la transmisión mecánica por otra eléctrica, desarrollando así el primer telemanipulador con servocontrol bilateral. Los telemanipuladores no han sufrido una evolución tan espectacular en los últimos años como la de los robots. Sus aplicaciones han quedado limitadas a campos como la industria nuclear, la industria química o la espacial. [3]

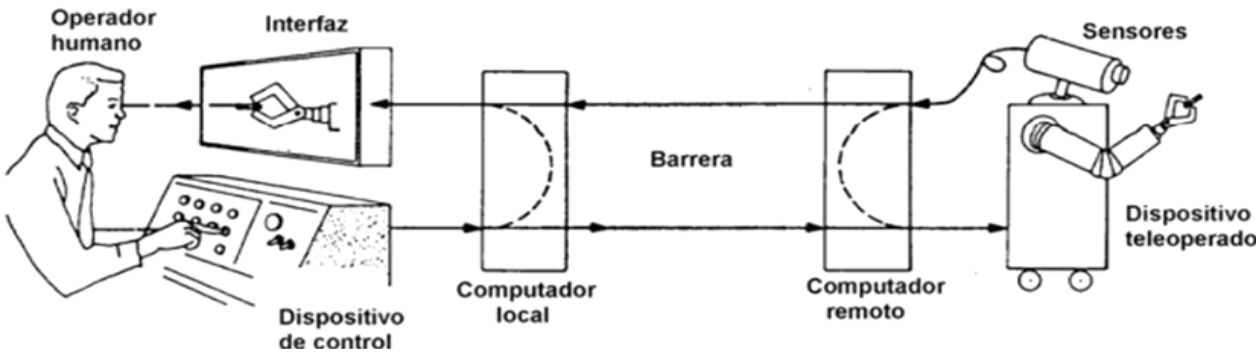


Ilustración 2. Elementos básicos de un telemanipulador [4]

La sustitución del mando continuo del operador por un programa de ordenador que controlase los movimientos del manipulador dio paso al concepto de robot. Fue George C. Devol, ingeniero norteamericano el que estableció las bases del robot industrial moderno. 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. Con esta tecnología no solo se valieron los primeros robots industriales, sino que sigue siendo la base del funcionamiento de la mayoría de los robots actuales. [3]

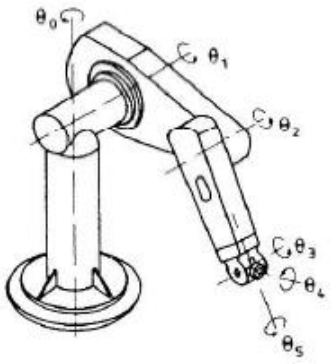


Ilustración 3. Brazo manipulador universal programable, PUMA [3]

La configuración de los primeros robots respondía a las denominadas configuraciones esférica y antropomórfica. En 1982, el profesor japonés Makino desarrolla el concepto de robot SCARA (Selective Compliance Assembly Robot Arm), una configuración con un número reducido de grados de libertad y coste limitado orientada al ensamblado de piezas.

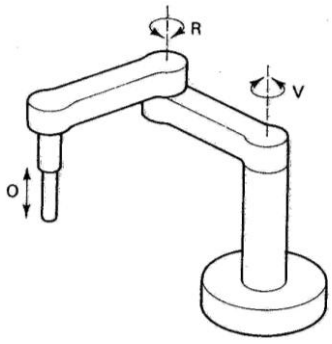


Ilustración 4. Modelo de robot SCARA [3]

Paralelamente a la robótica industrial se van desarrollando sistemas móviles autónomos capaces de desenvolverse por sí mismos en entornos desconocidos sin necesidad de supervisión humana. El robot Shakey, invento del SRI de 1966, fue el primer robot con inteligencia artificial de la historia.

La robótica se mantiene en continuo proceso de mejora en la actualidad. Alguna de las últimas novedades en robots implica a aquellos con capacidad de identificar emociones y responder ante ellas, a robots quirúrgicos Da Vinci, o a robots con capacidad de vuelo.

1.4.3. Clasificación de robots [1]

Podemos clasificar los tipos de robots según diversos criterios, como pueden ser:

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

   - Eléctricos: Obtienen la energía de un generador de corriente continua o alterna.
   - Neumáticos: Generan su movimiento a partir de aire comprimido y válvulas.
   - Hidráulicos: Son movidos gracias a la energía de un fluido en estado líquido, generalmente aceites.

2) Clasificación generacional:

   - 1ª generación o telemanipuladores, que ya han sido descritos previamente.
   - 2ª generación o robots de aprendizaje, que repiten una secuencia de movimientos que puede ser reprogramada.
   - 3ª generación o robots de control sensorizado, que responden al medio de diferente forma según la información recibida por los sensores de que dispone.
   - 4ª generación o robots inteligentes, con capacidad de autoaprendizaje y de toma de decisiones.

3) Según su arquitectura:

   - Poliarticulados: En este grupo están los robots estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número limitado de grados de libertad. En este grupo se encuentran los manipuladores y los robots industriales, empleados cuando es preciso acotar la zona de trabajo a una zona que debe ser debidamente protegida para evitar accidentes.

- Móviles: Robots con capacidad de movimiento que son guiados por entorno por telemando o con la información que reciben de sus sensores. Pueden llevar acoplado uno o varios brazos articulados.

  En este grupo se encuentran los robots para el transporte de piezas dentro de una cadena de producción.

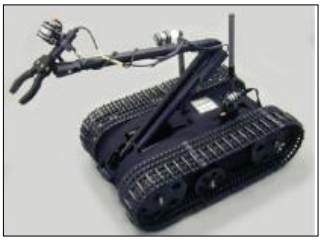


Ilustración 5. Robot móvil usado en la desactivación de minas [1]

- Androides: Reproducen 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.

Ilustración 6. Robot androide ASIMO de la empresa Honda [1]

- Zoomórficos: Son robots que pretenden imitar movimientos animales. Dentro de este grupo podemos diferenciar caminadores, como los robots mascota, o reptadores, que emulan serpientes o gusanos.

Ilustración 7. Robots zoomórficos: el primero reptante, el segundo robot mascota [1]

1.5. Control visual.

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: una en la que la cámara es independiente del movimiento del robot, (Eye-To-Hand, cámara fija) y otra en la que pertenece al cuerpo del robot y, por tanto, se mueve con él (Eye-In-Hand, cámara en mano).

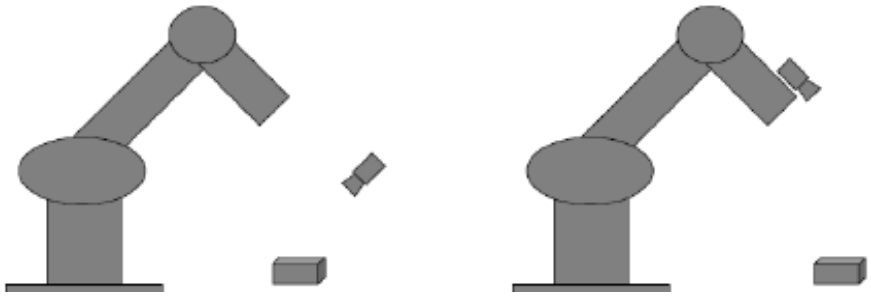


Ilustración 8 Disposición del sistema de visión: (a) Eye-To-Hand, (b) Eye-In-Hand. [5]

Dependiendo de las características de la imagen podemos distinguir: control basado en posición (PBVS, position-based visual servoing), control basado en imagen (IBVS, image-based visual servoing) y enfoques avanzados como VS híbrido y VS particionado, aunque los más utilizados son los dos primeros.

Para el PBVS (Ilustración 9) se realiza una estimación de la posición para compararla con la posición deseada, a partir de las características extraídas de las imágenes.

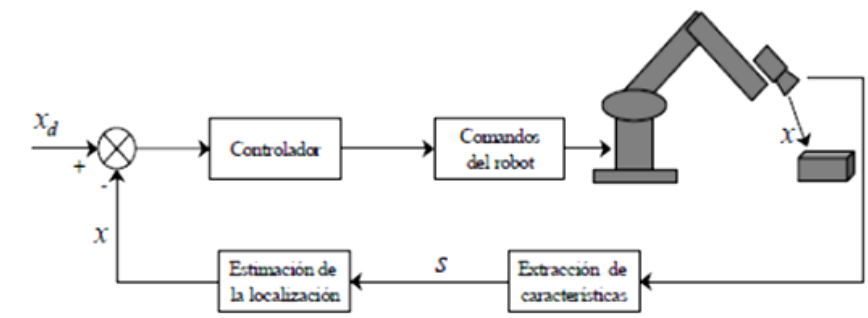


Ilustración 9. Control visual basado en posición. [5]

Por otro lado, en el IBVS se compara la imagen obtenida con una imagen deseada y, utilizando la matriz de interacción o jacobiana se relaciona el mundo real y la imagen 2D captada por la cámara.

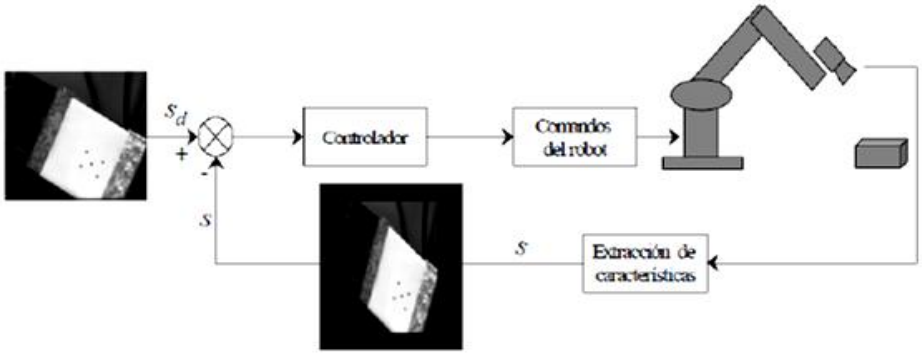


Ilustración 10. Control visual basado en imagen. [5]

Se ha realizado un programa de control tanto para cámara en mano como fija para poder comparar las diferencias que puede haber entre cada uno de los dos métodos. En la disposición Eye-In-Hand (apartado 3.2.4.1) se tendrá control IBVS, en el cual el algoritmo compara las características de la imagen captada con la de referencia, em una disposición con el objetivo centrado, generando un cambio de posición en el brazo robótico para lograr que el error existente entre ambas posiciones sea nulo. En la disposición Eye-To-Hand (apartado 3.2.4.2) el control será en lazo abierto, como se describirá posteriormente.

1.6. Estructura de la memoria

Tras esta breve introducción del mundo de la robótica industrial y de los aspectos de control visual que atañen a este proyecto se sigue con la organización estructural de los siguientes capítulos que conforman el trabajo realizado en el desarrollo del proyecto.

En el segundo capítulo se da a conocer el hardware utilizado en el sistema, que engloba tanto al brazo robótico como a las cámaras, así como el procedimiento de puesta en marcha. Además se detalla el proceso de elección de software usado para la programación y el software que proporciona Robotnik para el control de los módulos.

El tercer capítulo representa el grueso del proyecto, pues en el se presenta el desarrollo teórico y los pasos seguidos en la programación en los modos Eye-in-Hand y Eye-to-Hand. En una segunda parte se incluye la parte del programa que le da el carácter de selectivo.

El capítulo cuarto describe las limitaciones que presenta un programa como el realizado de control selectivo, en aspectos de superposición de objetos e iluminación.

Por último se presentan las conclusiones del proyecto y un apartado reservado a trabajos futuros.

Capítulo 2. Hardware y software utilizado

2. Introducción al capítulo.

En esta sección se pretende describir el hardware y software utilizado, incluyendo las características en los casos de hardware. Además, se debe justificar el proceso de selección del material disponible.

2.1. Software.

2.1.1. PowerCube

Este es el programa proporcionado junto al brazo robótico para el escaneo y reseteo de los diferentes módulos, para probar su correcto funcionamiento y otras operaciones como limitar el espacio de trabajo y estudiar velocidades y aceleraciones de forma controlada. El menú principal es el que aparece a continuación.

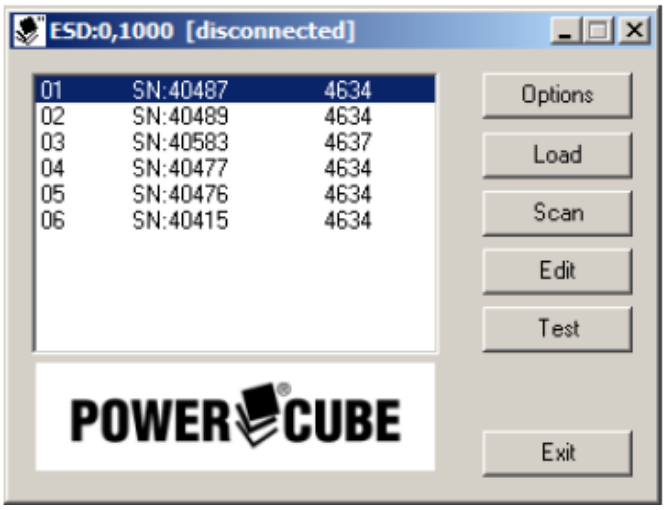


Ilustración 11. Ventana de control de módulos de PowerCube

Las acciones que se han utilizado han sido las siguientes:

- Scan. Permite escanear los dispositivos conectados al puerto CAN de nuestro equipo. Una vez escaneados, si no hay ninguna incidencia, deben aparecen los 6 módulos del brazo en la ventana principal del programa, definidos cada uno por un número de serie "SN". Si uno de los módulos no ha sido reconocido la solución puede consistir en reiniciar la fuente de alimentación del robot según el procedimiento indicado en la guía.

- Test. Esta opción nos permite teleoperar cada una de las articulaciones de forma independiente pudiendo modificar los valores de los módulos, su velocidad y la aceleración de los mismos.

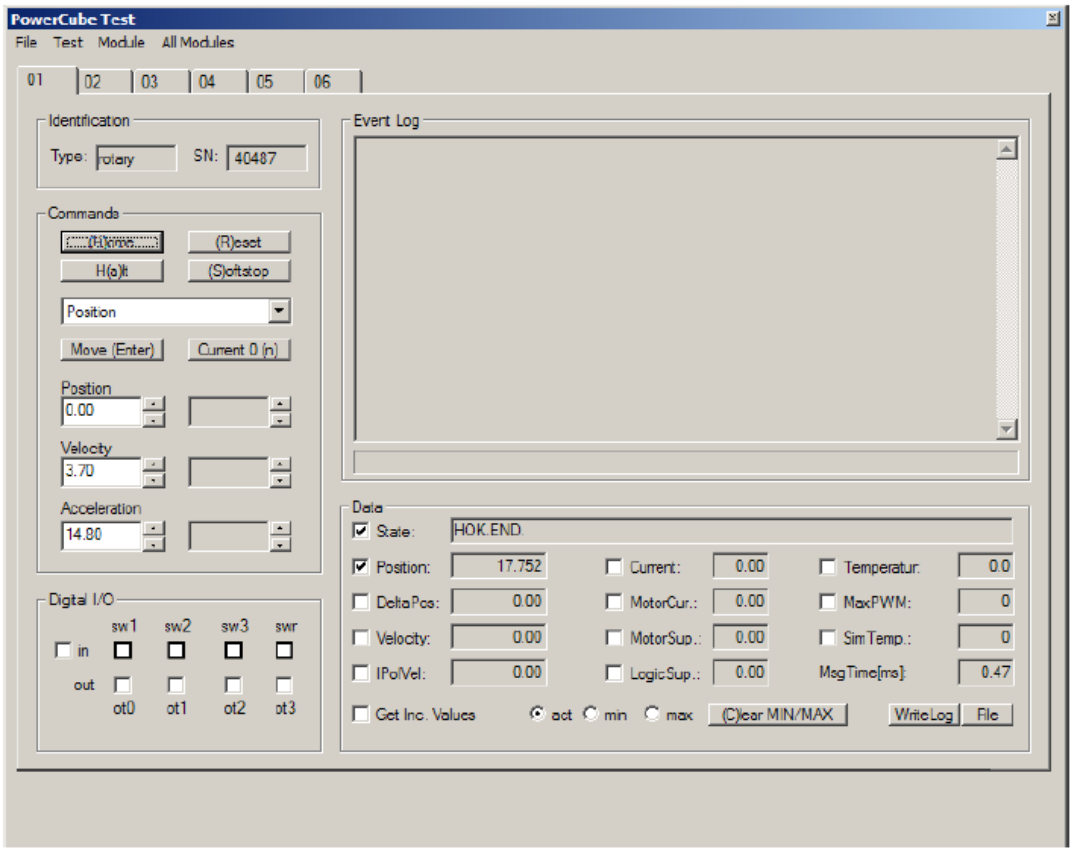


Ilustración 12. Ventana Test del programa

En la parte superior tenemos 6 pestañas, una por cada uno de los motores acoplados en los módulos de los motores. Existen las opciones de resetear los módulos por si ha quedado accidentalmente bloqueado, mover el módulo la posición deseada (como ya se describirá, todos los módulos del brazo Robotnik son giratorios y esta posición es angular) con la velocidad y la aceleración deseada, además de mandar la orden de detención del brazo.

En la parte derecha se dispone de variables de cada módulo del robot. La de estado muestra si hay algún error en el módulo. La otra variable que se dispone activa es la posición actual del módulo en actividad del brazo.

En la siguiente tabla se muestran los posibles errores que se pueden detectar en la inicialización del test del programa. En una ocasión apareció en pantalla un error -222 con el módulo 2 y se trataba de que el fusible se había fundido por exceso de potencia. Esto se solucionó colocando un fusible en paralelo para aguantar las sobreintensidades del arranque.

Value	Define	Description
-201	ERRID_DEV_FUNCTIONNOTAVAILABLE	The function called is not available.
-202	ERRID_DEV_NOINITSTRING	The InitString is missing during initialization.
-203	ERRID_DEV_NODEVICENAME	The device name specified in InitString is wrong or invalid.
-204	ERRID_DEV_BADINITSTRING	The InitString is incomplete or wrong.
-205	ERRID_DEV_INITERROR	Initialization of the interface failed. Check hardware and driver setup.
-206	ERRID_DEV_NOTINITIALIZED	The function was called before initializing the device.
-207	ERRID_DEV_WRITEERROR	Error during an attempt to write data to the interface.
-208	ERRID_DEV_READERROR	Error during an attempt to read data from the interface.
-209	ERRID_DEV_WRITETIMEOUT	Timeout while sending data on the bus.
-210	ERRID_DEV_READTIMEOUT	Timeout while reading data from a module.
-211	ERRID_DEV_WRONGMESSAGEID	The message received has an unexpected MessageID.
-212	ERRID_DEV_WRONGCOMMANDID	The message received has an unexpected CommandID.
-213	ERRID_DEV_WRONGPARAMETERID	The message received has an unexpected ParameterID.
-214	ERRID_DEV_EXITERROR	Error occured while closing the interface.
-215	ERRID_DEV_NOMODULES	No module found during initialization of the interface.
-216	ERRID_DEV_WRONGDEVICEID	The given DeviceID is wrong.
-217	ERRID_DEV_NOLIBRARY	A DLL file is missing to execute the function call.
-218	ERRID_DEV_ISINITIALIZED	The Interface has been already initialized.
-219	ERRID_DEV_WRONGEMSMODULEID	The given EMS module ID does not exist.
-220	ERRID_DEV_EMSNOTINITIALIZED	The EMS module has not been initialized.
-221	ERRID_DEV_EMSMAXNUMBER	The maximum number of EMS modules has been reached.
-222	ERRID_DEV_EMSINITERROR	Error initializing an EMS module.
-223	ERRID_DEV_WRONGEMSTYPE	This function is intended to use with a different EMS module type.
-224	ERRID_DEV_WRONGEMSCHANNELID	The given channel ID of the EMS module does not exist.
-225	ERRID_DEV_WRONGMP55MODULEID	The given MP55 module ID does not exist.
-226	ERRID_DEV_WRONGSCHUNKMODULEID	The given SCHUNK module ID does not exist.

Tabla 1. Errores de posible detección en los módulos Schunk. [6]

2.1.2. 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 otros lenguajes como C, Java y Phython. [8]

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

2.1.3. 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 Lab Windows
Suse Linux 6.4	Código abierto	GNU C/C++
QNX 4.25	Código abierto	Watcom C/C++ v.10

Tabla 2. Compatibilidades de la librería m5api.

Un sumario de las funciones incluidas dentro de esta librería se encuentran en el **Anexo** a disposición del lector.

2.1.4. Entorno de programación

Se estudiaron varias opciones para la realización del programa que incluyera la parte de procesamiento de imagen realizada en C por Alejandro y resolviera el control visual y la parte cinemática para la aplicación ya descrita. Se van a justificar las diferentes alternativas.

- **Microsoft Visual Studio** es un entorno para sistemas Windows que soporta múltiples lenguajes de programación tales como C++, C, Visual Basic, Java, Python o C#, entre otros.

  La gran ventaja del uso de este entorno es la compatibilidad de los lenguajes C y C++ con OpenCv, la librería que se ha comentado en el anterior apartado que permite simplificar problemas realmente complejos de visión en unas pocas líneas de código. La desventaja principal de usar este entorno es que los lenguajes mencionados tienen una potencia de cálculo media en operaciones y matrices, usadas en transformaciones propias de la calibración y en la cinemática del robot.

- El compilador **GNU de Linux** ofrece gran robustez y muy buen rendimiento. Posee un poderoso entorno gráfico llamado X Window System, un sistema de ventanas que aportan una mejor apariencia al programa.

  Como permite trabajar en lenguaje C o C++ se barajó la posibilidad de trabajar con este programa con una partición de Linux del ordenador, pero al final se optó por trabajar con Microsoft Visual Studio dada la comodidad que ofrecía.

- **Matlab** es un programa con un alto potencial de cálculo que no precisa de la instalación de librerías de cálculo y que además cuenta con una interfaz gráfica propia "GUI" muy fácil de implementar y con un diseño muy cuidado. Aunque su mayor potencial se debe a la velocidad de cálculo con vectores y matrices.

  Se investigó la forma de programar en C/C++ y ejecutarlo en Matlab haciendo uso de las ventajas citadas. El programa que se encontró para realizar la conversión es MexOpenCv, cuya misión es la de realizar una construcción del programa realizado en C/C++ para poder llamarlo desde Matlab. La página para descargar el programa si la versión de OpenCv instalada es anterior a la 3.0.0 (y posterior a la 2.4.0) es la siguiente: https://github.com/kyamagu/mexopencv/tree/v2.4. La misma página contiene un enlace a tutoriales sobre la programación con MexOpenCv.

2.2. Hardware.

2.2.1. Brazo Robotnik.

Ilustración 13. Brazo robot de Robotnik. [8]

Se trata de un brazo robot, diseñado por la empresa Robotnik Automation SSL., formado a partir de seis servoaccionamientos modulares PowerCube de Schunk.

Los servos no requieren de armario control externo ya que cada uno de ellos incluye motorreductor, etapa de potencia y controlador. La conexión externa consta de sólo 4 hilos, 2 para la alimentación y otros 2 hilos para la comunicación CAN bus.

Podemos destacar entre sus características generales [8]:

- Hasta 7 grados de libertad (en un mismo bus).
- Alcance de 400 a 1300 mm.
- Peso de 25 kg en una configuración de 6GL.
- 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 que posea una tarjeta de comunicaciones CAN.
- Alimentación a 24 VDC y 20 A. Para evitar posibles ruidos en la alimentación, esta debe ser dividida en dos suministros diferentes, uno para el control y otro para la potencia del motor. El bus CAN debe ser unido junto a la masa de control. Otra posibilidad que permite el brazo robótico es la que el suministro de potencia se realice a través de baterías
- Configuración modular.

2.2.1.1. 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. [9]

También 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.

2.2.1.2. Módulos.

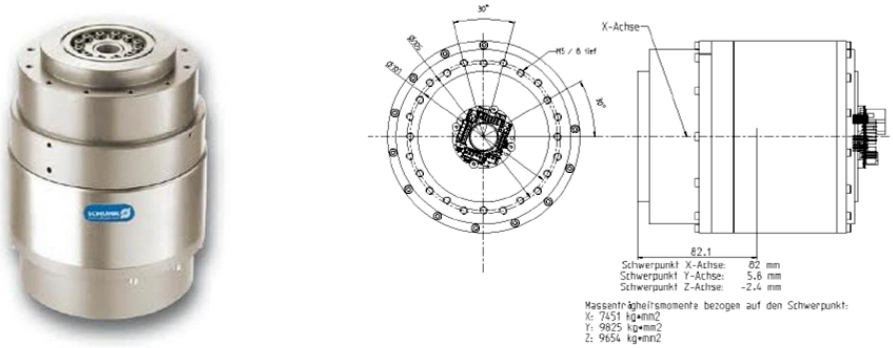


Ilustración 14. Módulos del brazo robótico [10]

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.

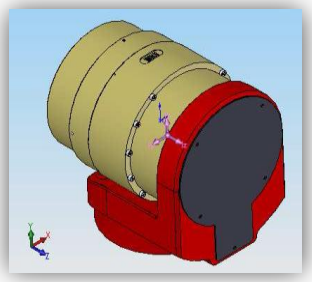
Entre las características de estos módulos tenemos [10]:

- Cada eje contiene su propio controlador y servo amplificador.
- Motores sin escobillas (brushless).
- Eje pasante que permite el paso interior de los cables
- Encoder absoluto para posicionamiento y control de velocidad.
- Monitorización de rango, limites, temperatura y corriente.
- Requerimientos de potencia de 20A y 24VDC (480W para todo el brazo).
- Integración inmediata con cualquier tipo de efector final: pinzas, manos robotizadas, taladros, útiles de soldadura, etc.

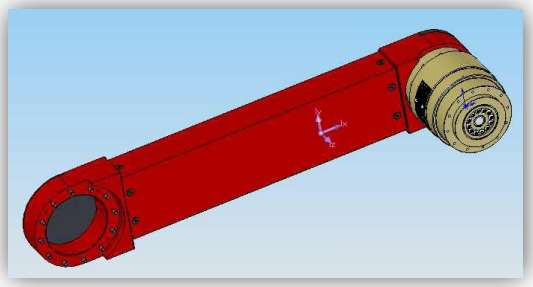
- Freno magnético integrado en todos los ejes.
- Arquitectura abierta (control de alto nivel sobre el movimiento de cada eje).

2.2.1.3. Enlaces

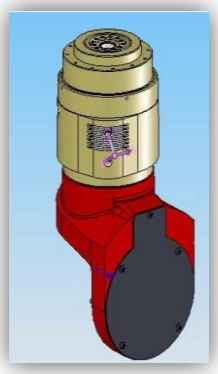
Los enlaces a los que se acoplan los diferentes módulos (y sus dimensiones) son los siguientes:

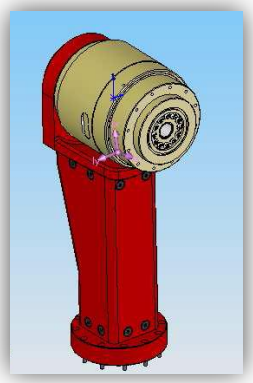


Eslabón 1 (L1=241.1 mm)

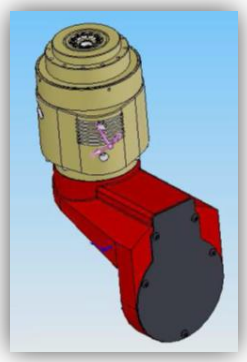


Eslabón 2 (L2=495 mm)

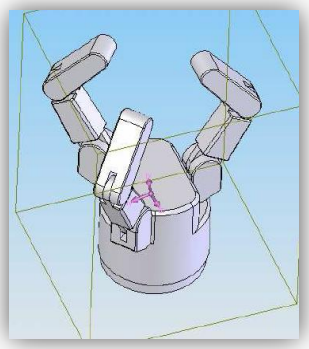




Eslabones 3 y 4 (L3=372 mm)



Eslabón 5 (L4=300 mm)



Manipulador final

Tabla 1. Lista de enlaces [10]

Como manipulador final se ha incluido la imagen proporcionada en los manuales de Robotnik, pero hay que indicar que es un componente independiente del fabricante y que durante la realización del proyecto se ha realizado el pedido pero por la tardanza no se ha podido incluir en el mismo.

A pesar de que no se va a realizar el estudio dinámico del brazo, se incluye una tabla resumen con sus parámetros característicos de centro de masas e inercial para futuros proyectos.

	COM	**Parámetros de inercia**
Eslabón 1	Cx = -0,00057 [m] Cy = -0,01133 [m] Cz = 0,00878 [m]	Ixx = 0 Iyy = 0,000030941 [kg*m^2] Izz = 0 Ixy = 0 Ixz = 0 Iyz = 0
Eslabón 2	Cx = 0,18128 [m] Cy = -0,00123 [m] Cz = -0,07017 [m]	Ixx = 0,00609233218 [kg*m^2] Iyy = 0,1845680479 [kg*m^2] Izz = 0,18838664807 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Eslabón 3	Cx = -0,00087 [m] Cy = -0,01087 [m] Cz = 0,10241 [m]	Ixx = 0,00069941799 [kg*m^2] Iyy = 0,00071684059 [kg*m^2] Izz = 0,00053893461 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Eslabón 4	Cx = -0,00041 [m] Cy = -0,04893 [m] Cz = 0,00485 [m]	Ixx = 0,00653606965 [kg*m^2] Iyy = 0,00097427479 [kg*m^2] Izz = 0,00637720193 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Eslabón 5	Cx = -0,00011 [m] Cy = -0,01824 [m] Cz = 0,09177 [m]	Ixx = 0,00083022261 [kg*m^2] Iyy = 0,00064595927 [kg*m^2] Izz = 0,00071374229 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0
Eslabón 6	Cx = -0,00018 [m] Cy = -0,00906 [m] Cz = =0,09540 [m]	Ixx = 0,00030738666 [kg*m^2] Iyy = 0,00015861153 [kg*m^2] Izz = 0,00045953777 [kg*m^2] Ixy = 0 Ixz = 0 Iyz = 0

Tabla 3. Parámetros dinámicos de los eslabones del brazo robótico. [10]

2.2.1.4. Tarjeta y buses CAN

El módulo CAN-PCI/331 es una tarjeta de PC diseñada por PCI bus que usa un microcontrolador 68331 para dirigir los datos del bus CAN. Estos datos son almacenados en la SRAM local. La seguridad y consistencia de los datos está garantizada para 1 Mbit/s, que es la capacidad de transferencia del bus por defecto.

La señal del bus está aislada eléctricamente de potenciales externos por medio de optoacopladores y convertidores DC/DC.

Para aplicaciones remotas con el brazo robótico con varios ordenadores es posible conectar el brazo vía Ethernet mediante la arquitectura JAUS, que permite la conexión cableada o inalámbrica.

2.2.1.5. Alcances y limitaciones

La siguiente figura muestra el espacio de trabajo establecido por los límites o alcance del robot. Las uniones del robot han sido limitadas para evitar autocolisiones del brazo. Esto contribuye a reducir la cantidad de soluciones redundantes del problema cinemático.

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

Tabla 4. Ángulos límite de cada módulo. [8]

Cuando aparece una restricción exterior es recomendable modificar los citados ángulos límite para evitar además de autocolisiones posibles colisiones con objetos en el entorno del brazo. Ejemplos de estas restricciones son, por ejemplo, el tronco al que va unido el brazo robótico o la superficie donde realiza su operación.

En el Anexo aparecen funciones de PCube para definir nuevos límites de los módulos. Se trata de las funciones PCube_setMinPos y PCube_setMaxPos.

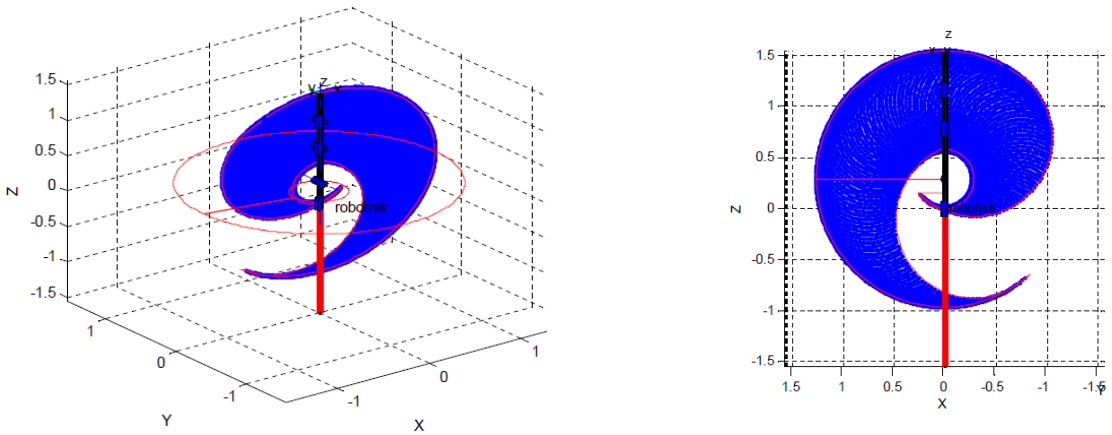


Ilustración 15. Alcance del brazo robótico sin restricciones externas.[8]

2.2.1.6. Secuencia de arranque

Para arrancar el brazo modular hay que seguir los siguientes pasos [9]:

1. Conectar la fuente de alimentación.
2. Asegurarse de que el botón de parada de emergencia no está pulsado.
3. Presionar el botón reinicio.
4. En este momento se deberá encender el PC controlador. El PC se pone en marcha y LinuxRT OS se inicia. El sistema es configurado para iniciar automáticamente los componentes de control.
5. Durante el inicio el sistema intentará conseguir la dirección IP del router mediante 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.

2.2.1.7. Botón de parada de emergencia

Este botón corta el suministro de energía de todos los módulos para evitar daño a los operadores o al entorno del robot. Dicho botón debe ser también presionado cuando se realiza cualquier trabajo sobre el robot, como cambiar cualquier componente, añadir nuevos, o en general, con cualquier caso de movimiento inesperado con el cual el robot pudiera generar peligro.

Para abandonar el modo parada de emergencia se deben seguir estos pasos [9]:

- Asegurarse de que la situación peligrosa por la cual se inició el modo de empergencia ha desaparecido.
- Tirar hacia fuera del botón de emergencia.
- Presionar ‘Restart’ en el panel de control.

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.

2.2.2. Cámaras de control

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

Ilustración 16. Cámaras utilizadas: uEye (izquierda) y Logitech Orbit (derecha)

2.2.2.1. 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 y alimentación: USB 2.0
- Tecnología del sensor: CMOS (Aptina)
- 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
- Dimensiones: Alto: 34.00 mm, Ancho: 32.00 mm, Largo: 27.40 mm
- Peso: 65.00 g

El Software de la cámara uEye es descargado de la página www.ids-imaging.com en la sección Software Package for USB and GigE uEye. 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 C, C++, C#, Microsoft .NET y Visual Basic.

Dentro de las herramientas proporcionadas están comprendidas:

- 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.

2.2.2.2. Cámara Orbit Sphere

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 fija: 3.85 mm
- Dimensiones: Alto: 109 mm, Ancho: 84 mm, Largo: 84 mm - Alimentación: USB
- Característica adicional: sensor y base motorizados.

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.

2.2.2.3. Calibración de las cámaras

Modelo pinhole

Dado que en los sistemas de monitoreo las imágenes son oblicuas, la estrategia más usada es obtener un modelo de la cámara que relaciones las coordenadas de un punto visible en la imagen con las coordenadas de dicho punto en el mundo real. Normalmente los modelos utilizados tienen en cuenta parámetros relacionados con el hardware de captura de las imágenes, llamados parámetros intrínsecos:

f : Distancia focal de las lentes

k : Coeficiente de distorsión radial de la lente

$C_{x,y}$: Coordenadas del centro de la distorsión de la lente radial

S_x : Representa la relación entre el tamaño del pixel y el del sensor CCD

u_0, v_0 : Punto del plano de la imagen por el que pasa el eje focal

Los parámetros intrínsecos pueden ser dados por el fabricante u obtenidos mediante pruebas de laboratorio.

Un modelo usualmente empleado es el modelo pinhole para cámaras (Abdel ZIZ Y Karara, 1971), basado en el principio de colinealidad, el cual asume que cada punto en el espacio es proyectado en la imagen por una línea recta que pasa por el foco de la imagen [11].

Los parámetros extrínsecos permiten obtener la orientación y posición de la cámara respecto a un sistema de coordenadas en el campo. En el modelo pinhole se definen:

$\propto, \tau, \phi$: Ángulos de Euler de rotación para la transformación entre el mundo y la cámara de coordenadas.

$T_{x,y,z}$: Componentes de traducción para la transformación entre el mundo y la cámara.

x_c, y_c, z_c : Coordenadas del centro de la imagen (coinciden con las del foco).

x, y, z : Coordenadas de un punto en el espacio (sistema de refrencia determinado)

u, v : Proyección en la imagen del punto en el plano de la imagen.

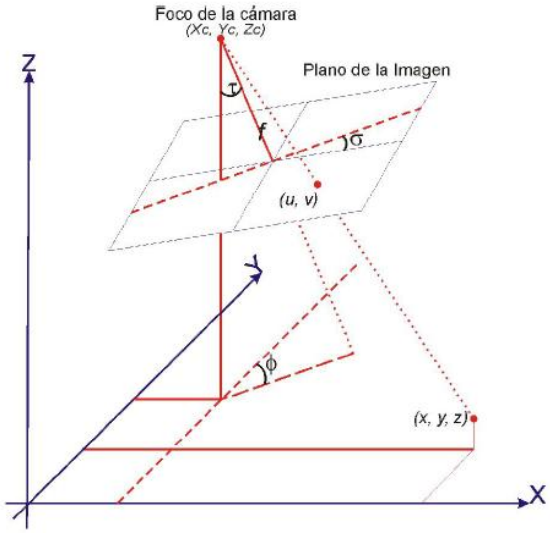


Ilustración 17. Proyección de la cámara bajo el principio de colinealidad [11]

La transformación de las coordenadas del punto respecto a un sistema referencia cualquiera (x, y, z) a las coordenadas respecto al centro de la imagen $(\hat{x}, \hat{y}, \hat{z})$ se resuelve mediante la ecuación:

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = R \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T$$

donde

$$\mathrm{R} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

$$\begin{cases} m_{11} = \cos\phi \cos\propto + \sin\phi \cos\tau \sin\propto \\ m_{22} = -\sin\phi \cos\propto + \cos\phi \cos\tau \sin\propto \\ m_{13} = \sin\tau \sin\propto \\ m_{12} = -\cos\phi \sin\propto + \sin\phi \cos\tau \cos\propto \\ m_{21} = \sin\phi \sin\propto + \cos\phi \cos\tau \cos\propto \\ m_{23} = \sin\tau \cos\propto \\ m_{31} = \sin\phi \sin\tau \\ m_{23} = \cos\phi \sin\tau \\ m_{33} = -\cos\tau \end{cases}$$

$$T^T = [x_C \quad y_C \quad z_C]$$

Los valores $m_{i,j}$ son cosenos directores por lo cual su valor numérico no varía si se usa otro sistema de rotación (Wolf, 2000).

La proyección del punto (x, y, z) en el plano de la imagen, definido por la terna de ejes coordenados (k, w, f) se expresa como:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{\hat{z}} \cdot \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$$

Si se desea obtener directamente las coordenadas en píxeles de la imagen se realiza en su lugar la multiplicación de las coordenadas de la imagen por la matriz de intrínsecos (Opencv calibration tutorial):

$$\begin{bmatrix} u_p \\ v_p \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}$$

siendo

f_x, f_y longitudes focales

c_x, c_y Centros ópticos (expresados en píxeles)

Como cualquier modelo, el modelo pinhole es una representación simplificada del sistema real (en este caso la proyección de la cámara). Por si solo no es suficiente si se requiere una alta precisión y generalmente se usa como una base que es ampliada con algunas correcciones para las coordenadas en la imagen, orientadas a contrarrestar las distorsiones sistemáticas causadas por los lentes.

Los lentes presentan dos tipos de distorsión principalmente: la radial, que causa un desplazamiento del punto real en el plano de la imagen en dicha dirección, y la distorsión tangencial que ocurre cuando los centros de curvatura de la superficie del lente no son colineales. Una expresión que permite representar el factor de corrección debido a la distorsión radial y tangencial es:

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} 2\,p_1\,u \cdot v + p_2(r^2 + 2u^2) + u(k_1 r + k_2 r^2 + \cdots) \\ p_1(r^2 + 2v^2) + 2p_2 u \cdot v + v(k_1 r + k_2 r^2 + \cdots) \end{bmatrix}$$

donde

$$p_1 \; y \; p_2 \; son \; coeficientes \; para \; la \; distorsión \; tangencial$$

$$k_1, k_2, \ldots, k_n \; son \; coeficientes \; para \; la \; distorsión \; radial$$

Teniendo en cuenta estos factores de corrección y el modelo pinhole, el modelo de proyección de la cámara se puede expresar como:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} D_u s_u (u + \delta u) \\ D_u (v + \delta v) \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix}$$

Para calcular los parámetros necesarios, tanto de la disposición interna de la cámara (intrínsecos), como de la lente (de distorsión) se emplean arreglos de laboratorio que constituyen métodos de calibración.

En el caso del presente proyecto se ha empleado el código proporcionado por Opencv para la calibración de una única cámara (también existe un código para calibrar los

parámetros necesarios para el uso de dos cámaras simultáneamente). El mismo requiere un patrón de calibración que consiste en un tablero de cuadrados blancos y negros de igual tamaño (2,5 x 2,5 mm)

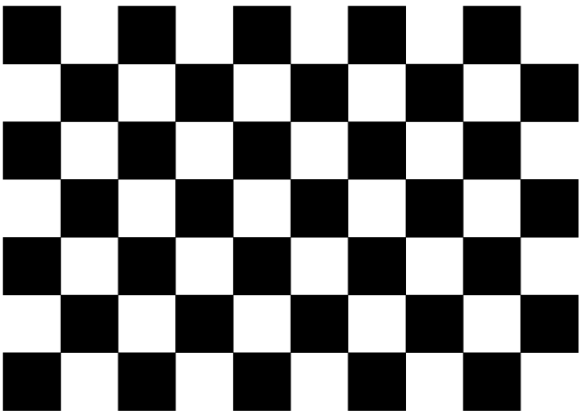


Ilustración 18. Pattern o modelo de calibración por el procedimiento de Opencv

En este procedimiento, donde aparecen puntos equidistantes y con la ubicación conocida, se puede calcular la proyección ideal de ellos en la imagen y calcular así los parámetros necesarios, intrínsecos y extrínsecos.

El método operativo es el siguiente:

1.- En primer lugar aparece una ventana que permite seleccionar el número de imágenes a tomar para la calibración o spanspots. Cuantas más capturas se realicen mayor es la fiabilidad de la calibración siendo el error menor. El valor que se ha introducido es de 20 capturas.

2.- En segundo lugar se pide el número de capturas que el programa debe ignorar entre cada par de capturas que van a ser utilizadas en el cómputo final para la calibración. El número introducido en este caso es de 90 capturas. La velocidad de captura de la cámara Logitech es de 30 fps, lo que permite un tiempo de 3 segundos entre la captura de una imagen y otra.

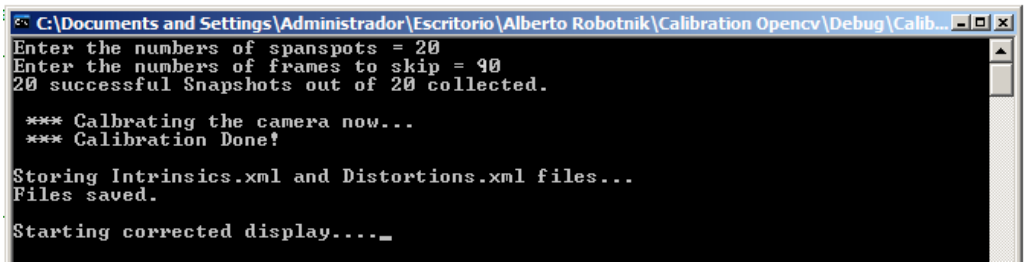


Ilustración 19. Pantalla del programa de calibración

3.- Una vez seleccionadas las condiciones de calibración se inicia una toma continua de capturas en la cámara hasta conseguir el número de spanspots

seleccionados inicialmente. Si alguna captura es errónea, es decir, si en la captura el patrón de calibración está cortado (no muestra el número total de intersecciones) o está suficientemente alejado para no permitir contabilizar los cuadros, la imagen no se utiliza para el cómputo.

4.- Tras finalizar la captura de imágenes el programa realiza las comparaciones necesarias en las imágenes para obtener los parámetros ya citados. Devuelve dos archivos de formato xml en los que aparecen los parámetros requeridos para la transformación de coordenadas: la matriz de intrínsecos y los coeficientes de distorsión. Los obtenidos para la cámara Logitech son:

$$Intrinsics: \quad \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 999.865 & 0 & 130.290 \\ 0 & 984.073 & 240.175 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Distortion: \quad [k_1 = 0.0166 \quad k_2 = -0.0233 \quad p_1 = 0.0567 \quad p_2 = -0.0337]$$

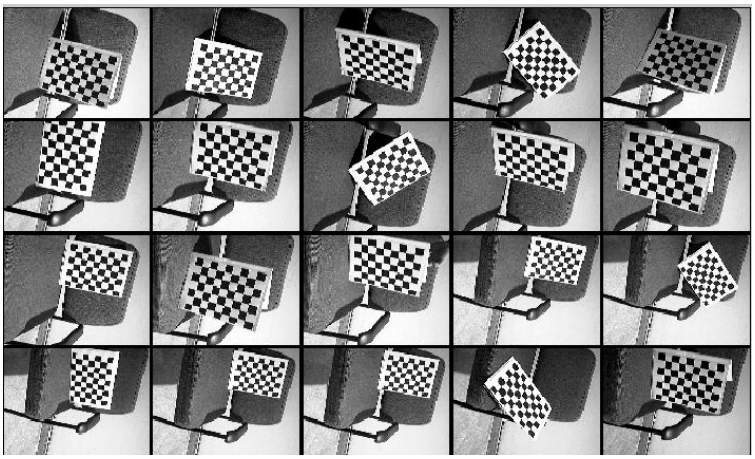


Ilustración 20. Imágenes de la calibración

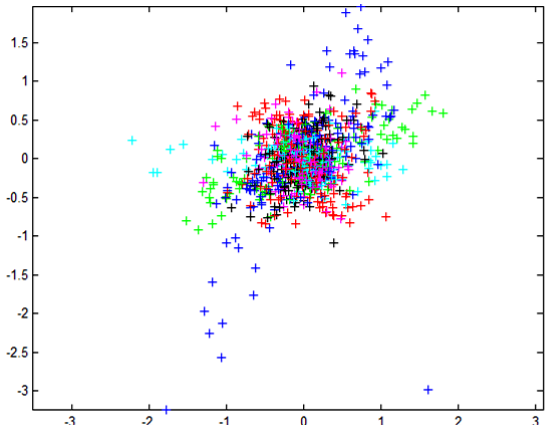


Ilustración 21. Representación del error cometido en cada punto debido a las distorsiones

Las operaciones matriciales necesarias para la transformación continua de las coordenadas en píxeles observadas por la cámara (2D) y las coordenadas en un sistema de referencia del mundo (3D) exigen que el programa de cálculo empleado admita una gran capacidad de cálculo. El programa adecuado para realizarlo es Matlab.

Ante la incapacidad de combinar Matlab con Visual Studio (y Opencv) se decidió realizar un control basado en la imagen en vez de basado en la posición. El control IBVS supone el control a partir de los píxeles de la imagen en bruto (control en 2D), lo cual implica que no es necesaria la transformación de coordenadas del mundo real a la imagen. Este control no es útil cuando se está trabajando con la totalidad de los grados de libertad del robot, ya que no aporta información acerca de la profundidad. En el caso de la aplicación del presente proyecto, en que el movimiento es lineal, la información de la profundidad es superflua. Se puede realizar este tipo de control de imagen.

Capítulo 3. Desarrollo y teoría del programa de control.

3. Introducción al capítulo.

En este capítulo se pretenden contemplar todos los aspectos que el lector debe conocer para la comprensión de aspectos clave, como son los relacionados con la cinemática en la robótica y el control visual, que han sido utilizados a lo largo del desarrollo de todo el proyecto.

3.1. Estudio cinemático del brazo robótico

3.1.1. Grados de libertad y configuraciones

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. Dicho movimiento 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. El brazo con el que se está trabajando dispone, como ya se ha comentado, de seis grados de libertad, uno por cada una de las articulaciones (cada una permite únicamente la rotación relativa).

La constitución física de la mayor parte de los robots industriales guarda cierta similitud con la anatomía del brazo humano, por lo que normalmente se usan términos del cuerpo para hacer referencia a los componentes del robot como cuerpo, brazo, codo y muñeca.

El control del brazo se traduce en el control del vector de posicionamiento que une el origen del brazo con la muñeca o herramienta (p_x, p_y, p_z) y de la terna de vectores ortogonales que definen la orientación de la misma (a, n, s). Son los grados de libertad los que permiten dicha misión. La redundancia de grados de libertad mejora la maniobrabilidad del brazo robótico [3].

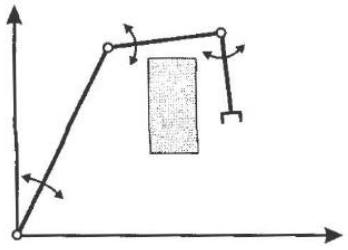


Ilustración 22. Robot planar con 3GL [3]

Atendiendo al brazo articulado de la Ilustración 23, si no hubiera estado presente el obstáculo, solo habrían sido necesarios dos de los grados de libertad para posicionar el brazo en el mismo sitio. Podemos entonces decir que el objetivo de incrementar un grado de libertad puede ser aumentar las dimensiones del área de trabajo, permitir una disposición de la herramienta en una orientación determinada o mejorar la maniobrabilidad del brazo para evitar obstáculos.

El brazo Robotnik posee 6 grados de libertad para permitir el posicionamiento y la orientación del extremo de cualquier forma. Esto también se traduce en mayor maniobrabilidad, ya que para cada posición existen diferentes configuraciones que se denominan a través de la parte del cuerpo humano a la que, por analogía, hace referencia.

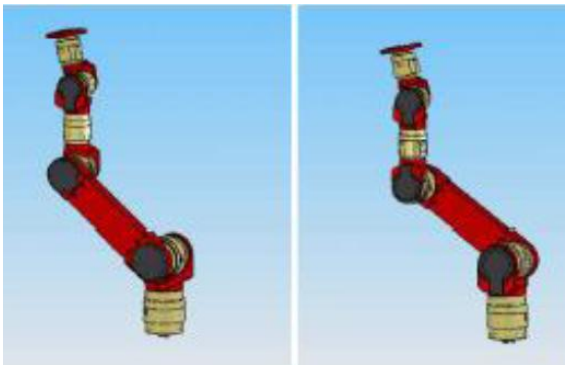


Ilustración 23. Configuraciones hombro izquierdo y hombro derecho [8]

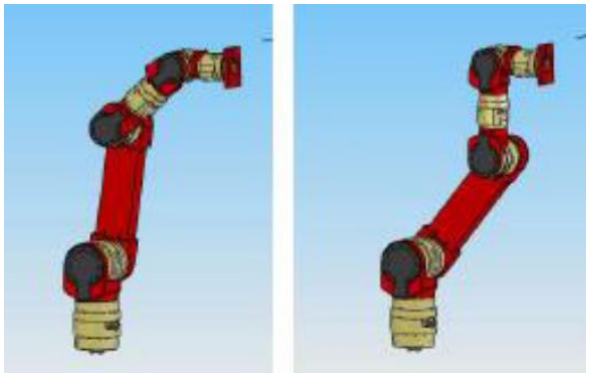


Ilustración 24. Configuración: codo arriba y codo abajo [8]

En la aplicación que se ha estudiado en el presente proyecto solo se ha hecho uso de tres grados de libertad, aquellos que permiten el movimiento en un plano (configuración planar del brazo).

3.1.2. Problema de la cinemática inversa

Existen dos problemas fundamentales en la robótica: la cinemática directa y la inversa. Ambos están basados en la relación existente entre las coordenadas articulares y la posición y orientación final del brazo.

Las diferencias entre los dos problemas es que el de la cinemática directa trata de hallar dicha posición y orientación final a través de las coordenadas relativas articulares y el problema de la cinemática inversa, por el contrario, permite obtener el valor que deben adquirir dichas coordenadas para que la herramienta de un robot alcance una posición y orientación.

En aplicaciones de control con manipuladores robóticos se utiliza la cinemática inversa para obtener los parámetros que deben adoptar las diferentes articulaciones para alcanzar una posición en el espacio determinada por un sensor.

Aunque se podría haber realizado la aplicación con dos grados de libertad se ha optado por incluir una restricción para que la recogida del objeto no estorbe a otros objetos en la cinta: la orientación vertical del último eslabón. Esto requiere un tercer grado de libertad, que consistirá en añadir el control del módulo 5 al control que se podía realizar con únicamente los módulos 2 y 3.

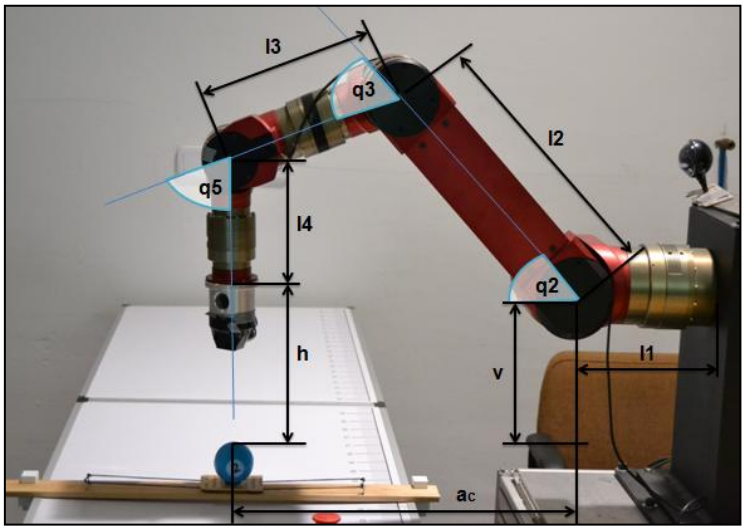


Ilustración 25. Parámetros de la cinemática inversa.

Se ha realizado el cálculo del problema cinemático inverso mediante el modelo geométrico, es decir, a través de las relaciones geométricas existentes entre los parámetros conocidos (fijos y variables) y los parámetros articulares ($q2, q3, q5$)

Parámetros conocidos fijos
$l_1 = 241,1\ mm$
$l_2 = 395\ mm$
$l_3 = 270\ mm$
$l_4 = 250\ mm$
$v = 250\ mm$: *distancia vertical del brazo*
$a_c = 600\ mm$: *alejamiento del centro*

Variables
h : *altura desde el objeto*
a : *alejamiento del objeto*

En primer lugar se define $a = a_c + desv$, siendo el último parámetro la desviación del objeto respecto al centro de la cinta.

Por el teorema del coseno se obtiene la primera relación de parámetros:

$$a^2 + (v - l_4 - h)^2 = l_2^2 + l_3^2 + 2l_2 l_3 \cos q_3$$

Se puede despejar así el parámetro articular q_3:

$$\boldsymbol{q_3} = \text{acos}\frac{(a^2 + (v - l_4 - h)^2 - l_2^2 - l_3^2)}{2l_2 l_3}$$

(en el programa se ha restado a esta expresión -0.1, que es el ángulo de desfase del enlace 3 respecto al 2 para que coincida con la definición del ángulo de la imagen)

Lo siguiente en calcular es el parámetro articular q_2. Lo haremos definiendo dos nuevos ángulos:

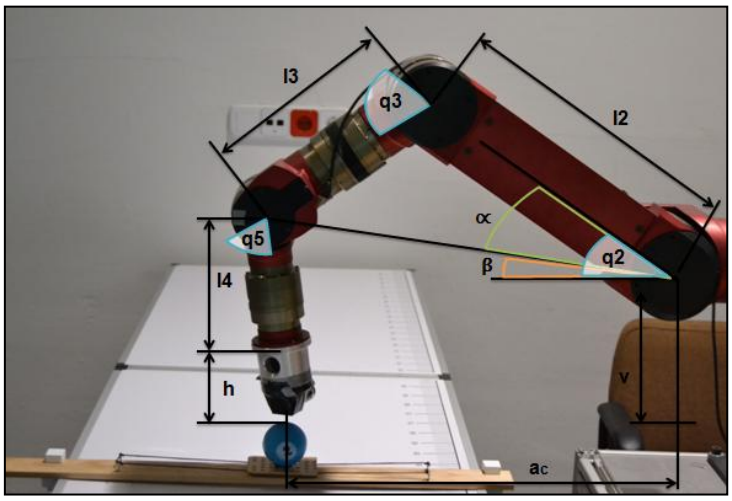


Ilustración 26. Ángulos ∝ y β auxiliares para el cálculo de q_2

$$\beta = \text{atan}\frac{(l_4 + h - v)}{a} \qquad \propto = \text{atan}\frac{(l_3 \cdot sen\, q_3)}{l_2 + l_3 \cos q_3}$$

De estos ángulos obtenemos el segundo parámetro articular:

$$\boldsymbol{q_2} = \propto + \beta = \text{atan}\frac{(l_3 \cdot sen\, q_3)}{l_2 + l_3 \cos q_3} + \text{atan}\frac{(l_4 + h - v)}{a}$$

El último parámetro articular se obtiene imponiendo la relación geométrica de verticalidad del último enlace:

$$q_2 - q_3 - q_5 = -90^{o} \qquad \rightarrow \qquad \boldsymbol{q_5} = \pi + q_2 - q_3$$

En el programa se ha implementado una función para determinar a partir de los parámetros conocidos ya descritos los parámetros articulares:

```
void Cininv(float a, float h6, float *q2, float *q3, float *q5)
{
      *q3=acos((a*a+(h-h6)*(h-h6)-l2*l2-l4*l4)/(2*l2*l4))-0.1;
      beta=atan((h-h6)/a);
      alfa=atan(l4*sin(*q3)/(l2+l4*cos(*q3)));
      *q2=alfa-beta;
      *q5=M_PI/2+(*q2)-(*q3);
}
```

3.2. Control visual

En un proceso que implica el uso de visión artificial se deben de seguir una serie de pasos desde la digitalización (adquisición de la imagen) a través de la cámara hasta la interpretación, que dependerá del control utilizado. Las etapas se muestran en el esquema a continuación:

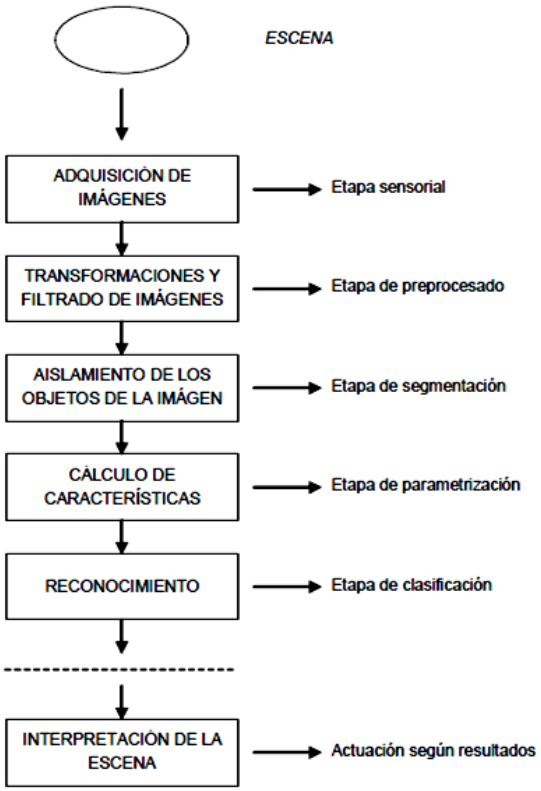


Ilustración 27. Diagrama de bloques de las etapas de un sistema de visión artificial [12]

Una vez que la imagen digitalizada ha sido obtenida, el siguiente paso consiste en el preprocesamiento de dicha imagen, con objeto de mejorarla y obtener un resultado con mayores posibilidades de éxito.

El siguiente paso es la segmentación. Su objetivo es dividir la imagen en las partes que la constituyen u objetos que la forman. El cálculo de las propiedades características de cada sección de la imagen, como el área o el centroide, se realiza en la etapa de parametrización. De esta manera se puede diferenciar una clase de objetos de otra.

En último lugar se encuentran el reconocimiento y la interpretación. El reconocimiento es el proceso que asigna una etiqueta a un objeto basada en la información que proporcionan los descriptores, mientras que la interpretación lleva a diferentes acciones basadas en las propiedades y el programa de control.

3.2.1. Conversión del color

El primer paso para el procesamiento de la imagen ha sido la conversión de RGB a HSV. Por ello se ha comenzado por explicar las diferencias entre ambos modelos.

3.2.1.1. Modelo RGB

El modelo RGB es el formado por la composición de los colores primarios de la luz. Con este modeloes posible representar un color mediante la mezcla de los colores Rojo (R), Verde (G) y Azul (B). 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.

3.2.1.2. Modelo HSV

El modelo HSV (Hue, Saturation, Value), también llamado HSB (Hue, Saturation, Brightness) se basa en la definición de cada color por la combinación de tres parámetros, que son los siguientes:

- Hue o matiz: Se representa como un grado de ángulo cuyos valores posibles van de 0 a 360°. Cada valor se corresponde a un color. En la librería de OpenCv se cuenta con 256 matices, del 0 al 255 (0-1).
- Saturation: Se representa como la distancia al eje de brillo negro-blanco. Los valores posibles van de 0 a 255 (0-1). Se entiende como la pureza del color.
- Value o Valor: Representa la altura en eje blanco-negro. Los valores posibles también van del 0 al 255 (0-1). Se entiende como intensidad de color o brillo.

El conjunto de todos los colores queda recogido en un espacio cónico o espacio de colores HSV, como se puede observar en la figura

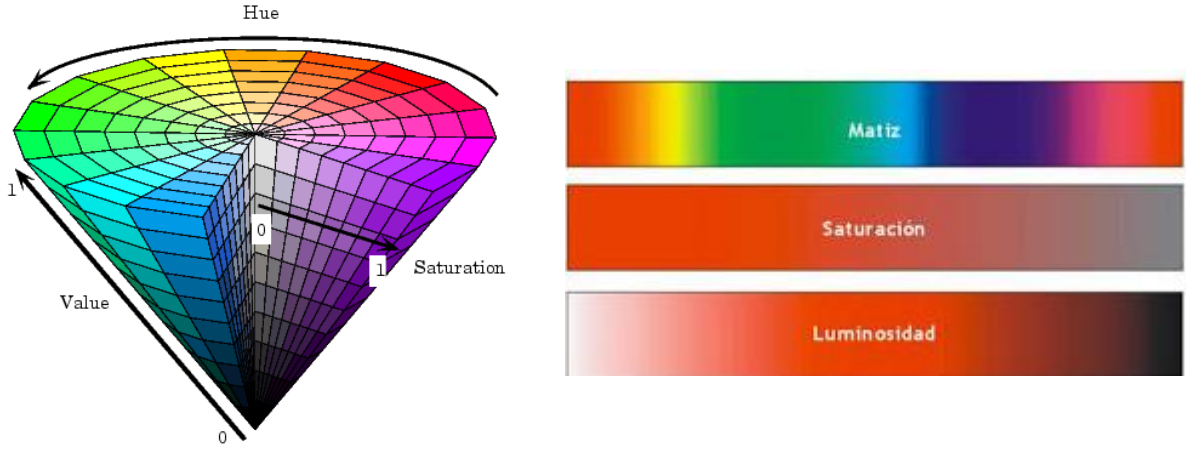


Ilustración 28. Modelo HSV. [14]

La principal ventaja de este modelo sobre el anterior es que supone una separación entre el luma, o información sobre intensidad del color o luminosidad, del croma, o la información del color. Entre otras cosas, esto permite una delimitación del rango de colores más intuitiva y permite ajustar la sensibilidad del programa ante cambios de luminosidad ambiente. [15]. Es por ello por lo que en el programa se realiza la conversión de la imagen del modelo RGB, predeterminado de OpenCv, al modelo HSV.

En el programa realizado la elección del intervalo de los valores H, S y V se realiza a través de unas barras de desplazamiento donde se ajustan los valores máximos y mínimos. También se incluyen unos valores predeterminados ofrecidos como opción al principio del programa para ahorrar tiempo de ejecución. En esta parte se ha adaptado el programa de Alejandro Sánchez para el funcionamiento en Microsoft Visual Studio

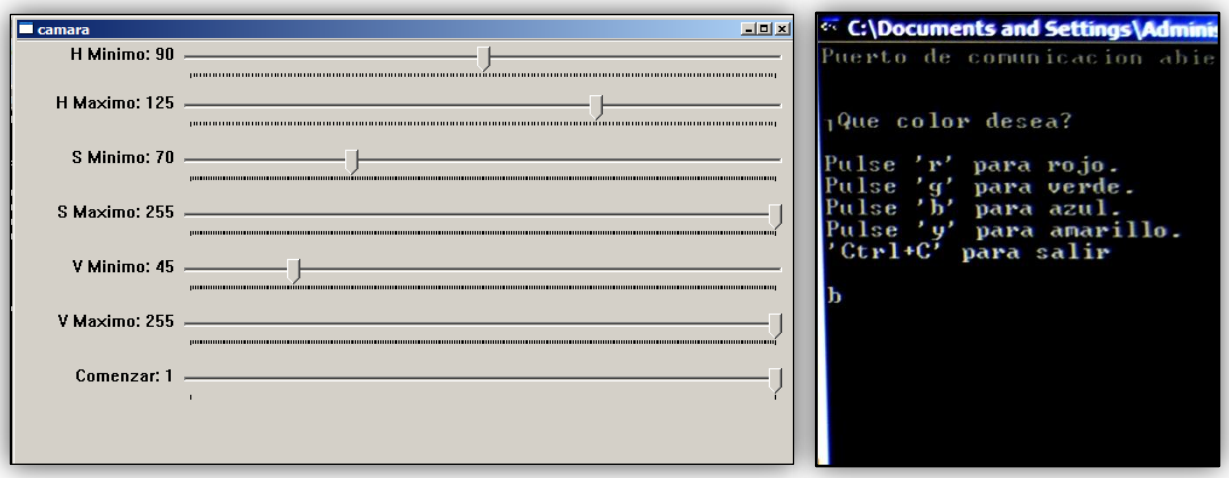


Ilustración 29. Interfaz de selección del rango de color en el programa.

A continuación, se pretende explicar el proceso de umbralizado a partir de los componentes máximos y mínimos HSV. Se incluye para ello una imagen en RGB y la descomposición de las diferentes componentes del modelo HSV por separado.

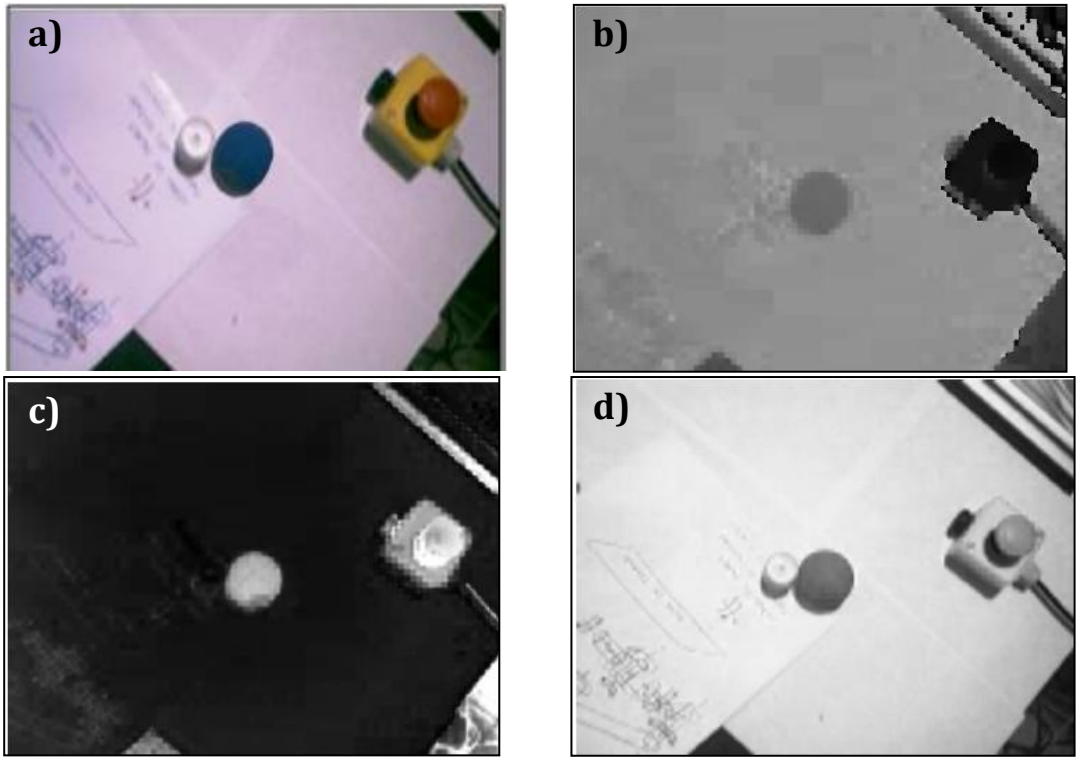


Tabla 5 (a) Imagen RGB (b) Componente Hue. (c) Componente Saturation. (d) Componente Value.

Como se ha indicado anteriormente, cada uno de los componentes del modelo o canal tiene un valor que puede variar entre 0 y 255 en la escala de grises. Para umbralizar se selecciona un rango de valores de cada uno de los componentes y se binariza como sigue: Se comprueba los valores H, S y V de cada píxel y si pertenecen todos a los respectivos intervalos seleccionados se escribe en el mismo píxel de la imagen umbralizada un 1 (objeto), mientras que si cualquiera de los valores está fuera de su intervalo se sustituye el valor por un cero (fondo). A continuación se va a mostrar las diferencias del filtrado de cada uno de los componentes:

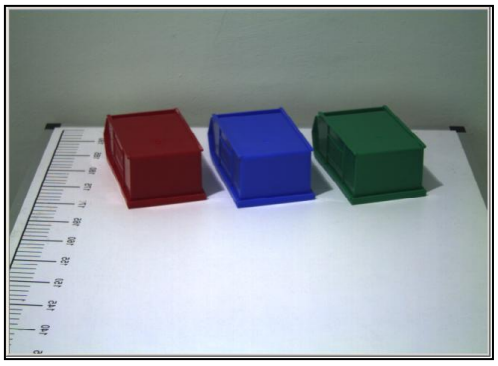


Ilustración 30. Imagen en RGB anterior al umbralizado

La imagen dada en RGB en primer lugar es acotada un rango de matiz (H) para incluir tonalidades azules únicamente. Los valores de S y V no se filtran (se selecciona todo el intervalo de 0 a 255).

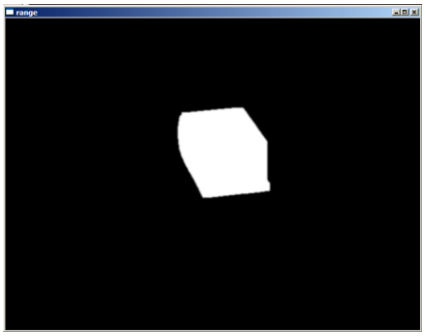


Ilustración 31. H: 85-125, S: 0-255, V: 0-255

La siguiente imagen se ha tomado para el mismo intervalo de matiz un intervalo acotado de valor (V). Esto puede ser útil cuando se pretende buscar objetos con una claridad definida. En este caso, habrá que controlar que la luminosidad sea la adecuada en la zona de trabajo del brazo robótico.

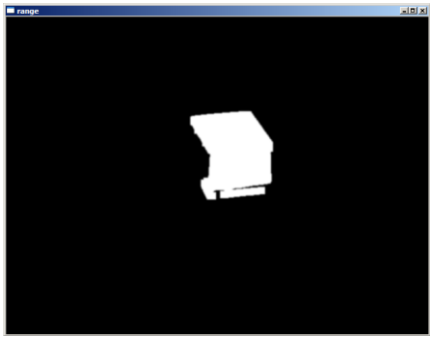


Ilustración 32 H: 85-125, S: 0-255, V: 105-255

Por último, se muestra la imagen que resulta de acotar el intervalo de saturación (S) manteniendo sin acotar el de valor. Se observa cómo la zona en la que aparece un reflejo de un color grisáceo aparece en negro en la imagen umbralizada. De esta manera, acotando el rango de saturación se pueden eliminar aquellas zonas en las que la pureza del color no es suficiente.

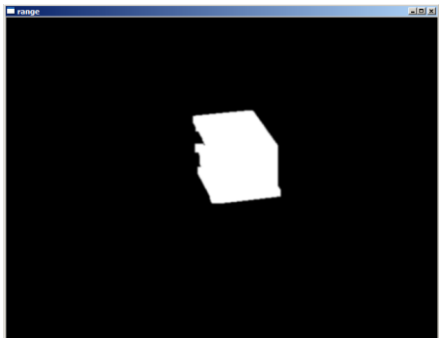


Ilustración 33. H: 85-125, S: 130-255, V: 0-255

Y a continuación se presenta la sección de código donde aparecen definidos los valores predefinidos de H, S y V para cada uno de los colores: rojo, verde, azul y amarillo.

```
printf ("¿Que color desea?\n\nPulse 'r' para rojo.\nPulse 'g' para
verde.\nPulse 'b' para azul.\nPulse 'y' para amarillo. \n'Ctrl+C' para
salir\n\n");
scanf ("%s",&color);
switch(color)
{
case 'r':   //Caso para el color rojo
h_max=10;  h_min=0;   s_max=255;  s_min=100;   v_max=255;  v_min=0;
break;
case 'g':   //Caso para el color verde
h_max=180;  h_min=0;   s_max=255;   s_min=0;   v_max=255;   v_min=0;
break;
case 'b':   //Caso para el color azul
h_max=125;  h_min=90;   s_max=255;  s_min=70;   v_max=255;  v_min=50;
break;
case 'y':   //Caso para el color amarillo
h_max=30;  h_min=20;   s_max=255;  s_min=100;   v_max=255;  v_min=80;
break;
}
```

3.2.2. Reducción del ruido

En el proceso de umbralización se consigue delimitar el área donde se encuentra el objetivo del control visual. A partir del área umbralizada se pueden realizar tareas como el cálculo del momento del objeto y, por tanto, de la posición del centro de masas. Pero aún después de la umbralización existen píxeles aislados en toda la imagen (que tienen el valor 1 en el entorno o el valor 0 dentro del objetivo) que deben ser eliminados para realizar ciertas operaciones, como la segmentación.

Los citados píxeles aislados son considerados como ruido de la imagen y es conveniente eliminarlos. Para ello se utilizan tanto herramientas de filtrado como de morfología matemática. En el presente proyecto se han aplicado únicamente operaciones morfológicas para obtener la imagen final, pues era suficiente para obtener una imagen sin píxeles aislados. La erosión sirve para eliminar el ruido y la dilatación, por su parte, para compensar los píxeles del objeto que son eliminados con la erosión. El uso de estas dos operaciones combinadas se conoce como apertura.

3.2.2.1. Erosión. [16]

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

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.

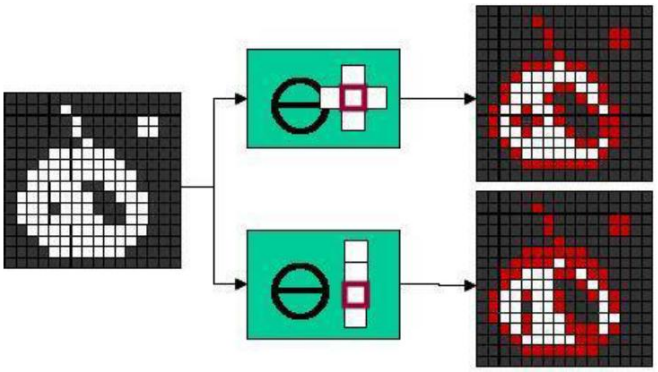


Ilustración 34. Operación de erosión [16]

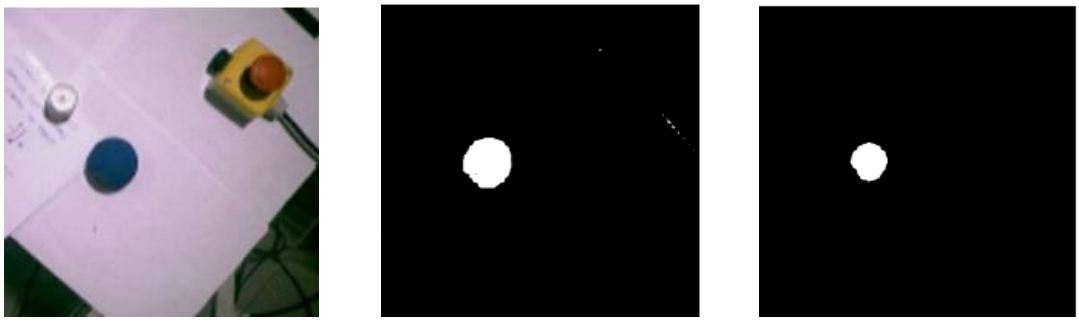


Ilustración 35. Ejemplo de erosión. Imagen original (izq), imagen umbralizada (cen) e imagen erosionada (der)

3.2.2.2. 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.

Propiedades:

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

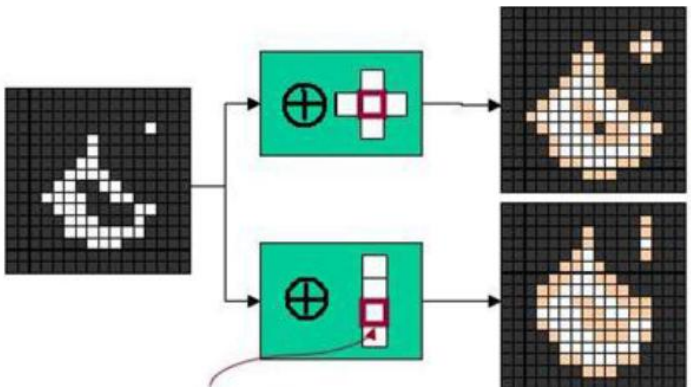


Ilustración 36. Operación de dilatación [16]

Como ya se ha comentado anteriormente, la combinación secuencial de una operación de erosión con una posterior de dilatación se conoce como la técnica de apertura, que es la que se ha aplicado en el programa de control visual en una etapa posterior al umbralizado y anterior al procesamiento. Se puede representar como sigue:

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

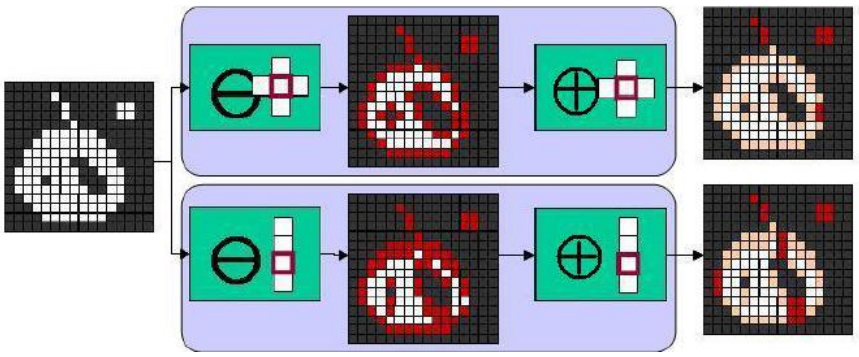


Ilustración 37. Técnica de apertura [16]

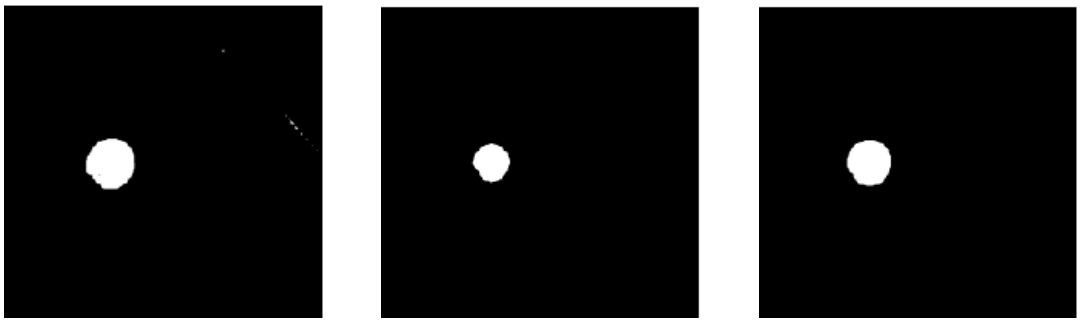


Ilustración 38. Ejemplo de apertura. Imagen umbralizada (izq), imagen erosionada (cen), imagen erosionada y dilatada (der)

A continuación, se muestran las líneas de código que se requieren desde la captura de la imagen desde la cámara (etapa sensorial) hasta el preprocesado de la imagen. La librería Opencv proporciona funciones a muy alto nivel que permiten realizar las operaciones morfológicas anteriormente explicadas en una sola línea de código.

```
captura=cvQueryFrame(capture);  //Función que toma un frame de la
imagen y lo almacena en captura.
cvResize(captura, img, CV_INTER_LINEAR);   //Escalar la imagen
capturada mediante interpolación lineal (esto se realiza para la
cámara uEye, pues el programa no admite trabajar con una
resolución de 1240x1024 y hay que redimensionar). Guardar la
imagen en img (640x480)
cvShowImage("camara",img);  //Muestreo de la imagen RGB


////SE UMBRALIZA SEGÚN LOS VALORES HSV INTRODUCIDOS///
AjusteHSV();  //Ajuste HSV automático, según el color introducido
cvCvtColor(img, hsv_frame, CV_BGR2HSV); //Conversión de BGR a HSV
IplImage* thresholded = ThresholdedColor(hsv_frame); //Umbralizar


////FUNCIÓN RGB-HSV-THRESHOLD CREA UNA IMAGEN BINARIA UMBRALIZADA
IplImage* ThresholdedColor(IplImage* hsv_frame)
{
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);  //Aplica los intervalos de Hue,
Saturation y Value a la imagen, es decir, la umbraliza en HSV
return thresholded;
}


/////////OPERACIONES MORFOLÓGICAS SOBRE LA IMAGEN UMBRALIZADA///
IplConvKernel *se11 = cvCreateStructuringElementEx(11, 11, 5,  5,
CV_SHAPE_RECT, NULL);  //Creación del kernel o núcleo para
realizar operaciones morfológicas
cvMorphologyEx(thresholded, thresholded,0, se11,CV_MOP_OPEN,1);
     //Aplicación de una apertura (erosión-dilatación)
cvReleaseStructuringElement(&se11);   //Liberar espacio del kernel
cvShowImage("range",thresholded);
```

3.2.3. Obtención del centro de masas objetivo del control visual

El trabajo de Alejandro Sánchez permitió demostrar la mayor precisión del método del centro de masas respecto de la transformada de Hough para el posicionamiento en el control visual. Se ha comentado que el control realizado en el proyecto es control basado en la imagen, por lo que la ley de control utilizada se referenciará mediante las coordenadas en píxeles del centro de masas del objeto en la imagen.

Se parte de la definición de momentos de una función $f(x,y)$ de un objeto [17]:

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

Esta integración es calculada a partir del área del objeto. En el caso de usar imágenes binarias, 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, que depende de los índices p y q del momento $m_{p,q}$, siendo el orden igual a la suma p+q. Teniendo esto en cuenta tenemos:

- Momentos de orden 0 o área del objeto:

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

- Momentos e orden 1:

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

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

El centro de gravedad puede ser descrito por el momento de primer orden $m_{1,0}$ y $m_{0,1}$ dividido por el momento de orden 0, es decir:

$$x_C = \frac{m_{1,0}}{m_{0,0}} = \frac{m_{1,0}}{A} \qquad y_C = \frac{m_{0,1}}{m_{0,0}} = \frac{m_{0,1}}{A}$$

El cálculo del centro de masas del objeto es necesario para realizar el seguimiento del mismo. El seguimiento también se podrá realizar en caso de visión parcial del objeto, aunque debe tenerse en cuenta de que el centro de masas del objeto visto parcialmente será un centro de masas aparente, impreciso en caso de recogida del objeto.

Si hay varios objetos dispuestos en la cámara, se calcula el centro de masas del conjunto, no de cada objeto por separado (c.m. aparente). Este caso especial está resuelto mediante una función especial que será desarrollada posteriormente en el apartado de control selectivo por segmantación. Estas operaciones son necesarias si se desea que el brazo robótico realice la correcta captura de al menos uno de los dos objetos cuando atraviesan la cinta con un corto intervalo de tiempo entre ambos.

Las coordenadas (en píxeles) del centro de gravedad han sido incrustadas en la imagen mostrada por pantalla a fin de facilitar el manejo del programa a través de la información continua de dichos parámetros. Se muestra en la siguiente imagen la colocación de la cámara con el objeto centrado (píxeles 320, 240)

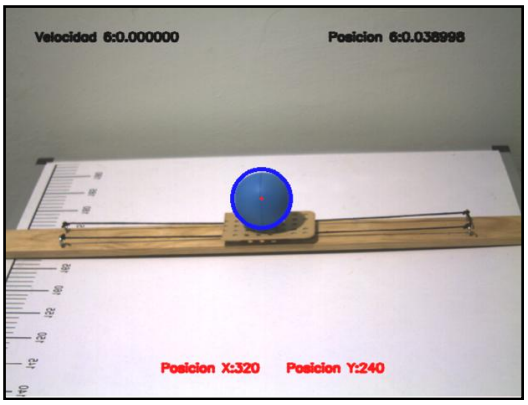


Ilustración 39. Información por pantalla en el modo Eye-in-Hand

Todo el código necesario para el cálculo del centro de masas de la zona umbralizada se muestra a continuación:

```
CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
cvMoments(thresholded, moments, 1);
double moment10 = cvGetSpatialMoment(moments, 1, 0);  //Momento1,0
double moment01 = cvGetSpatialMoment(moments, 0, 1);  //Momento0,1
area = cvGetCentralMoment(moments, 0, 0);  //Momento de orden 0
//Cálculo de las coordenadas del centro geométrico
y = moment01/area;    x = moment10/area;
if(lastX>=0 && lastY>=0 && x>=0 && y>=0)
{
cvLine(img,cvPoint(x,y),cvPoint(lastX,lastY),cvScalar(0,0,255),4);
//Dibuja una línea roja que representa la trayectoria del objeto
}
```

```
lastX = x;       //Último valor de la coordenada x
lastY = y;       //Último valor de la coordenada y
}
int radio;   radio=sqrt(area/M_PI);  //Radio de la circunferencia
if(area>200)
{
cvCircle(img,cvPoint(x,y),radio,CV_RGB(0,0,255),3,8,0);  //Se crea una circunferencia alrededor del centro geométrico del objeto
cvShowImage("camara",img);  //Img con trayectoria y circunferencia
}

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

3.2.4. Recogida de objetos.

Este apartado engloba todas las funciones necesarias para la recogida de cualquier objeto correctamente umbralizado que circule a cualquier distancia del brazo y una velocidad limitada en el espacio delimitado por la mesa que simula una cinta transportadora.

Se han realizado dos tipos de control diferentes que han sido desarrollado en la introducción: control Eye-in-Hand y control Eye-to-Hand. En un principio se pretendía realizar ambos programas con la cámara Logitech, ya que la otra cámara disponía de una resolución excesiva que no permitía que el programa se ejecutara correctamente, pero surgió un problema con el uso de dicha cámara en el extremo del brazo robótico que obligó a instalar y buscar una solución con la cámara uEye.

El problema que surgió con la cámara Logitech es el siguiente. La cámara dispone de un motor que permite rotar la lente en el interior de la carcasa de la cámara pero no permite controlar mediante ningún programa su posición. Solo permite control manual. De esta manera, cuando se realizaba un movimiento rápido con el brazo robótico la inercia provocada en la cámara del extremo del brazo afectaba al motor interior y provocaba un desajuste.

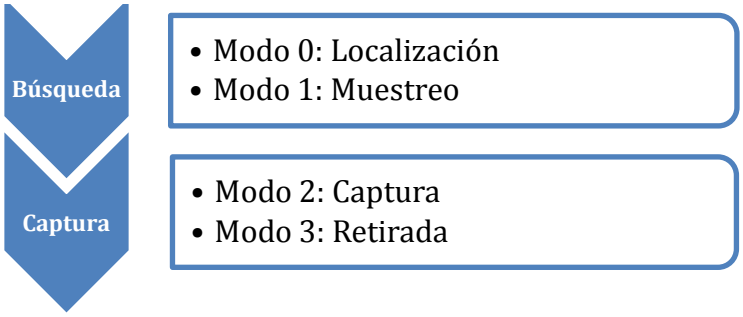
3.2.4.1. Modo de control Eye-in-Hand

Para llevar a cabo este modo de control se ajustó la cámara uEye en el extremo del brazo robótico tal como se muestra en la siguiente imagen.

Ilustración 40. Disposición de la cámara uEye en modo Eye-in-Hand

Como ya se ha explicado en la introducción, la demora de los suministradores de la pinza final del brazo obligó a realizar el proyecto sin hacer uso de la misma. Por este motivo se pudo situar la cámara justo en el extremo. En caso de que hubiera llegado la pinza la cámara se habría ajustado antes del extremo. En caso de robot industriales la cámara puede ir integrada en el mismo brazo.

El programa fue realizado mediante el método búsqueda-captura. Las fases o modos del programa de los que consta son los siguientes:



3.2.4.1.1. Modo búsqueda

Esta es la etapa inicial, en la cual el objetivo consiste en obtener distintos parámetros de entrada para llevar a cabo la correcta recogida del objeto. Se trata del alejamiento del objeto respecto del origen de coordenadas del brazo robótico, dispuesto en su base, y la velocidad que lleva a través de la misma. La disposición del brazo en la posición inicial se muestra a continuación.

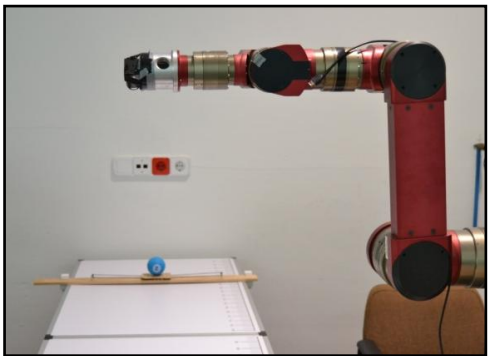


Ilustración 41. Posición de los distintos módulos en el modo búsqueda

Los valores de posición inicial de los distintos módulos son los siguientes:

	Módulo 1	**Módulo 2**	**Módulo 3**	**Módulo 4**	**Módulo 5**	**Módulo 6**
Posición angular (rad)	-0.05	1.57	1.49	0	0	0 (variante)

En este apartado se sigue una ley de control que permite el seguimiento del objeto por la cámara a través del movimiento del módulo 6 (giro alrededor del eje del brazo) para así permitir la recogida de los datos necesarios.

El control se basa en el objetivo de minimizar el error entre la posición actual en píxeles y la posición deseada (para el seguimiento emplearemos el centro de la imagen) tomadas por tanto directamente de la cámara.

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

siendo s* nuestra posición deseada, el centro de la imagen (320,240).

Igual que muchas de las leyes de control que existen, que se basan en iteraciones para el cálculo de la velocidad de todos los módulos, en este proyecto se ha pretendido realizar un cálculo sencillo y aproximado. Como no es posible trabajar con velocidades y tiempo a la vez (no lo permite la librería m5api), se ha optado por iterar el error de la posición y calcular por ensayo y error una velocidad proporcional al error que permita un seguimiento adecuado.

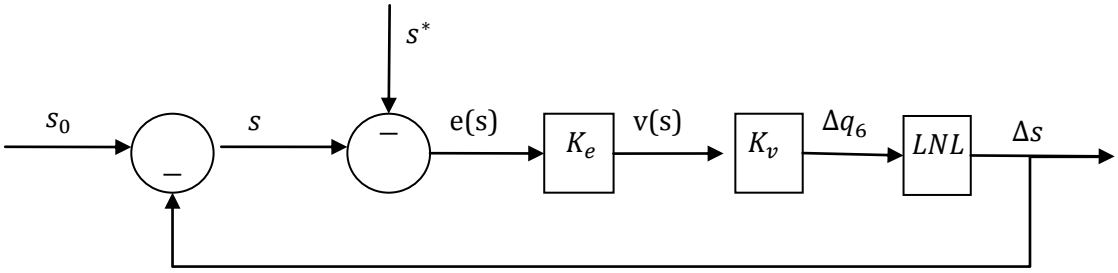


Ilustración 42. Esquema de control en modo Eye-in-Hand

Las constantes obtenidas por ensayo y error que permiten realizar un seguimiento adecuado del objeto son: $K_e = 0{,}02 \;\; y \;\; K_v = 0{,}01$. La relación que guarda la variación de la posición del módulo 6 con la diferencia de posición en la pantalla se traduce en una ley no lineal (será mayor cuanto más cerca esté el objeto) que no es preciso determinar ya que el modelo es aproximado y nos hemos asegurado que la toma de datos sea lo suficientemente alejada del robot para que la variación sea mínima y no afecte a la toma de medidas.

Como aceleración se ha tomado la máxima permitida por el brazo robótico: $a_{cc} = 2.7925\ rad/s^2$. La ley de control queda recogida en este segmento de código:

```
ey=ALTO/2-y; //Error de coordenada y respecto al centro
vel_y=ey/50;  //Control de velocidad en función del error anterior
incremento_y=vel_y/100;   //Incremento de la posición del módulo 6
PCube_getPos(dev, 6,&pos_act6);  //Obtenemos la posición actual
del motor 6, que se muestra en pantalla.
pos6=pos_act6+incremento_y;     //Posición a la que ha de moverse

if ((vel_y<0 && modo==0) || (vel_y<0 && modo==1))
 //Si la velocidad calculada es <0, módulo 6 en sentido horario
{
PCube_moveRamp( dev, 6, pos6, -vel_y, acc );
//Función que aplica velocidad en forma de Rampa
}
else if ((vel_y>0 && modo==0) || (vel_y>0 && modo==1))
//Si la velocidad calculada es >0, módulo 6 en sentido antihorario
{
PCube_moveRamp( dev, 6, pos6, +vel_y, acc );
}
else {}   //Si la velocidad calculada es=0 o no nos encontramos
en modo búsqueda no hay que realizar ninguna acción
```

Ensayo de tiempo centralización de la imagen

Con este ensayo se muestra la respuesta de control del brazo situado inicialmente en su posición inicial y estimulado por un objeto que se encuentra desplazado 150 píxeles en coordenada vertical.

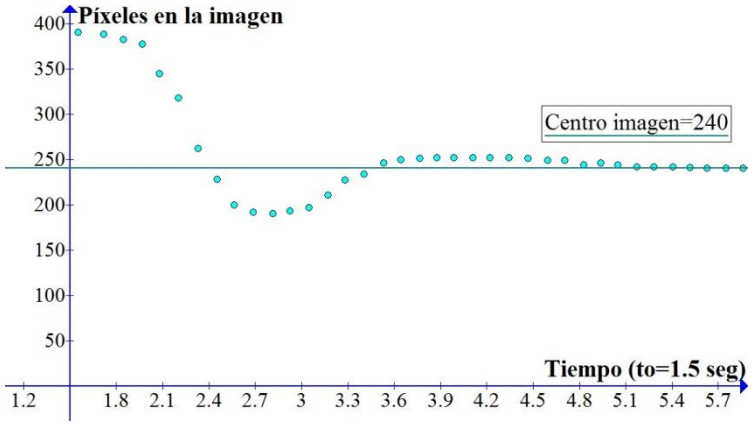


Ilustración 43. Ensayo de tiempo de centralización

Mediante este ensayo obtenemos que el tiempo de centralización de la imagen para un elevado alejamiento es aproximadamente igual a 2 segundos, una rápida respuesta que ha permitido un correcto funcionamiento a bajas velocidades de la cinta.

Forma de hallar el alejamiento del objeto

La toma del valor de alejamiento se realiza a una distancia determinada, que se ha determinado que sea a 130 mm del brazo. En ese punto, se hallan los píxeles de diferencia entre el centro de masas del objeto y el centro de la cinta. La cámara debe estar fija (por simplicidad se ha fijado su centro al de la cinta) y no desplazarse. Es por esto que la cámara Logitech fue sustituida. Una vez se halla la diferencia en píxeles se realiza la correspondencia para hallar la distancia en milímetros:

	Extremo izquierdo	Extremo derecho
Píxeles de la imagen	135	495
Alejamiento del centro (mm)	-16	+16

Tabla 6. Correspondencia de la distancia al centro en píxeles y milímetros

Las operaciones que realiza el programa según la correspondencia citada anteriormente se muestran en el siguiente segmento de código:

```
if(0.066<pos_act6 && pos_act6<0.0695 && modo==0)
//Intervalo aproximado en el que se puede encontrar el módulo 6 para que el objetivo esté fijo a un objeto que diste aproximadamente 130 mm del brazo
{t_130 =clock();
desv=x-ANCHO/2;
printf("\nLa desviacion respecto al centro de la cinta es %f \n\n", desv);
modo=1;   //Se pasa al muestreo de velocidades
}
a=ac+desv*32/(495-135);    //Conversión de la desviación hasta el objeto de píxeles a distancia real
ac es la distancia entre el brazo y el centro de la cinta. Está representado en el apartado de cinemática del proyecto.
```

Muestreo de velocidades para hallar de manera aproximada la velocidad del objeto en la cinta

Para ello se han definido los siguientes parámetros: t_130, t_120, t_110, t_100:

Estos parámetros representan los instantes de tiempo en los que el objeto atraviesa las posiciones determinadas a 130, 120, 110 y 100 milímetros desde la base del brazo robótico.

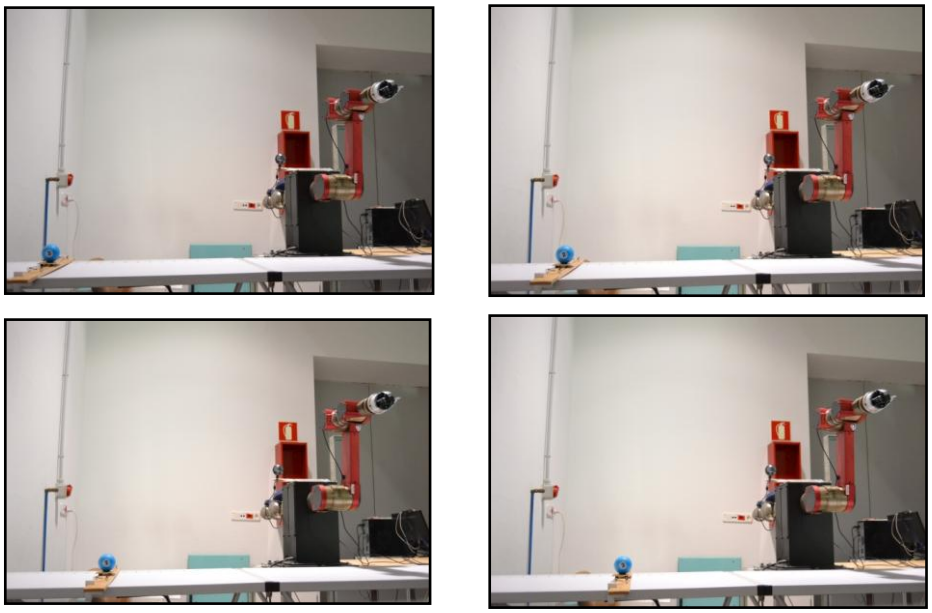


Tabla 7. Posiciones de muestreo de tiempo para el cálculo de la velocidad.

El método es aproximado, ya que el cálculo de los momentos no es un proceso continuo en el tiempo, sino que requiere la completa ejecución del programa y puede que durante este proceso el objeto haya sobrepasado el intervalo de posición que se había determinado. Por ello, el cálculo de la velocidad es preferible hacerlo de la siguiente manera.

Solo se toma cada muestra de velocidad si los instantes de tiempo correspondientes son distintos a cero (la inicialización). Cuando una de las velocidades es igual a cero solo se va a considerar la otra muestra de velocidad. En cambio, si se toman satisfactoriamente las dos muestras se realiza la media. El código es el siguiente:

```
if(modo==1)  //Cálculo de la velocidad del objeto
{
if(0.0325<pos_act6 && pos_act6<0.0355 && v1==0)
{t_120=clock();       v1=10/(t_120-t_130)*(double)CLOCKS_PER_SEC;
printf("Primera muestra de velocidad: %f [cm/s] \n", v1);}
if(-0.0035<pos_act6 && pos_act6<-0.001 && v2==0)
{t_110=clock(); printf("--\n");}

if(-0.046<pos_act6 && pos_act6<-0.043 && v2==0)
{t_100=clock();
if(t_110!=0)
{v2=10/(t_100-t_110)*(double)CLOCKS_PER_SEC;
```

```
printf("Segunda muestra de velocidad: %f [cm/s] \n", v2);}
CptPos();
modo=2;

if(v1>0 && v2>0 && vm==0) {vm=(v1+v2)/2;}
else if(v1==0 && vm==0) {vm=v2;}
else if(v2==0 && vm==0) {vm=v1;}
printf("\nLa velocidad media del objeto es: %f [cm/s]\n\n", vm);
}
}
```

3.2.4.1.2. Modo captura

Una vez que se ha calculado la velocidad que lleva el objeto el programa se encarga de situar el módulo 6 enfocando una posición más avanzada de la cinta. Solo cuando el objeto supera la frontera metálica de la cinta situada a 50 mm de donde está situado el brazo comienza el descenso. Por ello el tiempo de descenso se calcula como $50/v_m$.

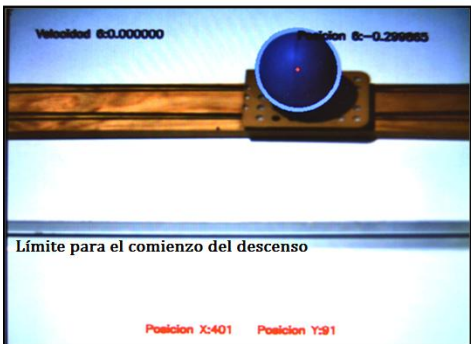


Ilustración 44. Línea de inicio del descenso del brazo.

Como ya se ha comentado, el descenso del brazo depende tanto de la velocidad que lleve el objeto como del alejamiento respecto de la base del brazo.

- El alejamiento se introduce en la función 'Cininv', ya definida en la cinemática inversa: **Cininv(a,h,&q2,&q3,&q5).** La función devuelve el valor de los parámetros modulares que ha de tomar el brazo para alcanzar la posición deseada.
- El tiempo de descenso se introduce justo a los parámetros modulares en la función 'PCube_moveStep'. En este caso la velocidad es aplicada en forma de escalón y no de rampa: **PCube_moveStep(dev, 3, q3, t_capt).** Se aplica la misma función en los módulos que participan en la captura, devices 2, 3 y 5.

Una vez capturado el objeto (en el caso en que hubiera una pinza en el extremo) se procede a la retirada del mismo. En toda la fase de captura son necesarias las funciones cuyo código se refleja a continuación:

Fúnción para alcanzar la posición deseada:

En el código del programa cada movimiento es accionado por una sentencia, pero el programa no espera a que el brazo con su velocidad ejecute el movimiento sino que continúa ejecutando la siguiente sentencia. La siguiente función obliga al programa a esperar a que el brazo robótico se haya situado prácticamente (admite un error de $\pm$0,001 radianes para cada módulo) en la posición deseada antes de continuar con la siguiente orden. Los parámetros de entrada son tanto la posición actual del brazo, que se va actualizando continuamente, como la deseada.

```
void GetPosition(float q2, float q3, float q5, float pos_act2,
float pos_act3, float pos_act5)
{
while(abs(pos_act2 - q2) > 0.001 && abs(pos_act3 - q3) > 0.001 &&
abs(pos_act5 - q5) > 0.001)
{
PCube_getPos(dev, 2,&pos_act2);
PCube_getPos(dev, 5,&pos_act5);
PCube_getPos(dev, 3,&pos_act3);
}
}
```

Función para mantener el brazo en una posición durante un tiempo de espera definido

El único parámetro que es por tanto necesario introducir en esta función es el tiempo de espera en segundos.

```
void Espera(float tespera)
{
paro=0;
inicioesp=clock();
while(paro < tespera) //Tiempo en que el robot está parado
{
ahora=clock();
paro=(ahora-inicioesp)/(double)CLOCKS_PER_SEC;
}
}
```

3.2.4.2. Modo de control Eye-to-Hand

En este segundo caso la cámara fue dispuesta en el tronco que sirve de bastidor del brazo robótico.

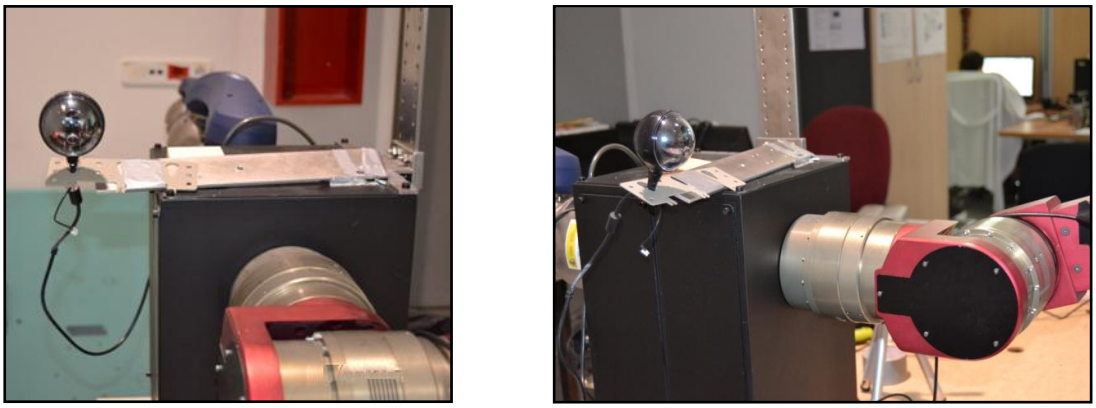


Ilustración 45. Disposición de la cámara en modo Eye-to-Hand

La ventaja de este modo de control es que al estar la cámara fija dispones de una referencia visual que no sufre traslaciones ni rotaciones en el espacio. Puedes conocer los instantes exactos en los que el móvil transcurre por determinados puntos, que pueden servir como límites para determinadas acciones con el brazo. Por tanto, la tarea a realizar en este modo es definir correctamente estos límites.

Se ha realizado un control por pasos, lo cual supone un descenso escalonado del mismo. Al primer paso se llega cuando el móvil supera la línea de 80 milímetros dispuesta sobre la mesa (que corresponde aproximadamente a la coordenada x=500 píxeles). El robot se dispone entonces en una posición intermedia antes de la captura. Se trata de una disposición central, que dista más o menos la misma distancia de cualquier punto en todo lo ancho de la cinta.

Además de servir como límite para pasar del step 0 al step 1, en el punto mencionado se toman también los parámetros necesarios para la correcta recogida del móvil: el alejamiento del objeto y el punto de recogida. La velocidad en este caso no es necesaria conocerla, ya que en este modo es posible la anticipación del brazo en la adecuada posición y el accionamiento rápido de la pinza en el punto exacto visto desde la cámara y correspondiente a la base.

<u>Alejamiento respecto del origen del brazo</u>

Como ya se ha visto en el apartado anterior, es necesario introducir este parámetro en la función de la cinemática inversa para obtener las posiciones angulares que deben tomar los distintos módulos para alcanzar correctamente al móvil.

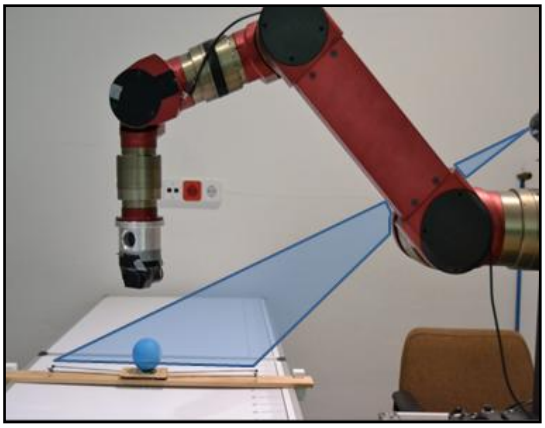


Ilustración 46. Toma del alejamiento en modo Eye-to-Hand

La relación entre la posición en píxeles de la imagen y el alejamiento del objeto no es lineal debido a la perspectiva. Es por ello por lo que se ha buscado obtener otro tipo de correlación entre otros métodos muestrales.

Se comenzó buscando una correlación basada en el polinomio de interpolación de **Lagrange.** En los textos de análisis numérico se define al polinomio de interpolación de Lagrange, correspondiente a $n+1$ valores dados, como aquella función polinomial de grado a lo sumo n que toma sobre los $n+1$ diferentes valores numéricos $\{x_0, x_1, \dots, x_n\}$, los $n+1$ valores dados $\{y_0, y_1, \dots, y_n\}$. Se propone una expresión del polinomio de la forma $a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$, en la que los coeficientes $a_0, \dots, a_n$ se calculan sabiendo que deben satisfacer el siguiente sistema de ecuaciones [17]:

$$a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n = y_0$$
$$a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n = y_1$$

$$\dots$$

$$a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n = y_n$$

A continuación se presentan los datos que se han recogido (n=17) haciendo uso de la interfaz gráfica del programa.

Píxeles en la imagen	251	266	278	290	301	313	324	335	345	355	364	374	384	392	401	410	420
Distancia desde la base	40	42	44	46	48	50	52	54	56	58	60	62	64	66	68	70	72

Tabla 8. Datos muestrales para interpolar

El polinomio ha sido obtenido mediante el uso del programa '*polint*' en el cual se introduce la tabla muestral anterior a través de dos arrays. La representación del polinomio se muestra a continuación:

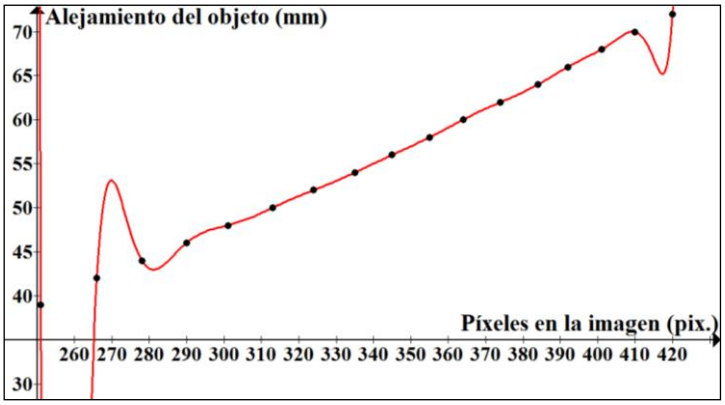


Ilustración 47. Interpolación por Lagrange

El polinomio obtenido representa el único de grado 18 que atraviesa todos los puntos anteriores. Lagrange no introduce ninguna condición que controle el suavizado de la función, por lo que en algunos casos (os extremos principalmente) la función difiere mucho de los puntos obtenidos. No es un método por tanto fiable.

El siguiente método probado sí introduce la condición de función suavizada. Este es conocido como el método de los **splines cúbicos interpolantes**. Consiste en interpolar con n funciones en los intervalos $[x_i, x_{i+1}], con\ i = 0,1, \dots, n-1$, que son polinomios de grado 3. Entonces

$$s_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \quad i = 0,1, \dots, n-1$$

Por lo que tenemos 4n incógnitas a determinar. Por otra parte, el spline tiene que cumplir las siguientes condiciones:

(i) Condiciones de interpolación:

$$s_i(x_i) = f(x_i), \qquad i = 0,1, \dots, n-1 \qquad s_{n-1}(x_n) = f(x_n)$$

(ii) Condiciones de continuidad (no se cumplen en los extremos):

$$s_i(x_{i+1}) = s_{i+1}(x_{i+1}), \qquad i = 0,1, \dots, n-2$$

(iii) Condiciones de suavidad (no se cumplen en los extremos):

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \qquad s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}), \qquad i = 0,1, \dots, n-2$$

Se obtienen en total $4n-2$ ecuaciones, lo que significa que para determinar el spline s(x) se deben imponer 2 condiciones adicionales. En el caso de spline natural son:

$$s_0''(a) = 0 \qquad s_{n-1}''(b) = 0$$

Haciendo uso del programa natural_spline de Mathworks [20] para la ejecución en Matlab se han obtenido los siguientes polinomios acotados en los 17 intervalos:

i	x_i	x_{i+1}	***Polinomio de tercer grado*** (a, b, c, d)
0	251	266	$(0.00004, -0.03445, 8.77029, -715.37004)$
1	266	278	$(-0.000083, 0.06834, 18.57305, 1709.07317)$
2	278	290	$(0.00007, -0.061127, 17.41959, -1626.24515)$
3	290	301	$(-0.0001, 0.09366, 27.46787, 2712.87698)$
4	301	313	$(0.0001, -0.089422, 27.63912, -2816.19108)$
5	313	324	$(-0.00007, 0.07375, 23.43297, 2512.33022)$
6	324	335	$(0.00008, -0.07879, 25.98966, -2825.31448)$
7	335	345	$(-0.0001, 0.09774, 33.15977, 3779.70523)$
8	345	355	$(0.00012, -0.12637, 44.17126, -5113.36338)$
9	355	364	$(-0.00017, 0.18793, 67.4057, 8089.91068)$
10	364	374	$(0.00004, -0.0518, 19.85584, -2497.82273)$
11	374	384	$(0.0002, -0.22839, 85.89865, -10731.15951)$
12	384	392	$(-0.00039, 0.45692, 177.261331, 22953.31786)$
13	392	401	$(0.00021, -0.25101, 100.250435, -13308.21966)$
14	401	410	$(-0.000136, 0.16561, -66.814715, 9022.82205)$
15	410	420	$(0.00007, -0.08918, 37.651263, -5254.19487)$

Tabla 9. Splines cúbicos por intervalos.

La interpolación por spines cúbicos puede ser representada en un gráfico, que se ajusta perfectamente a la distribución de puntos, que se muestra a continuación:

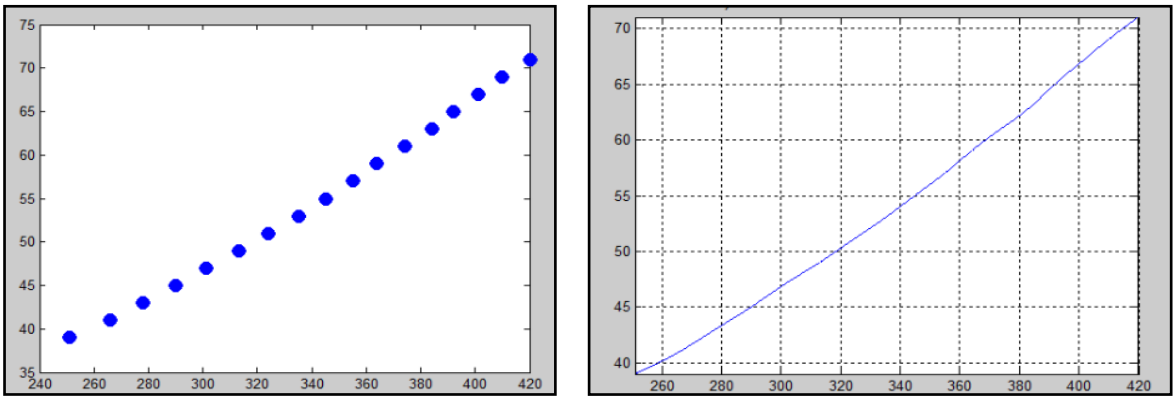


Ilustración 48. Gráfico de interpolación por splines cúbicos.

Otro modo de resolverlo es mediante regresión. Este caso se diferencia de los anteriores en que la función obtenida no tiene que pasar por todos los puntos (no se trata de una interpolación). Se presentan los resultados de regresión de grados 1 y 2.

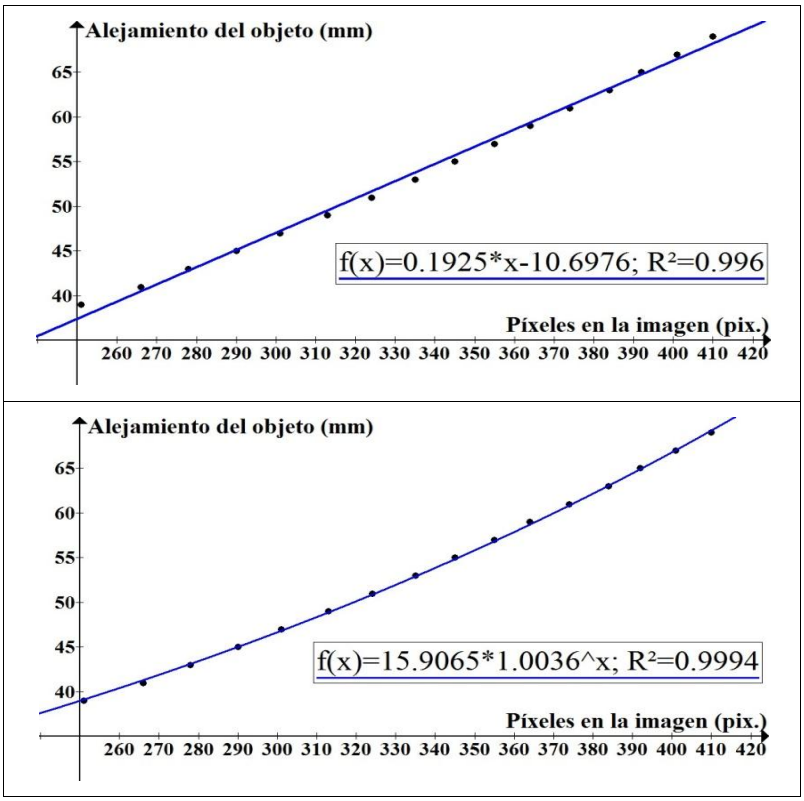
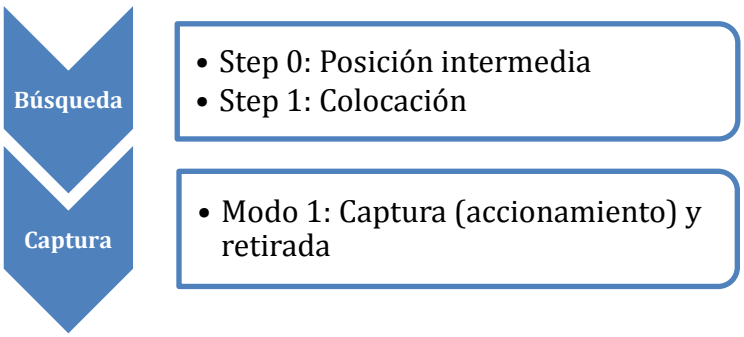


Tabla 10. Correlación lineal y exponencial

Tanto el método de los splines cúbicos como la regresión exponencial generan muy buenos resultados. Por otro lado, la regresión lineal genera en algunos puntos errores de hasta un centímetro que son inaceptables para la recogida del objeto. Esto se refleja en el parámetro de confianza del ajuste ($R = 0.996$ lineal frente a $R = 0.9994$ del ajuste exponencial).

El límite para pasar del step 1 al modo captura se corresponde con la coordenada x=300 en la imagen de la cámara. En ese punto se procede a la colocación anticipada del brazo robótico sobre el móvil para su posterior recogida. Se hace uso de la función anteriormente descrita '***Cininv***'.

El resumen del desarrollo del programa queda reflejado en el siguiente esquema:



Punto de recogida del objeto

Es tan importante saber cuándo se debe accionar la pinza del brazo robótico para recoger el objeto como saber a qué distancia del mismo se debe colocar. Por tanto, a cada alejamiento le corresponderá un punto en coordenada x para el accionamiento de la pinza. Es para ello necesario de la misma manera obtener una correlación de un conjunto de n=10 valores capturados manualmente.

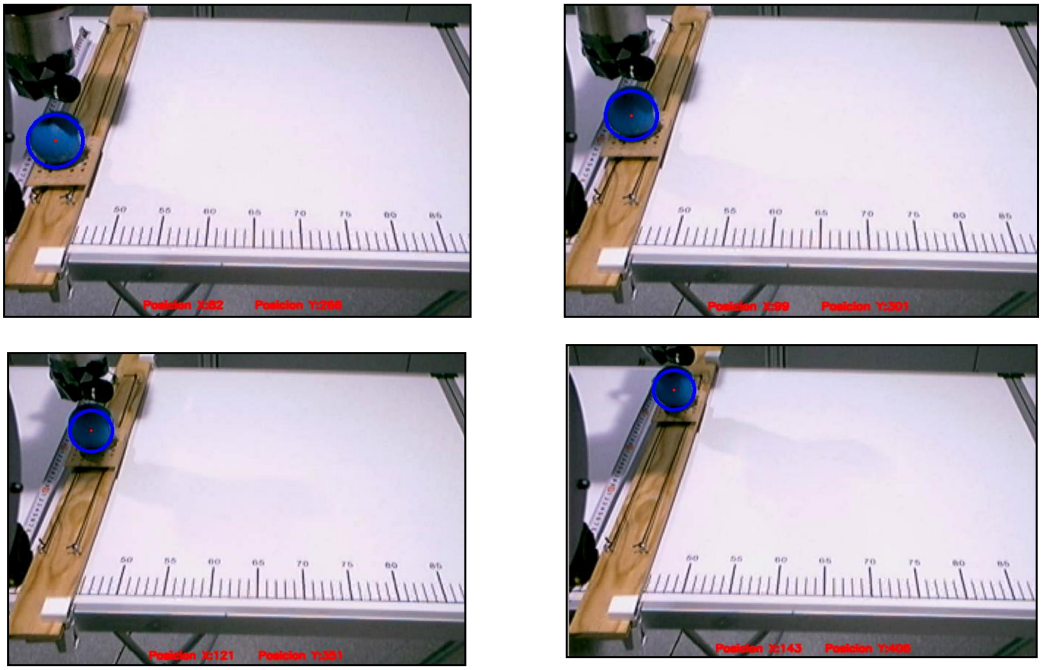


Tabla 11. Toma de datos (4/10 muestras) para el punto de accionamiento de la pinza

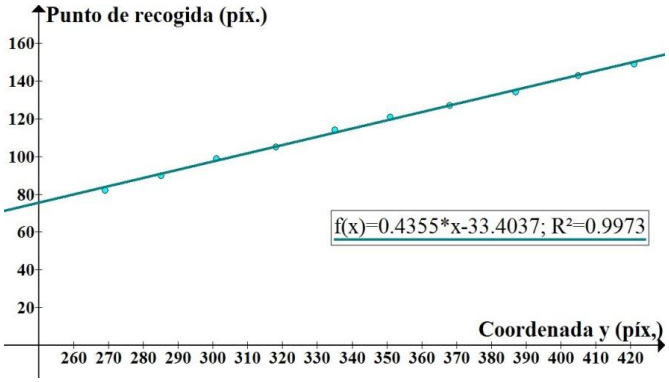


Ilustración 49. Correlación entre la coordenada 'y' del objeto y el punto de recogida

La función $f(x) = 0.4355 * x - 33.4037$ define el conjunto de puntos en los que se acciona la pinza que estaría situada en el extremo del brazo.

Inmediatamente se procede con el mismo conjunto de movimientos para la retirada del brazo que los ejecutados en el modo de captura Eye-in Hand.

```
if(modo==0)
{
if(step==0 && x<=500)
{
a=15.9065*pow(1.0036,y);
printf("\nEl alejamiento es %f\n", a);
step=1;
PCube_moveStep(dev, 5, 1.222, 3000); //5 en posición intermedia
PCube_moveStep(dev, 3, 1.099, 2500); //3 en posición intermedia
PCube_moveStep(dev, 2, 0.873, 2500); //2 en posición intermedia
Espera(1);
pos_rec=0.4355*y-33.4037; //Función que define posición de recogida
GetPosition(q2, q3, q5, pos_act2, pos_act3, pos_act5);
}
else if(step==1 && x<=300)
{
h=10;      Cininv(a,h,&q2,&q3,&q5);
PCube_moveStep(dev, 2, q2, 1500); printf("q2 : %f\n", q2);
PCube_moveStep(dev, 3, q3, 1500); printf("q3 : %f\n", q3);
PCube _moveStep(dev, 5, q5, 1500); printf("q5 : %f\n", q5);
GetPosition(q2, q3, q5, pos_act2, pos_act3, pos_act5);
modo=1;
}
}
```

3.2.4.3. Control selectivo

Para conocer el centro de masas del objeto hasta ahora se han realizado las operaciones pertinentes al conjunto mostrado por pantalla. Pero no se ha considerado hasta ahora la posibilidad de que en la misma cinta aparezcan dos objetos que puedan ser umbralizados. Este caso se complica, pues el centro de masas resultante no correspondería al centro de ninguno de los objetos, sino que se encontraría entre ambos, como se muestra en la imagen.

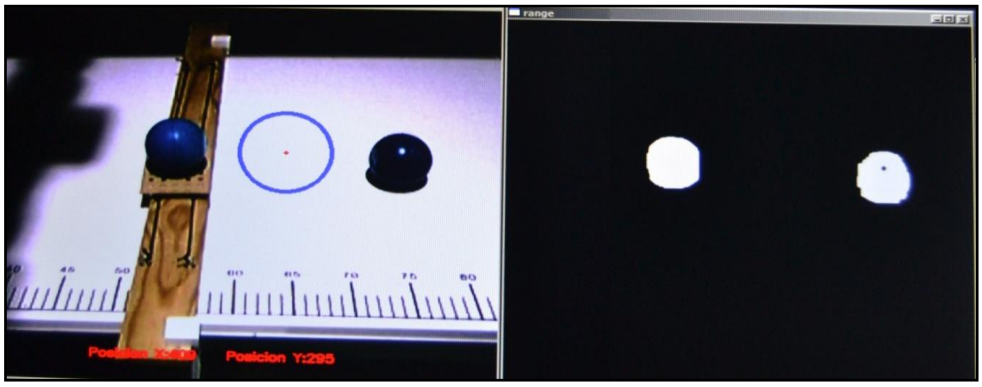


Ilustración 50. Modo de control no selectivo.

En este punto entra en juego la etapa de la segmentación, mirar Ilustración 31. El objetivo principal consiste en aislar los distintos elementos de la imagen para procesarlos (obtener sus características) por separado.

Para llevarlo a cabo, las librerías de imagen tienen funciones de tipo 'Connection', que permiten agrupar los píxeles que poseen el mismo valor en elementos con etiquetas diferentes para realizar un tratamiento diferente a cada uno.

Al investigar la forma de operar con OpenCv se llegó a la conclusión de que dicha librería de imagen dispone de dos formas diferentes igualmente válidas de solucionar el problema según la versión instalada.

En la versión 3.0.0 o superiores se dispone de una manera muy sencilla con la función ConnectedComponents [21] a través de la cual se etiquetan los elementos obteniendo una imagen por cada elemento en la que aparecen únicamente los píxeles de dicha etiqueta, pudiéndose realizar como antes se ha mencionado operaciones con las etiquetas por separado. Existe una función que simplifica aún más el problema. La función ConnectedComponentsWithStats permite obtener en un array las características (área y coordenadas del centro de masas) de cada uno de los elementos etiquetados.

Se instalaron varias versiones superiores a la 3.0.0, pero generaban incompatibilidades con las funciones de la librería m5apiw32 de PowerCube, por lo que hubo que recurrir a la solución que se presenta a continuación.

Versiones inferiores a la 3.0.0 no incluyen las funciones 'ConnectedComponents'. El método en este caso es más complejo, ya que requiere el uso de tres funciones diferentes:

- A través de la **función Canny** se obtiene una imagen con tan solo los bordes de los objetos en pantalla a través de gradientes de valor de los píxeles de una imagen monocanal en escala de grises o binaria. La función toma la imagen umbralizada original (`thresholded`) y cambia el valor de todos lo píxeles que no forman parte del contorno por el valor 0. Devuelve una imagen de las mismas características que la original. Los valores de thres1 y thres2 son valores umbral para el procedimiento de obtención por histéresis (en la documentación se recomienda utilizar thres =100).

- La **función findContours** permite, a partir de la imagen resultante de la función Canny (tratada como binaria), obtener un array de arrays con la información de la posición de los bordes de cada elemento. Se trata de un array principal cuyos elementos son a su vez arrays que contienen las coordenadas de los píxeles de cada uno de los contornos. El modo utilizado es el *CV_CHAIN_APPROX_SIMPLE* que almacena solo los puntos finales de cada segmento diagonal, en cuyo caso un rectángulo quedaría definido únicamente por cuatro puntos. La declaración `hierarchy` permite calcular también un vector de salida con información sobre la morfología de la imagen (no se ha hecho uso de este vector).

  En el proyecto la salida de esta función se la denomina 'contours'. De esta manera, el código `contours.size()` devuelve el número de elementos separados en la imagen, mientras que `contours[i]` devuelve los puntos asociados al elemento `i` almacenados por la función.

- **DrawContours,** por su parte, se emplea para dibujar el contorno del elemento diferenciado o segmento `i` obtenido por la función findContours. El color seleccionado es, como en las anteriores umbralizaciones, blanco (255,255,255) para el objeto próximo y negro (0,0,0) para cualquier otro objeto y el entorno. Mediante la declaración *CV_FILLED* no solo se dibuja el contorno del color deseado, sino que además se rellena de dicho color.

El resultado de la aplicación de las tres funciones anteriormente descritas implica la umbralización de únicamente el objeto más cercano a la base del brazo robótico. Es por esto el nombre de control selectivo, pues el brazo robótico selecciona qué objeto recoger mediante el criterio que se le ha impuesto en el programa. El resultado sería la recogida de un solo objeto y la omisión del resto cuando estos llegan muy seguidos. En cualquier caso es mejor resultado que el obtenido sin aplicar este criterio, que implicaría la recogida en un centro de masas situado entre los objetos, pero sin coincidir con el centro de ninguno.

La siguiente imagen muestra la interfaz del programa al que se le ha añadido la funcionalidad selectiva para la recogida de objetos.

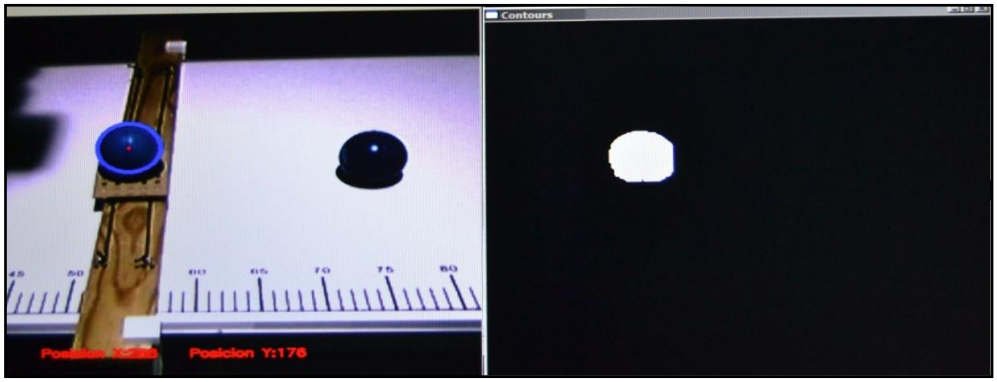


Ilustración 51. Resultado tras aplicar técnicas de segmentación.

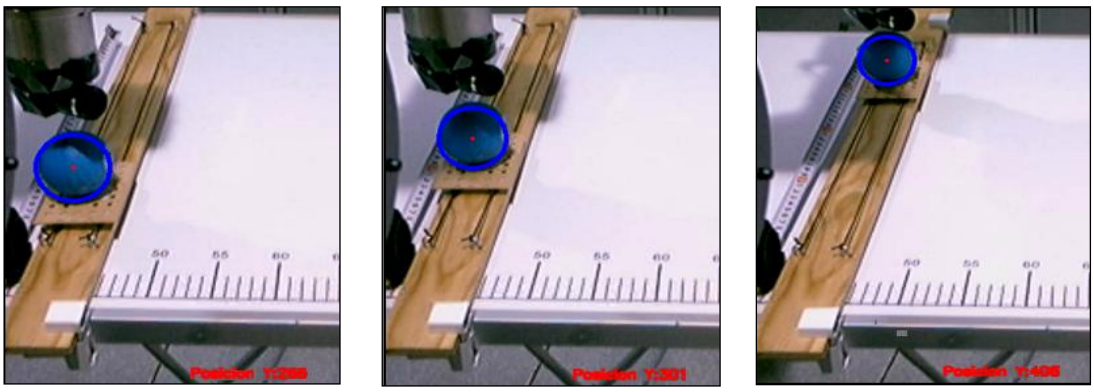
La funcionalidad que permite el control selectivo se ha denominado '*selección_objeto*', que incorpora las tres explicadas anteriormente y se presenta en el siguiente segmento de código.

```
void seleccion_objeto(Mat thresholded, float *xprox, float *yprox,
float *area)
{
Mat edges;
vector<vector<Point>> contours;
vector<Vec4i> hierarchy; //No se ha requerido al final su uso
int thresh=100; //Umbral recomendado para la histéresis
Canny(thresholded, edges, thresh, thresh*2, 3); //Detectar bordes
findContours(edges, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0,0)); //Guardar bordes

//CÁLCULO DEL MOMENTO DE LOS DISTINTOS OBJETOS
vector<Moments> mu(contours.size());
if(contours.size() != 0)
{
int contsize=5;
for(int i=0; i<contours.size(); i++)
  {mu[i] = moments(contours[i], false);
  contsize=i;} //Guarda el número de segmentos en la imagen
```

```
///CÁLCULO DEL CENTRO DE MASAS DEL OBJETO MÁS PRÓXIMO
std::vector<float> x(contsize);
std::vector<float> y(contsize);
Mat drawing = Mat::zeros(edges.size(), CV_8UC3);
*xprox=700.0; //Declaración de la variable que indica la coordenada x del objeto más próximo. Se empieza la iteración desde un valor muy alto (700)
for(int i=0; i<contsize; i++)
{
x[i]= mu[i].m10/mu[i].m00;
y[i]= mu[i].m01/mu[i].m00;
if(mu[i].m00<200) x[i]=600; //Cuando el área es muy pequeña se descarta asignándole un valor de x muy alejado del brazo
if(x[i]<*xprox)
   {*xprox=x[i]; *yprox=y[i];} //Coordenadas del objeto más próximo en un instante en la iteración hasta conseguir las del más próximo.
}

//REPINTADO DE TODOS LOS SEGMENTOS HALLADOS
for(int i=0; i<contsize;i++) //
{if(x[i]==*xprox) //Operaciones con el objeto más próximo
   {
   Scalar color = Scalar(255, 255, 255); //Color blanco
   drawContours(contours,contours,i,color,CV_FILLED,8,hierarchy,0);}
   *area=mu[i].m00;
   }
else if(x[i]!=*xprox) //Cualquier objeto que no sea el más próximo
   {Scalar color = Scalar(0,0,0); //Color negro
   drawContours(contours,contours,i,color,CV_FILLED,8,hierarchy,0);}
namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
imshow( "Contours", drawing ); //Mostrar en una imagen
}
}
```

Capítulo 4. Conclusiones del proyecto

Este proyecto ha sido realizado para la resolución de una aplicación específica mediante el uso de control visual. Se han presentado dos modos de control, descritos cada uno en la memoria, con diferente configuración de la cámara. La base de ambos métodos es la misma, el seguimiento del objeto se realiza basándose en la imagen y el problema de la cinemática inversa permite la colocación del brazo a cualquier profundidad en lo ancho de la cinta según la posición del objeto.



Sin embargo, la solución de cada uno se enfoca como se ha demostrado de una manera diferente, presentando cada modo ventajas e inconvenientes asociados.

En el **modo Eye-in-Hand** existe la problemática del movimiento solidario de la cámara con el brazo, lo cual ha obligado a establecer unas posiciones de referencia para la toma de los parámetros necesarios, independientes de la acción de recogida. Por otro lado, el movimiento del brazo ha obligado a disponer una cámara de lente fija, para evitar desajustes por efectos inerciales. Este modo es más recomendable cuando se otorga al brazo de más grados de libertad, como ocurre frecuentemente en robots de investigación, móviles o humanoides entre otros, ya que a diferencia del siguiente modo, no se produce el fenómeno típico de oclusión, en el cual se sitúa el brazo entre la cámara y el objetivo móvil.

El **modo Eye-to-Hand** tiene la ventaja de tener un plano de enfoque fijo, lo cual permite conocer el instante exacto en que el objetivo atraviesa los puntos definidos en la imagen, una ventaja significativa respecto al modo anterior. De este modo, el instante de accionamiento del sistema de recogida es más exacto y no aproximado como resultaba en el caso anterior.

El **control selectivo** permite, gracias a un criterio de selección establecido e implementado en el programa, discernir si sobre un objeto de la imagen debe actuar el brazo robótico. Esto posibilita el correcto funcionamiento del programa de control en problemas multiobjetivo y, a su vez, evita posibles fallos que se pudieran generar en la imagen por causas externas a la aplicación de estudio.

Durante el desarrollo del proyecto han surgido infinidad de adversidades en la instalación o actualización de librerías. Sobre todo fue muy costoso el fallido intento de combinar las posibilidades de Matlab y de Microsoft Visual Studio a través de mexOpencv, tal como se ha comentado en el apartado de software. Las dificultades de hardware no han sido menos importantes pues, por ejemplo, la instalación de las cámaras supuso probar varias alternativas hasta llegar a la más adecuada y el fusible del módulo 2 del brazo robótico sufrió un corte accidental en dos ocasiones por los cuales hubo que llamar al fabricante y solicitar ayuda técnica. Por otra parte, los proveedores de la pinza final del brazo demoraron tanto el envío de la misma que no fue posible incorporar su acción para la recogida de objetos.

El presente proyecto me ha servido para llevar a la realidad práctica los conocimientos adquiridos durante la carrera y a ajustar un trabajo de duración indeterminada a un plazo fijo (tres meses y redacción). También me ha servido para esforzarme en aprender los aspectos necesarios de robótica y visión artificial, materias que me interesan y me han llevado a decidirme por este proyecto de fin de carrera. Por último, cabe mencionar las mejoras notables que he conseguido en mis habilidades de programación.

5.1. Trabajos futuros

Son innumerables los diferentes proyectos que se pueden realizar en base a este. Solo se procede a explicar lo que se ha considerado más interesante.

El presente proyecto aborda una aplicación de tipo industrial, en la cual el robot conformaría un elemento más en una célula de proceso. Por el carácter lineal del movimiento se han limitado los grados de libertad del robot y en el modo Eye-in-Hand se ha considerado velocidad constante. Estas limitaciones simplifican el problema y no conducen al aprovechamiento de las capacidades del brazo. Por este motivo, un proyecto que podría resultar interesante sería la correcta calibración de dos cámaras para la localización de un objeto en las tres dimensiones del espacio (sin limitaciones como supone el movimiento lineal del presente proyecto), mediante la obtención de sus parámetros intrínsecos y extrínsecos, y el estudio completo de la cinemática inversa de los seis grados de libertad del brazo y de las distintas configuraciones que puede adoptar.

Una aplicación interesante puede ser la recepción de un objeto que se aproxima al brazo. Para realizar una recogida suave, el brazo puede esperar a que llegue el objeto en un área definida de recepción que podría ser esférica alrededor del brazo. Para evitar la ya explicada oclusión ha de estudiarse cuidadosamente la situación de ambas cámaras.

5.2. Limitaciones del control visual selectivo

Limitación 1: Objetos superpuestos en la imagen

Este programa presenta un problema en el caso de los objetos se encuentren muy próximos entre sí y aparezcan superpuestos en la imagen. En este caso no se diferenciarían los bordes de los objetos y se obtendría en la umbralización un área mayor a la de cualquiera de los objetos pero menor a la suma de las áreas de los objetos por separado, como se muestra en la imagen.

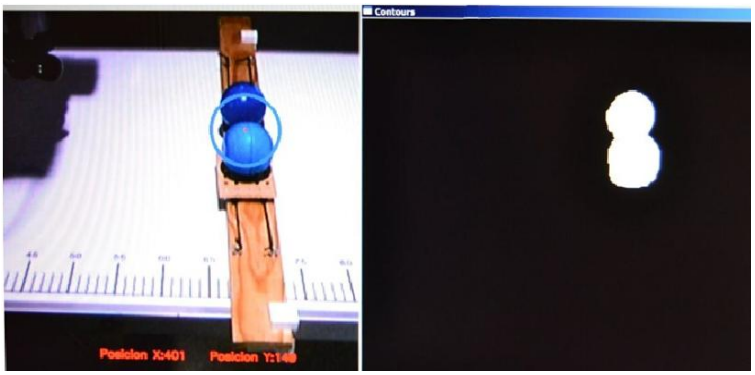


Ilustración 52. Limitación en el programa de control selectivo.

Para la aplicación de la recogida de objetos en una cinta transportadora esta limitación no es realmente importante, pues lo normal es que la distancia de separación entre dos objetos sea la suficiente para que no se produzca dicho problema. En cualquier aplicación en que no haya control de la distancia de separación de diferentes objetos habría que proceder de modo diferente.

Limitación 2: Defectos en la iluminación

La iluminación es un aspecto esencial en el control visual. El resultado del proceso de umbralización o binarización depende, como ya se ha explicado, de los valores exactos de tres parámetros combinados entre sí (en el modelo HSV son Matiz, valor y saturación).

El matiz solo depende del color del objeto en sí (Croma) mientras que los otros dos parámetros dependen de la intensidad del color (Luma). Haciendo más amplio el intervalo de los niveles de valor y saturación puedes reducir la sensibilidad del programa a cambios en la luminosidad pero también es posible que sean umbralizados elementos indeseados. Es por este motivo por el cual se debe controlar minuciosamente la iluminación dentro del área de enfoque de la cámara y evitar que un fallo eléctrico que afecte a algún elemento de iluminación imposibilite el correcto funcionamiento del programa de control tomando las medidas oportunas. Se pueden disponer para este fin focos de luz redundantes en diferentes circuitos iluminando el plano de enfoque de la cámara.

También es importante la posición relativa de los focos de luz respecto de la posición del brazo y del objetivo móvil. Existen dos formas diferentes de iluminar dicho objetivo: iluminación frontal y trasera. [12]

- La iluminación frontal, donde la luz incide directamente sobre el objeto, ya sea verticalmente, horizontalmente, de forma oblicua o difusa.

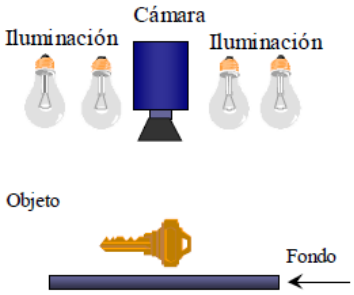


Ilustración 53. Iluminación frontal.[12]

En este tipo de iluminación hay que controlar la creación de sombras y de reflejos, que pueden dificultar el proceso de detección de las formas y detalles internos de los objetos en la imagen. Formas de evitar esto pueden ser la instalación de fuentes especiales de luz difusa (fibras ópticas), lámparas circulares que iluminan de forma homogénea o luz indirecta.

- La iluminación trasera o retroiluminación, en la cual se busca resaltar el contorno del objeto a modo de sombra chinesca. El objeto se suele situar delante o detrás de una pantalla.

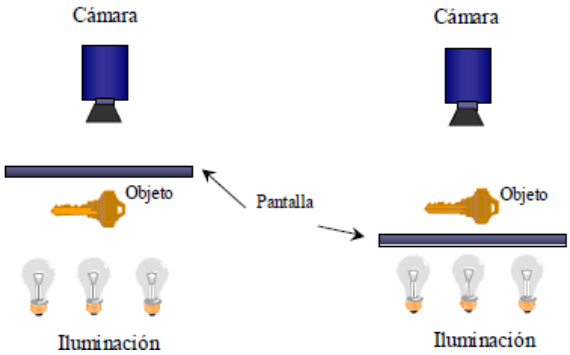


Ilustración 54. Retroiluminación: proyección sobre una pantalla (izq) e iluminación del fondo (der) [12]

La proyección del objeto en la pantalla solo permite el reconocimiento del contorno de este, teniendo como ventaja la rapidez del preprocesado y la segmentación. La pantalla de fondo del segundo método difumina la luz trasera produciendo un fuerte fondo blanco iluminado. Esta fuerte iluminación elimina las sombras que puede producir la iluminación ambiental consiguiendo un fuerte contraste entre el objeto y fondo, permitiendo detectar detalles pequeños del objeto.

Capítulo 6. Referencias bibliográficas

[1] DISAM Universidad Politécnica de Madrid, "*Robótica Industrial*", 2002.

[2] C. A. Montero et al., "*Del fordismo al toyotismo*", 2006.

[3] A. Barrientos. "*Fundamentos de robótica*", 2005.

[4] Nuño E. y L. Basañez, "*Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente*", 2004.

[5] L. A. Silva, UPM, "*Control visual de robots paralelos. Análisis, desarrollo y aplicación a la plataforma robotenis*", 2005.

[6] Amtec Robotics, "*Programmers guide for PowerCube*", pp. 1–16, 2005.

[7] G. Bradski, A. Kaehler, O'Reilly, "Learning OpenCv", 2008.

[8] Robotnik Automation S.L.L., "*Robot Kinematics and Control Manual*," 2008.

[9] Robotnik Automation S.L.L., "*System Startup Manual*", 2009.

[10] Robotnik Automation S.L.L., "*Robot Dynamics Manual*", 2009.

[11] A. Osorio, J. C. Pérez, C. A. Ortiz, and R. Medina, "*Horus: sistema de video para cuantificar variables ambientales en zonas costeras*," *Camera*, p. 15, 2007.

[12] A. Gonzales Marcos and F. Martínez de Pisón Ascacíbar, "*Técnicas y Algoritmos Básicos de Visión Artificial"*, 2008.

[13] B. Siciliano, O.Khatib. "*Handbook of robotics*," pp. 563-582. Springer, 2008.

[14] http://matlab.izmiran.ru/help/toolbox/images/color10.html

[15] Alejandro Sánchez Mur, UPCT, "*Control visual para el brazo Robotnik*", 2013.

[16] OpenCV Developers Team, "*Eroding and Dilating*", 2013.

[17] J. Kilian. "Simple image analysis by moments", 2001.

[18] R. Cantoral and G. Montiel, "*Visualización y pensamiento matemático*", 2001.

[19] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, "*Numerical Recipes: The Art of Scientific Computing"*, vol. 29, no. 4. 1987.

[20] http://www.mathworks.com/matlabcentral/fileexchange/25971-spline-natural

[21] OpenCV Developers Team, "*Structural Analysis and Shape Descriptors*", 2010.

Anexo: Librería m5apiw32

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 posición: 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.

Rango de operación: posición máxima	Función	Descripción
	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.