ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO DE FIN DE GRADO

Control de un brazo robótico mediante cámara *on-board*

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Nombre: Rafael Hernández Marrero

Tutor: Santiago Torres Álvarez

Cotutor: Rafael Arnay del Arco

Índice

Lista de tablas **IV**

Lista de figuras **V**

1. Resumen **VII**

2. Abstract **VIII**

3. Introducción **1**

3.1. Objetivos 2

3.2. Estructura de la memoria 3

4. Control visual **4**

4.1. Sistemas de visión en función de su arquitectura 4

4.1.1. Cámara externa al robot 6

4.1.2. Cámara en el extremo del robot 7

4.2. Clasificación de los sistemas de control visual 8

4.2.1. Ver y mover estático 8

4.2.2. Ver y mover dinámico 8

4.2.3. Control visual directo 9

4.2.4. Control basado en posición 10

4.3. Sistema de control ver y mover dinámico basado en posición . 10

4.3.1. Resolución de la cinemática del brazo: la Clase Conv . 11

5. Procesamiento digital de la imagen **12**

5.1. Introducción 12

5.1.1. OpenCv 13
5.2. Adquisición de imágenes 14
5.2.1. Cambio del espacio de color 16
5.2.2. Segmentación de la imagen: Umbralización 19
5.2.3. Operaciones morfológicas 23
5.3. Extracción de características 26
5.3.1. Orientación 27
5.3.2. Centro 29

6. Sistema de visión **32**
6.1. Componentes 32
6.1.1. Cámara 32
6.1.2. Sensor de distancia 33
6.2. Arquitectura 38

7. Comunicación serial **39**
7.1. Clase Serial 39
7.2. Clase ACL 40
7.3. Software Scorbot 41

8. Algoritmo de control visual con cámara on-board **43**
8.1. Selector de figuras por color 44
8.2. Seguimiento 51

9. Resultados obtenidos **63**

10. Conclusión **64**

11.Conclusion **66**

Referencias **67**

Anexo A. Seguimiento de objetos **69**

Anexo B. Selector de figuras por color **78**

Anexo C. Clases **82**

C.1. Clase Serial 82

C.2. Clase Acl 85

C.3. Clase Conv 92

C.4. Clase timer 96

Anexo D. Datasheet Sharp GP2Y0A21YK0F **99**

Anexo E. WebCam C270 HD Logitech **109**

Lista de tablas

1. Tabla con los píxeles mínimos y máximos para cada color y con iluminación diferente 22

2. Calibración de la cámara L-V 35

3. Longitud en píxeles 37

Lista de figuras

1. Componentes de un sistema de visión 5
2. a) Cámara montada en el extremo del robot. b) Cámara fija y externa al robot. 6
3. Sistemas de referencia de la cámara externa al robot 7
4. Sistemas de referencia de la cámara en el efector final 7
5. "Ver y mover" estático 8
6. "Ver y mover" dinámico 9
7. Control visual directo 9
8. Diagrama de bloques mostrando el proceso de extracción de características de la imagen 13
9. Espacio RGB. a) Modelo de color RGB. b) Cubo de color RGB. 16
10. Representación en 2D de las componentes Cb y Cr con luminancia constante Y = 0.5 17
11. Cono del modelo HSV 17
12. Representación por canales de los distintos espacios de color . 19
13. Ejemplo de iluminación para un cubo de *Rubik*. a) Exterior. b) Interior 21
14. Proceso morfológico de apertura. a) Imagen Original. b) Umbralización. c) Erosión. d) Dilatación. e) Apertura. 26
15. Orientación de figuras 29
16. Centro de figuras 31
17. Software Webcam C270 HD de Logitech 32
18. Relación voltaje-distancia del sensor 33
19. Gráfica de calibración L-V 34
20. Longitud en píxeles 36

21. Gráfica del factor de conversión píxel-milímetro 38

22. a) Diseño de la estructura del sistema de visión. b) Estructura del sistema de visión 39

23. Emulador de terminal ATS 42

24. a) Uso del comando `SHIFTC`. b) Uso del comando `TEACH`. . . . 43

25. Proceso de selección de la figura por color 45

26. Paleta de colores 47

27. a) Selector de colores a 0 con parámetros no inicializados. b) Selector de colores a 1 con parámetros HSV para el color rojo. 49

28. Proceso en el que se centra y se orienta el efector final 51

1. Resumen

El presente Trabajo de Fin de Grado (TFG) tiene como finalidad la localización y recogida de un objeto por un brazo robótico en tiempo real. Este proyecto se plantea también como una forma de ampliar los conocimientos que adquieren los alumnos en las sesiones de prácticas, debido a que amplía las posibilidades de control en manipulador.

Este TFG tiene como base dos pilares fundamentales: la localización del centro y de la orientación de un objeto, a partir de una imagen digital y la utilización de dichos datos en un sistema de control con realimentación de la información visual.

Para resolver estas dos premisas, primero se deberá construir una estructura que aloje el sistema de visión, compuesto por una cámara y un sensor de distancia, alrededor de la pinza. Para la detección de objetos se utilizará un algoritmo capaz de discriminar objetos por color de forma que se pueda obtener información manipulable (centro y orientación) de las imágenes. Una vez detectado el objeto, dicha información se traducirá en coordenadas respecto de la base del robot y se transmitirán a este por medio de una conexión serial.

Todos los procesos que se van a ejecutar en este proyecto se realizarán en el lenguaje de programación C++, utilizando clases ya establecidas o creando clases propias que ayudarán a facilitar la tarea. También se utilizará el lenguaje de programación Arduino para obtener las medidas de distancia que emita el sensor.

2. Abstract

The purpose of this Final Degree Project (FDP) is to define the location of an object and pick it up with a robotic arm in real time. This project is also proposed as a way to expand the knowledge acquired by students in the practice sessions, due to the new possibilities of control of the robotic arm.

This FDP is based on two fundamental pillars: the location of the center and the orientation of an objet from a digital image and the use of such data in a control system with feedback of visual information.

In order to solve these two premises, first a structure composed of a vision system (formed by a camera and a distance sensor) around the gripper must be built. For the detection of the objects, an algorithm capable of discriminating objects by color will be used to obtain manipulable information (center and orientation) of the images. Once the object is detected, this information will be translated into coordinates respect to the base of the robot and will be transmitted to the robot controller through a serial connection.

All the processes that will be executed in this project will be done in the C++ programming language, using already established classes or creating own classes that will help the task. The Arduino programming language will also be used to obtain the measurements of the distance sensor.

3. Introducción

En este proyecto se aborda un nuevo esquema de control de un brazo robótico utilizando una cámara *on-board*.

En asignaturas del grado como ”Sistemas Robotizados” o ”Ampliación de Sistemas Robotizados” y también en la asignatura ”Robótic” del Máster de Industriales , el objetivo de trabajo con los brazos robóticos es observar cómo actúan frente a un objeto de forma estática, es decir, capturando la posición y la orientación de dicho objeto en un momento determinado y actuando sobre ellas, sin tener en cuenta los posibles movimientos del objeto que puedan producirse con posterioridad a ser tomada la imagen del escenario o zona de trabajo del robot. Con este trabajo se intenta ampliar la funcionalidad del brazo robótico dotándole de algoritmos de control visual dinámico, de forma que se consiga capturar la posición y la orientación y al mismo tiempo poder manipular un objeto móvil.

Para llevar a cabo el proyecto se necesita seleccionar una cámara que cumpla con ciertas especificaciones. Ajustándose al método de cámara *on-board*, ésta se instalará en el efector final. La cámara elegida fue la WebCam C270 HD de Logitech que se colocará junto al sensor Sharp GP2Y0A21YK0F. Este sensor infrarrojo medirá la distancia al objeto que se quiere detectar, la cual se recogerá a través de una placa Elegoo Uno R3 que emula a una placa Arduino Uno.

El procesamiento de la imagen se realizará utilizando la librería OpenCv junto con el lenguaje de programación C++. Se utilizará la clase Serial de Arduino para poder obtener los valores de distancia ofrecidos por el sensor Sharp GP2Y0A21YK0F.

El modelo de robot del que se dispone es un Scorbot-Er IX de Eshed

Electronics, el cual está gobernado por un controlador tipo B de la citada empresa, que dispone de la electrónica de potencia necesaria para el movimiento de los distintos motores en las articulaciones del manipulador, así como la electrónica digital para la lectura de los sensores de posición y alarmas de la estructura. La vía de comunicación con el controlador B es a través de un PC utilizando el puerto serial RS232. Ayudándonos de una librería en C++ podremos enviar diferentes comandos al Scorbot, así como enseñarles las posiciones que queremos alcanzar.

En un primer planteamiento del trabajo se propuso que la posición del objeto se establecería en un plano XYZ respecto de la base del robot. Tras evaluar las características del sensor Sharp GP2Y0A21YK0F se tuvo que redefinir el planteamiento inicial, ya que debido a las condiciones del entorno de trabajo y las diferentes velocidades del sistema de visión y de la respuesta del robot, las medidas que ofrecía variaban demasiado para poder tenerlas en cuenta. Por consecuencia, se tomó la decisión de trabajar con el objeto apoyado sobre la mesa de trabajo de forma que quedase limitado a un plano paralelo al plano XY respecto de la base del robot.

3.1. Objetivos

- En la introducción se comenta que en la actualidad en el laboratorio se trabaja con un esquema en lazo abierto para la recogida de objetos. El objetivo es dotar de mayor versatilidad a la tarea de recogida, definiendo un sistema en bucle cerrado, con realimentación de la información visual.
- Se creará una estructura, que alojará el sistema de visión compuesto por la cámara y el sensor de distancia, alrededor de la pinza.
- La forma de comunicación con el robot y con el sensor de distancia se

realizará a través del puerto serial. Se establecerá una comunicación estable, que permita el envío y la recogida de datos.

- Se definirán las posiciones necesarias para colocar la pinza sobre el objeto, de forma que coincidan el centro y la orientación de la pinza con el centro y la orientación del objeto respectivamente. Una vez posicionado trazar una trayectoria de recogida.

3.2. Estructura de la memoria

En primer lugar, se van a comentar las diferentes estrategias de control visual que existen en función de su arquitectura. Más adelante, se explicará cómo se clasifican dependiendo si aplican o no un bucle de realimentación del sistema de visión, además de ver cómo actúan los sistemas de control basados en posición.

En segundo lugar, se va ha hablar sobre el procesamiento de la imagen y su posterior extracción de características. Se dará una breve explicación sobre qué es el procesamiento de la imagen y se nombrarán los algoritmos que reflejen aquellas características que sean útiles para la detección de los objetos.

En el siguiente apartado se explicará cómo funcionan los componentes del sistema de visión, introduciendo qué tipo de software utiliza la cámara elegida, además de incluir una explicación del funcionamiento del sensor Sharp y su implementación con la placa Elegoo Uno R3. Por último, se detallará cómo se creó la estructura que aloja estos componentes alrededor de la pinza.

A continuación, se describirá la comunicación serial entre el Scorbot y el PC que ejecuta el código escrito en C++. Para ello, se hará uso de una

librería serial, la cual enviará comandos al Scorbot a través de un software denominado ATS, que emula a una terminal, con la que se permitirá el acceso al entorno de programación ACL [5].

Una vez descrito lo anterior se explicará detalladamente cómo se han implementado todos estos procesos convergiendo en un único código, para más adelante, ofrecer de forma visual los resultados obtenidos.

Por último se adjuntarán las conclusiones del trabajo, además del apartado de referencias y los distintos códigos utilizados y datasheets recogidos en anexos.

4. Control visual

Como se comentó en la introducción, hasta ahora los sistemas robóticos implementados en el aula en los que se integraba la visión por computador, trabajaban en bucle abierto, en un esquema de control visual denominado "ver y mover" estático. La alternativa que se propone es un sistema de control en bucle cerrado, con realimentación de la información visual. A continuación se mostrarán las diferentes formas de construir un sistema de visión y seguidamente la clasificación de dichos sistemas dependiendo de la arquitectura y de la acción de control utilizada [10].

Todas las figuras utilizadas en este apartado se basan en los esquemas empleados en el capítulo 11 del libro Robots y Sistemas Sensoriales [10].

4.1. Sistemas de visión en función de su arquitectura

Gracias a la mejora en los sistemas de captura y procesamiento de imágenes se ha conseguido que existan diferentes tipos de arquitectura de

entornos robóticos. Esto ha permitido que la ubicación de una o varias cámaras influya positivamente en la tarea encomendada por el robot. Un ejemplo simple de un sistema robótico basado en la integración de un sistema de visión es el que muestra la figura 1.

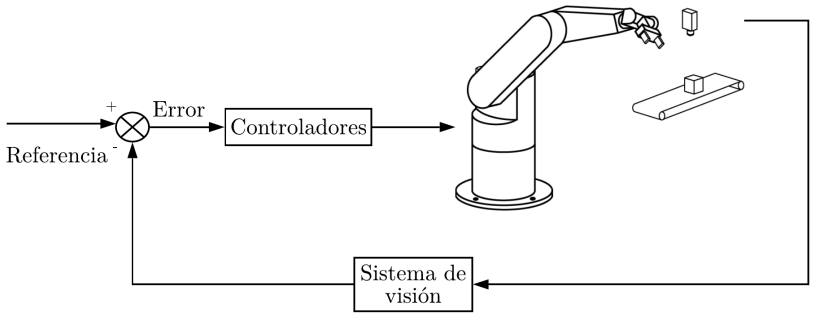
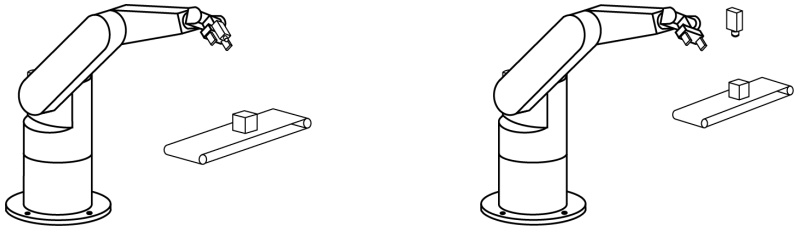


Figura 1: Componentes de un sistema de visión.

En la figura 1 se muestra un sistema de control en bucle cerrado en el que el bloque "Sistema de visión" se encarga de procesar y detectar los objetos. El valor de entrada al bucle es una referencia que se compara con la información ofrecida por el sistema de visión, generando una diferencia entre el valor deseado y el actual. De la corrección de este error se encarga el controlador aproximando la pinza al objeto.

En el sistema de visión que se muestra en la figura 1 la cámara se posiciona en un punto fijo, donde recoge información del entorno y de la localización del robot. Este tipo de configuración es muy utilizada, pero en otras ocasiones la localización del robot no es necesaria, por lo que otra ubicación de la cámara es montada junto a la pinza, esta opción permite un reconocimiento más amplio del entorno. Por lo tanto, se podrían considerar dos formas de colocar la cámara: montada sobre el efector final (Figura 2a), o en un punto fijo externo al robot (Figura 2b).



(a) *(b)*

Figura 2: a) Cámara montada en el extremo del robot. b) Cámara fija y externa al robot.

4.1.1. Cámara externa al robot

En esta configuración no existe ninguna relación física entre la pinza del robot y la cámara, pero sí se conoce la relación entre el sistema de coordenadas de la base del robot y la cámara. En la figura 3 se muestran los principales sistemas de coordenadas de un sistema de control visual en el que la cámara se encuentra en una posición fija, externa al robot. En este caso la cámara observa tanto el espacio de trabajo como la localización del extremo del robot. La ubicación del objeto se va a realizar en función del sistema de coordenadas del mundo, M, situado en la base del robot. Esto es posible gracias a que se conocen las relaciones entre los sistemas de coordenadas, lo que permitirá definir la posición del extremo del robot y del objeto en base a un mismo sistema de coordenadas.

Gracias al sistema de visión se puede procesar el objeto como una imagen en dos dimensiones de la que podemos obtener su centro y orientación. Esta información viene expresada en coordenadas respecto del sistema de la cámara, C, por lo que para obtener dichas coordenadas respecto a las del mundo, M, se debe conocer la posición relativa de la cámara y la base del robot, ${}^{M}\mathbf{T}_{C}$, que permita transformar dichas coordenadas.

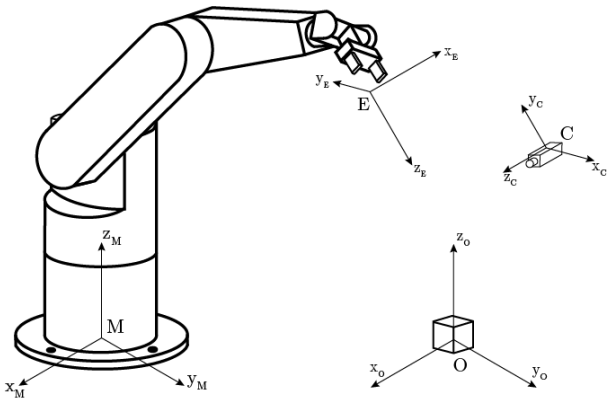


Figura 3: Sistemas de referencia de la cámara externa al robot.

4.1.2. Cámara en el extremo del robot

En este tipo de configuración, denominado en inglés como *on-board*, la cámara se encuentra solidaria a la pinza de forma que ahora la localización del objeto se realiza en función del sistema de coordenadas del extremo del robot. En la figura 4 se han representado los sistemas de coordenadas correspondientes a un sistema de visión con una cámara ubicada en el efector final. La relación que existe entre el sistema de coordenadas del efector final, E, y el de la cámara, C, ${}^{E}\mathbf{T}_{C}$, se conoce y permanece constante durante toda la tarea.

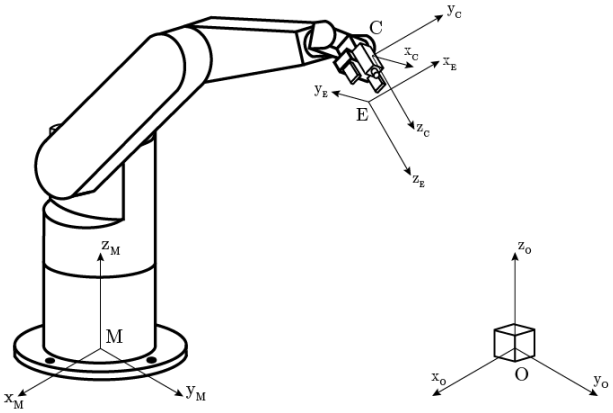


Figura 4: Sistemas de referencia de la cámara en el efector final.

4.2. Clasificación de los sistemas de control visual

En los siguientes apartados se verá la clasificación de los sistemas de control visual atendiendo a la ubicación del sistema de visión (Apartados 4.2.1, 4.2.2, 4.2.3) y también se verá cómo se comportan estos sistemas si el control se ejecuta basado en posición (Apartado 4.2.4).

4.2.1. Ver y mover estático

Los primeros sistemas que empezaron a integrar visión por computador funcionaban en bucle abierto, primero procesar la imagen, ver, y luego, de forma secuencial, mover el robot para alcanzar la posición del objeto. En la figura 5 se observa un sistema de visión como entrada al bucle que localiza la posición del objeto. Una vez localizado el objeto el robot irá a la posición establecida suponiendo que durante el trayecto dicha posición no ha variado. Por lo tanto, en dicho sistema se espera que el espacio de trabajo no se modifique ya que la única realimentación que existe es la de los bucles internos de control.

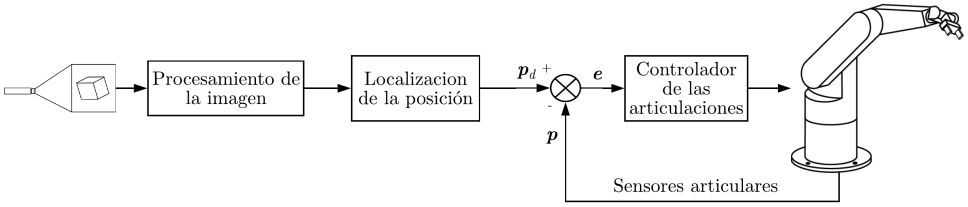


Figura 5: "Ver y mover" estático.

4.2.2. Ver y mover dinámico

Sobre el sistema "ver y mover" anterior que se ha denominado estático se puede añadir una realimentación de la información visual, definiendo otro

sistema “ver y mover” pero esta vez dinámico. En la figura 6 se observa un primer lazo de realimentación del sistema de visión, junto a un segundo que realimenta la información articular. Este lazo permite la actualización de la información visual mientras se realiza la tarea.

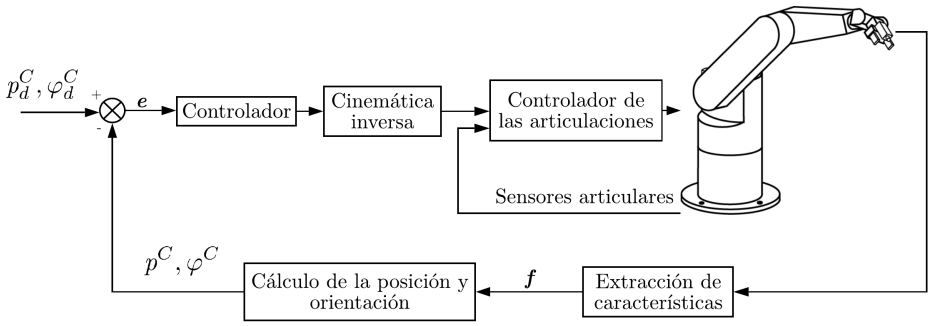


Figura 6: “Ver y mover” dinámico.

4.2.3. Control visual directo

Este tipo de sistema es igual al anterior pero, como aparece en la figura 7, el lazo que realimenta la información articular no es necesario ya que el mismo controlador que gestiona la información del sistema de visión se encarga de calcular las consignas para las articulaciones del robot.

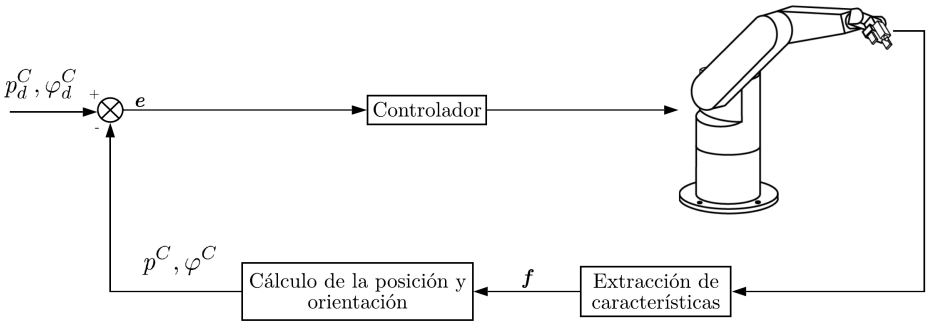


Figura 7: Control visual directo.

4.2.4. Control basado en posición

En este tipo de sistemas, se utiliza la información que ofrece el sistema de visión para localizar los objetos que se encuentren en el espacio de trabajo respecto del sistema de coordenadas de la cámara. Una vez obtenida la posición del objeto se compara con el valor de referencia y el error que produzca será la entrada al bucle de control. Atendiendo a la clasificación que se comentó en los apartados anteriores a continuación, se muestran las distintas configuraciones para los sistemas de control basados en posición.

La configuración "Ver y mover" estático basado en posición se corresponde con la figura 5, donde la información que ofrece el sistema de visión se almacena en la posición p_d. Las configuraciones "Ver y mover" dinámico y control visual directo basado en posición definen la información del sistema de visión por medio de dos valores de consigna: p^C, que corresponde a la posición y φ^C, que indica la orientación del objeto. Estos valores se comparan con la entrada de referencia al bucle, p_d^C, φ_d^C, y la diferencia resultante es la entrada al regulador (Figuras 6 y 7).

4.3. Sistema de control ver y mover dinámico basado en posición

En los apartados anteriores se han comentado los diferentes tipos de sistemas de control que integran un sistema de visión, para este proyecto, y como se comentó en la introducción, se parte de un entorno preparado para un sistema de control "ver y mover" estático y se quiere llegar a definir un sistema de control "ver y mover" dinámico basado en posición. Para este sistema se necesita primero, establecer la relación de transformación entre la cámara y el efector final, ${}^E\mathbf{T}_C$, comentada en el apartado 4.1.2. Una vez obtenida esta relación se debe resolver la cinemática que implica obtener la

posición dada por la imagen del objeto, respecto del sistema de coordenadas de la base del robot. Debido a que el software del Scorbot (Apartado 7) no dispone de un comando para conocer las variables articulares, se necesita resolver la cinemática inversa, a partir de una posición dada, para luego obtener de la cinemática directa las matrices de transformación. La resolución de la cinemática inversa y directa se implementó en la clase Conv que se describe a continuación.

4.3.1. Resolución de la cinemática del brazo: la Clase Conv

Esta clase es una transcripción de un código escrito en Octave para representar gráficamente las articulaciones del robot en una posición determinada. Se utilizó una librería de álgebra lineal para el lenguaje C++, denominada Armadillo [9], debido a que implementa métodos similares a las funciones de Octave.

La clase Conv posee un único método accesible por el usuario, `Trans()` al cual se le pasa por parámetro un vector matricial con la posición actual de la pinza respecto de la base del robot, y un vector matricial con la posición del objeto respecto a las coordenadas de la imagen. Este método devuelve un vector de tipo `float` con las coordenadas de la imagen respecto de la base del robot.

Los siguientes métodos que se van a comentar se ejecutan una vez se ha declarado el método `Trans()`. El método `Cip()` resuelve la cinemática inversa puntual de las tres primeras articulaciones. Una vez obtenidas sus variables articulares se definen los parámetros Denavit-Hartenberg junto con un *offset* que existe en el Scorbot entre las articulaciones 1 y 2. Con ayuda del método `matrizT()` se construyen las matrices de transformación homogénea que ayudan a obtener las variables articulares de las articulaciones

4 y 5 por métodos geométricos. Al obtener las variables articulares de todas las articulaciones se definen los parámetros Denavit-Hartenberg y se procede a resolver la cinemática directa. El método `cinemadirect()`, primero, construye las matrices de transformación homogénea a partir de los parámetros Denavit-Hartenberg y, a continuación, obtiene las coordenadas de la imagen en función de la base usando la relación de transformación, ${}^{E}\mathbf{T}_{C}$, anteriormente vista.

5. Procesamiento digital de la imagen

5.1. Introducción

El procesamiento digital de la imagen se define como un conjunto de técnicas que transforman una imagen digital en datos que luego podrán ser utilizados para extraer sus características. Esta información se empleará para mejorar la calidad de la imagen, o de ayuda para la localización y posicionamiento de cualquier elemento que se quiera detectar, camino por el que deriva este trabajo.

Las características que se extraigan de la imagen serán analizadas por una máquina y actuarán en consecuencia. Esta idea de dar a las máquinas una forma de visualizar elementos aparece en el concepto de visión artificial. Como se verá más adelante la capacidad de visión del robot será complementada con un sistema sensorial capaz de medir distancias [8].

La Figura 8 refleja los procesos a los que se va a someter la imagen para poder hallar su centro y su orientación.

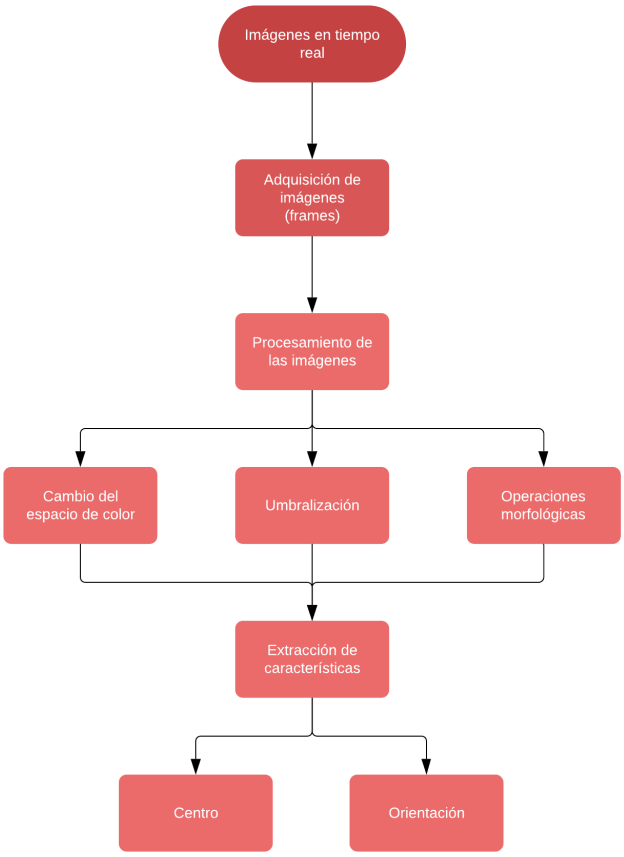


Figura 8: Diagrama de bloques mostrando el proceso de extracción de características de la imagen.

5.1.1. OpenCv

Como ya se expuso en la introducción, para el reconocimiento de imágenes se hará uso de OpenCv, biblioteca que contine diferentes clases y funciones relacionadas con la visión por computador y el *machine learning*, todo ello en código abierto [7].

Sabiendo que las imágenes digitales se reducen a una matriz que contiene todos los valores de intensidad de sus píxeles, OpenCv trata de procesar y manipular la información que contengan esas matrices.

OpenCv nació en 2001 y en aquel entonces se basó su estructura en el lenguaje C. Dado las limitaciones de este lenguaje la capacidad para gestionar la memoria se dejaba en manos del usuario lo cual acarreaba problemas cuanto mayor fuese el código que se estaba ejecutando.

El cambio vino con la llegada de C++ y la introducción de las clases que favoreció el manejo automático de la memoria. Con la clase `Mat` se automatizó el uso y la liberación de la memoria cuando ya no se necesitaba.

La clase `Mat` se divide en dos partes: una almacena el fichero de cabecera de la matriz, el cual contiene información sobre el tamaño de la matriz, qué método se usó para almacenar dicha información, la dirección de memoria de la matriz almacenada, etc; y otra indica el puntero a la matriz que contiene los valores de los píxeles.

Al almacenar las imágenes como se escribió anteriormente OpenCv es capaz de aumentar la velocidad computacional del código evitando copias innecesarias de grandes imágenes. Gracias a esto y a la gran cantidad de algoritmos optimizados para trabajar con imágenes en tiempo real se eligió esta librería para su uso en el proyecto [6].

5.2. Adquisición de imágenes

En este paso se inicializa el constructor de la clase `VideoCapture` pasándole el parámetro 0, el cual utiliza la cámara por defecto, a continuación, dentro de un bucle infinito, se almacenan los *frames* en una variable de tipo `Mat` utilizando el método `read()`.

Por seguridad se añade un condicional que comprueba si se ha logrado conectar a la cámara usando el método `isOpened()` que devuelve *false* cada vez que no se ha inicializado el constructor de la clase. Al igual que antes

se debe comprobar que la cámara esté pasando *frames* por lo que se usa el método `empty()` que devuelve *true* si la variable de tipo `Mat` no contiene elementos.

Por último, se muestra la imagen en una ventana con el comando `imshow()` pasándole como parámetros el nombre de la ventana y la variable de tipo `Mat` donde hemos almacenado los *frames*. Para terminar la ejecución es importante incluir al final del bucle el comando `waitKey()` ya que, una vez que se acabe el tiempo que le pasamos, en milisegundos, mantendrá abierta la ventana hasta que el usuario decida acabar con el programa.

```
VideoCapture cap(0);
if(!cap.isOpened()){
    cerr << "No se ha podido conectar con la cámara" << endl;
    return 0;
}
Mat frame;
while(true){
    cap.read(frame);
    if (frame.empty()){
        cerr << "No se ha detectado ningún frame" << endl;
        return 0;
    }
    imshow("Imagen", frame);
    waitKey(10);
}
```

Como se verá más adelante se ajustará el ancho y el alto de la ventana donde se muestra la imagen, definiendo dos propiedades de la clase `VideoCapture` mediante el comando `set()` y pasando por parámetro `CV_CAP_PROP_FRAME_WIDTH` para ajustar el ancho y `CV_CAP_PROP_FRAME_HEIGHT` para ajustar la altura.

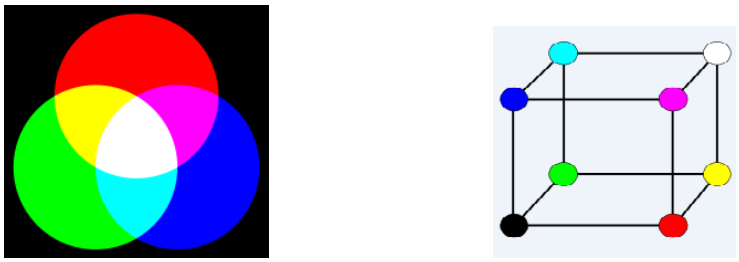
```
cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
```

A continuación, se procederá a detallar cómo se realizó la localización de los objetos teniendo en cuenta diferentes técnicas de procesamiento y extracción de características de la imagen.

5.2.1. Cambio del espacio de color

Dentro del concepto visión artificial se manejan diferentes espacios de color, esto es, una forma específica de organizar los colores para poder procesarlos y analizarlos por un ordenador.

Las imágenes que se obtienen de forma digital normalmente las observamos en el espacio de color RGB, el cual se forma a partir de la mezcla de sus colores primarios; rojo, verde y azul (Figura 9a). Este espacio de color se basa en la variación del color dependiendo de la radiación de luz con la que se incida a los colores primarios, abarcando una amplia muestra al mezclarlos, este concepto se conoce como síntesis aditiva del color [11]. La gama de colores RGB se puede representar en forma de cubo (Figura 9b) coincidiendo la escala de grises con la diagonal desde el punto negro hasta el blanco.



(a) *(b)*

Figura 9: Espacio RGB. a) Modelo de color RGB. b) Cubo de color RGB.

Otro espacio de color con el que se trabaja es el YCbCr, formado por la componente Y, que representa la luminancia (luma), y por Cb y Cr que son los dos componentes de crominancia (chroma), es decir, diferencia de

azul y diferencia de rojo respectivamente (Figura 10). Este espacio de color depende de la cromaticidad de los colores primarios del espacio RGB, por ello no se considera un espacio de color absoluto sino más bien, otra forma de ordenar la información que ofrece el espacio RGB [4].

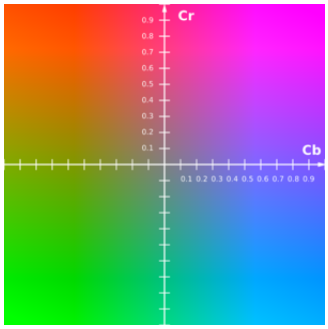


Figura 10: Representación en 2D de las componentes Cb y Cr con luminancia constante, Y = 0.5.

El último espacio de color que se va a comentar es el HSV, que define un modelo de color en función de las componentes *hue* (matiz), *saturation* (saturación) y *value* (valor). Este modelo de color es útil cuando se quiere buscar una amplia gama de colores dependiendo de las componentes saturación y valor que se apliquen al modelo RGB [4]. La representación más acorde a este espacio de color es la de un cono invertido en el que se diferencia a la base del cono con el matiz midiendo sus valores como grados de un ángulo de 0 a 360º; la altura con la componente valor que se ajusta al eje blanco-negro; y el radio de la base con la saturación, el cual define el eje de brillo negro-blanco (Figura 11).

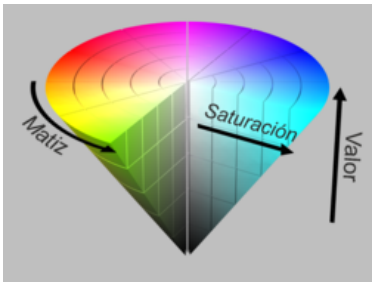


Figura 11: Cono del modelo HSV

A continuación, se va a explicar cómo se consigue el esquema de la figura 12, con los distintos espacios de color que se han comentado anteriormente separados por canales. La separación por canales se realiza en escala de grises, pero para una mejor visualización de cada modelo, se le ha aplicado una paleta de colores RGB.

En OpenCv existe una función con la que se puede pasar del espacio de color RGB a YCbCr y a HSV, y viceversa. Se llama a la función `cvtColor()`, pasándole por parámetros la imagen original y la imagen de salida e indicándole el código correspondiente al espacio de color al que se quiera pasar la imagen.

A continuación se muestra el código pasando la imagen RGB de la figura 12 a los espacios de color anteriormente mostrados:

```
Mat img = imread("lighthouse.png");
Mat HSV, YCbCr;
cvtColor(img, HSV, CV_BGR2HSV);
cvtColor(img, YCbCr, CV_BGR2YCrCb);
imshow("HSV", HSV);
imshow("YCbCr", YCbCr);
waitKey();
```

A diferencia del código que describía como adquirir imágenes en tiempo real, aquí basta con utilizar el método `imread()`, al cual se le pasa la ruta con el nombre de la imagen que se quiere procesar.

Para el proyecto se debía elegir alguno de estos espacios de color, por lo que se estudió cual de todos estos modelos representaba mejor los colores en una situación real. En la siguiente sección se detallará el proceso que se siguió para determinarlo además de ofrecer algunos ejemplos prácticos.

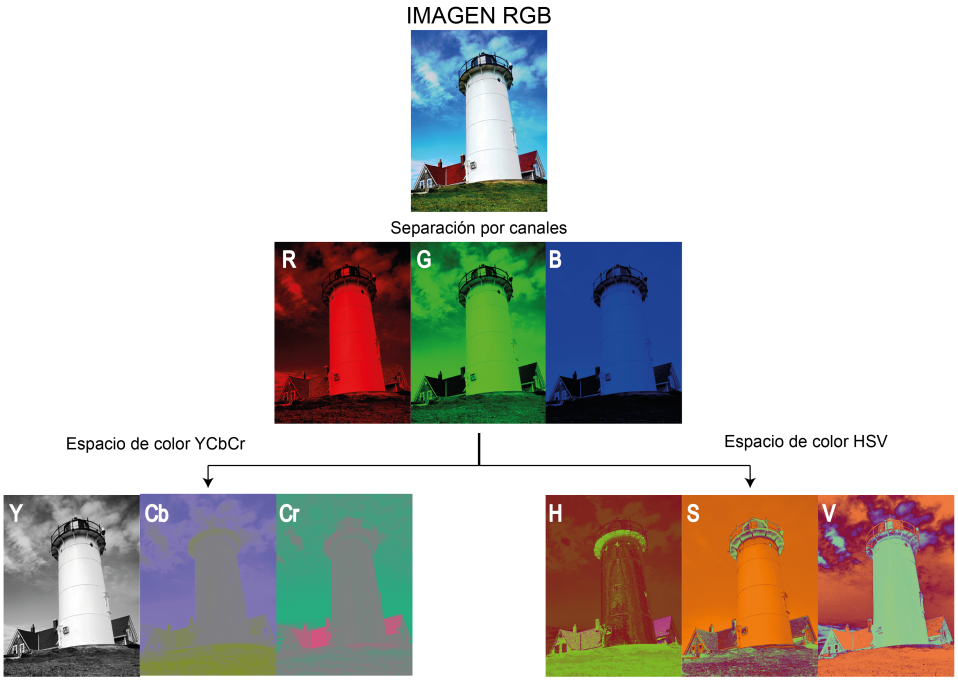


Figura 12: Representación por canales de los distintos espacios de color

5.2.2. Segmentación de la imagen: Umbralización

En cualquier imagen digital existen varios elementos que se diferencian entre sí y del entorno en el que se encuentran. La segmentación de una imagen consiste en extraer esos elementos para su estudio individualmente.

Uno de los métodos de segmentación, el cual utilizaremos en el proyecto, es la umbralización. Este método consiste en definir una imagen como dos niveles de gris, separando el fondo del resto de elementos. La umbralización es idónea cuando los objetos que aparecen en la imagen contrastan con el fondo de esta, lo cual coincide con nuestro caso [2].

Para la umbralización se debe definir un valor de umbralización T, por el que se diferenciaran los dos niveles de gris. Existen diferentes fórmulas dependiendo del uso de ese valor. En algunos casos se quiere que el valor de

los píxeles sea mayor que el valor umbral (Ecuación 1) y en otros que sea menor (Ecuación 2).

$$g(x,y) = \begin{cases} 1 & si\ T \leqslant f(x,y) \\ 0 & resto \end{cases} \tag{1}$$

$$g(x,y) = \begin{cases} 1 & si\ T \geqslant f(x,y) \\ 0 & resto \end{cases} \tag{2}$$

Otra fórmula para umbralizar define dos valores de T, un valor máximo y otro mínimo.

$$g(x,y) = \begin{cases} 1 & si\ T_{min} \leqslant f(x,y) \leqslant T_{max} \\ 0 & resto \end{cases} \tag{3}$$

Para el proyecto se usará la ecuación 3 definida para cada una de las componentes del espacio de color que se utilice (Ecuación 4).

$$g(x,y) = \begin{cases} & C1_{min} \leqslant f_{c1}(x,y) \leqslant C1_{max} \\ 1 & si\ C2_{min} \leqslant f_{c2}(x,y) \leqslant C2_{max} \\ & C3_{min} \leqslant f_{c3}(x,y) \leqslant C3_{max} \\ 0 & resto \end{cases} \tag{4}$$

Este tipo de umbralización se basa en que los diferentes componentes no siempre tienen un valor T concreto, sino que este varía dependiendo de la iluminación con la que se incida a la imagen.

A continuación se mostrará qué pasos se siguieron para determinar que espacio de color es el correcto para el proyecto. Se tomarán dos fotos de un

cubo de *Rubik*, una en el exterior, figura 13a y la otra en interior, figura 13b, simulando el cambio de iluminación que se podría tener durante la detección en tiempo real. Se comparará en qué espacio de color (RGB, YCbCr o HSV) se distinguen mejor los colores.

(a) *(b)*

Figura 13: Ejemplo de iluminación para un cubo de Rubik. a) Exterior. b) Interior

En las dos fotos se ha buscado el rango en el que se puede diferenciar cada color del resto, después, se han recogido esos valores en la tabla 1. A continuación, se comprueba que el espacio RGB en colores como el azul y el amarillo da valores muy diferentes dependiendo de la iluminación mientras que el HSV y el YCbCr se mantienen parejos. En el resto de colores sin embargo se comportan de forma homogénea, destacando el espacio YCbCr por encima del HSV.

Observando los datos de la tabla 1 es evidente que el espacio RGB cambia notablemente cuando el color del objeto está iluminado y cuando no, por lo que fue el primero en descartarse. En el caso del espacio HSV e YCbCr los datos reflejan una mejor adaptación a la luz por parte del espacio YCbCr por lo que en un principio se eligió este espacio.

		Exterior			Interior		
Amarillo hsv	Min	20	148	190	16	190	57
	Max	26	241	255	20	241	255
Amarillo rgb	Min	32	176	215	0	53	0
	Max	146	219	251	24	255	115
Amarillo ycbcr	Min	176	152	0	61	144	89
	Max	227	174	81	77	156	107
Azul hsv	Min	99	121	45	97	14	0
	Max	117	255	255	136	255	255
Azul rgb	Min	71	0	0	16	2	0
	Max	127	89	59	32	24	22
Azul ycbcr	Min	28	103	138	8	123	128
	Max	87	123	255	24	128	138
Naranja hsv	Min	10	154	178	0	215	91
	Max	14	255	255	10	253	256
Naranja rgb	Min	0	65	170	0	24	99
	Max	77	144	255	20	53	255
Naranja ycbcr	Min	89	180	0	45	158	0
	Max	170	255	95	71	255	256
Rojo hsv	Min	0	225	132	0	194	67
	Max	255	255	182	2	255	97
Rojo rgb	Min	0	0	127	0	0	67
	Max	38	36	255	16	22	87
Rojo ycbcr	Min	43	188	83	24	158	0
	Max	85	255	255	42	255	256
Verde hsv	Min	71	160	49	69	97	22
	Max	85	255	256	91	255	256
Verde rgb	Min	49	71	0	18	30	0
	Max	95	121	28	47	61	32
Verde ycbcr	Min	53	0	0	24	0	0
	Max	85	103	255	49	125	256

Tabla 1: Tabla con los píxeles mínimos y máximos para cada color y con iluminación diferente.

Las sucesiones pruebas, ya en tiempo real, hicieron que se replantease la utilización de dicho espacio de color. Por las especificaciones de la cámara o por la constante variante iluminación del lugar de trabajo, el espacio YCbCr no estaba dando los resultados que reflejaban los datos recogidos anteriormente. Debido a esto se probó con el espacio HSV, el cuál presentaba mayor robustez en las mediciones por lo que, finalmente se optó por utilizarlo

preferentemente antes que el espacio YCbCr.

Todo este proceso se realizó usando la función de OpenCv `inRange()` que se basa en la umbralización por color de la ecuación 4. A esta función se le pasa por parámetros la imagen original, la imagen de salida y dos vectores de 3 elementos indicando los valores máximos y mínimos de cada componente.

```
Mat CuboExt = imread("CuboExt.jpg");
Mat CuboInt = imread("CuboInt.jpg");
Mat HsvExt, HsvInt;
cvtColor(CuboExt, HsvExt, CV_BGR2HSV);
cvtColor(CuboInt, HsvInt, CV_BGR2HSV);
while(true){
    inRange(HsvExt,Scalar(H_MIN,S_MIN,V_MIN),
     ↪  Scalar(H_MAX,S_MAX,V_MAX),HsvExt);
    inRange(HsvInt,Scalar(H_MIN,S_MIN,V_MIN),
     ↪  Scalar(H_MAX,S_MAX,V_MAX),HsvInt);
    imshow("HsvExt", HsvExt);
    imshow("HsvInt", HsvInt);
    if(waitKey(30)== 'z') break;
}
```

En el código anterior se describe cómo se han obtenido los datos de la tabla 1 para el espacio de color HSV, y para el resto, se ha variado el último parámetro de la función `cvtColor()`. Los valores `H_MIN`, `S_MIN`, `V_MIN`, `H_MAX`, `S_MAX` y `V_MAX` variarán dependiendo del color que se quiera detectar.

5.2.3. Operaciones morfológicas

En las secciones anteriores se ha visto cómo detectar los objetos aplicando métodos de segmentación general, en esta última parte se definirán procesos que analizarán individualmente cada uno de los objetos que se encuentran en la imagen.

Las operaciones morfológicas ayudan a definir los objetos que se encuentran en la imagen variando la forma o la estructura que tienen estos. Con esto se consigue eliminar el ruido presente en la mayoría de las imágenes luego de un proceso de segmentación, además de ser de utilidad para la extracción de características que veremos en el apartado 5.3 [2].

Las operaciones morfológicas que se aplicarán en este proyecto serán la erosión y la dilatación.

Al trabajar con imágenes binarias existen únicamente dos niveles de gris, para nuestro caso 0 y 1. La erosión consiste en degradar progresivamente uno de esos dos campos. Si un elemento del campo a degradar está rodeado de elementos iguales, éste seguirá perteneciendo al campo que se quiere degradar, mientras que en el caso contrario pasaría al otro campo. La dilatación es el proceso contrario a la erosión, se basa en el progresivo crecimiento de uno de los dos campos. A diferencia de la erosión, si un elemento del campo contrario a dilatar es vecino de otro del campo a dilatar este pasará a pertenecer a este último, y si no es así, el elemento permanecerá en el mismo campo [2].

Una vez conocidos estos dos procesos, para las imágenes anteriores se aplicarán una combinación de ambas, primero se erosionará la imagen y luego se dilatará. Esta combinación se denomina de apertura y logra separar, con mayor precisión, los objetos del fondo de la imagen.

Para aplicar estos procesos en el código, primero debemos crear un elemento estructurante mediante la función `getStructuringElement()` a la que se le pasa por parámetros la forma del elemento y el tamaño de la matriz. El elemento estructurante se le pasa por parámetro a la función `erode()` y a la función `dilate()`, dependiendo si se quiere erosionar o dilatar la imagen respectivamente.

```
Mat img = imread("Estrella de mar.jpg");
Mat imgGray, imgErDi, Imgthreshold;
cvtColor(img, imgGray, CV_BGR2GRAY);
Mat Element = getStructuringElement( MORPH_RECT,Size(3,3));
while(true){
    threshold(imgGray, Imgthreshold, 188, 255 , 0);
    erode(Imgthreshold, imgErDi, Element);
    dilate(imgErDi, imgErDi, Element);
    imshow("apertura", imgErDi);
    if(waitKey(30)== 'z') break;
}
```

El código anterior es un ejemplo de como se obtuvieron las imágenes de la figura 14. Se aplicó, como se visualiza en el código, un elemento estructurante rectangular de tamaño 3x3, gracias al cual se pudo extraer de manera casi exitosa la forma de la estrella de mar del fondo de la imagen. Previamente a la imagen se le aplicó un tipo de umbralización que se basa en la ecuación 1 por la que el valor de los píxeles de la imagen deber ser mayor que 188. En OpenCv la forma en que se utiliza este tipo de umbralización es por medio de la función `threshold()` a la que se le pasa por parámetros la imagen de salida y de entrada, el valor de umbralización, el valor máximo y el tipo de umbralización, que en este caso es binaria.

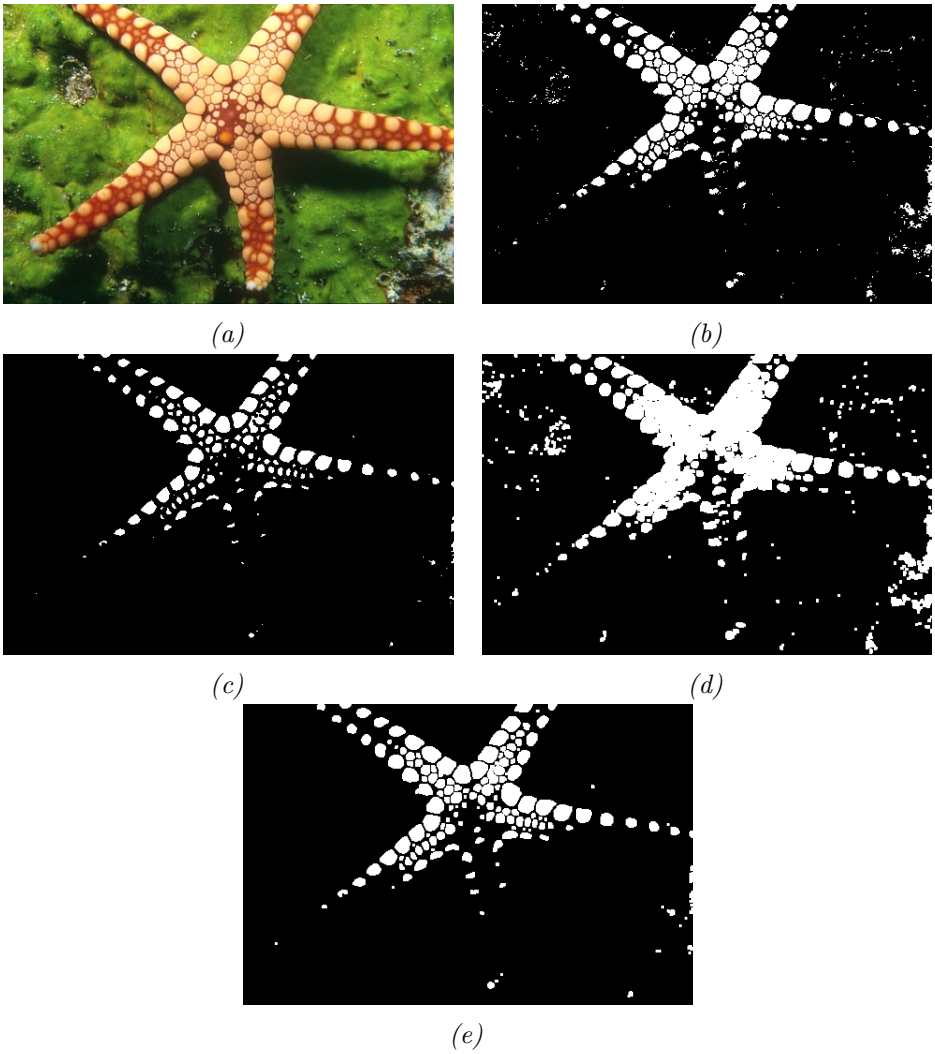


Figura 14: Proceso morfológico de apertura. a) Imagen Original. b) Umbralización. c) Erosión. d) Dilatación. e) Apertura.

5.3. Extracción de características

Este apartado se centrará en la búsqueda de las propiedades de los objetos que nos ayudarán a localizar su situación en el espacio de la cámara o a definir su longitud, su área o su forma. Teniendo en cuenta el apartado 4.2.4 las características que se necesitan conocer son: la posición del objeto respecto de las coordenadas de la imagen y la orientación de éste.

Una vez definidos los objetos de forma visual en los apartados anteriores tocaría traducir dicha información en datos para su manipulación. Existe una función en OpenCv que localiza y almacena los contornos de los objetos que aparecen en la imagen: `findContours()`. A esta función se le pasa por parámetros la imagen binaria, resultado de aplicar la umbralización con la función `inRange()` del apartado anterior; un vector donde se almacenarán los contornos como vectores de puntos; un vector donde se irán almacenando los contornos de forma ordenada mediante diferentes niveles de jerarquía (Apartado 8.2); un entero indicando en que modo se quiere almacenar los contornos; y por último un entero indicando que método de aproximación de contornos se va a utilizar.

Una vez almacenados los contornos de los objetos se aplicarán diferentes algoritmos para hallar la orientación y el posicionamiento.

5.3.1. Orientación

Para definir la orientación del objeto se debe elegir el modo en el que se quieren almacenar los contornos y el método que se quiere utilizar. Para este caso se elige el modo `CV_RETR_EXTERNAL`, el cual almacena solo el contorno más externo del objeto detectado y el método de aproximación `CV_CHAIN_APPROX_NONE` que recoge todos los puntos que forman el contorno sin ningún tipo de jerarquía.

A continuación se creará un elemento de la clase `RotatedRect` que busca rectángulos girados en un plano. Cada rectángulo está definido por su centroide, la longitud de sus lados y el ángulo rotado. Estos tres parámetros serán devueltos por la función `fitEllipse()` a la cual se le pasa el contorno almacenado. Esta función calcula la elipse que más se ajusta al contorno dado y devuelve el rectángulo girado un ángulo entre 0 y 180 grados. Para

esta función el contorno que se estudia debe almacenar al menos 5 puntos, por lo que antes de aplicar la función `fitEllipse()` se comprueba dicha restricción.

La función `fitEllipse()` calcula el ángulo tranzando una horizontal en el punto del contorno más cercano a la horizontal inferior de la imagen y dando el ángulo que existe entre el lado corto del rectángulo inscrito en la elipse y la horizontal.

```
Mat img = imread("Rectangulos Rotados.png");
Mat imgGray;
int angle;
cvtColor(img, imgGray, CV_BGR2GRAY);
vector<vector<Point> > contours;
RotatedRect elipse;
findContours(Imgthreshold, contours, CV_RETR_EXTERNAL,
 ↪  CV_CHAIN_APPROX_NONE);
for (size_t i = 1; i < contours.size(); i++){
    size_t count = contours[i].size();
    if( count < 6 ) continue;
    elipse = fitEllipse(Mat(contours[i]));
    angle = elipse.angle;
    putText(img, to_string(angle), Point(contours[i][0]), 1, 3,
     ↪  Scalar(0,0,255));
}
    imshow("imagen", img);
    if(waitKey(30)== 'z') break;
}
```

En la figura 15 aparecen varios rectángulos rotados un cierto ángulo. Con el código anterior se imprime en la imagen el ángulo calculado con `fitEllipse()` mediante la función `putText()` a la cual se le indica qué se quiere imprimir en la imagen y dónde.

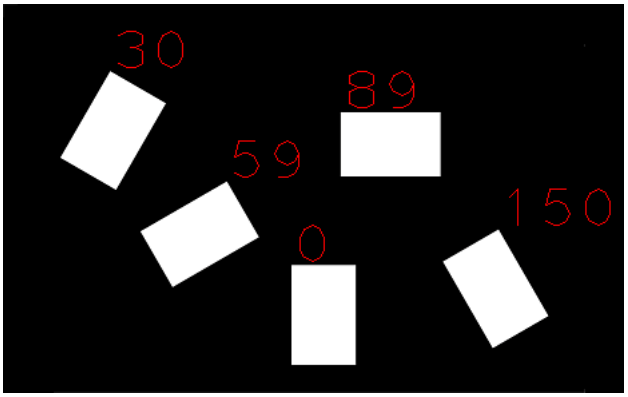


Figura 15: Orientación de figuras

5.3.2. Centro

Para esta parte, a la función `findContours()` se le pasará el modo `CV_RETR_CCOMP`, el cuál almacenará todos los contornos que encuentre en dos niveles de jerarquía, en el nivel superior se almacenarán los bordes más externos de las figuras y en el segundo nivel se almacenarán los bordes de los posibles agujeros que se encuentren en el objeto. Como método de aproximación se usará `CV_CHAIN_APPROX_SIMPLE` el cuál solo recogerá los puntos extremos del contorno que se forme, por ejemplo, si es un rectángulo recogerá los cuatro puntos correspondientes a las esquinas.

Para la localización del centro de una imagen se utilizará como base la teoría de los momentos de figuras planas [3]. Dicha teoría establece que para un conjunto de puntos determinados por una función $f(x, y) > 0$ el momento simple de orden (p, q) se define como:

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx \; dy \tag{5}$$

Siendo $p, q = 0, 1, 2, ...$

Adaptando la ecuación anterior a una imagen en escala de grises con la intensidad de los píxeles dada por la función $I(x, y)$ los momentos M_{ij} podrán ser calculados de la siguiente forma:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y) \tag{6}$$

El cálculo del centroide utilizará los momentos de orden 0 y 1, ecuaciones 7, 8, 9. El momento simple de orden 0 nos indica el área de la figura ya que suma todos los píxeles cuyo valor es 1 y los momentos simples de orden 1 se usan básicamente para hallar el centro de masa de la figura.

$$M(0, 0) = \sum_x \sum_y I(x, y) \tag{7}$$

$$M(1, 0) = \sum_x \sum_y xI(x, y) \tag{8}$$

$$M(0, 1) = \sum_x \sum_y yI(x, y) \tag{9}$$

Estos momentos dependen de las coordenadas (x, y) de la figura por lo que el área de la figura que queda a un lado o al otro del centro en x o del centro en y debe ser el mismo:

$$x = \frac{M(1, 0)}{M(0, 0)} \tag{10}$$

$$y = \frac{M(0, 1)}{M(0, 0)} \tag{11}$$

Los momentos de orden 0 y 1 los podemos definir en OpenCv gracias a

la clase `Moments`. Los momentos de la figura que se está evaluando se recogen por medio de la función `moments()` a la que se le pasa por parámetro el contorno seleccionado. El siguiente código calcula el centro de los rectángulos de la figura 15 usando las ecuaciones 10 y 11 y luego dibuja un punto junto con las coordenadas x e y del centro de la imagen. La figura 16 es el resultado de esta operación.

```
Mat img = imread("Rectangulos Rotados.png");
Mat imgGray;
int x, y;
double area;
cvtColor(img, imgGray, CV_BGR2GRAY);
vector<vector<Point> > contours;
findContours(Imgthreshold, contours, CV_RETR_CCOMP,
 ↪  CV_CHAIN_APPROX_SIMPLE);
for (size_t i = 1; i < contours.size(); i++){
    area = moment.m00;
    x = moment.m10/area;
    y = moment.m01/area;
    putText(img, "(" + to_string(x) + ", " + to_string(y) + ")",
     ↪  Point2f(x,y), 1, 1, Scalar(0,255,0),2);
    circle(img,Point(x,y),1,Scalar(0,0,255),2);
}
    imshow("imagen", img);
    if(waitKey(30)== 'z') break;
}
```

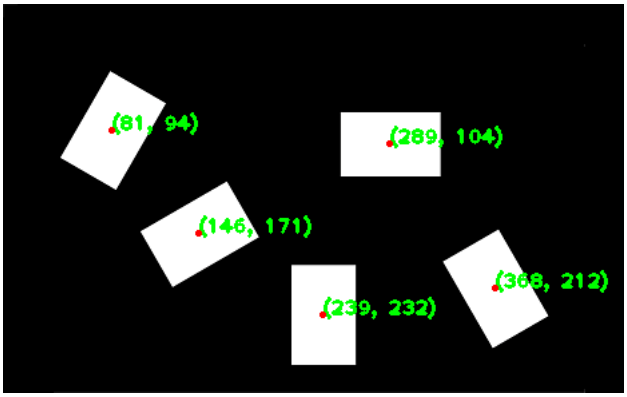


Figura 16: Centro de figuras

6. Sistema de visión

En este apartado se verá qué aparatos forman el sistema de visión y como se realizó el diseño de la estructura que los posicionará en torno a la pinza.

6.1. Componentes

6.1.1. Cámara

La cámara que se va a utilizar es la Webcam C270 HD de Logitech. Se eligió esta cámara debido a que la que se había utilizado en el laboratorio hasta entonces pertenecía a un modelo anterior y arrojaba buenos resultados durante las tareas, además de poseer un software de visión propio, figura 17, que facilita en gran medida su manejo. Con una resolución máxima de 1280x960 este modelo puede grabar a 720 píxeles con una frecuencia de 30 fps.

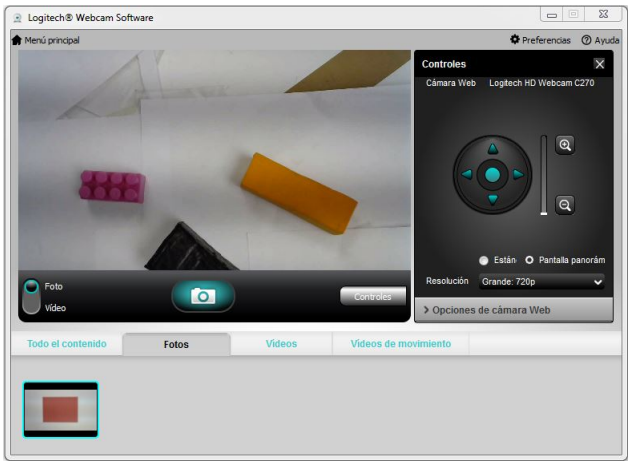


Figura 17: Software Webcam C270 HD de Logitech

6.1.2. Sensor de distancia

La longitud que separa el extremo de la pinza y el objeto la mediremos con el sensor de distancia Sharp, concretamente con el modelo GP2Y0A21YK0F. Este sensor permite calcular la distancia al objeto en un rango de 10 a 80 cm, se compone de un detector de posición (PSD), un diodo emisor de infrarrojos (IRED) y un circuito de procesamiento de señales.

Este dispositivo funciona por medio de triangulación, el sensor infrarrojo emite un haz de luz que al chocar con algún objeto es reflejado y recogido por el PSD. El ángulo que forme el haz reflejado con el PSD variará dependiendo de la distancia del objeto, convirtiendo esa posición en una señal de voltaje con un error en la medida de ±0,3 voltios. La relación entre voltaje y distancia viene dada por la gráfica de la figura 18 cuya expresión es una función potencial que podríamos expresar como $L = aV^b$, siendo L la distancia medida y V el voltaje que devuelve el sensor.

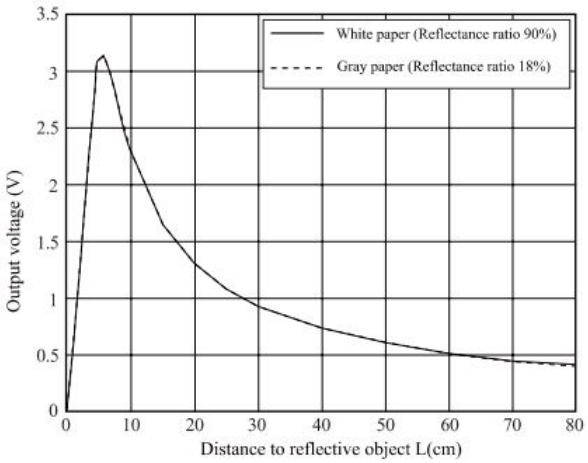


Figura 18: Relación voltaje-distancia del sensor

La salida analógica del sensor se puede leer desde Arduino. Para eliminar el ruido que se produzca usaremos un filtro, que para este caso será un conjunto de n muestras.

```
void setup() {
    Serial.begin(9600);
}
void loop() {
    int ADC_SHARP=ADC0_promedio(5000);
    Serial.println(ADC_SHARP);
}
int ADC0_promedio(int n){
    long suma=0;
    for(int i=0;i<n;i++){
        suma=suma+analogRead(A0);
    }
    return(suma/n);
}
```

Con el código anterior se tomaron 20 señales de voltaje para diferentes alturas (Tabla 2). El voltaje medido se transmite a través del puerto serie, como veremos en el apartado 7.2. Una vez volcado los datos en una tabla Excel se obtiene la ecuación L-V por medio de la gráfica siguiente.

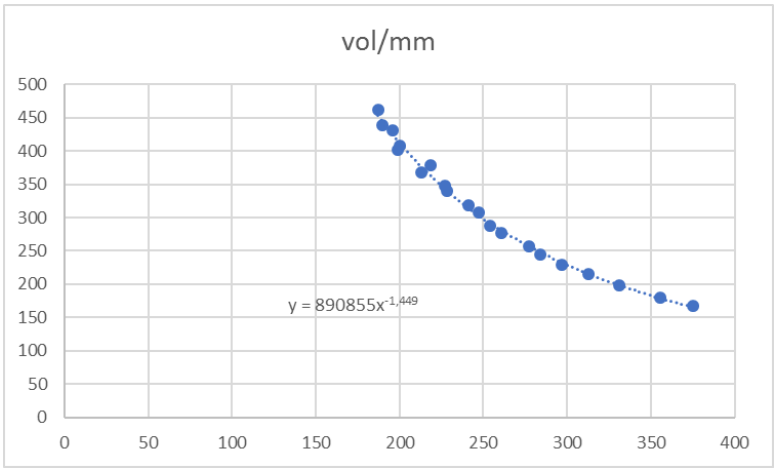


Figura 19: Gráfica de calibración L-V

	Distancia Medida (mm)	Voltaje (mV)
Altura 1	150,34	375
Altura 2	163,242	331
Altura 3	180,454	297
Altura 4	198,35	277
Altura 5	211,558	254
Altura 6	228,039	241
Altura 7	240,379	227
Altura 8	259,881	218,5
Altura 9	270,223	200
Altura 10	290,163	190
Altura 11	301,183	187
Altura 12	323,563	355,5
Altura 13	331,454	313
Altura 14	350,636	284
Altura 15	361,282	260,5
Altura 16	385,338	247
Altura 17	391,06	228
Altura 18	413,945	213
Altura 19	421,769	199
Altura 20	444,284	196

Tabla 2: Calibración de la cámara L-V

Una vez hallada la ecuación del sensor, $L = 890855V^{-1,449}$, se busca la conversión píxel-milímetro ya que para cada altura el valor de los píxeles varía. Para realizar esta operación primero, se definió un programa que reconociese la longitud en píxeles del objeto que estuviese recogiendo la cámara y, a continuación, se tomarían 20 medidas de dichos píxeles para diferentes alturas.

A diferencia del apartado 5.3.1 que utilizamos la función `fitEllipse()` para hallar la orientación de la figura, ahora usaremos la función `minAreaRect()` que almacena los contornos en un elemento de la clase `RotatedRect`. El uso de `minAreaRect()` en vez de `fitEllipse()` reside en que esta busca el rectángulo que encierra el área más pequeña del objeto. Gracias a esto se

calculan los vértices y hallando la distancia euclídea entre ellos nos dará la longitud en píxeles del lado del rectángulo que estamos midiendo (Figura 20).

```
Mat img = imread("Rectangulos Rotados.png");
Mat imgGray;
int angle;
cvtColor(img, imgGray, CV_BGR2GRAY);
vector<vector<Point> > contours;
RotatedRect box;
findContours(Imgthreshold, contours, CV_RETR_EXTERNAL,
    CV_CHAIN_APPROX_NONE);
for (size_t i = 1; i < contours.size(); i++){
    Point2f vertices[4];
    box.points(vertices);
    float x = vertices[1].x - vertices[2].x;
    float y = vertices[1].y - vertices[2].y;
    float dist = pow(x, 2) + pow(y, 2);
    dist = sqrt(dist);
    putText(img, to_string((int)dist), box.center, 0, 1,
     ↪  Scalar(0,0,255), 1);
    line(img, vertices[1], vertices[2], Scalar(255,0,0),3);
}
    imshow("imagen", img);
    if(waitKey(30)== 'z') break;
}
```

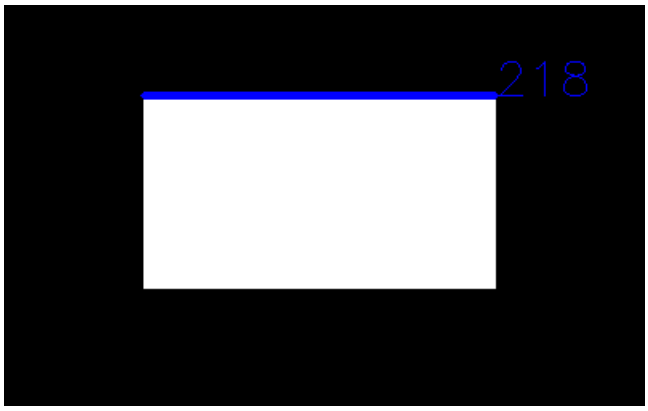


Figura 20: Longitud en píxeles

Para la calibración se tomó una figura de longitud 51 mm y se procedió a medir dicha longitud tomando las alturas que devuelve el sensor por medio

del código que se aplicó anteriormente, añadiéndole la conversión L-V a la salida de la señal analógica y transmitiendo los datos por el puerto serie, que se verá en el apartado 7.2. Una vez almacenados esos datos se calculó el factor de conversión pixel-milímetro dividiendo la longitud real, en mm, con la longitud en píxeles dada por cada altura (Tabla 3).

	Píxeles	**Distancia Medida (mm)**	**FC mm/píxel**
Altura 1	289	167,34	0,176470588
Altura 2	267	180,242	0,209876543
Altura 3	243	197,454	0,248780488
Altura 4	219	215,35	0,278688525
Altura 5	205	228,558	0,312883436
Altura 6	192	245,039	0,344594595
Altura 7	183	257,379	0,377777778
Altura 8	169	276,881	0,408
Altura 9	163	287,223	0,435897436
Altura 10	153	307,163	0,472222222
Altura 11	148	318,183	0,495145631
Altura 12	137	340,563	0,191011236
Altura 13	135	348,454	0,232876712
Altura 14	128	367,636	0,265625
Altura 15	125	378,282	0,301775148
Altura 16	116	402,338	0,333333333
Altura 17	117	408,06	0,372262774
Altura 18	109	430,945	0,3984375
Altura 19	108	438,769	0,439655172
Altura 20	103	461,284	0,467889908

Tabla 3: Longitud en píxeles

Los datos de la tabla 3, Distancia medida y FC mm/píxel, se volcaron en una hoja Excel y luego de varias pruebas con diferentes aproximaciones se tomó la ecuación potencial como la mejor que se adaptaba a los datos (Figura 21).

La ecuación que refleja la figura 21 se define como $FC = 0{,}001A^{1{,}0145}$ siendo A la altura dada por el sensor en mm y FC el factor de conversión

resultante.

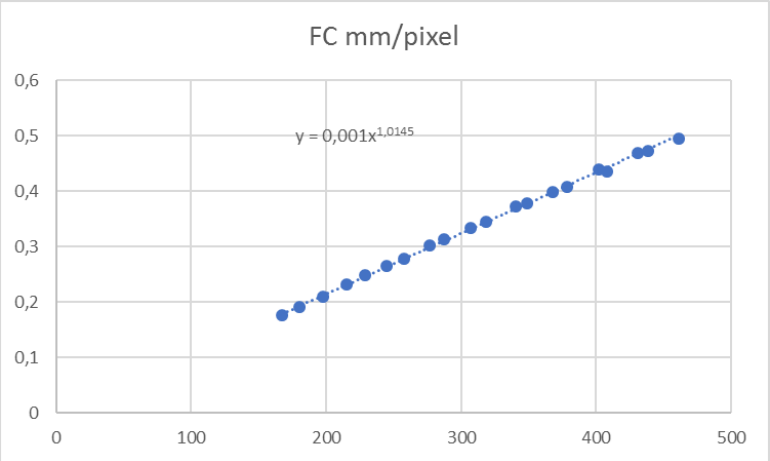


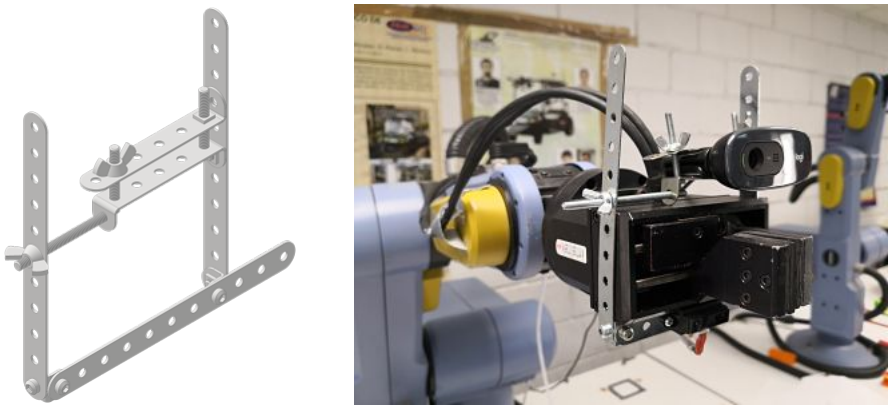
Figura 21: Gráfica del factor de conversión píxel-milímetro

En el apartado 8 se verá con más detenimiento como se implementa la ecuación de conversión de píxel a milímetro, la cuál será base para todo el sistema de medidas del proyecto.

6.2. Arquitectura

Desde un principio se decidió colocar la cámara solidaria a la pinza en la parte superior, con el sensor se hicieron varias pruebas y la posición donde había menor error en la medida era en la parte inferior de la pinza. Teniendo en cuenta estas restricciones, se diseñó una estructura capaz de fijar los componentes de forma que la vibración de la pinza no afectara a la imagen, además de poder ser desmontable fácilmente.

Para una mejor representación se realizó un esquema 3D de la estructura usando el software de diseño 3D, Inventor (Figura 22a). El resultado final de la estructura montada en la pinza queda reflejado en la figura 22b.



(a) *(b)*

Figura 22: a) Diseño de la estructura del sistema de visión. b) Estructura del sistema de visión

7. Comunicación serial

Tanto la comunicación con el Scorbot como con la placa Elegoo se realizan mediante el puerto serie. Para establecer la comunicación se utilizaron dos clases, la clase Serial, obtenida de la página oficial de Arduino [1], y la clase ACL, la cuál fue creada específicamente para el proyecto y facilita el envío y la recogida de datos de la serial.

7.1. Clase Serial

La clase se compone de los siguientes métodos: `Serial()` que inicializa la comunicación serial pasándole el nombre del puerto al que se quiere conectar; el destructor de la clase, ∼`Serial()`, que cierra la comunicación serial; el método `ReadData()` lee los datos que se transmiten a través de la serial almacenándolos en un búfer que se le pasa por parámetros, junto con el tamaño máximo permitido establecido en 255 caracteres; el método `WriteData()` envía los datos almacenados en el búfer, junto con el tamaño máximo permitido; y por último está el método `IsConnected()` el cual indica

si el puerto serie está abierto o no.

7.2. Clase ACL

Para la clase ACL se crearon en un principio más métodos de los que realmente se implementaron en el proyecto, por lo que a continuación se explicarán, primero, los métodos que se tuvieron en cuenta para el proyecto y segundo los que no.

El primer método de la clase es el `sendAcl()`, este se encarga de enviar comandos a través de la serial. Estos datos se pasan por parámetro, para mayor facilidad, como una cadena de texto de tipo `string`, luego se definen como un puntero a una cadena de caracteres de tipo `char` para pasárselo como parámetro al método de la clase Serial `WriteData()`. Es necesario incluir una pausa ya que el tiempo de respuesta del Scorbot no es inmediato.

El siguiente método es el `getPosition()`, el cual recoge la posición actual del Scorbot y la devuelve en un vector de tipo `float`. Este método usa el `sendAcl()` para enviar el comando `LISTPV POSITON`, que pertenece al lenguaje de programación ACL que se verá en el apartado 7.3. Los datos enviados por este comando son recogidos por el método de la clase Serial `ReadData()` y definidos como una cadena de texto de tipo `string` para un mejor análisis posterior. De dicha cadena se extraen los números correspondientes a cada una de las posiciones X, Y, Z, P y R del Scorbot y se almacenan como números de tipo `float` en un vector.

El conjunto de métodos que empiezan por `Shiftc` realizan todos la misma función pero para una coordenada diferente. Estos métodos modifican las coordenadas de una posición guardada del Scorbot añadiéndole un *offset*. A cada uno de los métodos se le pasa por parámetros el punto donde está

guardada la posición como una cadena de texto de tipo `string` y el *offset* como una variable de tipo `float`, a continuación se utiliza el método `sendAcl()` para enviar el comando `SHIFTC` *pos* `BY` *coord var* donde *pos* es el punto con la posición guardada, *coord* es la coordenada que se quiere variar y *var* es el *offset*. Además de la pausa que ya lleva incorporada el método `sendAcl()` se comprobó que hay que añadirle una más.

El método `getAltura()` recoge, por medio de `WriteData()`, la altura que envía el sensor de distancia a través de la placa Elegoo. Como hemos visto anteriormente se almacena la información en un búfer y luego se devuelve como un número de tipo `int`.

El siguiente grupo de métodos corresponden a los no utilizados en el proyecto debido a que en sucesivas pruebas se fueron descartando. El método `sendPosition()` envía un vector con las posiciones definidas por el usuario haciendo uso del comando `TEACH` *pos*. Los métodos que comienzan por `get` junto con el nombre de una de las 5 coordenadas, devuelven un número de tipo `float` con la posición actual de dicha coordenada. `getPositionPoint()` cumple la misma función que `getPosition()` salvo que se devuelve la posición de un punto determinado que se le pasa por parámetro, al igual que los métodos que empiezan por `getPoint` devuelven la posición de la coordenada de un punto que se le pasa por parámetro.

7.3. Software Scorbot

El software que rodea al Scorbot se compone del lenguaje de programación ACL y el emulador de terminal ATS (Figura 23). ACL es un lenguaje de programación multitarea desarrollado por Eshed Robotec programado para un conjunto de EPROMs que se encuentran dentro del controlador B. ATS es la interfaz de usuario del controlador de ACL que permite el acceso a este

mediante comunicación serial. Debido a la arquitectura basada en MS-DOS del ATS, este se ejecuta desde un emulador del sistema DOS denominado DOSBox.

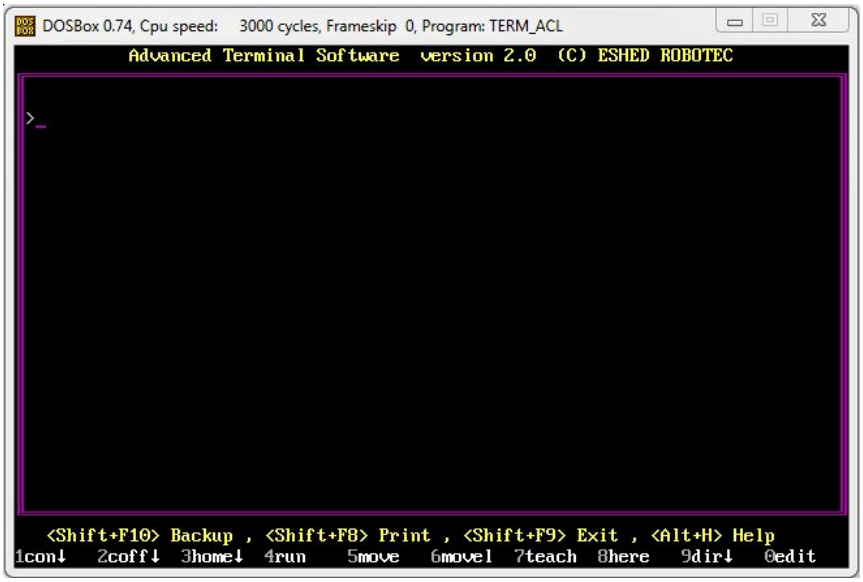
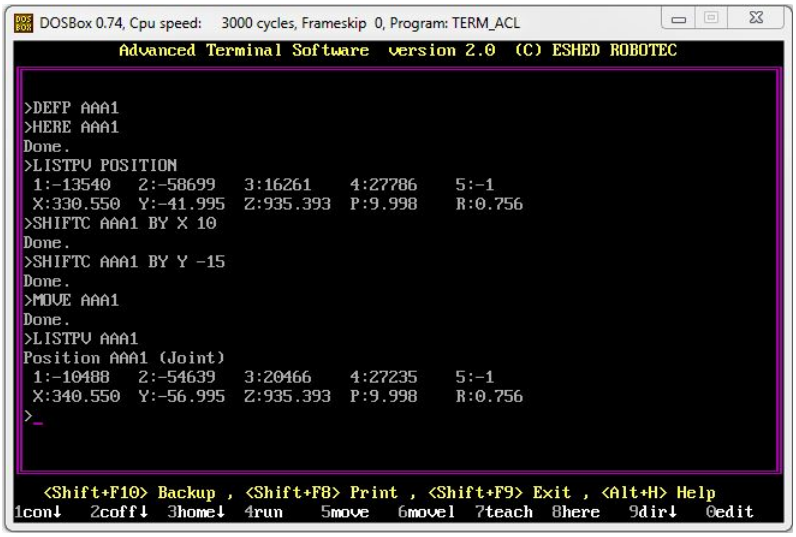
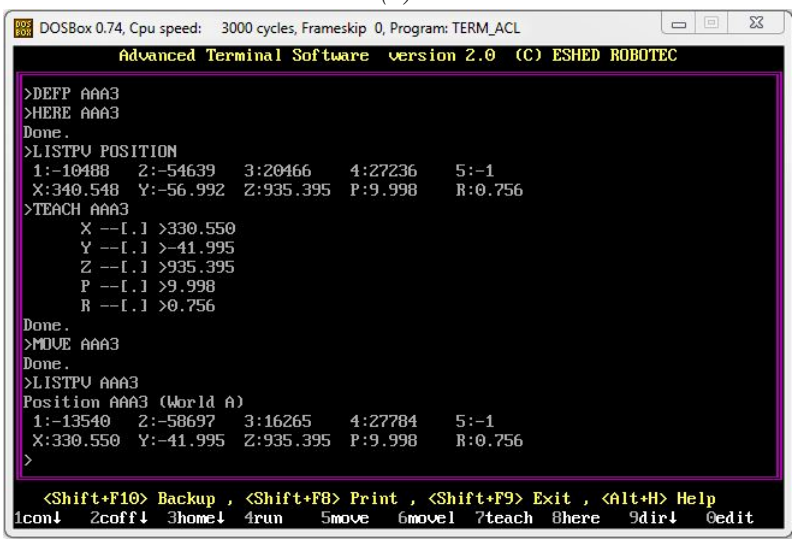


Figura 23: Emulador de terminal ATS

A continuación se van a comentar algunos de los comandos de ACL usados en el proyecto. El comando `DEFP` *pos* crea una variable de posición al que se le guardará la posición del robot. `HERE` *pos* guarda las coordenadas cartesianas de la posición actual del robot en el punto especificado. Los comandos `LISTPV POSITION` y `LISTPV` *pos* muestran por pantalla la posición actual del robot y la posición del punto guardado respectivamente. El comando `MOVE` *pos* desplaza el robot a la posición almacenada en el punto que se le indica. `TEACH` *pos* solicita al usuario que defina las posiciones cartesianas de forma individual de un punto. Por último el comando `SHIFTC` *pos* `BY` *coord var* cambia una coordenada cartesiana de una posición previamente guardada por un *offset*. La figura 24 muestra un ejemplo de uso de los comandos anteriores.



(a)



(b)

Figura 24: a) Uso del comando `SHIFTC`. b) Uso del comando `TEACH`.

8. Algoritmo de control visual con cámara on-board

El algoritmo planteado para el proyecto resuelve, en una primera parte, la detección de objetos dependiendo del color que se quiera detectar generando un fichero donde se almacenarán los datos referentes al color del objeto.

Esta información será utilizada en la segunda parte del código que extrae las características de la imagen y se encarga del control visual basado en posición.

Antes de empezar se debe hacer mención a la última clase creada para este proyecto, la clase `timer`. Esta clase resuelve la problemática de los tiempos de ejecución, es decir, la adquisición de imágenes, como veremos más adelante, ocurre dentro de un bucle infinito a una velocidad de milisegundos. Esto, comparado con la respuesta del Scorbot, produce una asincronía. Para resolver esta situación se usarán 4 métodos de la clase `timer`, `start()`, `stop()`, `reset()` y `getTime()`. Gracias a estos métodos y junto a bloques condicionales se puede controlar la ejecución de una línea de código dependiendo de los milisegundos que hayamos indicado.

8.1. Selector de figuras por color

Al iniciar el código, se muestran al usuario 4 ventanas en las que se visualizará la imagen real de la cámara, la imagen luego de aplicar los procesos del apartado 5.2 (vacía hasta que no se seleccione un color), las barras de control de los parámetros HSV y el selector del color. El usuario deberá elegir el color con el que se quiere realizar la detección utilizando el selector de color. En la ventana de umbralización se verá el resultado y dependiendo de la buena o mala detección de éste se podrán modificar los parámetros HSV con las barras de control. Una vez modificados o no, se finalizará el programa almacenando los valores en un fichero. El proceso descrito se refleja, de forma general, en la figura 25, y se describirá más detalladamente a continuación, según su orden de ejecución.

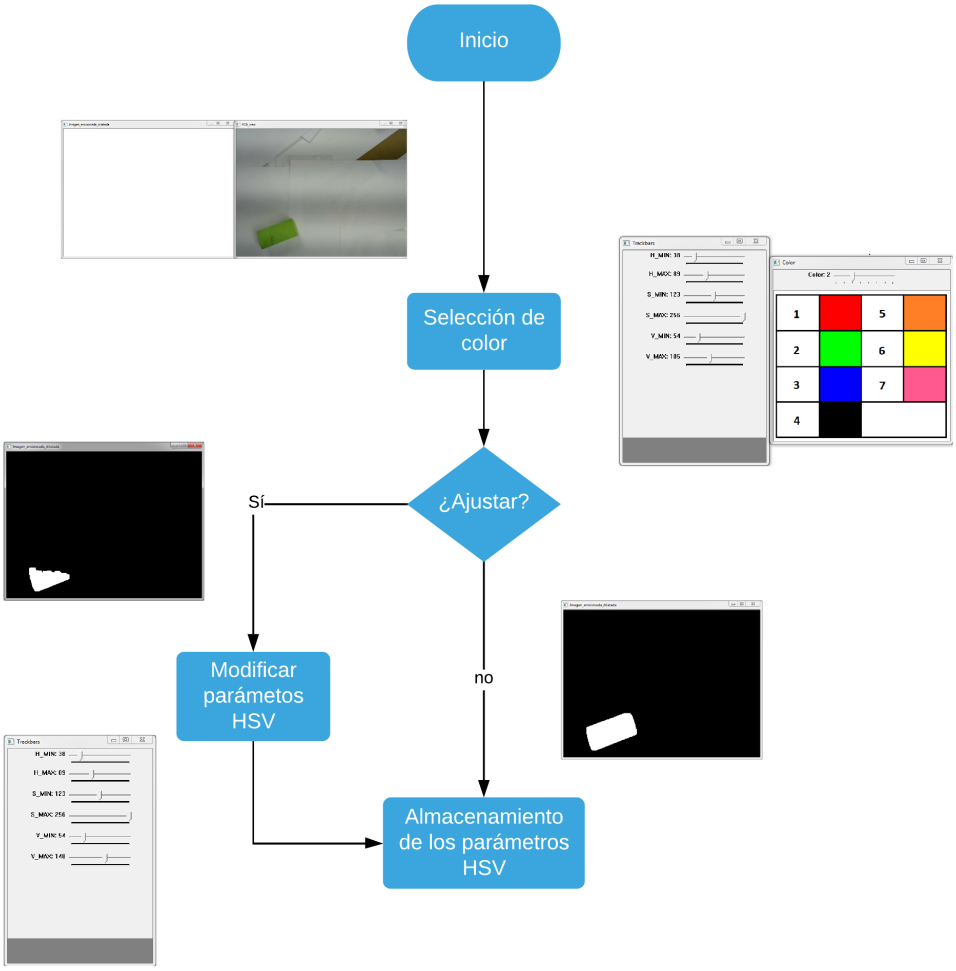


Figura 25: Proceso de selección de la figura por color

Inicializamos el código de la misma forma que se hizo en el apartado 5.2, declarando un elemento de la clase `VideoCapture`. Definimos el ancho y el alto de la ventana a 640 y 480 píxeles respectivamente, con el método `set()`.

```
VideoCapture cap(0);
if(!cap.isOpened()){
    cerr << "No se ha podido conectar con la cámara" << endl;
    return 0;
}
cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
```

Una vez definidas las propiedades de la ventana donde se van a mostrar las imágenes creamos un fichero de escritura definiendo un objeto `ofstream`, donde se van a guardar los valores HSV que se van declarando para luego volcarse en un vector de tipo `int`.

```
string nombre = "C:\\CppProjects\\Seguimiento\\Color.txt";
ofstream fichPixeles(nombre);
if (fichPixeles.fail()){
    cerr << "No se pudo abrir el archivo. TERMINAMOS." <<
    ↪  std::endl;
    return 3;
}
vector<int> pixeles(6);
```

A continuación se almacena en una variable de tipo `Mat` una imagen en la que se indican los números correspondientes a cada color.

```
cv::Mat img = imread("C:\\CppProjects\\Paleta1.png");
```

Se declara un bucle infinito donde se van a ir procesando los *frames*. Antes de entrar al bucle se llama a la función `BarrasColor()`, la cual, crea una barra de selección del color mediante la función `createTrackbar()`. A esta función se le pasa por parámetros el nombre de la barra, la ventana donde se va a mostrar, el valor que va a variar (por referencia), el valor máximo y por último un parámetro opcional que llama a la función que se le indica y le pasa el valor que varía. Además se debe indicar en que ventana se quiere mostrar la barra por medio de la función `namedWindow()`. Por último se inicializa la función a la que se le pasa el valor que varía.

```
void BarrasColor()
{
    namedWindow(Ventana_colores,1);
    createTrackbar( "Color", Ventana_colores, &colorIni, colorMax,
    ↪  Colores);
```

```
    Colores(colorIni, 0);
}
```

La imagen con los diferentes colores que se había guardado anteriormente se muestra en la misma ventana que la barra de selección (Figura 26).

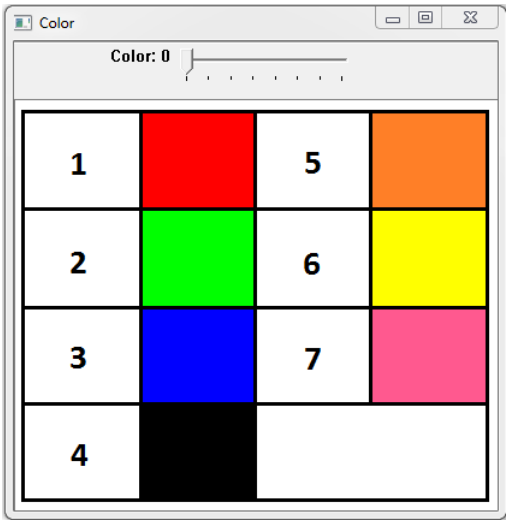


Figura 26: Paleta de colores

El valor seleccionado en la función `BarrasColor()` se envía a la función `Colores()` en la que, según el valor que se haya elegido, se seleccionarán unos valores iniciales para los parámetros HSV. Siendo el valor 0 el que inicializa los parámetros con sus valores máximos y mínimos, el resto coinciden con los colores mostrados en la figura 26.

```
void Colores(int, void*)
{
    if (colorIni == 0) {
        // Parámetros no inicializados
        H_MIN = 0;
        H_MAX = 256;
        S_MIN = 0;
        S_MAX = 256;
        V_MIN = 0;
        V_MAX = 256;
    }
```

```
    else if (colorIni == 1) {
        // rojo
        H_MIN = 0;
        H_MAX = 16;
        S_MIN = 109;
        S_MAX = 253;
        V_MIN = 34;
        V_MAX = 182;
    }
    else if (colorIni == 2){
        // Verde
        H_MIN = 38;
        H_MAX = 89;
        S_MIN = 123;
        S_MAX = 256;
        V_MIN = 54;
        V_MAX = 105;
    }[...]

}
```

Una vez elegido el color con el que se quiere comenzar se llama a la función `BarrasHSV()` que al igual que la anterior utiliza la función `createTrackbar()` para declarar 6 barras de selección correspondiente cada una a los valores máximos y mínimos de los parámetros HSV.

```
void BarrasHSV()
{
    namedWindow(Ventana_MinMax,2);
    createTrackbar( "H_MIN", Ventana_MinMax, &H_MIN, 255);
    createTrackbar( "H_MAX", Ventana_MinMax, &H_MAX, 255);
    createTrackbar( "S_MIN", Ventana_MinMax, &S_MIN, 255);
    createTrackbar( "S_MAX", Ventana_MinMax, &S_MAX, 255);
    createTrackbar( "V_MIN", Ventana_MinMax, &V_MIN, 255);
    createTrackbar( "V_MAX", Ventana_MinMax, &V_MAX, 255);

}
```

A diferencia de la anterior estas barras se muestran en una ventana aparte. En la figura 27a se muestran los parámetros no inicializados cuando

se elige la opción 0 y en la figura 27b se ha elegido la opción 1 por lo que los parámetros se inicializan acorde a los valores guardados previamente. Durante la ejecución del programa se puede cambiar el color que se quiera detectar, al igual que se pueden variar los valores de los parámetros.

(a) *(b)*

Figura 27: a) Selector de colores a 0 con parámetros no inicializados. b) Selector de colores a 1 con parámetros HSV para el color rojo.

Después de seleccionar el color que se quiere detectar pasamos a evaluar cada *frame* usando las funciones vistas en los apartados 5.2.1, 5.2.2 y 5.2.3. Primero, usando la función `cvtColor()` se cambia el espacio de color RGB a HSV, a continuación, con la función `inRange()` se aplican los valores de umbralización elegidos anteriormente dando como resultado la imagen umbralizada.

```
cvtColor(frame, HSV, CV_BGR2HSV);
inRange(HSV, Scalar(H_MIN,S_MIN,V_MIN), Scalar(H_MAX,S_MAX,V_MAX),
 ↪  threshold);
```

La imagen resultante se le pasa como parámetro a la función `ErodeDilate()` donde se le aplica un proceso doble de apertura, creando dos elementos estructurantes para la erosión (`erode()`) y la dilatación (`dilate()`) de dimensiones 3x3 y 8x8 respectivamente.

```
void ErodeDilate(cv::Mat &imgErDi)
{
    cv::Mat erodeElement =
     ↪  getStructuringElement(MORPH_RECT,Size(3,3));
    cv::Mat dilateElement =
     ↪  getStructuringElement(MORPH_RECT,Size(8,8));

    erode(imgErDi,imgErDi,erodeElement);
    erode(imgErDi,imgErDi,erodeElement);

    dilate(imgErDi,imgErDi,dilateElement);
    dilate(imgErDi,imgErDi,dilateElement);
}
```

Por último, los parámetros HSV que se hayan definido se guardan en un vector para que una vez se cierre el bucle infinito, se escriban en el fichero que se creó al inicializar el programa.

```
pixeles[0] = H_MIN;
pixeles[1] = S_MIN;
pixeles[2] = V_MIN;
pixeles[3] = H_MAX;
pixeles[4] = S_MAX;
pixeles[5] = V_MAX;
```

```
for (size_t i = 0; i < pixeles.size(); i++){
    fichPixeles << pixeles[i] << endl;
}
```

8.2. Seguimiento

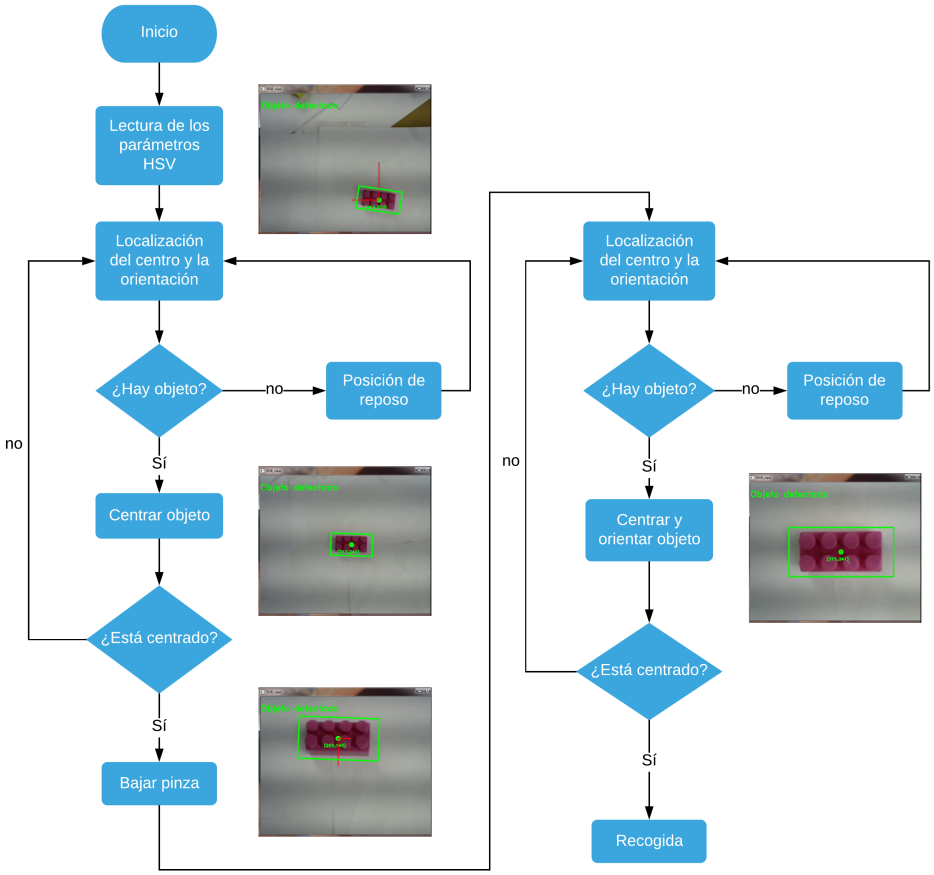


Figura 28: Proceso en el que se centra y se orienta el efector final

Al ejecutarse el código aparecen dos ventanas, en una se visualiza la imagen umbralizada y en la otra la imagen real. En una primera parte, el algoritmo intentará centrar el objeto realizando pequeñas variaciones en los ejes de la pinza. Una vez centrado, la pinza bajará hasta una distancia determinada y volverá a centrar el objeto a la vez que lo orienta paralelamente

a la pinza. Cuando esté centrado y orientado el programa terminará pasando al proceso de recogida. Hay que tener en cuenta que centrar y orientar el objeto conlleva varias iteraciones y, además, en cualquier momento la cámara puede no detectar ningún objeto, por lo que, en estos casos la pinza vuelve a una posición de reposo conocida. En la figura 28 se muestra un esquema general del funcionamiento del código y, a continuación, se detallarán los pasos descritos en orden de ejecución.

La inicialización del programa es la misma que en el comienzo del apartado anterior con la única diferencia que se debe pasar como argumento por la línea de comandos la altura del objeto que se quiere recoger.

```
if (argc < 2){
    cerr << "Hay que especificar la altura del objeto en mm.
    ↪  Terminamos" << endl;
    return 0;
}
float AltObjeto = stof(argv[1]);
VideoCapture cap(0);
if(!cap.isOpened()){
    cerr << "No se ha podido conectar con la cámara" << endl;
    return 0;
}
cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
```

A continuación se crea un fichero de lectura mediante el objeto `ifstream` con el que se leerán y almacenarán en un vector de tipo `int` los valores de los parámetros HSV que se definieron anteriormente.

```
string colores = "Color.txt";
ifstream fichColor(colores);
if (fichColor.fail()){
    cerr << "No se pudo abrir el archivo. Terminamos" << endl;
    return 0;
}
int valor;
vector<int> vColores;
```

```
fichColor >> valor;
while (fichColor.good()){
    vColores.push_back(valor);
    fichColor >> valor;
}
H_MIN = vColores[0];
S_MIN = vColores[1];
V_MIN = vColores[2];
H_MAX = vColores[3];
S_MAX = vColores[4];
V_MAX = vColores[5];
```

Debido a que la conexión serial con la placa Elegoo y con el Scorbot se va a realizar a través de distintas funciones, se declaran dos elementos de la clase Serial como puntero. A cada uno se le asigna un espacio dinámico que contiene un valor de tipo Serial inicializado con el nombre del puerto COM al que se quiere acceder.

```
Serial* arduino = new Serial("\\\\.\\COM4");
if (!arduino -> IsConnected()){
    cout << "\nNo se ha podido conectar con arduino" << endl;
    return 0;
}
Serial* scorbot = new Serial("\\\\.\\COM1");
if (!scorbot -> IsConnected()){
    cout << "\nNo se ha podido conectar con el scorbot" << endl;
    return 0;
}
```

En una variable de tipo `string` se almacena el nombre de un punto que previamente se ha creado con el comando `DEFP`. Una vez definido el nombre del punto se declara un elemento de la clase Acl y se utiliza el comando `HERE` para almacenar la posición actual del Scorbot en el punto que se acaba de definir.

```
string punto = "RE43";
Acl enviar;
enviar.sendAcl(scorbot, "here " + punto);
```

Después de inicializar estos parámetros se define un bucle infinito donde se van a ir ejecutando las distintas funciones hasta que se termine la localización del objeto. Al igual que en el apartado anterior, a la imagen se le va a cambiar del espacio de color de RGB a HSV con la función `cvtColor()`, a continuación, se le aplica el proceso de umbralización mediante la función `inRange()` con los parámetros almacenados en el fichero y, aunque para esta parte del proyecto no es necesario, también se muestra en una ventana el proceso de apertura que ejecuta la función `ErodeDilate()` debido a que si existe algún problema de calibración se pueden detectar mejor con la imagen umbralizada.

El siguiente paso es definir la localización del centro del objeto, para ello se llama a la función `Centro()` a la que se le pasa la imagen final del proceso anterior junto con el *frame* que se esté evaluando en ese momento; cuatro variables, dos de tipo `float` y dos de tipo `int`, y el elemento de la clase Serial. Salvo la imagen umbralizada y el elemento de la clase Serial los demás se pasan por referencia, ya que su valor va ha ser utilizado más adelante.

Para la localización del centro se va a proceder de igual forma que en el apartado 5.3.2 aunque para este caso se van a definir dos requisitos: el primero es que el objeto debe ser el que posea la mayor área posible y en el segundo se define un número máximo de objetos en la imagen, lo que hace que, en los dos casos, se reduzca la posibilidad de detectar objetos no deseados.

Para llevar a cabo estos requisitos se declara un vector de 4 dimensiones

donde cada valor es un entero, este vector almacena los contornos de forma ordenada mediante diferentes niveles de jerarquía. El tamaño de este vector indicará cuantos contornos existen en la imagen, además se iterará cada nivel de jerarquía trabajando solo con el contorno inicial.

El tamaño del objeto se definirá en base a su área, si el área del objeto es menor que 400 píxeles probablemente sea ruido, y si el área es mayor que 3/2 del tamaño total de la imagen el filtro que se le ha aplicado no es el idóneo. Para obtener el área mayor se almacena el área del contorno que se está evaluando y se compara con el área de los otros contornos en cada iteración.

```
vector< vector<Point> > contours;
vector<Vec4i> hierarchy;
findContours(temp, contours, hierarchy, CV_RETR_CCOMP,
 ↪  CV_CHAIN_APPROX_SIMPLE);
double refArea = 0;
bool objectFound = false;
if (hierarchy.size() > 0) {
    int numObjects = hierarchy.size();
    if(numObjects<MAX_NUM_OBJECTS){
        for (int index = 0; index >= 0; index =
         ↪  hierarchy[index][0]) {
            Moments moment = moments((cv::Mat)contours[index]);
            double area = moment.m00;
            if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA &&
             ↪  area>refArea){
                x = moment.m10/area;
                y = moment.m01/area;
                objectFound = true;
                refArea = area;
            }
        }
```

Una vez definido el centro se muestra en la ventana, junto con unas líneas que indican de forma visual la distancia del centro del objeto al centro de la imagen. También se visualiza un mensaje si el objeto se detecta correctamente y otro si no se cumple alguno de los dos requisitos vistos anteriormente.

La distancia en milímetros del objeto al centro de la imagen se calcula teniendo en cuenta la ecuación que refleja la figura 21. Con el método `getAltura()` de la clase Acl se almacena la distancia que ofrece el sensor Sharp y teniendo en cuenta que la altura almacenada está en milímetros y que el sensor no detecta objetos a partir de los 100 mm, se ajusta para que solo los valores correctos sean evaluados. Una vez procesada la altura se define el factor de conversión a utilizar. Este multiplica la distancia en el eje horizontal y vertical de la imagen que separa al objeto del centro de la imagen. Si no se está captando ningún objeto el valor de estos parámetros será diferente de cualquier número comprendido entre -1 y 1, esto se hace así para que no se activen funciones que se verán más adelante.

Las variables que almacenan el número total de objetos, la altura y la distancia en el eje horizontal y vertical corresponden con las variables de tipo `int` y de tipo `float` mencionadas anteriormente.

```
Objects = hierarchy.size();
Acl enviar;
int altura = enviar.getAltura(arduino);
if (altura != 0 && altura > 0 && altura >= 85){
    alt = altura;
    float mmXpixel =  0.001*pow(altura, 1.0145);
    if (Objects >= 1){
        DcentY = (y - MID_FRAME_HEIGHT)*mmXpixel;
        DcentX = (x - MID_FRAME_WIDTH)*mmXpixel;
    }else if (Objects == 0){
        DcentY = 3;
        DcentX = 3;
    }
}
```

Al finalizar la localización del centro se llama a la función `Orientacion()` que como su nombre indica, define la orientación del objeto respecto a la base del robot. A esta función se le pasa por parámetros la imagen umbralizada, el *frame* que se está evaluando y por referencia una variable de tipo `float`

que almacena el valor del ángulo calculado.

La forma en que la función `Orientacion()` calcula la orientación es similar a la vista en el apartado 5.3.1 aunque aquí se selecciona el contorno mayor y se trabaja con él. Esto se resuelve definiendo una variable que almacena el tamaño del contorno actual y lo compara con el tamaño del contorno siguiente.

```
vector< vector<Point> > contours;
findContours(temp, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);
unsigned max_size = 0;
int index = 0;
for (int i = 0; i < contours.size(); i++) {
    if (contours[i].size() > max_size) {
        max_size = contours[i].size();
        index = i;
    }
}
```

Después de calcular la elipse se muestra en la imagen el rectángulo donde está encerrado el objeto. Por último, se calcula el ángulo como el complementario al que ofrecía la función `fitEllipse()` para obtenerlo en relación al lado mayor del objeto.

```
if (contours[index].size() >= 5) {
    RotatedRect minEllipse = fitEllipse( cv::Mat(contours[index]));
    Point2f verts[4];
    minEllipse.points(verts);
    for(int i = 0; i < 4; ++i){
        line(frame,verts[i],verts[(i+1)%4],Scalar(0,255,0),4);
    }
    AngleR = minEllipse.angle-90;
}
```

A continuación se explicarán las funciones `Colocacion()` y `PosRecoger()` para más adelante explicar en que momento del programa se llama a cada una.

La función `Colocacion()` posiciona la pinza en el centro del objeto y una vez centrado la pinza baja hasta una altura de 100 milímetros con respecto a la mesa de trabajo para luego colocarse perpendicularmente al objeto. Todos estos pasos no se pueden realizar de forma consecutiva si no que deben pasar varias iteraciones entre ellos. La solución que se tomó fue definir 3 variables globales de tipo `bool` con las que se controlará cada movimiento.

```
bool Colocacion(Serial *scorbot, float &DcentX, float &DcentY,
↪  float &AngleP, int &alt2, string punto){

    Acl enviar;
    Conv matrix;
    vector<float> posicion;
    int restaAltura;
    float restaAngulo;

    posicion = enviar.getPosition(scorbot);
    vec p = { posicion[0], posicion[1], posicion[2] };
    vec DistPuntoCam = { DcentX, DcentY, 0, 1 };
    vector<float> distancias = matrix.Trans(p, DistPuntoCam);
    enviar.ShiftcX(scorbot, punto, distancias[0]);
    enviar.ShiftcY(scorbot, punto, distancias[1]);

    if (actAltscorbot2) {
        restaAngulo = 90 + posicion[3];
        if (posicion[3] > -90){
            enviar.ShiftcP(scorbot, punto, -restaAngulo);
        }else if (posicion[3] < -90){
            enviar.ShiftcP(scorbot, punto, restaAngulo);
        }
        enviar.sendAcl(scorbot, "MOVE " + punto);
        actAltscorbot2 = false;
        return true;
    }

    if (actAltscorbot){
        actAltscorbot = false;
        actAltscorbot2 = true;
    }

    if (actBajada){
        restaAltura = alt2 - 100;
        enviar.ShiftcZ(scorbot, punto, -restaAltura);
```

```
        actBajada = false;
        actAltscorbot = true;
    }

    if (DcentX >= -1 && DcentX <= 1.1 && DcentY >= -1 && DcentY <=
     ↪  1.1){
        actBajada = true;
    }

    enviar.sendAcl(scorbot, "MOVE " + punto);
    return false;

    }
```

En el inicio del código se definen dos vectores en los que se almacena la posición actual del Scorbot y la distancia al centro de la imagen dada por la función `Centro()`. Por medio del método `Trans()` de la clase Conv se obtienen las distancias del objeto respecto de la base del Scorbot. Una vez obtenidas se modifica la posición del punto guardado mediante los métodos `ShiftcX()` y `ShiftcY()` de la clase Acl. Como las variables de tipo `bool` se inicializan a *false* se corrige la posición del Scorbot mediante el comando `MOVE`.

El proceso anterior se repite varias veces hasta que la distancia al centro de la pinza se encuentra entre -1 y 1 milímetro cuando se activa el booleano de bajada. En la siguiente iteración se le resta a la altura actual el valor necesario para situar la pinza a una altura de 100 milímetros respecto al objeto. Debido a que el movimiento del Scorbot en el eje z se demora un tiempo una vez se ha ejecutado la orden de bajada se activan y desactivan unas variables booleanas que permiten que pasen varias iteraciones antes de que se ejecute el siguiente movimiento.

El último movimiento que ejecuta esta función corresponde a la colocación perpendicular de la pinza teniendo en cuenta que el ángulo puede ser

mayor o menor que -90. Una vez se ha ejecutado este movimiento se devuelve un valor de *true*.

La función `posRecoger()`, una vez que se ha colocado la pinza a 100 milímetros y perpendicular al objeto, se encarga de volver a posicionarla en el centro de la imagen. Se emplea el mismo código que se utilizó para centrar la pinza en la función anterior además de orientarla paralelamente al lado mayor del objeto. Una vez que los valores de distancia al centro y el ángulo entre la horizontal y el lado largo del objeto se encuentran entre los valores -1 y 1 milímetros y grados respectivamente la función devuelve un *true*.

```
bool PosRecoger(Serial *scorbot, float &AngleR, string punto, float
↪  &DcentX, float &DcentY){
    Acl enviar;
    Conv matrix;
    vector<float> posicion;

    posicion = enviar.getPosition(scorbot);
    vec p = { posicion[0], posicion[1], posicion[2] };
    vec DistPuntoCam = { DcentX, DcentY, 0, 1 };
    vector<float> distancias = matrix.Trans(p, DistPuntoCam);
    enviar.ShiftcX(scorbot, punto, distancias[0]);
    enviar.ShiftcY(scorbot, punto, distancias[1]);
    enviar.ShiftcR(scorbot, punto, AngleR);

    enviar.sendAcl(scorbot, "MOVE " + punto);

    if (DcentX >= -1 && DcentX <= 1.1 && DcentY >= -1 && DcentY <=
    ↪  1.1 && AngleR >= -1 && AngleR <= 1){
        return true;
    }else {
        return false;
    }
    return false;
}
```

Después de explicar los procesos que realizan la funciones anteriores se verá, a continuación, como actúan dentro del bucle infinito. Se ha tenido en cuenta la clase timer (Apartado 8) y un conjunto de activadores que

permitirán el control en ejecución del programa debido a que las diferentes velocidades de actuación del Scorbot y del sistema de visión no permiten la llamada de estas funciones de forma continua.

```
if (Objects != 0 && !act2){
    act1 = true;
    if (!colocado){
        t.start();
        t.stop();
        unsigned long duration = t.getTime();
        if (duration >= 700){
            colocado = Colocacion(scorbot, DcentX, DcentY, AngleP,
             ↪  altura, punto);
            t.reset();
            t.start();
        }else {
            t.start();
        }
    }
    if (colocado){
        t3.start();
        t3.stop();
        unsigned long duration = t3.getTime();
        if (duration >= 700){
            recogido = PosRecoger(scorbot, AngleR, punto, DcentX,
             ↪  DcentY);
            if (recogido){
                enviar.ShiftcX(scorbot, punto, 48);
                enviar.ShiftcY(scorbot, punto, 8);
                destroyAllWindows();
                break;
            }
            t3.reset();
            t3.start();
        }else {
            t3.start();
        }
    }
}else if(act1){
    enviar.sendAcl(scorbot, "MOVE CASA");
    act1 = false;
    act2 = true;
    actBajada = false;
    actAltscorbot = false;
    colocado = false;
    recogido = false;
    t2.start();
```

```
}
if (act2){
    t2.stop();
    unsigned long duration = t2.getTime();
    if (duration >= 1350){
        enviar.sendAcl(scorbot, "here " + punto);
        t2.reset();
        act2 = false;
    }else {
        t2.start();
    }
}
```

Se van a diferenciar dos partes: cuando el Scorbot está en reposo y cuando el Scorbot se está centrando teniendo en cuenta cuando no se detecta ningún objeto y cuando sí respectivamente. Cuando no se detecte ningún objeto primero se enviará al Scorbot a una posición conocida de reposo, en esta parte las variables globales retornan a su estado inicial. El movimiento desde cualquier posición a la posición de reposo requiere un tiempo por lo que mientras ese tiempo no se haya cumplido no se podrá acceder a la parte cuando el Scorbot se está centrando, aunque se haya detectado un objeto, ni a la parte donde se le indica que vuelva a la posición de reposo. Una vez finalizado se vuelve a redefinir el punto con el comando `HERE`.

La parte en la que el Scorbot se está centrando se divide en dos situaciones: cuando el Scorbot está centrado arriba y cuando el Scorbot está centrado abajo. La primera parte llama a la función `Colocacion()` cada cierto tiempo, para evitar que el Scorbot se sature, y una vez ésta devuelva un *true* se activará la segunda parte que, al igual que la anterior, llamará a la función `PosRecoger()` cada vez que se cumpla el tiempo especificado.

Teniendo en cuenta que hasta ahora el objeto se ha centrado respecto del centro de la imagen, una vez que la función `PosRecoger()` devuelva un *true*, se corregirá la posición de la pinza aplicándole un desfase en el eje

horizontal y vertical y, a continuación, se saldrá del bucle.

En esta última parte del código se explica como se ha diseñado la recogida. Por la línea de comandos se pasa la altura del objeto y para una mejor recogida se trabajará con la mitad de dicha medida. Teniendo en cuenta que el sensor a estas distancias no puede trabajar, se toma la distancia que separa la pinza de la mesa que proporciona el Scorbot. Esta altura tiene un desfase de 33 milímetros respecto del 0 por lo que la distancia que tenga que bajar el Scorbot será igual a la altura dada menos la suma de la mitad del ancho del objeto y el desfase de 33 milímetros. Por último, se girará la pinza 90º para posicionarla de forma perpendicular al objeto y se moverá al punto definido. Los comandos `OPEN` y `CLOSE` abren y cierran la pinza respectivamente.

```
AltScorbot = enviar.getZ(scorbot);
float MidAltObjeto = AltObjeto/2;
float resta = (AltScorbot - MidAltObjeto) - 33;
enviar.ShiftcZ(scorbot, punto, -resta);
enviar.ShiftcR(scorbot, punto, 90);
enviar.sendAcl(scorbot, "OPEN");
Sleep(500);
enviar.sendAcl(scorbot, "MOVE " + punto);
Sleep(500);
enviar.sendAcl(scorbot, "CLOSE");
Sleep(500);
```

9. Resultados obtenidos

En este apartado se mostrará la aplicación del algoritmo en diferentes situaciones. En cada vídeo aparece la ejecución del código mientras se ve el comportamiento de la pinza en tiempo real.

La trayectoria que se creó una vez el objeto es recogido se define a continuación.

```
enviar.ShiftcZ(scorbot, punto, 154);
enviar.sendAcl(scorbot, "MOVE " + punto);
Sleep(500);
enviar.sendAcl(scorbot, "MOVE ra1");
Sleep(500);
enviar.sendAcl(scorbot, "MOVE ra2");
Sleep(500);
enviar.sendAcl(scorbot, "OPEN");
Sleep(500);
enviar.sendAcl(scorbot, "MOVE casa");
Sleep(500);
enviar.sendAcl(scorbot, "CLOSE");
```

Utilizando el comando `ShiftcZ()` se le añade un *offset* positivo en el eje z al punto almacenado, por lo que la pinza se aleja de la mesa dicha cantidad. A continuación, se actualiza la posición del robot y se le envía a varias posiciones previamente definidas, colocando el objeto en la última. Por último, el robot vuelve a la posición de reposo.

Este primer vídeo es un ejemplo de aplicación del algoritmo de selección de figuras. Se elige el color naranja para detectar el objeto y al no obtener una imagen estable se modifican los parámetros HSV hasta conseguirla.

Tanto en el vídeo 1 como en el vídeo 2 se observa cómo se ejecuta el algoritmo de seguimiento y se produce la recogida, para dos objetos de color azul y rojo.

En el siguiente vídeo se muestra como se actualiza la posición del objeto aunque esta haya variado.

10. Conclusión

En el presente Trabajo de Fin de Grado (TFG) el objetivo principal ha sido la localización y recogida de un objeto en tiempo real, además de

tener la posibilidad de incluir este procedimiento en las sesiones prácticas. A día de hoy en el laboratorio de robótica solo se implementa un sistema de visión sin realimentación, con este trabajo se intentará dotar a los alumnos que realicen las prácticas en el laboratorio de una mejor comprensión de un sistema de visión en bucle cerrado.

En una primera parte, y partiendo de un conocimiento básico de las diferentes técnicas de visión actuales, se buscó la forma adecuada de localizar la posición y la orientación del objeto, con ayuda de la librería de visión artificial OpenCv. Al realizar la localización del objeto por medio del color uno de los inconvenientes fue la variación de la iluminación durante el desarrollo de la tarea, aunque ajustando los parámetros se podía conseguir una detección estable.

Una vez resuelta esta primera parte, se diseñó la estructura donde se iban a localizar los componentes del sistema de visión a la vez que se resolvía la calibración del sensor Sharp. Esta calibración se realizó varias veces debido a que la iluminación afectaba a la sensibilidad del sensor.

Para la comunicación serial con el Scorbot se decidió crear varias clases que simplificasen el envío y la recepción de información. El software del Scorbot no está pensado para enviar información que se pueda manipular, sino que los datos que transmite son meramente informativos. Esto genera que las clases no posean la robustez que se querría y generen fallos por mala lectura de dicha información.

Por último, se resolvió la cinemática y se decidió determinar la actualización de la posición mediante la suma de un *offset* en los ejes cartesianos correspondientes. Otra solución que se barajó, aunque no pudo probarse, definía la posición directamente enseñándole los puntos exactos donde debía dirigirse el Scorbot.

Desde un punto de vista docente, este trabajo puede servir de base para resolver, por ejemplo, el reconocimiento de objetos, ya que ofreció muy buenos resultados, en cambio, otros apartados como la comunicación con el Scorbot, entrañan muchas dificultades debido a las diferentes velocidades de trabajo por lo que convendría buscar otras formas de conexión.

11. Conclusion

In the present Final Degree Project (FDP) the main objective has been to define the location of an object and pick it up with a robotic arm in real time, besides having the possibility of including this procedure in the practical sessions of several degree's subjects, as well as some master degree's subject. Nowadays only a vision system without feedback is implemented in the robotics laboratory. This work tries to provide the students who perform the practices in the laboratory a better understanding of a closed loop vision system.

IFirst, starting from a basic knowledge of the different current vision techniques, an appropriate way to locate the position and orientation of the object was studied, with the help of the artificial vision library OpenCv. In the process of locating the object through color, one of the main disadvantages was the variation of the lighting during the development of the task, although adjusting the camera parameters could be achieved a stable detection.

Once this first part was solved, the structure where the components of the vision system were going to be located was designed, while the Sharp sensor calibration was solved. This calibration was performed several times because the lighting affected the sensitivity of the sensor.

It was decided to create several classes for serial communication with

the Scorbot, trying to simplify the sending and receiving information. The Scorbot software is not designed to send information that can be manipulated, but the information transmitted by the Scorbot is merely informative. This causes that the classes do not have the desirable robustness and could generate failures due to bad reading of the transmitted information.

Finally, the kinematics was solved. It was decided to determine the update of the position by adding an offset on the corresponding cartesian axes. Another solution that was taken into account, although it could not be proved, was to define the position directly by showing the exact points where the Scorbot should go.

From a teaching point of view, this work can serve as a base for solving, for example, the recognition of objects, since it offered very good results. However, other sections such as communication with the Scorbot, entail certain difficulties due to the different speeds of work, so it would be convenient to look for other ways of connection.

Referencias

[1] Arduino. Arduino and c++ (for windows). `https://playground.arduino.cc/Interfacing/CPPWindows/`, 2019.

[2] de la Escalera Hueso, A. *Visión por computador: fundamentos y métodos*. Pearson Educación, 2001.

[3] de Topología Computacional y Matemática Aplicada, G. Momentos. `http://grupo.us.es/gtocoma/pid/pid10/doc.htm`.

[4] de Wikipedia, C. Espacio de color. `https://es.wikipedia.org/wiki/Espacio_de_color`, 2018.

[5] Eshed Robotec. *Scorbot-ER IX User's Manual*, Marzo 1999.

[6] OpenCV. Opencv documentation. `https://docs.opencv.org/2.4.13.7/index.html`, 2018.

[7] OpenCV. About opencv. `https://opencv.org/about.html`, 2019.

[8] Pajares Martinsanz, G., and de la Cruz García, J. M. *Visión por computador. Imágenes digitales y aplicaciones*, 2 ed. RA-MA Editorial, 2008.

[9] Snderson, C., and Curtin, R. Armadillo: a template-based c++ library for linear algebra. `http://arma.sourceforge.net/armadillo_joss_2016.pdf`, 2016.

[10] Torres, F., Gil, P., Puente, S. T., Aracil, R., and Pomares, J. *Robots y sistemas sensoriales*. Pearson Educación, 2002.

[11] Weitzenfeld, D. A. Visión aibo. `http://cannes.itam.mx/Alfredo/Espaniol/Cursos/Robotica/Material/VisionAIBO.pdf`, 2011.

Anexo A Seguimiento de objetos

```
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <armadillo>
#include <vector>
#include <string>
#include <fstream>
#include "SerialClass.hpp"
#include "ACLclass.hpp"
#include "CronoClass.hpp"
#include "ConvCoordCam.hpp"

#define PI (4*atan(1))

using namespace cv;
using namespace std;
using namespace arma;

const string windowName_ErDi = "Imagen_erosionada_dilatada";
const string windowName_RGB = "RGB_view";

const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;
const int MID_FRAME_WIDTH = 317;
const int MID_FRAME_HEIGHT = 239;
const int MAX_NUM_OBJECTS=50;
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;

int H_MIN = 0;
int H_MAX = 0;
int S_MIN = 0;
int S_MAX = 0;
int V_MIN = 0;
int V_MAX = 0;

bool actBajada = false;
bool actAltscorbot = false;
bool actAltscorbot2 = false;

void Orientacion(cv::Mat threshold, cv::Mat &frame, float &AngleR)
{
    cv::Mat temp;
    threshold.copyTo(temp);
```

```
    vector< vector<Point> > contours;
    findContours(temp, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

    unsigned max_size = 0;
    int index = 0;
    for (int i = 0; i < contours.size(); i++) {
        if (contours[i].size() > max_size) {
            max_size = contours[i].size();
            index = i;
        }
    }

    if (contours.size() == 0) goto continue_loop;

    if (contours[index].size() >= 5) {

        RotatedRect minEllipse = fitEllipse(
        ↪  cv::Mat(contours[index])) ;

        Point2f verts[4];
        minEllipse.points(verts);
        for(int i = 0; i < 4; ++i){
            line(frame,verts[i],verts[(i+1)%4],Scalar(0,255,0),4);
        }
        AngleR = minEllipse.angle-90;
    }

    continue_loop:;
}

void Centro(cv::Mat threshold, cv::Mat &frame, Serial* arduino,
↪  float &DcentX, float &DcentY, int &Objects, int &alt)
{
    cv::Mat temp;
    threshold.copyTo(temp);
    int x, y;
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;
    findContours(temp, contours, hierarchy, CV_RETR_CCOMP,
    ↪  CV_CHAIN_APPROX_SIMPLE);
    double refArea = 0;
    bool objectFound = false;
    if (hierarchy.size() > 0) {
        int numObjects = hierarchy.size();
        if(numObjects<MAX_NUM_OBJECTS){
            for (int index = 0; index >= 0; index =
            ↪  hierarchy[index][0]){

                Moments moment = moments((cv::Mat)contours[index]);
```

```
                double area = moment.m00;

                if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA &&
                 ↪  area>refArea){
                    x = moment.m10/area;
                    y = moment.m01/area;
                    objectFound = true;
                    refArea = area;
                }else objectFound = false;
            }
            if(objectFound ==true){
                putText(frame,"Objeto detectado", Point(0,50), 2,
                 ↪  1, Scalar(0,255,0),2);
                circle(frame,Point(x,y), 9, Scalar(0,255,0), -1);
                putText(frame,
                 ↪  "("+to_string(x)+","+to_string(y)+")",
                 ↪  Point(x-50,y+30), 1, 1, Scalar(0,255,0),2);

                line(frame, Point2f(x,y), Point2f(x,
                 ↪  MID_FRAME_HEIGHT),Scalar(0, 0, 255), 2);
                line(frame, Point2f(x,y), Point2f(MID_FRAME_WIDTH,
                 ↪  y),Scalar(0, 0, 255), 2);

            }else putText(frame,"Mucho ruido,
             ↪  ajustar",Point(0,50),1,2,Scalar(0,0,255),2);
        }
    }
    Objects = hierarchy.size();
    Acl enviar;
    int altura = enviar.getAltura(arduino);
    if (altura != 0 && altura > 0 && altura >= 85){
        alt = altura;
        float mmXpixel =  0.001*pow(altura, 1.0145);

        if (Objects >= 1){
            DcentY = (y - MID_FRAME_HEIGHT)*mmXpixel;
            DcentX = (x - MID_FRAME_WIDTH)*mmXpixel;
        }else if (Objects == 0){
            DcentY = 3;
            DcentX = 3;
        }

    }
}

void ErodeDilate(cv::Mat &imgErDi)
{
    cv::Mat erodeElement = getStructuringElement(MORPH_RECT,
     ↪  Size(3,3));
```

```
    cv::Mat dilateElement = getStructuringElement(MORPH_RECT,
    ↪  Size(8,8));

    erode(imgErDi, imgErDi, erodeElement);
    erode(imgErDi, imgErDi, erodeElement);

    dilate(imgErDi, imgErDi, dilateElement);
    dilate(imgErDi, imgErDi, dilateElement);
}

string Punto(){
    srand (time(NULL));
    char cch = 97 + rand()%26;
    char cch2 = 97 + rand()%26;
    char cch3 = 97 + rand()%26;
    string nombre = string(1, cch) + string(1, cch2) + string(1,
    ↪  cch3) + to_string(rand()%100);
    return nombre;
}

bool Colocacion(Serial *scorbot, float &DcentX, float &DcentY,
↪  float &AngleP, int &alt2, string punto){

    Acl enviar;
    Conv matrix;
    vector<float> posicion;
    int restaAltura;
    float restaAngulo;

    posicion = enviar.getPosition(scorbot);
    vec p = { posicion[0], posicion[1], posicion[2] };
    vec DistPuntoCam = { DcentX, DcentY, 0, 1 };
    vector<float> distancias = matrix.Trans(p, DistPuntoCam);
    enviar.ShiftcX(scorbot, punto, distancias[0]);
    enviar.ShiftcY(scorbot, punto, distancias[1]);


    if (actAltscorbot2) {
        restaAngulo = 90 + posicion[3];
        if (posicion[3] > -90){
            enviar.ShiftcP(scorbot, punto, -restaAngulo);
        }else if (posicion[3] < -90){
            enviar.ShiftcP(scorbot, punto, restaAngulo);
        }
        enviar.sendAcl(scorbot, "MOVE " + punto);
        actAltscorbot2 = false;
        return true;
    }
```

```
    if (actAltscorbot){
        actAltscorbot = false;
        actAltscorbot2 = true;
    }

    if (actBajada){
        restaAltura = alt2 - 100;
        enviar.ShiftcZ(scorbot, punto, -restaAltura);
        actBajada = false;
        actAltscorbot = true;
    }

    if (DcentX >= -1 && DcentX <= 1.1 && DcentY >= -1 && DcentY <=
    ↪  1.1){
        actBajada = true;
    }

    enviar.sendAcl(scorbot, "MOVE " + punto);

    return false;

}

bool PosRecoger(Serial *scorbot, float &AngleR, string punto, float
↪  &DcentX, float &DcentY){

    Acl enviar;
    Conv matrix;
    vector<float> posicion;

    posicion = enviar.getPosition(scorbot);
    vec p = { posicion[0], posicion[1], posicion[2] };
    vec DistPuntoCam = { DcentX, DcentY, 0, 1 };
    vector<float> distancias = matrix.Trans(p, DistPuntoCam);
    enviar.ShiftcX(scorbot, punto, distancias[0]);
    enviar.ShiftcY(scorbot, punto, distancias[1]);
    enviar.ShiftcR(scorbot, punto, AngleR);

    enviar.sendAcl(scorbot, "MOVE " + punto);

    if (DcentX >= -1 && DcentX <= 1.1 && DcentY >= -1 && DcentY <=
    ↪  1.1 && AngleR >= -1 && AngleR <= 1){
        return true;
    }else {
        return false;
    }
    return false;
}
```

```
int main(int argc, char *argv[])
{
    if (argc < 2){
        cerr << "Hay que especificar la altura del objeto en mm.
        ↪  Terminamos" << endl;
        return 0;
    }
    float AltObjeto = stoi(argv[1]);
    VideoCapture cap(0);
    if(!cap.isOpened()){
        cerr << "No se ha podido conectar con la cámara" << endl;
        return 0;
    }
    cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
    cv::Mat frame;
    cv::Mat HSV;
    cv::Mat threshold;
    float AngleR, DcentY, DcentX, AngleP;
    string colores = "Color.txt";
    ifstream fichColor(colores);
    if (fichColor.fail()){
        cerr << "No se pudo abrir el archivo. Terminamos" << endl;
        return 0;
    }
    int valor;
    vector<int> vColores;
    fichColor >> valor;
    while (fichColor.good()){
        vColores.push_back(valor);
        fichColor >> valor;
    }
    H_MIN = vColores[0];
    S_MIN = vColores[1];
    V_MIN = vColores[2];
    H_MAX = vColores[3];
    S_MAX = vColores[4];
    V_MAX = vColores[5];
    Serial* arduino = new Serial("\\\\.\\COM4");
    if (!arduino -> IsConnected()){
        cout << "\nNo se ha podido conectar con arduino" << endl;
        return 0;
    }
    Serial* scorbot = new Serial("\\\\.\\COM1");
    if (!scorbot -> IsConnected()){
        cout << "\nNo se ha podido conectar con el scorbot" <<
        ↪  endl;
        return 0;
    }
```

```
string punto = Punto();
Acl enviar;
enviar.sendAcl(scorbot, "defp " + punto);
enviar.sendAcl(scorbot, "here " + punto);
int Objects = 0;
int x2 = 0;
int y2 = 0;
int x = 0;
int y = 0;
timer t;
timer t2;
timer t3;
timer t4;
t.start();
bool act1 = true;
bool act2 = false;
bool colocado = false;
bool recogido = false;
int altura = 0;
float AltScorbot;
while(true)
{
    cap.read(frame);
    if (frame.empty()){
        cerr << "No se ha detectado ningúng frame" << endl;
        return 0;
    }
    cvtColor(frame, HSV, CV_BGR2HSV);
    inRange(HSV, Scalar(H_MIN,S_MIN,V_MIN),
     ↪  Scalar(H_MAX,S_MAX,V_MAX), threshold);
    ErodeDilate(threshold);
    Centro(threshold, frame, arduino, DcentX, DcentY, Objects,
     ↪  altura);
    Orientacion(threshold, frame, AngleR);
    if (Objects != 0 && !act2){
        act1 = true;
        if (!colocado){
            t.start();
            t.stop();
            unsigned long duration = t.getTime();
            if (duration >= 700){
                colocado = Colocacion(scorbot, DcentX, DcentY,
                 ↪  AngleP, altura, punto);
                t.reset();
                t.start();
            }else {
                t.start();
            }
        }
```

```
            if (colocado){
                t3.start();
                t3.stop();
                unsigned long duration = t3.getTime();
                if (duration >= 700){
                    recogido = PosRecoger(scorbot, AngleR, punto,
                     ↪  DcentX, DcentY);
                    if (recogido){
                        enviar.ShiftcX(scorbot, punto, 48);
                        enviar.ShiftcY(scorbot, punto, 8);
                        destroyAllWindows();
                        break;
                    }
                    t3.reset();
                    t3.start();
                }else {
                    t3.start();
                    }
                }

        }else if(act1){
            enviar.sendAcl(scorbot, "MOVE CASA");
            act1 = false;
            act2 = true;
            actBajada = false;
            actAltscorbot = false;
            colocado = false;
            recogido = false;
            t2.start();
        }
        if (act2){
            t2.stop();
            unsigned long duration = t2.getTime();
            if (duration >= 1350){
                enviar.sendAcl(scorbot, "here " + punto);
                t2.reset();
                act2 = false;
            }else {
                t2.start();
            }
        }
        imshow(windowName_ErDi, threshold);
        imshow(windowName_RGB, frame);

        if(waitKey(30)== 'z') return 0;

    }
    AltScorbot = enviar.getZ(scorbot);
    float MidAltObjeto = AltObjeto/2;
```

```
    float resta = (AltScorbot - MidAltObjeto) - 33;
    enviar.ShiftcZ(scorbot, punto, -resta);
    enviar.ShiftcR(scorbot, punto, 90);
    enviar.sendAcl(scorbot, "OPEN");
    Sleep(500);
    enviar.sendAcl(scorbot, "MOVE " + punto);
    Sleep(500);
    enviar.sendAcl(scorbot, "CLOSE");
    Sleep(500);

    return 0;

}
```

Anexo B Selector de figuras por color

```
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <vector>
#include <fstream>

using namespace cv;
using namespace std;

const string windowName_RGB = "RGB_view";
const string Ventana_MinMax = "Trackbars";
const string windowName_ErDi = "Imagen_erosionada_dilatada";
const string Ventana_colores = "Color";

int colorIni = 0;
int colorMax = 4;


int H_MIN = 0;
int H_MAX = 256;
int S_MIN = 0;
int S_MAX = 256;
int V_MIN = 0;
int V_MAX = 256;

const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;

void Colores(int, void*)
{
    if (colorIni == 0) {
        // Parámetros no inicializados
        H_MIN = 0;
        H_MAX = 256;
        S_MIN = 0;
        S_MAX = 256;
        V_MIN = 0;
        V_MAX = 256;
    }
    else if (colorIni == 1) {
        // rojo
        H_MIN = 0;
        H_MAX = 16;
        S_MIN = 109;
```

```
        S_MAX = 253;
        V_MIN = 34;
        V_MAX = 182;
    }
    else if (colorIni == 2){
        // Verde
        H_MIN = 38;
        H_MAX = 89;
        S_MIN = 123;
        S_MAX = 256;
        V_MIN = 54;
        V_MAX = 105;
    }
    else if (colorIni == 3){
        // azul
        H_MIN = 99;
        H_MAX = 117;
        S_MIN = 109;
        S_MAX = 236;
        V_MIN = 38;
        V_MAX = 194;
    }
    else if (colorIni == 4){
        // negro
        H_MIN = 0;
        H_MAX = 255;
        S_MIN = 0;
        S_MAX = 255;
        V_MIN = 0;
        V_MAX = 43;
    }
    else if (colorIni == 5){
        // naranja
        H_MIN = 0;
        H_MAX = 22;
        S_MIN = 154;
        S_MAX = 255;
        V_MIN = 0;
        V_MAX = 255;
    }
    else if (colorIni == 6){
        // amarillo
        H_MIN = 0;
        H_MAX = 32;
        S_MIN = 93;
        S_MAX = 253;
        V_MIN = 103;
        V_MAX = 184;
    }
```

```
    else if (colorIni == 7){
        // rosa
        H_MIN = 156;
        H_MAX = 255;
        S_MIN = 38;
        S_MAX = 255;
        V_MIN = 0;
        V_MAX = 255;
    }

}

void BarrasColor()
{
    namedWindow(Ventana_colores,1);
    createTrackbar( "Color", Ventana_colores, &colorIni, colorMax,
     ↪  Colores);
    Colores(colorIni, 0);

}

void BarrasHSV()
{
    namedWindow(Ventana_MinMax,2);
    createTrackbar( "H_MIN", Ventana_MinMax, &H_MIN, 255);
    createTrackbar( "H_MAX", Ventana_MinMax, &H_MAX, 255);
    createTrackbar( "S_MIN", Ventana_MinMax, &S_MIN, 255);
    createTrackbar( "S_MAX", Ventana_MinMax, &S_MAX, 255);
    createTrackbar( "V_MIN", Ventana_MinMax, &V_MIN, 255);
    createTrackbar( "V_MAX", Ventana_MinMax, &V_MAX, 255);

}

void ErodeDilate(cv::Mat &imgErDi)
{
    cv::Mat erodeElement = getStructuringElement( MORPH_RECT,
     ↪  Size(3,3));
    cv::Mat dilateElement = getStructuringElement( MORPH_RECT,
     ↪  Size(8,8));

    erode(imgErDi,imgErDi,erodeElement);
    erode(imgErDi,imgErDi,erodeElement);

    dilate(imgErDi,imgErDi,dilateElement);
    dilate(imgErDi,imgErDi,dilateElement);
}

int main(int argc, char *argv[])
{
```

```
    VideoCapture cap(0);
    if(!cap.isOpened()){
        cerr << "No se ha podido conectar con la cámara" << endl;
        return 0;
    }
    cap.set(CV_CAP_PROP_FRAME_WIDTH, 640);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, 480);
    string nombre = "C:\\CppProjects\\Seguimiento\\Color.txt";
    ofstream fichPixeles(nombre);
    if (fichPixeles.fail()){
        cerr << "No se pudo abrir el archivo. TERMINAMOS." <<
        ↪  std::endl;
        return 3;
    }
    vector<int> pixeles(6);
    cv::Mat img = imread("C:\\CppProjects\\Paleta1.png");
    cv::Mat frame;
    cv::Mat HSV;
    cv::Mat threshold;
    while(true)
    {
        BarrasColor();
        BarrasHSV();
        cap.read(frame);
        if (frame.empty()){
            cerr << "No se ha detectado ningun frame" << endl;
            return 0;
        }
        cvtColor(frame, HSV, CV_BGR2HSV);
        inRange(HSV, Scalar(H_MIN,S_MIN,V_MIN),
        ↪  Scalar(H_MAX,S_MAX,V_MAX), threshold);
        ErodeDilate(threshold);
        pixeles[0] = H_MIN;
        pixeles[1] = S_MIN;
        pixeles[2] = V_MIN;
        pixeles[3] = H_MAX;
        pixeles[4] = S_MAX;
        pixeles[5] = V_MAX;
        imshow(Ventana_colores, img);
        imshow(windowName_ErDi, threshold);
        imshow(windowName_RGB, frame);
        if(waitKey(30)== 'z') break;
    }
    for (size_t i = 0; i < pixeles.size(); i++){
        fichPixeles << pixeles[i] << endl;
    }
    return 0;
}
```

Anexo C Clases

C.1 Clase Serial

Cabecera

```
#ifndef SERIALCLASS_H_INCLUDED
#define SERIALCLASS_H_INCLUDED

#define ARDUINO_WAIT_TIME 2000

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

class Serial
{
    private:
        HANDLE hSerial;
        bool connected;
        COMSTAT status;
        DWORD errors;
    public:
        Serial(const char *portName);
        ~Serial();
        int ReadData(const char *buffer, unsigned int nbChar);
        bool WriteData(const char *buffer, unsigned int nbChar);
        bool IsConnected();
};

#endif
```

Código fuente

```
#include "SerialClass.hpp"

Serial::Serial(const char *portName)
{
    this->connected = false;

    this->hSerial = CreateFile(portName,
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
```

```
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

    if(this->hSerial==INVALID_HANDLE_VALUE)
    {
        if(GetLastError()==ERROR_FILE_NOT_FOUND){

            printf("ERROR: Handle was not attached. Reason: %s not
             ↪  available.\n", portName);

        }
        else
        {
            printf("ERROR!!!");
        }
    }
    else
    {
        DCB dcbSerialParams = {0};

        if (!GetCommState(this->hSerial, &dcbSerialParams))
        {
            printf("failed to get current serial parameters!");
        }
        else
        {
            dcbSerialParams.BaudRate=CBR_9600;
            dcbSerialParams.ByteSize=8;
            dcbSerialParams.StopBits=ONESTOPBIT;
            dcbSerialParams.Parity=NOPARITY;

             if(!SetCommState(hSerial, &dcbSerialParams))
             {
                printf("ALERT: Could not set Serial Port
                 ↪  parameters");
             }
             else
             {
                 this->connected = true;
                 Sleep(ARDUINO_WAIT_TIME);
             }
        }
    }

}

Serial::~Serial()
{
```

```
    if(this->connected)
    {
        this->connected = false;
        CloseHandle(this->hSerial);
    }
}

int Serial::ReadData(const char *buffer, unsigned int nbChar)
{
    DWORD bytesRead;
    unsigned int toRead;

    ClearCommError(this->hSerial, &this->errors, &this->status);

    if(this->status.cbInQue>0)
    {
        if(this->status.cbInQue>nbChar)
        {
            toRead = nbChar;
        }
        else
        {
            toRead = this->status.cbInQue;
        }

        if(ReadFile(this->hSerial, const_cast<char*>(buffer),
        ↪  toRead, &bytesRead, NULL) && bytesRead != 0)
        {
            return bytesRead;
        }

    }

    return -1;

}


bool Serial::WriteData(const char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    if(!WriteFile(this->hSerial, (void *)buffer, nbChar,
    ↪  &bytesSend, 0))
    {
        ClearCommError(this->hSerial, &this->errors,
        ↪  &this->status);

        return false;
```

```
    }
    else
        return true;
}

bool Serial::IsConnected()
{
    return this->connected;
}
```

C.2 Clase Acl

Cabecera

```
#include "SerialClass.hpp"
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <vector>

using namespace std;
class Acl
{
    public:
        Acl() = default;
        void sendAcl(Serial *obj, string str);
        void sendPosition(Serial *obj, string pto, vector<float>
        ↪  vPos);
        vector<float> getPosition(Serial *obj);
        vector<float> getPositionPoint(Serial *obj, string punto);
        float getX(Serial *obj);
        float getY(Serial *obj);
        float getZ(Serial *obj);
        float getP(Serial *obj);
        float getR(Serial *obj);
        float getPointX(Serial *obj, string punto);
        float getPointY(Serial *obj, string punto);
        float getPointZ(Serial *obj, string punto);
        float getPointP(Serial *obj, string punto);
        float getPointR(Serial *obj, string punto);
        int getAltura(Serial *obj);
        void ShiftcX(Serial *obj, string pto, float dist);
        void ShiftcY(Serial *obj, string pto, float dist);
        void ShiftcZ(Serial *obj, string pto, float dist);
```

```
        void ShiftcR(Serial *obj, string pto, float angle);
        void ShiftcP(Serial *obj, string pto, float angle);
};
```

Código fuente

```
#include "ACLclass.hpp"
#include <string>
#include <stdio.h>
#include <iostream>
#include <cmath>
#include <stdexcept>

using namespace std;

void Acl::sendAcl(Serial *obj, string str){
    string str2;
    const char *strCC;
    str2 = str + string("\r");
    strCC = str2.c_str();
    obj->WriteData(strCC, strlen(strCC));
    Sleep(500);
}

void Acl::sendPosition(Serial *obj, string pto, vector<float>
↪  vPos){
    float X, Y, Z, P, R;

    X = vPos[0];
    Y = vPos[1];
    Z = vPos[2];
    P = vPos[3];
    R = vPos[4];

    sendAcl(obj, "TEACH " + pto);
    Sleep(350);

    string strX = to_string(X);
    string strY = to_string(Y);
    string strZ = to_string(Z);
    string strP = to_string(P);
    string strR = to_string(R);

    sendAcl(obj, strX);
    Sleep(350);
    sendAcl(obj, strY);
```

```
    Sleep(350);
    sendAcl(obj, strZ);
    Sleep(350);
    sendAcl(obj, strP);
    Sleep(350);
    sendAcl(obj, strR);
    Sleep(350);
}

vector<float> Acl::getPosition(Serial *obj){
    empezar:
    char incomingData[256] = "";
        int dataLength = 255;

    sendAcl(obj, "LISTPV POSITION");

    obj->ReadData(incomingData,dataLength);;
    string str2(incomingData);
    vector<float> pos;

    size_t foundError = str2.find("*** RUN TIME ERROR ***");

    if (str2.size() != 0 && foundError == string::npos){
        size_t foundX = str2.find("X:");
        if (foundX == string::npos){
            goto empezar;
        }
        try{
            pos.push_back(stof(str2.substr(foundX+2, 7)));
        }
        catch (const std::invalid_argument& ia){
            goto empezar;
        }

        size_t foundY = str2.find("Y:");
        if (foundY == string::npos){
            goto empezar;
        }
        try{
            pos.push_back(stof(str2.substr(foundY+2, 7)));
        }
        catch (const std::invalid_argument& ia){
            goto empezar;
        }

        size_t foundZ = str2.find("Z:");
        if (foundZ == string::npos){
            goto empezar;
        }
```

```
        try{
            pos.push_back(stof(str2.substr(foundZ+2, 7)));
        }
        catch (const std::invalid_argument& ia){
            goto empezar;
        }

        size_t foundP = str2.find("P:");
        if (foundP == string::npos){
            goto empezar;
        }
        try{
            pos.push_back(stof(str2.substr(foundP+2, 7)));
        }
        catch (const std::invalid_argument& ia){
            goto empezar;
        }

        size_t foundR = str2.find("R:");
        if (foundR == string::npos){
            goto empezar;
        }
        try{
            pos.push_back(stof(str2.substr(foundR+2, 7)));
        }
        catch (const std::invalid_argument& ia){
            goto empezar;
        }

        return pos;
    }else {
        goto empezar;
    }
    return pos;
}
float Acl::getX(Serial *obj){
    vector<float> vPos = getPosition(obj);
    return vPos[0];
}

float Acl::getY(Serial *obj){
    vector<float> vPos = getPosition(obj);
    return vPos[1];
}

float Acl::getZ(Serial *obj){
    vector<float> vPos = getPosition(obj);
    return vPos[2];
}
```

```
float Acl::getP(Serial *obj){
    vector<float> vPos = getPosition(obj);
    return vPos[3];
}

float Acl::getR(Serial *obj){
    vector<float> vPos = getPosition(obj);
    return vPos[4];
}

int Acl::getAltura(Serial *obj){
    char incomingData[256] = "";
        int dataLength = 255;

    obj->ReadData(incomingData,dataLength);
    int altura;
    altura = atoi(incomingData);

    return altura;
}

void Acl::ShiftcX(Serial *obj, string pto, float dist){
    string DistStr = to_string(dist);
    sendAcl(obj, "SHIFTC " + pto + " BY X " + DistStr);
    Sleep(350);
}

void Acl::ShiftcY(Serial *obj, string pto, float dist){
    string DistStr = to_string(dist);
    sendAcl(obj, "SHIFTC " + pto + " BY Y " + DistStr);
    Sleep(350);
}

void Acl::ShiftcR(Serial *obj, string pto, float angle){
    string AngleStr = to_string(angle);
    sendAcl(obj, "SHIFTC " + pto + " BY R " + AngleStr);
    Sleep(350);
}

void Acl::ShiftcZ(Serial *obj, string pto, float dist){
    string DistStr = to_string(dist);
    sendAcl(obj, "SHIFTC " + pto + " BY Z " + DistStr);
    Sleep(350);
}

void Acl::ShiftcP(Serial *obj, string pto, float angle){
    string AngleStr = to_string(angle);
    sendAcl(obj, "SHIFTC " + pto + " BY P " + AngleStr);
```

```
    Sleep(350);
}

vector<float> Acl::getPositionPoint(Serial *obj, string punto){
    char incomingData[256] = "";
        int dataLength = 255;

    sendAcl(obj, "LISTPV " + punto);
    obj->ReadData(incomingData,dataLength);;
    string str2(incomingData);

    vector<float> pos;
    size_t foundError = str2.find("*** RUN TIME ERROR ***");

    if (str2.size() != 0 && foundError == string::npos){
        size_t foundX = str2.find("X:");
        if (foundX == string::npos){
            cerr << "No se ha encontrado el valor de X en
             ↪  getPositionPoint" << endl;
            exit(0);
        }
        pos.push_back(stof(str2.substr(foundX+2, 7)));

        size_t foundY = str2.find("Y:");
        if (foundY == string::npos){
            cerr << "No se ha encontrado el valor de Y en
             ↪  getPositionPoint" << endl;
            exception();
        }
        pos.push_back(stof(str2.substr(foundY+2, 7)));

        size_t foundZ = str2.find("Z:");
        if (foundZ == string::npos){
            cerr << "No se ha encontrado el valor de Z en
             ↪  getPositionPoint" << endl;
            exception();
        }
        pos.push_back(stof(str2.substr(foundZ+2, 7)));

        size_t foundP = str2.find("P:");
        if (foundP == string::npos){
            cerr << "No se ha encontrado el valor de P en
             ↪  getPositionPoint" << endl;
            exception();
        }
        pos.push_back(stof(str2.substr(foundP+2, 7)));

        size_t foundR = str2.find("R:");
        if (foundR == string::npos){
```

```
            cerr << "No se ha encontrado el valor de R en
            ↪  getPositionPoint" << endl;
            exception();
        }
        pos.push_back(stof(str2.substr(foundR+2, 7)));

        return pos;
    }else {
        cerr << "No se ha recibido ning" << char(250) << "n
        ↪  parámetro o la posicion no es alcanzable" << endl;
        exit(1);
    }
    return pos;
}

float Acl::getPointX(Serial *obj, string punto){
    vector<float> vPos = getPositionPoint(obj, punto);
    return vPos[0];
}

float Acl::getPointY(Serial *obj, string punto){
    vector<float> vPos = getPositionPoint(obj, punto);
    return vPos[1];
}

float Acl::getPointZ(Serial *obj, string punto){
    vector<float> vPos = getPositionPoint(obj, punto);
    return vPos[2];
}

float Acl::getPointP(Serial *obj, string punto){
    vector<float> vPos = getPositionPoint(obj, punto);
    return vPos[3];
}

float Acl::getPointR(Serial *obj, string punto){
    vector<float> vPos = getPositionPoint(obj, punto);
    return vPos[4];
}
```

C.3 Clase Conv

Cabecera

```
#include <iostream>
#include <armadillo>
#include <iomanip>
#include <math.h>

#define PI (4*atan(1))

using namespace std;
using namespace arma;

class Conv
{
    private:
        mat cinemadirect(mat D, mat TITA, mat A, mat ALFA, vec
         ↪  Dist);
        mat Cip(float x, float y, float z, vec P, int d1, int d2,
         ↪  int l1, int l2, int l3, bool codo);
        mat matrizT(float d, float tita, float a, float alfa);
    public:
        Conv() = default;
        vector<float> Trans(vec P, vec DistCam);
};
```

Código fuente

```
#include "ConvCoordCam.hpp"
#include <iostream>
#include <armadillo>
#include <iomanip>
#include <math.h>

#define PI (4*atan(1))

using namespace std;
using namespace arma;

mat Conv::matrizT(float d, float tita, float a, float alfa){
    mat T;
    T = { {cos(tita), -cos(alfa)*sin(tita), sin(alfa)*sin(tita),
     ↪  a*cos(tita)},
```

```
          {sin(tita),  cos(alfa)*cos(tita), -sin(alfa)*cos(tita),
           ↪  a*sin(tita)},
          {0         ,  sin(alfa)          ,  cos(alfa)          , d
           ↪  },
          {0         ,  0                  ,  0                  , 1
           ↪  } };

    return T;
}

mat Conv::Cip(float x, float y, float z, vec P, int d1, int d2, int
 ↪  l1, int l2, int l3, bool codo){
    float ti1 = 0;
    float ti2 = 0;
    float ti3 = 0;
    float dist0 = 0;
    mat m1, m2, coff;
    float arg_ti3, tita, alfa;
    vec vectorNorm;
    mat ti13;

    if (x == 0 && y == 0){
        cerr << "Error: configuracion singular." << endl;
        return 0;
    }else {
        if (y == 0){
            ti1 = PI/2;
        }else {
            dist0 = sqrt( pow(P(0), 2) + pow(P(1), 2) + pow((P(2) -
             ↪  l1 ), 2) );
            ti1 = atan( P(1) / P(0) ) + asin( d2 / ( dist0 * cos(
             ↪  asin( ( P(2) - l1 ) / dist0 ) ) ) );
        }

        m1 = matrizT(l1,ti1,0,PI/2);
        m2=matrizT(d2,0,d1,0);
        vectorNorm = { 0, 0, 0, 1 };
        coff=m1*m2*vectorNorm;
        arg_ti3 = ( pow((x-coff(0)),2) + pow((y-coff(1)),2) +
         ↪  pow((z-l1),2) - pow(l2,2) - pow(l3,2) ) / ( 2*l2*l3 );
        arg_ti3 = (round(arg_ti3*10000))/10000;

        if (abs(arg_ti3) > 1){
            cerr << "Error: Punto no alcanzable." << endl;
            return 0;
        }else {
            if (codo == true){
                ti3 = -acos(arg_ti3); //Codo arriba
            }else{
```

```
                ti3 = acos(arg_ti3);  //Codo abajo
            }
            tita = atan((z-l1)/(sqrt(pow((x-coff(0)),2) +
             ↪  pow((y-coff(1)),2))));
            alfa = atan((l3*sin(ti3))/(l2+l3*cos(ti3)));
            ti2 = tita - alfa;
        }
    }
    ti13 = { ti1, ti2, ti3 };

    return ti13;
}

mat Conv::cinemadirect(mat D, mat TITA, mat A, mat ALFA, vec
 ↪  DistCam){
    cube vT(zeros(4,4,D.size()));

    for (int i = 0; i <= D.size()-1; i++){
        vT.slice(i) = matrizT(D(i),TITA(i),A(i),ALFA(i));
    }

    mat TC = {  {1, 0, 0, 8},
                {0, 1, 0,  52},
                {0, 0, 1, -25 },
                {0, 0, 0,  1 } };

    vec OrigenCam = { 0, 0, 0, 1 };
    mat CentCam =
     ↪  vT.slice(0)*vT.slice(1)*vT.slice(2)*vT.slice(3)*vT.slice(4)
     ↪  *vT.slice(5) * TC * OrigenCam;
    mat PuntoCam =
     ↪  vT.slice(0)*vT.slice(1)*vT.slice(2)*vT.slice(3)*vT.slice(4)
     ↪  *vT.slice(5) * TC * DistCam;
    mat Dist = CentCam - PuntoCam;

    return Dist;

}

vector<float> Conv::Trans(vec P, vec DistCam){
    double d1 = 70;
    double d2 = 42;
    double l1 = 390;
    double l2 = 280;
    double l3 = 230;
    double lm = 224;
    //double lp = 100;
    bool codo = true;
    mat D1, TITA1, A1, ALFA1, D2, TITA2, A2, ALFA2;
```

```
mat F, F1, F2, F3, F4;
mat R, U;
mat ti4, ti5;
mat TI;

vec a = { 0, 0, -1 };
vec S = { 0, 1, 0 };

a=a/norm(a);
S=S/norm(S);

vec N = cross(a,S);

vec O3 = P - lm*a;

mat ti13 =Cip(O3(0),O3(1),O3(2),P,d1,d2,l1,l2,l3,codo);


D1    = { l1,      d2, 0,       0       };
TITA1 = { ti13(0), 0,  ti13(1), ti13(2) };
A1    = { 0,       d1, l2,      l3      };
ALFA1 = { PI/2,    0,  0,       0       };

cube vT(zeros(4,4,4));

for ( int i = 0; i < 4; i++){
    vT.slice(i) = matrizT(D1(i),TITA1(i),A1(i),ALFA1(i));
}

F1 = vT(span(0,2),span(0,2),span(0,0));
F2 = vT(span(0,2),span(0,2),span(1,1));
F3 = vT(span(0,2),span(0,2),span(2,2));
F4 = vT(span(0,2),span(0,2),span(3,3));
F = F1 * F2 * F3 * F4;
R = join_horiz(S, N);
R = join_horiz(R, a);
U = F.t() * R;
ti4 = acos(-U(1,2)) * sign(U(0,2));
ti5 = acos(U(2,1)) * sign(U(2,0));

TI = join_horiz(ti13, ti4);
TI = join_horiz(TI, ti5);

D2    = { l1,     d2, 0,     0,     0,     lm    };
TITA2 = { TI(0),  0,  TI(1), TI(2), TI(3), TI(4) };
A2    = { 0,      d1, l2,    l3,    0,     0     };
ALFA2 = { PI/2,   0,  0,     0,     PI/2,  0     };

mat Dist = cinemadirect(D2, TITA2, A2, ALFA2, DistCam);
```

```
    vector<float> Distancia;

    for (int i = 0; i < Dist.size(); i++){
        Distancia.push_back(Dist(i));
    }
    return Distancia;

}
```

C.4 Clase timer

Cabecera

```
#include <iostream>
#include <conio.h>
#include <time.h>

using namespace std;

class timer {
    public:
        timer();
        void           start();
        void           stop();
        void           reset();
        bool           isRunning();
        unsigned long  getTime();
        bool           isOver(unsigned long seconds);
    private:
        bool           resetted;
        bool           running;
        unsigned long  beg;
        unsigned long  end;
};
```

Código fuente

```
#include "CronoClass.hpp"

timer::timer() {
    resetted = true;
    running = false;
    beg = 0;
    end = 0;
}

void timer::start() {
    if(! running) {
        if(resetted)
            beg = (unsigned long) clock();
        else
            beg -= end - (unsigned long) clock();
        running = true;
        resetted = false;
    }
}

void timer::stop() {
    if(running) {
        end = (unsigned long) clock();
        running = false;
    }
}

void timer::reset() {
    bool wereRunning = running;
    if(wereRunning)
        stop();
    resetted = true;
    beg = 0;
    end = 0;
    if(wereRunning)
        start();
}

bool timer::isRunning() {
    return running;
}

unsigned long timer::getTime() {
    if(running)
        return ((unsigned long) clock() - beg) / CLOCKS_PER_SEC;
    else
```

```
        return end - beg;
}

bool timer::isOver(unsigned long seconds) {
    return seconds >= getTime();
}
```

Anexo D Datasheet Sharp GP2Y0A21YK0F

GP2Y0A21YK0F

Distance Measuring Sensor Unit
Measuring distance: 10 to 80 cm
Analog output type

■Description

GP2Y0A21YK0F is a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector) , IRED (infrared emitting diode) and signal processing circuit.
The variety of the reflectivity of the object, the environmental temperature and the operating duration are not influenced easily to the distance detection because of adopting the triangulation method.
This device outputs the voltage corresponding to the detection distance. So this sensor can also be used as a proximity sensor.

■Features

1. Distance measuring range : 10 to 80 cm
2. Analog output type
3. Package size : 29.5×13×13.5 mm
4. Consumption current : Typ. 30 mA
5. Supply voltage : 4.5 to 5.5 V

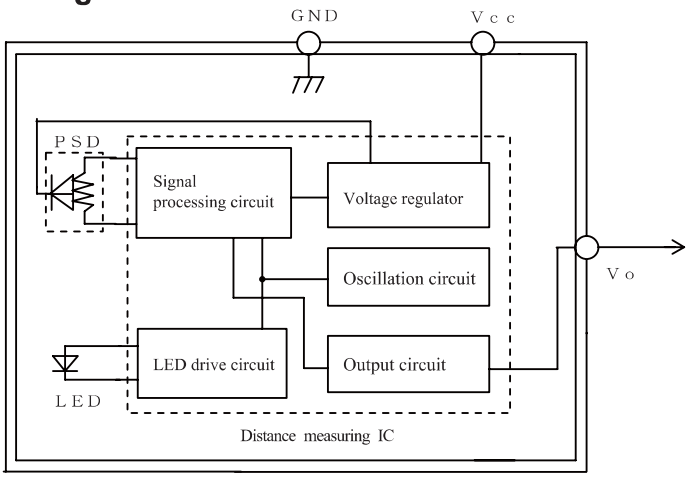
■Agency approvals/Compliance

1. Compliant with RoHS directive (2011/65/EU)

■Applications

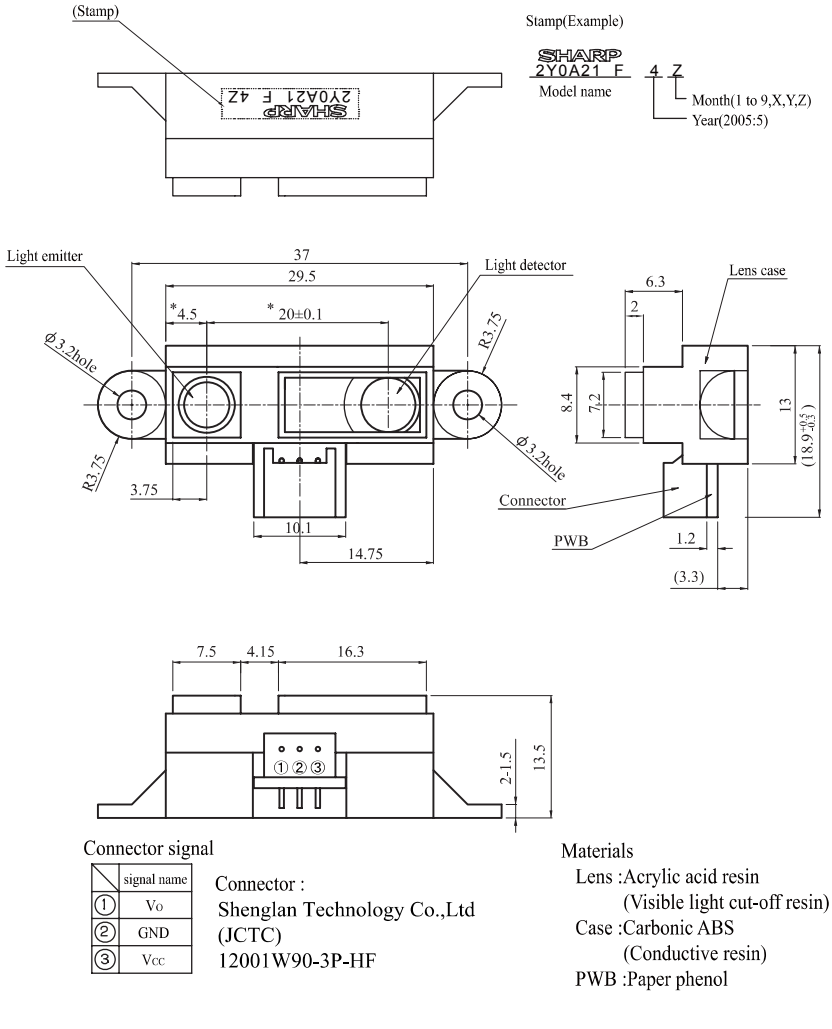
1. Touch-less switch
   (Sanitary equipment, Control of illumination, etc.)
2. Robot cleaner
3. Sensor for energy saving
   (ATM, Copier, Vending machine)
4. Amusement equipment
   (Robot, Arcade game machine)

■Block diagram



■Outline Dimensions

(Unit : mm)



Connector signal

	signal name
①	Vo
②	GND
③	Vcc

Connector :
Shenglan Technology Co.,Ltd
(JCTC)
12001W90-3P-HF

Materials
Lens :Acrylic acid resin
(Visible light cut-off resin)
Case :Carbonic ABS
(Conductive resin)
PWB :Paper phenol

Note 1. The dimensions marked * are described the dimensions of lens center position.
Note 2. Unspecified tolerances shall be ± 0.3 mm.
Note 3. The dimensions in parenthesis are shown for reference.

Product mass : Approx. 3.6g

■Absolute Maximum Ratings

(T_a=25℃,V_{CC}=5V)

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	-0.3 to +7	V
Output terminal voltage	V_O	-0.3 to V_{CC}+0.3	V
Operating temperature	T_{opr}	-10 to +60	℃
Storage temperature	T_{stg}	-40 to +70	℃

■Electro-optical Characteristics

(T_a=25℃,V_{CC}=5V)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Average supply current	I_{CC}	L=80cm (Note 1)	—	30	40	mA
Distance measuring	ΔL	(Note 1)	10	—	80	cm
Output voltage	V_O	L=80cm (Note 1)	0.25	0.4	0.55	V
Output voltage differential	ΔV_O	Output voltage differece between L=10cm and L=80cm (Note 1)	1.65	1.9	2.15	V

* L : Distance to reflective object

Note 1 : Using reflective object : White paper (Made by Kodak Co., Ltd. gray cards R-27・white face, reflectance; 90%)

■Recommended operating conditions

Parameter	Symbol	Rating	Unit
Supply voltage	V_{CC}	4.5 to 5.5	V

Fig. 1 Timing chart

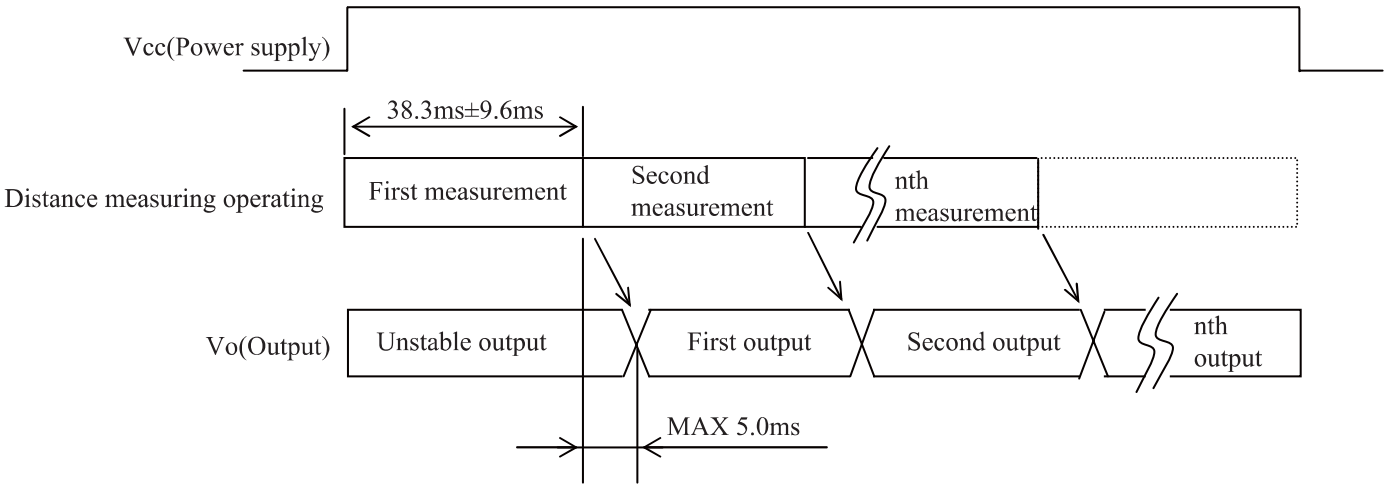
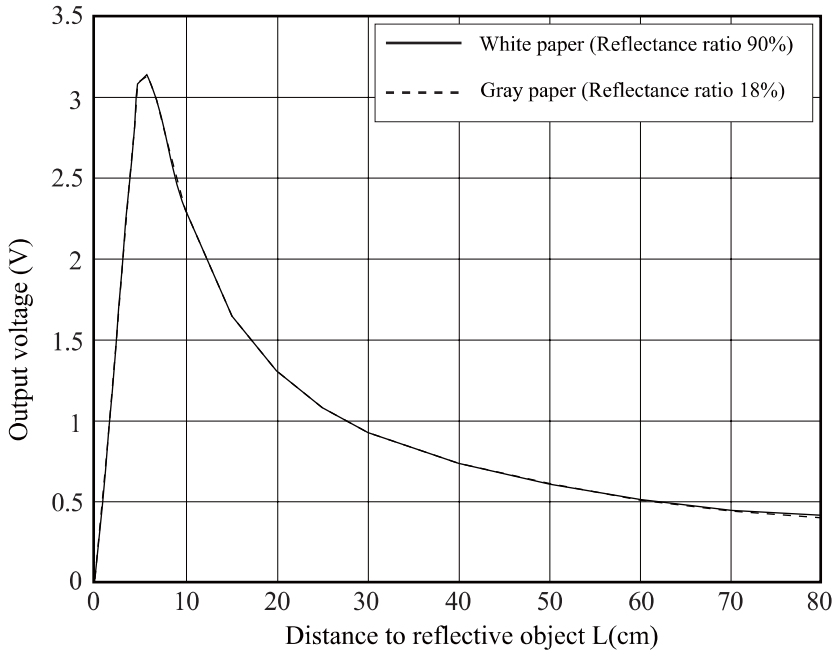
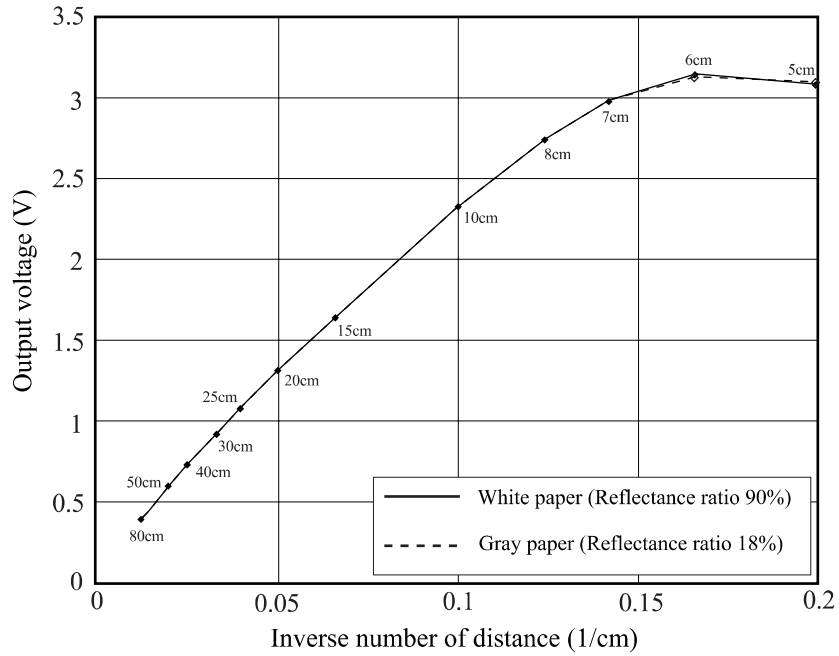


Fig. 2 Example of distance measuring characteristics(output)





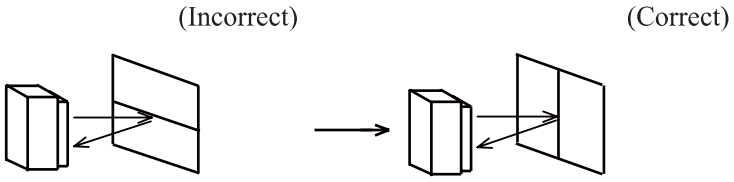
■Notes

●Advice for the optics

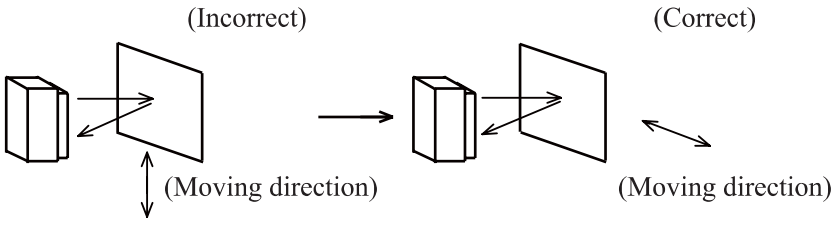
• The lens of this device needs to be kept clean. There are cases that dust, water or oil and so on deteriorate the characteristics of this device. Please consider in actual application.
• Please don't do washing. Washing may deteriorate the characteristics of optical system and so on. Please confirm resistance to chemicals under the actual usage since this product has not been designed against washing.

●Advice for the characteristics

• In case that an optical filter is set in front of the emitter and detector portion, the optical filter which has the most efficient transmittance at the emitting wavelength range of LED for this product ($\lambda = 870 \pm 70$nm), shall be recommended to use. Both faces of the filter should be mirror polishing. Also, as there are cases that the characteristics may not be satisfied according to the distance between the protection cover and this product or the thickness of the protection cover, please use this product after confirming the operation sufficiently in actual application.
• In case that there is an object near to emitter side of the sensor between sensor and a detecting object, please use this device after confirming sufficiently that the characteristics of this sensor do not change by the object.
• When the detector is exposed to the direct light from the sun, tungsten lamp and so on, there are cases that it can not measure the distance exactly. Please consider the design that the detector is not exposed to the direct light from such light source.
• Distance to a mirror reflector can not be sometimes measured exactly. In case of changing the mounting angle of this product, it may measure the distance exactly.
• In case that reflective object has boundary line which material or color etc. are excessively different, in order to decrease deviation of measuring distance, it shall be recommended to set the sensor that the direction of boundary line and the line between emitter center and detector center are in parallel.



• In order to decrease deviation of measuring distance by moving direction of the reflective object, it shall be recommended to set the sensor that the moving direction of the object and the line between emitter center and detector center are vertical.



●Advice for the power supply

• In order to stabilize power supply line, we recommend to insert a by-pass capacitor of 10μF or more between Vcc and GND near this product.

●Notes on handling

• There are some possibilities that the internal components in the sensor may be exposed to the excessive mechanical stress. Please be careful not to cause any excessive pressure on the sensor package and also on the PCB while assembling this product.

● Presence of ODC etc.

This product shall not contain the following materials.
And they are not used in the production process for this product.
Regulation substances : CFCs, Halon, Carbon tetrachloride, 1.1.1-Trichloroethane (Methylchloroform)

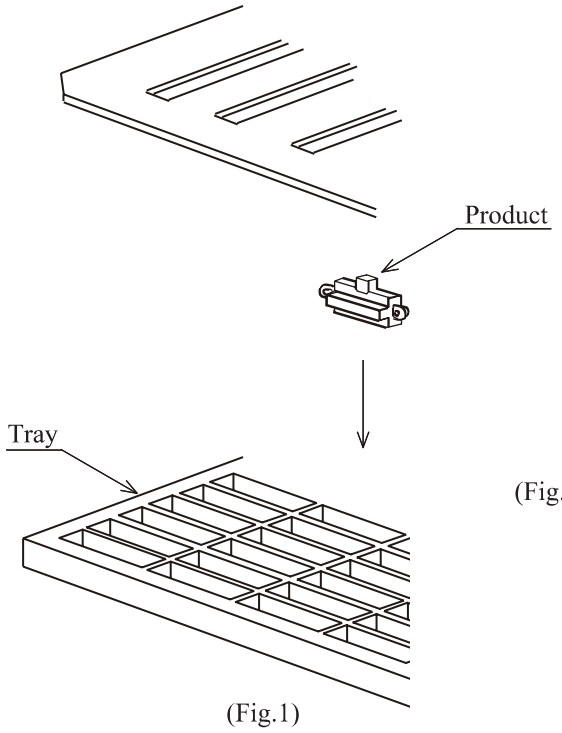
Specific brominated flame retardants such as the PBB and PBDE are not used in this product at all.

This product shall not contain the following materials banned in the RoHS Directive (2011/65/EU).

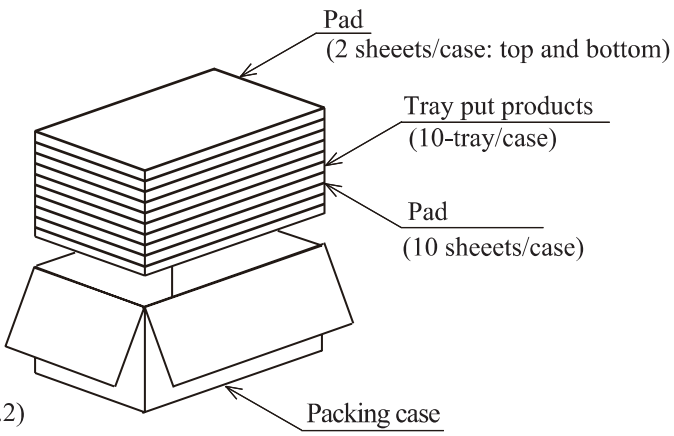
- Lead, Mercury, Cadmium, Hexavalent chromium, Polybrominated biphenyls (PBB), Polybrominated diphenyl ethers (PBDE).

■Package specification

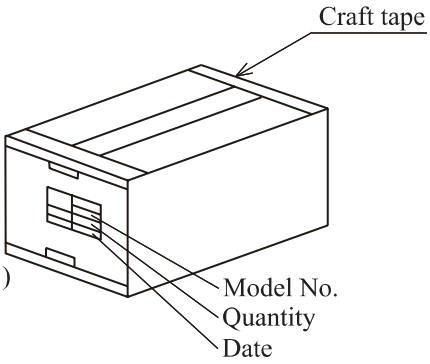
Package composition



(Fig.1)



(Fig.2)



(Fig.3)

Packaging method

1.Put products of 100pcs. in tray. packing method is showed in the above fig.(Fig.1)
2.Put them(10-tray) in the packing box. Put pads on their top and bottom.
And put pads on each trays(Total 10 sheets) (Fig.2).
3.Seal the packing box with craft tape.
Print the model No.,quantity,inspection date (1000 pcs./a packing box)(Fig.3).

■Important Notices

· The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property rights. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP's devices.

· Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structure, and other contents described herein at any time without notice in order to improve design or reliability. Manufacturing locations are also subject to change without notice.

· Observe the following points when using any devices in this publication. SHARP takes no responsibility for damage caused by improper use of the devices which does not meet the conditions and absolute maximum ratings to be used specified in the relevant specification sheet nor meet the following conditions:

(i) The devices in this publication are designed for use in general electronic equipment designs such as:
- --- Personal computers
- --- Office automation equipment
- --- Telecommunication equipment [terminal]
- --- Test and measurement equipment
- --- Industrial control
- --- Audio visual equipment
- --- Consumer electronics

(ii) Measures such as fail-safe function and redundant design should be taken to ensure reliability and safety when SHARP devices are used for or in connection with equipment that requires higher reliability such as:
- --- Transportation control and safety equipment (i.e., aircraft, trains, automobiles, etc.)
- --- Traffic signals
- --- Gas leakage sensor breakers
- --- Alarm equipment
- --- Various safety devices, etc.

(iii) SHARP devices shall not be used for or in connection with equipment that requires an extremely high level of reliability and safety such as:
- --- Space applications
- --- Telecommunication equipment [trunk lines]
- --- Nuclear power control equipment
- --- Medical and other life support equipment (e.g., scuba).

· If the SHARP devices listed in this publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Law of Japan, it is necessary to obtain approval to export such SHARP devices.

· This publication is the proprietary product of SHARP and is copyrighted, with all rights reserved. Under the copyright laws, no part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, in whole or in part, without the express written permission of SHARP. Express written permission is also required before any use of this publication may be made by a third party.

· Contact and consult with a SHARP representative if there are any questions about the contents of this publication.

Anexo E WebCam C270 HD Logitech

ESPECIFICACIONES

sobre el producto:		
Garantía / Autoayuda	En la sección de asistencia al producto, encontrará información sobre la duración de la garantía y respuestas a preguntas habituales.	
	Requisito básico	Requisito HD
Requisitos del sistema	1 GHz = CPU mínima 1,6 GHz = CPU recomendada 1 GB = RAM mínima 2 GB = RAM recomendada	Intel Core 2 Duo a 2,4 GHz = CPU recomendada 2 GB de RAM
Software compatible (en la fecha de lanzamiento)	Logitech Webcam Software 2.0 (LWS) (NOTA: En el sitio Web encontrará la versión más reciente del software)	
Sistemas operativos compatibles (en la fecha de lanzamiento)	Windows XP, Windows Vista, Windows 7	

Especificaciones de la cámara:	
Imágenes disponibles	[Imagen de lado izquierdo]
Tipo de conexión	USB con cable
Tipo de USB	USB 2.0 de alta velocidad
USB VID_PID	VID_046D&PID_081A
Micrófono	Integrado, supresión de ruido
Objetivo y tipo de sensor	Plástico
Tipo de enfoque	Fijo
Campo visual	60°
Longitud focal	4 mm
Resolución óptica (real)	1280 x 960 1,2 MP
Captura de imágenes (SD 4:3)	320 x 240, 640 x 480 1,2 MP, 3 MP
Captura de imágenes (W 16:9)	360p, 480p, 720p
Captura de vídeo (SD 4:3)	320 x 240, 640 x 480, 800 x 600
Captura de vídeo (W 16:9)	360p, 480p, 720p,
Frecuencia de cuadro (máx.)	30 fps: 640 x 480
Efectos de vídeo	N/A
RightLight	RightLight 2
Botones	Otros, no aplicable
Indicadores luminosos (LED)	Actividad/encendido
Tapa de privacidad	No
Tamaño de clip (máx.)	0 a infinito
Longitud del cable	1,5 metros (5 ft)

Hardware adicional en paquete:	
Auriculares con micrófono	No se incluye
Soporte de sobremesa	No se incluye