Universidad Autónoma Metropolitana Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Tecnológico.

Robot móvil que reconoce y busca objetos en base a su color y forma.

Efraín Ernesto Arévalo Vázquez, 206200579

Asesor: M. en C. Arturo Zúñiga López

22 de abril de 2015.

Yo, M. en C. Arturo Zúñiga López, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: 

Yo, Efraín Ernesto Arévalo Vázquez, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma: 

Resumen

En éste proyecto se describe el diseño e implementación de un sistema embebido que permite controlar un robot móvil que analice por color y forma un objeto mediante una cámara digital, se dé la vuelta y avance mientras identifica diferentes objetos con colores y formas distintas hasta detenerse enfrente del objeto buscado. En todo momento se puede monitorear parte del procesamiento de imágenes a través de una computadora por medio de una conexión inalámbrica.

Dentro del procesamiento de imagens realizadas en este trabajo, la primera etapa consiste en obtener el color predominante del objeto, posteriormente quitar el fondo para obtener una imagen sólida que representa la silueta o forma del objeto. De este análisis se obtiene información relevante como el área que ocupa dentro del campo de visión del robot, su centro de masa, etc. Posteriormente se aplican los momentos invariantes de *Hu*, obtenemos información que no cambia con la rotación del objeto ni con la escalabilidad, y con ello podemos identificar y encontrar objetos parecidos, ya que con ésta información no existen dos valores iguales para dos formas diferentes, y por lo tanto el resultado obtenido será confiable.

Los resultados obtenidos determinan que el sistema es capaz de reconocer objetos de color sólido y de formas regulares en tres tipos de iluminación: luz directa del sol, luz de día dentro de interiores y luz artificial. También se muestra que el procesamiento de imágenes ofrece un buen control del sistema de movimiento de un robot ofreciendo un buen grado de autonomía.

Este sistema aprovecho los recursos ofrecidos del sistema embebido logrando crear un dispositivo móvil capaz de reconocer objetos de manera eficiente y funcionando en diferentes condiciones de iluminación. Todo en un sistema que se puede adaptar a otras arquitecturas de procesamiento y diferentes tipos de robots móviles.

Tabla de Contenido

Índice de Figuras

1.- Introducción

Los sistemas de visión por computadora han evolucionado poco a poco a medida que la tecnología ha evolucionado y permitido la creación de cámaras, computadoras y algoritmos de procesamiento más potentes. De forma básica un sistema de visión por computadora está conformado por varios subsistemas capaces de realizar dos funciones fundamentales: captar la información de la escena real mediante la proyección en una imagen y analizar las imágenes para extraer la información que contienen [1].

El mayor peso del análisis de las imágenes captadas se encuentra a nivel de software, lo que proporciona una mayor flexibilidad en cuanto a lo que se puede implementar en base a los resultados obtenidos.

En éste proyecto se pretende realizar el diseño e implementación de un sistema embebido que permita controlar un robot móvil que analice por color y forma un objeto muestra mediante una cámara digital, se dé la vuelta y avance mientras identifica diferentes objetos con colores y formas distintas hasta detenerse enfrente del objeto buscado. En todo momento se podrá monitorear parte del procesamiento de imágenes a través de una computadora por medio de una conexión inalámbrica.

2.- Antecedentes

A continuación se muestran los proyectos que presentan alguna relación con el sistema que se pretende diseñar e implementar en esta propuesta:

2.1.- Trabajos Relacionados

- "Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot" [2]. En este proyecto, un brazo robot permite seleccionar objetos (tornillos, tuercas, llaveros, etc.) que se encuentran en una mesa, independiente de la posición y orientación. Implementa técnicas de aprendizaje y clasificación automática por medio de una cámara fija. La diferencia radica en que tanto la cámara como el brazo están fijos a una mesa. Aunque probablemente en un trabajo futuro se le podría añadir un brazo a nuestro diseño y que pueda tomar objetos.

- "Reconocimiento y localización de instrumental médico usando análisis automatizado de imágenes" [3]. En este artículo se propone un sistema que reconozca instrumentos de medicina para que un brazo robot pueda tomar el que se necesita. La diferencia principal es que las características los objetos se tienen que cargar anticipadamente en un a base de datos. En nuestro proyecto cada objeto a buscar se aprende en tiempo de ejecución.

- Estudio de los comportamientos de esquema de motores Lego [4]. Uno de los objetivos de este proyecto es equipar a los robots con algoritmos autómatas suficientes que le permiten ir de un punto a otro, evitando colisiones con obstáculos estáticos y dinámicos, evitando otros robots y asegurar una interacción ambiental correcta. Su comportamiento es producido por la interacción con el medio ambiente y la capacidad de asimilación. Dentro de se crean comportamientos recurrentes (esquemas).

- *Emulation System for a Distribution Center Using Mobile Robot, Controlled by Artificial Vision and Fuzzy Logic* [5]. El robot sigue una trayectoria libre que está controlado por visión por computadora y los módulos de la lógica difusa. Sistema de visión *Theartificial* incluye una cámara web situada en la parte superior del centro de distribución y se utiliza para calcular la ubicación del robot. La imagen técnica de segmentación del color rojo se implementó en el sistema de visión por computadora para determinar la posición y la orientación del robot. La información obtenida de la cámara también se utiliza por el controlador difuso para estimar la velocidad del robot móvil de manera inalámbrica. El

procesamiento de imágenes y control difuso se implementan en una computadora utilizando *Matlab*[1].

- Desarrollo de una visión artificial basada en un sistema de navegación para helicópteros modificados usando NDGPS/INS integrado [6]. Se presentan los resultados de desarrollo de un sistema de navegación NDGPS/INS basado en visión artificial. El sistema NDGPS/INS propuesto fue probado en el entorno dinámico de un vehículo de tierra y en vuelo. Los resultados han mostrado las ventajas significativas en la precisión de la posición y conocimiento de la situación. Se cumple la aproximación de precisión CAT-I y el aterrizaje utilizando integraciones NDGPS/INS. Sería útil cuando tenemos que operar de noche, en condiciones de mal tiempo y el propósito militar usando por pantalla la guía de navegación en 3D basado en la imagen artificial.

- An image matching evolutionary algorithm based on Hu invariant moments [7]. Es un juego algorítmico evolutivo basado en Momentos Hu. En primer lugar, se ha inicializado la población. Un grupo de subgrafos buscando se construye. En segundo lugar, la función de aptitud basado en Hu momentos invariantes está diseñado. Se calculan los siete Hu momentos invariantes de la imagen de la plantilla y el subgrafo buscado. La distancia euclidiana de Hu momentos invariantes se utiliza para medir la similitud entre la imagen plantilla y el subgrafo buscado. La imagen de la plantilla y el subgrafo buscado coinciden sí éstas distancias euclidianas son inferiores al umbral establecido. Por último, un nuevo subgrafo buscado se construye por medio de una nueva estrategia evolutiva. El nuevo subgrafo buscado reemplaza el subgrafo buscado cuyo valor de la función de aptitud es máxima. Los resultados experimentales demuestran la gran robustez y eficiencia del algoritmo IMEA.

- *Spatial color histograms for content-based image retrieval* [8]. El histograma de color es una técnica importante para una imagen en color de indexación y recuperación. En este trabajo, el histograma color tradicional es modificado para capturar la información de diseño espacial de cada color, y se introducen tres tipos de histogramas de color espaciales: anulares, histogramas de color angulares e híbridos. Los experimentos muestran que, con un equilibrio adecuado entre la granularidad en las dimensiones de color y espaciales, estos histogramas superan tanto el histograma de color tradicional y algunos de histograma existente refinamientos tales como el vector de color coherente.

[1] http://www.mathworks.com/products/matlab/

3.- Justificación

En la actualidad los avances de la tecnología nos permiten trabajar con sistemas embebidos que se acercan mucho al poder de procesamiento de una computadora personal. Esto permite realizar tareas más complejas en sistemas móviles y además se pueden utilizar en diferentes entornos y no únicamente en lugares cerrados.

Este proyecto pretende realizar el reconocimiento de objetos sin alterar o controlar su entorno, dependiendo solo de las condiciones actuales del objeto. La relevancia radica en que el control del desplazamiento del robot depende del procesamiento de las imágenes obtenidas de la cámara para interpretar si necesita avanzar o no en base a los resultados obtenidos. De esta manera el control se hace por software y no por señales recibidas de sensores electrónicos.

El robot en conjunto carecerá de sensores electrónicos para detectar su ambiente (como sensores de proximidad, acelerómetro, etc.), así como tampoco recibirá instrucciones de ningún usuario durante su funcionamiento. Únicamente utilizará el procesamiento de imágenes que capture con la cámara para recibir la imagen del objeto a buscar y controlar su movimiento para identificar y acercarse a él una vez que lo localice.

4.- Objetivo General

Diseñar e implementar un robot móvil que reconozca, busque y localice objetos de forma regular y de color sólido mediante el procesamiento de imágenes en tiempo real.

4.1.- Objetivos Específicos

- Diseñar e implementar la detección de la marca para enseñarle el objeto a buscar.
- Implementar los algoritmos para el reconocimiento y búsqueda de un objeto por color y forma.
- Diseñare implementar el sistema de control del desplazamiento del robot móvil.
- Realizar pruebas de los componentes del sistema de reconocimiento y búsqueda antes de implementarlos al sistema embebido.
- Implementar el sistema operativo y los componentes del sistema de reconocimiento y búsqueda en el sistema embebido, y además configurar los periféricos.
- Diseñar y configurar el sistema de monitoreo en el sistema embebido.
- Diseñar y ensamblar el robot móvil.
- Verificar que el sistema cumpla con los requerimientos mínimos establecidos en el diseño.

5.- Marco Teórico

Éste proyecto tiene la finalidad de realizar procesamiento de imágenes y desplazarse de forma autónoma, por lo cual es de importancia considerar los elementos utilizados; lo cuales fundamentan y sustentan el proyecto. Se muestran los conceptos utilizados, sus variables y sus antecedentes.

5.1.- Imagen

Una imagen [9] es una representación visual, que manifiesta la apariencia visual de un objeto real o imaginario.

5.1.1.- Imagen digital.

Una imagen digital es una representación bidimensional de una imagen a partir de una matriz numérica. Dependiendo de si la resolución de la imagen es estática o dinámica, puede tratarse de una imagen matricial (o mapa de bits) o de un gráfico vectorial.

Una imagen en mapa de bits es una estructura o fichero de datos que representa una rejilla rectangular de píxeles o puntos de color, denominada matriz, que se puede visualizar en un monitor, papel u otro dispositivo de representación. A las imágenes en mapa de bits se las suele definir por su altura y anchura (en píxeles) y por su profundidad de color (en bits por píxel), que determina el número de colores distintos que se pueden almacenar en cada punto individual, y por lo tanto, en gran medida, la calidad del color de la imagen.

El formato de imagen matricial está ampliamente extendido y es el que se suele emplear para tomar fotografías digitales y realizar capturas de vídeo. Para su obtención se usan dispositivos de conversión analógica-digital, tales como escáneres y cámaras digitales [10].

5.1.2.- Cámaras web

Una cámara web es una pequeña cámara digital conectada a una computadora la cual puede capturar imágenes y transmitirlas a través de Internet [11].

5.1.3.- Color

Cada punto representado en la imagen debe contener información de color[12], representada en canales separados que representan los componentes primarios del color que se pretende representar, en cualquier modelo de color, bien sea RGB, HSV, CMYK, LAB o cualquier otro disponible para su representación.

5.1.3.1.- Modelo de color RGB

RGB es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios. El modelo de color RGB no define por sí mismo lo que significa exactamente rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en diferentes dispositivos que usen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente [12].

5.1.3.2.- Modelo de color HSV

El modelo HSV [12], también llamado HSB (Matiz, Saturación, Brillo), define un modelo de color en términos de sus componentes. Se trata de una transformación no lineal del espacio de color RGB, y se puede usar en progresiones de color.

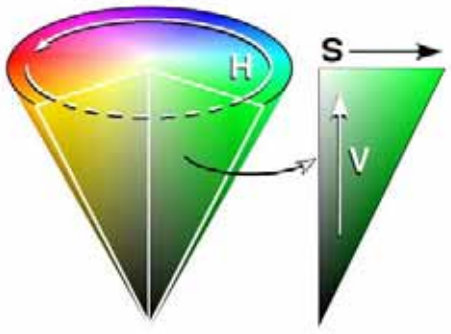

Figura 5.1.- Espacio de color HSV como una rueda de color

El sistema se puede representar como se muestra en la Figura. 5.1. En ella el matiz se representa por una región circular; una región triangular separada, puede ser usada para representar la saturación y el valor del color. El eje horizontal del triángulo denota la saturación, mientras que el eje vertical corresponde al valor del color. De este modo, un color puede ser elegido al tomar primero el matiz de una región circular, y después seleccionar la saturación y el valor del color deseados de la región triangular.

5.1.4.- Histograma de color

En las imágenes digitales un histograma [12] de color representa el número de pixeles que tienen colores en cada una de las listas fijas de rangos de colores, que se extienden sobre el espacio de color de la imagen, es decir el conjunto de todos los posibles colores.

El histograma de color puede ser construido para cualquier tipo de espacio de color, aun cuando el término es usado más frecuentemente en espacios tridimensionales como RGB o HSV. El histograma de color es una estadística que puede ser visto como una aproximación de distribución continua fundamental de valores de colores.

5.2.- Procesamiento de Imágenes

Una imagen puede ser definida como una función de dos dimensiones, *f (x, y)*, donde *x* y *y* representan las coordenadas del plano, y la amplitud de *f* en cualquier par de coordenadas *(x, y)* es la intensidad o llama nivel de gris (o de color) de la imagen en ese punto. Cuando *x*, y, *y* los valores de amplitud de *f* son todos cantidades discretas finitas, llamamos a la imagen digital.

El campo de procesamiento de imagen digital se refiere al procesamiento de imágenes [12] digitales por medio de un ordenador digital. Tenga en cuenta que una imagen digital se compone de un número finito de elementos, cada uno de los cuales tiene una ubicación y el valor particular. Estos elementos se denominan elementos de imagen, los elementos de imagen y los píxeles. Pixel es el término más utilizado para referirse a los elementos de una imagen digital.

Por otro lado, hay campos como la visión por computadora cuyo objetivo final es utilizar las computadoras para emular la visión humana, incluyendo el aprendizaje y ser capaz de hacer inferencias y tomar acciones basadas en entradas visuales. El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad o facilitar la búsqueda de información.

El procesamiento de imágenes se divide en las siguientes etapas:

- Captura.
- Pre procesamiento.
- Segmentación.
- Extracción de características.
- Identificación de objetos.

5.2.1.- Captura

Diseño de las propiedades de la captura como resolución y numero de bits de color, tipo de cámara, distancia al objeto, mega píxeles, etc.

5.2.2.- Pre-Procesamiento

El proceso de filtrado es un conjunto de técnicas englobadas dentro del pre-procesamiento de imágenes cuyo objetivo fundamental es obtener a partir de una imagen origen,

otra final cuyo resultado sea más adecuado para una aplicación específica mejorando ciertas características de la misma, que posibilite efectuar operaciones del procesamiento sobre ella.

Los principales objetivos que se persiguen con la aplicación de filtros [12] son:

- Cambiar una imagen de color a escala de grises.
- Suavizar la imagen: reducir la cantidad de variaciones de intensidad entre píxeles vecinos.
- Eliminar ruido: eliminar aquellos píxeles cuyo nivel de intensidad es muy diferente al de sus vecinos y cuyo origen puede estar tanto en el proceso de adquisición de la imagen como en el de transmisión.
- Realzar bordes: destacar los bordes que se localizan en una imagen.
- Detectar bordes: detectar los píxeles donde se produce un cambio brusco en la intensidad.

Por tanto, se consideran los filtros como operaciones que se aplican a los píxeles de una imagen digital para optimizarla, enfatizar cierta información o conseguir un efecto especial en ella.

5.2.3.- Segmentación

La segmentación [13] es el proceso de dividir una imagen digital en varias partes (grupos de píxeles) u objetos. El objetivo de la segmentación es simplificar y/o cambiar la representación de una imagen en otra más significativa y más fácil de analizar. Se usa tanto para localizar objetos como para encontrar los límites de estos dentro de una imagen.

El resultado de la segmentación de una imagen es un conjunto de segmentos que cubren en conjunto a toda la imagen, o un conjunto de las curvas de nivel extraídas de la imagen. Cada uno de los píxeles de una región son similares en alguna característica, como el color, la intensidad o la textura. Regiones adyacentes son significativamente diferentes con respecto a las mismas características.

5.2.3.1.- Métodos de Segmentación Utilizados para este Proyecto

5.2.3.1.1.- Métodos Basados en el Histograma

Los métodos basados en el histograma [13] son muy eficientes en comparación con otros métodos de segmentación de la imagen, ya que normalmente requieren sólo una pasada por los pixeles. En esta técnica, un histograma se calcula a partir de todos los píxeles de la imagen, y los picos y valles en el histograma se utilizan para localizar los grupos en la imagen (el color o la intensidad pueden ser usados como medida).

5.2.3.1.2.- Detección de Bordes

La detección de bordes [13] es un campo bien desarrollado por sí mismo en el procesamiento de imágenes. Los límites de regiones y los bordes están estrechamente relacionados, ya que a menudo hay un fuerte ajuste en la intensidad en los límites de las regiones. Las técnicas de detección de bordes pueden ser usadas como otra técnica de segmentación más. Los bordes identificados por la detección de bordes en ocasiones están desconectados. Para segmentar un objeto a partir de una imagen sin embargo, es necesario que los bordes formen figuras cerradas.

5.2.3.1.3.- Método del Valor Umbral (Umbralización)

Normalmente los métodos del valor umbral [14] "binarizan" la imagen de partida, es decir se construyen dos segmentos: el fondo de la imagen y los objetos buscados. La asignación de un pixel a uno de los dos segmentos (0 y 1) se consigue comparando su nivel de gris g con un cierto valor umbral preestablecido *t* (en inglés *threshold*). La imagen final es muy sencilla de calcular ya que para cada pixel sólo hay que realizar una comparación numérica. La regla de cálculo correspondiente *Tg* es:

$$T(g) = \begin{cases} 0, & g < t \\ 1, & g \geq t \end{cases}$$

Los métodos del valor umbral son métodos de segmentación completos, es decir cada pixel pertenece obligatoriamente a un segmento y sólo uno.

5.2.4.- Extracción de Características

Seleccionar y extraer “características” apropiadas para la identificación de los objetos deseados. En el procesamiento de imágenes, visión por computadora y otros campos relacionados, el momento de la imagen es un cierto promedio ponderado determinado de las intensidades de los píxeles de la imagen, o una función de este tipo de momentos, por lo general elegido a tener alguna propiedad atractiva o interpretación. Los momentos de imagen son útiles para describir los objetos después de la segmentación. Las propiedades simples de la imagen que se encuentran a través de momentos de imagen incluyen área (o intensidad total), su centro de gravedad, y la información sobre su orientación.

Momentos primarios de una imagen [14].

Para una función 2D continua $f(x, y)$ el momento de orden $(p + q)$ se define como

$$M_{pq} \iint_{-\infty-\infty}^{\infty\infty} x^p y^q f(x,y)dx\,dy$$

para p , q = 0,1,2, ... La adaptación de este a escalar (escala de grises) de imagen con intensidades de píxel $I(x, y)$, momentos de imagen en bruto M_{ij} se calculan

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

Un teorema de unicidad (Hu [1962]) establece que si $f(x, y)$ es continua y tiene valores distintos de cero sólo en una parte finita del plano xy , existen momentos de todas las órdenes, y la secuencia de momento (M_{pq}) es determinado unívocamente por $f(x, y)$. A la inversa, (M_{pq}) determina de forma única $f(x, y)$. En la práctica, la imagen se resume con las funciones de unos momentos de orden inferior.

Momentos centrales se definen como

$$\mu_{pq} \iint_{-\infty-\infty}^{\infty\infty} (x-\bar{x})^p (y-\bar{y})^q \, f(x,y) dx \, dy$$

Donde $\bar{x} = \frac{M_{10}}{M_{00}}$ y $\bar{y} = \frac{M_{01}}{M_{00}}$ son los componentes del centro de masa.

Si $f(x, y)$ es una imagen digital, entonces la ecuación anterior se convierte

$$\mu_{ij} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q \, f(x,y)$$

Los momentos centrales de orden de hasta 3 son:

$\mu_{00} = M_{00}$,
$\mu_{01} = 0$,
$\mu_{11} = M_{11} - \bar{x}M_{01} = M_{11} - \bar{y}M_{10}$,
$\mu_{20} = M_{20} - \bar{x}M_{10}$,
$\mu_{02} = M_{02} - \bar{y}M_{01}$,
$\mu_{21} = M_{21} - 2\bar{x}M_{11} - \bar{y}M_{20} + 2\bar{x}^2 M_{01}$,
$\mu_{12} = M_{12} - 2yM_{11} - \bar{x}M_{02} + 2\bar{y}^2 M_{10}$,
$\mu_{30} = M_{30} - 3\bar{x}M_{20} + 2\bar{x}^2 M_{10}$,
$\mu_{03} = M_{03} - 3\bar{y}M_{02} + 2\bar{y}^2 M_{01}$.

Momentos centrales son invariantes traslacional.

Escala momentos invariantes

Momentos η_{ij} donde $i + j \geq 2$ puede ser construido para ser invariante a la traslación y los cambios en la escala dividiendo el momento central correspondiente por el debidamente escalado (00)-ésimo momento, utilizando la siguiente fórmula:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+\frac{i+j}{2})}}$$

Momentos invariantes de rotación.

Es posible calcular momentos que son invariantes bajo traslación, cambios en la escala, y también la rotación. Los más utilizados son el conjunto de momentos invariantes de *Hu*:

$$I_1 = \eta_{20} + \eta_{02}$$
$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12}) - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

La primera es análoga al momento de inercia alrededor del centro de masa de la imagen, donde las intensidades de los píxeles son análogas a la densidad física. El último, I_7, es invariante a la inclinación, lo que le permite distinguir imágenes de espejo de imágenes de otro modo idénticas.

Con el uso de éstos momentos invariantes se determinarán las características del objeto a buscar en este proyecto.

5.2.5.- Identificación de Objetos

Tiene la tarea para encontrar e identificar objetos en una imagen o secuencia de video. [13] Los humanos reconocemos una multitud de objetos en imágenes con poco esfuerzo, a pesar del hecho que la imagen del objeto puede variar un poco en diferentes puntos de vista, en diferentes tamaños o escala e incluso cuando están trasladados o rotados. Los objetos pueden ser reconocidos cuando están parcialmente obstruidos desde una vista. No obstante esta tarea es un desafío para los sistemas de visión por computadoras. Para este problema se han implementado muchos métodos durante múltiples décadas, destacando los siguientes:

5.2.5.1.- Métodos Basados en Apariencia

Se basan principalmente en encontrar patrones en la distribución de colores, bordes, distribución de la probabilidad de tonalidades, así como el agrupamiento de colores. Son muy eficaces para encontrar figuras geométricas planas como líneas rectas, líneas curvas, polígonos, circunferencias, etc. También para detectar zonas de color similar.

5.2.5.2.- Métodos Basados en Características

Estos métodos se caracterizan por buscar descriptores de información contenida en la imagen como distribución de colores, agrupamiento de formas, regiones con texturas similares, etc. El tipo de información obtenida puede ser más completa, pero se necesitan muchas muestras y mucho tiempo de análisis para poder obtener buenos resultados. Un ejemplo es la detección de rostros en la cual se busca detectar objetos más pequeños como los ojos, la nariz y la boca y se busca que la distribución entre ellos corresponda a una cara.

5.3.- Tarjeta de Desarrollo *Raspberry Pi*

Raspberry Pi [15] es un ordenador de placa reducida o placa única de bajo coste desarrollado en Reino Unido por la Fundación *Raspberry Pi*, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.

El diseño incluye un *System-on-a-chip Broadcom* BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos "Turbo" para que el usuario pueda hacerle *overclock* de hasta 1 GHz sin perder la garantía), un procesador gráfico (GPU) *VideoCore* IV, y 512 MB de memoria RAM (aunque originalmente al ser lanzado eran 256 MB). El diseño no incluye un disco duro ni unidad de estado sólido, ya que usa una tarjeta SD[2] para el almacenamiento permanente; tampoco incluye fuente de alimentación ni carcasa. El 29 de febrero de 2012 la fundación empezó a aceptar órdenes de compra del modelo B, y el 4 de febrero de 2013 del modelo A.1.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, *Raspbian*[3] (derivada de *Debian*[4]), *RISC OS*[5], *Arch Linux* ARM[6] (derivado de *Arch Linux*[7]) y *Pidora*[8] (derivado de *Fedora*[9]); y promueve principalmente el aprendizaje del lenguaje de programación *Python*[10]. Otros lenguajes también soportados son *Tiny BASIC*[11], C++[12], *Perl*[13] y *Ruby*[14].

[2] http://tarjetasd.com/
[3] http://www.raspbian.org/
[4] https://www.debian.org/index.es.html
[5] https://www.riscosopen.org/content/
[6] http://archlinuxarm.org/
[7] https://www.archlinux.org/
[8] http://pidora.ca/
[9] https://getfedora.org/
[10] https://www.python.org/
[11] https://www.dmoz.org/Computers/Programming/Languages/BASIC
[12] http://es.wikipedia.org/wiki/C%2B%2B
[13] https://www.perl.org/
[14] https://www.ruby-lang.org/es/

5.4.- Robot Diferencial

Existen varios diseños de ruedas para elegir cuando se quiere construir una plataforma móvil sobre ruedas: diferencial, sincronizada, triciclo y de coche. Para el desarrollo de ésta práctica, se utilizará el **diseño diferencial** [16].

Se componen de dos ruedas en un eje común, cada rueda se controla independientemente, puede realizar movimientos en línea recta, en arco y sobre su propio eje de contacto de rodamiento, requiere de una o dos ruedas adicionales para balance o estabilidad. Sencillo mecánicamente, puede presentar problemas de estabilidad y su cinemática es sencilla, para lograr el movimiento en línea recta requiere que las dos ruedas de tracción giren a la misma velocidad.

5.5.- Servidores Web

Un servidor web [17] es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa.

5.6.- Wi-fi

Wi-Fi es un mecanismo de conexión de dispositivos electrónicos de forma inalámbrica [18]. Los dispositivos habilitados con *wi-fi*, tales como un ordenador personal, una consola de video juegos, un teléfono inteligente, o un reproductor de audio digital, pueden conectarse a Internet a través de un punto de acceso de red inalámbrica. Dicho punto de acceso tiene un alcance de unos 20 metros en interiores, una distancia que es mayor al aire libre.

6.- Desarrollo del Proyecto

En esta sección se hace referencia al conjunto de procedimientos que se desarrollaron para alcanzar el objetivo del proyecto; esto es, diseño e implementación de un robot móvil el cual reconoce, busca y localiza objetos de forma regular y de color sólido mediante el procesamiento de imágenes en tiempo real. En la Figura 6.1 se muestran los módulos del sistema y sus interacciones con los demás.

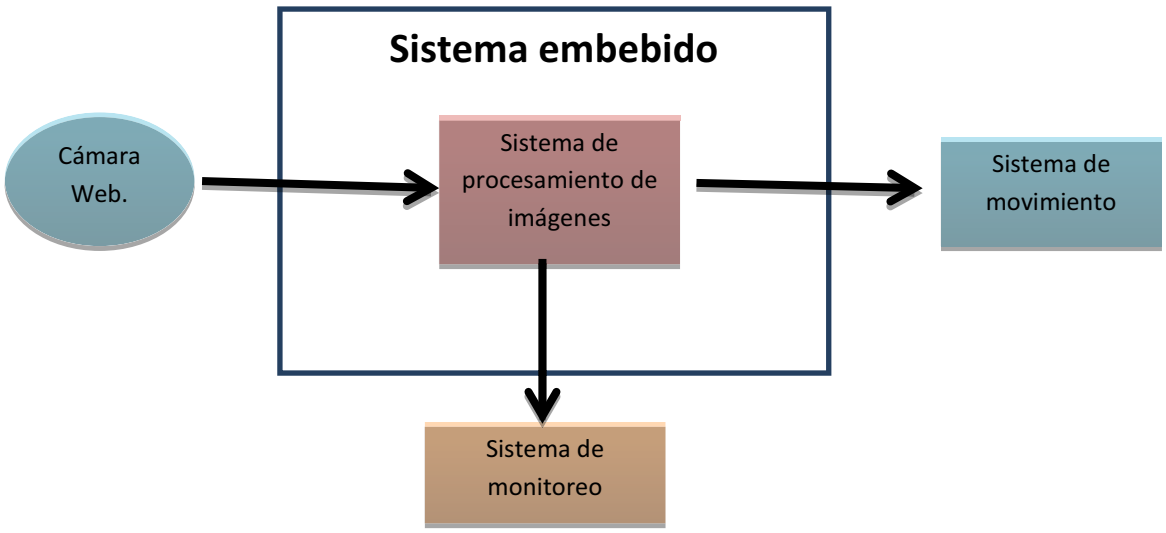

Figura 6.1.- Componentes que conforman el sistema.

6.1.- Sistema Embebido

La implementación del sistema embebido se realizó integrando los componentes de la figura 6.1, para que trabajen como un solo sistema. La parte del hardware está compuesto por la tarjeta de desarrollo *Rasberry PI*, La cámara web, la tarjeta de red inalámbrica y el sistema de movimiento.

La parte del software la integran el sistema operativo instalado en la tarjeta *SD* del *Raspberry PI*, la librería de visión por computadora *OpenCV* [19], Servidor Web *Lighttpd* [20], el cual es parte del sistema de monitoreo, y el código fuente del programa que representa el sistema de procesamiento de imágenes.

El sistema embebido tiene implementado lo siguiente:

- Sistema operativo *Raspbian* (*Debian Wheezy*), la librería *OpenCV* 2.4.9, el servidor *Lighttpd*. (*Véase apéndice A*).
- El programa desarrollado dentro una carpeta denominada, PT2 (*Véase apéndice D*).

6.2.- Sistema de monitoreo

El sistema de monitoreo fue instalado en el dispositivo *Raspberry Pi*. El servidor Web *Lighttpd* se configuró en el puerto 80 para acceder por medio de cualquier navegador web escribiendo la dirección IP del *Raspberry Pi*. Se creó un archivo index.html (*Véase apéndice E*), el cual representa la página web con cuatro imágenes que muestran los resultados del procesamiento que se lleva a cabo. Ver figura 6.2.

En la primera imagen se muestra su color original con señales que destacan el resultado del procesamiento de imágenes, la segunda muestra los cambios del proceso de filtrado, la tercera muestra el histograma de colores del objeto analizado; y la Cuarta muestra el contorno guardado del objeto que será la base para buscar coincidencias.

Las imágenes se van cambiando cada 200 milisegundos y son recibidas del programa principal en aproximadamente la misma cantidad de tiempo. La página solo proporciona imágenes y no contiene nada que interfiera en el funcionamiento del sistema.

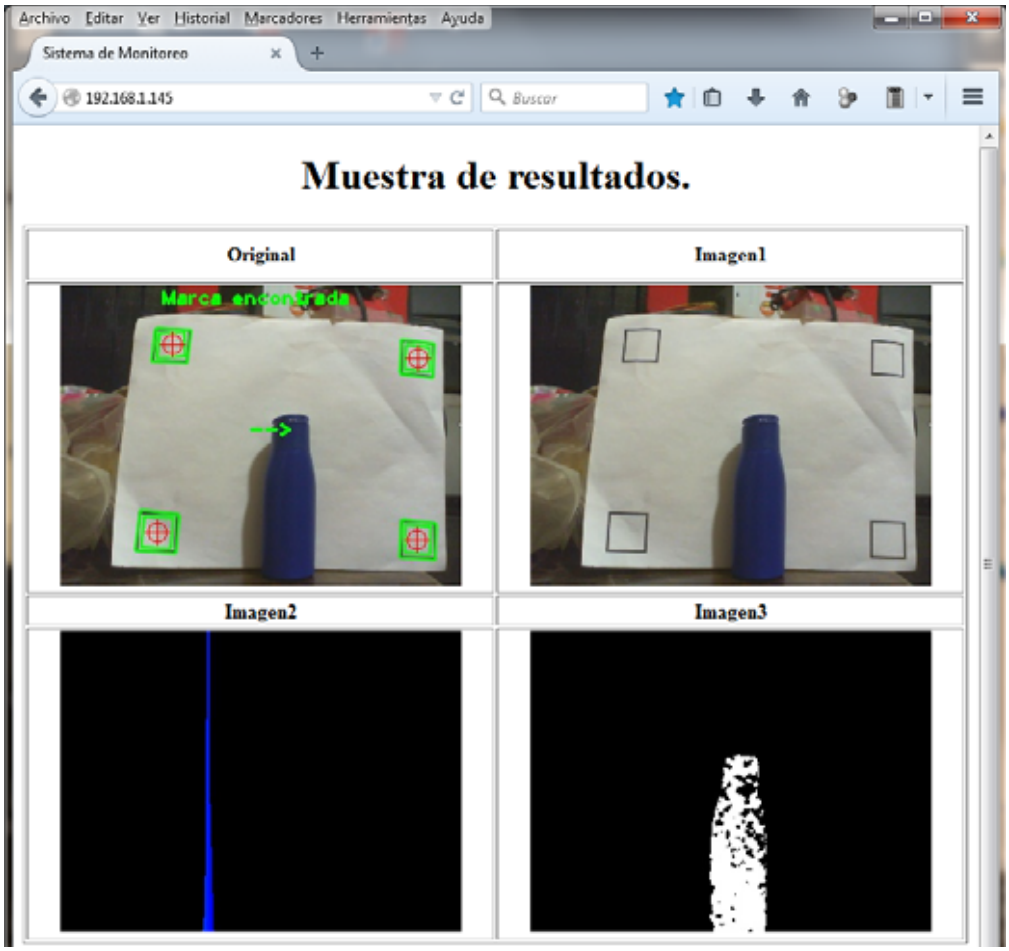

Figura 6.2 Página web abierta en un navegador de una PC que muestra el sistema de monitoreo funcionando.

6.3.- Sistema de movimiento

El sistema de movimiento se basa en un sistema formado por dos llantas conectadas cada una a un motor de corriente directa, y dos llantas pequeñas que solo sirven de soporte. Los motores giran las llantas para generar las funciones de avanzar hacia adelante y avanzar hacia atrás, girar hacia la derecha o girar hacia la izquierda sobre su propio eje, así como detenerse por completo.

El sistema está basado en el diseño de movimiento diferencial, utilizando un controlador de dos motores de corriente directa usando un puente en H (L293D[15]) que administran la corriente de dos baterías de litio de 3.6V el cual alimenta a los motores, y un circuito integrado (UNL2803A[16]) que convierte voltajes de 3.3V del sistema embebido a 5V para el controlador de los motores, ya que es un circuito TTL de 5V. Figura 6.3.

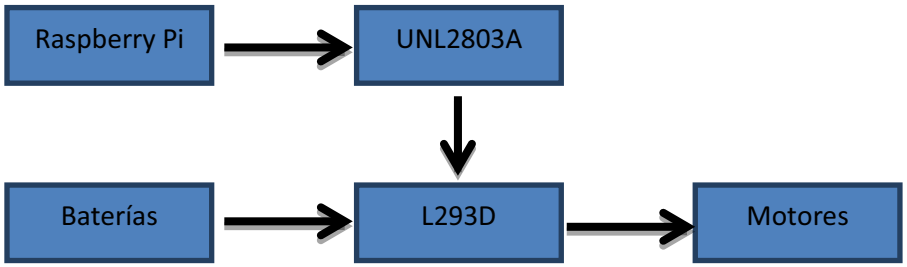

Figura 6.3 Diseño del sistema de movimientos

Los movimientos del robot se obtuvieron asignando un valor lógico 1 (HIGH) y 0 (LOW) a las dos terminales de cada motor para generar movimiento y el mismo valor a ambas terminales para detenerlos. Las instrucciones se envían por a los puertos GPIO'S de la siguiente manera:

Motores	**Raspberry PI B**		**Movimientos del robot**				
	Nombre del puerto.	Número de pin.	Detenido	Adelante	Atrás	Izquierda	Derecha
Izquierdo	GPIO17	11	HIGH	LOW	HIGH	HIGH	LOW
	GPIO18	12	HIGH	HIGH	LOW	LOW	HIGH
Derecho	GPIO22	15	HIGH	HIGH	LOW	HIGH	LOW
	GPIO23	16	HIGH	LOW	HIGH	LOW	HIGH

Movimientos en base a las señales enviadas del sistema embebido al sistema de movimiento.

[15] http://www.datasheetcatalog.com/datasheets_pdf/L/2/9/3/L293D.shtml
[16] http://search.datasheetcatalog.net/key/UNL2803A

6.4.- Sistema de procesamiento de imágenes

El procesamiento de imágenes se encarga de analizar las imágenes capturadas por medio de la cámara web para mostrar los resultados en el sistema de monitoreo y controlar la toma de decisiones para el sistema de movimiento.

Etapas del sistema de procesamiento de imágenes:

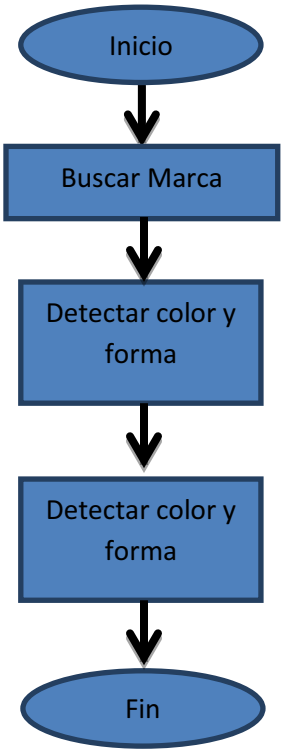

Figura 6.4 Diagrama a bloques del Sistema de procesamiento de imágenes.

6.4.1.- Inicio

Al encender el dispositivo se inician automáticamente los servicios web y SSH (*véase apéndices B y D*), así como los controladores de la cámara web y de la tarjeta inalámbrica. Se conecta también automáticamente a un ruteador vía wifi con IP fija 192.168.1.116 la cual se establece de manera manual y que no sea asignada por DHCP. Esto nos permite tener control todo momento sobre la dirección del sistema en la red local para el sistema de monitoreo y el acceso remoto.

La implementación del sistema se realizó en lenguaje C++ utilizando funciones de la librería OpenCV versión 2.4.9 la cual contiene algoritmos diseñados para el tratamiento de imágenes.

Las imágenes se capturan a color de 32 bits para obtener mayor detalle del color del objeto a buscar. La resolución se estableció de 320 x 240 pixeles debido a la capacidad limitada del procesador del sistema embebido. Con una resolución mayor se podrían obtener mayor detalle sobre la forma del objeto, pero el tiempo de procesamiento se cuadruplica a 640x480 pixeles, por ejemplo. Con una resolución menor se aumenta el procesamiento pero se pierde información. Con esta resolución se estableció que el sistema trabaje a 5 cuadros por segundo obteniendo un buen balance en velocidad y rendimiento.

Para el sistema de procesamiento de imagines se desarrolló en base al diagrama de la figura 6.5.

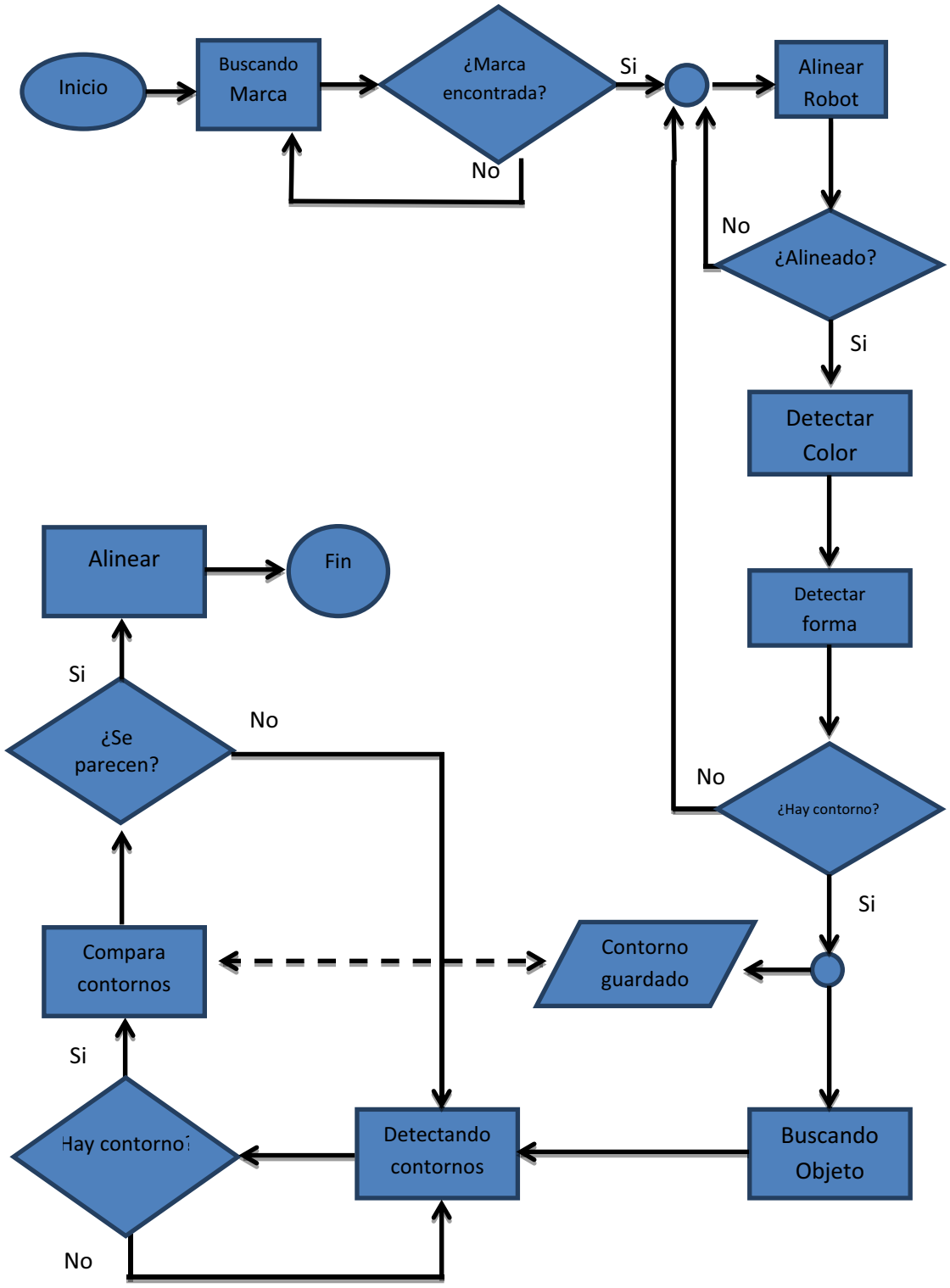

Figura 6.5 Diagrama a bloques del Sistema de procesamiento de imágenes.

6.4.2.- Buscar Marca

Se utiliza una marca distintiva para la colocación del objeto y para que el sistema identifique su ubicación, la cual se coloca frente al objeto y este es analizado.

Consta de una hoja carta de color blanco colocada de manera horizontal con cuatro cuadrados huecos de contorno negro dibujados cerca de las esquina de la hoja. El tamaño de los cuadrados es de 3 cm por lado y el grosor de la línea es de 3mm. Él tamaño puede variar, pero no deben quedar cubiertos al colocar el objeto a analizar. Él objeto se coloca cerca de la hoja y en la parte media de los dos cuadrados inferiores. Sí no se coloca en esta posición, el sistema no podrá analizarlo correctamente.

Al iniciar el programa, el robot busca la marca de los cuatro cuadrados, una vez encontrada toma las coordenadas de dos cuadrados inferiores para centrar la imagen y acercarse lo suficiente para analizar el color y la forma del objeto. La muestra se toma de un segmento interno del objeto para poder crear la muestra correcta; si no se logra establecer una muestra correcta, el robot retrocede y vuelve a acercarse.

En ésta etapa el procesamiento de imágenes se utiliza para reconocer la marca de los cuatro cuadrados, y obtener las coordenadas de los mismos para enviar las instrucciones de movimiento y centrar la marca en cuadro de visión de la cámara. Las coordenadas nos servirán para ubicar un rectángulo en el centro de los cuadros inferiores, que es donde se encuentra el objeto.

6.4.2.1.- Captura

En la etapa de captura se establece la resolución al momento de configurar la captura con las siguientes instrucciones:

```
...
const int FRAME_WIDTH  = 320; //ancho
const int FRAME_HEIGHT = 240; //alto
...
VideoCapture cap; // Clase para adquirir imágenes
...
cap.set(CV_CAP_PROP_FRAME_WIDTH ,FRAME_WIDTH );
cap.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
...
```

La imagen capturada guarda en una variable de la siguiente manera:

```
cap >> frame; // obtiene un cuadro de la cámara
```

6.4.2.2.- Pre-procesamiento

Se prepara la imagen para la segmentación aplicando diferentes filtros, el primero es convertir la imagen a escala de grises con la función `cvtColor();` el segundo filtro reduce la resolución de la imagen a la mitad y se vuelve a ampliar a la resolución original para eliminar ruido, con `pyrDown() y pyrUp()`, y el tercero filtro es un gaussiano para suavizar la imagen y eliminar el mayor ruido posible utilizando como parámetros los recomendados por OpenCV para la detección de líneas de la siguiente manera:

```
GaussianBlur(dilate, gaussian, Size(9,9), 1.5, 1.5);
```

6.4.2.3.- Segmentación

La imagen filtrada es segmentada mediante el algoritmo de *Canny*[17], con lo cual obtenemos una imagen binaria que muestra todos los bordes encontrados en color blanco y el fondo negro:

```
Canny(gaussian, canny, 0, 30, 3);
```

[17] http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

Los parámetros utilizados: 0 y 30, son los que mejor resaltaron los bordes de los cuadrados de la marca debido a que la hoja no se aprecia completamente blanca en la cámara. El valor 3 es el valor por default de la función.

6.4.2.4.- Extracción de Características

En la imagen binaria se buscan contornos, ya que los cuadrados de la marca son figuras cerradas. De éste proceso se obtiene un conjunto de coordenadas que representan cada uno de los contornos encontrados en la imagen. Éste arreglo se procesa para encontrar figuras geométricas, que en este caso; son cuadrados.

El proceso es el siguiente:

Encontrar contornos con la función *findContours()*.
Para cada contorno encontrado
Crear un polígono del área con *approxPolyDP()*.
Calcular su área con *contourArea(),*
    Sí polígono tiene 4 lados y es convexo
    Para cada lado
        Calcula el angulo entre ellos.
    Si es casi 90
        Guarda el polígono.

Al final del proceso tenemos un arreglo con las coordenadas de los cuadrados encontrados.

6.4.2.5.- Identificación de Objetos

Se dibuja un recuadro verde sobre la imagen original, resaltando cada cuadro encontrado utilizando el arreglo del proceso anterior y creando cada línea con la siguiente instrucción:

```
polylines(image, &p, &n, 1, true, color_verde, 1, CV_AA);
```

Sí el arreglo contiene cuatro objetos, entonces se considera que se ha encontrado la marca Cuando se detecta los cuatro cuadrados, se informa que se ha encontrado la marca, En la imagen original se dibuja la frase “Marca encontrada” y se resalta en rojo el interior de cada cuadro con la función *drawObject()*.

Una vez obtenida la marca, se identifican los dos cuadrados inferiores y se calculan dos datos: la distancia entre los centros y la coordenada *x* del punto medio de dicha distancia. El punto *x* nos sirve para centrar la imagen con respecto a la horizontal del campo de visión de la cámara. La distancia obtenida es utilizada para calcular la relación con respeto al ancho de la imagen y poder acercar el robot a la marca.

Los movimientos laterales que puede realizar el robot son determinados por la posición del punto x con respecto al centro de la imagen utilizando la función *movimientos()*, rotando primero a la izquierda o a la derecha hasta centrarlo en un margen de error de 16 pixeles. Una vez centrado, avanza hacia adelante hasta que la relación calculada no sea menor a 0.73. Ésta relación se determinó porque es la que representa un acercamiento que abarca la hoja completa y una distancia cercana al objeto.

Con el robot y cerca de la marca se detiene el sistema de movimiento y el proceso de Búsqueda de Marca, y se iniciara el análisis del objeto en la etapa Detectar Color y Forma.

6.4.3.- Detectar Color y Forma

En esta etapa se dibuja un rectángulo en el centro de los cuadrados inferiores de 25 pixeles de lado el cual debe cubrir una sección interna de la imagen del objeto a analizar. De esta sección se toma la muestra de los colores de cada pixel y se usa para crear el histograma de colores y la máscara que ayuda a separar la imagen del objeto y el fondo de la imagen. Una vez separados se analiza la forma del objeto y se guardan sus valores característicos que sirven para la identificar del objeto.

6.4.3.1.- Captura

Se obtiene una imagen de la cámara y se utiliza las coordenadas de la sección para crear una segunda imagen que solo contenga la sección recortada de la siguiente manera:

6.4.3.2.- Pre-procesamiento

Se reduce la resolución de la imagen a la mitad y se vuelve a ampliar a la resolución original para eliminar ruido.

```
pyrDown(temp, temp, Size(frame.cols/2, frame.rows/2));
pyrUp(temp, temp, frame.size());
```

Creamos una imagen llamada *corte* de la imagen original a partir de la sección del rectángulo central.

```
corte = Mat(temp, selection);
```

De la imagen *corte* se obtienen los mínimos y máximos de los valores de tono, saturación y brillo (HSV). Primero la convertimos al modelo de color HSV con la instrucción

```
cvtColor(m, corteHSV, COLOR_BGR2HSV);
```

Después recorremos todos los pixeles de la imagen y vamos guardando los valores mínimos y máximos para utilizarlos más adelante para crear el histograma de colores y el filtro HSV.

La imagen original de transforma de color RGB al modelo HSV usando los valores mínimos y máximos obtenidos del corte, y se obtiene una imagen en color HSV. De la imagen HSV se crea una imagen binaria que es la primera máscara en la que los pixeles que se encuentran dentro del rango de mínimos y máximos que se representan en blanco, y los demás en negro con la siguiente instrucción:

```
inRange(hsv, hsvMin, hsvMax, mask);
```

Se crea una imagen llamada *hue* (tinte) mezclando los canales tomados del corte de la imagen original, para realzar la tonalidad de colores del objeto y ayude a los cambios leves de iluminación. Se aplica un filtro a la imagen *hue* en base a los valores del histograma para eliminar el fondo, obteniendo una imagen en escala de grises que representa la máscara del color predominante del objeto. La imagen se convierte a binaria. Esta imagen se combina con la primera máscara y se obtiene otra con un mejor detalle del objeto y se obtiene con las siguientes instrucciones:

```
calcBackProject(&hue, 1, ch, hist, backproj, &phranges);
threshold(backproj, cuadros, 5, 255, THRESH_BINARY);
```

Ésta última imagen *cuadros* contiene un mayor detalle de la forma del objeto y los filtros aplicados eliminan del fondo todo lo que no coincida con la tonalidad del color.

6.4.3.3.- Segmentación

En la imagen anterior se buscan todos los contornos obteniendo como resultado solo uno, el del objeto. Con la función *findContours(),* obtenemos todos los contornos, obtenemos sus momentos con la clase *Moments,* y utilizamos el momento *00* que nos proporciona el área. Sí ésta área es te tamaño considerable, entonces la consideramos como el contorno del objeto y lo guardamos en una variable como referencia.

Si no se encuentra ningún contorno quiere decir que hubo una mala muestra y se debe volver a tomar.

6.4.4.- Buscar y Detectar Objeto.

Una vez completada la etapa Detectar Color y Forma, el robot gira a la derecha hasta perder de vista el objeto analizado y comienza la búsqueda de otro objeto con las mismas características. El robot se mueve lentamente a la derecha hasta que aparezca un objeto similar en el campo de visión de la cámara. Se utilizan los valores HSV obtenidos de la etapa anterior; por lo cual solo aparecerá en la imagen binaria cualquier objeto con la misma tonalidad que el objeto a buscar. Una vez encontrado un objeto, se compara con el contorno guardado y si su forma es similar, se moverá el robot para centrarlo en la cámara y marcarlo como encontrado.

6.4.4.2.- Pre-procesamiento

Se utilizan los mismos filtros de la etapa anterior, tanto el histograma de colores como los mínimos y máximos valores de la muestra de color. De esta manera cualquier objeto que no pertenezca al rango de colores de la muestra, no se verá en la imagen filtrada, y el robot seguirá girando a la derecha. Un objeto de color similar aparecerá en ventana dependiendo su grado de coincidencia.

El robot seguirá girando a la derecha hasta que aparezca en la imagen un objeto del mismo color, cuando esto suceda, se guarda el contorno para que en la siguiente fase del procesamiento sea analizado.

6.4.3.3.- Segmentación

De la misma manera que en la etapa anterior, se busca el contorno de la imagen para extraer sus características.

6.4.3.4.- Extracción de características.

Del contorno obtenido se extraen los puntos que lo conforman, incluyendo las coordenadas de su centro de masa.

6.4.3.5.- Identificación de objetos

Se compara con el contorno del objeto a buscar por medio de sus momentos invariantes *Hu*, obteniendo un valor númerico de dicha comparacíon. La comparación se realiza con las siguientes intrucciones:

```
double compara;
compara = matchShapes(contours1,contours2, CV_CONTOURS_MATCH_I1,0);
```

Los valores que arroja el resultado son muy pequeños debido al tipo de analisis que utiliza la librería con respecto a los momentos Hu, obteniendo mejor apreciacion al multiplicarlos por 100. Estos valores son inversamente proporcionales a la similitud entre los contornos. Por las pruebas realizadas se determinó que valores menores a 40 representan una mayor coincidencia con el objeto.

Se una señal en la imagen original, utilizando las coordenadas de su ubicación y el número obtenido de la comparación de contornos. El carro seguirá girando a la derecha, pero deja la señal para indicar la leve coincidencia.

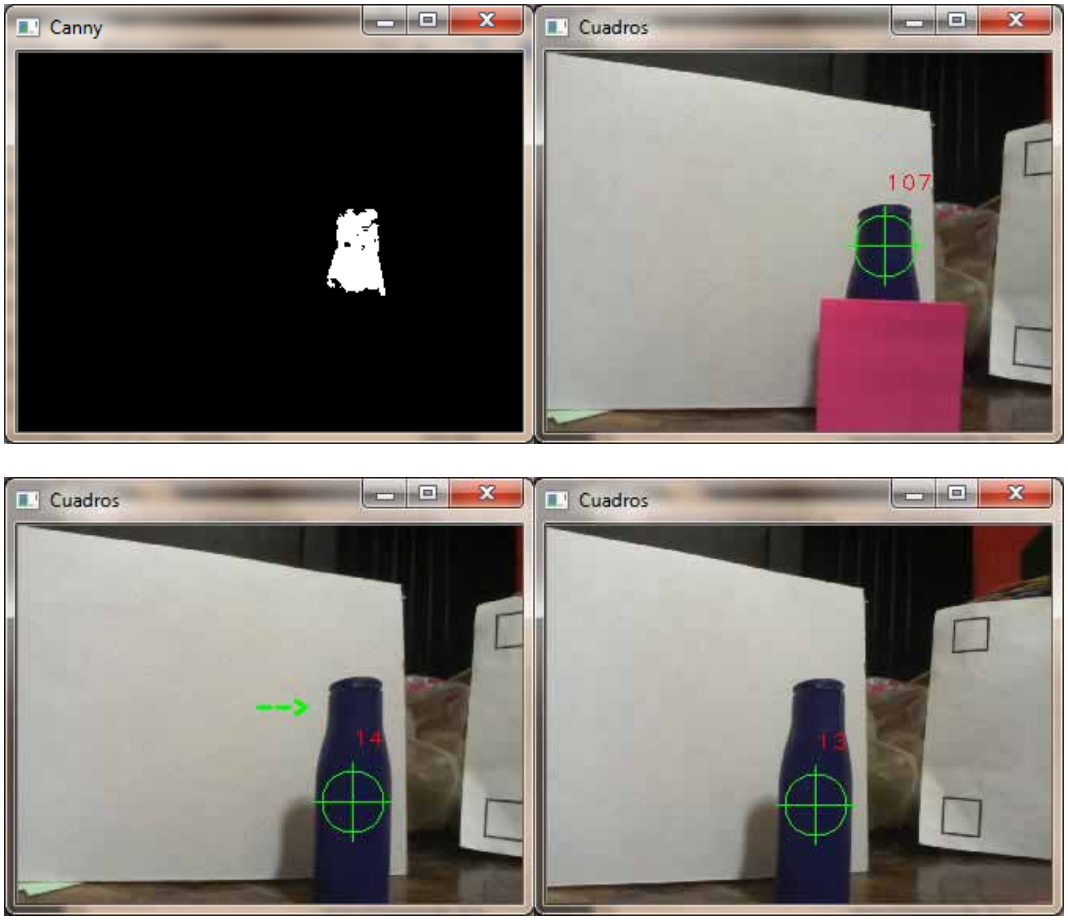

Figura 6.6.-Fase de detección del objeto.

Sí encuentra un objeto similar, entonces inicia movimientos laterales para centrar la imagen y colocarse frente al objeto y así dejar de moverse. Figura 6.6.

En este momento se da por encontrado el objeto.

7.- Resultados y análisis de resultados.

Se realizaron las pruebas en cinco objetos diferentes en color y forma. Figura 7.1. Cada objeto en condiciones de luz distinta, nombrados de izquierda a derecha y de arriba a abajo; objeto1, objeto2, objeto3, objeto4 y objeto5. Figura 7.14

Figura 7.1.- Objetos de prueba

Iluminación

- Luz artificial proporcionada por un foco, luz blanca.
- Luz directa del sol, en un área abierta.
- Iluminación indirecta de luz de día, sin fuente de iluminación artificial. Figura 7.2.

Figura 7.2.- Ejemplo de iluminación indirecta de luz de día

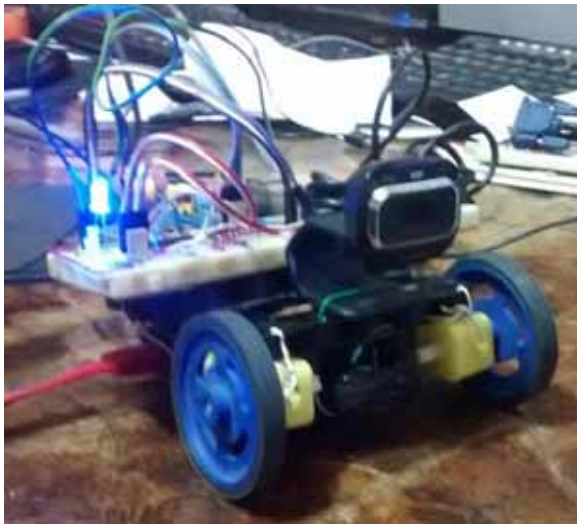

Figura 7.3 Vista frontal del prototipo

El sistema en lo general funciona como una sola unidad independiente, su encendido es mediante la conexión a una batería, la cual proporciona la energía suficiente para su correcto funcionamiento (Figura 7.3).

Sistema de monitoreo

Accedemos al sistema de monitoreo a través de un navegador web escribiendo la dirección IP 192.168.1.116. En la Figura 7.4 observamos las cuatro imágenes más significativas del procesamiento de imágenes que, al refrescarse cada 200 ms, ofrece la sensación de video, con lo cual se aprecia de mejor los resultados.

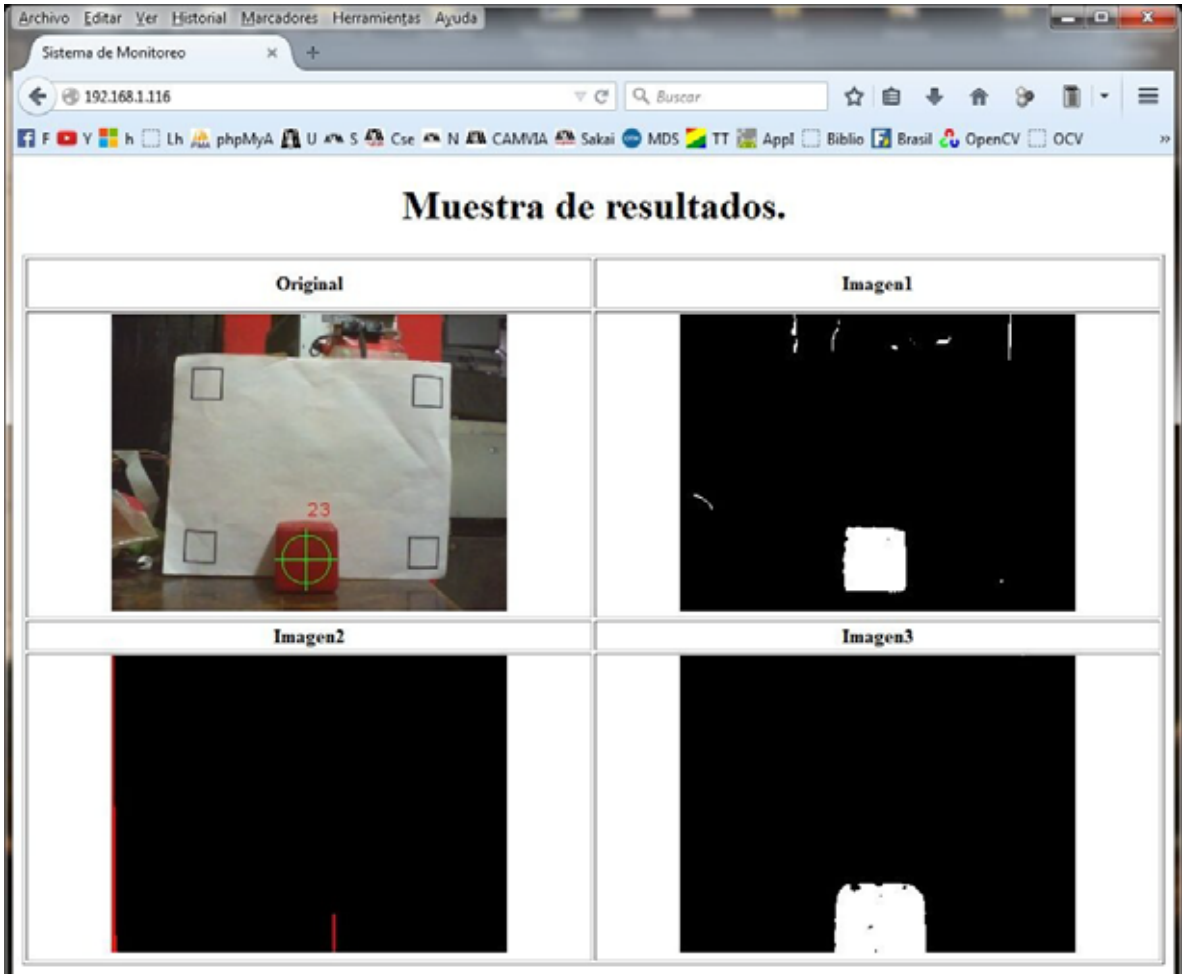

Figura 7.4 Navegador que muestra el sistema de monitoreo.

Sistema de movimiento

El sistema de movimiento responde bien a los desplazamientos generados por el sistema de procesamiento de imágenes, permitiendo al robot moverse para acercarse a la marca, así como durante la etapa de búsqueda. Se comprueba que nuestro dispositivo funciona de manera autónoma (Figura 7.5 y 7.6).

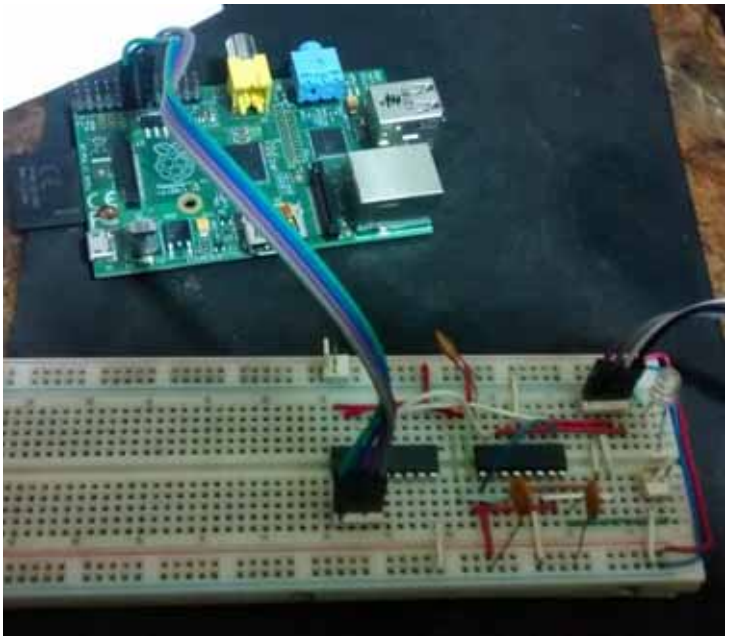

Figura 7.5 Sistema embebido y Sistema de movimiento.

Figura 7.6 Base de montaje con llantas y cámara web.

Sistema de procesamiento de imágenes.

Buscar marca

Figura 7.7 Etapas del procesamiento para encontrar la marca.

En las tres primeras imágenes de la figura 7.7 se observa que al pasar a escala de grises y aplicar el filtro gaussiano, se obtienen menos líneas al momento de segmentar con la función *Canny* en la cuarta imagen. Lo que facilita la detección de los cuadrados de la marca. En la cuarta imagen se dibujan cuadros verdes encima de los cuadrados detectados, como se puede apreciar el cuadrado interior izquierdo no está marcado ya que no aparece completo en la imagen de la cámara. En la última figura se encontraron los cuatro cuadrados se pone la leyenda "Marca encontrada" y comienza el análisis para centrar el robot y acercarse a la marca.

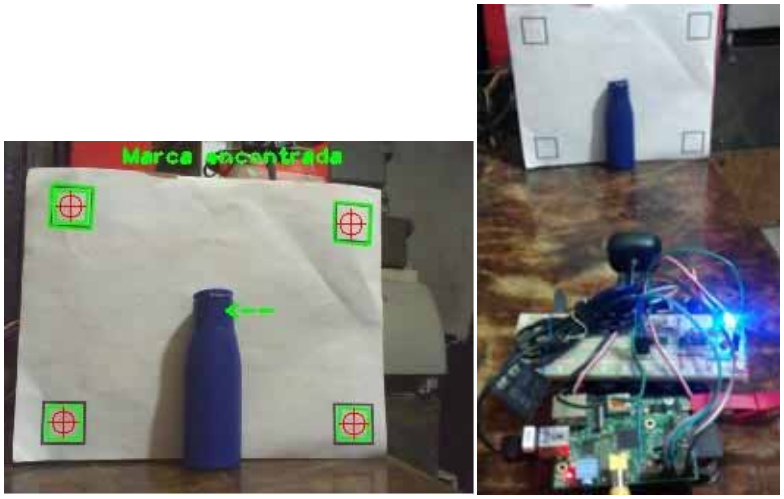

Figura 7.8 Movimientos laterales.

En la figura 7.8 se muestra la posición del robot con respecto a la marca, y se aprecia la leve inclinación a la derecha. El sistema busca con movimientos pequeños rotar a la izquierda en este caso hasta centrar la imagen.

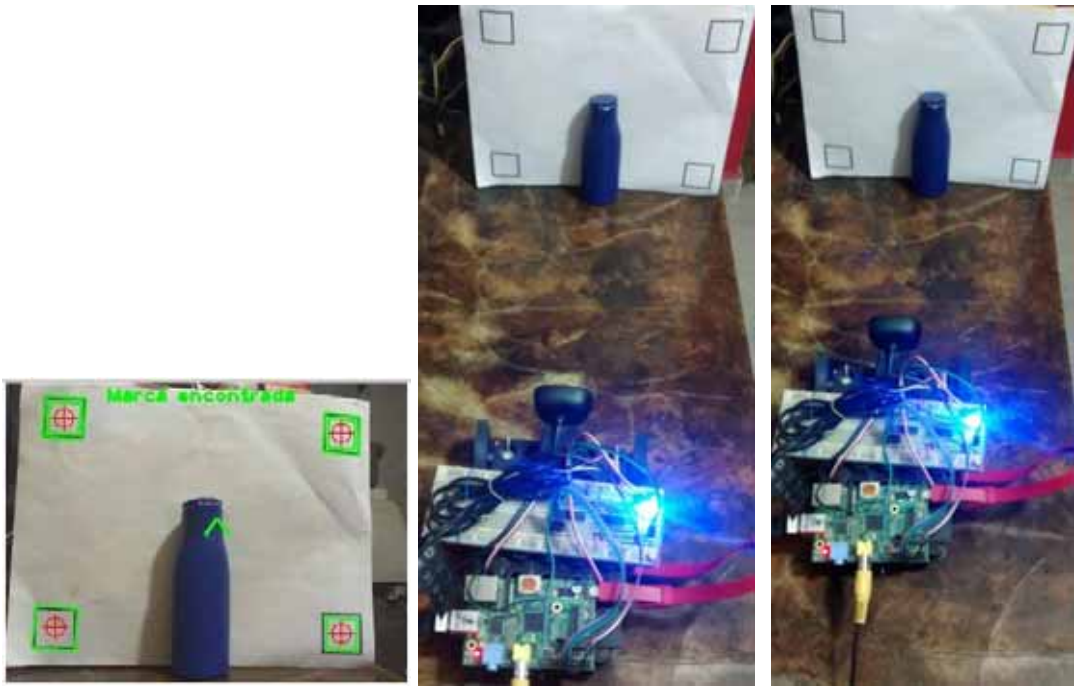

Figura 7.9 Movimientos de acercamiento.

En la figura 7.9 se aprecia cómo una vez centrado, se acerca con movimientos cortos hasta quedar cerca de la marca y frente al objeto. De esta manera la hoja ocupa la mayor parte del cuadro de la imagen y se pueden tener mejores detalles del color y forma del objeto.

Detectar color y forma.

El dispositivo es capaz de tomar una muestra del color del objeto, con ella se realiza el cálculo de colores, crea el histograma logrando separar el fondo y el objeto, obteniendo una imagen del contorno lo que el dispositivo interpreta como una imagen binaria del objeto.

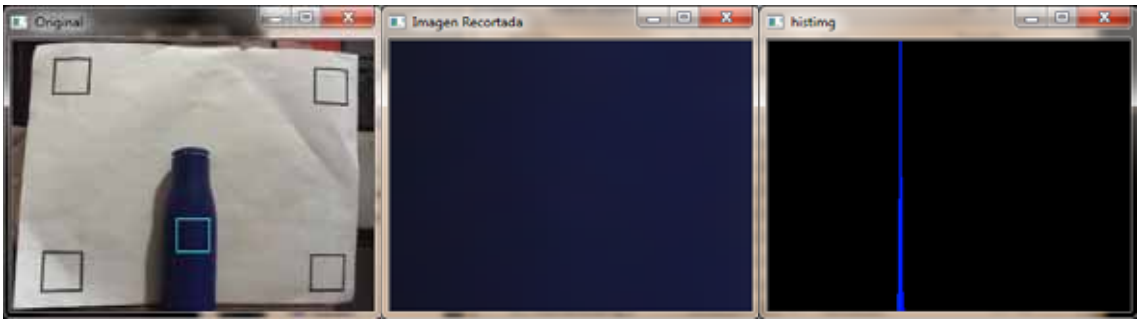

Figura 7.10 obtención de la muestra de color.

En la figura 7.10 se aprecia la muestra de color tomada de la imagen original. El histograma muestra los colores predominantes y si frecuencia, también se calculan los máximos y mínimos de los valores HSV.

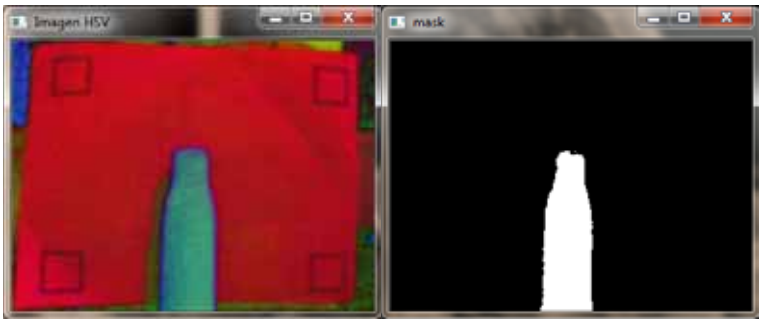

Figura 7.11 Creación de la máscara por colores HSV.

En la primera imagen figura 7.11 se muestra la conversión de la imagen original al esquema de colores HSV, ésta imagen se binariza utilizando los rangos obtenidos en la muestra como umbrales, obteniendo una imagen que resalta la silueta del objeto y servirá para filtrar el fondo en la etapa de búsqueda.

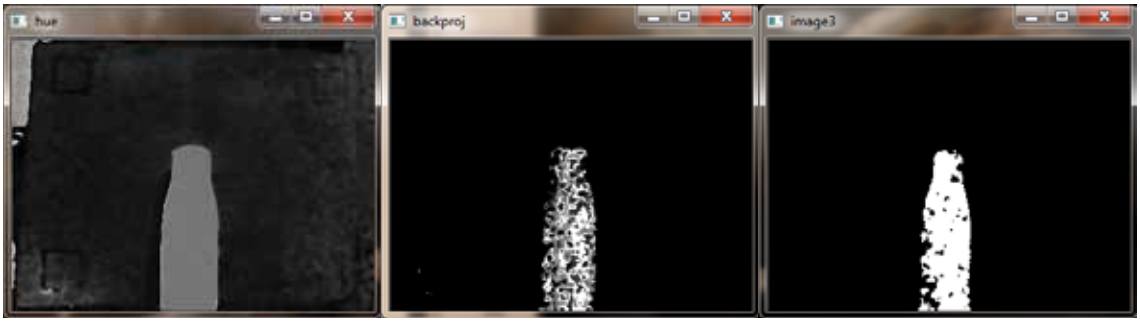

Figura 7.12 resaltado el color predominante e imagen binaria final.

La imagen original se convierte a colores HSV utilizando solo los valores de HSV como escala de grises resaltando del fondo el contorno del objeto como se muestra en la figura 7.12, al combinar la máscara obtenida en el paso anterior se muestra como se separa por completo el fondo de la imagen con el contorno. Por último se binariza la imagen para obtener la imagen que representa el contorno de la forma del objeto.

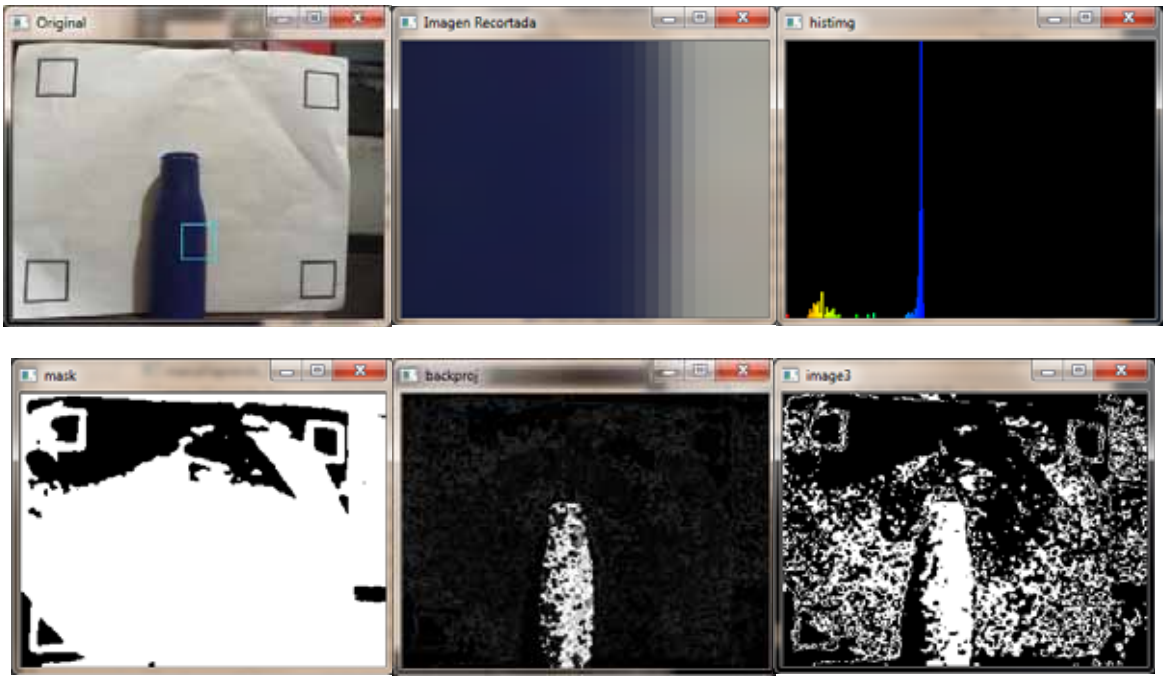

Figura 7.13.- Error en la muestra de color

En la figura 7.13 se muestra un caso de muestra mal tomada. El histograma reporta influencia de colores diferentes, obteniéndose valores muy amplios en la escala de tonalidades del color y resaltando más de una tonalidad en la imagen, y el resultado es una imagen binaria con mucho ruido que no permite resaltar solo el contorno. El robot retrocede y vuelve a acercarse para intentar tomar una mejor muestra.

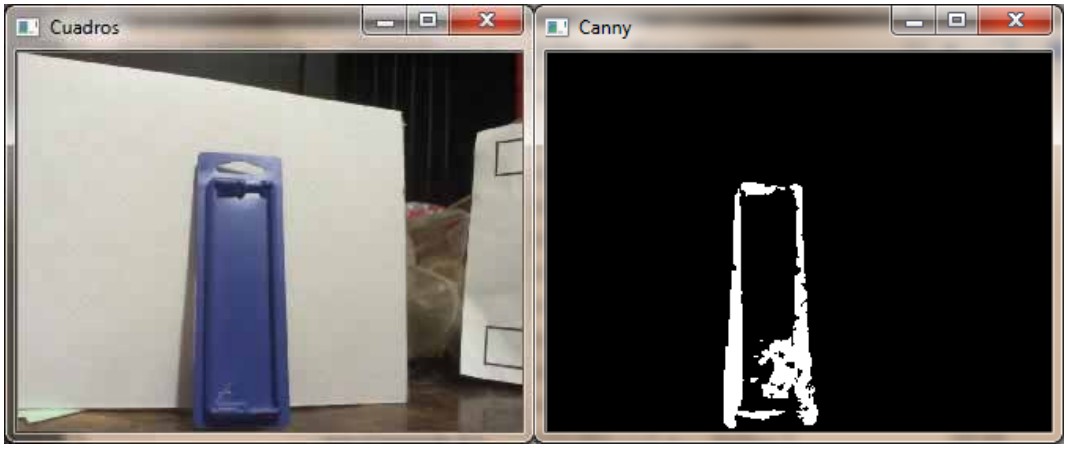

Figura 7.14.- Objeto de color similar y de forma diferente.

El sistema es capaz de detectar si el objeto encontrado corresponde o no a la forma solicitada; esto es, al realizar una prueba donde se colocó un objeto de color similar pero de forma diferente el dispositivo dio como resultado que el objeto no cumple con las especificaciones de la figura a buscar y por tanto es ignorado por el sistema (Figura 7.14). Esto quiere decir que el dispositivo es capaz de diferenciar formas de objetos.

Sí se encuentra un objeto similar, entonces inicia movimientos laterales para centrar la imagen y colocarse frente al objeto y así detenerse.

Figura 7.15 Centrando objeto encontrado.

El llegar seguir girando y encontrar un objeto parecido, el sistema le coloca una marca y comienza a centrar el robot como se muestra en la figura 7.15.

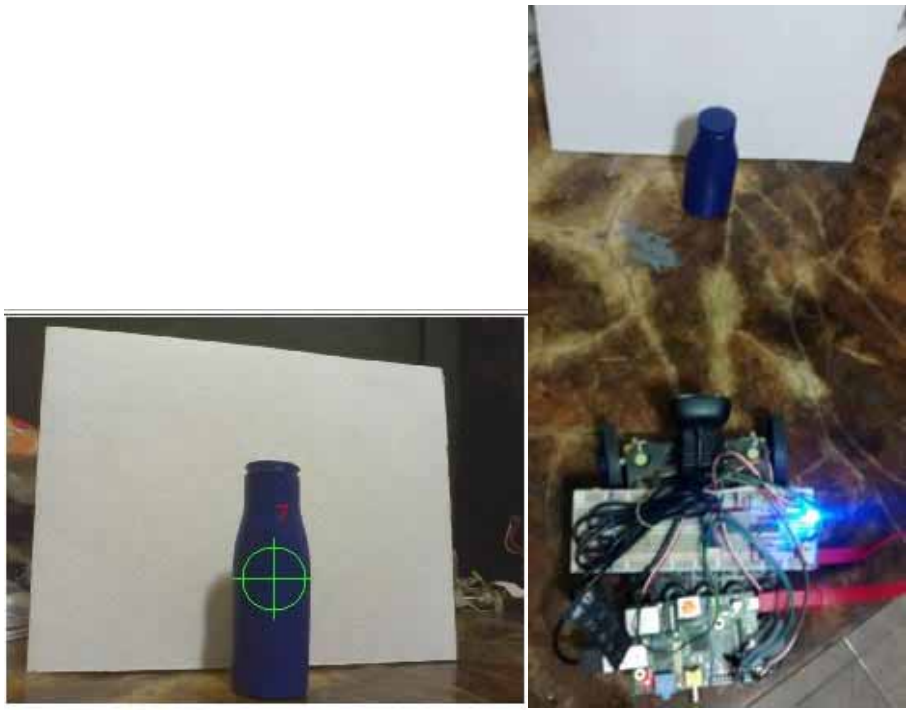

Figura 7.16 Objeto encontrado.

Ocurridos los eventos anteriores satisfactoriamente se da por encontrado el objeto (Figura 7.16).

De las pruebas realizadas con los cinco objetos de la figura 7.14 en las tres diferentes condiciones de luz se obtuvieron los siguientes resultados:

	Objeto 1	Objeto 2	Objeto 3	Objeto 4	Objeto 5
Luz artificial	si	si	si	si	si
Luz directa del sol	no	si	no	si	si
Luz indirecta de día	si	si	si	si	si

Tabla 7.1.- Resultados de detección de los objetos a buscar.

Se presentaron mejores coincidencias con objetos que no presentan texturas reflejantes como lo son los objetos 2, 4 y 5. Estos objetos mantuvieron mejor sus contornos en las tres condiciones de iluminación.

Los objetos que presentan un mayor reflejo de luz como el objeto 1 y 3, mostraron formas en el sistema que no coinciden con su forma original, debido a que se pierde mucho color en la zona donde es reflejada la fuente de iluminación en el objeto. El sistema encontró mayor coincidencia cuando la fuente de luz es reflejada casi en el mismo ángulo que en la muestra.

Los objetos con contorno más complejo como el 4 y el 5, ofrecen contornos con mayor información que ayudó al sistema a no confundir las dos figuras con el mismo color.

Para los objetos 4 y 5 se colocaron juntos para verificar que el robot pudiera distinguir la figura buscada e ignorar la silueta de un objeto diferente, obteniendo el resultado esperado.

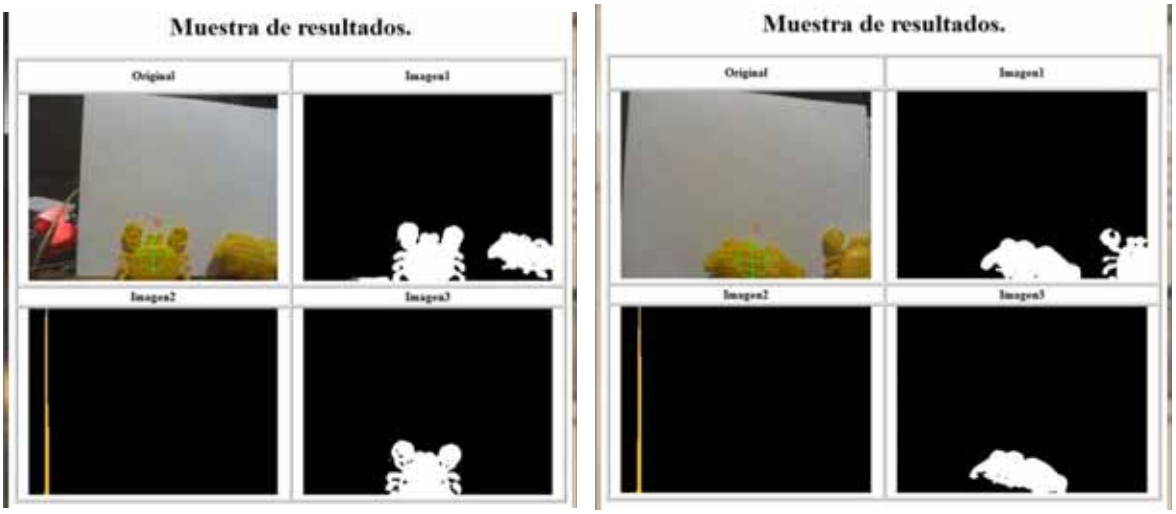

Figura 7.16.- Dos objetos del mismo color y diferente forma.

En la figura 7.16 se muestran dos procesos de búsqueda con objetos del mismo color y forma diferente y como aun así se detecta el objeto correctamente, gracias a la comparación de la forma.

8.- Conclusiones

La *Raspberry PI* representó una muy buena opción en este proyecto debido a la capacidad de procesamiento que tiene y siendo accesible económicamente. Durante la implementación de este proyecto respondió con buena velocidad de procesamiento. La librería OpenCV ofrece una gran variedad de funciones muy accesibles de implementar funciones de procesamiento de imágenes y trabajo de manera muy fluía en el sistema embebido..

La ventaja de crear software basado en sistemas embebidos; es que, por medio de programación se pueden crear funciones tan complejas como sean necesarias y que dependan solo de la habilidad del Ingeniero en Computación que lo desarrolla, para llevarlas a cabo y crear sistemas físicos (robots) autónomos que no se queden solo como una idea emulada en una computara de escritorio.

El procesamiento de imágenes ocupa muchos recursos del sistema embebido, para lo cual es de suma importante analizar en el tipo de algoritmos que se vayan a utilizar para con ello eficientar los recursos. En este caso, la detección de objetos se busca que sea en tiempo real; por ello se decidió trabajar a cinco cuadros por segundo y así analizar cada cuadro en un lapso de 200 ms y dar tiempo al sistema de realizar todas las operaciones necesarias. Los demás sistemas se adaptaron a este intervalo de tiempo para que el sistema en general trabaje adecuadamente.

El modelo a seguir en este proyecto para la detección de color y forma se basó, primero obtener una muestra del color y segundo su contorno; logrando un mejor procesamiento de imágenes al remover el fondo y disminuir en mucho la carga del sistema embebido.

Los objetos que son mejores para este sistema son aquellos que ofrezcan texturas opacas que no reflejen la fuente de luz para que mantengan ante la cámara un color uniforme en toda su superficie. Con ello la fuente de iluminación no afecta el buen funcionamiento del robot. Para objetos brillosos, se debe ocupar iluminación baja o que no presenten una sola fuente de luz para no perder información del color y de la forma del objeto.

El sistema se puede mejorar a futuro utilizando un sistema embebido con mayor capacidad de procesamiento. Carga de objetos ya analizados con anterioridad y solo se dedique a buscarlos. Así mimo se pueden agregar un sistema de navegación basado en seguimiento de líneas, caminos o de señales visuales para aumentar su autonomía; todos basados en reconocimiento de imágenes.

9.- Referencias bibliográficas

[1]Visión Artificial [En línea] Disponible:
http://www.ecured.cu/index.php/Visi%C3%B3n_Artificial

[2] E. D. Sobrado Malpartida, "Sistema de visión artificial para el reconocimiento y manipulación de objetos utilizando un brazo robot", tesis de maestría, Escuela de Graduados, Pontificia Universidad Católica DEL Perú, Perú, 2003.

[3] *Revista mexicana de ingeniería biomédica*, Reconocimiento y localización de instrumental médico usando análisis automatizado de imágenes, vol. *26*, Ciudad de México, México, 2005, pp. *75-85*.

[4] Gonzalez, E.A.B. Study of behaviors subsumed with the generation of motor schemas in LEGO NXT 2.0, Engineering Applications (WEA), 10.119, pp 1-6, mayo, 2012.

[5] Gonzalez, F. ; Guarnizo, J.G. ; Benavides, G. Emulation System for a Distribution Center Using Mobile Robot, Controlled by Artificial Vision and Fuzzy Logic, Latin America Transactions, IEEE (Revista IEEE America Latina), v12 , num 4, pp 555-563, june, 2014.

[6] Jae-Hyung Kim; Daejeon; Joon Lyou, The development of an artificial vision based navigation system for helicopter using modified NDGPS/INS integration, International Conference on Control Automation and Systems, vol. 7, pp. 2114-2118, oct,2007.

[7] Ruliang Zhang, Lin Wang, An image matching evolutionary algorithm based on Hu invariant moments, Image Analysis and Signal Processing (IASP), pp. 113-117, oct, 2011.

[8] Aibing Rao, Srihari, R.K. ; Zhongfei Zhang, Spatial color histograms for content-based image retrieval, Tools with Artificial Intelligence, Proceedings. vol. 11, pp. 183-186, nov, 1999.

[9]Imagen [En línea] Disponible: http://es.wikipedia.org/wiki/Imagen

[10] R. C. González y R. E Woods, "Introduction", en *Digital image processing,* tercera edición, Upper Saddle River, N.J.: Prentice Hall, 2009.

[11]Cámara Web [En línea] Disponible: http://es.wikipedia.org/wiki/C%C3%A1mara_web

[12] R. C. González y R. E Woods, "Color Image Processing", en *Digital image processing,* tercera edición, Upper Saddle River, N.J.: Prentice Hall, 2009.

[12] R. C. González y R. E Woods, "Image Segmentation", en *Digital image processing,* tercera edición, Upper Saddle River, N.J.: Prentice Hall, 2009.

[13] R. C. González y R. E Woods, "Object Recognition", en *Digital image processing,* tercera edición, Upper Saddle River, N.J.: Prentice Hall, 2009.

[14] Image moment [En línea] Disponible: http://en.wikipedia.org/wiki/Image_moment

[15] Raspberry Pi. [En línea]
Disponible: http://www.raspberryshop.es/hardware-raspberry-pi.php

[16] Robot con tracción diferencial [En línea] Disponible:
http://www.tamps.cinvestav.mx/~mgomez/Control_Lineal/node6.html

[17] Servidor web [En línea] Disponible: http://es.wikipedia.org/wiki/Servidor_web

[18] Wi-Fi [En línea] Disponible: http://es.wikipedia.org/wiki/Wi-Fi

[19] OpenCV, Open Source Computer Vision. [En línea]
Disponible: http://opencv.org/

[20] Lighttpd [En línea] Disponible: http://www.lighttpd.net/

10.- Apéndices

Apéndice A. Sistema embebido

Preparación de la *Raspberry Pi*

Tarjeta de desarrollo *Raspberry Pi* modelo B [9].

Procesador:	ARM1176JZF-S a 700 MHz
Memoria:	512 Mb SDRAM.
Puertos:	Dos puertos USB 2.0.
Almacenamiento:	16 Gb memoria flash SD.
Sistemas operativos compatibles:	*Debian* (*Raspbian*), *Fedora* (*Pidora*), *Arch Linux* (*Arch Linux* ARM), *Slackware Linux*, RISC OS.

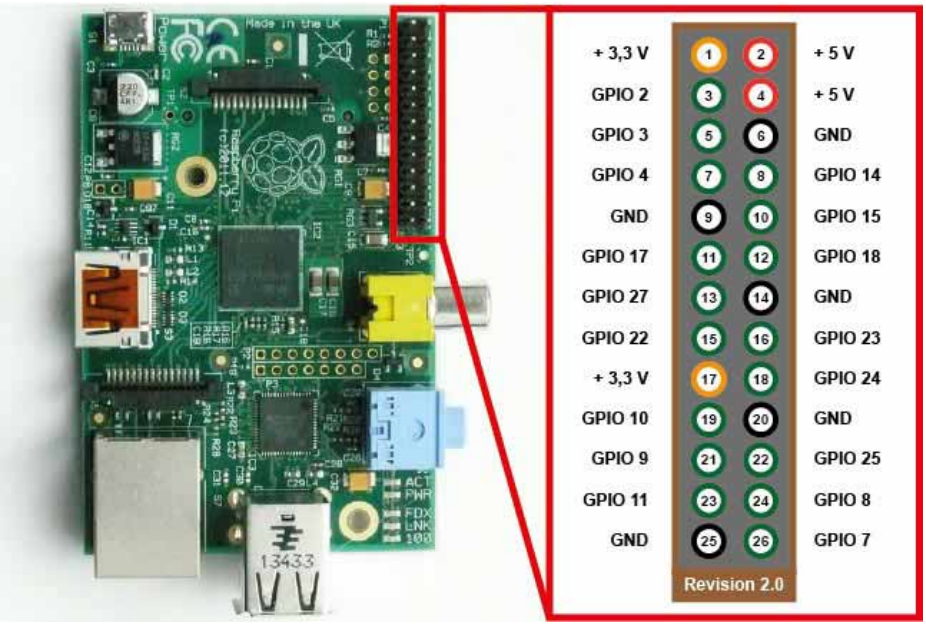

Figura A-A1 Diagrama de los puertos GPIO´s de la *Raspberry PI.*

La alimentación de corriente se lleva a cabo utilizando una batería para cargar celulares de 5V contactado mediando un cable mini USB.

El sistema de operativo *Raspbian* (*Debian Wheezy*) se instala en una memoria SD de 16 GB, siguiendo las instrucciones de la página oficial[18].

La librería *OpenCV 2.4.9* se instala siguiendo el tutorial de su página oficial[19].

El servidor *Lighttpd* se instaló siguiendo los pasos desde un tutorial en internet[20].

La conexión inalámbrica se realiza por medio del tutorial de la página oficial de *Raspberry PI*[21]. Se configura una IP fija 192.168.1.116 y con puerta de enlace 192.168.1.1.

[18] https://www.raspberrypi.org/documentation/installation/installing-images/README.md
[19] http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation
[20] http://apuntes.skamasle.com/2011/como-instalar-y-configurar-lighttpd-en-debian-linux-tutorial/
[21] https://www.raspberrypi.org/documentation/configuration/wireless/

Apéndice B. Sistema de monitoreo

El servidor *Lighttpd* atiende peticiones de página web utiliza la carpeta *"/var/www/*". En ésta ubicación se creó una carpeta llamada "*imgs*" donde se almacenan las imágenes a mostrar. Adicionalmente se creó el archivo "*index.html*" (*véase apéndice E*).

Apéndice C. Sistema de movimiento

El sistema de monitoreo se construyó utilizando los siguientes materiales:

- Dos motores con sistema reductor con llantas de plástico.
- Un circuito integrado UNL2803A
- Un circuito integrado L293D
- Dos baterías de litio de 3.6V conectadas en serie.
- Cables *Dupont*

Las conexiones se realizan en base al siguiente diagrama:

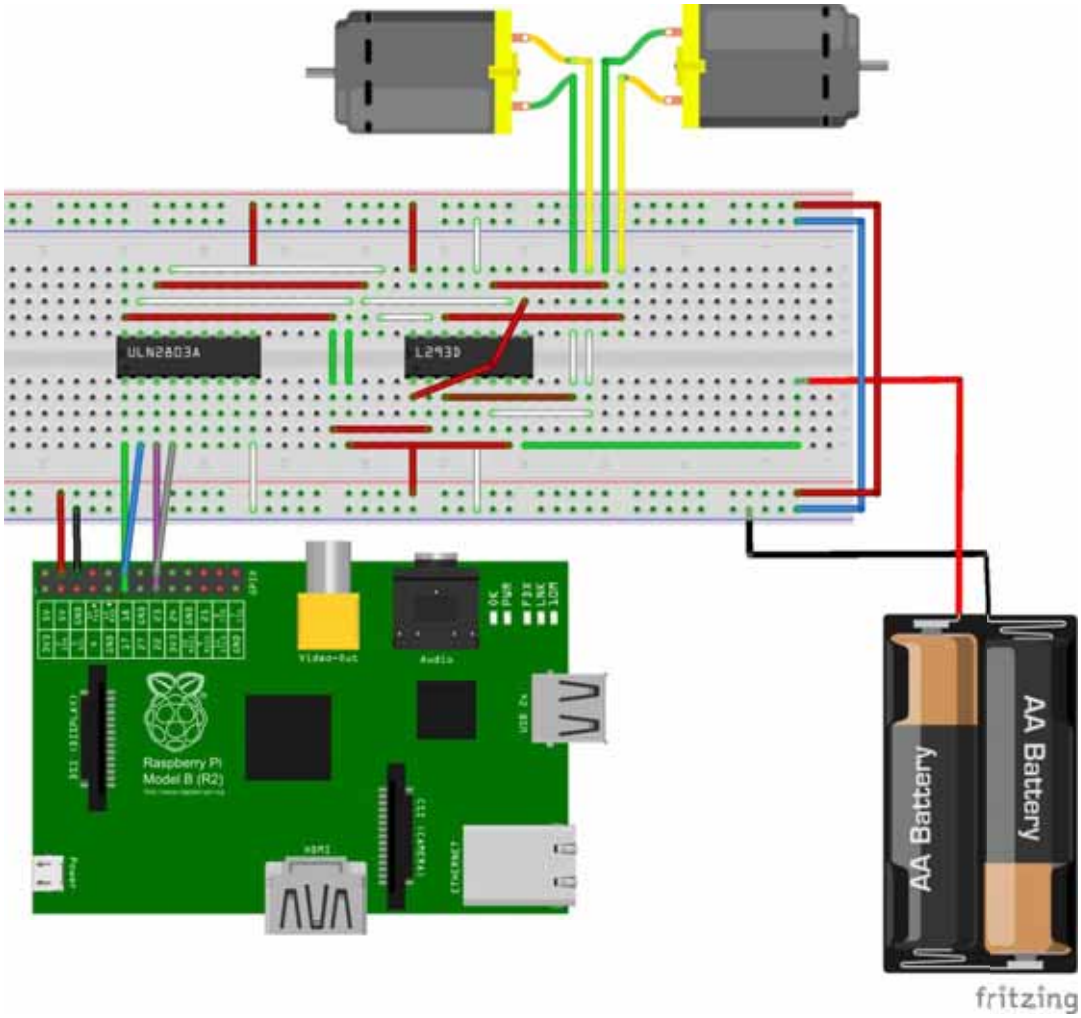

Figura A-C1 Diagrama de conexiones del sistema de movimiento.

Apéndice D. Sistema de procesamiento de imágenes.

El sistema de procesamiento de imágenes se implementó dentro la carpeta "*/home/pi/pruebas/pt2*", y consta de los siguientes archivos:

- CmakeLists.txt.- archivo de instrucciones para la compilación.
- gpio.h.- cabecera de la clase gnublin_gpio.
- gpio.cpp.- clase para el acceso a los puertos *GPIO´s* de la *Raspberry PI.*
- include.h.- cabecera de apoyo para la clase gnublin_gpio .
- main.cpp.- proyecto principal (Apéndice E.- main.cpp).

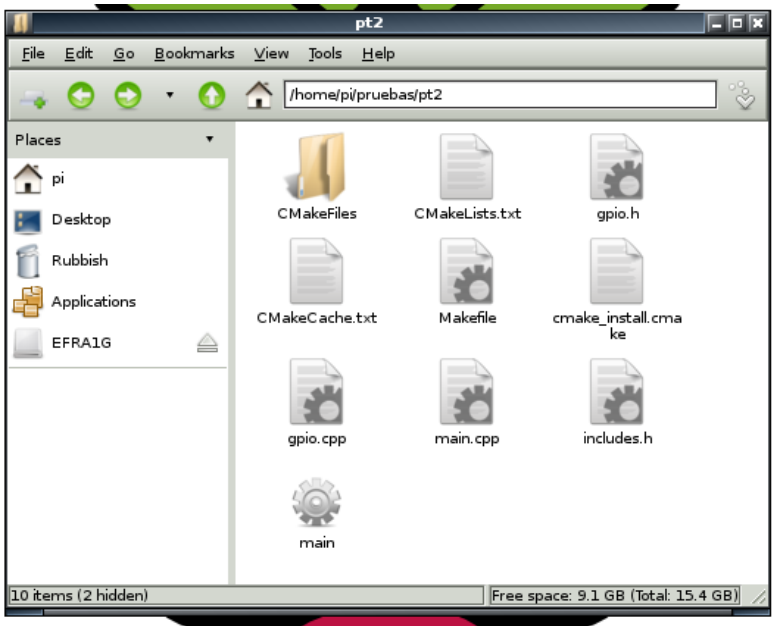

Figura A-D1 Los archivos extras se crean automáticamente al compilar el proyecto.

En el archivo CmakeLists.txt se incluyen opciones para añadir la *clase gpio*, y los paquetes de *OpenCV* y la librería *Threads* para la creación y manejo de hilos (*véase apéndice E*).

De la api "*gnublin-api*"[22] se utilizaron únicamente los archivos *gpio.h, gpio.h* e *includes.h* de la clase *gnublin_gpio*, ya que únicamente se utilizan las funciones de acceso a los puertos *GPIO´s* del *Raspberry Pi* para establecer valores lógicos *HIGH* y *LOW*.

[22] http://gnublin.org/gnublin-api/doc/html_en/classgnublin__gpio.html

La conexión remota al sistema de procesamiento de imágenes se puede realizar de dos maneras diferentes:

Por medio de la aplicación de "Escritorio Remoto", utilizando la dirección IP 192.168.1.116, el usuario "*pi*" y la contraseña "*raspberry*". Una vez cargado el ambiente gráfico, abrir una terminal para ejecutar instrucciones.

Utilizando el programa *SSH Secure Shell Client*[23] utilizando la dirección IP 192.168.1.116 con el puerto 22, el usuario "*pi*" y la contraseña "*raspberry*".

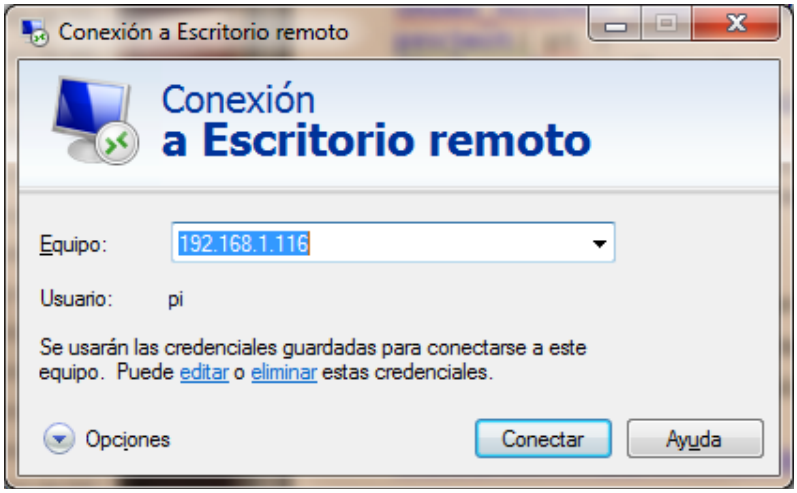

Figura A-D2 Ventana de conexion a Escritorio remoto

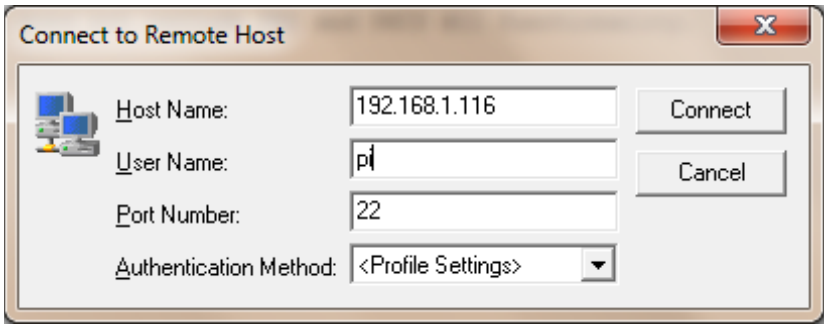

Figura A-D3 Ventana de conexión con SSH.

[23] http://sils.unc.edu/it-services/servers/using-ssh

Una vez conectado nos ubicamos en la carpeta del proyecto con la instrucción "*cd /home/pi/pruebas/pt2*".

La compilación se realiza con la instrucción "*cmake* " y posteriormente con la instrucción "*make*".

http://docs.opencv.org/doc/tutorials/introduction/linux_gcc_cmake/linux_gcc_cmake.html#linux-gcc-usage

Para ejecutar el programa se ejecuta la instrucción "*sudo ./main*".

Para terminar la ejecución se escribe la letra "*s*".

Apéndice E. Código fuente.

Index.html

```
<!doctype html>
<meta content='text/html; charset=UTF-8' http-equiv='Content-Type'/>
<html>
<head>
    <title>Sistema de Monitoreo</title>
  </head>
<body>
<h1 align="center"><strong>Muestra de resultados.</strong></h1>
<table width="100%" height="501" border="1" align="center">
  <tr>
    <th width="50%" height="37" align="center" scope="col">Original</th>
    <th width="50%" scope="col">Imagen1</th>
  </tr>
  <tr>
    <td align="center" valign="middle"><img name="imagen" id="imagen0"
alt="Imagenes"></td>
    <td align="center" valign="middle"><img name="imagen" id="imagen1"
alt="Imagenes"></td>
  </tr>
  <tr>
    <td align="center"><strong>Imagen2</strong></td>
    <td align="center"><strong>Imagen3</strong></td>
  </tr>
  <tr>
    <td align="center" valign="middle"><img name="imagen" id="imagen2"
alt="Imagenes"></td>
    <td align="center" valign="middle"><img name="imagen" id="imagen3"
alt="Imagenes"></td>
  </tr>
</table>
<p> </p>
<script type="text/javascript">
  var ruta = "./imgs/";
  var archivo0 = "img0.jpg";
  var archivo1 = "img1.jpg";
  var archivo2 = "img2.jpg";
  var archivo3 = "img3.jpg";

  //var indice = 10;
  var tiempo = 200;
  window.addEventListener('load', inicio, false);
  var recarga;
  //var imagen = document.getElementById("imagen");
    var imagen0 = document.getElementById("imagen0");
    var imagen1 = document.getElementById("imagen1");
    var imagen2 = document.getElementById("imagen2");
    var imagen3 = document.getElementById("imagen3");

  function inicio() {
```

```
    recarga = setInterval(procesar,tiempo);
  }

  function procesar()
  {
    //document.imagen.src = "./imagenes/01.jpg";
    imagen0.src = ""+ruta+archivo0+ "?d=" + new Date().getTime();
    imagen1.src = ""+ruta+archivo1+ "?d=" + new Date().getTime();
    imagen2.src = ""+ruta+archivo2+ "?d=" + new Date().getTime();
    imagen3.src = ""+ruta+archivo3+ "?d=" + new Date().getTime();
    //indice == 99 ? indice = 10 : indice++;
  }

</script>
</body>
</html>
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project( pt )
find_package( Threads )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( main main.cpp gpio.cpp)
target_link_libraries( main ${OpenCV_LIBS} ${CMAKE_THREAD_LIBS_INIT} )
```

main.cpp

```
// archivo main.cpp
// Efraín Ernesto Arévalo Vázquez. 206200579

#define ROBOT 1

#if ROBOT == 1
    #define BOARD RASPBERRY_PI
    #include "gpio.h" // librería para acceder a los puertos GPIO    #include
<pthread.h>
#endif

#include <opencv2/opencv.hpp>
#include <string>

using namespace std;
using namespace cv;

#if ROBOT == 1

bool vas = true;
// Hilo creado para terminar el programa principal.
void *EsperaTecla(void *threadid)
{
   long tid;
   tid = (long)threadid;
    char c;
    cout<<"hilo iniciado."<<endl;
    while(vas)
    {
        cin>>c;
        if (c=='s')
            vas = false;
    }
    cout<<"adios hilo"<<endl;
   pthread_exit(NULL);
}

// ----- inicia globales Robot -----
// funciones para mover el robot
void robotInicializar();
void robotFinalizar();
void robotAdelante();
void robotAtras();
void robotDerecha();
void robotIzquierda();
void robotAlto();

// pines GPIO Raspberry B Rev2
const int numa = 17; //GPIO17 PIN 11
const int numb = 18; //GPIO18 PIN 12
const int numc = 22; //GPIO22 PIN 15
const int numd = 23; //GPIO23 PIN 16
const int tiempo = 10000; // 20 milisegundos
gnublin_gpio gpio;
// ----- termina globales Robot -----
```

```
#endif

// ----- inicia globales constantes -----
const int FRAME_WIDTH  = 320; //640
const int FRAME_HEIGHT = 240; //360
const int FRAME_AREA   = FRAME_WIDTH * FRAME_HEIGHT;
const int MIN_AREA     = 300;
const int MAX_AREA     = FRAME_AREA * 1/6;

//max number of objects to be detected in frame
const int MAX_NUM_OBJECTS=200;
//minimum and maximum object area
const int MIN_OBJECT_AREA = 20*20;
const int MAX_OBJECT_AREA = FRAME_HEIGHT*FRAME_WIDTH/1.5;

const Scalar color_verde = Scalar(0,255,0);
const Scalar color_rojo  = Scalar(0,0,255);
const Scalar color_cyan  = Scalar(255,255,0);

const string V_ORIGINAL = "Original";
const string V_GRIS     = "Gris";
const string V_GAUSS    = "GaussianBlur";
const string V_CANNY    = "Canny";
const string V_CUADROS  = "Cuadros";

const int MARCA_WIDTH  = 25; //
const int MARCA_HEIGHT = 25; //
const int DELTA_MIN = 30; // rango hsv
const int DELTA_MAX = 30;

const float hranges[] = {0,180};
const float* phranges = hranges;
const int hsize = 180;
const int ENTRENAMIENTOS = 20;
// ----- termina globales constantes -----

// ----- inicia valores iniciales -----
// estados
bool activaRectangulo = false;
bool buscandoMarca = true;
bool entrenandoHistograma = true;
bool detectaCyF = false;
bool guardaContorno = false;
bool objectFound = false;
bool contornoGuardado = false;
bool buscandoObjeto = false;
bool buscaObjeto = false;

int entrena = 0;
int nada = 0;

int objX = 0; // modificados por
int objY = 0; // cont_sombra()

Rect selection;
Scalar hsvMin = Scalar(0,0,0);
Scalar hsvMax = Scalar(255,255,255);
```

```
double sumas[] = {0,0,0,0,0,0};
// ----- termina valores iniciales -----

// ----- inicia imagenes globales -----
Mat hue, hist;
// ----- termina imagenes globales -----

#if ROBOT == 1
void robotInicializar()
{
    //gnublin_gpio gpio;
    // puertos en modo Salida
    gpio.pinMode(numa,OUTPUT);
    gpio.pinMode(numb,OUTPUT);
    gpio.pinMode(numc,OUTPUT);
    gpio.pinMode(numd,OUTPUT);

    robotAlto();
}
void robotFinalizar()
{
    robotAlto();
    gpio.unexport(numa);
    gpio.unexport(numb);
    gpio.unexport(numc);
    gpio.unexport(numd);
}
void robotAdelante()
{
    gpio.digitalWrite(numa,LOW);
    gpio.digitalWrite(numb,HIGH);
    gpio.digitalWrite(numc,HIGH);
    gpio.digitalWrite(numd,LOW);
    usleep(tiempo);
    robotAlto();
}
void robotAtras()
{
    gpio.digitalWrite(numa,HIGH);
    gpio.digitalWrite(numb,LOW);
    gpio.digitalWrite(numc,LOW);
    gpio.digitalWrite(numd,HIGH);
    usleep(tiempo);
    robotAlto();
}
void robotDerecha()
{
    gpio.digitalWrite(numa,LOW);
    gpio.digitalWrite(numb,HIGH);
    gpio.digitalWrite(numc,LOW);
    gpio.digitalWrite(numd,HIGH);
    usleep(tiempo);
    robotAlto();
}
void robotIzquierda()
{
    gpio.digitalWrite(numa,HIGH);
```

```
    gpio.digitalWrite(numb,LOW);
    gpio.digitalWrite(numc,HIGH);
    gpio.digitalWrite(numd,LOW);
    usleep(tiempo);
    robotAlto();
}
void robotAlto()
{
    gpio.digitalWrite(numa,HIGH);
    gpio.digitalWrite(numb,HIGH);
    gpio.digitalWrite(numc,HIGH);
    gpio.digitalWrite(numd,HIGH);
    //usleep(tiempo/2);
}
#endif

string intToString(int number)
{
    std::stringstream ss;
    ss << number;
    return ss.str();
}

void drawObject3(Mat &frame, int valor)
{
    int x = objX;
    int y = objY;
    Point punto = Point(x,y);

    circle(frame,punto,20,color_verde,1);
    if(y-25>0)
        line(frame,punto,Point(x,y-25),color_verde,1);
    else line(frame,punto,Point(x,0),color_verde,1);
    if(y+25<FRAME_HEIGHT)
        line(frame,punto,Point(x,y+25),color_verde,1);
    else line(frame,punto,Point(x,FRAME_HEIGHT),color_verde,1);
    if(x-25>0)
        line(frame,punto,Point(x-25,y),color_verde,1);
    else line(frame,punto,Point(0,y),color_verde,1);
    if(x+25<FRAME_WIDTH)
        line(frame,punto,Point(x+25,y),color_verde,1);
    else line(frame,punto,Point(FRAME_WIDTH,y),color_verde,1);

    putText(frame,intToString(valor),Point(x,y-35),1,1,color_rojo,1);
}

static double angle(Point pt1, Point pt2, Point pt0)
{
    double dx1 = pt1.x - pt0.x;
    double dy1 = pt1.y - pt0.y;
    double dx2 = pt2.x - pt0.x;
    double dy2 = pt2.y - pt0.y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) +
1e-10);
}

static void findSquares( const Mat& image, vector<vector<Point> >& squares )
```

```
{
    vector<vector<Point> > contours;
    vector<Point> approx;
    int area=0;

    squares.clear();
    findContours(image, contours, CV_RETR_LIST, CV_CHAIN_APPROX_SIMPLE);
    approx.clear();
    int cont_size = contours.size();
    for( size_t i = 0; i < cont_size; i++ )
    {
        approxPolyDP(Mat(contours[i]), approx, 5, true);
        area = contourArea(Mat(approx));
        if( approx.size() == 4 && isContourConvex(Mat(approx)) &&
                MIN_AREA < area && area < MAX_AREA )
        {
            double maxCosine = 0;
            for( int j = 2; j < 5; j++ )
            {
                double cosine = fabs(angle(approx[j%4], approx[j-2],
approx[j-1]));
                maxCosine = MAX(maxCosine, cosine);
            }
            if( maxCosine < 0.2 )
                squares.push_back(approx);
        }
    }
}

static void drawSquares( Mat& image, const vector<vector<Point> >& squares )
{
    int tam = squares.size();
    for( size_t i = 0; i < tam; i++ )
    {
        const Point* p = &squares[i][0];
        int n = (int)squares[i].size();
        polylines(image, &p, &n, 1, true, color_verde, 1, CV_AA);
    }
}

Point pmedio(vector <Point> &c){
    Point p,p1,p2;

    p1.x = c[0].x + (c[2].x - c[0].x)/2;
    p1.y = c[0].y + (c[2].y - c[0].y)/2;
    p2.x = c[1].x + (c[3].x - c[1].x)/2;
    p2.y = c[1].y + (c[3].y - c[1].y)/2;

    p.x = p1.x + (p2.x - p1.x)/2;
    p.y = p1.y + (p2.y - p1.y)/2;

    return p;
}

void centroCuadros(const vector<vector<Point> >& squares, vector <Point>
&centros)
{
```

```
    int tam = squares.size();

    if(tam == 0)
        return;

    int tam2 = squares[0].size();
    Point cen;
    vector <Point> cuadro;

    centros.clear();
    for( size_t i = 0; i < tam; i++ )
    {
        cuadro.clear();
        for( size_t j = 0; j < tam2; j++ )
        {
            cuadro.push_back(squares[i][j]);
        }
        cen = pmedio(cuadro);
        centros.push_back(cen);
    }

    int dx,dy;

    for( size_t i = 0; i < centros.size() -1; i++ ){
        for( size_t j = i+1 ; j < centros.size(); j++ ){
            dx = abs(centros[i].x - centros[j].x);
            dy = abs(centros[i].y - centros[j].y);
            if (dx < 5 && dy < 5){
                centros.erase(centros.begin()+j);
                j--;
            }
        }
    }
}

bool movimientos(Mat &frame,int puntoX, double acercar){

    int centrarX = puntoX - FRAME_WIDTH/2;

    if(centrarX<-10){
        #if ROBOT == 1
        robotIzquierda(); // Robot
        #endif
        putText(frame,"<--",Point(150,120),1,1,color_verde,2);
        return false;
    }
    else if (10 < centrarX){
        #if ROBOT == 1
        robotDerecha(); // Robot
        #endif
        putText(frame,"-->",Point(150,120),1,1,color_verde,2);
        return false;
    }
    else{
        if (acercar<0.73){
            #if ROBOT == 1
            robotAdelante(); // Robot
```

```
            robotAdelante(); // Robot
            #endif
            putText(frame," /\\",Point(150,120),1,1,color_verde,2);
            return false;
        }
        else{
            return true;
        }
    }
}

void drawObject(Mat &frame,vector <Point> &centros)
{
    int cent[2];
    int cindx = 0;
    double prop;
    int centromarca;

    int mx,my;

    int cent_size = centros.size();
    for( size_t i = 0; i < cent_size; i++ )
    {
        int x = centros[i].x;
        int y = centros[i].y;
        if (y > FRAME_HEIGHT/2){
            cent[cindx] = i;
            cindx++;
        }
        circle(frame,Point(x,y),8,color_rojo,1);
        line(frame,Point(x-10,y),Point(x+10,y),color_rojo,1);
        line(frame,Point(x,y-10),Point(x,y+10),color_rojo,1);
    }
    putText(frame,"Marca encontrada",Point(80,15),1,1,color_verde,2);
    // crea coordenadas de Seleccion
    my = centros[cent[0]].y + (centros[cent[1]].y - centros[cent[0]].y)/2;
    mx = centros[cent[0]].x + (centros[cent[1]].x - centros[cent[0]].x)/2;

    prop = double(abs(centros[cent[1]].x -
centros[cent[0]].x))/double(FRAME_WIDTH);

    if (movimientos(frame, mx,prop)){
        activaRectangulo = true;
        buscandoMarca = false;
        selection = Rect(mx-MARCA_WIDTH/2, my - MARCA_HEIGHT/2,
MARCA_WIDTH,MARCA_HEIGHT);
    }
    else{
        activaRectangulo = false;
    }
}

void buscaMarca (Mat frame, Mat &cuadros){
    Mat  gray, gaussian, canny, dilate;
    vector<vector<Point> > squares;
    vector <Point> centros;
```

```
    cvtColor(frame, gray, CV_BGR2GRAY);
    pyrDown(gray, dilate, Size(frame.cols/2, frame.rows/2));
    pyrUp(dilate, dilate, frame.size());

    GaussianBlur(dilate, gaussian, Size(9,9), 1.5, 1.5);
    Canny(gaussian, canny, 0, 30, 3);

    // ---------------------- cuadros
    findSquares(canny, squares);
    drawSquares(cuadros, squares);
    centroCuadros(squares,centros);
    if((int)centros.size() == 4){
        activaRectangulo = true;
        drawObject(cuadros,centros);
    }

}
void minMaxHSV(Mat & m, Scalar &smi, Scalar &sma){
    Mat corteHSV;
    cvtColor(m, corteHSV, COLOR_BGR2HSV);

        float minh=300, mins=300, minv=300, maxh=0, maxs=0, maxv=0;
        Vec3b  s;
        float _h,_s,_v;

        for( int i = 0; i < MARCA_WIDTH ; i++ )
        {
            for( int j = 0; j < MARCA_HEIGHT; j++ )
            {
                s  = corteHSV.at<Vec3b>(i, j);
                _h = s.val[0];
                _s = s.val[1];
                _v = s.val[2];
                if(_h < minh)
                    minh= _h;
                if(_s < mins)
                    mins= _s;
                if(_v < minv)
                    minv= _v;
                if(_h > maxh)
                    maxh= _h;
                if(_s > maxs)
                    maxs= _s;
                if(_v > maxv)
                    maxv= _v;
            }
        }
        // mantiene dentro del rango permitido de 0 a 255
        minh = (0 >= minh-DELTA_MIN)? 0 : minh - DELTA_MIN;
        mins = (0 >= mins-DELTA_MIN)? 0 : mins - DELTA_MIN;
        minv = (0 >= minv-DELTA_MIN)? 0 : minv - DELTA_MIN;

        maxh = (255 <= maxh+DELTA_MAX)? 255 : maxh + DELTA_MAX;
        maxs = (255 <= maxs+DELTA_MAX)? 255 : maxs + DELTA_MAX;
        maxv = (255 <= maxv+DELTA_MAX)? 255 : maxv + DELTA_MAX;

        smi = Scalar(minh, mins, minv);
```

```
        sma = Scalar(maxh, maxs, maxv);
}

void promedioHSV(){
    for (int i=0; i<3;i++){
        sumas[i]+= hsvMin.val[i];
        sumas[i+3]+= hsvMax.val[i];
    }

    for (int i=0; i<3;i++){
        hsvMin.val[i] = sumas[i]/ entrena;
        hsvMax.val[i] = sumas[i+3]/ entrena;
    }
}

void detectaColorYforma(Mat frame, Mat &cuadros){

    Mat temp, corte,hsv;
    Mat roi,maskroi,mask, backproj;

    Mat histimg = Mat::zeros(FRAME_HEIGHT, FRAME_WIDTH, CV_8UC3);

    frame.copyTo(temp);
    pyrDown(temp, temp, Size(frame.cols/2, frame.rows/2));
    pyrUp(temp, temp, frame.size());
    corte = Mat(temp, selection);

    // obtener valores min y max hsv
    if(entrenandoHistograma)
        minMaxHSV(corte,hsvMin,hsvMax);
    //----------------------

    cvtColor(temp, hsv, COLOR_BGR2HSV);

    inRange(hsv, hsvMin, hsvMax, mask);
    int ch[] = {0, 0};
    hue.create(hsv.size(), hsv.depth());
    mixChannels(&hsv, 1, &hue, 1, ch, 1);

    if(entrenandoHistograma)
    {
        roi = Mat(hue, selection);
        maskroi= Mat(mask, selection);
        calcHist(&roi, 1, ch, maskroi, hist, 1, &hsize, &phranges);
        normalize(hist, hist, 0, 255, CV_MINMAX);

        histimg = Scalar::all(0);
        int binW = histimg.cols / hsize;
        Mat buf(1, hsize, CV_8UC3);
        for( int i = 0; i < hsize; i++ )
            buf.at<Vec3b>(i) = Vec3b(saturate_cast<uchar>(i*180./hsize), 255,
255);

        cvtColor(buf, buf, CV_HSV2BGR);

        for( int i = 0; i < hsize; i++ )
        {
```

```
            int val = saturate_cast<int>(hist.at<float>(i)*histimg.rows/255);
            rectangle( histimg, Point(i*binW,histimg.rows),
                    Point((i+1)*binW,histimg.rows - val),
                    Scalar(buf.at<Vec3b>(i)), -1, 8 );
        }
    }

    calcBackProject(&hue, 1, ch, hist, backproj, &phranges);
    threshold(backproj, cuadros, 5, 255, THRESH_BINARY);
    cuadros &= mask;

    if (entrena < ENTRENAMIENTOS){
        entrenandoHistograma = true;
        entrena++;
        promedioHSV();
        cout<<"Entrenando "<<entrena<<endl;
        if( !histimg.empty() ){
            //imshow("Histograma", histimg );
            imwrite("/var/www/imgs/img2.jpg", histimg);
        }
    }
    else{
        //entrena = ENTRENAMIENTOS;
        entrenandoHistograma = false;
    }
}


void cont_sombra(Mat threshold, vector<Point> &contorno)
{
    contorno.clear();
    int x, y;

    Mat temp;
    threshold.copyTo(temp);
    //these two vectors needed for output of findContours
    vector< vector<Point> > contours;
    vector<Vec4i> hierarchy;
    //find contours of filtered image using openCV findContours function
    findContours(temp,contours,hierarchy,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPLE
);
    //use moments method to find our filtered object
    double refArea = 0;
    //bool objectFound = false;
    objectFound = false;

    //cout<<hierarchy.size()<<endl;

    if (hierarchy.size() > 0)
    {
        int numObjects = hierarchy.size();
        //if number of objects greater than MAX_NUM_OBJECTS we have a noisy
filter
        if(numObjects<MAX_NUM_OBJECTS)
        {
            for (int index = 0; index >= 0; index = hierarchy[index][0])
            {
```

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

                double area = moment.m00;

                if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA &&
area>refArea)
                {
                    x = moment.m10/area;
                    y = moment.m01/area;
                    objX = x;
                    objY = y;
                    objectFound = true;
                    refArea = area;
                    contorno = contours[index];
                }
                else objectFound = false;
            }
        }
    }
}

void reset(){

    activaRectangulo = false;
    buscandoMarca = true;
    entrenandoHistograma = true;
    detectaCyF = false;
    guardaContorno = false;
    objectFound = false;
    contornoGuardado = false;
    buscandoObjeto = false;
    buscaObjeto = false;

    entrena = 0;

    objX = 0; // modificados por
    objY = 0; // cont_sombra()

    //Rect selection;
    hsvMin = Scalar(0,0,0);
    hsvMax = Scalar(255,255,255);
    nada = 0;

    for(int i=0; i<6 ; i++){
        sumas[i] = 0;
    }
    cout<<"Reseteado..."<<endl;
}

int main()
{
    cout<<"programa iniciado"<<endl;
    #if ROBOT == 1
    robotInicializar(); // inicia los puertos
    #endif
    cout<<"programa iniciado"<<endl;
```

```
pthread_t threads;
int rc;
long t=0;
rc = pthread_create(&threads, NULL, EsperaTecla, (void *)t);
if (rc){
   cout<<"ERROR; return code from pthread_create() is "<<rc<<endl;
   return -1;
}

// variables de im¶nes
Mat frame, resultado, resultado2;
vector<Point>  contours1 ,contours2;

// inicia cámara
VideoCapture cap; // open the default camera
cap.open(0);
waitKey(1000);
if(!cap.isOpened())  // check if we succeeded
    return -1;
waitKey(1000);
cap.set(CV_CAP_PROP_FRAME_WIDTH ,FRAME_WIDTH );
cap.set(CV_CAP_PROP_FRAME_HEIGHT,FRAME_HEIGHT);
cap >> frame; // get a new frame from camera
// finaliza cámara


// inicia ciclo infinito
//for(;;)
while(vas)
{
    // inicia fase de captura
    cap >> frame; // get a new frame from camera
    if( frame.empty() )
    {
        cout << "Couldn't load " << endl;
        continue;
    }
    // finaliza fase de captura

    frame.copyTo(resultado);
    frame.copyTo(resultado2);

    // busca marca y se ala
    if (buscandoMarca){
        buscaMarca(frame,resultado);
    }

    // empieza la fase de aprendizaje
    if (activaRectangulo){
        rectangle(resultado, selection, color_cyan);
        detectaCyF = true;
    }
    // fase de aprendizaje
    if (detectaCyF){
        detectaColorYforma(frame, resultado2);
        if(!entrenandoHistograma&&!contornoGuardado){
```

```
            //morphOps(resultado2);
            guardaContorno = true;
        }
    }
    // guarda contorno
    if (guardaContorno){
        cont_sombra(resultado2,contours1);
        if (objectFound){
            cout<<"Contorno guardado"<<endl;
            guardaContorno = false;
            contornoGuardado = true;
            //imshow("contorno guardado",resultado2);
            imwrite("/var/www/imgs/img3.jpg", resultado2);
            buscandoObjeto= true;
        }
        else{
            cout<<"Nada.."<<endl;
            nada++;
            if (nada == 10){
                #if ROBOT == 1
                for(int i=0; i < 20; i++)
                    robotAtras(); // Robot
                #endif
                resultado2 = Scalar::all(0);
                reset();

            }
        }
    }

    // empieza a buscar el objeto
    // 1 empieza a girar hasta buscar el objeto

    if (buscandoObjeto){

        activaRectangulo = false;
        buscandoObjeto = false;
        detectaCyF = false;
        buscaObjeto = true;
        cout<<"Girando.."<<endl;
        //waitKey(5000);
        #if ROBOT == 1
        for(int i=0; i < 50; i++)
            robotDerecha(); // Robot
        #endif
        cout<<"buscando.."<<endl;

        resultado = Scalar::all(0);

    }

    double compara= -1;
    if(buscaObjeto){
        int dist;
        detectaColorYforma(frame, resultado2);
        cont_sombra(resultado2,contours2);
        if (0<contours2.size()&& contours2.size()< 1000){
```

```
                double compara;
                compara = matchShapes(contours1,contours2,
CV_CONTOURS_MATCH_I1,0);

                if(compara < 10){
                    dist = (int) (compara*100);
                    drawObject3(resultado,dist);
                    if (dist < 50){
                        movimientos(resultado,objX,1);
                    }
                }
            }
            else{
                // si no encuentra candidatos se mueve
                //cout<<"Buscando similar..."<<endl;
                #if ROBOT == 1
                robotDerecha();
                robotDerecha();
                #endif
            }
        }

        // mostrar resultados

        //imshow(V_CANNY, resultado2);
        imwrite("/var/www/imgs/img1.jpg", resultado2);

        //imshow(V_CUADROS, resultado);
        imwrite("/var/www/imgs/img0.jpg", resultado);

        // -------
        #if ROBOT == 1
        usleep(150000);
        if(waitKey(1) >= 0) break;
        #else
        if(waitKey(150) >= 0) break;
        #endif

    }
    #if ROBOT == 1
    robotFinalizar();
    sleep(2);

    /* Last thing that main() should do */
    pthread_exit(NULL);
    #endif
    return 0;
}
```