Universidad
Carlos III de Madrid

Departamento de Ingeniería de sistemas y automática

PROYECTO FIN DE CARRERA

SISTEMA DE BAJO COSTE PARA EL SEGUIMIENTO DE PERSONAS

Autor: Rebeca Susana García Gallego

Tutor: Santiago Martínez de la Casa Díaz

Leganés, octubre de 2015

Título: SISTEMAS DE BAJO COSTE PARA EL SEGUIMIENTO DE PERSONAS

Autor: Rebeca Susana García Gallego

Director: Santiago Martínez de la Casa Díaz

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 28 de octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

Agradecer en primer lugar a Santiago Martínez de la Casa Díaz mi tutor por su infinita paciencia y perseverancia. Si no hubiera contado con tu ayuda y tu apoyo no lo habría conseguido.

En segundo lugar citar a mis padres por su incondicional apoyo en todo los proyectos que emprendo en la vida, nunca os podre agradecer suficientemente todo lo que habéis hecho y seguís haciendo por mí.

En tercer lugar me gustaría citar a mis hermanos, porque gracias a ellos soy quien soy y me siento orgullosa de la familia que tengo.

En cuarto lugar me gustaría citar a mis amigos. Por un lado a mis amigos Alberto, Pablo, Pilar y Marta, porque pese a todo siempre que les he necesitado han estado ahí, y espero seguir cuidándoles y que sigan a mi lado mucho tiempo. A mis 'Charlys' porque somos mucho más que un equipo de baloncesto de ex-estudiantes de la Universidad Carlos III. Dentro de mis charlys quiero hacer mención especial a mis 'Calcetas' que sin su apoyo el proyecto PP no habrías salido adelante, gracias Paus, muchas gracias Marta. Y por último citar a mi Gallega favorita, ya que pese a la distancia siempre está a mi lado.

Y por último quisiera hacer mención especial a todos esos que me han sufrido a lo largo de todos estos años, compañeros de trabajo, excompañeros de trabajo, compañeros de baloncesto, de todo se aprende en esta vida y parecía que este día no llegaría pero aquí estoy escribiendo mis agradecimientos.

Simplemente muchas gracias.

Resumen

Gracias a una Raspberry Pi, a la Pi-cam y a pocos componentes más, vamos a diseñar un sistema de seguimiento de objetos de bajo coste.

Para ello hemos diseñado varias aplicaciones que desempeñan tareas diferentes dentro de las que destacamos la siguientes: identificación de objetos en imágenes; detección y señalización de estos objetos; también hemos diseñado una aplicación que calcula el tamaño y la posición de nuestro objeto gracias a la cual sabremos si este se ha movido o no y gracias a la cual podremos mover nuestro sistema para posicionarnos en el lugar deseado.

Veremos como con un presupuesto muy reducido seremos capaces de desarrollar un trabajo bastante completo.

Hemos dividido el trabajo en dos tareas fundamentales, la localización del objeto deseado; la cual la fundamentaremos en la visión por computador. Y en un segundo lugar el movimiento del sistema; el cual gracias a la GPIO y un sistema actuador, seremos capaces de diseñar teóricamente.

En los capítulos siguientes veremos como con un sistema tan versátil como es una Raspberry Pi se pueden lograr infinidad de objetivos en cuanto a visión por computador; y de igual modo su pines de entrada/salida GPIO nos versatilizan este dispositivo dándonos muchísimas opciones de diseño y desarrollo.

Palabras clave: Raspberry Pi, visión por computador, movimiento.

Abstract

Using a Raspberry Pi, a Pi-cam and a few more components, we are going to design a low cost tracking system.

For that matter, we have designed several applications that play different tasks in which we highlight the following: object identification in pictures, detection and pointing out these objects. We also have designed an application that calculates the size and the position of our object and indicate if this object has moved or not so we can move our system to positioning in the desired place.

We will see how with a very little budget, we will be able to develop a quite complete work.

We have divided the work in two main tasks, the localization of the desired object, which we base in the computer sight. And in second place, the system movement, which thanks to the GPIO and a acting system, we will be able to theoretically design.

In the next chapters we will see how a so versatile system as the Raspberry Pi can achieve tons of goals in computer sighting, and similarly its pins in/out GPIO make this device versatile, offering us a lot of options of design and development.

Keywords: Raspberry Pi, computer vision, movement.

Índice

1. INDICE DE FIGURAS

2. INDICE DE TABLAS

3. INTRODUCCIÓN Y OBJETIVOS

3.1. INTRODUCCIÓN

Una de las temáticas clásicas del área de conocimiento de Ingeniería de Sistemas y Automática es la robótica.

El mundo de la robótica está experimentando un crecimiento exponencial impulsado por los avances tecnológicos en computación, sensores, electrónica, comunicaciones y software.

Los robots son los causantes de revolucionar los procedimientos que se emplean en la agricultura, minería, e industria en general.

Dentro de la robótica tenemos infinidad de ramas y posibilidades, y por ejemplo tenemos el campo de los robots manipuladores, el cual ha experimentado un alto desarrollo desde la década de los setenta. También tenemos la denominada robótica móvil, que ha cobrado una importancia creciente durante los años ochenta y noventa.

Tanto en los robots manipuladores como en la robótica móvil existen puntos de interés común: el modelado cinemático, el modelado dinámico, el control (arquitecturas, algoritmos…), la planificación, el reconocimiento del entorno, etc.

En nuestro caso hemos querido ser un poco ambiciosos ya que vamos a desarrollar un sistema de seguimiento de personas utilizando una Raspberry Pi, por lo que nuestro sistema será considerado de bajo coste. En este proyecto aunaremos modelado cinemático junto reconocimiento del entorno y de objetos. La aplicación a desarrollar formará parte de un sistema de seguimiento de personas mediante la utilización de identificación de patrones predefinidos que el individuo a seguir incorporará.

3.2. OBJETIVOS

El objetivo de este proyecto es la implementación de una aplicación para el desarrollo de un sistema de seguimiento de personas de bajo coste. A día de hoy estamos rodeados de tecnología por todas partes y tenemos a nuestra disposición infinidad de posibilidades. Por un precio muy reducido podemos disponer de dispositivos tan potentes a la vez que versátiles como es la Raspberry Pi. Con ella vamos a ser capaces de desarrollar un trabajo bastante completo el cual podemos subdividir en dos tareas básicas.

Por un lado nos centraremos en un primer lugar en la localización del objeto que deseamos seguir. Esta tarea la realizaremos gracias a la visión por computador. El mundo de la visión por computador es un campo muy amplio, y nos aporta infinidad de posibilidades.

El objeto a detectar lo hemos acotado dentro de un patrón dado según una forma o un color determinado. Esto nos facilitará la labor de detección del objeto a seguir.

Una vez tengamos el objeto detectado y adecuadamente posicionado, el segundo objetivo que vamos a completar es el que hemos denominado como 'movimiento del vehículo'. En este apartado implementaremos una aplicación básica que hará que nuestro sistema de bajo coste se dirija hacia la posición del objeto detectado previamente. Para ello diseñaremos un vehículo con unas características determinadas y calcularemos teóricamente las trayectorias que ha de seguir y como queremos que se comporte en cada momento en función de la información que estemos detectando.

Debido a que nuestro sistema es de bajo coste y disponemos de los elementos mínimos no seremos capaces de medir la distancia exacta a la que nos encontramos del objeto detectado. Pero por contrapunto si somos capaces de detectar el objeto y saber el tamaño que tiene. Gracias a una calibración previa sabremos que cuando el objeto tiene unas dimensiones dadas está a una distancia x del sistema, por lo que si deseamos acercarnos debemos conseguir que el tamaño del objeto detectado sea mayor al tamaño dado en las calibraciones; y si por el contrario nos queremos alejar, lo que debemos conseguir es que las medidas obtenidas del objeto detectado disminuya.

Entraremos en todos estos puntos más en detalle en los capítulos posteriores.

4. ESTADO DEL ARTE

4.1. VISIÓN POR COMPUTADOR

¿Qué es la visión por computador? Es muy importante tener claro el campo en el que nos adentramos ya que es la base del desarrollo de nuestro proyecto.

La visión por computador es un campo de la inteligencia artificial enfocado a que las computadoras puedan extraer información a partir de imágenes, ofreciendo así soluciones a problemas del mundo real. Podríamos decir que sería enseñar a ver a las computadoras.

Los objetivos básicos de la visión por computador son:

- La detección, segmentación, localización y reconocimiento de ciertos objetos.
- La evaluación de los resultados.
- Seguimiento de un objeto en una secuencia de imágenes.
- Mapeo de una escena para para generar un modelo tridimensional de la escena.

Todos estos objetivos se consiguen por medio de reconocimiento de patrones, aprendizaje estático, geometría de proyección, procesamiento de imágenes, teoría de grafos y otros campos.

4.1.1. HISTORIA DE ADQUISICIÓN Y REPRESENTACIÓN DE LA IMAGEN

Una cámara produce imágenes en dos dimensiones de un mundo físico percibido como tridimensional. Antes de que la fotografía llegara a nuestro mundo existía un gran interés por la representación de este mundo tridimensional de ahí las pinturas.

En la antigua Grecia ya se estudiaron las propiedades geométricas de la proyección, como es el caso de Thales de Mileto (filósofo, matemático, geómetra, físico y legislador griego - fue considerado uno de los 7 Sabios de Grecia); con sus conocimientos de la geometría pudo predecir un eclipse solar y también pudo medir la altura de una pirámide a partir de su sombra proyectada. Sin duda el griego más famoso es el matemático Euclides quien en el siglo IV AC ideó la geometría plana.

Posteriormente, los pintores italianos del Renacimiento fueron los primeros en entender la formación de las imágenes y fueron los primeros en estudiar la geometría para así poder reproducir correctamente los efectos de la perspectiva en las imágenes del mundo que observaban. La pintura anterior a esta época era plana y no mostraba la diferencia de profundidad en los objetos representados.

La perspectiva fue inventada por Filippo Brunelleschi (1377-1446) alrededor de 1413(gran arquitecto del Renacimiento Temprano). Fue también escultor y pintor. Sus principales obras se encuentran en Florencia, como por ejemplo la Catedral Santa María de Fiore, cuya cúpula es la más grande del mundo con más de 50m de diámetro. Artistas como Piero della Francesca (1415-1492), Leonardo da Vinci (1452-1519) y Albrecht DÄurer (1471-1528), los dos primeros italianos y el tercero alemán que viaja a Italia para posteriormente llevar el Renacimiento a Alemania, realizan serios estudios geométricos que se usan hasta el día de hoy. A partir de esta época se empieza a considerar el punto de fuga, en el que líneas paralelas que se alejan del observador convergen en un punto.
La teoría de la perspectiva se desarrolla en el siglo XVI, llegando incluso a la invención de Máquinas de Perspectiva como apoyo a los pintores para facilitar la reproducción de la perspectiva sin necesidad de cálculos matemáticos. Puede verse representada una de estas máquinas por Albrecht Dürer en la Figura - 1 La idea tras esta máquina concreta es mantener el ojo del dibujante fijo y usar un dispositivo para materializar la intersección de los rayos visuales con el plano de la imagen.

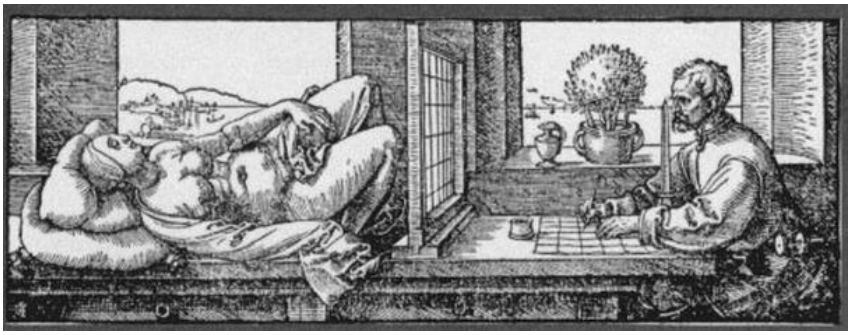


Figura - 1 -Ma1quina de Perspectiva por Albecht Dürer.

Podría considerarse a estas máquinas de la perspectiva como el primer intento de una cámara fotográfica. Utilizan un plano R dónde se forma la imagen (en Figura - 1, la rejilla) y un punto C como centro óptico (el ojo del dibujante) que no pertenece al anterior plano, y dónde se intersectan los rayos que forman la imagen.

En el año 1545 el astrónomo Germina Frisius publica un estudio donde presenta la cámara oscura. En la Figura - 2 se representa un esquema de la cámara oscura. Mediante un orificio muy pequeño C en una pared se deja entrar la luz externa que es proyectada en una pared interior de la cámara oscura. El resultado es una imagen invertida del mundo exterior. La cámara oscura sirvió a algunos pintores como a Vermeer (1632-1675) para representar de la manera más precisa posible la realidad.

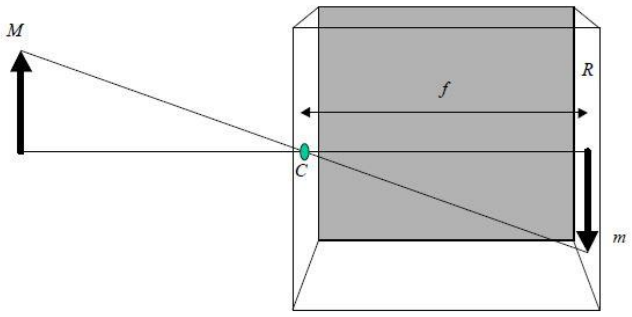


Figura - 2 - Cámara Oscura.

A partir de la teoría del plano cartesiano introducida por Descartes (1596-1650) se empieza a concebir la geometría desde un punto de vista algebraico. Así, las entidades geométricas son descritas como coordenadas y entidades algebraicas. Se cuenta que Descartes desarrollo esta teoría acostado en su cama al ver volar una mosca en la habitación. El movimiento de la mosca con respecto a los tres ejes conformados por la intersección de dos paredes y el techo, hizo que Descartes pensara en que con tres coordenadas (X, Y, Z) se podía definir la posición de la mosca en cualquier instante.
En el año 1826 el químico francés Niepce(1765-1833) llevó a cabo la primera fotografía, colocando una superficie fotosensible dentro de la cámara oscura para fijar la imagen. Posteriormente, en 1838 el químico francés Daguerre hizo el primer proceso fotográfico práctico. Daguerre utilizó una placa fotográfica que era revelada con vapor de mercurio y fijada con trisulfato de sodio.
En la actualidad se utilizan cámaras réflex y CCD que emplean lentes para incrementar la potencia de la luz y mejorar el enfoque de la imagen.

4.1.2. IMAGEN DIGITAL

¿Qué es una imagen digitalizada? Es un conjunto de elementos llamados pixeles. Cara pixel nos ofrece una información sobre una región elemental de la imagen. En imágenes en niveles de gris dicha información es el brillo; mientras que en imágenes en color la información corresponde a la intensidad de cada una de las componentes de una base de color (RGB).

En el proceso de obtención de una imagen digital podemos distinguir dos etapas. La primera es la conocida como la captura y utiliza un dispositivo, generalmente óptico, con el que obtenemos información relativa a la escena. La segunda etapa es la conocida como digitalización, se trasforma esa información en la imagen digital.

4.1.2.1.Modelos de captura de imágenes

Para capturar una imagen se suele distinguir entre dispositivos pasivos y dispositivos activos. Los primero son los que generalmente están basados en el principio de cámara oscura. Mientras que los segundos son los basados en el escaneo.

- **Cámara Oscura:** el principio de cámara oscura se suele usar para capturar imágenes de escenas tridimensionales del mundo real y proyectarlas en un plano bidimensional. Dispositivos de este tipo son las cámaras fotográficas y las cámaras de video. Este modelo de captura también puede ser usado para capturar imágenes de elementos bidimensionales como son fotografías y documentos. Un ejemplo de esto sería los escáneres de cámara. De igual forma podemos utilizar dos o más cámaras para capturar diferentes perspectivas de una misma escena y construir una representación 3D de la escena a capturar.

- **Escaneo**: Este esquema utiliza un elemento activo (generalmente un haz de luz láser) que recorre la escena que se desea capturar; por lo que son imprescindibles dos dispositivos, uno emisor del haz de luz y otro el receptor. El escáner emite el haz de luz y éste, tras chocar con la imagen que se escanea, es

recogido en el detector de luz. Si repetimos este proceso de marea continua se puede construir una señal que corresponde a una representación de la escena. Los escáneres-láser son capaces de capturar escenas en 3D directamente y los escáneres tambor permiten capturar imágenes de elementos bidimensionales.

Los dispositivos basados en cámara aventajan a los basados en escaneo en velocidad y simplicidad. También se parecen más al sistema visual humano. En un futuro los modelos de cámara terminaran superando a los de escaneo también en calidad de imagen; ya que a día de hoy es su principal cuello de botella.

4.1.2.2.Digitalización

La digitalización es el proceso de paso del mundo continuo o analógico al mundo discreto o digital. Distinguimos dos procesos: el muestreo y la cuantificación.

- **Muestreo:** consiste en la medición a intervalos (discretización) respecto de alguna variable (tiempo o espacio), siendo su parámetro fundamental la frecuencia de muestreo, la cual representa el número de veces que se mide un valor analógico por unidad de cambio.
  Mediante el Muestreo se convierte una imagen (que es algo continuo) en una matriz discreta de NxM píxeles. El número de muestras por unidad de espacio sobre el objeto original conduce al concepto de resolución espacial de la imagen. Esta se define como la distancia, sobre el objeto original, entre dos píxeles adyacentes. Sin embargo la unidad de medida de resolución espacial más habitual suele ser los píxeles por pulgadas siempre medidos sobre el objeto original. La imagen numero X nos muestra un claro ejemplo de resolución.

  Si nos centramos en las imágenes en 2D, tenemos que podemos discretizar la imagen de la siguiente manera:

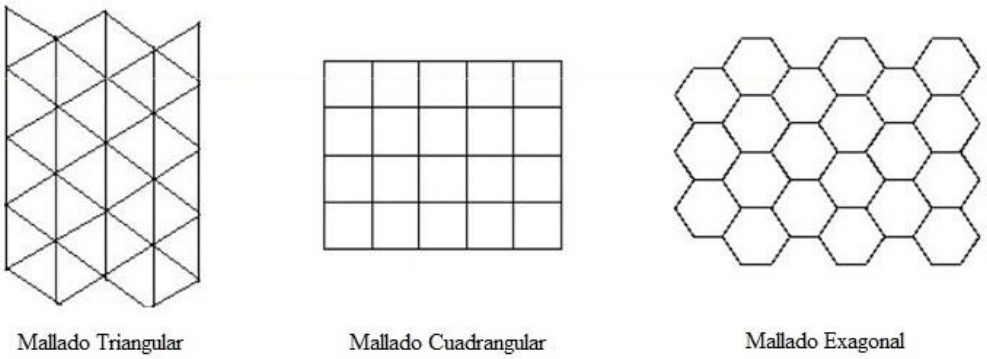


Figura - 3 - Mallados para el muestreo de imágenes

  Si tuviéramos que discretizar imágenes en 3D los distintos tipos de mallado son los siguientes:

  - Mallado BCC(Body-Centered-Cubic)
  - Mallado FCC(Face-Centered-Cubic)
  - Mallado Cúbico (que es el más usual)

- **Cuantificación:** La segunda operación es la cuantificación de la señal, que consiste en la discretización de los posibles valores de cada píxel. Los niveles de cuantificación suelen ser potencias de 2 para facilitar el almacenamiento en el computador de las imágenes, ya que éstos utilizan el byte como unidad mínima

de memoria directamente direccionable. Así, suelen usarse 2, 4, 16 o 256 niveles posibles.

Hay 256 valores de intensidad posibles, a esto se le denomina resolución de intensidad. Como mínimo en una imagen debe haber 2 valores, que designan el blanco (valor 255 de intensidad de escala de grises) y el negro (valor 0 de intensidad de escala de grises).En la Figura - 4 se observa la diferencia entre los distintos valores de intensidad de la escala de grises.

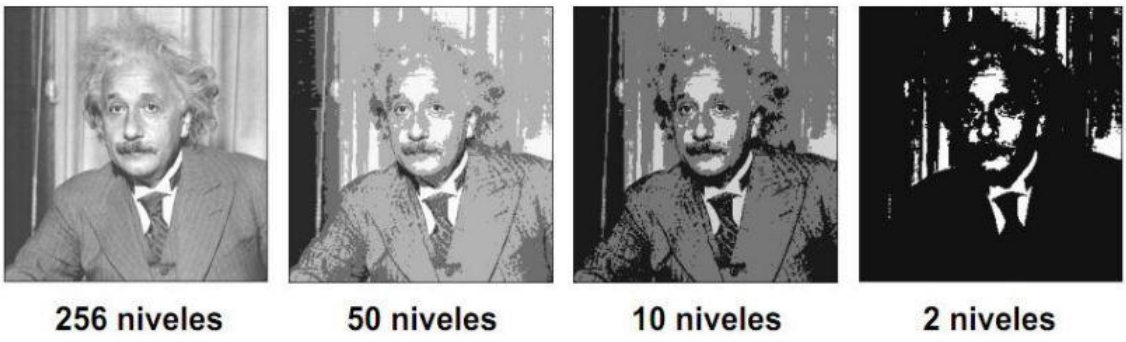


Figura - 4 - Distintos valores de intensidad en escala de grises

Ejemplo de digitalización:

El muestreo se ha hecho usando un mallado cuadrangular de 9 por 9 y la cuantificación consiste en una paleta de 256 niveles de gris (siendo 0 el color negro y 255 el color blanco).

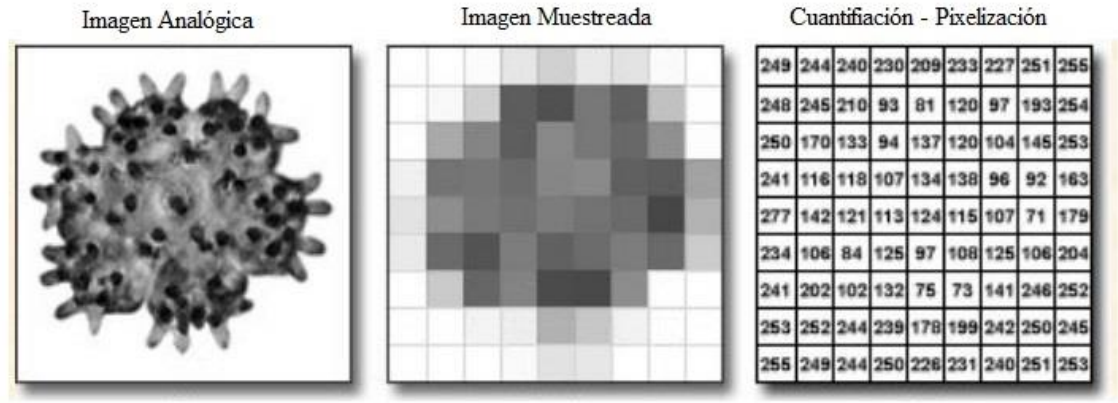


Figura - 5 - Creación de una imagen digital

Hemos de tener en cuenta que si el muestreo consiste en un mallado de M por N píxeles y el número de niveles de grises permitido es $L=2^k$, entonces el número de bits necesarios para almacenar una imagen digital es: M x N x k

Por lo que para una imagen de:

128 x 128 con 64 (2^6) niveles de gris necesitamos 98.304 bits o lo que es lo mismo 12 KB de memoria.

256 x 256 con 132 niveles de gris necesitamos 458.752 bits o lo que es lo mismo 56 KB de memoria.

1024 x 1024 con 256 niveles de gris necesitamos 8.388.608 bits o lo que es lo mismo 1024 KB = 1 MB de memoria.

4.1.3. PROCESAMIENTO DE IMÁGENES

Un paso previo a la segmentación y reconocimiento de una imagen son las operaciones que vamos a tratar en este apartado. El propósito de dichas operaciones es mejorar o destacar alguno de los elementos de las imágenes de manera que podamos garantizar las etapas posteriores de segmentación y clasificación.
Todas las operaciones que van a ser descritas en este capítulo se pueden explicar desde la perspectiva de la teoría de filtros. Un filtro puede verso como un mecanismo de cambio o trasformación de una señal de entrada a la que se le aplica una función, conocida como función de trasferencia, gracias a la cual obtenemos una señal de salida. En este contexto se entiende por señal una función de una o varias variables independientes. Los sonidos y las imágenes son ejemplos típicos de señales.

4.1.3.1. Operaciones básicas entre píxeles

Tenemos dos tipos de operaciones que podemos realizar entre pixeles:

- **Operaciones aritmético-lógicas:**
  Estas operaciones son las que se utilizan para leer y dar valor a los píxeles de las imágenes. Las operaciones básicas son: conjunción (AND), disyunción (OR), negación (NOT), suma, resta, multiplicación, división.
  En la Figura - 6 podemos observar ejemplos básicos de operaciones aritmético-lógicas donde los píxeles negros corresponden a bits 0 y los blancos a bits 255.

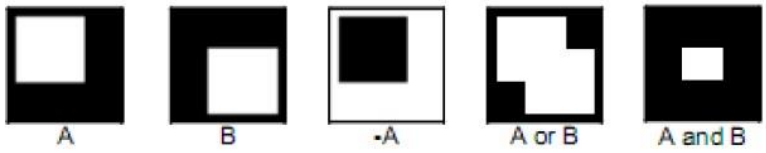


Figura - 6 - Operaciones aritmético-lógicas con píxeles

- Operaciones geométricas:
  Cuando hablamos de operaciones geométricas más concretamente nos referimos a operaciones de traslación, escalado y rotación.
  En la Figura - 7 observamos como en la imagen original se somete a procesos de traslación y rotación.

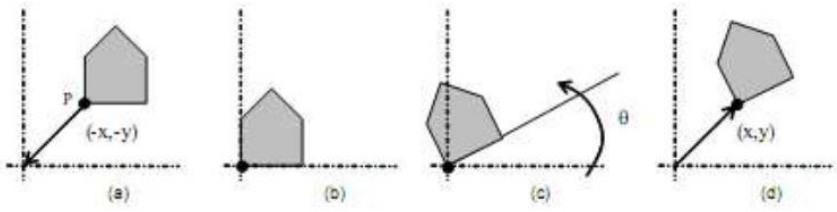


Figura - 7 - Ejemplo de operaciones geométricas

4.1.3.2.Operaciones sobre el histograma

En primer lugar tenemos que definir que es un histograma. Es un diagrama de barras en el que cada barra tiene una altura proporcional al número de píxeles que hay para un nivel de cuantificación determinado. Lo normal es que en el eje de abscisas se disponen los diferentes niveles de cuantificación de valores que pueden tomar los píxeles y en el eje de ordenadas tenemos el número de píxeles que habrá para cada nivel cuantificado. El histograma de una imagen en niveles de gris proporciona la información sobre el número de píxeles que hay para cada nivel de intensidad, mientras que en imágenes en color RGB se usan 3 histogramas, uno por cada componente de color.

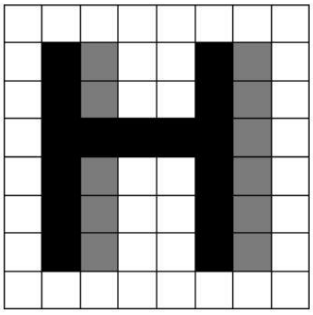


Figura - 8 - Imagen de 8x8 píxeles.

En la Figura - 8 tenemos una imagen en niveles de gris muy simple de 8x8 pixeles de tamaño (se han señalado los límites entre píxeles para facilitar su identificación). Sólo son posibles 4 niveles de gris, porque se van a usar 2 bits para codificar el brillo de cada píxel. De la forma habitual, los niveles de gris se numeran del 0 al 3, correspondiendo un brillo de cada píxel. De la forma habitual, los niveles de gris se numeran del 0 al 3 correspondiendo un brillo mayor a los valores más altos en la Tabla - 1 vemos el resumen.

Nivel de gris	Brillo
0	Negro
1	Gris oscuro
2	Gris claro
3	Blanco

Tabla - 1 - Identificación de los niveles de gris en la Figura - 8.

La gráfica de la Figura - 9 es el histograma correspondiente a la imagen de la Figura 8.

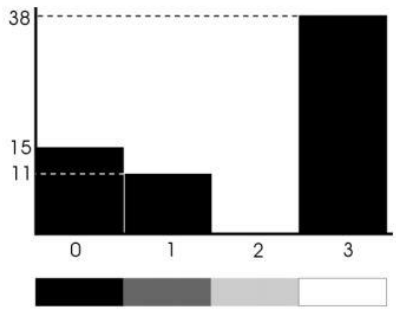


Figura - 9 - Histograma correspondiente a la Figura – 8

Los números que aparecen en el eje horizontal representan los niveles de gris que pueden aparecer en la imagen. A la izquierda está el valor más oscuro (negro) y en el extremo derecho el más claro (blanco). El resto de niveles se distribuyen uniformemente. Se ha puesto una escala con los tonos de gris correspondientes para facilitar la comprensión. En un histograma real habitualmente no encontraremos numerado el eje vertical, ni la escala de tonos para el eje horizontal. La altura de cada barra representa el número de píxeles de la imagen que presentan ese nivel de gris concreto. Se puede deducir entonces que la imagen tiene 15 píxeles completamente negros (con nivel 0), 11 de tono gris oscuro (nivel 1) y 38 píxeles completamente blancos (nivel 3). No hay ningún píxel en la imagen con un nivel de gris 2. Tenemos que tener en cuenta que la altura de todas las barras ha de sumar un total de 64, que es el número total de píxeles que tiene nuestra imagen.

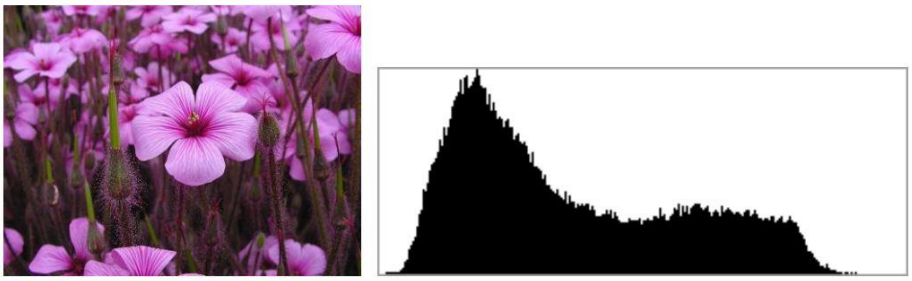


Figura - 10 - Imagen de flores con su histograma correspondiente.

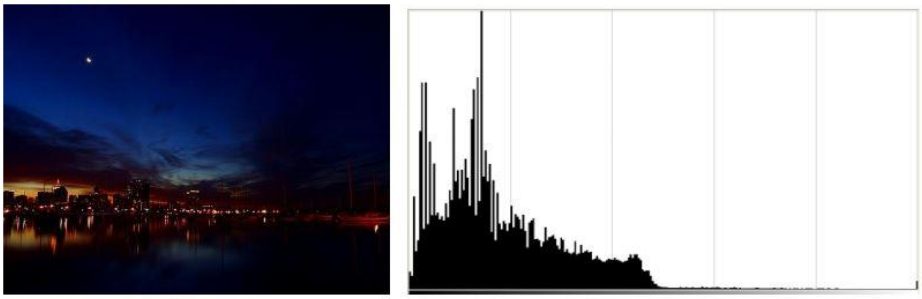


Figura - 11 - Imagen nocturna y su histograma correspondiente.

Las figuras 11 y 12 nos muestran los histogramas del contraste de las imágenes que están en el lado izquierdo de la figura correspondientemente. Podemos observar que en la figura 12 al tratarse de una imagen oscura el contraste existente entre los distintos tonos de la imagen es menor y la dispersión de las barras en el histograma es menor y obtenemos todas las barras agrupadas hacia los colores oscuros.

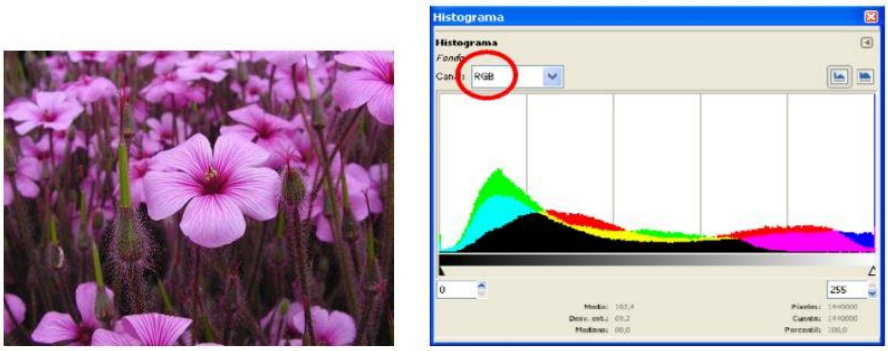


Figura - 12 - Histograma de canales de color.

En las Figuras 13 y 14 observamos los histogramas para cada canal de color, en primer lugar superpuestos y en segundo por separado.

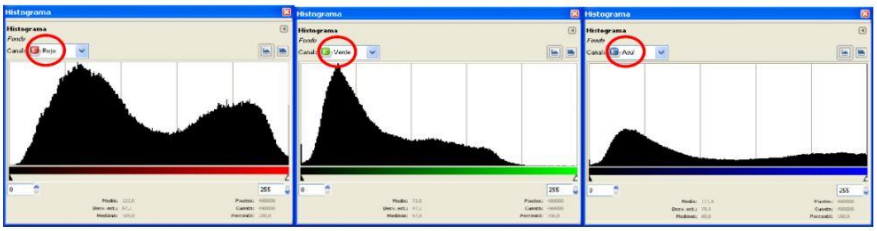


Figura - 13 - Histograma de canales de color por separado.

- **Aumento y reducción de contraste:**

Las modificaciones del histograma se pueden visualizar eficazmente mediante las funciones de trasferencia del histograma. Estas funciones corresponden a aplicaciones, pues para cada punto del domino solo tienen un valor imagen. Estas aplicaciones están acotadas entre 0 y 1 tanto en la abscisa, que se hace corresponder con la entrada del filtro (Imagen de Entrada = IE), como en la ordenada, que se corresponde con la salida (Imagen de Salida = IS) del filtro.

En la figura 14 se pueden ver tres ejemplos de funciones de transferencia. La función de trasferencia lineal (a) no introduce modificación alguna sobre el histograma, al coincidir exactamente los niveles de intensidad de la entrada y de la salida. La función cuadrado produce unos oscurecimientos generales de la imagen, lo podemos ver en la figura (b). Por último la función raíz cuadrada (c) produce un aclarado general de la imagen.

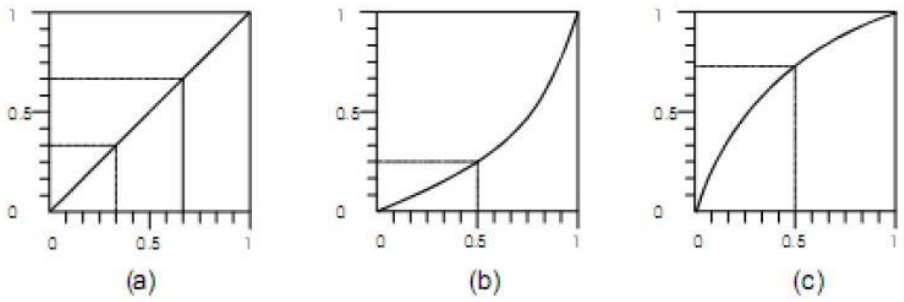


Figura - 14 - Funciones de trasferencia, (a) lineal, (b) cuadrado y (c) raíz cuadrada.

La función de transferencia que aclare los niveles claros y oscurezca los niveles oscuros conseguirá sobre el conjunto de la imagen un efecto visual de aumento del contraste. Una función tal se puede obtener componiendo una función de trasferencia del histograma que hasta el valor de 0,5 se comporte como la función cuadrado y que en adelante se comporte como la función raíz cuadrada. En la figura 15 se ha representado esta función de trasferencia. La función (b) de la misma figura produce un efecto contrario, esto es una disminución del contaste.

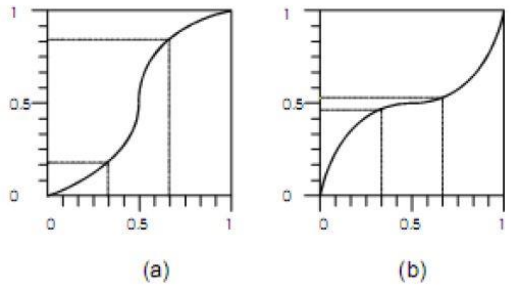


Figura - 15 - Funciones de trasferencia, (a) aumento del contraste y (b) disminución del contraste.

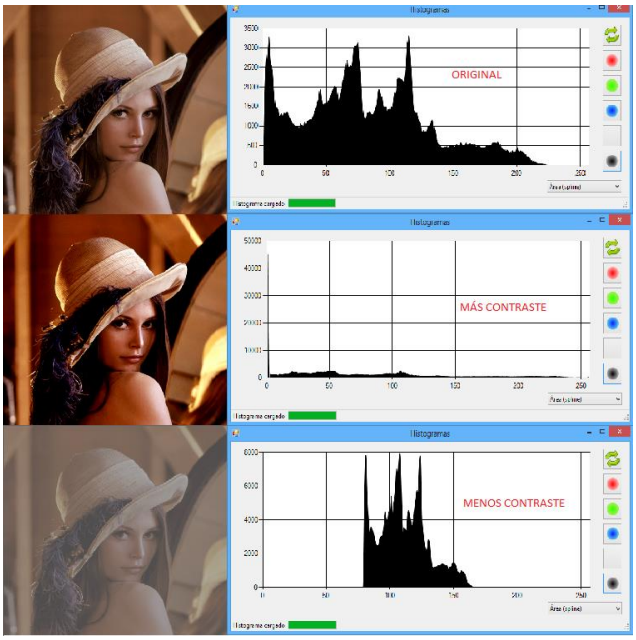


Figura - 96 - Transformaciones del contraste en una imagen a color.

4.1.4. SEGMENTACIÓN

La segmentación de una imagen permite separar o destacar zonas con características específicas de forma o color, con la finalidad de facilitar un análisis posterior. Así podremos localizar la cara de una persona dentro de la imagen, o encontrar los límites de una palabra dentro de una imagen.

La segmentación no tiene reglas estrictas a seguir y dependiendo del problema al que nos enfrentemos puede ser necesario idear técnicas a medida.

La segmentación debe verse como un proceso en el que a partir de una imagen se produce otra en la que cada píxel tiene asociada una etiqueta distintiva del objeto al que pertenece. Así una vez segmentada una imagen, se podría formar una lista de objetos consistentes en las agrupaciones de los píxeles que tengan la misma etiqueta. La segmentación no es un proceso fácil debido a que por un lado no se tiene una información adecuada de los objetos a extraer y por otro lado, a que la escena a segmentar aparece normalmente ruido.

Podemos diferenciar entre 3 grupos de técnicas: técnicas basadas en la umbralización, basadas en la detección de los contornos de los objetos y técnicas basadas en propiedades locales de las regiones.

4.1.4.1. Segmentación basada en la umbralización

La umbralización es el proceso que nos permite convertir una imagen de niveles de grises o de color en una imagen binaria; de tal forma que los objetos de interés se etiqueten con valor distinto al de los píxeles del fondo. Esta técnica de segmentación es rápida y tiene un coste computacional bajo y puede ser realizada en tiempo real durante la captura de la imagen.

En la figura X vemos un ejemplo de umbralización de una imagen y su imagen binaria.

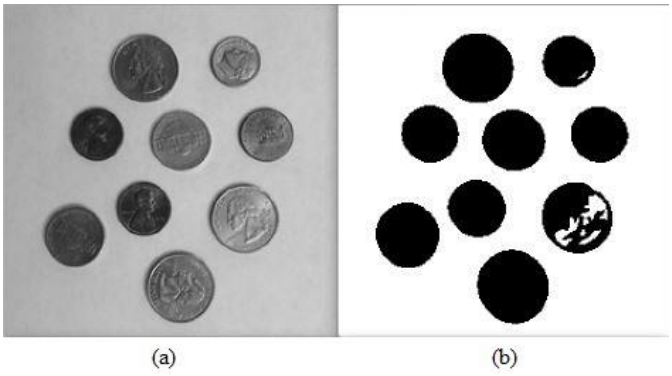


Figura - 17 - Umbralización de la imagen (a) en la imagen (b).

4.1.4.2.Segmentación basada en la detección de contornos.

En este tipo de segmentación podemos utilizar diversas técnicas, pero nos vamos a centrare en la utilización de filtros de gradiente, ya que es una de las más utilizadas y con las que obtenemos un mejor resultado.

Lo primero que tenemos que hacer es encontrar el "gradiente" de la imagen en escala de grises que queremos procesar; esto nos permitirá que encontremos bordes como regiones en las direcciones x e y.

- Detección de bordes Canny: etapa multiproceso que consta de las siguientes etapas:
  - Aplicamos el filtro gaussiano para suavizar la imagen con el fin de eliminar el ruido.
  - Encontrar los gradientes de intensidad de la imagen.
  - Aplicar doble umbral para determinar los bordes potenciales.
  - Seguimiento de borde por histéresis: Finalizar la detección de bordes mediante la supresión de todos los otros bordes que son débiles y no están conectados a los bordes fuertes.

No vamos a adentrarnos en las descripciones matemáticas de todas estas técnicas ya que no es el foco de nuestro proyecto, y explicar todas ellas en detalle nos separaría de nuestro objetivo.

En las imágenes 19 y 20 observamos el tratamiento de una imagen con la detección de bordes Canny.

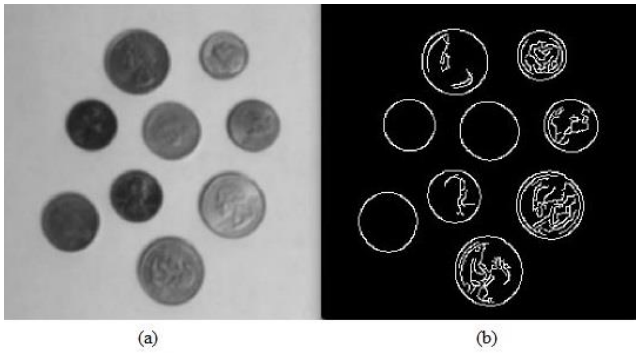


Figura - 18 - Imagen borrosa tratada con la detección de bordes Canny.

Figura - 19 - Fotografía tratada con la detección de bordes Canny.

4.1.5. SEGUIMIENTO DE OBJETOS EN MOVIMIENTO

Hasta el momento solo hemos hablado del tratamiento de imágenes, cuando tratamos con una fuente de video el método que debemos utilizar es diferente. En este apartado lo que intentaremos es comprender el moviente del objeto. Para ello tendremos dos componentes principales que calcular: identificación y modelado.

La identificación consiste en localizar el objeto en el que estamos interesados dentro de una de las imágenes, o frame, que componen la secuencia (que conocemos como video). Para ello utilizaremos las técnicas vistas anteriormente, ya que es mucho más complicado seguir cosas que no hemos identificado. El seguimiento de objetos no identificados es importante cuando queremos determinar algo con su movimiento o cuando el movimiento del propio objeto es lo que le hace interesante.
Las técnicas de seguimiento de objetos no identificados normalmente involucra el seguimiento de puntos clave muy significantes.
Como métodos de seguimiento de objetos destacamos: el algoritmo de Lucas-Kanade y el de Horn-Schunk, los cuales representan los flujos ópticos no compactos y compactos respectivamente.

Ambos algoritmos tratan de estimar el flujo óptico de un objeto.
¿Qué es el flujo óptico? Flujo óptico es el patrón de movimiento aparente de los objetos, superficies y bordes en una escena visual causado por el movimiento relativo entre un observador (un ojo o una cámara) y la escena. El concepto de flujo óptico fue presentado por el psicólogo estadounidense James J. Gibson en la década de 1940 para describir el estímulo visual suministrado a los animales que se mueven a través del mundo. Gibson hizo hincapié en la importancia de flujo óptico para la percepción, la capacidad de discernir las posibilidades de acción dentro del ambiente. Seguidores de Gibson y su enfoque ecológico a la psicología han demostrado aún más el papel de los estímulos de flujo óptico para la percepción del movimiento del observador en el mundo; la percepción de la forma, la distancia y el movimiento de los objetos en el mundo; y el control de la locomoción.

- **Algoritmo de Lucas-Kanade:** Es un método diferencial ampliamente utilizado para la estimación de flujo óptico desarrollada por Bruce D. Lucas y Takeo Kanade. Se supone que el flujo es esencialmente constante en una zona local del píxel bajo consideración, y resuelve las ecuaciones básicas de flujo óptico para todos los píxeles en ese barrio, por el criterio de mínimos cuadrados.
  Al combinar la información de varios píxeles cercanos, el método Lucas-Kanade menudo puede resolver la ambigüedad inherente de la ecuación de flujo óptico. También es menos sensible al ruido de imagen que los métodos de coma sabia. Por otra parte, ya que es un método puramente local, no puede proporcionar información de flujo en el interior de las regiones uniformes de la imagen.
  La idea básica de este algoritmo se basa en tres puntos:
  1. Brillo constante. Un píxel de la imagen o un objeto de una escena no puede cambiar de apariencia (en la medida de lo posible) en su movimiento. Para una imagen en escala de grises esto significa que el brillo de un píxel no puede cambiar.
  2. Persistencia temporal de pequeños movimientos. El movimiento de una zona de la imagen cambia lentamente en el tiempo. Esto significa que el incremento temporal es demasiado rápido para la escala del movimiento, el objeto no se mueve mucho entre frames.
  3. Coherencia espacial. Los puntos vecinos de una escena pertenecen a la misma superficie, que tiene un movimiento similar, y proyecta los puntos cercanos en el plano de la imagen.

- **Algoritmo de Horn-Schunck:** Fue creado en 1981 es una de las primeras técnicas que usaban la constancia en el brillo y derivaban las ecuaciones básicas del brillo constante. La solución de estas ecuaciones se hicieron bajo la hipótesis de suavizar la fuerza de las velocidades en el eje “x” y el eje “y”.

4.2. LENGUAJES DE PROGRAMACIÓN

En este apartado vamos a describir brevemente los lenguajes de programación más utilizados en y desarrollados en la visión por computador.

4.2.1. JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.
La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.
Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

4.2.2. C

C es un lenguaje de programación creado en 1972 por Kenneth L. Thompson y Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL. Al igual que B, es un lenguaje orientado a la implementación de Sistemas Operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones. Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.
Entre sus características se encuentran las siguientes: Un núcleo del lenguaje simple, con funcionalidades añadidas importantes, como funciones matemáticas y de manejo de ficheros, proporcionadas por bibliotecas. Es un lenguaje muy flexible que permite programar con múltiples estilos. Uno de los más empleados es el estructurado no llevado al extremo (permitiendo ciertas licencias rupturistas). Un sistema de tipos que

impide operaciones sin sentido. Usa un lenguaje de preprocesado, el preprocesador de C, para tareas como definir macros e incluir múltiples ficheros de código fuente. Acceso a memoria de bajo nivel mediante el uso de punteros. Interrupciones al procesador con uniones. Un conjunto reducido de palabras clave. Por defecto, el paso de parámetros a una función se realiza por valor. El paso por referencia se consigue pasando explícitamente a las funciones las direcciones de memoria de dichos parámetros. Punteros a funciones y variables estáticas, que permiten una forma rudimentaria de encapsulado y polimorfismo. Tipos de datos agregados (`struct`).

4.2.3. C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.
Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje *multiparadigma*.
Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. *C++* permite trabajar tanto a alto como a bajo nivel. El nombre **C++** fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

4.2.4. PYTHON

Python es un lenguaje de programación interpretado creado por Guido van Rossum en el año 1991. Se compara habitualmente con Tcl, Perl, Scheme, Java y Ruby.
Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.
Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.
Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software Foundation License,1 que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores.

4.3. LIBRERIAS PARA EL DESARROLLO DE NUESTRO TRABAJO

El lenguaje de programación en el que hemos desarrollado nuestro trabajo es Python, por lo que en este apartado nos vamos a centrar en las librerías de este lenguaje que hemos instalado, procediendo a una breve explicación de sus características básicas.

4.3.1. NUMPY Y SCIPY

Numpy es el paquete fundamental para la computación científica en Python. Es una biblioteca Python que proporciona un objeto de matriz multidimensional, diversos objetos derivados (tales como matrices y matrices enmascarados), y una variedad de rutinas para operaciones rápidas sobre matrices, incluyendo matemática, lógica, dar forma a la manipulación, clasificación, selección, I / O, transformada discreta de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

En el núcleo del paquete Numpy, es el objeto ndarray. Esto encapsula matrices n-dimensionales de tipos de datos homogéneos, con muchas de las operaciones que se realizan en el código compilado para el rendimiento.

La biblioteca SciPy, una colección de algoritmos que entre otras incluye el procesamiento de señales, la optimización, estadísticas y mucho más.

En muchas distribuciones de Linux, como Ubuntu, Numpy viene pre-instalado y configurado; pero si deseamos tener la última versión de Numpy y Scipy el método más fácil para obtenerlas es usar el gestor de paquetes de Ubuntu y con la sentencia `apt -get` obtener la actualización.

4.3.2. MATPLOTLIB

Matplotlib es una biblioteca de trazado, para los que están familiarizados con MATLAB les resulta muy cómodo trabajar con el entorno matplotlib. Esta librería nos permite el uso de histogramas o simplemente de imágenes tal cual. Es una librería que nos puede llegar a ser muy útil.

Si ya hemos instalado ScipySuperpack entonces ya tenemos instalado también matplotlib.

4.3.3. OPENCV

La principal característica de OpenCV es que es una librería de software libre de visión por computador que incluye cuatro componentes diferentes para la visión por computador en tiempo real.

El primer componente es CxCore, el cual nos proporciona los tipos genéricos que serán usados por las demás librerías como pueden ser árboles, listas, colas, secuencias, imágenes y las funciones que operan sobre estos.

El segundo componente es CvReference que incluye todas las funciones de análisis y procesamiento.

El tercer componente es la librería CvAux que contiene componentes experimentales y obsoletos del proyecto OpenCV. El último componente es la librería HighGUI que nos proporciona los elementos necesarios para la captura, presentación y almacenamiento de imágenes.

El componente HighGUI viene como una librería adicional. Esta librería nos proporciona una extensa lista de interfaces de captura. No provee herramientas para crear diferentes hilos ni flujos de datos, la adquisición, procesamiento y presentación de la imagen debe ser explicita.

Resumiendo OpenCV es una fuente fantástica de implementaciones estandarizadas de las funciones de visión por computador.
Su instalación no es algo evidente pero más adelante se procederá a su completa explicación.

4.3.4. MAHOTAS

Mahotas, al igual que OpenCV, se basa en matrices Numpy. Muchas de las funcionalidades implementadas en Mahotas se puede encontrar en OpenCV pero en algunos casos, la interfaz Mahotas es más fácil de usar. En nuestro caso la usaremos para complementar OpenCV.

4.4. SISTEMAS DE LOCOMOCIÓN DE ROBOTS

Existe una amplia gama de robots y cada uno de ellos es digno de estudio en este apartado nos vamos a centrar en el área de la robótica móvil más concretamente en los robots que utilizan como medio de locomoción ruedas. Realizaremos una revisión del estado del arte de dichos robots, y de las configuraciones más empleadas, los actuadores comúnmente utilizados y algunas de las técnicas básicas de control, con el objetivo de conseguir cada vez una mayor autonomía en los problemas de seguimiento de trayectoria y en la evasión de obstáculos.

La robótica ha jugado un papel muy importante durante el desarrollo de la humanidad y su evolución ha ido de la mano con la construcción de artefactos que materializan el deseo de crear objetos que facilitan el trabajo al hombre. Ya en antiguas civilizaciones como la griega se hablaba de seres mecánicos con vida que eran movidos por mecanismos construidos con poleas y bombas hidráulicas. Pero el concepto de robot como tal comenzó a fraguarse en la civilización árabe, donde se le dio sentido a dichos mecanismos para confort del ser humano, y ahí se inició la evolución de la robótica.

El primero en escribir sobre los robots fue Capek, escritor checo, que en 1920 escribió 'Rossum's Universal Robots (R.U.R.)', fue el primero en acuñar el término robot, a partir de la palabra checa robota, que significa servidumbre o trabajo forzado.

Por su parte Asimov fue el primero en utilizar el término robótica, en la historia Runaround de su obra I, 'Robot'; incluyendo en este mismo trabajo las tres leyes de la robótica.

1. Un robot no hará daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la 1ª Ley.
3. Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la 1ª o la 2ª Ley.1

Los robots en la actualidad no solo se limitan a semejarse a los seres humanos, si no que han tomado formas diversas para satisfacer ciertas necesidades de la mejor manera.

Los verdaderos intentos de robot se implementaron en la década de los cuarenta. En 1952 en el MIT, aparece la primera máquina de control numérico para automatizar algunas tareas industriales.

Por otro lado la compañía Unimates, en 1961 introdujo el primer robot industrial en la General Motors.

En cuanto a robots móviles se refiere, entre 1966 y 1972 se desarrolló en el SRI (Stanford Research Institute, ahora conocido como Artificial Intelligence Center) el primer robot móvil llamado Shakey; era una plataforma móvil independiente controlada por visión mediante una cámara y dotada con un detector táctil. A partir de este momento la investigación y el diseño de robots móviles crecerán de manera exponencial.

Los robots móviles brindan la posibilidad de "navegar" en distintos terrenos y tienen como propósito aplicaciones como: la explotación minera, explotación planetaria, misiones de búsqueda y rescate de personas, limpieza de desechos peligrosos,

automatización de procesos, vigilancia, reconocimiento de terrenos…. entre otras muchas.

Los robots móviles se clasifican por el tipo de locomoción utilizado de la siguiente manera:

- Por ruedas.
- Por patas
- Orugas.

Destacamos que pese a que la locomoción por patas y orugas ha sido ampliamente estudiada el mayor desarrollo se presenta en los Robots Móviles con Ruedas (RMR).

Entro de las características más relevantes de los RMR destacamos su eficiencia en cuanto a energía en superficies lisas y firmes, a la vez que no causan desgaste en la superficie donde se mueven y requieren un número menor de partes y menos complejas en comparación con los robots de patas y de orugas, lo que a su vez permite que su construcción se a más sencilla.

Por lo que podemos definir un robot móvil de ruedas como un sistema electromecánico controlado, que utiliza como locomoción ruedas de algún tipo y que es capaz de trasladarse de forma autónoma a una meta preestablecida en un determinado espacio de trabajo.

La autonomía de un robot móvil es el dominio que tiene éste para determinar su curso de acción mediante un proceso propio de razonamiento en base a sensores.

Las partes que componen un RMR son: la configuración cinemática y el sistema de actuadores. Ambos sistemas están íntimamente ligados aun realizaremos su explicación por separado.

4.4.1. CONFIGURACIONES CINEMÁTICAS DE LOS RMR

Existen diferentes configuraciones cinemáticas para los RMR, aunque de manera general se tienen las siguientes configuraciones:

- Ackerman.
- Triciclo clásico.
- Tracción diferencial.
- Skid steer o por cintas de desplazamiento.
- Síncrona.
- Tracción omnidireccional u omniwheels.

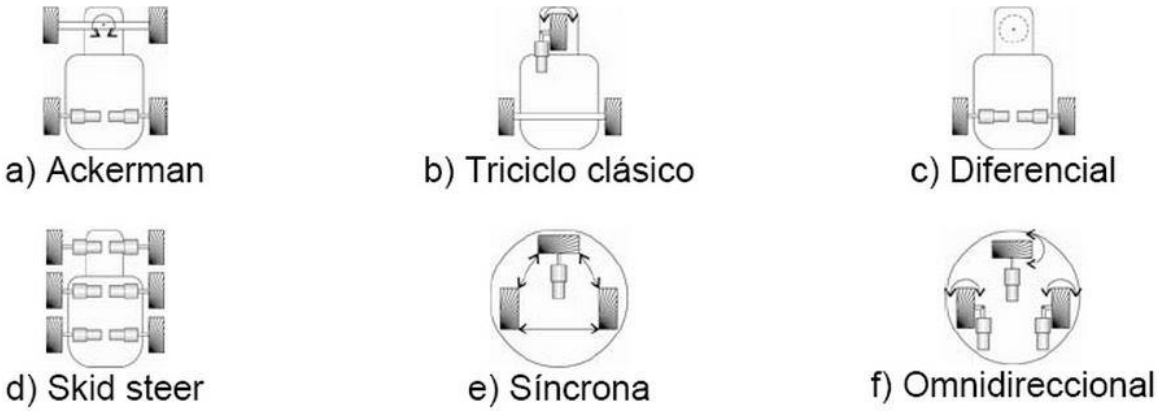


Figura - 20 - Configuración de los RMR.

Dependiendo de la configuración cinemática que conformen el RMR puede utilizar cuatro tipos de ruedas diferentes para su locomoción. Estas son las siguientes:

- Convencionales.
- Tipo castor.
- Ruedas de bolas.
- Omnidireccionales.

Figura - 21 - Tipos de ruedas.

4.4.1.1. Conceptos previos

Antes de explicar un poco más en detalle las diferentes configuraciones cinemáticas, tenemos que dejar claro ciertos conceptos básicos.

- Rueda motriz: rueda que nos proporciona la fuerza de tracción al robot.
- Rueda directriz: ruedas de direccionamiento de orientación controlable.
- Ruedas fijas: sólo giran en torno a su eje sin tracción motriz.
- Ruedas locas o ruedas de castor: ruedas orientables no controladas.

Tenemos que dejar claro también que es el centro instantáneo de rotación (CIR) o centro instantáneo de curvatura (CIC). Lo definimos como el punto de intersección de todos los ejes de las ruedas.

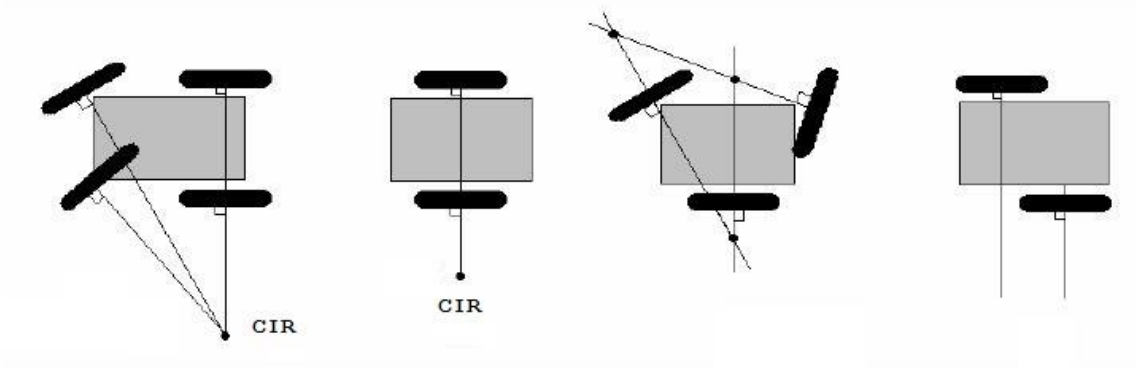
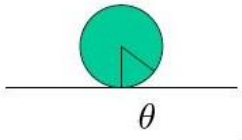


Figura - 22 - Centro instantáneo de rotación (CIR).

- Restricciones no holónomas: Conceptualmente esto significa que el robot puede moverse instantáneamente adelante o atrás pero no lateralmente por el desplazamiento de las ruedas.
  Matemáticamente hablando:

$$G(p, \dot{p}, t) = 0$$

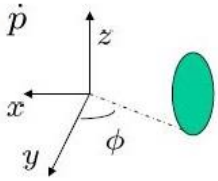
R. Holónoma no depende de $\dot{p}$



$$\dot{x} = R \cdot \dot{\theta}$$

$$\int dt \rightarrow x - R \cdot \theta = cte$$

R. No Holónoma depende de $\dot{p}$ **y no es integrable**



$$-\dot{x}\sin\phi + \dot{y}\cos\phi = \dot{\theta} \cdot R$$

$$\dot{x}\cos\phi + \dot{y}\sin\phi = 0$$

NO INTEGRABLE

Figura - 23 - Restricciones Holónomas.

4.4.1.2. Locomoción diferencial

En este caso no disponemos de ruedas directrices, por lo que los cambios de dirección se realizan modificando la velocidad relativa de las ruedas según deseemos girar a la izquierda o a la derecha.

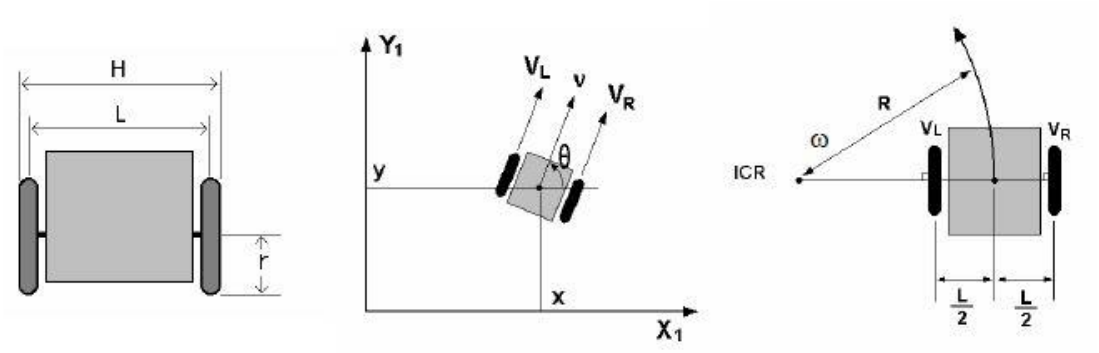


Figura - 24 - Diagrama vectorial de locomoción diferencial.

Los sistemas diferenciales tienes las ventajas de que son sistemas baratos, fáciles de implementar y su diseño es simple. Como contrapunto nos encontramos que son difíciles de controlar, incluso en trayectorias rectas, donde requieren un control de precisión.

Otros problemas asociados que nos encontramos en este tipo de sistemas es que la deformación de los neumáticos nos afecta de forma directa, ya que el cambio de diámetro de las ruedas distorsiona el control de dirección del vehículo.

4.4.1.3. Locomoción Síncrona

En la locomoción síncrona todas las ruedas se mueven de forma síncrona actuadas por el mismo motor. Los sistemas disponen de dos motores, uno para la rotación del sistema y otro para la traslación.

Como ventaja destacamos justo esa que al tener los motores de rotación y traslación separados simplifican mucho el control del sistema.

En este caso el control de la línea recta está garantizado mecánicamente.

Otra ventaja de este sistema es que posee restricciones holónomas.

Los inconvenientes de estos sistemas de locomoción son que su diseño es muy complejo y son difíciles de implementar.

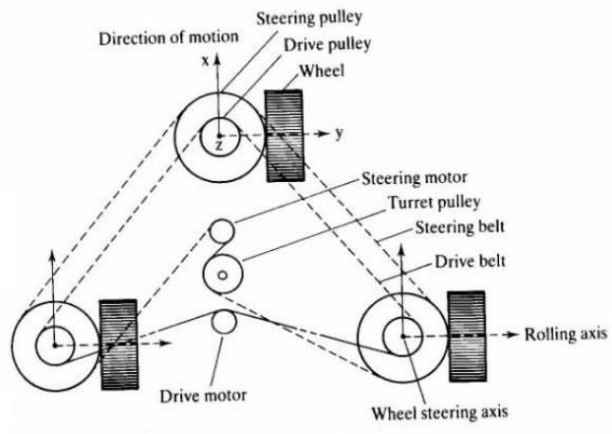


Figura - 25 - Diagrama de locomoción síncrona.

4.4.1.4. Triciclo clásico

Los triciclos cuentan como ventaja principal con que no sufren desplazamiento. Pero su gran desventaja es que se requiere guiado no holónomo.

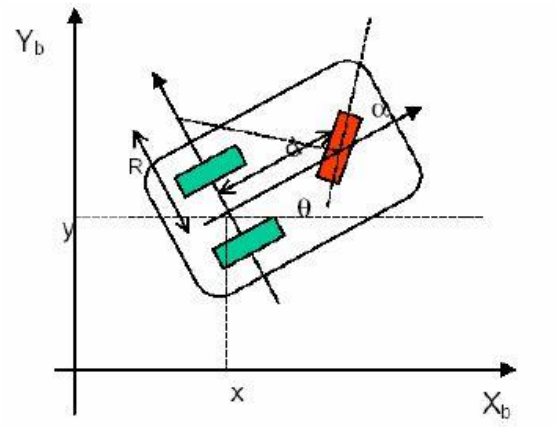
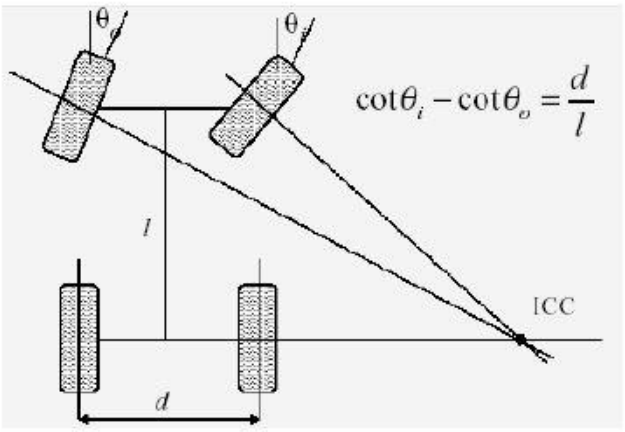


Figura - 26 - Triciclo clásico.

4.4.1.5. Locomoción Ackerman

Estos sistemas al igual que los diferenciales son fáciles de implementar, y el sistema direccional es simple y está formado por 4 barras.

La principal desventaja es que también posee restricciones no holónomas.



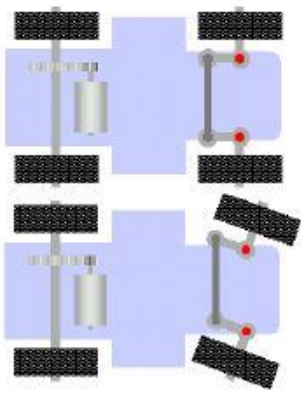


Figura - 27 - Diagrama de sistema de locomoción Ackerman.

4.4.1.6. Locomoción Omnidireccional u Omniwheels

Estos sistemas poseen un diseño complejo que a la vez nos permite una mayor libertad de movimiento que los sistemas de ruedas clásicos. Un ejemplo sería un sistema con Ruedas Suecas.

Una de las ventajas principales de estos sistemas es que nos permite movimientos complicados (reduciendo las restricciones cinemáticas).

Y el mayor inconveniente que presentan es que el movimiento en línea recta no está garantizado por restricciones mecánicas, por lo que precisan de control. Otra desventaja es que la implementación es bastante complicada.

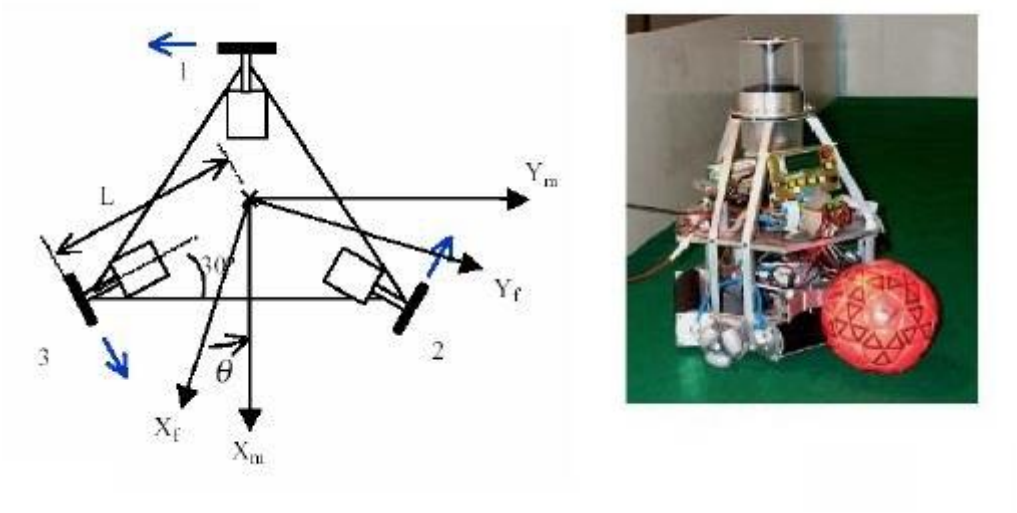


Figura - 28 - Diagrama de locomoción Omnidireccional.

4.4.1.7. Locomoción por cintas de desplazamiento

Las ventajas principales de estos sistemas es que es un sistema simple de controlar. Pero los inconvenientes son varios:

- El deslizamiento conduce a resultados pobres en odometría.
- No se dispone de un modelo preciso de giro.
- Consumen mucha potencia para girar.

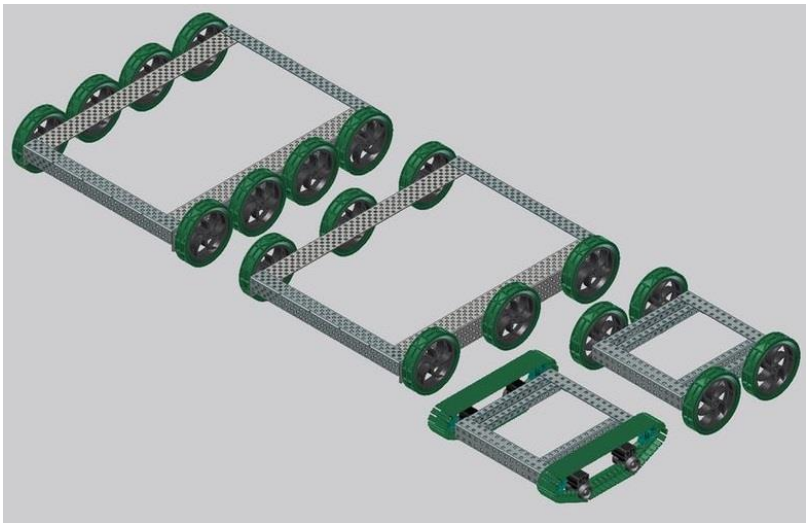


Figura - 29 – RMR por cintas de desplazamiento.

4.4.2. CINEMÁTICA DE ROBOTS MÓVILES

La cinemática de un robot móvil describe la evolución de la posición/orientación del mismo en función de las variables de actuación (vamos a desarrollarlo únicamente para RMR).

Vamos a partir de la siguiente hipótesis:

- El robot se mueve sobre una superficie plana.
- Los ejes de guiado son perpendiculares al suelo.
- Rodadura pura (No hay deslizamiento).
- El robot será considerado como un sólido rígido (no hay flexión).
- Las trayectorias se pueden aproximar como arcos de circunferencia entre dos periodos.

4.4.2.1.Modelo básico del monociclo:

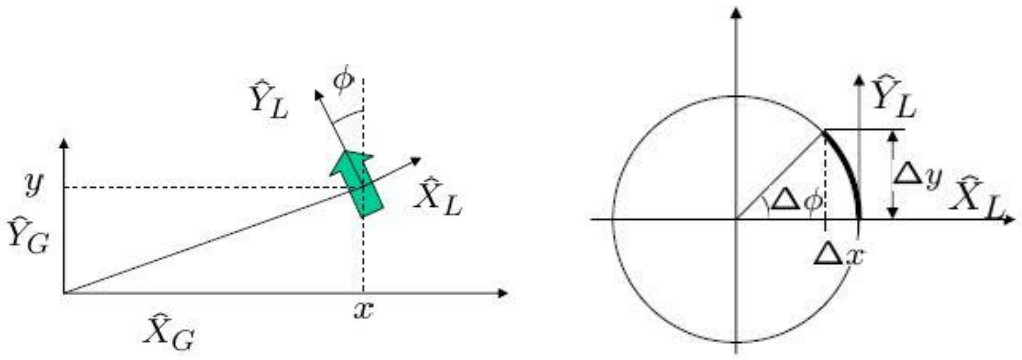


Figura - 30 - Diagrama de modelo básico de monociclo.

$$Velocidad\ lineal\ del\ vehículo \quad v = \frac{\Delta s}{\Delta t}$$

$$Velocidad\ angular\ del\ vehículo \quad \omega = \frac{\Delta\phi}{\Delta t}$$

$$En\ coordenadas\ locales \quad L(\Delta x) = -(\mathcal{R} - \mathcal{R}\cdot\cos\Delta\phi)$$
$$L(\Delta y) = \mathcal{R}\cdot\sin\Delta\phi$$

El cambio de posición en coordenadas globales lo obtenemos aplicando una rotación pura ϕ.

$$\Delta x = \mathcal{R}\cdot(1-\cos\Delta\phi)\cdot\cos\phi - \mathcal{R}\cdot\sin\Delta\phi\cdot\sin\phi$$
$$\Delta y = \mathcal{R}\cdot(1-\cos\Delta\phi)\cdot\sin\phi + \mathcal{R}\cdot\sin\Delta\phi\cdot\cos\phi$$
$$\text{Suponiendo } \Delta\phi \ll 1 \Rightarrow \cos\Delta\phi \simeq 1 \quad \sin\Delta\phi \simeq \Delta\phi$$
$$\Delta x = -\mathcal{R}\cdot\Delta\phi\cdot\sin\phi$$
$$\Delta y = \mathcal{R}\cdot\Delta\phi\cdot\cos\phi$$

O equivalentemente

$$\Delta x = -\Delta s \cdot \sin\phi$$
$$\Delta y = \Delta s \cdot \cos\phi$$

Pasando al límite podemos obtener el modelo cinemático diferencial como:

$$\dot{x} = -v \sin\phi$$
$$\dot{y} = v \cos\phi$$
$$\dot{\phi} = \omega$$

Evolución de la posición y orientación del monociclo en función de su velocidad lineal y angular de guiado (variables de actuación)

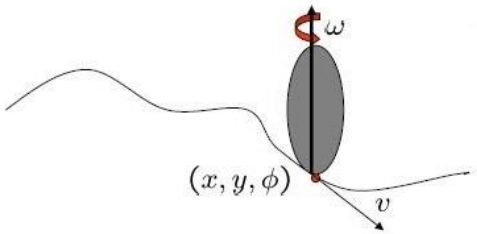


Figura - 31 - Evolución de la posición y orientación del monociclo.

4.4.2.2. Modelo cinemático Jacobiano

En general la cinemática de un robot puede explicarse como:

$$\dot{p} = J \cdot (p) \cdot \dot{q}$$

Donde:

- p Coordenadas generalizadas.
- q Variables de actuación.
- $J(p)$ Matriz Jacobiana

Si tomamos como ejemplo el monociclo definido anteriormente, obtenemos lo siguiente:

$p = (x, y, \phi)^{\mathbb{T}}$

$$\dot{p} = J(p)\dot{q} \quad \text{con} \quad J(p) = \begin{pmatrix} -\sin\phi & 0 \\ \cos\phi & 0 \\ 0 & 1 \end{pmatrix}$$

$\dot{q} = (v, \omega)^{\mathbb{T}}$

4.4.2.3.Modelo cinemático inverso (Jacobiano)

Este modelo nos permite obtener las variables de actuación necesarias para seguir una determinada trayectoria. Si partimos del modelo Jacobiano podemos obtener el modelo inverso como:

(Continuamos con el ejemplo del monociclo).

$$\dot{q} = J^*(p)\dot{p} \qquad \text{Con} \qquad J^*(p) = \left(J^{\mathbb{T}}(p) \cdot J(p)\right)^{-1} J^{\mathbb{T}}(p)$$

4.4.2.4. Modelos cinemáticos de algunas configuraciones

- Locomoción síncrona:
  Las ruedas se mueven síncronamente en velocidad y orientación por lo que el modelo es idéntico al del monociclo.

  $\dot{x} = -v \sin \phi$

  $\dot{y} = v \cos \phi$

  $\dot{\phi} = \omega$

- Locomoción diferencial:
  El modelo se reduce al del monociclo donde:

  $$v = \frac{v_L + v_R}{2} = \frac{R(\omega_L + \omega_R)}{2}$$

  $$\omega = \frac{v_R - v_L}{L} = \frac{R(\omega_L - \omega_R)}{L}$$

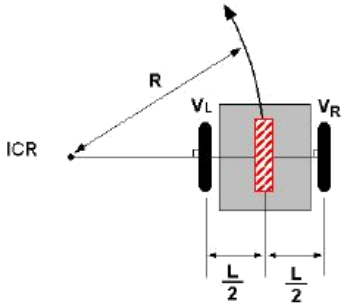


Figura - 32 - Modelo cinemático de locomoción diferencial.

- Triciclo / Bicicleta:
  El modelo se reduce al del monociclo con:

  $v = v_t \cos \alpha$

  $\dot{\alpha} = \omega_\alpha$

  Y el cambio de orientación del vehículo:

  $$\dot{\phi} = \frac{v_t}{d} \sin \alpha$$

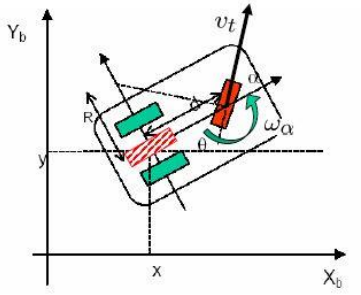


Figura - 33 - Modelo cinemático de locomoción en triciclo.

4.4.3. ACTUADORES DE LOS MRM

Un actuador es un dispositivo inherentemente mecánico cuya función es proporcionar fuerza para mover o "actuar" otro dispositivo mecánico. La fuerza que provoca el actuador proviene de tres fuentes posibles: Presión neumática, presión hidráulica, y fuerza motriz eléctrica (motor eléctrico o solenoide). Dependiendo del origen de la fuerza el actuador se denomina "neumático", "hidráulico" o "eléctrico".

Actualmente hay dos tipos de actuadores, lineales o rotatorios. Los actuadores lineales generan una fuerza en línea recta, tal como haría un pistón. Los actuadores rotatorios generan una fuerza rotatoria, como lo haría un motor eléctrico.

Es importante entender el funcionamiento de los actuadores para su correcta aplicación.

Nos centramos en los actuadores rotatorios, ya que son los que debe utilizar nuestro sistema.

El objetivo del actuador rotatorio es generar un movimiento giratorio. El movimiento debe estar limitado a un ángulo máximo de rotación:

- Actuadores de cuarto de vuelta, o 90º.
- Actuadores de fracción de vuelta, para ángulos diferentes a 90º, por ejemplo 180º.
- Actuadores multivuelta, para válvulas lineales que poseen un eje de tornillo o que requieren de múltiples vueltas para ser actuados.

La variable básica a tomar en cuenta en un actuador rotatorio es el torque o par; también llamado momento.

4.4.3.1. Actuadores eléctricos

Este tipo de actuadores presenta gran control sencillez y precisión, por lo tanto son los más utilizados en robótica.

Existen tres tipos de actuadores eléctricos:

- Motores de corriente continua DC.
- Motores paso a paso.
- Motores de corriente alterna.

Figura - 34 - Actuador eléctrico.

MOTORES DE CORRIENTE CONTINUA:

Desde el punto de vista mecánico el motor está formado por:

- Estator: parte fija.
- Rotor: parte móvil o giratoria.

El rotor es una pieza giratoria cilíndrica, un electro imán móvil, con varios salientes laterales, que llevan cada uno a su alrededor un bobinado de hilo de cobre por el que pasa la corriente eléctrica. El estator, situado alrededor del rotor, es un electroimán fijo, cubierto de un aislante. Al igual que el rotor, dispone de una serie de salientes con bobinados eléctricos por los que circula la corriente.

Para permitir el movimiento del rotor, entre rotor y estator, existe un espacio de aire llamado entrehierro, que debe ser lo más reducido posible para evitar pérdidas del flujo magnético.

Tenemos que el inductor está situado en el estator y crea un campo magnético de dirección fija.

Por otro lado el inducido está situado en el rotor y hace girar al mismo debido a la fuerza de lorentz, como combinación de la corriente circulante por él y del campo magnético de excitación.

En cuando a cómo controlar estos motores podemos dividir la clasificación en dos tipos:

- Controlados por inducido.
- Controlados por excitación.

Al aumentar la tensión del inducido aumenta la velocidad de la máquina. En el caso de control por inducido, la intensidad del inductor se mantiene constante, mientras que la tensión del inducido se utiliza para controlar la velocidad de giro.

En los controlados por excitación se actúa de manera contraria. A demás en los motores controlados por inducido se produce un efecto estabilizador de la velocidad e giro, originado por la velocidad de giro.

MOTORES PASO A PASO:

En estos motores, la señal de control son trenes de impulsos que van actuando rotativamente sobre una serie de electroimanes dispuestos en el estator. Por cada pulso recibido el rotor del motor gira un determinado número discreto de grados.

Para conseguir el giro del motor en un determinado número de grados, las bobinas del estator deben ser excitadas a una frecuencia que determina la velocidad de giro.

Se dividen en:

- Imanes permanentes. El rotor posee una polarización magnética constante, por lo que gira para orientar sus polos de acuerdo al campo magnético creado por las fases del estator.
- Reluctancia variable. El rotor está formado por un material ferromagnético que tiende a orientarse de modo que facilite el camino de las líneas de fuerza del campo magnético generada por las bobinas del estator.
- Híbridos.

MOTORES DE CORRIENTE ALTERNA:

Los podemos clasificar en:

- **Motores síncronos.** La velocidad de giro depende únicamente de frecuencia de la tensión de alimenta el inducido, para poder variar esta con precisión, el control de velocidad se realiza mediante un convertidor de frecuencia.

$$n = \frac{60 \cdot f}{P} = \frac{120 \cdot f}{p}$$

Donde:

f: $Frecuencia\ de\ la\ red\ a\ la\ que\ está\ conectada\ la\ máquina.$
P: $Número\ de\ pares\ de\ polos\ que\ tiene\ la\ máquina.$
p: $Número\ de\ polos\ que\ tiene\ la\ máquina.$
n: $velocidad\ de\ sincronismo\ de\ la\ máquina.$

- Motores asíncronos o de inducción. La corriente eléctrica del rotor necesaria para producir torsión es inducida por inducción electromagnética del campo magnético de la bobina del estator. Por lo que un motor de inducción no requiere una conmutación mecánica a parte de su misma excitación o para todo o parte de la energía trasferida del estator al rotor como en los DC y motores asíncronos.

5. HERRAMIENTAS Y PLATAFORMA UTILIZADA

5.1. DISPOSITIVOS HARDWARE

5.1.1. RASPBERRY PI (Sistema de bajo coste)

Como ya hemos comentado el hardware que vamos a utilizar para la implementación del proyecto es una Raspberry Pi.

¿Qué es una RaspberryPi? Vulgarmente se la conoce como un ordenador de placa reducida de bajo coste desarrollado por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de las ciencias de la computación en las escuelas.

Sus características básicas son:

Pequeño tamaño.

Versatilidad

Potencia

Precio

Figura - 35 - Raspberry Pi Modelo B

5.1.1.1.Descripción técnica - Hardware

Existen diferente modelos disponibles en el mercado, a continuación en la siguiente tabla resumen se concentran la mayoría de las características que diferencias dichos modelos.

	RPI Modelo A	**RPI Modelo A+**	**RPI Modelo B**	**RPI Modelo B+**	**RPI2 Modelo B**
SoC	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2836
CPU	ARM1176 JZF-S a 700 MHz	ARM1176 JZF-S a 700 MHz	ARM1176 JZF-S a 700 MHz	ARM1176 JZF-S a 700 MHz	CORTEX-A7 4 NÚCLEOS 900 MHZ.
GPU	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV
Memoria Ram	256 MB LPDDR	256 MB LPDDR	512 MB LPDDR	512 MB LPDDR	1 GB LPDDR2
Puertos USB	1	1	2	4	4
GPIO	26 Pines	40 Pines	26 Pines	40 Pines	40 Pines
Vídeo	HDMI 1.4 1920X1200	HDMI 1.4 1920X1200	HDMI 1.4 1920X1200	HDMI 1.4 1920X1200	HDMI 1.4 1920X1200
Almacenamiento	SD	MicroSD	SD	MicroSD	MicroSD
Ethernet 10/100MBPS	NO	NO	SI	SI	SI
Tamaño (mm)	85,60X56,5 MM	85,60X56,5 MM	85,60X56,5 MM	85,60X56,5 MM	85,60X56,5 MM
Peso (gramos)	45	23	45	45	45

Tabla - 2 – Modelos de Raspberry Pi existentes en el mercado y sus características

Para nuestro trabajo hemos utilizado la RPI Modelo B, la cual nos proporciona todo lo que necesitamos.

5.1.2. CÁMARA Raspberry Pi

El módulo de la cámara Raspberry Pi es un diseño personalizado para este dispositivo. Se conecta mediante un bus de datos directamente a la placa base a través de la interface dedicada CSi; diseñada especialmente para la conexión de esta cámara.

La placa del módulo de la cámara consta de unas dimensiones de 25x20x9mm. Una de sus características es su poco peso (3 g), característica que la hace idónea para aplicaciones móviles donde el peso y el tamaño es algo importante. Este sería nuestro caso, por lo que la cámara se ajusta perfectamente a nuestras necesidades.
El sensor tiene una resolución de 5 megapíxeles compuesto por una lente de foco fijo. (2.592 x 1.944 píxeles imágenes estáticas y 1080p30, 720p60 y 640x480p60 / 90 vídeo). Por último la cámara es compatible con la última versión de Raspbian (sistema operativo preferido de Raspberry Pi).

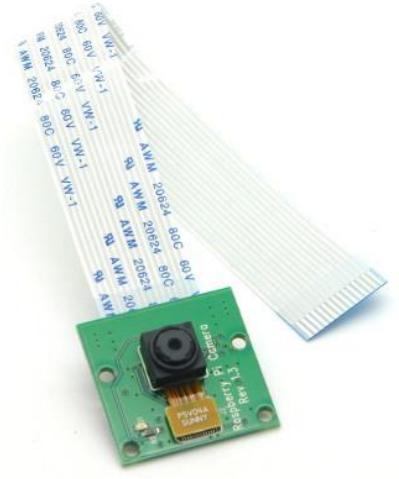


Figura - 36 – Módulo Cámara Raspberry Pi

5.1.3. GPGIO

Estos pines son una interfaz física entre el Pi y el mundo exterior. Al nivel más simple, se puede pensar en ellos como interruptores que puede activar o desactivar (entrada) o que el Pi puede activar o desactivar (salida). Diecisiete de los 26 pines son pines GPIO; los otros son pines de alimentación o de tierra.

¿Qué opciones nos dan estos pines?

Las entradas no tienen que venir de un interruptor físico; podría ser la entrada de un sensor o una señal de otro ordenador o dispositivo, por ejemplo. La salida también puede hacer cualquier cosa, desde encender un LED con el envío de una señal o de datos a otro dispositivo. Si el Raspberry Pi está en una red, puede controlar los dispositivos que están conectados a él desde cualquier lugar y esos dispositivos pueden enviar datos de nuevo. Conectividad y control de los dispositivos físicos a través de Internet es algo muy poderoso y emocionante, y la Raspberry Pi es ideal para esto.

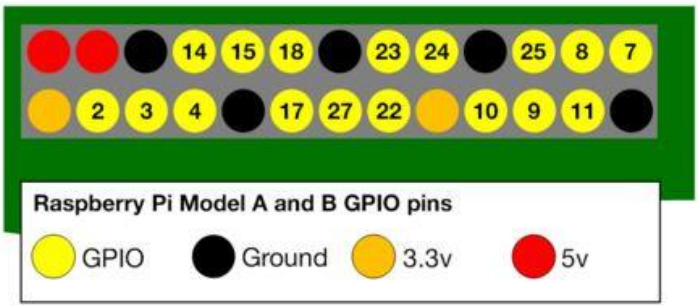


Figura - 37 - Descripción de los pines GPIO Raspberry Pi Modelos A y B.

Tenemos dos maneras de referirnos a los pines: Numeración GPIO y numeración física.

- Numeración GPIO (o BCM): Los números no tienen ningún sentido para los seres humanos, saltan sobre todo el lugar, así que no hay manera fácil de recordar.
- Numeración física: La otra manera de referirse a los pines es simplemente contando a través y por debajo de la pata 1 en la parte superior izquierda (la más cercana a la tarjeta SD). Se ve así:

Figura - 38 - Doble nomenclatura de los Pines GPIO de la Raspberry Pi.

5.1.4. ESTO DE HARDWARE NECESITADO

Para la realización del proyecto a parte de la Raspberry Pi y el módulo de la cámara, hemos necesitado los siguientes elementos:

- Cable de Ethernet.
- Cable HDMI.
- Ratón – USB.
- Teclado – USB.
- Tarjeta SD (16GB).
- Cable de alimentación de 5V.

5.2. ELEMENTOS SOFTWARE

5.2.1. SOFTWARE (RASPBIAN)

Lo normal es utilizar sistemas operativos basados en Linux, más en concreto el software que hemos instalado y que va a correr en nuestra Raspberry Pi es un Raspbian. Este Linux es una distribución derivada de Debian optimizada para el hardware de nuestra Raspberry Pi. Dicho software se lanzó en Julio de 2012 y es la distribución recomendada por la fundación para iniciarse en este entorno de desarrollo.

Un sistema operativo es el conjunto de programas básicos y utilidades que hacen que su funcionamiento Raspberry Pi. Sin embargo, Raspbian ofrece más que un SO puro; viene con más de 35.000 paquetes, software pre- compilado en un formato que hace más fácil su instalación.

Raspbian está todavía en desarrollo activo con un énfasis en la mejora de la estabilidad y el rendimiento de la mayor cantidad de paquetes de Debian como sea posible.

Raspbian no está afiliada con la Fundación Raspberry Pi. Raspbian fue creado por un pequeño equipo de desarrolladores que son fans del hardware Raspberry Pi, los objetivos educativos de la Fundación Raspberry Pi, y, por supuesto, el proyecto Debian.

5.2.2. PYTHON

La fundación Raspberry Pi recomienda el uso de Python como lenguaje de programación, sin embargo, no limitan el uso de lenguaje C, C++, Java, Scratch y Ruby ya que vienen instalados por defecto en Raspbian. Es posible utilizar cualquier software de programación siempre y cuando pueda ser ejecutado en un procesador ARMv6.

En nuestro caso hemos elegido programar en Python, ya que es un lenguaje de programación flexible y poderoso, además es ideal para aprender a programar incluso si no estás familiarizado con ningún lenguaje.

Python es de alto nivel, esto significa que codificar es muy fácil; es parecido a dar simples instrucciones en inglés. Es un lenguaje interpretado, quiere decir que es posible escribir y ejecutar directamente un programa o script sin necesidad de un compilador. Programar usando un lenguaje interpretado es en cierta medida un poco más rápido y fácil. Por ejemplo, en Python no es necesario indicar si una variable es un número, una cadena de caracteres o un arreglo; el intérprete identifica el tipo de datos al ejecutar el script.

El entorno de desarrollo integrado (IDE) para programar en Python viene incluido en Raspbian, se llama IDLE.

Figura - 39 - Icono Python en el escritorio.

Puede parecer un poco confuso ver tres iconos parecidos, esto se debe a que hay dos versiones de Python instaladas en Raspbian y una aplicación con juegos desarrollados en Python. La versión más reciente es Python 3, los cambios realizados en esta versión hacen que la versión 2 no sea del todo compatible con esta última.

6. DESARROLLO DE LA APLICACIÓN

El objetivo a alcanzar es el de detectar un objeto de forma y dimensiones preestablecidas y localizarlo en el mundo real. Una vez lo tengamos localizado realizaremos una comprobación con unos patrones previamente establecidos de posición y tamaño, para analizar hacía donde debe moverse nuestro sistema.

Tendremos configurado un tamaño y una posición de referencia que será a la posición a la que nos queremos dirigir de forma óptima. Instantáneamente capturaremos la imagen de nuestro objeto, calcularemos la posición del centro de masas y el radio del objeto y en función de los valores obtenidos realizaremos un movimiento hacia delante, hacia atrás, a la derecha o a la izquierda; o una combinación de varios de ellos.

Para ello hemos dividido la tarea en dos partes, la primera donde nos centramos en la localización del objeto, y la segunda donde nos centramos en el movimiento del sistema. En el apartado 6.3 realizamos una aplicación que aúna el trabajo de ambas partes.

6.1. LOCALIZACIÓN DEL OBJETO

Para la localización del objeto hemos realizado un estudio de diferentes métodos de detección de diferentes formas y colores. Para ellos seguiremos el siguiente diagrama de flujos el cual nos muestra los pasos básicos que debemos seguir para poder conseguir nuestros objetivos de forma exitosa.

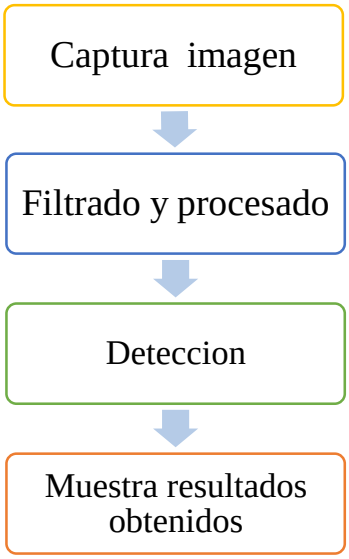


Figura - 40 – Diagrama de flujos básico de procedimiento para detectar objetos.

La forma fundamental en la que vamos a centrar el trabajo de esta memoria es la detección de formas circulares ya que hemos decidido que nuestra aplicación sigue la posición de una pelota de características dadas (tamaño, color).

Hemos dividido nuestro trabajo en dos metodologías diferentes, la primera de ellas consiste en la detección de formas, en este caso más concretamente en la detección de círculos. Esta metodología esta detallada en el apartado 6.1.1. Ahí podemos ver con detalle todas las funciones que intervienen en la aplicación anexa.

Haciendo un pequeño inciso me gustaría destacar que gracias a la función Cv2.HoughCircles() vamos a ser capaces de detectar círculos en nuestras imágenes, y de conocer su posición y su radio. Esta información será la que utilicemos para poder detectar cambios de posición de nuestro objetivo a seguir, y será gracias a la cual establezcamos los parámetros de movimiento de nuestro sistema.

La segunda de ellas es la detección de contornos y la umbralización. Toda esta metodología esta detallada en el apartado 6.1.2. Donde podemos ver las funciones que intervienen en la aplicación anexa.

Finalmente hemos desarrollado varias aplicaciones y en el apartado 7 de este proyecto podemos ver los resultados obtenidos para cada metodología.

6.1.1. DETECCIÓN DE FORMAS – CÍRCULOS.

La detección de figuras geométricas es una parte fundamental en la detección de objetos. En este apartado hemos basado nuestro trabajo básicamente en la utilización de las funciones que se explican a continuación. Para ello debemos seguir la siguiente estructura en nuestro programa:

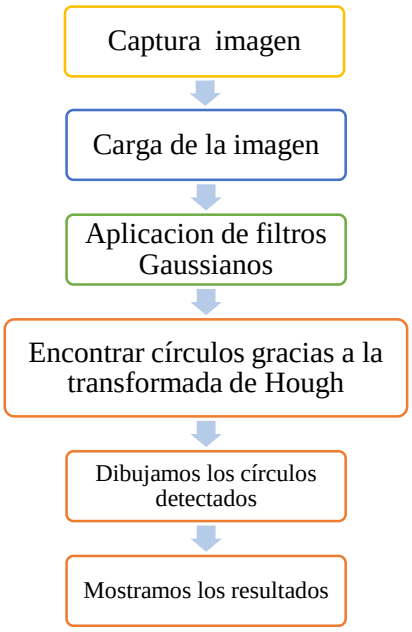


Figura - 41 - Diagrama de flujos básico de procedimiento para detectar círculos.

Las funciones que hemos utilizado para conseguir el objetivo planteado son las siguientes:

6.1.1.1. Picamera.capture ()

Para capturar una imagen en un archivo es tan simple como especificar el nombre del archivo como la salida de cualquier captura ().

```
import picamera

with picamera.PiCamera() as camera:
    camera.capture('imagen.jpg')
    camera.close()
```

```
import picamera

camera= picamera.PiCamera()
camera.capture('imagen.jpg')
camera.close()
```

Figura - 42 - scripts para captura de imágenes.

6.1.1.2. Cv2.imread()

Utilizamos la función cv2.imread () para leer una imagen. La imagen debe estar en el directorio de trabajo o debemos dar la ruta completa de donde se encuentra la imagen a leer.

```
cv2.imread(imagen, flag)
```

En segundo argumento es una bandera que especifica la manera en la imagen ha de ser leída:

- `cv2.IMREAD_COLOR`: carga una imagen en color. Cualquier transparencia de imagen se descuida. Es la bandera por defecto.
- `cv2.IMREAD_GRAYSCALE`: Cargas imagen en escala de grises.
- `cv2.IMREAD_UNCHANGED`: Una imagen transparente tiene cuatro canales - 3 de color, y una para la transparencia. Estas imágenes se pueden leer en OpenCV usando esta bandera.

Nota: En lugar de estas tres banderas, sólo tiene que pasar enteros 1, 0 o -1, respectivamente.

```
Img1 = cv2.imread ("image.jpg"   )
img2 = cv2.imread ("image.jpg", 1)
img3 = cv2.imread ("image.jpg", 0)
img4 = cv2.imread ("image.jpg",-1)
```

Figura - 43 - Ejemplo de tipos de cargas de una imagen.

En la Figura - 36 podemos ver como img1 = img2, ya que si no especificamos el valor de la bandera toma por defecto el valor 1. En este caso estaríamos leyendo la imagen como imagen a color.

En cambio img3 estaría cargando la imagen en escala de grises, e img4 la estaríamos leyendo la imagen en color con su cuarto canal de trasparencia incluido.

6.1.1.3. Cv2.medianBlur y Cv2.GaussianBlur

Estas funciones son las que aplicamos para de algún modo "filtrar" nuestra imagen y así obtener un análisis de la imagen más efectivo.

Aplicar filtro gaussiano para reducir el ruido y evitar la detección círculos falsos. (Este paso es conocido como el suavizado de imagen)

1. **Cv2.medianBlur:**
   La función suaviza una imagen con el filtro de mediana. Cada canal de una imagen multicanal se procesa independientemente.

   ```
   cv2.medianBlur(src,ksize[,dst])
   ```

   Donde:

- `Src:` Entrada de 1, 3, o una imagen de 4 canales; cuando ksize es 3 o 5, la profundidad de la imagen debe ser CV_8U, CV_16U o CV_32F, para tamaños de apertura más grandes, sólo puede ser CV_8U.
- `Dst:` matriz de destino del mismo tamaño y tipo que src.
- `Ksize:` tamaño de apertura lineal; que debe ser impar y mayor que 1, por ejemplo: 3, 5, 7 ...

2. **Cv2.GaussianBlur:**
   La función consiste en la imagen de origen con el kernel gaussiano especificado. En el lugar se admite el filtrado.

```
cv2.GaussianBlur(src,ksize,sigmaX[,dst[,sigmaY[,borderType]]])
```

Donde:

- `Src:` imagen de entrada; la imagen puede tener cualquier número de canales, que se procesan de forma independiente, pero la profundidad debe ser CV_8U, CV_16U, CV_16S, CV_32F o CV_64F.
- `Dst:` imagen de salida del mismo tamaño y tipo que `src`.
- `Ksize`: tamaño Gaussian kernel. ksize.width y ksize.height pueden diferir, pero ambos deben ser positivos y extraño. O bien, pueden ser cero y después de que se calculan a partir sigma.
- `Sigmax:` desviación estándar del núcleo de Gauss en dirección X.
- `SigmaY`: desviación estándar del núcleo de Gauss en la dirección Y; Si sigmaY es cero, se establece para que sea igual tosigmaX, si ambos son ceros sigmas, que se calculan a partir ksize.width y ksize.height,; para controlar totalmente el resultado sin tener en cuenta las posibles modificaciones futuras de todos esta semántica, se recomienda especificar todos ksize, Sigmax y sigmaY.
- `BorderType`: método de extrapolación de píxeles.

6.1.1.4. Cv2.HoughCircles

Esta función encuentra los círculos en una imagen en escala de grises utilizando la transformada de Hough. Solo puede aplicarse a imágenes de 8 bits, de un solo canal, o lo que es lo mismo, la imagen de entrada en escala de grises.

Como salida obtenemos un vector de salida por cada círculo que se encuentra. Cada vector se codifica como un vector de punto flotante 3-elemento (**x, y, radio** → Esto es muy importante ya que nos servirá para localizar nuestro objeto).

```
cv2.HoughCircles(image,method,dp,minDist[,circles[,param1[,param2[,
                                minRadius[,maxRadius]]]]])
```

Donde:

- `Image`: de 8 bits, la imagen de un solo canal. Si vamos a trabajar con una imagen en color, previamente la tenemos que convertir a escala de grises.
- `Method`: Define el método para detectar los círculos en las imágenes. Actualmente, el único método implementado es cv2.HOUGH_GRADIENT. Será por lo tanto el método que utilicemos.
- `Dp`: Este parámetro es la relación inversa de la resolución del acumulador a la resolución de la imagen. Esencialmente, cuanto mayor sea el dp consigue, cuanto menor sea la matriz acumulador consigue.
- `MinDist`: Distancia mínima entre el centro (x, y) las coordenadas de los círculos detectados. Si el DISTMÍN es demasiado pequeño, podemos detectar falsamente varios círculos vecinos. Si por el contrario el MinDist es demasiado grande, entonces algunos círculos pueden no ser detectados en absoluto.

- `Param1:` valor del gradiente utilizado para manejar la detección de bordes.
- `Param2:` valor umbral del acumulador para el método cv2.HOUGH_GRADIENT. Cuanto más pequeño es, detectamos un mayor número de falsos círculos.
- `MinRadius:` Tamaño mínimo del radio (en píxeles).
- `MaxRadius:` Tamaño máximo del radio (en píxeles).

6.1.1.5. Cv2.cvtColor

Para poder utilizar la función HoughCircles necesitamos previamente que la imagen que vamos a tratar venga en escala de grises. Para ellos podemos usar entre otras la siguiente función:

```
cv.CvtColor(src, dst, code)
```

Donde:

- `Src:` imagen de entrada: 8 bits sin signo, de 16 bits sin signo (CV_16UC ...), o de precisión simple de punto flotante.
- `Dst:` imagen de salida del mismo tamaño y profundidad como src.
- `Código:` código de color conversión del espacio (véase la descripción más adelante).
- `DstCn`: número de canales de la imagen de destino en; si el parámetro es 0, el número de los canales se deriva automáticamente a partir de src y el código.

En cuanto a los distintos códigos que podemos tener destacamos los siguientes: CV_BGR2Luv, CV_RGB2GRAY, CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB.

En nuestro caso para poder trasformar a escala de grises la imagen usaremos la función CV_RGB2GRAY.

Nota: Otra forma de pasar una imagen a escala de grises es utilizando la función `cv2.imread` utilizando la bandera 0; como ya hemos visto en el apartado 4.1.1.2.

6.1.1.6. Cv2.circle()

Esta función sirve para dibujar un círculo simple o lleno de un centro y un radio dado.

```
Cv2.circle(img,center,radius,color,thickness=1,lineType=8,shift=0)
```

Donde:

- `Img:` Imagen donde se dibuja el círculo.
- `Centro`: Centro del círculo.
- `Radio`: Radio del círculo.
- `Color`: de color Círculo.
- `Espesor`: Espesor del círculo contorno, si es positivo. Espesor negativo significa que un círculo relleno se va a dibujar.
- `Tipo de línea:` Tipo del círculo límite. Ver la descripción de línea ().
- `Cambiar`: Número de bits fraccionales en las coordenadas del centro y en el valor del radio.

6.1.1.7. Cv2.imshow()

Esta función sirve para mostrar una imagen por pantalla, creando una ventana emergente con la imagen que deseamos ver.

```
Cv2.imshow(nombre,image)
```

Donde:

- `Nombre`: Nombre de la ventana que mostrará la imagen.
- `Imagen`: imagen que queremos mostrar.

6.1.2. DETECCIÓN DE CONTORNOS

El método que hemos seleccionado para la detección de contornos en una imagen es el basado en la umbralización de imágenes. Para el desarrollo de esta aplicación hemos seguido el siguiente flujo

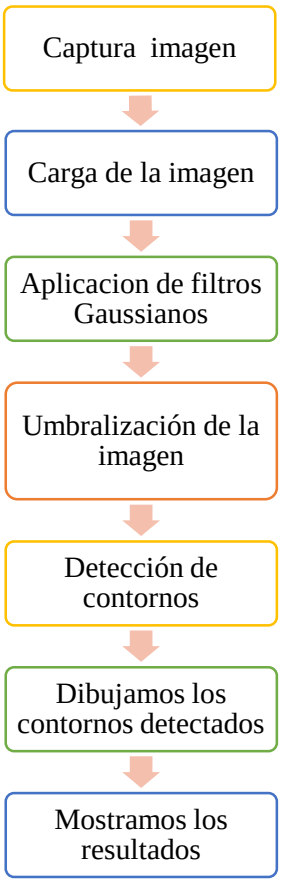


Figura - 44 - Diagrama de flujos básico de procedimiento para detectar contornos.

Por lo que para el desarrollo de este método hemos añadido la utilización de las siguientes funciones:

6.1.2.1. Cv2.threshold()

La función umbral convierte una imagen en escala de grises en una imagen binaria. Es muy útil para la segmentación de imágenes, creación de marcadores, máscaras, etc.

```
Cv2.threshold(src, thresh, maxval, type[,dst])
```

Donde:

- `Src:` matriz de entrada (de un solo canal, 8 bits o de punto flotante de 32 bits).
- `Dst:` matriz de salida del mismo tamaño y tipo que `src`.
- `Trillar:` valor de umbral.
- `Maxval:` valor máximo de usar con la THRESH_BINARY y tipos de umbrales THRESH_BINARY_INV.
- `Tipo:` de umbralización.
    - cv2.THRESH_BINARY

- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

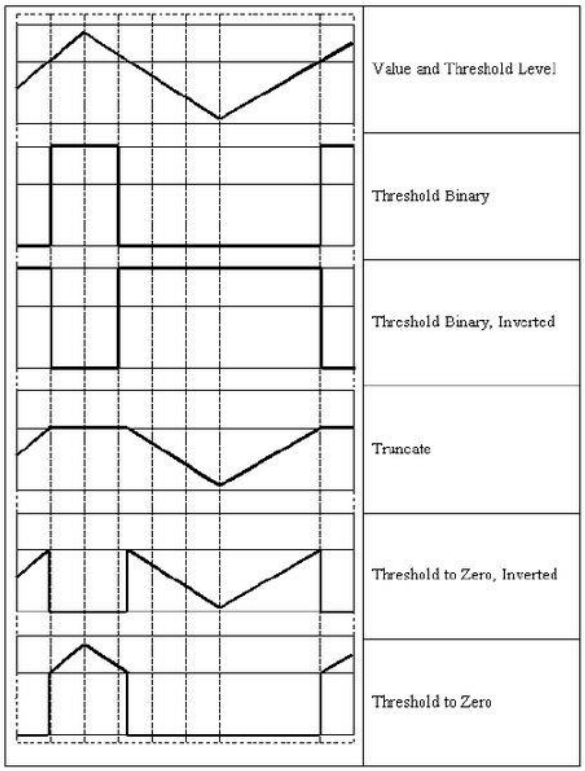


Figura - 45 - Representación gráfica de los distintos tipos de umbralización de imágenes.

Como variante para la umbralización hemos utilizado la siguiente función:

6.1.2.2. Cv2.Canny()

Para encontrar los bordes de una imagen utilizamos la siguiente función:

```
cv2.Canny(image,threshold1,threshold2[,edges[,apertureSize
                    [,L2gradient]]])
```

Donde:

- `Image`: imagen de entrada de un solo canal de 8 bits.
- `Threshold1`: primer umbral para el procedimiento de histéresis.
- `Threshold2`: segundo umbral para el procedimiento de histéresis.
- `ApertureSize`: tamaño de la abertura para el operador Sobel ().
- `L2gradient`: una bandera, que indica si una norma más precisa se debe utilizar para calcular la magnitud del gradiente de la imagen (L2gradient = true), o si la norma predeterminada es suficiente (L2gradient = false).

6.1.2.3. Cv2.findContours()

Los contornos se pueden explicar simplemente como una curva que une todos los puntos continuos (a lo largo de la frontera), que tiene el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de la forma y la detección de objetos y reconocimiento.

Para una mayor precisión, utilizamos imágenes binarias. (Por eso previamente aplicaremos la función cv2.threshold() o la función cv2.Canny()).

La función cv2.findContours() modifica la imagen de origen.

En OpenCV, la búsqueda de contornos es como encontrar objeto blanco de fondo negro. Así que recuerda, objeto que se encuentran deben ser de color blanco y el fondo debe ser de color negro.

```
cv2.findContours(image,mode,method[,contours[,hierarchy[,offset]]])
```

Donde:

- `Image:` una imagen de un solo canal de 8 bits. Los píxeles no-cero son tratados como 1 de. La imagen se trata como binario.
  Puede utilizar compare (), inRange (), umbral (), adaptiveThreshold (), Canny (), y otros para crear una imagen binaria de una escala de grises o color una. Si el modo es igual a CV_RETR_CCOMP o CV_RETR_FLOODFILL, la entrada también puede ser una imagen entero de 32 bits de etiquetas (CV_32SC1).
- `Contours:` Cada contorno detectado se almacena como un vector de puntos.
- `hierarchy:` vector de salida opcional, que contiene información acerca de la topología de la imagen. Tiene tantos elementos como el número de contornos.
- `Mode:` Modo de recuperación de contornos.
  - CV_RETR_EXTERNAL recupera sólo el contorno exterior extremo. Establece jerarquía para todos los contornos.
  - CV_RETR_LIST recupera todos los contornos sin establecer relaciones jerárquicas.
  - CV_RETR_CCOMP recupera todos los contornos y los organiza en una jerarquía de dos niveles. En el nivel superior, hay límites externos de los componentes. En el segundo nivel, hay límites de los agujeros. Si hay otro contorno dentro de un agujero de un componente conectado, todavía se pone en el nivel superior.
  - CV_RETR_TREE recupera todos los contornos y reconstruye una jerarquía completa de los contornos anidados.
- `Method`: método de aproximación.
  - CV_CHAIN_APPROX_NONE almacena absolutamente todos los puntos de contorno. Es decir, cualquiera de los 2 puntos posteriores (x1, y1) y (x2, y2) del contorno será o bien vecinos horizontal, vertical o diagonal, es decir, max (abs (x1-x2), abs (y2-y1)) == 1.
  - CV_CHAIN_APPROX_SIMPLE comprime segmentos horizontales, verticales y diagonales y deja sólo los puntos finales. Por ejemplo, un contorno rectangular arriba-derecha se codifica con 4 puntos.
  - CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS aplica uno de los sabores del algoritmo de aproximación cadena Teh-Chin.

- `Offset:` (Opcional) Offset por la que se desplaza cada punto del contorno. Esto es útil si los contornos se extraen de la ROI imagen y entonces deben ser analizados en el contexto global de la imagen.

6.1.2.4. Cv2.drawContours()

Para dibujar los contornos, se utiliza la función cv2.drawContours (). También se puede utilizar para dibujar cualquier forma siempre y cuando tenga sus puntos de frontera.

```
cv2.drawContours(image,contours,contourIdx,color[,thickness[,lineTyp
            e[,hierarchy[,maxLevel[,offset]]]]])
```

Donde:

- `Image:` la imagen de destino sobre la que vamos a dibujar los contornos.
- `Contornos:` Todos los contornos de entrada. Cada contorno se almacena como un vector de punto.
- `ContourIdx:` Parámetro que indica un contorno a dibujar. Si es negativo, todos los contornos se dibujan.
- `Color:` El color de los contornos.
- `Thinckness:` Espesor de las líneas de los contornos que vamos a dibujar. Si es negativo (por ejemplo, espesor = CV_FILLED), los interiores de nivel se dibujan (o lo que es lo mismo se rellenan).
- `LineType:` línea de unión.
- `Hierarchy:` (Opcional) información sobre las jerarquías. Será necesario si solo desea dibujar algunas de las curvas de nivel (ver maxLevel).
- `maxLevel`: nivel máximo de contornos dibujados. Si es 0, sólo se dibuja el contorno especificado. Si es 1, la función dibuja el contorno (s) y todos los contornos anidados. Si es 2, la función dibuja los contornos, todos los contornos anidados, todos los contornos-anidados para-anidada, y así sucesivamente. Este parámetro sólo se tiene en cuenta cuando hay jerarquías disponibles.
- `Desplazamiento:` (opcional) parámetro de cambio contorno. Cambie todos los contornos dibujados por la especificada \ texttt {compensar} = (dx, dy).
- `Contorno:` Puntero al primer contorno.
- `ExternalColor:` Color del contorno externo.
- `holeColor:` Color de contornos internos (agujeros).

6.1.3. MÁS FUNCIONES UTILIZADAS.

A continuación vamos a describir el resto de funciones que hemos utilizado para ambas metodologías. Son funciones básicas de openCV que se utilizan cuando programas en Python.

6.1.3.1. Cv2.waitKey()

Es una función de unión teclado, o lo que es lo mismo espera o bien un tiempo especificado previamente o la recepción de la pulsación de una tecla (una en concreto o cualquiera, según se configure).

Su argumento es el tiempo en milisegundos. La función de espera milisegundos especificados para cualquier evento de teclado. Si pulsa cualquier tecla en ese momento, el programa continúa. Si se pasa 0, espera indefinidamente para un golpe de tecla. También se puede configurar para detectar pulsaciones de teclas específicas.

6.1.3.2. Cv2.destroyAllWindows()

Esta función simplemente destruye todas las ventanas que hemos creado.

Si queremos destruir cualquier ventana específica, utilizamos la función cv2.destroyWindow función (), donde se pasa el nombre exacto ventana como argumento.

6.1.3.3. Cv2.resize()

Con esta función lo que conseguimos es redimensionar imágenes.

```
cv2.resize(src,dsize[,dst[,fx[,fy[,interpolation]]]])
```

Donde:

- `Src:` imagen de entrada.
- `Dst:` imagen de salida; tiene la DSIZE tamaño (cuando no es cero) o el tamaño calculado a partir de src.size (), fx, fy y; el tipo de dst es el mismo que de src.
- `DSIZE:` tamaño de la imagen de salida; si es igual a cero, se calcula como:

```
dsize = Size(round(fx*src.cols), round(fy*src.rows))
```

De cualquier DSIZE o ambos fx y fy debe ser distinto de cero.

- `Fx`; factor de escala a lo largo del eje horizontal; cuando es igual a 0, se calcula como:
  `(double)dsize.width/src.cols`
- `Fy`: factor de escala a lo largo del eje vertical; cuando es igual a 0, se calcula como:
  `(double)dsize.height/src.rows`
- `Interpolación`: métodos de interpolación:
  - INTER_NEAREST: interpolación del vecino más cercano.
  - INTER_LINEAR: interpolación bilineal (usado por defecto)
  - INTER_AREA: remuestreo usando relación área de píxeles. Cuando se amplía la imagen, es similar al método INTER_NEAREST.
  - INTER_CUBIC - una interpolación bicúbica sobre barrio 4x4 píxeles.
  - INTER_LANCZOS4 - una interpolación Lanczos sobre el vecindario 8x8 píxeles.

6.1.3.4. Cv2.equalizeHist()

Esta función la utilizamos para ecualizar el histograma de una imagen en escala de grises. El algoritmo normaliza el brillo y aumenta el contraste de la imagen.

```
cv2.equalizeHist(src[, dst])
```

Donde:

- `Src:` imagen de entrada de un único canal de 8 bits.
- `Dst:` imagen de destino del mismo tamaño y tipo que src.

6.2. MOVIMIENTO DEL VEHICULO

Para el movimiento del vehículo hemos diseñado el siguiente sistema:

Mediante la Raspberry Pi vamos a generar una onda PWM para controlar la velocidad de un motor de corriente continua. Esta señal de salida la obtenemos gracias a la GPGIO.

Esta señal la pasaremos por el chip controlador de motores L293D, gracias al cual obtendremos la intensidad necesaria y podremos invertir la dirección de giro de nuestro motor.

La salida del chip la conectaremos directamente al motor de continua para así generar el movimiento necesario en nuestro sistema.

Requisitos:

- Raspberry Pi (GPIO Pins).
- Placa protoboard.
- Chip L293 o SN755410.
- Cables para realizar las conexiones necesarias.
- Dos motores de corriente continua nominal de 6V.
- 4 baterías AA y un adaptador.

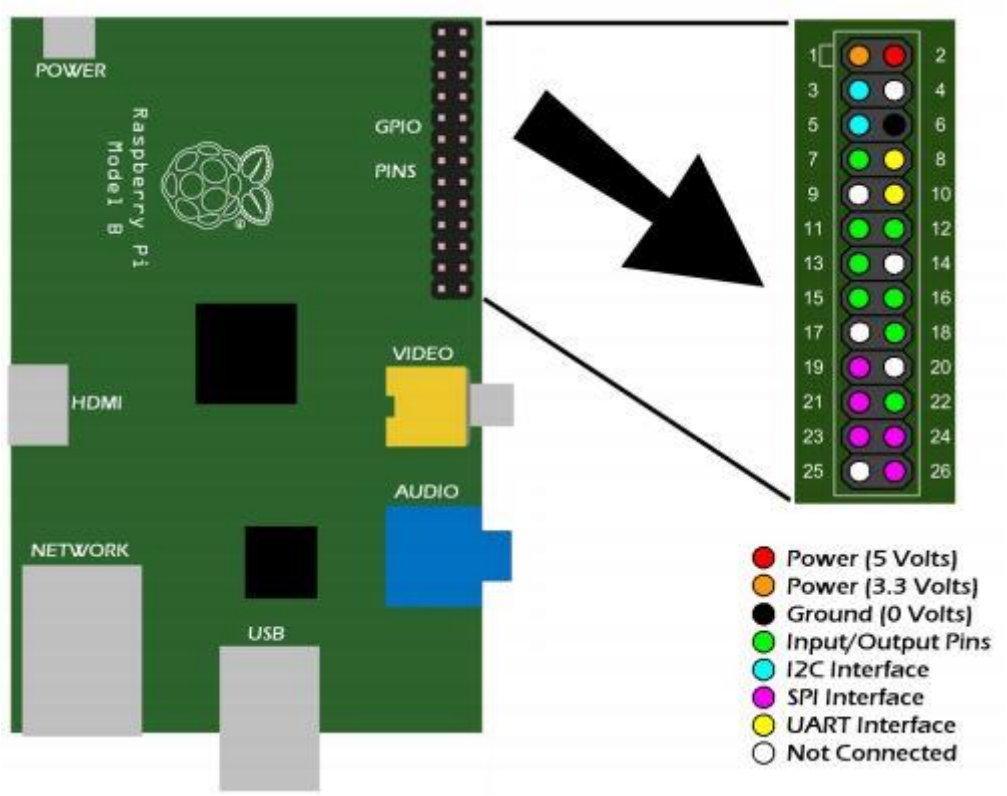


Figura - 46 - Descripción de los pines GPIO Raspberry Pi.

6.2.1. CHIP L293D

El chip L293D nos las opción de controlar dos motores de forma independiente. El chipo nos permite que una fuente de alimentación independiente alimente tanto a los motores como al chip en sí. Esto es importante ya que la potencia que nos ofrece la GPIO de la Raspberry Pi es suficiente para alimentar el chip, pero no es suficientemente potente para activar los motores.

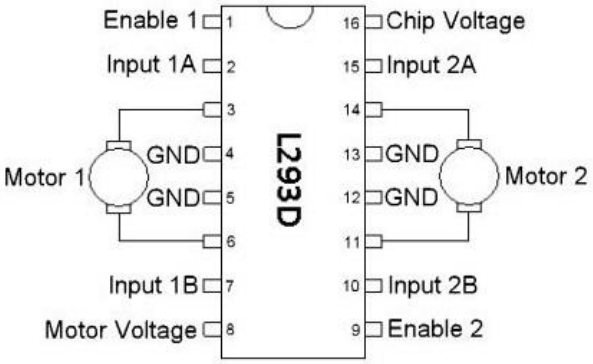


Figura - 47 - Diagrama del chip L293D.

El chip dispone de tres señales de control de entrada, la entrada Enable, la entrada A y la entrada B. Gracias a esto somos capaces de controlar la dirección de giro del motor como podemos ver en la siguiente tabla:

Enable	Input A	Input B	Motor Behaviour
Off	On or Off	On or Off	Stopped
On	On	Off	Turn Forwards
On	Off	On	Turn Backwards
On	Off	Off	Stopped
On	On	On	Stopped

Figura - 48 - Control de motores DC con chip L293D.

Por lo que podemos observar que para que nuestro motor gire tenemos que cumplir las siguientes condiciones:

En primer lugar la entrada Enable ha de estar en ‘on’. (Si esta entrada no está en ‘on’ nunca se moverá nuestro motor).

Una vez que cumplimos esta primera condición, tenemos que tener que la entrada A y la entrada B han de tener valores opuestos, o lo que es lo mismo si A esta a ‘on’ B debe estar a off para que el motor gire, y al contrario.

Ya que si las entradas A y B tienen el mismo valor, el motor permanece parado.

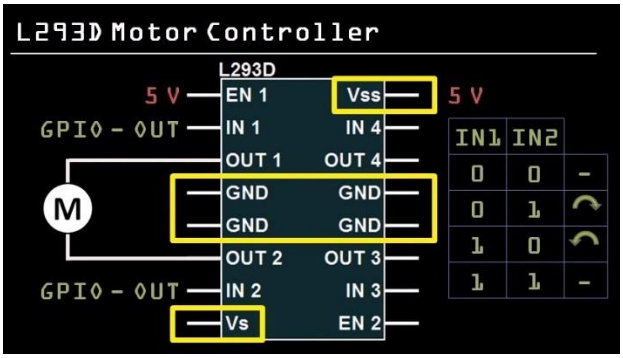


Figura - 49 - Resumen control de motor DC con L293D.

El siguiente paso sería realizar las conexiones pertinentes, para ello es muy importante cerciorarse de que nuestra Raspberry Pi está apagada y nuestra fuente de alimentación también lo está para evitar cortocircuitos.

El procedimiento a seguir es el siguiente:

- Colocamos el chip L293D en nuestra placa protoboard.
- Conectamos el pin número 6 de GPIO a la cabecera de la protoboard. (Cable negro, conexión de tierra).
- Conectamos el pin número 2 de GPIO a la barra de alimentación de la protoboard del lado contrario. (Cable rojo, alimentación).
- Conectamos el pin número 16 del chip L293D a la barra de alimentación de la protoboard. (Mediante un cable naranja).
- Conectamos el pin número 9 del chip L293D a la barra de alimentación de la protoboard. (Mediante un cable purpura).
- Conectamos el pin número 5 del chip L293D al carril de tierra de la parte izquierda de la placa protoboard. (Cable negro).
- Conectamos el pin número 16 de GPIO al pin número 15 del chip L293D. (Cable verde).
- Conectamos el pin número 18 de GPIO al pin número 10 del chip L293D.
- Conectamos el motor a los pines número 11 y 14 del chip L293D.
- Conectamos la batería a los carriles de alimentación de la placa protoboard, de modo que el cable negro lo conectamos a tierra y el rojo a la alimentación.
- Conectamos el pin número 8 del chip L293D a la barra de alimentación (Mediante un cable rojo).

El resultado obtenido es el siguiente:

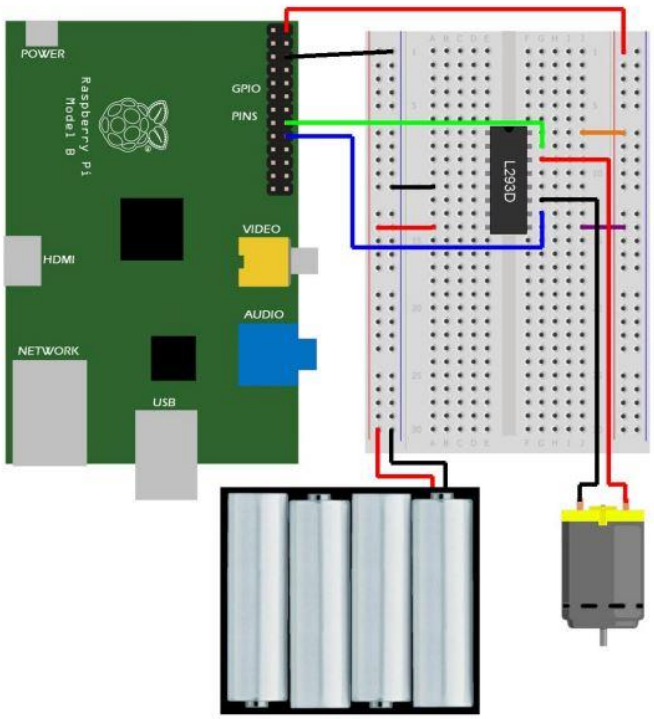


Figura - 50 - Conexiones del sistema para un motor DC.

6.2.2. CONTOL DE LA VELOCIDAD DEL MOTOR PWM

Lo que vamos a realizar es controlar la velocidad del motor utilizando modulación de ancho de pulso o lo que es lo mismo PWM.

Se podría pensar que la mejor manera de controlar la velocidad de un motor sería variar la tensión, pero el uso de este método es problemático porque se necesita una tensión mínima para obtener el comienzo de la rotación del motor, y si la tensión se reduce a continuación para hacer que el motor funcione lentamente, el par producido por el motor a bajos voltajes es muy pequeño, lo que no nos resultaría útil.

La mejor manera de controlar la velocidad del motor es cambiar muy rápidamente la alimentación del motor de encendido a apagado.

Definimos el ciclo de trabajo como el tiempo que la tensión esta activa dentro de un periodo.

La trama de tensión frente al tiempo para ciclos de trabajo de 100%, 90%, 50%, 10% y 0% son de la siguiente manera:

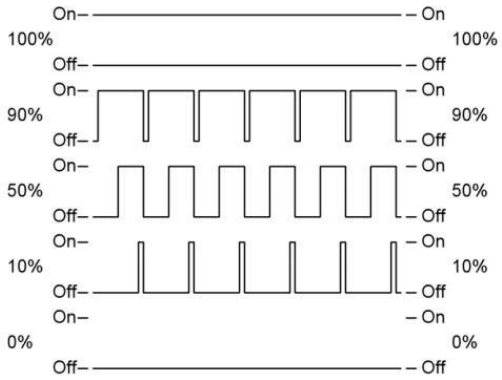


Figura - 51 - Diferentes ciclos de trabajo.

La Raspberry Pi es capaz de proporcionarnos una salida PWM en el pin 22 de GPIO, esta salida se la aplicamos al pin "Enable" (pin 9 en el gráfico de la Figura 45), y así el chip modulará la potencia enviada al motor.

Por lo que según la configuración anterior, tenemos que quitar el cable purpura, y conectar en el Pin número 9 del chip L293D, a la clavija número 22 de la GPIO. (Mediante el cable amarillo).

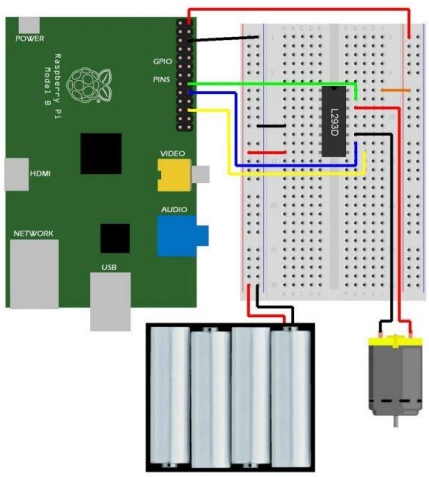


Figura - 52 - Conexiones del sistema para un motor DC con señal PWM.

6.3. APLICACIÓN

Hemos realizado diversas pruebas y estudios sobre cómo llevar a cabo la localización de nuestro objeto, y debido a nuestras necesidades establezco como la opción más óptima, la de la detección de círculos utilizando la función Cv2.HoughCircles, ya que gracias a esta función somos capaces de posicionar nuestro objeto y saber las dimensiones lo mismo (nos facilita las coordenadas x e y del centro del círculo detectado junto con el radio del mismo).

Vamos a usar esta información para poder saber hacia dónde ha de dirigirse nuestro sistema; para ello debemos establecer unas coordenadas y un radio dado como referencia, para así saber si nuestro sistema ha de moverse hacia adelante, hacia atrás, a la derecha o a la izquierda.

Para unos valores de:

Radio = 100
Coordenada x = 350
Coordenada y =300.

C.M.ref = (coordenada x, coordenada y) = (350,300).
Tamaño = size = Radio = r = 100.

Utilizaremos estos valores como nuestra posición objetivo. Para ello la aplicación ha de seguir el diagrama de flujo que se muestra en la Figura 53.

Como nos muestra el diagrama en primer lugar realizamos la operación de detección del objeto, almacenando las variables C.M. (x, y) y por otro lado el tamaño (r).

En primer lugar evaluaremos el C.M. (centro de masas que viene determinado por (x, y) de la imagen actual, y lo compararemos con C.M.ref.

En función de los resultados obtenidos moveremos el sistema hacia la derecha o hacia la izquierda, o permanecerá quieto.

El movimiento hacia la derecha o hacia la izquierda lo conseguimos actuando de manera independiente uno de los dos motores de los que dispone nuestro sistema. Si deseamos girar a la derecha debemos dejar quieto el motor que actúa nuestra rueda derecha y mover el de la rueda izquierda. Y si deseamos girar a la izquierda al contrario, fijaremos el motor que actúa la rueda izquierda y actuaremos el que gira la rueda derecha. En ambos casos los motores se moverían hacia adelante.

En el anexo del código podemos ver este caso explicado.

En segundo lugar evaluaremos el tamaño del radio obtenido en la fotografía, por lo que si r es mayor debemos retroceder con nuestro sistema, y si es menor debemos avanzar para acercarnos.

Una vez corregida la posición, volvemos a empezar, ya que nuestro objeto puede estar en movimiento y debemos ser capaces de seguirlo.

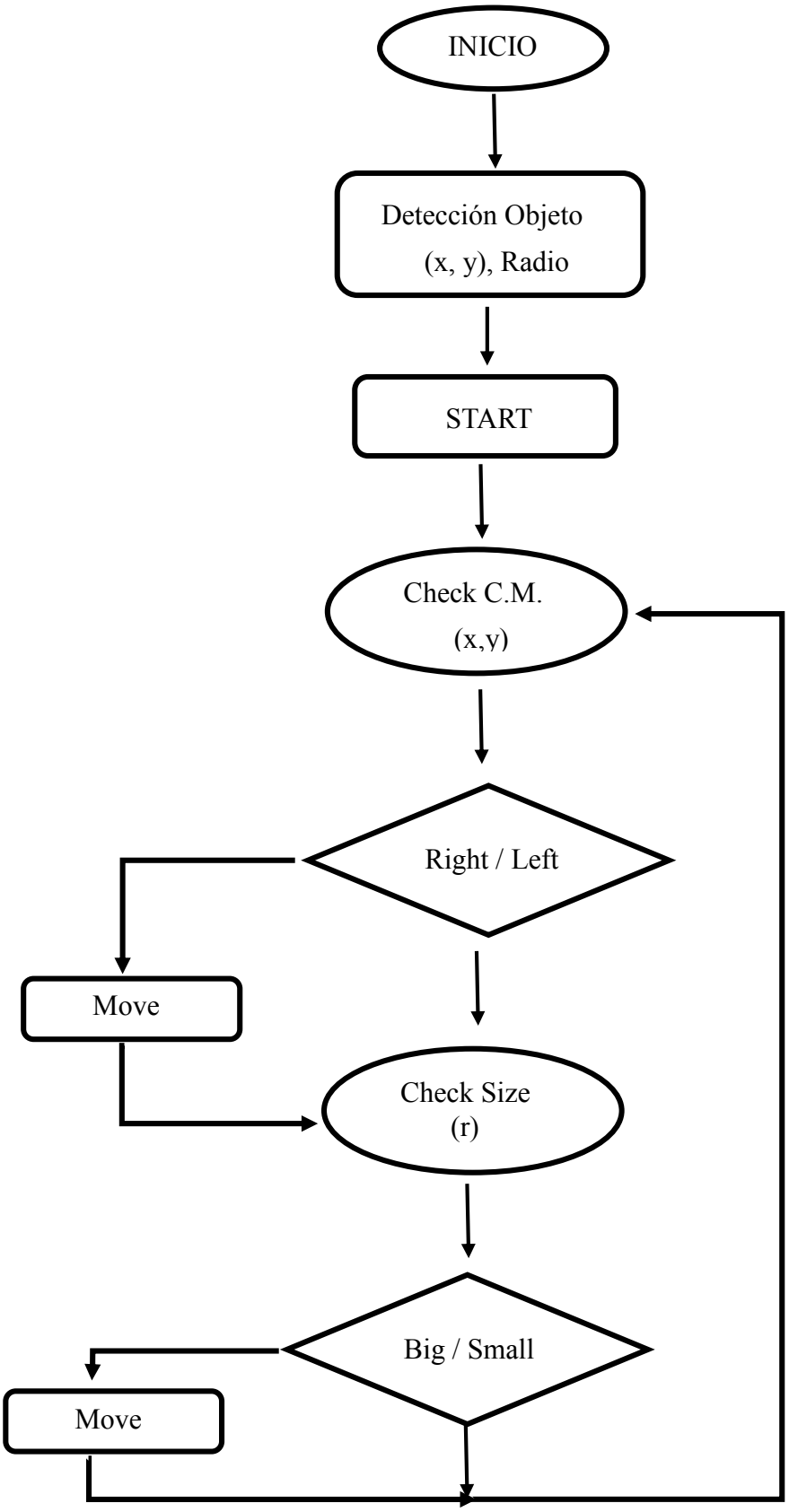


Figura - 53 - Diagrama de flujo de la aplicación.

7. RESULTADOS, PROBLEMÁTICA Y LIMITACIONES OBTENIDOS

7.1. LOCALIZACIÓN DEL OBJETO

En el siguiente apartado vamos a detallar con imágenes los resultados obtenidos tras el estudio y análisis de las funciones explicadas en el apartado 4.

7.1.1. BUSQUEDA DE CONTORNO

- En primer lugar tenemos que jugar con el brillo y el contraste de la imagen para que se ajuste a nuestras necesidades. En la siguiente figura vemos imágenes de nuestra muestra tomadas con diferentes combinaciones de las funciones `cv2.equalizeHist()` y `cv2.medianBlur()`.

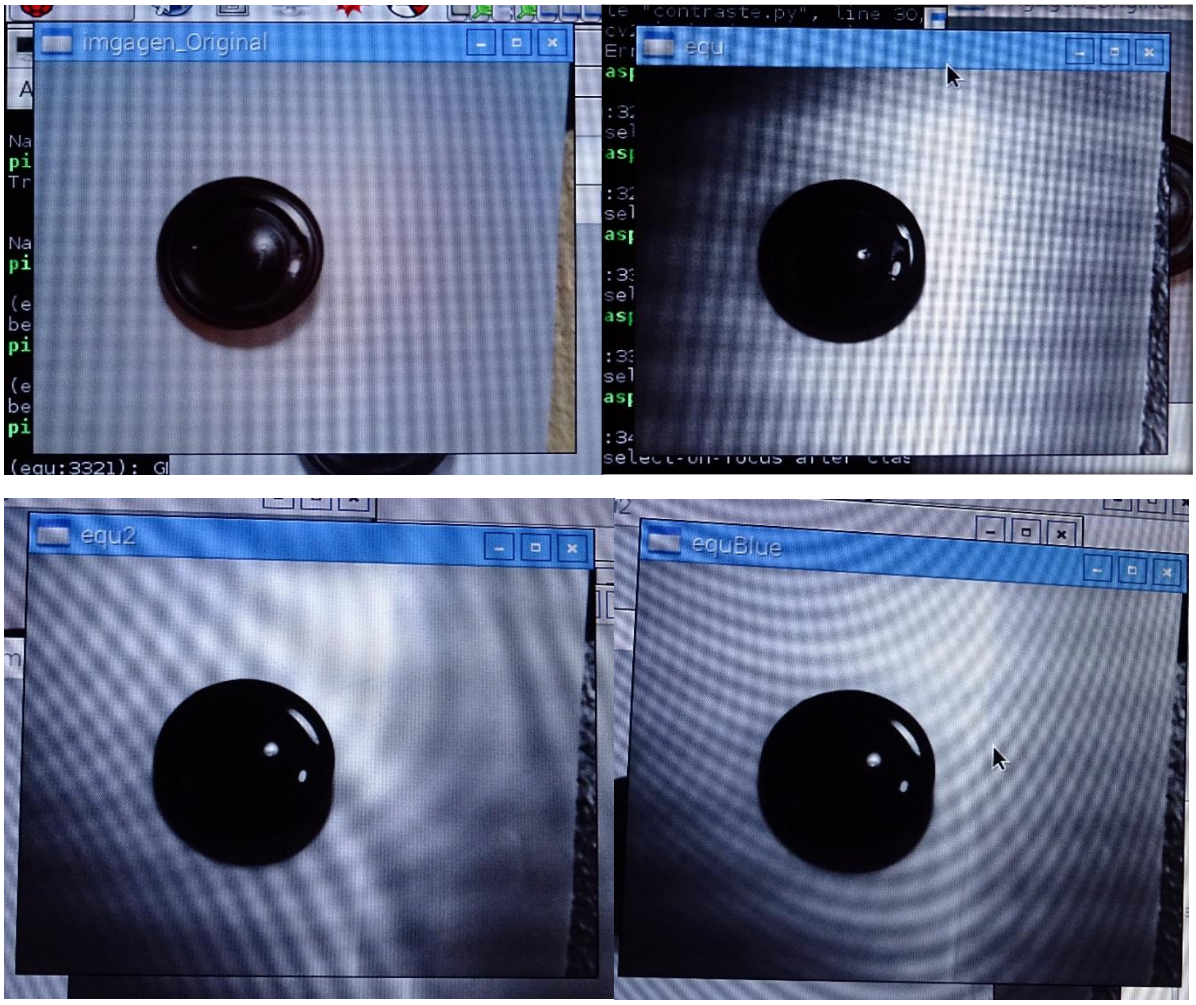


Figura - 54 - Ecualización de la imagen original

Podemos observar como el mejor resultado nos lo aporta la imagen equ2, la cual resulta de en primer lugar aplicar el filtro cv2.medianBlur(img_gris) y al resultado de este filtro le aplicamos la función cv2.equalizeHist().

- **Pruebas utilizando diferentes tipos de umbralización.**
  En primer lugar vamos a mostrar los resultado obtenidos utilizando la función **`cv2.Canny()`** con diferentes valores de umbralización máxima y mínima.

  En este caso vamos a aplicar la función canny a la anterior imagen ya ecualizada. Un resultado bastante bueno para nuestras necesidades sería o bien en el que los niveles de umbralización máximos y mínimos toman el mismo valor (120) o en el que tenemos (80-120). En los siguientes apartados veremos cuál de los dos es el más óptimo para nuestras necesidades.

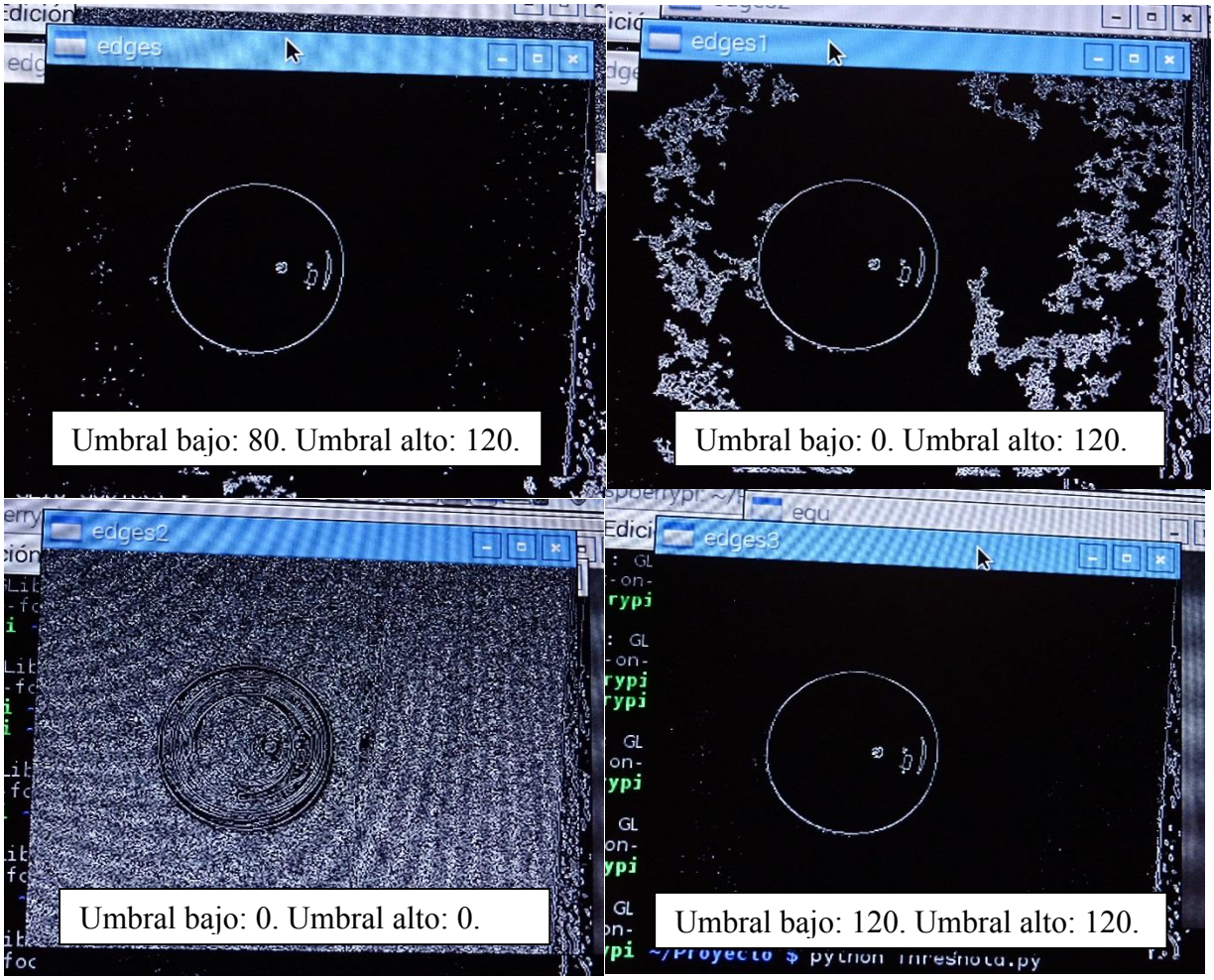


Figura - 55 - Resultados obtenidos de la función cv2.Canny().

Si a estos resultados obtenidos le aplicamos la función cv2.findContours() y la cv2.drawContours(). Obtenemos lo siguientes contornos:

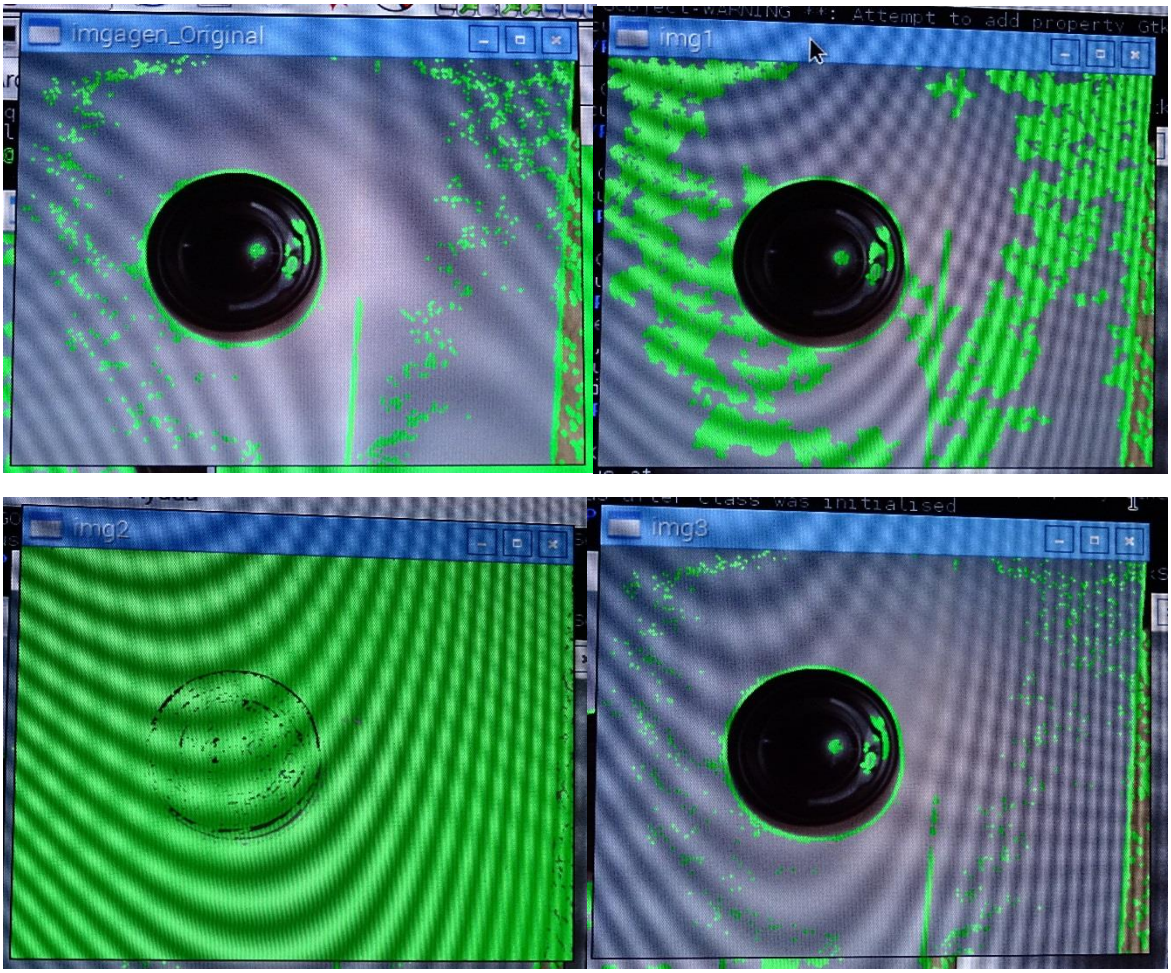


Figura - 56 - Resultados de aplicar las funciones Cv2.findContours() y cv2.drawContours().

Ahora si podemos observar claramente como el mejor resultado de los 4 es el último en el que en la función cv2.Canny() aplicábamos 120 para el umbral mínimo y máximo.

En segundo lugar vamos a mostrar los resultado obtenidos utilizando la función **`cv2.threshold()`** con diferentes valores de umbralización máxima y mínima. Esta función la vamos a probar usando los siguientes tipos de umbralización:

- cv2.THRESH_BINARY (thresh1)
- cv2.THRESH_BINARY_INV (thresh2)
- cv2.THRESH_TRUNC (thresh3)
- cv2.THRESH_TOZERO (thresh4)
- cv2.THRESH_TOZERO_INV (thresh5)

A continuación comprobaremos que la mejor opción en nuestro caso no está clara. Una vez que apliquemos la función de búsqueda de contornos será cuando tengas clara la mejor opción a escoger.

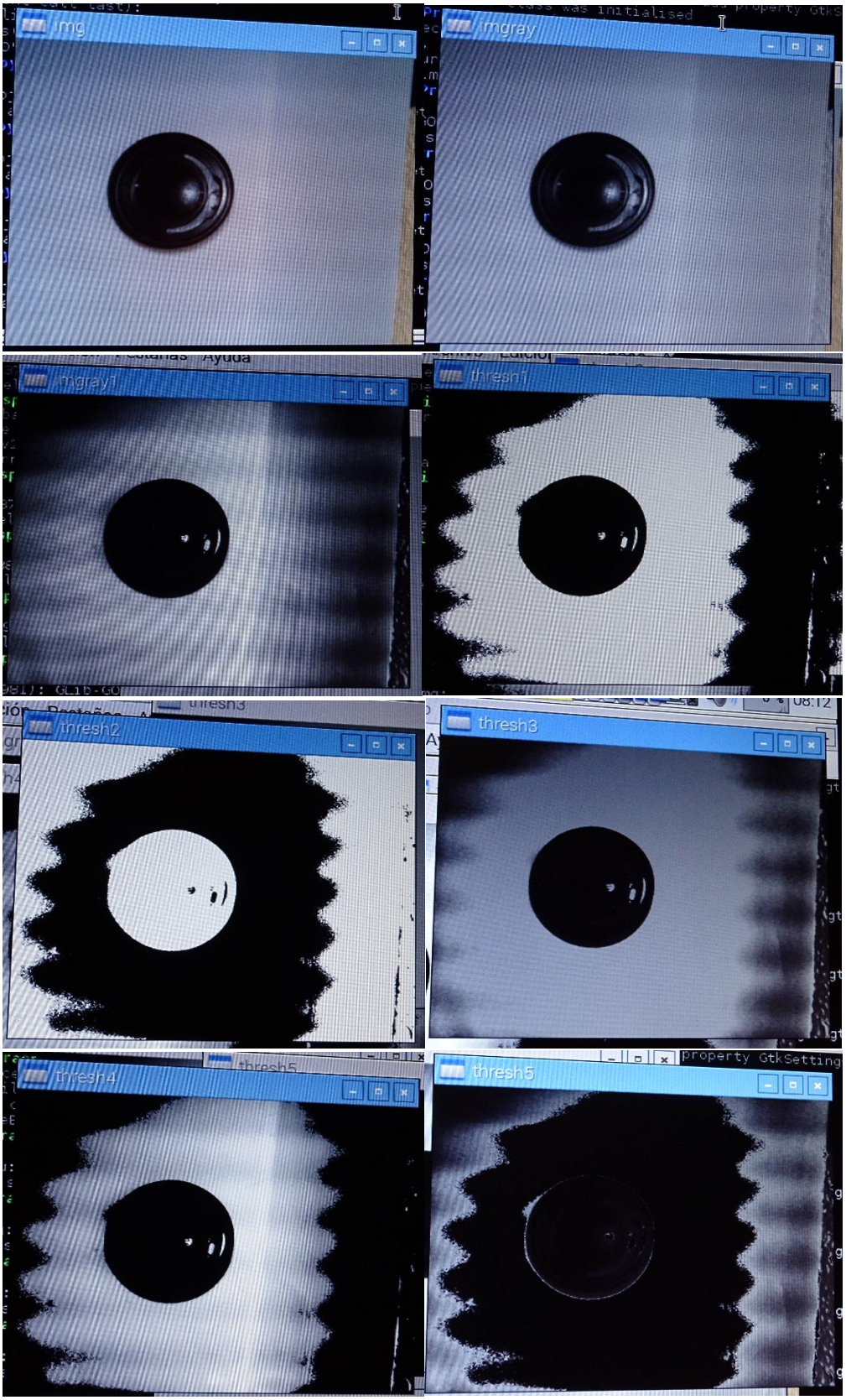


Figura - 57 - Resultados de aplicar las funciones Cv2.threshold().

Aplicamos las funciones cv2.findContours() y la cv2.drawContours() y obtenemos los siguientes resultados.

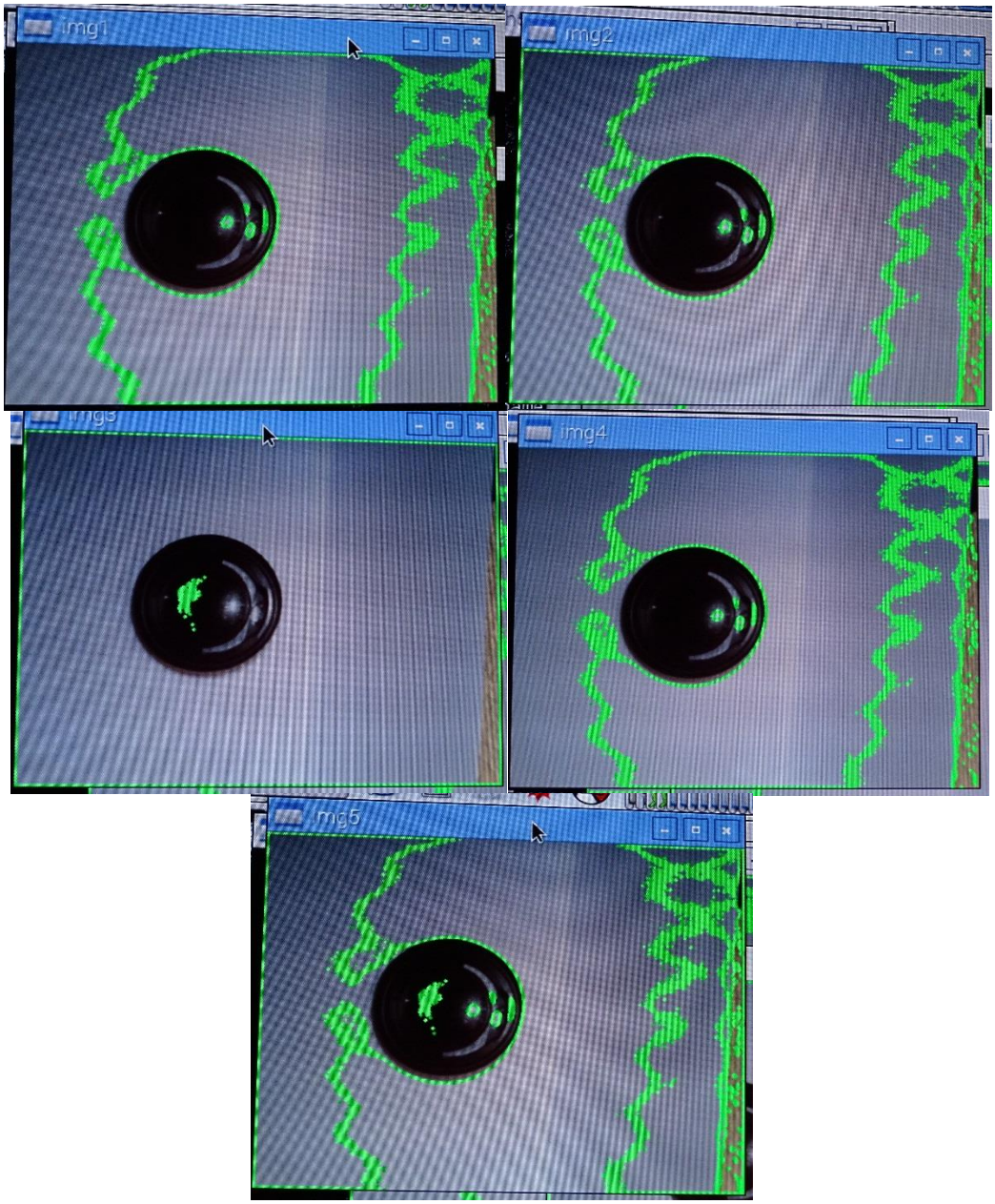


Figura - 58 - Resultados de aplicar las funciones Cv2.findContours() y cv2.drawContours() sobre la imagen tratada con cv2.threshold().

Podríamos seguir mostrando opciones de umbralización utilizando la función cv2.threshold(), pero debido a que los objetos que queremos detectar son redondos, nos interesa ver los resultados que nos aporta la función cv2.HoughCircles().

7.1.2. DETECCIÓN DE CÍRCULOS

- La librería OpenCV nos facilita entre otras la función cv2.HoughCircles(), gracias a la cual podemos detectar círculos, devolviéndonos un vector con las coordenadas del centro del círculo y el radio del mismo.

  Vamos a aplicar esta función en primer lugar a los resultados obtenidos utilizando el filtro proporcionado por la función cv2.threshold() y en segundo lugar por la función cv2.Canny().

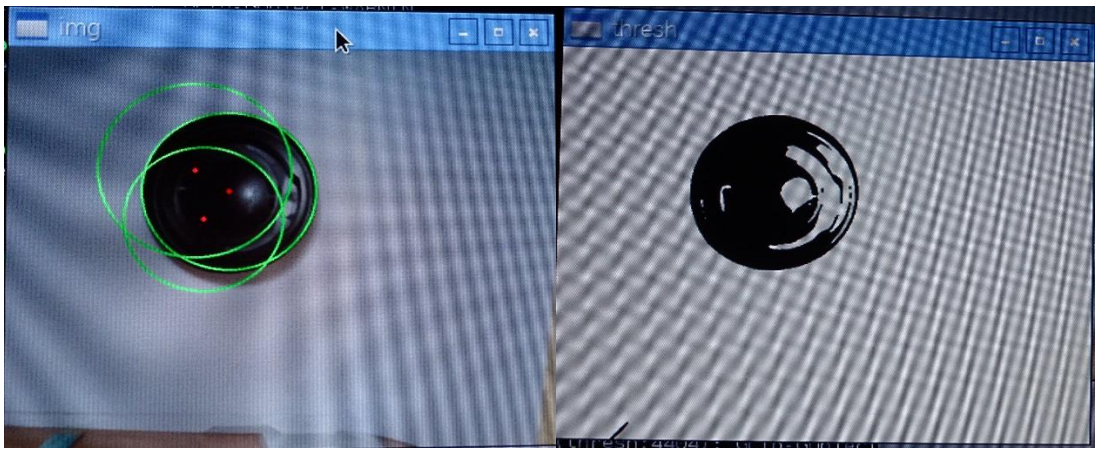


Figura - 59 - Resultado obtenido utilizando cv2.threshold() y cv2.HoughCircles().

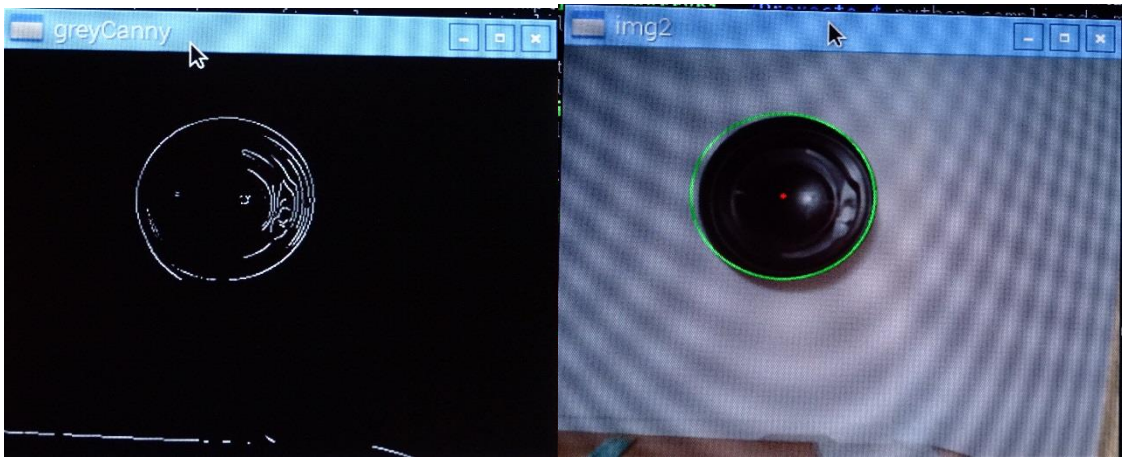


Figura - 60 - Resultado obtenido utilizando cv2.Canny() y cv2.HoughCircles(),

En este ejemplo se ve claramente como es mucho mejor opción optar por la función cv2.Canny() para obtener los contornos de nuestra imagen. Podemos observar como tenemos el centro de nuestro círculo detectado marcado con un punto rojo. A parte de representarlo gráficamente también disponemos de las coordenadas (x,y) para posicionarlo y poder llevar a cabo un correcto movimiento de nuestro vehículo en función del cambio de posición detectado.

- Si ajustamos los valores en la función cv2.threshold() tenemos como mejor la detección de círculos aplicando los filtros adecuados.

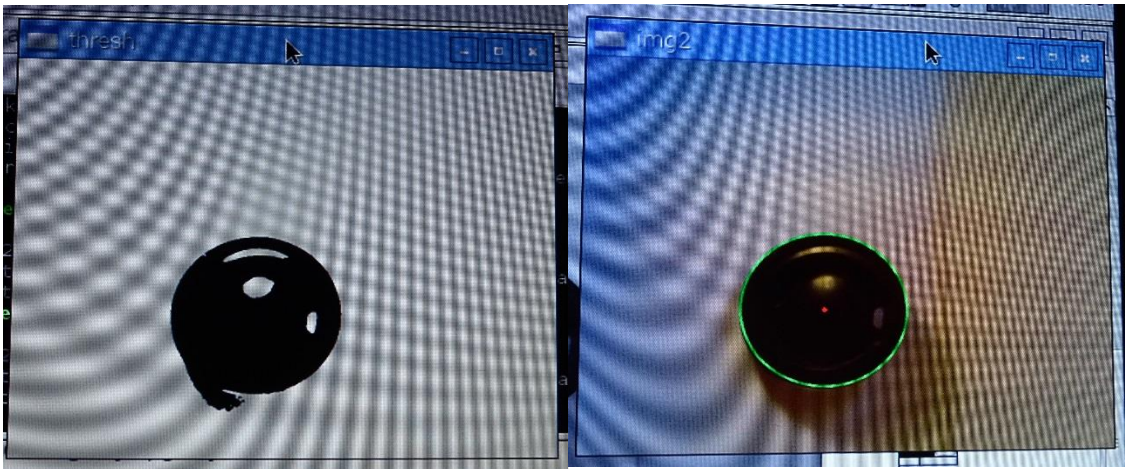


Figura - 61 - Resultado obtenido utilizando cv2.threshold() y cv2.HoughCircles() variando los umbrales en la función threshold.

7.2. MOVIMIENTO DEL VEHÍCULO

En este apartado añadimos las pautas que ha de tener nuestra aplicación para poder llevar a cabo el movimiento de nuestro sistema.

En primer lugar tenemos que ser capaces de que los motores se muevan hacia adelante y hacia atrás; porque nuestro sistema dispone de dos motores, uno para la rueda derecha y otro para la rueda izquierda.

Para ello hemos pensado el siguiente código:

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)

Motor1A = 16
Motor1B = 18
Motor1E = 22

Motor2A = 23
Motor2B = 21
Motor2E = 19

# Establecemos las variables de salida
GPIO.setup(Motor1A,GPIO.OUT)
GPIO.setup(Motor1B,GPIO.OUT)
GPIO.setup(Motor1E,GPIO.OUT)

GPIO.setup(Motor2A,GPIO.OUT)
GPIO.setup(Motor2B,GPIO.OUT)
GPIO.setup(Motor2E,GPIO.OUT)

print "Going forwards"
GPIO.output(Motor1A,GPIO.HIGH)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.HIGH)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Going backwards"
GPIO.output(Motor1A,GPIO.LOW)
GPIO.output(Motor1B,GPIO.HIGH)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.LOW)
GPIO.output(Motor2B,GPIO.HIGH)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor1E,GPIO.LOW)
GPIO.output(Motor2E,GPIO.LOW)

GPIO.cleanup()
```

Con este código básico lo que conseguimos es que el motor vaya hacia adelante 2 segundos y otros dos segundos hacia atrás. Porque en las patillas 19 y 22 estamos aplicando una señal constante de valor HIGH.

En el siguiente ejemplo vamos a ver como usamos la señal PWM.

Para controlar la PWM, tenemos las siguientes funciones:
Para crear la instancia:
p = GPIO.PWM(channel, frequency)
Para arrancar PWM:
p.start(dc)
# Donde dc es el duty cicle o lo que es lo mismo el ciclo de trabajo (0.0 <= dc <= 100.0)
Para cambiar la frecuencia:
p.ChangeFrequency(freq) # where freq is the new frequency in Hz
Para cambiar el ciclo: (ancho de pulso)
p.ChangeDutyCycle(dc) # where 0.0 <= dc <= 100.0
Para parar la PWM:
p.stop()

Añadiendo estas variantes al código anterior obtenemos el siguiente resultado:

```
# Importamos modulos
import time
import RPi.GPIO as GPIO

# Establecemos GPIO y PWM
GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
motor=GPIO.PWM(22,50)

# motor forwards al 10% durante 10 segundos
motor.start(10)
GPIO.output(16,0)
GPIO.output(18,1)
time.sleep(10)
GPIO.output(18,0)
motor.stop()
# Pausa de 1 segundo
time.sleep(1)

# motor forwards al 50% durante 5 segundos
motor.start(50)
GPIO.output(16,0)
GPIO.output(18,1)
time.sleep(5)
GPIO.output(18,0)
motor.stop()
# Pausa de 1 segundo
time.sleep(1)

# motor backwards al 30% durante 10 segundos
motor.start(30)
GPIO.output(18,0)
GPIO.output(16,1)
time.sleep(10)
GPIO.output(16,0)
motor.stop()
# cleanup and close GPIO
GPIO.cleanup()
```

Si lo que queremos ahora es que nuestro sistema se mueva a derecha o a izquierda, o lo que es lo mismo que realice un movimiento de pivote, tendríamos que ejecutar el siguiente código.

Nota: el Motor 1 es el que actúa la rueda izquierda y el Motor 2 el que actúa la rueda derecha.

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)

Motor1A = 16
Motor1B = 18
Motor1E = 22

Motor2A = 23
Motor2B = 21
Motor2E = 19

# Establecemos las variables de salida
GPIO.setup(Motor1A,GPIO.OUT)
GPIO.setup(Motor1B,GPIO.OUT)
GPIO.setup(Motor1E,GPIO.OUT)

GPIO.setup(Motor2A,GPIO.OUT)
GPIO.setup(Motor2B,GPIO.OUT)
GPIO.setup(Motor2E,GPIO.OUT)

print "Going Right"
GPIO.output(Motor1A,GPIO.HIGH)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.LOW)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Going Left"
GPIO.output(Motor1A,GPIO.LOW)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.HIGH)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor1E,GPIO.LOW)
GPIO.output(Motor2E,GPIO.LOW)

GPIO.cleanup()
```

8. PRESUPUESTO

En este capítulo vamos a estimar los datos económicos referentes al coste del desarrollo de este proyecto. En este presupuesto incluimos:

- Costes por uso de equipos y materiales utilizados.
- Coste del personal.
- Costes totales.
- Costes de ejecución por contrata.
- Presupuesto total.

Para el cálculo de costes, se tiene que considerar que la duración del proyecto es de 6 meses.

8.1. Costes por uso de equipos y materiales utilizados.

Concepto	Descripción	Coste	Amortización	Uso	Coste/uso
Ordenador Personal	Sony Vaio W7 Home Premium	895€	120 meses	6 meses	45 €
Equipo Pruebas	Raspberry Pi	40,95€	60 meses	6 meses	5€
Software de ofimática	Microsoft Office 2013	498€	72 meses	6 meses	42€
Librerías de programación	OpenCv, numpy, raspicam, math, RPi.GPIO	0€	-	-	0€
Cámara de grabación	Pi Cam	28,95€	60 meses	6 meses	29€
Monitor	Samsung S22D300HY 22"	109€	60 meses	6 meses	11€
Teclado + Ratón	Gigabyte KM6150 Kit Teclado + Ratón USB	13,95€	120 meses	6 meses	6€
Total					138€

Tabla - 3 - Costes por uso de equipos y materiales.

8.2. Coste del personal

Concepto	Trabajo Realizado	Coste / Mes	Dedicación	Coste
Ingeniero Industrial Jr.	Estudio Investigación Diseño Programación Pruebas Redacción	1.500€	50%	4.500€
Total				4.500€

Tabla - 4 - Coste del personal.

8.3. Costes totales

Concepto	Porcentaje	Coste
Costes por uso de equipos y materiales utilizados	-	138€
Coste del personal	-	4.500€
Costes indirectos	20%	928€
Total		5.566€

Tabla - 5 - Costes totales.

8.4. Costes de ejecución por contrata

Concepto	Porcentaje	Coste
Coste total	-	5.566€
Beneficio	6%	334€
Honorarios de dirección y redacción.	10%	557€
Total		6.457€

Tabla - 6 - Costes de ejecución por contrata.

8.5. Presupuesto total

Concepto	Porcentaje	Coste
Costes de ejecución por contrata	-	6.457€
I.V.A.	21%	1.356€
Total		7.813€

Tabla - 7 - Presupuesto total.

El presupuesto total del proyecto asciende a SIETEMIL OCHOCIENTOS TRECE EUROS.

9. CONCLUSIONES Y FUTURAS APLICACIONES

El objetivo principal de este proyecto fin de carrera era la implementación de una aplicación para el desarrollo de un sistema de seguimiento de personas de bajo coste. En el cual teníamos dos tareas fundamentales a desarrollar.

La primera de ellas era gracias a la visión por computador, localizar el objeto a seguir. Para ello se ha realizado un estudio de las principales metodologías para poder implementarlo. Como hemos visto de forma detallada en la memoria, existen muchas formas de llevar a cabo esta tarea.

La segunda tarea que teníamos que desarrollar era el 'movimiento del vehículo'. En este apartado el objetivo era familiarizarnos con las opciones que nos ofrece GPIO para así ser capaces de lograr el movimiento deseado en nuestro sistema.

Uniendo estas dos tareas hemos implementado una aplicación básica que haría posible que nuestro sistema de bajo coste se dirigiese hacia la posición del objeto detectado previamente de forma satisfactoria. Esta aplicación no se ha podido probar, ya que no hemos implementado el sistema completo, pero se ha estudiado y diseñado de forma que garantizamos, al menos de forma teórica, que alcanzamos el objetivo.

Por lo que a partir de los resultados obtenidos en las pruebas y de las conclusiones halladas de forma "teórica", podemos decir que hemos logrado los objetivos que nos planteamos al inicio del proyecto.

Las líneas de trabajo futuro pueden ir orientadas a la implementación del vehículo de bajo coste, construyéndolo según las características físicas y teóricas tenidas en cuenta en este proyecto. Para poder hacer un seguimiento exacto deberíamos incluir a nuestro sistema o bien un sensor de posición, otra cámara para así poder calcular la posición exacta del objeto detectado, o un puntero laser para la calibración del sistema y de igual modo poder proceder al cálculo de la distancia exacta.

10. BIBLIOGRAFÍA

1. Fundamentals of Digital Image Processing, Anil K. Jain, Ed. Prentice Hall, 1989.
2. A. de la Escalera, Visión por computador: Fundamentos y métodos, Pearson-Prentice Hall, 2001.
3. Learning OpenCV: Computer Vision with OpenCV, Gary Bradsky & Adrian Kaehler, O'Reilly, 2008
4. A. Barrientos, L. F. Peñín, C. Balaguer, and R. Aracil, Fundamentos de robótica. Madrid: McGRAW-HILL, 1997.
5. Learn Raspberry Pi Programing with Python, Wolfram Donat. Ed. Tecnology in Action.
6. http://www.sdcard.org/
7. https://opencv-python-tutroals.readthedocs.org/en/latest/
8. https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_tutorials.html
9. http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=houghcircles#houghcircles
10. http://raspi.tv/
11. https://www.raspberrypi.org/
12. http://www.ti.com/lit/ds/symlink/l293.pdf
13. http://www.raspberrypi.org/faqs.
14. http://www.wikipedia.org/

11. ANEXOS

11.1. INTALACIÓN DE SOFTWARE

El primer paso que tenemos que completar es la instalación del sistema operativo seleccionado, en nuestro caso Raspbian. El siguiente paso sería la instalación y actualización de todas las librerías que hemos mencionado en el apartado 2.3 para poder ponernos manos a la obra con la programación de nuestro dispositivo.

11.1.1. INSTALACIÓN DEL SISTEMA OPERATIVO (RASPBIAN)

¿Qué necesitamos para empezar con la instalación?

- Evidentemente nuestra Raspberry Pi (en nuestro caso como ya hemos comentado disponemos del modelo B).
- Adaptador de red a micro USB. (Cualquier cargador de los actuales móviles nos debería valer).
- Una tarjeta SD. (La capacidad mínima es de 4GB, pero se recomienda utilizar una de al menos 8 GB. En nuestro caso la que vamos a utilizar es una de 16GB; ya que la disponíamos inicialmente de 4GB se nos corrompió y dejó de funcionar).
- Un cable HDMI. (Para la instalación del sistema operativo es necesario conectar la Raspberry Pi a un monitor o un televisor; una vez instalado ya no será necesario, ya que podremos acceder a ella por SSH, por lo que podremos desconectarlo y acceder remotamente).
- Teclado y ratón USB. (Su utilidad es necesaria al menos para la instalación del sistema operativo, al igual que el cable HDMI).
- Cable de red. (Es necesario si queremos acceder a internet desde la Raspberry, o si queremos acceder a la Raspberry remotamente).

Tenemos dos maneras de instalar Raspbian, una es usando directamente la imagen de Raspbian y otra es usando NOOBS. Esta segunda opción es la más sencilla y nos permite instalar a parte de Raspbian otras distribuciones de Linux para Raspberry Pi; como contrapunto nos ocupará espacio extra en la tarjeta SD (en nuestro caso esto no es un problema ya que disponemos de una SD de 16GB).

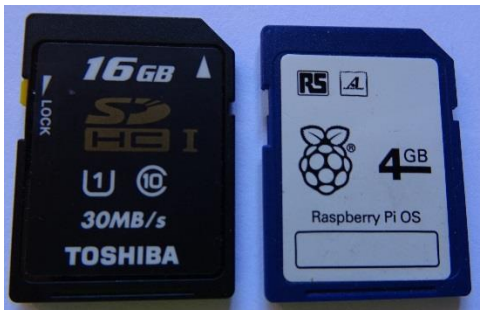


Figura - 62 -Tarjetas SD

Lo primero que tenemos que hacer es adquirir el software, para ello simplemente tenemos que acceder a la web oficial de la fundación Raspberry. En el siguiente enlace obtenemos el paquete NOOBS (V1.4.1) el cual nos ofrece un instalador sencillo de utilizar para la instalación de Raspbian.

https://www.raspberrypi.org/downloads/noobs/

Una vez que tenemos descargada la imagen tenemos que descomprimirla para obtener el fichero .img. El siguiente paso es seguir las instrucciones de instalación de Noobs que nos vienen indicadas en el fichero INSTRUCTIONS-README.

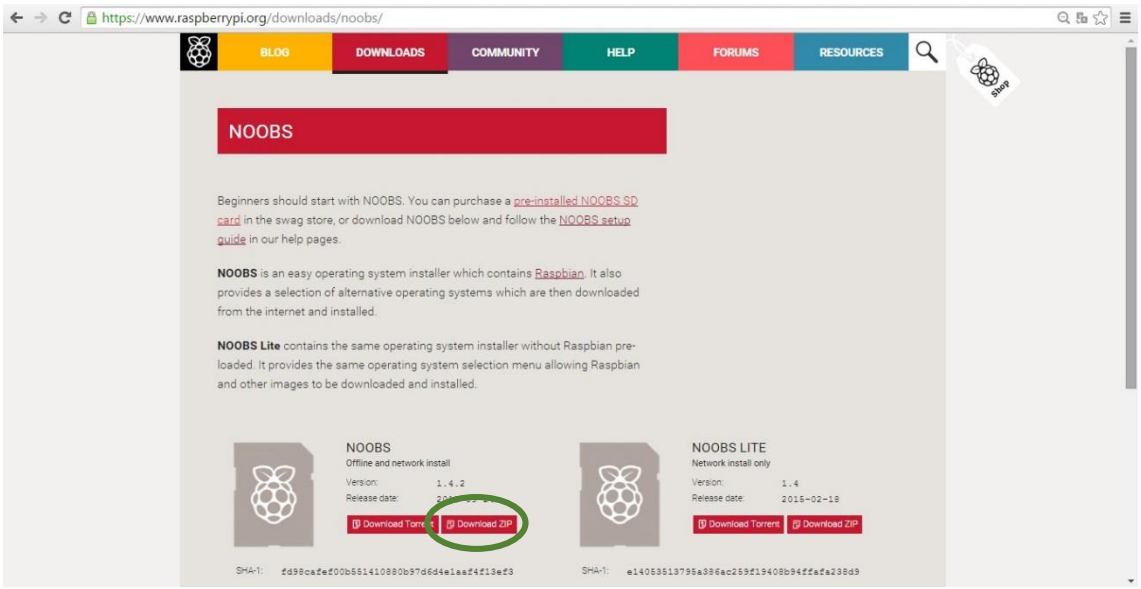


Figura - 63 - Web oficial de la fundación Raspberry Pi para la descarga del S.O.

Antes de copiar los ficheros de NOOBS en la tarjeta SD, tenemos que formatearla.

La web oficial de Raspberry Pi nos aconseja descargar el software SD Formatter 4.0 de la web de la asociación SD (http://www.sdcard.org/); más concretamente del siguiente enlace (https://www.sdcard.org/downloads/formatter_4/)

Descargamos e instalamos el software, y una vez instalado insertamos la tarjeta SD, en el ordenador personal. Una vez que la tarjeta SD ha sido formateada en formato FAT, copiamos en ella todos los archivos de la carpeta NOOBS que previamente hemos descomprimido en el ordenador personal.

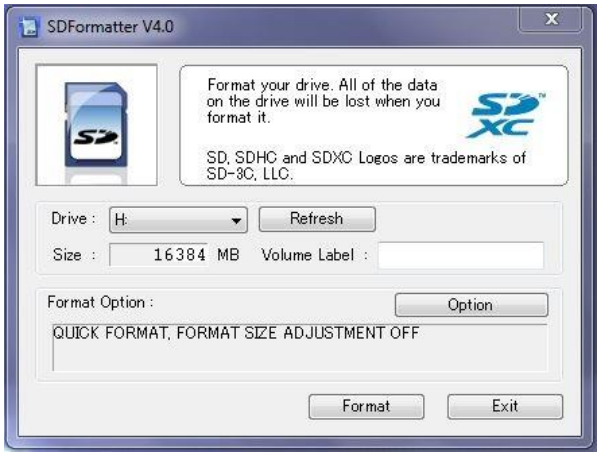


Figura - 64 - SDFormatter V4.0

Una vez que todos los archivos se han copiado de forma correcta, extraemos la tarjeta SD de forma segura, y la introducimos en la Raspberry Pi.

Primer inicio del sistema:

- Conectamos el monitor (cable HDMI), el ratón y el teclado.
- A continuación conectamos el cable de alimentación USB de nuestra Raspberry Pi.
- Al conectar la alimentación la Raspberry se iniciará y aparecerá una ventana con los diferentes sistemas operativos que podemos instalar. Como ya hemos visto seleccionamos para su instalación Raspbian, y pulsamos en instalar.

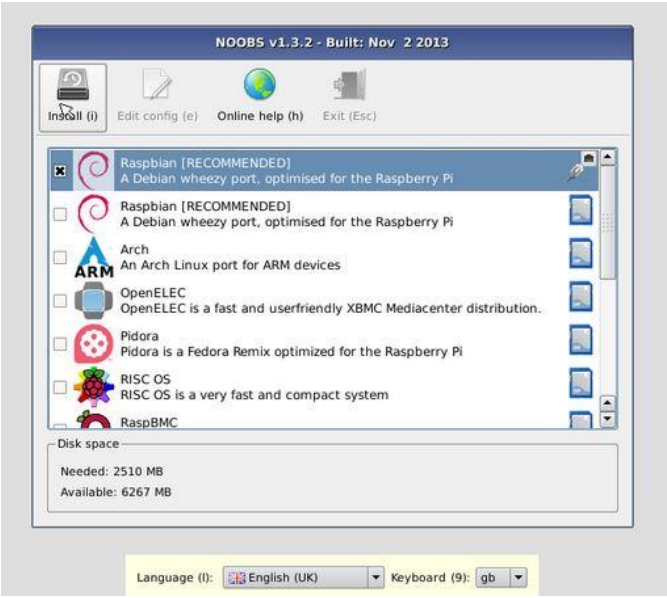


Figura - 65 - Ejemplo de menú de selección NOOBS

- Raspbian comenzara con el proceso de instalación, lo cual llevará un tiempo.
- Cuando el proceso de instalación termine se carga el menú de configuración de Raspberry Pi. Desde este menú podemos seleccionar la fecha y la hora, habilitar o deshabilitar la cámara, crear usuarios, etc. Este menú se puede lanzar en cualquier momento que deseamos ejecutando el siguiente comando:

```
pi@raspberrypi ~ $ sudo raspi-config
```

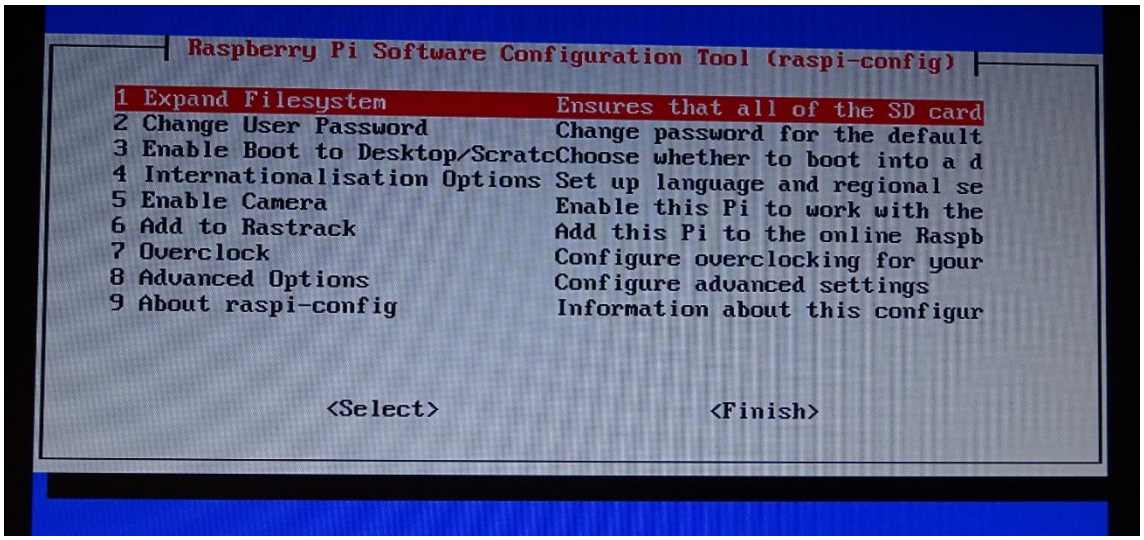


Figura - 66 - Menú de configuración de Raspberry Pi

Una vez que ya hemos instalado el sistema operativo y configurado el lenguaje y las configuraciones básicas del sistema operativo, tenemos que instalar las librerías de visión por computador de las que hemos hablado en el capítulo 2.

11.1.2. INSTALACIÓN DE OPENCV Y PYTHON

En este apartado vamos a explicar paso por paso como realizar una correcta instalación del software OPENCV y PYTHON para nuestra Raspberry Pi.

- En primer lugar tenemos que actualizar el firmaware de nuestra Raspberry Pi.

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get upgrade
pi@raspberrypi ~ $ rpi-update
```

- El siguiente paso que debemos realizar es instalar las herramientas y paquetes de desarrollo requeridos.

```
pi@raspberrypi ~ $ sudo apt-get install build-essential cmake pkg-config
```

- A continuación instalamos los paquetes de E/S (entrada salida), estos paquetes nos permitirán cargar varios formatos de archivo de imagen como por ejemplo JPEG, PNG, TIFF, etc.

```
pi@raspberrypi ~ $ sudo apt-get install libjpeg8-dev libtiff4-dev
                   libjasper-dev libpng12-dev
```

- Continuamos instalando la biblioteca de desarrollo GTK. Esta biblioteca se utiliza para construir interfaces gráficas de usuario (GUI) y es necesaria para la biblioteca highgui de OpenCV, que le permite ver las imágenes en su pantalla.

```
pi@raspberrypi ~ $ sudo apt-get install libgtk2.0-dev
```

- El siguiente paso es instalar los paquetes de E/S de vídeo necesarios. Estos paquetes se utilizan para cargar archivos de vídeo usando OpenCV.

```
pi@raspberrypi ~ $ sudo apt-get install libavcodec-dev libavformat-dev
                   libswscale-dev libv4l-dev
```

- Instalamos las bibliotecas que se utilizan para optimizar diversas operaciones dentro de OpenCV.

```
pi@raspberrypi ~ $ sudo apt-get install libatlas-base-dev gfortran
```

- El siguiente paso es instalar pip.

```
pi@raspberrypi ~ $ wget https://bootstrap.pypa.io/get-pip.py
pi@raspberrypi ~ $ sudo python get-pip.py
```

- Instalamos virtualenv y virtualenvwrapper

```
pi@raspberrypi ~ $ sudo pip install virtualenv virtualenvwrapper
pi@raspberrypi ~ $ sudo rm -rf ~/.cache/pip
```

Una vez instalado tenemos que incluirlo al ~.profile:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
source /usr/local/bin/virtualenvwrapper.sh
```

Creamos un entorno virtual de visión por ordenador:

```
pi@raspberrypi ~ mkvirtualenv cv
```

- Ahora ya podemos proceder a instalar la herramienta de desarrollo Python 2.7.

```
pi@raspberrypi ~ sudo apt-get install python2.7-dev
```

También tenemos que instalar NumPy desde los enlaces de OpenCV Python representan imágenes como matrices NumPy multidimensionales:

```
pi@raspberrypi ~ pip install numpy
```

- El siguiente paso es descargar OpenCV y descomprimirlo.

```
pi@raspberrypi ~ wget -O opencv-2.4.10.zip
http://sourceforge.net/projects/opencvlibrary/files/opencv-
unix/2.4.10/opencv-2.4.10.zip/download
pi@raspberrypi ~ unzip opencv-2.4.10.zip
pi@raspberrypi ~ cd opencv-2.4.10
```

Configuramos:

```
pi@raspberrypi ~ mkdir
pi@raspberrypi ~ cd build
pi@raspberrypi ~ cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_NEW_PYTHON_SUPPORT=ON -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D
BUILD EXAMPLES=ON ..
```

Compilamos OpenCV:

```
pi@raspberrypi ~ make
```

Nota: es muy importante asegurarse de que estamos en el entorno virtual cv ya que OpenCV se compila contra el entorno virtual de Python y NumPy.

El último paso de este apartado sería la instalación de OpenCV:

```
pi@raspberrypi ~ sudo make install
pi@raspberrypi ~ sudo ldconfig
```

- Una vez llegados a este punto podemos comprobar como tenemos instalado OpenCV en el siguiente directorio: `/usr/local/lib/python2.7/site-packages.`

A continuación creamos los enlaces simbólicos:

```
pi@raspberrypi ~ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/
pi@raspberrypi ~ -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
pi@raspberrypi ~ ln -s /usr/local/lib/python2.7/site-packages/cv.py cv.py
```

- Por último comprobamos que todo funciona correctamente.

```
pi@raspberrypi ~ workon cv
pi@raspberrypi ~ python
>>> import cv2
>>> cv2.__version__
'2.4.10'
```

11.2. CODIGO UTILIZADO

A continuación vamos a desgranar el código utilizado para cada parte del proyecto.

11.2.1. BUSQUEDA DE CONTORNOS

Filtramos y ecualizamos las imágenes, gracias a las funciones cv2.equalizeHist() y cv2.medianBlur().

```
import cv2
import picamera
import math
import numpy as np

## Capturamos la imagen
camera = picamera.Picamera()
camera.capture('imagen.jpg')
camera.close()

## Cargamos la imagen
img0 = cv2.imread("imagen.jpg")
img = cv2.imread("imagen.jpg") # 0 para escala de grises

## Aplicamos Filtros
img2 = cv2.medianBlur(img,5)

## Ecualizamos el histograma
equ  = cv2.equalizeHist(img)
equ2 = cv2.equalizeHist(img2)  # Esta imagen va filtrada previamente

equBlue = cv2.medianBlur(equ,5)

## Redimensionamos las imagens a mostrar
equ = cv2.resize(equ,(0,0), fx=0.5,fy=0.5)
equ2 = cv2.resize(equ2,(0,0), fx=0.5,fy=0.5)
img = cv2.resize(img,(0,0), fx=0.5,fy=0.5)
equBlue = cv2.resize(equBlue,(0,0), fx=0.5,fy=0.5)
img0 = cv2.resize(img0,(0,0), fx=0.5,fy=0.5)

## Mostramos las imagenes
cv2.imshow("equ",equ)
cv2.imshow("equ2",equ2)
cv2.imshow("img", img)
cv2.imshow("equBlue", equBlue)
cv2.imshow("Imagen_original", img0)
```

Búsqueda de contornos utilizando las función cv2.Canny() y cv2.findContours() como base.

```
import cv2
import picamera
import math
import numpy as np

## Capturamos la imagen
camera = picamera.Picamera()
camera.capture('imagen.jpg')
camera.close()

## Cargamos la imagen
img0 = cv2.imread("imagen.jpg")
img1 = cv2.imread("imagen.jpg")
img2 = cv2.imread("imagen.jpg")
img3 = cv2.imread("imagen.jpg")
img = cv2.imread("imagen.jpg") # 0 para escala de grises

## Ecualizamos el histograma
equ  = cv2.equalizeHist(img)

## Aplicamos la función cv2.Canny()
edges = cv2.Canny(equ,80,120)
edges1 = cv2.Canny(equ,0,120)
edges2 = cv2.Canny(equ,0,0)
edges3 = cv2.Canny(equ,120,120)

## Detectamos y guardamos los contornos.
contours, hierarchi = cv2.findContours(edges,cv2.RETR_TREE,cv2.CHAIN_APROX_SIMPLE)
contours1, hierarchi = cv2.findContours(edges1,cv2.RETR_TREE,cv2.CHAIN_APROX_SIMPLE)
contours2, hierarchi = cv2.findContours(edges2,cv2.RETR_TREE,cv2.CHAIN_APROX_SIMPLE)
contours3, hierarchi = cv2.findContours(edges3,cv2.RETR_TREE,cv2.CHAIN_APROX_SIMPLE)

## Dibujamos los contornos.
cv2.drawContours(img0,contours,-1,(0,255,0),3)
cv2.drawContours(img1,contours1,-1,(0,255,0),3)
cv2.drawContours(img2,contours2,-1,(0,255,0),3)
cv2.drawContours(img3,contours3,-1,(0,255,0),3)

## Redimensionamos las imágenes a mostrar
equ = cv2.resize(equ,(0,0), fx=0.5,fy=0.5)
img = cv2.resize(img,(0,0), fx=0.5,fy=0.5)
img0 = cv2.resize(img0,(0,0), fx=0.5,fy=0.5)
img1 = cv2.resize(img1,(0,0), fx=0.5,fy=0.5)
img2 = cv2.resize(img2,(0,0), fx=0.5,fy=0.5)
img3 = cv2.resize(img3,(0,0), fx=0.5,fy=0.5)
edges = cv2.resize(edges,(0,0), fx=0.5,fy=0.5)
edges1 = cv2.resize(edges1,(0,0), fx=0.5,fy=0.5)
edges2 = cv2.resize(edges2,(0,0), fx=0.5,fy=0.5)
edges3 = cv2.resize(edges3,(0,0), fx=0.5,fy=0.5)
```

```
## Mostramos las imagenes
cv2.imshow("equ",equ)
cv2.imshow("img", img)
cv2.imshow("img1", img1)
cv2.imshow("img2", img2)
cv2.imshow("img3", img3)
cv2.imshow("imgagen_Original", img0)
cv2.imshow("edges", edges)
cv2.imshow("edges1", edges1)
cv2.imshow("edges2", edges2)
cv2.imshow("edges3", edges3)
```

Búsqueda de contornos utilizando las funciones cv2.threshold() y cv2.findContours() como base.

```
import cv2
import cv
import picamera

## Capturamos la imagen
camera = picamera.Picamera()
camera.capture('imagen.jpg')
camera.close()

## Cargamos la imagen
imoriginal = cv2.imread("imagen.jpg")
imoriginal2 = cv2.imread("imagen.jpg")
imoriginal3 = cv2.imread("imagen.jpg")
imgris = cv2.imread("imagen.jpg",0) # 0 para escala de grises

## Aplicamos Filtros
imgris2 = cv2.GaussianBlur(imgris, (5,5),0)
imgris3 = cv2.medianBlur(imgris, 5)

## Ejecutamos la función Threshold
ret, thresh2 = cv2.threshold (imgris2, 150, 255, cv2THRESH_BINARY)
ret, thresh3 = cv2.threshold (imgris3, 0  , 255, cv2THRESH_BINARY)
#ret, thresh4 = cv2.threshold (imgris , 50 , 255, cv2THRESH_BINARY_INV)
#ret, thresh5 = cv2.threshold (imgris , 0  , 255, cv2THRESH_TRUNC)
#ret, thresh6 = cv2.threshold (imgris , 0  , 255, cv2THRESH_TOZERO)
#ret, thresh7 = cv2.threshold (imgris , 0  , 255, cv2THRESH_TOZERO_INV)
#ret, thresh8 = cv2.threshold (imgris , 127, 255, cv2THRESH_BINARY)

## Buscamos contornos
Contours2, hierarchi = cv2.findContours(thresh2,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
Contours3, hierarchi = cv2.findContours(thresh3,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
#Contours4, hierarchi = cv2.findContours(thresh4,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
#Contours5, hierarchi = cv2.findContours(thresh5,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
#Contours6, hierarchi = cv2.findContours(thresh6,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
#Contours7, hierarchi = cv2.findContours(thresh7,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)
#Contours8, hierarchi = cv2.findContours(thresh8,cv2.RETR_TREE,cv2.CHAIN:APPROX_SIMPLE)

## Dibujamos contornos
cv2.drawContours(imoriginal2, Cotours2, -1, (0,255,0), 3)
cv2.drawContours(imoriginal3, Cotours3, -1, (0,255,0), 3)

## Mostramos las imágenes
cv2.imshow("imagen original", imoriginal)
cv2.imshow("imagen 2", imoriginal2)
cv2.imshow("imagen 3", imoriginal3)
cv2.imshow("threshold 2", thresh2)
cv2.imshow("threshold 3", thresh3)
```

11.2.2. DETECCIÓN DE CÍRCULOS

Utilizamos las funciones cv2.threshold() y cv2.HoughCircles() como base.

```
import cv2
import cv
import picamera

## Capturamos la imagen
camera = picamera.Picamera()
camera.capture('imagen.jpg')
camera.close()

## Cargamos la imagen
imagen = cv2.imread("imagen.jpg")
imgris = cv2.imread("imagen.jpg",0) # 0 para escala de grises

## Aplicamos Filtros
imagen = cv2.medianBlur(imagen, 5)
imgris  = cv2.medianBlur(imgris, 5)

## Ejecutamos la función Threshold
ret, thresh = cv2.threshold (imgris, 50, 255, cv2THRESH_BINARY)

## Usamos la función Houghcircles para determinar el centro del círculo
Circles = cv2.HoughCircles(imgris, cv2.cv.CV_HOUGH_GRADIENT,1,50,param1=50,param2=30,
          minRadius=130,maxRadius=175)
for I in circles[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(imgris,(i[0],i[1]),i[2],(0,255,0),2)
      cv2.circle(imagen,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(imgris,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(imagen,(i[0],i[1]),2,(0,0,255)3)

## Mostramos las imágenes
cv2.imshow("imagen original", imagen)
cv2.imshow("imagen gris", imgris)
cv2.imshow("imagen threshold", thresh)
```

Utilizamos las funciones cv2.Canny (),cv2.threshold() y cv2.HoughCircles() como base.

```
import cv2
import picamera
import math
import numpy as np

## Capturamos la imagen
camera = picamera.Picamera()
camera.capture('imagen.jpg')
camera.close()

## Cargamos la imagen
img = cv2.imread("imagen.jpg")
img = cv2.medianBlur(img,5)

img2 = cv2.imread("imagen.jpg")
img2 = cv2.medianBlur(img2,5)

grey = cv2.imread("imagen.jpg",0) # 0 para escala de grises
grey = cv2.medianBlur(grey,5)
grey2 = cv2.equalizeHist(grey)

## Aplicamos Canny
greyCanny = cv2.Canny(grey, 120,120)

## Aplicamos Threshold
ret, thresh = cv2.threshold(grey,50,255,cv2.THRESH_BINARY)

## Usamos la función Houghcircles para determinar el centro del círculo
circles = cv2.HoughCircles(grey, cv2.cv.CV_HOUGH_GRADIENT,1,50,param1=50,param2=30,
          minRadius=130,maxRadius=175)
for i in circles[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(grey,(i[0],i[1]),i[2],(0,255,0),2)
      cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(grey,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(img,(i[0],i[1]),2,(0,0,255)3)
circulos = np.round(circles[0,:].astype("int")
for (x,y,r) in circulos:
      radio = r
      coordenadax = x
      coordenaday = y

## Usamos la función Houghcircles para determinar el centro del círculo
Circles2 = cv2.HoughCircles(greyCanny,
cv2.cv.CV_HOUGH_GRADIENT,1,50,param1=50,param2=30,
          minRadius=130,maxRadius=175)
for i in circles[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(grey2,(i[0],i[1]),i[2],(0,255,0),2)
      cv2.circle(img2,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(grey2,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(img2,(i[0],i[1]),2,(0,0,255)3)
```

```
circulos2 = np.round(circles2[0,:].astype("int")
for (x,y,r) in circulos:
      radio2 = r
      coordenadax2 = x
      coordenaday2 = y

## Redimensionamos las imagenes
grey = cv2.resize(grey,(0,0),fx=0.5,fy=0.5)
grey2 = cv2.resize(grey2,(0,0),fx=0.5,fy=0.5)
thresh = cv2.resize(thresh,(0,0),fx=0.5,fy=0.5)
greyCanny = cv2.resize(greyCanny,(0,0),fx=0.5,fy=0.5)
img = cv2.resize(img,(0,0),fx=0.5,fy=0.5)
img2 = cv2.resize(img2,(0,0),fx=0.5,fy=0.5)

## Mostramos las imagenes
cv2.imshow("thresh",thresh)
cv2.imshow("img",img)
cv2.imshow("img2", img2)
cv2.imshow("grey",grey)
cv2.imshow("grey2", grey2)
cv2.imshow("greyCanny", greyCanny)

## Imprimimos por pantalla el tamaño del radio del círculo detectado
print "El radio del circulo es %d" %radio
print "La coordenada x es %d" %coordenadax
print "La coordenada y es %d" %coordenaday
print "El radio del ciruclo2 es %d" %radio2
print "La coordenada x2 es %d" %coordenadax2
print "La coordenada y2 es %d" %coordenaday2
```

11.2.3. DETECCIÓN DE MOVIMIENTO

En este apartado hemos desarrollado un código que lo que hace es detectar diferencia entre dos frames consecutivos, con un lapso de tiempo entre ellas, y como consecuencia determinar si ha habido movimiento en nuestra imagen, y como consecuencia a esto realizar un movimiento de nuestro sistema.

```
import cv2
import picamera
import math
import numpy as np
from time import sleep

## Capturamos la imagen 1
camera = picamera.Picamera()
camera.capture('imagen1.jpg')
camera.close()

sleep(2)  # espero 2 segundos entre frame y frame

## Capturamos la imagen 2
camera = picamera.Picamera()
camera.capture('imagen2.jpg')
camera.close()

## Cargamos las imagens
Img1 = cv2.imread("imagen1.jpg")
Img1 = cv2.medianBlur(img1,5)
Img2 = cv2.imread("imagen2.jpg")
Img2 = cv2.medianBlur(img2,5)

Grey1 = cv2.imread("imagen1.jpg",0) # 0 para escala de grises
Grey1 = cv2.medianBlur(grey1,5)
Grey2 = cv2.imread("imagen2.jpg",0) # 0 para escala de grises
Grey2 = cv2.medianBlur(grey2,5)

## Aplicamos Canny
greyCanny1 = cv2.Canny(grey1, 120,120)
greyCanny2 = cv2.Canny(grey2, 120,120)

## Usamos la función Houghcircles
circles = cv2.HoughCircles(greyCanny, cv2.cv.CV_HOUGH_GRADIENT,1,50,
              param1=50,param2=30,minRadius=130,maxRadius=175)
for i in circles[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(grey,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(img,(i[0],i[1]),2,(0,0,255)3)
circulos1 = np.round(circles[0,:].astype("int")
for (x,y,r) in circulos1:
      radio1 = r
      coordenadax1 = x
      coordenaday1 = y
```

```
## Usamos la función Houghcircles para determinar el centro del círculo
circles2 = cv2.HoughCircles(greyCanny2, cv2.cv.CV_HOUGH_GRADIENT,1,50,
              param1=50,param2=30,minRadius=130,maxRadius=175)
for i in circles2[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(img2,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(img2,(i[0],i[1]),2,(0,0,255)3)
circulos2 = np.round(circles2[0,:].astype("int")
for (x,y,r) in circulos2:
      radio2 = r
      coordenadax2 = x
      coordenaday2 = y

if coordenadax1 < coordenadax2  # el objeto se ha movido a la derecha
      Print "Tenemos que mover a la izquierda"
else
if coordenadax1 > coordenadax2  # el objeto se ha movido a la izquierda
      print "Tenemos que mover a la derecha"
else
if ((coordenaday1 > coordenaday2) or (radio1 < radio2)) # se ha movido adelante
      print "Tenemos que mover hacia atrás"
else
if ((coordenaday1 < coordenaday2) or (radio1 > radio2)) # se ha movido hacia atrás
      print "Tenemos que movernos hacia Adelante"
```

11.2.4. APLICACIÓN COMPLETA

Siguiendo el diagrama de flujo diseñado en el apartado 6.3. obtenemos el siguiente programa:

```
import cv2
import picamera
import math
import numpy as np
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)

## Declaramos datos referencia
Radio0 = 100
Coordenadax0 = 350
Coordenaday0 = 300

Motor1A = 16
Motor1B = 18
Motor1E = 22

Motor2A = 23
Motor2B = 21
Motor2E = 19

# Establecemos las variables de salida
GPIO.setup(Motor1A,GPIO.OUT)
GPIO.setup(Motor1B,GPIO.OUT)
GPIO.setup(Motor1E,GPIO.OUT)

GPIO.setup(Motor2A,GPIO.OUT)
GPIO.setup(Motor2B,GPIO.OUT)
GPIO.setup(Motor2E,GPIO.OUT)

Bucle = 1

While (bucle = 1)

       ## Capturamos la imagen
       camera = picamera.Picamera()
       camera.capture('imagen.jpg')
       camera.close()

       ## Cargamos la imagen
       img = cv2.imread("imagen.jpg")
       img = cv2.medianBlur(img,5)

       grey = cv2.imread("imagen.jpg",0) # 0 para escala de grises
       grey = cv2.medianBlur(grey,5)

       ## Aplicamos Canny
       greyCanny = cv2.Canny(grey, 120,120)
```

```
## Usamos la función Houghcircles para determinar el centro del círculo
circles = cv2.HoughCircles(greyCanny, cv2.cv.CV_HOUGH_GRADIENT,1,50,
              param1=50,param2=30,minRadius=130,maxRadius=175)
for i in circles[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle(img,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(grey,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(img,(i[0],i[1]),2,(0,0,255)3)
circulos = np.round(circles[0,:].astype("int")
for (x,y,r) in circulos:
      radio = r
      coordenadax = x
      coordenaday = y

if coordenadax < coordenadax0  # muevo a la derecha 2 segundos y paro.
print "Going Right"
GPIO.output(Motor1A,GPIO.HIGH)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.LOW)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor1E,GPIO.LOW)

else
if coordenadax > coordenadax0  # muevo a la izquierda 2 segundos y paro.
print "Going Left"
GPIO.output(Motor1A,GPIO.LOW)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.HIGH)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor2E,GPIO.LOW)

else
if ((coordenaday > coordenaday0) or (radio < radio0)) # muevo adelante
```

```
print "Going forwards"
GPIO.output(Motor1A,GPIO.HIGH)
GPIO.output(Motor1B,GPIO.LOW)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.HIGH)
GPIO.output(Motor2B,GPIO.LOW)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor1E,GPIO.LOW)
GPIO.output(Motor2E,GPIO.LOW)

else
if ((coordenaday < coordenaday0) or (radio > radio0)) # muevo atrás
print "Going backwards"
GPIO.output(Motor1A,GPIO.LOW)
GPIO.output(Motor1B,GPIO.HIGH)
GPIO.output(Motor1E,GPIO.HIGH)

GPIO.output(Motor2A,GPIO.LOW)
GPIO.output(Motor2B,GPIO.HIGH)
GPIO.output(Motor2E,GPIO.HIGH)

sleep(2)  # espero 2 segundos

print "Now stop"
GPIO.output(Motor1E,GPIO.LOW)
GPIO.output(Motor2E,GPIO.LOW)

## Capturamos la imagen para comprobar
camera = picamera.Picamera()
camera.capture('imagen2.jpg')
camera.close()

## Cargamos la imagen
img2 = cv2.imread("imagen2.jpg")
img2 = cv2.medianBlur(img2,5)

grey2 = cv2.imread("imagen2.jpg",0) # 0 para escala de grises
grey2 = cv2.medianBlur(grey2,5)

## Aplicamos Canny
greyCanny2 = cv2.Canny(grey2, 120,120)
```

```
## Usamos la función Houghcircles para determinar el centro del círculo
circles2 = cv2.HoughCircles(greyCanny2, cv2.cv.CV_HOUGH_GRADIENT,1,50,
              param1=50,param2=30,minRadius=130,maxRadius=175)
for i in circles2[0,:]:
      # Dibujamos el exterior del círculo
      cv2.circle2(img2,(i[0],i[1]),i[2],(0,255,0),2)
      # Dibujamos el centro del círculo
      cv2.circle(grey2,(i[0],i[1]),2,(0,0,255)3)
      cv2.circle(img2,(i[0],i[1]),2,(0,0,255)3)
circulos = np.round(circles2[0,:].astype("int")
for (x,y,r) in circulos:
      radio2 = r
      coordenadax2 = x
      coordenaday2 = y

if (coordenadax2 = coordenadax0 and coordenaday2 = coordenaday0 and
    radio2 = radio0)
      bucle = 2          # salimos del bucle
      GPIO.cleanup()
      print "Objetivo localizado"
else
      print "Continuamos recalculando"
```