PROYECTO FINAL DE CARRERA

ROBÓTICA PARA SEGUIMIENTO DE LÍNEAS

(ROBOTICS FOR LINE TRACKING)

Estudis: Enginyeria Electrónica

Autor: David Ortiz Martínez

Director/a: Sergi Bermejo

Año: 2015-2016

Índice general

Índice general 1
Colaboraciones 3
Agradecimientos 4
Resumen del Proyecto 5
Abstract 6
1. Introducción 7
1.1 Contexto del proyecto 7
1.2 Objetivos 7
1.3 Estructura de la memoria 8
2. Robots seguidores de línea: Fundamentos y arquitecturas. 9
2.1 Robots seguidores de línea 9
2.2 Componentes 10
2.3 Funcionamiento 17
2.4 Sensores CCD. 18
2.5 Aplicaciones Industriales 22
3. Linebot: El Robot seguidor de línea modular 25
3.1 Herramientas de Desarrollo: Open Hardware y OpenCV 25
3.2 Fases de desarrollo del proyecto: Fase 1 – CCD Lineal 32
3.3 Fases de desarrollo del proyecto: Fase 2 – Raspberry pi 42
3.4 Fases de desarrollo del proyecto: Fase 3 – Montaje e integración del Robot 51
4. Control del Robot 62
4.1 Controlador PID 62
4.2 Planteamiento del problema 65
4.3 Algoritmo de control 67
4.4 Ajustes y pruebas 70
5. Conclusiones 81
6. Apendice 82

6.1 Código Raspberry 82
6.2 Código Arduino 97
6.3 Código Windows 101
7. Referencias 103

Colaboraciones

Agradecimientos

Quiero agradecer este proyecto a mi madre y a mi hermana, porque son la razón de que haya acabado este proyecto.

También quiero agradecer todo el apoyo brindado por mi tutor de proyecto, Sergi Bermejo, porque, aunque ha sido un camino tortuoso, me ha dado mil oportunidades y facilidades para que este proyecto saliera adelante.

Por último, quiero agradecer a toda la gente que me ha dado las fuerzas para conseguir terminar este proyecto a tiempo.

Gracias a todos por vuestro apoyo,

David Ortiz

Barcelona, 16 de Octubre de 2016

Resumen del Proyecto

Los robots seguidores son apasionantes, ya que son una puerta de entrada a la robótica, y además permiten a la persona que los realiza tener la libertad de poderlo diseñar utilizando una variada gama de tecnología.

En este proyecto se intentará mejorar el sistema de detección de un seguidor de líneas, pasando de los sensores de infrarrojos a un sensor CCD.

Para ello, y en primer lugar, se analizarán los distintos tipos de robots seguidores de línea, que tecnologías típicas utilizan, y las mejoras que se pueden aplicar.

Posteriormente se detallarán las decisiones que se han tomado en este proyecto, qué tecnologías se quieren utilizar, cómo se van a utilizar y qué problemas se han tenido al implementarlas.

Seguiremos con el desarrollo específico del robot detallando: en qué punto se encontraba en cada fase, qué se hizo para intentar solucionar los problemas, y cómo quedo la electrónica asociada. Este punto nos llevará al montaje final del robot, al que hemos bautizado con el nombre de **Linebot**.

Por último se analizará el funcionamiento del robot, en particular cómo afronta los retos y toma las decisiones a partir de los estímulos de entrada que recibe, y las mejoras al diseño introducidas de cara a mejorar su rendimiento.

Abstract

The line following robots are exciting, because they are a gateway to robotics, and also allow the person who performs them have the freedom of being able to design using a variety of technology.

In this project is intended to improve the detection system of a line follower, moving from an infrared sensor to a CCD Sensor.

To do this, first and foremost, the different types of line follower robots are analyzed, typical technologies used, and improvements that can be applied.

Later we will talk about the decisions that have been taken in this project, which technologies are to be used, as will be used and what problems have been taken to implement them.

We will continue with detailing the specific development of the robot: at what point was in each stage, what was done to try to solve the problems, and how the electronics stayed. This point takes us to the final assembly of the robot, which we have named **Linebot**.

Finally the operation of the robot will be discussed, particularly how it faces the challenges and makes decisions based on the input stimuli it receives, and the design improvements introduced in order to improve their performance.

1. Introducción

1.1 Contexto del proyecto

En el mundo de la robótica, cuando hablamos de un robot seguidor de línea se piensa en un dispositivo que, utilizando unos fotodiodos, va siguiendo una línea negra sobre un fondo blanco. La reacción de estos fotodiodos es muy rápida y, por tanto, si se dispone de un buen dispositivo de control, el robot se mueve rápidamente por el mapa siguiendo ágilmente la línea. Este tipo de robots son muy populares y se pueden realizar verdaderas maravillas utilizando pocos recursos.

Partiendo de esta premisa, ¿qué pasaría si se utilizasen más sensores? Teóricamente, cuántos más sensores se colocarán antes se detectaría la posición del mismo respecto a la línea. Así, la reacción sería mucho más suave y los movimientos más fluidos. La pregunta que nos podemos plantear ahora es: ¿cómo añadir más sensores posibilitando a un robot de tamaño reducido un procesado eficiente?

Este proyecto surge a partir de la pregunta de qué pasaría si se sustituyen los típicos fotodiodos y se colocaran otro tipo que no hiciera que el tamaño y procesado del robot crecieran. La respuesta más obvia sería cambiar los sensores por una cámara CCD. Este tipo de dispositivos nos proporcionaran una información muy parecida a la que nos proporciona el fotodiodo, pero además nos proporciona información adicional, como la intensidad del color que se ha detectado, el tipo del color, etc.

1.2 Objetivos

Los objetivos de este proyecto se pueden dividir en hitos:

- Primero: Ser capaces de adquirir unas señales desde un sensor CCD para poderlas procesar.
- Segundo: Captar estas imágenes y tratarlas adecuadamente, para poder extraer el máximo de información posible.
- Tercero: Tratar la información que nos proporcionan estas imágenes y extraer todo lo que nos interesa para poder ser capaces de detectar líneas, colores, objetos, etc.
- Cuarto: Implementar una plataforma robótica para montar nuestro robot y toda la electrónica necesaria.
- Quinto: Añadir el software de captación de imágenes a la plataforma robótica, y desarrollar un sistema de control que haga mover el robot.
- Sexto: Realizar distintos escenarios de prueba para verificar el correcto funcionamiento del robot.

1.3 Estructura de la memoria

La memoria se puede dividir en tres grandes bloques. El primer bloque habla de toda la teoría en la que se basa este proyecto, desde las ideas iniciales hasta la base del desarrollo final. Se habla de los robots seguidores de línea, de las tecnologías para implementar las distintas partes del proyecto, y de los retos que nos podemos encontrar.

El segundo bloque entra en la parte práctica. En este bloque se quiere hablar de todo el hardware utilizado, las características y las razones por las que se ha decidido utilizarlo. Tan importante como el hardware, será el software, y en este proyecto se utilizan varias librerías que facilitan la interacción con los distintos módulos hardware. En este capítulo se habla de los pasos dentro del diseño del robot hasta llegar a su forma final. Por último, se hablará de todas las pruebas que se ha realizado con el robot.

La última parte será para recapitular todo lo que se ha aprendido y añadir unas conclusiones para un futuro trabajo en este robot.

2. Robots seguidores de línea: Fundamentos y arquitecturas.

Exactamente, ¿qué es un robot seguidor de línea? En este capítulo se explicará qué es, cuáles son los diferentes módulos que lo componen y cómo funcionan. Además, se quiere explicar el resto de tecnologías que se pretenden implementar en este proyecto.

2.1 Robots seguidores de línea

Estos tipos de robots se caracterizan porque son capaces de seguir un camino trazado por una línea. Esta línea normalmente es de un color que contrasta con el color del resto del suelo, es decir, si la línea es negra, el suelo será blanco, y viceversa.

Debido a su facilidad de implementación, estos robots son los que normalmente se utilizan para introducirse en la robótica, y se les suele llamar el "*Hello World*" de los robots, siguiendo la analogía con el "*Hello World*" típico de los lenguajes de programación.

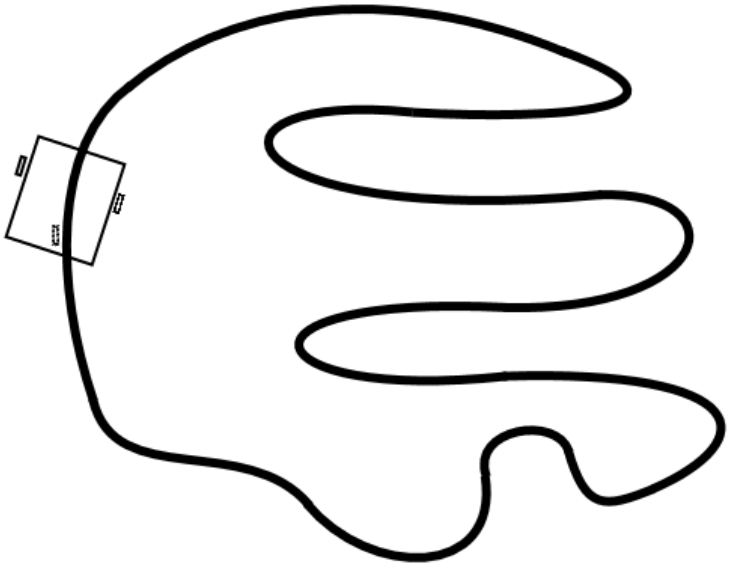

Figura 1.- Seguidor de línea recorriendo un circuito.

La gran virtud de esta configuración es que se puede realizar de múltiples maneras, y por tanto, pueden ser tan complicados como la persona que los realice quiera. Es decir, se pueden diseñar robots que funcionen con electrónica analógica y digital, utilizando lógica combinacional o amplificadores operacionales, hasta utilizar un microprocesador de última generación. Usualmente estos diseños suele utilizar microprocesadores de 8 o 16 bits.

Al ser tan fáciles de montar, existen multitud de competiciones que fomentan el desarrollo y mejora de los robots, añadiendo dificultades diversas que hagan que se tenga que diseñar un robot lo más versátil y veloz posible. Estas competiciones suelen tener niveles de dificultad, como por ejemplo:

- Líneas discontinuas.
- Bifurcaciones y cortes del camino.

- Subidas y bajadas.
- Falsos caminos
- Etc.

Estas competiciones suelen proporcionar unas reglas claras sobre los pesos y tamaños máximos y/o mínimos de los robots, tipos de circuitos, sistemas de puntuación, etc. Los robots reciben mejores puntuaciones contra más rápidos vayan, contra menos pérdidas del camino tengan, si se recuperan de una situación de error, etc.

Este tipo de robots también se utilizan para competiciones de velocistas, en las que dos robots se colocan en el mismo circuito con dos líneas, y gana el que antes recorre el circuito.

2.2 Componentes

Al haber tanta diversidad de montajes, no se puede entrar a explicar el funcionamiento de un tipo de componente u otro, ya que no se acabaría nunca. Pero lo que sí que se puede hacer es explicar las partes genéricas de este tipo de montaje, ya que esto sí que no varía en la multitud de configuraciones que existen:

- **CHASIS:**

La base del robot es su cuerpo. Este cuerpo puede ser de varias formas, colores y materiales, pero siempre debe ser capaz de soportar y mover el resto de componentes del robot. Este chasis puede ser móvil, puede ser modular, con niveles, de madera, de acero, etc. Las características de los materiales y diseños pueden influir en el funcionamiento del mismo. Un robot seguidor de línea debe contar con una estructura que el facilite el desplazamiento que lo mantenga en equilibrio en todo momento y que le permita cambiar de dirección fácilmente.

- **TRACCIÓN:**

Para generar el movimiento, se necesita algún dispositivo que proporcione una fuerza motriz para desplazar el chasis. Normalmente estos robots utilizan ruedas para desplazarse, porque son más ágiles y fáciles de manejar.

Estos equipos suelen utilizar baterías para alimentar los motores y electrónica, y por tanto, la fuente de alimentación suele ser corriente continua. Utilizando este tipo de fuente de alimentación, el tipo de dispositivo a utilizar suele variar respecto de las necesidades de cada equipo, aunque se suelen utilizar de tres tipos, de los cuáles comentaremos sus ventajas y desventajas:

  - **Motores de corriente continua [1]**: Este tipo de motores son muy interesantes porque proporcionan velocidades finales muy elevadas. El problema que tienen es que estos motores no se pueden clavar, es decir, no se pueden dejar parados en una posición, y para frenar este tipo de motores, se tiene que aplicar una inversión de polarización y emular una clavada de motor. Esto provoca un tiempo de reacción más lento que con otro tipo de motores. Además, igual que en la parada, también tiene una inercia de arranque mayor.

    Otra característica es que suministran una velocidad de giro muy elevada, pero su par de potencia es muy bajo. Eso obliga a colocar una caja reductora que proporcionará una potencia mayor a cambio de una velocidad de giro inferior.

    Un motor de corriente continua se compone de dos partes, el estator y el rotor. El rotor constituye la parte móvil del motor con el devanado y un núcleo, proporciona

el esfuerzo de torsión para mover a la carga. El estator constituye la parte fija de la máquina. Su función es suministrar el flujo magnético que será usado por el bobinado del rotor para realizar su movimiento giratorio.

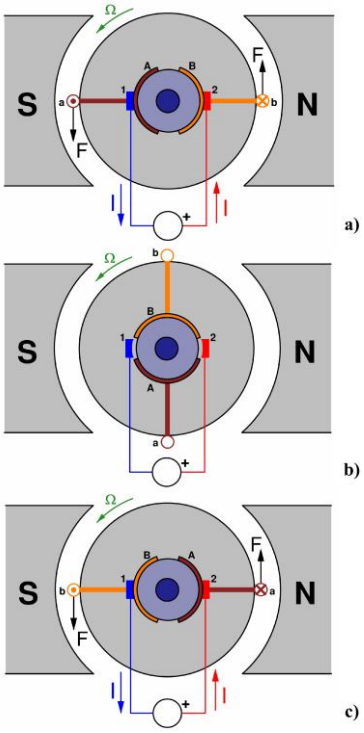

Figura 2.- Funcionamiento Motor CC

El movimiento giratorio de los motores de C.C. se basa en el empuje derivado de la repulsión y atracción entre polos magnéticos. Creando campos constantes convenientemente orientados entre el estator y el rotor, se origina un par de fuerzas que obliga a que el rotor gire buscando la posición de equilibrio.

En la imagen puede verse como aplicando una tensión en el rotor, se genera un movimiento.

- **Servomotores**: Son dispositivos capaces de llevar el motor a posiciones angulares específicas al enviar una señal codificada. Mientras esta señal exista en la entrada, el motor mantendrá la posición angular del engranaje. Si esta señal cambia, la posición cambia, y si desaparece, el motor deja de mantener la posición.

  Dentro de un servomotor hay un motor de corriente continua, una caja reductora, un potenciómetro y una electrónica de control.

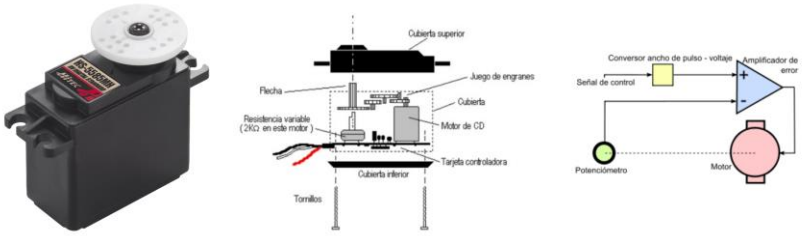

Figura 3.- Partes de un Servomotor

Para controlar estos dispositivos se utiliza un PWM (Pulse Width Modulation), y dependiendo de su anchura de pulso, es capaz de controlar el ángulo de giro, tal y como se puede ver en la siguiente imagen:

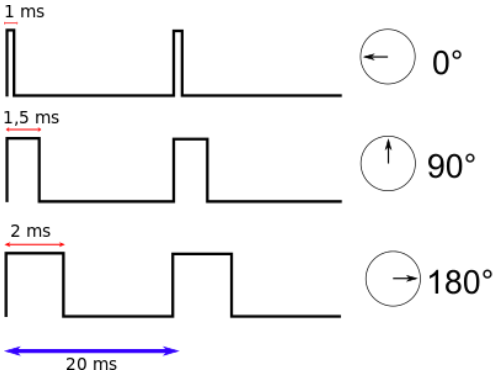

Figura 4.- Codificación de ángulo

Como se puede ver, si el pulso es de 1 ms de duración, el servomotor estará la posición de 0 grados. Si el pulso va aumentando y llega hasta los 2ms, el motor se quedará en la posición de 180 grados. Estos valores son indicativos, cada fabricante tiene su rango de valores.

El componente encargado de controlar el ángulo es el potenciómetro, es el sistema que realimenta el circuito de control para saber en qué posición está el servo. Si se quita este tope mecánico, se consigue que el motor gire como un motor de corriente continua. El control funcionará de la misma manera aunque, cuando se pongan el valor mínimo (por ejemplo, 1ms), el motor girará sin parar hacia la izquierda sin detenerse. Si por el contrario se pone el valor máximo, por ejemplo, 2ms, el motor girará sin parar hacia la derecha. Esta modificación hace que el servo pase a llamarse *servo de rotación continua.*

Como la señal enviada tiene una frecuencia de 50 Hz (periodo de 20ms) hace que el control de este dispositivo sea muy rápido, ya que cada 20 ms se puede realizar un cambio que el motor detectará. Además, con una frecuencia de refresco determinada, este motor puede quedarse completamente quieto. El problema general de este tipo de motores es que están pensados para realizar movimientos cortos y necesitar mucho par, y por tanto, hace que sean más lentos.

- **Motores paso a paso**: Es un dispositivo electromagnético que convierte impulsos eléctricos en movimientos mecánicos de rotación. La principal característica de estos motores es que se mueven un paso por cada impulso que reciben. Normalmente los pasos pueden ser de 1,8º a 90º por paso, dependiendo del motor. Son motores con mucha precisión, que permiten quedar fijos en una posición (como un servomotor) y también son capaces de girar libremente en un sentido u otro (como un motor DC).

  Cuando circula corriente por una o más bobinas del estator se crea un campo magnético creando los polos Norte-Sur. Luego el rotor se equilibrará magnéticamente orientando sus polos Norte-Sur hacia los polos Sur-Norte del

estator. Cuando el estator vuelva a cambiar la orientación de sus polos a través de un nuevo impulso recibido hacia sus bobinas, el rotor volverá a moverse para equilibrarse magnéticamente. Si se mantiene esta situación, obtendremos un movimiento giratorio permanente del eje. El ángulo de paso depende de la relación entre el nombre de polos magnéticos del estator y el nombre de polos magnéticos del rotor. En esta imagen se puede ver el funcionamiento:

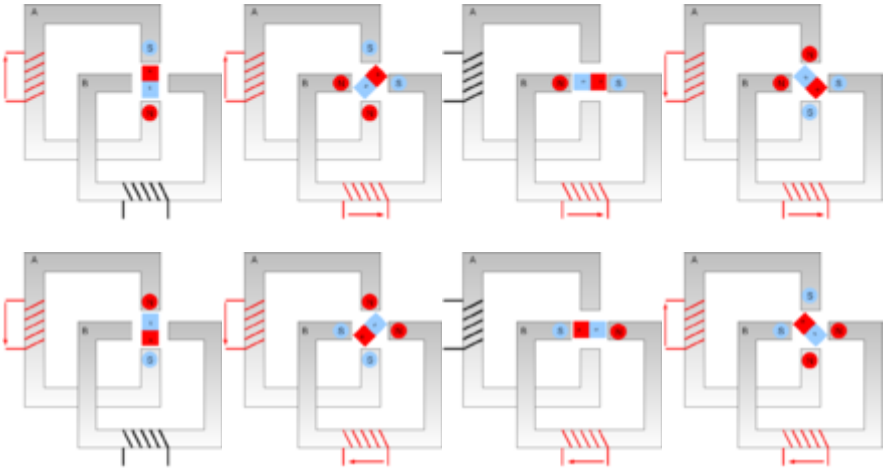

Figura 5.- Funcionamiento motor paso a paso (fuente Wikipedia [2])

La contraprestación de estos motores es su elevado consumo, que los hacen poco viables en un entorno donde el consumo es crítico.

Por último, hay que definir qué tipo de rueda se va a utilizar, y esta elección hará que el funcionamiento sea de una manera u otra. Ruedas pequeñas proporcionarán una velocidad inicial muy elevada, y ruedas de mayor diámetro garantizarán una velocidad final muy elevada. Ruedas anchas significará que tendrán mucho agarre en curvas, pero ruedas estrechas garantizarán menos superficie de rozamiento, y por tanto, mayor velocidad. Hay que escoger que se quiere sacrificar para poder ganar por otro lado.

Si se utiliza una configuración de cuatro ruedas, se pierde mucho de maniobrabilidad, y por tanto, muchas veces se utiliza una configuración de 3 ruedas, en las que las dos ruedas motrices se colocan o bien delante o detrás, y la tercera rueda es una rueda loca o rueda libre. Este tipo de rueda permite que el robot pueda girar libremente como si fuera un tanque, cambiando rápidamente de dirección.

Figura 6.- Rueda Loca

- **SENSADO:**

Aunque el equipo puede disponer de otro tipo de sensores –como por ejemplo, sensores de distancia–, el sensor clave en el diseño de un sigue-líneas es un sensor de infrarrojos. Es el que se encargará de detectar si el equipo está dentro o fuera de la línea.

Un sensor de infrarrojos se compone de dos partes, un diodo emisor infrarrojo y un fototransistor que opera en la misma longitud de onda. Estos sensores suelen ser de corto alcance, y cuando hay una superficie suficientemente plana y sólida lo bastante cerca, la señal emitida por el diodo es recibida por el fototransistor.

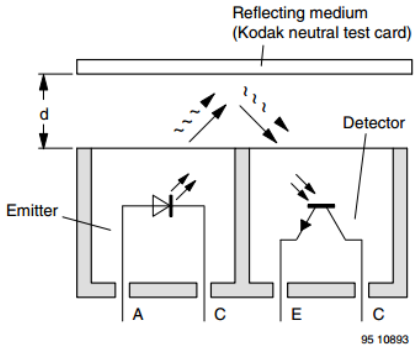

Figura 7 .- Detalle de funcionamiento del sensor

El nivel de la señal recuperada por el fototransistor dependerá de cuan cerca se encuentre el sensor de la superficie de referencia. De la misma manera, una superficie blanca refleja mejor la señal que una superficie negra. Esta es la característica que proporciona la capacidad de implementar la detección de línea con estos sensores. Dicho de otro modo, gracias a este tipo de sensor y una buena colocación se puede discriminar el color blanco del negro.

El problema que tienen estas señales es que son analógicas, y por tanto, se deberá añadir una electrónica capaz de transformar esta señal analógica en una señal digital que la unidad de control sea capaz de procesar.

Ejemplos de estos tipos de sensores podría ser el ***PC817*** de Sharp o el ***CNY70*** de Vishay.

- **ALIMENTACIÓN:**

Al ser un dispositivo móvil, la alimentación debe ser móvil también, y por tanto, la mejor solución es utilizar algún medio de almacenamiento de tensión, ya sean pilas, baterías, o paneles solares. La elección de la fuente de alimentación es especialmente importante, porque dependiendo de la capacidad que tenga, condicionará la velocidad y autonomía del robot.

Por tanto, cuando se escoja la batería se deben tener en cuenta los siguientes puntos:

- **Consumo máximo del robot:** A partir de las características de todos los componentes, se tiene que mirar cuál será su consumo máximo (motores a pleno rendimiento, electrónica consumiendo al máximo, etc.), y una vez determinado, buscar una fuente de alimentación que sea capaz de proporcionar esta corriente.

- **Pico de consumo máximo:** Cuando un motor se pone en marcha, se origina un pico de consumo por la resistencia que tiene a moverse. Este pico será mayor cuanta más velocidad se da y/o mayor carga se desee mover. La batería debe ser capaz de proporcionar estos picos y no ver comprometido su funcionamiento. Si no fuera capaz, provocaría una bajada de tensión para proporcionar esta corriente, y podría apagar el resto de la electrónica. Este valor se suele encontrar en las baterías como C. Por ejemplo, si la batería es de 1000 mAh con un C de 10, podrá suministrar picos de hasta 10000 mA, o lo que es lo mismo, 10 Amperios.

- **Capacidad:** Todas las baterías recargables dan una cifra de carga, se suele expresar en Ah (Amperios/hora) o mAh (miliamperios/hora). Esta cifra indica cuantos amperios/hora es capaz de dar la batería antes de descargarse. Por tanto, si el consumo es de 10 mA y la batería es de 100 mAh, el equipo podrá estar en marcha durante 10 horas, si por el contrario el consumo es de 100mA y la batería es de 10mAh, el equipo solo podrá estar en funcionamiento 6 minutos.

- **Voltaje:** Este valor dependerá de las tensiones que se necesiten en el circuito, ya sea por parte del motor, o por parte de la electrónica de control.

Existen muchas tecnologías de baterías: Alcalinas, NiCd, NiMH, Li-ion, etc. En esta gráfica se pueden ver la diferencia entre ellas:

	Nickel-cadmium	Nickel-metal-hydride	Lead-acid sealed	Lithium-ion cobalt	Lithium-ion manganese	Lithium-ion phosphate
Gravimetric Energy Density (Wh/kg)	45-80	60-120	30-50	150 - 190	100 - 135	90 - 120
Internal Resistance in mΩ	100 to 200[1] 6V pack	200 to 300[1] 6V pack	<100[1] 12V pack	150 - 300[1] pack 100 -130 per cell	25 – 75[2] per cell	25 – 50[2] per cell
Cycle Life (to 80% of initial capacity)	1500[2]	300 to 500[3,4]	200 to 300[3]	300 - 500[3]	Better than 300 – 500[4]	>1000 lab conditions
Fast Charge Time	1h typical	2 to 4h	8 to 16h	1.5 - 3h	1h or less	1h or less
Overcharge Tolerance	moderate	low	high	Low. Cannot tolerate trickle charge.		
Self-discharge / Month (room temperature)	20%[5]	30%[5]	5%	<10%[6]		
Cell Voltage Nominal Average	1.25V[7]	1.25V[7]	2V	3.6V 3.7V[8]	Nominal 3.6V Average 3.8V[8]	3.3V
Load Current peak best result	20C 1C	5C 0.5C or lower	5C[9] 0.2C	<3C 1C or lower	>30C 10C or lower	>30C 10C or lower
Operating Temperature[10] (discharge only)	-40 to 60°C	-20 to 60°C	-20 to 60°C	-20 to 60°C		
Maintenance Requirement	30 to 60 days	60 to 90 days	3 to 6 months[11]	not required		
Safety	Thermally stable, fuse recommended	Thermally stable, fuse recommended	Thermally stable	Protection circuit mandatory; stable to 150°C	Protection circuit recommended; stable to 250°C	Protection circuit recommended; stable to 250°C
Commercial use since	1950	1990	1970	1991	1996	2006
Toxicity	Highly toxic, harmful to environment	Relatively low toxicity, should be recycled	Toxic lead and acids, harmful to environment	Low toxicity, can be disposed in small quantities		

Figura 8.- Comparativa entre tecnologías de fabricación de baterías [3]

En muchos diseños, por facilidad de uso, se suelen utilizar baterías de usar y tirar, ya que no son muy caras y se pueden adquirir con facilidad.

Las baterías de NiMH también se utilizan mucho, porque son baratas, no tienen efecto memoria (efecto que se produce cuando una no se descarga completamente antes de volverla a cargar), y no son excesivamente contaminantes. Para este tipo de equipos tienen bastantes inconvenientes, como su tamaño, ya que como el valor de voltaje nominal por celda es de 1,2V, para conseguir valores de tensión elevados, se deben poner muchas celdas en serie, no son capaces de dar mucha corriente de pico, tienen una tasa de descarga sin uso muy elevada y su tiempo de carga es más lento.

Las baterías que más se utilizan son las de Litio, más concretamente las llamadas LiPo (Litio Polímero). Tienen un voltaje nominal de 3,7 V, y por tanto, con pocas celdas consigues valores de tensión muy elevados. Tienen una potencia de descarga muy elevada, llegando hasta 60C o más. Además se cargan mucho más rápido que las de NiMH. Como es natural, también tienen problemas.

Uno de ellos es la inversión de polaridad. Si se deja descargar la batería durante mucho tiempo se produce un efecto llamado pasivación, que es parecido al efecto memoria, y hace que la batería pierda capacidad de carga o que hasta deje de funcionar. Por otro lado, cuando se cargan, se debe tener especial cuidado en los márgenes de tensión de cada celda, ya que si se supera una tensión de 4,2 V la batería se puede destruir. Otra manera de destruir una batería de Litio es cargarla con mucha corriente, ya que puede hacer subir la temperatura del conjunto y que se destruya.

Después de escoger la tensión de la batería, se deberán ajustar las tensiones para cada parte de la electrónica, y para ello se utilizan reguladores de tensión lineales. Un regulador de tensión lineal, a partir de una tensión de entrada, suministra una tensión de salida estable inferior. Lo que este tipo de dispositivos nos proporciona es que la tensión de salida será estable, mientras la tensión de entrada este dentro de los márgenes. Un ejemplo se puede ver en la siguiente figura:

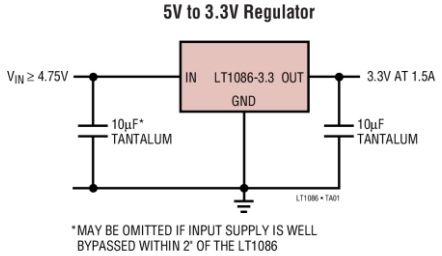

Figura 9.- Regulador de tensión LT1086 de Linear

En la imagen se puede ver un regulador de tensión de Linear, el ***LT1086***, que a partir de una tensión de entrada mayor o igual de 4,75V, proporciona una salida estable de 3,3 Voltios y 1,5 Amperios. Esta tensión podría ser la que se utilizase para alimentar la electrónica de control

En muchos robots se utilizan dos baterías separadas. Utilizando esta configuración separamos la parte digital (sistema de control y sensado) de la parte analógica (motores). Los motores son muy ruidosos, y a través de la alimentación pueden inducir ruidos al resto de circuitería. En el mejor de los casos el sistema será lo suficientemente robusto para soportar estos problemas, pero muchas veces este ruido falsea las medidas, o hasta puede provocar que la parte del control se resetee.

- **CONTROL:**

La parte del control es la que más difieren en los distintos seguidores de línea. Como ya se ha comentado, utilizando un adaptador de analógico/digital en la parte de los sensores, y colocando lógica combinacional se puede realizar un seguidor de línea. Pero obviamente, se puede complicar, y puede ser que lo que se quiera poner es un microprocesador de última generación que efectúe un control lo más velozmente posible.

Por tanto, cada aplicación tiene unos requisitos, y hay que ajustar la parte de control para poder cumplir holgadamente estos requisitos. Si por ejemplo se necesita un sistema para efectuar el control utiliza un sistema de redes neuronales, lógica difusa y técnicas de aprendizaje automático, se debe utilizar un procesador lo más potente posible, de manera que sea capaz de efectuar todas las operaciones necesarias para que las reacciones del robot sean lo más fluidas posible.

2.3 Funcionamiento

Utilizando todos los componentes que se han comentado, se puede montar un robot seguidor de línea. La versión más sencilla utiliza dos sensores, ajustados a la anchura de la pista, y cuando uno de los dos se active, significará que el robot está saliendo de la línea o que se aproxima un giro, y por tanto, se tiene que ajustar el movimiento.

Como hay dos sensores, los resultados se pueden resumir en esta tabla de estados

Sensor izquierda	Sensor derecha	Acción a realizar
0	0	No se detecta línea, el robot está parado
0	1	Se detecta que el sensor izquierda está fuera de la línea, y por tanto, se tiene que iniciar un giro a la derecha para volver a la trazado
1	0	Se detecta que el sensor derecho ha salido fuera de la línea, y en este caso se activa el giro a la izquierda para volver al trazado.
1	1	El robot está dentro de la línea, continua recto.

Tabla 1.- Tabla de estados de los sensores

Este funcionamiento se puede ver en el siguiente dibujo:

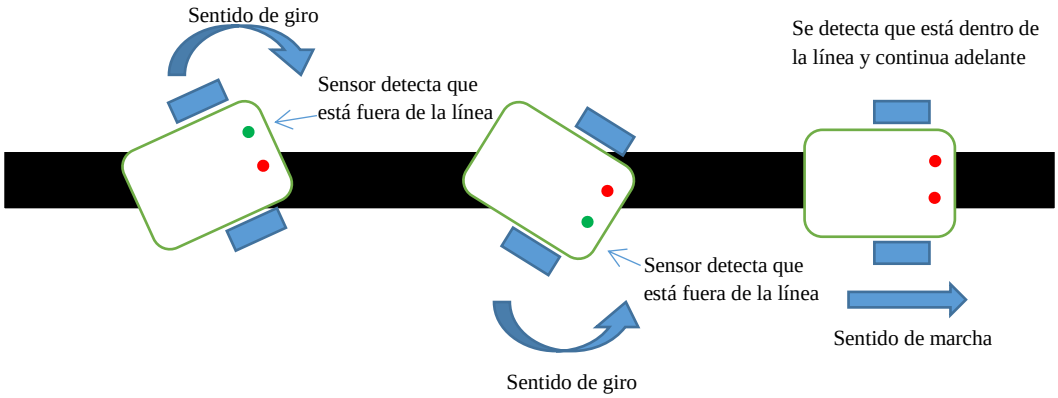

Figura 10.- Esquema de detección de línea

Como se puede apreciar en la figura, cuando uno de los dos sensores detecta que esta fuera de la línea, se gira en el sentido contrario, para volver a colocarse en la línea. El giro se suele efectuar parando la rueda contraría al giro, y dejando en marcha la otra, tal y como funciona un tanque.

Si los dos sensores están activos, el robot está dentro de la línea y así se pueden activar los dos motores a la vez avanzando.

Esta configuración funcionará muy bien en robots que funcionen en una línea continua y con giros muy suaves, porque el robot no será capaz de detectar bifurcaciones o giros muy abruptos. Por otro lado, los movimientos también serán muy bruscos, porque el movimiento se suele realizar parando una de las dos ruedas para efectuar un giro rápido.

Una manera de detectar bifurcaciones y tener movimientos más fluidos, es utilizando más sensores, colocados de forma que sean capaces de detectar los cambios del robot de una manera más suave y precisa, y de esta manera, se podrán detectar las bifurcaciones. Además, al tener más

sensores, se tiene una mayor capacidad para detectar la posición en la que se encuentra el robot, y por tanto, el movimiento será mucho más fluido. Un ejemplo de configuración podría ser la mostrada en la figura de abajo.

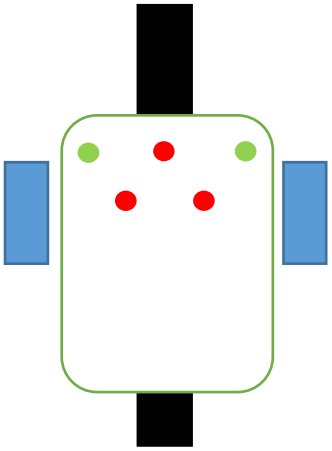

Figura 11.- Configuración con 5 sensores

Con esta configuración se podrá detectar cuando haya una bifurcación o giro de 90º, ya que mientras detecte línea funcionarán los tres sensores centrales, pero si hay una bifurcación brusca, los sensores externos se activarán, y el control podrá efectuar un movimiento para adaptarse al tipo de pista.

2.4 Sensores CCD.

Cuando se quiere mejorar la sensibilidad del sistema, la idea más obvia es añadir más sensores, de esta manera se podrá afinar mucho más la posición del robot respecto a la línea. El problema es que contra más sensores se añadan, más consumo tendrá el robot y más espacio para colocar sensores se necesitará. Para añadir sensibilidad al sistema y no aumentar el tamaño del robot se añaden sensores CCD, que pueden detectar la línea en un ángulo de visión muy amplio.

- **¿Qué es un sensor CCD?**

  El CCD se inventó a finales delos 60 por investigadores de *Bell Laboratories*. Originalmente se concibió como un nuevo tipo de memoria de ordenador, pero pronto se observó que tenía muchas más aplicaciones potenciales, tales como el proceso de señales y sobretodo la captación de imagen, esto último debido a la sensibilidad a la luz que presenta el silicio.

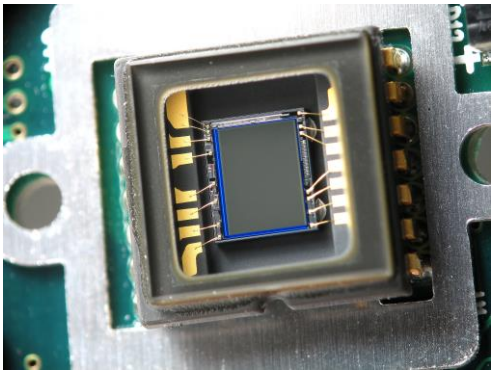

Figura 12.- Detalle sensor CCD

Las siglas CCD provienen del inglés *charge-coupled device*, o lo que es lo mismo, dispositivo de carga acoplada. Es una superficie sólida sensible a la luz, dotada de unos circuitos que permiten leer y almacenar electrónicamente las imágenes que se proyectan sobre ella. El funcionamiento de los CCD se basa en el fenómeno físico del efecto fotoeléctrico. Ciertas sustancias tienen la propiedad de absorber quantos de luz, o fotones, y liberar un electrón.

Tienen dos diferencias básicas con los fotomultiplicadores:

- Los sensores CCD son de menor tamaño y están construidos de semiconductores lo que permite la integración de millones de dispositivos sensibles en un solo chip.
- La eficiencia cuántica de los CCD (sensibilidad) es mayor para los rojos. Los fotomultiplicadores son más sensibles a los azules.

Una de las características que tienen los CCD's es que son muy sensibles al espectro infrarrojo. Eso hace que las cámaras dispongan de filtros para estas longitudes de onda.

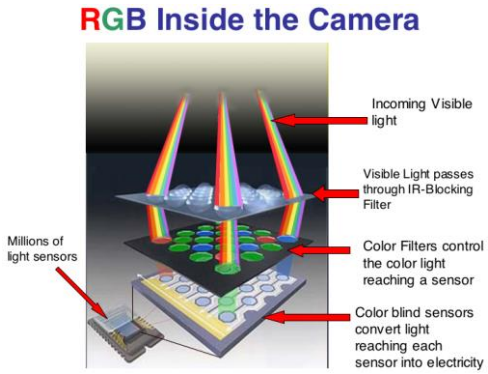

Figura 13.- Dentro de un sensor de una Cámara (Fuente Photoaxe [4])

- **¿Cómo funciona un CCD?:**

Para la fabricación de los detectores CCD se utiliza el silicio, que es un material semiconductor. Una de las caras de una placa de silicio se recubre con una red de electrodos microscópicos cargados positivamente. En virtud del efecto fotoeléctrico, la luz incidente genera electrones, de carga negativa, que son atraídos por los electrodos y se acumulan a su alrededor. La imagen final captada por el detector CCD es un mosaico formado por tantos elementos, o teselas, como electrodos hay en la placa de silicio. Se suele llamar píxeles a las teselas de los mosaicos digitales.

Periódicamente se lee el contenido de cada pixel haciendo que los electrones se desplacen físicamente desde la posición donde se originaron (en la superficie del chip), hacia el amplificador de señal con lo que se genera una corriente eléctrica que será proporcional al número de fotones que llegaron al pixel. Para coordinar los periodos de almacenamiento (tiempo de exposición) y vaciado del pixel (lectura del pixel) debe existir una fuente eléctrica externa que marque el ritmo de almacenamiento-lectura: el reloj del sistema. La forma y amplitud de reloj son críticas en la operación de lectura del contenido de los pixeles.

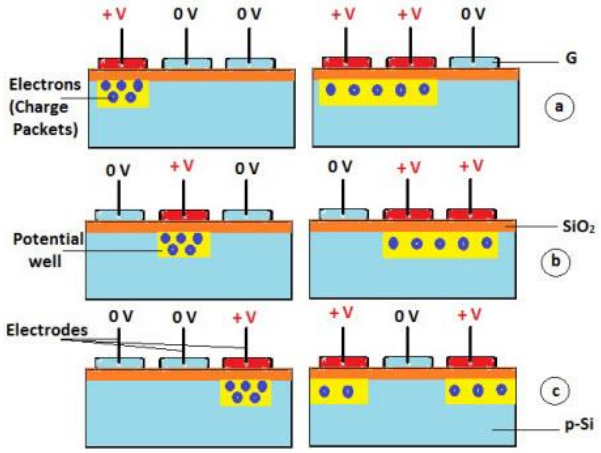

Figura 14.- Funcionamiento CCD (Fuente www.elprocus.com [5])

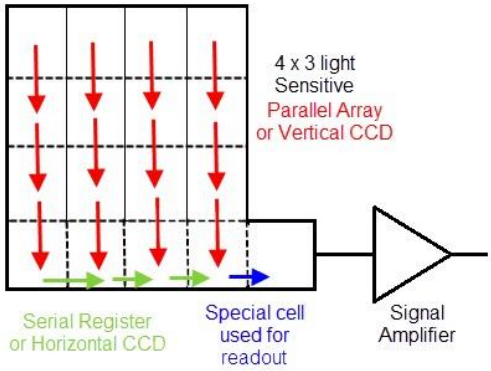

Figura 15.- Envío de datos CCD (Fuente www.elprocus.com)

- **Tipos de CCD:**

Existen varios métodos de captura de imágenes con CCD:

Arrays lineales

- **Sensor lineal.** Los conjuntos lineales usan una fila única de pixeles que escanea linealmente la imagen. Los que tienen un solo CCD hacen tres exposiciones por separado: rojo/verde/azul (RGB). Estos modelos se empezaron a utilizar en los primeros escáneres Todavía son usados para capturar imágenes de objetos que no se mueven.
- **Sensor Trilineal.** Se trata de tres CCD lineales unidos que se combinan para capturar cada uno de los canales RGB en un solo barrido, son los que dan la resolución más alta y la gama espectral más rica. Se emplean en los escáneres de sobremesa y diapositivas.

Array de superficie

Son los más empleados actualmente en cámaras digitales, consisten en una superficie donde existen miles de pixeles sensibles a la luz organizados en filas y columnas (una matriz). El CCD es sensible a los fotones de cualquier longitud de onda en mayor o

menor grado (en general es más sensible a los rojos e infrarrojos y menos a los azules). Todos los CCD son, por tanto, monocromáticos, y no tendremos ningún problema para capturar imágenes monocromas. Para obtener fotografías en color con dispositivos CCD se han desarrollado distintas tecnologías, las más empleadas son:

- **MOSAICO DE CCD.** El CCD único con máscara de color (CCD en mosaico) es el que se emplea en la mayor parte de las cámaras de video digital o analógico y en las cámaras fotográficas digitales de color.

  Antes de llegar al pixel, la luz pasa por un filtro que solo deja pasar los fotones de la longitud de onda deseada. Cada pixel solo puede tener un filtro y por tanto solo es sensible a un color, el CCD se convierte en un mosaico de pixeles sensibles respectivamente al rojo, verde y azul. Como es lógico, en el pixel en el que se recoge información de un color, rojo por ejemplo, no se puede captar la información del resto de los colores. La información de un color en los pixeles que no son sensibles al mismo se deduce por interpolación a partir de los pixeles vecinos de ese color. Debido a la interpolación, que por óptima que sea nunca es real, las imágenes captadas con CCD en mosaico dan un cierto grado de borrosidad lo que las hace ser de baja calidad.

  Una solución a éste problema, que se emplea en las cámaras domésticas de video y fotografía digital, es aumentar porcentualmente los pixeles sensibles al verde (el ojo humano es mucho más sensible a éste color) de modo que los tonos verdes tienen mucha menos borrosidad que los rojos o azules y el conjunto de la imagen gana en definición.

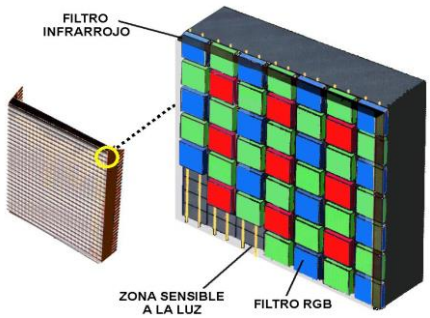

Figura 16.- Detalle de un CCD en mosaico de color (Fuente UAM)

- **CCD triple (triCCD).** La luz es descompuesta por prismas ópticos y desviada a tres sensores CCD, uno para cada color básico. Los sensores para el verde y rojo suelen ser idénticos pero el sensor azul suele estar optimizado para este color. Las cámaras construidas con esta tecnología son mucho más caras que el resto no solo porque tienen que triplicarse los componentes sino porque los CCD deben estar perfectamente ajustados para que la luz de un mismo punto del objeto incida exactamente en las mismas coordenadas de pixel de cada uno de los CCD. Las cámaras tri-CCD son la mejor opción: permiten capturar imágenes en movimiento con una gran resolución y calidad cromática, el gran inconveniente es su precio por lo que esta tecnología solo se emplea en cámaras profesionales.

- **CCD único con exposición triple**. Consiste en un único CCD que es expuesto sucesivamente a los tres colores. El modo de conseguir imágenes de los tres colores es a través de un filtro que se coloca delante del CCD, luego se superponen las tres para obtener la imagen de color. Los filtros pueden ser de cristal (implica que la cámara debe disponer de un dispositivo mecánico que vaya cambiando cada filtro de modo secuencial), o de cuarzo líquido, este último permite cambiar de

color al aplicarle distintos voltajes lo que abarata y simplifica el funcionamiento del sistema.

El método de exposición triple permite obtener imágenes de una calidad equivalente al tri-CCD pero solo de objetos estáticos ya que se necesita un tiempo para captar las tres imágenes.

2.5 Aplicaciones Industriales

En los entornos industriales se utilizan robots seguidores de línea, pero unas versiones mucho más optimizadas y preparadas, ya que en la implementación que se ha descrito hasta ahora hay muchas variables del entorno que no se controlan.

Este tipo de robot se le llama Vehículo guiado automatizado (*Automatic Guided Vehicle* o en siglas *AGV*). Además de seguir una línea, suele añadir sensores de distancia para evitar chocar con gente u otros vehículos, radares, visión artificial, etc.

Las aplicaciones de estos robots suelen estar relacionadas con el trasporte de mercancías dentro de la misma empresa o almacenaje. Se suele crear una red de robots que mueven la producción de un sitio hacia un almacén, o hacia otro punto de la cadena de fabricación. Como ejemplos podríamos encontrar:

- **Modelo Weasel® de la marca SCHAEFER [6]**

  Este modelo está pensado para transportar pequeñas cargas (hasta 35 Kg) por una empresa, automatizando los desplazamientos internos. Para funcionar se debe crear unos caminos utilizando unas bandas de seguimiento óptico, que no son más que líneas negras, aunque también dispone de otros medios de seguridad para evitar posibles accidentes.

Figura 17.- Robot Weasel (Fuente Schaefer)

Utilizando todos los accesorios de la compañía, se pueden instalar puestos donde el Weasel recoge o deposita la mercancía, como se puede apreciar en la figura que sigue.

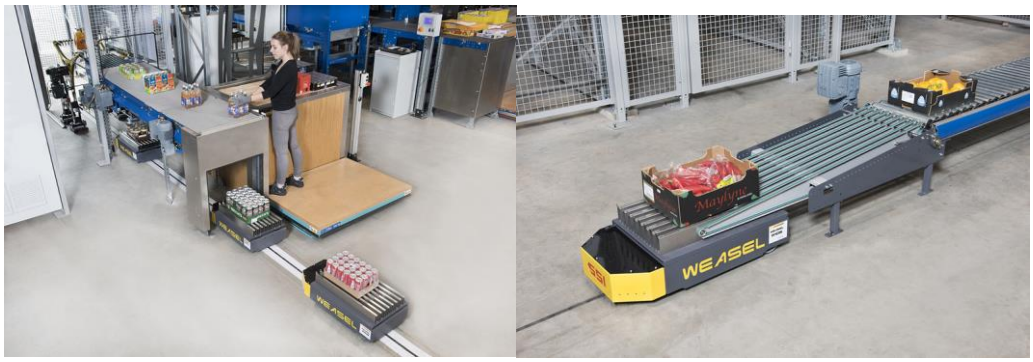

Figura 18.- Carga y descarga de mercancías (Fuente Schaefer)

La misma empresa suministra un software para crear una red de caminos y puntos de control y poder gestionar así una red de robots (ver **Figura 18**).

Figura 19.- Ejemplo de instalación (Fuente: Schaefer)

- **Modelo CEITruck® de la marca CEIT Technical Innovation [7]:**

  La marca CEIT dispone de un modelo parecido al Weasel, pero en vez de cargar el material encima, lo lleva con remolque.

Figura 20.- Robot CEITruck (Fuente CEIT TI)

La gran diferencia respecto al anterior modelo es que es capaz de desplazar mucha más carga (existe modelos para desplazar desde los 500 kg hasta los 3000kg), y está más pensado para desplazar pallets de un lugar de la fábrica a otro lugar. De la misma manera que el otro modelo, este robot se integra en un sistema global de control (**Figura 21**).

Figura 21.- Sistema integrado (Fuente CEIT TI)

Con estos dos ejemplos se quiere ilustrar que los robots seguidores de línea son un buen punto de partida para sistemas más complejos.

3. Linebot: El Robot seguidor de línea modular

En el proyecto, se han producido varias iteraciones en el diseño del robot. La primera utilizaba un sensor CCD lineal y el módulo de control se implementaba utilizando un PIC. Durante su desarrollo surgieron varios problemas que hicieron descartar el proyecto tal y como se había planteado, y se recondujo a un proyecto con un sistema de control basado en un módulo con un microprocesador, la *Raspberry PI*, y añadiendo la cámara oficial para implementar el sistema de visión artificial. Por último, se ha decidido implementar un sistema con módulos comprados en diferentes proveedores, tanto físicos como por Internet, y utilizando como sistema de control la *Raspberry PI*, apoyada con un módulo de *Arduino* (Este es su nombre comercial en EEUU, en Europa se llama *Genuino*, aunque el nombre de *Arduino* es el más popular). Se quiere demostrar que con pequeños módulos se puede montar un robot de una manera sencilla, pero con una potencia de procesado elevada.

3.1 Herramientas de Desarrollo: Open Hardware y OpenCV

Desde hace unos años se ha extendido por todo el mundo la cultura *Maker*. Esta cultura se caracteriza por seguir una filosofía DIY (*Do It Yourself* o hágalo usted mismo) e introducirse en desarrollos utilizando Hardware y software sin la necesidad de tener profundos conocimientos en el tema. Esta cultura ha venido acompañada de la creación de lugares de desarrollo, como los *Makerspaces* y los *FabCafes*, donde la gente puede reunirse para desarrollar sus nuevos proyectos, compartir ideas y conocimiento, y se tiene acceso a herramientas de elevado coste por un precio moderado, como impresoras 3D, cortadoras por láser, etc.

Además, se ha abierto al gran público lenguajes de programación sencillos e intuitivos para desarrollar programas como el Scratch (creado en el MIT y que trata de acercar la programación a niños utilizando bloques para programar), se ha extendido el uso del lenguaje de programación Python (Creado por Guido van Rossum como un lenguaje de alto nivel muy orientado a realizar un código fácilmente legible), y además se han desarrollado multitud de tarjetas con licencia *Open Hardware*[8] (o sin esta licencia) que resultan muy económicas, y disponen de muchos dispositivos orientados para trabajar con ellas (como sensores, displays, etc).

En el software desde hace tiempo existe el término de *Software Libre* [9]. Se refiere al conjunto de software que por elección manifiesta del autor, puede ser copiado, estudiado, modificado, utilizado libremente con cualquier fin y redistribuido con o sin cambios o mejoras. Su definición está asociada al nacimiento del movimiento del software libre, encabezado por Richard Stallman, y con su asociación, la *Free Software Foundation* [10], coloca la libertad del usuario informático como propósito ético-fundamental. En este movimiento el software sí que dispone de licencia, es decir, el software es del autor, pero el autor ha permitido que se pueda utilizar por otros usuarios. Hay muchos tipos de licencias: GPL, MIT, Copyleft, AGBL, etc, y cada una dispone de sus restricciones y consideraciones.

En el mundo electrónico del hardware, esta licencia funciona de un modo parecido, pero con matices. Además de *Open* Hardware, se puede llamar de muchas otras maneras, Hardware libre, hardware de código abierto, electrónica libre, o maquinas libres, y se llama así a los dispositivos de

hardware cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea bajo algún tipo de pago o de forma gratuita.

En un diseño open hardware están integrados todos los documentos necesarios para fabricar el dispositivo (Usualmente Gerbers, o en su defecto los ficheros PCB para poder generarlos, y las listas de materiales), aunque a veces, las licencias son limitadas, y no se incluyen los códigos exactos de los componentes utilizados, solo un esquema genérico.

Como dentro del hardware se utilizan dispositivos que pueden necesitar ser programados, como por ejemplo Dispositivos de lógica programable (FPGA) y dispositivos hechos a medida (ASICs), también se distribuye el código HDL, para poder programarlos.

- **ARDUINO / GENUINO [11]:**

  Dentro del Hardware libre, el desarrollo que más ha marcado ha sido el de Arduino. La idea del Arduino [12] se comenzó a gestar en el instituto IVREA (Italia), para desarrollar una placa de evaluación para los estudiantes de bajo costo, ya que las placas de evaluación que se utilizaban en aquellos tiempos en el instituto costaban 100 $ y resultaban muy caras. El desarrollo fue promovido por Massimo Banzi, y se comenzó a realizar como partes de tesis de estudiantes, y su primer nombre fue *Wiring*. Una vez finalizado el proyecto, se incorporaron David Cuartielles, Tom Igoe, David Mellis, Nicholas Zambetti y Gianluca Martino, para poder comenzar a desarrollar un prototipo funcional, y un entorno de programación realizado en Processing.

  Debido a que se sabía que el instituto iba a cerrar, y por miedo a perder todo el trabajo realizado, se decidió abrir el diseño a la comunidad y declararla como proyecto *Open Hardware*, de tal manera, cualquiera podría fabricar una tarjeta, pagando unas regalías a los creadores.

  Sin que los otros fundadores de Arduino lo supiesen, Gianluca Martino, a través de su empresa de montajes electrónicos, adquirió la patente del nombre en Europa, lo que llevo, al cabo del tiempo, a que la sociedad se dividiese, con la pérdida de las patentes de uso del nombre de Arduino en Europa. Es por ello que en Europa el Arduino tiene el nombre de *Genuino*.

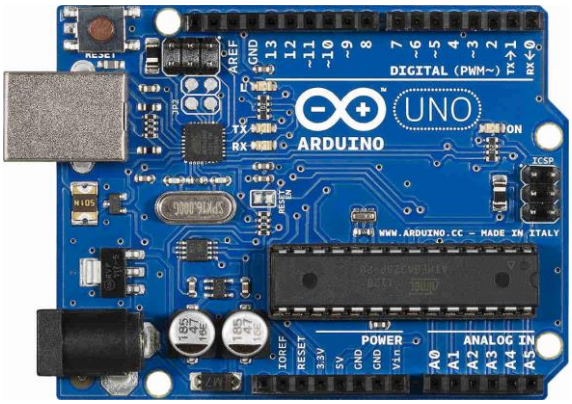

Figura 22.- Arduino UNO

El hardware es una placa de circuito impreso con un microcontrolador, normalmente de la marca Atmel, con un puerto USB para programar y alimentar el equipo, una entrada de alimentación, y múltiples puertos digitales y analógicos de entrada/salida a los que se le

pueden conectar tarjetas de expansión que dan capacidades diversas al sistema, desde añadir un giroscopio, diplays, módulos Ethernet, etc.

La targeta Arduino más popular es el *Arduino Uno*, que tiene un microprocesador ***ATMega328P*** de 8 bits, pero existen versiones de Arduino que incluyen procesadores de 32 bits basados en arquitecturas ***ARM-M3*** (Arduino Due), o el chip ***Intel Curie***, cuya arquitectura es x86 (Arduino 101). Esto si estamos hablando de las versiones oficiales, porque también existen una gran cantidad de versiones no oficiales con las mismas características de las tarjetas originales, como por ejemplo *Freeduino*, *Diabolino*, etc… Por último, también existen tarjetas que adaptan el sistema de puertos de entrada y salida del Arduino, pero añaden muchas más funcionalidades, incluyendo un IDE de programación diferente.

Los microprocesadores de Arduino tienen instalado un Bootloader, que hace que se puedan programar sin necesidad de utilizar el programador oficial del chip de ***Atmel***.

El lenguaje de programación de Arduino es C, y para compilarlo, existe un programa gratuito que se puede descargar desde la página oficial (Llamado Arduino también), que incluye todas las herramientas de programación de todas las tarjetas oficiales, un editor de texto para el código, y diversas herramientas para poder depurar el código, como por ejemplo un puerto serie. Lo mejor de esta herramienta es que es compatible para Windows, Linux y Mac.

Figura 23.- Pantalla IDE Arduino

- **RASPBERRY PI [13]:**
    - Historia [14]

La idea de Raspberry PI comenzaría a gestarse en el 2006, pero no acabaría viendo la luz hasta el Febrero del 2012. Así como Arduino quería acercar la programación y la electrónica a la gente sin conocimientos previos, Raspberry surgió con la idea de fomentar la enseñanza de las ciencias de computación en las escuelas. Igual que en el caso de Arduino, también existe una fundación Raspberry para gestionar todo lo relacionado con descargas de software, gestión de la comunicación entre usuarios, etc.

El plan inicial era el de construir 1.000 unidades para estudiantes de la Universidad de Cambridge. El precio de venta sería de 35 dólares, aunque el coste de fabricación de estos

dispositivos era de 36 dólares. Haciendo números no es difícil darse cuenta que este movimiento les llevaría a unas pérdidas iniciales de 1.000 dólares, pero era un coste relativo, ya que esperaban poder utilizar a los universitarios como testeadores de la tarjeta, y de esta manera poder depurar, documentar y mejorar el dispositivo. Este coste inicial de 1.000 dólares se planteó como una inversión más que como unas pérdidas.

Durante finales del 2011 se pusieron 10 tarjetas a la venta en eBay, para comenzar a preparar el terreno para la llegada de la tarjeta final en febrero del 2012. Todo lo que parecía una estrategia de marqueting comedida, se les fue totalmente de las manos, y a tres semanas del lanzamiento de la Raspberry Pi, los servidores se colapsaron con el aumento de los pedidos en más de 200.000 unidades. Contando que el dispositivo inicial tenía unas pérdidas de 1 $ por tarjeta, aquello estuvo a punto de acabar con el proyecto antes de iniciarse. A raíz de esta gran cantidad de pedidos a solo tres semanas del lanzamiento, hizo inviable construirla en el Reino Unido como se esperaba al principio, y por tanto la fabricación se envió a China, y la compra de los componentes se gestionó a través de las compañías de distribución de componentes Farnell y RS para reducir costes.

Figura 24.- Raspberry Pi 1 B

El siguiente paso era como iban a publicar información sobre el dispositivo y a la vez evitar que alguien copiara el proyecto y sacara un dispositivo igual o parecido, pero más barato, ya que se ahorrarían el coste del diseño. Esto era más una preocupación de los socios, que al fin y al cabo eran los que iban a arriesgar el capital. La solución fue el de publicar el esquema del dispositivo, pero sin dar detalles de los componentes que llevaría el mismo. Esto haría que el copiarlo fuera casi tan costoso como hacer el diseño desde cero.

- Componentes

Como se ha comentado, la idea de esta tarjeta empezó con una siendo una manera barata de aprender ciencias de la computación, y para ello querían que esta tarjeta fuera el núcleo de un pequeño ordenador, al que se le pudiera añadir una conexión a la red, un teclado, un ratón y un monitor. Los componentes de esta tarjeta son:

- Procesador:

Aunque los primeros prototipos incorporaban un chip de Atmel, el ATMega644, en las versiones finales se incorporó un chip con una arquitectura ARM de la empresa Broadcom. El primer modelo disponía de un procesador ARM11 a 700 MHz, en la revisión 2 el procesador paso a ser de cuatro núcleos y a 900 MHz, un ARM Cortex A7. La última revisión dispone de un procesador ARM Cortex A53 (ARMv8) de 64 Bits a 1,2 GHz.

- Memoria:

La Memoria de la tarjeta esta compartida con la tarjeta de vídeo. En los primeros modelos disponía de 512 Mb o 256 Mb, pero en los dos últimos modelos la memoria ha subido hasta 1 Gb.

- Conectividad:

La Raspberry PI 1 y 2 disponen de un modelo con RJ45 para poder conectarse a una Red LAN, cualquier conectividad extra se podía implementar conectando un adaptador en alguno de los puertos USB. En la última revisión se ha añadido, integrado en la tarjeta, un adaptador WI-FI 802.11 y BlueTooth v4.1.

- Entradas y salidas

En la primera revisión había un conector HDMI para conectar un monitor, una salida de audio, dos puertos USB, un RJ-45, un conector microUSB de alimentación, y un puerto de expansión con acceso a 8 pines I/O del micro, el bus I^2C, SPI, etc. En la revisión 2 y 3 este puerto ha crecido y existe la posibilidad de acceder hasta 17 pines de I/O, y además hay el doble de puertos USB, 4.

Figura 25.- Raspberry Pi 3

- Almacenamiento

Al principio se utilizaba una tarjeta SD, que se utilizaba como si fuera un disco duro de un PC. En las revisiones 2 y 3 se ha reducido el tamaño y ha pasado a ser una tarjeta microSD.

- Alimentación:

Se puede disponer de la alimentación del equipo a través del micro USB de carga o a través de los pines de alimentación del puerto de expansión.

  - Modelos

El primer modelo puesto a la venta fue la Raspberry Pi Modelo B, pero debido al éxito que tuvo, se decidió construir un modelo con las características recortadas y más pequeño, al que se llamó modelo A. Este modelo tenía menos RAM, no tenía Conector RJ45 para conectarse a la red, y solo tiene un puerto USB.

Figura 26.- Raspberry Pi modelo A

En la actualidad, existen tres revisiones del modelo original, con nuevas características, y además, existe un nuevo modelo de bajo coste, Raspberry pi Zero, que tiene las características de hardware del primer modelo, aunque ligeramente mejoradas, pero con muy pocos conectores para poder hacer una tarjeta de un tamaño muy reducido (65mm x 35mm x 5mm) y a un precio de compra muy bajo, 5 $.

Figura 27.- Raspberry Pi Zero

Una comparativa de modelos se puede ver en la siguiente tabla.

	Raspberry Pi 1 Modelo A	**Raspberry Pi 1 Modelo B**	**Raspberry Pi 1 Modelo B+**	**Raspberry Pi 2 Modelo B**	**Raspberry Pi 3 Modelo B**
SoC:	Broadcom BCM2835 (CPU + GPU + DSP + SDRAM + puerto USB)			Broadcom BCM2836 (CPU + GPU + DSP + SDRAM + Puerto USB)	Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB
CPU:	ARM 1176JZF-S a 700 MHz (familia ARM11)			900 MHz quad-core ARM Cortex A7	1.2GHz 64-bit quad-core ARMv8
Juego de instrucciones:	RISC de 32 bits				
GPU:	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC				
Memoria (SDRAM):	256 MB(compartidos con la GPU)	512 MB (compartidos con la GPU) desde el 15 de octubre de 2012		1 GB (compartidos con la GPU)	
Puertos USB 2.0:	1	2 (vía hub USB integrado)	4		
Entradas de vídeo:	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF				
Salidas de vídeo:	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD				
Salidas de audio:	Conector de 3.5 mm, HDMI				
Almacenamiento integrado:	SD / MMC / ranura para SDIO		MicroSD		
Conectividad de red:	Ninguna	10/100 Ethernet (RJ-45) via hub USB			10/100 Ethernet (RJ-45) vía hub USB, Wifi 802.11n, Bluetooth 4.1
Periféricos de bajo nivel:	8 x GPIO, SPI, I²C, UART			17 x GPIO y un bus HAT ID	
Reloj en tiempo real:	Ninguno				
Consumo energético:	500 mA, (2.5 W)	700 mA, (3.5 W)	600 mA, (3.0 W)	800 mA, (4.0 W)	
Fuente de alimentación:	5 V vía Micro USB o GPIO header				
Dimensiones:	85.60mm × 53.98mm (3.370 × 2.125 inch)				
Sistemas operativos soportados:	GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux. RISC OS				

Tabla 2.- Comparativa modelos Raspberry Pi

- Sistema operativo:

A diferencia del Arduino, la Raspberry Pi necesita un sistema operativo para trabajar. Desde el principio ha tenido mucho apoyo por parte de la comunidad, y se ha desarrollado un software que es *Open Source*, y por tanto, de libre distribución.

Aunque la gran mayoría de sistemas operativos compatibles con la Raspberry están basados en Linux, en la última versión de la tarjeta se puede instalar una versión adaptada de Windows 10, llamada Windows 10 IOT Core. La gran mayoría de empresas ven esta herramienta como una posibilidad de desarrollo de aplicaciones para IoT (Internet of Things).

La distribución oficial es *Raspbian*, un sistema operativo Linux basado en la distribución *Debian*, pero existen muchos más, como por ejemplo una distribución basada en Ubuntu, otra basada en Fedora, Kali Linux, etc. Hay distribuciones que se basan en Unix, como FreeBSD. También existen distribuciones basadas en Android y el sistema operativo de Google, Chromium OS.

La potencia de la tarjeta también ha abierto nuevas aplicaciones para la misma, como centros multimedia, donde se pueden conectar al televisor y convertirlo en una *Smart TV.*

- **OPENCV [15]:**

OpenCV (Open-Source Computer Vision, opencv.org) es una librería para trabajar con algoritmos de visión artificial. Dispone de un amplio rango de módulos para solucionar problemas en el ámbito de la visión artificial. Posiblemente el punto más útil de esta librería es la arquitectura y el manejo de la memoria. Proporciona un *framework* en el que se puede trabajar con imágenes y video sin tener que preocuparte de la asignación y deasignación de la memoria.

- Historia de OpenCV [16] y [17]:

OpenCV se lanzó oficialmente como un proyecto de investigación dentro del *Intel Research to advance technologies in CPU-intensive aplications* en 1999. Los objetivos de este proyecto fueron:

- Avanzar en la investigación de la visión artificial proporcionando un código optimizado y de código abierto.
- Difundir el conocimiento de visión artificial al proporcionar una infraestructura común, de manera que el código desarrollado sea más legible y transferible.
- Avanzar en el desarrollo de aplicaciones comerciales haciendo un código portable, optimizable y libre, de manera que cualquiera pueda avanzar y desarrollar nuevas aplicaciones.

La primera versión alpha del OpenCV se lanzó al público en la *IEEE Confenference on Computer Vision and Pattern Recognition* en el 2000.

Esta librería se ha utilizado en infinidad de aplicaciones: Sistemas de seguridad, detección de movimiento, reconocimiento de objetos, etc. Esto es gracias a que la librería se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación, siempre siguiendo las condiciones que esta licencia tiene.

Desde agosto del 2012, el soporte de OpenCV está controlado por una fundación sin ánimo de lucro OpenCV.org, que se encarga de mantener la web de desarrollo.

- Características de OpenCV

OpenCV es una librería multiplataforma, así que se pueden encontrar versiones para GNU/Linux, Android, Mac OS y Windows. Contiene más de 2500 algoritmos optimizados que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos, calibración de cámaras, machine learning, realidad aumentada, visión estéreo para múltiples cámaras, visión robótica, etc.

La comunidad de OpenCV es de más de 47.000 personas, y la cantidad de descargas realizadas se estima que superan los 7 millones.

La librería ha sido escrita en forma nativa en C++, y dispone de interfaces para C++, C, Python, Java y Matlab. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel (IPP).

La lista de compañías que utilizan estas librerías es muy extensa e incluyen a Google, Yahoo, Microsoft, Intel, IBM, Honda, Toyota, etc.

En octubre del 2016, la última versión oficial de la librería es la 3.1, aunque debido a la gran difusión de la anterior versión, la 2.4, se ha lanzado una revisión 2.4.13 que soluciona problemas en las anteriores versiones.

3.2 Fases de desarrollo del proyecto: Fase 1 – CCD Lineal

Cuando se comenzó el proyecto, la idea era implementar un seguidor de línea con un sensor CCD lineal.

- Sensor CCD:

El sensor que se había pensado utilizar era el ***TCD132D*** de Toshiba, un sensor CCD en B/N.

Figura 28.- Toshiba TCD132D

Este sensor es un sensor de imagen CCD de 1024 elementos que incluye la electrónica de análisis de señal.

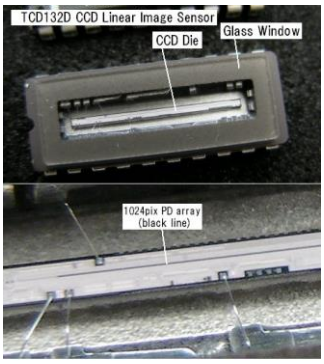

Figura 29.- Detalla del sensor

Como se puede ver en la imagen, la parte del CCD dispone de un cristal protector que deja pasar la luz protegiendo el interior del circuito.

El sensor dispone de un bus de datos con tres entradas de frecuencia, y una de salida de datos. El esquema del CCD se puede ver en el diagrama del datasheet:

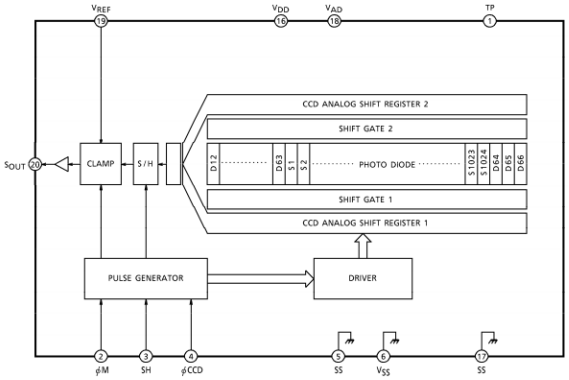

Figura 30.- Diagrama de bloques interno (Fuente Datasheet componente)

Para funciona, al CCD se le deben introducir dos señales de reloj: *CCD clock* (ΦCCD) y *Master Clock* (ΦM). Estas dos señales deben estar sincronizadas, y además la frecuencia de ΦCCD debe ser 4 veces menor que la ΦM. La siguiente entrada es la de *Shift pulse*, que tendrá la misma frecuencia que la señal de ΦCCD. Por último, los datos saldrán por el pin S_{OUT}, con una frecuencia de la mitad de la frecuencia ΦM.

Para alimentar el CCD se utilizan dos fuentes de tensión, una para los valores digitales, que es la tensión de referencia V_{REF}, y va de 4,5 V a 5,5 V, y una tensión para la parte analógica (V_{AD}) y otra para la parte del driver (V_{DD}), que va de los 11 a los 13 Volts.

El sistema necesita que siempre reciba ciclos de datos para que vaya capturando imágenes. Un ciclo de datos se produce cuando se envía una señal de SH, y en este momento el sensor comienza a extraer los datos que tiene almacenados. Primero extrae 64 valores *dummy*, y a partir del bit 65 comienza a extraer los 1024 datos que había en cada CCD del sensor. Por último, y para cerrar el ciclo, se envían 3 datos *dummy* más, momento en el que se espera un nuevo ciclo de datos. El funcionamiento se puede ver claramente en el siguiente cronograma:

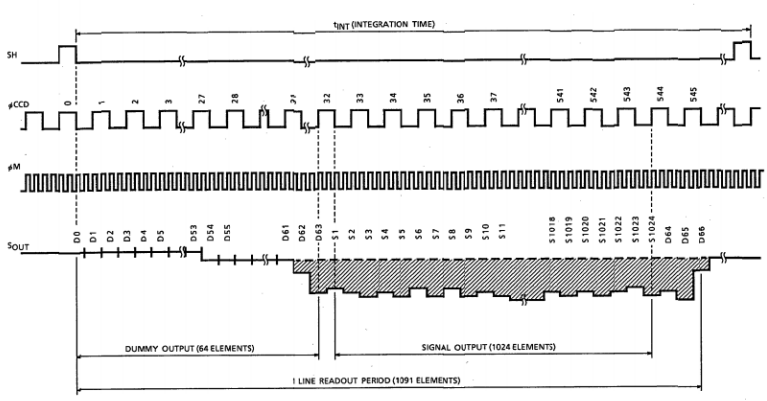

Figura 31.- Cronograma de envío de datos (Fuente Datasheet componente)

En el cronograma se puede ver bien que lo datos extraídos son analógicos, es decir, es el valor en gris captado por el sensor.

- Controlador:

Para poder configurar el sensor CCD y leer los datos se necesitaban un procesador capaz de generar señales síncronas de las frecuencias que aceptaba el sensor (de 0,4 MHz a 4 MHz en la entrada de ΦM), y además poder realizar lecturas analógicas a una velocidad lo suficientemente rápida para detectar los cambios suministra el CCD. Por esas características, y por incorporar un módulo DSP que se podría necesitar en alguna de las fases del proyecto, se escogió el ***dsPIC33FJ128*** [18], en su formato DIP-28.

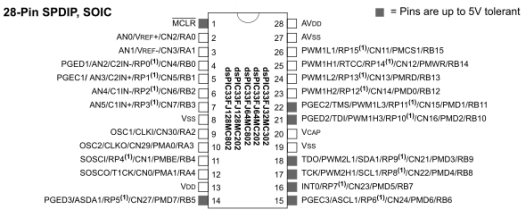

Figura 32.- Encapsulado PIC (Fuente Datasheet componente)

Al escoger este PIC se añadía una variable más, ya que este dispositivo, aunque alguna de sus entradas toleran los 5 V, funciona con una tensión de + 3,3 V.

Las características más importantes de este PIC son:

- Hasta 40 MIPS.
- Bus de datos de 16 bits, y de direcciones hasta 24 bits.
- Listado de instrucciones optimizadas para compilar en C.
- DMA interno.
- 128 Kb de memoria flash interna
- ADC de 10 bits con 1Msps o 12 bits con 500 Ksps.
- 6 canales de PWM.
- Regulador *on-chip* de 2,5 V.
- Buses de datos: I^2C, SPI, UART, CAN, etc.
- Formato SDIP

Lo más importante que necesitábamos era un ADC que tuviera una resolución aceptable y que fuera a una velocidad, como mínimo, de la frecuencia ΦM. Esto era debido a que la señal de

salida del CCD sería de como mínimo 200 KHz, y con un periodo de muestreo inferior a 400 KHz no se cumpliría Nyquist y no se podría reconstruir la señal.

El otro punto a tener en cuenta es el control de las frecuencias del CCD. Para ello se quería utilizar el PIC, pero con código puro no se conseguiría un algoritmo eficiente y síncrono. Este dispositivo dispone de unos módulos independientes, los PWM, que suministran, sin necesidad de control por parte del microprocesador, una frecuencia de salida programada, especialmente pensada para controlar motores de corriente continua. Tener 6 canales de PWM permitía pensar que la gestión de los motores se realizaría utilizando estas salidas.

Para programar el dsPIC se utiliza un programa de Microchip, el MPLAB, en su versión 8.66, ya que el programador que se utilizaba, el ICD2 no funcionaba en versiones posteriores. Para aprender y practicar se utilizó un libro de carácter práctico [19]

- Explicación esquema de interconexión

El conexionado entero de todo el circuito era el que se muestra en la **Figura 33**.

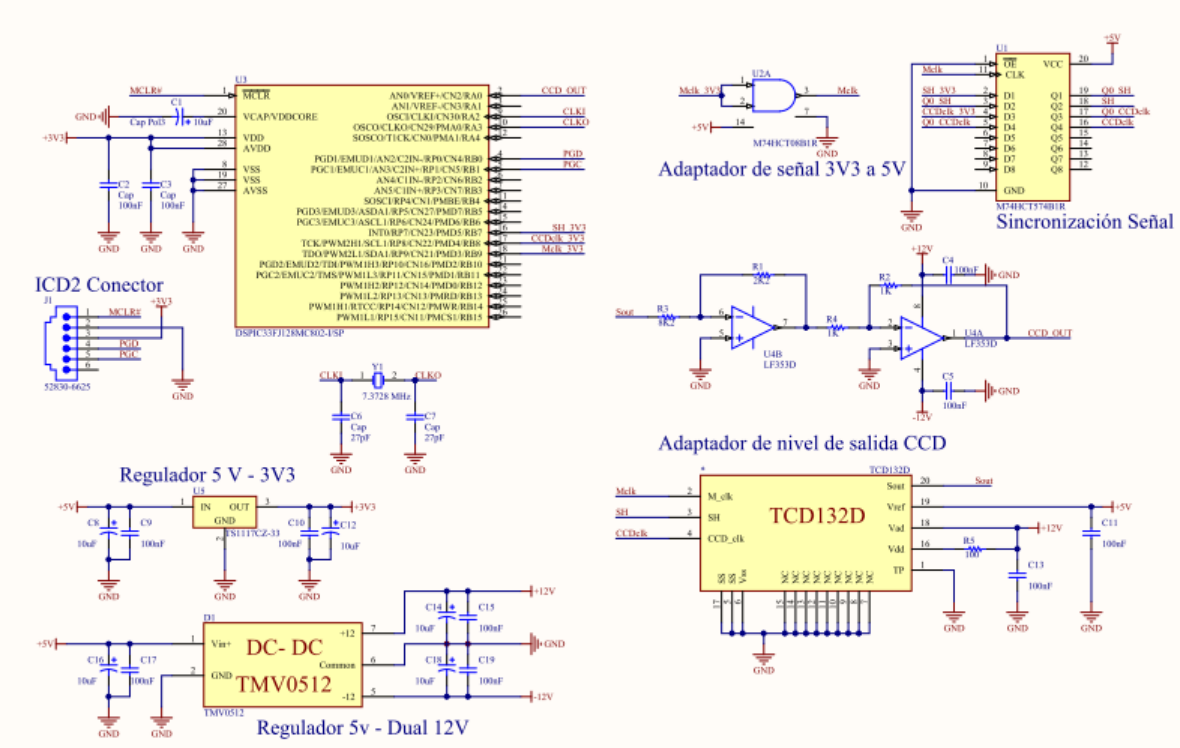

Figura 33.- Esquemático control CCD

Este primer esquemático se realizó utilizando una fuente de alimentación que proporcionaba 5V, y era para comenzar a testear el funcionamiento del CCD. Dado que la alimentación era de 5 V, se colocó un regulador lineal modelo ***TS1117CZ-33***, que proporciona una salida de 3,3 V. Se escogió este regulador porque tenía un rango de tensiones de entrada muy amplio, hasta 12 V en algunas versiones y eso nos permitiría poder cambiar la entrada de tensión sin problemas, una buena corriente de salida de hasta 800 mA, 1,3 voltios de Dropout, una muy buena regulación de línea y de carga, es decir, la tensión es muy estable independientemente de la carga que se coloque, y un encapsulado que permitía, si fuera necesario, incorporar un disipador. Además, este modelo es genérico, y aunque el que se ha utilizado es de la marca *TSC*, el equivalente de Texas Instruments ***LM1117*** sirve igual.

Con los 3,3V, se necesitaba también una tensión de alimentación del CCD, ya que este dispositivo necesitaba una alimentación de 12 Voltios. Para conseguir esta tensión se ha utilizado un elevador de tensión aislado, ***TMV0512*** de Traco Power. Este dispositivo lo que hace es transformar la tensión de entrada de los 5 Vcc a una salida dual de ±12 V. Tiene una potencia de 1W, la conversión tiene una eficiencia típica del 72%, y es capaz de dar 40 mA por salida. La gran ventaja de estos circuitos es que toda la electrónica de potencia (transformadores para aislar la señal, inductores, etc…) está dentro del encapsulado, únicamente hay que colocar unos condensadores para obtener una tensión de salida estable. Lo malo es que la eficiencia es bastante baja, lo que quiere decir que hay muchas pérdidas, y eso para un sistema con batería puede ser crítico. Otro punto negativo es que al ser aislado, su precio es muy elevado.

Como el Pic funciona a 3,3 Voltios, y el CCD funciona a 5 Voltios, hay que adaptar las señales entre los dos dispositivos. Para hacer un cambio de tensión de una señal digital lo más fácil es utilizar una puerta lógica que funcione a la tensión de salida deseada y asegurarse que la puerta lógica sea capaz de distinguir los valores de entrada como ceros y unos. Para realizar esta conversión se utiliza una puerta lógica AND, que tiene la peculiaridad de comportarse como un buffer de señal cuando se le conecta la señal a adaptar en las dos entradas. El modelo de puerta lógica es de la familia TTL 74, y el modelo que se escogió fue uno capaz de detectar los cambios a la entrada a la frecuencia de funcionamiento del reloj maestro. El modelo es la ***74HCT08*** en formato DIP-14, capaz de trabajar con señales de cientos de Mhz.

Para que el CCD proporcione los datos correctamente, las entradas de reloj y de SH deben estar sincronizadas. El mejor sistema para sincronizar dos señales es utilizar dos básculas D en serie que, aunque añaden un retardo de propagación inherente, proporcionan una señal sincronizada fiable:

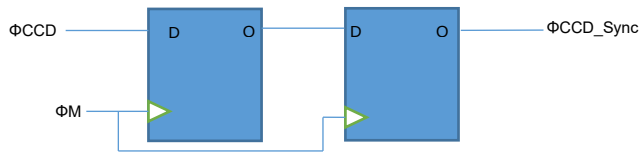

Figura 34.- Sincronismo con Básculas D

Para sincronizarlas, y a la vez adaptar las señales de 3,3 Voltios a 5 Voltios, se utiliza un dispositivo lógico de la misma manera que en la puerta lógica AND, un ***74HCT574*** en encapsulado DIP-20, que incluye 8 básculas lógicas del tipo D. De estas 8 utilizaremos únicamente 4, dos para sincronizar el SH y dos para el ΦCCD.

Una vez adaptadas las señales digitales de entrada al CCD, hay que adaptar la señal de salida S_{OUT}. Esta salida tendrá un valor entre 5 Voltios y 3,3 V, y como el ADC del PIC tiene un fondo de escala de como máximo la tensión de alimentación, se debe reducir. Como esta señal es analógica, la manera que se tiene que realizar la reducción de tensión es también analógica, y por tanto, se utiliza un Amplificador operacional, en su configuración inversora (ver figura de abajo).

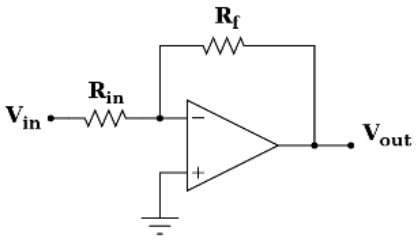

Figura 35.- Amplificador Inversor

Se utiliza esta configuración porque es capaz de atenuar la señal. La función de transferencia de un amplificador diferencial es $Vout = -(Rf/Rin) \cdot Vin$. Calculando la ganancia con los valores del esquema queda $G = Vout/Vin = -(2K2/8K2) = 0.268$, así que la tensión de salida será aproximadamente 4 veces la tensión de entrada. Se configuraron estos valores de partida, para evitar que se saturase la entrada del ADC, para posteriormente ir modificándolos y mejorándolos durante el desarrollo.

El amplificador operacional escogido es el ***LF353D*** de ST, aunque el de Texas Instruments es equivalente. Un ancho de banda de 3 MHz, un slew rate muy rápido de 13 V/µs, muy fácil de conseguir en tiendas y que en el encapsulado hay dos Operacionales hicieron que esta fuera la elección lógica. El único posible problema era que para funcionar correctamente necesita una fuente de alimentación dual de ±3.5 a ±18 Voltios, pero como en el circuito se había montado el DC-DC que proporcionaba los -12 V, la elección estaba clara.

El PIC utilizaba un reloj externo de 7.3728 Mhz, que se ha escogido para poder tener una frecuencia de 400.000 KHz a la salida del PWM, y se programaba a través de un conector RJ11, J1.

- Pruebas de software

El algoritmo de trabajo del PIC, en grandes bloques, era el de la **Figura 36**.

En el primer paso se configuraba el ADC para que leyese a la misma velocidad que el *master clock*, ya que los datos llegarían el doble de lentos que la frecuencia del reloj principal del ADC, se obtendrían 2 muestras por ciclo, suficiente para cumplir el criterio de Nyquist. Había que tener en cuenta que se almacenarían las 1024 muestras de datos, las 64 dummy del principio, y 3 extra de final, y como de cada bit se tenían 2 muestras, serían 2182 muestras de 10bits. En un dispositivo como un PIC tanta memoria era importante, y se tuvo especial cuidado en que estas muestras no saturaran el programa. La entrada del ADC está configurada para que funcione con el pin 2.

El PWM también se tuvo que configurar para que proporcionara las dos frecuencias de reloj necesarias, con unas frecuencias de 400 KHz y 100 KHz. Se utilizan los Pines 18 y 17 del chip para estas salidas.

Una vez configurados los dispositivos del sistema, se debe iniciar un ciclo de lectura. Lo primero es activar la señal SH tal y como aparece en la **Figura 31**. A la vez que se comienza a muestrear la señal que proviene del CCD.

Una vez leídas todas las muestras, se deben analizar, para verificar donde está detectando la línea. Con estas muestras, se debería tomar la decisión de hacia dónde mover el robot y ejecutar la rutina de control de motores.

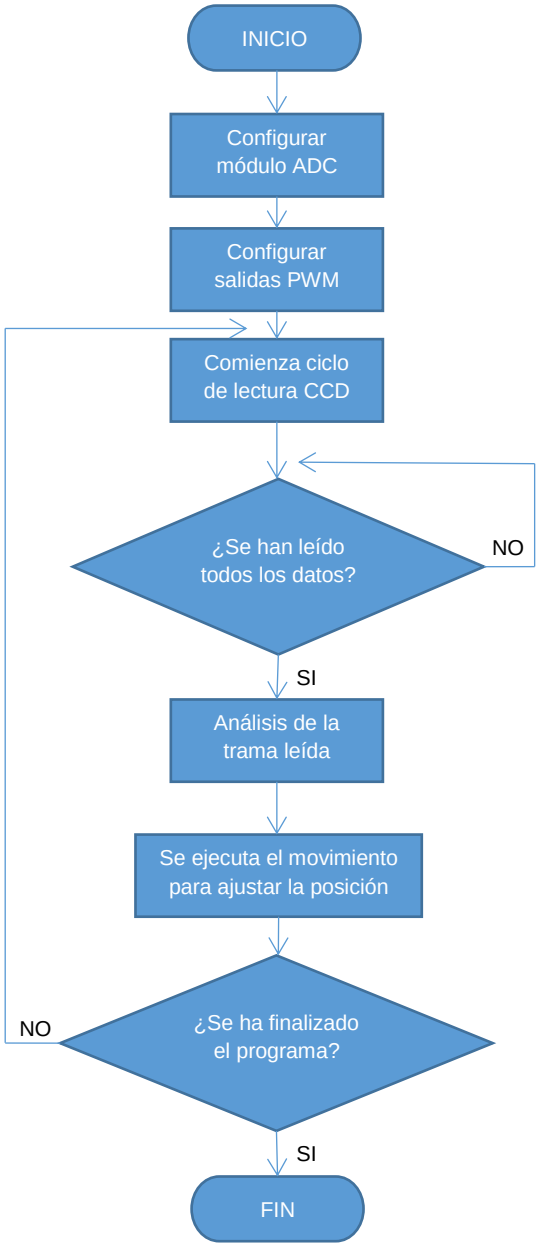

Figura 36.- Diagrama Control PIC

Con todos los pasos completados, si no se finaliza el programa, se continúa leyendo y moviendo el robot tal y como se ha explicado.

- Resultado de las pruebas

Toda la configuración del PWM, ADC y el resto de dispositivos llevo un tiempo, pero se consiguió que funcionara correctamente. Con la ayuda de un osciloscopio se comprobó que la señal digitalizada desde el ADC era muy parecida a la que suministraba el CCD, que era de esta forma:

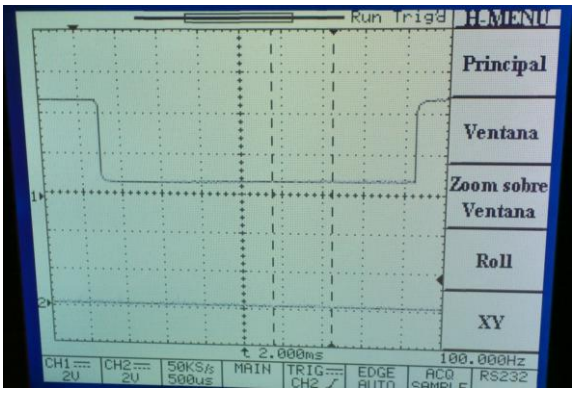

Figura 37.- Captura salida CCD con luz ambiental

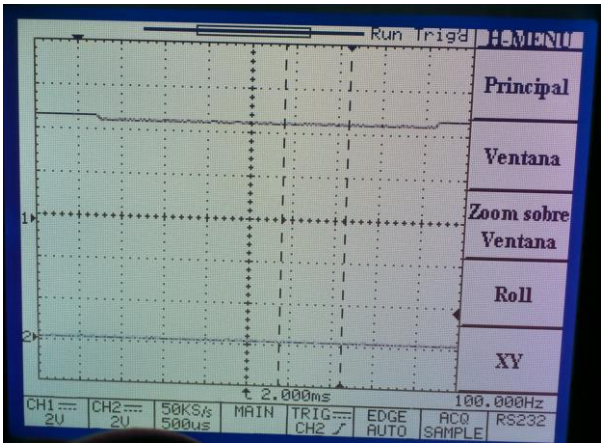

Figura 38.- Captura salida CCD sin luz

Viendo las capturas se interpreta que cuando el CCD este a oscuras, o lo que detecte sea un color oscuro, la salida será muy parecida al valor máximo, en cambio, si el CCD recibe luz, la salida será aproximadamente 0. Hay que comentar que la luz que se utilizó era una luz ambiental que iluminaba la zona donde estaba el sensor, la fuente de luz no estaba muy cerca de donde estaba el sensor. Este era el rango que pensábamos utilizar.

Comenzamos las pruebas, y vimos que colocando un objeto tapando la imagen, el sensor detectaba correctamente una ausencia de luz, y nos proporcionaba una señal de este estilo:

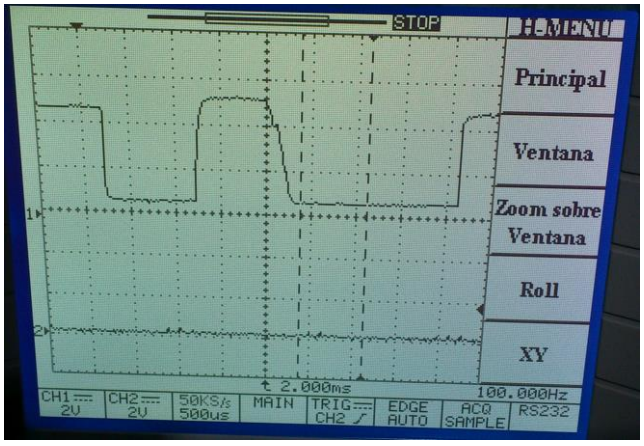

Figura 39.- CCD con zona tapada

Además de obtener estos resultados, verificado los datos recibidos desde el ADC, se observaba que eran idénticos, y por tanto, el proyecto parecía estar bien encarado.

El problema apareció cuando se quiso detectar una línea, ya que pusiéramos lo que pusiéramos delante del Sensor, no parecía que detectase nada, detectaba muy bien la ausencia y presencia de luz, pero no las tonalidades de gris que debería estar detectando. Parecía que la cámara no recibía suficiente luz, así que se comenzó a implementar un sistema con LED's cerca del sensor para iluminar la zona y recibir correctamente lo que había delante. Pero aunque parecía que había suficiente luz, el sensor seguía funcionando de forma Binaria, o detectaba luz, o no detectaba.

En una de las pruebas, con un flexo con una bombilla halógena de 5700k de brillo, una lámpara a la que se le denomina de luz solar, encendido a unos 60 cm del sensor, se observó que la salida del sensor proporcionaba esta salida:

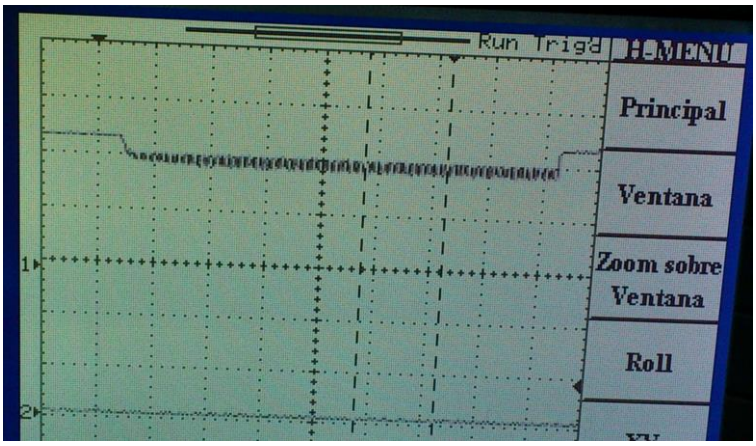

Figura 40.- CCD con luz de flexo a 60 Cm

Este comportamiento era muy extraño, ya que normalmente no debería funcionar de esta manera, parecía, según lo que se había extraído hasta ahora, que el sensor se había saturado.

Observando atentamente el datasheet del componente, se comenzó a observar la gráfica de sensibilidad a la respuesta espectral del circuito, y se detectó que el sensor era sensible a las longitudes de onda que recibe el ojo humano, pero además también era sensible a parte del espectro de infrarrojo.

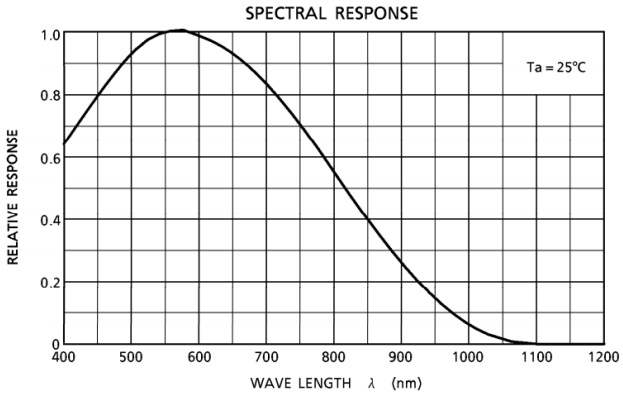

Figura 41.- Respuesta espectral del CCD

El rango de espectro que detecta el ojo humano va desde aproximadamente los 400 nm hasta los 750 nm, y a partir de allí entra la componente de Infrarrojo [20].

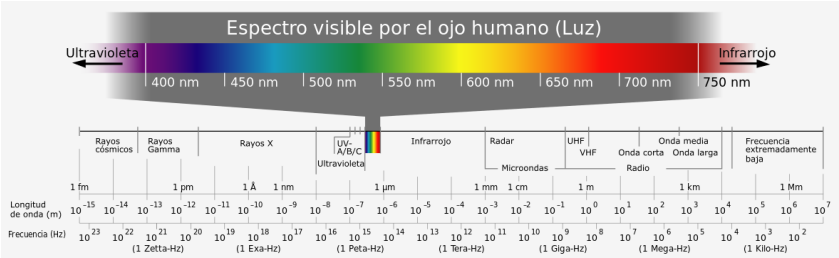

Figura 42.- Espectro visible por el ojo humano (Fuente Wikipedia)

Si se comparan las dos gráficas, se puede ver que hay una respuesta muy importante de las bandas de infrarrojos, lo que hace que la señal recibida por la cámara no sea realmente lo que el ojo humano percibe. Buscando soluciones para este problema se encontró una web que había desarrollado un proyecto con este sensor, utilizándolo como una cámara lineal [21]. Este problema lo solucionaba añadiendo un filtro de cristal que bloquease la banda Infrarrojo de la señal, llamados IRCF (Infra Red Crystal Filter):

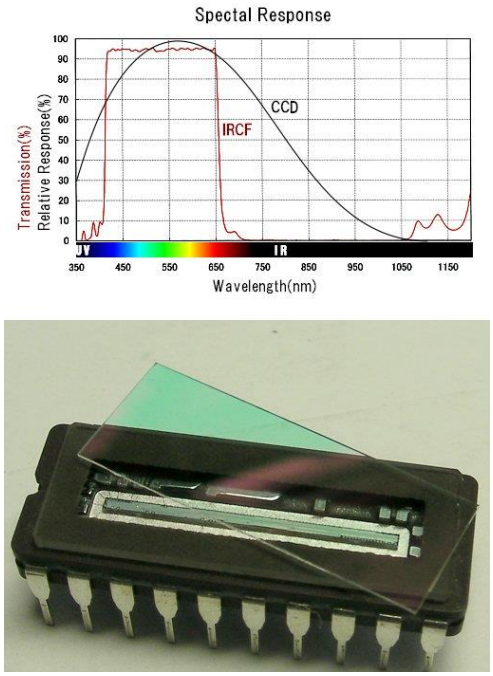

Figura 43.- Filtro IRCF

Este filtro, como se puede ver en la anterior figura, al aplicarse deja pasar únicamente la parte del espectro visible, y bloquea la componente espectral que está en el Ultravioleta y en el Infrarrojo.

La complejidad a la hora de encontrar sitios para comprar este IRCF, y que no se sabía a ciencia cierta si era esta la solución o no, nos hizo decantarnos por enfocar el proyecto desde otro prisma, y de esta manera añadir más funcionalidades al proyecto.

3.3 Fases de desarrollo del proyecto: Fase 2 – Raspberry pi

Buscando soluciones para el problema del CCD, se encontró un nuevo dispositivo que nos proporcionaba una manera de empezar el proyecto de nuevo de otra manera diferente, intentando llegar de esta manera hasta el objetivo final.

- Cámara Raspberry Pi [22]:

Hasta el momento todos los proyectos que se realizaban con cámara con la Raspberry Pi utilizaban una cámara USB externa tipo webcam, y utilizando los drivers de Linux se podían realizar capturas y videos. Pero en mayo del 2013 (aproximadamente un año después del lanzamiento de la primera Raspberry), se puso a la venta la cámara oficial de la Raspberry, que utilizaba uno de los conectores de cable plano flexible de la tarjeta y que hasta ahora no se utilizaban para nada.

Figura 44.- Cámara Raspberry pi conectada

La cámara dispone de un sensor de OmniVision OV5647 de 5 Megapíxeles. El sensor tiene una resolución de 2592 x 1944 pixeles, y es capaz de grabar videos de hasta 1080p30.

Como la cámara se conecta a un puerto de la tarjeta, para activarla basta con activar la opción del configurador de la Raspberry, reiniciar y ya está operativa.

- Poniendo en Marcha el sistema:

Una vez escogido que se iba a trabajar con una Raspberry, se comenzó a preparar el sistema. Los pasos a seguir son los siguientes:

1. Bajar la última versión de Raspbian, la distribución de Linux oficial de la Raspberry, desde su página oficial.
2. Descargar el programa *win23diskimager*, que se utilizará para descomprimir la imagen descargada a una SD, preparándola para poder arrancar el sistema operativo.
3. Conectar la tarjeta en la Raspberry, y arrancar la tarjeta.
4. Si no se dispone de un monitor y teclado al que conectar la tarjeta, conectar la tarjeta a la Red, obtener su dirección IP y conectarse por SSH, utilizando un programa como *putty*.
5. Para empezar, el usuario es *pi*, la contraseña *raspberry*.

6. Configurar la Raspbery introduciendo el comando: *sudo raspi-config*.
7. Una vez configurada la Cámara y todo lo necesario, reiniciar el sistema.
8. Actualizar el firmware y software de la tarjeta a la última versión y volver a reiniciar.

Siguiendo estos pasos la Raspberry estaría actualizada a la última versión y lista para trabajar. En la bibliografía se encontrarán enlaces para configurar la Raspberry Pi tal y como se hizo en este proyecto [23].

- Preparando OpenCV y la cámara:

Para tratar las imágenes se utilizará la librería OpenCV. Como el sistema operativo de la tarjeta es Linux, hay que descargar la versión de Linux de la librería.

1. Descargar de la página web de opencv.org o hacer descargar del *github* del proyecto la última versión disponible, en este caso se ha utilizado la versión 3.1 de la librería.
2. Instalar todas las dependencias en el sistema.
3. Crear un nuevo directorio para todos los ficheros del OpenCV.
4. Realizar un CMake para configurar que módulos se quieren instalar y que otro nos.
5. Compilar el OpenCV para este entorno. Esta compilación puede llevar tiempo, para una Raspberry Pi 1 podía llegar a tardar 10 horas, para una revisión 3, que dispone de 4 cores, se puede reducir hasta la hora y media.
6. Instalar todo lo compilado en el sistema.

Aunque el OpenCV normalmente trabaja con cámaras de manera bastante sencilla y transparente, en este caso la cámara del sistema no es así, y es necesario instalar unos drivers para acceder a la cámara desde un programa en C++.

Los drivers los ha realizado un grupo de investigación de la universidad de Córdoba llamado **Aplicaciones para la visión artificial** (AVA) [24], y permiten tener un acceso rápido y sencillo a la cámara con o sin el OpenCV. La librería se llama ***Raspicam***.

Con la instalación de estos drivers, el sistema está configurado para trabajar. Los pasos para instalar estos drivers se pueden ver en el punto [25] de la bibliografía.

Para practicar y documentarme con distintas funciones de OpenCV se han utilizado dos libros, con un aspecto más práctico que teórico [26] y [27].

- Entorno de trabajo:

Existen distintas configuraciones para poder escribir y compilar código en la Raspberry, ya que utiliza un sistema operativo Linux, en este caso se ha escogido la siguiente configuración para trabajar desde un ordenador con Windows 10, donde todos los programas son gratuitos o tienen un modo de trabajo gratuito:

- **Editor de texto:** Para trabajar con el código, se ha decidido que lo más fácil es escribirlo en el ordenador, y para ello se han utilizado dos programas: Notepad++ y Sublime Text. La primera opción es gratuita y existen muchos módulos instalables que hacen que puedas crear un entorno de

programación muy fácilmente, por ejemplo, se puede instalar un terminal SSH, un FTP, etc. La segunda opción es un programa de pago, pero dispone de una versión gratuita con limitaciones. Se ha escogido la versión gratuita porque es muy cómodo para escribir código, ya que gráficamente es muy agradable.

- **Servidor FTP:** Una vez hecho el código, el siguiente paso es enviarlo a la Raspberry, y para ello se necesita un servidor FTP. Para realizar esta tarea se ha escogido el WinSCP, que es un programa gratuito pero muy fácil de manejar. En alguna ocasión se ha utilizado el FTP del Notepad++, pero la gran mayoría de veces se ha utilizado este programa.

- **Terminal SSH:** Para conectarse con la tarjeta se puede utilizar el programa que incluye el mismo Windows de escritorio remoto, pero este tipo de conexión resulta algo lenta e incómoda. Como la gran mayoría de instrucciones se escriben utilizando el terminal, se ha decidido utilizar una conexión SSH a través del programa Putty, que cumple con todas las necesidades que se tienen.

- **Compilación de código en la Raspberry:** Con el código enviado a un directorio de la tarjeta, se debe compilar para obtener el ejecutable. Aunque siempre se puede utilizar la instrucción que llama directamente al compilador de C++, para hacerlo más sencillo, se ha utilizado el CMAKE, que con un script, es capaz de buscar las dependencias necesarias para el proyecto, y generar el ejecutable.

Con estas cuatro aplicaciones es suficiente para crear, enviar, compilar y ejecutar todo el código que se va a realizar para el proyecto. A veces se ha utilizado el escritorio remoto para entrar a la Raspberry y observar algún proceso, pero rara vez.

- Detectando la línea:

Una vez todo el entorno de trabajo está preparado, se puede comenzar a crear y probar las funcionalidades del robot. La primera funcionalidad que se debe tener es la detección de la línea adquiriendo las imágenes a través de la cámara y analizándolas utilizando OpenCV. Hay que implementar un código que sea capaz de detectar dónde está la línea, tamaño y donde está su centro. El código debe ejecutar el diagrama de flujo de la figura adjunta.

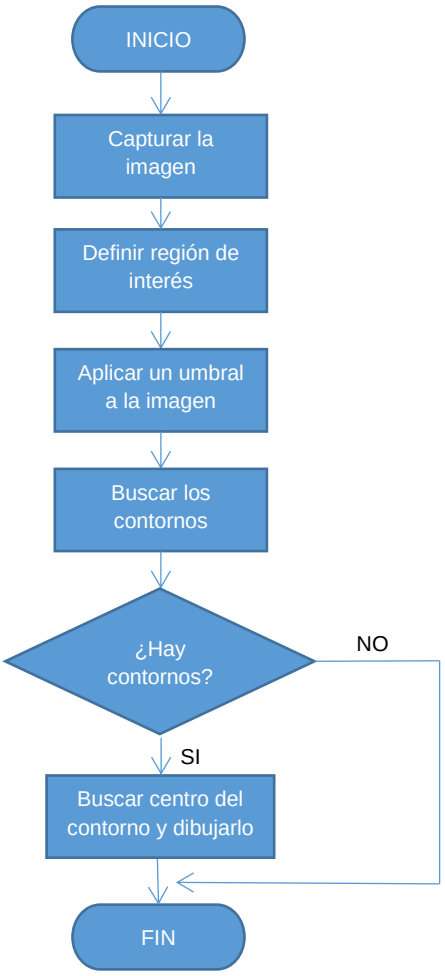

Figura 45.- Diagrama de bloques de búsqueda de contornos

Los pasos a realizar son los siguientes:

- **Captura de la imagen:**

OpenCV dispone de una función para capturar una imagen que proviene de una cámara conectada al sistema. En el caso de Raspberry, y como ya se ha comentado, no se puede aplicar directamente, y por tanto se utilizará la librería ***Raspicam*** [24]. Para utilizar esta librería, además de incluirla en el fichero, se debe crear una instancia a un objeto que será el que manejará todos los datos que provienen de la cámara:

```
#include <raspicam/raspicam_cv.h>
…
// Variable para acceder a la cámara
raspicam::RaspiCam_Cv Camera;
cv::Mat image;                          // Variable para almacenar los datos de la cámara

// Configuración de la cámara
Camera.set( CV_CAP_PROP_FORMAT, CV_8UC3 ); // En color
Camera.set(CV_CAP_PROP_FRAME_WIDTH, 640); // Las imágenes serán de 640 pixeles en X
Camera.set(CV_CAP_PROP_FRAME_HEIGHT, 480); // y 480 pixeles en y

// Se verifica que la cámara se ha abierto
if(!Camera.open())
{
   std::cerr << "Error opening the camera" << std::endl;
   return -1;
```

```
}
...
// Adquiero los datos de la cámara
Camera.grab();
Camera.retrieve(image);
```

Antes de empezar a explicar el código, toda la documentación de las diferentes funciones del sistema se puede consultar online en la documentación oficial de OpenCV [28].

Las dos primeras instrucciones crear el objeto para acceder a la cámara y una variable de OpenCV llamada ***Mat***, que es un vector donde se guardan todos los datos de una imagen para poder trabajar desde ellos con las funciones de OpenCV. Puede almacenar cualquier tipo de imagen, desde en blanco y negro, hasta en color.

Las tres siguientes instrucciones configuran el funcionamiento de la cámara: De que tipo tiene que ser las capturas, en este caso de 8 bits con 3 canales, es decir, en color, y de un tamaño de 640 x 480 pixeles.

La función ***open()*** verifica que la se ha podido acceder a la cámara, si no hubiera cámara, el programa fallaría, y por tanto, se tiene que salir antes de que se provoque cualquier problema.

Por último, para adquirir las imágenes desde la cámara, se utilizan dos funciones, primero ***grap()***, que recoge la información de la cámara, y después ***retrieve****(cv::Mat image)*, que transforma lo recibido por la cámara a una variable tipo Mat.

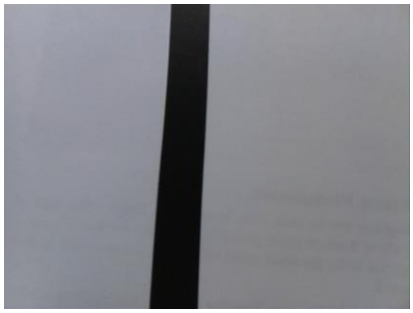

Figura 46.- Captura obtenida desde la cámara

- **Región de interés:**

Con la imagen capturada, se tiene que escoger una región de interés (ROI) donde se buscará la línea y su centro. Se define un área de interés porque así se puede acotar mejor el área donde está el punto de detección del robot. Para esta primera aproximación se utiliza un área de interés centrada (ver figura adjunta).

Figura 47.- Captura de la ROI

Esta ROI afectará sobre todo a la velocidad del robot, contra más arriba esté la zona de interés, antes reaccionará, y contra más abajo, más lento será. Obviamente, estos ajustes se irán verificando en pasos posteriores.

Para obtener esta región de intereses, hay que tratar esta imagen para obtener un trozo y poder utilizarlo como el ROI:

```
cv::Rect roiTemp;
roiTemp.width = 640; // Anchura de la imagen
roiTemp.height = 100; // Altura de la imagen
roiTemp.x = 0; // Posición en X del punto inferior izquierdo de la imagen
roiTemp.y = 190; // Posición en Y del punto inferior izquierdo de la imagen

// Se crea una imagen temporal recortada a partir de la imagen original
cv::Mat tempCrop(image, roiTemp);
// Se paso a blanco y negro
cv::cvtColor(tempCrop, tempCrop, CV_RGB2GRAY);
```

Lo primero que se debe hacer es definir un rectángulo con el área que se quiere recortar de la imagen original. Este rectángulo será de 640 x 100 pixeles, y se colocará en el centro de la imagen.

Con este rectángulo creado, ya se puede crear una nueva variable ***Mat*** que realizará un recorte de la imagen original al tamaño que se ha definido como rectángulo de trabajo.

Por último, y para poder realizar mejor los pasos posteriores, se utiliza la función ***cv::cvtColor (InputArray src, OutputArray dst, int code, int dstCn=0)***, que transforma las imágenes de un dominio de color a otro. En este caso se pasa del dominio de color RGB al dominio de la escala de grises.

- **Definir umbral de la imagen:**

A partir de esta imagen se debe obtener la posición de la línea, si es que la hay. Para ello hay que preparar esta imagen para que los contornos se encuentren de la manera más sencilla posible. El código que realiza esta función es el siguiente:

```
// Se aplica el filtro umbral
cv::threshold(tempCrop, tempCrop, 50, 255, 0);
// Se negativiza la imagen
cv::bitwise_not(tempCrop, tempCrop);
// Se realiza una filtrado de erosión y dilatación
// Se definen las estructuras de erosión y dilatación
cv::Mat erodeElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
cv::Mat dilateElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(8, 8));
// Filtro de erosión
cv::erode(tempCrop, tempCrop, erodeElement);
// Filtro de dilatación
cv::dilate(tempCrop, tempCrop, dilateElement);
```

La primera función que aplicamos en la imagen es la función umbral ***double threshold(InputArray src, OutputArray dst, double thresh, double maxval, int type).*** La función recibe una imagen a tratar (***src***), devuelve una imagen tratada (***dst***), recibe un valor umbral (***thresh***), el valor máximo que puede tener un pixel (***maxval***) y el tipo de filtrado (***type***).

En nuestro caso el filtro será binario, y tiene el comportamiento descrito por la ecuación de abajo.

$$dst(x,y) = \begin{cases} maxval, & src(x,y) > thresh \\ 0, & otherwise \end{cases}$$

Ecuación 1.- Formula filtro Threshold

Si el ***maxval*** está configurado a 255, si el valor es mayor al thresh, devolverá un 255, blanco, si no, devolverá un 0.

Figura 48.- Filtro Threshold

Como el resultado de la función está invertida, se realiza una operación de inversión de la imagen, con la función ***void bitwise_not(InputArray src, OutputArray dst, InputArray mask=noArray())***. Esta función devuelve el valor negado de la imagen que se pasa, por tanto, la línea seleccionada ahora será blanca, y la zona sin línea negra.

Figura 49.- Filtro Bitwise

Por último se aplica a la imagen un filtrado de erosión y dilatación, que lo que persigue es "suavizar" los contornos, para que la detección sea más sencilla. Para aplicar el filtro primero se tiene que crear unos filtros morfológicos para los dos filtros, utilizando la función ***Mat getStructuringElement(int shape, Size ksize, Point anchor=Point(-1,-1))***, que crea un filtro del tamaño definido por ***ksize*** y del tipo que se defina en ***shape***, en nuestro caso se utiliza un filtro de erosión de 3x3 rectangular, y un filtro de dilatación de 8x8 rectangular.

Para erosionar la imagen se utiliza la función ***void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue()),*** que lo que hace es pasar un filtro rectangular de 3x3 por todos los pixeles de la señal, donde el valor que se obtiene es el mínimo valor en el rectángulo que rodea al pixel. Esta función lo que hace es realzar los puntos negros en contra de los puntos blancos.

Para dilatar la imagen se utiliza la función ***void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue()),*** que hace justo lo contrario, es decir, el valor del pixel se sustituye con el valor máximo de los valores que rodean al pixel en un rectángulo de 8x8.

Se realiza de esta manera para maximizar el efecto de los blancos, primero se maximizan los negros, y al pasar el filtro de dilatación haces que la imagen quede lo más detallada posible, eliminando las pequeñas imperfecciones de la detección que puedan hacer que la imagen no quede lo suficientemente nítida.

Figura 50.- Imagen después de pasar el filtro

- **Encontrar contornos en la imagen:**

Con las anteriores operaciones se ha preparado la imagen para poder detectar los contornos con el menor error posible. Aclarar que contornos son áreas cerradas. El código para realizar la detección es el siguiente:

```
// Se definen las variables para guardar los contornos
std::vector<std::vector<cv::Point> > contours;
std::vector<cv::Vec4i> hierarchy;
// Se ejecuta la función para detectar contornos
cv::findContours(tempCrop, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0));
```

Lo primero que se hace es crear unas variables para guardar los valores de los contornos, el primero, ***contours***, devuelve el número de contornos, y todos los puntos x e y que componen cada contorno cerrado. La segunda variable, ***hierarchy***, que no se utilizará, nos proporciona información sobre la forma de la topología de la imagen.

La función para detectar contornos es ***void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())***, a partir de una imagen ***image***, devuelve todos los contornos que puede encontrar en la imagen (si los hubiera) ***contours***, los valores de topología de la imagen ***hierarchy***, siguiendo el modo de trabajo ***mode***, y el método de aproximación ***method***. En este caso se utiliza el ***mode*** CV_RETR_TREE, que establece todos los contornos y todas las relaciones de las topologías, y el ***method*** CV_CHAIN_APPROX_SIMPLE, que comprime los datos al máximo, minimizando el número de puntos que se obtienen, por ejemplo, si se detecta un rectángulo, este se almacenará con cuatro puntos, que son los que definen el espacio.

- **Búsqueda del centro del contorno y dibujado en la imagen original:**

La anterior función devuelve todos los contornos detectados en la imagen, y lo que ahora se debe hacer es dilucidar si estos contornos son correctos y corresponden a una línea, y dibujar un punto en el lugar que se encuentran de la imagen original.

```
// Se recorren todos los contornos encontrados
for (size_t i = 0; i < contours.size(); i++)
{
        float area = cv::contourArea(contours[i]);
        // Si el área es mayor de 2000, se acepta el contorno
        if (area > 2000)
        {
                // Se crea una variable para guardar los momentos
                cv::Moments mu;
                mu = moments(contours[i], false);
                // Se crean dos variables para guardar los datos del punto central
                xPoint[j] = mu.m10 / mu.m00;
                // Se suma el desfase en y de la imagen original.
                yPoint[j] = (mu.m01 / mu.m00) + roiTemp.y;
                cv::Point2f center(xPoint[j], yPoint[j]);
                // Dibujo un circulo basado en el centro
                cv::circle(image, center, 5, cv::Scalar(0, 255, 0), -1, 8, 0);
                // Dibujo el rectangulo
                cv::rectangle(image, cv::Point(xPoint[j] - 40, yPoint[j] - 20), cv::Point(xPoint[j] + 40, yPoint[j]
                + 20), cv::Scalar(0, 0, 255), 1, 8, 0)
        }
}
```

Se ejecuta un bucle que recorre todos los contornos encontrados. Para que el contorno sea correcto, el tamaño debe ser mayor de 2000 pixeles, un contorno menor significa que es ruido. El área total se devuelve en pixeles, y para calcularla se utiliza la función ***double contourArea(InputArray contour, bool oriented=false)***, que devuelve el valor en pixeles del área del contorno.

Si el contorno es mayor que 2000 pixeles, es correcto, y se deben calcular los momentos de este área, utilizado una variable del tipo ***cv::Moments***, y la función ***Moments moments(InputArray array, bool binaryImage=false)***.

Los momentos es un tipo de medida utilizada en mecánica, estadística y visión artificial, que describe la distribución espacial de un grupo de puntos. En visión artificial, el momento de orden cero m_{00}, es el área total de la imagen, en este caso el contorno. Los momentos de primer orden, m_{10} y m_{01}, definen el centro de gravedad de la imagen. Si se combinan los momentos de orden cero y primer orden, se obtiene:

$$\bar{X} = \frac{m_{10}}{m_{00}} \; ; \; \bar{Y} = \frac{m_{01}}{m_{00}}$$

Ecuación 2.- Coordinadas del centro del contorno

Se obtiene las coordenadas del centro de gravedad de la imagen, que en este caso serán las coordenadas del centro del contorno detectado. En este caso, como se quiere dibujar el centro del contorno en la imagen original, se tiene que sumar el desplazamiento de la imagen recortada respecto la imagen original.

Por último, y para visualizarlos en la imagen, se dibuja un circulo verde en el punto central de la imagen con ***void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0)***, y un rectángulo que rodee el punto central con ***void rectangle(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)***.

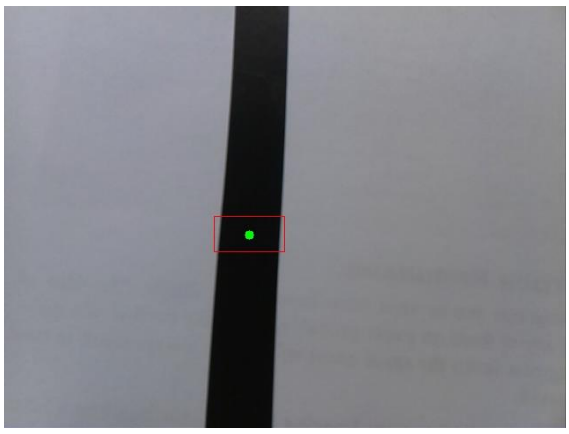

Figura 51.- Representación del punto central detectado

- **Mejora de la detección del punto central de la línea:**

Al detectar un solo punto, puede haber algún error, que deje de detectar, o que la línea este fuera del ROI. Para intentar minimizar este problema, cuando se detecta el punto central, se añaden 2 ROI más, desfasadas 50 pixeles entre ellas. De esta manera, cuando el primer punto no tenga información, se pasará a probar en los otros dos. Es una manera de intentar

evitar que se pierda la línea con los movimientos del robot, ya que a veces pueden ser bruscos y añadir algún que otro movimiento demasiado rápido, y por tanto, la línea se puede perder.

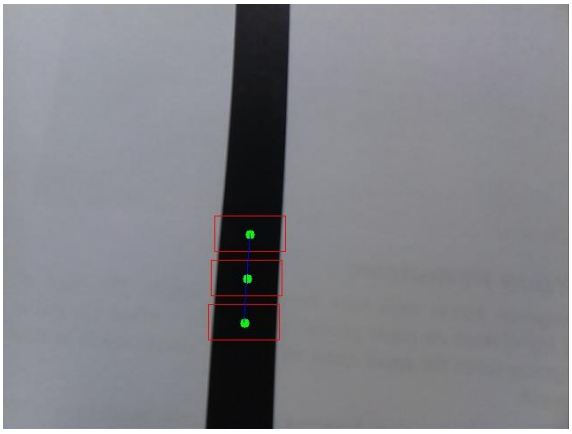

Figura 52.- Con tres puntos detectados

3.4 Fases de desarrollo del proyecto: Fase 3 – Montaje e integración del Robot

Con la detección de línea implementada, el siguiente punto es integrar todo el sistema en el robot. Ahora se procederá a definir todo lo que se ha utilizado para montar el robot.

- **Magician chasis**

En el mercado actual del aficionado a la electrónica, existen bastantes tipos diferentes de chasis, cosa que hace que escoger uno sea un trabajo complicado. Se escogió este chasis por dos razones. La primera razón es porque disponía de dos niveles, y se podían colocar las baterías en una sección, y la electrónica en el otro nivel. La segunda razón fue que la base del nivel superior dispone de muchísimos taladros, y por tanto, permitía colocar todo lo necesario de una manera más sencilla sin tener que hacer modificaciones mecánicas al chasis.

Este chasis viene desmontado, y el kit incluye el chasis, 2 motores de corriente continua con sus ruedas, una rueda loca, toda la tornillería y un soporte de batería.

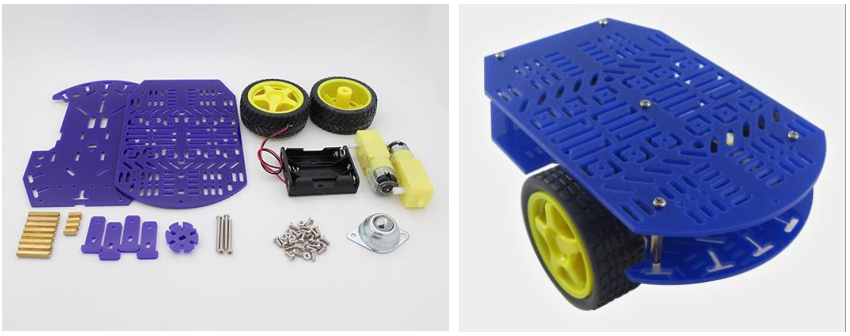

Figura 53.- Detalle del despiece (izquierda) y del chasis montado (derecha)

- **Protoboard**

Para unir las diferentes partes del circuito se utiliza una protoboard pequeña. Se utilizará sobre todo para unir las conexiones de los motores y la parte de la Raspberry.

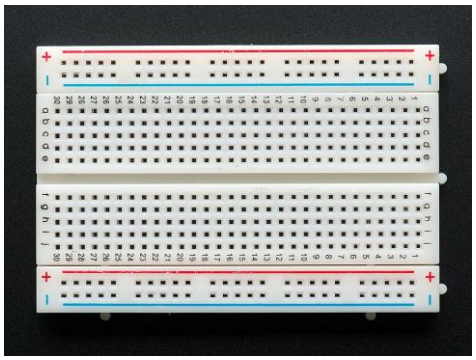

Figura 54.- Protoboard

- **Control de motores.**

El control de los motores necesita un puente en H, que ayuda a cambiar la dirección de giro de un motor utilizando un control con señales lógicas. El controlador seleccionado es uno muy típico para estas aplicaciones, el ***L293D***, ya que es multimarca y es relativamente fácil de encontrar.

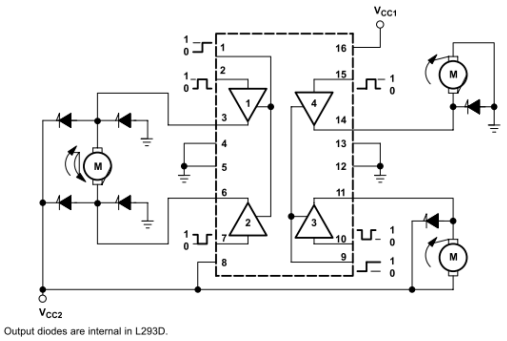

Figura 55.- Diagrama de conexión típico del L293D

Una de las razones por la que se ha escogido este controlador es porque los diodos de protección del motor son internos (Aunque en el dibujo parezca que están fuera), cosa que simplifica bastante la circuitería. Otra buena razón ha sido porque dispone de dos entradas de tensión, una para controlar las señales digitales, y la otra para controlar las salidas del motor, y aceptan tensiones de hasta 36 Volts, de esta manera se pueden aislar las señales digitales de las señales analógicas, que son más ruidosas.

- **Módulo de control.**

El núcleo de control del sistema será la Raspberry pi, será el que recibirá los datos desde todos los módulos del sistema y los gestionará. El módulo que se utilizará será el de la Raspberry Pi 3.

- **Cámara.**

La cámara que se utilizará será la primera versión de la cámara de la Raspberry, es decir, la que tiene una resolución de 5 megapíxeles. Como el cable que viene en la cámara original es muy pequeño, se ha añadido un cable de 300 mm, más que suficiente para poder colocar la cámara sin problemas.

Para poder fijar la cámara, se utilizar una funda de plástico con color carbono.

Figura 56.- Funda cámara Rasberry Pi

- **Soporte cámara.**

Para poder ir variando la posición de la cámara dependiendo de las necesidades, se ha decidió colocar un soporte de la cámara móvil. La mejor opción que se encontró fue un soporte en el que se pueda controlar el giro y la inclinación de la cámara, lo que normalmente se llama soporte Pan & Tilt.

Figura 57.- Pan & Tilt

Este tipo de dispositivos se basa en dos servomotores de 180º conectados entre sí, de manera que el inferior controla el giro, en este caso la dirección a la que mira la cámara, y el segundo controla la inclinación de la cámara.

Estos servos son unos ***DGServo S3003***, y tiene un conector de tres pines, con la siguiente configuración:

Marrón → GND
Rojo → +Vcc
Naranja → Señal de control

La alimentación de este motor va de 4,8 V a 6 V. La señal de control, como ya se explicó, funciona con una señal PWM de 20 ms de periodo.

- **Control de motores.**

Aunque con la Raspberry Pi se pueden gestionar los motores, no es su punto fuerte, y además se estaría añadiendo carga de trabajo a la tarjeta. Para intentar descargar el control, se ha añadido un Arduino al robot. Este Arduino se comunicará con un puerto I^2C con la Raspberry, y esta le indicará como controlar la posición de los servos del soporte de la cámara, la dirección a la que giran los motores, etc.

Como el Arduino se puede alimentar directamente de lo que le llega desde el puerto USB, recibirá la alimentación desde un conector USB de la Raspberry, ya que no consume mucho, y así la interconexión será más sencilla.

- **Batería.**

Al ser un sistema autónomo con motores, y dos tarjetas que alimentar, se ha pensado en utilizar una batería de gran potencia. Se han descartado las pilas ya que ocupan mucho y son poco prácticas, ya que para recargarlas hay que sacarlas, se necesitarían muchas pilas unidas en serie, etc.

Como batería se ha decidido utilizar una batería externa USB, como las que se utilizan para cargar los móviles. La batería escogida es de la marca ***Tecknet Powerzen G2 2nd Generation***.

Figura 58.- Batería

Esta batería tiene varias características que la hacen interesante:

1. Se recarga utilizando un cargador de móvil.
2. Tiene una potencia de 9600 mAh, y por tanto, la autonomía del robot será buena.
3. Dispone de dos salidas, de 2,5 A, y por tanto, se podrá separar la alimentación de la parte digital (Raspberry y Arduino) y la alimentación de los motores (Servos y motores).

Para el control de los motores de corriente continua se ha decidido utilizar una pila de 9V, porque con la batería de 5 V no tenía una buena respuesta.

- **Alimentación USB/Protoboard.**

Para alimentar la protoboard sin tener que modificar un cable USB, se ha decidido utilizar una pequeña tarjeta que se conecta en los carriles de tensión de la tarjeta Protoboard, y proporciona los 5 V de entrada del USB (o de un conector Jack) a las líneas de alimentación de la protoboard. Además este dispositivo integra un interruptor, que permitirá activar y desactivar los motores, y puede regular la tensión a 3,3 Voltios.

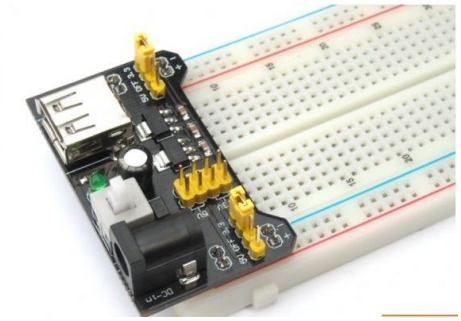

Figura 59.- Adaptador de entrada USB a protoboard

- **Interconexión entre módulos y fijación.**

La conexión entre los diferentes módulos se realiza utilizando cables USB, y cables de interconexión entre la Raspberry/Arduino con la protoboard.

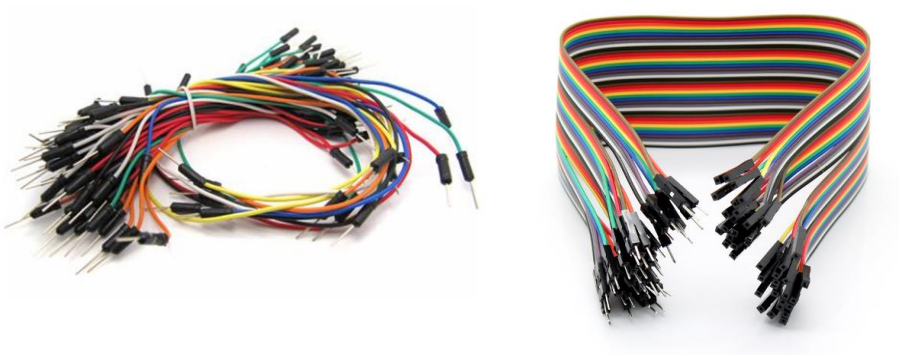

Figura 60.- Cables de interconexión para protoboard

Para fijar los módulos se utilizan bridas, ya que de esta manera el movimiento de los diferentes módulos será rápido. En el caso del módulo Pan & Tilt la fijación se realiza con una torreta metálica, ya que el movimiento de los servos puede ser muy brusco.

- **Adafruit NeoPixel 12 Leds.**

Para evitar problemas con la luz ambiental, se ha decidido implementar un sistema de iluminación para la cámara. El sistema funcionará como un foco de una cámara, y se activará cuando el programa esté funcionando.

El módulo utilizado es una versión del módulo de Adafruit, en su versión de 12 LEDS. Este módulo utiliza 12 Leds RGB configurables ***WS2812B*** de Worldsemi, funciona a 5 V, y dispone de un puerto de entrada de datos para configurar los LEDS, y un módulo de salida de datos, que en este proyecto no se utilizará.

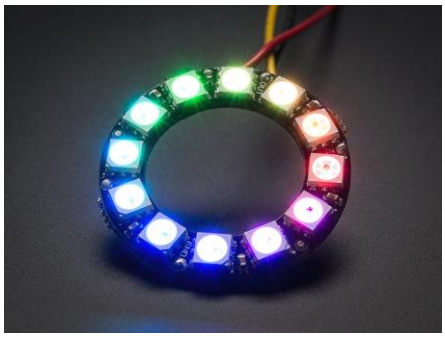

Figura 61.- NeoPixel Ring 12 LEDs

- **Librerías adicionales.**

Para poder implementar de manera sencilla el control de los diferentes módulos, se han utilizado diferentes librerías de código abierto.

- Wiring Pi [29]:

La Raspberry tiene un conector de 40 pines, con GPIO's, buses de comunicación, etc. El diagrama del bus de la Raspberry pi 3 es el siguiente:

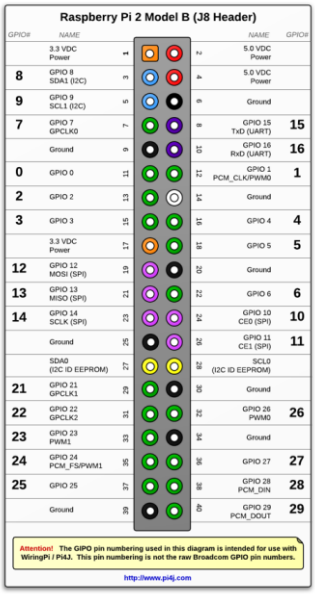

Figura 62.- Detalle del bus GPIO Raspberry Pi 3

Para acceder a los distintos pines hay que acceder a nivel de registros del microprocesador BCM2837. El creador de esta librería, Gordon Henderson, quiso realizar unas funciones que facilitarán el acceso a estas señales GPIO.

El acceso a los pines se realiza configurando los GPIO siguiendo los números que están en la figura anterior. Es decir, si se quiere acceder al pin 11, se deberá configurar el GPIO0.

Además del acceso a estos GPIO's, la biblioteca es capaz de configurar y trabajar con los accesos al bus SPI, I^2C, etc.

A diferencia de la primera versión de la tarjeta, que solo disponía de 1, la Raspberry pi dispone de 2 PWM integrados, y tienen salida directa desde el GPIO 1 y GPIO 24. Esta librería además es capaz de generar, de una manera sencilla y transparente para el usuario, PWM's por software, y por lo tanto puedes ampliar la disponibilidad de PWMs utilizando otros GPIO's de la tarjeta, a expensas de tener un error inherente en la señal, ya que la señal tendrá un pequeño desfase provocado por el clock del sistema y la generación de las señales que salgan de esos pines.

Para configurar la librería y configurar los pins de salida como PWM, el código es el siguiente:

```
// Se arranca la librería Wiring Pi
wiringPiSetup();

// Se incilizan los GPIO's de los PWM
pinMode(1,PWM_OUTPUT);
pinMode(24,PWM_OUTPUT);
// Los valores de los pwm serán  en ms
pwmSetMode(PWM_MODE_MS);
pwmSetClock(PWM_SET_CLOCK);
pwmSetRange (PWM_SET_RANGE) ;
// Se arrancan los PWM en el valor de reposo.
// Con 0 ms haces que el pwm esté siempre a 0
pwmWrite (1, 0);
pwmWrite (24, 0);
```

- Librería de control de servos Arduino:

El control del soporte de la cámara se ha dejado al Arduino porque dispone de una librería, ***Servo.h***, que permite a Arduino controlar un servo de 180 grados de giro. Directamente, con una función ajustas la salida, como si se accediera a un PWM y se configurará. Para que el código funcione se debe incluir la librería en el proyecto Arduino y configurar las salidas.

```
// Libreria de los servos
#include <Servo.h>
…
// Variable para los servos
Servo servoYaw; // Servo para el giro
Servo servoPitch; // Servo para la inclinación
```

Lo primero que se hace es crear unas variables para los servos, que se llamarán *servoYaw* y *servoPitch*.

```
// Se enlazan las salidas de los servos con los pines.
  servoYaw.attach(YawPin);
  servoPitch.attach(PitchPin);

  // Se inicializa la posición
  servoYaw.write(90);
  servoPitch.write(75);
```

El siguiente paso, ya en la parte del ***setup()***, se debe asociar la variable *Servo* a un pin, y después ya se puede escribir. El valor a introducir es directamente el ángulo, por tanto los valores irán de los 0 grados a 180 grados.

- Librería NeoPixel para Arduino:

Aunque se puede utilizar con Raspberry, el NeoPixel está más optimizado para trabajar con el Arduino. Como Adafruit dispone de varios modelos de NeoPixel (Con distinta cantidad de Leds

(8, 12, etc.), y distintas formas (circular, línea)), y por ello ha creado una librería de uso libre para controlar el dispositivo. Se puede descargar directamente desde la web de ***Adafruit*** [30], y grabarla en el directorio de librerías de Arduino, o utilizar el gestor de librerías interno del programa Arduino e instalarlo desde ahí.

```
// Se incluye la librería de anillo de leds (Flash)
#include <Adafruit_NeoPixel.h>
…

// Defines de las constantes del Flash
const int Flash_PIN = 2; // El pin de control será el 1
const int STRIPSIZE = 12; // El anillo tiene 12 LEDS.
…

// Parametros para inicializar el anillo de Leds
// Parametro 1 = Número de pixeles en el anillo
// Parametro 2 = Número de pin
// Parametro 3 = Descriptores del tipo de pin, se pueden combinar:
//   NEO_KHZ800  Flujo de datos de 800 KHz (La mayoria de los productos NeoPixel con leds WS2812)
//   NEO_KHZ400  400 KHz (Para el resto de versiones classic 'v1' (not v2) FLORA pixels, WS2811
drivers)
//   NEO_GRB     Los pixeles estan cableados para flujo de datos GRB (La mayoria de los productos
NeoPixel)
//   NEO_RGB     Los pixeles estan cableados para flujo de datos RGB (v1 FLORA pixels, not v2)
Adafruit_NeoPixel Flash = Adafruit_NeoPixel(STRIPSIZE, Flash_PIN, NEO_GRB + NEO_KHZ800);
```

Lo primero es utilizar la librería del dispositivo. Después se indica el pin por donde saldrá la señal de datos hacia el anillo y la cantidad de Leds que tiene. Por último se crea una variable que controlará el flash

```
// Inicialización del Flash
  Flash.begin();
  Flash.setBrightness(75);  // Control de Brillo de los leds
  Flash.show(); // Se inicializan los Leds a 0.
```

Dentro de la rutina de setup, se debe inicializar el flash, en este caso se pone un brillo del sistema al 75 %, y se inicializan todos los Leds a 0, que es el color negro, es decir, que no están encendidos.

```
colorWipe(Flash.Color(255, 255, 255), 25); // Se enciende el flash
```

Cuando se recibe la orden de poner encender el flash, se ponen todos los Leds a 255, de esta manera se iluminan todos y el color que aparece es el blanco.

```
colorWipe(Flash.Color(0, 0, 0), 25); // Se apaga el flash
```

En cambio, cuando se pone todo a 0, el led se apaga.

```
// Función para pintar los pixeles, uno a uno
void colorWipe(uint32_t c, uint8_t wait)
{
  // Se recorren todos los Leds
  for(uint16_t i=0; i<Flash.numPixels(); i++)
  {
      Flash.setPixelColor(i, c);
      Flash.show();
      delay(wait);
  }
}
```

La función colorWipe recorre todos los Leds que hay en el sistema, en este caso 12, y le cambia el color, añadiendo un delay entre cada escritura. Con esta función es muy fácil hacer un efecto de encendido progresivo de los Leds, jugando con el valor de retraso.

- Comunicación I^2C entre Raspberry y Arduino:

La comunicación entre las dos tarjetas se implementa utilizando los pins dedicados, es decir, en la Raspberry el pin de la SDA es el pin 3 (GPIO 8), y el de SCL es el pin 5 (GPIO 9), y en el Arduino son el pin A4 y A5.

Desde la Raspberry, se accederá al bus como se acceden a los dispositivos en Linux, es decir, se tiene que abrir el fichero que gestiona el recurso, en este caso el bus I2C-1, y acceder a él como si fuese un fichero.

```
// Includes para controlar las comunicaciones con el arduino
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <ctime>
#include <iostream>
#include <unistd.h>


// Constantes para el bus I2C
// The Arduino address
#define ADDRESS 0x04

// The I2C bus
static const char *devName = "/dev/i2c-1";

int file;

// Creo un puntero para el I2C
if ((file = open(devName, O_RDWR)) < 0)
{
        fprintf(stderr, "I2C: Failed to access %d\n", devName);
        exit(1);
}

printf("I2C: acquiring bus to 0x%x\n", ADDRESS);

if (ioctl(file, I2C_SLAVE, ADDRESS) < 0)
{
    fprintf(stderr, "I2C: Failed to acquire bus access/talk to slave 0x%x\n", ADDRESS);
    exit(1);
}
```

Primero se tienen que añadir las librerías necesarias para poder acceder al fichero de gestión del bus, y después, utilizando las variables de control del bus, se intenta abrir el fichero para comunicarse. Si el fichero se ha abierto, se verifica si el Arduino (Cuya dirección es la 0x04) está conectado. Si no lo esta se finaliza el programa. Si está conectado, se continúa.

Una vez creado el acceso al fichero que gestiona el recurso del I2C-1, ya se puede acceder a él escribiendo funciones de escritura y lectura.

```
unsigned char cmd[16];
cmd[0] = 'Y'; // Se quiere modificar el angulo de giro
write(file, cmd, 1);
```

En este ejemplo se envía por el bus I^2C el carácter 'Y'.

Para configurar el bus por el Arduino, es más sencillo, ya que existe una biblioteca que se encarga de gestionar el bus, ***Wire.h***. Los pasos para conectarse son los siguientes:

```
// Libreria para la comunicación I2C
#include <Wire.h>

// Definición de la dirección I2C
const int I2CADDRESS = 4;
```

Aquí se especifica que la dirección de trabajo del I2C será la 4. Dentro del código de Setup se inicializa el I^2C.

```
// Inicializo el puerto I2C
  Wire.begin(I2CADDRESS); // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event
```

La segunda instrucción ***orReceive()***, genera una atención a la interrupción, es decir, que cuando llegue un dato desde el bus, el sistema lo sabrá e irá a ver que es. Está rutina proporciona la cantidad de datos que se han recibido en el bus. En esa rutina primero se decodificará el valor que envíe la Raspberry, y se realizarán las acciones necesarias. Los códigos son:

Código	Acción
Y	Cambio del valor de giro del Pan & Tilt
P	Cambio de valor de la inclinación del Pan & Tilt
M	Modificar dirección de giro de los motores
F	Modificar estado del anillo de Leds

- Diagrama de bloques: El diagrama de conexión entre los bloques es el que sigue.

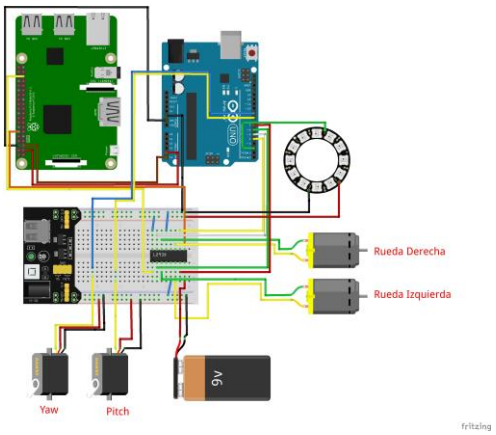

Figura 63.- Interconexión de la electrónica

- Montaje final: El aspecto final del Robot es el de las figuras de abajo.

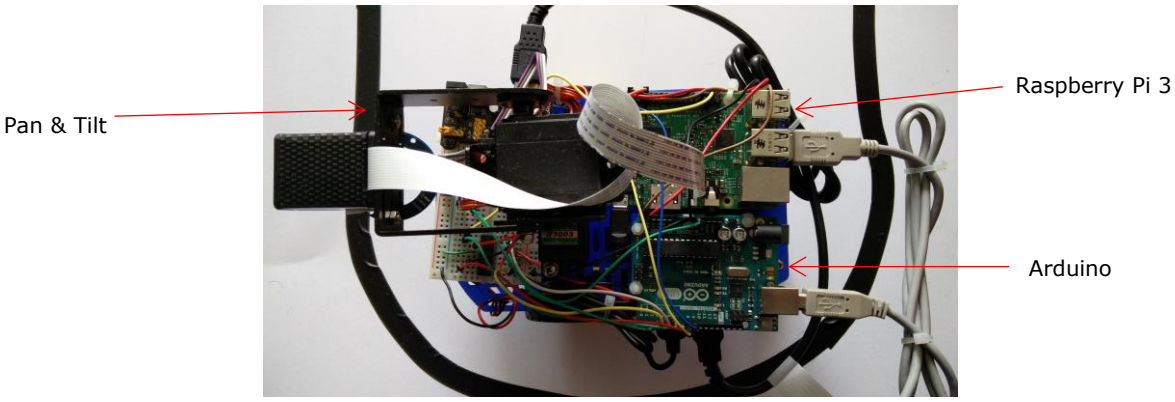

Figura 64.- Vista superior LineBot

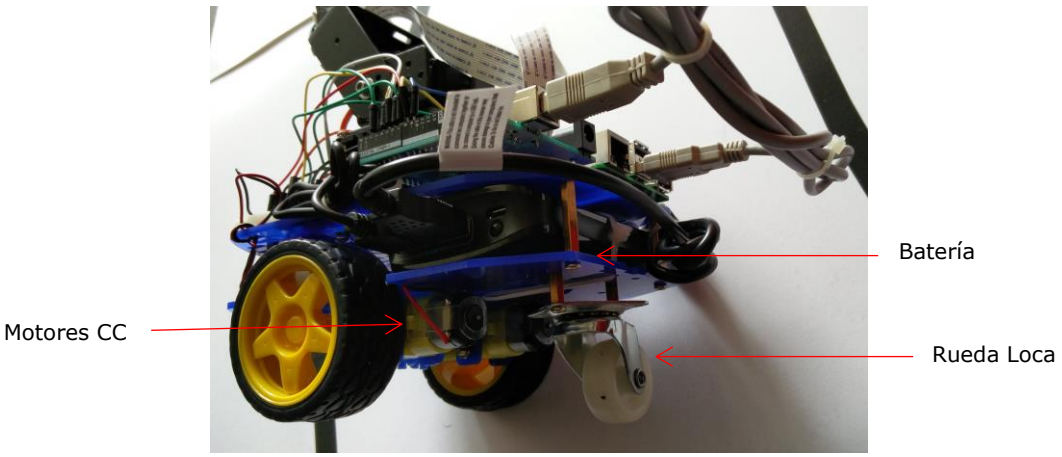

Figura 65.- Vista lateral Linebot

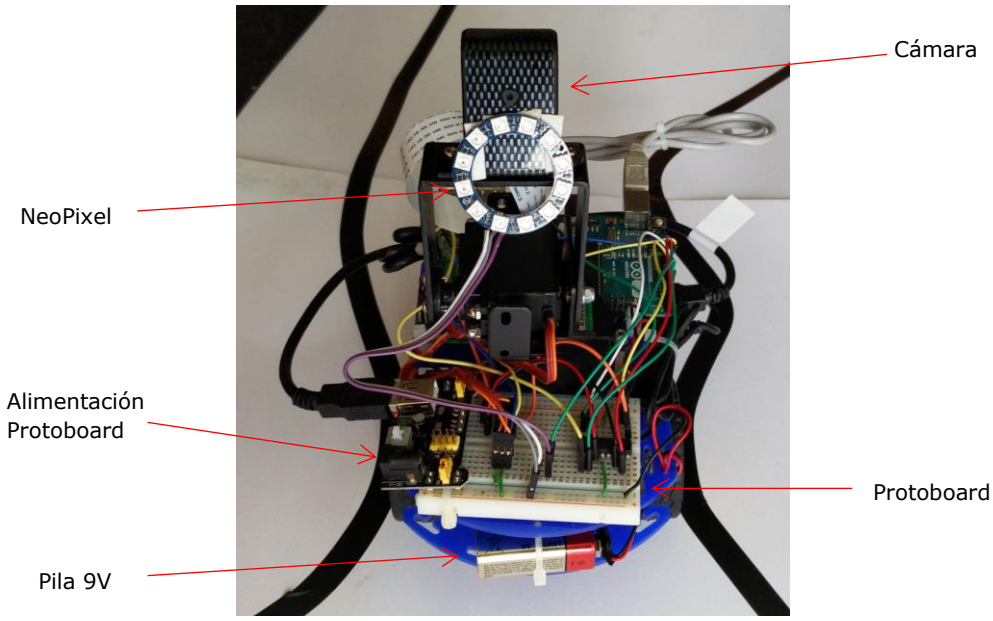

Figura 66.- Vista frontal Linebot

4. Control del Robot

Con el robot ensamblado, ya se puede comenzar a trabajar en el control del robot. En este caso, el control del movimiento del robot se realizará utilizando un control PID, que, con la información obtenida desde la cámara, realizará los cálculos necesarios para que el robot siga la línea.

4.1 Controlador PID

Un controlador PID (Que son las iniciales de Proporcional Integrativo Derivativo) es un mecanismo de control sobre la realimentación de bucle cerrado ampliamente utilizado en sistemas de control industrial. Este controlador calcula el error como la diferencia entre el valor actual del sistema y el valor al que se desea llegar. El controlador intenta minimizar el error ajustando la entrada del sistema [31].

El controlador viene determinado por tres parámetros: El proporcional, el integral y el derivativo. La parte proporcional P depende del error actual, la integral I depende de la suma de todos los errores pasados, y la derivativa D es la predicción de errores futuros. La suma ponderada de los tres términos permite ajustar el proceso mediante un elemento de control, como por ejemplo, en este proyecto se ajustará la velocidad de las ruedas del robot.

Ajustando estas constantes en el algoritmo de control del PID, el regulador es capaz, de proporcionar acciones de control específicas a los requerimientos de un sistema. La respuesta del controlador se puede describir como la capacidad de respuesta ante un error, el grado en el que el regulador llega más allá del punto de ajuste, y la oscilación del sistema.

Los valores para ajustar el controlador se obtienen analizando el sistema que se quiere regular, y los ajustes tienen que ver con el tiempo de respuesta, los errores que puede tener hasta ajustar la salida y las sobre oscilaciones del sistema hasta llegar al valor deseado. Estos valores están interrelacionados entre sí, ya que si, por ejemplo, se quiere una reacción muy rápida, el sistema sobrepase el valor deseado al intentar ajustar y se cree una sobre oscilación inevitable. Lo que está claro es que un PID no tiene por qué funcionar siempre, y hay sistemas que pueden ser inestables y no pueden ser corregidos.

Dentro de los controladores PID, hay controladores que no necesitan utilizar todos los parámetros del control, y por tanto, existen combinaciones con sus parámetros. Existen controladores proporcionales (P), Proporcionales-Derivativos(PD), Proporcionales-Integrativo, Integrativo, etc. Los más comunes son los PI, porque la acción derivativa suele ser más sensible al ruido, y la ausencia de la acción integral lentifica el sistema y puede provocar que la salida nunca llegue al valor deseado.

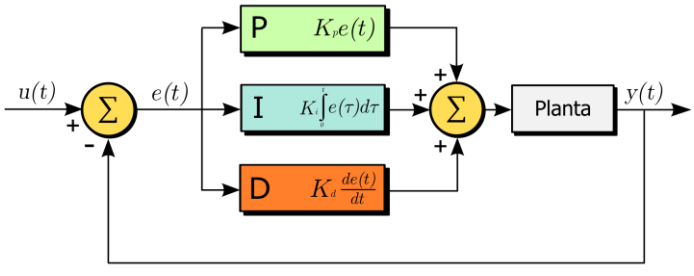

Figura 67.- Controlador PID (Fuente Wikipedia)

- Proporcional:

El término proporcional modifica la salida proporcionalmente con el error actual. La respuesta proporcional se puede ajustar multiplicando el error por una constante, típicamente K_P, conocida como ganancia proporcional. Este término crece cuando el error es muy amplio, y se reduce cuando el error es muy pequeño, hasta que el error llega a cero y el valor se va a 0. La fórmula para expresar la ganancia proporcional es:

$$P = K_P \cdot e(t)$$

Ecuación 3.- Ganancia Proporcional

Donde *e(t)* es el error en un instante de tiempo.

- Integral:

El término integral el proporcional a la magnitud y duración del error. En otras palabras, es la suma de todos los errores en cada instante de tiempo, o lo que es lo mismo, la integración de los errores.

La suma de estos errores compensa la diferencia que debería haber sido corregida anteriormente. El error acumulado se multiplica por la ganancia integral, K_I, que es la cantidad de acción integral se debe sumar al control.

Al usar la variable proporcional junto a la variable integral, el movimiento del sistema se acelera, llegando antes al valor deseado. Pero como el término integral tiene en cuenta los errores acumulados en el pasado, puede ser que el valor actual sobrepase el valor deseado, y creara una oscilación alrededor del valor final deseada hasta conseguir ajustarse y entonces estabilizarse. Esta sobreoscilación es crítica, porque si el sistema tarda mucho en estabilizarse, entonces puede entrar en una inestabilidad y estar siempre oscilando.

La ganancia integral se representa con la siguiente ecuación:

$$I = K_I \cdot \int_0^t e(t) \cdot dt$$

Ecuación 4.- Ganancia integral

- Derivativa:

El término derivativo calcula la variación del error mediante su pendiente en cada instante de tiempo, es la primera derivada con respecto al tiempo. Este error se multiplica con la ganancia derivativa, K_P.

Este término ralentiza la salida del controlador, y cuanto más cerca está del valor deseado, más lento se hace. La mayor utilidad de este componente es disminuir las oscilaciones producidas por la parte integral, y así mejorar la estabilidad del proceso. El inconveniente que tiene es que al hacer la derivada, el ruido se amplifica, y hace que el regulador sea más sensible a las interferencias, pudiendo llevar el sistema a la inestabilidad.

El término derivativo se suele formular como:

$$D = K_D \cdot \frac{d}{dt} e(t)$$

Ecuación 5.- Ganancia derivativa

- Ecuación del controlador:

La fórmula que socia todos los términos es la siguiente:

$$u(t) = K_P \cdot e(t) + K_I \cdot \int_0^t e(t) \cdot dt + K_D \cdot \frac{d}{dt} e(t)$$

Ecuación 6.- Formula PID

Aunque a veces se escribe de la siguiente manera

$$u(t) = K_P \cdot e(t) + \frac{K_P}{T_I} \cdot \int_0^t e(t) \cdot dt + K_P \cdot T_D \cdot \frac{d}{dt} e(t)$$

Ecuación 7.- Formula PID utilizando Tiempos integrales y derivativos

Donde T_I es el tiempo integral y T_D es el tiempo derivativo.

De la ecuación se pueden extraer los siguientes puntos:

1. Contra más grande sea K_P, más rápida será la respuesta, ya que el error será mayor, pero si es muy grande puede provocar oscilaciones e inestabilidad.
2. Valores grandes en K_I eliminarán los errores estacionarios más rápidamente, pero provocará mayor sobre oscilación.
3. Si la K_D aumenta mucho, la sobre oscilación se reduce, pero aumenta el tiempo de respuesta, y además, al amplificar el ruido en el cálculo diferencial, el aumento del error puede provocar que el sistema sea inestable.

- Ecuación del discreta del controlador:

Para poder controlar un PID con un procesador, no se puede trabajar con señales continuas, sino que se debe trabajar con señales discretas y finitas. Por tanto, se debe discretizar la ecuación del controlador y utilizar los valores muestreados como valores anteriores.

$$u(i) = K_P \cdot e(i) + K_I \cdot T_S \cdot \sum_{k=0}^{i} e(k) + \frac{K_D}{T_S} \cdot (e(i) - e(i-1))$$

Ecuación 8.- Formula PID discreto

Donde T_S es el tiempo de muestreo de la señal [32].

Esta será la ecuación que se utilizará en el sistema para controlar el funcionamiento del robot.

- Ajuste del controlador:

Existen diversos métodos de ajuste numérico, como por ejemplo el Método de Ziegler-Nichols o el Método de Cohen-Coon, que son un buen punto de partida para elegir los valores del controlador. En

este caso se ha utilizado un método más sencillo, y ha sido un método heurístico o de prueba y error.

Los pasos que se siguen son los siguientes:

1. Acción proporcional: Se aumenta poco a poco para disminuir el error y aumentar la velocidad de respuesta. Si se vuelve inestable, se aumenta la acción derivativa, sino, el PID está sintonizado

2. Acción derivativa: Si el sistema es inestable, se aumentará poco a poco la constante derivativa para conseguir de nuevo la estabilidad

3. Acción integral: Si el error es muy grande, se aumenta la constante integral hasta que el error se minimice con la rapidez deseada. Si el sistema se vuelve inestable, se debe aumentar la acción derivativa.

4.2 Planteamiento del problema

El modelo del que del robot del que se ha basado este proyecto es el siguiente [33]:

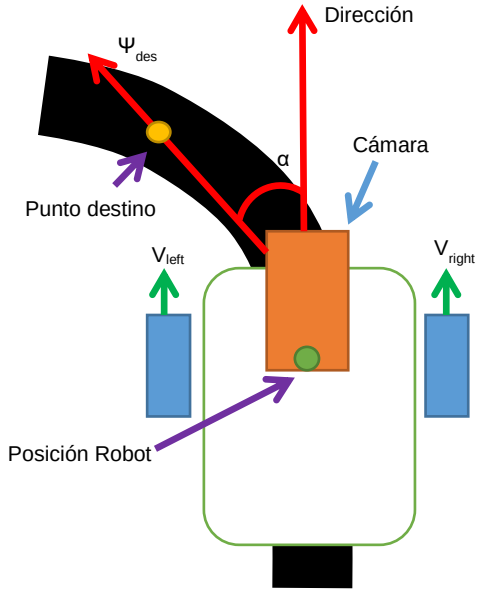

Figura 68.- Diagrama de movimiento Robot.

En el dibujo se puede observar el robot sobre una línea negra. El robot dispone de dos ruedas motrices y una cámara de control.

La posición actual del robot es el punto medio entre las dos ruedas motrices, donde está el círculo verde, y se puede notar como $x_{inicial}$ e $y_{inicial}$. Además, el robot se mueve en la dirección que marca la flecha, que, aplicando una analogía con los ejes cartesianos, sería en y, y su velocidad es:

$$V_y = \frac{v_{right} + v_{left}}{2}$$

Ecuación 9.- Velocidad total robot

En el dibujo la línea continua hacia la izquierda, y por tanto, si el robot no cambia su dirección, perdería la línea. Para continuar siguiendo la línea, debería poder llegar al punto destino, marcado con las coordenadas x_{dest} e y_{dest}, y de esta manera estaría siguiendo la línea correctamente.

Para poder llegar al punto de destino, la dirección del robot debe adquirir la dirección que marca el vector Ψ_{dest}, cuyo diferencia de ángulo con la dirección actual se marca con el ángulo α. Esta corrección de ángulo se realizará modificando la velocidad de las dos ruedas, con las siguientes formulas:

$$v_{right} = V + corrección$$

$$v_{left} = V - corrección$$

Ecuación 10.- Velocidad en cada rueda

V será una constante, y la corrección será lo que proporcionará el controlador PID. Toda la corrección se basará sobre todo en el ángulo de desfase entre el punto destino y el punto en el que está el robot.

Para calcular el ángulo se utilizará la arcotangente, utilizando los puntos conocidos. El primero es la posición del robot, y por tanto, es una posición conocida. El segundo es la posición deseada, o hablando en parámetros de detección de línea con OpenCV sería donde está el centro de la línea negra. El bloque del controlador quedará como:

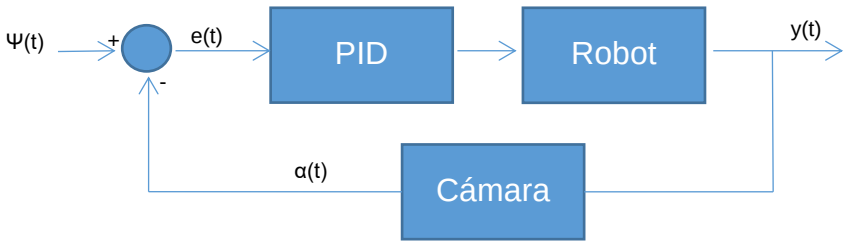

Figura 69.- Controlador Robot

A partir de una entrada *Ψ(t)*, se le restará el ángulo real que hay en el robot, obtenido desde la Cámara. Esta resta proporcionará el error del sistema, que entrará al controlador PID, y este lo transformará en valores de velocidad para los motores.

La única incógnita que queda por resolver es como asociar las distancias entre los puntos, y de esa manera calcular correctamente el desfase de posición. La fórmula para calcular el ángulo de desfase es la siguiente:

$$\alpha = -\tan^{-1}\left(\frac{x_{dest} - x_{inicial}}{y_{dest} - y_{inicial}}\right)$$

Ecuación 11.- Formula para cálculo del ángulo de defase

Y aquí falta saber dos variables, $x_{inicial}$ e $y_{inicial}$, ya que x_{dest} e y_{dest} las conseguimos al calcular la posición de la línea con el OpenCV. De las dos variables iniciales, $x_{inicial}$ siempre será 0, ya que será el punto medio del robot.

Para calcular el valor de $y_{inicial}$ se tiene que tener en cuenta lo que "ve" el robot.

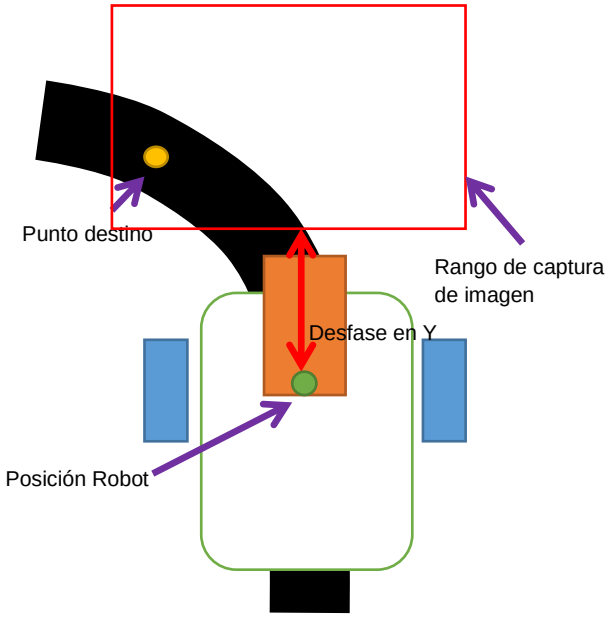

Figura 70.- Diagrama con la imagen de la cámara.

Como se puede ver en la imagen, hay que ajustar este valor de desfase en Y para poder hacer bien la arcotangente. Dentro del control de la cámara, los valores se trabajarán con pixeles, por lo tanto, hay que adaptar este desfase a pixeles. Para realizar esta conversión se utilizará una cinta métrica y una captura de pantalla. Se verificarán cuantos cm de la cinta métrica aparecen en pantalla, y eso se dividirá por la cantidad de pixeles captados.

$$R_{Pix/cm} = \frac{num\ pixeles\ en\ eje}{cm\ medidos}$$

Ecuación 12.- Relación entre pixeles y mm.

El valor obtenido será el que se utilizará para calcular los cm entre el eje del robot y la detección del punto deseado.

4.3 Algoritmo de control

A partir del planteamiento del problema, se ha establecido un diagrama de flujo:

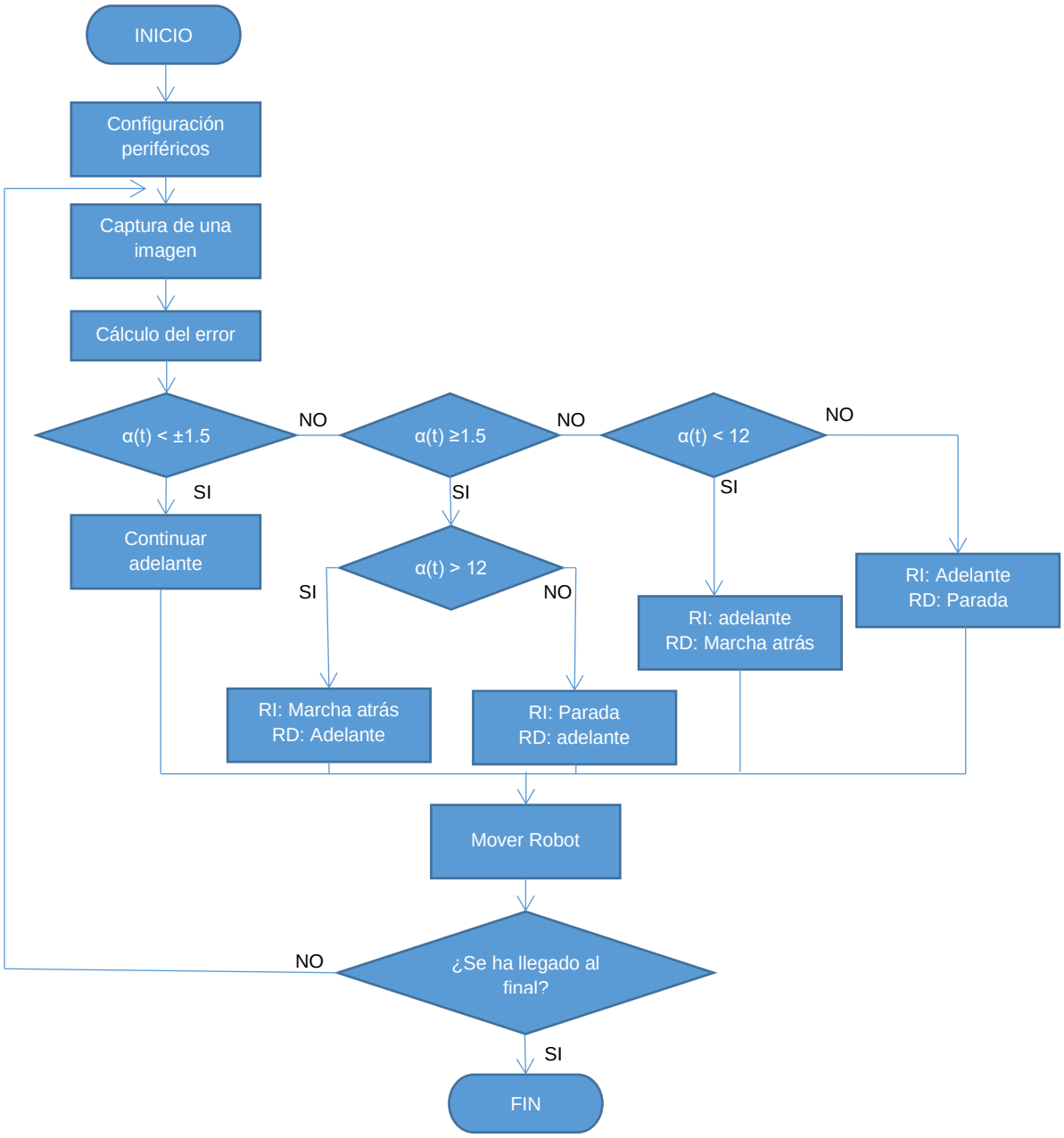

Figura 71.- Diagrama de flujo del PID

El programa empieza configurando todos los dispositivos preparados para el control del robot, como la cámara, los PWM, la conexión con Arduino y el estado de todos los motores.

Una vez está todo preparado, inicia el bucle del programa. Se captura una imagen, y se calcula el error. Para gestionar el error se han preparado cinco escenarios posibles:

1. El error es menor a ±1,5 grados, y por tanto, para que no entre en un estado inestable, se continúa hacia adelante, sin modificar la dirección
2. Si el error esta entre 1,5 grados y 12 grados, se debe girar a la izquierda, y la corrección se realiza parando la rueda contraría al movimiento, y dejando otra rueda en movimiento. Este sería un giro moderado.

3. Si el error es mayor que 12, el giro a la izquierda debe ser más brusco, y por tanto se configura la rueda que estaba parada para que funcione en sentido opuesto y el giro sea como el de un tanque, más rápido.
4. Si el error está entre -1,5 grados y -12 grados, se debe girar a la derecha, y se tiene que hacer el movimiento opuesto del punto 2.
5. Si el error es mayor que 12, se debe realizar un giro a la derecha más brusco, y realizar el movimiento contrarío del punto 3.

Por último, se verifica si se ha llegado al final. Si no es así, se continua el movimiento, si se ha llegado al final de la línea, el robot se para.

El código generado a partir de este diagrama de flujo es:

```
   // Función para establecer el movimiento del robot
   float lineControl(int &leftDirection, int &rightDirection, int &leftSpeed, int &rightSpeed, float
&samplePeriod, int i2cFile)
   {
       // Se calcula la arcotangente entre los dos puntos
       angleDiff = 90 - atan2(static_cast<float>((CAMERA_HEIGHT - yPointDec) + 210),
static_cast<float>((CAMERA_WIDTH / 2) - xPointDec)) * 180 / PI;
       // Error integral
       integralError = integralError + angleDiff;
        // Calculo del PID
       outputDegree = kMotorProp*angleDiff + (kMotorDeriv*(angleDiff - errorAntDegre)) +
kMotorInt*integralError;
       errorAntDegre = angleDiff; // Se guarda el error anterior para el control diferencial.
       // Se calculan los valores de la corrección
       if(angleDiff > 1.5f)
       {
           // Se para la rueda izquierda y se deja en funcionamiento la derecha
           if(abs(outputDegree) >= 200)
           {
               rightSpeed = motorSpeed + 200;
           }
           else
           {
               rightSpeed = motorSpeed + outputDegree;
           }
           if(angleDiff > 12.0f)
           {
               // Se activa el movimiento en las dos ruedas, el motor derecho hacia adelante y el
               izquierdo hacia detrás.
               if(leftDirection == 1)
               {
                   rightDirection = 1;
                   leftDirection = 0;
                // Se envía a Arduino el cambio de dirección de motores
                   i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
               }
               leftSpeed = 300;
           }
           else
           {
               if(leftDirection == 0)
               {
                   rightDirection = 1;
                   leftDirection = 1;
                // Se envía a Arduino el cambio de dirección de motores
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
               }
               leftSpeed = 0;
           }
       }
       else if(angleDiff < -1.5f)
       {
           // Se para la rueda derecha y se deja en funcionamiento la izquierda
           if(abs(outputDegree) >= 200)
           {
               leftSpeed = motorSpeed + 200;
           }
```

```
        else
        {
            leftSpeed = motorSpeed - outputDegree;
        }
        if(angleDiff < -12.0f)
        {
            // Se activa el movimiento en las dos ruedas, el motor izquierdo hacia adelante y el
            derecho hacia detrás.
            if(rightDirection == 1)
            {
                rightDirection = 0;
                leftDirection = 1;
            // Se envía a Arduino el cambio de dirección de motores
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            rightSpeed = 300;
        }
        else
        {
            if(rightDirection == 0)
            {
                rightDirection = 1;
                leftDirection = 1;
            // Se envía a Arduino el cambio de dirección de motores
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            rightSpeed = 0;
        }
    }
    else
    {
        // Por si viene de un estado con movimiento brusco, se vuelven a colocar los motores hacia
            adelante
        if(rightDirection == 0 || leftDirection == 0)
        {
            rightDirection = 1;
            leftDirection = 1;
            i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
        }
        // Avanzo sin corrección
        leftSpeed = (motorSpeed - 100) - outputDegree/2;
        rightSpeed = (motorSpeed - 100) + outputDegree/2;

    }
    // Se retorna el angulo de corrección
    return angleDiff;
}
```

En la función del cálculo de la arcotangente, aparece el desfase entre el centro del eje del robot y el ángulo de visión de la cámara, que en este caso es 210. Por lo demás, el código, después de calcular el error con el PID, ejecuta exactamente lo mismo que hay en el diagrama de flujo.

4.4 Ajustes y pruebas

Con la teoría de funcionamiento clara, y con el código probado y funcionando, se comenzaron a probar todas las implementaciones realizadas

- Visualización de los resultados en remoto

Para depurar el código se necesitaba poder visualizar las capturas que realizaba el programa. Como la utilización del escritorio remoto era muy lenta, se decidió minimizar la transmisión de datos y enviar solo las capturas en streaming.

La conexión se realizó entre el ordenador y la Raspberry utilizando un socket.

```
// Includes para los sockets
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>

// Variable para guardar el socket
int thisSocket;
// Variable para guardar los datos de conexión
struct sockaddr_in server;
```

Lo primero que se tiene que hacer es incluir las librerías para crear un socket. Lo siguiente crear las variables que guardarán los datos de la conexión.

```
// Función para inicializar el socket.
int socketInit()
{
    ////Se crea el socket
    // La función socket lo que hace es crear un socket del estilo ipv4 con AF INET, y
    // que es del tipo envío de datos con Sock stream. 0 es para indicar que es TCP.
    if ((thisSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        //Si no se ha podido crear el socket, se cierra.
        printf("Could not create socket!!\n");
        if(thisSocket)
        {
            close(thisSocket);
        }
        exit(0); // Se sale del programa, si no hay socket, no funciona.
    }

    // Todo continua perfecto.
    printf("Socket created.\n");

    // Ahora se crea la conexión en este caso se va a utilizar la conexión con el portatil
    server.sin_addr.s_addr = inet_addr("192.168.1.43");
    // Introduzco el tipo de conexión inet
    server.sin_family = AF_INET;
    // Se conecta al puerto 8888, uno dedicado para trabajar entre los dos dispositivos.
    server.sin_port = htons(8888);

    // Se intenta la conexión
    if (connect(thisSocket, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        std::cout << "connect error\n";
        return 1;
    }

    std::cout << "Connected\n";

    return 0;
}
```

Lo siguiente es realizar la inicialización del socket, creando una conexión directa con el ordenador, donde habrá un programa que estará escuchando el puerto 8888 para establecer la conexión.

```
// Función para enviar una imagen
bool socketSend(cv::Mat camImg, int imgSize)
{
    if((bytes = send(thisSocket, camImg.data, imgSize, 0)) < 0)
    {
        return false;;
    }
    return true;
}
```

Para enviar los datos se utiliza la función *send*, que crea un paquete con el tamaño de los datos recibidos y los envía.

La imagen que se transmite es del formato *Mat*, y por tanto, para poder visualizarse en el ordenador se debe hacer una transformación de los datos. La función que realiza esta transformación es la siguiente:

```
// Transformación de imagenes en color.
int ptr = 0;
for (int i = 0; i < img.rows; i++)
{
        for (int j = 0; j < img.cols; j++)
        {
                img.at<cv::Vec3b>(i, j) = cv::Vec3b(sockData[ptr + 0], sockData[ptr + 1], sockData[ptr + 2]);
                ptr = ptr + 3;
        }
}
```

Como la imagen se compone de tres canales (Rojo, Verde y Azul), se debe ir creando la imagen pixel a pixel. El programa funciona con imágenes de 320 x 240 pixeles, que es la resolución a la que se trabaja, ya que la resolución de 640 x 480 pixeles era demasiado pesada y hacía el sistema muy lento.

- Ajuste valores PID:

La primera acción que se realizo es ajustar los valores del PID. Los pasos que se siguieron fueron los siguientes:

1. Se comenzó ajustando el PWM para el control del motor. El PWM de la Raspberry funciona a una velocidad base de 19,2 MHz. Para configurar la frecuencia y rango de la salida del PWM se deben inicializar utilizando las siguientes funciones del Wiring Pi:

```
pwmSetMode(PWM_MODE_MS);        // Los valores de los pwm serán en ms
pwmSetClock(PWM_SET_CLOCK);     // Se establece del divisor de reloj
pwmSetRange (PWM_SET_RANGE);    // Se configura el rango del PWM
```

   La primera función configura el PWM para trabajar con entradas expresadas en milisegundos. La segunda función establece el divisor para la frecuencia base de salida, cuyo valor va de 2 a 4096. La tercera función establece en cuantas partes se puede dividir el periodo.

   Para calcular el valor de frecuencia de salida se utiliza la siguiente fórmula:

$$F_{PWM} = \frac{\left(\frac{19.2MHz}{PWM\_SET\_CLOCK}\right)}{PWM\_SET\_RANGE}$$

Ecuación 13.- Cálculo salida PWM Raspberry Pi

   En este caso se ha establecido que las variables *PWM_SET_CLOCK* = 384 y *PSM_SET_RANGE* = 1000, por lo que la frecuencia de salida es 50 Hz.

   Con el ajuste de la frecuencia de salida del PWM fijado, y sabiendo que para controlar el periodo podríamos partir de 0 hasta los 1000, fuimos cambiando los valores utilizando esta función:

```
pwmWrite (LEFT_MOTOR_PIN, Speed);     // Se configura la velocidad de salida
```

   Al final, se comprobó que hasta los 400 el motor no se comenzaba a mover, por tanto, se estableció que la velocidad mínima de control sería 400.

2. Se estableció un margen de trabajo, para evitar que fuera muy rápido. El margen de trabajo que se escogió es que el motor iría desde los 400 a los 800, y el valor de salida sería 500.

3. Se creó un pequeño circuito para ajustar los valores del PID.

4.

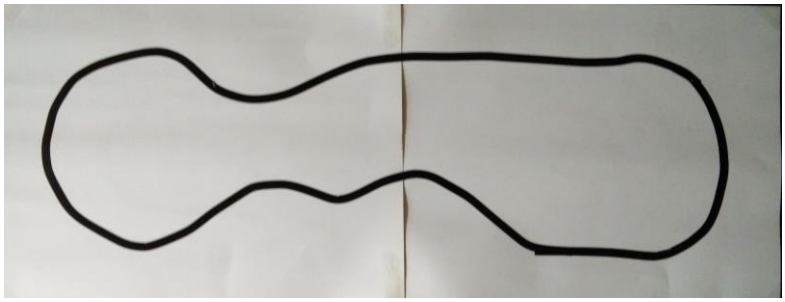

Figura 72.- Imagen del circuito de pruebas

5. Por último, y a partir de unos valores iniciales, se fue enfrentando el robot al circuito de pruebas hasta que el funcionamiento fue el más óptimo. Los valores fueron:

K_P = 6
K_D = 0.1
K_I = 0.3

Al principio la alimentación de los motores venía desde el la batería de 5 V, pero al ver que el motor no iba muy rápido, y que incluso muchas veces ni se movía, se decidió incorporar una pila de 9V externa, y de esta manera acelerar el movimiento de los motores. Cuando se estuvo trabajando con la batería, la tensión de 5 V permanecía estable, pero cuando se introdujo la pila, esta tensión iba variando, cosa que hizo bastante complicado el ajuste del PID. En la siguiente figura se puede observar una gráfica típica de descarga de una pila de 9V

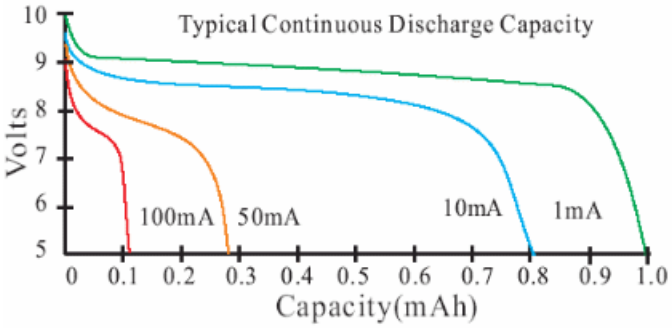

Figura 73.- Imagen del circuito de pruebas

Como se puede ver, al conectar la pila, en la primera parte de la descarga la tensión baja bruscamente de los 9 Voltios nominales a 8 V. Después el valor se mantiene medianamente estable, pero al final va decayendo, hasta que se descarga totalmente. Esta curva hizo que el cálculo final de valores PID fuera complicado para todo el margen de tensiones, ya que el motor tiene una velocidad proporcional a la tensión de entrada, es decir, contra más tensión, y misma corriente, más velocidad.

- Detección de colores:

El código hasta ahora hacía que el robot se moviese de una manera controlada por la línea, pero no existía ningún control de las bifurcaciones. Se decidió que una manera interesante de establecer el control de las bifurcaciones podría ser utilizar una detección de los colores.

En OpenCV las imágenes que se leen desde la cámara se adquieren en espacio de color BGR, en este espacio de color la imagen este dividida en 3 canales cada uno de ellos representa en azul el verde y el rojo respectivamente. El espacio de color RGB funciona de la misma manera excepto por que el orden de los canales está invertido.

El problema con los espacios de color RGB y BGR consiste en la información del color está presente en los 3 canales de la imagen lo cual hace más difícil realizar un filtrado. La mejor manera de filtrar es convertir la imagen al espacio de color HSV, en el que la información que representa el color se encuentra solo en un canal.

El modelo HSV (del inglés Hue, Saturation, Value – Matiz, Saturación, Valor), también llamado HSB (Hue, Saturation, Brightness – Matiz, Saturación, Brillo), define un modelo de color en términos de sus componentes.

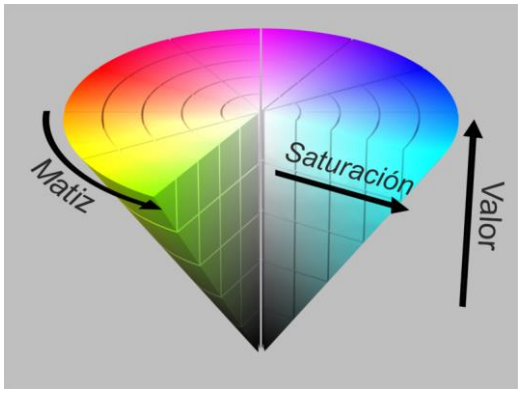

Figura 74.- Cono de colores del espacio HSV

En este caso el Matiz (H) es el color, y se representa con un ángulo que va desde los 0 a los 360. La saturación(S) se representa como la distancia al eje de brillo-blanco, y va de 0% al 100%, donde el 100 será el color más puro. Por último, el Valor (V) representa la altura del eje blanco-negro, y va del 0% al 100%, donde el 0 es el negro, y el 100 % sería el color blanco o un color más o menos saturado.

El cambio de espacio de color en OpenCV es bastante sencillo, ya que existe una función que realiza esta conversión:

```
// Para guardar los datos de la cámara
cv::Mat camImg;
// Para guardar la imagen en HSV
cv::Mat imgHSV;

// Se pasa la imagen de RGB a HSV
cv::cvtColor(camImg, imgHSV, cv::COLOR_RGB2HSV);
```

La función ***void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)*** transforma la imagen *camImg*, que es RGB, a *imgHSV* que es HSV.

Como se ha explicado, el HSV funciona en rangos de color, de saturación y de brillo, y por tanto hay que usar una función que detecte un color dentro de un rango de una imagen, y devuelva una imagen únicamente con la zona en la que se encuentra el color. En OpenCV, los rangos de H van de 0 a 180, y los de S y V van de 0 a 255. Teniendo esto en cuenta, se puede definir el siguiente código:

```
// Se definen los rangos del rojo
const int HSVMIN_RED[3] = {150, 100, 100};
const int HSVMAX_RED[3] = {180, 255, 255};

//Se filtra la imagen para el color Rojo
cv::inRange(imgIn, cv::Scalar(HSVMIN_RED[0], HSVMIN_RED[1], HSVMIN_RED[2]), cv::Scalar(HSVMAX_RED[0],
HSVMAX_RED[1], HSVMAX_RED[2]), imgOut);
```

Esta función, ***void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst),*** devuelve una imagen filtrada con la zona que está dentro de los rangos *lowerb* y *upperb*.

La señal es muy parecida a la que se veía en el filtrado *Threshold* que se ha visto cuando se estaba detectando la línea, por tanto, para mejorar la detección se puede filtrar de la misma manera:

```
// Se crean las estructuras que se utilizarán para erosionar y dilatar las imágenes
// Para la erosión se utiliza un rectangulo de 3x3
cv::Mat erodeElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
// Para la dilatación, se utiliza un rectángulo más grande, para que se
// vea mejor
cv::Mat dilateElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(8, 8));

// Se aplica dos veces los filtros para que la imagen sea lo mejor posible.
cv::erode(imgOut, imgOut, erodeElement);
cv::erode(imgOut, imgOut, erodeElement);

cv::dilate(imgOut, imgOut, dilateElement);
cv::dilate(imgOut, imgOut, dilateElement);
```

Se filtra dos veces para mejorar aún más el filtrado. Como el color detectado se puede interpretar como un área, se puede utilizar el mismo algoritmo de búsqueda de contornos utilizado en la detección de línea.

```
// Se crean las variables para almacenar los contornos
std::vector< std::vector<cv::Point> > contornos;
std::vector<cv::Vec4i> jerarquia;

// Variable para guardar el dato del area máxima
float largeArea = 0.0f;

// Se buscan todos los contornos en la imagen
cv::findContours(temp, contornos, jerarquia, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);

// Se buscan contornos que cumplan las condiciones
if(jerarquia.size() > 0)
{
        // Se verifica que no hay muchos objetos detectados
        if(jerarquia.size() < MAX_NUM_OBJECTS)
        {
            for(int i = 0; i < contornos.size(); i++)
            {
                // se detecta si hay una area
                float area = cv::contourArea(contornos[i]);
                //std::cout << "El tamaño es = " << area << std::endl;

                // Se consigue el mayor contorno
                if(area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > largeArea)
                {
                    // Se ha detectado un contorno, por tanto se cambia el estado
                    objectFound = true;
```

```
                    // se ajustan los valores x e y del centro del contorno
                    cv::Moments mu = cv::moments(contornos[i], false);
                    xPointDecObj = mu.m10 / area;
                    yPointDecObj = mu.m01 / area;
                    largeArea = area;
                }
            }
        }
        else
        {
            // se pone una texto en la imagen indicando que hay mucho ruido
            putText(cameraIn, "TOO MUCH NOISE! ADJUST FILTER", cv::Point(0, 50), 1, 2, cv::Scalar(0, 0,
255), 2);
        }
    }
```

De la misma manera que en la detección de línea, si hay un área con color, se detectará su centro, y por tanto, su posición.

Para detectar las bifurcaciones, se ha ideado un sistema que cuando detecte una zona de color, en este caso Verde será el color que indicará las bifurcaciones, el robot escogerá valores de la línea que se aproximen a ese punto, ignorando los que estén más lejos de este punto. En el momento que ya no detecte el color, seguirá detectando la línea de la misma manera que hasta entonces.

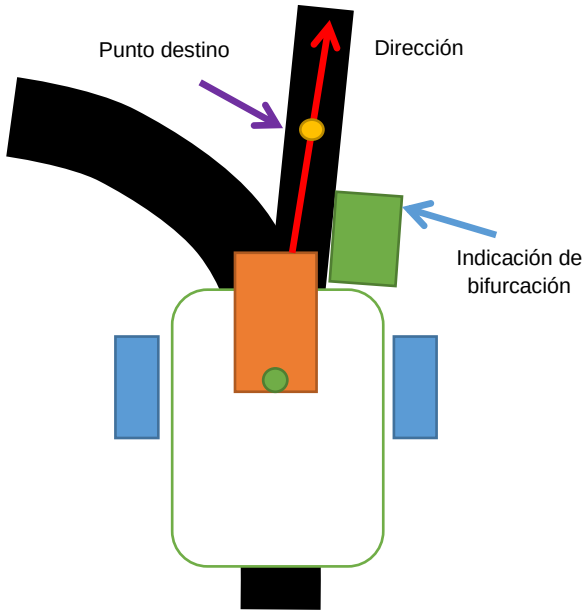

Figura 75.- Diagrama de detección de bifurcación

Lo primero que hay que definir es el rango del verde:

```
// Color Verde, asociado a una bifurcación
const int HSVMIN_GREEN[3] = {15, 77, 80};
const int HSVMAX_GREEN[3] = {35, 220, 215};
```

Cuando se detecta el área con el color verde, se obtienen unas coordenadas. Estas coordenadas se pasarán a la función de detección de línea, que las utilizará para establecer cuáles son los puntos que están dentro de una línea

```
// Se buscan los contornos de las tres imagenes.
for (int j = 0; j < 3; j++)
{
        // Defino las variables para guardar los contornos
```

```
        std::vector<std::vector<cv::Point> > contours;
        std::vector<cv::Vec4i> hierarchy;
        // Se ejecuta la función para detectar contornos
        cv::findContours(crop[j], contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE,
cv::Point(0, 0));
        for (size_t i = 0; i < contours.size(); i++)
        {
            float area = cv::contourArea(contours[i]);
            if (area > 1500)// Si el area es mayor que 1500, es una línea
            {
                cv::Moments mu;
                mu = moments(contours[i], false);
                int xtempPoint;
                int ytempPoint;
                // Se crean dos variables para guardar los datos del punto central
                xtempPoint =  mu.m10 / mu.m00;
                ytempPoint = (mu.m01 / mu.m00) + roi[j].y;
                // Si se ha detectado una bifurcación
                if(colorDetect)
                {
                    // Si solo se ha detectado un punto, se cogen estos valores como correctos
                    if(!bAreaDetected[j])
                    {
                        xPoint[j] = xtempPoint;
                        yPoint[j] = ytempPoint;
                        // Apunto que ha habido una detección de área
                        bAreaDetected[j] = true;
                        bArea[j] = true;
                        val[j].x = xPoint[j];
                        val[j].y = yPoint[j];
                    }
                    else
                    {
                       // Si el punto detectado esta más cerca de la bifurcación que el anterior,
                       // se selecciona este
                        if(abs(xtempPoint - colorPoint.x) < abs(xPoint[j] - colorPoint.x))
                        {
                            xPoint[j] = xtempPoint;
                            yPoint[j] = ytempPoint;
                            // Apunto que ha habido una detección de area
                            bAreaDetected[j] = true;
                            bArea[j] = true;
                            val[j].x = xPoint[j];
                            val[j].y = yPoint[j];
                        }
                    }
                }
                else
                {
                   // Si no se ha detectado color, se funciona igual
                    xPoint[j] = xtempPoint;
                    yPoint[j] = ytempPoint;
                    // Apunto que ha habido una detección de area
                    bAreaDetected[j] = true;
                    bArea[j] = true;
                    val[j].x = xPoint[j];
                    val[j].y = yPoint[j];
                }
            }
        }
        // Se pinta el centro
        cv::Point2f center(xPoint[j], yPoint[j]);
        // Dibujo un circulo basado en el centro
        cv::circle(cameraIn, center, 5, cv::Scalar(0, 255, 0), -1, 8, 0);
        // Dibujo el rectangulo
        cv::rectangle(cameraIn, cv::Point(xPoint[j] - 40, yPoint[j] - 20),
        cv::Point(xPoint[j] + 40, yPoint[j] + 20), cv::Scalar(0, 0, 255), 1, 8, 0);
    }
```

En el código lo que se hace es detectar todas las áreas donde hay línea. Y una vez detectadas, si se ha detectado una bifurcación (la variable *colorDetect*) es verdadera, se escoge el valor que está más cerca en X del valor de la bifurcación. Asi se consigue que el robot seleccione como valores correctos los que están cerca de la bifurcación.

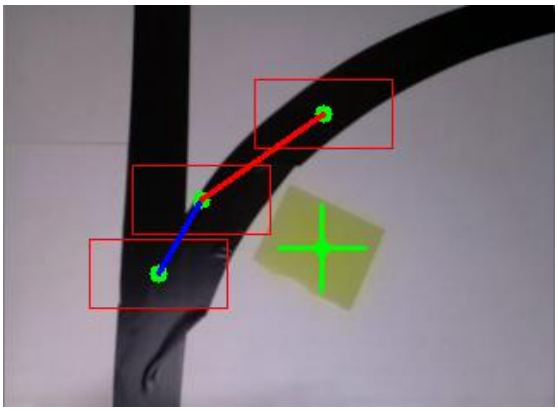

Figura 76.- Captura con bifurcación a la derecha

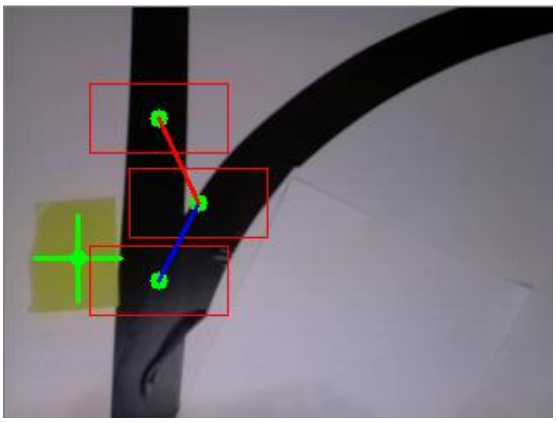

Figura 77.- Captura con bifurcación a la izquierda

- Detección de final de circuito:

Después de probar que la detección de color funcionaba, y que el código era capaz de manejar el giro utilizando los indicadores de color, se decidió que se podría añadir una detección de color extra, en este caso el Rojo, para detectar el final del circuito, y por tanto, que el robot se parase.

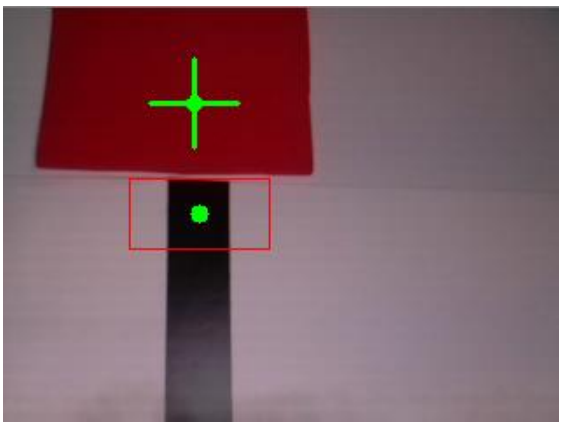

Figura 78.- Detección de final de línea

- Detección de línea:

Como se fue viendo que la detección de color era muy optima, y eliminaba problemas con las sombras del robot, en vez de realizar un filtro de ***threshold*** para detectar la línea, se ha implementado una detección de color negro, y ha mejorado bastante la respuesta e inmunidad a pequeñas sombras.

- Perdida de línea:

Con la detección de final con la detección de un color, se quiso solucionar un problema que aparecía cuando el robot iba muy rápido, que era que perdía la línea ya que no giraba a la suficiente velocidad o porque reaccionaba tarde en algunos casos. Para evitar que el robot se quede colgado, se añadió una funcionalidad para que volviese marcha atrás hasta encontrar la línea de nuevo.

- Problemas con luz:

Cuando se estuvieron haciendo pruebas, se comenzó a ver que había problemas con la luz. Dependiendo de la luz ambiental o que incedía en el robot, este podía detectar sombras, o no saber distinguir los colores. Por ello, y se decidió integrar en el diseño una fuente de luz estable, y así poder eliminar la variable incontrolada de la luz ambiental. Para realizar esta iluminación se utilizó un módulo de LEDs RGB de Adafruit, el Adafruit NeoPixel.

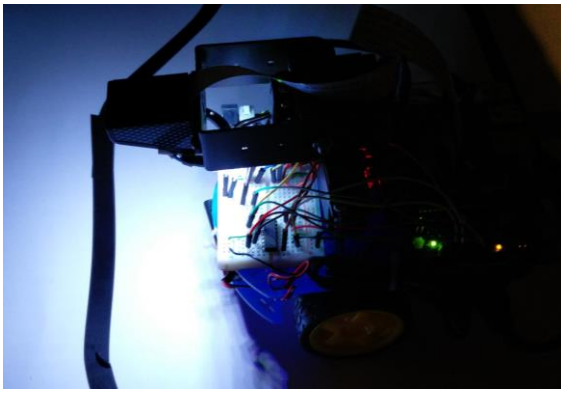

Figura 79.- Robot con los LEDs encendidos

- Diagrama de flujo final:

Con todas estas mejoras, se replanteo el diagrama de flujo del sistema, que quedo como diagrama de flujo final.

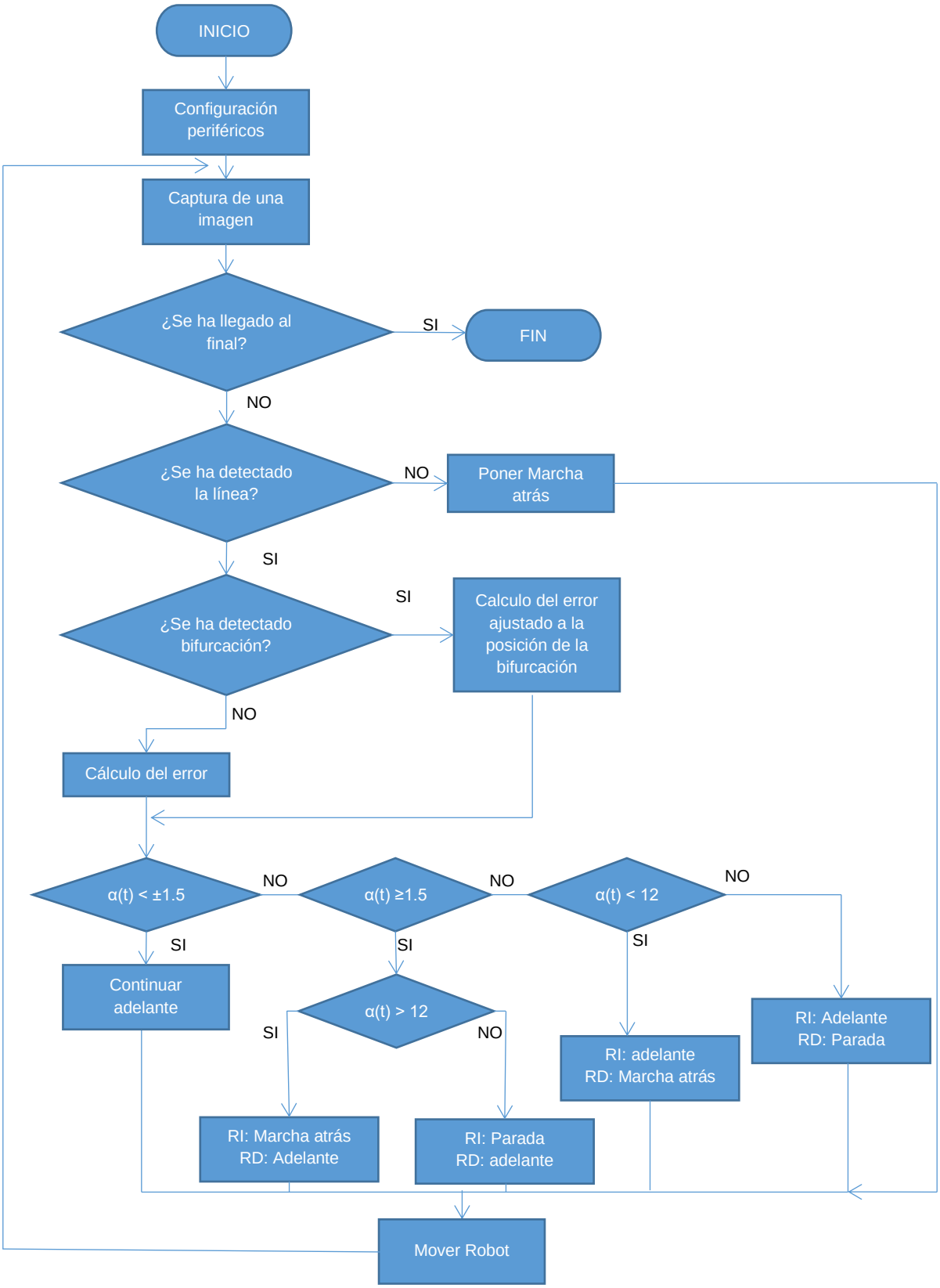

Figura 80.- Diagrama de flujo del PID

5. Conclusiones

El diseño se dio por concluido al incoporar la detección de colores, aunque la funcionalidad del robot pudiera seguir creciendo dada su flexibilidad. Por ejemplo, se había pensado introducir nuevos sensores, como de distancia, o hasta un LIDAR para intentar dar más autonomía al robot, pero la complejidad y el tiempo para las pruebas lo hicieron inviable.

Dentro de las mejoras que se podrían realizar están las siguientes:

- Añadir un control mejor y más estable, de manera que se pueda acelerar la velocidad.
- Intentar utilizar unos motores mejores, ya que los que se utilizaron eran muy lento, y su respuesta no era muy buena.
- Mejorar la comunicación entre tarjetas, utilizando, por ejemplo, el cable USB que se utiliza para alimentar el Arduino, o quizás un SPI, que puede ser más rápido que un bus I2C.
- Implementar funciones de detección de patrones, para poder indicar al robot los sitios a los que se puede mover, que hacer en cada sitio, etc.
- Añadir funcionalidades de machine learning, para que el robot sea capaz de detectar objetos, y sepa reaccionar. Por ejemplo, se pueden añadir Stops, semáforos, etc, y que el robot sepa que hacer dependiendo de la señal, o del color del semáforo.

Así que como se puede ver, hay muchas cosas para poder integrar en el robot. La detección de la línea no deja de ser la punta del iceberg de todo lo que puede hacer un robot con un módulo de visión artificial y los sensores adecuados. Realmente este proyecto no deja de ser una base de partida para poder llegar a realizar un robot autónomo, que funcione de manera similar a otros vehículos autónomos.

6. Apendice

6.1 Código Raspberry

- main.cpp: Fichero principal.

```
/*
*
*   Main.cpp- Fichero principal del sistema
*
*
*   Realizado por David Ortiz 2016
*/

// Librerias para incluir
#include <iostream>
#include <vector>
#include <unistd.h>

// Includes para el PID
#include <ctime>
#include <cmath>

// Includes de los ficheros del proyecto
#include "lineDetect.h"
#include "variables.h"
#include "arduinoI2C.h"
#include "objectDetect.h"
#include "peripherals.h"

// Bloque principal
int main(int argc, char **argv)
{
    // Variable para ajustar la detección de area
    bool bAreaDetected;

    // Variable para guardar el puntero al fichero I2C.
    int i2cFile;

    // Variables para controlar la dirección por si pierde la línea
    bool bAtras = false;

    // Se crean los Mats para guardar los datos
    // Para guardar los datos de la cámara
    cv::Mat camImg;
    // Para guardar la imagen en HSV
    cv::Mat imgHSV;
    // para guardar la versión binaria de la imagen
    cv::Mat imgThres;

    // Variable para almacenar la posición del centro del area del color
    cv::Point turnPoint;

    // Variable para guardar el periodo, que variará dependiendo de lo que tarde en ejecutar el código
    float samplePeriod = 0.03f;

    // Variables para el cálculo del tiempo
    clock_t reloj;


    // Se incializa los valores de pitch y yaw
    int pitch = 25; // Colocado un poco hacia abajo, mirando al suelo.
    int yaw = 90; // Colocado en el centro de giro

    // Se inicializa el I2C
    i2cFile = i2cInit();

    // Se le envían los valores por defecto para inicializar
    // Se envía el valor por defecto del giro
    i2cWriteYaw(yaw, i2cFile);
    // Se envía el valor por defecto de la inclinación
    i2cWritePitch(pitch, i2cFile);

    // Se incializan los valores de los motores
    int leftDirection = 1; // Rueda izquierda adelante
    int rightDirection = 1; // Rueda derecha adelante
    int leftSpeed = 0; // Velocidad 0
    int rightSpeed = 0; // Velocidad 0

    // Pruebas de velocidades

    // Se inicializa la velocidad del motor a 0
    i2cWriteMotor(leftDirection, rightDirection, leftSpeed, rightSpeed, i2cFile);

    // Se activa el flash Enciende y intensidad del 100
    i2cWriteFlash(1, 100, i2cFile);

    // Variables para el control, para combinar entre los tres puntos de detección
    bool bPointDetected;

    // Se inicializa el socket
```

```
if(socketInit() != 0)
{
    std::cout << "Fallo al crear el Socket!!!\n";
    exit(0); // Se sale del programa porque no se ha podido incializar el socket
}

// Se inicializa la cámara
if(!cameraInit())
{
    std::cout << "Fallo al inicializar la camara!!!\n";
    exit(0); // Se sale del programa porque no se ha podido inicializar la cámara
}

// Se realiza una captura de imagen para calcular el tamaño que se enviará en el socket
cameraCapture(camImg);
// Se preparan los datos
int  imgSize = camImg.total()*camImg.elemSize();

// Se inicializa la parte de los PWM's
pwmInit();

// Se incializa el detector de línea
initLineDetector();

while(1)
{
    // Se adquieren los datos de la cámara
    cameraCapture(camImg);

    // Variable para los datos enviados
    int bytes = 0;

    // Se captura el tiempo de inicio del bucle
    reloj = clock();

    // Se pasa la imagen de RGB a HSV
    cv::cvtColor(camImg, imgHSV, cv::COLOR_RGB2HSV);

    // Se pasa la imagen de BGR a RGB, para enviar el código al programa de recepción de datos
    cv::cvtColor(camImg, camImg, cv::COLOR_BGR2RGB);

    // Se efectuan las operaciones de objetos de color amarillo.

    // Se filtra la imagen en el rango HSV y suavizando la imagen
    imageFilter(imgHSV, imgThres, GREEN_COLOR); // filtrando para el color verde.

    // Se detecta la posición del objeto de giro
    bool turnDetection = detectObject(imgThres, camImg, bAreaDetected, turnPoint);

    // Se vuelve a filtrar la imagen, para detectar el cólor rojo
    imageFilter(imgHSV, imgThres, RED_COLOR);
    // Se detecta si se ha llegado al final
    cv::Point temp;
    bAreaDetected = false;
    bool detected = detectObject(imgThres, camImg, bAreaDetected, temp);

    // Se inicializan los valores
    bPointDetected = false;

    // Variable para anotar si ha habido o no una detección
    bool bAreaDetec[3];
    // Puntos para la detección de la línea
    cv::Point mp[3];
    for(int i = 0; i < 3; i++)
    {
        bAreaDetec[i] = false;
        mp[i].x = 0;
        mp[i].y = 0;
    }

    // Se detecta la línea.
    lineDetect(camImg, bPointDetected, bAreaDetec, mp, turnDetection, turnPoint);

    // Se efectua el control de la línea.
    if(bPointDetected && !bAreaDetected)
    {
        if(bAtras)
        {
            // Si se viene de marcha atrás, se vuelve a colocar el robot moviendo hacia adelante
            bAtras = false;
            leftDirection = 1;
            rightDirection = 1;
            i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
        }
        lineControl(leftDirection, rightDirection, leftSpeed, rightSpeed, samplePeriod, i2cFile);
        // Se mueven los servos
        pwmSetVelocity(leftSpeed, rightSpeed);
    }
    else if(bAreaDetected)
    {
        pwmSetVelocity(0, 0);
    }
    else
    {
        if(!bAtras)
        {
            // Si se ha salido del camino, tira marcha atrás hasta que vuelve a cogerlo.
            bAtras = true;
            leftDirection = 0;
            rightDirection = 0;
            i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
```

```
            }
            // Se busca otra vez la línea yendo marcha atrás
            pwmSetVelocity(400, 400);
        }

        // Se envía la imagen por el socket
        if(!socketSend(camImg, imgSize))
        {
            break;
        }

        reloj = clock() - reloj;

        // Actualizo el valor del periodo de muestreo
        samplePeriod = static_cast<float>(reloj) / CLOCKS_PER_SEC;
    }

    puts("Se ha salido del bucle");

    // Se paran los motores.
    pwmSetVelocity(0, 0);

    // Se activa se apaga el flash, y se pone la intensidad al 1
    i2cWriteFlash(0, 25, i2cFile);

    cameraRelease();

    // cierro el fichero I2C
    close(i2cFile);

    // Se cierra el socket y se limpia todo.
    closeSocket();

    // Salgo del programa
          return 0;
}
```

- variables.h : Conjunto de variables globales

```
/*
*
*   variables.h.-   Fichero donde se guardarán todas la constantes y variables globales
*                                                 que se utilizarán entre procesos.
*
*
*   Realizado por David Ortiz 2016
*/

#ifndef VARIABLES_H
#define VARIABLES_H

// Includes del OpenCV
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/core/core_c.h>

// Constantes

// Variables para detectar el color del posit, en código HSV

const int H_MIN = 0;
const int H_MAX = 10;
const int S_MIN = 50;
const int S_MAX = 100;
const int V_MIN = 100;
const int V_MAX = 255;

// Color Rojo, asociado al Final
const int HSVMIN_RED[3] = {150, 100, 100};
const int HSVMAX_RED[3] = {180, 255, 255};

// Color Verde, asociado a una bifurcación
const int HSVMIN_GREEN[3] = {15, 77, 80};
const int HSVMAX_GREEN[3] = {35, 220, 215};

// Color negro, asociado a la línea
const int HSVMIN_BLACK[3] = {0, 0, 0};
const int HSVMAX_BLACK[3] = {180, 255, 30};

// Variables variar la detección de colores
const int GREEN_COLOR = 0;
const int RED_COLOR = 1;
const int BLACK_COLOR = 2;

// Defino variables matemáticas
const float PI = 3.14159265f;

// Valores constantes del tamaño de alto y ancho de captura de la cámara
const int CAMERA_WIDTH = 320;
const int CAMERA_HEIGHT = 240;

// Numero máximo de objetos detectados en una imagen
const int MAX_NUM_OBJECTS = 50;

// Tamaño máximo y mínimo de area a encontrar
const int MIN_OBJECT_AREA = 10 * 10;
const int MAX_OBJECT_AREA = CAMERA_HEIGHT*CAMERA_WIDTH / 1.5;

// Constantes para el bus I2C
// The Arduino address
#define ADDRESS 0x04

// The I2C bus: This is for V2 pi's. For V1 Model B you need i2c-0
static const char *devName = "/dev/i2c-1";

// Velocidad media del motor
const int motorSpeed = 500;

// Constantes de configuración del PWM de control del motor
const int PWM_SET_CLOCK = 384;
const int PWM_SET_RANGE = 1000;

// Pins para los motores
const int LEFT_MOTOR_PIN = 1;
const int RIGHT_MOTOR_PIN = 24;

// Variables de control para el control del motor
const float kMotorProp = 6.0f; // Ganancia proporcional
const float kMotorInt = 0.1f; // Ganancia integral
const float kMotorDeriv = 0.3f; // Ganancia derivativa

// Variable para el control de la máxima ganancia
const float maxCorrection = 300.f;

#endif // HEADER_H
```

- lineDetect.cpp: Funciones de detección de línea

```
/*
*
*   lineDetect.cpp.-  Funciones para detectar las líneas.
*
*
*   Realizado por David Ortiz 2016
*/

#include "lineDetect.h"

// Como habrá transmisiones I2C, se incluye tambien las funciones del arduino
#include "arduinoI2C.h"

#include <cmath>

// Variables locales
std::vector<cv::Rect> roi; // Region of interest

// Se crea un vector con Mats para los fichero crop
std::vector<cv::Mat> crop;

// Variables para la posicion
int xPointDec;
int yPointDec;

// Variables para el control
float angleDiff= 0.0f;

// Variable para los errores
float outputDegree = 0.0f;
float errorAntDegre = 0.0f;
float integralError = 0.0f;

// Función para inicalizar las region of interest
void initLineDetector()
{
    // Se incializan las areas de interés
    // Los puntos estarán en 140, 90 y 40.
    for (int i = 0; i < 3; i++)
    {
        cv::Rect temp;
        temp.width = CAMERA_WIDTH;
        temp.height = 50;
        temp.x = 0;
        temp.y = ((CAMERA_HEIGHT) - 100) - 50 * i; // Distancia entre puntos
        roi.push_back(temp);
    }

}

// Función detectar la línea
void lineDetect(cv::Mat &cameraIn, bool &lineAreaDetect, bool bArea[], cv::Point *val, bool colorDetect, cv::Point
colorPoint)
{
    // Se crea una matriz para guardar las coordinadas de los puntos y poder dibujar luego las líneas
    float xPoint[3];
    float yPoint[3];

    // Variable para anotar si ha habido o no una detección
    bool bAreaDetected[3];

    // Se vacian las imagenes previas
    crop.clear();

    // Utilizo la función de crear una imagen teniendo en cuenta el tamaño establecido por el rectangulo
    // que se ha definido como borde
    // Se crea una función para crear las imagenes dependiendo del rectangulo, y hago todos las operaciones
    // para preparar la imagen.
    for (int i = 0; i < 3; i++)
    {
        // Creo una imagen con la primera region of interest
        cv::Mat tempCrop(cameraIn, roi[i]);

        // Se detecta el color negro
        cv::inRange(tempCrop, cv::Scalar(HSVMIN_BLACK[0], HSVMIN_BLACK[1], HSVMIN_BLACK[2]), cv::Scalar(HSVMAX_BLACK[0],
HSVMAX_BLACK[1], HSVMAX_BLACK[2]), tempCrop);

        // Realizo el filtrado de erosión y dilatación
        // Defino las estructuras de erosión y dilatación
        cv::Mat erodeElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
        cv::Mat dilateElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(8, 8));

        // Filtro de erosión
        cv::erode(tempCrop, tempCrop, erodeElement);
        cv::erode(tempCrop, tempCrop, erodeElement);
        // Filtro de dilatación
        cv::dilate(tempCrop, tempCrop, dilateElement);
        cv::dilate(tempCrop, tempCrop, dilateElement);

        crop.push_back(tempCrop);
    }

    // Se inicializan las variables
    for(int i = 0; i < 3; i ++)
    {
        xPoint[i] = 0.0f;
        yPoint[i] = 0.0f;
        bAreaDetected[i] = false;
```

```
}

// Se buscan los contornos de las tres imagenes.
for (int j = 0; j < 3; j++)
{
    // Se definen las variables para guardar los contornos
    std::vector<std::vector<cv::Point> > contours;
    std::vector<cv::Vec4i> hierarchy;
    // Se ejecuta la función para detectar contornos
    cv::findContours(crop[j], contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, cv::Point(0, 0));
    // Se dibuja el contorno encontrado
    for (size_t i = 0; i < contours.size(); i++)
    {
        float area = cv::contourArea(contours[i]);
        if (area > 1500)// Si el area es mayor que 1500, Se considera que se ha detectado una línea.
        {
            cv::Moments mu;
            mu = moments(contours[i], false);
            int xtempPoint;
            int ytempPoint;
            // Se crean dos variables para guadar los datos del punto central
            xtempPoint =  mu.m10 / mu.m00;
            ytempPoint = (mu.m01 / mu.m00) + roi[j].y;
            if(colorDetect)
            {
                if(!bAreaDetected[j])
                {
                    // Si solo se ha detectado un punto, se cogen estos valores como correctos
                    xPoint[j] = xtempPoint;
                    yPoint[j] = ytempPoint;
                    // Se apunta que ha habido una detección de area
                    bAreaDetected[j] = true;
                    bArea[j] = true;
                    val[j].x = xPoint[j];
                    val[j].y = yPoint[j];
                }
                else
                {
                    if(abs(xtempPoint - colorPoint.x) < abs(xPoint[j] - colorPoint.x))
                    {
                        // Si el punto detectado esta más cerca de la bifurcación que el anterior,
                        // se selecciona este
                        xPoint[j] = xtempPoint;
                        yPoint[j] = ytempPoint;
                        // Se apunta que ha habido una detección de area
                        bAreaDetected[j] = true;
                        bArea[j] = true;
                        val[j].x = xPoint[j];
                        val[j].y = yPoint[j];
                    }
                }
            }
            else
            {
                // Si no se ha detectado color, se funciona igual
                xPoint[j] = xtempPoint;
                yPoint[j] = ytempPoint;
                // Se apunta que ha habido una detección de area
                bAreaDetected[j] = true;
                bArea[j] = true;
                val[j].x = xPoint[j];
                val[j].y = yPoint[j];
            }
        }
    }
    // Se pinta el centro si se ha detectado
    if(bAreaDetected[j])
    {
        cv::Point2f center(xPoint[j], yPoint[j]);
        // Se dibuja un circulo basado en el centro
        cv::circle(cameraIn, center, 5, cv::Scalar(0, 255, 0), -1, 8, 0);
        // Se dibuja el rectangulo
        cv::rectangle(cameraIn, cv::Point(xPoint[j] - 40, yPoint[j] - 20),
        cv::Point(xPoint[j] + 40, yPoint[j] + 20), cv::Scalar(0, 0, 255), 1, 8, 0);
    }
}
// Se dibujan las líneas si existen los puntos
if(bAreaDetected[0] && bAreaDetected[1])
{
    cv::line(cameraIn, cv::Point(xPoint[0], yPoint[0]), cv::Point(xPoint[1], yPoint[1]), cv::Scalar(255, 0, 0), 2, 8, 0);
}

if(bAreaDetected[1] && bAreaDetected[2])
{
    cv::line(cameraIn, cv::Point(xPoint[1], yPoint[1]), cv::Point(xPoint[2], yPoint[2]), cv::Scalar(0, 0, 255), 2, 8, 0);
}

// Se ajusta si se han detectado alguno de los puntos para efectuar el control con el
if(bAreaDetected[2])
{
    // Si se ha detectado un area en el punto superior, este
    // Sera prioritario
    lineAreaDetect = true;
    xPointDec = xPoint[2];
    yPointDec = yPoint[2];
}
else if(bAreaDetected[1])
{
    // El segundo en prioridad será el punto central
    lineAreaDetect = true;
    xPointDec = xPoint[1];
```

```
        yPointDec = yPoint[1];
    }
    else if(bAreaDetected[0])
    {
        // Por último esta el punto inferior
        lineAreaDetect = true;
        xPointDec = xPoint[0];
        yPointDec = yPoint[0];
    }
    else
    {
        lineAreaDetect = false;
        xPointDec = 0;
        yPointDec = 0;
    }

}

// Función para establecer el movimiento del robot
float lineControl(int &leftDirection, int &rightDirection, int &leftSpeed, int &rightSpeed, float &samplePeriod, int i2cFile)
{
    // Se calcula la arcotangente entre los dos puntos
    angleDiff = 90 - atan2(static_cast<float>((CAMERA_HEIGHT - yPointDec) + 210), static_cast<float>((CAMERA_WIDTH / 2) -
xPointDec)) * 180 / PI;
    // Error integral
    integralError = integralError + angleDiff;
    // Calculo de la corrección - PID
    outputDegree = kMotorProp*angleDiff + (kMotorDeriv*(angleDiff - errorAntDegre)) + kMotorInt*integralError;
    std::cout << "El error angulo es: "  << angleDiff << " y el error " << outputDegree << std::endl;
    errorAntDegre = angleDiff; // Guardo el error anterior para el control diferencial.
    // Calculo los nuevos errores
    if(angleDiff > 1.5f)
    {
        // Se para la rueda izquierda y dejo en funcionamiento la derecha
        if(abs(outputDegree) >= maxCorrection)
        {
            rightSpeed = motorSpeed + maxCorrection;
        }
        else
        {
            rightSpeed = motorSpeed + outputDegree;
        }
        if(angleDiff > 12.0f)
        {
            // Si hay mucho error, se pone la rueda contraria marcha atrás
            if(leftDirection == 1)
            {
                rightDirection = 1;
                leftDirection = 0;
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            leftSpeed = 300;
        }
        else
        {
            // Si no se sigue hacia adelante
            if(leftDirection == 0)
            {
                rightDirection = 1;
                leftDirection = 1;
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            leftSpeed = 200;
        }
    }
    else if(angleDiff < -1.5f)
    {
        // Se para la rueda derecha y dejo en funcionamiento la izquierda
        if(abs(outputDegree) >= maxCorrection)
        {
            leftSpeed = motorSpeed + maxCorrection;
        }
        else
        {
            leftSpeed = motorSpeed - outputDegree;
        }
        if(angleDiff < -12.0f)
        {
            // Si hay mucho error, se pone la rueda contraria marcha atrás
            if(rightDirection == 1)
            {
                rightDirection = 0;
                leftDirection = 1;
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            rightSpeed = 300;
        }
        else
        {
            // Si no se sigue hacia adelante
            if(rightDirection == 0)
            {
                rightDirection = 1;
                leftDirection = 1;
                i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
            }
            rightSpeed = 200;
        }
    }
    else
    {
```

```
        // Se colocan los motores hacia adelante por si hay algún problema
        if(rightDirection == 0 || leftDirection == 0)
        {
            rightDirection = 1;
            leftDirection = 1;
            i2cWriteMotor(leftDirection, rightDirection, 0, 0, i2cFile);
        }
        // Se avanza sin corrección
        leftSpeed = (motorSpeed - 100) - outputDegree/2;
        rightSpeed = (motorSpeed - 100) + outputDegree/2;

    }

    // Se retorna el angulo de corrección
    return angleDiff;
}
```

- objectDetect.cpp: Función para la detección de objetos.

```
/*
*
*  objectDetect.cpp.-         Código de las funciones para detectar los objetos.
*
*
*   Realizado por David Ortiz 2016
*/

#include "objectDetect.h"

// Como habrá transmisiones I2C, se incluye tambien las funciones del arduino
#include "arduinoI2C.h"

#include <cmath>

// Variables para la posicion
int xPointDecObj;
int yPointDecObj;

// Variables para el control
float angleDiffObj = 0.0f;

// Variable para los errores
float outputDegreeObj = 0.0f;
float errorAntDegreeObj = 0.0f;

// Función para filtrar la imagen
void imageFilter(cv::Mat &imgIn, cv::Mat &imgOut, int color)
{
    //Se filtra la imagen para el color escogido
    if(color == GREEN_COLOR)
    {
          cv::inRange(imgIn, cv::Scalar(HSVMIN_GREEN[0], HSVMIN_GREEN[1], HSVMIN_GREEN[2]), cv::Scalar(HSVMAX_GREEN[0],
HSVMAX_GREEN[1], HSVMAX_GREEN[2]), imgOut);
    }
    else if(color == RED_COLOR)
    {
          cv::inRange(imgIn, cv::Scalar(HSVMIN_RED[0], HSVMIN_RED[1], HSVMIN_RED[2]), cv::Scalar(HSVMAX_RED[0],
HSVMAX_RED[1], HSVMAX_RED[2]), imgOut);
    }
    else if(color == BLACK_COLOR)
    {
          cv::inRange(imgIn, cv::Scalar(HSVMIN_BLACK[0], HSVMIN_BLACK[1], HSVMIN_BLACK[2]), cv::Scalar(HSVMAX_BLACK[0],
HSVMAX_BLACK[1], HSVMAX_BLACK[2]), imgOut);
    }
    else
    {
          // El evento else será para el color rosa
          cv::inRange(imgIn, cv::Scalar(H_MIN, S_MIN, V_MIN), cv::Scalar(H_MAX, S_MAX, V_MAX), imgOut);
    }

    // Se crean las estructuras que se utilizarán para erosionar y dilatar las imagenes
    // Para la erosión se utiliza un rectangulo de 3x3
    cv::Mat erodeElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
    // Para la dilatación, se utiliza un rectangulo más grande, para que se
    // vea mejor
    cv::Mat dilateElement = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(8, 8));

    // Se aplica dos veces los filtros para que al imagen que de lo mejor posible.
    cv::erode(imgOut, imgOut, erodeElement);
    cv::erode(imgOut, imgOut, erodeElement);

    cv::dilate(imgOut, imgOut, dilateElement);
    cv::dilate(imgOut, imgOut, dilateElement);
}

// Función para detectar el objeto
bool detectObject(cv::Mat thres, cv::Mat &cameraIn, bool &areaDetec, cv::Point &dp)
{
    // Se crea una copia del fichero filtrado
    cv::Mat temp;
    thres.copyTo(temp);

    // Se crean las variables para almacenar los contornos
    std::vector< std::vector<cv::Point> > contornos;
    std::vector<cv::Vec4i> jerarquia;

    //Variable para detectar los contornos
    bool objectFound = false;

    // Variable para guardar el dato del area máxima
    float largeArea = 0.0f;

    // Se buscan todos los contornos en la imagen
    cv::findContours(temp, contornos, jerarquia, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE);

    // Se buscan contornos que cumplan las condiciones
    if(jerarquia.size() > 0)
    {
        // Se verifica que no hay muchos objetos detectados
        if(jerarquia.size() < MAX_NUM_OBJECTS)
        {
            for(int i = 0; i < contornos.size(); i++)
            {
                // se detecta si hay una area
                float area = cv::contourArea(contornos[i]);
                //std::cout << "El tamaño es = " << area << std::endl;
```

```
                // Se consigue el mayor contorno
                if(area > MIN_OBJECT_AREA && area < MAX_OBJECT_AREA && area > largeArea)
                {
                    // Se ha detectado un contorno, por tanto se cambia el estado
                    objectFound = true;

                    // se ajustan los valores x e y del centro del contorno
                    cv::Moments mu = cv::moments(contornos[i], false);
                    xPointDecObj = mu.m10 / area;
                    yPointDecObj = mu.m01 / area;
                    largeArea = area;
                }
            }
        }
        else
        {
            // se pone una texto en la imagen indicando que hay mucho ruido
            putText(cameraIn, "TOO MUCH NOISE! ADJUST FILTER", cv::Point(0, 50), 1, 2, cv::Scalar(0, 0, 255), 2);
        }
    }

    // Si se ha encontrado un contorno, se pone un texto en pantalla
    if (objectFound)
    {
        // Se ha detectado un contorno.
        // Se pinta el centro de la imagen
        drawCenter(static_cast<float>(xPointDecObj), static_cast<float>(yPointDecObj), cameraIn, cv::Scalar(0,255,0));
        // Se actualiza la posición donde esta el area
        dp.x = xPointDecObj;
        dp.y = yPointDecObj;
    }

    if(largeArea >= 10000.0f)
    {
        // Se ha detectado un area correcta
        areaDetec = true;
    }
    else
    {
        areaDetec = false;
    }

    return objectFound;
}

// Función para pintar el centro del contorno
void drawCenter(float x, float y, cv::Mat &cameraIn, cv::Scalar color)
{
    // Se dibuja un circulo central
    cv::circle(cameraIn, cv::Point(x, y), 3,  color, 2);

    // Ahora se dibuja una cruz de imagenes.
    if (y - 25 > 0)
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x, y - 25), color, 2);
    }
    else
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x, 0), color, 2);
    }

    if (y + 25 < CAMERA_HEIGHT)
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x, y + 25), color, 2);
    }
    else
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x, CAMERA_HEIGHT), color, 2);
    }

    if (x - 25 > 0)
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x - 25, y), color, 2);
    }
    else
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(0, y), color, 2);
    }

    if (x + 25 < CAMERA_WIDTH)
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(x + 25, y), color, 2);
    }
    else
    {
        cv::line(cameraIn, cv::Point(x, y), cv::Point(CAMERA_WIDTH, y), color, 2);
    }

}
```

- periphreals.cpp : Código de configuración de los dispositivos del sistema

```
/*
*
*  peripherals.cpp.-          En esta función estarán todos los datos necesarios para ejecutar
*                        las funciones del PWM, utilizar la cámara y enviar datos con los threads.
*
*
*   Realizado por David Ortiz 2016
*/

#include "peripherals.h"

// Variable para guardar el socket
int thisSocket;
// Variable para guardar los datos de conexión
struct sockaddr_in server;
// Variables para el mensaje enviado, mensaje recibido y tamaño del mensaje recibido.
char *message, server_reply[2000];

// Send data here
int bytes = 0;

// Creo la variable de la cámara
raspicam::RaspiCam_Cv Camera;

// Función para inicializar el socket.
int socketInit()
{
    //Se crea el socket
    // La función socket lo que hace es crear un socket del estilo ipv4 con AF_INET, y
    // que es del tipo envío de datos con Sock_stream. 0 es para indicar que es TCP.
    if ((thisSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        //Si no se ha podido crear el socket, se cierra.
        printf("Could not create socket!!\n");
        if(thisSocket)
        {
            close(thisSocket);
        }
        exit(0); // Salgo del programa, si no hay socket, no funciona.
    }

    // Todo continua perfecto.
    printf("Socket created.\n");

    // Ahora se crea la conexión en este caso voy a utilizar la conexión con google
    server.sin_addr.s_addr = inet_addr("192.168.1.43"); // direción portátil.
    // Se introduce el tipo de conexión inet
    server.sin_family = AF_INET;
    // El socket se conecta al puerto 8888, uno especifico para trabajar los dos dispositivos.
    server.sin_port = htons(8888);

    // Se conecta al servidor remoto
    if (connect(thisSocket, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        std::cout << "connect error\n";
        return 1;
    }

    std::cout << "Connected\n";

    return 0;
}

// Función para enviar una imagen
bool socketSend(cv::Mat camImg, int imgSize)
{
    if((bytes = send(thisSocket, camImg.data, imgSize, 0)) < 0)
    {
        return false;;
    }
    return true;
}

// Función para cerrar el socket
void closeSocket()
{
    close(thisSocket);
}


// Función para inicializar la cámara
bool cameraInit()
{
    // Se crea una variable mat para las lecturas dummy.
    cv::Mat img;
    int count = 5;

    // Se ajustan los parametros de la camara
    Camera.set( CV_CAP_PROP_FORMAT, CV_8UC3 ); // En color
    Camera.set(CV_CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH);
    Camera.set(CV_CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT);

    // Se verifica que la cámara se ha abierto
    if(!Camera.open())
    {
        std::cerr << "Error opening the camera" << std::endl;
        return false;
    }
```

```
    std::cout << "El tamaño de la captura es ancho: " << CAMERA_WIDTH;
    std::cout << " y el alto es:" << CAMERA_HEIGHT << std::endl;

    // Inico que la cámara se ha abierto bien
    std::cout << "Camara abierta correctamente!!\n";

    // Para que la cámara funcione bien, se tiene que hacer unas lecturas dummy
    // En este caso voy a hacer 5, a ver que tal va. Si empiezo a leer de vacío, hay
    // algún problema de ajuste.
    for(int i = 0; i < count; i++)
    {
        // Adquiero los datos de la cámara
        Camera.grab();
        Camera.retrieve(img);
    }

    return true;
}

// Captura de una imagen
bool cameraCapture(cv::Mat &cam)
{
    // Se adquieren los datos de la cámara
    Camera.grab();
    Camera.retrieve(cam);
    return true;
}

// Función para liberar la camara
void cameraRelease()
{
    // Se libera la cámara
    Camera.release();
}

// Función para inicializar los PWM's de la Raspberry
bool pwmInit()
{
    // Se arranca el Wiring pi.
    wiringPiSetup();

    // Se inicializan los motores como salidas
    pinMode(LEFT_MOTOR_PIN,PWM_OUTPUT);
    pinMode(RIGHT_MOTOR_PIN,PWM_OUTPUT);

    // Los valores de los pwm serán  en ms
    pwmSetMode(PWM_MODE_MS);
    pwmSetClock(PWM_SET_CLOCK);
    pwmSetRange (PWM_SET_RANGE);

    // Se arrancan los motores asignando un valor de reposo.
    // Con 0 ms haces que el pwm esté siempre a 0
    pwmWrite (LEFT_MOTOR_PIN, 0);
    pwmWrite (RIGHT_MOTOR_PIN, 0);

    return true;
}

// Se ajusta la velocidad del PWM
void pwmSetVelocity(int leftSpeed, int rightSpeed)
{
    pwmWrite (LEFT_MOTOR_PIN, leftSpeed); // Motor izquierdo.
    pwmWrite (RIGHT_MOTOR_PIN, rightSpeed); // Motor derecho
}
```

- arduinoI2C.cpp: Código de envío de datos a Arduino.

```
/*
*
*  arduinoI2C.cpp.-          Fichero con todas la funciones para implementar las comunicaciones entre
*                                                    la Raspberry y el Arduino.
*
*
*   Realizado por David Ortiz 2016
*/

#include "arduinoI2C.h"

// Includes para controlar las comunicaciones con el arduino
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <ctime>
#include <iostream>
#include <unistd.h>

// Include a las variables globales del sistema
#include "variables.h"


// Función para inicializar el I2C
int i2cInit()
{
    int file;

    // Creo un puntero para el I2C
    if ((file = open(devName, O_RDWR)) < 0)
    {
        fprintf(stderr, "I2C: Failed to access %d\n", devName);
        exit(1);
    }

    printf("I2C: acquiring buss to 0x%x\n", ADDRESS);

    if (ioctl(file, I2C_SLAVE, ADDRESS) < 0)
    {
        fprintf(stderr, "I2C: Failed to acquire bus access/talk to slave 0x%x\n", ADDRESS);
        exit(1);
    }

    return file;
}

// Función para escribir el angulo de Giro
void i2cWriteYaw(int yawAngle, int file)
{
    unsigned char cmd[16];
    cmd[0] = 'Y'; // Se quiere modificar el angulo de giro
    write(file, cmd, 1);
    usleep(1000);
    // Se envía el valor a grabar en el motor.
    cmd[0] = yawAngle;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
}

// Función para escribir el angulo de inclinación
void i2cWritePitch(int pitchAngle, int file)
{
    unsigned char cmd[16];
    cmd[0] = 'P'; // Se quiere modificar el angulo de giro
    write(file, cmd, 1);
    usleep(1000);
    // Se envía el valor a grabar en el motor.
    cmd[0] = pitchAngle;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
}

// Función para mover los dos motores
void i2cWriteMotor(int leftDirection, int rightDirection, int leftSpeed, int rightSpeed, int file)
{
    unsigned char cmd[16];
    cmd[0] = 'M'; // Se quiere modificar la velocidad de los motores
    write(file, cmd, 1);
    usleep(1000);
    // Se envía la dirección de la rueda izquierda
    cmd[0] = rightDirection;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
    // Se envía la dirección de la rueda derecha
    cmd[0] = leftDirection;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
    // Se envía la velocidad de la rueda izquierda
    cmd[0] = leftSpeed;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
    // Se envía la velocidad de la rueda derecha
    cmd[0] = rightSpeed;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
```

```
}

// Función para modificar el estado del Flash
void i2cWriteFlash(int color, int bright, int file)
{
    unsigned char cmd[16];
    cmd[0] = 'F'; // Se quiere modificar el estado del flash
    write(file, cmd, 1);
    usleep(1000);
    // Se envía el color
    cmd[0] = color;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
    // Se envía la intensidad lúminica de los LEDS
    cmd[0] = bright;
    write(file, cmd, 1);
    usleep(1000); // El programa se espera 1 milisegundo
}
```

- CMake: Fichero de configuración para compilar.

```
#####################################
cmake_minimum_required (VERSION 2.8)
project (raspicam_test)
set(CMAKE_MODULE_PATH "/usr/local/lib/cmake/${CMAKE_MODULE_PATH}")

# Locate libraries and headers
find_package(Threads REQUIRED)

find_package(raspicam REQUIRED)
find_package(OpenCV)
IF  ( OpenCV_FOUND AND raspicam_CV_FOUND)
MESSAGE(STATUS "COMPILING OPENCV TESTS")

SET(CMAKE_CXX_FLAGS "-pthread -I/usr/local/include -L/usr/local/lib -lwiringPi")
# Create executable
add_executable (linebot main.cpp arduinoI2C.cpp objectDetect.cpp peripherals.cpp lineDetect.cpp)

# Link against libraries
target_link_libraries (linebot ${raspicam_CV_LIBS} ${CMAKE_THREAD_LIBS_INIT} /usr/local/lib/libwiringPi.so)
ELSE()
MESSAGE(FATAL_ERROR "OPENCV NOT FOUND IN YOUR SYSTEM")
ENDIF()
#####################################
```

Una vez se haya compilado el código, se debe escribir esta instrucción para ejecutar el código:

sudo ./linebot

Esto es debido a que la librería *wiring.h* requiere permisos de administrador.

6.2 Código Arduino

- ComRasp.ino: Fichero de código para compilar y grabar desde el programa Arduino en una placa Arduino Uno R3

```
/*****************************************************
    Programa para gestión de Arduino proyecto Linebot:
                  - Código para comunicación con Raspberry por I2C
                  - Control de los motores
                  - Control de Anillo de leds

    Realizado por David Ortiz
*/

// Includes
// Libreria de los servos
#include <Servo.h>
// Libreria para la comunicación I2C
#include <Wire.h>
// Se incluye la librería de anillo de leds (Flash)
#include <Adafruit_NeoPixel.h>

// Definición de la dirección I2C
const int I2CADDRESS = 4;

const int I1 = 7;  // Control pin 1 for motor 1 Left
const int I2 = 6;  // Control pin 2 for motor 1 Left
const int I3 = 5;  // Control pin 1 for motor 2 Right
const int I4 = 4;  // Control pin 2 for motor 2 Right

// Defines de los controles de los servos
const int PitchPin = 8;
const int YawPin = 9;

// Defines de las constantes del Flash
const int Flash_PIN = 2; // El pin de control será el 1
const int STRIPSIZE = 12; // El anillo tiene 12 Leds.

// Variable para el envío de datos para controlar el LED
bool bFlash;
int i2cFlashByte;
int flashColor;
int flashBright;

// Variable de control de la velocidad  y dirección de los motores
int leftMotorDirection;
int rightMotorDirection;
int leftMotorSpeed;
int rightMotorSpeed;

// Variable para los servos
Servo servoYaw; // servo para el giro
Servo servoPitch; // Servo para la inclinación

// Variable globales de control del estado de los servos del Pan & Tilt
bool bYaw;
bool bPitch;

// Variables para actualizar la posición de los motores al inicio.
bool bMotorUpdate;

// Variables para controlar la transmisión de datos de los motores
bool bMotor;
int i2cMotorByte;

// Variable para guardar la dirección hacia donde se movia antes.
int actualRightDirection;
int actualLeftDirection;  // Las variables pueden ser:
                          //  1: Adelante
                          //  0: Atrás

// Parametros para inicializar el anillo de Leds
// Parametro 1 = Número de pixeles en el anillo
// Parametro 2 = Número de pin
// Parametro 3 = Descriptores del tipo de pin, se pueden combinar:
//   NEO_KHZ800  Flujo de datos de 800 KHz (La mayoria de los productos NeoPixel con leds WS2812)
//   NEO_KHZ400  400 KHz (Para el resto de versiones classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Los pixeles estan cableados para flujo de datos GRB (La mayoria de los productos NeoPixel)
//   NEO_RGB     Los pixeles estan cableados para flujo de datos RGB (v1 FLORA pixels, not v2)
Adafruit_NeoPixel Flash = Adafruit_NeoPixel(STRIPSIZE, Flash_PIN, NEO_GRB + NEO_KHZ800);

// Función para controlar el movimiento de los motores
void updateMotorSpeed(int leftMovement, int rightMovement, int leftMotor, int rightMotor)
{
  // Se actualizan la salida de la dirección.
  // Motor Izquierdo
  if(leftMovement == 1)
  {
    // Si el movimiento es hacia adelante
    digitalWrite(I1, HIGH);
    digitalWrite(I2, LOW);
  }
  else
  {
    digitalWrite(I1, LOW);
```

```
    digitalWrite(I2, HIGH);
  }

  // Motor Derecho
  if(rightMovement == 1)
  {
    // Si el movimiento es hacia adelante
    digitalWrite(I3, HIGH);
    digitalWrite(I4, LOW);
  }
  else
  {
    // Si es hacia atrás
    digitalWrite(I3, LOW);
    digitalWrite(I4, HIGH);
  }

  // Guardo la dirección en la que se están moviendo los motores
  actualLeftDirection = leftMovement;
  actualRightDirection = rightMovement;
}

// Función para pintar los Leds, uno a uno
void colorWipe(uint32_t c, uint8_t wait)
{
  // Se recorren todos los Leds
  for(uint16_t i=0; i<Flash.numPixels(); i++)
  {
      Flash.setPixelColor(i, c);
      Flash.show();
      delay(wait);
  }
}

// Función de recepción de datos I2C
void receiveEvent(int howMany)
{
  int a=0;
  while(Wire.available()>0)
  {
    // Si el primer valor enviado es para controlar el motor, se almacenan los dos valores
    if(bMotor == true)
    {
      a = Wire.read();
      switch(i2cMotorByte)
      {
        case 0:
          // Se recibe el primer dato
          leftMotorDirection = a;
          i2cMotorByte++;
          break;
        case 1:
          // Se recibe el segundo dato
          rightMotorDirection = a;
          i2cMotorByte++;
          break;
        case 2:
          // Se recibe la velocidad del motor izquierdo
          leftMotorSpeed = a;
          i2cMotorByte++;
          break;
        case 3:
          // Se recibe la velocidad del motor derecho
          rightMotorSpeed = a;
          // Se actualiza la velocidad del motor;
          updateMotorSpeed(leftMotorDirection, rightMotorDirection, leftMotorSpeed, rightMotorSpeed);
          // Se inicializan las variables de recepción
          bMotor = false;
          i2cMotorByte = 0;
          break;
      }
      Serial.print("Datos de motor: ");
    }
    // Si el primer bit enviado es para controlar el giro, se actualiza el valor
    else if(bYaw == true)
    {
      // Se ha detectado un envío para el control del giro o Yaw
      a = Wire.read();
      // Se reinicia el flag
      bYaw = false;
      Serial.print("Datos de giro: ");

      // Se escriben los datos del servo de giro.
      servoYaw.write(a);
    }
    // Si el primer bit enviado es para controlar la inclinación, se actualiza el valor
    else if (bPitch == true)
    {
      // Se ha detectado un envío para el control de la inclinación o Pitch
      a = Wire.read();
      // Se reinicia el flag
      bPitch = false;
      Serial.print("Datos de inclinación: ");

      // Se escriben los datos del servo de inclinación
      servoPitch.write(a);
    }
    // Si el primer bit se dirige al flash
    else if(bFlash == true)
    {
      a = Wire.read();
```

```
      switch(i2cFlashByte)
      {
        case 0:
          // Se recibe el color del flash 0 apagado, 1 encendido(blanco)
          flashColor = a;
          i2cFlashByte++;
          break;
        case 1:
          // Se recibe la el brillo del flash
          flashBright = a;
          // Se actualiza el brillo y el color del flash
          if(flashColor == 0)
          {
            colorWipe(Flash.Color(0, 0, 0), 25); // Se apaga el flash
          }
          else if(flashColor == 1)
          {
            colorWipe(Flash.Color(255, 255, 255), 25); // Se activa en blanco
          }
          Flash.setBrightness(flashBright);
         // Se reinicia la variable de recepción del flash
          bFlash = false;
          i2cFlashByte = 0;
          break;
      }
      Serial.print("Datos del Flash");
    }
    else
    {
      // Si es el primer dato, entonces, se descifra que se va a enviar.
      a = Wire.read();
      switch(a)
      {
        case 77:
          // Se ha detectado que se va enviar una actualización de la posición del motor
          bMotor = true;
          break;
        case 89:
          //Se ha detectado que se va enviar el giro
          bYaw = true;
        break;
        case 80:
          // Se ha detectado que se va a enviar la inclinación
          bPitch = true;
        break;
        case 70:
          // Se detecta una trama para los el control de flash
          bFlash = true;
          Serial.print("No entro aqui?");
        break;
      }
    }
    //a++;
  }
  Serial.println(a);
}

// Función de inicialización del microprocesador
void setup()
{
  // Se inicializa el puerto serie
  Serial.begin(115200); // start serial for output
  Serial.println("Hello!");

  // Se inicializa el puerto I2C
  Wire.begin(I2CADDRESS); // join i2c bus with address #4
  Wire.onReceive(receiveEvent); // register event

  // Se enlazan las salidas de los servos con los pines.
  servoYaw.attach(YawPin);
  servoPitch.attach(PitchPin);

  // Se inicializa la posición
  servoYaw.write(90);
  servoPitch.write(15);

  // Se incializan los pines de control del chip L293D

  pinMode(I1, OUTPUT);
  pinMode(I2, OUTPUT);
  pinMode(I3, OUTPUT);
  pinMode(I4, OUTPUT);

  // Se inicializan los valores de los motores por defecto
  leftMotorSpeed = 0;
  rightMotorSpeed = 0;

  // Se inicializa la dirección de los motores
  digitalWrite(I1, HIGH);
  digitalWrite(I2, LOW);
  digitalWrite(I3, HIGH);
  digitalWrite(I4, LOW);

  // Se inicializan las variables globales de control de envíos
  bYaw = false;
  bPitch = false;
  bMotor = false;
  bMotorUpdate = true;
  bFlash = false;

  // Se inicializan los datos para recibir I2C
```

```
  i2cMotorByte = 0;

  // Inicializo la variable de dirección. La pongo hacia adelante
  actualLeftDirection = 1; // 1 es adelante, 0 es atrás.
  actualRightDirection = 1;

  // Inicialización del Flash
  Flash.begin();
  Flash.setBrightness(75);  // Control de Brillo de los leds
  Flash.show(); // Se inicializan los Leds a 0.

  delay(200);
}

void loop()
{
  // Si ha habido algún cambio en la velocidad del motor
  if(bMotorUpdate == true)
  {
    // Se actualizan las direciones de los motores
    digitalWrite(I1, HIGH);
    digitalWrite(I2, LOW);
    digitalWrite(I3, HIGH);
    digitalWrite(I4, LOW);

    bMotorUpdate = false;
  }
}
```

6.3 Código Windows

- main.cpp: Fichero para crear un servidor en un ordenador con Windows.

```
/*
*
*   Paso 7.- Probando los sockets
*       En este fichero voy a intentar comenzar a establecer los envíos entre la
*                   Raspberry y el ordenador.
*   Realizado por David Ortiz 2016
*/

// Librerias para incluir
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

// Incluyo las librerias para establecer un socket
#include <WinSock2.h>

// Incluyo la llamada a la librería.
#pragma comment(lib,"ws2_32.lib") //Winsock Library

// Bloque principal
int main(int argc, char **argv)
{
            // Variable para guardar los datos del WinSock
            WSADATA wsa;
            // Creo la variable para crear un puerto para escuchar....
            struct sockaddr_in serverBind;
            // Variables para el mensaje enviado, mensaje recibido y tamaño del mensaje recibido.
            char *message, server_reply[2000];
            int recv_size;

            // Variables para consultar la ip del servidor al que se conecta
            char *hostname = "www.google.com";
            char ip[100];
            struct hostent *he;
            struct in_addr **addr_list;

            // Variable para crear un servidor escuchando
            int c;
            SOCKET new_socket, s1;
            struct sockaddr_in client;

            // Variables para la cámara
            cv::Mat img;

            // 320 x 240
            const int height = 240;
            const int width = 320;

            std::cout << "\nInitialising Winsock...\n";
            // Para inicializar los sockets se utiliza esta función, y se inicializa ocmo 2,2, y se pasa la variable
            // wsa, que incluirá información sobre el socket.
            if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
            {
                        printf("Failed. Error Code : %d", WSAGetLastError());
                        return 1;
            }

            std::cout << "Initialised.\n";

            // Se crea un socket para escuchar, es decir, para esperar si alguien se conecta
            if ((s1 = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
            {
                        printf("Could not create socket : %d", WSAGetLastError());
            }

            printf("Socket created.\n");

            // Se redefinen primero las caracteristicas del server
            serverBind.sin_addr.S_un.S_addr = INADDR_ANY;
            serverBind.sin_family = AF_INET;
            serverBind.sin_port = htons(8888);

            // Se crea un servicio para escuchar
            if (bind(s1, (struct sockaddr *)&serverBind, sizeof(serverBind)) == SOCKET_ERROR)
            {
                        printf("Bind failed with error code : %d\n", WSAGetLastError());
            }

            puts("Bind done");

            // Escuchando a nuevas conexiones
            listen(s1, 3);

            // Se aceptan las conexiones que lleguen
            puts("Waiting for incoming connections...");

            // El tamaño de la estructura del socket que se almacenará, con las caracteristicas de servidor
            // etc.
            c = sizeof(struct sockaddr_in);
```

```
	// Se crea un nuevo socket de conexión con este nuevo servidor.
	new_socket = accept(s1, (struct sockaddr *)&client, &c);
	// Si se ha creado bien, pues adelante!
	if (new_socket == INVALID_SOCKET)
	{
		printf("accept failed with error code : %d\n", WSAGetLastError());
	}

	puts("Connection accepted");

	// Añado un código para visualizar la dirección que se ha conetado
	char *client_ip = inet_ntoa(client.sin_addr);
	int client_port = ntohs(client.sin_port);

	puts(client_ip);

	// Creo una ventana donde se verá la foto
	cv::namedWindow("Lo recibido");
	// Ajusto la posición
	cv::moveWindow("Lo recibido", 0, 0);

	// Se crean los parametros de la cámara
	img = cv::Mat::zeros(height, width, CV_8UC3); // Imagenes en color
	int  imgSize = img.total()*img.elemSize();
	char *sockData;
	sockData = (char *)malloc(sizeof(char) * imgSize + 1);
	// Se añade un bucle de recepción
	while (1)
	{
		// Recepción de datos
		int bytes = 0;
		for (int i = 0; i < imgSize; i += bytes) {
			if ((bytes = recv(new_socket, sockData + i, imgSize - i, 0)) == -1) {
				return 0;
			}
		}

		// Transformación de imagenes en color.
		int ptr = 0;
		for (int i = 0; i < img.rows; i++) {
			for (int j = 0; j < img.cols; j++) {
				img.at<cv::Vec3b>(i, j) = cv::Vec3b(sockData[ptr + 0], sockData[ptr + 1], sockData[ptr + 2]);
				ptr = ptr + 3;
			}
		}

		// Se muestra la imagen
		cv::imshow("Lo recibido", img);

		if (cv::waitKey(1) == 'q')
			break;
	}
	puts("Reply received\n");
	cv::waitKey(1);

	// Se cierra el socket y se limpia todo.
	closesocket(s1);
	WSACleanup();

	// Se finaliza el programa.
	return 0;
}
```

7. Referencias

[1] Wikipedia (s.f.). Motor de corriente continua (Online). Disponible en: https://es.wikipedia.org/wiki/Motor_de_corriente_continua

[2] Wikipedia (s.f.). Motor de paso a paso (Online). Disponible en: https://es.wikipedia.org/wiki/Motor_paso_a_paso

[3] Emir, Michael (01 – 02 – 2012) Baterías: Comparativa Litio, NiMh, NiCd, LiPo, Plomo. Disponible en: http://topcoding.blogspot.com.es/2012/02/baterias-comparativa-litio-nimh-nicd.html

[4] Photoaxe (s.f.), A Guide for Understanding the Camera Sensor (Online). Disponible en: http://www.photoaxe.com/a-guide-for-understanding-the-camera-sensor/

[5] Agarwal, Tarun (2014), Know about a Charge Coupled Device's Working Principle (Online). Disponible en: https://www.elprocus.com/know-about-the-working-principle-of-charge-coupled-device/

[6] SCHÄFER (s.f.), Weasel® vehículo guiado automatizado (Online). Disponible en: http://www.ssi-schaefer.es/sistemas-logisticos/vehiculos-de-guiado-automatico-agv/weasel-R.html

[7] CEIT Technical innovation (s.f.), AGV System (Online). Disponible en: http://www.ceittechinnovation.eu/index.php/en/agv-system

[8] Wikipedia (s.f.). Hardware Libre (Online). Disponible en: https://es.wikipedia.org/wiki/Hardware_libre

[9] Wikipedia (s.f.). Software Libre (Online). Disponible en: https://es.wikipedia.org/wiki/Software_libre

[10] FSF (s.f.). Free software foundation (Online). Disponible en: http://www.fsf.org/

[11] Arduino (s.f.). Página web oficial Arduino (Online). Disponible en: https://www.arduino.cc/

[12] Wikipedia (s.f.). Arduino (Online). Disponible en: https://es.wikipedia.org/wiki/Arduino

[13] Raspberry Pi (s.f.). Página web oficial Raspberry pi (Online). Disponible en: https://www.raspberrypi.org/

[14] Blog Historia de la informática (18 de Diciembre, 2013), Raspberry PI (Online). Disponible en: http://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

[15] OpenCV (s.f.). Página web oficial OpenCV (Online). Disponible en: http://opencv.org/

[16] Wikipedia (s.f.). OpenCV (Online). Disponible en: https://es.wikipedia.org/wiki/OpenCV

[17] OpenCV (s.f.). History OpenCV (Online). Disponible en: http://opencv.org/about.html

[18] Microchip (s.f.), Documentación del dsPIC33FJ128MC802 (Online). Disponible en: http://www.microchip.com/wwwproducts/en/dsPIC33FJ128MC802

[19] J.M. Angulo Usategui, A. Etxebarría Ruiz, I. Angulo Martínez y I. Trueba Parra (2010), dsPic Diseño práctico de aplicaciones, Madrid, McGraw Hill

[20] Wikipedia (s.f.). Espectro visible por el ser humano (Online). Disponible en: https://es.wikipedia.org/wiki/Espectro_visible

[21] The Electronic Lives Manufacturing (4 de Febrero, 2011), Fundamentals and experiences of Line Scan Camera (Online). Disponible en: http://elm-chan.org/works/lcam/report.html

[22] Raspberrypi.org (2016), Camera Module - Raspberry Pi Documentation (Online). Disponible en: https://www.raspberrypi.org/documentation/hardware/camera/README.md

[23] Cobo Lopez, Alejandro (11 de Octubre, 2013), Guía de Raspberry Pi 2 y Raspberry Pi 3 (Online). Disponible en: https://hardlimit.com/guia-raspberry-pi/#__RefHeading__1658_924516217

[24] Aplicaciones de la Visión Artificial (28 de Febrero, 2015), RaspiCam: C++ API for using Raspberry camera with/without OpenCv (Online). Disponible en: http://www.uco.es/investiga/grupos/ava/node/40

[25] Rosebrock, Adrian (18 de Marzo, 2016), Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3 (Online). Disponible en: http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/

[26] Dawson-Howe, K. (2014) A practical introduction to computer vision with OpenCV, Dublin, Wiley

[27] Brahmbhatt, S. (2013). Practical OpenCV. Berkeley, California, Apress.

[28] OpenCV (s.f.). Documentation OpenCV (Online). Disponible en: http://docs.opencv.org/3.1.0/

[29] Henderson, Gordon (s.f.). Página oficial librería Wiring Pi (Online). Disponible en: http://wiringpi.com/

[30] Adafruit (s.f.). Librería control NeoPixel (Online). Disponible en https://github.com/adafruit/Adafruit_NeoPixel

[31] Kuo, B. (1996). Sistemas automáticos de control. España, Prentice Hall

[32] Codermonkey (21 de Septiembre, 2012), PID Controller in C++ (Online). Disponible en: http://codermonkey65.blogspot.com.es/2012/09/basic-control-theory-in-c.html

[33] Cook, G. (2011). Mobile robots. Piscataway, NJ: IEEE.