UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

VEHÍCULO AUTÓNOMO CON VISIÓN ARTIFICIAL UTILIZANDO OpenCV

GRADO EN INGENIERÍA ELÉCTRICA

TRABAJO FINAL DE GRADO

AUTOR:

Adrián Benavent Pla

TUTOR:

D. Juan Ramón Rufino Valor

Convocatoria de defensa: Septiembre de 2017

Índice General

AGRADECIMIENTOS:

En primer lugar, me gustaría agradecer a mis padres, mis hermanos y a mi novia, el apoyo que siempre me han brindado durante mis estudios, y en especial en este último año con el que pongo fin a mis estudios de grado.

En segundo lugar, me gustaría agradecer el asesoramiento y la ayuda recibida por parte del profesor y tutor de este trabajo fin de grado, Don Juan Ramón Rufino Valor, de la Universidad Politécnica de Valéncia, Campus d'Alcoi.
Así mismo, también agradezco al profesor Don Jaime Masiá Vañó, de la UPV Campus d'Alcoi, la ayuda recibida en este trabajo fin de grado ya que me ayudo en la realización e impresión de la caja que contiene todos los componentes.
Además, me gustaría agradecer a Don Juan Ramón Rufino Valor y a Don Jaime Masiá Vañó la enseñanza prestada durante estos años de estudio, gracias a que tuve la oportunidad de formar parte en el grupo de robótica y mecatrónica GROMEP, de la UPV Campus d'Alcoi.

Por último, me gustaría agradecer a todas y cada una de esas personas, ya sean profesores, compañeros de clase o personas que he conocido a lo largo de esta travesía, en especial a Misael Sandoval Salvatierra, que de alguna forma han influido en mi desarrollo como graduado en ingeniería y que, gracias a sus consejos y enseñanzas, han hecho posible que hoy esté aquí, defendiendo mi trabajo fin de grado.

¡Muchas gracias!

1. RESUMEN

El presente proyecto recoge la información necesaria para la movilidad de un vehículo que circulará de forma autónoma siguiendo un circuito limitado por dos líneas paralelas (carril) simulando una carretera. El vehículo llevará una Raspberry Pi 2 y el software estará realizado en C++ utilizando las librerías OpenCV, las cuáles son necesarias para la aplicación de la tecnología de la visión artificial.

RESUM

El present projecte recull la informació necessària per a la mobilitat d'un vehicle que circularà de manera autònoma seguint un circuit limitat per dues línies paral·leles (carril) simulant una carretera. El vehicle ha de portar una Raspberry Pi 2 i el software estarà realitzat en C ++ utilitzant les llibreries OpenCV, les quals són necessàries per a l'aplicació de la tecnologia de la visió artificial.

ABSTRACT

The present project collects the necessary information for the mobility of a vehicle which will circulate autonomously following a circuit limited by two parallel lines (lane) simulating a road. The vehicle will carry a Raspberry Pi 2 and the software will be made in C ++ using the OpenCV libraries, which are necessary for the application of artificial vision technology.

2. INTRODUCCIÓN

Hoy en día, la tecnología forma parte de nuestra vida de un modo que hace unos años nadie hubiera podido predecir.
Cada vez es más frecuente la aparición de nuevos dispositivos cuya finalidad sea mejorar o ayudar en algún aspecto nuestra vida diaria.
Gracias a la aparición de nueva tecnología, actualmente, hay muchos sectores de la industria que están innovando y con ello creciendo gracias a los avances tecnológicos.

Uno de los sectores que ha crecido y mejorado gracias a los avances tecnológicos es el sector automovilístico, o incluso el sector de las redes sociales gracias a la tecnología de visión artificial. Cada vez más, vemos a grandes empresas automovilísticas y tecnológicas apostando por los vehículos autónomos como medio de transporte del futuro, sin la necesidad de tener a una persona al mando del vehículo.

Empresas como Google, Toyota, Mercedes, BMW o Audi están haciendo pruebas con prototipos, cuyo fin es el transporte de personas o mercancía. Como podemos imaginar, no es tan fácil de hacer, ya que existen infinidades de variables las cuales no son fáciles de predecir, como por ejemplo, un accidente, pero aun así la visión artificial puede llegar a reducir gran parte de estos.

3. PRESENTACIÓN DE LA SOLUCIÓN

3.1. NECESIDAD QUE CUBRE

Desde hace ya algún tiempo se habla mucho de los vehículos autónomos, especialmente de los coches que vendrán en el futuro, pero también de aviones (UAVs), submarinos (AUVs), barcos… etc. Este tipo de sistemas prometen traer grandes mejoras en seguridad, reducción de costes y optimización de recursos, minimizando o eliminando el factor humano.

Para conseguir todo esto se utilizan una gran cantidad de sensores, como GPS, radares o cámaras, y una multitud de técnicas procedentes de disciplinas como son la ingeniería de control, la inteligencia artificial, la automatización y la visión artificial.

Una de las principales necesidades de este tipo de vehículos es la de capturar una gran cantidad de información en sólo un instante de tiempo, por lo que la visión artificial es perfecta, ya que una cámara puede capturar muchos datos del entorno en centésimas de segundo

3.2. OBJETIVO DEL PROYECTO

El objetivo del presente proyecto es el desarrollo de un prototipo de vehículo autónomo, tanto a nivel de software como de hardware, capaz de detectar, mediante el uso de visión artificial, las líneas de un circuito simulando las líneas de un carril de la carretera, para posteriormente mantener el vehículo lo más centrado posible en el carril.

3.3. PLANTEAMIENTO DE LA SOLUCIÓN

Para conseguir nuestro objetivo, la solución que en este proyecto se plantea, ante la necesidad anteriormente expuesta, consiste en el desarrollo de un vehículo (solución), capaz de circular de forma autónoma por el interior de unas líneas que forman un carril.

Para lograrlo, se ha desarrollado una solución basada en visión artificial, la cual una cámara adquiere imágenes de las líneas del carril. Tras ello, se van analizando las imágenes adquiridas y en función del resultado del análisis, se envía la orden de girar a la derecha, girar a la izquierda o seguir recto.

El software desarrollado (programa), el cual lleva a cabo el análisis y las ordenes a los motores, se ha escrito en el lenguaje de programación C++ y es ejecutado por una Raspberry Pi 2 Modelo B (microordenador). Este microordenador permite analizar las imágenes adquiridas mediante las librerías OpenCV, y además generar las señales (ordenes) de control para el movimiento de los motores.

Las imágenes a analizar se adquieren mediante una cámara de 5 megapíxeles, la cual está fabricada para usarse con la Raspberry Pi 2.

El motor del vehículo se mueve a través de una placa de control (shield), la cual lleva el circuito integrado L293D y el servomotor de la dirección se mueve directamente desde la señal que le envía la Raspberry Pi 2.

Tanto la cámara, Raspberry Pi 2, shield de los motores, cámara, y los diversos componentes auxiliares como cables y baterías de alimentación, van encapsulados en una caja impresa con una impresora 3D, cuyo diseño y fabricación se especifican más adelante.

En resumen, la solución planteada en este proyecto se centra en el desarrollo del software, aunque también se describe la parte del ensamblaje en el vehículo.

Para probar y demostrar que la solución funciona correctamente, se ha realizado un circuito hecho con seis cartulinas de color blanco, las cuales albergan un circuito dibujado. Se hará circular el prototipo de vehículo autónomo por el carril dibujado, con lo que la idea de este montaje es probar y demostrar que, los coches autónomos formarán parte de nuestras vidas.

4. DESARROLLO DE LA SOLUCIÓN

4.1. DESCRIPCIÓN DETALLADA DE LA SOLUCIÓN

Tal y como se ha descrito anteriormente en el planteamiento de la solución, ésta está formada por:

4.1.1. Elementos físicos (hardware)

- ***Raspberry Pi 2***

El dispositivo seleccionado para que ejecute nuestro programa que controla nuestra solución, es la Raspberry Pi 2 Modelo B.

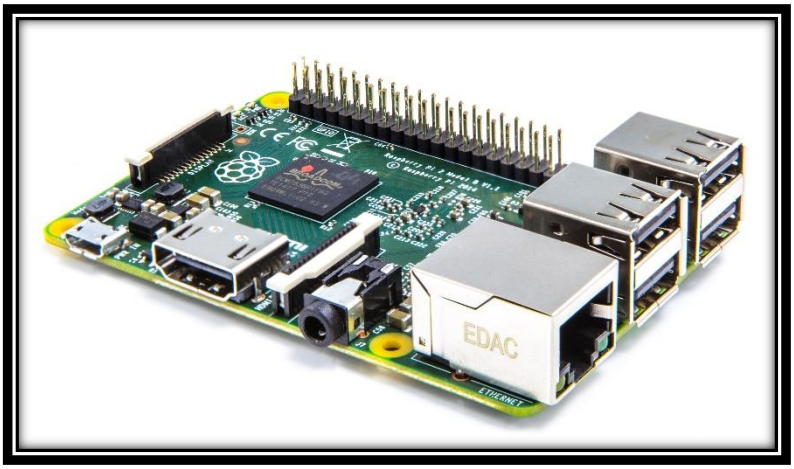


Imagen 1. Raspberry Pi 2 Modelo B.

- ***Cámara***

La cámara seleccionada para que nuestro dispositivo adquiera las imágenes a analizar es la cámara fabricada por la fundación Raspberry Pi (*RaspiCam)*, en su versión 1.3 que tiene 5 Megapíxeles y se comunica por CSI (Interfaz Serie para Cámaras).

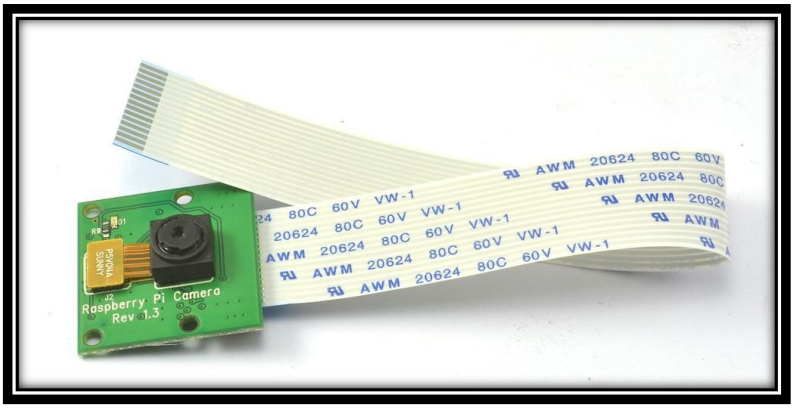


Imagen 2. Cámara Raspberry Pi versión 1.3 (*RaspiCam*).

- ***Placa de control para motores***

La placa de control seleccionada para mover el motor de las ruedas traseras es una placa que lleva incorporado el circuito integrado L293D, el cual permite controlar dos motores de forma bidireccional.

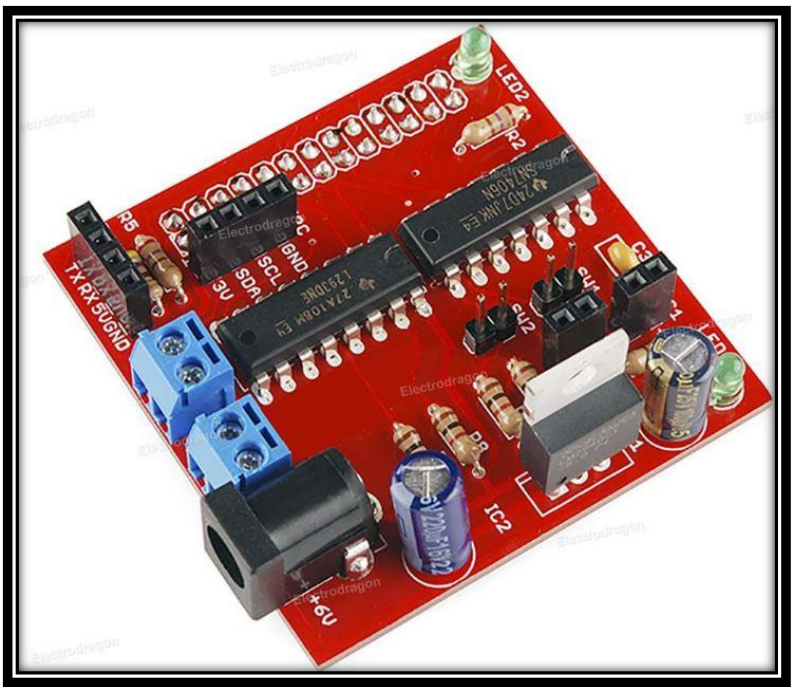


Imagen 3. Placa con circuito integrado L293D para control de motores bidireccionales.

- ***Motor de corriente continua***

Para mover las ruedas traseras del vehículo, que son las que dan la tracción, se ha utilizado un motor de corriente continua de 6 V y 6800 RPM.

Imagen 4. Motor de corriente continua de 6 VDC.

- ***Servomotor***

El movimiento de las ruedas delanteras, que son las ruedas de la dirección de vehículo, se ha conseguido mediante un servomotor analógico de la marca EMAX, de 6 VDC y que es capaz de proporcionar un par motor de 1,8 Kgf·cm a 6 V, o lo que es lo mismo, 17,65 N·cm.

$$\boxed{\mathbf{m \cdot g = F}} \rightarrow 1\ Kg \cdot 9{,}81\ m/s^2 = 9{,}81\ N \rightarrow \mathbf{1\ Kgf = 9{,}81\ N}$$

$$\boxed{\mathbf{T} = 1{,}8\ Kgf{\cdot}cm \cdot 9{,}81 = \mathbf{17{,}65\ N{\cdot}cm}}$$

Imagen 5. Servomotor EMAX ES08A.

- ***Power Bank para alimentar la Raspberry Pi 2***

Para alimentar la Raspberry Pi 2 y dotarla de un funcionamiento autónomo se escogió una "Power Bank" de 5200 mAh con salida USB de 5 VDC y 1 A.

Imagen 6. Power Bank Ion-Litio con capacidad de 5200 mAh.

- ***Batería de Ni-Cd para alimentar la placa de control de los motores***

Para alimentar la placa de control de los motores y los propios motores se escogió una batería de Ni-Cd de 700 mAh de 4,8 VDC.

Imagen 7. Batería de Ni-Cd con capacidad de 5200 mAh.

- ***WiFi USB***

Para proporcionar conexión a internet a la Raspberry Pi 2, se optó por conectarle un pin WiFi, de la marca EDUP, al puerto USB.

Imagen 8. Pin WiFi USB.

- ***Caja que contiene todos los componentes***

La caja que aloja a todos los componentes de nuestro vehículo, se ha diseñado con el programa SolidWorks.

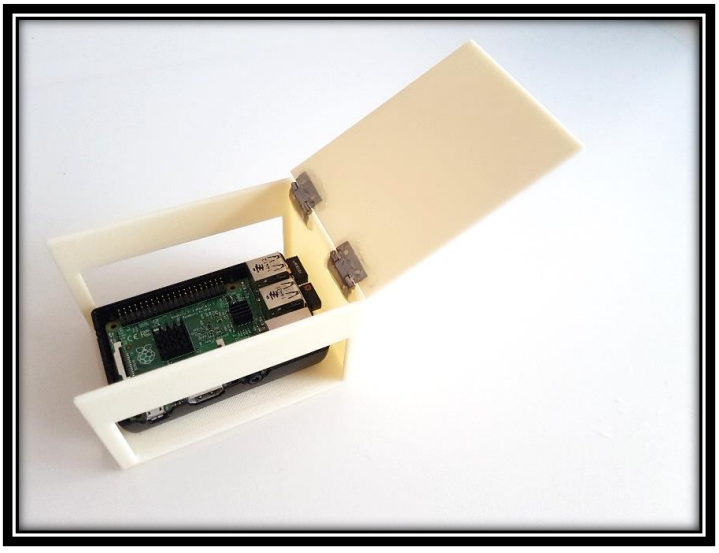


Imagen 9. Caja con los componentes en su interior sin cablear.

- ***Cables de conexión***

Los cables necesarios para realizar las conexiones de los diversos componentes y la alimentación son:

- Cable USB a micro USB para la alimentación del dispositivo Raspberry Pi 2.
- Cable rojo y negro de 1 mm^2 para la alimentación de la placa de control de los motores y para la interconexión de los motores a la placa de control de estos.
- Cable CSI de la cámara a la Raspberry Pi 2. Éste viene como parte de la propia cámara.
- Cables de colores macho/hembra para la conexión de la placa de control de los motores y el servomotor con la Raspberry Pi 2.

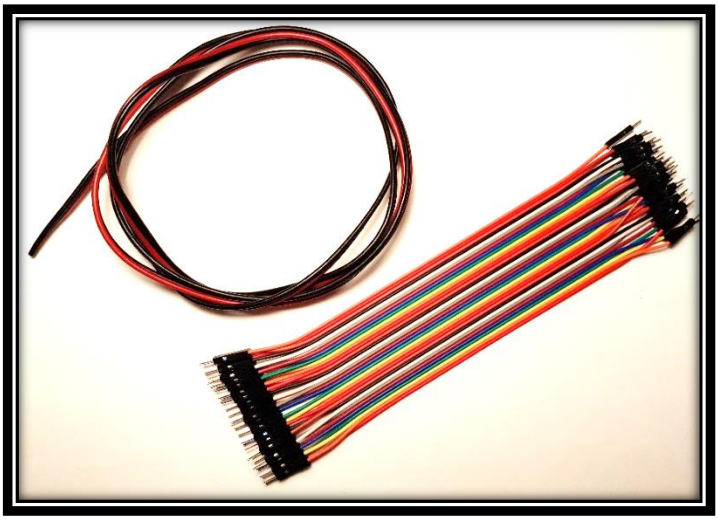


Imagen 10. Cables rojo, negro y macho/macho.

4.1.2. Software

- Sistema Operativo de la Raspberry Pi 2
- Programa desarrollado para que funcione nuestro vehículo

4.1.3. Herramientas de desarrollo

Editor de texto:
Para escribir el código de la aplicación que controla nuestro dispositivo, se ha utilizado el editor de texto NANO que incluye el sistema operativo de la propia Raspberry Pi 2.
Este editor de texto se encuentra disponible de forma gratuita y está especialmente enfocado a la creación de archivos de cualquier tipo, ya sea C, C++, Pyton, etc.

Software:
El lenguaje de programación escogido para escribir la aplicación que se ejecuta en la Raspberry Pi 2 es el C++. Se decidió utilizar este lenguaje puesto que es muy potente y es comúnmente utilizado.
El Sistema Operativo con el que la Raspberry Pi 2 funciona es Raspbian Jessie en su versión PIXEL desktop.

Librerías:
Para capturar imágenes y posteriormente ser tratadas se ha utilizado la librería RaspiCam.
Para el tratamiento de las imágenes capturadas, con el fin de obtener la información necesaria se ha utilizado las librerías OpenCV.
Para crear la señal PWM y controlar el servomotor y el motor se ha utilizado la librería WiringPi.

4.2. JUSTIFICACIÓN DE LOS COMPONENTES SELECCIONADOS

4.2.1. Herramienta de desarrollo (editor de texto NANO)

Para la realización del programa que se encargará de capturar imágenes, tratarlas y mover el vehículo por el interior de las líneas se optó por el editor de líneas de comandos NANO, el cual viene instalado de forma predeterminada en el software de la Raspberry Pi 2 y se ejecuta a través del "*Terminal*".

Elegimos esta herramienta de desarrollo y no otra porque, es fácil de usar, es gratuita y además te permite modificar gran variedad de archivos desde la propia Raspberry Pi 2, sin necesidad de utilizar una herramienta de desarrollo externa a esta, como por ejemplo Visual Studio 2015.

Cabe destacar que, aunque realices programas con el editor de líneas de comandos NANO, después puedes copiar fácilmente los programas desde la Raspberry Pi 2 a un ordenador para posteriormente poder modificarlos como por ejemplo con la herramienta Visual Studio 2015.

Para poder copiar programas desde la Raspberry Pi 2 a un ordenador se puede utilizar el programa VNC Viewer, el cual es utilizado para manejar la Raspberry Pi 2 a distancia.

4.2.2. Software de la Raspberry Pi 2 (Raspbian Jessie con PIXEL)

La elección de Raspbian Jessie fue porque es el último software desarrollado por Raspbian, el cual es el sistema operativo oficial de la Fundación Raspberry Pi.

Además, se optó por utilizar la versión con el nuevo entorno de escritorio PIXEL, el cual está basado en Debian (sistema operativo libre para el ordenador).

4.2.3. Raspberry Pi 2 Modelo B

Se decantó por utilizar la Raspberry Pi en nuestra solución, y no utilizar Arduino, ya que la Raspberry Pi es un microordenador y el Arduino es un microcontrolador. Por esta razón fue por la que la Raspberry Pi es más adecuada para realizar las capturas y el análisis de imágenes, puesto que tiene mayor potencia de cálculo y está más enfocada a este tipo de soluciones que el Arduino, el cual podría servir de apoyo en un futuro.

Se eligió la Raspberry Pi 2 Modelo B puesto que ya la tenía y no hacía falta comprarla, aunque actualmente existe en el mercado la Raspberry Pi 3, la cual podría ser utilizada en un futuro para una nueva versión de nuestra solución.

4.2.4. Cámara

La elección de la cámara fue definida porque tiene un precio asequible y además es compatible con la Raspberry Pi 2.

En un principio se pensó en utilizar una cámara USB, pero como no se necesitaba de una gran resolución para nuestra solución se optó por utilizar la *RaspiCam*, pudiendo en un futuro optar por utilizar una cámara por USB con mayor resolución.

4.2.5. Placa de control L293D

Para poder mover el motor que acciona las ruedas traseras se decantó por una placa de circuito impreso, la cual tenía el circuito integrado L293D. Este circuito integrado te permite controlar el giro bidireccional de dos motores fácilmente.

4.2.6. Motor de corriente continua

La elección de utilizar un motor de corriente continua para el accionamiento de las ruedas traseras de vehículo fue porque son motores fáciles de controlar y baratos. No son motores muy fiables, pero para la finalidad de este trabajo era más que suficiente.

En este caso no se ha utilizado ningún control de velocidad para este motor, sino que funciona a pleno rendimiento, con lo que una posible mejora sería utilizar una señal PWM para controlar la velocidad del motor.

4.2.7. Servomotor

Para mover las ruedas delanteras se decantó por utilizar el servomotor EMAX ES08A, ya que con un servomotor y con una señal PWM podemos hacer un control más exhaustivo del movimiento de estas ruedas

Se utilizó un servomotor para accionar las ruedas delanteras y no un motor de corriente continua porque en el servomotor se puede controlar la posición de este y en un motor de corriente continua no, con lo que si utilizaba uno de estos lo que sucedería es que el vehículo giraría dando golpes bruscos en la dirección.

4.2.8. Alimentación autónoma

Tal y como se ha descrito anteriormente, fue necesario dotar a la solución de un funcionamiento totalmente autónomo, ya que de esta forma no sería necesario tener conectado cables de alimentación directamente a los enchufes, los cuales pueden causar un mal funcionamiento del vehículo.

Para ello se optó por alimentar a la Raspberry Pi 2 con una "Power Bank" de 5200 mAh y a la placa de control de los motores y los propios motores con una batería de Ni-Cd de 700 mAh, comúnmente utilizada para vehículos radiocontrol.

4.2.9. WiFi USB

Para proporcionar a nuestra solución una conexión a internet fue necesario utilizar un WiFi USB.

Con el pin WiFi USB lo que conseguimos es que la Raspberry Pi 2 esté conectada a la misma red a la que está conectado el ordenador, el cual a través de VNC Viewer podrá tener acceso al entorno de escritorio PIXEL de la Raspberry Pi 2 y de esta forma se podrá modificar, actualizar, compilar y ejecutar el programa creado para la solución a distancia.

A la hora de seleccionar el pin WiFi USB para usarlo en nuestra solución, se tuvo en cuenta que fuera compatible con la Raspberry Pi 2 y Raspbian Jessie.

4.2.10. Software para el manejo a distancia (VNC Viewer 6.0)

Para poder tener acceso de forma remota a la Raspberry Pi 2 y poder crear, modificar, compilar, y ejecutar cualquier archivo creado, fue necesario la activación del software VNC en la Raspberry Pi y posteriormente la instalación del programa VNC Viewer en su versión 6.1.1 en el ordenador.

A continuación, se muestra el link a través del cual se puede descargar la última versión del programa VNC Viewer para Windows:

https://www.realvnc.com/download/viewer/windows/

4.2.11. Caja que contiene todos los componentes

Para poder tener comodidad en el manejo del vehículo se decidió diseñar una caja que reuniera todos los componentes en su interior. De esta forma se puede transportar fácilmente de un lugar a otro y además era totalmente necesario porque es un dispositivo que se tiene que estar moviendo continuamente.

Se quiso diseñar la caja que contiene todos los componentes porque de esta forma se consigue una mejor adaptación de las necesidades.

Se decidió usar una impresora 3D y el programa llamado SolidWorks, tal y como se ha mencionado anteriormente, porque es un programa potente, se tenía acceso gracias a ser alumno de la universidad y porque es un programa compatible con la mayoría de impresoras 3D.

4.2.12. Cables de conexión

Se seleccionaron los cables de nuestra solución de acuerdo a las necesidades que se deseaban, buscando flexibilidad y teniendo en cuenta que no ocuparan mucho espacio para no tener que hacer la caja contenedora más grande de lo necesario.

4.2.13. Otras justificaciones

- **Entorno de trabajo**

El entorno en el que se hará uso de nuestro vehículo dispone de suficiente luz, con lo que no será necesario agregar a nuestro dispositivo una fuente de luz adicional.

- **Dispositivo económico**

Como era de esperar, lo que se intentó en todo momento era crear un vehículo autónomo lo más económico posible, ya que como bien se ha dicho anteriormente, es un prototipo meramente demostrativo.

4.3. DESCRIPCIÓN DETALLADA DE LOS COMPONENTES SELECCIONADOS

4.3.1. Herramienta de desarrollo (editor de texto NANO)

NANO es un editor de texto sencillo que se usa directamente en el "Terminal" de la Raspberry Pi 2 y actualmente se encuentra instalado por defecto en todas, o casi todas, las distribuciones Linux. Tiene una apariencia bastante tosca, pero aun así es lo suficientemente potente como para crear aplicaciones con los diferentes lenguajes de programación que soporta:

- C
- C++
- Python
- C#
- TXT
- HTML

Como ya se ha comentado anteriormente, el editor de texto NANO es una herramienta gratuita, completa y fácil de usar para desarrolladores que crean aplicaciones que no sean empresariales, como es nuestra solución.

Para crear un programa solamente tenemos que escribir en el "Terminal" de la Raspberry Pi 2 el comando *nano archivo.cpp* (para un archivo C++).

A continuación, se puede encontrar las anotaciones de los atajos presentes en el editor de texto NANO.

- Ctrl + G Solicitar el menú de ayuda
- Ctrl + X Salir del editor de texto NANO
- Ctrl + O Guardar el fichero actual en el disco
- Ctrl + R Insertar otro fichero en el actual
- Ctrl + \ Reemplazar texto en el editor
- Ctrl + W Buscar un texto en el editor
- Ctrl + Y Moverse a la página anterior

- Ctrl + V Moverse a la página siguiente
- Ctrl + K Cortar la línea actual y guardarla en el cutbuffer
- Ctrl + U Pegar el cutbuffer en la línea actual
- Ctrl + C Mostrar la posición del cursor
- Ctrl + T Solicitar el corrector ortográfico (si está disponible)
- Ctrl + P Moverse una línea hacia arriba
- Ctrl + N Moverse una línea hacia abajo
- Ctrl + F Moverse hacia adelante un carácter
- Ctrl + B Moverse hacia atrás un carácter
- Ctrl + A Moverse al principio de la línea actual
- Ctrl + E Moverse al final de la línea actual
- Ctrl + L Redibujar la pantalla actual
- Ctrl + ^ Marcar texto en la posición actual del cursor
- Ctrl + D Borrar el carácter bajo el cursor
- Ctrl + H Borrar el carácter a la izquierda del cursor
- Ctrl + I Insertar un carácter tab
- Ctrl + J Justificar el párrafo actual
- Ctrl + _ Ir a un número de línea en concreto

Para más información se puede escribir en el "Terminal" **man nano**, **info nano, /usr/doc/nano** o visitar la página oficial:

https://www.nano-editor.org/

4.3.2. Software de la Raspberry Pi 2 (Raspbian)

El Sistema Operativo escogido para la Raspberry Pi 2 es Raspbian en su versión Jessie, como ya se ha mencionado anteriormente.
Este Sistema Operativo debe su nombre a otro sistema operativo llamado Debian, el cual está compuesto por software completamente libre. Ha sido desarrollado por la fundación Raspberry Pi recientemente, concretamente en 2015.

Raspbian = **Rasp**berry + De**bian**

Podemos decir que Raspbian Jessie es una versión de Linux optimizada para pequeños dispositivos, como son la Raspberry Pi 2 y 3.

Raspbian Jessie ha sido desarrollado no solo para ofrecer un ordenador barato para la educación, sino también para ofrecer un ordenador pequeño, pero lo suficientemente potente como para realizar tareas desde prototipos simples hasta tareas complejas. Con esto lo que se pretende es que se use la Raspberry Pi para hacer el tipo de cosas que se haría en un Mac o un PC.

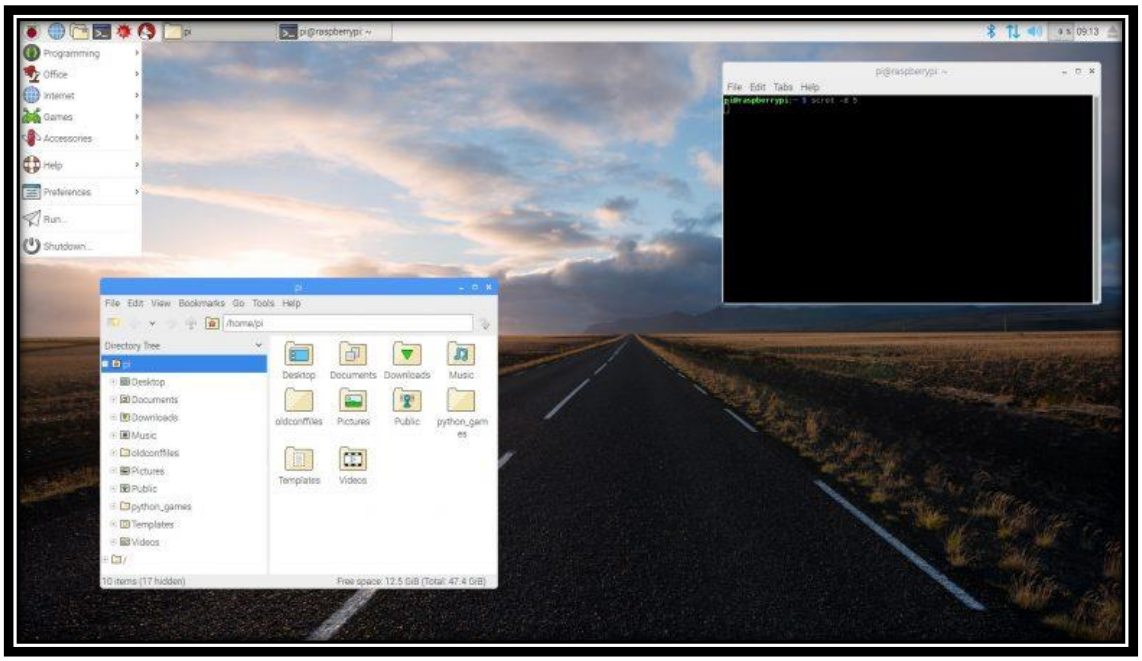


Imagen 11. Escritorio Raspbian Jessie con Pixel.

4.3.3. Raspberry Pi 2 modelo B

La Raspberry Pi que se seleccionó para nuestra solución fue la Raspberry Pi 2 modelo B, tal y como se ha descrito anteriormente.

La Raspberry Pi 2 es la segunda generación de la Raspberry Pi, ya que esta reemplaza la Raspberry Pi 1 modelo B+.
Si comparamos la Raspberry Pi 2 con la Raspberry Pi 1 podemos ver que la Raspberry Pi 2 tiene una CPU quad-core ARM Cortex-A7 de 900 MHz y 1 GB de memoria RAM, y la Raspberry Pi 1 modelo B+ tiene una CPU single-core ARM1176JZF-S de 700 MHz y 512 MB de memoria RAM.

Datos y especificaciones de la Raspberry Pi 2 modelo B:

- Procesador Broadcom BCM2836 ARMv7 quad-core corriendo a 900 MHz.
- 1 GB de memoria RAM LPDDR2.
- 4 puertos USB 2.0.
- 40 pines GPIO (*general purpose input/output*).
- Puerto HDMI de 1920 x 1200 píxeles.
- Puerto Ethernet de 10/100 Mbps.
- Puerto micro USB para la alimentación.
- Conector de audio de 3,5 mm combinado y vídeo compuesto.
- Interfaz para cámara (CSI).
- Interfaz para pantalla (DSI).
- Ranura para tarjeta micro SD.
- Núcleo de gráficos Broadcom 3D VideoCore IV.
- Tamaño de 85 x 56 mm aprox.
- Peso de 45 g.
- Consumo de 5 V y 900 mA, aunque depende de la carga de trabajo de los cuatro nucleos.

Podemos destacar que la Raspberry Pi 2 es completamente compatible con la Raspberry Pi 1.

Aspectos relevantes de la Raspberry Pi 2 modelo B:

- Puede proporcionar hasta 1,2 A al puerto USB, lo que tendremos la posibilidad de conectar más dispositivos USB con gran consumo de energía directamente a la Raspberry Pi 2. Para aprovecharse de esta característica es necesario que la fuente de alimentación micro USB sea de 2 A.
- Con 1 GB de memoria RAM, se puede ejecutar aplicaciones más grandes y potentes que necesiten de más recursos.
- Conector de 4 polos combinado para conectar su salida de audio estéreo de 3,5 mm y salida de vídeo compuesto.

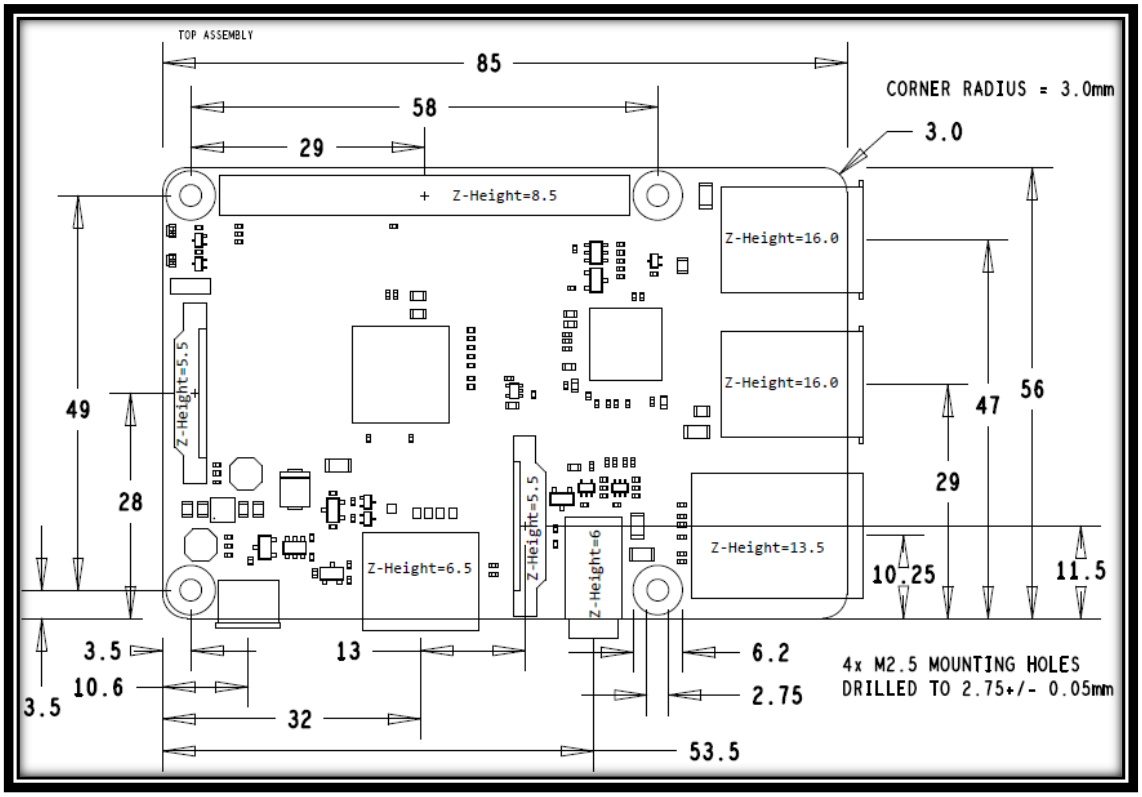


Imagen 12. Medidas de la Raspberry Pi 2 modelo B.

Como se puede observar en la imagen las medidas de la Raspberry Pi 2 modelo B son iguales a las medidas de su predecesora, la Raspberry Pi 1 modelo B+.

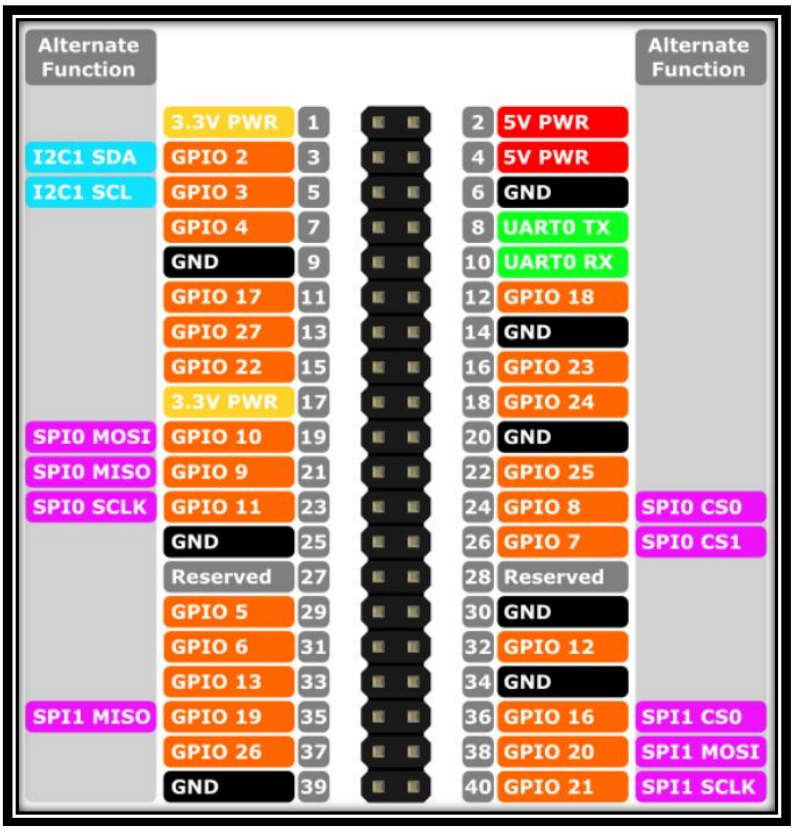


Imagen 13. Pines de propósito general de la Raspberry Pi 2 modelo B.

4.3.4. Cámara

La cámara utilizada en nuestro dispositivo es la cámara fabricada por la fundación Raspberry Pi (*RaspiCam o Raspberry Pi Camera Board)*, en su versión 1.3 (v1) que tiene 5 Megapíxeles y se comunica por CSI (Interfaz Serie para Cámaras).

Características de la cámara:

- Alta velocidad de captura de imágenes
  - 1080p (1920 x 1080 px.) @ 30 fps.
  - 720p (1280 x 720 px.) @ 60 fps.
  - 640 x 480 px. @ 60/90 fps.
- Resolución fija de 5 Megapixels.
- Sensor CMOS OmniVision OV5647 en un módulo de enfoque fijo con filtro IR integrado.
- Angulo de visión de 54 grados en horizontal y 41 grados en vertical.

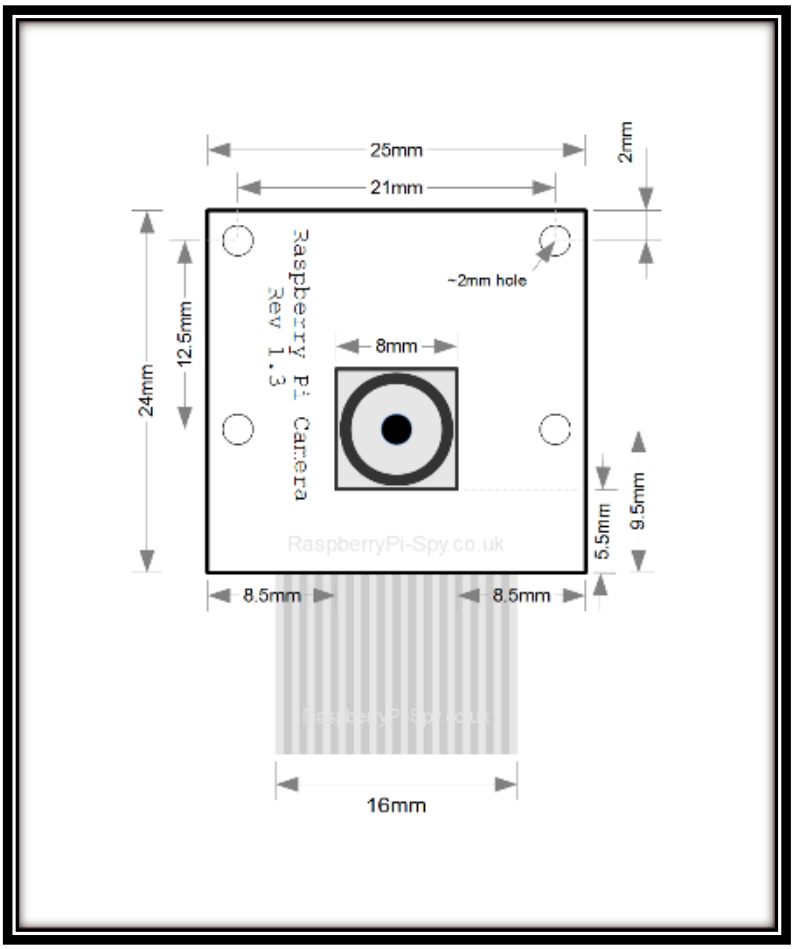


Imagen 14. Medidas de la cámara.

4.3.5. Placa de control L293D

La placa de control utilizada para dirigir el motor que controla la tracción del vehículo es una placa de circuito impreso fabricada por la empresa *Electrodragon* de China. Esta placa de circuito impreso lleva incorporado el circuito integrado *L293D* y el circuito integrado *SN74LS06N* de puertas lógicas NOT (inversor).

Componentes de la placa de control

- 1 x Circuito integrado L293D
- 1 x Circuito integrado SN74LS06N
- 1 x Regulador de voltaje LM2940
- 3 x Resistencia de 270 Ohm
- 1 x Resistencia de 470 Ohm
- 4 x Resistencia de 1 kOhm
- 1 x Condensador de 220 uF
- 1 x Condensador de 100 uF
- 1 x Condensador de 0,1 uF
- 2 x Led de 3 mm

- 2 x Conector de 2 Pin hembra
- 2 x Conector de 2 Pin macho
- 2 x Conector de 4 Pins hembra
- 1 x Conector hembra para Jack 3.5 mm
- 2 x Terminal de 2 Pins para cables de 3.5 mm
- 1 x Conector macho para Jack 3.5 mm

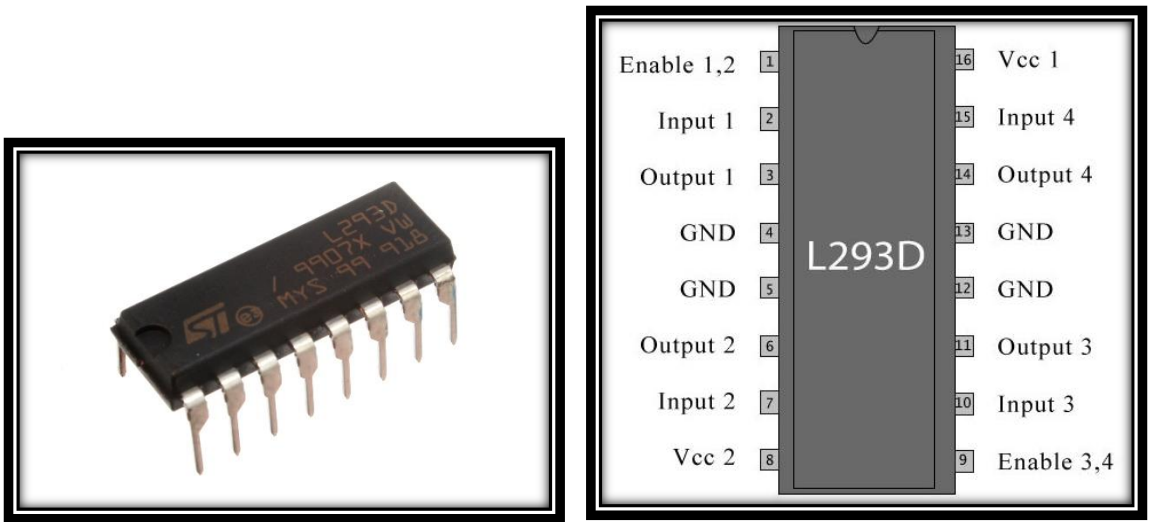


Imagen 15. Circuito integrado L293D.

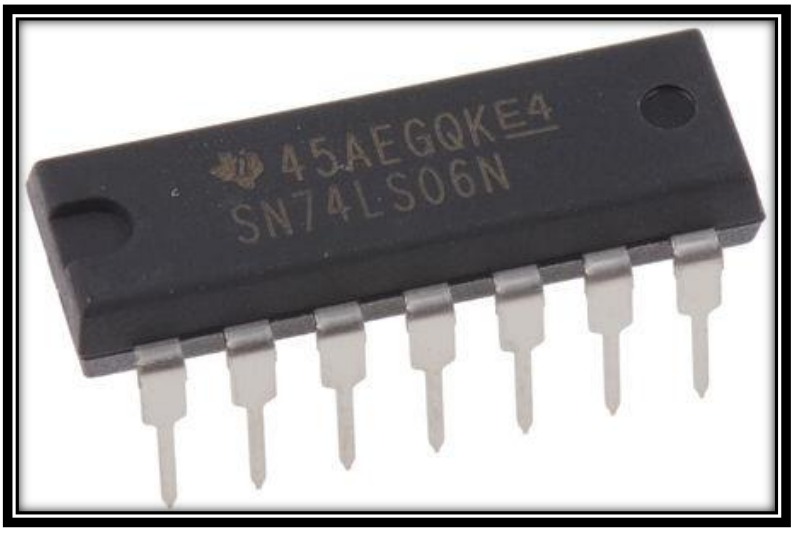


Imagen 16. Circuito integrado SN74LS06N.

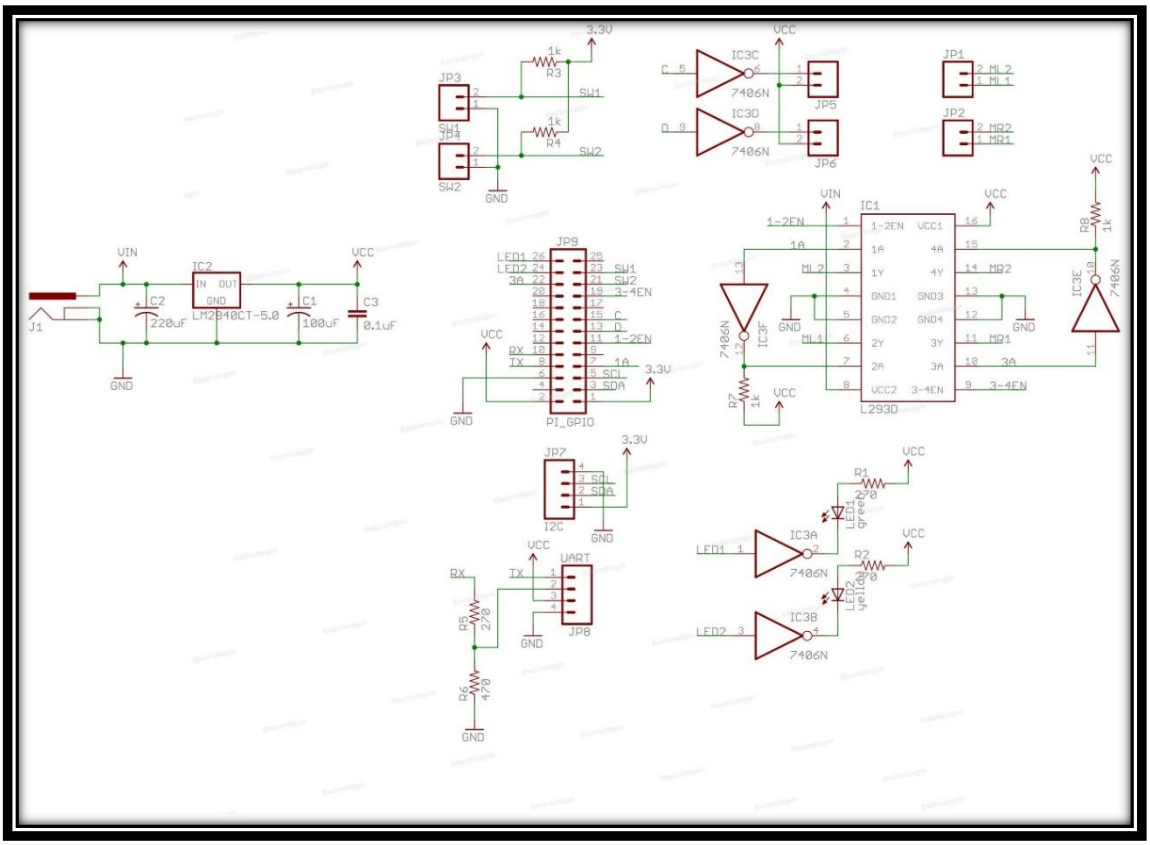


Imagen 17. Circuito eléctrico de la placa.

4.3.6. Motor de corriente continua

El motor de corriente continua que se ha usado para accionar las ruedas traseras (tracción del vehículo) es un motor de 6 V como los que puedes encontrar en los coches radiocontrol de juguete.

Características principales del motor de corriente continua:

- Tensión nominal: 6 VDC
- Tensión de funcionamiento: 3 – 9 VDC
- Corriente de eficiencia máxima: 0,25 A
- Velocidad de eficiencia máxima: 6800 RPM @ 6 V
- Diámetro del eje: 2 mm
- Longitud del eje: 8 mm
- Tamaño: 20,4 mm de diámetro x 25 mm de profundidad

4.3.7. Servomotor

Tal y como se ha descrito anteriormente, el servomotor seleccionado para mover las ruedas directrices (ruedas delanteras) es el ES08A de la marca EMAX, comúnmente utilizado para aeromodelismo y radiocontrol.

Este servomotor es analógico a diferencia de otros que son digitales, y es capaz de proporcionar un par motor de 1,8 Kgf·cm a 6 V, o lo que es lo mismo, 17,65 N·cm.

$$\boxed{\mathbf{m \cdot g = F}} \rightarrow 1\ Kg \cdot 9{,}81\ m/s^2 = 9{,}81\ N \rightarrow \mathbf{1\ Kgf = 9{,}81\ N}$$

$$\boxed{\mathbf{T} = 1{,}8\ Kgf{\cdot}cm \cdot 9{,}81 = \mathbf{17{,}65\ N{\cdot}cm}}$$

Especificaciones del servomotor EMAX ES08A:

- Tensión de alimentación: 6 VDC
- Tensión de funcionamiento: 4,8 – 6 VDC
- Par máximo: 1,5 Kg a 4,8 V y 1,8 Kg a 6 V
- Dimensiones: 22 x 11,5 x 24 mm
- Peso: 9 gr.

Imagen 18. Servomotor EMAX ES08A.

4.4. ACOPIO DE MATERIALES Y PRESUPUESTO

4.4.1. Compra de los componentes

La compra de los componentes utilizados en este proyecto se hizo a través de las siguientes páginas web:

- Raspberry Pi 2 Modelo B:
  http://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/8326274/ (RS-Online)
- Tarjeta micro SD de 8 GB:
  https://tiendas.mediamarkt.es/p/tarjeta-microsd-sandisk-8gb-1251528 (MediaMarkt)
- Vehículo:
  (Juguettos)
- Shield para los motores:
  http://www.electrodragon.com/product/raspberry-pi-motor-robot-shield-kit-l293d/ (Electrodragon)
- Motor de corriente continua:
  Material propio. Valor estimado en presupuesto.
- Servomotor EMAX ES08A:
  http://www.rctecnic.com/servos-micro-mini-3-30gr/479-servo-analgico-emax-es08a-9gr-18kg (RC Tecnic)
- Cámara RaspiCam v1.3:
  Material proporcionado por GROMEP. Valor estimado en presupuesto.
- Adaptador WiFi USB para Raspberry Pi:
  http://www.electan.com/modulo-wifi-compatible-raspberrypi-p-3391.html (Electan)
- Cables macho/macho, macho/hembra y hembra/hembra:
  https://www.amazon.es/gp/product/B01FXPC2HU/ref=oh_aui_detailpage_o07_s00?ie=UTF8&psc=1 (Amazon)
- Power Bank de 5200 mAh:
  Material propio. Valor estimado en presupuesto.
- Batería de Ni-Cd de 700 mAh:
  http://www.rctecnic.com/baterias-nimh/5071-bateria-ni-cd-4-8-v-700-mAh-hb-p1802 (RC Tecnic)
- Carcasa para la Raspberry Pi 2:
  http://www.electan.com/caja-para-raspberry-model-negra-p-6182.html (Electan)
- Disipadores para Raspberry Pi:
  http://www.electan.com/disipadores-para-raspberry-con-adhesivo-p-6587.html (Electan)
- Cable de conexión jack 5 VDC:
  Material propio. Valor estimado en presupuesto.
- Interruptor para 5 VDC:
  Material propio. Valor estimado en presupuesto.
- Aluminio para sujeción de la cámara:
  Material propio. Valor estimado en presupuesto.

4.4.2. Presupuesto

El presupuesto del prototipo realizado para este proyecto se muestra a continuación:

ELEMENTO	CANTIDAD	PRECIO USD	PRECIO €
Raspberry Pi 2 Modelo B (propio, valor estimado)	1	36,25 USD	31,90 €
Tarjeta micro SD de 8 GB	1	6,81 USD	5,99 €
Vehículo	1	22,67 USD	19,95 €
Shield para motores	1	27,62 USD	24,31 €
Motor de corriente continua (propio, valor estimado)	1	5,67 USD	4,99 €
Servomotor EMAX ES08A	1	13,75 USD	12,10 €
Cámara RaspiCam v1.3 (cortesia de GROMEP, valor estimado)	1	20,40 USD	17,95 €
Adaptador WiFi USB para Raspberry Pi	1	9,11 USD	8,02 €
Cables macho/macho macho/hembra colores	15	1,13 USD	1,00 €
Power Bank de 5200 mAh (propio, valor estimado)	1	18,13 USD	15,95 €
Batería de Ni-Cd de 700 mAh	1	17,95 USD	15,80 €
Carcasa para la Raspberry Pi 2	1	14,68 USD	12,92 €
Disipadores para Raspberry Pi	1	1,36 USD	1,20 €
Cable de conexión jack 5 VDC (propio, valor estimado)	1	6,81 USD	5,99 €
Interruptor para 5 VDC (propio, valor estimado)	1	1,99 USD	1,75 €
Aluminio para sujeción de la cámara (propio, valor estimado)	1	0,32 USD	0,28 €
Impresión 3D (cortesia de GROMEP)	1	-	-
TOTAL		**204,65 USD**	**180,09 €**

4.5. PREPARACIÓN DE LA RASPBERRY PI 2

4.5.1 Descarga y preparación de Raspbian Jessie

Para poder utilizar el sistema operativo Raspbian Jessie, primero tenemos que descargarlo y posteriormente instalarlo en una tarjeta Micro SD.

Desde nuestro ordenador descargamos el sistema operativo Raspbian Jessie y lo descomprimimos:

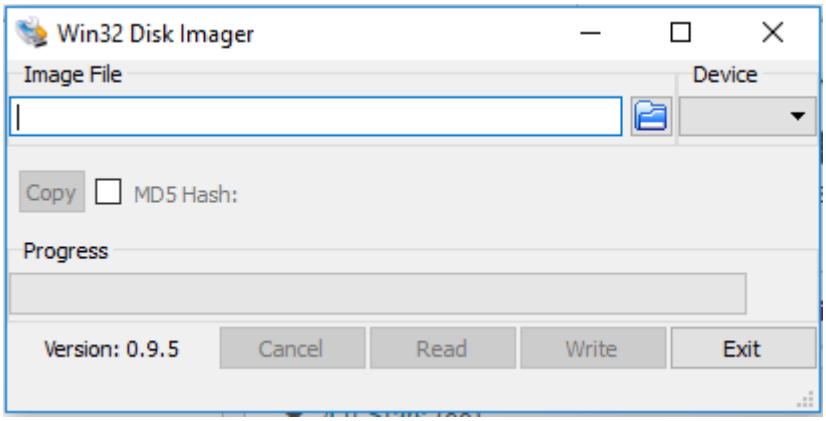
https://downloads.raspberrypi.org/raspbian_latest

A continuación, vamos a proceder a la instalación del sistema operativo en la tarjeta Micro SD usando el programa "*Win32 Disk Imager*", el cual se puede descargar a través de la siguiente pagina web:

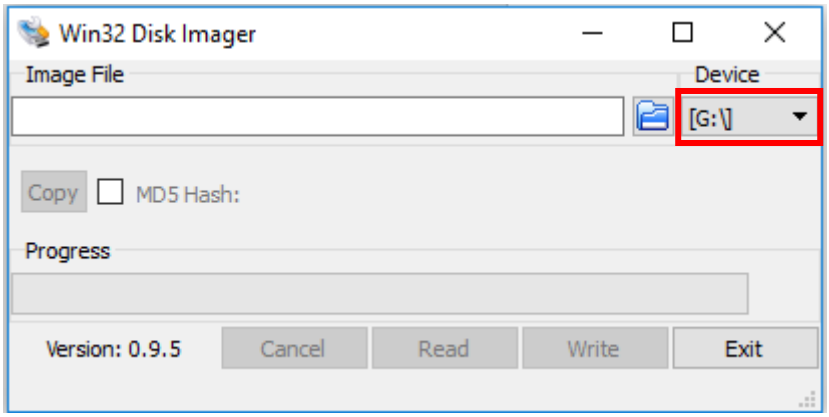
https://sourceforge.net/projects/win32diskimager/

Grabación del sistema operativo Raspbian Jessie en la tarjeta Micro SD

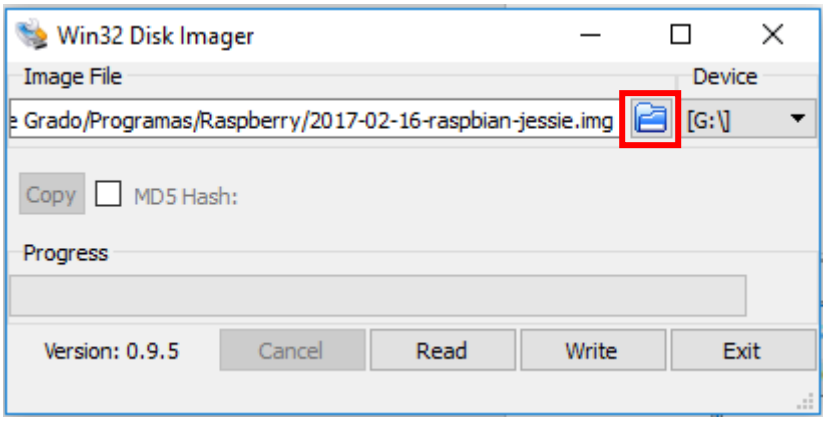
1º - Ejecutamos el programa Win32 Disk Imager:



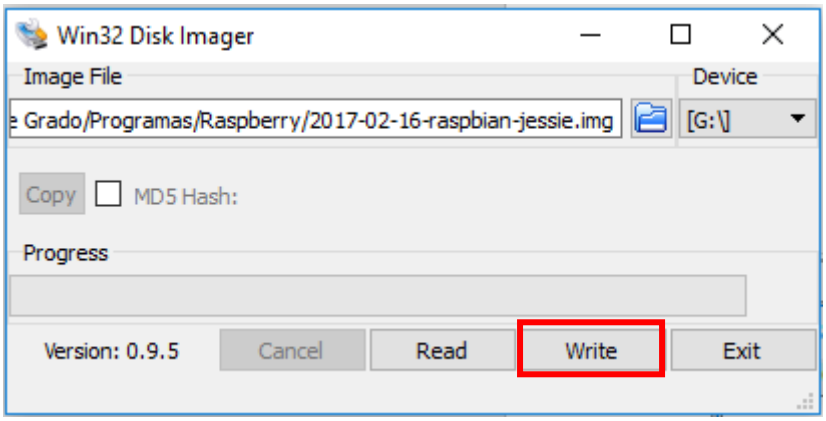
2º - Seleccionamos la unidad de la tarjeta Micro SD introducida en nuestro ordenador:



3º - Buscamos la imagen iso descargada anteriormente del sistema operativo Raspbian Jessie:



4º - Presionamos el botón "*Write*" y esperamos a que finalice la grabación:



Preparación de la Raspberry Pi 2

Una vez insertemos la tarjeta Micro SD en la Raspberry Pi 2 y la encendamos, automáticamente se ejecutará el sistema operativo Raspbian Jessie en su versión Pixel.

A partir de ahora vamos a trabajar con el "*Terminal*" de la Raspberry Pi 2 y siempre que se use "*>> pi@raspberrypi $*" en una línea significará que lo que viene detrás se está escribiendo en el "*Terminal*" que tiene el siguiente aspecto:

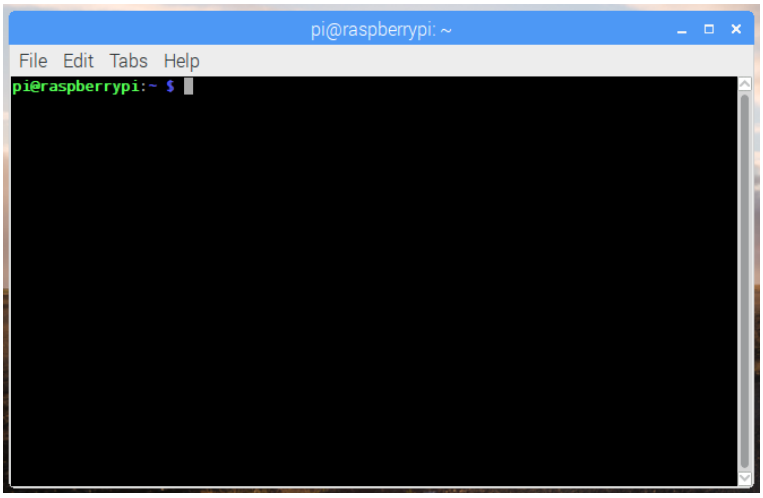


Imagen 19. Terminal de la Raspberry Pi 2.

Una vez abierto el Terminal de la Raspberry Pi 2 procedemos a actualizarla:

>> pi@raspberrypi $ sudo apt-get update
>> pi@raspberrypi $ sudo apt-get upgrade

sudo apt-get update actualiza los repositorios (ve si hay algo nuevo), es decir, actualiza la lista de todos los paquetes, con la dirección de dónde obtenerlos para que, a la hora de hacer la búsqueda y su posterior descarga, sea más rápida.

sudo apt-get upgrade actualiza nuestro sistema con todas las posibles actualizaciones que pudiera haber, es decir, no sólo actualiza nuestro sistema operativo, sino que también las aplicaciones que están contenidas en los repositorios.

Para proporcionar acceso como superusuario se pone delante la palabra ***sudo***.

4.5.2. Activación VNC Viewer y cambio de resolución

Para poder conectarse a la Raspberry Pi 2 desde nuestro ordenador se tiene que activar VNC Viewer a través del **menú Inicio-Preferencias-Configuración de Raspberry Pi**.

Le damos click en la pestaña **Interfaces** y activamos VNC Viewer:

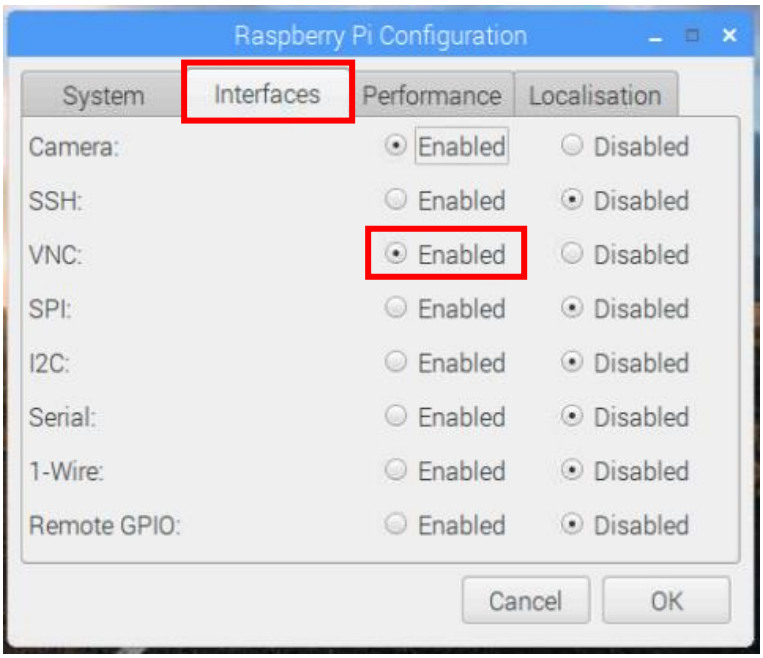


Imagen 20. Configuración de la Raspberry Pi 2.

A continuación, vamos a cambiar la resolución de nuestra Raspberry Pi 2 para que cuando nos conectemos a través del programa VNC Viewer nos muestre la misma resolución que tiene nuestro ordenador.

Abrimos el Terminal de la Raspberry Pi 2 y abrimos el documento *config.txt*:

```
>> pi@raspberrypi $ sudo nano /boot/config.txt
```

Quitamos los comentarios a las siguientes líneas de texto:

```
framebuffer_width=1280
framebuffer_height=720
```

Guardamos (Ctrl + O), salimos del documento (Ctrl + X) y reiniciamos la Raspberry Pi 2 ejecutando:

```
>> pi@raspberrypi $ sudo reboot
```

4.5.3. Conectarse a una red WiFi

Para proporcionar a la Raspberry Pi 2 de conexión internet se utiliza un WiFi USB de la marca EDUP el cual lleva el chip interno de la marca Ralink.

Lo primero que vamos a hacer es instalar los drivers de Ralink para que la Raspberry Pi 2 reconozca el WiFi USB. Para ello utilizamos el siguiente comando:

```
>> pi@raspberrypi $ sudo apt-get install ralink
```

Seguidamente, abrimos el archivo de las interfaces de la conexión a internet.

```
>> pi@raspberrypi $ sudo nano /etc/network/interfaces
```

Una vez abierto, tenemos que escribir lo siguiente y guardarlo:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
    wpa-ssid "mi SSID"
    wpa-psk "mi contraseña"
```

Al escribir esto automáticamente se conectará a la red WiFi que le hemos indicado poniendo el *SSID* y la *contraseña*.

4.5.4. Instalación compilador g++

G++ es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran, también llamado GCC (GNU Compiler Collection). Es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.

Para instalarlo solamente tenemos que escribir la siguiente línea:

```
>> pi@raspberrypi $ sudo apt-get install g++
```

4.5.5. Instalación CMAKE

CMake es una familia de herramientas de código abierto y multiplataforma diseñada para construir, probar y empaquetar software.

Para poder instalar CMake primero vamos a descargar la última versión disponible (cmake-3.7.2.): https://cmake.org/download/

Desde la Raspberry Pi 2 tenemos que introducir la siguiente línea:

```
>> pi@raspberrypi $ wget https://cmake.org/files/v3.7/cmake-3.7.2.tar.gz
```

Una vez descargada vamos a descomprimir y extraer los archivos:

```
>> pi@raspberrypi $ tar –xvzf cmake-3.7.2.tar.gz
```

Por último, procederemos a compilar e instalar CMake:

```
>> pi@raspberrypi $ cd cmake-3.7.2/
>> pi@raspberrypi/cmake-3.7.2 $ sudo ./bootstrap
>> pi@raspberrypi/cmake-3.7.2 $ sudo make
>> pi@raspberrypi/cmake-3.7.2 $ sudo make install
```

4.5.6. Instalación librería RaspiCam

Como bien se ha dicho antes, se ha optado por utilizar la libería RaspiCam, la cual ha sido desarrollada por profesores de la Universidad de Cordoba y que se centra en la captura de imágenes para posteriormente ser tratadas.

Lo primero que tenemos que hacer es descargar la última versión disponible (raspicam-0.1.6.): https://sourceforge.net/projects/raspicam/files/

Para ello, vamos a crear una carpeta llamada TFG para usarla durante el desarrollo de nuestra aplicación.

```
>> pi@raspberrypi $ mkdir TFG
>> pi@raspberrypi $ cd TFG/
```

Usando la siguiente línea descargaremos la librería Raspicam-0.1.6. desde la Raspberry Pi 2:

```
>> pi@raspberrypi/TFG $ wget https://sourceforge.net/projects/raspicam/files/raspicam-0.1.6.zip
```

Una vez descargada, vamos a descomprimirla y extraer los archivos:

```
>> pi@raspberrypi/TFG $ unzip raspicam-0.1.6.zip
```

Construimos la librería Raspicam-0.1.6. usando CMake:

```
>> pi@raspberrypi/TFG $ cd raspicam-0.1.6/
>> pi@raspberrypi/TFG/raspicam-0.1.6/ $ mkdir build
>> pi@raspberrypi/TFG/raspicam-0.1.6/ $ cd build
>> pi@raspberrypi/TFG/raspicam-0.1.6/build/ $ cmake ..
```

Finalmente compilamos, instalamos y actualizamos ldconfig:

```
>> pi@raspberrypi/TFG/raspicam-0.1.6/build/ $ make
>> pi@raspberrypi/TFG/raspicam-0.1.6/build/ $ sudo make install
>> pi@raspberrypi/TFG/raspicam-0.1.6/build/ $ sudo ldconfig
```

ldconfig crea los vínculos necesarios y la caché para las bibliotecas compartidas más recientes encontradas en los directorios especificados en la línea de comandos, en el archivo /etc/ld.so.conf, y en los directorios de confianza (/usr/lib y /lib). **ldconfig** comprueba los nombres de cabezera y los nombres de los archivos de las bibliotecas.

4.5.7. Instalación librería OpenCV

Tal y como se ha especificado anteriormente, las librerías utilizadas para el tratamiento de las imágenes capturadas son las librerías OpenCV.

OpenCV (Open Source Computer Vision Library) es una biblioteca de software de visión artificial. Fue construido para proporcionar una infraestructura común para aplicaciones de visión por ordenador y para acelerar el uso de la percepción de la máquina en los productos comerciales.

Lo primero que hay que hacer es instalar los paquetes requeridos por OpenCV:

Compilador:
>> pi@raspberrypi $ sudo apt-get install build-essential

Requeridos:
>> pi@raspberrypi $ sudo apt-get install cmake pkg-config libgtk2.0-dev libgtk2.0 zlib1g-dev libavcodec-dev swig unzip

Opcionales:
>> pi@raspberrypi $ sudo apt-get install python-dev libpng-dev libjpeg-dev libtiff-dev libjasper-dev

Seguidamente vamos a descargar la última versión disponible (opencv-3.2.0.): https://sourceforge.net/projects/opencvlibrary/files/opencv-unix/3.2.0/

>> pi@raspberrypi $ cd TFG/
>> pi@raspberrypi/TFG $ wget https://sourceforge.net/projects/opencvlibrary/files/opencv-unix/3.2.0/opencv-3.2.0.zip

Descomprimimos y extraemos los archivos:

>> pi@raspberrypi/TFG $ unzip opencv-3.2.0

Construimos OpenCV-3.2.0. usando CMake:

>> pi@raspberrypi/TFG $ cd opencv-3.2.0/
>> pi@raspberrypi/TFG/opencv-3.2.0/ $ cmake -DCMAKE_BUILD_TYPE=RELEASE -DCMAKE_INSTALL_PREFIX=/usr/local -DBUILD_PERF_TESTS=OFF -DBUILD_opencv_gpu=OFF -DBUILD_opencv_ocl=OFF

Finalmente instalamos OpenCV-3.2.0.:
>> pi@raspberrypi/TFG/ opencv-3.2.0/ $ sudo make
>> pi@raspberrypi/TFG/ opencv-3.2.0/ $ sudo make install

4.5.8. Instalación librería WiringPi

La librería WiringPi está escrita en C y es utilizada para controlar los pins GPIO de la Raspberry Pi 2.

La cosa más importante a tener en cuenta es que la librería WiringPi tiene una numeración propia para los pins GPIO de la Raspberry Pi 2. A continuación se puede ver una tabla de equivalencias para la utilización de los pins GPIO, que nos será útil para comprender parte del programa escrito en C++.

WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11

Tabla 1. Tabla de equivalencias pins GPIO WiringPi.

Vamos a proceder a la instalación de la librería WiringPi. Para ello empezaremos por descargar la librería https://git.drogon.net/?p=wiringPi;a=summary

Esta vez la tenemos que descargar desde el ordenador porque el formato, el cual va comprimido, no se puede descargar directamente desde la Raspberry Pi 2. Con lo cual, después de descargar el archivo tenemos que copiarlo a la Raspberry Pi 2, y una vez copiado seguiremos con la instalación.

Descomprimimos y extraemos los archivos:

```
>> pi@raspberrypi/TFG $ tar xfz wiringPi-96344ff.tar.gz
```

Finalmente construimos la librería WiringPi:

```
>> pi@raspberrypi/TFG $ cd wiringPi-96344ff/
>> pi@raspberrypi/TFG $ ./build
```

4.6. DESARROLLO DE LA APLICACIÓN SOFTWARE

4.6.1. Consideraciones preliminares

Existen varios aspectos a tener en cuenta, antes de seguir con el desarrollo del código. Uno de los aspectos más importantes es el tema de la compresibilidad de los programas. Lo que se quiere decir es que, a medida que se va desarrollando el programa, este será cada vez más grande y complejo, con lo que la elección de los nombres a usar para las variables y funciones cada vez será más engorroso.

Lo más importante, y que cabe destacar, es que cuando un programa alcanza cierto tamaño, es funcional dividirlo en partes más pequeñas.

Para desarrollar el código, se ha partido de un ejemplo proporcionado por la librería RaspiCam, la cual usa una cámara para adquirir imágenes, siendo al mismo tiempo compatible con la utilización de la librería OpenCV. Una vez las imágenes hayan sido obtenidas, y ya dentro del programa desarrollado, éstas son tratadas y procesadas para el propósito que se desea, tal y como se explicará más adelante mediante la librería OpenCV.

Tras numerosas pruebas, el código elegido como base es *simpletest_raspicam_cv*, el cual ha sido desarrollado por la Universidad de Cordoba y que se caracteriza por tomar imágenes, sacar unos tiempos estadísticos y guardar las imagenes usando la cámara *Raspberry Pi Camera Board* y la librería RaspiCam.

4.6.2. Desarrollo del código de la aplicación

Antes de empezar la explicación del código, se van a crear dos carpetas, una dentro de otra, de tal modo que estas queden dentro de la carpeta general que hemos creado anteriormente llamada *"TFG"*.

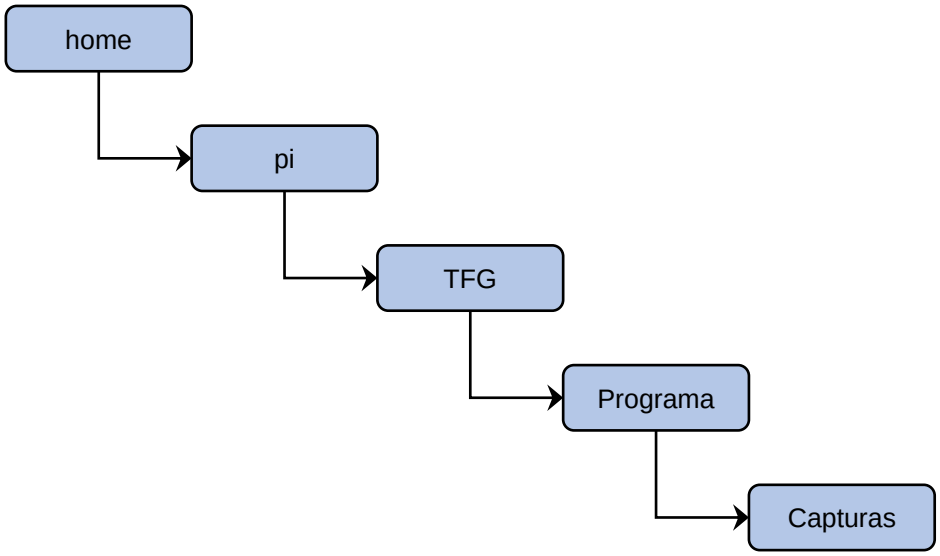


Diagrama 1. Carpetas de la Raspberry Pi 2.

En la carpeta *"Programa"*, se escribirá el código que se explicará a continuación, y que se accederá a él por medio del editor de texto NANO escribiendo la siguiente línea:

>> pi@raspberrypi/TFG/Programa $ sudo nano tfg.cpp

Dentro de la carpeta *"Capturas"*, se guardarán las imágenes capturadas y modificadas durante la ejecución del código.

Una vez aclarado esto, se van a describir los pasos principales que se hacen en este, con lo que conseguiremos un mayor entendimiento cuando se proceda a explicar el código "*in situ*".

El programa empieza capturando una imagen y guardándola como imagen_RGB.

Seguidamente se recorta la imagen, de modo que siempre nos centraremos en la parte de la imagen en la cual podemos sacarle información. Esta imagen se guarda como imgen_CORTADA.

Una vez la imagen está cortada, se procede a aplicarle un filtro llamado *Filtro Canny*, el cual detecta los bordes por medio de su función dejando toda la imagen, menos los bordes, de color negro, y posteriormente se guarda como imagen_CANNY. Imagen_CANNY es una imagen RGB.

Lo siguiente que se hace para que se vean mejor las líneas, es utilizar una función que dibuja los contornos de la imagen_CANNY. Estos contornos son pintados de color rojo, con lo que la imagen guardada como imagen_CONTORNOS es una imagen RGB.

Para poder normalizar un color que pueda ser detectado por su tonalidad lo que se hace es, convertir la imagen_CONTORNOS a una imagen con escala de grises, la cual se guarda como imagen_GRAY.

Después de hacer todo el proceso del tratamiento de la imagen, lo que se hace es encontrar el centro de la imagen_GRAY y posteriormente encontrar las líneas grises de esta por medio de un proceso, el cual recorre toda la imagen en busca de los pixeles de color gris.

Finalmente, se aplica el movimiento de los motores de acuerdo a lo que se ha detectado anteriormente.

Para que quede más claro, a continuación, se encuentra un diagrama con todos los procesos que se llevan a cabo de forma general.

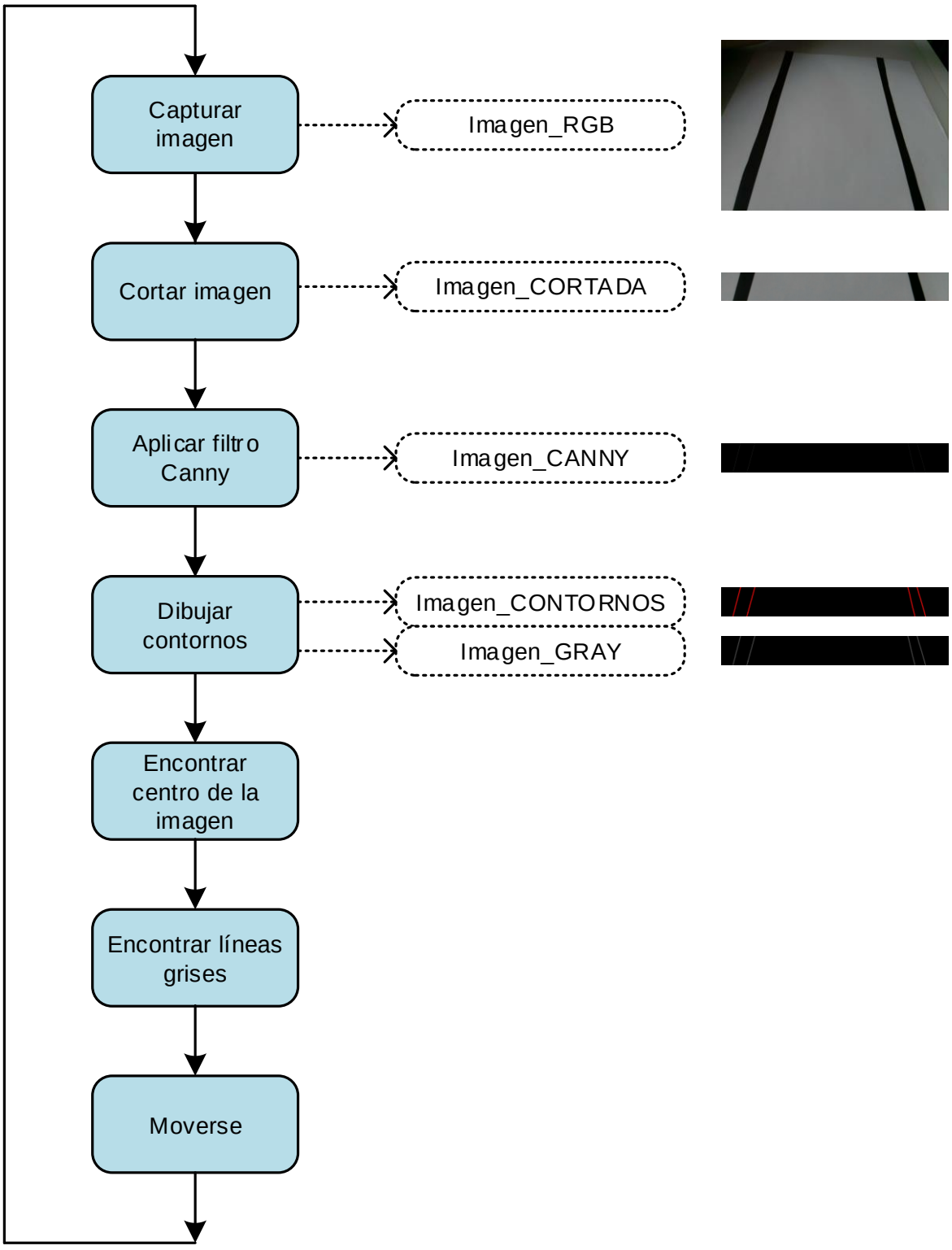


Diagrama 2. Pasos principales del programa.

Explicación del código

- **Librerías**

Lo primero que se hace es incluir todas las librerías y ficheros necesarios. Las librerías son cierto tipo de archivos que podemos importar o incluir en nuestro programa. Estos archivos contienen las especificaciones de diferentes funcionalidades ya construidas y utilizables que podremos agregar a nuestro programa, como por ejemplo leer del teclado o mostrar algo por pantalla entre muchas otras más.

La llamada de las librerías se debe hacer al principio de todo nuestro código, antes de la declaración de cualquier función o línea de código. Para ello usaremos la siguiente sintaxis: *#include <nombre de la librería o fichero>* o alternativamente ***#include** "nombre de la librería o fichero"*.

Al utilizar la palabra *include*, lo que hacemos es incluir nuestras librerías o ficheros dentro de nuestro código, de modo que cuando nuestro programa encuentre una línea que ponga #include "fichero"**,** entonces reemplazará esta línea por la librería o el fichero incluido.

```
#include <ctime>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <raspicam/raspicam_cv.h>
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <wiringPi.h>
#include <softPwm.h>
```

Librería "ctime": Este archivo de encabezado contiene definiciones de funciones para obtener y manipular información sobre la fecha y la hora. En este trabajo es necesaria para calcular una estadística del tiempo que tarda en capturar una imagen.

Librería "iostream": Encabezado que define los objetos de entrada / salida estándares. Incluyendo <iostream> automaticamente también se incluyen las librerías <ios>, <streambuf>, <istream>, <ostream> i <iosfwd>.

Librería "stdio.h": Las operaciones de entrada y salida también se pueden realizar en C++ usando la biblioteca de entrada y salida estándar C (cstdio, conocida como stdio.h en el lenguaje C). Esta biblioteca utiliza lo que se llama flujos para operar con dispositivos físicos como teclados, impresoras, terminales o con cualquier otro tipo de archivos soportados por el sistema.

Librería "stdlib.h" (cstdlib): Esta cabecera define varias funciones de propósito general, incluyendo gestión de memoria dinámica, generación de números aleatorios, comunicación con el entorno, aritmética entera, búsqueda, clasificación y conversión.

Librería "raspicam/raspicam_cv.h": Las operaciones para capturar las imágenes se realizan usando la librería raspicam_cv. Esta librería se utiliza para tener un control total y sencillo de la cámara con OpenCV.

Librería "opencv2/core/core.hpp": Esta biblioteca es utilizada para el tratamiento de imágenes con OpenCV.

Librería "opencv2/highgui/highgui.hpp": Esta biblioteca es utilizada para el tratamiento de imágenes con OpenCV.y es necesaria para aplicar el filtro Canny y para dibujar contornos.
Librería "opencv2/imgproc/imgproc.hpp": Esta biblioteca es utilizada para el tratamiento de imágenes con OpenCV. y es necesaria para aplicar el filtro Canny y para dibujar contornos.

Librería "wiringPi.h": Esta librería contiene todas las funciones necesarias para el control de los pines GPIO de la Raspberry Pi 2.

Librería "softPwm.h": Esta librería contiene funciones para la realización de señales PWM.

- **Definir motores**

Lo siguiente que se hace es definir las variables de los motores:

```
// Pins de los motores del coche
#define MotorDelante 4                  // Pin 16 (GPIO 23)
#define MotorDetras 7                   // Pin 7 (GPIO 4)
#define MotorEnableDetras 0             // Pin 11 (GPIO 17)
```

Para definir las variables necesarias para el accionamiento de los motores se utiliza la siguiente sintaxis: *#define -Nombre de la variable- -Nº correspondiente al pin GPIO que se quiere utilizar-*
El número que se tiene que poner para utilizar un pin GPIO se tiene que ver en la tabla expuesta anteriormente en el apartado "***Instalación librería WiringPi***". Como bien se ha dicho anteriormente, este número está definido de acuerdo a la librería WiringPi.

- **Using namespace**

Declaramos los espacios de nombres necesarios:

```
using namespace std;
using namespace cv;
```

El lenguaje de programación C++ utiliza palabras reservadas en un espacio de nombres (namespace). Para poder utilizar estas palabras es necesario declararlo al principio del programa. Algunas palabras del *namespace std* podrían ser, *cout* y *cin*.

Si quisiéramos utilizar *cout* sin declarar el *namespace std*, tendríamos que escribir cada vez que usemos *cout* lo siguiente:

```
std::cout << "Hola";
```

Para solucionar esto, lo que se hace es declararlo al principio, tal y como se ha descrito antes, y así cada vez que queramos utilizar *cout* solo tendríamos que escribir:

```
cout << "Hola";
```

En este caso solamente necesitamos declarar los espacios de nombres del lenguaje de programación C++ y de la librería de OpenCV.

- **Variables utilizadas para capturar imágenes**

```
// Variables Capturar Imagen
time_t timer_begin,timer_end;
raspicam::RaspiCam_Cv Camera;
Mat imagen_RGB;
int nCount = 5;
```

time_t, timer_begin y timer_end: Son variables utilizadas para mostrar una estadística del tiempo que tarda en capturar una imagen.

Raspicam::RaspiCam_Cv Camera: *Camera* es una variable utilizada para ejecutar la cámara y hacer fotos, que como bien se puede observar se sitúa dentro del espacio de nombres de la librería *raspicam_cv*.

Mat imagen_RGB: *imagen_RGB* es una variable, la cual es del tipo *Mat* que proviene de la librería OpenCV. *Mat* es básicamente una clase con dos partes de datos: la matriz de encabezado (que contiene información como el tamaño de la matriz, el método utilizado para almacenar, en qué dirección se almacena la matriz, etc.) y un puntero de la matriz que contiene los valores de los píxeles (tomando cualquier dimensionalidad dependiendo del método elegido para almacenar). El tamaño de la matriz de encabezado es constante, sin embargo, el tamaño de la propia matriz puede variar de una imagen a otra y generalmente es mayor en órdenes de magnitud.
En este caso *Mat* es utilizada para una variable que guardará una imagen en la memoria de la Raspberry Pi 2.

int nCount = 5: *nCount* es una variable de tipo entero que es utilizada para capturar un numero de frames, los cuales servirán para formar una imagen.

- **Variables utilizadas para cortar la imagen**

```
// Variables Cortar Imagen
Mat imagen_CORTADA;
int xMin = 0;
int xMax = 1279;
int yMin = 800;
int yMax = 959;
```

Mat imagen_CORTADA: *imagen_CORTADA* es una variable, la cual es del tipo *Mat* que proviene de la librería OpenCV. *Mat* es básicamente una clase con dos partes de datos: la matriz de encabezado (que contiene información como el tamaño de la matriz, el método utilizado para almacenar, en qué dirección se almacena la matriz, etc.) y un puntero de la matriz que contiene los valores de los píxeles (tomando cualquier dimensionalidad dependiendo del método elegido para almacenar). El tamaño de la matriz de encabezado es constante, sin embargo, el tamaño de la propia matriz puede variar de una imagen a otra y generalmente es mayor en órdenes de magnitud.
En este caso *Mat* es utilizada para una variable que guardará una imagen en la memoria de la Raspberry Pi 2.

int xMin, xMax, yMin, yMax: Variables que tienen los píxeles del rectángulo que define la región a cortar de la imagen.

- **Variables utilizadas para aplicar el filtro Canny (también llamado Sobel)**

```
// Variables Filtro Canny
Mat src_gray, imagen_CANNY, detected_edges;
int lowThreshold = 40;
int ratio = 2;
int kernel_size = 3;
```

Mat src_gray, imagen_CANNY, detected_edges: Variables del tipo *Mat* que proviene de la librería OpenCV. *Mat* es básicamente una clase con dos partes de datos: la matriz de encabezado (que contiene información como el tamaño de la matriz, el método utilizado para almacenar, en qué dirección se almacena la matriz, etc.) y un puntero de la matriz que contiene los valores de los píxeles (tomando cualquier dimensionalidad dependiendo del método elegido para almacenar). El tamaño de la matriz de encabezado es constante, sin embargo, el tamaño de la propia matriz puede variar de una imagen a otra y generalmente es mayor en órdenes de magnitud.
En este caso *Mat* es utilizada para dos variables (*src_gray, detected_edges*) que guardarán dos imágenes de forma temporal y otra variable (*imagen_CANNY*) que guardará una imagen en la memoria de la Raspberry Pi 2.

int lowThreshold: Variable que define el umbral mínimo para la función Canny.

int ratio: Variable que define la relación entre el umbral máximo y mínimo para la función Canny.

int kernel_size: Variable que define el tamaño de apertura del operador Canny.

- **Variables utilizadas para dibujar los contornos**

```
// Variables Dibujar Contornos
Mat imagen_find, imagen_CONTORNOS, imagen_GRAY;

    // Variables para findContours
    int modo = CV_RETR_TREE;
    int metodo = CV_CHAIN_APPROX_SIMPLE;

    // Variables para drawContours
    int grosorLinea = 3;
    int conectividadLinea = 8;
    int maxNivelContornos = 0;
```

Mat imagen_find, imagen_CONTORNOS, imagen_GRAY: Variables del tipo *Mat* que proviene de la librería OpenCV. *Mat* es básicamente una clase con dos partes de datos: la matriz de encabezado (que contiene información como el tamaño de la matriz, el método utilizado para almacenar, en qué dirección se almacena la matriz, etc.) y un puntero de la matriz que contiene los valores de los píxeles (tomando cualquier dimensionalidad dependiendo del método elegido para almacenar). El tamaño de la matriz de encabezado es constante, sin embargo, el tamaño de la propia matriz puede variar de una imagen a otra y generalmente es mayor en órdenes de magnitud.
En este caso *Mat* es utilizada para dos variables (*imagen_CONTORNOS, imagen_GRAY*) que guardarán dos imágenes en la memoria de la Raspberry Pi 2 y otra variable (*imagen_find*) que guardará una imagen de forma temporal.

int modo: Variable que especifica el modo de recuperación del contorno de la función *findContours()*.

int metodo: Variable que especifica el método de aproximación del contorno de la función *findContours()*.

Int grosorLinea, conectividadLinea, maxNivelContornos: Variables que describen como serán las líneas que se dibujan con la función *drawContours()*.

- **Variables utilizadas para encontrar el centro de la imagen**

```
// Variables Encontrar centro de la imagen
int centroImagen;
```

Para encontrar el centro de la imagen, que para nosotros sería la referencia, primero se crea una variable llamada *centroImagen*. Esta variable almacenará el pixel central en horizontal de la imagen capturada.

- **Variables utilizadas para encontrar la línea izquierda y derecha**

```
// Variables Encontrar Lineas Izquierda y Derecha
int y, xIzq, xDer;
int vectorPixelesIzq[2];
int posVectorIzq = 0;
int vectorPixelesDer[2];
int posVectorDer = 0;
int salirBucle = 0;
int lineaIzqEncontrada = 0; // 0 -> NO ENCONTRADA; 1 -> ENCONTRADA
int lineaDerEncontrada = 0; // 0 -> NO ENCONTRADA; 1 -> ENCONTRADA
```

int y, xIzq, xDer: Variables que almacenarán la posición de los pixeles donde ha encontrado una línea izquierda y otra derecha.

int vectorPixelesIzq[2]: Vector que almacenará la posición de los pixeles de la línea izquierda encontrada.

int posVectorIzq: Variable que indica la posición del vector descrito en el párrafo anterior, donde se guardará la posición del pixel encontrado perteneciente a la línea izquierda.

int vectorPixelesDer[2]: Vector que almacenará la posición de los pixeles de la línea derecha encontrada.

int posVectorDer: Variable que indica la posición del vector descrito en el párrafo anterior, donde se guardará la posición del pixel encontrado perteneciente a la línea derecha.

int salirBucle: Variable que se utiliza para indicar la salida del bucle donde se procede a buscar la línea izquierda y derecha. Cuando esta variable valga 1, saldrá del bucle donde busca las líneas.

lineaIzqEncontrada y **lineaDerEncontrada:** Estas variables nos confirmarán si ha encontrado la línea izquierda y la línea derecha.

- **Variables utilizadas para encontrar el centro de las líneas**

```
// Variables Encontrar Centro Lineas
int centroPixeles;
```

Para encontrar el centro que forman las líneas encontradas, que para nosotros sería la posición actual, primero se crea una variable llamada *centroPixeles*. Esta variable almacenará el pixel central a las dos líneas encontradas de la imagen capturada.

- **Variables utilizadas para crear la señal PWM que moverá el servomotor**

```
// Variables señal PWM
int tiempoCicloPwm = 200;
int tiempoPulsPwm = 15;
```

int tiempoCicloPwm: Esta variable nos indica el tiempo del ciclo de la señal PWM. Un valor de 200 significa que el ciclo de la señal es de 20 ms.

int tiempoPulsPwm: Esta variable nos indica el tiempo el cual hace referencia a la señal PWM cuando esta está HIGH. Se inicializa en un valor de 15 porque es el centro de las ruedas delanteras, aunque más adelante veremos que cambiará de valor según la posición del vehículo.
Un valor de 15 significa un tiempo de pulsación de 1,5 ms.

- **Variables utilizadas para calcular el tiempo de ejecución del programa**

```
// Variables calculo tiempo programa
clock_t start, end;
float tiempoActual, sumaTiempoActual, tiempoMedio, tiempoMaximo, numCapturas;
```

clock_t start, end: Estas variables son las encargadas de medir el tiempo al principio del programa y al final del programa.

float tiempoActual, sumaTiempoActual, tiempoMedio, tiempoMaximo, numCapturas: Estas variables se usan para realizar el cálculo del tiempo de ejecución del programa, incluyendo el tiempo medio y el tiempo máximo durante la ejecución del mismo.

- **Declaración de funciones**

```
/*********************************************/
/*          DECLARACION DE FUNCIONES         */
/*********************************************/

void CapturarImagen();
void CortarImagen();
void FiltroCanny();
void DibujarContornos();
void EncontrarLineas();
void EncontrarCentroLineas();
void CalcularTiempoPulsPwm();
void Moverse();
void MoverMotorDelante();
void MoverMotorDetras();
```

Una de las cosas que hace mejorar un código es la división de este en partes.
Lo que se ha llevado a cabo en este trabajo es la separación del código en diferentes partes, sin hacer otros archivos, con lo que conseguimos tener una mayor organización y a la hora de trabajar en el código se hace mucho más ameno. Sin embargo, para poder mejorar la compilación del código se podrían crear diversos archivos, y estos ser llamados por un archivo principal.

En esta parte del código se declaran las funciones y se especifica el tipo de retorno que van a tener. En este caso son funciones `void`, lo que quiere decir que son funciones que no retornan ningún valor, o mejor dicho, que retornan un valor nulo.

Una función es un conjunto de instrucciones que se pueden llamar desde cualquier parte del programa.

- **Función Main**

A continuación, se presenta la función Main completa, y siguiendo a esta su explicación.

```
/*************************************************/
/*                FUNCION MAIN                   */
/*************************************************/
int main ( )
{

        cout << "\n************************************************" <<
                "\n*           CONFIGURANDO MOTORES ...           *" <<
                "\n************************************************\n" << endl;

        // Configurar parametros de los motores
        if (wiringPiSetup() == -1)
        {
               cout << "Fallo la configuracion de wiringPi !" << endl;
               return 1;
        }

        pinMode(MotorDelante, OUTPUT);
        pinMode(MotorDetras, OUTPUT);
        pinMode(MotorEnableDetras, OUTPUT);

        // Poner las ruedas delanteras en posicion central
        digitalWrite(MotorDelante, LOW);
        softPwmCreate(MotorDelante, 0, tiempoCicloPwm);
        softPwmWrite(MotorDelante, 15);
        delay(1000);   // milisegundos
        digitalWrite(MotorDelante, LOW);


        cout << "\n************************************************"<<
                "\n*      CONFIGURANDO Y ABRIENDO CAMARA ...      *"<<
                "\n************************************************\n"<<endl;

        // Configurar parametros de la camara
        Camera.set(CV_CAP_PROP_FORMAT, CV_8UC3);
```

```
/* CV_8UC1: Escala de grises. 1 byte por color.
CV_8UC3: Modo color RGB. 3 bytes por color. */

// Abriendo camara
cout << "Abriendo camara..." << endl;
if (!Camera.open())
{
	cerr << "Error abriendo la camara" << endl;
	return -1;
}

while(1)
{

	start = clock();

	/*********************************************/
	/*               CAPTURAR IMAGEN             */
	/*********************************************/

	// Llamar a la funcion Capturar Imagen
	CapturarImagen();  //Llamada de la funcion CapturarImagen

	// Mostrar estadisticas del tiempo
	time ( &timer_end );
	double secondsElapsed = difftime ( timer_end,timer_begin );
	//cout<< secondsElapsed<<" segundos para "<< nCount<<"  frames : FPS =
	"<<  ( float ) ( ( float ) ( nCount ) /secondsElapsed ) <<endl;

	// Guardar imagen RGB
	imwrite("/home/pi/TFG/Programa/Capturas/imagen_RGB.jpg",imagen_RGB);


	/*********************************************/
	/*                CORTAR IMAGEN                */
	/*********************************************/

	// Llamar a la funcion Cortar Imagen
	CortarImagen();

	// Guardar imagen cortada
	imwrite("/home/pi/TFG/Programa/Capturas/imagen_CORTADA.jpg",
	imagen_CORTADA);


	/*********************************************/
	/*            APLICAR FILTRO CANNY           */
	/*********************************************/

	// Llamar a la funcion Filtro Canny
	FiltroCanny();

	// Guardar imagen con filtro canny
	imwrite("/home/pi/TFG/Programa/Capturas/imagen_CANNY.jpg",
	imagen_CANNY);
```

```
/*********************************************/
/*              DIBUJAR CONTORNOS              */
/*********************************************/

// Llamar a la funcion Dibujar Contornos
DibujarContornos();

// Guardar imagen con contornos dibujados en rojo
imwrite("/home/pi/TFG/Programa/Capturas/imagen_CONTORNOS.jpg",
imagen_CONTORNOS);

// Guardar imagen con contornos dibujados en escala de grises
imwrite("/home/pi/TFG/Programa/Capturas/imagen_GRAY.jpg",imagen_GRAY);


/*********************************************/
/*        ENCONTRAR CENTRO DE LA IMAGEN      */
/*********************************************/

// Centro de la imagen con margenes
centroImagen = imagen_GRAY.cols/2;


/*********************************************/
/*     ENCONTRAR LINEAS GRISES Y SU CENTRO   */
/*********************************************/

// Llamar a la funcion Encontrar Lineas Izquierda y Derecha
EncontrarLineas();

// Llamar a la funcion Encontrar Centro Lineas
EncontrarCentroLineas();


/*********************************************/
/*                  QUE HACER                */
/*********************************************/

// Llamar a la funcion Calcular Tiempo Pulsacion PWM
CalcularTiempoPulsPwm();

// Llamar a la funcion que hace mover el servomotor
MoverMotorDelante();

// Llamar a la funcion que hace mover el motor trasero
MoverMotorDetras();


/*********************************************/
/*          CALCULOS TIEMPOS PROGRAMA        */
/*********************************************/

end = clock();
tiempoActual = end - start; //en microsegundos
numCapturas = numCapturas + 1;
sumaTiempoActual = sumaTiempoActual + tiempoActual; //en microsegundos
```

```
            tiempoMedio = (sumaTiempoActual / numCapturas); //en microsegundos

            if (tiempoActual > tiempoMaximo)
            {
                  tiempoMaximo = tiempoActual; //en microsegundos
            }


            /*********************************************/
            /*         MOSTRAR VALORES POR PANTALLA      */
            /*********************************************/

            cout << "\nlineaIzqEncontrada = " << lineaIzqEncontrada << endl;
            cout << "\nlineaDerEncontrada = " << lineaDerEncontrada << endl;
            cout << "\ncentroImagen = " << centroImagen << endl;
            cout << "\ncentroPixeles = " << centroPixeles << endl;
            cout << "\ntiempoPulsPwm = " << tiempoPulsPwm << endl;
            cout << "\nNumero de capturas: " << numCapturas << endl;
            cout << "\nTiempo actual: " << tiempoActual / 1000 << " milisegundos"
            << endl;
            cout << "\nTiempo medio: " << tiempoMedio / 1000 << " milisegundos"
            << endl;
            cout << "\nTiempo maximo: " << tiempoMaximo / 1000 << " milisegundos"
            << endl;

      }

    cout << "\nParar camara..." << endl;
    Camera.release();

    return 0;
}
```

- **Explicación función Main**

```
            /*********************************************/
            /*           CONFIGURANDO MOTORES ...        */
            /*********************************************/
```

Al principio de esta parte lo que se hace es comprobar si la configuración de los parámetros de los motores se llevaron a cabo de forma correcta a través de la librería *wiringPi*. En el caso de que sucediera una mala configuración, el código lo mostraría por pantalla.

En las siguientes líneas se configuran los pines correspondientes al movimiento de los motores como pines de salida.

```
      pinMode(MotorDelante, OUTPUT);
      pinMode(MotorDetras, OUTPUT);
      pinMode(MotorEnableDetras, OUTPUT);
```

Y finalmente, el servomotor que controla las ruedas delanteras se mueve a la posición central con las siguientes líneas:

1º - Le indicamos al pin que controla el servomotor que ponga la salida a 0 V (LOW).

```
digitalWrite(MotorDelante, LOW);
```

2º - Creamos la señal PWM, la cual tiene un tiempo de ciclo de 20 ms, y que es indicado a través de la variable `tiempoCicloPwm`.

```
softPwmCreate(MotorDelante, 0, tiempoCicloPwm);
```

3º - Enviamos la señal PWM a través del pin que controla el servomotor, la cual poniendo el número 15 significa que la señal debe permanecer en HIGH durante 1,5 ms.

```
softPwmWrite(MotorDelante, 15);
```

4º - Se proporciona una espera de 1 segundo y luego se vuelve a poner el pin que le envía la señal al servomotor a 0 V (LOW).

```
delay(1000);   // milisegundos
digitalWrite(MotorDelante, LOW);
```

```
/*********************************************/
/*      CONFIGURANDO Y ABRIENDO CAMARA ...     */
/*********************************************/
```

Esta parte es para configurar los parámetros de la cámara para posteriormente ser abierta. Primero se configuran los parámetros de la cámara escribiendo `Camera.set(CV_CAP_PROP_FORMAT, CV_8UC3) en una línea.`

Según las librerías OpenCV y Raspicam si ponemos en la línea anterior `CV_CAP_PROP_FORMAT` y `CV_8UC1`, le estamos diciendo a la cámara que se configure para tomar fotos con escala de grises. 1 byte por color. Sin embargo, si ponemos `CV_8UC3`, le estamos diciendo a la cámara que se configure para tomar fotos con el modo a color RGB. 3 bytes por color.

Seguidamente, se abre la cámara y se comprueba que no ha habido ningún error durante el proceso de apertura de esta.

```
/*********************************************/
/*               CAPTURAR IMAGEN             */
/*********************************************/
```

Lo que se hace en esta parte es llamar a la función que capturará una imagen, posteriormente se muestra una estadística sobre el tiempo que tarda en capturar una imagen, y finalmente guardará la imagen capturada como `imagen_RGB`.

```
/*********************************************/
/*                CORTAR IMAGEN                */
/*********************************************/
```

Luego de capturar la imagen, se llama a la función que cortará la imagen según le hemos indicado al principio del código, en el apartado donde se declaran las variables, y finalmente guardará esta imagen como `imagen_CORTADA`.

```
/*********************************************/
/*            APLICAR FILTRO CANNY           */
/*********************************************/
```

Una vez se ha cortado la imagen según nuestro propósito, se llama a la función que aplicará el filtro Canny (en muchas ocasiones también llamado Sobel) según los parámetros que le hemos indicado al principio del código, en el apartado donde se declaran las variables, y finalmente guardará esta imagen como `imagen_CANNY`.

```
/*********************************************/
/*              DIBUJAR CONTORNOS            */
/*********************************************/
```

En este apartado se llama a la función que, dibujará los contornos encontrados en la imagen anterior y posteriormente la convertirá a escala de grises. Finalmente, se guardarán las dos imágenes. La imagen con los contornos dibujados (`imagen_CONTORNOS`), la cuál es una imagen a color, y la misma imagen, pero con escala de grises (`imagen_GRAY`).

```
/*********************************************/
/*        ENCONTRAR CENTRO DE LA IMAGEN      */
/*********************************************/
```

En la siguiente sección, se calcula el centro de la imagen a través del comando `imagen_GRAY.cols`, el cual cuenta los pixeles en el eje horizontal que tiene la imagen, que anteriormente se ha guardado como imagen_GRAY.

```
/*********************************************/
/*     ENCONTRAR LINEAS GRISES Y SU CENTRO   */
/*********************************************/
```

Esta parte es una de las más importantes del programa. En ella se llama a la función que encuentra la línea de la parte izquierda y la línea de la parte derecha, teniendo en cuenta que pueden o no, aparecer en la imagen.

Una vez haya encontrado las dos líneas, se llama a la función que calcula el centro de las dos.
Si por alguna de las dos líneas no aparece en la imagen, la variable que define el centro de las líneas tomará un valor diferente, tal y como se explicará más adelante.

```
/*********************************************/
/*                 QUE HACER                 */
/*********************************************/
```

Esta parte llama a la función que calcula el tiempo de la pulsación PWM, que variará según donde se encuentre el centro de las líneas detectadas anteriormente, lo que equivale a la posición del vehículo.

Y posteriormente llamaremos a las funciones que mueven el servomotor de las ruedas delanteras y el motor de las ruedas traseras.

```
/*********************************************/
/*          CALCULOS TIEMPOS PROGRAMA        */
/*********************************************/
```

Para finalizar, esta parte lo que hace es calcular los tiempo actual, medio y máximo de ejecución del programa para posteriormente mostrarlo por pantalla.

```
/*********************************************/
/*         MOSTRAR VALORES POR PANTALLA      */
/*********************************************/
```

Esta parte del programa solamente muestra por pantalla datos sobre la posición de las líneas izquierda y derecha encontradas anteriormente, el centro de la imagen (referencia), el centro de las dos líneas encontradas, y el valor de la variable `tiempoPulsPwm`. Esta variable es la que hace que se mueva el servomotor de las ruedas delanteras.

Además, también se muestra el número de capturas realizadas, el tiempo de ejecución actual del programa, el tiempo de ejecución medio del programa y el tiempo de ejecución máximo del programa.

- **Finalización función Main**

Una vez se quiera finalizar el programa, el código mandará parar la cámara para que no provoque errores la siguiente vez que se quiera utilizar.

- **Función capturar imagen**

```
/*********************************************/
/*           FUNCION CAPTURAR IMAGEN         */
/*********************************************/

void CapturarImagen()
{
    //Capturar imagen
    cout<<"Capturando "<<nCount<<" frames ...."<<endl;
    time ( &timer_begin );
    for ( int i=0; i<=nCount; i++ )
    {
        Camera.grab();
        Camera.retrieve (imagen_RGB);
        if ( i%5==0 )  cout<<"\r captured "<<i<<" images"<<std::flush;
    }

    return;
}
```

Explicación

La función empieza poniendo en marcha un temporizador que nos servirá para mostrar una estadística del tiempo que tarda en tomar una imagen:

```
    time ( &timer_begin );
```

Y luego empieza a tomar los frames que formarán una imagen:

```
    for ( int i=0; i<=nCount; i++ )
    {
        Camera.grab();
        Camera.retrieve (imagen_RGB);
        if ( i%5==0 )  cout<<"\r captured "<<i<<" images"<<std::flush;
    }
```

`Camera.grab()` capturará el siguiente frame desde el archivo de vídeo o el dispositivo de captura y lo guardará en el buffer.
`Camera.retrieve` decodificará y devolverá el frame de video grabado.

La imagen capturada tomará el nombre de `imagen_RGB` tal y como se puede observar.

Cuando se pone `i%5==0` en el condicional `if`, lo que hace es verificar si la variable `i` es divisible por 5. Si es divisible, el residuo será 0. De este modo lo que conseguimos es mostrar por pantalla los frames que se van tomando, los cuales irán subiendo de 5 en 5.

- **Función cortar imagen**

```
/*********************************************/
/*            FUNCION CORTAR IMAGEN            */
/*********************************************/

void CortarImagen()
{
    // Configurar un rectangulo para definir la region a cortar
    Rect miRectangulo(xMin, yMin, xMax-xMin, yMax-yMin);

    // Cortar la imagen completa usando el rectángulo definido
    // Observe que esto no copia los datos
    Mat imagenRect (imagen_RGB, miRectangulo);

    // Copiar los datos en la nueva matriz
    imagenRect.copyTo(imagen_CORTADA);

    return;
}
```

Explicación

Empezamos configurando un rectángulo con las dimensiones que le hemos indicado al principio del código, en el apartado de las variables:

```
    Rect miRectangulo(xMin, yMin, xMax-xMin, yMax-yMin);
```

Luego cortamos la imagen original usando el rectángulo creado anteriormente:

```
    Mat imagenRect (imagen_RGB, miRectangulo);
```

Y finalmente copiamos los datos a la nueva matriz (imagen):

```
    imagenRect.copyTo(imagen_CORTADA);
```

- **Función aplicar filtro Canny**

```
/*********************************************/
/*            FUNCION FILTRO CANNY             */
/*********************************************/

void FiltroCanny()
{
    // Convertir la imagen_CORTADA en escala de grises
    cvtColor(imagen_CORTADA, src_gray, CV_BGR2GRAY);

    // Reducir ruido con un kernel 3x3
    blur( src_gray, detected_edges, Size(3,3) );

    /// Detector Canny
    Canny( detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
    kernel_size );

    /// Rellenamos la imagen_CANNY con ceros (lo que significa que la imagen es
    completamente negra)
    imagen_CANNY = Scalar::all(0);

    // Copiamos la imagen_CORTADA en imagen_CANNY con la matriz detected_edges.
    imagen_CANNY es RGB
    imagen_CORTADA.copyTo( imagen_CANNY, detected_edges);

    return;
}
```

Explicación

1º - Convertimos la imagen cortada anteriormente en una imagen con escala de grises a través del comando `cvtColor()`. Esta imagen, será una imagen que no se guardará, lo que quiere decir que solo servirá para llevar a cabo este proceso.

```
    cvtColor(imagen_CORTADA, src_gray, CV_BGR2GRAY);
```

2º - Creamos una matriz de 3x3, la cual al aplicarla con la función `blur()` conseguimos difuminar la imagen con escala de grises anterior.

```
    blur( src_gray, detected_edges, Size(3,3) );
```

3º - Aplicamos la función `Canny()`, la cual se le tiene que especificar los umbrales para llevar a cabo la transformación de la imagen. Los umbrales seleccionados, han sido probados y se ha visto que son los que mejor se adaptan a nuestro interés.

```
    Canny( detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
    kernel_size );
```

Función Canny:

Canny(image, edges, threshold1, threshold2, apertureSize)

image: Imagen de un solo canal de 8 bits.
edges: Salida del mapa de los bordes. Tiene el mismo tamaño y tipo que image.
threshold1: Primer umbral para el procedimiento de histéresis.
threshold2: Segundo umbral para el procedimiento de histéresis.
apertureSize: Tamaño de apertura para el operador `Sobel()`.

4º - Creamos una imagen que tendrá todos los pixeles con un valor de 0, lo que significa que la imagen es completamente negra.

```
imagen_CANNY = Scalar::all(0);
```

5º - Copiamos los pixeles de la imagen_CORTADA que están indicados en la matriz detected_edges y los pegamos en la imagen_CANNY que habíamos creado anteriormente, con lo que esta imagen será a color (RGB).

```
imagen_CORTADA.copyTo( imagen_CANNY, detected_edges);
```

- **Función dibujar contornos**

```
/*********************************************/
/*         FUNCION DIBUJAR CONTORNOS         */
/*********************************************/

void DibujarContornos()
{
    vector<vector<Point>> contornos;
    vector<Vec4i> jerarquia;

    // Convertir la imagen_CANNY en escala de grises
    cvtColor(imagen_CANNY, imagen_find, CV_BGR2GRAY);

    // Encontrar contornos
    findContours(imagen_find, contornos, jerarquia, modo, metodo, Point(0, 0));

    // Dibujar contornos
    imagen_CONTORNOS = Mat::zeros(imagen_find.size(), CV_8UC3);
    for( int i = 0; i< contornos.size(); i++ )
    {
        Scalar color = Scalar(0, 0, 255);

        drawContours(imagen_CONTORNOS, contornos, i, color, grosorLinea,
        conectividadLinea, jerarquia, maxNivelContornos, Point());
    }

    // Convertir la imagen_CONTORNOS en escala de grises
    cvtColor(imagen_CONTORNOS, imagen_GRAY, CV_BGR2GRAY);

    return;
}
```

Explicación

1º - Creamos un vector de vectores llamado `contornos`, el cual almacenará los puntos de los contornos encontrados en la imagen binaria, y otro vector llamado `jerarquia`, el cual contendrá información sobre la topología de la imagen.

```
vector<vector<Point>> contornos;
vector<Vec4i> jerarquia;
```

2º - Convertimos la imagen_CANNY anterior en una imagen con escala de grises (binaria).

```
cvtColor(imagen_CANNY, imagen_find, CV_BGR2GRAY);
```

3º - Buscamos los contornos de la imagen binaria `imagen_find`.

```
findContours(imagen_find, contornos, jerarquia, modo, metodo, Point(0, 0));
```

Función buscar contornos:

---

findContours(image, contours, hierarchy, mode, method, Point offset=Point())

image: Imagen de un solo canal de 8 bits.
contours: Vector con los puntos de los contornos.
hierarchy: Vector de salida opcional, que contiene información sobre la topología de la imagen.
mode: CV_RETR_TREE recupera todos los contornos y reconstruye una jerarquía completa de contornos anidados.
method: CV_CHAIN_APPROX_SIMPLE comprime segmentos horizontales, verticales y diagonales y deja sólo sus puntos finales.
Por ejemplo, un contorno rectangular arriba-derecho se codifica con 4 puntos.
Point offset: Desplazamiento opcional por el cual se desplaza cada punto de contorno.
Esto es útil si los contornos se extraen del ROI (region of interest) de la imagen y luego deben analizarse en todo el contexto de la imagen.

4º - Dibujamos todos los contornos encontrados. Para ello creamos la imagen_CONTORNOS de 8 bits y luego nos desplazamos por dentro de ella para pintar los pixeles guardados, en el vector de vectores `contornos`, de color rojo. Esto último se indica en la variable `color`, indicándole que es igual a un vector llamado `Scalar`, proveniente de la librería OpenCV. Este vector es usado para pasar valores de pixeles.
Si le indicamos (0, 0, 255), quiere decir que es un color BGR, el cual tiene los valores de configuración Blue = 0, Green = 0, Red = 255.

```
imagen_CONTORNOS = Mat::zeros(imagen_find.size(), CV_8UC3);
for( int i = 0; i< contornos.size(); i++ )
{
    Scalar color = Scalar(0, 0, 255);

    drawContours(imagen_CONTORNOS, contornos, i, color, grosorLinea,
    conectividadLinea, jerarquia, maxNivelContornos, Point());
}
```

Función dibujar contornos:

---

drawContours(image, contours, contourIdx, color, thickness, lineType, hierarchy, maxLevel, Point offset=Point())

image: Imagen de destino.
contours: Todos los contornos de entrada. Cada contorno se almacena como un vector de puntos.
contourIdx: Parámetro que indica un contorno a dibujar. Si es negativo, se dibujan todos los contornos.
color: Color de los contornos.
thickness: Grosor de las líneas con las que se dibujan los contornos. Si es negativo se dibujan los interiores del contorno.
lineType: Conectividad de línea. Ver la función `line()` para más detalles.
- 8 (u omitido): 8 líneas conectadas.
- 4: 4 líneas conectadas.
- CV_AA: líneas no conectadas.

hierarchy: Información opcional sobre jerarquía. Sólo es necesario si desea dibujar sólo algunos de los contornos (consulte maxLevel).
maxLevel: Nivel máximo para contornos dibujados. Si es 0, sólo se dibuja el contorno especificado. Si es 1, la función dibuja el contorno (s) y todos los contornos anidados. Si es 2, la función dibuja los contornos, todos los contornos anidados, todos los contornos anidados a anidados, y así sucesivamente. Este parámetro sólo se tiene en cuenta cuando hay jerarquía disponible.
Point offset: Parámetro opcional de cambio de contorno. Desplazar todos los contornos dibujados por el offset=(dx,dy).

5º - Convertimos la imagen_CONTORNOS en una imagen de 8 bits (con escala de grises), llamada imagen_GRAY.

```
cvtColor(imagen_CONTORNOS, imagen_GRAY, CV_BGR2GRAY);
```

- **Función encontrar líneas grises izquierda y derecha**

```
/*********************************************/
/*       FUNCION ENCONTRAR LINEAS GRISES       */
/*             IZQUIERDA Y DERECHA             */
/*********************************************/

void EncontrarLineas()
{
    y = 1;
    xIzq = 10;
    xDer = imagen_GRAY.cols - 10;
    vectorPixelesIzq[0] = 0;
    vectorPixelesIzq[1] = 0;
    vectorPixelesDer[0] = 0;
    vectorPixelesDer[1] = 0;

    do
    {
        // Ponemos las variables por defecto
        lineaIzqEncontrada = 0;
```

```
lineaDerEncontrada = 0;

// Construccion del vector de pixeles izquierdo
do
{

    Scalar intensidadPixelIzq = imagen_GRAY.at<uchar>(y, xIzq);

    if (intensidadPixelIzq.val[0] > 50)
    {
        cout << "El punto [" << y << "," << xIzq << "] vale: " <<
        intensidadPixelIzq << endl;
        posVectorIzq = 0;
        vectorPixelesIzq[posVectorIzq] = y;
        posVectorIzq = posVectorIzq + 1;
        vectorPixelesIzq[posVectorIzq] = xIzq;
    }
    xIzq = xIzq + 1;

} while (xIzq < imagen_GRAY.cols / 2);

if (vectorPixelesIzq[1] != 0)
{
    lineaIzqEncontrada = 1;
}

// Construccion del vector de pixeles derecho
do
{

    Scalar intensidadPixelDer = imagen_GRAY.at<uchar>(y, xDer);

    if (intensidadPixelDer.val[0] > 50)
    {
        cout << "El punto [" << y << "," << xDer << "] vale: " <<
        intensidadPixelDer << endl;
        posVectorDer = 0;
        vectorPixelesDer[posVectorDer] = y;
        posVectorDer = posVectorDer + 1;
        vectorPixelesDer[posVectorDer] = xDer;
    }
    xDer = xDer - 1;

} while (xDer > imagen_GRAY.cols / 2);

if (vectorPixelesDer[1] != 0)
{
    lineaDerEncontrada = 1;
}

if (lineaIzqEncontrada == 0 || lineaDerEncontrada == 0)
{
    y = y + 1;      // Indicamos que pase a la siguiente fila
    xIzq = 10;      // Volvemos a la columna "" para volver a buscar la
                    linea izquierda
```

```
            xDer = imagen_GRAY.cols - 10;    // Volvemos a la columna "" para volver
                                              a buscar la linea derecha
            posVectorIzq = 0;      // Volvemos a la primera posicion del vector
                                   izquierdo, para que empiece a guardar los puntos
                                   desde ahi
            posVectorDer = 0;      // Volvemos a la primera posicion del vector
                                   derecho, para que empiece a guardar los puntos
                                   desde ahi
        }
        else if (lineaIzqEncontrada == 1 && lineaDerEncontrada == 1)
        {
            salirBucle = 1;
        }

        if (y >= imagen_GRAY.rows)
        {
            salirBucle = 1;

            // Indicacion de que a llegado a la ultima fila y no ha encontrado nada
            cout << "\n****** El buscador de pixeles a llegado a la ultima fila y no
            ha encontrado ninguna linea guardada en el vector izquierdo ******" <<
            endl;
        }
    } while (salirBucle == 0);

/*****************************************************************************/

    // Aunque haya terminado de examinar toda la foto, observamos los vectores para
    saber si habia encontrado alguna línea
    if (vectorPixelesIzq[1] != 0)
    {
        lineaIzqEncontrada = 1;
    }

    if (vectorPixelesDer[1] != 0)
    {
        lineaDerEncontrada = 1;
    }

/*****************************************************************************/

    // Mostrar por pantalla el vectorPixelesIzq con los valores guardados
    cout << "\nvectorPixelesIzq [ P1y  P1x ]" << endl;
    cout << "vectorPixelesIzq [";
    for (int i=0; i<2; i++)
    {
        cout << " " << vectorPixelesIzq[i] << " ";
    }
    cout << "]" << endl;

    // Mostrar por pantalla el vectorPixelesDer con los valores guardados
    cout << "\nvectorPixelesDer [ P2y  P2x ]" << endl;
    cout << "vectorPixelesDer [";
```

```
    for (int i = 0; i<2; i++)
    {
        cout << " " << vectorPixelesDer[i] << " ";
    }
    cout << "]" << endl;

    return;
}
```

Explicación

1º - La función empieza poniendo los valores iniciales a las variables que se van a usar.

Para indicar que empiece desde la primera fila de la imagen a buscar pixeles de color gris, se emplea la variable `y`.

```
    y = 1;
```

Al igual ocurre con `xIzq` y `xDer`, que son las variables para indicar que la función empiece buscando pixeles de color gris, ya sea desde la parte izquierda para la línea izquierda, como desde la parte derecha para la línea derecha. Se le deja un margen de 10 pixeles sin examinar porque de esta forma evitamos los posibles errores que pueda causar el borde de la imagen.

```
    xIzq = 10;
    xDer = imagen_GRAY.cols - 10;
```

Y por último ponemos los valores de los dos vectores a 0, para indicar que no tienen nada dentro de ellos.

```
    vectorPixelesIzq[0] = 0;
    vectorPixelesIzq[1] = 0;
    vectorPixelesDer[0] = 0;
    vectorPixelesDer[1] = 0;
```

2º - Acto seguido entraremos a un "*do while"*, lo cual significa que tendrá una condición para salir de este bucle.

Cada vez que empecemos este bucle las variables que indican si se ha encontrado la línea izquierda o derecha se ponen a 0. Esto se hace porque lo que se busca es que los dos vectores, que son los que nos indicarán si han encontrado las líneas, tienen que estar en la misma fila de vectores, lo cual lo indica la variable *y*.

```
    // Ponemos las variables por defecto
    lineaIzqEncontrada = 0;
    lineaDerEncontrada = 0;
```

3º - Lo siguiente que se hace es construir el vector de pixeles izquierdo y seguidamente el derecho. Estos vectores guardarán las posiciones de los últimos pixeles encontrados en una misma fila.

Las construcciones de los dos vectores son exactamente igual, salvo la condición de salida del bucle la cual se explica a continuación.

La línea izquierda se detecta analizando la mitad izquierda de la imagen, yendo de izquierda a derecha y de arriba abajo, y la línea derecha se detecta analizando la mitad derecha de la imagen, yendo de derecha a izquierda y de arriba abajo.

```
// Construccion del vector de pixeles izquierdo
do
{

    Scalar intensidadPixelIzq = imagen_GRAY.at<uchar>(y, xIzq);

    if (intensidadPixelIzq.val[0] > 50)
    {
        cout << "El punto [" << y << "," << xIzq << "] vale: " <<
        intensidadPixelIzq << endl;
        posVectorIzq = 0;
        vectorPixelesIzq[posVectorIzq] = y;
        posVectorIzq = posVectorIzq + 1;
        vectorPixelesIzq[posVectorIzq] = xIzq;
    }
    xIzq = xIzq + 1;

} while (xIzq < imagen_GRAY.cols / 2);

if (vectorPixelesIzq[1] != 0)
{
    lineaIzqEncontrada = 1;
}
```

Scalar intensidadPixelIzq: Lee el valor de intensidad que tiene el pixel de la posición indicada como (y, xIzq).

Si el pixel analizado tiene un valor por encima de 50, significa que es el color gris, por lo que se muestra por pantalla su valor y se guarda la posición del pixel en el vector correspondiente.

xIzq = xIzq + 1: Indica que pase a la siguiente columna de pixeles, para ser analizada en el caso de que siga cumpliéndose la condición del bucle.

Para poder salir del bucle, se tiene que haber llegado a la mitad de la imagen, indicado como `xIzq < imagen_GRAY.cols / 2`, donde `imagen_GRAY.cols` indica las columnas que tiene la imagen anteriormente tratada.

Como podemos ver a continuación, el vector derecho se construye exactamente igual que el izquierdo, salvo que la condición de salida del bucle es diferente. Para poder salir de este, se tiene que haber llegado a la mitad de la imagen, desde la parte derecha, indicado como `xDer > imagen_GRAY.cols / 2`.

```
// Construccion del vector de pixeles derecho
do
{

    Scalar intensidadPixelDer = imagen_GRAY.at<uchar>(y, xDer);

    if (intensidadPixelDer.val[0] > 50)
    {
        cout << "El punto [" << y << "," << xDer << "] vale: " <<
        intensidadPixelDer << endl;
        posVectorDer = 0;
        vectorPixelesDer[posVectorDer] = y;
```

```
            posVectorDer = posVectorDer + 1;
            vectorPixelesDer[posVectorDer] = xDer;
        }
        xDer = xDer - 1;

    } while (xDer > imagen_GRAY.cols / 2);

    if (vectorPixelesDer[1] != 0)
    {
        lineaDerEncontrada = 1;
    }
```

xDer = xDer - 1: Indica que pase a la siguiente columna de pixeles, para ser analizada en el caso de que siga cumpliéndose la condición del bucle.

Para terminar este punto, vemos que al finalizar los bucles de construcción de los vectores, existen dos *"if"* los cuales indican si se han encontrado las líneas, poniendo sus variables a 1. Para ello se consulta la posición 1 de los vectores, y si son distinto de 0, significa que se ha guardado algo anteriormente.

4º - Una vez construidos los vectores, si se han encontrado líneas, se procede a verificarlo, y en el caso de que no se hayan encontrado las dos se vuelve a repetir el proceso pero con la fila siguiente de pixeles, tal y como se describe a continuación.

```
if (lineaIzqEncontrada == 0 || lineaDerEncontrada == 0)
        {
            y = y + 1;      // Indicamos que pase a la siguiente fila
            xIzq = 10;      // Volvemos a la columna "" para volver a buscar la
                            linea izquierda
            xDer = imagen_GRAY.cols - 10;    // Volvemos a la columna "" para volver
                                              a buscar la linea derecha
            posVectorIzq = 0;      // Volvemos a la primera posicion del vector
                                   izquierdo, para que empiece a guardar los puntos
                                   desde ahi
            posVectorDer = 0;      // Volvemos a la primera posicion del vector
                                   derecho, para que empiece a guardar los puntos
                                   desde ahi
        }
        else if (lineaIzqEncontrada == 1 && lineaDerEncontrada == 1)
        {
            salirBucle = 1;
        }

        if (y >= imagen_GRAY.rows)
        {
            salirBucle = 1;

            // Indicacion de que a llegado a la ultima fila y no ha encontrado nada
            cout << "\n****** El buscador de pixeles a llegado a la ultima fila y no
            ha encontrado ninguna linea guardada en el vector izquierdo ******" <<
            endl;
        }
    } while (salirBucle == 0);
```

Saldremos del bucle si sucede una de las siguientes afirmaciones:

- Si da el caso que ha encontrado las dos líneas, las cuales quedan confirmadas porque las variables `lineaIzqEncontrada` y `lineaDerEncontrada` tienen un valor de 1.
- Si hemos llegado al final de la imagen y no ha encontrado los pixeles de las líneas en la misma fila de pixeles.

5º - Cuando salgamos del bucle donde hemos buscado los pixeles que caracterizan haber encontrado las líneas, llegamos al punto donde se procede a verificar si hay algún pixel guardado en los vectores, lo que quiere decir que efectivamente había encontrado una línea.

Esta verificación se hace porque puede haber el caso que haya encontrado una línea y la otra no, y aun así, se haya salido del bucle. Repito, que este caso sucede cuando ha encontrado los pixeles que caracterizan haber encontrado una o las dos líneas en filas de pixeles diferentes, lo que quiere decir que el valor de y de los vectores, no estarían a la misma altura.

```
// Aunque haya terminado de examinar toda la foto, observamos los vectores para
saber si habia encontrado alguna línea
if (vectorPixelesIzq[1] != 0)
{
    lineaIzqEncontrada = 1;
}

if (vectorPixelesDer[1] != 0)
{
    lineaDerEncontrada = 1;
}
```

6º - Por último, se muestra por pantalla los dos vectores con sus valores guardados, los cuales indicarán la posición donde se han encontrado los pixeles que caracterizan haber localizado las líneas.

```
// Mostrar por pantalla el vectorPixelesIzq con los valores guardados
cout << "\nvectorPixelesIzq [ P1y  P1x ]" << endl;
cout << "vectorPixelesIzq [";

for (int i=0; i<2; i++)
{
    cout << " " << vectorPixelesIzq[i] << " ";
}
cout << "]" << endl;

// Mostrar por pantalla el vectorPixelesDer con los valores guardados
cout << "\nvectorPixelesDer [ P2y  P2x ]" << endl;
cout << "vectorPixelesDer [";
for (int i = 0; i<2; i++)
{
    cout << " " << vectorPixelesDer[i] << " ";
}
cout << "]" << endl;
```

- **Función encontrar centro que forman las líneas izquierda y derecha**

```
/*******************************************/
/*     FUNCION ENCONTRAR CENTRO DE LINEAS      */
/*******************************************/

void EncontrarCentroLineas()
{
    // Calcular el punto medio de los pixeles encontrados de los vectores
    if  (vectorPixelesIzq[0] == vectorPixelesDer[0] && lineaIzqEncontrada == 1 &&
    lineaDerEncontrada == 1)
    {
        centroPixeles = (vectorPixelesIzq[1] + vectorPixelesDer[1]) / 2;
    }
    else if (lineaIzqEncontrada == 1 && lineaDerEncontrada == 0)
    {
        centroPixeles = imagen_GRAY.cols;       // Esto hara que tiempoPulsPwm = 5
    }
    else if (lineaIzqEncontrada == 0 && lineaDerEncontrada == 1)
    {
        centroPixeles = imagen_GRAY.cols - imagen_GRAY.cols;  // Esto hara que
                                                              tiempoPulsPwm = 25
    }

    return;
}
```

Explicación

En esta función lo que se hace es calcular el centro que forman los pixeles encontrados que caracterizan la posición de las líneas.

Si encuentra las dos líneas y estas, están situadas en la misma fila de pixeles, se calcula el centro de la siguiente forma:

```
centroPixeles = (vectorPixelesIzq[1] + vectorPixelesDer[1]) / 2;
```

Si encuentra solo la línea izquierda, se calcula el centro de la siguiente forma, de tal modo que posteriormente se le indicará un valor de 5 a la variable `tiempoPulsPwm`, tal y como veremos más adelante.

```
centroPixeles = imagen_GRAY.cols;           // Esto hara que tiempoPulsPwm = 5
```

Si encuentra solo la línea derecha, se calcula el centro de la siguiente forma, de tal modo que posteriormente se le indicará un valor de 25 a la variable `tiempoPulsPwm`, tal y como veremos más adelante.

```
centroPixeles = imagen_GRAY.cols - imagen_GRAY.cols;      // Esto hara que
                                                          tiempoPulsPwm = 25
```

- **Función que calcula la variable `tiempoPulsPwm`**

```
/*********************************************/
/*       FUNCION CALCULAR tiempoPulsPwm      */
/*********************************************/

void CalcularTiempoPulsPwm()
{
    tiempoPulsPwm = 15 - ((centroPixeles - centroImagen) / (16));

    if (tiempoPulsPwm <= 5)
    {
        tiempoPulsPwm = 5;
    }
    else if (tiempoPulsPwm >= 25)
    {
        tiempoPulsPwm = 25;
    }

    return;
}
```

Explicación

Para calcular el tiempo que se le envía al servomotor, se ha hecho uso del control PID,

Control PID:

Acción proporcional

Un controlador con acción proporcional, se caracteriza porque la señal de salida es proporcional al error o diferencia entre el valor actual de la variable controlada y el punto de consigna.

La variación en la señal de salida S viene dada por siguiente relación, que es la ecuación de una recta:

$$S = K \cdot E + K'$$

Siendo K una constante llamada ganancia. Esta constante representa la relación entre el cambio en el valor de salida y el cambio en la señal de entrada que lo causó, después de haber logrado un nuevo estado de equilibrio.
Lo más habitual es que K' tenga un valor del 50 %.

S → Señal de salida
E → (M – SP) → Error ó desviación
M → Medida
SP → Punto de consigna o Set Point
K' → Valor de S cuando E = 0, M = SP
K → Ganancia

Dentro de esta acción proporcional, entra en juego otro dato importante, llamado *Banda Proporcional*, la cuál es la variación de la variable controlada para provocar un recorrido completo del elemento final de control. Se expresa en %.

La relación entre la banda proporcional BP y la ganancia K es la siguiente:

$$\mathbf{BP = 100\ \% / K}$$

BP = 100 %, K = 1 → La medida varía un 100 % y la salida varía un 100 %.
BP = 50 %, K = 2 → La medida varía un 50 % y la salida varía un 100 %.
BP = 200 %, K = 1/2 → La medida varía un 200 % y la salida varía un 100 %.

BP pequeña supone un controlador muy sensible (ganancia K grande), en el cual un pequeño error es suficiente para producir un gran cambio en la señal de salida, y por tanto en la posición del elemento final de control.

Aparte de todo esto, también existe la posibilidad de tener un controlador de acción directa o inversa, donde la acción directa responde con una mayor salida ante aumentos de la señal de medida y en el caso de acción inversa daría menor señal de salida ante aumentos de la señal de medida.

$$\mathbf{Acción\ Directa \rightarrow S = K \cdot E + K'}$$

$$\mathbf{Acción\ Inversa \rightarrow S = - K \cdot E + K'}$$

Acción integral

Un controlador con acción integral, se caracteriza porque proporciona la ayuda suficiente a la acción proporcional para llegar al Set Point sin provocar demasiada oscilación de la señal. A esto se le llama reducción automática del Off-Set.

Disminuyendo la banda proporcional disminuye el Off-Set, pero para una banda demasiado estrecha empieza a oscilar.

En realidad, el problema del control proporcional reside en el hecho de que para un error fijo (offset) da una salida fija, es decir que, teniendo una situación de error, la posición del servomotor no varía, con lo cual no puede corregir dicho error. Lo que necesitamos es un controlador que varíe su salida constantemente hasta que el error desaparezca. Esto es el concepto del control integral.

La variación en la señal de salida S viene dada por siguiente relación, que es la suma de la acción proporcional más la acción integral (controlador P+I):

$$\mathbf{Acción\ Directa \rightarrow S = K \cdot E + K \cdot E \cdot (t / T_i) + K'}$$

$$\mathbf{Acción\ Indirecta \rightarrow S = - K \cdot E - K \cdot E \cdot (t / T_i) + K'}$$

t → Tiempo de la perturbación
T_i → Tiempo integral (Tiempo que tarda la acción integral en alcanzar a la proporcional y de este modo llegar al Set-Point)

En un primer momento, la acción integral es un factor despreciable.
Primero actúa la acción proporcional y luego la integral.

- Mucha acción integral supone un Ti corto y poca acción integral supone un T_i largo.
- Mucha acción integral añade oscilaciones al sistema.
- T_i se expresa en minutos o segundos, o en repeticiones por minuto (inversa de min).

Acción derivativa

Un controlador con acción derivativa, se caracteriza porque no corrige errores, solamente reacciona, pero muy rápidamente.

Con la acción derivativa lo que intentamos es adelantarnos a la variación de aquella variable que queremos controlar, con lo que por muy mínimo que sea el error, la acción derivativa entrará en juego y reaccionará muy rápidamente para volver al Set-Point.

- Se busca la rapidez del controlador en procesos inerciales.
- La acción derivada es la variación del error, es decir, la velocidad de cambio del error.
- La acción diferencial es como si tuviéramos una K variable (ganancia variable).

La variación en la señal de salida S viene dada por siguiente relación, que es la suma de la acción proporcional más la acción derivativa (controlador P+D):

Acción Directa → $S = K \cdot E + K \cdot E \cdot (T_d / t) + K'$

Acción Indirecta → $S = - K \cdot E - K \cdot E \cdot (T_d / t) + K'$

t → Tiempo de la perturbación
T_d → Tiempo derivativo

Muchas veces hay que limitar la acción derivativa, sino nos provocaría oscilaciones innecesarias.

En este caso se realizó un control Proporcional básico, ya que no se tuvo más tiempo para la realización de un control PID más completo.

Lo ideal hubiese sido la realización completa de este control, ya que de este modo el vehículo hubiera funcionado mejor, pero haciendo pruebas nos percatamos que el vehículo funcionaba bien con solo el control Proporcional. Así mismo, vimos que de momento no hacía falta un completo control PID, ya que lo que se pretendía era demostrarlo con una prueba.

A los servomotores solo se les puede enviar 21 valores, los cuales están comprendidos entre 5 y 25, los cuales son los límites de giro del propio servomotor.

- Si se envía al servomotor una señal PWM, la cual tiene la variable `tiempoPulsPwm` con un valor de 5, el servomotor hará girar las ruedas delanteras lo más a la derecha posible.

- Si se envía al servomotor una señal PWM, la cual tiene la variable `tiempoPulsPwm` con un valor de 15, el servomotor pondrá las ruedas delanteras lo más rectas posible.

- Si se envía al servomotor una señal PWM, la cual tiene la variable `tiempoPulsPwm` con un valor de 25, el servomotor hará girar las ruedas delanteras lo más a la izquierda posible.

Sabiendo esto, la siguiente función lo que hará es cambiar el valor de la variable `tiempoPulsPwm`, de modo que, cuanto más se desvíe el vehículo del centro, más girarán las ruedas delanteras, con la intención de volver a llevar el vehículo al centro del carril.

```
tiempoPulsPwm = 15 - ((centroPixeles - centroImagen) / (16));
```

Podemos ver que se asemeja a una acción proporcional inversa, tal y como la hemos descrita anteriormente con la ecuación:

$$S = -K \cdot E + K'$$

Donde:

BP = 1600 %
K = 100 % / BP =100 / 1600 = 1 / 16 = 0,0625
E = centroPixeles – centroImagen
K' = 15

- **Funciones mover motores**

```
/*********************************************/
/*           FUNCIONES MOVER MOTORES           */
/*********************************************/

void MoverMotorDelante()
{
    digitalWrite(MotorDelante, LOW);
    softPwmCreate(MotorDelante, 0, tiempoCicloPwm);
    softPwmWrite(MotorDelante, tiempoPulsPwm);

    return;
}

void MoverMotorDetras()
{
    digitalWrite(MotorDetras, HIGH);
    digitalWrite(MotorEnableDetras, HIGH);
    int tiempoMotorCC;

    if (tiempoPulsPwm < 15)
    {
        tiempoMotorCC = 300 - ((15 - tiempoPulsPwm) * 5);
    }
    else if (tiempoPulsPwm > 15)
    {
        tiempoMotorCC = 300 - ((tiempoPulsPwm - 15) * 5);
    }
    else
    {
        tiempoMotorCC = 300;
    }
```

```
    delay(tiempoMotorCC);
    digitalWrite(MotorEnableDetras, LOW);
    digitalWrite(MotorDetras, LOW);

    return;
}
```

Explicación

1º - Para mover el servomotor de las ruedas delanteras se han utilizado unas funciones que crean una señal PWM.

```
    softPwmCreate(MotorDelante, 0, tiempoCicloPwm);
    softPwmWrite(MotorDelante, tiempoPulsPwm);
```

Cuando se pone **`softPwmCreate(MotorDelante, 0, tiempoCicloPwm)`** quiere decir que se crea una señal PWM con un tiempo de ciclo de 200 (variable `tiempoCicloPwm`), lo que equivale a 20 ms.

Cuando se pone **`softPwmWrite(MotorDelante, tiempoPulsPwm)`** quiere decir que la señal PWM se mantiene en HIGH (tensión alta) durante el tiempo que ponga en la variable `tiempoPulsPwm`). Por ejemplo, si la variable `tiempoPulsPwm` = 15, quiere decir que la señal se mantendrá en HIGH durante 1,5 ms, lo que equivale a centrar el servomotor.

2º - Para mover el motor de las ruedas traseras se ha utilizado la placa de control L293D, la cual proporcionará alimentación al motor a través de ella siempre y cuando pongamos en tensión las entradas 1A y 1-2EN de su circuito integrado L293D.

- **1A**: proporciona el giro del motor
- **1-2EN:** activa la entrada 1A, dejando pasar la alimentación al motor

El tiempo de giro del motor irá en función de la posición de las ruedas delanteras, de modo que, si las ruedas están rectas girará más tiempo y así el vehículo avanzará más, y conforme las ruedas vayan girando más, el tiempo de giro del motor irá disminuyendo proporcionalmente gracias a la utilización de las siguientes formulas.

```
    if (tiempoPulsPwm < 15)
    {
        tiempoMotorCC = 300 - ((15 - tiempoPulsPwm) * 5);
    }
    else if (tiempoPulsPwm > 15)
    {
        tiempoMotorCC = 300 - ((tiempoPulsPwm - 15) * 5);
    }
    else
    {
        tiempoMotorCC = 300;
    }
```

4.6.3. Errores aparecidos en la compilación y soluciones

Para poder compilar el código creado anteriormente, es necesario escribir la siguiente línea:

>> pi@raspberrypi/TFG/Programa $ sudo g++ tfg.cpp -o tfg -I/usr/local/include/ -lraspicam -lraspicam_cv -lmmal -lmmal_core -lmmal_util -lopencv_core -lopencv_highgui -lopencv_imgcodecs -lopencv_imgproc -lwiringPi -lpthread

Mientras se estaba desarrollando el código del programa, aparecieron los siguientes errores durante la compilación del mismo.

- **Error 1:**

  `/usr/bin/ld: cannot find -lmmal`

  - ✓ **Solución Error 1:**

    Se tuvo que copiar el archivo `libmmal.so`, situado en la dirección `/opt/vc/lib/`, y pegarlo en la dirección `/usr/local/lib/`.

    >> /opt/vc/lib/ $ sudo cp libmmal.so /usr/local/lib/libmmal.so

- **Error 2:**

  `/usr/bin/ld: cannot find -lmmal_core`

  - ✓ **Solución Error 2:**

    Se tuvo que copiar el archivo `libmmal_core.so`, situado en la dirección `/opt/vc/lib/`, y pegarlo en la dirección `/usr/local/lib/`.

    >> /opt/vc/lib/ $ sudo cp libmmal_core.so /usr/local/lib/libmmal_core.so

- **Error 3:**

  `/usr/bin/ld: cannot find -lmmal_util`

  - ✓ **Solución Error 3:**

    Se tuvo que copiar el archivo `libmmal_util.so`, situado en la dirección `/opt/vc/lib/`, y pegarlo en la dirección `/usr/local/lib/`.

    >> /opt/vc/lib/ $ sudo cp libmmal_util.so /usr/local/lib/libmmal_util.so

Para poder encontrar donde estaban situadas las librerías, se ha tenido que escribir la siguiente línea y ver donde se alojaban:

>> pi@raspberrypi/TFG/Programa $ ldconfig -p | grep libmmal

- **Error 4:**

```
fatal error: raspicam/raspicam_cv.h: No such file or directory
```

  - ✓ **Solución Error 4:**

    Se tuvo que copiar la librería `raspicam_cv.h`, situada en la dirección `/home/pi/TFG/raspicam-0.1.6/src/`, y posteriormente pegarla dentro de `/usr/local/include/raspicam/`, para que al compilar, poniendo `-I/usr/local/include/`, encontrara la librería:

    >> home/pi/TFG/raspicam-0.1.6/src/ $ sudo cp raspicam_cv.h /usr/local/include/raspicam/raspicam_cv.h

- **Error 5:**

```
/usr/bin/ld: cannot find –lraspicam_cv
```

  - ✓ **Solución Error 5:**

    Se tuvo que actualizar `ldconfig` de la siguiente forma:

    >> pi@raspberrypi/TFG/raspicam-0.1.6/build/ $ sudo ldconfig

- **Error 6:**

```
/usr/bin/ld: /tmp/ccfv9yo2.o: undefined reference to symbol
'_ZN2cv7imwriteERKNS_6StringERKNS_11_InputArrayERKSt6vectorIiSaIiEE'

//usr/local/lib/libopencv_imgcodecs.so.3.2: error adding symbols: DSO
missing from command line
```

  - ✓ **Solución Error 6:**

    Estos dos últimos errores se solucionan añadiendo la librería `opencv_imgcodecs` cuando se compila con g++, de tal forma que se tendría que escribir `-lopencv_imgcodecs`. Al principio de este punto se puede observar cómo quedaría la línea que compilaría el programa.

4.6.4. Puesta en marcha del programa

Una vez creado el programa utilizando:

>> pi@raspberrypi/TFG/Programa $ sudo nano tfg.cpp

Y compilado utilizando:

>> pi@raspberrypi/TFG/Programa $ sudo g++ tfg.cpp -o tfg -I/usr/local/include/ -lraspicam -lraspicam_cv -lmmal -lmmal_core -lmmal_util -lopencv_core -lopencv_highgui -lopencv_imgcodecs -lopencv_imgproc -lwiringPi -lpthread

Toca el turno de darle permisos de ejecución al archivo ejecutable que nos ha creado el compilador g++. Para ello escribiremos la siguiente línea:

>> pi@raspberrypi/TFG/Programa $ sudo chmod +x tfg

Finalmente, para ejecutar el programa tendremos que escribir la siguiente línea:

```
>> pi@raspberrypi/TFG/Programa $ ./tfg
```

Nota: Para detener el programa utilizamos la combinación de teclas *Ctrl* + *C*.

4.7. DESCRIPCIÓN DE LA CAJA CONTENEDORA

La caja situada encima del vehículo contendrá en su interior una Raspberry Pi 2, la cuál será el cerebro del vehículo, y la placa controladora de los motores. En la parte de debajo de la caja se situará la batería de Ni-Cd de 700 mAh para proporcionar alimentación a la placa de control de los motores, y a los propios motores, y en la parte superior de la caja se albergará la "Power Bank" de 5200 mAh para alimentar la Raspberry Pi 2.

Además de todo esto, en la parte superior habrá un soporte de aluminio, el cuál sujetará la cámara utilizada para la captura de imágenes. Este soporte se ha cortado y doblado con el ángulo que mejor se adaptaba a las necesidades de la cámara, para poder capturar imágenes del carril por el cual se circulará.

La única pieza móvil de la caja es la tapa situada en su parte superior, tal que permita el acceso al interior de la caja.

Las partes izquierda y derecha de la caja se han diseñado con un agujero, el cual permitirá realizar las conexiones que se consideren oportunas, y de este modo también se favorece la refrigeración de la Raspberry Pi 2.

La parte trasera de la caja se dejó totalmente descubierta para poder realizar todas las conexiones de los motores, alimentación e interruptor de encendido.

La caja está formada por una única pieza, cuyas paredes tienen un grosor tal que, cuando cerremos la parte superior, pieza móvil, esta se apoyará sobre las paredes, haciendo que no sobresalga nada por los lados.

Cabe destacar que la caja fue diseñada única y exclusivamente para un uso no profesional, de tal modo que el vehículo fuera solamente un prototipo. Es por ello que, no se pensó en darle una forma más bonita.

La Raspberry Pi 2 se decidió colocar dentro de la caja con el micro USB situado lo más cercano a la parte trasera del vehículo, de forma que molestara lo menos posible cuando se tenga que abrir la caja por la parte de arriba y a la cámara. La distancia que se dejó desde la parte superior de la Raspberry Pi 2 situada en el interior de la caja y la parte superior (tapa) fue la que se consideró suficiente para poder albergar las conexiones sin que se fuercen o se doblen excesivamente los cables y pines.

Cuando se diseñó la caja, aparte de tener en cuenta las medidas de cada componente (Raspberry Pi 2, cámara y placa de control), también se tuvo en cuenta que el tamaño fuera el adecuado para que la caja no se viera muy grande cuando se situase encima del vehículo.

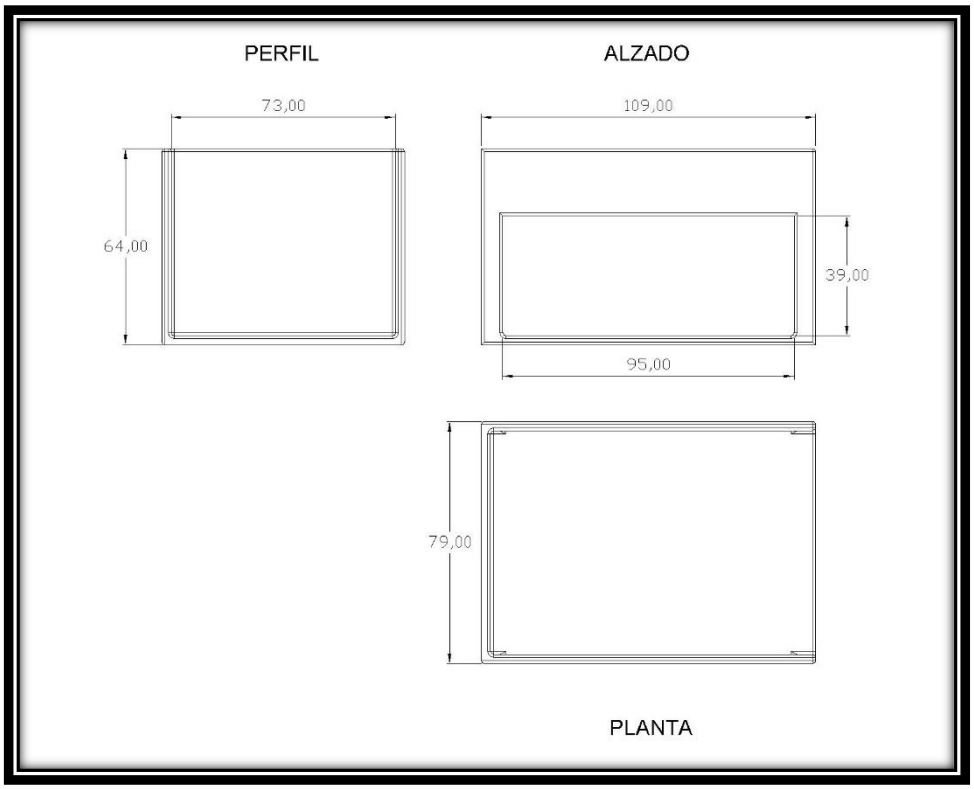


Imagen 21. Alzado, planta y perfil de la caja contenedora de los componentes.

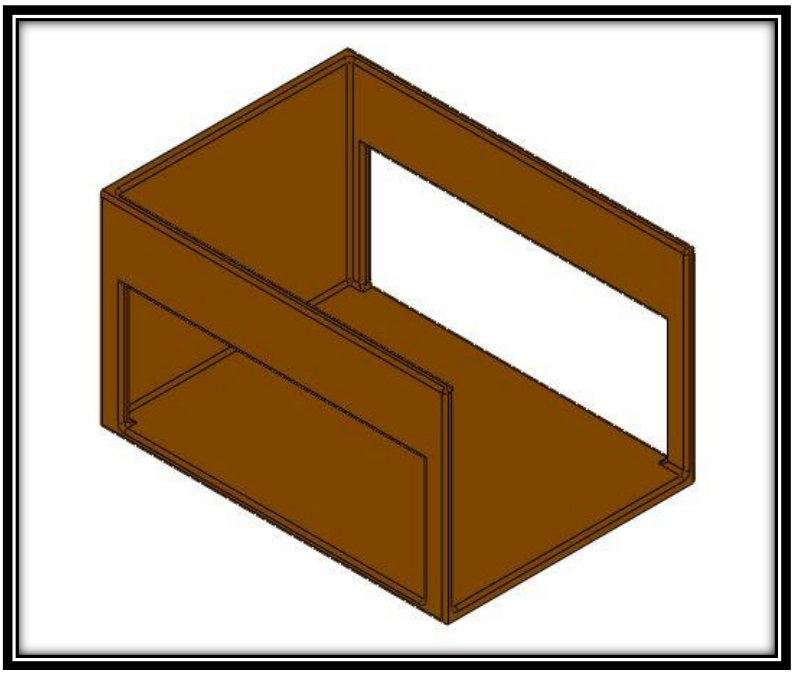


Imagen 22. Vista 3D de la caja contenedora de los componentes.

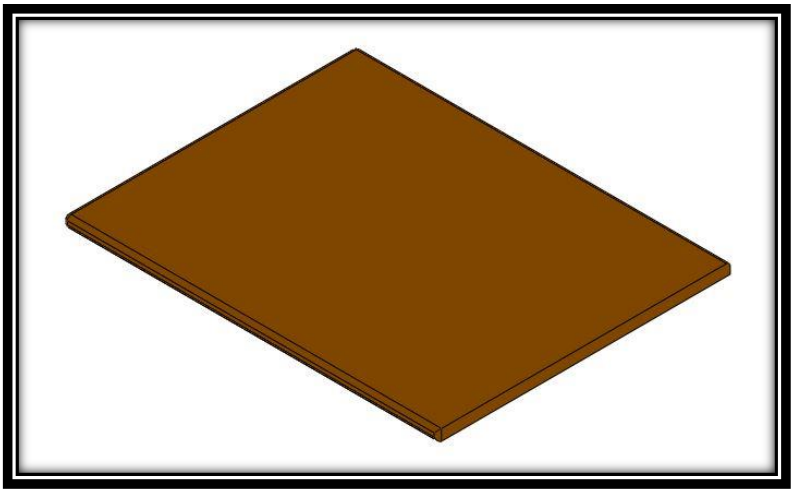


Imagen 23. Vista 3D de la tapa de la caja.

4.8. DISEÑO E IMPRESIÓN 3D DE LA CAJA CONTENEDORA

El diseño de la caja que contiene a la Raspberry Pi 2, la cámara, el shield que controla los motores y los cables de conexión necesarios, se ha llevado a cabo mediante el uso del programa SolidWorks 2017, en su versión Educación, descargada a través de la página web de la Universidad Politécnica de Valéncia.

Una vez diseñada la caja, tras haberse puesto en contacto con el tutor de este proyecto, Juan Ramón Rufino Valor y con Jaime Masiá Vañó, ambos profesores encargados del grupo de robótica y mecatrónica GROMEP, de la UPV Campus d'Alcoi, se decidió imprimirla en el departamento del grupo anteriormente mencionado.

5. MONTAJE

Una vez impresa la caja mencionada anteriormente, se procedió al montaje de la misma.

Se partió de un vehículo de radiocontrol de venta en tienda, y lo que se hizo fue desmontarlo entero y ponerle un motor de corriente continua detrás, para las ruedas traseras, y un servomotor para las ruedas delanteras, con sus cables previamente soldados.

Una vez realizado esto, se cogió la carcasa del vehículo que traía de serie y se le corto el techo para hacerlo totalmente recto y horizontal, de modo que la caja contenedora se apoyará sobre esta parte.

La tapa, como ya se dijo anteriormente, es la única pieza móvil de la caja, por lo que irá unida al resto de la caja por medio de unas bisagras.

En la parte de arriba de la tapa irá un soporte de aluminio en forma de ángulo atornillado a esta, donde irá sujeta la cámara.

Imagen 24. Montaje final del vehículo_1.

Imagen 25. Montaje final del vehículo_2.

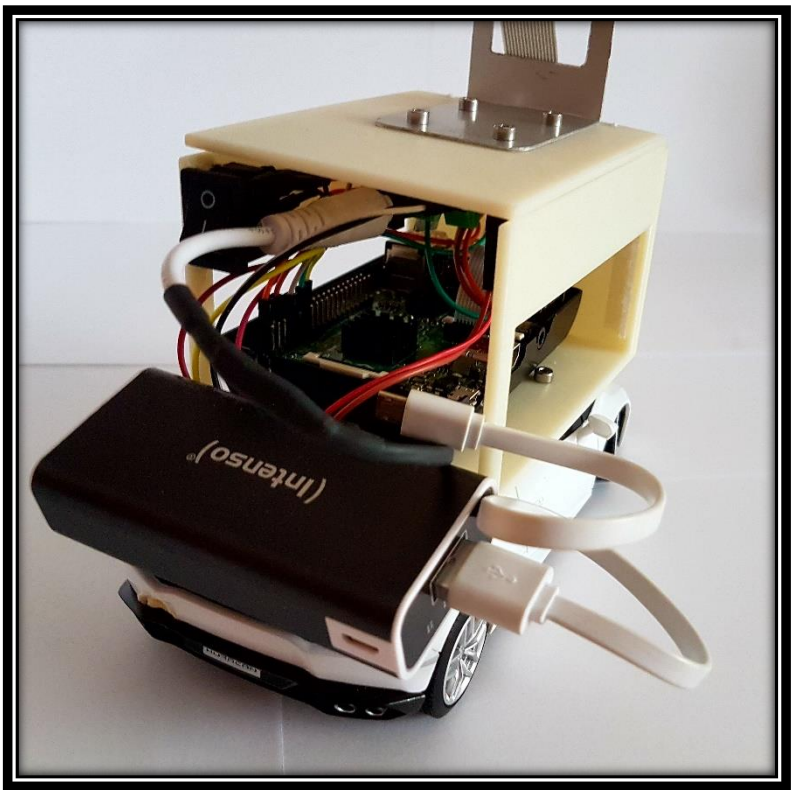


Imagen 26. Montaje final del vehículo_3.

6. ESQUEMA ELÉCTRICO

Para poder entender mejor las conexiones eléctricas entre todos los componentes del vehículo, se ha dibujado un esquema eléctrico donde se especifica dónde van conectados todos los componentes.

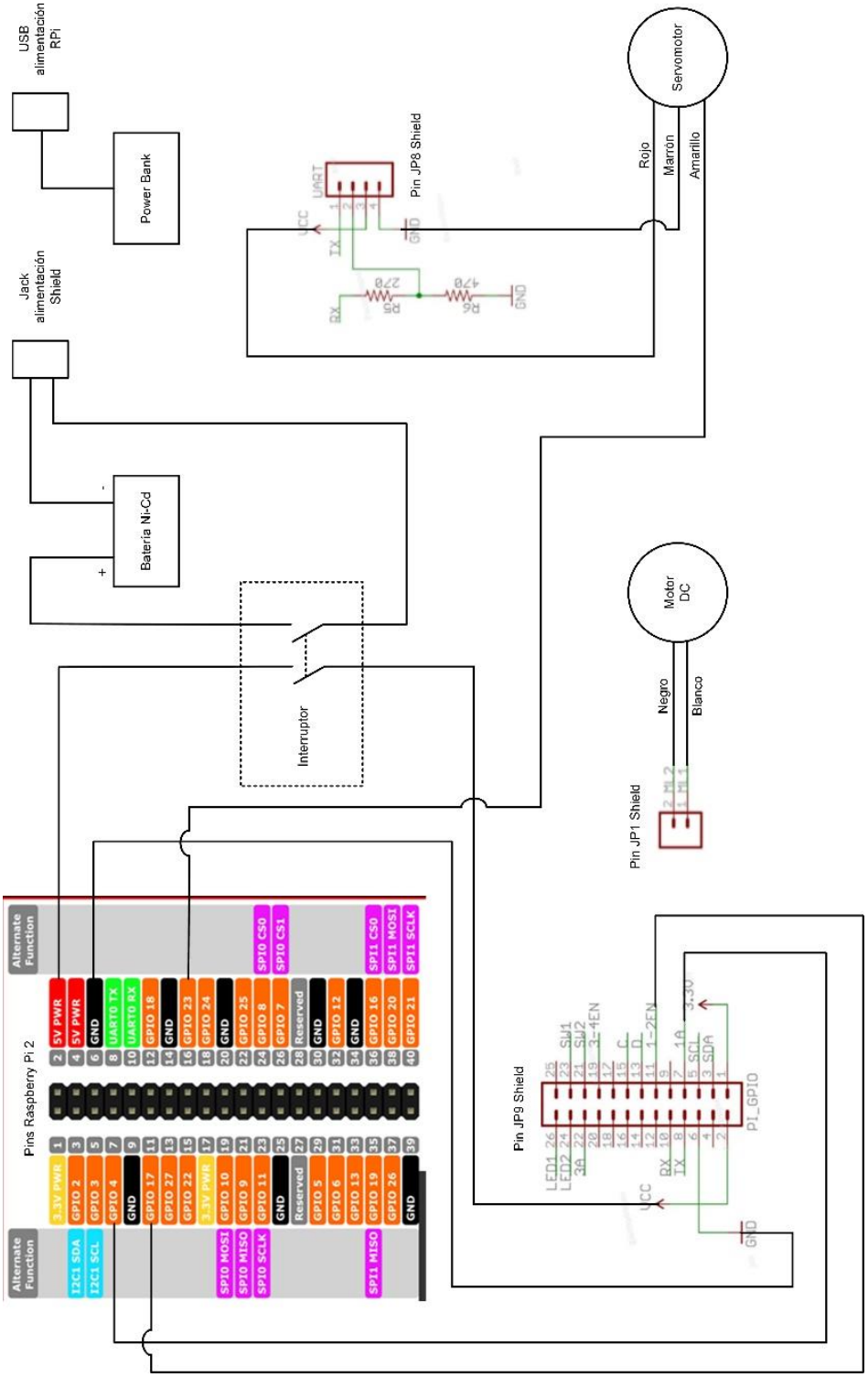


Imagen 27. Esquema eléctrico del montaje.

7. PRUEBAS DE ESTABILIDAD

Para comprobar el buen funcionamiento y su estabilidad, lo que se hizo es ejecutar un cronometro dentro del programa, tal y como se ha dicho anteriormente, para medir el tiempo que se tarda desde que se da la orden de capturar una imagen hasta el final del programa, una vez terminado el movimiento de los motores.

Para ello se dibujó en unas cartulinas blancas un trozo de circuito, por el cual tendría que circular el vehículo.

Mientras se realizaba el circuito con el vehículo, se ejecutó el programa 110 veces, lo cual equivale a decir que se realizaron 110 capturas de imágenes.

Como resultado se obtuvo que:

- El tiempo medio fue de: ***590 milisegundos***
- El tiempo máximo fue de: ***853 milisegundos***

8. POSIBLES MEJORAS

Como bien se puede observar, este vehículo no deja de ser un mero prototipo, que en un futuro y siguiendo las modificaciones y mejoras que se describirán a continuación, puede llegar a ser un prototipo competitivo en el mercado de la automatización en los vehículos, y demostrando que la visión artificial puede ser nuestro futuro aliado.

Una de las principales mejoras podría ser la utilización de la nueva Raspberry Pi 3 o el Odroid-XU4, el cual tiene cuatro núcleos y soportaría mejor la intensiva carga de la CPU creada por la utilización de la visión artificial, la cual consume muchos recursos.

La utilización de un Arduino u otra placa capaz de manejar varios motores por control PWM, junto con lo descrito anteriormente, podría ser un conjunto de mejoras mucho más potente que el actual, llegando a ser capaz de crear señales PWM mucho más exhaustivas y precisas que la señal PWM actual, realizada desde la Raspberry Pi 2 y a través de software.

Ligado a esto, otra mejora sería la utilización de una señal PWM para controlar el motor de corriente continua de las ruedas traseras, o la utilización de un motor más potente como es el ejemplo del motor Brushless, lo que sería necesario cambiar de vehículo para optar a uno más grande, ya que el problema principal es el tamaño de este.

Por otra parte, la utilización de una cámara más potente por medio del USB seria otra mejora para este prototipo.

Finalmente, una de las mejoras más importantes que se podrían hacer es, partir el código en varios ficheros, ya que actualmente existe solo un fichero el cual contiene todo el código. Con ello se mejoraría la compilación del código.

9. PROBLEMAS ENCONTRADOS

El principio, el vehículo se movía con dos motores de corriente continua, uno en las ruedas traseras y otro en las ruedas delanteras.

Para poder manejar estos motores se necesitaba el circuito integrado L293D. el cuál permite controlar el giro bidireccional de dos motores fácilmente.

Se busco por internet si había alguna placa de circuito impreso, y efectivamente había una, pero era muy grande para el propósito que se deseaba, ya que podía manejar cuatro motores. Finalmente se encontró un shield de circuito impreso en una página web llamada *Electrodragon* que vendía desde China. Este shield tenía un circuito integrado L293D y podía manejar dos motores, lo cual venía perfecto para el proyecto, aunque el único problema era que venía sin los componentes soldados, por lo que se tuvieron que soldar todos a la propia placa de circuito impreso.

Este fue el primer problema que se tuvo, aunque finalmente la placa se sigue utilizando como alimentador de 5 VDC y para controlar solamente un motor, ya que el que controla las ruedas delanteras es un servomotor.

Otro inconveniente que se podría convertir en un problema es la utilización de un servomotor para provocar el giro de las ruedas delanteras, ya que, por culpa de este, las ruedas delanteras tienen el giro muy limitado.

Por último, decir que tuve que modificar por dentro el muelle que llevaban las ruedas delanteras, ya que al tener demasiado peso encima, estos se hundían provocando el roce de las ruedas con el chasis y con ello también provocaba el no avance del vehículo. Esto no se pudo mejorar ya que el coche es demasiado pequeño para poner un motor más grande y más potente, además que se percató de este problema al final del proyecto, cuando se hizo la prueba definitiva.

10. CONCLUSIONES

Durante la realización de este proyecto he podido mejorar lo poco que sabía sobre la programación en lenguaje C++. Aprendí a programar un poco con este lenguaje en tercero del Grado en Ingeniería Eléctrica, el cual estoy terminando ahora. Pero sin lugar a dudas, el reto principal era la escritura del código en lenguaje C++ y usando la tecnología de la visión artificial. Decir que gracias a este campo ahora poseo muchos más conocimientos con los que defenderme en el futuro, ya sea en el lenguaje C++, como en la visión artificial.

Aparte de haber aprendido sobre programación, también tuve la oportunidad de manejarme tanto en Linux, como en el programa de diseño SolidWorks, con el que se pueden diseñar objetos para posteriormente imprimirlos en 3D.

Cuando comencé con este proyecto, no tenía idea que la visión artificial fuera tan potente. Pero a raíz de trabajar con ella he podido observar que es una herramienta muy potente, la cual puede adquirir mucha información a través de imágenes o videos realizados anteriormente o en tiempo real. A día de hoy se encuentran muchas empresas, como por ejemplo redes sociales, automovilísticas o fabricantes de drones, que quieren utilizar la tecnología de la visión artificial para mejorar nuestro día a día, ya que, gracias a esto, podríamos incluso predecir un accidente automovilístico a base de ir adquiriendo datos de imágenes e ir guardándolos.

Resumiendo, puedo afirmar que la realización del presente proyecto, me ha dotado de nuevos conocimientos que pueden ser utilizados en diversas disciplinas.

Para terminar, he de decir que nunca antes me había enfrentado a problemas de esta magnitud y solventarlos por mí mismo.

11. BIBLIOGRAFÍA

- Información del lenguaje C++:

http://www.cplusplus.com/

- Información de la librería OpenCV:

http://opencv.org/

- Información sobre la librería WiringPi:

http://wiringpi.com/

http://rpiplus.blogspot.com.es/2013/06/libreria-wiring-pi.html

- Información sobre la Raspberry Pi 2:

https://i1.wp.com/www.pighixxx.com/test/wp-content/uploads/2015/06/raspberry.png

https://es.pinout.xyz/pinout/pin2_alimentacion_5v

https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/Raspberry-Pi-2B-V1.2-Schematics.pdf

- Información sobre el shield L293D:

http://www.electrodragon.com/w/index.php?title=L293&redirect=no

- Información sobre el servomotor:

http://hertaville.com/rpiservo.html

http://www.rctecnic.com/servos-micro-mini-3-30gr/479-servo-analgico-emax-es08a-9gr-18kg

http://projectedneuralactivity.blogspot.com.es/2012/12/controlling-servo-using-raspberry-pi.html