UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Sistema empotrado para el control autónomo de un coche a través de las imágenes recibidas por una cámara

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Gao Yao, Jia Cheng

Tutor/a: Simó Ten, José Enrique

Cotutor/a: Balbastre Betoret, Patricia

CURSO ACADÉMICO: 2021/2022

Sistema empotrado para el control autónomo de un coche a través de las imágenes recibidas por una cámara

Autor
JIA CHENG GAO YAO
(alfredo_gao@hotmail.com)

Tutor: JOSÉ ENRIQUE SIMÓ TEN
(jsimo@disca.upv.es)

TRABAJO FIN DE MÁSTER
MÁSTER EN AUTOMÁTICA E INFORMÁTICA INDUSTRIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
Valencia, 2022

AGRADECIMIENTOS

Quería aprovechar la culminación de este proyecto, para agradecer el haber llegado hasta aquí, a todas las personas que me han acompañado día a día. Primero de todo, agradecer a mis padres, quienes han sacrificado todo lo necesario para que yo siga estudiando.

Por otro lado, agradecerles de corazón, a mi pareja y su familia, por haberme dado un hogar donde seguir realizando este proyecto y un apoyo crucial para acabarlo.

Por último, y no menos importante, agradecerle a mi tutor José Enrique Simó Ten, la paciencia que ha tenido para transmitirme todos sus conocimientos y experiencia, haciendo que este proyecto sea posible.

RESUMEN

En este Trabajo de Fin de Máster (TFM) se ha buscado desarrollar la programación para el control de un vehículo radiocontrol y la creación de un sistema para la implementación de una aproximación a la conducción autónoma. Para ello, se han analizado las diferentes alternativas existentes en el mercado actual y finalmente, se ha decidido trabajar con un kit “PiRacer Pro AI” de la marca Waveshare. Para la programación se ha utilizado el lenguaje C++. Además, se han creado tres modos de funcionamiento: control por mando, conducción asistida y conducción autónoma. El método de funcionamiento más completo, la conducción autónoma, contará con diferentes funcionalidades como: la detección de señales de “stop”, la frenada de emergencia o la detección de la posición del vehículo en la carretera. Por otro lado, se analizarán diversos métodos para aplicar la visión artificial, con el fin de obtener la combinación de técnicas que ofrezca mayor robustez al funcionamiento del sistema.

RESUM

En aquest Treball de Fi de Màster s'ha buscat desenvolupar la programació per al control d'un vehicle radiocontrol i la creació d'un sistema per a la implementació d'una aproximació a la conducció autònoma. Per a això, s'han analitzat les diferents alternatives existents en el mercat actual i finalment, s'ha decidit treballar amb un kit "PiRacer Pro AI" de la marca Waveshare. Per a la programació s'ha utilitzat el llenguatge C++. A més, s'han creat tres mètodes de funcionament: control per comandament, conducció assistida i conducció autònoma. El mètode de funcionament més complet, la conducció autònoma, comptarà amb diferents funcionalitats com: la detecció de senyals de "stop", la frenada d'emergència o la detecció de la posició del vehicle en la carretera. D'altra banda, s'analitzaran diversos mètodes per a aplicar la visió artificial, amb la finalitat d'obtindre la combinació de tècniques que oferisca major robustesa al funcionament del sistema.

ABSTRACT

In this Master's Thesis we have sought to develop the programming for the control of a radio-controlled vehicle and the creation of a system for the implementation of an approach to autonomous driving. For this, the different alternatives existing in the current market have been analyzed and finally, it has been decided to work with a “PiRacer Pro AI” kit from the Waveshare brand. For programming, the C++ language has been used. In addition, three operating modes have been created: remote control, assisted driving and autonomous driving. The most complete operating mode, autonomous driving, will have different functionalities such as: detection of stop signs, emergency braking or detection of the vehicle's position on the road. On the other hand, some methods to apply artificial vision will be analyzed, in order to get the combination of techniques that offers greater robustness to the operation of the system.

DOCUMENTOS

1. Memoria
2. Planos y código fuente
3. Presupuesto
4. Pliego de condiciones

1. MEMORIA

Índice de contenido

Índice de tablas

Índice de figuras

1. INTRODUCCIÓN

En una sociedad donde se busca la similitud entre las capacidades de las máquinas y los seres humanos, está aumentando la creación de tecnologías como la visión artificial. Ésta es un campo de la Inteligencia Artificial (IA) que permite a las máquinas la captura, el procesamiento y el análisis de imágenes, con el fin de obtener información para orientar su funcionamiento.

Antes de la existencia de la IA ya se empleaban métodos clásicos de la visión artificial que permitían procesar imágenes y reconocer características básicas de estas. Como, por ejemplo, detectar los extremos de los objetos o diferenciar entre colores. Aunque debían ser operaciones simples, con formas previsibles y ajustarse a un patrón fijo. Aun así, las técnicas clásicas de visión artificial han supuesto un impacto industrial enorme. Esto es debido a que, entre otras cosas, las máquinas no se cansan, pueden funcionar a mayor velocidad que los seres humanos, no sufren limitaciones en la visión y pueden incorporar tecnologías muy avanzadas, como la visión térmica o los rayos X.

La IA ha permitido la aplicación de redes neuronales para mejorar la precisión y versatilidad de los métodos clásicos. Para ello, las redes neuronales realizan un entrenamiento con una gran cantidad de imágenes. A partir de las cuales obtienen patrones con los que pueden hallar los detalles más sutiles, que les permiten detectar los objetos en cuestión.

Gracias a estas tecnologías, han surgido otras nuevas como la conducción autónoma. Ésta, junto a la conectividad y la seguridad, son algunas de las últimas tendencias en la industria del automóvil. Buscando como objetivos, evitar errores humanos, como pueden ser las distracciones; reducir el tiempo de reacción o mejorar la experiencia de usuario del conductor. Concretamente, la conducción autónoma es una modalidad de conducción que permite el manejo del vehículo sin necesitar un control activo por parte del conductor.

Por otro lado, en la industria se está apreciando un auge en los vehículos de guiado automático (AGV). Estos consisten en vehículos autónomos encargados del transporte de materiales sin la necesidad de un operario que lo maneje. De esta forma, este proyecto también podría plantearse como un AGV con control por visión artificial.

El objetivo de este TFM es desarrollar un proyecto donde seamos capaces de programar una Raspberry Pi para controlar un coche radiocontrol. Así como la creación de diferentes modos de funcionamiento, entre los que se incluyen, la utilización de la sensorización y la visión artificial.

2. JUSTIFICACIÓN

La programación de un coche radiocontrol con conducción autónoma es un proyecto completo que combina la programación de sistemas empotrados y la aplicación de la visión artificial. Sin embargo, la programación de una conducción autónoma completa es una tarea muy difícil por lo que se han ido planteando diferentes modos de funcionamiento que han ido incorporando progresivamente mayores prestaciones. Creando así un modo completamente controlado por el usuario, un modo semi-autónomo que ayuda al usuario en casos extremos para evitar que el vehículo se salga del circuito y, por último, un modo con la conducción autónoma completa.

Por otro lado, este proyecto está relacionado con varias áreas temáticas que se han tratado en el máster, por lo que es una buena oportunidad para adquirir experiencia en cómo estas diferentes disciplinas, interactúan en un único proyecto.

Además, la inteligencia artificial y en concreto, la visión artificial, son unas tecnologías que aún se están en fase de desarrollo, pudiendo lograrse infinidad de mejoras tanto para el proceso industrial como para el día a día de las personas. Es por esto que se desea desarrollar un proyecto en el que se pueda profundizar más en el funcionamiento de estas tecnologías.

3. OBJETIVOS

En este apartado se van a describir los objetivos generales y específicos del proyecto.

3.1 OBJETIVO GENERAL

La finalidad del proyecto es el desarrollo de una aproximación a la implementación de un sistema de conducción autónoma para un vehículo de modelismo. Para ello se deberá programar el control de la dirección y la velocidad de un coche radiocontrol para luego, utilizar sensorización y visión artificial con el fin de lograr una conducción autónoma.

3.2 OBJETIVOS ESPECÍFICOS

3.2.1 Establecer una infraestructura software para acceder a los actuadores y sensores del vehículo.

- Evaluar el uso de proyecto de código abierto como “Donkeycar”.
- Desarrollo de módulos específicos en lenguaje C++.

3.2.2 Desarrollar una infraestructura software para la ejecución de comunicaciones y comportamientos.

3.2.3 Introducir la capacidad de teleoperación.

- Transmisión de señales de operación.

3.2.4 Desarrollo de comportamientos básicos.

- Transmisión de imagen en directo.
- Detección y seguimiento de líneas en el suelo.
- Detección de obstáculos.

3.2.5 Desarrollo de comportamientos avanzados:

- Detección de señales de tráfico.
- Modificación del comportamiento de acuerdo con las señales detectadas.

4. ANTECEDENTES

En este apartado se van a analizar los proyectos que existen actualmente en el mercado y que aplican la visión artificial para el control de un coche radiocontrol.

4.1 PROYECTO “DONKEYCAR”

Figura 1. Vehículo autónomo construido en el proyecto “Donkeycar”.

Donkeycar es una plataforma online que guía al usuario para construir un coche con conducción autónoma a partir de un coche radiocontrol convencional. Destaca por la facilidad de funcionamiento ya que se puede montar sobre una gran variedad de coches radiocontroles y puedes tener el prototipo funcionando en menos de un día (tomado de [1]). Sin embargo, es un código complejo y difícil de entender si quieres aprender cómo se ha llevado a cabo su programación.

Entre otras cosas ofrece:

- Instrucciones completas para crear tu propio modelo físico.
- Manual completo para realizar toda la configuración software.
- Control del vehículo con el teclado del ordenador, un control remoto o con el móvil a través de una APP.
- Entrenar un piloto automático que sea capaz de seguir un circuito.
- Utilizar un simulador para evitar la construcción de un vehículo físico.
- Funcionalidades adicionales como detectar señales de “stop”, control por voz, sensor IMU o LIDAR.

Para su funcionamiento únicamente se utiliza un coche similar al “Exceed RC Magnet”, una Raspberry Pi 3B+, una batería para la Raspberry Pi y una cámara gran angular. Por último, cabe destacar que este proyecto utiliza el lenguaje de programación Python.

4.2 PROYECTO "*BUILD YOUR OWN SELF DRIVING CAR"*

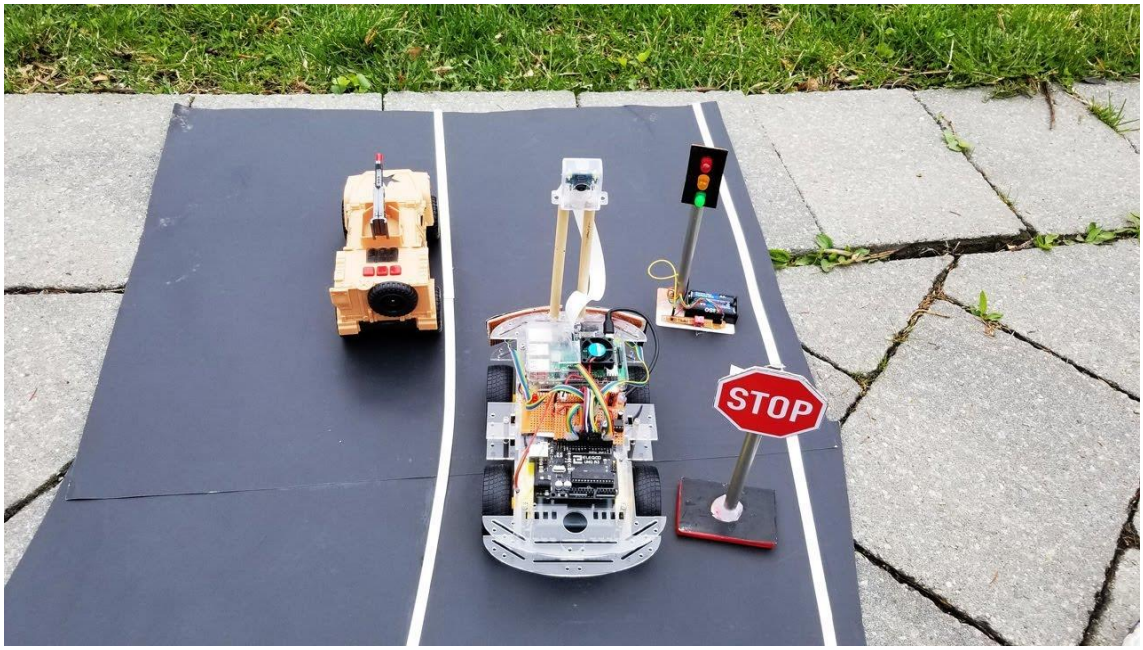


Figura 2. Vehículo autónomo construido en el proyecto "Build your own self driving car".

Se trata de un proyecto programado en C++ el cual es capaz de detectar semáforos en rojo, seguir la carretera, detectar y esquivar objetos, detectar señales de "stop" y utilizar luces de freno e intermitentes. Para ello combina un chasis simple, una Raspberry Pi 3B+, un Arduino UNO, un driver para los motores, una batería portátil y una cámara gran angular (tomado de [2]).
Durante el desarrollo de este curso te enseñan:

- Realizar el montaje del hardware.
- Configurar el software.
- Configurar la cámara.
- Utilizar OpenCV en la Raspberry.
- Utilizar el Arduino UNO en combinación con la Raspberry.
- Técnicas clásicas de visión artificial.
- Entrenar crear tu propio Clasificador en cascada para detectar cualquier objeto.

4.3 PROYECTO “SELF DRIVING RC CAR”

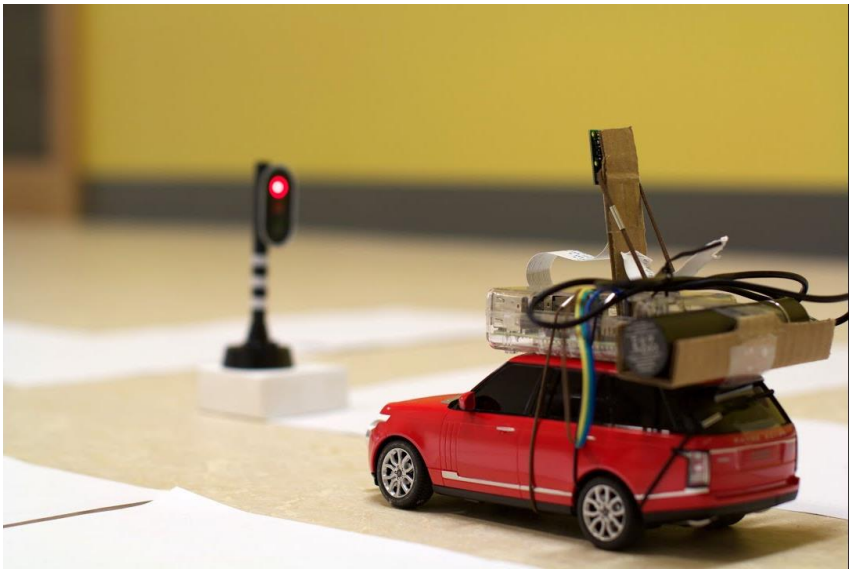


Figura 3. Vehículo autónomo construido en el proyecto “Self driving rc car”.

Proyecto capaz de detectar semáforos en rojo, señales de “stop”, seguir un circuito y detenerse ante la presencia de obstáculos. Para ello, utiliza una red neuronal para calcular la dirección que debe seguir el vehículo, un clasificador Haar Cascade para detectar las señales de “stop” y los semáforos, un sensor de ultrasonidos para evitar la colisión delantera con un obstáculo, una Raspberry para procesar toda la información y un Arduino para el control del coche radiocontrol. El lenguaje de programación utilizado en la Raspberry Pi para procesar toda la información es Python (tomado de [3] y [4]).

5. SELECCIÓN DE LA PLATAFORMA EXPERIMENTAL

La elección entre las diferentes alternativas se va a realizar, en base a los componentes, tales como el micro ordenador, el sensor de medición de distancia, la cámara, el chasis del coche y los kits completos.

5.1 SOLUCIONES HARDWARE

5.1.1 Microordenador

Se trata de un ordenador reducido u ordenador de placa única “Single Board Computer” (SBC) de bajo coste. Va a ser el encargado de procesar toda la información y ejecutar un comportamiento u otro, según la información recibida. Teniendo en cuenta que buscamos trabajar con un sistema Embedded Linux y wifi las mejores alternativas son (tomado de [5]):

5.1.1.1 Raspberry Pi 4B

Figura 4. Ordenador reducido Raspberry Pi 4B.

Éste fue desarrollado en Reino Unido con fines académicos, fomentando la enseñanza informática en las escuelas. Desde su lanzamiento en 2012 han ido sacando diferentes modelos, de los cuales vamos a tomar como referencia el último modelo ya que se adapta mejor a las últimas tecnologías, además de garantizar mejores prestaciones. La Raspberry pi 4B cuenta con un sistema en un chip SOC (“System On a Chip”) Broadcom BCM2711 con una CPU ARM Cortex-A72 (ARM v8) de 64 bits con cuatro núcleos a 1.5GHz y una GPU VideoCore VI 500MHz. Está disponible con 1, 2, 4 u 8 GB de memoria RAM LPDDR4. Dispone de conectividad Wi-Fi 2,4GHz/5GHz, Bluetooth 5.0, Gigabit Ethernet (tomado de [6]). También cuenta con las siguientes entradas y salidas:

- 40 pines GPIO.
- 2 puertos micro-HDMI que ofrecen una resolución máxima de 4K a 60Hz. Compatibles con la tecnología HDMI-CEC.
- 2 puertos USB 2.0.
- 2 puertos USB 3.0.
- CSI para cámara MIPI.
- DSI para pantalla MIPI.
- Salida estéreo de 4 polos y puerto de video compuesto.
- Lector tarjetas microSD.
- Alimentación por vía USB-C o por cabezal GPIO.

5.1.1.2 NVIDIA Jetson Nano

Figura 5. Ordenador reducido NVIDIA Jetson Nano.

Jetson es la familia de SBC creada por NVIDIA. Combina su familia de SoC Tegra y sus GPU para ordenadores, donde los Tensor Cores con la ayuda de núcleos ARM muy potentes, cuentan con aceleradores y procesadores diseñados para ser utilizados en la robótica e inteligencia artificial,

Incorpora una CPU de ARM de cuatro núcleos A57 a 1,43 GHz y una GPU NVIDIA Maxwell de 128 núcleos. También se puede destacar la presencia de una memoria RAM LPDDR4 de 4 GB, 64 bits y 25,6 GB/s (tomado de [7] y [8]). Otras características que podemos encontrar son:

- Conectividad Gigabit Ethernet.
- Dos conectores CSI-2 para cámaras MIPI.
- Un conector HDMI y un “DisplayPort” para conectar un monitor.
- Un lector de tarjetas microSD.
- Cuatro conectores USB 3.0
- Cuarenta pines (GPIO, I2C, I2S, SPI, UART), 12 pines (Alimentación, UART) y 4 pines para el ventilador.

5.1.1.3 ASUS Tinker Board S

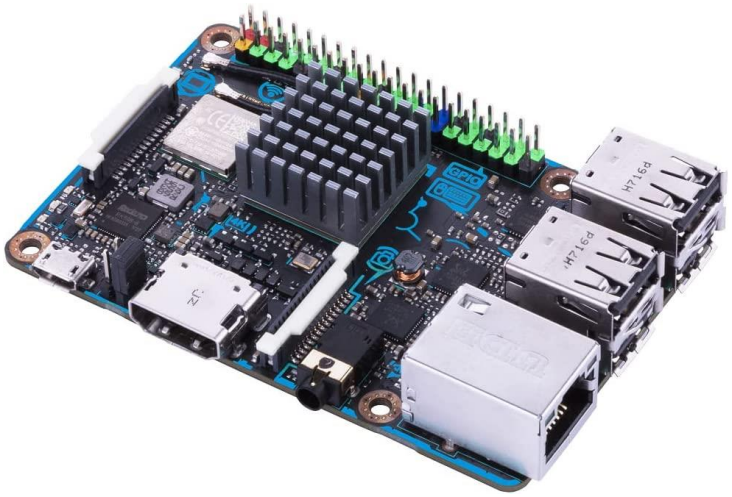


Figura 6. Ordenador reducido ASUS Tinker Board S.

Esta placa SBC cuenta con un potente procesador de cuatro núcleos (Rockchip RK3288) y 2 GB de memoria RAM LPDDR3 de doble canal. También cuenta con una GPU Mali-T764. Además, ofrece la posibilidad de crear tu propio proyecto al contar con 40 conectores GPIO. También cuenta con un conector DSI MIPI para pantallas y conexión CSI MIPI para cámaras compatibles. Incluye 16 GB de memoria eMMC integrada y una ranura microSD para ofrecer una mayor versatilidad (tomado de [9]). Otras características que podemos encontrar son:

- Detección de entradas de alto voltaje para evitar problemas con la alimentación.
- Un puerto HDMI compatible con HDMI-CEC al igual que la Raspberry Pi.
- Conexión Wifi, Bluetooth y Ethernet.
- Cuatro conectores USB 2.0.

5.1.2 Sensor de distancia

Teniendo en cuenta que no se necesita un dispositivo con una gran precisión, encontramos las siguientes opciones:

5.1.2.1 Sensor de ultrasonidos HC-SR04

Figura 7. Sensor de distancia por ultrasonidos HC-SR04.

Este sensor tiene un rango de detección de 3 cm a 4 m, con una sensibilidad de 3 mm. Con este sensor se puede realizar un máximo de 50 mediciones por segundo. Su voltaje de funcionamiento son 5V y consume una corriente de aproximadamente 2 mA por medición (tomado de [10]). Su método de funcionamiento es sencillo, el gatillo o *trigger* envía un pulso de ultrasonido y detecta cuánto tiempo tarda en volver. Teniendo en cuenta este tiempo, la velocidad del sonido y que se realiza el recorrido dos veces (ida y vuelta) se puede calcular fácilmente la distancia.

5.1.2.2 Sensor IR

Figura 8. Sensor de distancia por luz IR.

Su funcionamiento se basa en un pulso de luz infrarroja que se emite desde el emisor y ante la presencia de un objeto, rebota y es detectado por el receptor (tomado de [11]). Sin embargo, a diferencia del sensor de ultrasonidos, este sensor no nos devuelve un valor analógico proporcional a la distancia, sino que posee un comparador con un potenciómetro, a partir de los cuales podemos establecer una distancia de referencia. De esta forma, el sensor nos devuelve un valor digital. Siendo este valor “0”, si la distancia es menor, y “1”, si la distancia es mayor. Este sensor posee un rango de detección de entre 2 y 30 cm, con un ángulo de detección de 35 grados. Además, puede funcionar con una alimentación de entre 3 y 5 V.

5.1.3 Chasis del coche

Este compondrá la estructura principal del proyecto y estará formado principalmente por carrocería, las ruedas, el ESC, el motor con escobillas para el desplazamiento y el servomotor para la dirección.

5.1.3.1 Dilwe

Figura 9. Coche radiocontrol Dilwe.

Este coche RC cuenta con tracción a las cuatro ruedas y diferenciales en la parte delantera y trasera. También cuenta con una suspensión independiente en cada rueda, consiguiendo una conducción muy estable. Incluye un control remoto de 2,4 GHz y un receptor resistente al agua que permite el control hasta un máximo aproximado de 80 metros. Utiliza batería de iones de litio de alta eficiencia (7,4 V 1500 mAh) y un motor con escobillas que le permite alcanzar una velocidad máxima de 50 km/h y una duración aproximada de 25 minutos (tomado de [12]).

5.1.3.2 Exceed Buggy Blaze

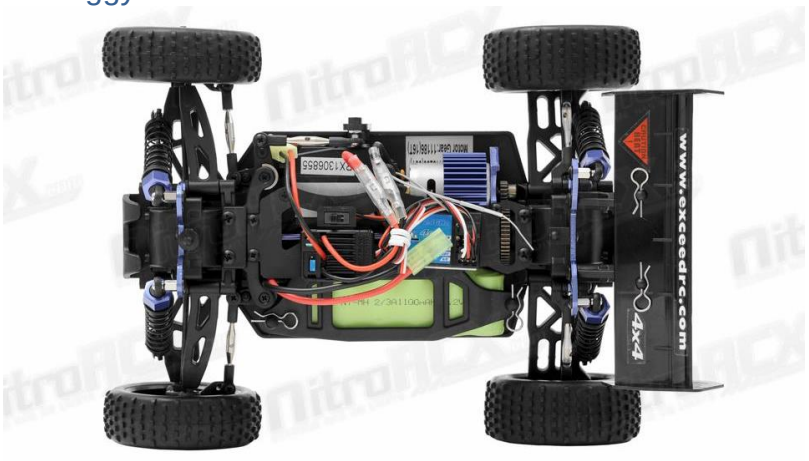


Figura 10. Coche radiocontrol Exceed Buggy Blaze.

El Exceed Metallic Blaze RC buggy es un coche radiocontrol a escala 1/16 que posee transmisión en las cuatro ruedas. Cuenta con una batería Ni-MH de 7,2 V 1100 mAh. También podemos destacar la suspensión independiente en cada rueda, con resortes helicoidales y amortiguadores hidráulicos. Además, posee protección delantera y trasera para reducir el impacto de los golpes en la carrocería. Por último, destacar que el control se realiza a una frecuencia de 2,4 GHz (tomado de [13]).

5.1.3.3 YONCHER YC200

Figura 11. Coche radiocontrol YONCHER YC200.

Se trata de un coche radiocontrol de estala 1/16 con el que se puede alcanzar una velocidad máxima de 45 Km/h. Con un sistema de tracción a las cuatro ruedas, gracias a un eje de transmisión y un diferencial independiente para las ruedas delanteras y uno para las ruedas traseras. Además, cuenta con suspensión formada por amortiguadores hidráulicos y resortes helicoidales. Garantiza la impermeabilidad IPX5 y un control a una frecuencia de 2,4 GHz que permite una distancia máxima entre el controlador y el coche de aproximadamente 100 metros.

Además, posee un sistema de protección contra sobrecorrientes y altas temperaturas lo que aumenta la seguridad del vehículo y su integridad. Por último, cuenta con una batería de 7,4 V y 1500 mAh que ofrecen una duración de funcionamiento de aproximadamente 25 minutos (tomado de [14]).

5.1.4 Cámara

5.1.4.1 AZDelivery Cámara Raspberry Pi

Figura 12. Cámara AZDelivery para Raspberry Pi.

Esta cámara de 5 megapíxeles y sensor Omnivision 5647 CMOS permite tomar imágenes de hasta 2592 x 1944 px y videos 1080p a 30 fps o 720p 60 fps. Cuenta con un cable plano flexible para una fácil conexión con un conector CSI-2 (tomado de [15]).

5.1.4.2 KEYESTUDIO Módulo de Fisheye Cámara para Raspberry Pi

Figura 13. Cámara KEYESTUDIO para Raspberry Pi.

Esta cámara de 25 x 24 mm cuenta con un chip fotosensible Omnivision 5647 CMOS, 5 MP, longitud focal ajustable y ángulo de disparo de 130 grados. Además, funciona a una tensión de 3,3 V y posee visión nocturna (tomado de [16]).

Por último, destacar que es compatible con conector CSI mediante un cable plano.

5.1.4.3 Electreeks® Raspberry Pi Camera Module

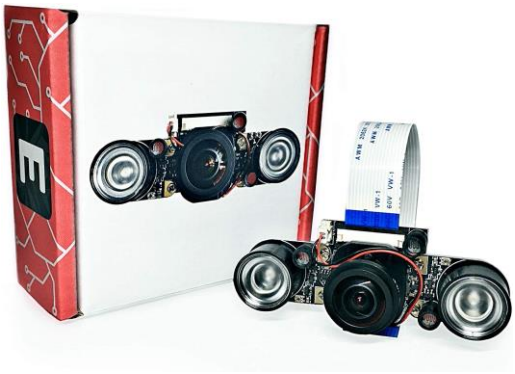


Figura 14. Cámara Electreeks para Raspberry Pi.

Esta cámara posee un ángulo de captura de 175 grados con un sensor de 5 MP Omnivision 5647 CMOS. Además, consigue una resolución máxima de 2592 x 1944 píxeles y en video 1080p a 30 fps. Con unas dimensiones de 80 x 30 x 28 mm y un cable plano de 15 cm, es totalmente compatible con cualquier conector CSI.

Gracias a las fotorresistencias que posee, la cámara activará automáticamente los leds IR de los laterales asegurando una buena toma de las imágenes incluso en situaciones con luminosidad reducida (tomado de [17]).

5.1.5 Kits completos

5.1.5.1 Donkey Car S1

Figura 15. Kit completo Donkey Car S1.

Se trata de un vehículo listo para usar, controlado por una Raspberry Pi 4B y una cámara gran angular. Utiliza una PCB llamada Donkey HAT para facilitar las conexiones y evitar que éstas se realicen de forma incorrecta. Viene con una pantalla de 0,91 pulgadas OLED, que te muestra diferentes informaciones sobre el sistema como puede ser la dirección IP para conectarse de forma remota (tomado de [18]). Cabe destacar que se necesitará una batería portátil para alimentar a la Raspberry ya que el producto no lo incluye.

Otras características que tienes son:

- Batería NiMH que ofrece una duración de entre 15 y 30 minutos.
- Chasis HSP 94186.
- Tarjeta microSD de 16 GB.
- Conectividad Wifi.

5.1.5.2 PiRacer Pro AI

Figura 16. Kit completo “PiRacer Pro AI”.

Vehículo con tracción a las cuatro ruedas y suspensión independiente en cada una de ellas. Esta suspensión está compuesta por amortiguadores hidráulicos y resortes helicoidales. Además, cuenta con diferenciales en los ejes delanteros y traseros reduciendo así, el deslizamiento de las ruedas. También tiene un motor RC380 con escobillas de carbono, engranajes metálicos, un servomotor E6001 con 6kg/cm de torque y una esponja frontal para prevenir daños por impactos (tomado de [19]). Por otro lado, cuenta con otros elementos como:

- Raspberry Pi 4B
- Cámara gran angular con 5MP, 160 grados de ángulo de captura y un sensor de alta calidad OV5647
- ESC (Electronic Speed Controller) impermeable, con control bidireccional y protección de bajas tensiones.
- Mando inalámbrico para el control remoto.
- Circuito de 3 x 2 m.
- Sistema de protección ante sobrecorrientes.

También hay que destacar que cuenta con una placa de expansión con diferentes funciones:

- Regulador de tensión automático de tipo Buck-Boost
- Sistema de protección para la batería
- Control del nivel de la batería
- 4 baterías de alta capacidad 18650, formado una tensión total de 8,4 V
- Driver para el motor
- Pantalla OLED de 0,91 pulgadas y 128x32 pixeles

5.1.5.3 JetRacer Pro

Figura 17. Kit completo JetRacer Pro.

Es kit es muy parecido al anterior. Sin embargo, se pueden encontrar varias diferencias como el SBC, que no es una Raspberry Pi 4B sino un Jetson nano, con un sistema de ventilación formado por un disipador y un ventilador

Además, cuenta con una campara de 8MP, con un ángulo de captura de 160 grados y un sensor IMS219 que ofrece una resolución máxima de 3280 x 2464 píxeles (tomado de [20]).

Respecto a todas las demás características, son las mismas que para el kit anterior.

6. SOLUCIÓN ADOPTADA

En esta sección se van a describir todos los aspectos relativos a la solución final adoptada para este proyecto

6.1 SELECCIÓN DE LA PLATAFORMA

Para llevar a cabo la realización del proyecto se ha optado por el kit “PiRacer Pro AI" de la marca Waveshare. Esto es debido a que por una parte el tutor ya disponía de este kit, y, por otra parte, tiene todos los componentes que necesitamos para desarrollar nuestro proyecto; a falta del sensor para medir la distancia. En este caso, se utilizará un sensor de ultrasonidos HC-SR04, ya que su funcionamiento no se ve afectado por condiciones ambientales (al sensor IR puede afectarle la luz del sol) y, además, con el sensor seleccionado se puede obtener un valor analógico de la distancia.

6.2 METODOLOGÍA DEL DESARROLLO

Para llevar a cabo el desarrollo del proyecto, se ha seguido una estructura en espiral. Donde en un principio se han partido de unos objetivos simples, y se ha ido desarrollando el proyecto según se han ido cumpliendo estos objetivos. Al mismo tiempo, se han ido solventando los problemas que han aparecido.

Para ello, se va a trabajar desde el SO (sistema operativo) “Raspbian” instalado en una Raspberry Pi 4B. Esta va a ser controlada remotamente desde un ordenador mediante la comunicación por VNC. La programación se va a realizar en un código CPP que se va a compilar a través de un archivo de tipo “makefile”.

6.3 DISEÑO DEL SOFTWARE

Respecto a la programación del software se propone una estructura de actividades concurrentes o hilos de ejecución. De esta forma, podremos realizar varios procesos en paralelo y reducir así, el tiempo de procesamiento.

Sin embargo, como se va a trabajar con variables globales que serán utilizadas por varios hilos, utilizaremos las variables tipo “mutex” para evitar que se corrompan. Además, para garantizar el funcionamiento sincronizado entre determinados hilos, utilizaremos las variables tipo “condition”. Las cuales nos permiten pausar un hilo hasta que se activa con una llamada.

6.3.1 Main

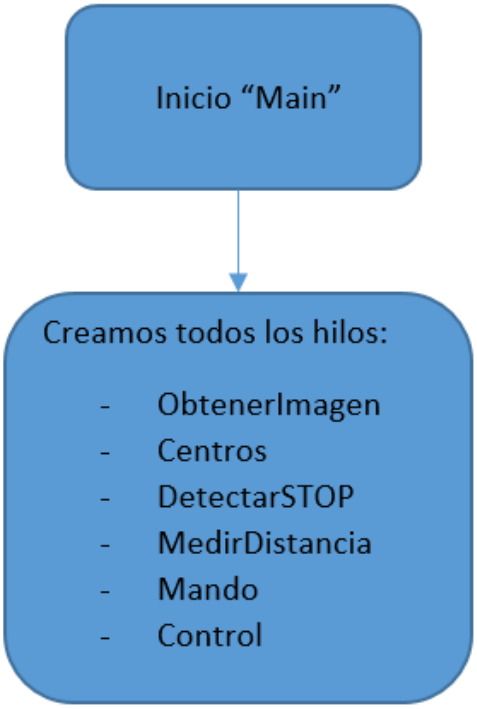


Figura 18. Flujograma de la función principal del código (Main).

En la función principal del código se inicializan los seis hilos que componen la totalidad del código. Los cuales se explicarán en los siguientes apartados.

6.3.2 ObtenerImagen

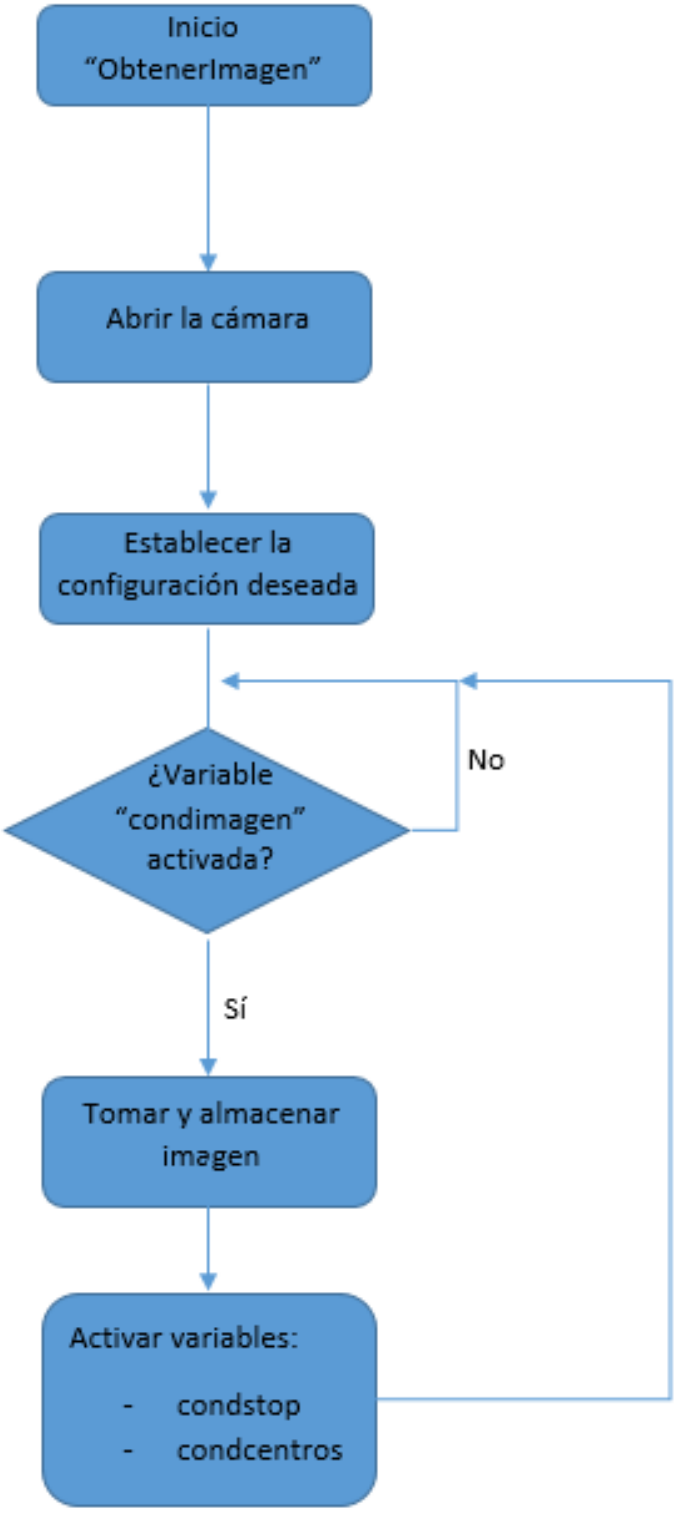


Figura 19. Flujograma del hilo "ObtenerImagen".

El hilo “ObtenerImagen” será el encargado de proporcionar al sistema una imagen sobre la que trabajar. Para ello, primero se abre la cámara y se establece la configuración deseada. En este caso, se define un tamaño de “frame” de 640 x 480 píxeles (ancho x alto) y un tamaño de “buffer” de 0 imágenes.

Posteriormente, se comprueba si se ha activado la variable tipo “condition” para que se ejecute, en este caso “condimagen”. Esta variable será la encargada de pausar el hilo hasta que se active. Una vez se haya activado, se toman 5 imágenes. Esto se realiza debido a que el sistema tiene establecido por hardware un buffer de dos imágenes y de esta forma, se llena ese buffer de imágenes con el instante actual. Si no se hace esto, el proceso trabajará con una imagen retrasada de aproximadamente 1 segundo, debido al tiempo de procesamiento de nuestro código.

Una vez tomadas estas imágenes, se rota la última 180 grados y se almacena como una variable global dentro de un mutex, para evitar problemas con otros hilos.

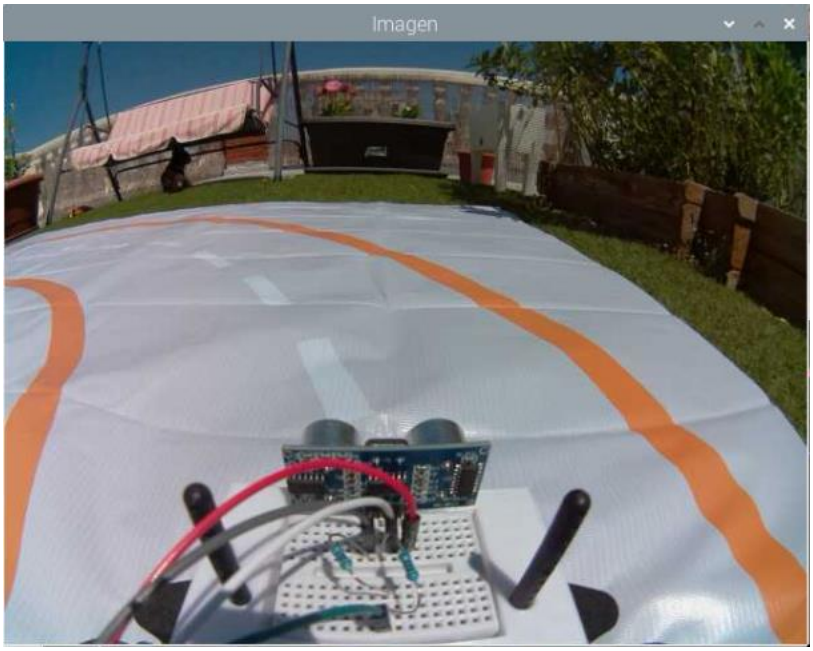


Figura 20. Imagen tomada desde el hilo “ObtenerImagen”.

Una vez hecho esto, se activan las variables “condstop” y “condcentros” que permiten la reanudación del funcionamiento de los hilos “Centros” y “DetectarSTOP”, ya que estos necesitan la imagen obtenida para poder trabajar.

Este proceso se repite en bucle desde que se comprueba si la variable “condimagen” está activada hasta que se activan las variables “condstop” y “condcentros”.

6.3.3 Centros

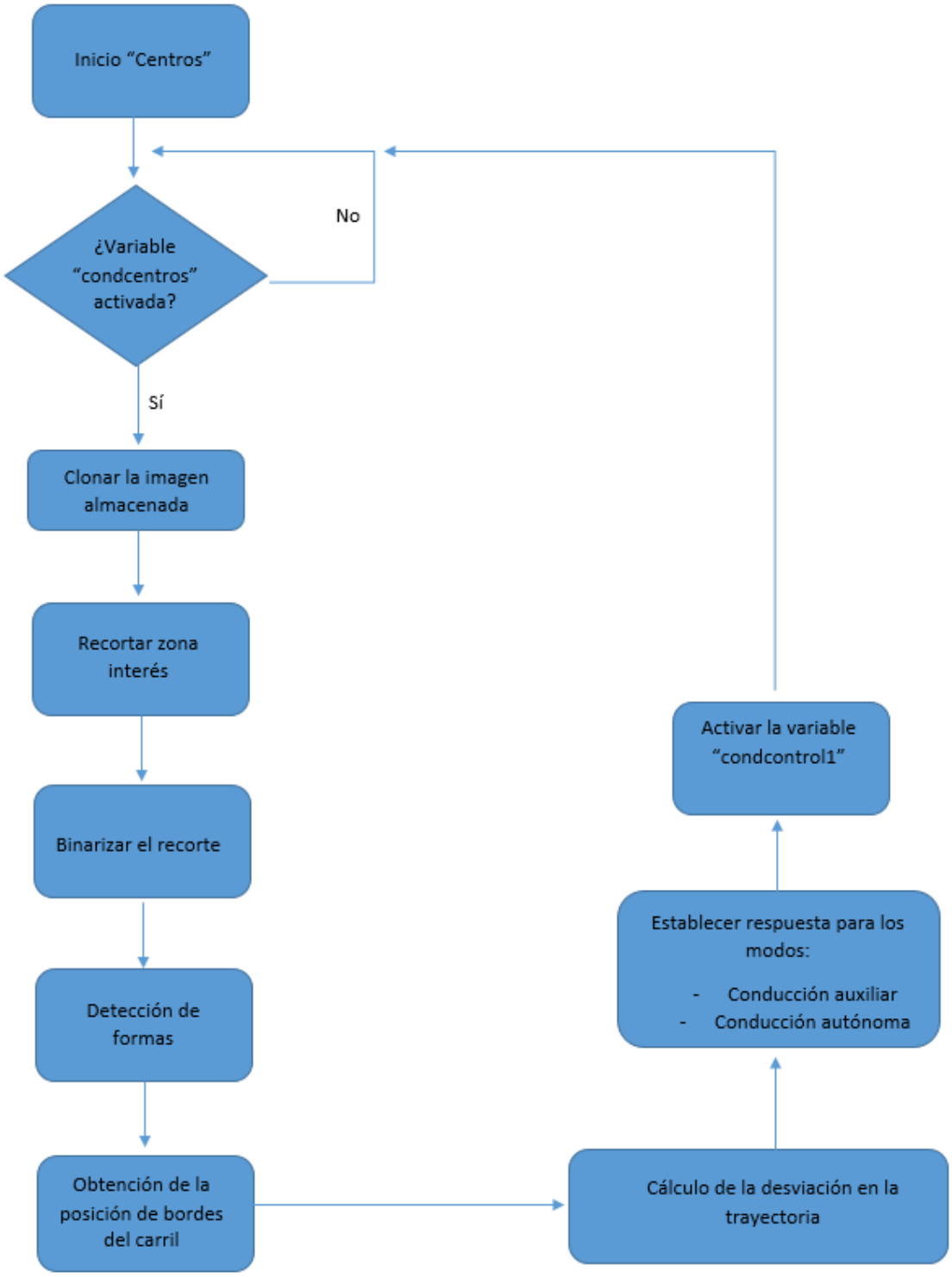


Figura 21. Flujograma del hilo "Centros".

Este hilo será el encargado de calcular la posición del vehículo en la carretera. Para ello, primero comprueba si se ha activado la variable que lo activa (“condcentros”) y de ser así, realiza el siguiente proceso.

Primero se clona la imagen que se ha almacenado como variable global en el apartado anterior y se recorta un rectángulo de 640 x 40 píxeles, con un origen en (0, 250) respecto al origen de coordenadas globales. Cabe destacar que siempre que se hable de coordenadas, el origen de coordenadas se situará en la esquina superior izquierda de la imagen, siendo el eje x en horizontal y con valores positivos hacia la derecha; y el eje y en vertical y con valores positivos hacia abajo. Este recorte se ha hecho para determinar una zona en concreto donde se realizará la detección de la posición de las líneas de la carretera respecto al centro del vehículo. Para determinar esta zona, se ha situado el vehículo en el circuito y se ha observado aproximadamente en qué punto debería empezar a detectar la curva y la zona de la imagen en la que se puede apreciar ese cambio en la orientación.

Figura 22. Posición determinada donde se ha situado el vehículo considerando que debería empezar a detectar la curva.

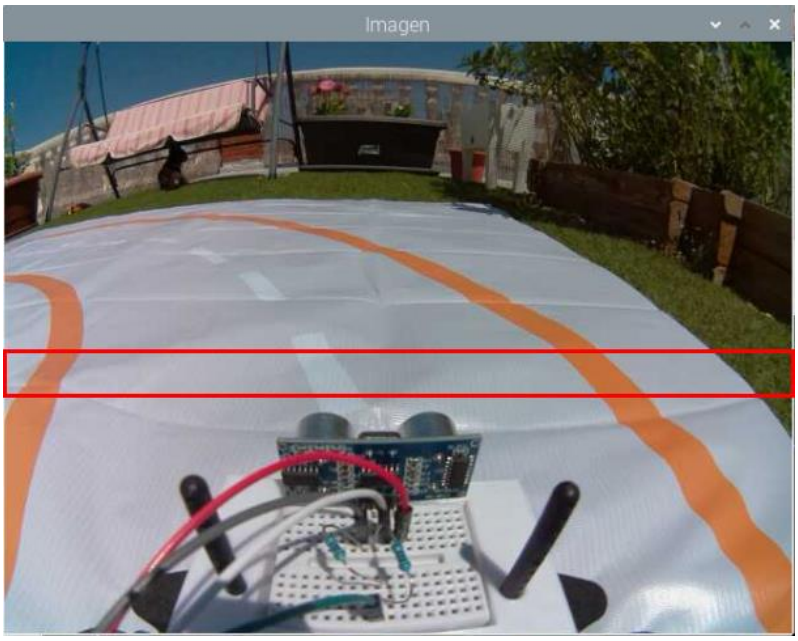


Figura 23. Imagen captada desde la cámara en la posición determinada, donde se ha establecido la zona en la que se debe detectar la desviación en la trayectoria (marcada en rojo).

Figura 24. Imagen recortada resultante.

Posteriormente se convierte el espacio de color de la imagen recortada de BGR (Blue Green Red) a HSV (Hue Saturation Value) y se binariza según los parámetros determinados. Estos parámetros se obtienen tomando una imagen en vivo y variando los parámetros hasta obtener la imagen binarizada que mejor aísle los objetos a detectar, en este caso las líneas del circuito.

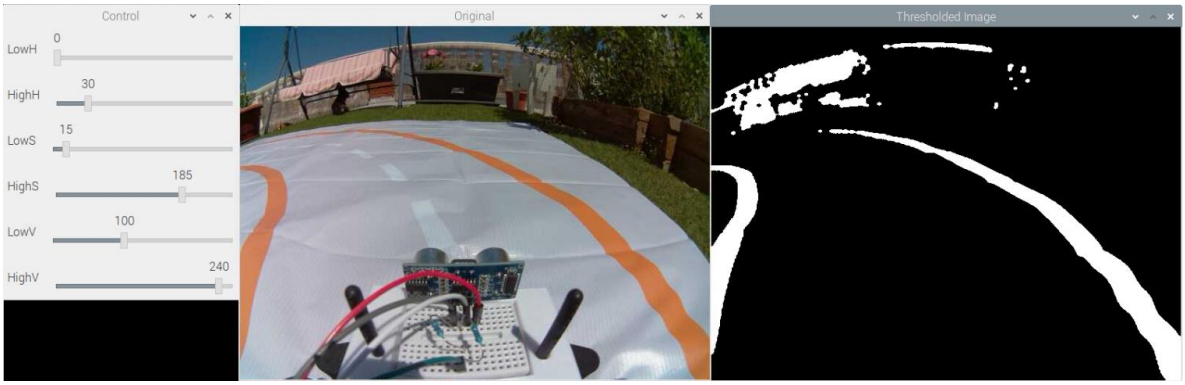


Figura 25. Proceso de obtención de los parámetros para la binarización. Donde se puede apreciar de izquierda a derecha: el panel de control, la imagen original y la imagen binarizada.

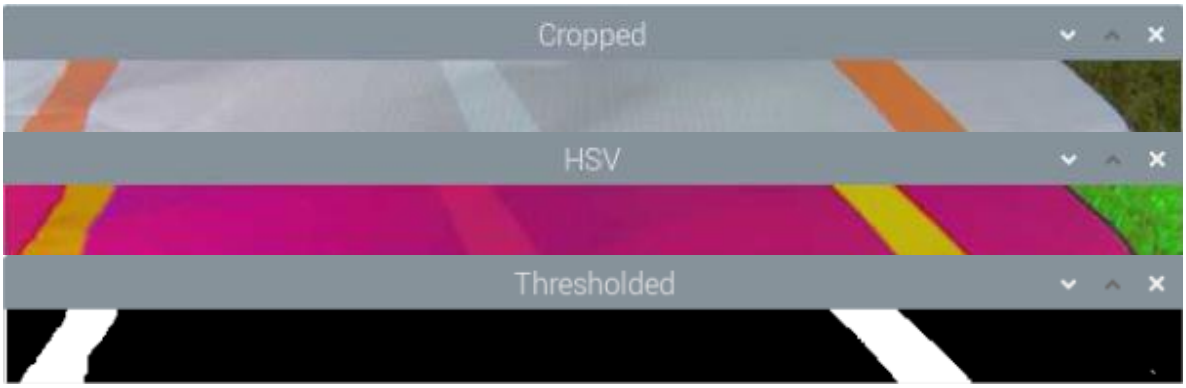


Figura 26. Resultado del proceso de binarización de la zona recortada. Donde se puede apreciar de arriba a abajo: el recorte original, el recorte convertido a espacio de color HSV y, por último, el recorte binarizado.

Una vez hecho esto, calculamos los contornos de las formas obtenidas y para cada contorno, obtenemos su momento central y centroide. Además, se reduce la presencia de ruido, filtrando los contornos detectados por el tamaño de su área, eliminando así los pequeños objetos que puedan ser detectados.

Después, se calcula independientemente la posición media de los objetos detectados a la derecha y los detectados a la izquierda. Esto se realiza ya que muchas veces por problemas de iluminación como reflejos o sombras, en el proceso de binarización se obtiene la forma completa dividida en dos o más partes.

Una vez hecho esto, se dibuja un punto en la imagen en las posiciones medias detectadas, representando así el centro de las líneas del circuito. Además, también se trazan dos líneas rectas para mejorar la visualización del resultado. Una de estas líneas, representará una línea tangente a la dirección recta del vehículo (azul); y la otra, mostrará la desviación desde el punto medio de la parte delantera del vehículo hasta el punto medio de la carretera (verde).

Figura 27. Resultado del hilo “Centros”, donde sobre la imagen tomada, se ha dibujado un punto en el centro de cada línea de la carretera detectada. Además, se ha trazado una línea azul que marca la trayectoria recta del vehículo y una línea verde que marca la desviación en la trayectoria.

Posteriormente, calculamos el ángulo de la línea verde, que marca la desviación en la trayectoria; y según este ángulo, indicaremos un resultado u otro para los modos de conducción auxiliar y conducción autónoma. El resultado para ambos modos de funcionamiento puede ser diferente ya que para la conducción auxiliar se otorga un mayor margen en la “posición correcta del vehículo” evitando una corrección excesiva, permitiendo así que el usuario sea el que controle el vehículo a no ser que la desviación en la trayectoria sea muy grande.

Además, en ambos modos de funcionamiento, si no se detecta ninguna línea de la carretera, el vehículo dará marcha atrás hasta que pueda volver a realizar un funcionamiento normal.

Por último, se almacenan estos resultados como variables globales y se activa la primera variable (“condcontrol1”) que pausa el hilo “Control” cuando se activa la conducción autónoma o la conducción asistida.

Este proceso se repite en bucle desde que se comprueba si la variable “condcentros” está activada hasta que se activa la variable “condcontrol1”.

6.3.4 DetectarSTOP

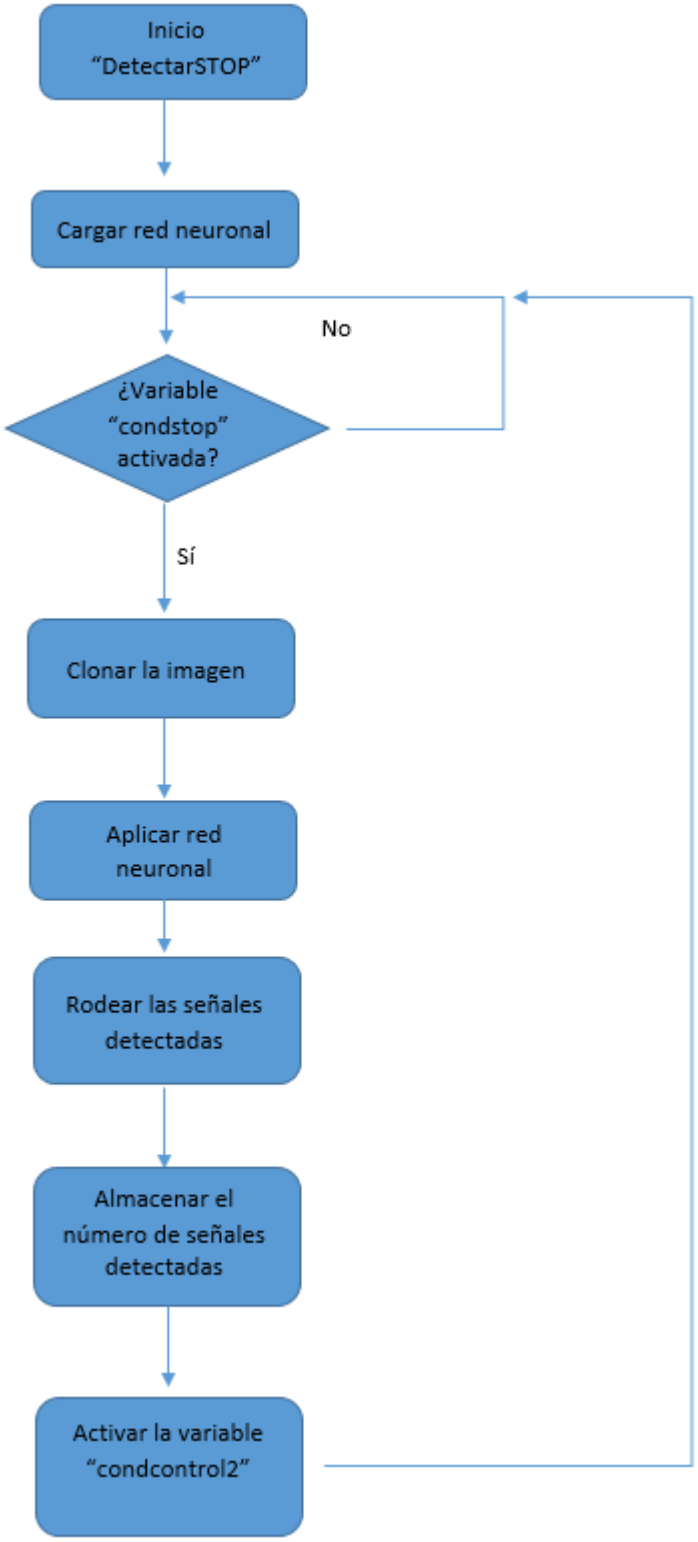


Figura 28. Flujograma del hilo "DetectarSTOP".

Este hilo será el encargado de aplicar una red neuronal para detectar las señales de “stop”. Primero de todo, se carga la red neuronal entrenada anteriormente (explicado en el apartado 7.1 Red Neuronal). A continuación, al igual que los dos hilos anteriores, espera la activación de una variable de tipo “condition” que lo pausa, en este caso “condstop”. Una vez se activa, clona la imagen que ha sido almacenada como variable global en el primer hilo.

Posteriormente convierte esta imagen clonada en escala de grises y aplica un ecualizador de histograma para mejorar la calidad de la imagen obtenida.

Figura 29. Imagen clonada de la imagen almacenada en hilo “ObtenerImagen” (figura 20) y convertida a escala de grises.

Figura 30. Imagen anterior (figura 29) tras aplicar un ecualizador de histograma que mejora el contraste.

Una vez hecho esto, se aplica la detección de objetos con la red neuronal que hemos entrado previamente. A continuación, se rodea en la imagen dónde se han detectado los objetos identificados, en este caso, señales de “stop”.

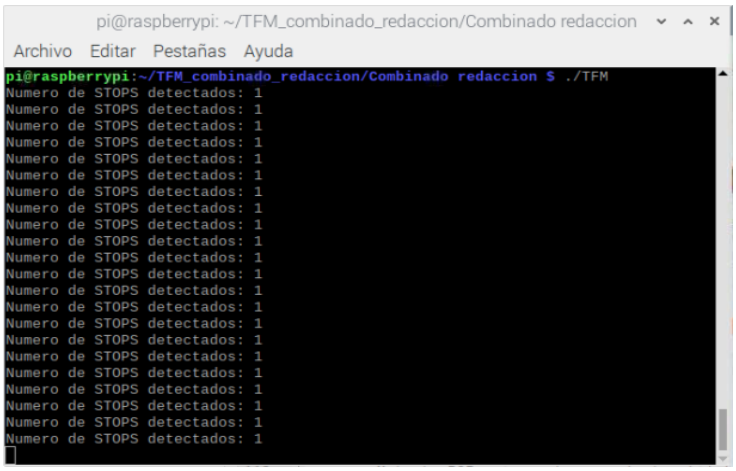


Figura 31. Resultado del hilo “DectectarSTOP” expresado por el terminal.

Figura 32. Resultado del hilo “DetectarSTOP” representado visualmente. Donde se puede apreciar en la imagen clonada, la señal de “stop” que se ha detectado.

Finalmente se almacena el número de señales detectadas como una variable global y se activa la segunda variable (“condcontrol2”) que pausa el hilo “Control” cuando se activa la conducción autónoma o la conducción asistida.

Este proceso se repite en bucle desde que se comprueba si la variable “condstop” está activada hasta que se activa la variable “condcontrol2”.

6.3.5 MedirDistancia

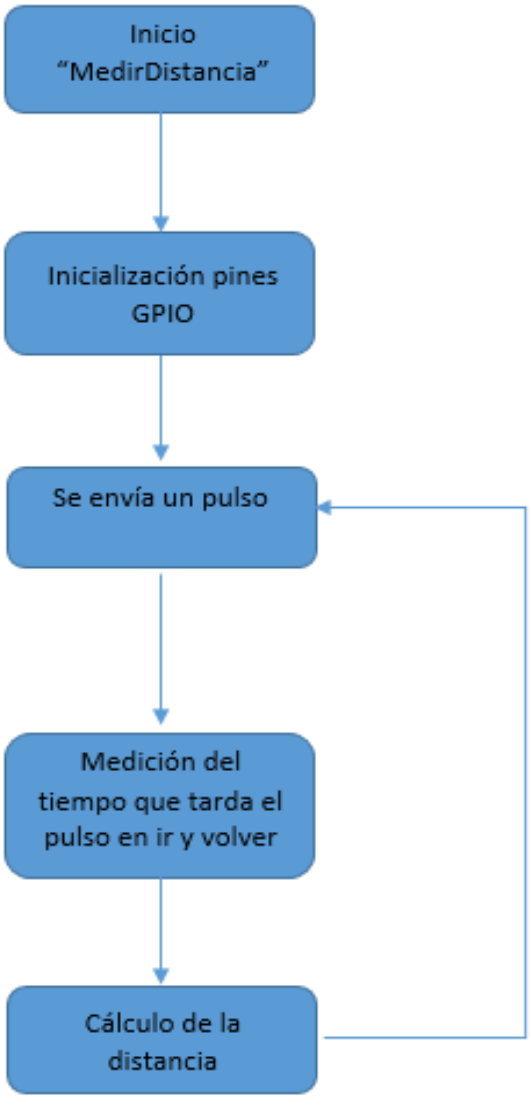


Figura 33. Flujograma del hilo "MedirDistancia".

Por otro lado, se encuentra el hilo “MedirDistancia” que obtiene la distancia libre en la parte delantera del vehículo a través del sensor. Para ello, primero se inicializan los pines GPIO. Después, se envía un pulso de activación del “trigger” al sensor de ultrasonidos de 10 microsegundos. Con esta activación, el sensor emite un tren de 8 pulsos a 40 kHz y en el momento en el que envía el último pulso, envía un “1” lógico al pin “echo”. Este “1” lógico se mantendrá hasta que el último pulso regrese al sensor. De esta forma, el tiempo en el que se mantenga este valor en “1” lógico será el mismo tiempo que tarda este último pulso en ir y volver al rebotar contra un objeto. A partir de este tiempo, se calcula la distancia teniendo en cuenta la velocidad del sonido (343 m/s); y que el recorrido es de ida y vuelta, a partir de la siguiente fórmula:

$$distancia\ (cm) = \frac{tiempofinal - \ tiempoinicial}{1.000.000} \cdot \frac{343}{2} \cdot 100$$

Hecho esto, se almacena el resultado en una variable global y se repite este proceso en bucle desde el momento en el que se envía el pulso de activación del “trigger” hasta que se almacena como variable global la distancia medida.

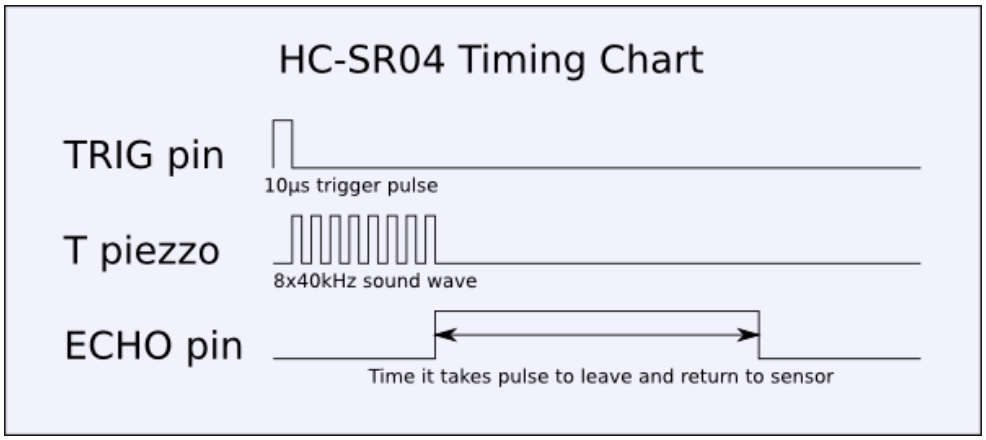


Figura 34. Tabla de tiempos del sensor de ultrasonidos HC-SR04.

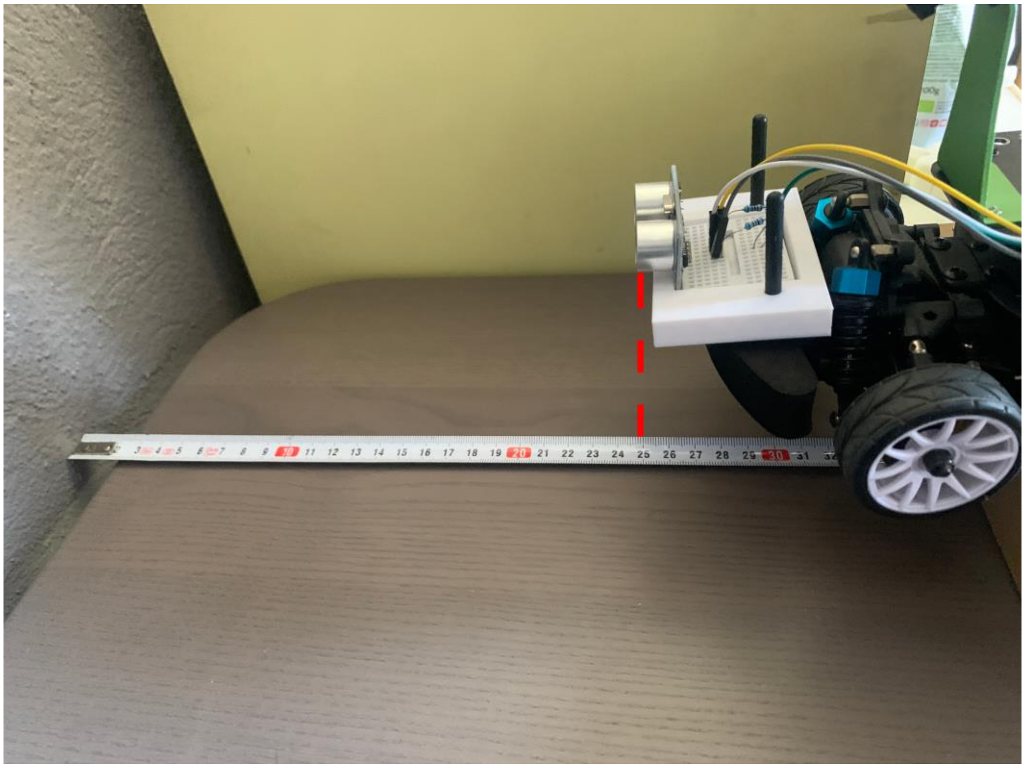


Figura 35. Montaje empleado para verificar el correcto funcionamiento del sistema creado para medir la distancia. Donde podemos apreciar que el sensor está aproximadamente a 25 cm de la pared.

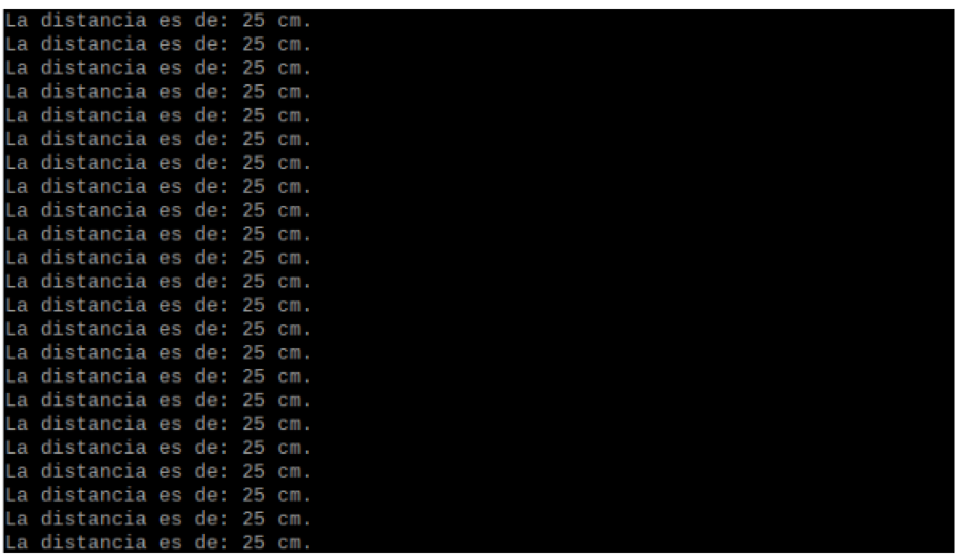


Figura 36. Resultado obtenido por el terminal, obteniendo la distancia teórica y verificando el correcto funcionamiento del sistema.

6.3.6 Mando

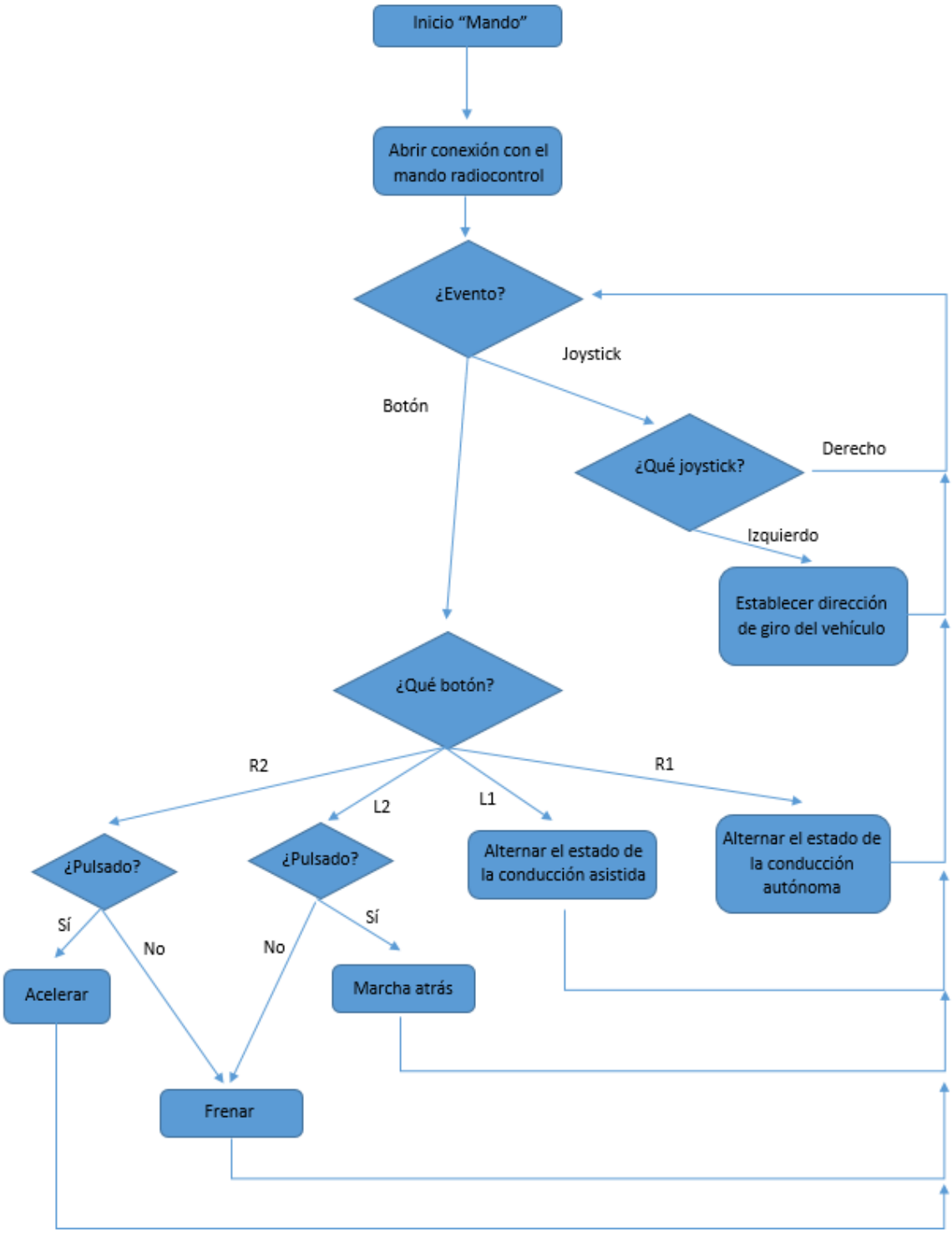


Figura 37. Flujograma del hilo "Mando".

Figura 38. Mando utilizado en el proyecto visto desde la parte frontal.

Figura 39. Mando utilizado en el proyecto visto desde la parte superior.

El hilo “Mando” es el encargado de comunicarnos todas las instrucciones recibidas a través del control remoto. Para ello, primero se abre la conexión con el dispositivo remoto en modo lectura. Posteriormente se lee el evento recibido desde el control remoto y lo clasificará según si la acción se ha dado en un botón o en un “joystick”. En caso de ser un botón podemos identificar cuál es y si la acción ha sido presionar el botón o soltarlo. En caso de ser un “joystick” podremos saber el desplazamiento en x e y que se ha dado. De esta forma, lograremos el siguiente funcionamiento:

Botón	Función presionado	Función soltado
R2	Acelerar	Frenar
L2	Marcha atrás	Frenar
R1	Alternar activación de conducción autónoma	-
L1	Alternar activación de conducción auxiliar	-

Tabla 1. Funcionamiento planteado a partir de las acciones llevadas a cabo en los botones.

“Joystick”	Función
Izquierdo	Desplazamiento en eje “x” proporcional al giro del vehículo

Tabla 2. Funcionamiento planteado a partir de las acciones llevadas a cabo en los “joystick”.

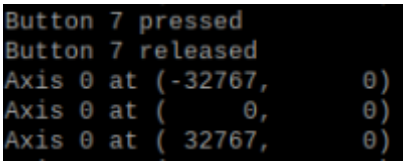


Figura 40. Ejemplo de la información que podemos leer a través del mando. Donde las primeras dos líneas pertenecen a la pulsación y liberación del botón “R1”, que tiene asignado el identificador “7”. Y las ultimas 3 líneas, pertenecen al joystick izquierdo respectivamente desplazado completamente a la izquierda, en la posición de reposo y desplazado completamente a la derecha.

Finalmente, almacenamos como variables globales; por un lado, el estado actual de la activación de la conducción auxiliar y la conducción autónoma. Y, por otro lado, el comportamiento que debe tener el vehículo según la información recibida por el mando.

Este proceso se repite en bucle desde que se lee el evento recibido por el mando hasta que se almacenan las variables globales.

6.3.7 Control

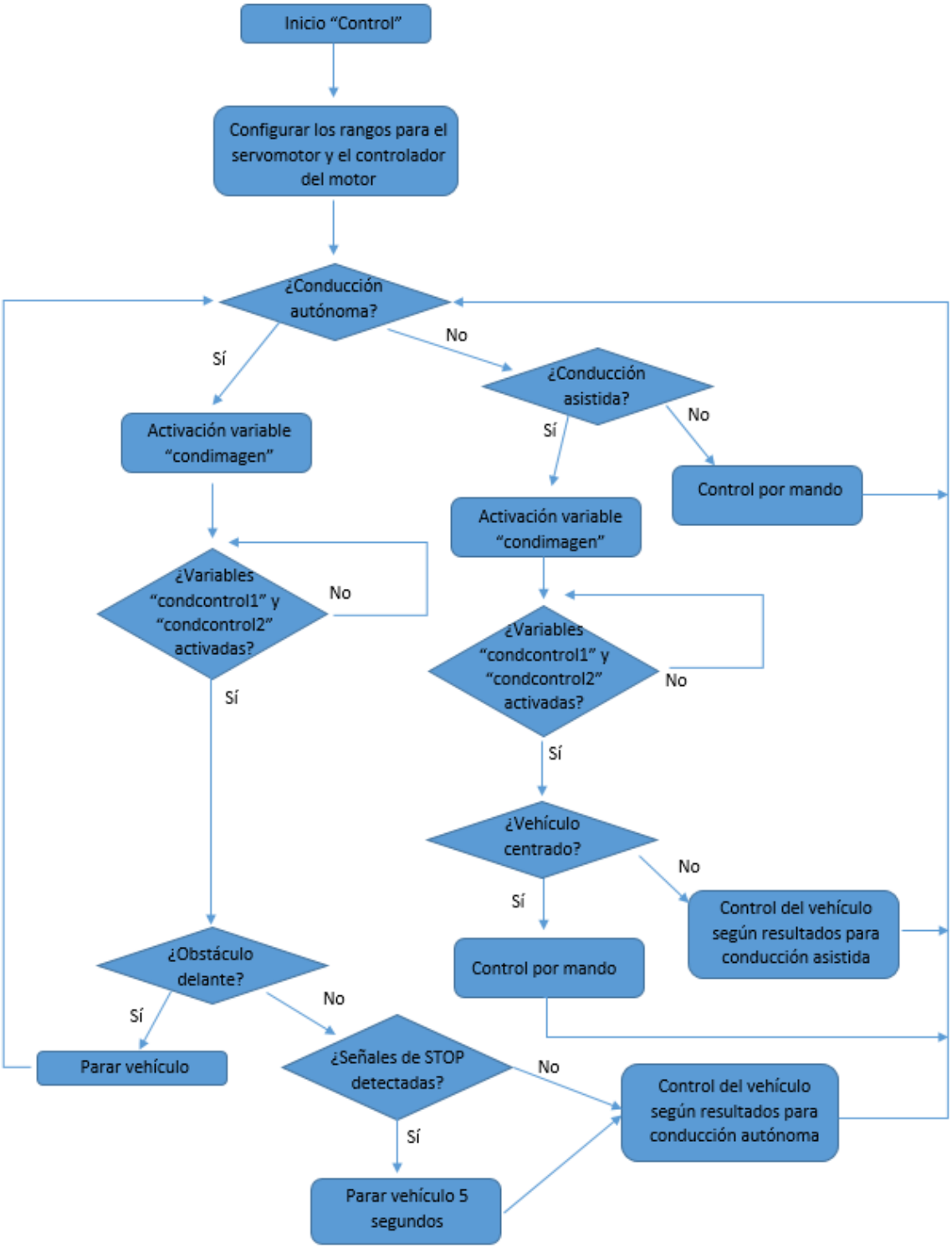


Figura 41. Flujograma del hilo "Control".

El último hilo que encontramos es “Control”, este será el encargado de procesar toda la información y hacer que el vehículo actúe en consecuencia. Para ello, toma dentro de varios “mutex” todas las variables globales obtenidas de los resultados de hilos anteriores. Posteriormente comprueba si el control autónomo está activado o no. En caso afirmativo, primero envía la señal que activa el hilo “ObtenerImagen”, ya que para la conducción autónoma se necesita el tratamiento de imagen para conocer la posición del vehículo en la carretera y la presencia de señales de “stop”. Posteriormente se pausa el hilo hasta que se activen las variables de tipo condition “condcontrol1” y “condcontrol2”, activadas por los hilos “Centros” y “DetectarSTOP”.

Una vez hecho esto, comprobamos que la distancia delante del vehículo no sea menor de 20 cm. En caso de que la distancia sí sea menor a este valor, frenamos el vehículo; y en caso de que sea mayor seguiremos con el siguiente proceso. Comprobaremos si se ha detectado alguna señal de “stop”, en caso afirmativo se frenará el coche 5 segundos e iniciaremos un contador que nos permitirá pasar la señal de “stop” sin que vuelva a ser detectada. Este contador es necesario ya que, en caso de no estar, el coche realiza una parada de 5 segundos y al volver a analizar la imagen, volvería a detectar la misma señal de “stop”, por lo que entraría en un bucle infinito. Después de esperar los 5 segundos o en el caso de que no se haya detectado ninguna señal de “stop” y la distancia delante sea mayor a los 20 cm, se envía al vehículo la velocidad y dirección correspondiente según el resultado que se ha obtenido en el hilo “Centros”.

En el caso de que la conducción autónoma no esté activada, comprobaremos si la conducción auxiliar lo está. En caso negativo se procederá a realizar el control del vehículo a partir del resultado obtenido en el hilo “Mando”. En caso afirmativo, enviaremos la condición que activa el hilo “ObtenerImagen”, ya que al igual que en el caso de la conducción autónoma, necesitamos el procesamiento de la imagen para conocer la posición del vehículo en el carril. Una vez hecho esto, al igual que en la conducción autónoma, se pausa el hilo hasta que se activan las “condcontrol1” y “condcontrol2”. Esto nos indicará que ya conocemos la posición del vehículo en la carretera. A partir de esta, si el vehículo está centrado dentro del margen establecido, permitiremos el control libre por parte del usuario a través del control remoto. En caso de que se salga del margen corregiremos su dirección hasta que el vehículo vuelva a posicionarse dentro del margen.

Finalmente se imprimirá por pantalla la información relevante obtenida en los hilos anteriores, como, por ejemplo: el número de señales de “stop” detectadas, el ángulo en la línea trazada que marca la desviación en la trayectoria, la distancia delante del vehículo o el estado de la activación de la conducción autónoma y la conducción auxiliar.

Este proceso se repite en bucle desde que se comprueba si la conducción autónoma está activada hasta que se imprime por pantalla la información de interés.

6.4 IMPLEMENTACIÓN

La implementación de este comportamiento se ha llevado a cabo con la ayuda de las siguientes librerías:

6.4.1 Joystick

Con la ayuda de la librería “joystick” de Linux y las funciones creadas por Jason White en el código (https://gist.github.com/jasonwhite/c5b2048c15993d285130) podremos realizar todas las funciones necesarias con el mando de control remoto. Para ello primero abrimos la conexión con el dispositivo al que nos queremos conectar.

```
dispositivo= "/dev/input/js0";
js= open(dispositivo, Modo);
```

En este caso, el dispositivo en cuestión es un mando, el cual está representado por el archivo en la dirección “/dev/input/js0”. La dirección exacta del archivo estará contenida en una variable como cadena de carácteres y será utilizado en la función “open”, al cual le pasaremos esta dirección y “Modo” que indica el modo de apertura. Además, “js” es una variable de tipo “int” y en caso de no poder abrir la conexión con el mando el mando nos devolverá un “-1”.

Nomenclatura	Significado
O_RDONLY	Sólo lectura
O_WRONLY	Sólo escritura
O_RDWR	Lectura y escritura

Tabla 3. modos de apertura en la función “open”.

Una vez abierto el mando, se pueden leer los eventos detectados con la función:

read_event(js, &event)

Al cual se le introduce como entradas: el valor entero que hemos utilizado para abrir la conexión con el mando (en este caso “js”) y una variable con estructura “js_event” (estructura predefinida en la librería “joystick” de Linux). Además, esta función nos devuelve “0” si se ha podido leer el evento correctamente y “-1” si no se ha podido hacer.

Una vez leído el evento, lo podemos clasificar por el tipo de evento con la variable “event.type”. El cual puede tomar los siguientes valores:

Valor de “event.type”	Evento leído
JS_EVENT_BUTTON	Pulsación o liberación de un botón
JS_EVENT_AXIS	Desplazamiento de un “joystick”

Tabla 4. Valores que puede tomar la variable “event.type”.

Además, en caso de haber leído el evento en un botón se podrá identificar de qué botón se trata según el valor que toma la variable “event.number” como se muestra en la siguiente tabla.

Valor de “event.number”	Botón pulsado
0	A
1	B
3	X
4	Y
6	L1
7	R1
8	L2
9	R2
10	Select
11	Start
13	“Joystick” izquierdo
14	“Joystick” derecho

Tabla 5. Valores que puede tomar la variable “event.number” y el botón al que hace referencia.

También, cuando el evento se refiere a un botón, podemos diferenciar si la acción se trata de pulsar el botón o de soltarlo. Este se realiza gracias al valor que toma la variable “event.value”. Siendo “1” cuando se detecta la pulsación del botón y “0” cuando se suelta.

Por otro lado, cuando se detecta un evento de desplazamiento en algún “joystick”, podemos identificar de qué “joystick” se trata y cuánto se ha desplazado en cada eje. Para ello se utiliza la siguiente función:

get_axis_state(&event, axes);

Donde “event” es el evento leído en la función “read_event” y “axes” es una variable con estructura “axis_state” donde se almacenarán las posiciones de ambos “joysticks”. A partir de la función anterior, podemos obtener los valores para los diferentes ejes en ambos “joysticks” con las siguientes variables:

Variable	Significado
axes[0].x	Eje “x” del “joystick” izquierdo
axes[0].y	Eje “y” del “joystick” izquierdo
axes[1].x	Eje “x” del “joystick” derecho
axes[1].y	Eje “y” del “joystick” derecho

Tabla 6. Lista de variables donde se almacenan la posición en los ejes “x” e “y” de ambos “joysticks”.

Por último, se puede cerrar la conexión con el mando con la función:

close(js);

Donde “js” es la misma variable que hemos utilizado para abrir la conexión.

6.4.2 I2C

Esta librería proporcionada por el tutor nos facilita la utilización del protocolo de comunicación serial I2C (“inter integrated circuits”). Para utilizarlo, definiremos una clase utilizando la siguiente función:

BBB::I2C m_i2c(X);

Donde m_i2c será la variable que se va a declarar con clase I2C y la “X” indicará el bus al que nos queremos conectar.

6.4.3 PCA9685_PWM

Esta librería, también proporcionada por el tutor, nos permite definir una variable de tipo PCA9685, que es el controlador para servomotores que utiliza el kit empleado. Esto lo realizaremos de la siguiente forma:

BBB::PCA9685 PWMController(&m_i2c, Dirección);

Donde PWMController es la variable que hemos definido como el controlador, m_i2c es la variable de clase “I2C” creada anteriormente y “Dirección” es la dirección I2C a la que nos queremos conectar, en nuestro caso la dirección es “0x40”.

6.4.4 Servo_PCA

Esta librería también ha sido proporcionada por el profesorado; y nos permite, declarar los servomotores dentro del controlador PCA9685. Para ello utilizamos la siguiente función:

BBB::Servo_PCA NombreServo(&PWMController,X);

Donde NombreServo es el nombre de la variable que asignamos al servomotor con clase “Servo_PCA”, PWMController es el controlador PCA9685 que hemos definido en el apartado anterior y “X” será el servomotor que queremos controlar. En nuestro caso “1” es el servomotor que controla la dirección y “2” es el controlador para el motor de tracción. Por otro lado, deberemos establecer el rango de trabajo que será diferente para cada servomotor. para ello utilizaremos la siguiente función:

NombreServo.setRange(X,Y);

Donde “NombreServo” es la variable asignada en la función anterior al servomotor que queremos controlar, “X” es el valor mínimo del rango e “Y” es el valor máximo del mismo. Estos valores de “X” e “Y” se obtienen experimentalmente para cada servomotor. Una vez hecho esto, podemos enviar valores al servomotor con la siguiente función:

NombreServo.setPosition(Valor);

Donde “NombreServo” sigue siendo la variable asignada al servomotor como en los casos anteriores y “Valor” es un valor numérico entre 0 y 1. Donde el valor “0” corresponde al valor mínimo del rango y va aumentando proporcionalmente hasta llegar al valor “1”, que se corresponde con el valor máximo del rango.

6.4.5 Pthread

Esta librería de hilos “POSIX” (“Portable Operating System Interface for Unix”) para Linux nos facilita la creación de aplicaciones utilizando una estructura con hilos (tomado de [21]). Para ellos primero declararemos las variables de tipo thread de la siguiente forma:

pthread_t NombreVariable;
Una vez tenemos la variable definida, podemos crear el hilo con la función:

pthread_create (&NombreVariable, Atributos, Ejecutar, Argumentos);

Donde “NombreVariable” es el nombre de la variable asignada en la función anterior, “Atributos” se puede utilizar si queremos definir algunos atributos para el hilo. En caso de definirlos se debe utilizar una estructura “pthread_attr_t” o también se puede utilizar “NULL” para utilizar los atributos por defecto. “Ejecutar” es la función que queremos iniciar con el hilo y “Argumentos” se puede utilizar si queremos especificar algún argumento para la ejecución del hilo, en caso contrario

utilizaremos “NULL” en su lugar. Una vez creado el hilo, se puede utilizar la siguiente función para esperar a que el hilo termine.

pthread_join (NombreVariable, Cancelado);

Donde “NombreVariable” es la variable que hemos utilizado para crear el hilo, y “Cancelado” es una variable que nos devuelve “PTHREAD_CANCELED” si se cancela el hilo en cuestión. Si no queremos obtener el valor que nos devuelve se puede utilizar “NULL”.

Por otro lado, para evitar problemas durante la lectura y/o escritura de variables globales, vamos a utilizar los mutex (mecanismos de exclusión mútua). Estos se pueden bloquear y desbloquear para controlar cuando una variable está accesible para ser leída o modificada. Para crear e inicializar un mutex con los atributos por defecto podemos hacerlo de la siguiente manera:

pthread_mutex_t NombreMutex= PTHREAD_MUTEX_INITIALIZER;

Una vez creado e iniciado el mutex, podremos utilizar estas dos funciones:

pthread_mutex_lock(&NombreMutex);

Esta función nos permite bloquear el mutex “NombreMutex”.

pthread_mutex_unlock(&NombreMutex);

Ésta en cambio, nos permite desbloquear el mutex “NombreMutex”.

Por último, si se necesita garantizar la sincronización entre ciertos hilos, se pueden utilizar las variables de tipo “condition”. Este tipo de variables permiten pausar un hilo hasta que se llame a su activación. Primero de todo se pueden crear e iniciar con la siguiente función:

pthread_cond_t NombreCondición= PTHREAD_COND_INITIALIZER;

Una vez creado e iniciada la variable, podremos utilizar las siguientes funciones:

pthread_cond_wait (&NombreCondición, &NombreMutex);

Esta función nos permite pausar un hilo hasta que se active la condición “NombreCondición”. Además, debe ser utilizado dentro de un mutex con nombre “NombreMutex”.

pthread_cond_signal (&NombreCondición);

Esta función nos permite activar la variable “NombreCondición”.

6.4.6 OpenCV2

Esta librería nos va a permitir utilizar todas las funciones necesarias relacionadas con el trabajo de imágenes (tomado de [22]). Para ello, utilizamos la siguiente función para establecer la comunicación con la cámara desde la que queremos capturar las imágenes.

```
VideoCapture cap(CV_CAP_ANY);
```

Donde "cap" es una variable que declaramos de clase "VideoCapture" con "CV_CAP_ANY" como argumento. Éste último parámetro nos permite la detección automática de la cámara detectada. Posteriormente se puede realizar una configuración básica con las siguientes funciones.

```
cap.set(CV_CAP_PROP_BUFFERSIZE, X);
```

Esta función nos permite establecer el tamaño de "buffer" por software. Donde "X" indica el número de imágenes que queremos almacenar.

```
cap.set (CV_CAP_PROP_FRAME_WIDTH, X);
cap.set (CV_CAP_PROP_FRAME_HEIGHT, Y);
```

Estas funciones nos permiten establecer la altura y la anchura del "frame" que vamos a tomar. También podemos comprobar si la comunicación con la cámara se ha abierto con la función:

```
cap.isOpened()
```

Que nos devuelve "0" si ha habido algún error o "1" si se ha abierto correctamente. Una vez abierta la comunicación podemos tomar una imagen con la siguiente función:

```
cap.read(NombreImagen);
```

Donde "NombreImagen" es una variable de tipo "Mat". En nuestro caso, por cuestión de comodidad del cableado; la cámara está situada al revés, por lo que se puede rotar la imagen obtenida con la siguiente función:

```
cv:rotate(NombreImagen, NombreImagen, cv::ROTATE_X);
```

Donde "X" indica la cantidad de grados con los que se quiere realizar el giro. Además, para copiar una imagen de una variable a otra, se puede utilizar la siguiente función:

```
imagenclonada = imagenroriginal.clone();
```

Por otro lado, se puede realizar un recorte de la siguiente manera. Primero de todo, se declara la zona que se desea recortar.

cv::Rect ZonaCorte(OrigenX,OrigenY,Ancho,Alto);

Ahora se aplica el recorte de la región deseada de la siguiente manera:

imagenrecortada= imagenoriginal(ZonaCorte);

Por otro lado, la imagen tomada desde la función “cap.read” está en un espacio de color BGR (“Blue Green Red”), lo que indica que es una imagen a color. Se puede cambiar entre diferentes espacios de color con la siguiente función

cvtColor(ImagenOriginal, ImagenCambiada, ModoCambio);

Donde “ModoCambio” indica el cambio en el espacio de color que se desea realizar. Siendo algunos de los más conocidos:

- COLOR_BGR2GRAY: de “BGR” a escala de grises.
- COLOR_BGR2HSV: De “BGR” a “HSV”(“Hue Saturation Value”).
- COLOR_BGR2HLS: De “BGR” a “HLS” (“Hue Saturation Lightness”).

En caso de convertir una imagen de “BGR” a escala de grises, se le puede aplicar un ecualizador de histograma con la siguiente función.

equalizeHist(Original, Resultante);

Por otro lado, podemos binarizar una imagen con la siguiente función:

inRange(ImagenHSV, Scalar(Hmin, Smin, Vmin), Scalar(Hmax, Smax, Vmax), ImagenBinarizada);

Donde “ImagenHSV” será una imagen en el espacio de color “HSV”, “Scalar” es un vector que puede contener hasta 4 elementos. En este caso, primero se declara un vector de tipo “Scalar” que contiene los valores mínimos de “hue” (matiz) “saturation” (saturación) y “value” (valor); y posteriormente otro con los valores máximos de los mismos parámetros.

Para mostrar la imagen por pantalla podemos utilizar la siguiente función:

imshow("Titulo", Imagen);

Donde “Título” es el título que queremos poner a la ventana donde se va a mostrar la imagen y “Imagen” es la variable de tipo “Mat” que queremos mostrar por pantalla. Además, debemos poner un “waitKey” para que se muestre la imagen.

waitKey(ms);

Donde “ms” es el tiempo mínimo en milisegundos durante el que se mostrará la imagen

Por otro lado, se pueden detectar los contornos de los objetos (tomado de [23]) con la función “findContours”, que funciona de la siguiente manera:

findContours(ImagenBinarizada, contours, hierarchy, Mode, Method);

Donde “contours” es un vector de vectores de variables de tipo “Point”, “hierarchy” es un vector de estructuras de tipo “Vec4i”, “Mode” indica el modo de detección de contornos y “Method” indica el método para detectarlos. En nuestro caso utilizamos el modo “RETR_TREE” y el método “CHAIN_APPROX_SIMPLE). Una vez tenemos los contornos detectados, podemos calcular los momentos centrales para cada uno de ellos. Para ello utilizaremos la siguiente función:

mu[h] = moments(contours[h]);

Donde “mu” es un vector de clase “moments” con el mismo tamaño que el vector “contours” y “[h]” indica la posición en el vector. Ahora se pueden calcular los centroides de los momentos centrales obtenidos anteriormente con la siguiente función:

mc[h] = Point2f(mu[h].m10/mu[h].m00 , mu[h].m01/mu[h].m00);

Donde “mc” es un vector de clase “Point2f” con el mismo tamaño que el vector “contours”, “[h]” indica la posición en el vector, “Point2f” nos permite expresar un punto con coordenadas “x” e “y” como “float” y “mu[h]” indica el momento central con la posición “h” en el vector. Por otro lado, “m10” indica el momento espacial calculado utilizando “1” y “0” como valores para “j” e “i”. “m00” y “m01” realizan la misma función, pero calculados respectivamente con los valores “0”,”0” y “0”,”1”. Una vez obtenidos los centroides, podemos expresar su posición de la siguiente manera:

mc[h].eje

Donde “mc[h]” es el momento central con la posición “h” en el vector y “eje” puede tomar el valor de “x” o “y” según el eje en el que queramos saber la posición.

También, se puede calcular el área de los contornos obtenidos a partir de la función “findContours”, Para ello utilizamos la función “contourArea” de la siguiente manera:

area = contourArea(contours[h]);

Se pueden dibujar formas sobre las imágenes; y para ello, podemos usar funciones como las siguientes:

circle(Imagen,Centro, Radio, color, grosor);

Esta función nos permite dibujar circunferencias en una imagen. También se puede dibujar elipses con la siguiente función:

ellipse(Imagen, Centro, TamañoRadios, Ángulo, ÁnguloInicial, ÁnguloFinal, Color, Grosor);

También, se pueden dibujar puntos:

Point center(stops[h].x + stops[h].width/2, stops[h].y + stops[h].height/2);

Y también se pueden dibujar líneas:

line(Imagen, Punto1, Punto2, Color, Grosor, TipoLinea);

Por otro lado, para realizar la detección de objetos utilizando un clasificador en cascada, OpenCV ofrece la posibilidad de realizarlo mediante “Haar-cascade”. Para ello cargamos la red neuronal previamente entrenada, con la siguiente función.

stop_cascade.load(NombreRedNeuronal)

Donde “stop_cascade” es una variable de clase “CascadeClassifier”. Una vez cargada la red neuronal, se puede utilizar para detectar objetos con la siguiente función:

stop_cascade.detectMultiScale(Imagen, Detectados, FactorEscala, VecinosMín);

Una vez detectados los objetos en la imagen, podemos utilizar las siguientes variables:

Detectados.size();

Esta variable nos permite identificar cuántos objetos se han detectado.

Detectados[h].x
Detectados[h].y
Detectados[h].width
Detectados[h].height

Estas variables nos permiten identificar para el objeto número “h” detectado, su posición en el eje “x”, en el eje “y”, su anchura y su altura.

6.4.7 WiringPi

Esta librería permite la utilización de los pines de entradas y salidas GPIO de la Raspberry Pi 4B. Para ello, se dispone de las siguientes funciones:

wiringPiSetupGpio();

Esta función permite inicializar la librería. Una vez inicializada, podemos definir los pines de entradas y salidas de la siguiente manera:

pinMode(NumPin, Modo);

Donde “Modo”, puede tomar el valor de “OUTPUT” para pines de salida o “INPUT” para pines de entrada. Para los pines de salida, podemos establecer su valor con la función:

digitalWrite(NumPin, Valor);

Donde “Valor” puede ser “HIGH” para un valor alto o “LOW” para un valor bajo. Por otro lado, también podemos leer los valores de los pines de entrada con la siguiente función:

digitalRead(NumPin)

Que devuelve “HIGH” o “LOW” según el valor del pin.

6.5 DISEÑO ELECTRONICO

Para la detección de objetos delante del vehículo se utilizará un sensor de ultrasonidos HC-SR04 conectado a la Raspberry Pi. Para el funcionamiento de este desarrollaremos el circuito que podemos encontrar en el “Plano 001”.

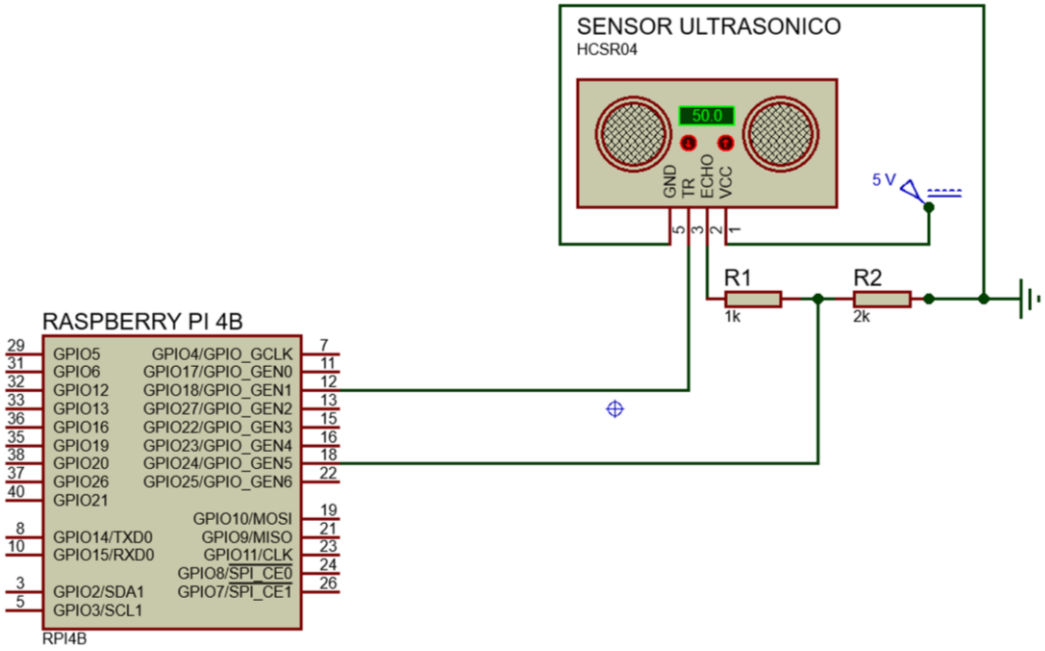


Figura 42. Recorte del “Plano 001” con el circuito electrónico diseñado para el sensor de distancia.

En este circuito se utilizan dos resistencias con el objetivo de realizar un divisor de tensión. Esto es debido a que la respuesta máxima que nos devuelve el sensor es de 5 V y las entradas GPIO de la Raspberry Pi toleran un máximo de 3.3 V (tomado de [24]). Para ello se ha diseñado el siguiente divisor de tensión:

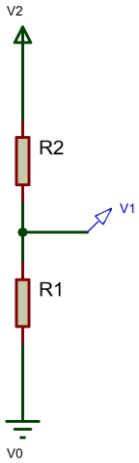


Figura 43. Representación gráfica de un divisor de tensión.

Donde:

$$V0 = 0\,V$$
$$V1 = 3.3\,V$$
$$V2 = 5\,V$$

Además, se sabe que:

$$V1 = R1 \cdot I$$

$$I = \frac{V2 - V0}{R1 + R2} = \frac{V2}{R1 + R2}$$

Combinando las dos últimas ecuaciones:

$$V1 = R1 \cdot \frac{V2}{R1 + R2}$$
$$\frac{V1}{V2} = \frac{R1}{R1 + R2}$$

Sustituyendo los valores:

$$\frac{3.3}{5} = \frac{R1}{R1 + R2}$$

Lo que equivale aproximadamente a:

$$\frac{2}{3} = \frac{R1}{R1 + R2}$$

De esta forma:

$$2 \cdot (R1 + R2) = 3 \cdot R1$$
$$R1 + R2 = \frac{3}{2} \cdot R1$$
$$R1 - \frac{3}{2} \cdot R1 = -R2$$
$$\frac{1}{2} \cdot R1 = R2$$
$$R1 = 2 \cdot R2$$

Así, sabemos que el valor de la resistencia R1 tiene que ser aproximadamente el doble que el de la resistencia R2.

7. PRUEBAS Y RESULTADOS EXPERIMENTALES

La selección de la solución adoptada se ha realizado experimentalmente, para ello se han desarrollado las alternativas que se han planteado y se ha seleccionado la que mejores resultados han ofrecido.

7.1 RED NEURONAL

Se disponen de dos métodos para llevar a cabo el entrenamiento de una red neuronal (tomado de [25] y [26]).

7.1.1 Entrenamiento red neuronal capturando las imágenes positivas

Para llevar a cabo el entrenamiento de una red neuronal, primero se tienen que crear dos bases de datos de imágenes. Una de ellas está formada por imágenes negativas, las cuales consisten en fondos de imagen que no contienen el objeto a detectar. Y la otra está formada por imágenes positivas, es decir, imágenes que sí que contienen el objeto a detectar.

Por un lado, las imágenes negativas se almacenan todas dentro de una carpeta. Y, por otro lado, se crea un archivo con extensión “txt” donde se incluyen las direcciones de estos archivos. Quedando de la siguiente manera:

Figura 44. Imagen negativa donde no se puede apreciar el objeto a detectar (señal de “stop”).

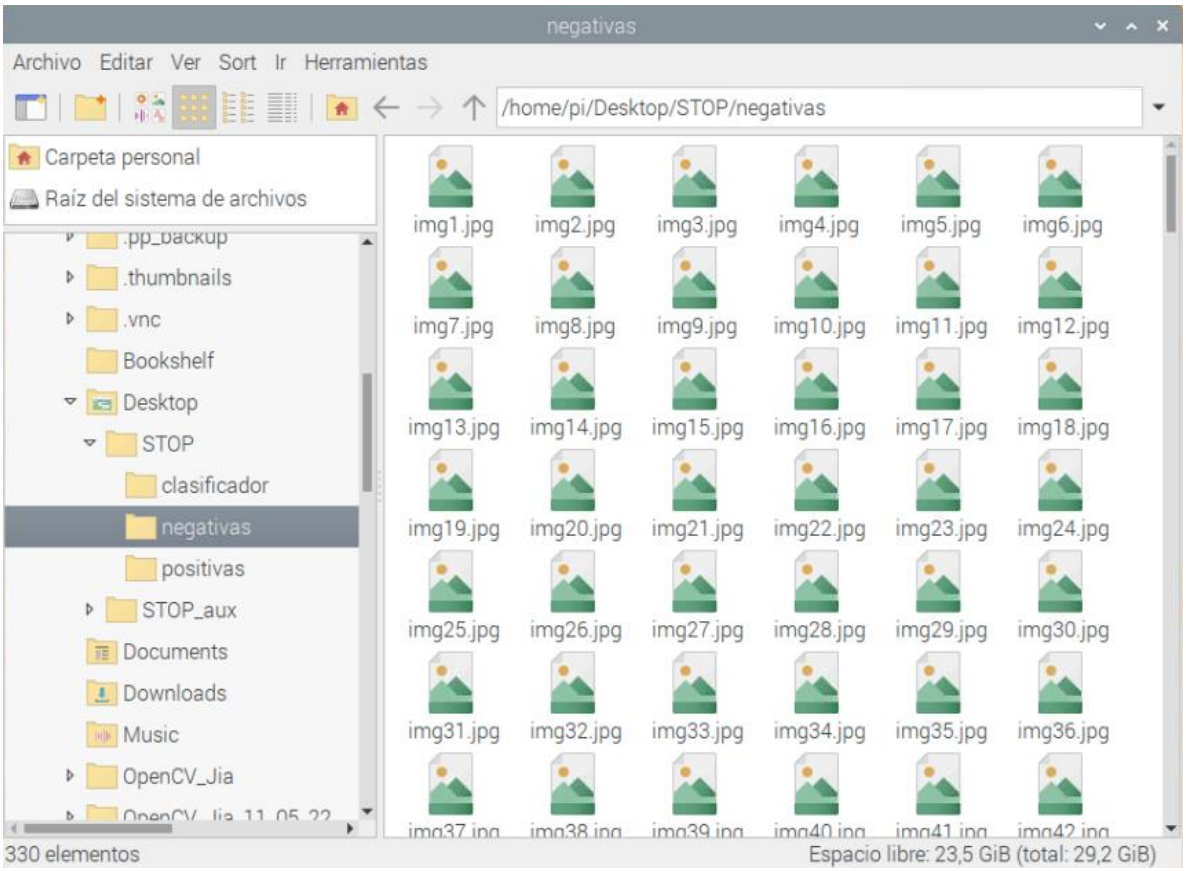


Figura 45. Base de datos de imágenes negativas.

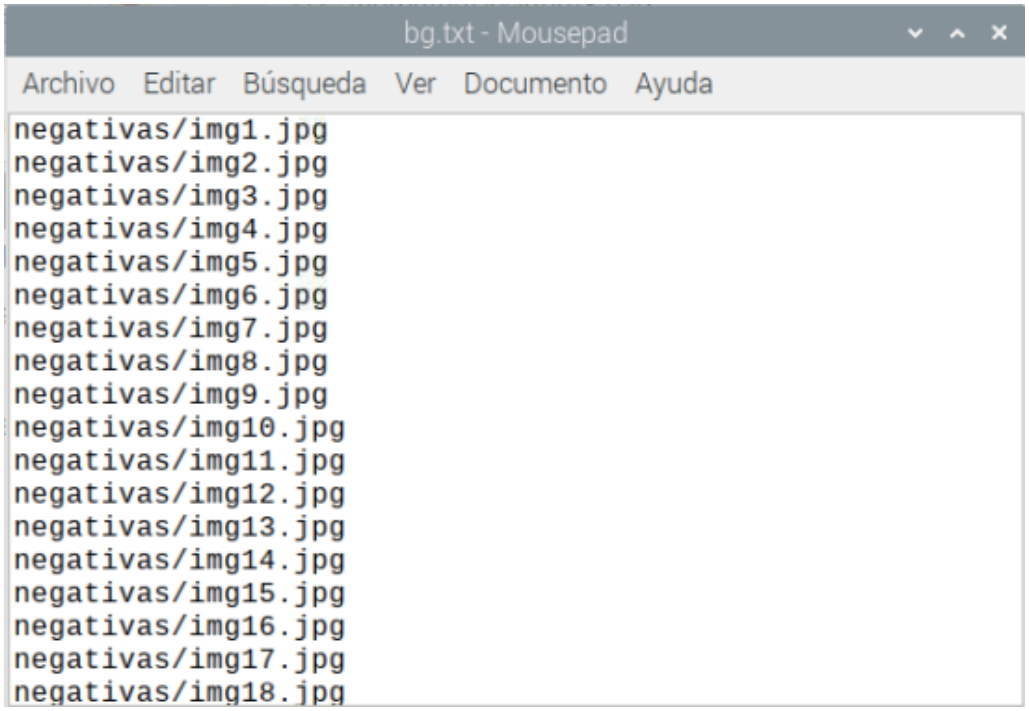


Figura 46. Archivo “txt” donde se indican las direcciones de todas las imágenes negativas.

Para realizar este proceso, primero necesitamos tomar las imágenes. Para ello, repetimos el siguiente comando donde únicamente tendremos que cambiar el nombre entre una imagen y otra:

raspistill -o Dirección/Nombre.jpg -w Ancho -h Alto

En esta función, se establece, por un lado, la dirección y el nombre de la imagen que queremos guardar; y, por otro lado, el ancho y alto de la imagen.

Una vez se tienen las imágenes negativas tomadas, se procede a realizar el mismo proceso, pero situando el objeto que se quiere detectar dentro de la imagen, para crear la base de datos de imágenes positivas.

Figura 47. Imagen positiva donde sí se puede apreciar el objeto a detectar (señal de “stop”).

Posteriormente, se aplica el comando “opencv_annotation” para seleccionar en la imagen dónde están los objetos a detectar y así, crear un archivo “txt” donde se indican el número de objetos detectados y las coordenadas de los mismos. Para ello se emplea la siguiente función:

opencv_annotation --annotations = DireccionTXT --images=DirecciónPositivas

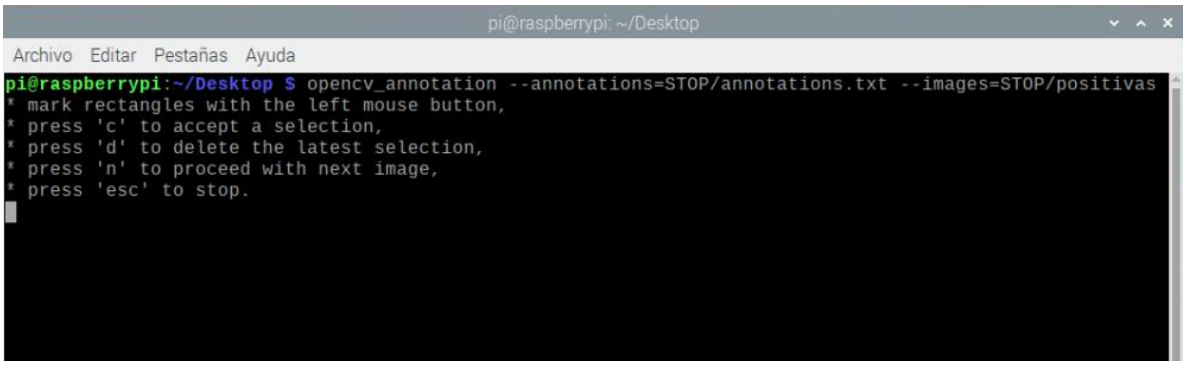


Figura 48. Terminal tras ejecutar el comando “opencv_annotation”, donde se indican las instrucciones para llevar a cabo el proceso de creación del archivo “txt”.

Figura 49. Proceso de la función anterior, donde se tiene que marcar dónde están los objetos a detectar.

Quedando así el directorio:

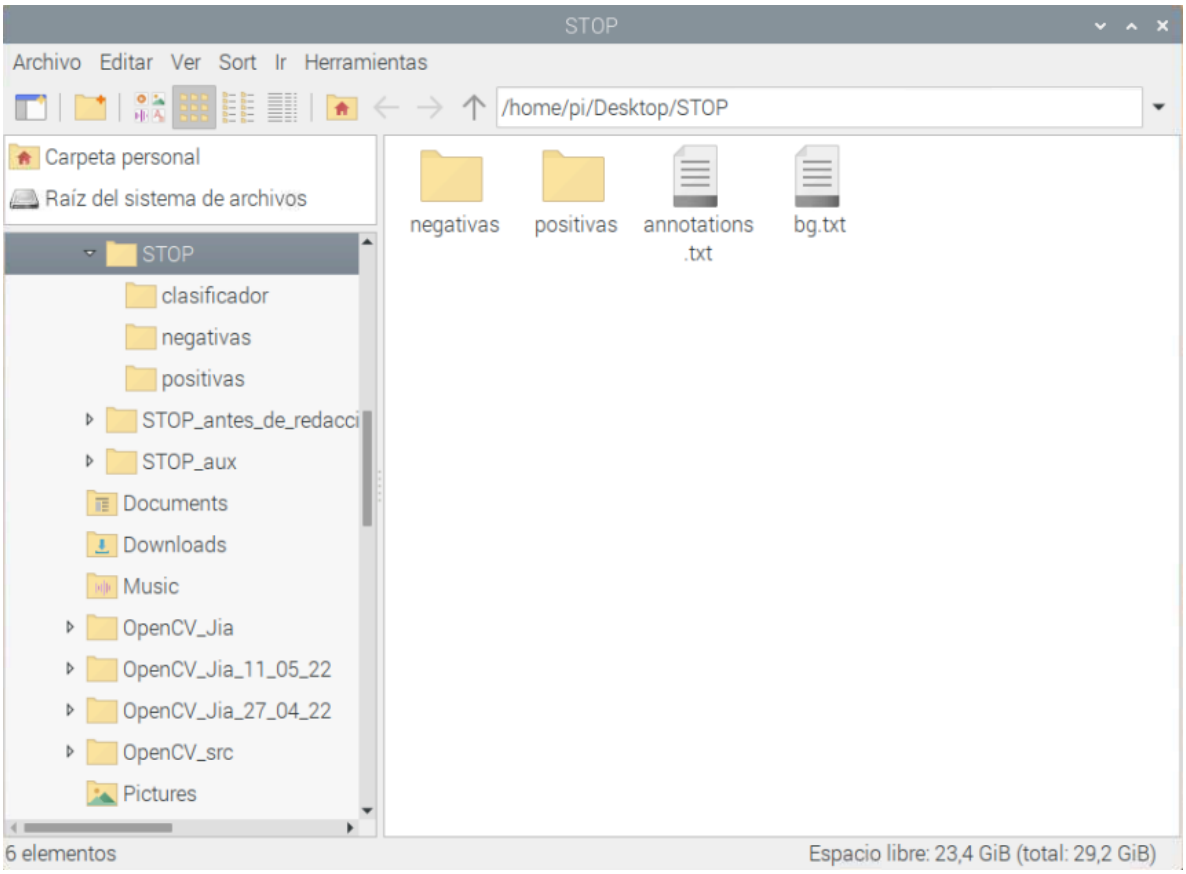


Figura 50. Directorio con las carpetas de imágenes positivas y negativas, y los dos archivos "txt" que indican las direcciones de las diferentes imágenes. En este caso "annotations.txt" para las imágenes positivas y "bg.txt" para las negativas.

Donde el archivo "annotations.txt" tiene un contenido con la siguiente estructura:

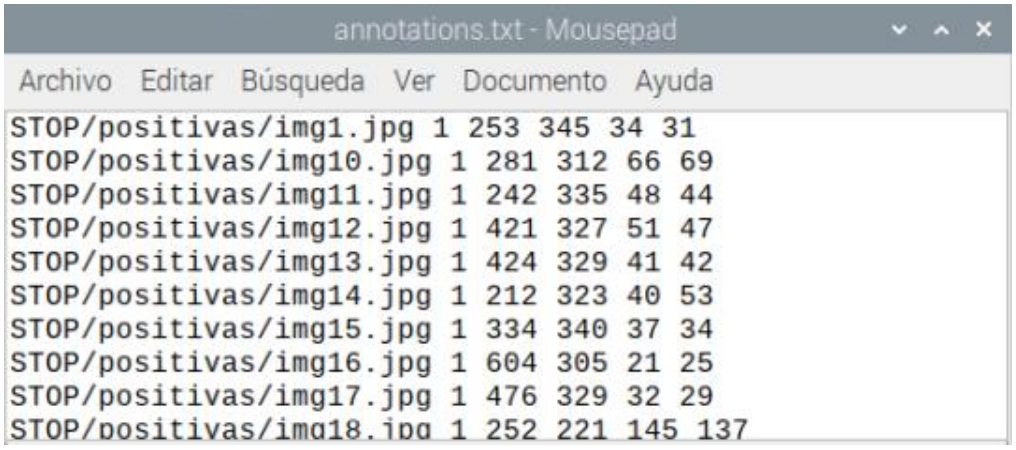


Figura 51. Archivo "txt" con las direcciones de las imágenes positivas, el número de objetos a detectar en la imagen, y las posiciones de estos.

A partir de esto, necesitaremos un archivo con extensión “vec” que utilizaremos para el entrenamiento de la red neuronal. Para crearlo utilizaremos el siguiente comando:

opencv_createsamples -info info.txt -bg bg.txt -vec prueba.vec -num 5 -w 30 -h 30

Donde los tres primeros parámetros son los arhivos que hemos creado en los apartados anteriores, “-num 5” al igual que en el apartado anterior indica el número de imágenes positivas, y, por último, “-w 30” y “-h 30” indican el ancho y el alto de las muestras que serán utilizadas para el entrenamiento de la red neuronal. Estos dos últimos parámetros, deberán tener un valor pequeño ya que sino, el sistema requeriría de una computación muy potente, pudiendo incluso saturarse.

Por último, vamos a utilizar el comando “opencv_traincascade” para entrenar la red neuronal. Antes de eso, deberemos crear una carpeta donde almacenaremos los archivos que irá creando opencv a la vez que va realizando el entrenamiento. una vez hecho esto, utilizamos el comando de la siguiente forma:

opencv_traincascade -data clasificador -vec prueba.vec -bg bg.txt -numPos 5 -numNeg 5 -numStages 10 -w 30 -h 30

Donde “clasificador” es la carpeta que hemos nombrado anteriormente, “prueba.vec” y “bg.txt” son los archivos que hemos creado en los pasos anteriores, “-numPos 5” indica el numero de imágenes positivas que se han utilizado, “-numNeg 5” indica el número de imágenes negativas que se han utilizado, “-numStages 10” indica el numero de etapas que queremos que realice el entrenamiento y por último, “-w 30” y “-h 30” indican el ancho y el alto de las muestras que hemos creado en el paso anterior.

Esto nos crea un archivo con extensión “xml” que podemos utilizar para realizar la detección de objetos.

Figura 52. Detección de señales de “stop” con red neuronal desarrollada a partir de una base de datos de imágenes positivas tomadas por el usuario.

Además, durante la detección de objetos se pueden ajustar algunos parámetros como el “minneighbours” o “scale”, a partir de los cuales podemos ajustar la precisión en la detección.

Figura 53. Detección de señales de “stop” con red neuronal desarrollada a partir de una base de datos de imágenes positivas tomadas por el usuario y con los parámetros “minneighbours” o “scale” ajustados.

7.1.2 Entrenamiento red neuronal creando las imágenes positivas

A parte de la metodología explicada en el apartado anterior, donde se crea una base de datos de imágenes positivas y se le aplica la función opencv_annotations para crear un archivo ".txt" donde se indica dónde están los objetos detectados. Se puede aplicar otra metodología por la cual no se necesita suministrar esa base de datos de imágenes positivas, sino que la crea el sistema.

Para ello, el sistema genera automáticamente toda la base de datos de imágenes positivas a partir de una imágen con el objeto a detectar y unos fondos que no contengan la imagen. A partir de estos, crea toda la base de datos utilizando diferentes combinaciones, aplicando transformaciones, rotaciones, etc.

Para realizar este proceso, primero se necesita tomar varias imágenes negativas para utilizar de fondo. Una vez se dispone de las imágenes negativas, se procede a la creación de las imágenes positivas con el siguiente comando:

opencv_createsamples -img ImagenPositiva -num NumeroImagenes -bg ArchivoTXTnegativas -info ArchivoTXTpositivas -maxxangle ÁnguloX -maxyangle ÁnguloY -maxzangle ÁnguloZ

Donde "ImagenPositiva" es la imagen del objeto para el cual queremos crear la red neuronal para que lo detecte, "NumeroImagenes" indica el número de imágenes positivas que queremos crear, "ArchivoTXTnegativas" es el archivo que se ha creado con las direcciones de las imagenes negativas, "ArchivoTXTpositivas" es el archivo que vamos a crear y que contendrá las direcciones de todas las imágenes positivas, y el resto de parámetros son los ángulos máximos que queremos que aplique en la imagen del objeto a la hora de crear la base de datos de imágenes positivas. A partir de este punto, el proceso es el mismo que para la metodología anterior.

Figura 54. Detección de señales de “stop” con red neuronal desarrollada a partir de una base de datos de imágenes positivas creadas por el sistema.

Como podemos comprobar en la figura 53 y la figura 54, la primera metodología permite la creación de una red neuronal mucho más robusta ya que incluye una mayor variedad de parámetros como la iluminación, reflejos o las formas, no contemplando únicamente transformaciones en imágenes. En la figura 53 vemos que solo detecta la señal de “stop”. Sin embargo, en la figura 54, se detecta la señal de “stop” y otros muchos objetos más.

7.2 DETECCIÓN LÍNEAS

Para la detección de las líneas de la carretera, a parte del método empleado explicado en el apartado anterior “Solución Adoptada”, existe el método por “Houghlines” Este método es capaz de detectar automáticamente las líneas presentes en la imagen.

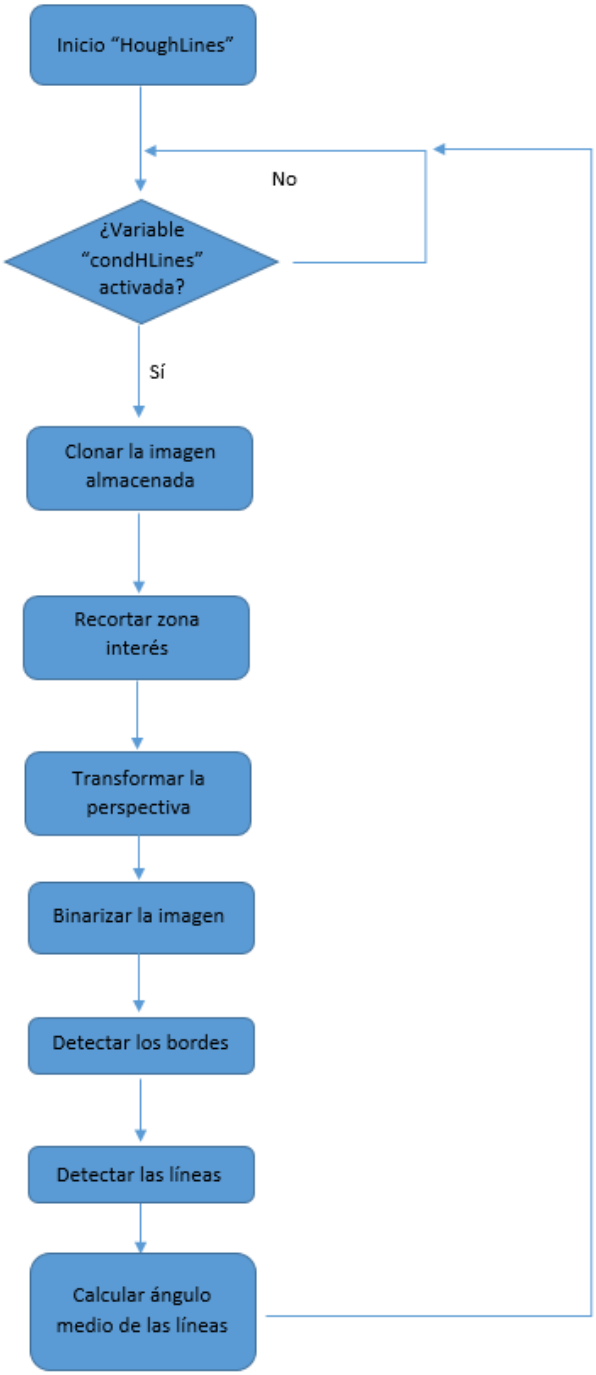


Figura 55. Flujograma para la detección de líneas mediante “houghlines”.

Para realizar este proceso lo primero que hacemos es recortar una zona de 540 píxeles de ancho y 150 píxeles de alto. Con un origen en las coordenadas (0, 150) respecto al origen de coordenadas de la imagen original.

A partir de este punto, se selecciona en la imagen las esquinas que captaría la cámara si pusiéramos un cuadrado en la vida real. Esto servirá para transformar la perspectiva y conseguir que esa figura tenga la forma de cuadrado que tendría si lo captáramos desde una vista superior. Una vez hecho esto, se transforma la imagen de BGR a HSV y se binariza para, igual que en el caso empleado en la "Solución Adoptada", aislar las líneas de la carretera y eliminar objetos que puedan perjudicar nuestro resultado. Posteriormente, se utiliza la función "Canny" para obtener los bordes de las formas que han quedado aisladas en la binarización.

Finalmente, aplicamos la función "HoughLinesP" para que el sistema detecte automáticamente las líneas que puede trazar en estos bordes. Para cada una de estas líneas, calculamos el ángulo que forma respecto al eje x positivo. En caso de ser un ángulo negativo, lo convertimos en positivo para facilitar el cálculo del ángulo medio de todas las líneas detectadas.

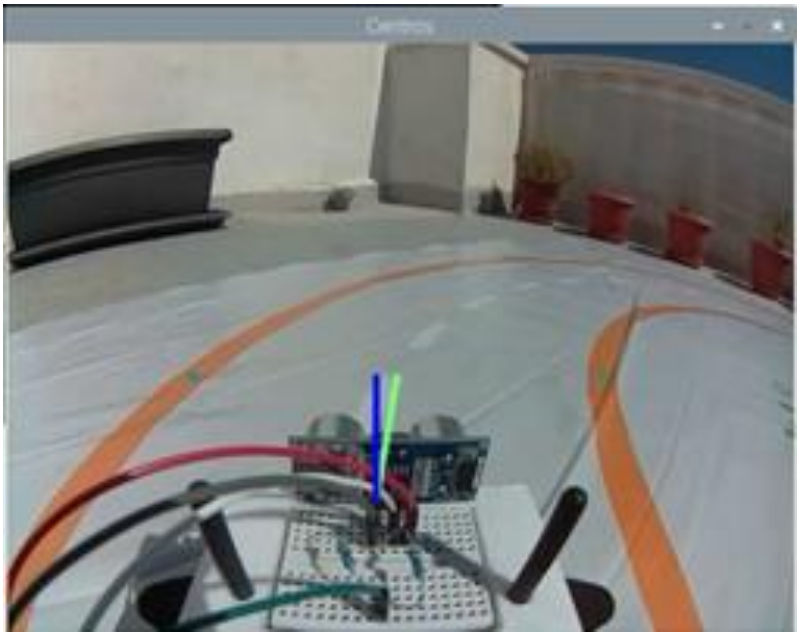


Figura 56. Detección de líneas por el método empleado en la "Solución adoptada"

Figura 57. Resultado obtenido para la detección de líneas por el método “houghlines”.

En comparación con el método empleado en la solución adoptada, es una metodología que tiene más facilidad en ofrecer resultados incorrectos, ya que para cada objeto que detecte va a calcular el ángulo de todas sus líneas. Por ejemplo, para una carretera recta, para la cual se debería detectar unas líneas con un ángulo medio de aproximadamente 90 grados. Si se obtiene una forma rectangular debido a que se trata de una parte de la carretera que no se ha binarizado correctamente, esas líneas horizontales que cierran la forma en la parte superior e inferior se detectarían como unos ángulos de 0 o 180 grados, lo que modificará en gran medida el ángulo medio detectado, produciendo errores significativos.

8. DESARROLLO Y GENERACIÓN DEL PROGRAMA.

8.1 CONFIGURACION RASPBERRY PI

Para poder utilizar la Raspberry Pi se necesitará instalar el sistema operativo en la tarjeta microSD. Para ello se descarga la versión deseada desde la página oficial de Raspberry, se descomprime y se graba la imagen del SO en la tarjeta de memoria microSD con el programa para Windows “Win32DiskImager”. Posteriormente se le conecta un monitor, un teclado, un ratón y el cable de la alimentación para realizar la configuración correspondiente.

Primero se modifica el archivo “/boot/config.txt” para forzar la salida de la imagen por el conector HDMI, aunque no detecte ningún monitor conectado. Posteriormente vamos ejecutamos el comando “sudo raspi-config”, este nos abrirá una ventana donde habilitaremos las interfaces (cámara, SSH, VNC y I2C), seleccionaremos nuestra localización y ampliaremos el tamaño de la memoria de la GPU a 256 MB (megabytes).

Ahora conectamos la Raspberry Pi a la red WIFI (tiene que ser la misma a la que se conecta el ordenador). Después, se procede a la instalación de OpenCV. Para ello, primero se instalan las dependencias correspondientes y posteriormente se descarga el código fuente de OpenCV3 (en este caso versión 3.4.9). Finalmente se compila e instala la versión de OpenCV que hemos descargado.
Por otro lado, se instala un servicio que nos muestra por la pantalla del vehículo información relevante como el porcentaje de carga de la batería, el porcentaje de uso de la CPU o la dirección IP. Además, realizamos la instalación de la librería WiringPi para utilizar las entradas y salidas GPIO de la Raspberry Pi.

8.2 VNC VIEWER

Para poder conectarnos mediante VNC a la Paspberry Pi y así no tener que tener conectados los periféricos al vehículo, se utiliza el programa VNC Viewer para Windows. Para ello se descarga el software desde la página oficial (www.realvnc.com) y se instala. Una vez instalado, al abrir el programa se abre una ventana donde se tiene que introducir la dirección IP a la que se quiere conectar. Posteriormente se debe introducir el usuario y contraseña de la Raspberry Pi. El usuario y la contraseña de fábrica son (“pi” y “raspberry”). Una vez establecida la conexión, ya se puede controlar la Raspberry Pi remotamente desde el ordenador.

8.3 COMPILAR EL CÓDIGO

Para compilar el código que se ha diseñado, primero de todo se debe abrir un terminal. Desde el terminal, abriremos la carpeta donde tenemos nuestro código y ejecutaremos “make –f Makefile”. Donde “Makefile” es un archivo previamente diseñado para realizar la compilación de nuestro código. Indicando entre otras cosas, todas las librerías que se necesitan incluir, el archivo ejecutable que se va a crear, los mensajes que queremos enviar por el terminal durante la compilación o los comandos que queremos ejecutar durante este proceso.

8.4 CONECTAR EL SENSOR DE DISTANCIA

Para que la Raspberry Pi pueda medir la distancia a través de nuestro código, debemos realizar el montaje del Plano 001 sobre una protoboard. Para la alimentación del sensor, podemos conectar 5V y GND a las salidas correspondientes de la Raspberry Pi. Este sistema se situará en la parte frontal del vehículo permitiendo así medir la distancia delantera del vehículo.

9. PRUEBAS FINALES

9.1 PRUEBA DE LA ALIMENTACIÓN

Primero, se debe comprobar la correcta conexión del sistema de alimentación con la placa de extensión y la Raspberry Pi. Para ello accionaremos el interruptor situado en la placa de extensión, si comienzan a iluminarse los leds en la Raspberry Pi y el ESC emite un pitido, se considera que el funcionamiento es el correcto

9.2 PRUEBA DE LA PANTALLA

Una vez verificado el correcto de la alimentación, se ha de comprobar la correcta configuración de la pantalla OLED que incorpora la placa de extensión. Al encender la Raspberry Ri se debería mostrar:

- La dirección IP
- El porcentaje de uso de la CPU, RAM y memoria de almacenamiento.
- La tensión, corriente y el porcentaje de la batería.

En caso de ser así, se considera que el funcionamiento es correcto.

9.3 PRUEBA DE CONEXIÓN VNC

Posteriormente verificaremos si se puede establecer la conexión VNC entre el ordenador y la Raspberry Pi. Para ello encendemos la Raspberry Pi y en el ordenador, abrimos el programa “VNC Viewer”. En este programa se introduce la dirección IP de la Raspberry, la cual la podemos encontrar en la pantalla de la placa de extensión o en la configuración de red de la Raspberry. Una vez seleccionada la dirección IP e introducidos el usuario y la contraseña, se debe de abrir una ventana que muestre el escritorio de la Raspberry Pi. En caso de ser así, se considera que el funcionamiento es correcto.

9.4 CONFIGURACIÓN DE VARIABLES

Antes de ejecutar el código principal diseñado en este proyecto, se debe compilar y ejecutar el código auxiliar (“calibracionparametros.cpp”) que se ha creado con el fin de obtener los parámetros óptimos que nos permiten realizar la mejor binarización posible según las condiciones del entorno. Al ejecutar este código, se deben ir desplazando las barras deslizantes hasta que en la imagen binarizada, obtengamos las líneas de la carretera de manera aisladas y nítidas.

Una vez obtenidos estos parámetros, abriremos el código “TFM.cpp” y los cambiaremos en la sección creada con este fin. Hecho esto, guardamos el archivo y lo compilamos.

9.5 PRUEBA DE FUNCIONAMIENTO

Una vez tenemos el archivo “TFM.cpp” configurado y compilado, lo ejecutamos. Una vez ejecutado se realizarán las siguientes comprobaciones.

Al inicio del código, se activa el modo “control con mando”. Con este modo comprobaremos si se realizan correctamente las siguientes comunicaciones:

- El mando con la Raspberry Pi.
- La Raspberry Pi con el servomotor de dirección.
- La Raspberry Pi con el controlador de velocidad del motor.

Para ello desplazaremos el joystick izquierdo al extremo izquierdo, posteriormente lo soltaremos, lo volveremos a desplazar hasta el extremo derecho y finalmente lo soltaremos. Durante este proceso, la dirección del vehículo debe girar a la izquierda, volver al centro, girar a la derecha, y volver finalmente al centro. En caso de seguir este comportamiento, verificaremos que las dos primeras comunicaciones se realizan correctamente. Para comprobar si la última comunicación se lleva a cabo de forma correcta, pulsaremos el botón R2, posteriormente lo soltaremos, pulsaremos el botón L2 y lo soltaremos. Siguiendo este comportamiento el vehículo debería acelerar, frenar, echar marcha atrás y volver a frenar. En caso de ser así, se considerará que se realiza la comunicación de forma correcta.

Ahora se va a proceder a verificar la correcta activación del modo “conducción asistida” y el correcto comportamiento de este modo. Para ello se pulsa el botón “L1” y en el terminal nos debe indicar el estado de la conducción asistida como “1”. Además, si el vehículo está centrado, nos debe permitir controlar la dirección de éste. En caso de tener una trayectoria muy desviada, el vehículo debe corregir automáticamente la dirección. Además, en el caso de haberse salido de la carretera, el vehículo debe echar marcha atrás hasta volver a una posición a partir de la cual se pueda seguir la trayectoria de la carretera correctamente. Además, exceptuando este último caso, el usuario debe ser siempre el que controla el desplazamiento del vehículo (hacia delante, parado o marcha atrás). Por último, volveremos a pulsar “L1” para desactivar la conducción asistida. En el caso de cumplirse todas estas condiciones, se considerará el comportamiento como adecuado.

Ahora se va a verificar la correcta activación del modo “conducción autónoma” y el correcto comportamiento de este modo. Para ello, se pulsa el botón “R1” y en el terminal nos debe indicar que el estado de la conducción autónoma es “1”. A partir de esta activación, el vehículo debe empezar a desplazarse por el circuito sin salirse de la carretera. Además, si colocamos una señal de “stop”, debe identificarla e indicarlo por el terminal, esperar 5 segundos y continuar con el seguimiento del circuito. Por otro lado, si colocamos un objeto en la trayectoria, el vehículo debe frenar e indicarlo por el terminal.

En este caso, el vehículo se mantendrá parado hasta que se retire el objeto, momento a partir del cual el vehículo continuará con el seguimiento del circuito. Al igual que en la conducción asistida, si el vehículo se sale de la carretera, éste debe echar marcha atrás hasta volver a una posición a partir de la cual se pueda seguir la trayectoria de la carretera correctamente. Por último, desactivaremos la conducción autónoma pulsando el botón “R1”. En el caso de cumplirse todas estas condiciones, se considerará el comportamiento como adecuado.

10. RESULTADOS

Una vez realizado el montaje del divisor de tensión previamente diseñado para la señal de salida del sensor de ultrasonidos, se obtiene una tensión de salida exacta de 3.30 V.

Durante las pruebas finales se han podido apreciar los siguientes resultados. El código se inicia en el control por mando. Donde podemos apreciar que el control se realiza de forma muy rápida, logrando una respuesta inmediata. En este modo controlamos la dirección del vehículo con el joystick izquierdo, utilizamos el botón “R2” para acelerar y el botón “L2” para echar marcha atrás. Por otro lado, si pulsamos el botón “L1”, se aprecia en el terminal que se activa la conducción asistida. En el caso de que el modo anterior fuera la conducción autónoma, se desactivaría ésta y se activaría la conducción asistida. En el caso de que el modo de control anterior fuera el control por mando, únicamente se activaría la conducción asistida. En este modo, si estamos situados en el centro del carril podemos controlar el vehículo con el mando a no ser que nos desviemos mucho de la trayectoria idónea, si lo hacemos el coche corregirá automáticamente la dirección para que, si seguimos acelerando, el vehículo vuelva al centro de la carretera. Por otro lado, si situamos el vehículo fuera de la carretera (como si se hubiese salido), este echa marcha atrás automáticamente hasta que se sitúe en una posición desde la que podamos seguir con un comportamiento normal. Ahora podemos desactivar la conducción autónoma volviendo a pulsar el botón “L1”.

En cambio, si pulsamos el botón “R1” en cualquier momento, activaremos la conducción autónoma. Por un lado, en el caso de que el modo de control anterior fuera control por mando, únicamente se activaría la conducción autónoma. Sin embargo, en el caso de que el modo de control anterior fuera la conducción asistida, al pulsar el botón se desactivaría la conducción asistida y se activaría la conducción autónoma. En este modo, el vehículo es capaz de detectar las líneas de la carretera y seguir el circuito completamente sin salirse. Además, por un lado frena automáticamente ante la presencia de un objeto que se interponga en la trayectoria del vehículo y por otro lado, frena 5 segundos si detecta una señal de “stop”. Este modo de conducción se puede desactivar si se vuelve a pulsar el botón “R1”

Por último, el tiempo de procesamiento total es de aproximadamente 6 ms para el control por mando y 0.4 segundos para los modos de conducción auxiliar y conducción autónoma.

11. ANALISIS Y DISCUSION DE RESULTADOS

El resultado obtenido a partir del divisor de tensión ha sido exactamente el esperado, por lo que podemos confirmar que nuestro cálculo se ha realizado correctamente. Además, la utilización del kit para el montaje del hardware ha facilitado mucho el proceso de desarrollo del proyecto. Por otro lado, la elección de la metodología empleada podemos considerarla correcta ya que se ha logrado conseguir el comportamiento deseado.

Además, respecto a la red neuronal entrenada, se ha conseguido desarrollar un sistema robusto que detecta correctamente las señales de “stop”. Sin embargo, si se cambia de ambiente puede que algunos objetos den lugar a falsos positivos, esto es debido a que no se ha empleado un base de datos de imágenes muy grande que contemple diferentes escenarios.

Sin embargo, los tiempos de procesamiento no se han conseguido reducir más allá de los tiempos obtenidos, por lo que el comportamiento conseguido en la conducción autónoma y la conducción auxiliar no llega a ser rápido y fluido. De esta forma, se tiene que parar el motor después de realizar la acción durante un corto periodo de tiempo. Esto es necesario ya que en caso de no realizar esta parada, el vehículo continuaría realizando la última orden durante mucho tiempo hasta recibir una nueva orden, por lo que podría chocar o salirse de la carretera.

12. CONCLUSIONES

Finalizado el proyecto se puede afirmar que hemos cumplido con todos los objetivos establecidos al principio de este documento. De esta forma, hemos conseguido controlar los actuadores del vehículo a través del mando remoto y mediante los diferentes modos de control que se han creado. Además, hemos logrado recibir información de un sensor de distancia y actuar en consecuencia. Por último, se han desarrollado funcionalidades que emplean el uso de visión artificial, tanto por técnicas clásicas como por inteligencia artificial.

Así, se ha logrado desarrollar un sistema completo a partir de la programación de un sistema empotrado. Habiendo diseñado un modo de funcionamiento semi-autónomo que hemos llamado “conducción asistida” y un modo de funcionamiento totalmente autónomo que es capaz de seguir un circuito, detectar señales de “stop” y detenerse ante la presencia de un obstáculo.

De esta forma, se han adquiridos nuevos conocimientos sobre cómo funciona la visión artificial y cómo se puede incorporar en un proyecto.

Así, a partir del proyecto desarrollado se podría diversificar en proyectos de diferentes ámbitos. Como puede ser un vehículo de guiado automático para la industria, donde se podría utilizar la visión artificial para detectar el entorno y el objeto que se ha introducido en el vehículo. De esta forma, él mismo sería capaz de desplazarse hasta su destino sin necesitar el control de un operario, reconociendo siempre su entorno y evitando así cualquier accidente.

13. RECOMENDACIONES Y TRABAJOS FUTUROS

En un futuro podría mejorarse la conducción auxiliar para que cuando vaya a realizar una corrección de trayectoria, varie la dirección de giro según el sentido de desplazamiento. Por ejemplo, si detecta una curva hacia la derecha y nosotros seguimos recto. En nuestro comportamiento diseñado, el vehículo giraría a la derecha para que el usuario acelere y así vuelva a una posición correcta. Sin embargo, si el usuario da marcha atrás, empeoraría la situación. En este caso, el vehículo debería de girar a la derecha en caso de que el usuario quiera acelerar; y girar a la izquierda en caso de que el usuario quiera dar marcha atrás.

En este mismo modo de funcionamiento, se podría controlar el tamaño de las señales de “stop” detectadas para que en el caso de que sea muy grande, es decir, que esté muy cerca. Frene automáticamente si el usuario no lo hace.

Por otro lado, en la solución propuesta, cuando el sistema hace una corrección, lo hace girando la dirección hasta el extremo directo o izquierdo. Esto para el modo de funcionamiento de conducción asistida no es un problema ya que el margen establecido es más grande y cuando hace falta la actuación del giro automático, tiene que girar mucho. Sin embargo, en la conducción autónoma donde el vehículo gira continuamente para mantener una posición lo más centrada posible, un giro proporcional al ángulo de desviación de la trayectoria lograría un resultado mucho más suave y fluido.

Además, si se desea utilizar el método de detección de las líneas de la carretera por “houghlines”, se podría aplicar ignorando los ángulos que sean muy horizontales (cercanos a 0 o 180 grados).

Por último, se podría incorporar un modo de funcionamiento del vehículo que se accionara mediante un botón, para entrar en un modo de configuración de los parámetros para la binarización. Donde se mostraría por pantalla la imagen recibida por la cámara, el panel de control con los parámetros de interés y la imagen binarizada. Sincronizando así estos parámetros con los que se utilizarán posteriormente durante el funcionamiento ordinario. O incluso, la aplicación de una red neuronal para que esta calibración se llevara a cabo automáticamente. De esta forma, el usuario no tendría que ejecutar un código diferente para realizar esta configuración.

14. REFERENCIAS BIBLIOGRÁFICAS

1 - https://docs.donkeycar.com/

2 - https://youtu.be/0Gkx57dOeYU

3 - https://github.com/hamuchiwa/AutoRCCar/find/master

4 - https://youtu.be/BBwEF6WBUQs

5 - https://www.adslzone.net/listas/gadgets/alternativas-raspberry-pi/

6 - https://www.pccomponentes.com/caracteristicas-raspberry-pi-4

7 - https://developer.nvidia.com/embedded/jetson-nano-developer-kit

8 - https://uelectronics.com/producto/nvidia-jetson-nano-developer-kit/

9 - https://www.asus.com/es/Networking-IoT-Servers/AIoT-Industrial-Solutions/All-series/Tinker-Board-S/

10 - https://amzn.to/3z4UYdh

11 - https://amzn.to/3z52GnP

12 - https://amzn.to/3cdjPCR

13 - https://www.nitrorcx.com/51c852-wildblue-24ghz.html

14 - https://amzn.to/3c97nnA

15 - https://amzn.to/3uOqTw1

16 - https://amzn.to/3z6qaJo

17 - https://amzn.to/3PoGCd9

18 - https://www.robocarstore.com/products/donkey-car-starter-kit?variant=40002462941232

19 - https://www.waveshare.com/piracer-pro-ai-kit.htm?sku=18492

20 - https://www.waveshare.com/jetracer-pro-2gb-ai-kit.htm

21 - Mastering C++ multithreading : a comprehensive guide to developing effective multithreading applications in C++. Maya Posch, 2017.

22 - OpenCV 3 computer vision application programming cookbook : recipes to help you build computer vision applications that make the most of the popular C++ library OpenCV 3. Robert Laganiere, 2017

23 - Object Tracking by Color and Active Contour Models Segmentation. A.S Silva, F.M.Q Severgnini, M.L Oliveira, V.M.S Mendes, Z.M.A Peixoto, 2016

24 - https://descubrearduino.com/hc-sr04-con-tu-raspberry-pi/

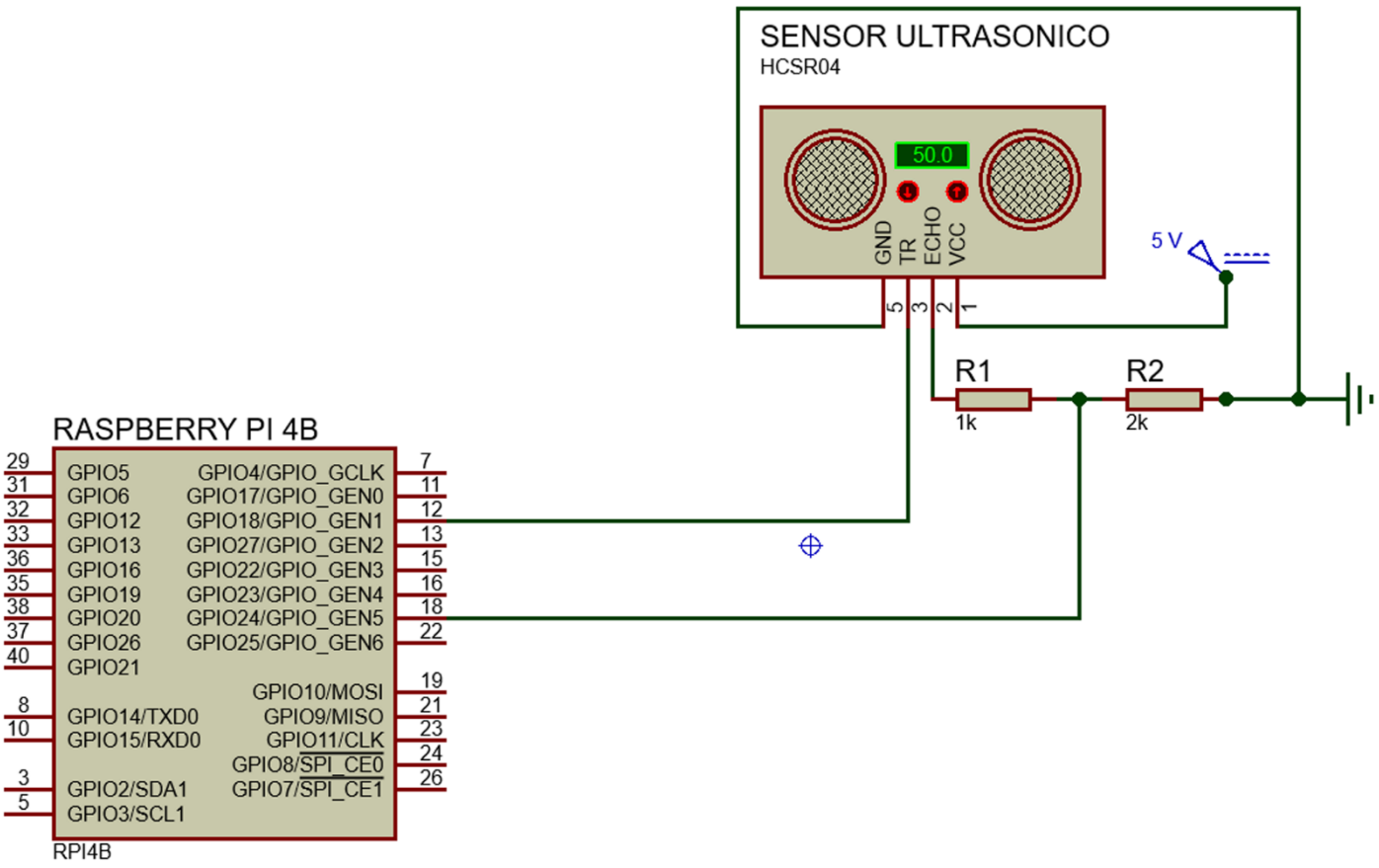
25 - https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html

26 - https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

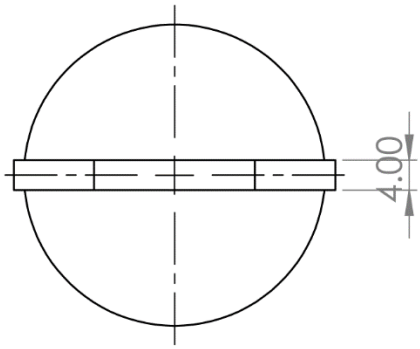
Sistema empotrado para el control autónomo de un coche a través de las imágenes recibidas por una cámara

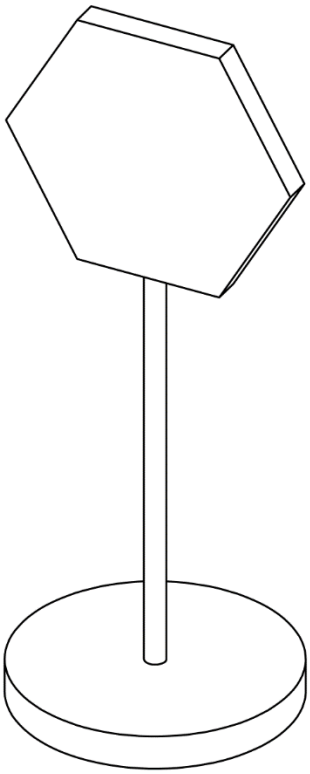
2. PLANOS Y CÓDIGO FUENTE

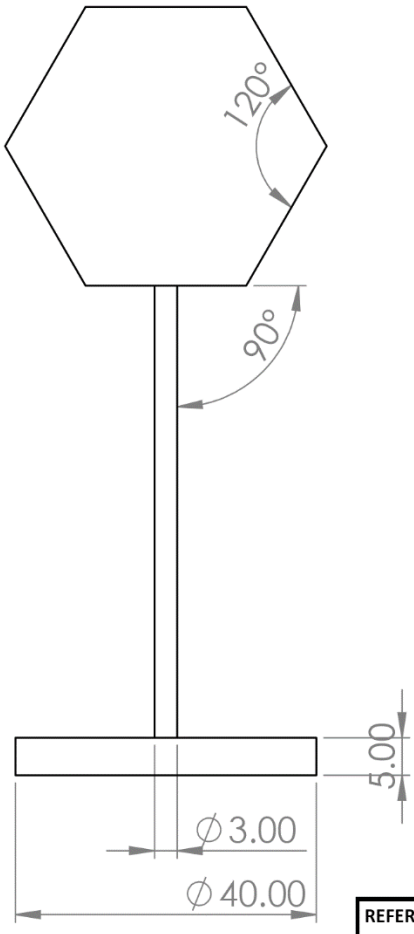
Índice de contenido

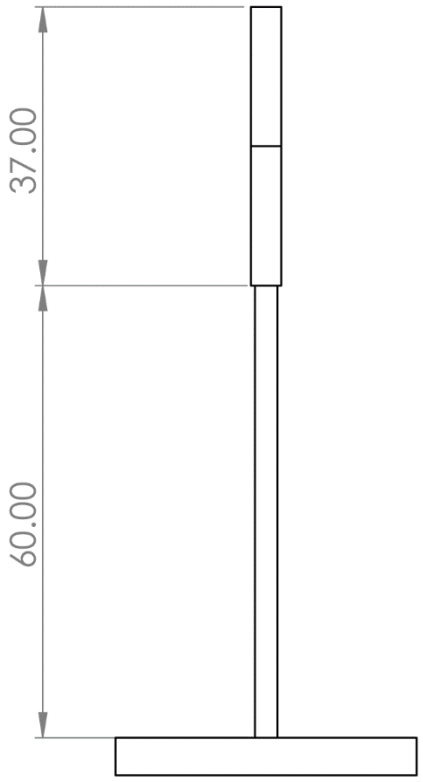


REFERENCIA: **P-001**	NOMBRE PLANO: **Circuito electrónico sensor ultrasonidos**	FECHA: **10-08-2022**
PROMOTOR: **TRABAJO FIN DE MÁSTER**		Nº PLANO: **001**
AUTOR: **JIA CHENG GAO YAO**		
TÍTULO DEL PROYECTO: **SISTEMA EMPOTRADO PARA EL CONTROL AUTÓNOMO DE UN COCHE A TRAVÉS DE LAS IMÁGENES RECIBIDAS POR UNA CÁMARA**		









REFERENCIA: **P-002**	NOMBRE PLANO: **Señal de stop**	FECHA: **10-08-2022**
PROMOTOR: **TRABAJO FIN DE MÁSTER**		Nº PLANO:
AUTOR: **JIA CHENG GAO YAO**		
TÍTULO DEL PROYECTO: **SISTEMA EMPOTRADO PARA EL CONTROL AUTÓNOMO DE UN COCHE A TRAVÉS DE LAS IMÁGENES RECIBIDAS POR UNA CÁMARA**		**002**

Código principal “TFM.cpp”

```
#include <wiringPi.h>
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <linux/joystick.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <iostream>
#include "I2C.h"
#include "PCA9685_PWM.h"
#include "Servo_PCA.h"
#include <pthread.h>
#include <opencv2/opencv.hpp>
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/videoio.hpp"

//Definimos los pines de entrada y salida para la medida de distancia

#define trigger 18
#define echo 24

//Establecemos los diferentes namespaces para las funciones que necesitaremos

using namespace cv;
using namespace std;
using namespace BBB;


//Creamos diferentes conjuntos de parámetros para la realizar la binarización según las condiciones ambientales

/*
//Con sol
int thresh =20;
int iLowH = 0;
int iHighH = 30;
int iLowS = 15;
int iHighS = 180;
int iLowV = 100;
int iHighV = 240;
*/
/*
//Con sombra
int thresh =20;
int iLowH = 0;
int iHighH = 15;
int iLowS = 50;
int iHighS = 140;
int iLowV = 100;
int iHighV = 255;
*/
/*
//Verano por la tarde
int thresh =20;
int iLowH = 0;
int iHighH = 179;
int iLowS = 0;
int iHighS = 100;
int iLowV = 0;
int iHighV = 150;
*/
/*
//Laboratorio
int thresh =20;
int iLowH = 0;
int iHighH = 50;
int iLowS = 33;
```

```
int iHighS = 200;
int iLowV = 115;
int iHighV = 255;
*/
//Cocina
int thresh =20;
int iLowH = 0;
int iHighH = 30;
int iLowS = 15;
int iHighS = 255;
int iLowV = 100;
int iHighV = 240;


//Almacenamos los resultados obtenidos en los diferentes procesos
int resultadocentros;// 0 = no se detectan lineas, 1 = girar a la izquierda, 2 = seguir recto, 3 = girar a la derecha
int resultadoSTOP; //Número de señales de STOP detectadas
int resultadodistancia; //Distnacia delante del vehículo en centímetros
double gradoscentros ; //Grados de desviación entre el centro del vehículo y el de la carretera

// Almacenamos la ultima direccion del motor del vehículo
int velocidad_anteriormando=0, velocidad_anteriorautonoma=0; // "0" Es marcha adelante, y "1" es marcha atras

// Almacenamos el estado del control autónomo y la conduccion asistida
bool conduccionautonoma = false, conduccionasistida = false;

// Almacenamos la imagen obtenida por la cámara
Mat imagenreferencia;

// Hilos y variables tipo "condition"
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; //Para la obtención de la imagen
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER; //Para medir la distancia
pthread_mutex_t mutex3 = PTHREAD_MUTEX_INITIALIZER; //Para almacenar las variables del mando
pthread_mutex_t mutex4 = PTHREAD_MUTEX_INITIALIZER; //Para almacenarlas variables del proceso de cálculo de
centros
pthread_mutex_t mutex5 = PTHREAD_MUTEX_INITIALIZER; //Para las almacenar las variables del proceso  STOP
pthread_mutex_t mutexcond = PTHREAD_MUTEX_INITIALIZER; //Para las variables tipo "condition"
pthread_cond_t condimagen = PTHREAD_COND_INITIALIZER; //Para activar el hilo "ObtenerImagen"
pthread_cond_t  condcontrol1  =  PTHREAD_COND_INITIALIZER;  //Primera  condición  para  activar  la  conducción
autónoma o la conducción asistida en el  hilo control
pthread_cond_t  condcontrol2  =  PTHREAD_COND_INITIALIZER;  //Segunda  condición  para  activar  la  conducción
autónoma
pthread_cond_t condcentros = PTHREAD_COND_INITIALIZER; //Para activar el hilo "centros"
pthread_cond_t condstop = PTHREAD_COND_INITIALIZER; //Para activar el hilo "detectarSTOP"


//Inicio de funciones para el control del servomotor y el controlador de la velocidad del motor
BBB::I2C m_i2c(1);
BBB::PCA9685 PWMController(&m_i2c, 0x40);
BBB::Servo_PCA servoDireccion(&PWMController,1);
BBB::Servo_PCA servoMotor(&PWMController,2);

//Variables globales para indicar el giro de la direccion y la velocidad del vehículo
float                                                                posDireccionmando=0.5,posMotormando=0.5,
posDireccionautonoma=0.5,posMotorautonoma=0.5,posDireccionasistida=0.5,posMotorasistida=0.5;

//Variables para el cálculo de los tiempos de procesamiento
long tcentros, timagen, tresultados, tstop, tdistancia, tmando;

//Definimos el clasificador
CascadeClassifier stop_cascade;

// Se declaran las funciones para la interaccion con el  mando

//Leemos si se ha producido algún evento en el mando.
//En caso afirmativo se develve un 0. En caso contrario, se duelve un -1.
int read_event(int fd, struct js_event *event)
{
    ssize_t bytes;

    bytes = read(fd, event, sizeof(*event));

    if (bytes == sizeof(*event))
        return 0;

    //Si no se ha podido leer el evento, devolvemos "-1"
```

```
    return -1;
}

//Definimos la estructura que tendrán los joysticks del mando
struct axis_state {
    short x, y;
};

//Obtenemos el valor de los joysticks.
//Para ello tendremos en cuenta que el eje "x" está identificado con un número de evento par y el eje "y" con un número de evento impar

size_t get_axis_state(struct js_event *event, struct axis_state axes[3])
{
    size_t axis = event->number / 2;

    if (axis < 3)
    {
        if (event->number % 2 == 0) //Si el identificador del evento es par
            axes[axis].x = event->value; //Asignamos el valor al eje "x"
    else //Si el identificador del evento es impar
        axes[axis].y = event->value; //Asignamos el valor al eje "y"
    }

    return axis;
}

//Hilo para medir la distancia
void *MedirDistancia (void *)
{
    int resultadodistanciaaux;
    long iniciodistancia, finaldistancia, difdistancia;
    long startTimeUsec, endTimeUsec, travelTimeUsec, now;
    double distanceMeters;

    //Iniciamos los pines GPIO
    wiringPiSetupGpio();
    pinMode(trigger, OUTPUT); //Trigger
    pinMode(echo, INPUT); //Echo
    digitalWrite(trigger, LOW);

    while(1)
    {

    //Establecemos el tiempo inicial del hilo
    iniciodistancia = micros();
        //Enviamos un pulso de 10 ms
    digitalWrite(trigger, HIGH);
    usleep(10);
    digitalWrite(trigger, LOW);
    //Medimos el final del envio del pulso
    while (digitalRead(echo) == LOW)
    {
        startTimeUsec = micros();
    }
    //Medimos el tiempo final de la recepción del pulso

    while ( digitalRead(echo) == HIGH )
    {
        //Leemos el tiempo final de la recepción del pulso
        endTimeUsec = micros();
    }
    //Calculamos la diferencia entre ambos tiempos
    travelTimeUsec = endTimeUsec - startTimeUsec;
    //Calculamos la distancia teniendo el cuenta la velocidad del sonido y que el recorrido es de ida y vuelta
    resultadodistanciaaux = ((travelTimeUsec/1000000.0)*34300.0)/2;

    //Calculamos el tiempo que ha tardado en procesar este hilo
        finaldistancia = micros();
        difdistancia = finaldistancia - iniciodistancia;

    //Almacenamos los resultados como variables globales
        pthread_mutex_lock(&mutex2);
        resultadodistancia = resultadodistanciaaux;
        tdistancia = difdistancia;
        pthread_mutex_unlock(&mutex2);
```

```
    usleep(300000);
    }
}
void *Mando (void *)
{
    const char *device;
    int js;
    struct js_event event;
    struct axis_state axes[3] = {0};
    size_t axis;
    float posDireccionmandoauxiliar=0.5,posMotormandoauxiliar=0.5;
    int velocidad_anteriormandoauxiliar=0;
    bool conduccionautonomaauxiliar, conduccionasistidaauxiliar;
    long iniciomando, finalmando, difmando;



    //Abrimos el mando al que nos queremos conectar
    device = "/dev/input/js0";
    js = open(device, O_RDONLY);

    //Notificamos en caso de no poder abrir el mando
    if (js == -1)
        perror("No se ha podido abrir el mando");


    while (read_event(js, &event) == 0)
    {
    //Anotamos el tiempo inicial de procesamiento del hilo
        iniciomando = micros();

    //Leemos los valores de las variables globales de interés
        pthread_mutex_lock(&mutex3);
        conduccionautonomaauxiliar=conduccionautonoma;
        conduccionasistidaauxiliar=conduccionasistida;
        pthread_mutex_unlock(&mutex3);

        //Clasificamos el evento recibido por el mando
        switch (event.type)
        {
        //En caso de ser un botón. Podemos obtener información de qué botón se trata y si la acción ha sido "pulsar" o
"soltar"
            //printf("El botón %u ha sido %s\n", event.number, event.value ? "presionado" : "soltado");

        //Establecemos un comportamiento para los botones que deseamos.
        case JS_EVENT_BUTTON:
        switch (event.number)
        {
            //En el caso del acelerador
            case 9:
            //Si la acción es "pulsar" establecemos que el coche debe acelerar y la ultima velocidad ha sido "hacia delante"
            if( event.value==1)
            {
                posMotormandoauxiliar=0.7;
                velocidad_anteriormandoauxiliar=0;
            }
            //En caso contrario, si la acción es "soltar", establecemos que el coche debe frenar
            else
            {
                posMotormandoauxiliar=0.5;
            }
                break;

            //En el caso de la marcha atras
                case 8:
            //Si la acción es "pulsar" establecemos que el coche debe ir marcha atrás
            if( event.value==1)
            {
                posMotormandoauxiliar=0.3;
            }
            //En caso contrario, si la acción es "soltar", establecemos que el coche debe frenar
            else
            {
                posMotormandoauxiliar=0.5; //paramos el acelerador cuando dejemos de pulsar
            }
```

```
                break;

            // En el caso de la activación del control autónomo.
                case 7:
            //Si la acción es "pulsar", alternamos el valor del control autónomo.
            if( event.value==1)
            {
               //Si esta desactivado, lo activamos y desactivamos la conduccion asistida
               if (conduccionautonomaauxiliar==false)
               {
               conduccionautonomaauxiliar=true;
               conduccionasistidaauxiliar=false;
               }
               //Si por el contrario está activado, lo desactivamos
               else{
               conduccionautonomaauxiliar=false;
               }
            }
                break;

            //En el caso de la activación de la conducción asistida
                case 6:
            //Si la acción es "pulsar", alternamos el valor de la conducción asistida
            if( event.value==1)
            {
               //Si esta desactivado, lo activamos y desactivamos el control autónomo
               if (conduccionasistidaauxiliar==false)
               {
               conduccionasistidaauxiliar=true;
               conduccionautonomaauxiliar=false;
               }
               //Si por el contrario está activado, lo desactivamos
               else
               {
               conduccionasistidaauxiliar=false;
               }
            }
                break;
                default:
                break;
               }

               break;
         //En caso de ser un joystick. Podemos obtener información de que joystick se trata y su desplazamiento en el eje "x" e "y" respecto a la posición de reposo
            case JS_EVENT_AXIS:
               axis = get_axis_state(&event, axes);
               if (axis < 3)
         {
            //printf("Joystick %zu con las posiciones (%6d, %6d)\n", axis, axes[0].x, axes[1].y);
                  //Almacenamos el desplazamiento en el eje "x" del joystick izquierdo como la direccion enviada por el mando
            posDireccionmandoauxiliar = -1*(-0.50+(float(axes[0].x)/70000));
               }
         break;
               default:
         break;
         }

      //Establecemos el tiempo final que ha tardado en procesar el hilo "mando"
      finalmando = micros();
      //Calculamos la diferencia entre el tiempo final y el tiempo inicial
      difmando = finalmando - iniciomando;

      //Almacenamos como variables globales:
      pthread_mutex_lock(&mutex3);
      posMotormando=posMotormandoauxiliar; //El comportamiento que debe tener el motor según el mando
      posDireccionmando = posDireccionmandoauxiliar; //El comportamiento que debe tener el servomotor de dirección según el mando
      conduccionautonoma = conduccionautonomaauxiliar; //El estado del control autónomo
      conduccionasistida = conduccionasistidaauxiliar; //El estado de la conducción asistida
      velocidad_anteriormando = velocidad_anteriormandoauxiliar; //La última dirección del motor de tracción
      tmando = difmando; //El tiempo de procesamiento que ha tardado el hilo "mando"
      pthread_mutex_unlock(&mutex3);
      }
      //Cerramos el mando al que nos hemos conectado
```

```
    close(js);
    return 0;
}
void *Centros (void *)
{
    Mat imagecentros, greycentros, cropcentros;
    double sumaizq=0.0;
    double contadorizq=0.0;
    double mediaizq=0.0;
    double sumader=0.0;
    double contadorder=0.0;
    double mediader=0.0;
    double alphacentros, area, gradoscentrosauxiliar;
    long iniciocentros, finalcentros, difcentros;
    float posDireccionasistidaauxiliar,posMotorasistidaauxiliar, posDireccionautonomaauxiliar,posMotorautonomaauxiliar;
    int resultadocentrosauxiliar, velocidad_anteriorautonomaauxiliar;
    Scalar color = Scalar(167,151,0); // Definimos un color por sus valores B G R

    while(1)
    {

    //Pausamos el hilo hasta que se cumpla la condicion "condcentros"
        pthread_mutex_lock(&mutexcond);
        pthread_cond_wait ( &condcentros, &mutexcond);
        pthread_mutex_unlock(&mutexcond);

    //Anotamos el tiempo inicial de procesamiento del hilo
        iniciocentros = micros();

    //Clonamos la imagen almacenada como una variable global en el hilo "ObtenerImagen"
        pthread_mutex_lock(&mutex);
        imagecentros = imagenreferencia.clone();
        pthread_mutex_unlock(&mutex);

        //Recortamos zona de interes
        cv::Rect crop_regioncentros(0,250,640,40);
        cropcentros= imagecentros(crop_regioncentros);

    //Comienzo binarizacion
    //Convertimos la imagen de GBR a HSV
        Mat imgHSVcentros;
        cvtColor(cropcentros, imgHSVcentros, COLOR_BGR2HSV);

    //Binarizamos la imagen resultante según los parámetros establecidos anteriormente
        Mat imgThresholdedcentros;
        inRange(imgHSVcentros, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholdedcentros);
//Threshold the image
    //Final de la binarizacion

        //Calculamos contornos
        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;
        findContours( imgThresholdedcentros, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE );

        //Calculamos los centros

    // Primero obtenemos los momentos
    vector<Moments> mu(contours.size());
    for( int i = 0; i<contours.size(); i++ )
    {
        mu[i] = moments( contours[i]);
    }
    // Ahora obtenemos los centroides de las figuras
    vector<Point2f> mc(contours.size());
    for( int i = 0; i<contours.size(); i++)
    {
        mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 );
    }

    //Solo tendremos en cuenta las figuras con un area considerablemente grande, logrando así reducir la presencia de
ruido.
    for( int i = 0; i<contours.size(); i++ )
    {
        area = contourArea(contours[i]);
        if (area>200)
        {
```

```
        //Si sus coordenadas en "x" están a la izquierda del centro aproximado del vehículo
        if (mc[i].x<300)
        {
            contadorizq=contadorizq+1;
            sumaizq=sumaizq+mc[i].x;
        }
        //Si sus coordenadas en "x" están a la derecha del centro aproximado del vehículo
        else if (mc[i].x>300)
        {
            contadorder=contadorder+1;
            sumader=sumader+mc[i].x;
        }
        circle( imagecentros, cv::Point(mc[i].x, 270), 4, color, 6); //Ponemos un punto en las coordenadas en "x" del centro y a una altura determinada (altura media de la imagen recortada)
        }
    }

    //Una vez obtenidos todos los centros para cada lado, calculamos la posición media
        mediaizq=sumaizq/contadorizq;
        mediader=sumader/contadorder;

    //Trazamos dos lineas que muestren:
    //- Una primera linea recta vertical desde el centro de la parte delantera del vehículo
    //- Una segunda linea que muestre la desviación entre el centro de la carretera y el del vehículo
    line(imagecentros, Point((mediader-(mediader-mediaizq)/2), 270), Point(300, 370), Scalar(122, 255, 122), 3, LINE_AA);
    line(imagecentros, Point(300, 270), Point(300, 370), Scalar(255, 0, 0), 3, LINE_AA);

    //Mostramos la imagen por pantalla
    imshow( "Centros", imagecentros );

    //Calculamos el ángulo de la segunda linea en radianes
    alphacentros=atan(double(370-270)/((mediader-(mediader-mediaizq)/2)-300));
    // Y lo convertimos a grados
    gradoscentrosauxiliar = alphacentros*180.0/M_PI;

    //En caso de ser negativo el ángulo, lo convertimos a positivo.
    if(gradoscentrosauxiliar<0)
    {
        gradoscentrosauxiliar=gradoscentrosauxiliar+180;
    }

    //Si no se ha detectado ninguna de las lineas de la carretera
    if (isnan(mediaizq)&&isnan(mediader))
    {
        //El coche debería ir marcha atrás en el modo de conducción autónoma y en la conducción asistida
        resultadocentrosauxiliar = 0;
        posMotorautonomaauxiliar=0.3;
        posDireccionautonomaauxiliar = 0.5;
        posMotorasistidaauxiliar=0.3;
        posDireccionasistidaauxiliar = 0.5;

    }
    //Si no se detecta la linea izquierda pero sí la derecha
    else if (isnan(mediaizq))
    {
        resultadocentrosauxiliar = 1; // Resultado es girar a la izquierda
        posMotorasistidaauxiliar=0.5; // En la conducción asistida, el desplazamiento del vehículo se controlará con el mando
        posMotorautonomaauxiliar=0.7; // En el control autónomo el vehículo se desplazará hacia delante
            velocidad_anterioraautonomaauxiliar=0; // Almacenaremos que la ultima dirección de desplzamiento del vehículo con el control autónomo es hacia delante
        //Orientación del vehículo hacia la izquierda
        posDireccionasistidaauxiliar = 1;
        posDireccionautonomaauxiliar = 1;
    }
    //Si no se detecta la linea derecha pero sí la izquierda
    else if (isnan(mediader))
    {
        resultadocentrosauxiliar = 3; // Resultado es girar a la derecha
        posMotorasistidaauxiliar=0.5; // En la conducción asistida, el desplazamiento del vehículo se controlará con el mando
        posMotorautonomaauxiliar=0.7; // En el control autónomo el vehículo se desplazará hacia delante
            velocidad_anterioraautonomaauxiliar=0; // Almacenaremos que la ultima dirección de desplzamiento del vehículo con el control autónomo es hacia delante
        //Orientación del vehículo hacia la derecha
```

```
            posDireccionautonomaauxiliar = 0;
        posDireccionasistidaauxiliar = 0;

    }
    //Si detecta ambas lineas y el centro de la carretera está al lado derecho del centro del vehículo
    else if(gradoscentrosauxiliar<87)
    {
        resultadocentrosauxiliar = 3; // Resultado es girar a la derecha
        posMotorautonomaauxiliar=0.7; // En el control autónomo el vehículo se desplazará hacia delante
            velocidad_anteriorautonomaauxiliar=0; // Almacenaremos que la ultima dirección de desplzamiento del vehículo
con el control autónomo es hacia delante
            posDireccionautonomaauxiliar = 0; //Orientación del vehículo hacia la derecha
        //Para la conducción asistida aumentamos el margen para que no esté interviniendo continuamente
        if (gradoscentrosauxiliar<80)
        {
        posDireccionasistidaauxiliar = 0; //Orientación del vehículo hacia la derecha
        posMotorasistidaauxiliar=0.5; // En la conducción asistida, el desplazamiento del vehículo se controlará con el
mando
        }
    }
    //Si detecta ambas lineas y el centro de la carretera está al lado izquierdo del centro del vehículo
    else if(gradoscentrosauxiliar>97)
    {
        resultadocentrosauxiliar = 1; // Resultado es girar a la izquierda
        posMotorautonomaauxiliar=0.7; // En el control autónomo el vehículo se desplazará hacia delante
            velocidad_anteriorautonomaauxiliar=0; // Almacenaremos que la ultima dirección de desplzamiento del vehículo
con el control autónomo es hacia delante
            posDireccionautonomaauxiliar = 1; //Orientación del vehículo hacia la izquierda
        // Aumentamos el margen como hemos hecho para la conducción asistida en el lado derecho
        if (gradoscentrosauxiliar>100)
        {
        posDireccionasistidaauxiliar = 1; //Orientación del vehículo hacia la izquierda
        posMotorasistidaauxiliar=0.5; // En la conducción asistida, el desplazamiento del vehículo se controlará con el
mando
        }
    }
    //Si se detectan ambas lineas y el centro de la carretera no se desvia más allá de los margenes establecidos
    else
    {
        resultadocentrosauxiliar = 2; // Resultado es seguir recto
        posMotorautonomaauxiliar=0.7; // En el control autónomo el vehículo se desplazará hacia delante
        posDireccionautonomaauxiliar = 0.5; //Orientación recta del vehículo
            velocidad_anteriorautonomaauxiliar=0; // Almacenaremos que la ultima dirección de desplzamiento del vehículo
con el control autónomo es hacia delante
        posDireccionasistidaauxiliar = 0.5; //Orientación recta del vehículo
        posMotorasistidaauxiliar=0.5; // En la conducción asistida, el desplazamiento del vehículo se controlará con el
mando
    }

    //Establecemos el tiempo final que ha tardado en procesar el hilo "centros"
    finalcentros = micros();
    //Calculamos la diferencia entre el tiempo final y el tiempo inicial
    difcentros = finalcentros - iniciocentros;

    //Almacenamos como variables globales:
    pthread_mutex_lock(&mutex4);
    gradoscentros = gradoscentrosauxiliar; //Los grados de la línea que muestra la desviación entre el centro de la
carretera y el del vehículo
    resultadocentros = resultadocentrosauxiliar; //El resultado obtenido en el hilo centros
    posMotorasistida = posMotorasistidaauxiliar; //El comportamiento que debe tener el motor del vehículo para la
conducción asistida
    posDireccionasistida = posDireccionasistidaauxiliar; //El comportamiento que debe tener el servomotor de dirección
del vehículo para la conducción asistida
    posMotorautonoma = posMotorautonomaauxiliar; //El comportamiento que debe tener el motor del vehículo para la
conducción autónoma
    posDireccionautonoma = posDireccionautonomaauxiliar;  //El comportamiento que debe tener el servomotor de
dirección del vehículo para la conducción autónoma
    velocidad_anteriorautonoma = velocidad_anteriorautonomaauxiliar; // Almacenamos la ultima dirección de
desplzamiento del motor del vehículo
    tcentros = difcentros; //El tiempo de procesamiento que ha tardado el hilo "centros"
    pthread_mutex_unlock(&mutex4);

    //Enviamos la primera condición para activar la conducción autónoma o la conducción asistida en el  hilo control
    pthread_mutex_lock(&mutexcond);
    pthread_cond_signal ( &condcontrol1);
    pthread_mutex_unlock(&mutexcond);
```

```
    //Reestablecemos los valores
    sumaizq=0.0;
    contadorizq=0.0;
    mediaizq=0.0;
    sumader=0.0;
    contadorder=0.0;
    mediader=0.0;

        usleep(100000);
        waitKey(1);
    }
}

void *DetectarSTOP (void *)
{
    float scale = 1.2;
    int minne = 8;
    int resultadoSTOPauxiliar, contador=0;
    long iniciostop, finalstop, difstop;
    Mat imagestop, imagestop_gray;

    //Seleccionamos la red neouronal entrenada
    String stop_cascade_name = "cascadestop.xml";

    //Lo cargamos
    if( !stop_cascade.load( stop_cascade_name ) )
    {
        cout << "--(!)Error cargando la red neuronal\n";
    }

    while (1)
    {
        //pausamos el hilo hasta que se active la condicion "condstop"
        pthread_mutex_lock(&mutexcond);
        pthread_cond_wait ( &condstop, &mutexcond);
        pthread_mutex_unlock(&mutexcond);

    //Anotamos el tiempo inicial de procesamiento del hilo
        iniciostop = micros();

    //Clonamos la imagen almacenada como una variable global en el hilo "ObtenerImagen"
        pthread_mutex_lock(&mutex);
        imagestop=imagenreferencia.clone();
        pthread_mutex_unlock(&mutex);
    //Convertimos la imagen de BGR a escala de grises
        cvtColor( imagestop, imagestop_gray, COLOR_BGR2GRAY );
    //Le aplicamos un ecualizador de histograma para mejorar el resultado
    equalizeHist( imagestop_gray, imagestop_gray );
    //Aplicamos la detección de objetos utilizando nuestra red neuronal entrenada
    std::vector<Rect> stops;
    stop_cascade.detectMultiScale( imagestop_gray, stops , scale,minne);
    //Rodeamos cada objeto detectado
    for ( size_t i = 0; i < stops.size(); i++ )
    {
        if(stops[i].width*stops[i].height>2500)
        {
        Point center( stops[i].x + stops[i].width/2, stops[i].y + stops[i].height/2 );
        ellipse( imagestop, center, Size( stops[i].width/2, stops[i].height/2 ), 0, 0, 360, Scalar( 255, 0, 255 ), 4 );
        contador++;
        }
    }
    imshow("Image STOP", imagestop);
    //almacenamos el numero de objetos detectados
    resultadoSTOPauxiliar = contador;

    //Establecemos el tiempo final que ha tardado en procesar el hilo "detectarSTOP"
        finalstop = micros();
    //Calculamos la diferencia entre el tiempo final y el tiempo inicial
        difstop = finalstop - iniciostop;
    //Almacenamos como variables globales:
        pthread_mutex_lock(&mutex5);
    resultadoSTOP = resultadoSTOPauxiliar; //El número de objetos detectados
    tstop = difstop; //El tiempo de procesamiento que ha tardado el hilo "detectarSTOP "
    pthread_mutex_unlock(&mutex5);
```

```
    //Enviamos la segunda condición para activar la conducción autónoma
        pthread_mutex_lock(&mutexcond);
        pthread_cond_signal ( &condcontrol2);
        pthread_mutex_unlock(&mutexcond);
    contador=0;
    waitKey(1);
    }

}

void *ObtenerImagen (void *)
{
    Mat image1, image2, image3, image4, image5;
    int contador = 0;
    long inicioimagen, finalimagen, difimagen;
    //Establecemos la captura a través de cualquier cámara
    VideoCapture cap(CV_CAP_ANY);
    //Establecemos por Software un tamaño de buffer de 0 imágenes
    cap.set(CV_CAP_PROP_BUFFERSIZE, 0);
    //Establecemos un tamaño de fram de 640 x 480
    cap.set (CV_CAP_PROP_FRAME_HEIGHT, 480);
    cap.set (CV_CAP_PROP_FRAME_WIDTH, 640);

    //Establecemos un método para indicar si no se ha podido abrir la cámara
    if( !cap.isOpened() )
    {
        cout << "Dispositivo de captura no encontrado.\n";
        return 0;
    }

    while(1)
    {
    //Pausamos el hilo hasta que se active la condición
    pthread_mutex_lock(&mutexcond);
        pthread_cond_wait ( &condimagen, &mutexcond);
        pthread_mutex_unlock(&mutexcond);

    //Anotamos el tiempo inicial de procesamiento del hilo "ObtenerImagen"
        inicioimagen = micros();

    //Tomamos 5 imágenes
        cap.read(image1);
        cap.read(image2);
        cap.read(image3);
        cap.read(image4);
        cap.read(image5);

    //Rotamos la última
        cv::rotate(image5, image5, cv::ROTATE_180);

    //Establecemos el tiempo final que ha tardado en procesar el hilo "ObtenerImagen"
        finalimagen = micros();

    //Calculamos la diferencia entre el tiempo final y el tiempo inicial
    difimagen = finalimagen - inicioimagen;

    //Almacenamos como variables globales:
        pthread_mutex_lock(&mutex);
        imagenreferencia=image5.clone(); //La última imagen tomada
        timagen = difimagen;//El tiempo de procesamiento que ha tardado el hilo "ObtenerImagen"
        pthread_mutex_unlock(&mutex);

    //Enviamos las condiciones que activan los hilos "centros" y "detectarSTOP"
        pthread_mutex_lock(&mutexcond);
        pthread_cond_signal ( &condstop);
        pthread_cond_signal ( &condcentros);
        pthread_mutex_unlock(&mutexcond);

    }
}

void *Control (void *)
{
    Mat imagenaux;
    //Establecemos los rangos para el servomotor y el controlador del motor
    servoDireccion.setRange(245, 345); //Todo derecha 245 //Todo izquierda 345 //centro entre 295 y 300
```

```
    servoMotor.resetStandardRange();

    float posDireccionautonomaR,posMotorautonomaR;
    int velocidad_anteriorautonomaR;
    int resultadoSTOPR;
    float posDireccionasistidaR,posMotorasistidaR;
    int resultadocentrosR;
    float posDireccionmandoR,posMotormandoR;
    int velocidad_anteriormandoR=0;
    bool conduccionautonomaR, conduccionasistidaR;
    double gradoscentrosR;
    int resultadodistanciaR;
    long tcentrosR, timagenR, tstopR, tdistanciaR, tmandoR, tfinalR, tinicioR, ttotalR;
    int contadorSTOP=0;

    while(1)
    {
    //Almacenamos el valor inicial del tiempo total
    tinicioR = micros();

    //Tomamos los valores de todas las variables globales almacenadas
    pthread_mutex_lock(&mutex);
    timagenR = timagen;
    pthread_mutex_unlock(&mutex);

    pthread_mutex_lock(&mutex2);
    resultadodistanciaR=resultadodistancia;
    tdistanciaR = tdistancia;
    pthread_mutex_unlock(&mutex2);

    pthread_mutex_lock(&mutex3);
    posMotormandoR=posMotormando;
    posDireccionmandoR = posDireccionmando;
    conduccionautonomaR = conduccionautonoma;
    conduccionasistidaR = conduccionasistida;
    velocidad_anteriormandoR = velocidad_anteriormando;
    tmandoR = tmando;
    pthread_mutex_unlock(&mutex3);

    pthread_mutex_lock(&mutex4);
    gradoscentrosR = gradoscentros;
    resultadocentrosR = resultadocentros;
    posMotorasistidaR = posMotorasistida;
    posDireccionasistidaR = posDireccionasistida;
    posMotorautonomaR = posMotorautonoma;
    velocidad_anteriorautonomaR = velocidad_anteriorautonoma;
    posDireccionautonomaR = posDireccionautonoma;
    tcentrosR = tcentros;
    pthread_mutex_unlock(&mutex4);

    pthread_mutex_lock(&mutex5);
    resultadoSTOPR = resultadoSTOP;
    tstopR = tstop;
    pthread_mutex_unlock(&mutex5);

        //Comprobamos si la conducción autónoma está activadada
        switch(conduccionautonomaR)
        {
        case false:
        //En caso de estar la conducción autónoma desactivada, comprobamos si la conducción asistida está activada
        switch (conduccionasistidaR)
        {
            //En caso de estar desactivada también, procederemos a controlar el vehículo mediante los resultados obtenidos
en el hilo "mando"
            case false:
            printf("--------------Control por Mando-----------------\n");
            //printf("--------------pos Motor: %f Pos Direccion: %f-----------------\n", posMotormandoR, posDireccionmandoR);
            //En caso de querer dar marcha atrás y que la última dirección del motor fuera hacia delante:
            if(posMotormandoR<0.31 && velocidad_anteriormandoR==0)
            {
                //Simulamos un punto muerto antes de dar marcha atrás
                servoMotor.setPosition(0.3);
                usleep(100000);
                servoMotor.setPosition(0.5);
                usleep(100000);
                //Anotamos que ahora la última dirección del motor es marcha atrás
```

```
            velocidad_anteriormandoR=1;
        }

        //Enviamos el comportamiento deseado al motor y al servomotor
        servoDireccion.setPosition(posDireccionmandoR);
        servoMotor.setPosition(posMotormandoR);
        break;
        case true:
        //En caso de que esté la conducción asistida activada
        printf("--------------Conducción asistida-----------------\n");
        //Activamos la condición del hilo "ObtenerImagen"
        pthread_mutex_lock(&mutexcond);
        pthread_cond_signal ( &condimagen);
        pthread_mutex_unlock(&mutexcond);

        //Esperamos a la condición "condcontrol1", que es la que proviene del hilo "centros". Siendo el único necesario para este modo de control.
        pthread_mutex_lock(&mutexcond);
        pthread_cond_wait ( &condcontrol1, &mutexcond);
        pthread_cond_wait ( &condcontrol2, &mutexcond);
        pthread_mutex_unlock(&mutexcond);


        //printf("--------------posMotorasistida:  %f  Direccionasistida:  %f-----------------\n", posMotorasistidaR, posDireccionasistidaR);

        //En caso de querer dar marcha atrás y que la última dirección del motor fuera hacia delante:
        if(posMotorasistidaR<0.31 && velocidad_anteriormandoR==0)
        {
            //Simulamos un punto muerto antes de dar marcha atrás
            servoMotor.setPosition(0.3);
            usleep(100000);
            servoMotor.setPosition(0.5);
            usleep(100000);
            //Damos marcha atras
            servoMotor.setPosition(0.3);
            usleep(200000);
            servoMotor.setPosition(0.5);
            usleep(100000);
            //Anotamos que ahora la última dirección del motor es marcha atrás
            velocidad_anteriormandoR=1;
        }

        //En caso de que la conducción asistida no requiera de actuación sobre el motor, éste será controlado con el mando
        if (posMotorasistidaR==0.5)
        {
            servoMotor.setPosition(posMotormandoR);
            usleep(100000);
            servoMotor.setPosition(0.5);
        }
        else
        {
            //En caso de que sí se requiera actuación sobre el motor, se realizará esta actuación y posteriormente se parará
            servoMotor.setPosition(posMotorasistidaR);
            usleep(100000);
            servoMotor.setPosition(0.5);
        }
        //En caso de que la conducción asistida no requiera de actuación sobre el servomotor, éste será controlado con el mando
        if (posDireccionasistidaR==0.5)
        {
        servoDireccion.setPosition(posDireccionmandoR);
        }
        else
        {
            servoDireccion.setPosition(posDireccionasistidaR);
        }


        break;
        default:
        break;
    }
    break;
        case true:
```

```
        //En caso de estar activada la conducción autónoma
        printf("---------- Control autónomo----------\n");
        //Enviamos la condición que activa el hilo "ObtenerImagen"
        pthread_mutex_lock(&mutexcond);
        pthread_cond_signal ( &condimagen);
        pthread_mutex_unlock(&mutexcond);
        //Esperamos las condiciones de control 1 y 2.
        pthread_mutex_lock(&mutexcond);
        pthread_cond_wait ( &condcontrol1, &mutexcond);
        pthread_cond_wait ( &condcontrol2, &mutexcond);
        pthread_mutex_unlock(&mutexcond);

        //Paramos el motor si detectamos la presencia de algún objeto delante
        if (resultadodistanciaR <= 20)
        {
            printf("**\n*\n*\n*\n*\n**********Se ha detectado un obstaculo a menos de 20 cm. Distancia detectada: %d **********\n*\n*\n*\n*\n", resultadodistanciaR);
            //Paramos el motor
            posMotorautonomaR=0.5;
            usleep(100000);

        }
        //Si no hay objetos delante
        else
        {
            //Comprobamos si se ha detectado alguna señal de stop
            if (resultadoSTOPR>=1 && contadorSTOP > 5)
            {
            //En caso afirmativo, paramos 5 segundos e iniciamos un contador para poder avanzar sin volver a detectar la misma señal
            printf("**\n*\n*\n*\n*\n**********Se ha detectado una señal de STOP **********\n*\n*\n*\n*\n");
            usleep(5000000);
            contadorSTOP = 0;
            }
            contadorSTOP++;
            //printf("----------pos Motor: %f pos Direccion: %f----------\n", posMotorautonomaR, posDireccionautonomaR);
            //En caso de querer dar marcha atrás y que la última dirección del motor fuera hacia delante:
            if(posMotorautonomaR ==0.3 && velocidad_anteriorautonomaR==0)
            {
            //Simulamos un punto muerto antes de dar marcha atrás
            servoMotor.setPosition(0.3);
            usleep(100000);
            servoMotor.setPosition(0.5);
            usleep(100000);
            //Damos marcha atras
            servoMotor.setPosition(0.3);
            usleep(200000);
            servoMotor.setPosition(0.5);
            usleep(100000);
            //Anotamos que ahora la última dirección del motor es marcha atrás
            velocidad_anteriorautonomaR=1;
            }
            //Enviamos el comportamiento deseado al motor y al servomotor
            servoDireccion.setPosition(posDireccionautonomaR);
            servoMotor.setPosition(posMotorautonomaR);
            usleep(100000);
            //Paramos el motor para que no siga avanzando mientras procesamos la información de las imágenes
            servoMotor.setPosition(0.5);
        }
            break;
            default:
            break;

        }
    //Anotamos el tiempo final para el que hemos acabado el proceso entero
        tfinalR = micros();
    //Calculamos la diferencia con el tiempo establecido como inicial en el hilo "ObtenerImagen"
        ttotalR = tfinalR - tinicioR;

    //Imprimimos por pantalla toda la información relevante obtenida en el proceso completo
    printf("Control: \n");
        cout << "Conducción asistida: " << conduccionasistidaR << endl;
    cout << "Conducción autónoma: " << conduccionautonomaR << endl;
        printf("Grados trayectoria deseada : %lf \n", gradoscentrosR);
        cout << "Señales de STOP detectadas: " << resultadoSTOPR << endl;
```

```
        cout << "Distancia: " << resultadodistanciaR << "cm "<< endl;

        //Incluyendo los tiempos de procesamiento

        cout << "Tiempo TOTAL: " << ttotalR << "us "<< endl;
        cout << "Tiempo cálculo de centros: " << tcentrosR << "us "<< endl;
        cout << "Tiempo obtención de imagen: " << timagenR << "us "<< endl;
        cout << "Tiempo detección STOPs: " << tstopR << "us "<< endl;
        cout << "Tiempo cálculo distancia: " << tdistanciaR << "us "<< endl;
        cout << "Tiempo comunicación con mando: " << tmandoR << "us "<< endl;

        pthread_mutex_lock(&mutex3);
        velocidad_anteriormando = velocidad_anteriormandoR;
        pthread_mutex_unlock(&mutex3);

        pthread_mutex_lock(&mutex4);
        velocidad_anteriorautonoma = velocidad_anteriorautonomaR;
        pthread_mutex_unlock(&mutex4);

        usleep(100000);
    }
}
int main (void)
{

    //Creamos hilos
    pthread_t threads1, threads2, threads3 , threads4, threads5, threads6;

    pthread_create (&threads1, NULL, MedirDistancia, NULL);
    pthread_create (&threads2, NULL, Mando, NULL);
    pthread_create (&threads3, NULL, ObtenerImagen, NULL);
    pthread_create (&threads4, NULL, Centros, NULL);
    pthread_create (&threads5, NULL, DetectarSTOP, NULL);
    pthread_create (&threads6, NULL, Control, NULL);
    pthread_join (threads1, NULL);
    pthread_join (threads2, NULL);
    pthread_join (threads3, NULL);
    pthread_join (threads4, NULL);
    pthread_join (threads5, NULL);
    pthread_join (threads6, NULL);
}
```

Código auxiliar “calibracionparametros.cpp”

```
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

using namespace cv;
using namespace std;

 int main( int argc, char** argv )
 {
    //Establecemos la captura a través de cualquier cámara
   VideoCapture cap(CV_CAP_ANY);
   //Si no se ha podido abrir la cámara
   if ( !cap.isOpened() )
   {
       cout << "No se puede abrir la cámara" << endl;
       return -1;
   }

//Creamos una ventana
namedWindow("Control", CV_WINDOW_AUTOSIZE);

//Establecemos los valores iniciales
int iLowH = 0;
int iHighH = 179;
int iLowS = 0;
int iHighS = 255;
int iLowV = 0;
int iHighV = 255;

//Creamos las barras deslizantes para ajustar los valores
cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
cvCreateTrackbar("HighH", "Control", &iHighH, 179);

cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)
cvCreateTrackbar("HighS", "Control", &iHighS, 255);

cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
cvCreateTrackbar("HighV", "Control", &iHighV, 255);

while (true)
   {
   Mat imgOriginal;
   //Tomamos una imagen
   cap.read(imgOriginal);
   //La rotamos 180 grados
   cv::rotate(imgOriginal, imgOriginal, cv::ROTATE_180);

   //Se convierte al espacio de color HSV
   Mat imgHSV;
   cvtColor(imgOriginal, imgHSV, COLOR_BGR2HSV);

   //Se binariza segun los parámetros actuales
   Mat imgThresholded;
   inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded);

   //Se muestran ambas imágenes
   imshow("Thresholded Image", imgThresholded);
   imshow("Original", imgOriginal);

   waitKey(1);
   }

  return 0;

}
```

Sistema empotrado para el control autónomo de un coche a través de las imágenes recibidas por una cámara

3. PRESUPUESTO

Índice de contenido

Índice de tablas

1. Costes

En este apartado se analizan los costes de diseño y desarrollo de este proyecto. Divididos en tres tablas, una que representa los costes materiales, una para la mano de obra y la última para la amortización de equipos. Finalmente, se calculará la cuantía total de costes y se le añadirán los gastos generales, el beneficio industrial y el IVA (Impuesto sobre el Valor Añadido) aplicables a un contrato en el sector público.

Descripción	Unidad	Cantidad	Precio (euros)	Total (euros)
Kit PiRacer	ud	1	315	315
Protoboard	ud	1	1,5	1,5
Sensor ultrasonidos	ud	1	2,6	2,6
Resistencias	ud	2	0,01	0,03
Filamento PLA	g	14	0,02	0,32
Total				319,45

Tabla 1: Costes de los materiales.

Descripción	Unidad	Cantidad	Precio (euros/unidad)	Total (euros)
Ordenador	meses	6	18,90	113,39
Monitor	meses	6	2,45	14,69
Teclado	meses	6	1,02	6,12
Ratón	meses	6	0,76	4,53
Impresora 3D	horas	2,15	0,04	0,10
Total				138,83

Tabla 2: Amortización del equipo.

Descripción	Unidad	Cantidad	Precio (euros)	Total (euros)
Ingeniero electrónico	h	500	30	15000

Tabla 3: Mano de obra.

Descripción	Unidad	Cantidad
Selección de hardware y software	h	100
Configuración software (programas, librerías…)	h	100
Implementación control remoto	h	40
Implementación procesamiento de imagen	h	40
Implementación seguimiento carretera	h	20
Implementación detección stop	h	80
Implementación evitación obstáculo	h	20
Configuración modos funcionamiento	h	80
Realización de pruebas y análisis de resultados	h	20
Total		500

Tabla 4: Desglose de la mano de obra.

2. Resumen del presupuesto

En este apartado se va a resumir los diferentes costes explicados anteriormente. Además, se le va añadir el 13% por gastos generales, 6% por beneficio industrial y 21% de IVA.

Descripción	Total (euros)
1. Coste de materiales	319,45
2. Amortización del equipo	138,83
3. Mano de obra	15.000
4. Suma de "1, 2 y 3"	15.458,28
5. Gastos generales(13% de "4")	2.009,58
6. Suma de "4 y 5"	17.467,86
7. Beneficio industrial (6% de "6")	1.048,07
8. Suma de "6 y 7"	18.515,93
9. IVA (21% de "8")	3.888,34
10. Total (suma de "8 y 9")	22.404,27

Tabla 5: Resumen del presupuesto.

El coste total del presupuesto es de 22.404,27€.

Sistema empotrado para el control autónomo de un coche a través de las imágenes recibidas por una cámara

4. PLIEGO DE CONDICIONES

Índice de contenido

1. OBJETIVO

La presente documentación tiene como objetivo la especificación de las propiedades mínimas que se deben cumplir durante el diseño y la programación de la conducción autónoma en un coche radiocontrol.

2. CONDICIONES DE LOS MATERIALES

En este apartado se especifican los principales elementos materiales necesarios para llevar a cabo el proyecto.

2.1 Ordenador

2.1.1 Hardware

Para poder desarrollar el proyecto se necesita de un ordenador cuyos componentes hardware deben cumplir estos requisitos mínimos:

- 8 GB de memoria “RAM” (“Random Access Memory”)
- Procesador Intel Core i5-8400 2.8GHz
- 500 GB (gigabyte) de almacenamiento en disco duro
- Monitor de 22 pulgadas
- Teclado
- Ratón
- Conectividad Wifi

2.1.2 Software

Para poder desarrollar el proyecto se necesita de un ordenador cuyos componentes software deben cumplir estos requisitos mínimos:

- Sistema operativo Windows 10
- Programa “VNC Viewer” instalado

2.2 Coche radiocontrol

Vehículo con tracción a las cuatro ruedas y suspensión independiente en cada una de ellas. Cuenta con diferenciales en los ejes delanteros y traseros, un motor RC380 con escobillas de carbono y engranajes metálicos. También dispone de un servomotor E6001 con 6kg/cm de torque y un controlador de velocidad (ESC) impermeable, con control bidireccional y protección de bajas tensiones. Por otro lado, posee una placa de expansión con 4 Baterías de alta capacidad 18650 (formado una tensión total de 8,4 V) y una pantalla OLED de 0,91 pulgadas.

2.3 Microordenador

Raspberry pi 4B con 4 GB de memoria RAM y una tarjeta microSD de 32 GB. Conectividad Wi-Fi (2,4GHz/5GHz), Bluetooth 5.0 y Gigabit Ethernet. Entradas y salidas entre las que encontramos 40 pines GPIO, 2 puertos USB 2.0, 2 puertos USB 3.0, CSI para cámara MIPI y 2 puertos micro-HDMI que ofrecen una resolución máxima de 4K a 60Hz.

2.4 Sensor de ultrasonidos HC-SR04

Sensor de distancia con un rango de detección de 3 cm a 4 m y una sensibilidad de 3 mm. Voltaje de funcionamiento de 5V y una corriente aproximada de 2 mA por medición.

2.5 Cámara
Cámara gran angular de 5MP, 160 grados de ángulo de captura y conector CSI.

2.6 Control remoto
Mando inalámbrico con un receptor USB. Cuenta con 17 botones y 2 joystick.

2.7 Señal de stop
Señal de stop impresa en 3D utilizando PLA (ácido poliláctico) y siguiendo el plano 002.

3. REQUISITOS
En este apartado se van a definir los requisitos que se deben cumplir en el proyecto.

3.1 Requisitos funcionales
Como requisitos relacionados con la funcionalidad del vehículo se obtienen los siguientes.

RF01: El vehículo debe soportar tres modos de conducción: control remoto, semi-autónomo y autónomo.

RF02: El vehículo debe iniciarse en el modo de conducción por control remoto.

RF03: Siempre se deberá poder cambiar entre los modos de conducción a través del mando.

RF04: En el modo de conducción por control remoto se debe poder controlar la dirección y el desplazamiento del vehículo en todo momento.

RF05: El modo de conducción semi-autónoma debe permitir el control de la dirección y el desplazamiento del vehículo por parte del usuario. Sin embargo, si el vehículo se va a salir del circuito, se corregirá su dirección automáticamente.

RF06: El modo de conducción autónoma consistirá en el seguimiento de una carretera pintada en el suelo. Además, deberá ser capaz de detectar obstáculos y reaccionar deteniéndose hasta que la presencia del obstáculo desaparezca. También, deberá detectar señales de “stop” y reaccionar deteniéndose durante 5 segundos. Posteriormente deberá seguir con un comportamiento normal, ignorando la última señal de “stop” detectada.

RF07: En los modos de conducción semi-autónoma y autónoma, si el vehículo se sale del carril, deberá retroceder hasta situarse en una posición en la que pueda continuar su funcionamiento con normalidad.

RF08: El tiempo de procesamiento en el modo de conducción por control remoto deberá ser inferior a 10 ms.

RF09: El tiempo de procesamiento en los modos de conducción semi-autónoma y autónoma deberá ser inferior a 500 ms.

3.2 Requisitos no funcionales

Como requisitos no relacionados con la funcionalidad del vehículo se obtienen los siguientes.

RNF01: La duración de la batería en cualquiera de los modos de conducción deberá ser superior a los 15 min.

RNF02: El proyecto deberá realizarse con el prototipo de vehículo tele-dirigido “PiRacer Pro AI” de la marca Waveshare.

RNF03: La programación se realizará sobre Linux empotrada en la plataforma Rasberry Pi 4B.

RNF04: La programación se realizará en el lenguaje de programación C++.