Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Autor/es

Marcos Bronchal Paricio

Director/es

Ana María López Torres
Alfonso Blesa Gascón

Escuela Universitaria Politécnica de Teruel

2017

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer la labor de mis directores, Ana María López Torres y Alfonso Blesa Gascón, así como todo su esfuerzo, dedicación y ayuda prestada, cada uno en su especialidad, que me han proporcionado para que este Trabajo Fin de Grado se haya podido llevar a cabo. Con el desarrollo del mismo, he podido evolucionar dentro de mi formación como ingeniero, adquiriendo ciertas competencias que me resultarán tremendamente útiles para el mundo laboral.

En segundo lugar, me gustaría agradecerle a mi familia, amigos y compañeros, la implicación en este proyecto. Ellos han sido quienes me han proporcionado un punto de vista diferente, que me ha ayudado a mejorar mi prototipo.

En especial, quiero agradecer la labor de dos buenos amigos, Andrés Yús Belenguer, por su amabilidad y generosidad de proporcionarme el circuito de *Scalextric*, y Ángel Peralta Romero, por la ayuda prestada en el diseño de placas de circuito impreso.

Por último, especial agradecimiento para Miguel Ángel García Monclús, oficial de laboratorio de la EUPT, por su paciencia, tiempo, consejo y trabajo en las tareas mecánicas del proyecto.

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Marcos Bronchal Paricio. Febrero, 2017

"Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*"

RESUMEN

En este Trabajo Fin de Grado se ha desarrollado el diseño y la implementación de un prototipo de radar de tramo, en el que un computador *Raspberry Pi* es el encargado de gestionar los datos necesarios para el correcto funcionamiento de la aplicación. Su implementación se ha llevado a cabo en un circuito de *Scalextric*.

El sistema calcula la velocidad media de los coches de *Scalextric* en un determinado tramo, y en función de si superan un cierto valor de velocidad, decide si se trata de un infractor. En caso afirmativo, el sistema realiza una serie de fotografías mediante una *PiCamera* para identificar su matrícula, que es comparada con una base de datos para redactar la correspondiente multa.

Palabras clave: radar, *Raspberry Pi*, *PiCamera*, *Python*, *Scalextric*.

"Design and implementation of a stretch radar prototype based on Raspberry Pi"

ABSTRACT

This work describes the design of a stretch radar and its implementation on a prototype. A Raspberry Pi computer manages the data necessary for the correct operation of the application. Its implementation has been carried out in a Scalextric circuit.

The system calculates the average speed of Scalextric cars in a specific stretch, and depending on whether it exceeds a predefined value, it decides if these vehicles have committed a traffic offence. If so, the system takes a series of photographs using a PiCamera to identify its number plate. The corresponding fine is written using the data associated to this number. This information is obtained from a database.

Keywords: radar, Raspberry Pi, PiCamera, Python, Scalextric.

Marcos Bronchal Paricio. Febrero, 2017

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Marcos Bronchal Paricio. Febrero, 2017

ÍNDICE GENERAL

PRIMERA PARTE.
LISTADOS

Marcos Bronchal Paricio. Febrero, 2017

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Marcos Bronchal Paricio. Febrero, 2017

Lista de figuras

Lista de tablas

Lista de siglas y abreviaturas

ASCII	*American Standard Code for Information Interchange*
AWB	*Automatic White Balance*
BACK	*Backlight*
BCM	*Broadcom*
BUZZ	*Buzzer*
C	Condensador
CON	*Control Voltage*
CONTR	Contraste
CSI	*Camera Serial Interface*
D	*Data*
DGT	Dirección General de Tráfico
DIS	*Discharge*
DSI	*Display Serial Interface*
Dst	*Destination*
E	*Enable*
f	Frecuencia
fo	Frecuencia de salida
Fps	*Frames per second*
GND	*Ground*
GPIO	*General Purpose Input/Output*
HD	*High Definition*
HDMI	*High-Definition Multimedia Interface*
IP	*Internet Protocol*
IR	Radiación Infrarroja
ISO	*International Organization for Standardization*
I2C	*Inter-Integrated Circuit*
J	*Jack*

kHz	kilohercios
kΩ	kiloohmios
LAN	*Local Area Network*
LCD	*Liquid Crystal Display*
LED	*Light-Emitting Diode*
Ln	Logaritmo neperiano
m	Metros
MB	*Megabyte*
mm	Milímetros
Mpx	*Megapixel*
Ms	Milisegundos
mV	Milivoltios
nF	Nanofaradios
OCR	*Optical Character Recognition*
OS	*Operating System*
OUT	*Output*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PDF	*Portable Document Format*
PIL	*Python Imaging Library*
POT	Potenciómetro
Px	*Pixel*
R	Resistencia
RAM	*Random Access Memory*
RCA	*Radio Corporation of America*
RES	*Reset*
RGB	*Red Green Blue*
ROI	*Region Of Interest*
RS	*Registry*

R/W	*Read/Write*
SD	*Secure Digital*
Seg	Segundos
SoC	*System on Chip*
Src	*Source*
TRES	*Threshold*
TRI	*Trigger*
UK	*United Kingdom*
USB	*Universal Serial Bus*
V	Voltios
VCC	*Power Supply*
VO	*Output Voltage*
Vs	*Versus*
Ω	Ohmios
µF	Microfaradios

SEGUNDA PARTE.
MEMORIA

Marcos Bronchal Paricio. Febrero, 2017

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Marcos Bronchal Paricio. Febrero, 2017

1. Introducción

El exceso de velocidad es una de las causas más importantes de la siniestralidad en las carreteras de nuestro país. Por ello, la DGT se esfuerza cada año para reducirlo en la medida de lo posible. A pesar de las restricciones de velocidad de las señalizaciones, muchos conductores hacen caso omiso de ellas, y deciden poner en riesgo su vida y la de los demás conductores. Es por ello que la DGT se ha visto obligada a instalar radares para el control de velocidad.

Existen multitud de tipos de radares[1], pero en una primera clasificación general, se pueden diferenciar radares fijos y radares móviles. Los primeros de ellos, son radares que están colocados en una posición fija, en un poste, pórtico o dentro de una cabina, y deben estar siempre señalizados. A diferencia de estos, los radares móviles suelen estar camuflados en diferentes puntos, o incluso instalados dentro de un coche o helicóptero, para intentar sorprender a los conductores infractores.

A pesar de que algunos funcionan mediante láser, el funcionamiento de la mayoría de radares se basa en el efecto *Doppler*[2]. Este fenómeno físico calcula la velocidad de un determinado vehículo mediante la diferencia de frecuencias, entre la emitida (radar) y la recibida (vehículo). Así pues, el radar emite ondas electromagnéticas a una determinada frecuencia que rebotan en el vehículo y vuelven a ser recibidas por el radar.

Todos los dispositivos anteriores calculan velocidades instantáneas, es decir, velocidades en un determinado instante de tiempo. Sin embargo, los denominados radares de tramo, instalados recientemente en muchos puntos de las carreteras de nuestro país, calculan la velocidad media de un vehículo en un determinado tramo. Su funcionamiento es mucho más simple que el de los radares convencionales, puesto que las únicas variables que manejan son el espacio y el tiempo.

Figura 1: Escenificación de un radar de tramo (http://www.antena3.com)

Como se puede observar en la *Figura 1*, conociendo la distancia de un determinado tramo delimitado por dos puntos, estos dispositivos calculan el tiempo que un vehículo tarda en recorrer dicha longitud para calcular su velocidad media mediante el cociente espacio-tiempo.

1.1. Objetivos

1.1.1. Objetivo principal

El objetivo principal de este Trabajo Fin de Grado es el diseño y la implementación de un sistema capaz de identificar automáticamente a aquellos vehículos que superen un valor de velocidad. El diseño se concreta en un circuito de *Scalextric*, en el que un conjunto de detectores de posición proporciona la información necesaria a una *Raspberry Pi* para determinar la velocidad, y las imágenes capturadas por una cámara digital permiten identificar al vehículo reconociendo los dígitos de su matrícula.

1.1.2. Objetivos específicos

1. Configuración del sistema operativo de la *Raspberry Pi* (*Raspbian*) y puesta en marcha de la *PiCamera*.
2. Búsqueda y selección de los sensores y periféricos más adecuados para el correcto funcionamiento de la aplicación.
3. Diseño de circuitos electrónicos e implementación en *protoboard*.
4. Elección del lenguaje de programación y desarrollo de las primeras líneas de código.
5. Instalación y configuración de las diferentes librerías necesarias en la *Raspberry Pi*.
6. Construcción de soportes para el posicionamiento de sensores y periféricos en la maqueta.
7. Caracterización del escenario de toma de fotografías. Posicionamiento de la cámara, parámetros de captura e iluminación.
8. Tratamiento de la información recibida de los sensores para el correcto funcionamiento automático de la aplicación.
9. Fabricación de las placas de circuito impreso de los diseños electrónicos definitivos de sensores y periféricos.
10. Colocación y cableado de las PCBs en sus correspondientes soportes de la maqueta.
11. Desarrollo del código.
12. Calibración del sistema y corrección de errores.

2. Diseño *hardware*

A continuación, se detalla la elección e implementación de todos los sensores y periféricos del prototipo que proporcionan la información necesaria al sistema para el correcto funcionamiento del mismo.

Por su bajo coste, facilidad de interacción de entradas-salidas y cierto conocimiento previo en la materia, se ha optado por una *Raspberry Pi* como sistema de gestión y procesamiento de la información del prototipo.

Puesto que se desea calcular la velocidad media de un coche de *Scalextric* en un determinado tramo, dicho tramo deberá estar perfectamente delimitado. Además de esto, se deberá detectar con precisión el paso del coche por los dos puntos que delimitan el tramo. Por su relativa sencillez y bajo coste, se ha optado por la implementación de dos parejas emisor-receptor IR perfectamente enfrentadas.

Para la identificación de matrículas es indispensable disponer de un dispositivo capaz de capturar imágenes con relativa rapidez, puesto que se deben realizar capturas en movimiento. Para solventar esta necesidad, y por temas de compatibilidad con *Raspberry Pi*, se ha seleccionado una *PiCamera*.

Para que la identificación de matrículas sea correcta, las imágenes deberán ser tomadas con la iluminación adecuada. Por su facilidad de orientación, el sistema de iluminación seleccionado ha sido un módulo orientable de ojo de buey, y una lámpara flexible USB de 28 LEDs.

Se ha implementado una LCD para la simulación de los paneles informativos que existen en las carreteras. El mensaje de la LCD cambia de manera periódica, el primer mensaje muestra un aviso de la existencia del radar de tramo, mientras que el segundo mensaje informa sobre la hora y la fecha actual. Se ha optado por este dispositivo porque su tamaño es acorde con el del prototipo y su implementación es relativamente sencilla.

Para que el usuario sea consciente de la infracción, es necesario que el sistema responda ante el paso de un coche que sobrepase el umbral de velocidad fijado. Por disponibilidad y bajo coste, se ha implementado un LED que emite una señal óptica, y un zumbador (BUZZ) que emite una señal acústica.

Todos los dispositivos han sido fijados en soportes de aglomerado. Se ha optado por este material en la fabricación de los soportes por su fácil manejo y bajo coste. Por temas de estética, el aglomerado es blanco y canteado de 16 mm de grosor.

Un esquema básico de conexión de todos los sensores y periféricos descritos anteriormente puede observarse en la *Figura 2.*

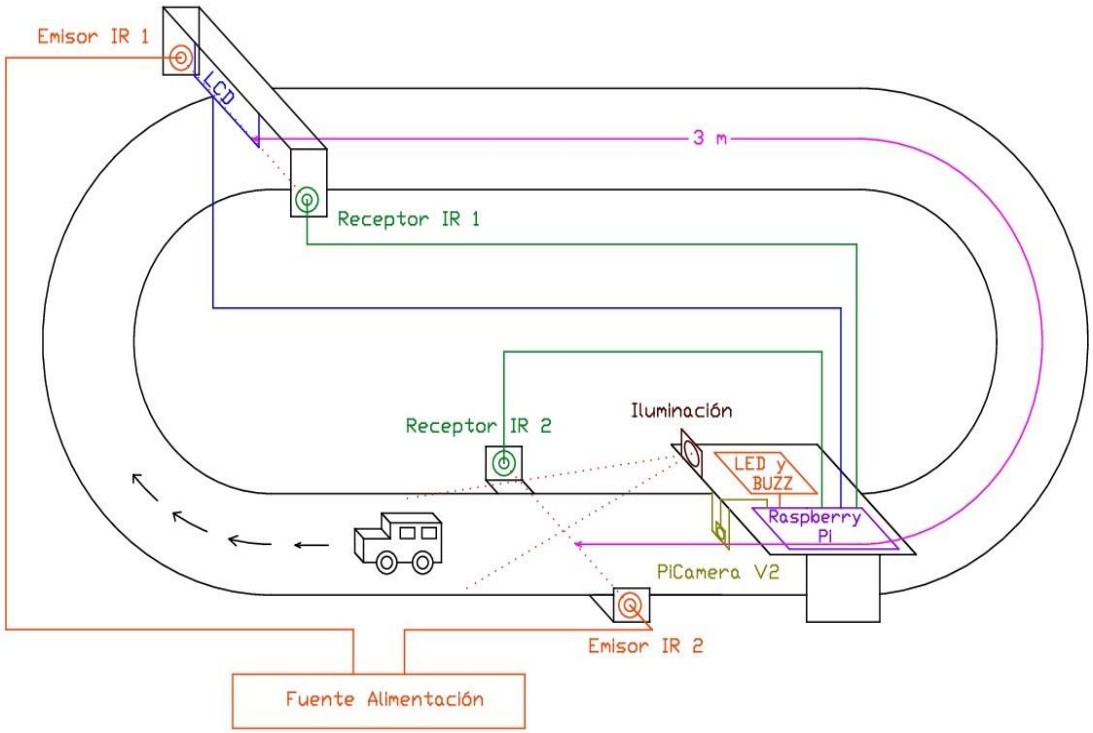

Figura 2: Esquema general del prototipo

En la figura anterior se observa claramente las dos parejas emisor-receptor IR que delimitan el tramo de interés, cuya longitud es de 3 m. Las líneas discontinuas muestran la perfecta alineación del emisor con su correspondiente receptor IR para que la lectura del paso de un vehículo sea correcta.

En el soporte en el que está fijada la primera pareja de sensores (emisor IR 1 y receptor IR 1) se ha colgado la LCD a una distancia acorde para que el coche no impacte en ella tras su paso por la misma.

Como es lógico, la cámara se encuentra detrás de la segunda pareja de sensores (emisor IR 2 y receptor IR 2), para que sea capaz de capturar imágenes del vehículo si su velocidad es inadecuada. La cámara se ha colgado en este soporte a una altura acorde para que el coche no impacte en ella tras su paso por la misma. Para que el encuadre y enfoque del coche sea el correcto, también existe una cierta distancia entre la cámara y la segunda pareja de sensores.

El posicionamiento del sistema de iluminación permite iluminar correctamente al coche cuando pasa por la segunda pareja de sensores.

En la *Figura 3* se muestra el esquema electrónico general del prototipo, en el que se puede observar todas las conexiones de los diferentes sensores y periféricos. A continuación, se detallan cada uno de ellos.

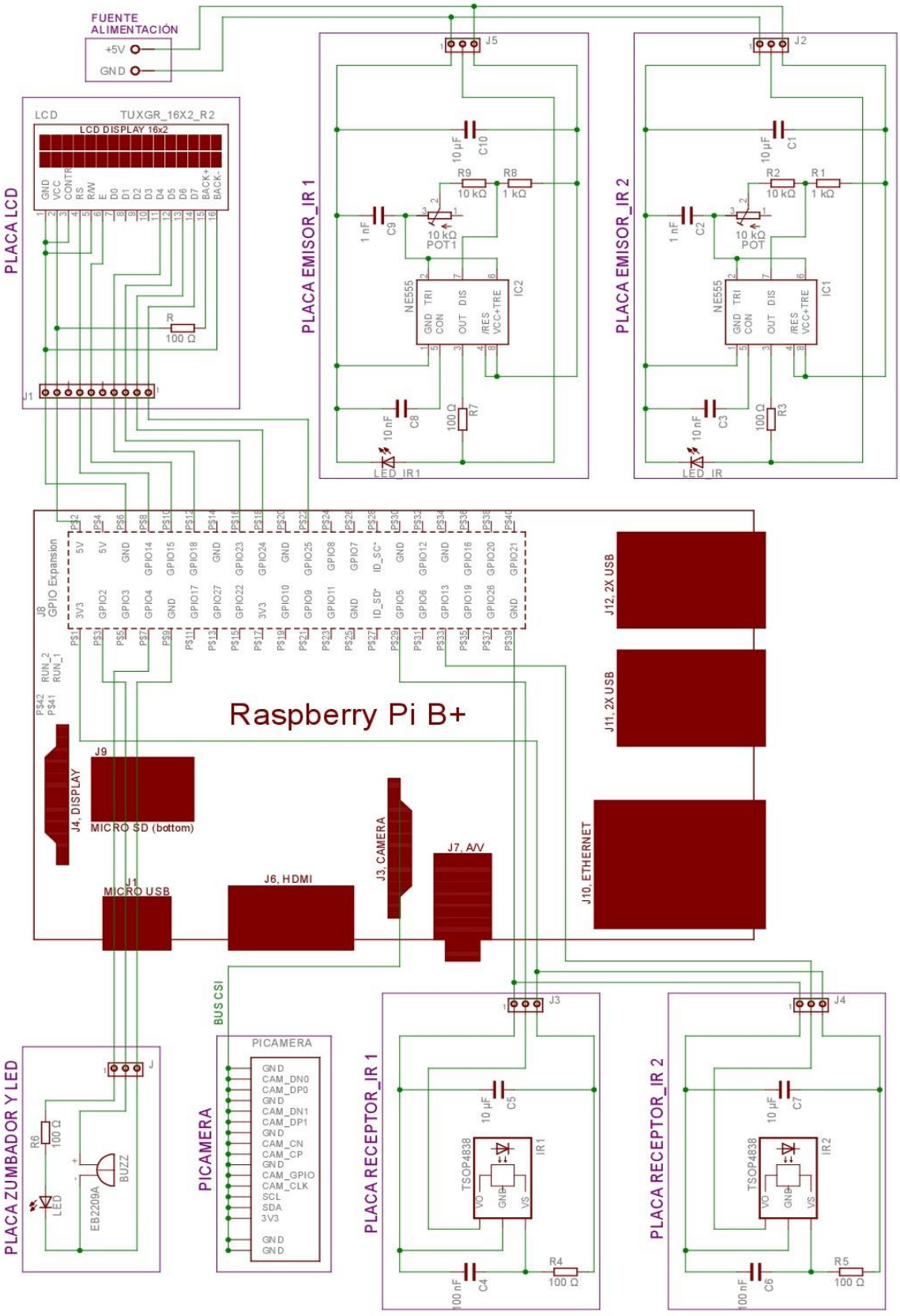

Figura 3: Esquema electrónico general del prototipo

2.1. *Raspberry Pi*

Raspberry Pi[3], [4] es un computador de placa reducida de bajo coste desarrollado en UK por la Fundación *Raspberry Pi*. Actualmente, en el mercado existen varios modelos, aunque para este proyecto se ha utilizado *Raspberry Pi Model B+*, pues dispone de más GPIOs y mejor gestión de consumo que modelos anteriores. Entre sus especificaciones, destaca el procesador BCM 2835 SoC, y los 512 MB de RAM.

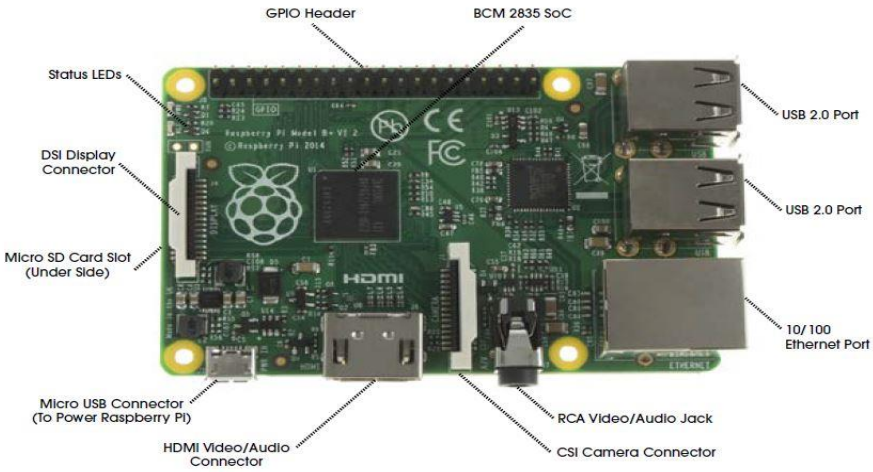

Figura 4: Partes de Raspberry Pi Model B+

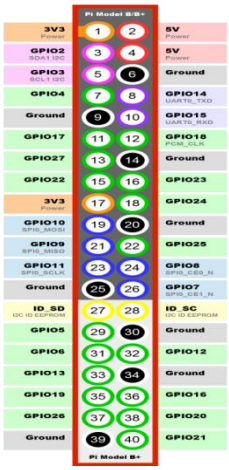

Figura 5: GPIOs de Raspberry Pi Model B+

Como se puede observar en la *Figura 4*, dicho modelo dispone de 4 puertos USB (de los cuales se han utilizado tres, uno para el ratón, otro para el teclado, y otro para la iluminación), un puerto de *Ethernet* (para la comunicación con otro PC, y la obtención de la fecha y la hora de la red, puesto que no dispone por defecto de reloj en tiempo real), una salida HDMI (para la visualización por monitor), un puerto micro-USB (para alimentar la placa), 40 GPIOs (para conectar los dos sensores de posición, la LCD, un zumbador y un LED), un conector CSI (para la *PiCamera*), y una ranura para insertar una tarjeta micro SD (instalación del sistema operativo *Raspbian* y de diferentes librerías necesarias para el funcionamiento de la aplicación). En la *Figura 5* se ilustra la disposición de pines y de GPIOs de dicho modelo. Los *datasheet* de esta *Raspberry Pi* y de su procesador pueden consultarse en el *ANEXO 1*.

El sistema operativo seleccionado para la *Raspberry Pi* ha sido *Raspbian*, pues al tratarse de una distribución *Linux*, es de *software* libre. Como cualquier otra distribución *Linux*, y para una mayor comodidad, se ha optado por el acceso remoto a la aplicación con un PC mediante *Putty*. Se ha utilizado una dirección IP pública, pero perfectamente podría haberse utilizado una dirección IP privada en una LAN. Para la visualización en el PC de la organización de directorios de la *Raspberry Pi* se ha hecho uso del protocolo *Samba*. Pese al acceso remoto, todo el procesamiento es llevado a cabo íntegramente por la *Raspberry Pi*.

2.2. *PiCamera*

En este proyecto se ha trabajado con una *PiCamera V2*[5]. Se trata de una cámara HD compatible con todos los modelos de *Raspberry Pi*, que se ha utilizado para la captura de fotografías de vehículos infractores. Dispone de un sensor *Sony* IMX219, y es capaz de capturar fotografías de 8 Mpx de resolución. El módulo *PiCamera V2* puede visualizarse en la *Figura 6*, y sus modos de funcionamiento en la *Tabla 1*[6]. El *datasheet* de esta *PiCamera* puede consultarse en el *ANEXO 1*.

La banda CSI se conecta al conector CSI de la *Raspberry Pi*, tal y como se muestra en la *Figura 3*.

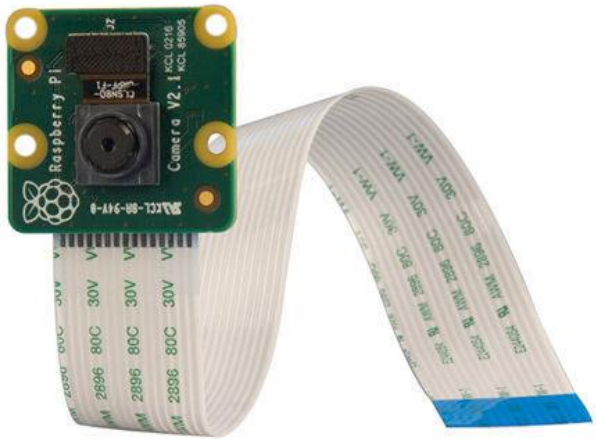

Figura 6: PiCamera V2

#	Resolución (Px)	Relación de aspecto	Tasa Fps	Video	Imagen
1	1920 x 1080	16:9	0.1 - 30	x	
2	3280 x 2464	4:3	0.1 - 15	x	x
3	3280 x 2464	4:3	0.1 - 15	x	x
4	1640 x 1232	4:3	0.1 - 40	x	
5	1640 x 922	16:9	0.1 - 40	x	
6	1280 x 720	16:9	40 - 90	x	
7	640 x 480	4:3	40 - 90	x	

Tabla 1: Modos de funcionamiento de PiCamera V2

2.2.1. Parámetros de configuración

En esta cámara pueden ser configurados infinidad de parámetros. A continuación, se detallan solamente los parámetros que se han configurado para la aplicación:

- Nitidez[7]: claridad de los detalles de la fotografía. Una buena nitidez se consigue con una iluminación adecuada, y un buen enfoque. En esta cámara, la nitidez se expresa como un número entero entre -100 y 100. Por defecto es 0.
- Contraste[8], [9]: diferencia relativa de la intensidad entre píxeles de la imagen. En esta cámara, el contraste se expresa como un número entero entre -100 y 100. Por defecto es 0.
- Brillo[9], [10]: altera la gama tonal. Su modificación en exceso puede reducir el contraste de la imagen. En esta cámara, el brillo se expresa como un número entero entre 0 y 100. Por defecto es 50.
- Saturación[9], [11]: mayor o menor pureza del color. Está asociada a la cantidad de blanco que se añade al tono dominante. En esta cámara, la saturación se expresa como un número entero entre -100 y 100. Por defecto es 0.
- Sensibilidad ISO[9], [10]: determina la cantidad de energía que necesita la cámara para realizar una fotografía. Con el incremento de este parámetro se favorece la aparición de ruido, pero es recomendable en condiciones de poca iluminación. En esta cámara, la sensibilidad ISO se expresa como un número entero entre 0 y 1600. Por defecto es 0.
- Resolución[12]: indica la cantidad de detalles que pueden observarse en la fotografía. Se expresa en píxeles. Las diferentes resoluciones de esta cámara se pueden observar en la *Tabla 1,* con sus correspondientes relaciones de aspecto.
- Rotación vertical[9]: permite rotar la imagen verticalmente. En esta cámara, su valor puede ser verdadero (habilitar rotación) o falso (deshabilitar rotación). Su valor por defecto es falso.
- Rotación horizontal[9]: permite rotar la imagen horizontalmente. En esta cámara, su valor puede ser verdadero (habilitar rotación) o falso (deshabilitar rotación). Su valor por defecto es falso.
- Velocidad de fotogramas: indica la tasa de fps que puede capturar el dispositivo. Los valores máximos de esta cámara aparecen en la *Tabla 1*, y en ella se observa que esta tasa puede ser mayor cuanto menor sea la resolución.

Tras la realización de varios ensayos con diferentes posicionamientos de la cámara y la iluminación, se ha optado por configurar la *PiCamera* con los parámetros que aparecen en la *Tabla 2*.

Parámetro	Configuración	Parámetro	Configuración
Resolución	1280 x 720 Px	Brillo	50 %
Velocidad fotogramas	90 fps	Contraste	100 %
Rotación vertical	Verdadero	Saturación	0 %
Rotación horizontal	Verdadero	Sensibilidad ISO	1600 ISO

Tabla 2: Parámetros de configuración fijados en PiCamera V2

2.3. Sensores de posición

Como se ha explicado anteriormente, para calcular la velocidad media de los coches de *Scalextric* en un determinado tramo, se necesita tener dicho tramo perfectamente delimitado. Los dispositivos encargados de realizar esta tarea son dos parejas de emisor-receptor IR.

El emisor IR puede observarse en la *Figura 7*, mientras que el receptor IR puede observarse en la *Figura 8*.

Figura 7: LED IR Kodenshi

Figura 8: Receptor IR Vishay TSOP4838 38 kHz

Así pues, equipados con la circuitería electrónica correspondiente que posteriormente se detalla, el emisor y el receptor IR se encuentran perfectamente enfrentados, cada uno en un lado de la pista de *Scalextric*, creando lo que comúnmente se denomina una "barrera infrarroja", para poder detectar el paso de un coche.

Las dos parejas de emisor-receptor IR se encuentran separadas en el circuito de *Scalextric* una distancia de 3 m, por lo que la *Raspberry Pi* calculará el tiempo que le cuesta a un coche completar dicho tramo para calcular su velocidad media.

2.3.1. Receptor IR

El comportamiento de este sensor puede apreciarse en la *Figura 9*. Su salida tiene valor alto, es decir, 1 lógico (voltaje de alimentación), cuando el receptor no recibe IR. Cuando el receptor es excitado por IR con forma de onda cuadrada de 0 a 5 V, de 38 kHz de frecuencia, y con un ciclo de trabajo del 50%, su salida tiene valor bajo, es decir, 0 lógico (0 V).

El resto de información y características sobre este receptor IR puede consultarse en el *ANEXO 1*.

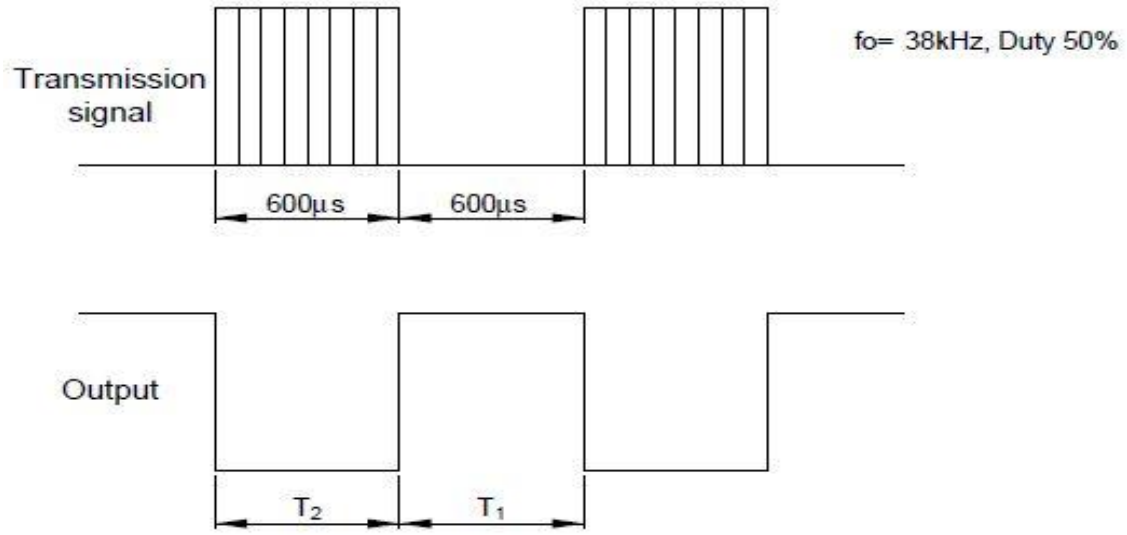

Figura 9: Comportamiento del receptor IR

Para mejorar la robustez del sensor, se ha diseñado la circuitería electrónica que se muestra en la *Figura 10*, que se integrará en una PCB. El circuito es alimentado por 3.3 V que son proporcionados por la *Raspberry Pi*, para que la salida VO sea de 0 V (en el caso de que reciba IR con las características descritas anteriormente) o 3.3 V (en el caso de que no reciba IR, es decir, cuando un coche atraviesa la barrera IR). Se ha colocado un condensador de filtrado C1.

El borne VO de los receptores IR 1 y 2 se conecta a GPIO5 y a GPIO13 de la *Raspberry Pi* respectivamente, tal y como se muestra en la *Figura 3*.

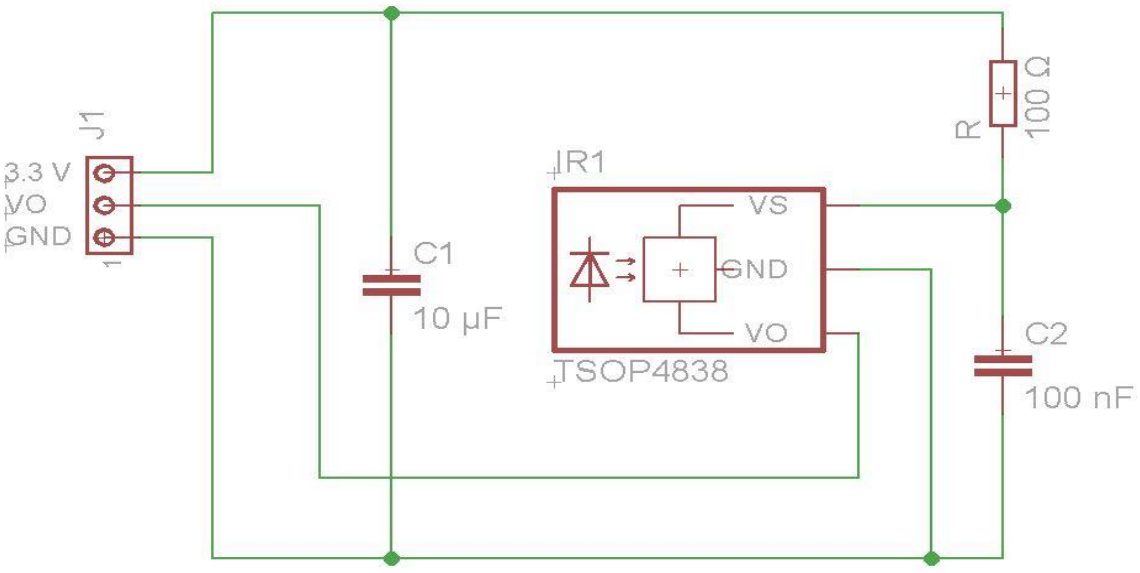

Figura 10: Esquema electrónico del receptor IR

2.3.2. Emisor IR

Para conseguir la señal de transmisión IR que se muestra en la *Figura 9*, se ha diseñado la circuitería electrónica que aparece en la *Figura 11*, en la que un circuito integrado 555 configurado como astable[13] es el encargado de proporcionar la excitación al LED IR.

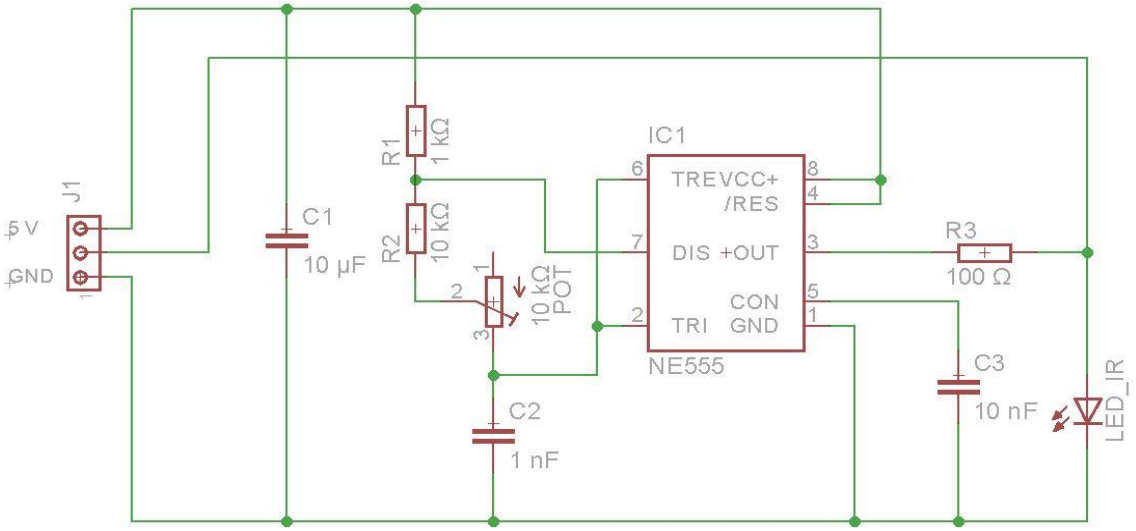

Figura 11: Esquema electrónico del emisor IR

El circuito se implementará en una PCB, que será alimentada con 5 V de una fuente de alimentación externa, tal y como se muestra en la *Figura 3,* con el objetivo de aislar las masas del emisor y el receptor IR. Se ha colocado un condensador C1 de filtrado, y el valor del resto de componentes del circuito determinan la frecuencia de excitación al LED IR.

Debido a la tolerancia de los componentes, tanto en el circuito emisor IR como en el circuito receptor IR, se ha colocado un potenciómetro POT en serie con la resistencia R2 para poder variar la frecuencia entre f1 y f2. Así pues, una vez enfrentados emisor y receptor IR, se ajustará manualmente el potenciómetro para que el comportamiento del receptor IR sea el deseado.

La frecuencia de transmisión viene determinada por la siguiente ecuación:

$$f = \frac{1}{\ln(2) \cdot C2 \cdot [R1 + 2 \cdot (R2 + POT)]} = \frac{1}{\ln(2) \cdot (1\ nF) \cdot [1\ \text{k}\Omega + 2 \cdot (10\ \text{k}\Omega + POT)]}$$

Si POT = 10 kΩ, la frecuencia es $f1 \approx 35.2\ kHz$
Si POT = 0 kΩ, la frecuencia es $f2 \approx 68.7\ kHz$

El resto de características del circuito integrado 555 y del LED IR pueden consultarse en el *ANEXO 1*.

2.4. LCD

Se ha conectado una LCD de 2 líneas y 16 caracteres por línea, tal y como se muestra en la *Figura 12*, para la visualización de algunos mensajes de interés.

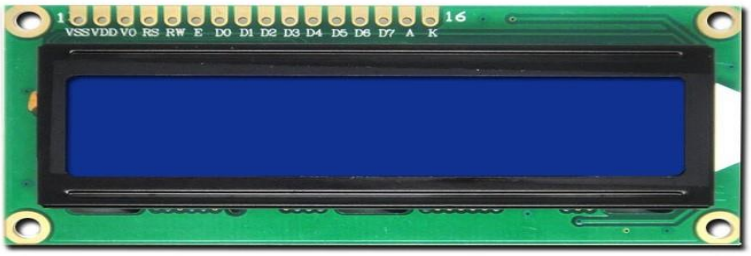

Figura 12: LCD 16x2 ERM1602-6

El *datasheet* de esta LCD puede consultarse en el *ANEXO 1*.

Para un mejor manejo de la misma, se ha realizado el diseño electrónico que se muestra en la *Figura 13,* que se implementará en una PCB.

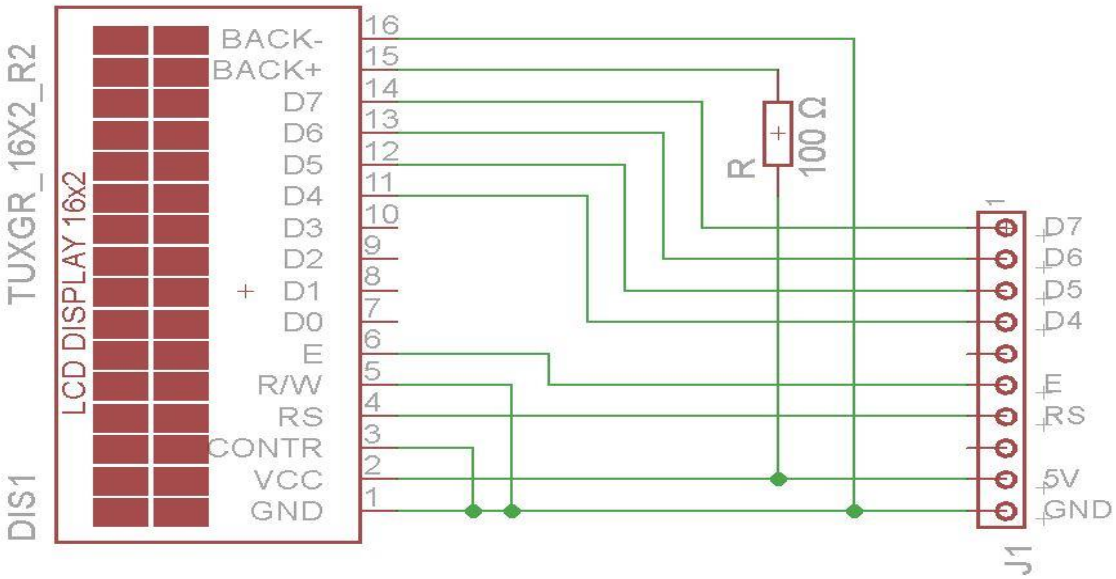

Figura 13: Esquema electrónico de LCD

Esta placa será alimentada con 5 V proporcionados por la *Raspberry Pi*. La función de la resistencia R es limitar la intensidad de la retroiluminación de la LCD. Las conexiones a la *Raspberry Pi*, que pueden observarse en la *Figura 3*, son las siguientes:

D7: GPIO25
D6: GPIO24
D5: GPIO23
D4: GPIO18
E: GPIO15
RS: GPIO14

2.5. Emisor óptico y acústico

Cuando la velocidad de un determinado coche supera el umbral de velocidad estipulado (8 m/s), el sistema emite una señal óptica a través de un LED azul, y una señal acústica a través de un zumbador. El primer dispositivo puede apreciarse en la *Figura 14*, y el segundo, en la *Figura 15*.

Figura 14: LED Kingbright azul

Figura 15: Zumbador piezoeléctrico Kingstate KPEG242

Los *datasheet* del LED y del zumbador pueden consultarse en el *ANEXO 1*.

Se ha diseñado un circuito electrónico, tal y como se muestra en la *Figura 16*, para conectar ambos dispositivos, que será implementado en una PCB.

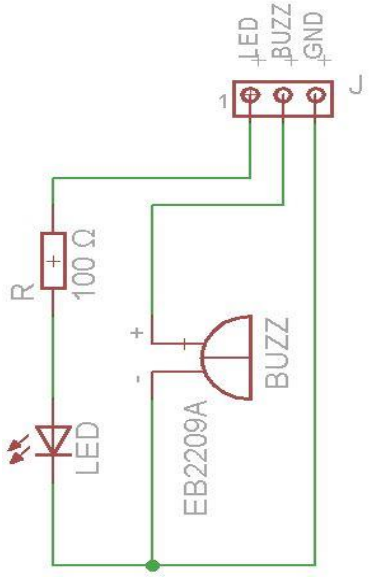

Figura 16: Esquema electrónico de LED y BUZZ

La función de la resistencia R es limitar la intensidad en el LED. El borne LED se conecta a la GPIO4 de la *Raspberry Pi*, y el borne BUZZ a la GPIO2, tal y como se muestra en la *Figura 3.*

3. Diseño *software*

3.1. Elección del lenguaje de programación: *Python* Vs. *C++*

Debido a las recomendaciones de los repositorios oficiales de *Raspberry Pi* en cuanto a la *PiCamera*, y a la poca documentación acerca de librerías en *C++* de la *PiCamera*, se ha decidido desarrollar la aplicación en lenguaje *Python*, con las ventajas e inconvenientes que ello conlleva.

En primer lugar, cabe destacar que *Python* es un lenguaje de programación interpretado, es decir, que realiza la traducción a lenguaje máquina a medida que se necesita, a diferencia de *C++*. Esta característica de *Python* hará que se reduzca el rendimiento del sistema en cuanto a velocidad de procesamiento. Sin embargo, tras la realización de varios ensayos, se ha considerado que el rendimiento del prototipo es aceptable.

Por otro lado, cabe destacar algunas de las ventajas que ofrece la programación en *Python* frente a la programación en *C++*, como es la mayor legibilidad del código y la menor extensión del mismo. La razón principal de estas dos ventajas radica en que *Python* es un lenguaje de más alto nivel que *C++*.

3.2. Librerías utilizadas

Para el desarrollo de la aplicación, se ha hecho uso de las diferentes librerías que se describen a continuación.

3.2.1. *Numpy*

Esta librería permite trabajar con matrices, y puesto que una imagen no es más que una matriz de valores de intensidad, es fundamental para el desarrollo de la aplicación. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 3*[14].

Función	Parámetros	Descripción
ones	*matrix* = np.ones(*shape*)	Devuelve una matriz de unos de tamaño *shape* y la guarda en la variable *matrix.*

Tabla 3: Funciones utilizadas de Numpy

3.2.2. *OpenCV*

OpenCV[15] es una librería libre originalmente desarrollada por *Intel*, que contiene funciones para el procesado de imagen. En este proyecto, dicha librería ha sido utilizada para el procesado de las imágenes capturadas por la *PiCamera*. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 4*[16].

Función	Parámetros	Descripción
imread	*img = cv2.imread(filename, flags)*	Lee una imagen desde un archivo *filename*, y la guarda en la variable *img*. El segundo parámetro indica el tipo de color de la imagen, los más comunes son: - CV_LOAD_IMAGE_COLOR: color. - CV_LOAD_IMAGE_GRAYSCALE: grises.
imwrite	cv2.imwrite*(filename, img)*	Guarda una imagen *img* en un archivo *filename.*
cvtColor	*dst* = cv2.cvtColor*(src, flags)*	Convierte una imagen *src* de un espacio de color a otro, y guarda esa conversión en la variable *dst*. El segundo parámetro indica el tipo de conversión, los más comunes son: - cv2.COLOR_BGR2GRAY: de RGB a gris. - cv2.COLOR_GRAY2BGR: de gris a RGB.
threshold	*ret, dst =* cv2.threshold*(src, thresh, maxval, type)*	Guarda en la variable *dst* la binarización de una imagen *src*. Los píxeles de intensidad mayor a *thresh* toman el valor de intensidad *maxval*. El parámetro *type* indica el tipo de binarización, las más comunes son: - cv2.THRESH_BINARY: simple. - cv2.THRESH_OTSU: Otsu.
erode	*dst* = cv2.erode*(src, element, iterations)*	Guarda en la variable *dst* la erosión de la imagen *src* mediante una máscara *element*, un determinado número de iteraciones *iterations.*
findContours	*contours, hierarchy =* cv2.findContours*(scr, mode, method)*	Guarda en la variable *contours* los contornos de una imagen *src* utilizando un modo *mode*, y un método *method*. El modo más común es: - cv2.RETR_TREE El método más común es: - cv2.CHAIN_APPROX_SIMPLE

contourArea	*area* = cv2.contourArea*(contour)*	Guarda en la variable *area* el área encerrada en un determinado contorno *contour.*
boundingRect	*x, y, w, h* = cv2.boundingRect*(cnt)*	Guarda en las variables *x, y, w, h* las coordenadas del rectángulo mínimo que encierra a un determinado contorno *cnt.* - *x, y*: coordenadas de los píxeles de la esquina superior izquierda del rectángulo. - *w*: largo del rectángulo en píxeles. - *h*: alto del rectángulo en píxeles.

Tabla 4: Funciones utilizadas de OpenCV

3.2.3. PIL

PIL es otra librería asociada a labores de procesado de imágenes. En este proyecto, dicha librería se ha utilizado para leer las imágenes procesadas por *OpenCV* para que pueda realizarse sobre ellas un posterior reconocimiento óptico de caracteres. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 5*[17].

Función	Parámetros	Descripción
Image.open	*img = PIL.Image.open(filename)*	Lee una imagen desde un archivo *filename*, y la guarda en la variable *img.*

Tabla 5: Funciones utilizadas de PIL

3.2.4. *Time*

Permite la gestión del tiempo en *Python*. Dicha librería ha sido utilizada en este proyecto para calcular el tiempo que le cuesta a un coche recorrer el tramo de interés, así como para la obtención de la hora y la fecha actual, para su posterior visualización en la LCD. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 6*[18].

Función	Parámetros	Descripción
strftime	time.strftim("%H:%M %d/%m/%y*")*	Obtiene la fecha y hora actual en el formato hora:minutos día/mes/año.
sleep	time.sleep(*seg*)	Retardo de *seg* segundos.
time	*seg* = time.time	Almacena en la variable *seg* los segundos transcurridos desde la época del 1 de enero de 1970.

Tabla 6: Funciones utilizadas de Time

3.2.5. *Picamera*

Es la librería recomendada en los repositos oficiales de *Raspberry Pi* para configurar los parámetros de *PiCamera*, y realizar capturas de fotografías y vídeos. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 7*[6].

Función	Parámetros	Descripción
resolution	camera.resolution = *resolution*	Establece una resolución *resolution* en la cámara.
framerate	camera.framerate = *frames*	Establece los fps *frames* en la cámara.
vflip	camera.vflip = *param*	Permite la rotación vertical de la cámara. El parámetro *param* puede ser: - True: rotación habilitada. - False: rotación deshabilitada.
hflip	camera.hflip = *param*	Permite la rotación horizontal de la cámara. El parámetro *param* puede ser: - True: rotación habilitada. - False: rotación deshabilitada.
iso	camera.iso = *iso*	Establece el valor de ISO *iso* en la cámara.
contrast	camera.contrast = *contrast*	Establece un contraste *contrast* en la cámara.
brightness	camera.brightness = *brightness*	Establece un brillo *brightness* en la cámara.
saturation	camera.saturation = *saturation*	Establece una saturación *saturation* en la cámara.
start_preview	camera.start_preview()	Vista previa de la cámara por el puerto HDMI de la *Raspberry Pi*.
awb_mode	camera.awb_mode = *mode*	Establece el modo *mode* de las ganancias de balance de blancos de la cámara. Normalmente *mode* es 'auto'.
exposure_mode	camera.exposure_mode = *mode*	Establece el modo de exposición *mode* de la cámara. Normalmente *mode* es 'auto'.
capture_sequence	camera.capture_sequence (*filenames, use_video_port*)	Captura una secuencia de imágenes consecutiva y las guarda en los archivos *filenames*. *Use_video_port = True* para realizar secuencias rápidas.

Tabla 7: Funciones utilizadas de Picamera

3.2.6. *Pytesseract*

Esta librería está basada en *Tesseract*[19], que es un motor OCR libre desarrollado actualmente por *Google*, que está catalogado como uno de los motores OCR más potentes hoy en día.

Tesseract realiza el reconocimiento óptico de caracteres de una determinada imagen (binaria a poder ser), mediante la comparación de sus características (geométricas por lo general). Así pues, *Tesseract* extrae las características de una determinada imagen, y las compara con las características de las plantillas de los caracteres del código ASCII. Para el reconocimiento de palabras completas, *Tesseract* tiene en cuenta la estadística en cuanto a palabras más frecuentes, o la distinción entre mayúsculas y minúsculas.

Pueden producirse errores en el reconocimiento por la distancia entre caracteres o la conexión de algunos píxeles de diferentes caracteres.

Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 8*[20].

Función	Parámetros	Descripción
image_to_string	*string* = pytesseract.image_to_string*(img)*	Realiza el reconocimiento óptico de caracteres de una imagen *img*, y los resultados son guardados en una cadena de texto *string.*

Tabla 8: Funciones utilizadas de Pytesseract

3.2.7. *Shutil*

Esta librería se ha utilizado para la gestión de directorios, y para cambiar archivos de directorio. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 9*[21].

Función	Parámetros	Descripción
rmtree	shutil.rmtree(*path*)	Borra el directorio o archivo no vacío de la ruta *path.*
move	shutil.move(*src, dst*)	Mueve el directorio o archivo desde una ubicación *src* a otra *dst.*

Tabla 9: Funciones utilizadas de Shutil

3.2.8. *ReportLab*

Reportlab es un motor de código abierto para la creación y edición de documentos PDF. En este proyecto, esta librería se ha utilizado para la generación de multas en PDF. Las funciones de esta librería utilizadas para la aplicación se detallan en la *Tabla 10*[22].

Función	Parámetros	Descripción
canvas	*c* = canvas.Canvas(*filename*)	Permite la creación de un documento PDF de nombre *filename.* Las modificaciones que se realicen en dicho documento serán realizadas en la variable *c.*
drawImage	c.drawImage(*filename, x, y, w, h*)	Permite la inserción de una imagen de nombre *filename* en un documento PDF *c.* Su inserción se realiza en la posición del plano *(x, y)* del documento. La altura de la imagen es *h* y el largo *w.*
drawString	c.drawString(*x, y, string*)	Permite la inserción de una cadena de caracteres *string* en la posición del plano *(x, y)* de un documento PDF *c.*
rect	c.rect(*x, y, w, h*)	Permite la inserción de un rectángulo de altura *h* y de largo *w* en un documento PDF *c*. *X* e *y* representan las coordenadas de la esquina superior izquierda del rectángulo.
save	c.save()	Guarda los cambios realizados en un documento PDF *c.*

Tabla 10: Funciones utilizadas de ReportLab

3.2.9. Os

Esta librería también ha sido utilizada para la gestión de directorios. Las funciones utilizadas de esta librería se detallan en la *Tabla 11*[23].

Función	Parámetros	Descripción
getcwd	os.getcwd()	Obtiene el directorio actual.
access	os.access(*path*, os.F_OK)	Devuelve verdadero si el directorio de la ruta *path* existe.
mkdir	os.mkdir(*path*)	Crea un nuevo directorio en la ruta *path.*
remove	os.remove(*path*)	Borra el directorio o archivo de la ruta *path.*

Tabla 11: Funciones utilizadas de Os

3.2.10. RPi.GPIO

Esta librería permite configurar e interactuar con las entradas y salidas de la *Raspberry Pi*. Permite también la configuración de entradas como interrupciones. Las funciones utilizadas de esta librería se detallan en la *Tabla 12*[24].

Función	Parámetros	Descripción
setmode	GPIO.setmode(*mode*)	Permite configurar los pines de la *Raspberry Pi* según el modo *mode*: - GPIO.BOARD: pines numerados sucesivamente en la placa. - GPIO.BCM: pines numerados según la numeración GPIO.
setwarnings	GPIO.setwarnings(*mode*)	Habilita o deshabilita warnings según el modo *mode*: True: warnings habilitados. False: warnings deshabilitados.
setup	GPIO.setup(*pin, mode*)	Configura un determinado pin *pin* según el modo *mode*: - GPIO.IN: pin configurado como entrada. - GPIO.OUT: pin configurado como salida.
output	GPIO.output(*pin, mode*)	Pone una determinada salida *pin* en valor alto (3.3 V) o en valor bajo (0 V) según el modo *mode*: - GPIO.LOW: valor bajo (0 V). - GPIO.HIGH: valor alto (3.3 V).
input	GPIO.input(*pin*)	Devuelve un 1 si en la entrada pin *pin* hay 3.3 V o devuelve 0 si hay 0 V.
cleanup	GPIO.cleanup()	Devuelve las GPIOs a su estado inicial de funcionamiento.
add_even_detect	GPIO.add_event_detect(*pin, mode, callback*)	Configura un determinado pin *pin* como una interrupción, que ejecuta una función *callback* según el modo *mode*: - GPIO.RISING: flancos de subida. - GPIO.FALLING: flancos de bajada. - GPIO.BOTH: flancos de subida y de bajada.

Tabla 12: Funciones utilizadas de RPi.GPIO

3.2.11. LCDLIB16x2

Esta librería permite la configuración de una LCD de 16x2, así como la visualización de mensajes en ambas líneas. Las funciones utilizadas de esta librería se detallan en la *Tabla 13*[25].

Función	Parámetros	Descripción
lcd_init	LCD.lcd_init()	Configuración e inicialización de la LCD.
lcd_string	LCD.lcd_string(*string, line*)	Escritura en la LCD de una cadena de caracteres *string*, en una determinada línea *line:* - LCD.LINE_1: línea 1. - LCD.LINE_2: línea 2.

Tabla 13: Funciones utilizadas de LCDLIB16x2

3.2.12. *Threading*

Este módulo permite la existencia de hilos en la programación. Particularmente se ha utilizado en este proyecto para cambiar el mensaje de la LCD. Las funciones utilizadas de esta librería se detallan en la *Tabla 14*[26].

Función	Parámetros	Descripción
start	th.start()	Inicia la actividad del hilo *th.*
daemon	th.daemon = *True*	Hilo *th* de utilidad, para poder salir del programa principal satisfactoriamente con una interrupción de teclado.
Timer	*th* = threading.Timer(*time, function, [param])*	Una vez iniciada su actividad, el hilo *th* ejecuta una función *function* después de un tiempo *time*. Se debe especificar los parámetros *param* que necesita la función.

Tabla 14: Funciones utilizadas de Threading

3.3. Rasgos generales de la programación

Aunque el código completo de la programación puede consultarse en el *ANEXO 2* y en el *ANEXO 3*, a continuación, se detallan los aspectos básicos sobre la misma.

Destacar que el programa principal "solamente" se encarga de procesar las imágenes capturadas por la *Picamera* para el reconocimiento de caracteres, y de redactar la multa correspondiente. Así pues, el programa principal permanece en un bucle de espera mientras no existan imágenes que procesar.

Para una mejor organización de ficheros y directorios, lo primero que hace el programa es crear un directorio donde se incluirán las multas redactadas. Si este directorio ya existe, se borra y se crea de nuevo para eliminar las multas antiguas. En cuanto a las imágenes, el programa principal elimina ráfagas de fotos conforme van siendo procesadas, así como las imágenes intermedias que se van creando.

Los sensores IR están configurados como interrupciones por flancos de subida. Así pues, cuando un determinado coche es detectado por el sensor 1, el programa principal es interrumpido y se ejecuta una función en la que se obtiene el tiempo en ese instante. Cuando el coche es detectado por el sensor 2, el programa principal vuelve a ser interrumpido, y se ejecuta otra función en la que se vuelve a obtener el tiempo. La resta de los tiempos obtenidos permite calcular el tiempo transcurrido en el tramo de interés, y se procede al cálculo de la velocidad media utilizando la distancia del tramo. Si esta velocidad es superior a un cierto umbral, la *PiCamera* realiza una ráfaga de capturas que serán procesadas por el programa principal, y la activación de un LED y de un zumbador indica que ese coche ha sobrepasado el límite de velocidad.

El rozamiento de las escobillas por la pista de *Scalextric* produce ruido en los receptores IR, y hace que aparezcan picos de tensión en los mismos. Puesto que las interrupciones están programadas por flancos de subida, estos picos hacen que se interrumpa el programa principal y se ejecute la función correspondiente. Para solventar este problema, se ha programado un filtro *software*, que se detalla en el apartado 3.5 de este proyecto.

Se dispone también de una base de datos en la que una serie de matrículas está asociada a una serie de propietarios. También se especifica el modelo y el color del coche. Cuando se identifica una determinada matrícula, se comprueba su coincidencia con la base de datos, para redactar la multa correctamente. La base de datos del prototipo puede consultarse en el *ANEXO 4*.

La existencia de hilos en la programación permite el cambio periódico del mensaje de la LCD.

3.4. Diagramas de flujo de la programación

En las siguientes figuras se muestra el diagrama de flujo del programa principal (*Figura 17*), de la interrupción del sensor 1 (*Figura 18*), de la interrupción del sensor 2 (*Figura 19*), y del hilo (*Figura 20*).

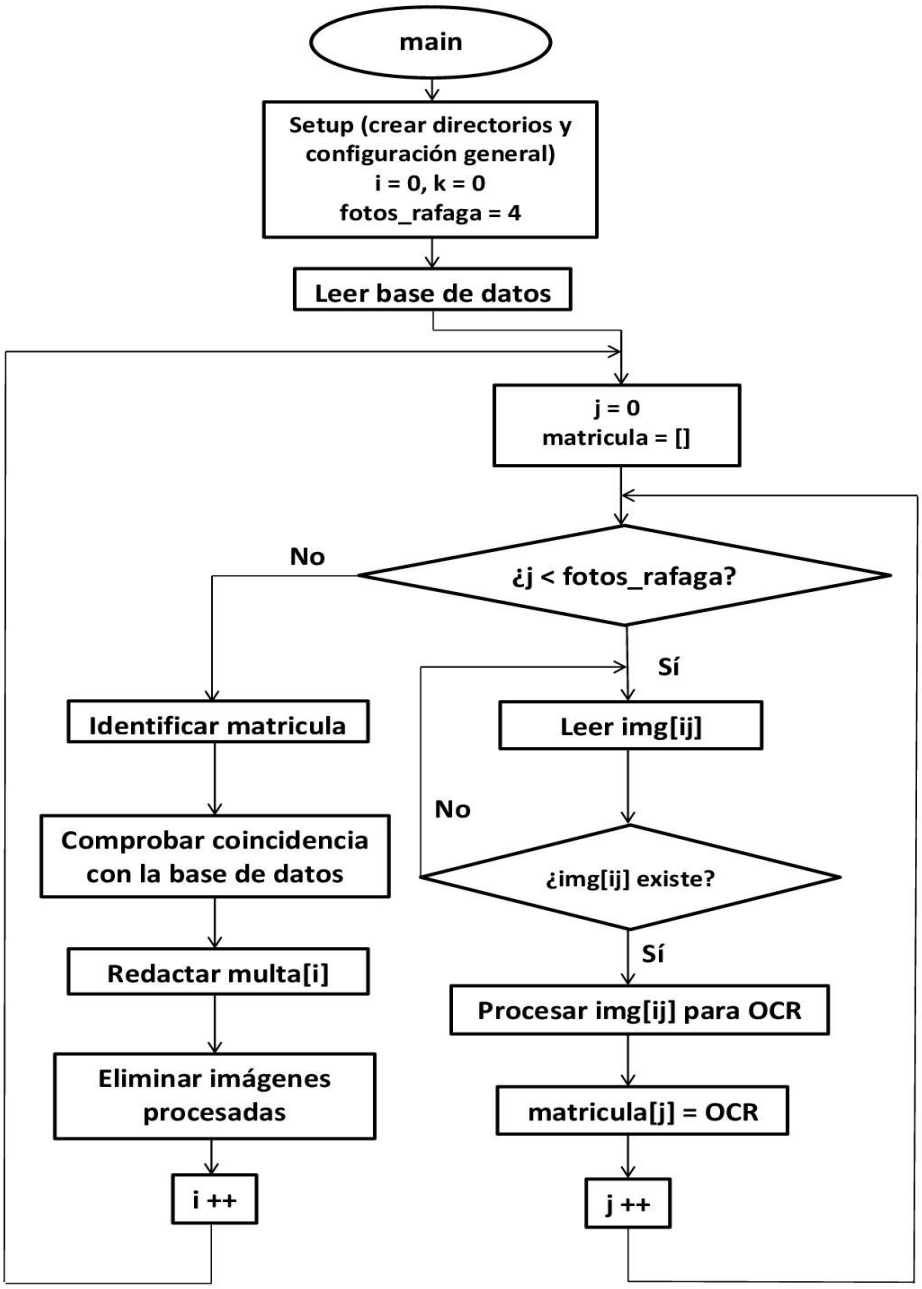

Figura 17: Diagrama de flujo del programa principal

La variable *i* expresa la ráfaga de fotografías que se está procesando, mientras que la variable *j* expresa el número de fotografía dentro de una determinada ráfaga.

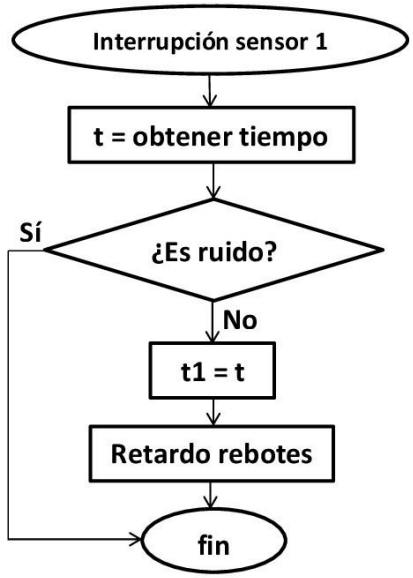

Figura 18: Diagrama de flujo de la interrupción del sensor 1

Figura 20: Diagrama de flujo del hilo

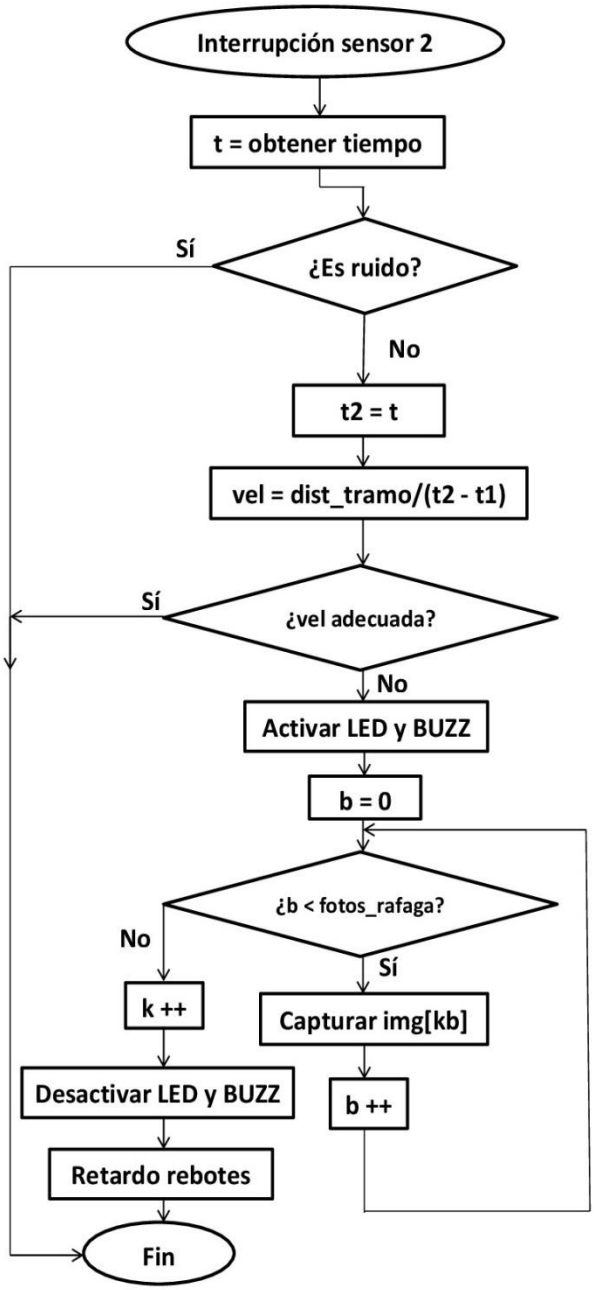

Figura 19: Diagrama de flujo de la interrupción del sensor 2

La variable *k* expresa la ráfaga de fotografías que captura la *PiCamera*, mientras que la variable *b* expresa el número de fotografía dentro de una determinada ráfaga.

Por tanto, se debe diferenciar que la labor de las variables *i* y *j* difiere de la labor de las variables *k* y *b*. Todas ellas son índices de las imágenes, pero mientras que las variables *i* y *j* se encargan de la gestión de las imágenes en cuanto al procesado de las mismas, la función de las variables *k* y *b* es la gestión de las imágenes capturadas por la *PiCamera*.

En la *Figura 21* se muestra el intercambio de información mediante las variables globales que realiza el programa principal con los diferentes procesos.

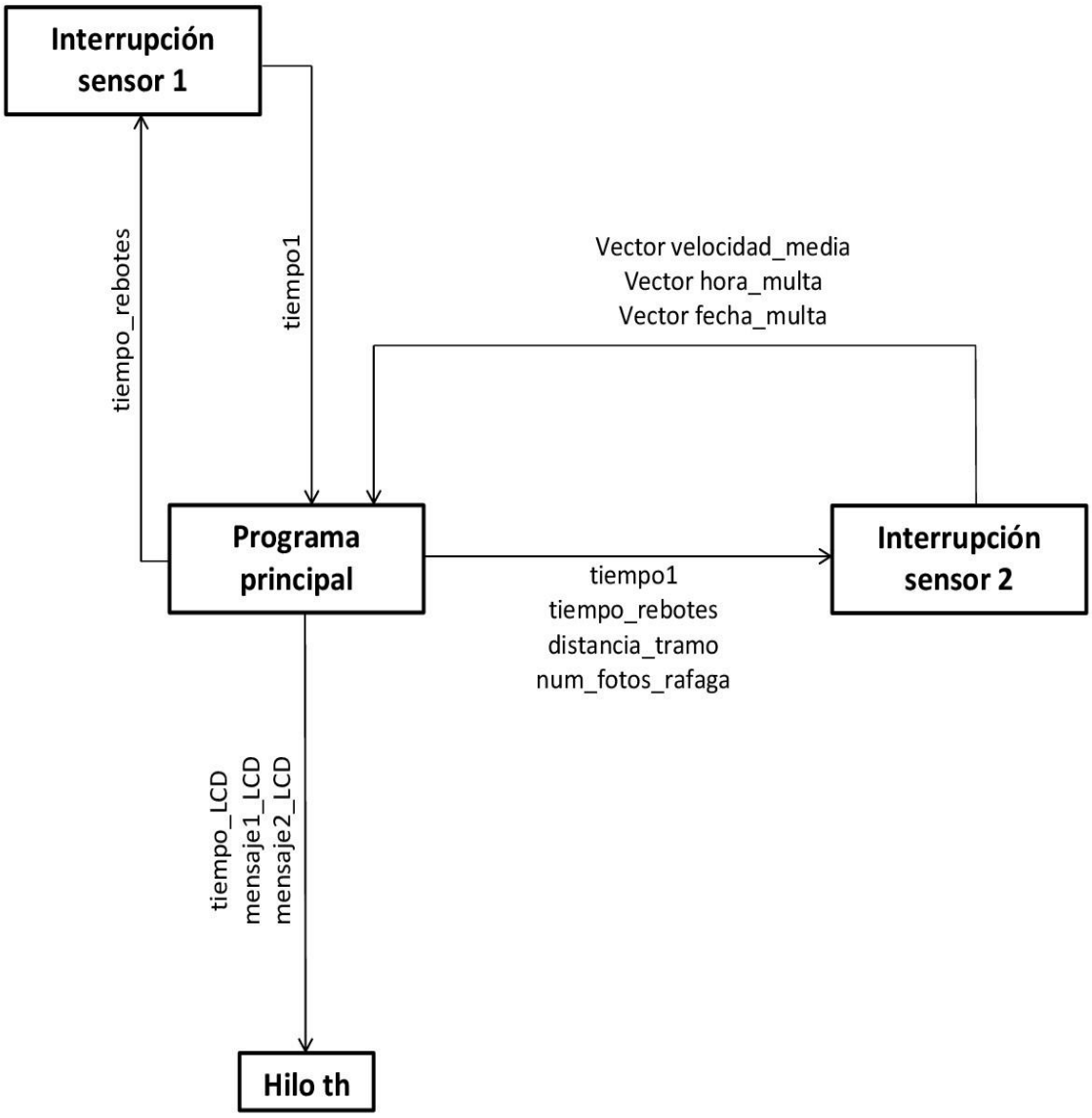

Figura 21: Esquema del intercambio de información entre procesos

Con la información que contienen las variables globales, el programa principal es capaz de procesar las imágenes para redactar la multa correspondiente. El sentido de las flechas informa del sentido del intercambio de información.

3.5. Filtro *software*

Los receptores IR captan algunos picos de tensión procedentes del rozamiento de las escobillas del coche con las pistas del circuito de *Scalextric*, sobretodo con los cambios bruscos de tensión que se producen en los tramos de mayor velocidad. Puesto que las interrupciones asociadas a dichos sensores están configuradas en la *Raspberry Pi* por flancos de subida, y que los picos de tensión son comparables a 3.3 V, las interrupciones se disparan de manera errónea con la captación de estos picos, cuando en realidad, el coche no ha atravesado la barrera infrarroja correspondiente.

Para solventar este problema, y puesto que los picos de tensión tienen una duración mínima, se ha implementado un filtro *software* en las interrupciones de los receptores IR. Dicho filtro consiste en la verificación del estado alto (3.3 V) del correspondiente sensor. Así pues, cuando un pico de tensión hace saltar a la interrupción, el filtro *software* actúa para identificar esta señal como ruido, e inmediatamente el programa sale de la interrupción.

Tras la realización de tareas de mantenimiento mecánico en el prototipo, tales como la limpieza de las pistas y el cambio de escobillas de los coches, estos picos de tensión se han reducido notablemente. Pese a ello, el filtro *software* se ha seguido manteniendo. La *Figura 22* muestra la motorización de uno de los receptores IR, donde pueden observarse claramente los mencionados picos. En dicha figura, se puede apreciar que la amplitud del mayor pico de tensión alcanza los 800 mV.

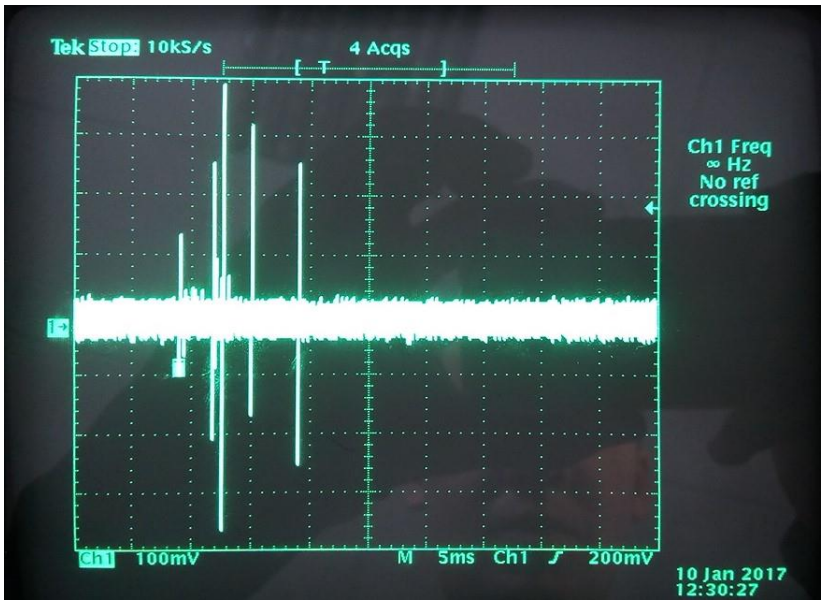

Figura 22: Monitorización en el osciloscopio de los picos de tensión ocasionados en el receptor IR 2

Cuando un coche atraviesa una determinada barrera infrarroja, se producen una serie de rebotes en el receptor IR correspondiente. Para evitar que la interrupción salte de manera continuada ante estos rebotes, se ha programado un retardo de 80 ms (*tiempo_rebotes*) en las interrupciones desde el primer flanco de subida del pulso de 3.3 V. La *Figura 23* muestra la motorización de uno de los receptores IR tras el paso de un vehículo. En ella puede observarse el nivel alto (3.3 V) que produce el paso de un vehículo, así como los rebotes mencionados. La duración total del pulso es de aproximadamente 60 ms, por tanto, queda justificada la elección de 80 ms como tiempo de retardo.

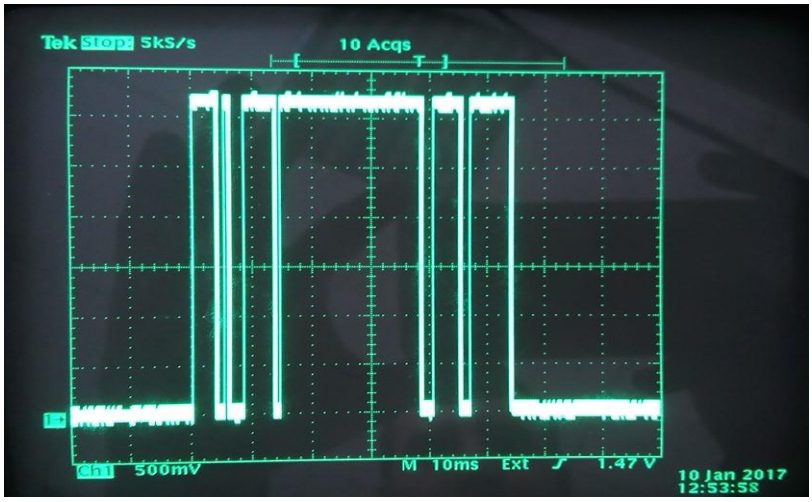

Figura 23: Monitorización en el osciloscopio del receptor IR 2 tras el paso de un vehículo

La *Figura 24* muestra la motorización de uno de los receptores tras el paso de dos vehículos. En ella pueden apreciarse los dos niveles altos (3.3 V) que producen los pasos de los vehículos, así como los picos de tensión captados entre los mismos. Cabe destacar que estos picos no son comparables con 3.3 V.

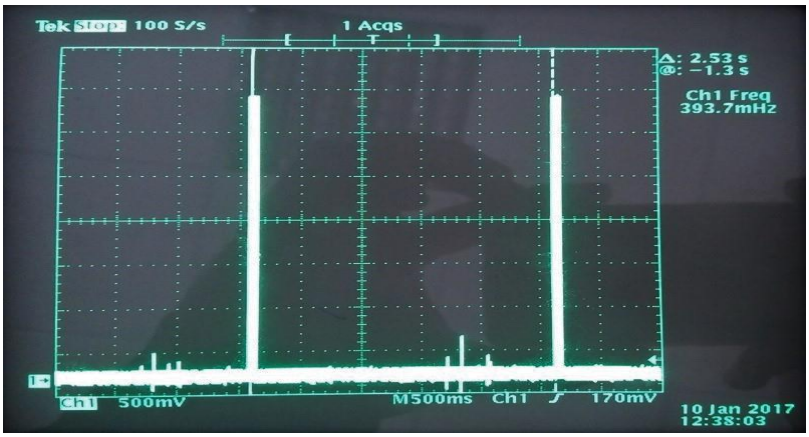

Figura 24: Monitorización en el osciloscopio del receptor IR 2 tras el paso de dos vehículos

4. Procesado de imagen

Aunque para el reconocimiento óptico de caracteres se ha utilizado la librería de *Python Pytesseract*, que es un motor de OCR muy potente como se ha visto anteriormente, la verdadera labor que se ha llevado a cabo en este proyecto en cuanto a procesado digital de imagen ha sido el procesamiento previo de la misma para el correcto funcionamiento de *Pytesseract*. En rasgos generales, el procesado de la imagen tiene como objetivo conseguir una imagen en la que solamente aparezcan en color negro los caracteres de la matrícula a reconocer, sobre un fondo blanco.

Tras la realización de varios ensayos, se ha optado por capturar una secuencia de 4 imágenes cuando un determinado coche supera la velocidad media estipulada del tramo (8 m/s), con el objetivo de que el reconocimiento óptico de caracteres sea lo más fiable posible. Así pues, tras la captura de la secuencia, las imágenes son procesadas una a una, para servir de entrada posteriormente a la función OCR. Una vez reconocidas las 4 matrículas, el programa decide mediante un algoritmo de selección (se desarrolla en el apartado 5 de este proyecto) la matrícula correcta.

Puesto que el procesado de imagen es una técnica que conlleva mucho coste computacional, la identificación de una determinada matrícula no se realiza de manera inmediata.

4.1. Conceptos teóricos básicos

Antes de proceder al desarrollo del algoritmo de procesado de imagen, a continuación, se detallan algunos conceptos teóricos básicos indispensables.

- Imagen digital[27]: proyección bidimensional de una escena tridimensional. Para cada posición dentro de un plano (x, y) se define una magnitud asociada a la escena representada.
- Procesado digital de imagen[27]: tiene como objetivo favorecer la obtención de información de la imagen, pero nunca puede incrementar la cantidad de información que una imagen contiene.
- Representación del color en imágenes digitales[11]: se obtienen a través de los valores RGB, que son los colores primarios (rojo, verde y azul, respectivamente). La combinación de estos colores da lugar a colores secundarios.
- Imágenes de intensidad[27]: aquellas en las que se almacena en cada pixel un valor de intensidad o nivel de gris.

- Imagen binaria[28]: aquella en la que sus píxeles solamente pueden tomar dos valores: blanco o negro. A partir de una imagen de niveles de intensidad, se puede establecer un umbral que actué como frontera. En una binarización simple, este umbral es seleccionado por el usuario a partir del histograma de la imagen (distribución de los niveles de intensidad de una imagen), sin embargo, la binarización *Otsu*[29] selecciona automáticamente el umbral para que la varianza de la dispersión de niveles de grises sea la mínima. La binarización *Otsu* es recomendable para imágenes cuyo histograma tenga dos picos distinguidos en niveles de intensidad diferentes.
- Contornos[28]: representan los puntos de frontera entre un objeto y el entorno en contacto con él.
- Erosión[28]: proceso de eliminación de los puntos de frontera de un objeto, reduciendo su área en pixeles según el tamaño y la forma de la máscara utilizada. Las máscaras más comunes son las cuadradas y las de tipo cruz.
- Dilatación[28]: proceso de incorporación al objeto de los puntos del entorno en contacto con él. Se produce un incremento de su área en píxeles según el tamaño y la forma de la máscara utilizada. Las máscaras más comunes son las mismas que en el proceso de erosión.
- ROI: selección de una zona de interés dentro de una imagen.

En cuanto a la erosión y a la dilatación, se debe tener clara la asociación que realiza el *software* a los objetos y al entorno. Dependiendo de si la herramienta identifica a los objetos como un conjunto de píxeles negros o como un conjunto de píxeles blancos, una erosión puede realizar la función de una dilatación, y viceversa. En el caso de *OpenCV*, los objetos son identificados como un conjunto de píxeles blancos.

Otro aspecto importante es la iluminación[10]. Para obtener buenas fotografías, la intensidad de luz que llega a la imagen debe ser la adecuada. La energía no debe ser excesiva (sobreexposición) ni insuficiente (subexposición). Además, la iluminación debe ser la suficiente para que el tiempo de exposición de la cámara sea mínimo, y así poder capturar imágenes de manera rápida, lo que es fundamental en este caso al tratarse de imágenes de un objeto en movimiento.

4.2. Diagrama de flujo del procesado de imagen

En la *Figura 25* se puede observar el diagrama de flujo en cuanto a la programación del procesado digital de imagen.

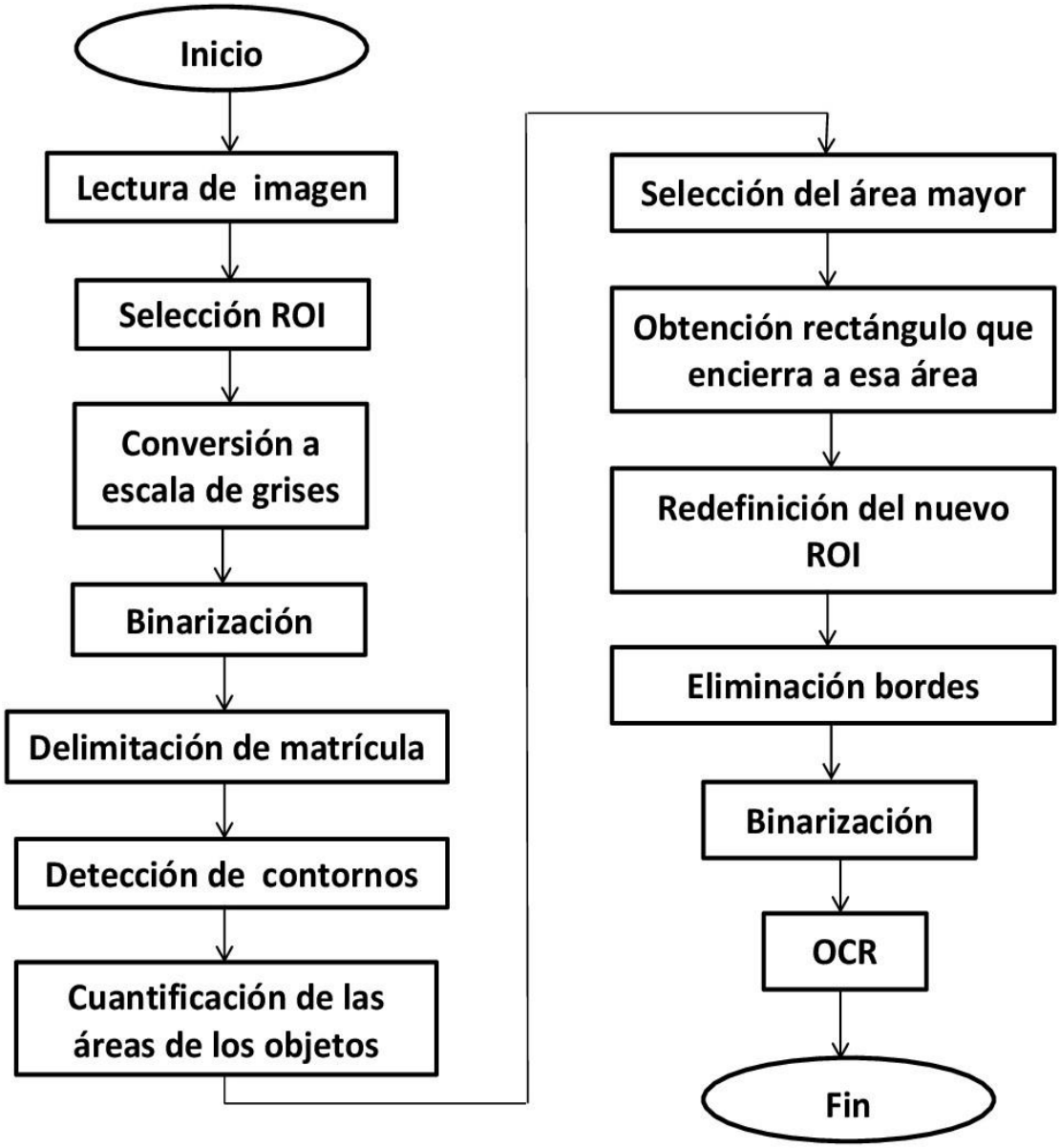

Figura 25: Diagrama de flujo del procesado digital de imagen

A continuación, se detallan todos los pasos anteriormente descritos con la ilustración de un coche circulando por el *Scalextric*.

Lectura de imagen

Las siguientes cuatro imágenes muestran las capturas realizadas por la *PiCamera* tras el paso de un coche de *Scalextric*.

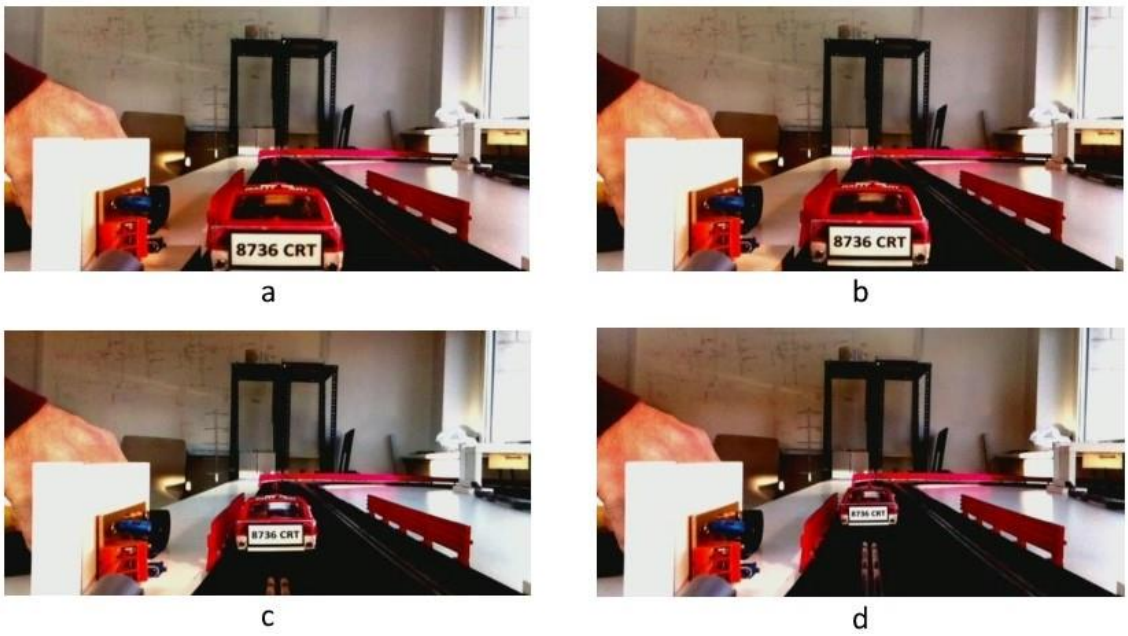

Figura 26: Capturas de PiCamera

Selección ROI

Las imágenes anteriores muestran mucha información que no es útil, por ello se selecciona una zona de interés común para todas las capturas, como muestran las siguientes cuatro imágenes. La definición de este ROI se ha llevado a cabo teniendo en cuenta que la cámara ocupa una posición fija, y que los coches circulan por el carril izquierdo del *Scalextric*.

Figura 27: ROI de las capturas

Conversión a escala de grises

Para realizar la binarización de las imágenes, estas deben ser imágenes en escala la grises. Las siguientes cuatro imágenes muestran esta conversión.

Figura 28: ROI en escala de grises

Binarización

Las siguientes cuatro imágenes muestran la binarización de las imágenes anteriores mediante el método de *Otsu*. Se ha optado por este método porque los histogramas de las anteriores imágenes permiten que la selección automática del umbral sea la adecuada.

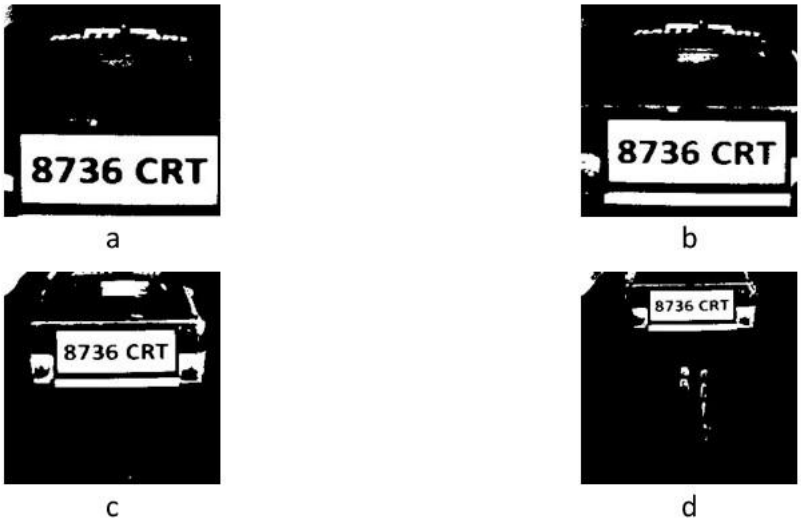

Figura 29: Binarización de ROI

Delimitación de matrícula

En cada imagen, se desea definir una nueva área de interés que se corresponda con la placa de la matrícula. Para evitar que el objeto matrícula se junte con otros objetos, tales como el parachoques, se ha realizado una erosión en la imagen mediante una máscara 2x2 de 3 iteraciones. Este paso solamente se utiliza para identificar el objeto matrícula, puesto que como se observa en las siguientes 4 imágenes, los caracteres se deterioran bastante.

Figura 30: Erosión de ROI

Una vez que el objeto matrícula está perfectamente separado, se obtienen los contornos en las imágenes, y se identifica aquel contorno que encierre al área mayor, es decir, el contorno que encierra a la matricula. Posteriormente, se obtiene el rectángulo mínimo que encierra a este contorno.

Redefinición del nuevo ROI

El rectángulo obtenido anteriormente se utiliza como nuevo ROI, para conseguir que la imagen solamente contenga información sobre los caracteres de la matrícula, como muestran las siguientes 4 imágenes.

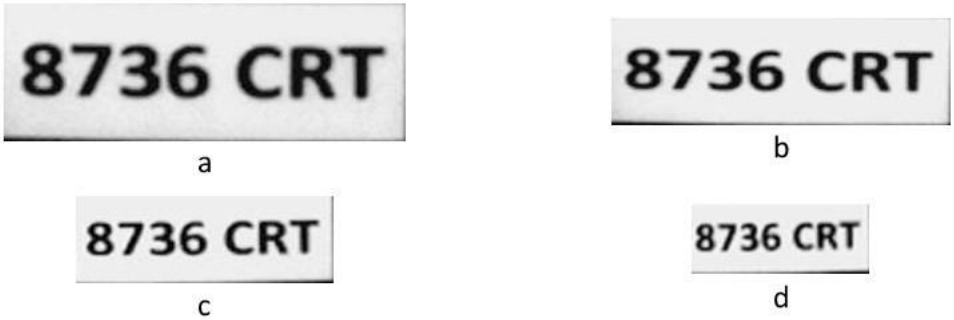

Figura 31: Objeto matrícula en escala de grises

Eliminación de bordes

Para que el resultado del OCR sea el deseado, se ha vuelto a seleccionar otro ROI en las imágenes anteriores. Se ha eliminado un 10 % en altura y un 1.5 % en anchura para eliminar los posibles bordes de la matrícula, y conseguir eliminar el fondo blanco de la matrícula que no aporta información. Este paso se observa en las siguientes 4 imágenes.

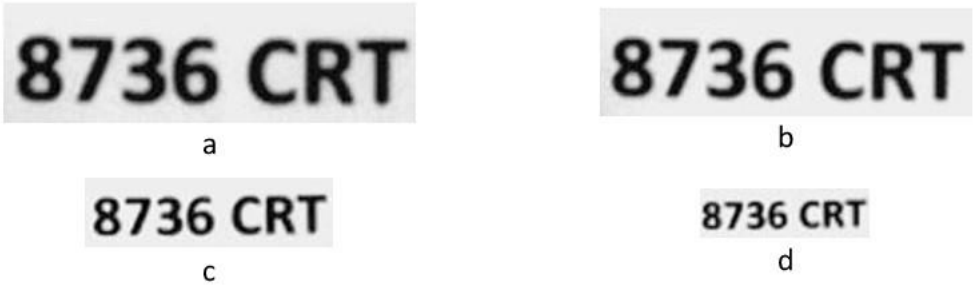

Figura 32: ROI de la matrícula

Binarización

Como paso final en el procesado digital de imagen, se han binarizado mediante el método de *Otsu* las imágenes anteriores para que el reconocimiento OCR sea el correcto. Este paso se observa en las siguientes 4 imágenes.

Figura 33: Binarización de la matrícula

Así pues, la labor en cuanto al procesado digital de imagen se resume en la siguiente imagen, que muestra una captura de la *PiCamera* y el resultado de su procesado digital de imagen descrito anteriormente.

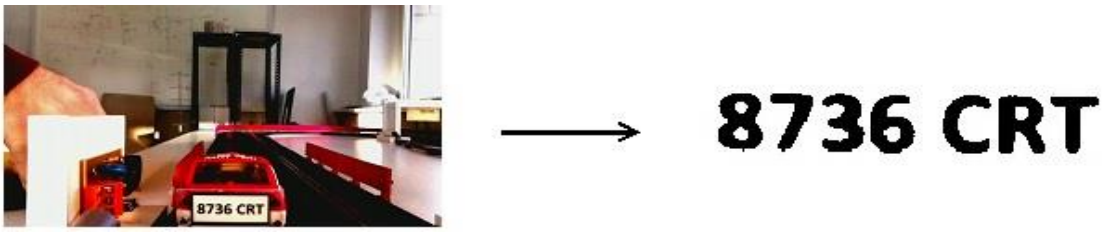

Figura 34: Labor del procesado digital de imagen

5. Identificación de la matrícula

Una vez procesadas todas las imágenes de una determinada ráfaga, se obtiene un vector fila (*vector[]*) de cuatro elementos. Cada elemento del vector contiene los caracteres del OCR de su respectiva imagen.

Así pues, el programa deberá identificar la matrícula correcta. Para ello, se ha implementado en la programación el algoritmo mostrado en la *Figura 35.* A partir del vector fila, dicho algoritmo comprueba si coinciden dos cadenas de caracteres de longitud 8 (las matrículas están compuestas por 4 números, 1 espacio, y 3 letras). En caso afirmativo, el programa seleccionará esa cadena de caracteres como la matrícula correcta. En caso de que ninguna cadena de caracteres de longitud 8 coincida, el programa identificará como matrícula correcta la primera cadena de caracteres cuya longitud sea 8. En caso de que ninguna cadena de caracteres sea de longitud 8, el programa no identificará ninguna matrícula como correcta.

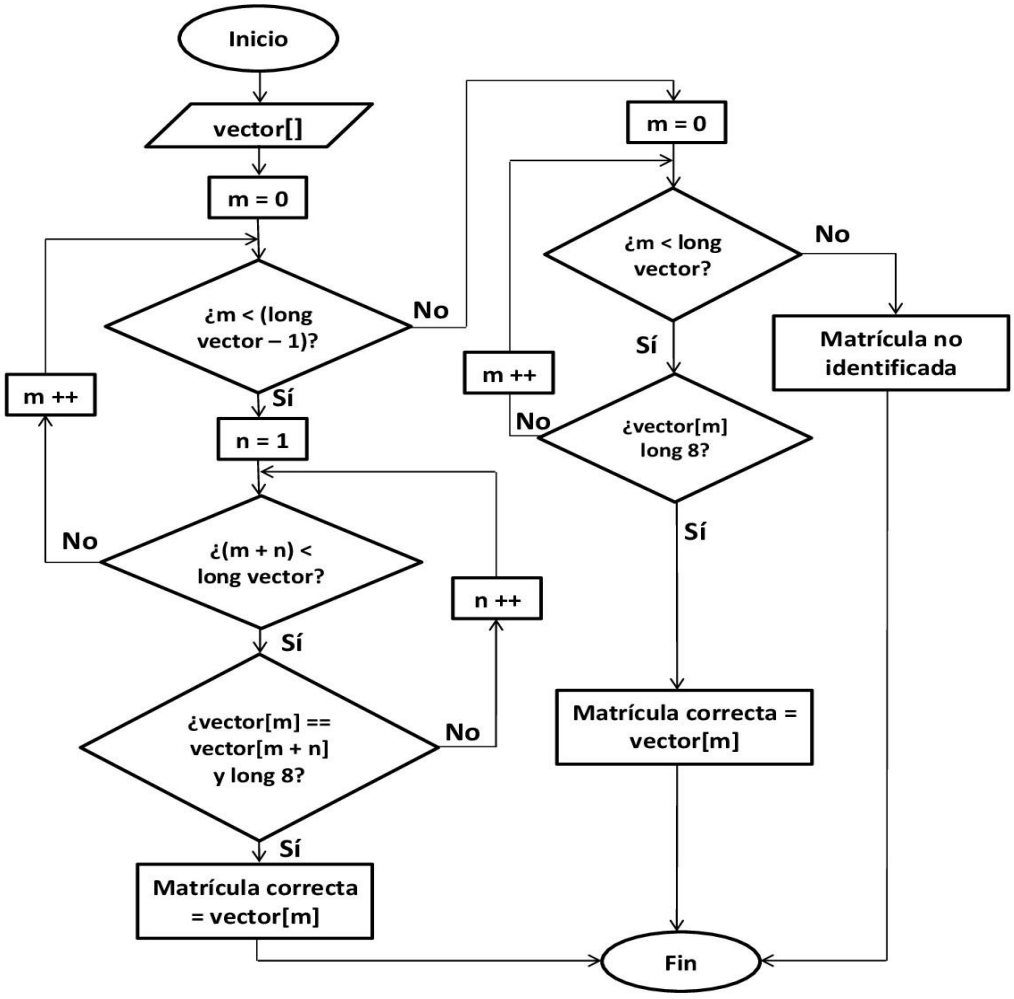

Figura 35: Diagrama de flujo de la identificación de la matrícula

6. Placas de circuito impreso

Todos los diseños electrónicos se han implementado en PCBs. Debido a su relativa sencillez, y a que el coste de fabricación es menor, todas las PCBs se han fabricado a simple cara.

Los diseños de todas las PCBs se han llevado a cabo mediante el programa informático *Eagle.* Para la realización de los mismos, se han seguido las reglas de diseño[30] que se detallan a continuación:

- Las pistas se han diseñado cortas y anchas, para minimizar la inductancia. Los cambios de dirección de las mismas se han realizado a 45º para evitar problemas de reflexión.
- Una vez soldados los componentes, todas sus patillas han sido cortadas para evitar que actúen como antenas.
- Todas las placas se han diseñado con plano de masa, para que la corriente retorne por el camino de menor impedancia. Con ello se evita la existencia de grandes áreas de bucles que puedan actuar como antena para la captación de ruido.
- Se ha colocado zócalo en el circuito integrado 555 con el objetivo de que pueda ser remplazado con facilidad en caso de avería, puesto que se trata de un elemento sensible.

En las siguientes imágenes se muestran todas las PCBs de los diferentes diseños electrónicos. Las dimensiones se muestran en mm.

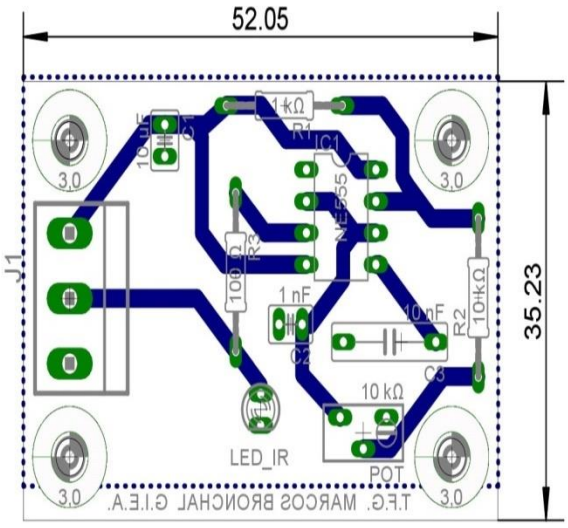

Figura 36: Board emisor IR

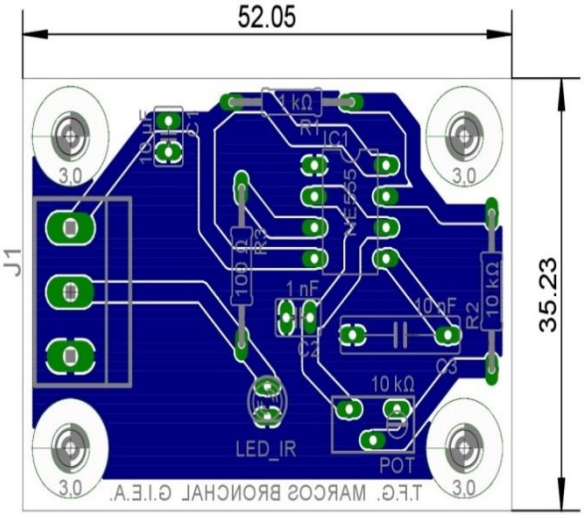

Figura 37: Board emisor IR con plano de masa relleno de cobre

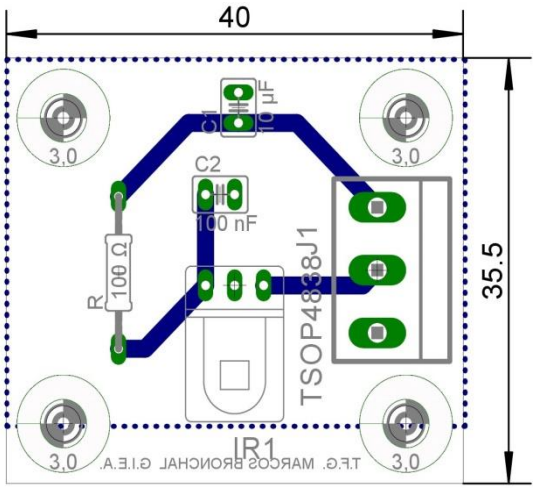

Figura 38: Board receptor IR

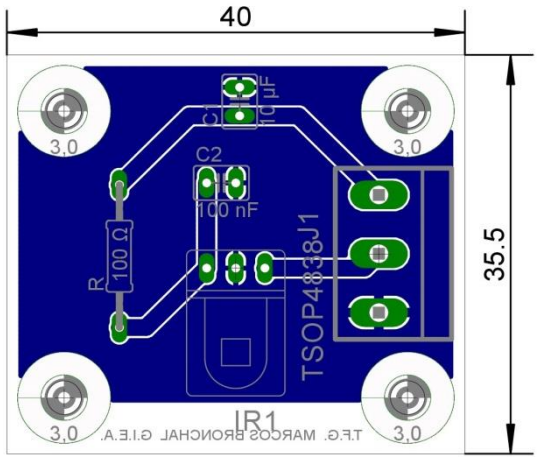

Figura 39: Board receptor IR con plano de masa relleno de cobre

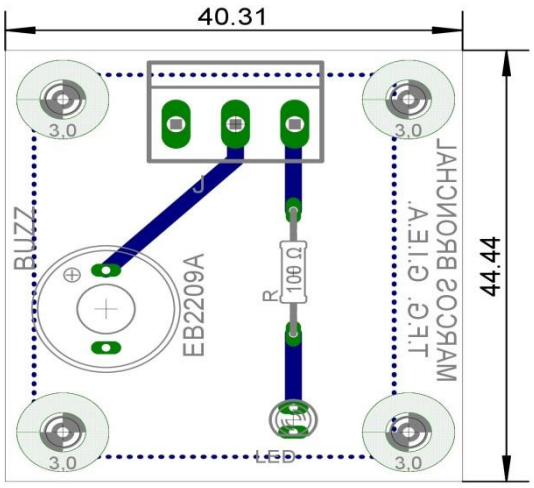

Figura 40: Board LED y BUZZ

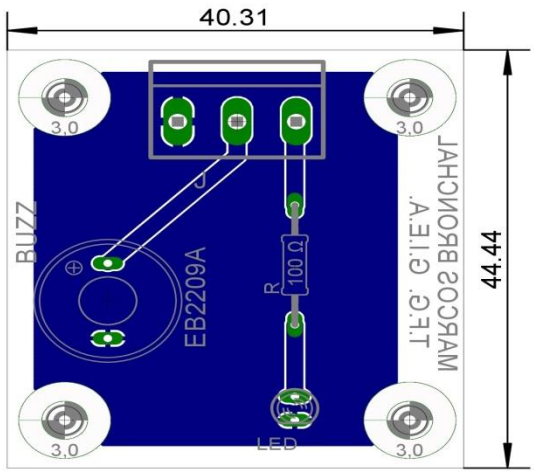

Figura 41: Board LED y BUZZ con plano de masa relleno de cobre

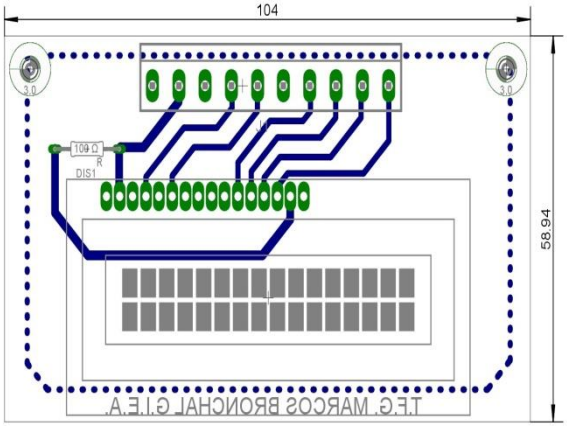

Figura 42: Board LCD

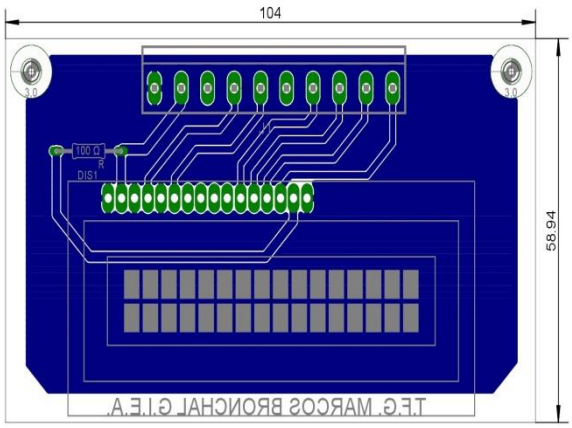

Figura 43: Board LCD con plano de masa relleno de cobre

7. Pruebas del sistema

Para la determinación de la tasa de éxito del prototipo, entiendo como éxito la identificación correcta de las matrículas, se han realizado algunas pruebas en diferentes condiciones.

Básicamente, una vez calibrado el sistema, las condiciones que modifican la tasa de éxito son la luz ambiental y la propia velocidad del vehículo. Por ello, se han hecho pasar 40 coches por el circuito de *Scalextric* en las condiciones que se detallan a continuación, para que el sistema proceda a la identificación de su matrícula. Las pruebas realizadas, con sus correspondientes condiciones, han sido las siguientes:

- Prueba 1: luz ambiental alta, y velocidad alta del vehículo.
- Prueba 2: luz ambiental alta, y velocidad baja del vehículo.
- Prueba 3: luz ambiental baja, y velocidad alta del vehículo.
- Prueba 4: luz ambiental baja, y velocidad baja del vehículo.

El término luz ambiental alta implica realizar la prueba con las persianas abiertas en un día soleado, mientras que el término luz ambiental baja significa realizar la prueba con las persianas cerradas. Por otro lado, el término velocidad baja implica que el vehículo circula por el circuito entre un rango de velocidades de 8-9 m/s, mientras que el término velocidad alta significa que el coche circula por el circuito a una velocidad superior a 9 m/s, teniendo en cuenta que el coche puede alcanzar una velocidad máxima de aproximadamente 10.5 m/s.

La tasa de éxito de cada prueba, expresada en porcentaje, ha sido determinada mediante la siguiente ecuación:

$$Tasa\ de\ éxito = \frac{nº\ éxitos}{nº\ pruebas} \cdot 100$$

Puesto que el número de ensayos de todas las pruebas es el mismo, se ha realizado una media aritmética para la estimación de las tasas de éxito globales del sistema, que se detallan en el apartado de análisis de los resultados.

A continuación, se muestran los resultados de las pruebas, incluyendo la tasa de éxito de cada una.

7.1. Prueba 1

Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito	Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito
1	3698 GGT	3698 GGT	✔	21	8736 CRT	8736 CRT	✔
2	9339 GVP	9339 GVP	✔	22	1975 BVL	1975 BVL	✔
3	5989 FXB	5989 FXB	✔	23	6223 GFH	6223 GFH	✔
4	7799 HGT	7799 HGT	✔	24	1507 GJV	1507 GJV	✔
5	9364 BLS	9364 BLS	✔	25	0913 CFT	0913 CFT	✔
6	9369 FFR	9369 FFR	✔	26	5693 BYT	5693 BYT	✔
7	1556 GST	1556 GST	✔	27	9985 CCR	9985 CCR	✔
8	6739 DGH	6739 DGH	✔	28	0007 BLF	0007 BLF	✔
9	3990 FHD	3990 FHD	✔	29	6963 HDS	6963 HDS	✔
10	0113 FJD	0113 F10	✗	30	4598 HHS	4598 HHS	✔
11	1278 FDD	1278 F00	✗	31	7177 GTC	7177 GTC	✔
12	9223 JRD	9223 JRD	✔	32	6380 HHG	6380 HHG	✔
13	6544 HCR	6544 HCR	✔	33	8869 JJD	8869 LID	✗
14	1474 GFG	1474 GFG	✔	34	3399 GYX	3399 GYX	✔
15	4963 CBD	4963 C80	✗	35	3971 DLZ	3971 DLZ	✔
16	8896 CDC	8896 CDC	✔	36	6587 BLL	6587 BLL	✔
17	5477 GTV	5477 GTV	✔	37	3365 DTR	3365 DTR	✔
18	4697 BBV	4697 BBV	✔	38	6422 HGC	6422 HGC	✔
19	6423 GTH	6423 GTH	✔	39	1133 HFF	1133 HFF	✔
20	7622 BBD	7622 380	✗	40	7143 HFV	7143 HFV	✔

Tabla 15: Resultados de la prueba 1

Tasa de éxito = 87.5 %

7.2. Prueba 2

Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito	Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito
1	3698 GGT	3698 GGT	✔	21	8736 CRT	8736 CRT	✔
2	9339 GVP	9339 GVP	✔	22	1975 BVL	1975 BVL	✔
3	5989 FXB	5989 FXB	✔	23	6223 GFH	6223 GFH	✔
4	7799 HGT	7799 HGT	✔	24	1507 GJV	1507 GJV	✔
5	9364 BLS	9364 BLS	✔	25	0913 CFT	0913 CFT	✔
6	9369 FFR	9369 FFR	✔	26	5693 BYT	5693 BYT	✔
7	1556 GST	1556 GST	✔	27	9985 CCR	9985 CCR	✔
8	6739 DGH	6739 DGH	✔	28	0007 BLF	0007 BLF	✔
9	3990 FHD	3990 FHD	✔	29	6963 HDS	6963 HDS	✔
10	0113 FJD	0113 FJD	✔	30	4598 HHS	4598 HHS	✔
11	1278 FDD	1278 FDD	✔	31	7177 GTC	7177 GTC	✔
12	9223 JRD	9223 JRD	✔	32	6380 HHG	6380 HHG	✔
13	6544 HCR	6544 HCR	✔	33	8869 JJD	8869 JJD	✔
14	1474 GFG	1474 GFG	✔	34	3399 GYX	3399 GYX	✔
15	4963 CBD	4963 C30	✗	35	3971 DLZ	3971 DLZ	✔
16	8896 CDC	8896 CDC	✔	36	6587 BLL	6587 ELL	✗
17	5477 GTV	5477 GTV	✔	37	3365 DTR	3365 DTR	✔
18	4697 BBV	4697 BBV	✔	38	6422 HGC	6422 HGC	✔
19	6423 GTH	6423 GTH	✔	39	1133 HFF	1133 HFF	✔
20	7622 BBD	7622 BBD	✔	40	7143 HFV	7143 HFV	✔

Tabla 16: Resultados de la prueba 2

Tasa de éxito = 95 %

7.3. Prueba 3

Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito	Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito
1	3698 GGT	Mon (.6!	✗	21	8736 CRT	I736 CRT	✗
2	9339 GVP	9339 GVP	✔	22	1975 BVL	1975 BVL	✔
3	5989 FXB	59.9 FXI	✗	23	6223 GFH	6223 GFH	✔
4	7799 HGT	7799 HGT	✔	24	1507 GJV	1507 GJV	✔
5	9364 BLS	9364 [RS	✗	25	0913 CFT	0913 CFT	✔
6	9369 FFR	9369 FFR	✔	26	5693 BYT	5693 BVT	✗
7	1556 GST	1556 GST	✔	27	9985 CCR	9935 C01	✗
8	6739 DGH	6739 DGN	✗	28	0007 BLF	“dor Inn	✗
9	3990 FHD	3990 FH0	✗	29	6963 HDS	6963 HDS	✔
10	0113 FJD	0113 F10	✗	30	4598 HHS	459. HHS	✗
11	1278 FDD	1278 F00	✗	31	7177 GTC	7177 61C	✗
12	9223 JRD	9223 1RD	✗	32	6380 HHG	QIIO “Nd	✗
13	6544 HCR	6544 HCR	✔	33	8869 JJD	5369 110	✗
14	1474 GFG	---- ---	✗	34	3399 GYX	3399 GYX	✔
15	4963 CBD	4963 (ED	✗	35	3971 DLZ	---- ---	✗
16	8896 CDC	3.0% (DC	✗	36	6587 BLL	6587 BLI	✗
17	5477 GTV	5477 GTV	✔	37	3365 DTR	3365 DTR	✔
18	4697 BBV	4697 BBV	✔	38	6422 HGC	6422 HGC	✔
19	6423 GTH	6423 GT”	✗	39	1133 HFF	III. MD.	✗
20	7622 BBD	7622 BIO	✗	40	7143 HFV	7143 NFV	✗

Tabla 17: Resultados de la prueba 3

Tasa de éxito = 37.5 %

7.4. Prueba 4

Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito	Nº ensayo	Matrícula a identificar	Matrícula identificada	Éxito
1	3698 GGT	3698 GGT	✔	21	8736 CRT	8736 CRT	✔
2	9339 GVP	9339 GVP	✔	22	1975 BVL	1975 BVL	✔
3	5989 FXB	QOIO VII	✗	23	6223 GFH	6223 GFH	✔
4	7799 HGT	7799 HGT	✔	24	1507 GJV	1507 GJV	✔
5	9364 BLS	9364 BLS	✔	25	0913 CFT	0913 CFT	✔
6	9369 FFR	9369 FFR	✔	26	5693 BYT	5693 BYT	✔
7	1556 GST	1556 GST	✔	27	9985 CCR	9985 CCR	✔
8	6739 DGH	6739 DGH	✔	28	0007 BLF	0007 BLF	✔
9	3990 FHD	3990 IND	✗	29	6963 HDS	6963 HDS	✔
10	0113 FJD	0113 F10	✗	30	4598 HHS	4598 HHS	✔
11	1278 FDD	127! FDD	✗	31	7177 GTC	7177 GTC	✔
12	9223 JRD	9223 JRD	✔	32	6380 HHG	6380 HHG	✔
13	6544 HCR	6544 HCR	✔	33	8869 JJD	---- ---	✗
14	1474 GFG	1474 GFG	✔	34	3399 GYX	3399 GYX	✔
15	4963 CBD	4963 C80	✗	35	3971 DLZ	397(DLZ	✗
16	8896 CDC	8896 CDC	✔	36	6587 BLL	6587 BLL	✔
17	5477 GTV	5477 GTV	✔	37	3365 DTR	3365 UT!	✗
18	4697 BBV	4697 BBV	✔	38	6422 HGC	6422 HGC	✔
19	6423 GTH	6423 GTH	✔	39	1133 HFF	1133 HFF	✔
20	7622 BBD	7622 880	✗	40	7143 HFV	7143 HFV	✔

Tabla 18: Resultados de la prueba 4

Tasa de éxito = 77.5 %

7.5. Análisis de los resultados

Observando nuevamente los resultados, se aprecian errores en el OCR comunes en varios ensayos. Dichos errores se producen por la similitud entre caracteres. Los errores más comunes se muestran en la *Tabla 19*.

Caracter	Caracter reconocido	Caracter	Caracter reconocido	Caracter	Caracter reconocido
D	0	J	1	G	6
B	8	J	L	C	(
B	3	J	I	H	“

Tabla 19: Errores comunes en el OCR

En *Tabla 20* se muestran las tasas de éxito del prototipo en la identificación de matrículas, tras las pruebas realizadas.

	Luz ambiental alta	Luz ambiental baja
Velocidad alta del vehículo	87.5 %	37.5 %
Velocidad baja del vehículo	95 %	77.5 %

Tabla 20: Tasas de éxito del prototipo

A la vista de los resultados obtenidos, se observa que la mayor tasa de éxito se ha logrado en condiciones de buena iluminación ambiental y con velocidades relativamente bajas de los vehículos. La menor tasa de éxito se ha obtenido en condiciones de mala iluminación ambiental y con velocidades relativamente altas de los vehículos.

La tasa de éxito es inversamente proporcional a la velocidad de los vehículos, puesto que a mayor velocidad es más difícil realizar el reconocimiento de caracteres. Unas buenas condiciones de luz ambiental favorecen el éxito en el reconocimiento de caracteres, puesto que cuando la iluminación no es adecuada se incrementa el tiempo de exposición de la cámara, lo que produce que se capturen imágenes borrosas.

Según los resultados, la aplicación funciona relativamente bien cuando las condiciones de luz ambiental son favorables, mientras que cuando las condiciones de luz ambiental no son tan favorables, se reduce notablemente la tasa de acierto.

Tras la realización de la media aritmética de las tasas de éxito en condiciones de velocidad alta y velocidad baja, se puede afirmar que la tasa de éxito estimada del prototipo es del **91.25 %** cuando las condiciones de luz ambiental son favorables, y del **57.5 %** en condiciones desfavorables de luz ambiental.

8. Plantilla multas

La *Figura 44* muestra un ejemplo de la generación de una multa después de que un coche haya completado el tramo de interés a una velocidad superior a la estipulada (8 m/s). Como se puede apreciar, en la multa aparecen datos relativos sobre la fecha y la hora de la denuncia, el lugar, el hecho que se denuncia, los datos del vehículo y del propietario, y una prueba fotográfica de la infracción.

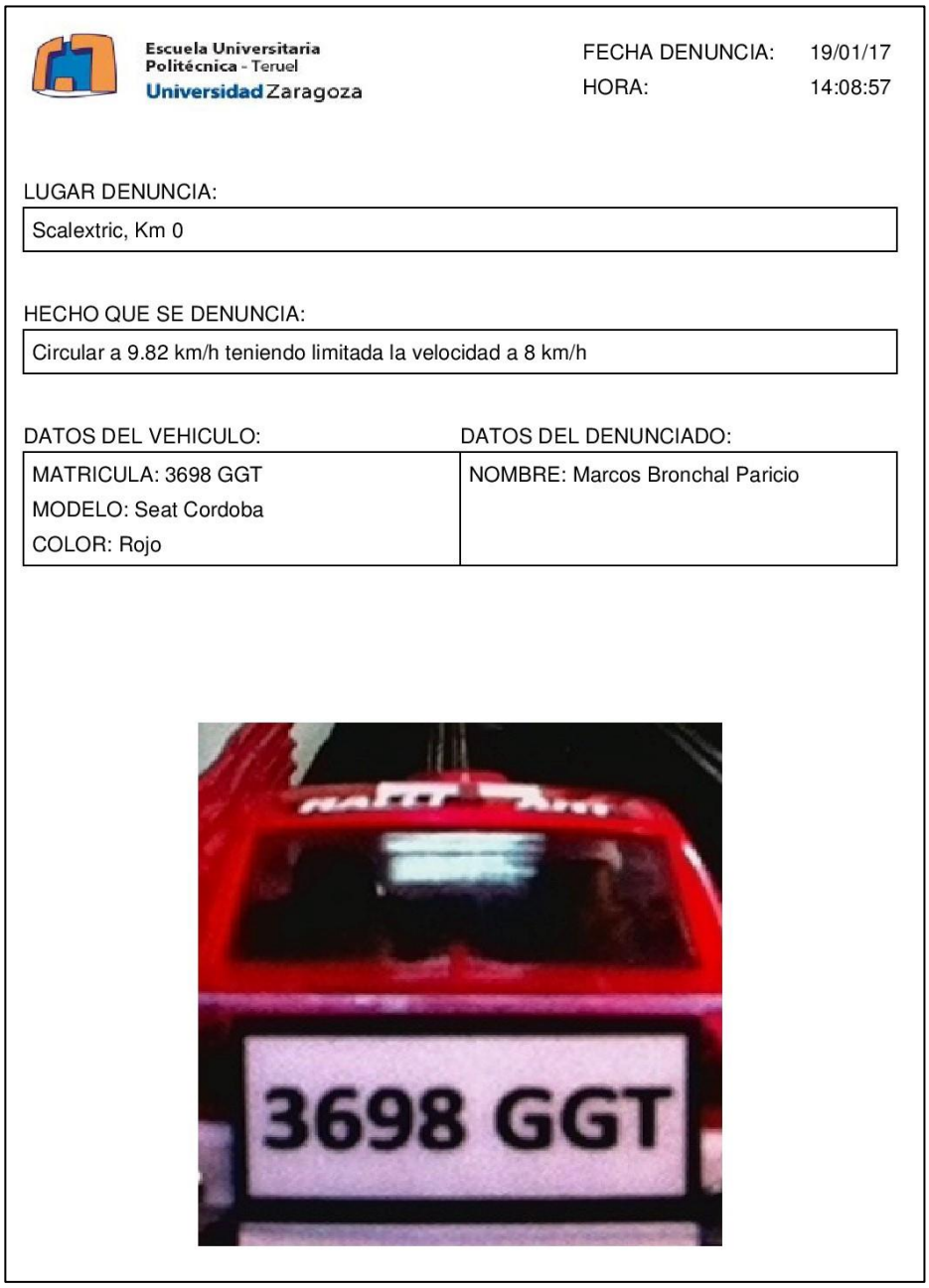

Figura 44: Ejemplo de multa

9. Conclusiones y trabajo futuro

Para finalizar la memoria, y como conclusión final, se puede afirmar que el alcance del proyecto ha sido el esperado, es decir, que la aplicación es capaz de calcular la velocidad media de un coche de *Scalextric* en un determinado tramo, identificando correctamente los caracteres de su matrícula cuando este circula a una velocidad inadecuada (teniendo en cuenta las tasas de éxito).

Por otro lado, la configuración de los receptores IR como interrupciones hacen que el sistema sea bastante robusto, puesto que el paso de un vehículo por cualquiera de las dos barreras infrarrojas es un evento que sucede de manera no programada. Cuando se produce este evento, se requiere una atención inmediata por parte del programa principal, ya que es prioritario atender al paso de un vehículo por una determinada barrera infrarroja, que seguir con el procesado digital de imagen de una ráfaga de fotografías anterior.

A continuación, se detallan algunos aspectos relevantes sobre el funcionamiento del prototipo:

- Es imprescindible que el sistema de iluminación esté enfocado correctamente para que la tasa de éxito en el OCR de la matrícula sea la más alta posible.
- Se deberá comprobar periódicamente que las dos parejas emisor-receptor IR estén perfectamente sincronizadas, es decir, que la frecuencia de excitación de los LED's IR sea la adecuada para que los correspondientes receptores IR funcionen correctamente. Una frecuencia de emisión errónea provoca un mal comportamiento del receptor IR, lo que implica que el funcionamiento de la aplicación no sea el deseado.
- Se ha optado por inicializar la LCD cada vez que se cambia el mensaje en la misma, puesto que en caso de no hacerlo, la LCD mostraba caracteres erróneos producidos por la captación del ruido provocado por el rozamiento de las escobillas del coche con la pista de *Scalextric*.
- El color oscuro del coche de *Scalextric* (rojo) ha facilitado la labor del procesado de imagen. Si el color del coche hubiese sido más claro (blanco en el peor de los casos), la delimitación de la matrícula podría haber sido más laboriosa, es decir, puesto que para la obtención de la región que encierra a la matrícula se ha utilizado el área mayor de los objetos de la imagen binarizada (regiones blancas), el área de un techo blanco podría ser comparable con el área de la matrícula. Al tratarse de un coche rojo, y debido a los niveles de intensidad de las imágenes, la binarización *Otsu* asocia píxeles negros a la región del techo del coche, quedando la matrícula como objeto de mayor área.

A modo personal, la realización de este Trabajo Fin de Grado ha supuesto todo un reto para mí. La existencia de problemas muy diversos (mecánicos, electrónicos, eléctricos, de ruido electromagnético, de procesado de imagen, de diseño electrónico, de programación, etcétera) me ha hecho ponerme a prueba, enfrentándome a un problema propio de la vida real. Por otro lado, he adquirido conocimientos básicos de programación en *Python*.

Como trabajo futuro, se han propuesto las siguientes ideas para mejorar el prototipo:

- Implementación de otro sistema de identificación de vehículos al inicio del tramo de interés. Este sistema de identificación podría ser otra *Raspberry Pi* y otra *Picamera*. La identificación de vehículos al principio y al final del tramo posibilitaría la circulación múltiple de vehículos por dicho tramo.
- Implementación de dos microcontroladores, uno para la gestión de la LCD, y otro para la gestión del tiempo, comunicados con la *Raspberry Pi* mediante el bus I2C. Su implementación aumentaría notablemente el rendimiento del prototipo, puesto que la *Raspberry Pi* sólo se encargaría del procesamiento de las imágenes capturadas por la *Picamera*, además de atender a los microcontroladores. También se reduciría el cableado de la LCD, y se lograría una mayor precisión en el cálculo de la velocidad del coche, puesto que un microcontrolador es muy preciso en el cálculo de tiempos.
- Diseño de un sistema de iluminación con modo *flash*, y mejora de la función OCR, para incrementar la tasa de éxito en la identificación de la matrícula.

10. Bibliografía

[1] CEA, «Tipos de radares. ¿Cuáles son y dónde están?» [En línea]. Disponible en: http://www.cea-online.es/reportajes/radares.asp. [Accedido: 19-ene-2017].
[2] «Efecto Doppler», *Wikipedia, la enciclopedia libre*. 04-nov-2016.
[3] «Raspberry Pi», *Wikipedia, la enciclopedia libre*. 09-ene-2017.
[4] «Raspberry Pi - Teach, Learn, and Make with Raspberry Pi», *Raspberry Pi*. [En línea]. Disponible en: https://www.raspberrypi.org/. [Accedido: 13-ene-2017].
[5] Raspberry Pi Foundation, «Camera Module - Raspberry Pi». [En línea]. Disponible en: https://www.raspberrypi.org/products/camera-module-v2/. [Accedido: 19-ene-2017].
[6] Dave Jones, «Picamera — Picamera 1.12 documentation», 2014. [En línea]. Disponible en: http://picamera.readthedocs.io/en/release-1.12/. [Accedido: 11-ene-2017].
[7] FotoNostra, «Nitidez». [En línea]. Disponible en: http://www.fotonostra.com/glosario/nitidez.htm. [Accedido: 19-ene-2017].
[8] «Contraste», *Wikipedia, la enciclopedia libre*. 10-dic-2016.
[9] Raspberry Pi Foundation, «Raspberry Pi Camera Module - Raspberry Pi Documentation». [En línea]. Disponible en: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md. [Accedido: 19-ene-2017].
[10] Ana María López Torres, «Imagen. Conceptos básicos (I)». Apuntes visión por computador. Ingeniería Electrónica y Automática, 2015.
[11] Ana María López Torres, «Imagen. Conceptos básicos (II)». Apuntes visión por computador. Ingeniería Electrónica y Automática, 2015.
[12] «Resolución de imagen», *Wikipedia, la enciclopedia libre*. 26-oct-2016.
[13] «Circuito integrado 555», *Wikipedia, la enciclopedia libre*. 22-nov-2016.
[14] «NumPy — NumPy». [En línea]. Disponible en: http://www.numpy.org/. [Accedido: 19-ene-2017].
[15] «OpenCV | OpenCV». [En línea]. Disponible en: http://opencv.org/. [Accedido: 19-ene-2017].
[16] Alexander Mordvintsev & Abid K., «OpenCV-Python Tutorials — OpenCV-Python Tutorials 1 documentation». [En línea]. Disponible en: http://opencv24-python-tutorials.readthedocs.io/en/stable/py_tutorials/py_tutorials.html. [Accedido: 19-ene-2017].
[17] «The Image Module». [En línea]. Disponible en: http://effbot.org/imagingbook/image.htm. [Accedido: 02-ene-2017].
[18] «Time access and conversions — Python 2.7.13 documentation». [En línea]. Disponible en: https://docs.python.org/2/library/time.html. [Accedido: 19-ene-2017].
[19] R. Smith, «An Overview of the Tesseract OCR Engine», en *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, 2007, vol. 2, pp. 629-633.

[20] R. Saxena, «Optical Character Recognition in Python», *Polymath Bequeath*, 04-oct-2014. .
[21] «Shutil — High-level file operations — Python 2.7.13 documentation». [En línea]. Disponible en: https://docs.python.org/2/library/shutil.html. [Accedido: 05-ene-2017].
[22] Mike Driscoll, «A Simple Step-by-Step Reportlab Tutorial», *Mouse Vs. The Python*.
[23] «Os — Miscellaneous operating system interfaces — Python 2.7.13 documentation». [En línea]. Disponible en: https://docs.python.org/2/library/os.html. [Accedido: 19-ene-2017].
[24] Chris Hager, «RPIO's documentation! — RPIO 0.10.0 documentation», 2013. [En línea]. Disponible en: https://pythonhosted.org/RPIO/index.html. [Accedido: 02-ene-2017].
[25] «Librería LCD 16×2 con Python», *botboss*. .
[26] «Threading — Higher-level threading interface — Python 2.7.13 documentation». [En línea]. Disponible en: https://docs.python.org/2/library/threading.html. [Accedido: 05-ene-2017].
[27] Ana María López Torres, «Introducción. ¿Qué es la visión por computador?» Apuntes visión por computador. Ingeniería Electrónica y Automática, 2015.
[28] Ana María López Torres, «Segmentación. Detección de bordes y regiones (I)». Apuntes visión por computador. Ingeniería Electrónica y Automática, 2015.
[29] Otsu N., «A Threshold Selection Method from Gray-Level Histograms», vol. SMC-9, n.º 1, pp. 62-66, 1979.
[30] Guillermo Palacios, «Principios de diseño frente a EMI». Apuntes diseño electrónico. Ingeniería Electrónica y Automática, 2015.

TERCERA PARTE.

ANEXOS

Marcos Bronchal Paricio. Febrero, 2017

Diseño e implementación de un prototipo de radar de tramo basado en *Raspberry Pi*

Marcos Bronchal Paricio. Febrero, 2017

ANEXO 1. Enlaces *web* de los *datasheet*

- *Raspberry Pi B+*

http://docs-europe.electrocomponents.com/webdocs/1310/0900766b813109e6.pdf

- *Procesador BCM2835 Soc*

http://docs-europe.electrocomponents.com/webdocs/1419/0900766b814199b9.pdf

- *PiCamera V2*

http://docs-europe.electrocomponents.com/webdocs/14db/0900766b814db308.pdf

- *Receptor de infrarrojos Vishay TSOP4838 38kHz*

http://docs-europe.electrocomponents.com/webdocs/0e30/0900766b80e30b9e.pdf

- *LED infrarrojo Kodenshi*

http://docs-europe.electrocomponents.com/webdocs/0325/0900766b80325c6f.pdf

- *Timer LM555*

http://docs-europe.electrocomponents.com/webdocs/12cf/0900766b812cfcbd.pdf

- *LCD 16x2 ERM1602-6*

http://www.buydisplay.com/download/manual/ERM1602-6_Series_Datasheet.pdf

- *LED Kingbright azul*

http://docs-europe.electrocomponents.com/webdocs/1395/0900766b813955f4.pdf

- *Zumbador Piezoeléctrico Kingstate KPEG242*

http://docs-europe.electrocomponents.com/webdocs/068f/0900766b8068f3a2.pdf

ANEXO 2. Código del programa principal en *Python*

Fichero: *radar.py*

```
#########################################################################
#                                                                       #
#   ESCUELA UNIVERSITARIA POLITECNICA DE TERUEL                         #
#   GRADO EN INGENIERIA ELECTRONICA Y AUTOMATICA                        #
#   TRABAJO DE FIN DE GRADO                                             #
#   CURSO: 2016/2017                                                    #
#                                                                       #
#   RADAR DE TRAMO RASPBERRY PI                                         #
#                                                                       #
#   Fichero : radar.py                                                  #
#   Autor   : Marcos Bronchal Paricio                                   #
#   Fecha   : Febrero de 2017                                           #
#                                                                       #
#   Breve descripcion: funcionamiento de un radar de tramo. Calculo de  #
#   la velocidad de un coche, reconocimiento de matricula, y generacion#
#   de multa en PDF                                                     #
#                                                                       #
#   Acentos eliminados intencionadamente                                #
#                                                                       #
#########################################################################

# Import
import cv2                                  # Libreria imagenes OpenCV
import PIL                                  # Libreria imagenes PIL
import picamera                             # RaspiCam
import pytesseract                          # OCR
import numpy as np                          # Matrices
import time                                 # Control tiempo
from reportlab.pdfgen import canvas         # Generacion PDF
import RPi.GPIO as GPIO                     # GPIO Raspberry Pi
import os                                   # Directorios
import shutil                               # Directorios
import LCDLIB16x2 as LCD                    # LCD
import threading                            # Hilos

# Variables globales
nombre_conductor = []
matricula_conductor = []
modelo_coche = []
color_coche = []
tiempo1 = 0
tiempo2 = 0
th = 0
camera = 0
rebotes_sensor1 = 0
rebotes_sensor2 = 0
i = 0
j = 0
k = 0
vel_media = []
hora_multa = []
fecha_multa = []
directorio_trabajo = ""
```

```
directorio_multas = ""

# Parametros configuracion
nombre_fichero = 'base_datos.txt'
texto1_l1_LCD = "VEL. CONTROLADA "
texto1_l2_LCD = "POR RADAR TRAMO"
texto2_LCD = "MODERE VELOCIDAD"
retardo_rebotes = 0.08              # En segundos
tiempo_mensaje_LCD = 3              # En segundos
vel_umbral = 8                      # En km/h
longitud_tramo = 3.00               # En metros
num_fotos_rafaga = 4                # Numero fotos de cada rafaga
pin_sensor1 = 5                     # GPIO5 Raspberri Pi
pin_sensor2 = 13                    # GPIO13 Raspberri Pi
pin_zumbador = 2                    # GPIO2 Raspberri Pi
pin_LED = 4                         # GPIO4 Raspberri Pi
esq_izq_vert = 455                  # Pixel esquina sup izq vertical
esq_izq_horiz = 540                 # Pixel esquina sup izq horizontal
ancho_matri = 250                   # Longitud en pixeles matricula
porc_alto_ocr = 10.0/100            # Porcentaje eliminar bordes alto
porc_largo_ocr = 1.5/100            # Porntaje eliminar bordes ancho

# Escribir 1er mensaje LCD
def escribir_texto_LCD(texto_l1, texto_l2):

    global th

    LCD.lcd_init()  # Inicializar LCD

    LCD.lcd_string(texto_l1, LCD.LINE_1)    # Escribir mensaje linea 1
    LCD.lcd_string(texto_l2, LCD.LINE_2)    # Escribir mensaje linea 2

    # Hilo ejecuta funcion despues de un intervalo
    th = threading.Timer(tiempo_mensaje_LCD, escribir_fecha_hora_LCD,
                         [texto2_LCD])

    # Hilo de utilidad para salir con KeyboardInterrupt
    th.daemon = True

    th.start()          # Iniciar la actividad del hilo

    return

# Escribir 2o mensaje LCD
def escribir_fecha_hora_LCD(texto):

    global th

    LCD.lcd_init()

    LCD.lcd_string(texto, LCD.LINE_1)
    LCD.lcd_string(time.strftime(" %H:%M %d/%m/%y"), LCD.LINE_2)

    th = threading.Timer(tiempo_mensaje_LCD, escribir_texto_LCD,
                         [texto1_l1_LCD, texto1_l2_LCD])
    th.daemon = True
    th.start()
```

```
    return

# Leer base datos
def leer_fichero(nom_fichero):

    nom_conductor = []
    matr_conductor = []
    mod_coche = []
    col_coche = []

    fichero = open(nom_fichero, 'r')    # Abrir fichero para lectura

    b = 0

    for linea in fichero:   # Leer datos linea a linea

        datos_conductor = linea.strip('\n').strip('\r').split('\t')
        nom_conductor.insert(b, datos_conductor[0])
        matr_conductor.insert(b, datos_conductor[1])
        mod_coche.insert(b, datos_conductor[2])
        col_coche.insert(b, datos_conductor[3])

        b += 1

    fichero.close() # Cerrar fichero

    return nom_conductor, matr_conductor, mod_coche, col_coche

# Configuracion
def setup():

    print "Configurando parametros..."

    global camera
    global directorio_trabajo
    global directorio_multas

    # Directorios
    directorio_trabajo = os.getcwd()    # Obtener directorio trabajo

    # Comprobar si existe directorio
    if (os.access((directorio_trabajo + '/multas'), os.F_OK)):

        # Borrar directorio no vacio
        shutil.rmtree(directorio_trabajo + '/multas')

    # Crear directorio multas
    os.mkdir(directorio_trabajo + '/multas')
    directorio_multas = (directorio_trabajo + '/multas')

    # RaspiCam
    camera = picamera.PiCamera()

    camera.resolution = (1280, 720)                 # Resolucion
    camera.framerate = 90                           # FPS
    camera.vflip = True                             # Rot vertical
    camera.hflip = True                             # Rot horizontal
    camera.iso = 1600                               # ISO
```

```
    camera.contrast = 100                           # Contraste
    camera.saturation = 0                           # Saturacion
    camera.brightness = 50                          # Brillo
    time.sleep(1)

    camera.shutter_speed = camera.exposure_speed    # Vel exposicion
    camera.exposure_mode = 'auto'
    g = camera.awb_gains                            # Ganancia
    camera.awb_mode = 'auto'
    camera.awb_gains = g

    camera.start_preview()                          # Vista previa
    time.sleep(1)

    # Sensores
    GPIO.setmode(GPIO.BCM)  # BCM GPIO
    GPIO.setwarnings(False) # Warnings deshabilitados

    GPIO.setup(pin_sensor1, GPIO.IN)    # sensor1 --> entrada
    GPIO.setup(pin_sensor2, GPIO.IN)    # sensor2 --> entrada

    # Zumbador
    GPIO.setup(pin_zumbador, GPIO.OUT)  # zumbador --> salida
    GPIO.setup(pin_LED, GPIO.OUT)       # LED --> salida
    GPIO.output(pin_zumbador, GPIO.LOW) # zumbador = 0 V
    GPIO.output(pin_LED, GPIO.LOW)      # LED = 0 V

    # LCD
    LCD.lcd_init()  # Inicializar LCD
    escribir_texto_LCD(texto1_l1_LCD, texto1_l2_LCD)

    # Interrupciones
    GPIO.add_event_detect(pin_sensor1, GPIO.RISING ,
                          callback = interr_sensor1)    # Flanco subida
    GPIO.add_event_detect(pin_sensor2, GPIO.RISING ,
                          callback = interr_sensor2)    # Flanco subida

    print "Configuracion completada"

    return

# Interrupcion sensor1
def interr_sensor1(channel):

    tiempo = time.time()    # Tiempo actual en seg

    global rebotes_sensor1

    if (GPIO.input(pin_sensor1) & (rebotes_sensor1 == 0)):  # Filtro
                                                            # software
        rebotes_sensor1 = 1

        global tiempo1

        tiempo1 = tiempo
        time.sleep(retardo_rebotes)

        rebotes_sensor1 = 0
```

```
    return

# Interrupcion sensor2
def interr_sensor2(channel):

    tiempo = time.time()

    global rebotes_sensor2

    if (GPIO.input(pin_sensor2) & (rebotes_sensor2 == 0)):

        rebotes_sensor2 = 1

        global tiempo1
        global vel_media
        global k
        global hora_multa
        global fecha_multa

        tiempo2 = tiempo

        vel_media.insert(k, ((longitud_tramo / (tiempo2 - tiempo1)))
                         * 3.6)      # Velocidad en km/h

        if (vel_media[k] > vel_umbral):      # Velocidad superior umbral

            global camera

            GPIO.output(pin_zumbador, GPIO.HIGH)     # Zumbador = 3.3 V
            GPIO.output(pin_LED, GPIO.HIGH)          # LED = 3.3 V

            hora_multa.insert(k, time.strftime("%H:%M:%S"))
            fecha_multa.insert(k, time.strftime("%d/%m/%y"))

            b = 0

            # Rafaga fotos
            camera.capture_sequence(['captura%d%d.jpg' % (k, b)
            for b in range(num_fotos_rafaga)], use_video_port = True)

            GPIO.output(pin_zumbador, GPIO.LOW)      # Zumbador = 0 V
            GPIO.output(pin_LED, GPIO.LOW)           # LED = 0 V

            k += 1

        time.sleep(retardo_rebotes)
        rebotes_sensor2 = 0

    return

# Procesado de imagen para reconocimiento de caracteres
def procesar_imagen(img):

    global i
    global j

    alto_img = img.shape[0] # Pixeles alto imagen
```

```
# Zona de interes
roi = img[esq_izq_vert : alto_img, esq_izq_horiz : esq_izq_horiz
          + ancho_matri]
cv2.imwrite('roi%d%d.jpg' % (i, j), roi)

# Escala grises
roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

# Binarizar ROI con OTSU
ret2, bin_roi = cv2.threshold(roi_gray, 0, 255,
                              cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Erosionar para detectar bordes
kernel = np.ones((2, 2), np.uint8)  # Mascara de unos 2x2
ero = cv2.erode(bin_roi, kernel, iterations = 3)

# Detectar contornos
contours, hierarchy = cv2.findContours(ero, cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)

# Detectar areas de los objetos
areas = []

b = 0

while (b < len(contours)):

    area = cv2.contourArea(contours[b])
    areas.insert(b, area)

    b += 1

# Detectar objeto de mayor area (matricula)
cnt = contours[areas.index(max(areas))]

# Encerrar area en rectangulo
x,y,w,h = cv2.boundingRect(cnt)
matricula = roi_gray[y:y+h,x:x+w]

# Eliminar bordes
alto_ocr = matricula.shape[0]       # Pixeles alto matricula
largo_ocr = matricula.shape[1]      # Pixeles largo matricula

matricula = matricula[porc_alto_ocr * alto_ocr : (alto_ocr -
                      2 * porc_alto_ocr * alto_ocr), porc_largo_ocr
                      * largo_ocr : (largo_ocr - 2 * porc_largo_ocr
                      * largo_ocr)]

# Binarizar matricula OTSU
ret2, matricula_bin = cv2.threshold(matricula, 0, 255,
                                    cv2.THRESH_BINARY + cv2.THRESH_OTSU)
cv2.imwrite('matricula_bin.jpg', matricula_bin) # Guardar imagen

# Reconocimiento caracteres
foto = PIL.Image.open('matricula_bin.jpg')
texto = pytesseract.image_to_string(foto)
```

```
    return texto

# Identificar matricula correcta
def iden_matricula(matr):

    global i    # Guardar ROI para multa

    encontrado = 0

    b = 0

    # Coinciden 2 matriculas
    while ((b < (num_fotos_rafaga - 1)) & (encontrado != 1)):

        c = 1

        while (((b + c) < (len(matr))) & (encontrado != 1)):

            if ((not(cmp(matr[b], matr[b + c]))) & (len(matr[b]) == 8)):

                matricula_iden = matr[b]

                roi = cv2.imread('roi%d%d.jpg' % (i, b))
                cv2.imwrite('roi.jpg', roi)

                encontrado = 1

            c += 1

        b += 1

    # No coinciden 2 matriculas, pero alguna tiene longitud 8
    if (encontrado != 1):

        b = 0

        while ((b < len(matr)) & (encontrado != 1)):

            if (len(matr[b]) == 8):

                matricula_iden = matr[b]

                roi = cv2.imread('roi%d%d.jpg' % (i, b))
                cv2.imwrite('roi.jpg', roi)

                encontrado = 1

            b += 1


        # Matricula no encontrada
        if (encontrado != 1):

            matricula_iden = "---- ---"

            roi = cv2.imread('roi%d%d.jpg' % (i, 0))
            cv2.imwrite('roi.jpg', roi)
```

```
    return matricula_iden

# Identificar infractor
def iden_infractor(matricula):

    global nombre_conductor
    global matricula_conductor
    global modelo_coche
    global color_coche

    encontrado = 0

    b = 0

    # Coincide la matricula con alguna de la base de datos
    while ((b < len(matricula_conductor)) & (encontrado != 1)):

        if (not(cmp(matricula, matricula_conductor[b]))):

            nombre = nombre_conductor[b]
            modelo = modelo_coche[b]
            color = color_coche[b]
            encontrado = 1

        b += 1

    if (encontrado != 1):

        nombre = "Conductor no registrado"
        modelo = "Modelo no registrado"
        color = "Color no registrado"

    return (nombre, modelo, color)

# Redactar multa PDF
def crear_multa(matricula, nombre, modelo, color):

    global i

    # Crear PDF
    c = canvas.Canvas("multa%d.pdf" % i)

    # Cabezera multa
    c.drawImage("logo_eupt.jpg", 50, 750, width=200, height=50)
    c.drawString(370, 780,"FECHA DENUNCIA: ")
    c.drawString(500, 780, "%s" % fecha_multa[i])
    c.drawString(370, 760, "HORA: ")
    c.drawString(500, 760, "%s" % hora_multa[i])

    # Lugar de la denuncia
    c.drawString(50, 700, "LUGAR DENUNCIA: ")
    c.drawString(55, 678, "Scalextric, Km 0")
    c.rect(50,670,500,25)

    # Motivo de la denuncia
    c.drawString(50, 630, "HECHO QUE SE DENUNCIA: ")
    c.drawString(55, 608, "Circular a %.2f km/h teniendo limitada la"
                 " velocidad a %d km/h" % (vel_media[i], vel_umbral))
```

```
    c.rect(50,600,500,25)

    # Datos del conductor
    c.drawString(50, 560, "DATOS DEL VEHICULO: ")
    c.drawString(55, 538, "MATRICULA: %s" % matricula)
    c.drawString(55, 518, "MODELO: %s" % modelo)
    c.drawString(55, 498, "COLOR: %s" % color)
    c.rect(50,490,250,65)
    c.drawString(300, 560, "DATOS DEL DENUNCIADO: ")
    c.drawString(305, 538, "NOMBRE: %s" % nombre)
    c.rect(300,490,250,65)

    # Foto radar
    c.drawImage("roi.jpg", 150, 100, width=300, height=300)

    # Guardar PDF
    c.save()

    # Mover multa al directorio multas
    shutil.move((directorio_trabajo + ('/multa%d.pdf' % i)),
                (directorio_multas + ('/multa%d.pdf' % i)))

    return

# Eliminar imagenes procesadas (temporales)
def eliminar_imagenes_procesadas():

    global i
    global j

    j -= 1

    while (j >= 0):
        os.remove(directorio_trabajo + ('/captura%d%d.jpg' % (i, j)))
        os.remove(directorio_trabajo + ('/roi%d%d.jpg' % (i, j)))

        j -= 1

    os.remove(directorio_trabajo + '/matricula_bin.jpg')
    os.remove(directorio_trabajo + '/roi.jpg')

    return

# Funcion principal
def main():

    global i
    global j
    global nombre_conductor
    global matricula_conductor
    global modelo_coche
    global color_coche

    setup()

    (nombre_conductor, matricula_conductor, modelo_coche,
     color_coche) = leer_fichero(nombre_fichero)
```

```
# Bucle principal del programa
    while (True):

        matricula = []

        j = 0

        while (j < num_fotos_rafaga):

            #Leer imagen
            imagen = cv2.imread('captura%d%d.jpg' % (i, j))

            while (imagen == None):
                imagen = cv2.imread('captura%d%d.jpg' % (i, j))

            matricula.insert(j, procesar_imagen(imagen))

            j += 1

        matricula_identificada = iden_matricula(matricula)

        (nombre_infract, modelo_coche_infract,
         color_coche_infract) = iden_infractor(matricula_identificada)

        crear_multa(matricula_identificada, nombre_infract,
                    modelo_coche_infract, color_coche_infract)

        eliminar_imagenes_procesadas()

        i += 1

if __name__ == '__main__':

    try:
        main()
    except KeyboardInterrupt:
        pass
    finally:
        GPIO.cleanup()
```

ANEXO 3. Código de la librería LCD en *Python*

Fichero: *LCDLIB16x2.py*

```
##########################################################################
#                                                                        #
#   ESCUELA UNIVERSITARIA POLITECNICA DE TERUEL                          #
#   GRADO EN INGENIERIA ELECTRONICA Y AUTOMATICA                         #
#   TRABAJO DE FIN DE GRADO                                              #
#   CURSO: 2016/2017                                                     #
#                                                                        #
#   RADAR DE TRAMO RASPBERRY PI                                          #
#                                                                        #
#   Fichero : LCDLIB16x2.py                                              #
#   Autor   : Marcos Bronchal Paricio                                    #
#   Fecha   : Febrero de 2017                                            #
#                                                                        #
#   Breve descripcion: funcionamiento de una LCD 16x2 para transmitir    #
#   informacion en paquetes de 4 bits (D4 - D5)                          #
#                                                                        #
#   Acentos eliminados intencionadamente                                 #
#                                                                        #
##########################################################################

# Import
import RPi.GPIO as GPIO      # GPIO Raspberry Pi
import time                  # Control tiempo

# GPIO Raspberry Pi
LCD_RS = 14      # GPIO14 Raspberri Pi
LCD_E = 15       # GPIO15 Raspberri Pi
LCD_D4 = 18      # GPIO18 Raspberri Pi
LCD_D5 = 23      # GPIO23 Raspberri Pi
LCD_D6 = 24      # GPIO24 Raspberri Pi
LCD_D7 = 25      # GPIO25 Raspberri Pi

# Constantes
LCD_WIDTH = 16  # Maximo de caracteres por linea
LCD_CHR = True
LCD_CMD = False

LINE_1 = 0x80   # LCD direccion RAM de la 1a linea
LINE_2 = 0xC0   # LCD direccion RAM de la 2a linea

# Constantes de tiempo
E_PULSE = 0.0005     # 500 us
E_DELAY = 0.0005     # 500 us

GPIO.setwarnings(False)          # Warnings deshabilitados
GPIO.setmode(GPIO.BCM)           # BCM GPIO
GPIO.setup(LCD_E, GPIO.OUT)      # E --> salida
GPIO.setup(LCD_RS, GPIO.OUT)     # RS --> salida
GPIO.setup(LCD_D4, GPIO.OUT)     # DB4 --> salida
GPIO.setup(LCD_D5, GPIO.OUT)     # DB5 --> salida
GPIO.setup(LCD_D6, GPIO.OUT)     # DB6 --> salida
GPIO.setup(LCD_D7, GPIO.OUT)     # DB7 --> salida
```

```
# Inicializacion Display
def lcd_init():

    lcd_byte(0x33,LCD_CMD)  # 110011 Inicializacion
    lcd_byte(0x32,LCD_CMD)  # 110010 Inicializacion
    lcd_byte(0x06,LCD_CMD)  # 000110 Mover cursor
    lcd_byte(0x0C,LCD_CMD)  # 001100 Display ON, Cursor OFF, Blink OFF
    lcd_byte(0x28,LCD_CMD)  # 101000 Long. datos, n lineas, tamano car
    lcd_byte(0x01,LCD_CMD)  # 000001 Limpiar Display
    time.sleep(E_DELAY)     # Retardo E_DELAY

# Enviar datos pins
def lcd_byte(bits, mode):

    GPIO.output(LCD_RS, mode)    # RS

    # High bits
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)

    if bits&0x10==0x10:
        GPIO.output(LCD_D4, True)

    if bits&0x20==0x20:
        GPIO.output(LCD_D5, True)

    if bits&0x40==0x40:
        GPIO.output(LCD_D6, True)

    if bits&0x80==0x80:
        GPIO.output(LCD_D7, True)

    lcd_toggle_enable()

    # Low bits
    GPIO.output(LCD_D4, False)
    GPIO.output(LCD_D5, False)
    GPIO.output(LCD_D6, False)
    GPIO.output(LCD_D7, False)

    if bits&0x01==0x01:
        GPIO.output(LCD_D4, True)

    if bits&0x02==0x02:
        GPIO.output(LCD_D5, True)

    if bits&0x04==0x04:
        GPIO.output(LCD_D6, True)

    if bits&0x08==0x08:
        GPIO.output(LCD_D7, True)

    lcd_toggle_enable()
```

```
# Habiilitar Toggle
def lcd_toggle_enable():

    time.sleep(E_DELAY)             # Retardo E_DELAY
    GPIO.output(LCD_E, True)
    time.sleep(E_PULSE)             # Retardo E_PULSE
    GPIO.output(LCD_E, False)
    time.sleep(E_DELAY)             # Retardo E_DELAY

# Escribir Display
def lcd_string(message, line):

    message = message.ljust(LCD_WIDTH," ")

    lcd_byte(line, LCD_CMD)

    for i in range(LCD_WIDTH):
        lcd_byte(ord(message[i]),LCD_CHR)
```

ANEXO 4. Base de datos del prototipo

Fichero: *base_datos.txt*

#	Nombre conductor	Matrícula coche	Modelo coche	Color coche
1	Marcos Bronchal Paricio	3698 GGT	Seat Cordoba	Rojo
2	Paloma Bronchal Paricio	9339 GVP	Seat Cordoba	Rojo
3	Amado Bronchal Martinez	5989 FXB	Seat Cordoba	Rojo
4	Concepcion Paricio Dolz	7799 HGT	Seat Cordoba	Rojo
5	Pilar Paricio Tomas	9364 BLS	Seat Cordoba	Rojo
6	Jose Paricio Tomas	9369 FFR	Seat Cordoba	Rojo
7	Javier Alvaro Paricio	1556 GST	Seat Cordoba	Rojo
8	Daniel Alvaro Paricio	6739 DGH	Seat Cordoba	Rojo
9	Beatriz Alvaro Paricio	3990 FHD	Seat Cordoba	Rojo
10	Patricia Alvaro Paricio	0113 FJD	Seat Cordoba	Rojo
11	Maria Alvaro Paricio	1278 FDD	Seat Cordoba	Rojo
12	Ruben Paricio Hernandez	9223 JRD	Seat Cordoba	Rojo
13	Maria Paricio Hernandez	6544 HCR	Seat Cordoba	Rojo
14	Paula Alvaro Tabernero	1474 GFG	Seat Cordoba	Rojo
15	Lara Alvaro Tabernero	4963 CBD	Seat Cordoba	Rojo
16	Ivan Alvaro Moron	8896 CDC	Seat Cordoba	Rojo
17	Marcos Alvaro Moron	5477 GTV	Seat Cordoba	Rojo
18	Aroa Garcia Alvaro	4697 BBV	Seat Cordoba	Rojo
19	Amara Garcia Alvaro	6423 GTH	Seat Cordoba	Rojo
20	Adrian Villamon Alvaro	7622 BBD	Seat Cordoba	Rojo
21	Olivia Villamon Alvaro	8736 CRT	Seat Cordoba	Rojo
22	Victoria Yus Paricio	1975 BVL	Seat Cordoba	Rojo
23	Raquel Yus Paricio	6223 GFH	Seat Cordoba	Rojo
24	Jorge Yus Paricio	1507 GJV	Seat Cordoba	Rojo

25	Jose Yus Paricio	0913 CFT	Seat Cordoba	Rojo
26	Andres Yus Belenguer	5693 BYT	Seat Cordoba	Rojo
27	Adrian Lazaro Dalda	9985 CCR	Seat Cordoba	Rojo
28	Pablo Javiva Ubeda	0007 BLF	Seat Cordoba	Rojo
29	Victor Lazaro Cebrian	6963 HDS	Seat Cordoba	Rojo
30	Alejandro Dolz Bellido	4598 HHS	Seat Cordoba	Rojo
31	Victor Rubio Parrilla	7177 GTC	Seat Cordoba	Rojo
32	Raquel Torrijo Castelar	6380 HHG	Seat Cordoba	Rojo
33	Eva Torrijo Castelar	8869 JJD	Seat Cordoba	Rojo
34	Rocio Alegre Pascual	3399 GYX	Seat Cordoba	Rojo
35	Ester Alegre Pascual	3971 DLZ	Seat Cordoba	Rojo
36	Adrian Galiando Gomez	6587 BLL	Seat Cordoba	Rojo
37	Ruben Lazaro Dalda	3365 DTR	Seat Cordoba	Rojo
38	Jorge Lazaro Cebrian	6422 HGC	Seat Cordoba	Rojo
39	Angel Peralta Romero	1133 HFF	Seat Cordoba	Rojo
40	Sergio Barea Dolz	7143 HFV	Seat Cordoba	Rojo

Tabla 21: Base de datos del prototipo

ANEXO 5. Galería fotográfica del prototipo

Figura 45: Prototipo completo 1

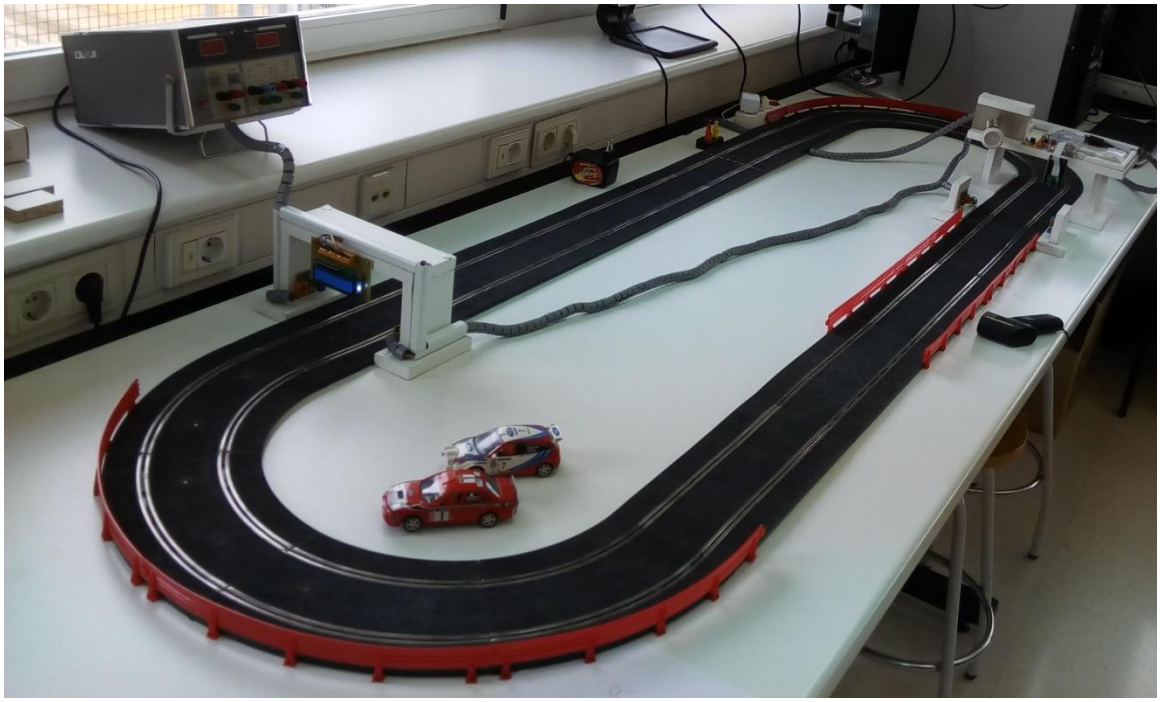

Figura 46: Prototipo completo 2

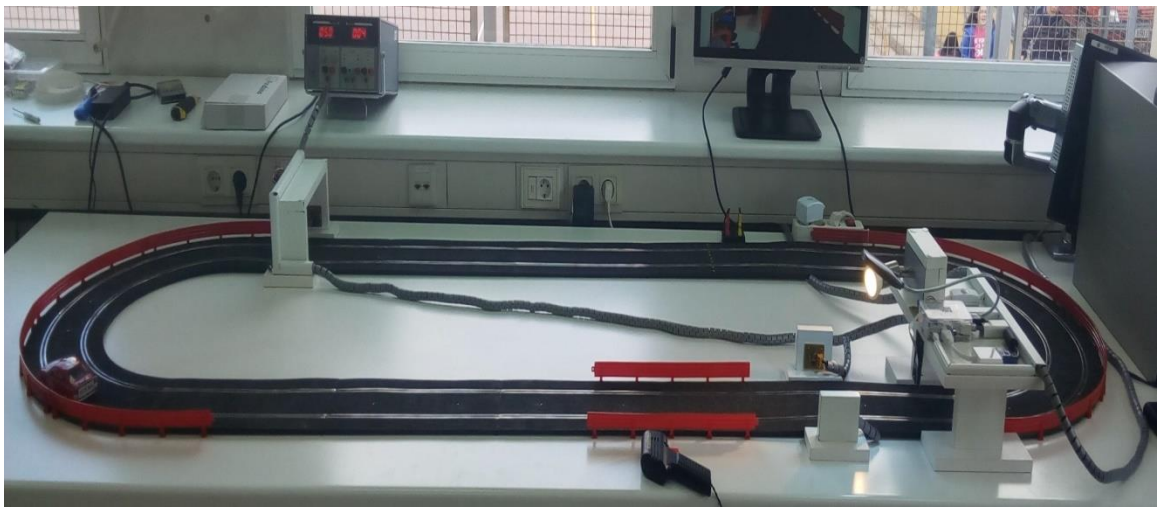

Figura 47: Prototipo completo 3

Figura 48: Puente Raspberry Pi 1

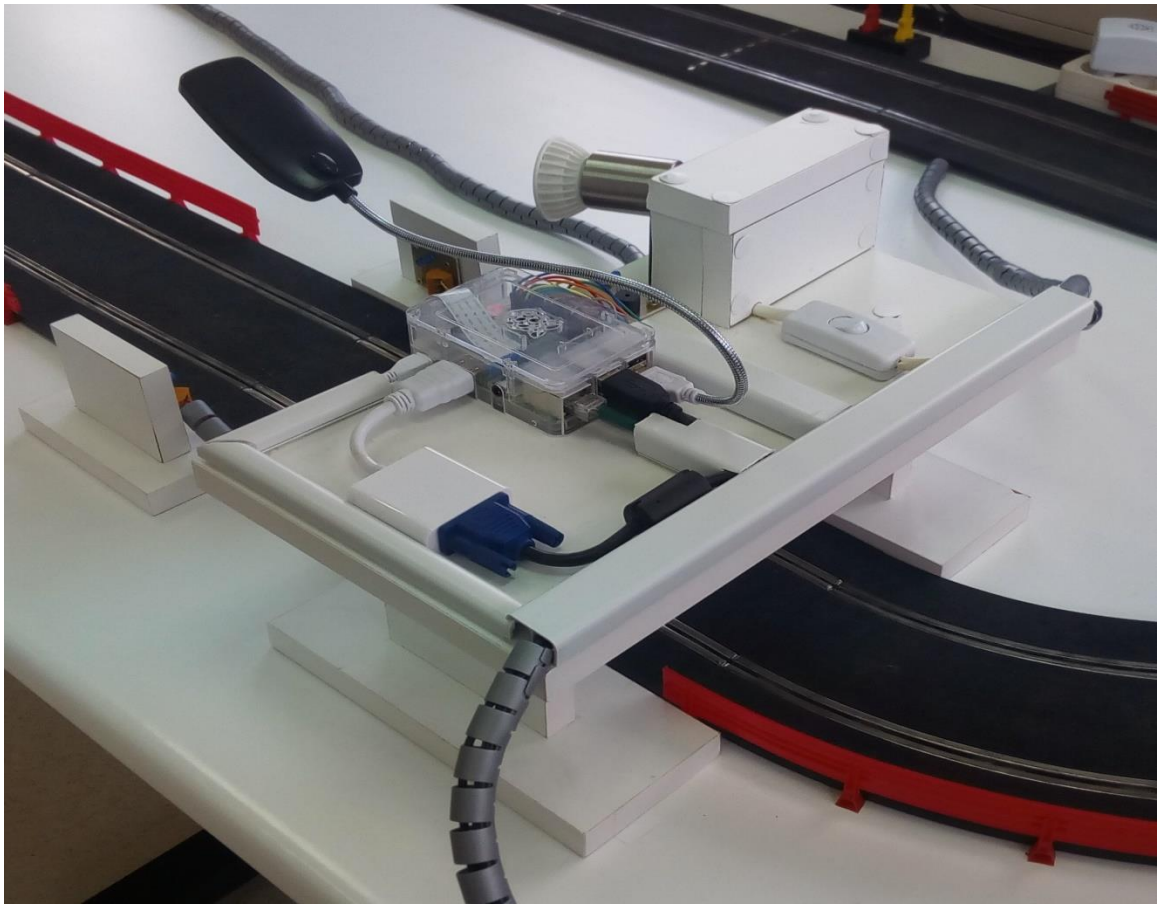

Figura 49: Puente Raspberry Pi 2

Figura 50: Puente Raspberry Pi 3

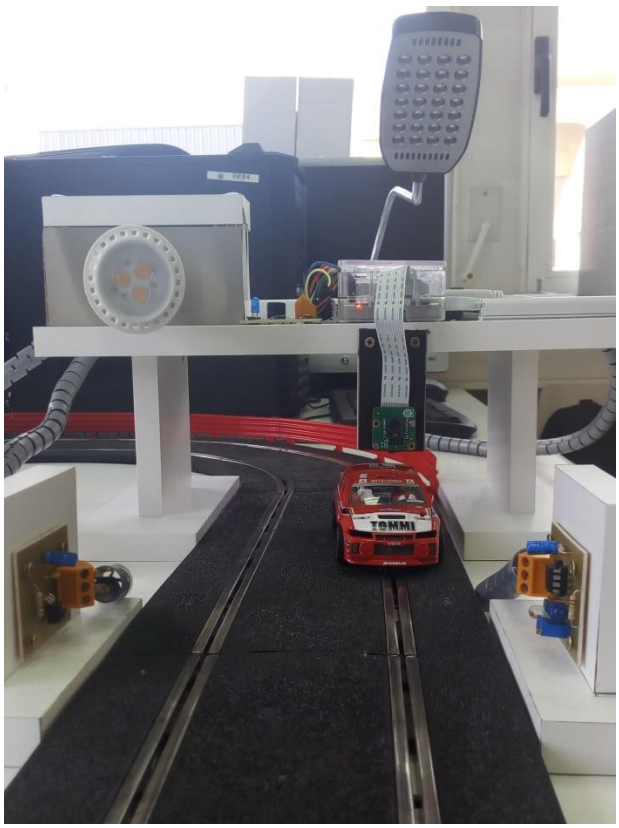

Figura 51: Coche pasando por radar

Figura 52: Coche Scalextric

Figura 53: Coche pasando por el primer puente 1

Figura 54: Coche pasando por el primer puente 2

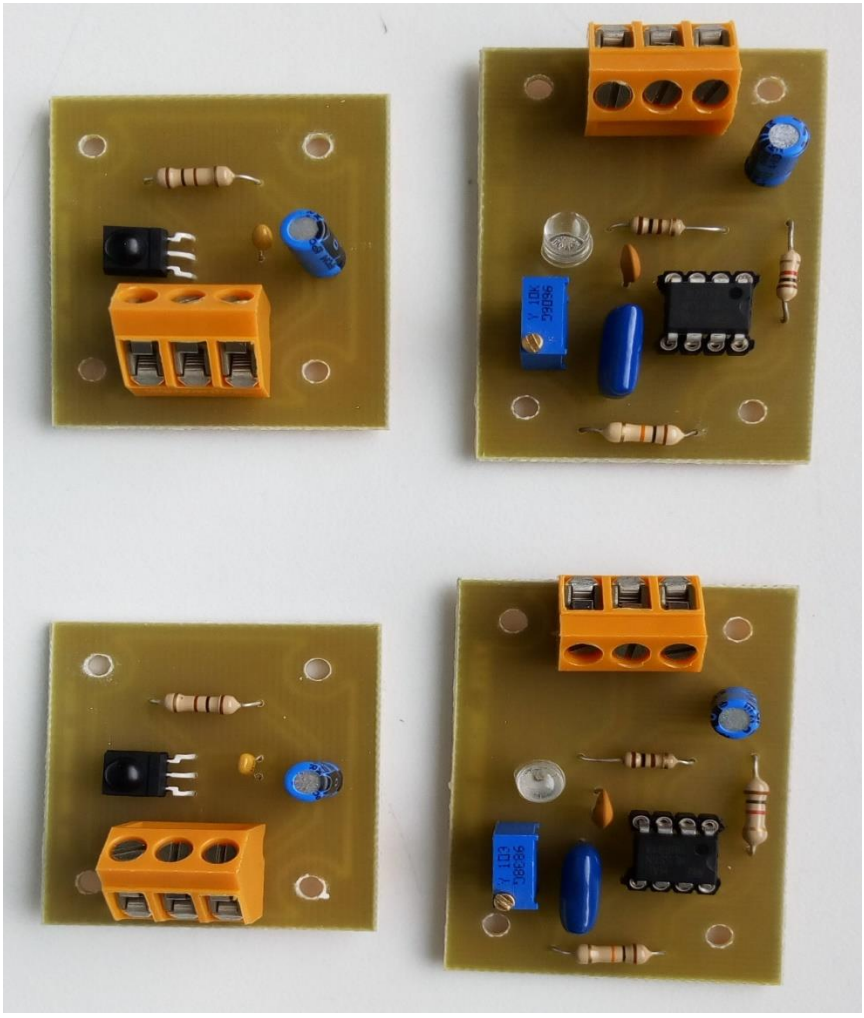

Figura 55: Cara top PCBs emisores y receptores IR

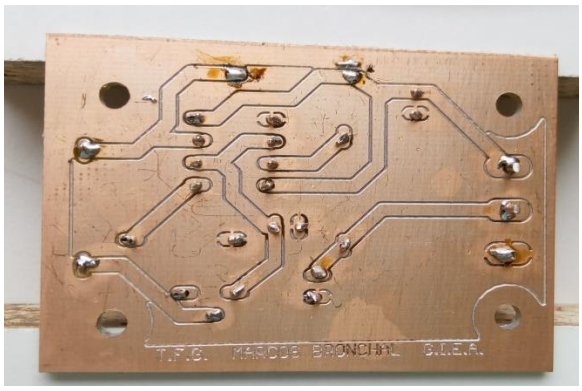

Figura 56: Cara botton PCB emisor IR

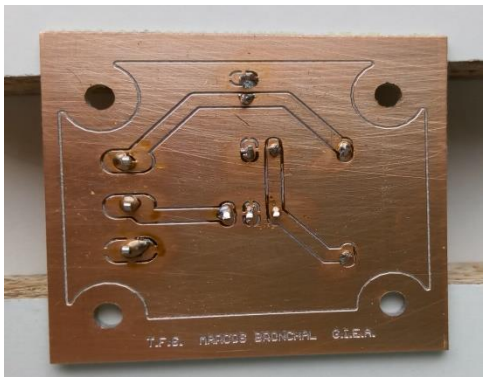

Figura 57: Cara botton PCB receptor IR

Figura 58: Cara top PCB LCD

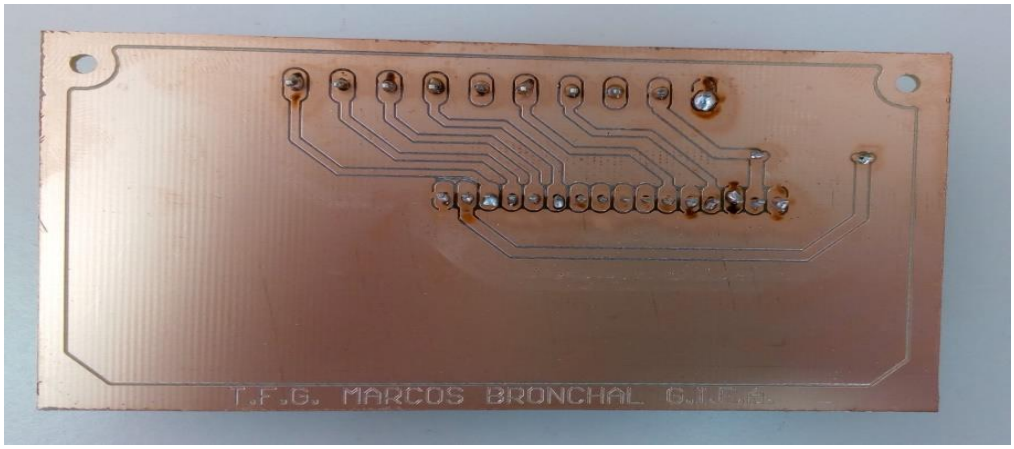

Figura 59: Cara botton PCB LCD

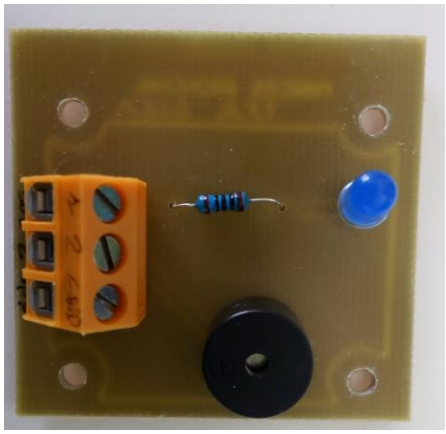

Figura 60: Cara top PCB BUZZ y LED