Universidad Zaragoza
1542

Trabajo Fin de Grado

SISTEMA DE AUTOMATIZACIÓN DE UN PARKING MEDIANTE EMPLEO DE VISIÓN ARTIFICIAL

AUTOMATION SYSTEM FOR A PARKING LOT USING ARTIFICIAL VISION

Autor

Octavio Marín Carnicer

Director

Javier Esteban Escaño

Escuela Universitaria Politécnica La Almunia

Junio 2021

Página intencionadamente en blanco.

Escuela Universitaria
Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

MEMORIA

SISTEMA DE AUTOMATIZACIÓN DE UN PARKING MEDIANTE EMPLEO DE VISIÓN ARTIFICIAL

AUTOMATION SYSTEM FOR A PARKING LOT USING ARTIFICIAL VISION

424-23-90

Autor: Octavio Marín Carnicer

Director: Javier Esteban Escaño

Fecha: 11 2023

Página intencionadamente en blanco.

INDICE DE CONTENIDO BREVE

INDICE DE CONTENIDO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1. RESUMEN

En este trabajo fin de grado se pretende documentar la realización de un proyecto en el que se realiza un sistema de automatización de un parking mediante empleo de visión artificial.

Para ello se analizarán las principales alternativas para la realización de este proyecto y se justificarán las decisiones tomadas.

El objetivo es llegar a cumplir con todos los requerimientos del sistema e implementarlos en un prototipo lo más realista y completo posible.

Así mismo durante la implementación de software del sistema también es posible que algunos de los elementos como el OCR se podrían haber implementado de forma más sencilla con librerías de más alto nivel. Sin embargo uno de los objetivos personales de este proyecto es comprender con la mayor profundidad posible el empleo de tecnologías como la inteligencia artificial, el machine Learning o las redes neuronales.

1.1. PALABRAS CLAVE

- Reconocimiento automático de matrículas
- OCR
- Inteligencia artificial
- Red neuronal
- Python

2. ABSTRACT

In this final degree project, the aim is to document the implementation of a project in which an automation system for a parking lot is developed using computer vision.

To achieve this, the main alternatives for the project will be analyzed, and the decisions made will be justified.

The objective is to meet all the requirements of the system and implement them in a prototype as realistic and complete as possible.

Similarly, during the implementation of the system's software, it is also possible that some elements, such as OCR, could have been implemented more easily using higher-level libraries. However, one of the personal goals of this project is to understand as deeply as possible the use of technologies such as artificial intelligence, machine learning, or neural networks.

2.1. KEY WORDS

- Automatic License Plate Recognition
- OCR (Optical Character Recognition)
- Artificial Intelligence
- Neural Network
- Python

3. INTRODUCCIÓN

El proyecto que se va a realizar tiene como principal característica el empleo de un sistema de visión artificial para el reconocimiento automático de las matrículas de los coches que entren o salgan del parking.

Se ha escogido este proyecto debido al atractivo en estos momentos de las tecnologías derivadas de la inteligencia artificial y sus aplicaciones en la industria.

El objetivo de este proyecto es el de diseñar un sistema autónomo que cumpla con las funciones especificadas.

Para ello se analizaran los antecedentes, se estudiará el marco teórico, se desarrollara un diseño en base a los anteriores puntos y por último se construirá una maqueta que represente de alguna forma los elementos más importantes del sistema.

3.1. OBJETIVOS

- Investigar las tecnologías existentes de reconocimiento de matrículas mediante inteligencia artificial e investigar metodos de implementación de reconocimiento de imágenes mediante inteligencia artificial.
- Diseñar e implementar un algoritmo de inteligencia artificial para el reconocimiento de matrículas.
- Integrar el sistema de reconocimiento de matrículas con una barrera automática y una pantalla que muestre la información de la matricula reconocida.
- Realizar una maqueta.
- Realizar pruebas del sistema y evaluar su rendimiento.

3.2. ANTECEDENTES

Para empezar con este proyecto vamos a comenzar analizando los antecedentes, toda la información analizada antes de comenzarlo que se ha considerado relevante y que ha servido para enfocar el proyecto y su realización.

Para ello partimos de la idea principal de este proyecto, realizar un sistema que sea capaz de identificar los caracteres que contiene una placa de matrícula.

En este punto debemos preguntarnos qué características tiene una placa de matrícula, pues encontramos que las hay de distintos tamaños, formas, colores. Dependiendo de las características del vehículo o remolque al que se encuentran asociadas así como el país donde fueron emitidas y otros factores. Para el proyecto que aquí se realiza tan solo se ha tomado como referencia la placa de matrícula para vehículos generales de España en su versión larga (520 x 110 mm) con sistema nacional alfanumérico posterior al año 2000 que es la que se encuentra vigente en el momento de realización del proyecto.

Una vez definido el objetivo principal analicemos cómo se puede realizar, los sistemas actuales que realizan esta función de forma automática lo hacen mediante la combinación de hardware y software diseñados para tal propósito. Esto se logra mediante un dispositivo de captación de imágenes que transforma la información de las ondas electromagnéticas en información digital y un sistema que posteriormente analice esa información. Esto se puede acompañar de otros dispositivos que añadirán funciones al sistema pero siempre tendremos como mínimo ese sistema principal.

Analizando los sistemas que hay en el mercado nos encontramos con distintos acrónimos que parecen describir lo mismo pero que tienen sutiles diferencias, LPR (License Plate Recognition), ALPR (Automatic License Plate Recognition) y ANPR (Automatic Number Plate Recognition).

Aquí lo que cobra mayor relevancia es la diferencia del LPR con las otras dos, la tecnología LPR consiste en sistema de reconocimiento básico que contiene el mínimo, una cámara que captura imágenes y un sistema que las analiza, siendo este un OCR (Optical Character Recognition) que se implementa normalmente mediante inteligencia artificial. Y después tenemos las tecnologías ALPR y ANPR cuya característica diferenciadora es que poseen conexión con algún tipo de base de datos

ya sea local o remota que les permite además de reconocer las matriculas almacenar datos relacionados con ellas. Los términos ALPR y ANPR aparecen indistintamente para referirse a lo mismo y parece que su uso depende de la persona o empresa que los menciona.(Conure, 2022; Satria, 2021; Survision, 2020)

Así que tenemos por ahora en nuestro sistema tres grandes bloques, una cámara, un sistema OCR y posiblemente una base de datos (esta dependerá de las características específicas de nuestro sistema y sus capacidades).

Profundizando en la cámara y tomando como referencia productos comerciales del mercado podemos ver que estas cumplen con una serie de características. La resolución viene determinada por la distancia de detección, si se quiere reconocer una matrícula que se encuentra lejos se necesitará una cámara con mayor resolución. La velocidad de captura será relevante para aquellas aplicaciones en las que el vehículo se encuentra en movimiento, siendo todavía más crítica cuando hablamos de velocidades elevadas. Así también se ha encontrado información sobre la utilización de sensores infrarrojos en condiciones de baja luminosidad.(Coltec, 2022; Conure, 2022).

El sistema OCR se encarga de transformar una imagen digital a caracteres. Esto se logra mediante software, actualmente mediante algoritmos que usan las empresas o mediante inteligencia artificial (IA).

Toda la información recopilada apunta a que actualmente la mejor manera de realizar un OCR es mediante inteligencia artificial y técnicas de Machine Learning (ML).(Everen, 2023)

La base de datos depende enormemente de la aplicación concreta, por ejemplo puede darse el caso de que en un sistema más complejo haya varios puntos con un sistema de reconocimiento de matrículas y que todos ellos necesiten compartir información, en este caso sería imprescindible que la base de datos estuviera conectada fuera única para todos ellos. También pueden existir casos en los que bastara con la memoria interna del microprocesador o del PC en su defecto que se utilice para realizar el OCR, también dependerá de si se quiere tener un acceso remoto a los datos, mediante una red o internet.

Después de ver en que consiste un sistema de este tipo vamos a acercarnos un poco más a nuestro caso en concreto que es el diseño y la realización de un prototipo. Así pues siendo este un proyecto final de grado tiene peculiaridades que lo alejan de un proyecto con fines comerciales.

Así pues se disponen de recursos limitados, tanto económicos como en términos de tiempo.

Las condiciones son autoimpuestas, aunque el objetivo general tenga que tener sentido no responde a unos requerimientos del mercado o a un cliente, tanto las características concretas como el alcance y profundidad del proyecto se definen por su autor.

Aunque el objetivo este marcado el camino para llegar a él no tiene por qué ser el más corto o el más fácil. Pues el objetivo último de este proyecto, así entiendo yo, es poner en práctica los conocimientos adquiridos así como adquirir nuevos durante su realización.

Con estas premisas queda claro que no se pretende ejecutar un proyecto competitivo pero si acercarse cuanto se pueda a la mayor funcionalidad posible. Vamos pues ahora a analizar proyectos y casos más cercanos al nuestro.

Si tomamos como referencia los proyectos que se pueden encontrar en internet referentes al nuestro veremos que son todos bastante similares. (Bronchal Paricio, 2017; Redondo Sanz, 2020; Sánchez-Agustino, 2016)

En todos se usa un microcontrolador integrado en una placa de desarrollo, en algunos con la ayuda de un PC. En la mayoría se usan sensores ópticos económicos o incluidos en las placas de desarrollo de los microcontroladores.

En cuanto a los elementos mecánicos como motores o barreras no hay referencias claras en este tipo de documentos por lo que tendremos que recabar información en productos comerciales.

3.3. REQUISITOS DE DISEÑO

Con el fin de acotar el proyecto se establecerán unos requisitos de diseño que se intentarán satisfacer en todo lo posible en el diseño que se pretende realizar.

Con el fin de enfocar el diseño se tomara como referencia sistemas reales. En estos sistemas vemos que es importante tener en cuenta los ángulos de visualización que las cámaras tienen respecto a la posición ortogonal de las placas de matrícula.

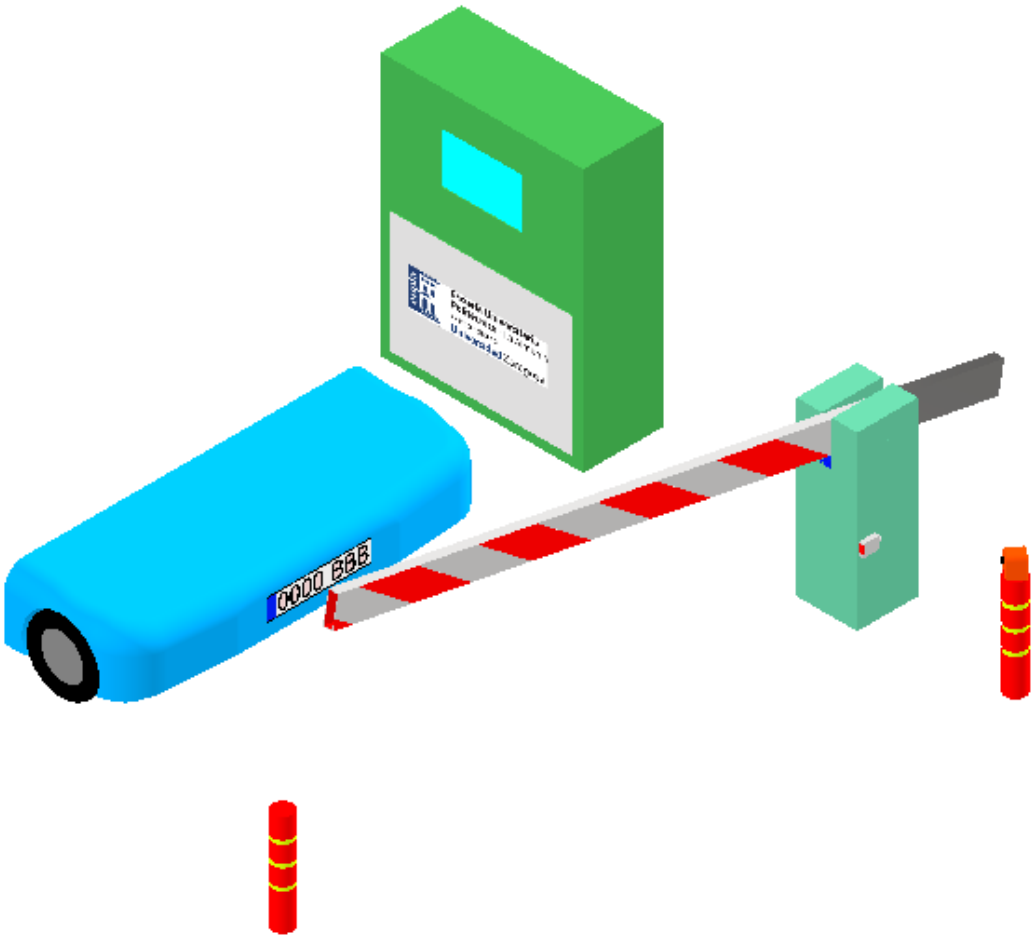
Además vemos que la cámara se encuentra relativamente alejada del vehículo, esto es así para conseguir esos ángulos y que la cámara se encuentre fuera de la trayectoria del vehículo.

Estos ángulos son necesarios respetarlos debido a que para una correcta identificación es preferible un ángulo menor.

Viendo esto se podría decir que si es posible se pusiera la cámara lo más alejada posible, sin embargo esto conlleva otros problemas. El principal que se requeriría una cámara con mayor resolución y esto a su vez conlleva que o bien el tiempo de computo se incremente o de utilizar un hardware más potente debido a que la cantidad de datos a analizar sería mayor.

También hay que tener en cuenta el ángulo de visión que tiene la cámara, o más bien la lente del sensor óptico. Si tuviéramos una situación en la que la cámara estuviera muy cerca del vehículo sería apropiada una lente de mayor ángulo sin embargo para la situación que planteamos de una cámara relativamente lejana sería más conveniente una lente de ángulo más cerrado. Esto es así también porque en el momento de la identificación el vehículo siempre se va a encontrar en un determinado margen espacial.

Por estas razones hay que llegar a un compromiso en las medidas de nuestro sistema si queremos que este sea eficaz.



En cuanto a las características de funcionamiento vamos a definir las funciones que debe realizar el sistema.

Ser capaz de identificar los vehículos según su placa de matrícula de forma eficaz, aquí podríamos añadir una condición adicional la cual sería que el sistema sea capaz de saber cuándo hay un vehículo esperando a ser identificado y posteriormente cuando el vehículo rebasa la barrera.

Disponer de un sistema RTC que permita al sistema almacenar y relacionar datos de los vehículos y el tiempo que se encuentran dentro del parking.

Además de estos datos también será capaz de saber cuántos vehículos hay dentro y cuantas plazas hay disponibles.

El sistema también tendrá que comunicarse con el usuario de forma que le muestre los datos anteriormente mencionados.

4. DESARROLLO

Dividido en epígrafes. En este apartado, el más amplio, se expondrá el cuerpo fundamental del trabajo y se argumentarán las ideas principales y secundarias del mismo.

Detalla el proceso que se ha llevado a cabo para la elaboración del TFG y la metodología adoptada dependiendo de la naturaleza del TFG escogido (contenido teórico, caso práctico, proyecto, estudio técnico, revisión bibliográfica y estado de la cuestión, justificación, resultados...), pero en todo caso debe estar dividido en capítulos homogéneos y numerados.

4.1. MARCO TEÓRICO

En este apartado se analizarán todos los elementos que componen el sistema, una descripción de su funcionamiento aportando características y justificando su uso así como interrelaciones entre ellos.

4.1.1. Elementos de hardware

Aquí analizaremos los elementos que componen la parte hardware del sistema, es decir los elementos físicos.

4.1.1.1. Sensor óptico

Para valorar los sensores ópticos primero tenemos que saber cómo funcionan. Los módulos que podemos encontrar en el mercado constan de 3 partes diferenciadas, una lente, el sensor y un procesador de imágenes.

4.1.1.1.1. Lente

La lente de la cámara se encarga de dirigir los rayos de luz hacia el sensor. Además según sea esta lente tendremos un ángulo de visión diferente. También podemos encontrar esta información como distancia

focal aunque es más común en otros tipos de usos, una distancia focal menor implica un ángulo mayor y viceversa.

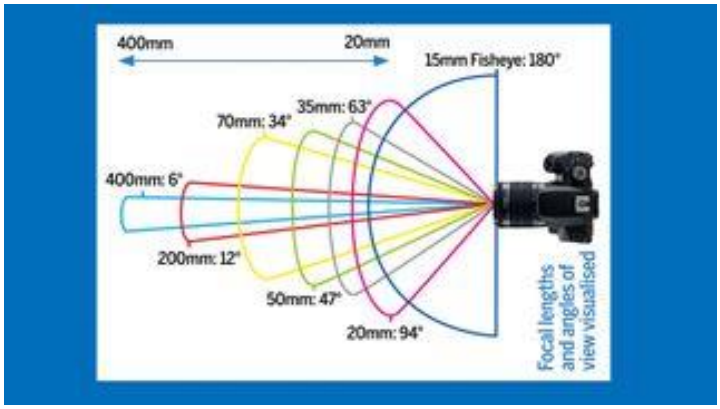


Ilustración 1: Campo de visión,(published, 2022)

4.1.1.1.2. Sensor

El sensor convierte la radiación electromagnética en corriente eléctrica. De esto se encarga un elemento semiconductor.

4.1.1.1.3. Procesador

La función del procesador es recibir los datos de cada pixel del sensor y montar la imagen para ello realiza ajustes automáticamente.

Esto tiene que ver con el filtro Bayer y las interpolaciones que se realizan para pasar a un formato RAW o a otro formato con compresiones de imagen.

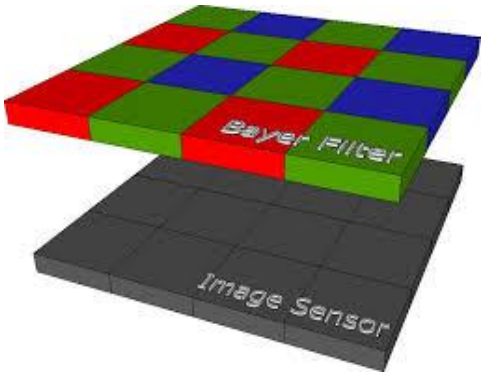


Ilustración 2: Filtro Bayer, (historyphotography, 2014)

4.1.1.2. Microcontrolador

Un microcontrolador es un dispositivo electrónico integrado que se utiliza para controlar y gestionar diferentes funciones en sistemas electrónicos. Está diseñado para ejecutar tareas específicas de procesamiento y control en tiempo real. Los microcontroladores constan de una unidad central de procesamiento (CPU), memoria, periféricos de entrada/salida y, en algunos casos, interfaces de comunicación.

COMPARATIVA

Para un proyecto como este hay varias opciones en el mercado, a continuación se mencionan algunas, Arduino, ESP32, Raspberry Pi, STM32.

Tabla 1: Comparativa Microcontroladores

Característica	ESP32	ATMEGA 328P	Raspberry Pi3	STM32
Arquitectura	Xtensa LX6 (32-bit)	AVR (8-bit)	ARM Cortex (64-bit)	ARM Cortex (32-bit)
WiFi	Sí	Modulo adiccional	Sí	Modulo adiccional
Bluetooth	Sí, Bluetooth Low Energy	Modulo adiccional	Si	Modulo adiccional
Número de Núcleos	Doble núcleo	Un solo núcleo	Cuatro núcleos	Doble núcleo
Velocidad de Reloj	240 MHz	20 MHz	1.2GHz	72MHz
Memoria RAM	520 KB	2 KB	1 GB	20 KB
Memoria Flash	16 MB	32 KB	Según almacenamiento, puede llegar a 1 TB	64 KB
GPIO	34	23	40	16
Sistema Operativo Soportado	FreeRTOS	No (programación en bare-metal)	Linux (Raspbian, otros)	Depende del modelo y configuración

Ademas de esto hay que tener en cuenta que tanto ESP32 como Raspberry Pi poseen placas de desarrollo a las que es muy fácil conectar una cámara. Tanto Arduino como STM32 tienen posibilidad de conectar una cámara pero no usan un conector estándar.

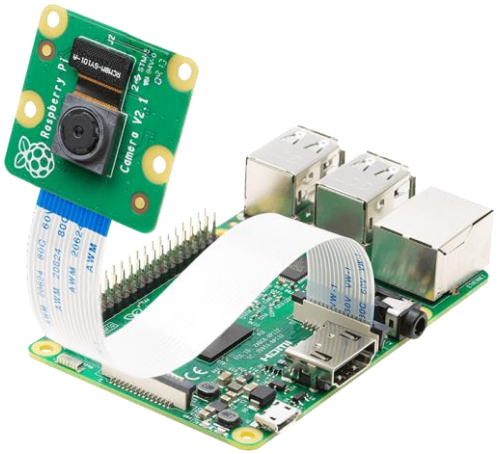


Ilustración 3: Cámara Raspberry Pi, (Arducam, 2020)

A grandes rasgos podemos diferenciar a Rspberry Pi del resto puesto que cuenta con mucha más potencia y es capaz de correr sistemas operativos complejos, se podría decir que es más parecido a un ordenador, los demás son microcontroladores más modestos.

Sin embargo hay otros aspectos a evaluar, el consumo de energía de una Raspberry Pi también es muy superior al de las demás opciones.

Otro factor muy importante es el precio, en estos últimos años y sobre todo a partir de 2020 la industria de los semiconductores a atravesado momentos convulsos que en algunos casos han aumentado los precios significativamente, este es el caso de la Raspberry Pi llegando incluso al punto de que haya un stock limitado de ete producto.

En cuanto a Arduino, ESP32 y STM32 el microcontrolador desarrollado por Espressif es claramente superior a sus competidores, tiene una potencia superior, mejor conectividad y además es más económico que Arduino.

Aunque el STM32 tenga un menor precio respecto al ESP32, este último compensa la diferencia de precio.

4.1.1.3. Pantalla

Podríamos definir como principales requisitos para la pantalla y para el tipo de nuestra aplicación que en ella se puedan mostrar todos los datos que queremos, que el tamaño sea el adecuado teniendo en cuenta las distancias de visualización y que la resolución sea la adecuada teniendo en cuenta distancia y tamaño.

PROTOCOLOS DE COMUNICACIÓN

Para la conexión de una pantalla se suele utilizar un protocolo de comunicación así que tenemos que tener en cuenta esto si es nuestro caso para comprobar la compatibilidad de estos protocolos con nuestro microcontrolador.

TIPO DE PANEL

LED

Utiliza diodos emisores de luz para iluminar los píxeles. Cada píxel es una fuente de luz individual.

Ofrece buena calidad de imagen, buenos colores y alto brillo. Adecuado para aplicaciones como televisores y pantallas grandes.

Consumo moderado de energía.

OLED

Cada píxel es una fuente de luz orgánica individual. No requiere retroiluminación separada.

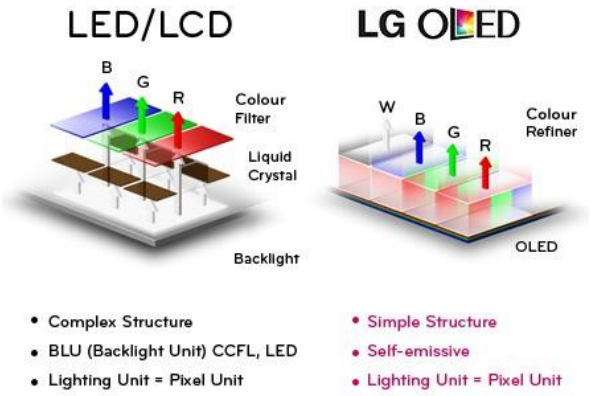


Ilustración 4: Pantallas OLED, (Rosalie, 2016)

Ofrece una excelente calidad de imagen con negros profundos, alto contraste y reproducción precisa de colores. Se utiliza en dispositivos móviles, televisores de alta gama y monitores de ordenador de alta gama.

Consumo bajo de energía, especialmente al mostrar contenido oscuro debido a la falta de retroiluminación.

LCD

Dependiente de una fuente de luz de fondo.

Ofrece una calidad de imagen decente, pero puede sufrir problemas como fugas de luz y ángulos de visión limitados. Común en monitores de ordenador y televisores.

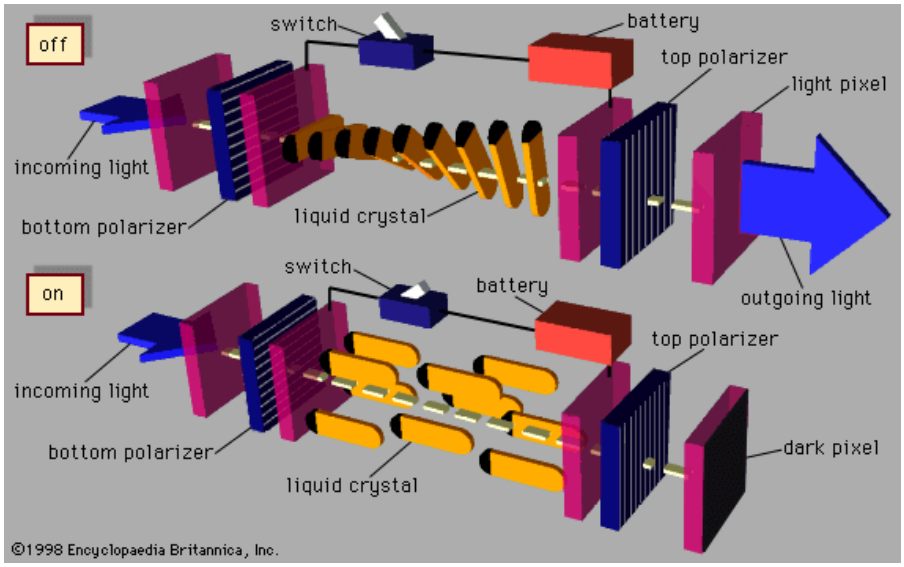


Ilustración 5: Pantalla LCD, (Walton, 2020)

Pantallas de Tinta Electrónica

No emiten luz propia. Utilizan partículas microscópicas que se reorganizan para mostrar contenido.

Ofrecen una calidad de imagen similar a la del papel impreso, con bajo consumo de energía. Ideales para lectores de libros electrónicos y aplicaciones de lectura.

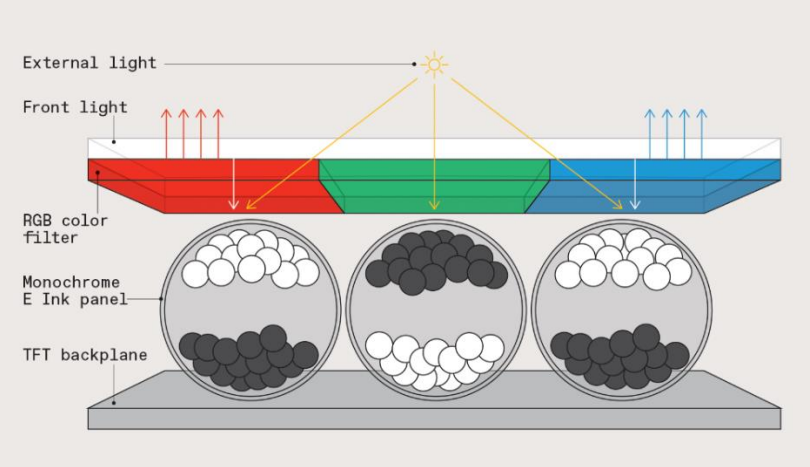


Ilustración 6: Pantalla de tinta electrónica, (Huitema & French, 2022)

Altamente eficientes en términos de consumo de energía, ya que solo requieren energía para cambiar el contenido de la pantalla.

ELECCIÓN

Se ha escogido una pantalla OLED ya que los ángulos de visión pueden ser determinantes en una aplicación.

Los vehículos no siempre se van a posicionar en el mismo punto, y el ángulo desde el que se observe la pantalla cambiara considerablemente.

4.1.1.4. Barrera

FUNCION

En nuestro sistema, para su correcto funcionamiento, es necesaria la presencia de una barrera u obstáculo que impida el paso al vehículo hasta que este sea identificado. Una vez el proceso de identificación y otras funciones han sido completadas el obstáculo se retirara para que el vehículo continúe y cuando este haya superado el mismo el obstáculo recuperara la posición para repetir el ciclo con el siguiente vehículo.

CARACTERISTICAS FÍSICAS

Según se ha podido establecer en la normativa si la barrera no supera los 2.5m de longitud no se le aplica la normativa CE así que en ese caso se han buscado datos de productos comerciales similares a lo que se necesita para nuestra aplicación viendo que medidas son las más habituales así como materiales y tipos de motores empleados.(RFC Puertas Automaticas, 2013)

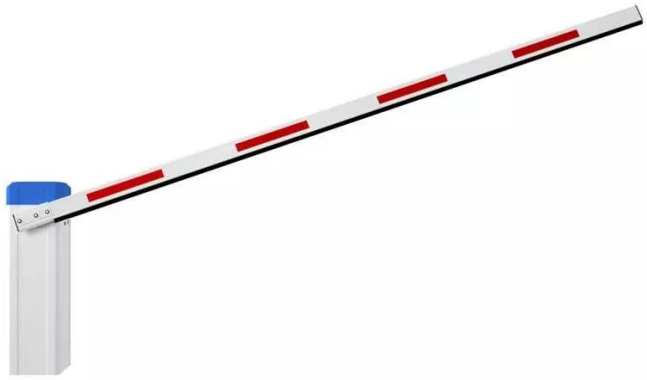


Ilustración 7: Barrera de Parking, (Segurdoma ACS®, 2016)

En cuanto a los materiales para el cuerpo fijo parece que la mejor opción es una cubierta de acero inoxidable o en su defecto con un recubrimiento anticorrosivo que lo proteja.

Para el mástil tenemos más opciones, siendo las principales las aleaciones de aluminio, el acero y la fibra de carbono.

OPCIÓN COMERCIAL

Se ha decidido optar por una solución comercial que incorpora gran parte de los sistemas. Esta es de la empresa Aprimatic.

Ilustración 8: Barrera Aprimatic, (Aprimatic, 2019)

Esta nos viene con un módulo de control que incorpora cierta lógica de la barrera.

El modulo es de alimentación de 220VAC.

4.1.1.5. Sensores

Para nuestra aplicación necesitamos sensores que nos indiquen la presencia o ausencia de vehículo en una determinada posición. Incluso se podría plantear la ausencia de estos sensores si se incorporara una función mediante software aprovechando las imágenes capturadas para averiguar si hay o no vehículo y su posición pero esta es una tarea bastante compleja además de que una de los objetivos que perseguimos con la incorporación de sensores es precisamente que el sistema de reconocimiento este activo solo cuando haya vehículo.

Dicho esto hay varios tipos de sensores que satisfacen nuestros requerimientos, estos son los siguientes, sensor capacitivo, sensor de distancia por ultrasonidos, sensor de distancia por infrarrojos, células/barreras fotoeléctricas.

4.1.1.5.1. Sensor de distancia ultrasonidos

Los sensores de distancia por ultrasonidos son dispositivos electrónicos que utilizan ondas sonoras ultrasónicas para medir la distancia entre el sensor y un objeto.

Principio de Funcionamiento

El funcionamiento de un sensor de distancia por ultrasonidos se basa en la emisión de pulsos de sonido ultrasónico y la medición del tiempo que tarda el eco en regresar al sensor. El sensor emite un pulso de sonido de alta frecuencia y luego mide el tiempo que tarda en recibir el eco del pulso cuando rebota en un objeto. Utilizando el tiempo de vuelo del sonido y la velocidad del sonido en el aire, el sensor calcula la distancia entre el sensor y el objeto.

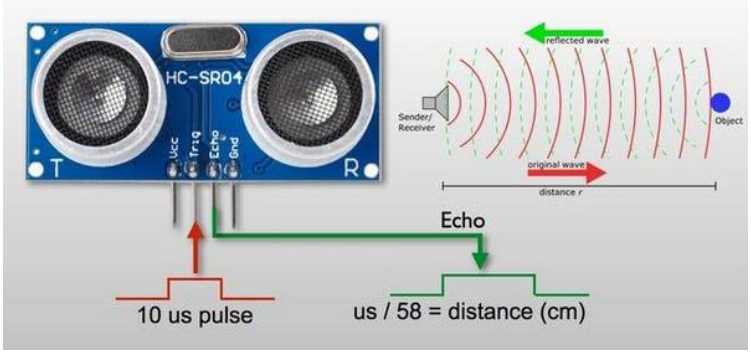


Ilustración 9: Sensor de Ultrasonidos, (Techmake, 2020)

Detección de Distancia Precisa

Los sensores de ultrasonidos son conocidos por su capacidad para medir distancias con alta precisión en una variedad de condiciones ambientales.

Aplicaciones Comunes

Los sensores de distancia por ultrasonidos se utilizan en una variedad de aplicaciones, que incluyen:

Sistemas de estacionamiento de automóviles: Para medir la distancia entre el automóvil y obstáculos al estacionar.

Robots móviles: Para evitar colisiones y navegar en entornos desconocidos.

Control de nivel en tanques: Para medir el nivel de líquidos en tanques y contenedores.

Detección de objetos: En líneas de ensamblaje para asegurar la presencia o ausencia de objetos.

Medición de distancias en aplicaciones industriales: En maquinaria y equipos de producción para control de procesos.

4.1.1.5.2. Barrera fotoeléctrica

Las barreras fotoeléctricas son dispositivos que utilizan la interrupción de un haz de luz para detectar la presencia o ausencia de un objeto en un área específica.

Principio de Funcionamiento

El principio fundamental detrás de las barreras fotoeléctricas se basa en la interrupción del haz de luz emitido por una fuente de luz y su detección por un receptor fotosensible. Estas barreras consisten en una unidad emisora de luz y una unidad receptora. Cuando un objeto atraviesa el haz de luz y bloquea la luz que llega al receptor, se detecta la presencia del objeto. La interrupción del haz de luz genera una señal eléctrica que indica la detección.

Detección de Objeto y Seguridad

Las barreras fotoeléctricas se utilizan para detectar objetos en movimiento o para monitorear el acceso a áreas sensibles. Su capacidad para detectar cambios en el haz de luz las hace valiosas en aplicaciones de seguridad, control de procesos industriales y automatización.

Aplicaciones Comunes

Las barreras fotoeléctricas se utilizan en una variedad de aplicaciones, que incluyen:

Sistemas de seguridad: En puertas automáticas, sistemas de alarma y control de acceso.

Detección de objetos en movimiento: En líneas de ensamblaje, cintas transportadoras y sistemas de seguridad en maquinaria industrial.

Control de procesos industriales: Para monitorear la presencia y movimiento de objetos en procesos de fabricación.

Control de acceso: Para controlar el acceso a áreas restringidas o parques de estacionamiento.

4.1.2. Elementos de software

En este apartado se pretende analizar las opciones de software que pueden ser utilizadas para controlar adecuadamente el sistema.

4.1.2.1. OCR

El Reconocimiento Óptico de Caracteres (OCR) es una tecnología que revoluciona la forma en que interactuamos con el texto impreso o manuscrito. Se trata de un proceso que permite convertir imágenes de texto en texto digital editable, lo que facilita la búsqueda, el almacenamiento y la manipulación de información. El OCR tiene aplicaciones en diversas áreas, desde la digitalización de documentos hasta la mejora de la accesibilidad y la automatización de procesos empresariales.

Principio de Funcionamiento

El OCR se basa en un proceso complejo que implica varias etapas:

Adquisición de imágenes: En esta fase, se captura una imagen de un documento físico o una superficie que contiene texto, ya sea mediante un escáner, una cámara o cualquier dispositivo de captura de imágenes.

Preprocesamiento de la imagen: Las imágenes capturadas pueden ser imperfectas debido a distorsiones, ruido o problemas de iluminación. El preprocesamiento se encarga de corregir estas imperfecciones para mejorar la calidad de la imagen.

Segmentación de caracteres: Una vez que la imagen ha sido preprocesada, se realiza la segmentación, que consiste en identificar la ubicación de cada carácter individual en la imagen.

Reconocimiento de caracteres: En esta fase, se aplica un algoritmo de reconocimiento para identificar cada carácter en las regiones segmentadas de la imagen. Este proceso se basa en comparar las características de la imagen con un conjunto de datos de formas de caracteres conocidas. Los algoritmos pueden variar desde métodos tradicionales hasta técnicas más avanzadas basadas en inteligencia artificial y aprendizaje profundo.

Postprocesamiento: Después de la etapa de reconocimiento, se realiza un postprocesamiento para mejorar la precisión. Esto puede incluir la corrección de errores y el análisis del contexto.

Generación de texto digital: Finalmente, los caracteres reconocidos se combinan para formar texto digital que se puede editar, buscar y almacenar.

Precisión y Entrenamiento

La precisión del OCR puede variar dependiendo de varios factores, como la calidad de la imagen, la legibilidad del texto y la tecnología utilizada. Para lograr una alta precisión, es común que los sistemas OCR se entrenen con grandes conjuntos de datos de texto y se ajusten para reconocer fuentes y estilos específicos. Además, algunos OCR permiten la adaptación y el entrenamiento personalizado para mejorar el rendimiento en aplicaciones específicas.

4.1.2.2. C/C++

El lenguaje C es un lenguaje muy utilizado en los microcontroladores, se trata de un lenguaje compilado y calificado como de tipado fuerte.

Este lenguaje es muy usado en microcontroladores como Arduino o ESP32, en el caso de Arduino es el único lenguaje en el que se puede programar, para ESP32 tenemos otras opciones como Micropython o incluso JavaScript.

En el caso de ESP32, si queremos utilizar C podemos usar la capa de Arduino o la de Espressif. A favor de usar la capa de Arduino hay que decir que existe más documentación sobre ella.

AL ser un lenguaje compilado también es necesario un compilador, para ello podemos usar el IDE de Arduino o usar otro IDE y agregarle un plugin como PlatformIO. Esta última opción tiene las ventajas de que podremos aprovechar facilidades y herramientas de entornos de desarrollo como Visual Sutudio Code

Ilustración 10: Logo PlatformIO, (Wikipedia, 2023)

4.1.2.3. Python

Python es un lenguaje de programación versátil, de alto nivel e interpretado. Su diseño se centra en la legibilidad del código y la facilidad de uso, lo que lo convierte en una excelente opción.

Características de Python

Sintaxis Clara y Legible: Python se destaca por su sintaxis simple y fácil de leer. Utiliza sangrías en lugar de llaves y paréntesis para estructurar el código.

Gran Comunidad y Ecosistema: Python tiene una comunidad de desarrolladores activa y una amplia base de usuarios. Existe una abundancia de librerías de código abierto que extienden su funcionalidad y permiten el desarrollo rápido de aplicaciones.

Multiplataforma: Python es compatible con una amplia gama de sistemas operativos.

Interpretado: Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea por un intérprete en lugar de compilarlo previamente.

OpenCV

OpenCV es una biblioteca de visión por computadora muy popular y ampliamente utilizada que proporciona una variedad de herramientas y algoritmos para procesar y analizar imágenes y videos.

Ilustración 11: OpenCV, (Rouizi, 2022)

4.1.2.4. Micropython

MicroPython es una implementación de Python diseñada específicamente para sistemas embebidos y microcontroladores. Esta especialmente pensada para el campo de la Internet de las Cosas (IoT).

Características de MicroPython

Sintaxis de Python: MicroPython utiliza la misma sintaxis que Python, lo que facilita a los programadores familiarizados con Python empezar a trabajar en proyectos de hardware sin tener que aprender un nuevo lenguaje.

Eficiencia: A pesar de ser una implementación de Python, MicroPython es lo suficientemente eficiente para ejecutarse en microcontroladores y sistemas con recursos limitados, como los utilizados en dispositivos IoT.

Soporte para hardware: MicroPython ofrece una amplia gama de librerías que facilitan la interacción con hardware, incluyendo sensores, actuadores y comunicación con periféricos.

Portabilidad: MicroPython es altamente portable y se puede ejecutar en una variedad de microcontroladores populares, como los de la familia ESP8266, ESP32 y STM32.

OpenCV

Cabe destacar que openCV no cuenta con soporte para micropython. Este hecho es relevante ya que openCV es una de las librerías más usadas para el procesamiento y reconocimiento de imágenes en microcontroladores.

4.1.2.5. Pytorch

PyTorch es una biblioteca de aprendizaje profundo de código abierto desarrollada por Facebook's AI Research lab (FAIR). Es una de las bibliotecas más populares para el desarrollo de aplicaciones de Deep Learning y redes neuronales.

Aplicaciones de PyTorch

PyTorch se utiliza en una amplia variedad de aplicaciones de Deep Learning, que incluyen:

Visión artificial: Para tareas como clasificación de imágenes, detección de objetos, segmentación semántica y generación de imágenes.

Procesamiento de lenguaje natural (NLP): Para la construcción de modelos de procesamiento de texto, traducción automática y generación de texto.

Reconocimiento de Voz: En aplicaciones de reconocimiento de voz y procesamiento de audio.

Robótica: En robótica para tareas de percepción, planificación y control.

Investigación en IA: PyTorch es ampliamente utilizado en la investigación académica y la experimentación en inteligencia artificial.

4.1.2.6. TensorFlow

TensorFlow es una librería de código abierto desarrollada por Google.

Características de TensorFlow

Flexibilidad de Modelado: TensorFlow ofrece una amplia flexibilidad para la creación y personalización de modelos Deep Learning. Podemos definir y entrenar modelos desde cero o utilizar modelos pre entrenados.

Modelos pre Entrenados: La librería ofrece una amplia selección de modelos pre entrenados en una variedad de dominios, como visión por computadora y procesamiento de lenguaje natural.

Keras como Interfaz de alto nivel: TensorFlow proporciona Keras, una API de alto nivel que simplifica la construcción y el entrenamiento de modelos de aprendizaje profundo.

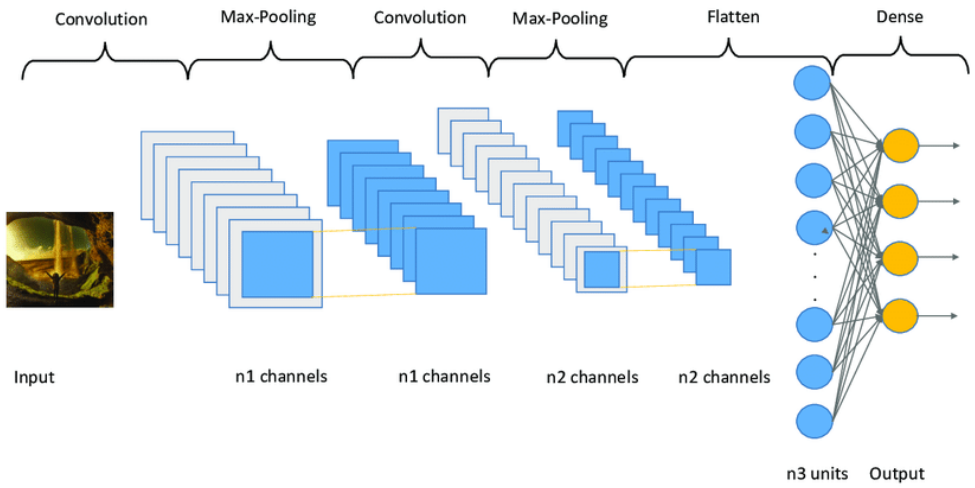


Ilustración 12: Capas de Keras, (jaintarum, 2022)

Soporte de GPU: TensorFlow proporciona una integración eficiente con GPU, lo que permite una aceleración significativa en el entrenamiento de modelos.

Aplicaciones de TensorFlow

TensorFlow se utiliza en una amplia variedad de aplicaciones de inteligencia artificial y aprendizaje profundo, que incluyen:

Visión por Computadora: Para tareas como clasificación de imágenes, detección de objetos, segmentación semántica y generación de imágenes.

Procesamiento de Lenguaje Natural (NLP): En aplicaciones de procesamiento de texto, traducción automática, generación de texto y análisis de sentimientos.

Reconocimiento de Voz: En sistemas de reconocimiento de voz y procesamiento de audio.

Robótica: En robótica para tareas de percepción, planificación y control.

Investigación en IA: TensorFlow se utiliza en investigaciones académicas y proyectos de vanguardia en inteligencia artificial.

4.1.3. Procesamiento de imágenes

El procesamiento de imágenes es un proceso por el cual se transforma una imagen en otras con el objetivo de extraer información de ellas, esto se logra mediante la aplicación de filtros.

Funcionamiento del proceso

La imagen que se recibe del sensor óptico es una matriz de datos que tiene información de la imagen con color. Esto quiere decir que cada pixel de la imagen tiene tres valores que representan la intensidad de rojo, verde y azul.

Para hacer los datos más manejables se convierte la imagen a una escala de grises de forma que así cada pixel tiene un único valor.

Ahora podemos aplicar transformaciones sobre la imagen, esto se refiere a aplicar operaciones matemáticas sobre la matriz. Esto se denomina aplicar un filtro.

Unas transformaciones comunes consisten en sobre cada pixel aplicar una matriz de convolución, también llamado núcleo o kernel, cada uno con una función específica. Esta recoge información de los pixeles cercanos.

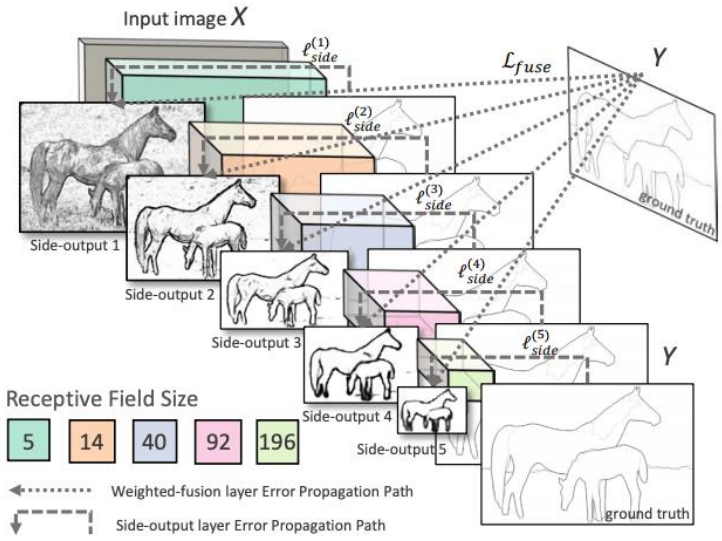


Ilustración 13: Procesamiento de imágenes, (Pardo Herrera, 2019)

Así conseguimos por ejemplo transformar la matriz para que únicamente los pixeles con cambios bruscos de tono tengan un valor

alto en esta nueva matriz, el resto de pixeles tendrán un valor de 0. Con esto podemos identificar los bordes de los objetos, ya que estos producen un cambio brusco.

Ahora mediante un algoritmo podemos identificar contornos cerrados y además buscar entre esos contornos los que cumplan con las características de los objetos que nos interesan.

Una vez tenemos identificada la zona que nos interesa la aislamos y aplicamos sobre ella un reconocimiento óptico de caracteres.

Filtros y matrices de convolución

Existen diferentes tipos de filtros, los denominados filtros de paso bajo cuya función es reducir el ruido y suavizar la imagen, filtros de paso alto que resaltan zonas, filtros direccionales y filtros de detección de bordes. (Gimenez-Palomares, Monsoriu, Alemany-Martinez, 2016)

4.1.4. Redes neuronales

Las redes neuronales son un concepto fundamental en el campo de la inteligencia artificial y el aprendizaje automático. Se inspiran en la estructura y el funcionamiento del cerebro humano y se utilizan para abordar una amplia gama de problemas, desde la visión por computadora hasta el procesamiento de lenguaje natural.

Estructura de una Red Neuronal

Una red neuronal está compuesta por capas de neuronas artificiales interconectadas. Estas capas se dividen en tres tipos principales:

Capa de Entrada: Esta capa recibe los datos de entrada, que pueden ser imágenes, texto, señales, etc. Cada neurona en esta capa representa una característica de entrada.

Capas Ocultas: Estas son capas intermedias que procesan la información. Cada neurona en estas capas realiza cálculos basados en las entradas y transmite los resultados a las siguientes capas.

Capa de Salida: Esta capa produce los resultados finales de la red neuronal, que pueden ser clasificaciones, valores numéricos, o cualquier otro tipo de salida deseada.

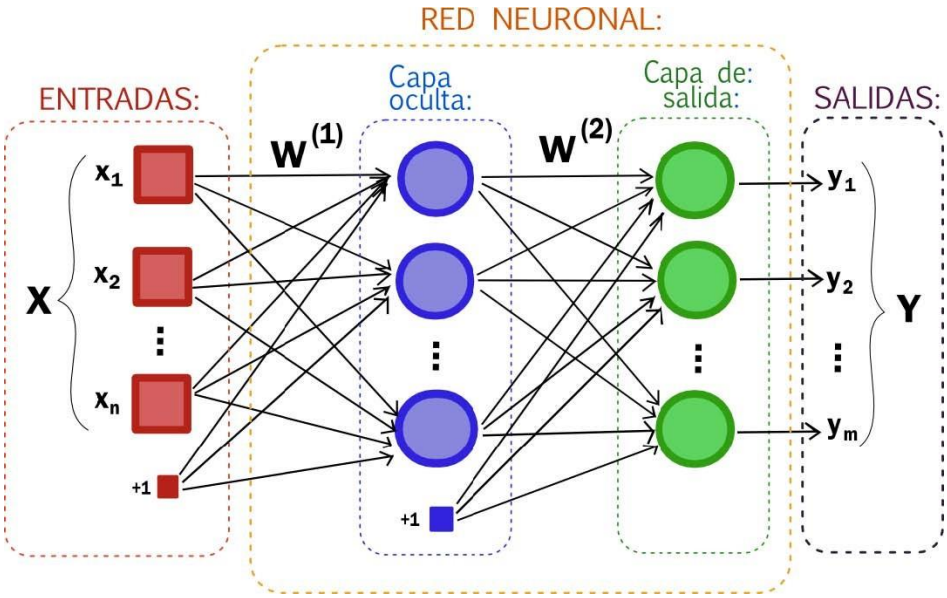


Ilustración 14: Red Neuronal, (Rodríguez, 2023)

Funcionamiento de una Neurona Artificial

Cada neurona artificial es como una pequeña unidad de procesamiento. Toma las entradas, las multiplica por pesos (números que ajustan la importancia de cada entrada), suma estos valores ponderados y luego aplica una función de activación. La función de activación determina si la neurona debe activarse y transmitir su salida a la siguiente capa.

Entrenamiento de una Red Neuronal

El proceso de entrenamiento es crucial en una red neuronal. Se basa en el principio de aprendizaje supervisado, donde la red se entrena con ejemplos etiquetados. Durante el entrenamiento, la red ajusta los pesos de sus conexiones para minimizar la diferencia entre sus predicciones y las etiquetas reales. Esto se logra mediante algoritmos de optimización.

4.1.5. DETECCIÓN DE CONTORNOS

La detección de contornos es un paso muy importante para el reconocimiento óptico de caracteres.

El objetivo es detectar don se produce un cambio brusco de color o de iluminación, es decir cuando el gradiente sea alto.

Una de las formas más habituales de hacerlo es mediante matrices de convolución. Estas matrices se aplican a caa pixel de la imagen dando como resultado otra imagen.

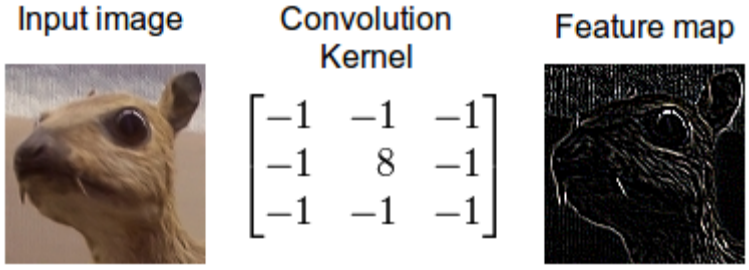


Ilustración 15: Matriz de convolución, (NVIDIA, 2021)

En este ejemplo podemos ver cómo funciona, tomamos como referencia un pixel cualquiera y sus adyacentes formando una matriz 3x3.

Tomamos el valor de los pixeles en escala de grises y multiplicamos cada pixel por su peso correspondiente en la matriz de convolución, después sumamos los resultados y ese será el nuevo valor del pixel de referencia en la nueva imagen.

Además, podemos incluir una función que haga que todos los valores por debajo de 0 sean iguales a 0, todos los valores por encima del máximo, por ejemplo 255, sean ese valor máximo y añadir un valor de threshold para que un pixel con un resultado superior a este valor umbral obtenga un valor máximo.

FILTRO SOBEL

Existen muchas formas de detectar estos gradientes, una de las más usadas es el filtro Sobel.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Ilustración 16: Filtro Sobel, (Fisher et al., 2003)

Este filtro además de poder detectar bordes también es capaz de identificar si el borde es vertical, horizontal o diagonal.

Esto se consigue modificando los pesos de la matriz de convolución, la de la izquierda detecta bordes verticales mientras que la de la derecha lo hace con los horizontales.

4.2. DISEÑO

Para comenzar el diseño partiremos de los requisitos del sistema y elaboramos un esquema con las principales funciones del sistema.

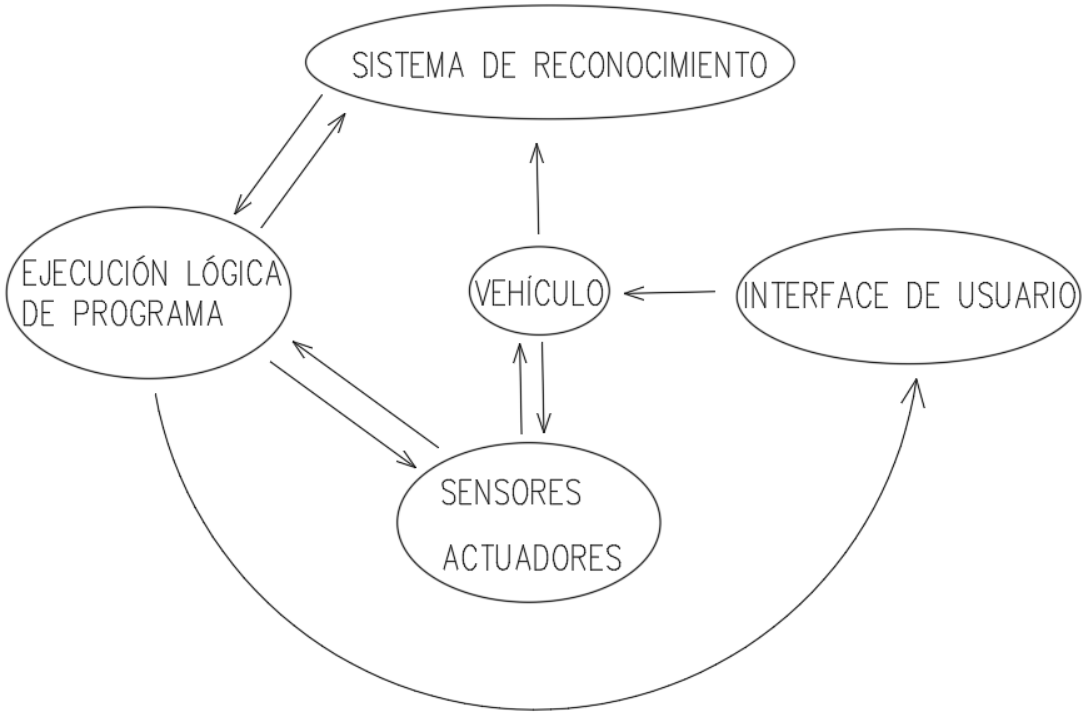


Ilustración 17: Diagrama de flujo

También tenemos que definir un lugar y unas condiciones para enmarcar el proyecto. Fijaremos estas según datos que se han recopilado de entornos reales.

De estos obtendremos las dimensiones de partes del sistema como la altura de la barrera, la distancia del sensor óptico a la matrícula o las dimensiones de los carriles.

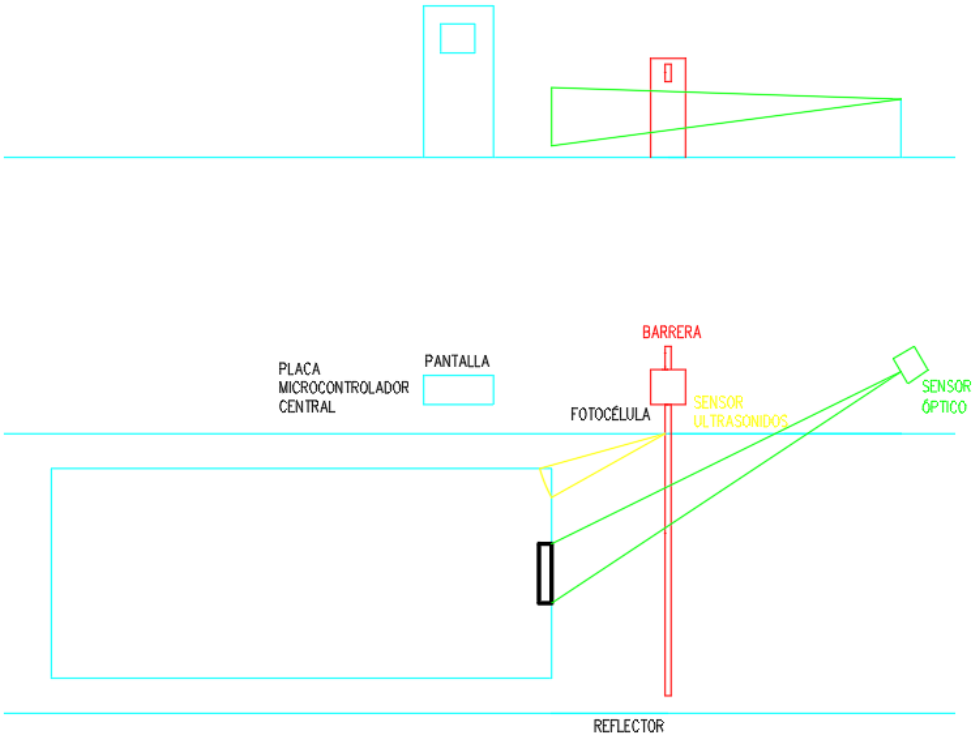


Ilustración 18: Boceto

Se ha fijado que la altura del sensor óptico sea de 0.5m.

También que se detecte cuando el vehículo se encuentre a menos de 1m de la barrera.

Por lo tanto para tener una visión adecuada desde la posición del sensor óptico este debe encontrarse a 2m de la barrera, esto nos deja una distancia a la línea ortogonal del vehículo de 3m. Si contamos con un ángulo de 30º nos deja una distancia entre la cámara y la matrícula de 3.5m.

Con este marco sobre el que trabajar colocaremos los elementos, los sensores para detectar la posición del vehículo, la pantalla dispuesta de tal forma que desde el vehículo se vea con claridad y el sensor óptico enfocando a la matricula con un ángulo adecuado.

Así definimos los elementos concretos que definirán nuestro sistema.

4.2.1. Elementos del sistema

Cabe mencionar, como se explicó anteriormente, que para escoger los elementos que conforman el sistema se ha tenido en cuenta con los medios de los que se disponía antes de comenzar la realización del proyecto.

4.2.1.1. Sensor óptico

Se ha optado por un sensor óptico incorporado en una placa de desarrollo de la plataforma ESP32, una ESP32-CAM.

El sensor en concreto es un OV2640.

Attribute	Values
Array Size	1600 x 1200 (UXGA)
Power Supply	Core: 1.3V DC ± 5% Analog: 2.5~3.0V DC I/O: 1.7V to 3.3V
Power Consumption	YUV mode full res & framerate: 125mW Compressed mode full Res & framerate: 140mW Standby:600μA
Image Sensor Format	Type 1/4"
Maximum Image Transfer Rate	1600×1200@15fps, SVGA@30fps, CIF@60fps
Sensitivity	0.6V/Lux-sec
S/N ratio	40dB
Dynamic Range	50dB
Pixel Size	2.2 x 2.2 μm

Ilustración 19: Sensor OV2640, (Arducam, 2017)

4.2.1.2. Microcontrolador

Para el microcontrolador encargado de comunicar todos los elementos y controlar la ejecución lógica del proceso se ha elegido un ESP32 de la empresa Espressif Systems.

Ilustración 20: ESP32 WROOM, (DigiKey, 2023)

Las razones han sido la experiencia con este microcontrolador y con los entornos de desarrollo asociados así como sus buenas prestaciones, en potencia de cómputo, número de entradas y salidas y comunicaciones disponibles.

Y como se expuso anteriormente la dificultad de encontrar en el mercado placas como la Raspberry Pi o Arduino R4.

4.2.1.3. Sistema de reconocimiento

El sistema de reconocimiento es el encargado de recibir las imágenes, procesarlas y dar un resultado de una sucesión de números y letras de forma que se adecue al formato de una matrícula.

En él está englobado tanto el tratamiento de las imágenes como el OCR.

Así pues en nuestro caso será un programa de Python que se ejecutará en un PC, este programa de Python constará de varias partes, una de las cuales es el empleo de un modelo de predicción basado en una red neuronal de TensorFlow al que previamente hemos entrenado.

Ilustración 21: Logos TensorFlow y Python, (Mora, 2020)

4.2.1.4. Pantalla

Según la información recopilada en el marco teórico analizando los diferentes tipos de pantallas lo más adecuado sería una pantalla LED debido a que posee buenos ángulos de visión y nos permite mostrar la información de una forma más visual.

Sin embargo tenemos también que encajar todos los elementos que conforman el sistema, así pues como se habló anteriormente el protocolo que use una pantalla es muy importante.

Las pantallas de panel LED por lo general usan interfaces de comunicación propias de PC o de sistemas con grandes capacidades de memoria y de cómputo como podría ser una placa RaspberryPI que podría encajar con nuestro proyecto. Estas formas de conexión son HDMI, DisplayPort o USB.

Por las razones anteriormente recomendadas se decide buscar una pantalla más acorde con el hardware usado eligiendo una pantalla LCD de 5 pulgadas con resolución 480x320

Model number	H28A-IC	H35B-IC	H43A-IC	H50A-IC
Resolution	240xRGBx320	480xRGBx320	480xRGBx272	480xRGBx272
LCD size	2.8 inch	3.5 inch	4.3 inch	5.0 inch
Outline	89x61mm	100x67mm	122x72mm	137x82mm
Active area	57x41mm	72x46mm	100x59mm	110x60mm
LCD mode	65K TFT	65K TFT	65K TFT	65K TFT
Viewing angle	12 O'clock	12 O'clock	12 O'clock	12 O'clock
Voltage	3.3~5.0V	3.3~5.0V	3.3~5.0V	3.3~5.0V
Backlight	White LED	White LED	White LED	White LED
Touch panel	Capacitive	Capacitive	Capacitive	Capacitive
Communicate port	I2C	I2C	I2C	I2C
Com voltage	3.3/5.0V TTL	3.3/5.0V TTL	3.3/5.0V TTL	3.3/5.0V TTL
Flash	8M	8M	16M	16M

Ilustración 22: Características Pantalla 5 Pulgadas, (Aliexpres, 2023)

4.2.1.5. Sensores

Para el sensor de seguridad de la barrera se ha elegido una célula fotoeléctrica.

Ilustración 23: Fotocélula, (Carlo Gavazzi, 2003)

Este elemento requiere de una alimentación de corriente alterna de 24/240VAC o bien corriente continua 12/240VCC.

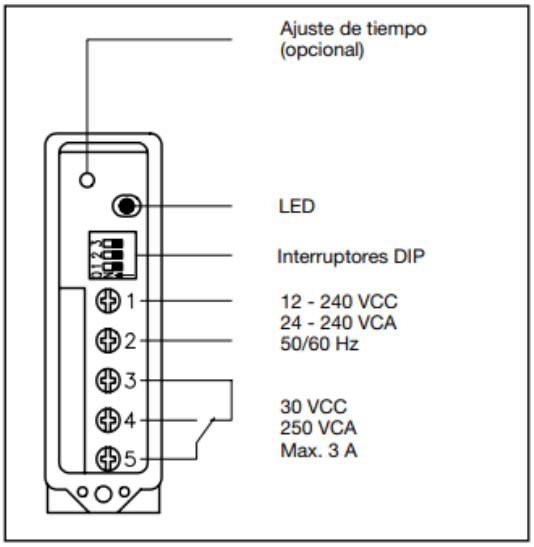


Ilustración 24: Conexiones Fotocélula, (Carlo Gavazzi, 2003)

La distancia máxima de alcance son de 10m, mas que suficiente para nuestra aplicación.

4.2.1.6. Barrera

El sistema que hemos escogido para la barrera es una central de control comercial junto con el mecanismo de accionamiento correspondiente.

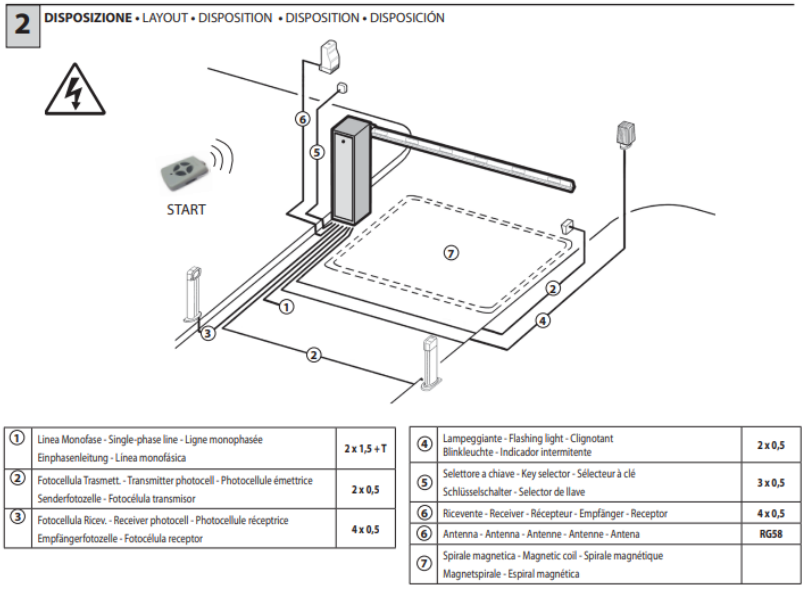


Ilustración 25: Conjunto Barrera, (Aprimatic, 2023a)

A continuación veremos cómo configurar esta central de control e interconectarla con nuestro sistema.

Podemos ver que el cuadro de control cuenta con una placa electrónica de donde parten todas las conexiones y donde tenemos elementos como potenciómetros o microrruptores con los que establecer la configuración del sistema.

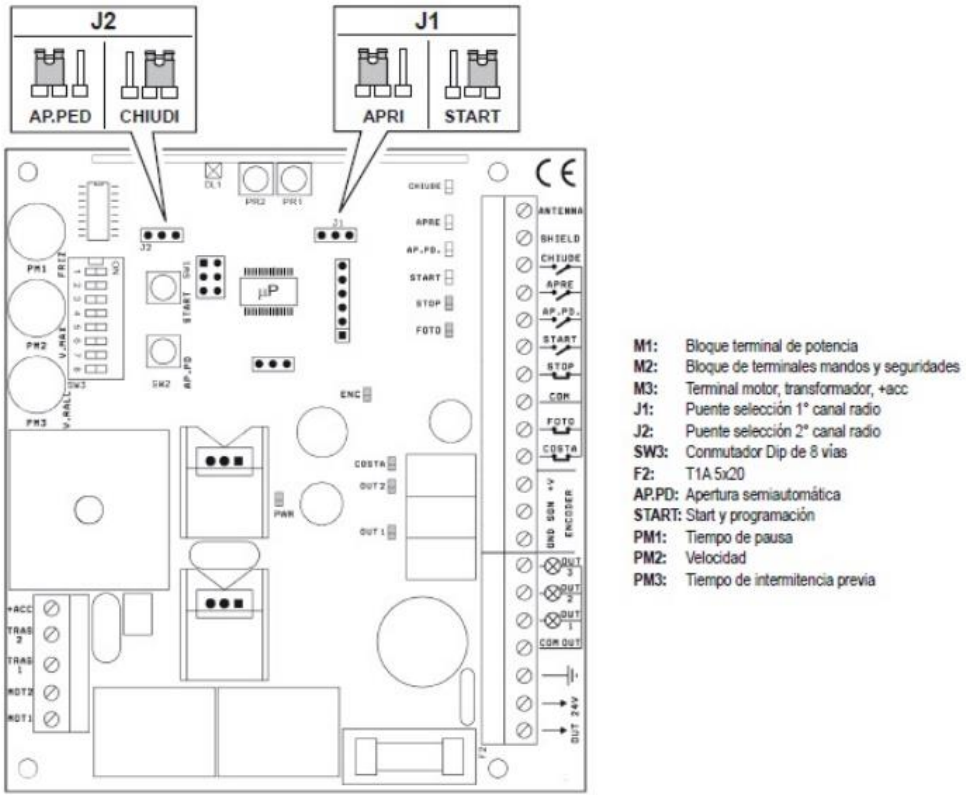


Ilustración 26: Placa de control Aprimatic, (Aprimatic, 2023c)

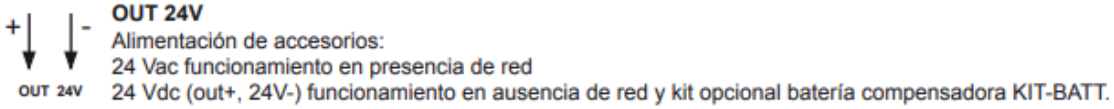


Ilustración 27: Traffic Park 24 1, (Aprimatic, 2023c)

Dispone de una conexión para los elementos externos de 24VAC.

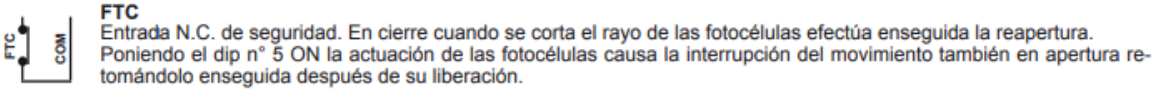


Ilustración 28: Traffic Park 24 2, (Aprimatic, 2023c)

Al contacto FTC conectaremos una fotocélula que será la encargada del sistema de seguridad para evitar que la barrera baje cuando haya un vehículo debajo de esta.

START
Entrada N.O. que permite controlar la automatización según la lógica programada mediante los dip 1& 2.

Ilustración 29: Traffic Park 24 3, (Aprimatic, 2023c)

El otro contacto al que tenemos que prestar atención es esta, es el de START, este lo tendrá que controlar el microcontrolador y su funcionamiento además dependerá de cómo configuremos esta central de control.

L configuración se realiza mediante un microswitch como el que podemos ver en la imagen.

Ilustración 30: Traffic Park 24 Microrruptor, (Aprimatic, 2023b)

Dip switch n° 1 y 2: Seleccionan la lógica de funcionamiento
Off-Off: Lógica de hombre presente.
El automatismo funciona para mandos mantenidos. El mando de start una vez abre y otra vez cierra.
La norma prohíbe comandos por vía radio en la lógica de hombre presente.
On-Off: Lógica semiautomática.
El automatismo funciona para mandos de impulsos sin el cierre automático.
On-On: Lógica automática.
El automatismo funciona para mandos de impulsos con cierre automático después del tiempo de pausa regulado en el trimmer PM1.

Ilustración 31: Traffic Park 24 4, (Aprimatic, 2023c)

En el switch número 1 y número 2 tenemos la opción de cómo queremos que funcione el proceso de abrir y cerrar, si queremos que se cierre al cerrar el contacto START o tras un tiempo.

Nosotros escogemos la opción de Lógica Automática, tras un impulso, la barrera abre, una vez arriba cuenta un tiempo y después cierra.

Para ello los switch 1 y 2 tienen que estar en ON.

Dip 3: Selecciona la lógica del mando de start.
Off: El mando de start repetido realiza la secuencia: abre-para-cierra-abre
On: El mando de start efectúa sólo la apertura; en pausa, cierra; en cierre, vuelve a abrir.

Ilustración 32: Traffic Park 24 5, (Aprimatic, 2023c)

En el switch 3 esccogemos la opción ON para que el contacto de START solo pueda abrir la barrera.

Dip 5: Selecciona la actuación de la fotocélula de seguridad.
Off: Fotocélulas activas sólo en cierre: en caso de taparlas, invierten la maniobra.
On: Fotocélulas activas tanto en apertura como en cierre. Cuando son tapadas en apertura interrumpen la maniobra para luego reanudarla tan pronto el rayo queda libre, mientras que en cierre invierten la maniobra sólo cuando está destapadas.

Ilustración 33: Traffic Park 24 6, (Aprimatic, 2023c)

En cuanto a la opción de la fotocélula solo queremos que se pare la barrera en el caso de que se encuentre bajando así que este switch se encontrará en OFF.

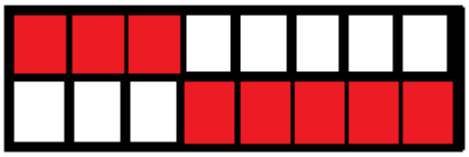
Dip 6: Selecciona la función de recierre enseguida después de la actuación de la fotocélula.
Off: Fotocélulas habilitadas como en el Dip 5.
On: Fotocélulas activadas sea como dispositivo de seguridad, según programación del Dip 4, que como comando de cierre. Su hay programado un tiempo de pausa éste es reducido a 3 s. cuando las fotocélulas son interrumpidas en apertura o en pausa.

Ilustración 34: Traffic Park 24 7, (Aprimatic, 2023c)

En el switch 6 seleccionamos la opción OFF porque queremos que actué el switch 5.

Los switches 4, 7 y 8 no afectan para nuestro funcionamiento así que los dejaremos en OFF.

Quedaría de la siguiente forma.



Como hemos configurado un cierre automático tras un tiempo, tenemos que configurar ese tiempo, esto lo hacemos mediante un potenciómetro situado al lado del microinterruptor.

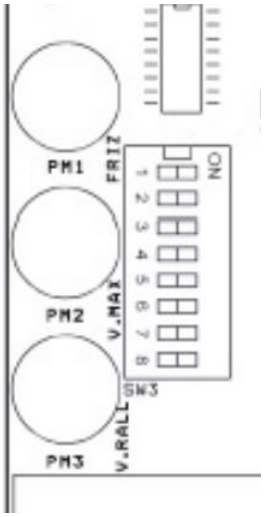


Ilustración 35: Traffic Park 24 Potenciometro,(Aprimatic, 2023c)

Hay tres potenciómetros, el que determina el tiempo es el PM1.

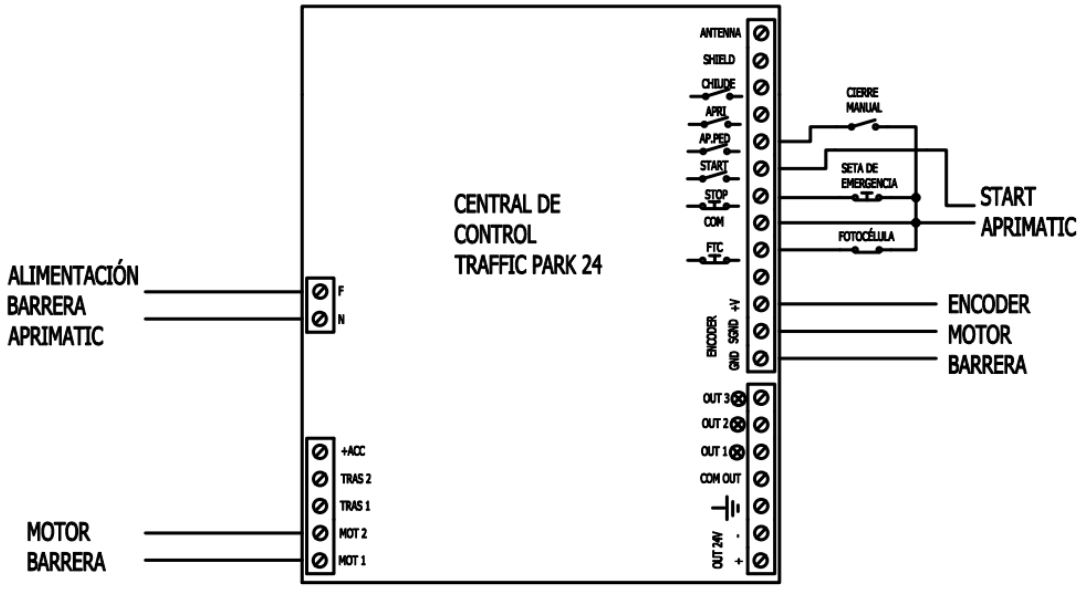


Ilustración 36: Central de control Traffic Park 24

Realizamos las conexiones de la fotocélula, una seta de emergencia que abre la barrera, un pulsador manual para abrir y cerra de forma manual y la conexión con la tarjeta electrónica del microcontrolador central.

Además conectamos el motor y su encoder como nos dice el fabricante.

4.2.2. Sistema de alimentación

Para la electrónica de control necesitamos una alimentación de 3.3V y de 5V para el puerto USB y para el sensor de distancia de ultrasonidos.

De lo que disponemos es de una toma de corriente de 230 VAC.

Así que lo más óptimo es una fuente de alimentación con doble salida, una de 5V y otra de 3.3V.

Ilustración 37: TMLM 04253, (Traco Power, 2023)

Una que cumple con nuestras necesidades es la siguiente, se trata de la fuente TMLM04253 de TRACO.

Input Specifications		
Input voltage	– Nominal	100 – 240 VAC
	– Range	90 – 264 VAC (universal input)
	– DC range	120 – 370 VDC

Ilustración 38: TMLM 04253 Entrada, (Traco Power, 2023)

Models

Order Code	Output Power max.	Output 1	Output 2	Efficiency
TMLM 04103	4.0 Watt	3.3 VDC / 1200 mA		67 %
TMLM 04105	4.0 Watt	5.0 VDC / 800 mA		69 %
TMLM 04109	4.0 Watt	9.0 VDC / 444 mA		72 %
TMLM 04112	4.0 Watt	12 VDC / 333 mA		70 %
TMLM 04115	4.0 Watt	15 VDC / 267 mA		74 %
TMLM 04124	4.0 Watt	24 VDC / 167 mA		73 %
TMLM 04253	3.5 Watt	+5.0 VDC / 600 mA	+3.3 VDC / 150 mA	69 %
TMLM 04225	3.6 Watt	+12 VDC / 250 mA	+5.0 VDC / 120 mA	69 %

Ilustración 39: TMLM 04253 Salida, (Traco Power, 2023)

La corriente de salida en ambos casos es suficiente, el micro ESP32 consume a pleno rendimiento unos 100mA.

Así pues agregamos a nuestra placa de control una conexión de 2 pines que corresponderán con la alimentación de red 230 VAC, conectada a la fuente y a la salida unos condensadores para estabilizar la tensión.

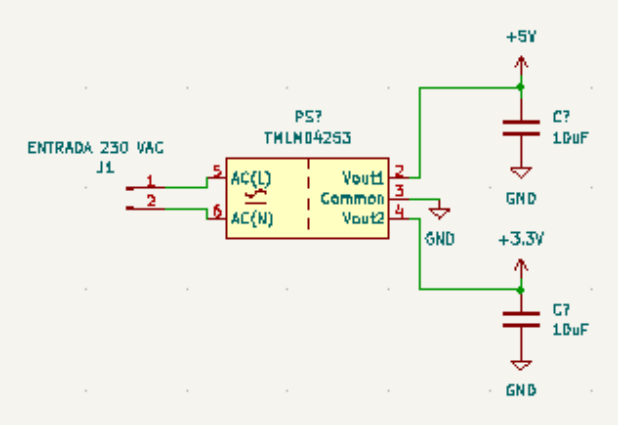


Ilustración 40: Fuente de alimentacion 5V y 3.3V

4.2.3. Interconexiones y comunicaciones

Una de las partes más importantes del sistema es la transmisión de datos entre el sensor óptico y donde se procesan las imágenes.

Ya que el sensor óptico se encuentra integrado dentro de una placa del ecosistema de ESP32 las opciones que tenemos para esta comunicación dependen de ese microcontrolador.

Así pues tenemos diferentes opciones para la comunicación, analizando el flujo de datos que va a tener esta comunicación vemos que tiene que tener una tasa de transferencia relativamente alto siendo las más acertadas las siguientes:

UART → 115200 Hasta 5Mbps

WiFi → Hasta 150 Mbps

Bluetooth → Hasta 24 Mbps

Pero además hemos de tener en cuenta otros factores, la comunicación no va a ser constante, solo se enviaran imágenes cuando haya un vehículo.

Cuando se envíe una imagen la comunicación no requerirá de más transferencias hasta que el procesamiento de la imagen termine, y si este es exitoso no se requerirá de más transmisiones.

Otro factor importante es el lugar donde se van a situar los elementos y la distancia entre ellos, el entorno en principio no es agresivo para las comunicaciones, un parking no representa un problema en este sentido en cuanto a ruido eléctrico o interferencias electromagnéticas, por otro lado si es subterráneo podría representar un problema para las comunicaciones inalámbricas. Sobre todo si los elementos se encuentran distanciados o en otras plantas.

En resumen ambas opciones son válidas en cuanto a tasa de transferencia de datos pero tienen diferencias según el lugar del sistema. Donde una comunicación inalámbrica puede representar una ventaja porque nos permite salvar obstáculos sin necesidad de realizar instalación también implica una posibilidad de que la comunicación se vea interrumpida.

Por esta razón esta comunicación se realizará mediante la interfaz UART de la placa ESP32-CAM.

4.2.4. Lógica de programa(Microcontrolador)

En el microcontrolador que llamaremos central se ejecutaran las funciones que afectan al sistema en su conjunto, esto quiere decir que se encargara de recibir información de los demás subsistemas y controlar a estos en consecuencia.

A su vez también se encargara de controlar procesos exclusivamente internos como tener un control sobre el tiempo absoluto.

Al tratarse de un ESP32 se programará mediante el entorno de desarrollo de Visual Studio Code con el plugin de PlatformIO y el framework de Arduino (también tenemos la opción en PlatformIO de usar el framework propio Espresiff).

Comenzamos nuestro código incluyendo las librerías que vamos a usar.

```
#include <Arduino.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <RTClib.h>
```

Es necesario incluir la librería de Arduino porque estamos fuera del entorno de desarrollo de la plataforma.

La librería Wire es para controlar buses I2C. Y las otras dos librerías corresponden a la pantalla y al RTC.

A continuación definimos los pines que usaremos, el número de filas y de columnas que tiene la pantalla y la velocidad del puerto serie.

```
// Pin para dar la señal al ESP-CAM y que capture una imagen
#define SENAL_SENSOR 25
// Pin para dar la señal al controlador de Aprimatic
#define START 35
// Pines I2C para pantalla y RTC
#define SDA 21
#define SCL 22
// Pines sensor ultrasonidos
#define TRIG 18
#define ECHO 19
#define COLUMNAS 20
#define FILAS 4
#define DIRECCION_LCD 0x3F
#define SERIAL_BAUDRATE 115200
```

Definimos las variables que vamos a usar y definimos los objetos de la pantalla y el RTC.

```
float distancia = 2000;
bool control = 0;
unsigned long pulse_time;

LiquidCrystal_I2C lcd(DIRECCION_LCD, COLUMNAS, FILAS, POSITIVE);

RTC_DS3231 rtc;
```

En el apartado de Setup inicializamos la pantalla, las GPIO diciendo si son entradas o salidas e inicializamos el puerto serie.

```
void setup() {

  lcd.begin(COLUMNAS, FILAS);

  pinMode(SENAL_SENSOR, OUTPUT);

  pinMode(START, OUTPUT);
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
  digitalWrite(TRIG,LOW);

  //Puerto serie
  Serial.begin(SERIAL_BAUDRATE);
  Serial.setTimeout(2000);
  Serial.println("ESP32-CAM inicializado.");
}
```

Ahora en el bucle infinito lo primero que hacemos es almacenar los datos del RTC en un objeto tipo DateTime y lo nombramos como “now”.

```
void loop() {
  DateTime now = rtc.now(); //Actualiza los datos de tiempo
```

Lo siguiente es comprobar si tenemos un vehículo enfrente del sensor de ultrasonidos y a que distancia.

```
// Leer sensor ultrasonidos
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);          //Enviamos un pulso de 10us
  digitalWrite(TRIG, LOW);
```

```
  pulse_time = 0;
  pulse_time = pulseIn(ECHO, HIGH, 75000); //Espermos 75000 que es lo que tardaria a 5m
  if(pulse_time != 0)   // Esto es para que si ha pasado el timeout no ponga distancia 0
    distancia = pulse_time * 0.033 / 2.0; // Calcula la distancia en centímetros, el dos es por ida y vuelta
```

Si hay un vehiculo y este se encuentra a menos de 1m mandamos una señal al ESP32-CAM para que capture una imagen y se la envie al PC.

```
  // Orden de tomar imagen
  if (distancia < 100){     // Si la distancia es menor a 1m enviamos la señal para que tome la imagen
    digitalWrite(SENAL_SENSOR,HIGH);
  }else{
    digitalWrite(SENAL_SENSOR,LOW);
  }
```

Cuando el PC haya procesado la imagen nos enviará una cadena por el puerto Serie. Leemos esa cadena de caracteres y la almacenamos en un string. Además nos guardamos que tenemos que abrir la barrera en una variable bool.

```
  // Recibir información
  if(Serial.available() && control == 0){
    matricula = Serial.read();
    control = 1;
  }
```

Si en este punto hemos recibido una matrícula la mostraremos por pantalla durante 1s. Si no en la pantalla aparecerá la hora.

```
  // Pantalla
  lcd.setCursor(0,0);
  if(matricula != ""){
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print(matricula);
    delay(1000);
    lcd.clear();
    matricula = "";
  }else{
```

```
    lcd.setCursor(0, 0);
    lcd.print("Hora: ");
    lcd.print(now.hour(), DEC);
    lcd.print(':');
    lcd.print(now.minute(), DEC);
  }
```

Por último, si anteriormente hemos decidido que teníamos que abrir la barrera mandamos una señal al controlador de Aprimatic.

```
  // Barrera
  if (control){
    digitalWrite(START,HIGH);
    delay(100);
    digitalWrite(START,LOW);
    control = 0;
  }
}
```

4.2.4.1. Conexión con sensores de entrada y salida

Uno de los elementos que enviará información a este microcontrolador central será una célula fotoeléctrica cuya función es informar de que un vehículo se encuentra frente a la barrera.

Esta conexión se realizará mediante pines de propósito general de entradas/salidas GPIO.

Para interconectar los sensores con las entradas del microcontrolador se desarrollará una placa electrónica con optoacopladores para adaptar el nivel de tensión y además separar eléctricamente los circuitos.

4.2.4.2. Conexión con sistema de reconocimiento

En función de la información recibida de los sensores desde el microcontrolador central este le envía la orden al subsistema de reconocimiento para que capte imágenes y las procese cuando sea necesario. Una vez obtenido un resultado este se le comunicará al microcontrolador central.

Esta conexión se realizará mediante una comunicación UART.

4.2.4.3. Conexión con pantalla

En la pantalla se mostraran diferentes mensajes según sean las entradas recibidas por el microcontrolador central, cuando no haya actividad se mostrará un mensaje que indique que el sistema está listo y esperando así como la hora en tiempo real.

Cuando se detecte un vehículo aparecerá un mensaje indicando que se están procesando los datos.

Cuando el sistema obtenga los resultados del sistema de reconocimiento estos se mostrarán en la pantalla junto con el tiempo correspondiente si procede.

Posteriormente al levantamiento de la barrera se indicará en la pantalla que el vehículo puede avanzar.

Cuando el sensor de salida detecte que el vehículo ya ha cruzado la barrera, la pantalla volverá al estado de espera.

La conexión entre esta pantalla y el microcontrolador central se realizará mediante una comunicación I2C.

4.2.4.4. Conexión con la central de control Traffic Park 24

A la central de control lo único que tenemos que enviarle es la señal de abrir, esto lo tenemos que hacer mediante un contacto abierto que tiene el cuadro.

Se ha optado por solucionar esto mediante un optoacoplador, otra opción podría haber sido un relé pero tienen un consumo y un desgaste mucho mayor.

El optoacoplador cumple perfectamente con las especificaciones que necesitamos ya que la central de control trabaja con 24V

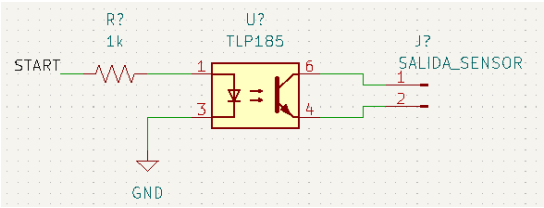


Ilustración 41: Señal de Start

Conectamos pues la entrada del optoacoplador a una salida del microcontrolador.

4.2.5. Implementación de sensor ultrasónico de distancia HC-SR04

El sensor HC-SR04 cumple con nuestras necesidades, tiene un rango de detección de 3-400cm y una resolución de 3mm parámetros más que suficientes para nuestra aplicación.

Este sensor funciona con una alimentación de 5V, esto representa un inconveniente ya que nuestro microcontrolador trabaja con 3.3V.

Además de la alimentación, el sensor tiene un pin de disparo TRIG y otro por donde recibimos el rebote ECHO.

Implementamos pues un adaptador de nivel de 3.3V a 5V, donde la parte de 3.3V va a unas entradas salidas del micro.

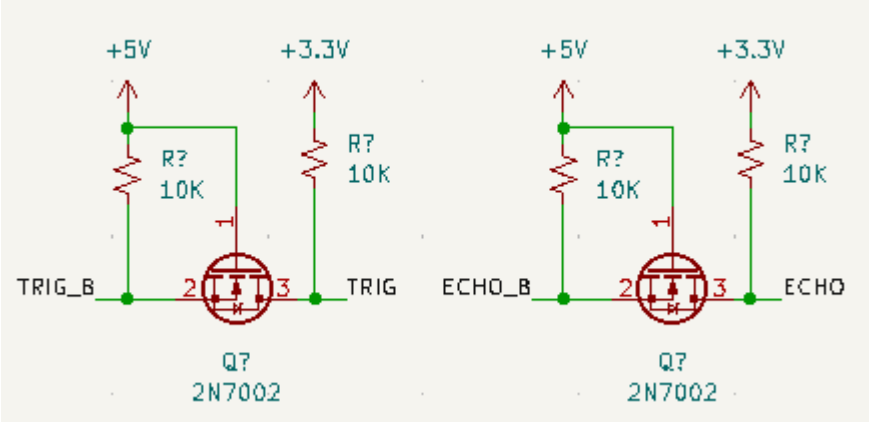


Ilustración 42: Adaptador de nivel

Y la otra parte junto con la alimentación a un conector de 4 pines.

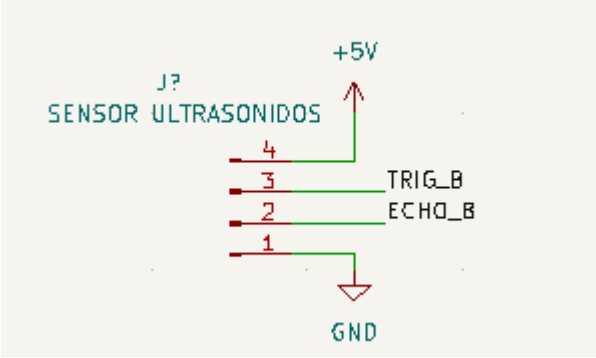


Ilustración 43: Conector sensor de ultrasonidos

4.2.6. Implementación RTC (Real Time Clock)

Para facilitar el control del tiempo se decide usar el chip DS3231, que tiene un oscilador interno y actúa como un reloj.

Podemos acceder a los datos que tiene almacenados en cualquier momento usando el bus I2C.

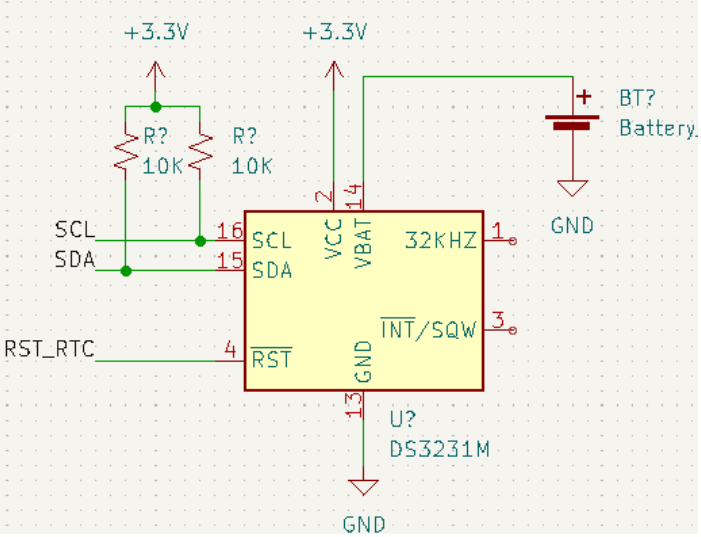


Ilustración 44: DS3231

Además se le puede añadir una batería para que aunque se vaya la luz o se apague el sistema no pierda la cuenta.

4.2.7. Implementación sensor óptico

Como se ha mencionado el sensor óptico tiene su propio microcontrolador, esto es así por el gran número de pines que requiere conexión del sensor óptico.

El software de este microcontrolador tiene como función capturar imágenes cuando se lo ordene el microcontrolador central y enviárselas posteriormente a un computador para su procesamiento y análisis.

Para poder realizar la captura de imágenes lo primero es configurar el microcontrolador para que pueda interactuar con el sensor OV2640.

```
#include <Arduino.h>
#include "esp_camera.h"

//Configuracion de pines de la placa ESP32-CAM
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
```

Además de ello vamos a definir un pin con el que podremos controlar la toma de imágenes desde el otro microcontrolador.

Así como especificar la velocidad del puerto serie con el PC.

```
//Definir un pin de entrada para comunicar con el otro microcontrolador
// No se puede escoger un pin del puerto serie
#define PIN_CONTROL    2

// Configuración del puerto serie, tiene que ser el mismo que
// en el script de python
```

```
#define SERIAL_BAUDRATE 115200
```

Después inicializamos el sensor y el puerto serie en la .

```
void setup() {

  // Inicializa la cámara
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;        //  FRAMESIZE_VGA  -
640x480   FRAMESIZE_HD  -1280x720
  config.jpeg_quality = 50;                 //Con 10 funciona
  config.fb_count = 2;

  esp_err_t err = esp_camera_init(&config);

  if (err != ESP_OK) {
    Serial.printf("Error al inicializar la cámara: %s\n",
esp_err_to_name(err));
    return;
  }
  // Inicializa el puerto serie
  Serial.begin(SERIAL_BAUDRATE);
  Serial.setTimeout(2000);
  Serial.println("ESP32-CAM inicializado.");
```

```
}
```

Por último en el bucle infinito realizamos las capturas y las enviamos por puerto serie siempre y cuando recibamos la orden del microcontrolador central.

```
void loop() {

if (PIN_CONTROL){

  // Toma una foto y envía los datos por el puerto serie
  camera_fb_t* fb = NULL;
  fb = esp_camera_fb_get();//Esta es la función que toma la imagen
  if (!fb) {
  Serial.println("Error al capturar la imagen");
  return;
  }

  Serial.write((const uint8_t*)fb->buf, fb->len);

  esp_camera_fb_return(fb);

  delay(600); //Con 1000 funciona bien pero lento
  //Es necesario para no saturar el puerto serie
  }

delay(100);
}
```

4.2.8. Implementación red neuronal

4.2.8.1. Entrenamiento red neuronal

Para entrenar una red neuronal necesitamos muchos datos, además estos datos deben estar clasificados.

En nuestro caso los datos son imágenes de caracteres de matrículas y se encuentran clasificados en carpetas cuyo nombre es el del carácter de las imágenes.

A esto se le llama dataset, el que hemos escogido es de acceso público.

Ilustración 45: Imagen 1 dataset

Ilustración 46: Imagen 2 dataset

Ilustración 47: Imagen 3 dataset

De cada carácter tenemos 1000 imágenes, estas representan el número o la letra desde diferentes ángulos y con distintos tamaños e iluminaciones.

La red neuronal que vamos a entrenar se ejecutara y realizaremos su aprendizaje en Python mediante la librería TensorFlow y usando la capa Keras que nos simplifica el proceso.

Comenzando con el código que vamos a ejecutar en Python lo primero que debemos hacer es importar todas las librerías que posteriormente usaremos

```
import tensorflow as tf
import numpy as np
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

Despues indicamos la ruta donde se encuentra el dataset que queremos usar, la ruta exacta coincide con la carpeta que contiene todas las carpetas clasificadas.

```
# Define la ruta de la carpeta principal que contiene todas las subcarpetas
ruta_principal =
"C:\\Users\\Usuario\\Desktop\\TFG\\RECURSOS\\datasets\\CNN_letter_Dataset"
```

Definimos el tamaño que tienen las imágenes, esto será importante mas adelante, para saber el tamaño seleccionamos una imagen y miramos sus propiedades.

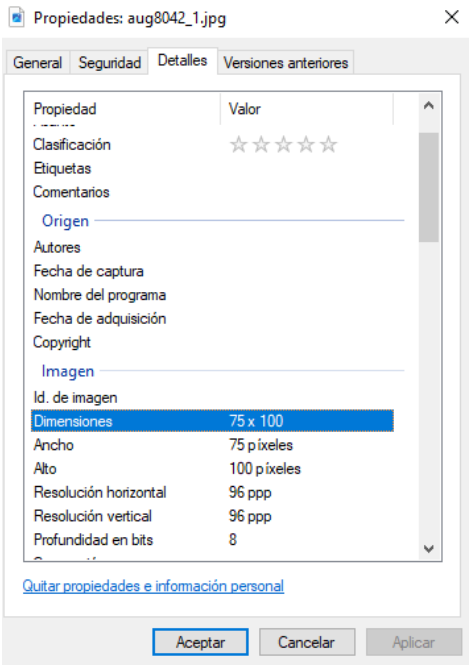


Ilustración 48: Propiedades Imagenes

Todas las imágenes tienen que tener el mismo tamaño, si queremos podemos modificarlo en el mismo programa de Python al importarlas.

```
# Define el tamaño deseado de las imágenes
altura_imagen = 100
ancho_imagen = 75
```

Lo siguiente que tenemos que hacer es importar las imágenes y ordenarlas mediante listas.

```
# Crea una lista vacía para almacenar tus datos y etiquetas
datos = []
etiquetas = []

# Recorre todas las subcarpetas en la carpeta principal y carga los datos en
la lista de datos
# y las etiquetas en la lista de etiquetas
for etiqueta, nombre_carpeta in enumerate(os.listdir(ruta_principal)):
    ruta_carpeta = os.path.join(ruta_principal, nombre_carpeta)
    for nombre_archivo in os.listdir(ruta_carpeta):
        ruta_archivo = os.path.join(ruta_carpeta, nombre_archivo)
        imagen = tf.keras.preprocessing.image.load_img(ruta_archivo,
target_size=(altura_imagen, ancho_imagen), color_mode='grayscale')
        imagen_array = tf.keras.preprocessing.image.img_to_array(imagen)
        datos.append(imagen_array)
        etiquetas.append(etiqueta)
```

Para poder operar con las listas necesitamos convertirlas a un tipo llamado numpy.

```
# Convertir las listas a matrices numpy
X = np.array(datos)
y = np.array(etiquetas)
```

Dividimos los datos del dataset en dos partes, una usada para entrenar y la otra para la fase de validación. También especificamos el número de categorías.

```
# Dividir los datos en conjuntos de entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3,
random_state=42)

y_train = to_categorical(y_train, num_classes=35)
```

```
y_val = to_categorical(y_val, num_classes=35)
```

Por último normalizamos los valores, esto quiere decir dejarlos entre 0 y 1.

```
# Normalizar las imágenes
X_train = X_train / 255.0
X_val = X_val / 255.0
```

Con esto tenemos preparados los datos para usarlos en el entrenamiento de la red neuronal.

Ahora vamos a configurar nuestra red neuronal, estableciendo en primer lugar las capas que va a tener el modelo.

Estas serán la capa de entrada, las capas ocultas (si las hay) y la capa de salida. En nuestro caso la capa de entrada es una matriz que coincide con los pixeles de las imágenes y la capa de salida es una lista que corresponde con las etiquetas que hemos establecido.

Las capas ocultas quedan a nuestra elección. A priori más capas crean redes más complejas y más precisas pero también depende de otros factores y luego el tiempo necesario para entrenar la red aumenta enormemente y también el tiempo que tarda en ejecutarse el modelo cuando se usa.

También hay que tener en cuenta los activadores de las neuronas, para este caso la capa de salida lo más adecuado es que sean de tipo softmax, esta función de activación hace que de entre todas las salidas la que tenga un valor más alto se active con un 1 y las demás se ponen a 0. Además usaremos la función relu que convierte todos los valores negativos en 0 y no produce modificaciones sobre los positivos.

Por último especificamos el orden de las capas y el nombre del modelo.

Compilamos el modelo con los parámetros de optimizador, función de perdida (esto evaluará las diferencias entre las predicciones y los datos de validación) y un último parámetro para que podamos ver lo preciso que es nuestro modelo.

```
layer1 = tf.keras.layers.Flatten(input_shape= (100,75,1))   #100x75 pixels, el 1 es porque es escala de grises
layer2 = tf.keras.layers.Dense(150, activation = 'relu')     #150 neuronas, buscar informacion sobre relu (funciones de activación)
layer3 = tf.keras.layers.Dense(150, activation = tf.nn.relu)
layer4 = tf.keras.layers.Dense(35, activation = tf.nn.softmax)
```

```
model = tf.keras.Sequential([layer1, layer2, layer3, layer4])
model.compile(optimizer = 'adam', loss =
tf.keras.losses.CategoricalCrossentropy(), metrics = ['accuracy'])
```

Ahora ya solo queda entrenar el modelo, para ello usamos fit en el modelo que es un objeto de Keras y posee este método. En el indicamos los valores de entrenamiento, el tamaño de los lotes, el número de veces que se repetirá el proceso (épocas), los datos de validación y un verbose para que no muestre por pantalla todo lo que ejecuta.

```
model.fit(X_train, y_train, batch_size=32, epochs=100,
validation_data=(X_val, y_val), verbose=1)
```

Guardamos el modelo ya entrenado en un archivo con extensión .h5 propia de TensorFlow y de Keras, en el guardamos tanto los pesos como la configuración de cómo ha sido entrenado el modelo.

```
model.save('modelo_gpu.h5')
```

4.2.8.2. Uso de la red neuronal entrenada

Para poder usar el modelo que hemos entrenado en un código de Python tenemos que importar la librería de TensorFlow y especificar la rauta donde hemos guardado el modelo.

```
import tensorflow as tf
# Cargar el modelo
model =
tf.keras.models.load_model('C:\\Users\\Usuario\\Desktop\\TFG\Programas\\Pyth
on\\modelo_gpu.h5')
```

Ahora podemos usar el modelo cuando queramos, tan solo tenemos que proporcionarle una imagen en las mismas condiciones con las que ha sido entrenado, en escala de grises y de un tamaño de 100x75 pixeles.

```
# Usar el modelo
prediction = model.predict(imagen_array)
```

4.2.9. Procesamiento de imágenes

El procesamiento de imágenes se realizará en un dispositivo con capacidad para ejecutar Python.

Importamos todas las librerías que usaremos.

```
import serial
import time
import os                   # Para manejar archivos, rutas y ficheros
from PIL import Image       # Para trabajar con imagenes
import struct
import cv2                  # Necesario instalar con "pip install opencv-
python"
import numpy as np
import io
import matplotlib.pyplot as plt
from datetime import datetime
import tensorflow as tf
```

Las imágenes las recibe a través de un puerto serie. Configuramos el puerto serie al que conectaremos el sensor óptico. Las opciones RTS (Ready To Send) y DTR (Data Terminal Ready) las ponemos en LOW para que el puerto este escuchando.

```
# Configuración del puerto serie
serial_connection = serial.Serial(port = 'COM3', baudrate = 115200, timeout
= 0.5) #0.5
serial_connection.setRTS(False)
serial_connection.setDTR(False)
```

Ahora iniciamos un bucle infinito y leemos los datos que haya en el buffer del puerto serie.

```
    # Lee los datos recibidos por el puerto serie
    data = serial_connection.read(2000000)
```

Hacemos diferentes operaciones transformando los datos para conseguir que la imagen pueda ser usada por la librería openCV.

En todo momento comprobamos que no hay errores, si algún paso falla no seguimos avanzando y volvemos al inicio del bucle. Esto lo

conseguimos con sentencias "if" anidadas, si no se cumplen las condiciones vamos directamente al final del bucle.

```
    #Creamos una variable vacia
    img = None

    # Pasamos los datos a un array de numpy
    data_np = np.frombuffer(data, dtype=np.uint8)

    # Verificamos si el array no está vacío
    if len(data_np) == 0:
        print('El búfer de datos está vacío')
    else:
    # Decodificar la imagen a partir del búfer de datos
        img = cv2.imdecode(data_np, cv2.IMREAD_COLOR)
```

Después se aplican filtros de la librería openCV, primero convertimos la imagen a escala de grises, después reducimos el ruido y por último aplicamos un detector de bordes.

Además guardamos una copia de la imagen.

```
    # Verificar si la imagen se cargó correctamente
    if img is None:
            print('La imagen no se pudo cargar')
    else:
        # Copia de imagen
        img_cpy = img

        # Convertir a escala de grises
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Reducir ruido
        median = cv2.medianBlur(gray, 5)

        # Filtro canny detector de bordes
        edges = cv2.Canny(median, 40, 100)
```

El problema que nos surge en este punto es que los bordes que nos da el filtro canny no son perfectos, suelen estar abiertos en las esquinas.

Para solucionar esto engrosamos las líneas que se han creado con canny.

```
        # Aplicar operación de dilatación
        kernel = np.ones((3,3), np.uint8)   # Creamos un kernel con una
matriz 3x3 con todo unos
        # Se aplica el kernel sobre los bordes
        dilation = cv2.dilate(edges, kernel, iterations=1)

        # Buscar los contornos _ es para retornos que no se utilizan, los
retornos de findContours han cambiado y tenemos que dejar el primer y tercer
argumento vacío
        _, contours, _ = cv2.findContours(dilation, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

Tenemos que determinar cuál de los contornos que se han creado es el de la matrícula. Para ello definimos las características de la matrícula en pixeles, aquí dependerá mucho de la distancia entre el vehículo y el sensor óptico. El ratio siempre será el mismo pero también le daremos un margen ajustable de error.

Guardamos todos los contornos que cumplan los requisitos en un array.

```
        '''PROPIEDADES DE UNA MATRICULA
        520x110mm   relacion de aspecto = 4.27'''
        ratio = 4.27
        min_w = 100
        max_w = 600
        min_h = 25
        max_h = 150

        # Convierte la matriz NumPy en un objeto de imagen de Pillow
        imagen2 = Image.fromarray(dilation)

        candidates = []

        for cnt in contours:
            cnt_copy = cnt
            x, y, w, h = cv2.boundingRect(cnt)
            if (h != 0):
                aspect_ratio = float(w) / h
                if (np.isclose(aspect_ratio, ratio, atol=5.0) and
                (max_w > w > min_w) and
                (max_h > h > min_h)):
```

```
                    print(cv2.contourArea(cnt))
                    if (cv2.contourArea(cnt) >= min_h * min_w) and
(cv2.contourArea(cnt) <= max_h * max_w):
                        candidates.append(cnt_copy)
```

Comprobamos que ha encontrado algún contorno valido y luego descartamos todos los contornos menos el más grande.

```
        if candidates == []:
            print('No hay candidatos')
        else:
            # Seleccionar el contorno más grande como la matrícula
            plate_contour = max(candidates, key=cv2.contourArea)
```

Aproximamos el contorno que hemos escogido a un polígono que si coincide con el de la matricula debería tener 4 esquinas.

```
            # Obtener un polígono de aproximación a partir del contorno
            epsilon = 0.05 * cv2.arcLength(plate_contour, True)
            approx = cv2.approxPolyDP(plate_contour, epsilon, True)

            # Obtener las coordenadas de las esquinas del polígono
            corners = approx.reshape(-1, 2)
```

Comprobamos que todo está correcto.

Después de esto nos encontramos con un problema, al aplicar esta función en ocasiones las esquinas se desordenan.

Para solucionar esto analizamos el orden que tienen que seguir, e ideamos un algoritmo que las ordene, por último comprobamos que todas las esquinas tienen un valor adecuado, que no es 0.

```
            # Comprobar que tiene 4 esquinas, sino eliminar
            if len(corners)!=4  :
                print('IMAGEN NO VÁLIDA')
                print(corners)
                print('---------')
                print(len(corners))
                print('---------')

            else:
                print('---OK---')
                # Imprimir las coordenadas de las esquinas
```

```
                print(corners)
                print('---------')
                print(len(corners))
                print('---------')

                # Dibujar un polígono con las esquinas
                cv2.polylines(img, [corners], True, (0, 255, 0),
thickness=2)

                #Realizar algoritmo para ordenar porque a veces es aleatorio
                corners_cpy = []
                corners_ord = [[], [], [], []]

                # Calcular el centroide del contorno
                M = cv2.moments(plate_contour)

                if M['m00'] != 0:
                    cx = int(M['m10'] / M['m00'])
                    cy = int(M['m01'] / M['m00'])

                    print(cx)
                    print(cy)

                    for i in range(0,4):
                        if (corners[i][0] - cx < 0 ) and (corners[i][1] - cy
< 0 ):
                            corners_ord[0] = corners[i]
                        if (corners[i][0] - cx > 0 ) and (corners[i][1] - cy
< 0 ):
                            corners_ord[1] = corners[i]
                        if (corners[i][0] - cx > 0 ) and (corners[i][1] - cy
> 0 ):
                            corners_ord[2] = corners[i]
                        if (corners[i][0] - cx < 0 ) and (corners[i][1] - cy
> 0 ):
                            corners_ord[3] = corners[i]

                    print(corners_ord)

                    if corners_ord[0] != [] and corners_ord[1] != [] and
corners_ord[2] != [] and corners_ord[3] != []:
                        corners_ord = np.float32(corners_ord)
                        print(corners_ord)
```

Después se corrige el error de perspectiva que tiene la imagen al ser tomada desde un lateral y al mismo tiempo se redimensiona para tener un tamaño específico de 520x110 pixeles.

```
                    # RECTIFICAR PERSPECTIVA
                    dst = np.float32([[0,0], [520,0], [520,110],
[0,110]])
                    M = cv2.getPerspectiveTransform(corners_ord, dst)
                    perspective = cv2.warpPerspective(img_cpy, M, (520,
110))
```

4.3. PRUEBAS

Una vez definida la parte del procesamiento de imágenes se comienzan a realizar pruebas

Se comienza con matrículas impresas en papel.

Ilustración 49: Prueba funcionamiento 1

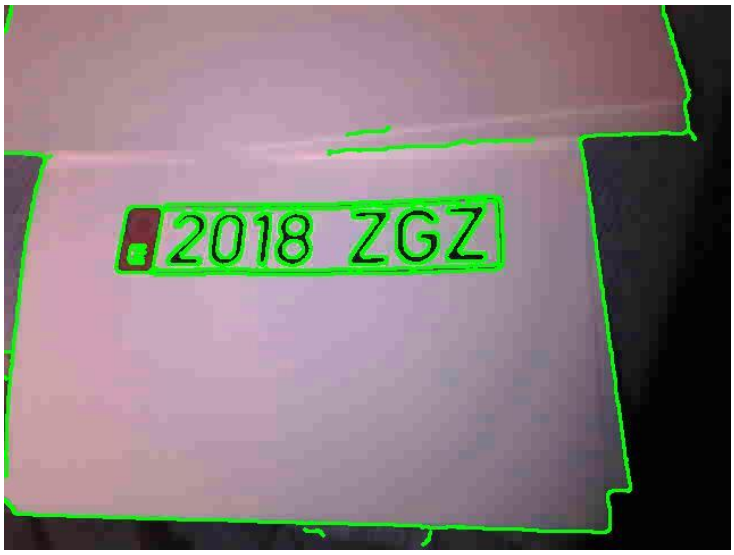


Ilustración 50: Prueba funcionamiento 2

Los resultados son muy buenos, hay un gran porcentaje de acierto.

Incluso con ángulos bastante inclinados siguen siendo buenos los resultados.

Después probamos con imágenes más complejas en las que aumenta el ruido.

Ilustración 51: Prueba funcionamiento 3

Ilustración 52: Prueba funcionamiento 4

El resultado aquí es bastante bueno, se puede apreciar como cierra por completo el contorno de la matricula mientras qu los demás se encuentran abiertos, esto haría que el algoritmo identificara como bueno el de la matrícula.

Es decir cómo queremos que suceda.

Ilustración 53: Prueba funcionamiento 5

Ilustración 54: Prueba funcionamiento 6

Con esta última imagen no se obtuvieron buenos resultados, tal vez sea porque se encontraba algo pixelada, el filtro canny es demasiado sensible y se marcan los cambias de color cuando no tendría que ser así.

4.4. PROTOTIPO

Con el proyecto más avanzado se decide fabricar una plantilla para probar diferentes combinaciones.

Para ello se imprime en 3D con PLA un marco y unos dígitos

Ilustración 55: Prototipo 1

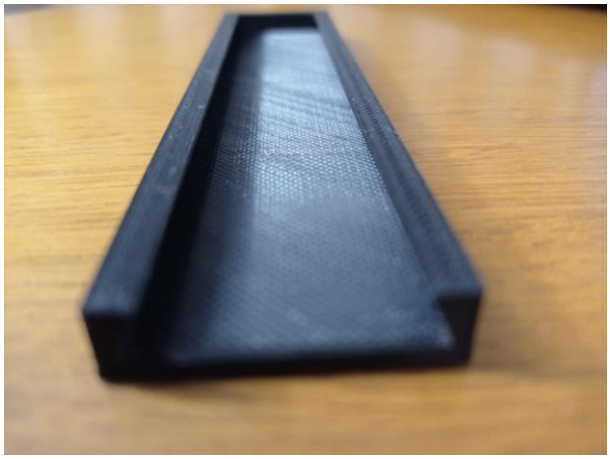


Ilustración 56: Prototipo 2

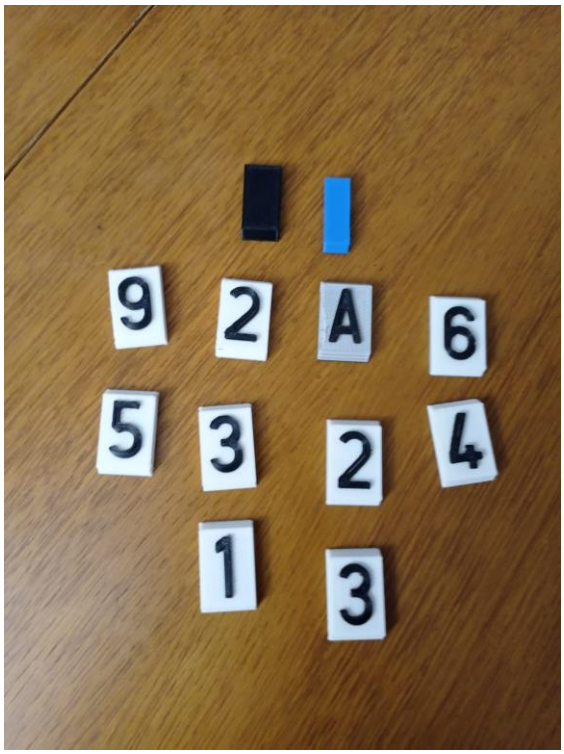


Ilustración 57: Prototipo 3

Ilustración 58: Prototipo 4

Se realizan pruebas de reconocimiento con el OCR incluido.

Ilustración 59: Prototipo 5

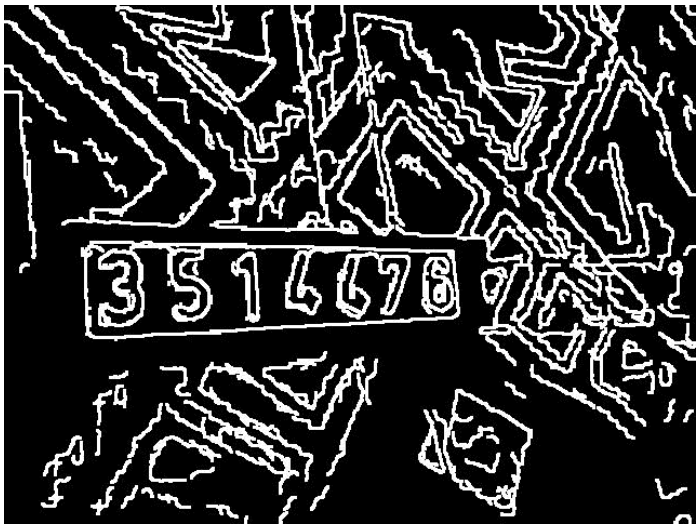


Ilustración 60: Prototipo 6

Podemos ver que a pesar del ruido el resultado es bastante bueno,

Ilustración 61: Prototipo 7

Después de pasar por el corrector de perspectiva

Ilustración 62: Prototipo 7

Aquí se puede ver los dígitos separados.

Despues se añade un contorno blanco para que respete las dimensiones de entrada al modelo de predicción y obtenemos un resultado.

Por ahora las pruebas son satisfactorias, se producen algunos errores de confusión de caracteres. Sobre todo la letra I con el número 1.

Despues se conecta al PC un ESP32 con una pantalla LCD para visualizar el resultado.

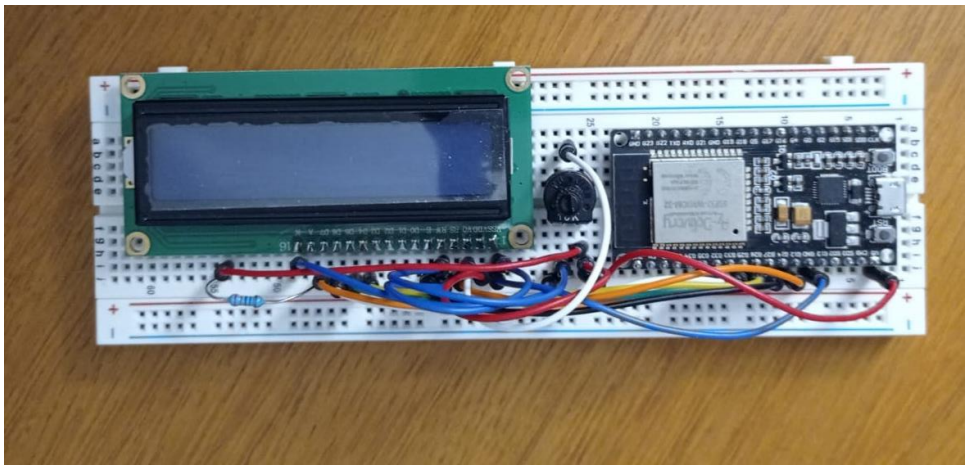


Ilustración 63: Prototipo 8

5. CONCLUSIONES

A lo largo del proceso de realización de este proyecto he aprendido muchas habilidades y conocimientos.

Mi objetivo personal con este proyecto era aprender sobre las redes neuronales, su funcionamiento, sus aplicaciones y su creación.

Este objetivo ha quedado se sobras cumplido y me ha hecho ver el potencial que tiene para mejorar muchos procesos.

Y es que me he dado cuenta de que con la cantidad suficiente de datos bien clasificados un algoritmo puede hacer casi cualquier cosa. El potencial de este tipo de tecnologías es enorme y este año (2023) ha sido muestra de ello.

Otro aprendizaje importante que me llevo es aprender a gestionar un proyecto de esta envergadura. Con varias fases y elementos diferentes que al final se unen para crear un sistema.

Los objetivos iniciales se han cumplido, se ha diseñado un sistema capaz de reconocer matriculas, se ha implementado dentro de una barrera automática y se ha probado de forma experimental que funciona.

Para continuar con el desarrollo de este proyecto se podría ahondar en el mundo de las bases de datos e incluir una a todo el sistema que se ha desarrollado.

6. OBJETIVOS DE DESARROLLO SOSTENIBLE

Los objetivos de este Trabajo Fin de Grado están alineados con los siguientes Objetivos de Desarrollo Sostenible (ODS) y metas, de la Agenda 2030:

- Objetivo 9 - Conseguir infraestructuras sostenibles, resilientes y de calidad para todos, impulsar una nueva industria bajo criterios de sostenibilidad que adopte tecnologías y procesos industriales limpios y ambientalmente racionales, fomentar la tecnología, la innovación y la investigación y lograr el acceso igualitario a la información y al conocimiento, principalmente a través de internet.

- Meta 9.4 De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas

- Objetivo 12 - Cambiar el modelo actual de producción y consumo para conseguir una gestión eficiente de los recursos naturales, poniendo en marcha procesos para evitar la pérdida de alimentos, un uso ecológico de los productos químicos y disminuir la generación de desechos.

- Meta 12.2 De aquí a 2030, lograr la gestión sostenible y el uso eficiente de los recursos naturales

7. BIBLIOGRAFÍA

Aliexpres. (2023). *Módulo de pantalla táctil capacitiva HMI I2C IIC LCD para Arduino 2,8, 3,5, 4,3 y 5 pulgadas, 480x320—AliExpress*. https://es.aliexpress.com/item/1005001580662616.html?src=google&src=google&albch=shopping&acnt=439-079-4345&slnk=&plac=&mtctp=&albbt=Google_7_shopping&albagn=888888&isSmbAutoCall=false&needSmbHouyi=false&albcp=20330

Aprimatic. (2023a). *Manual-Barrera-Park-40-60.pdf*.

Aprimatic. (2023b). *T24M.pdf*.

Aprimatic. (2023c). *TRAFFIC PARK 24 APRIMATIC MANUAL.pdf*.

Aprimatic. (2019). *Barrera Aprimatic Park 60 24v con mástil de 6m (4m + 2m)*. AUTOMATISMOS W2B. https://www.webdosb.com/barreras-automaticas/6251-barrera-aprimatic-park-60-24v-con-mastil-de-6m-4m-2m.html?gad_source=1&gclid=Cj0KCQiApOyqBhDlARIsAGfnyMp1fHPlG5Wb2S3R-5zptK_ijrSFy70-qi2nHqrGGfR9Y2FKdw9iY_MaAmhXEALw_wcB

Arducam. (2017). *OV2640: Specs, Camera, Datasheet & Alternative (2022 Report)*. https://www.arducam.com/ov2640/

Arducam. (2020). *Raspberry Pi Camera Pinout—Arducam*. https://www.arducam.com/raspberry-pi-camera-pinout/

Bronchal Paricio, M. (2017). *Diseño e implementación de un prototipo de radar de tramo basado en Raspberry Pi*.

Carlo Gavazzi. (2003). *PMR10RGT.pdf*.

Coltec. (2022, febrero 15). *Lector de matrículas para el control de vehículos*. https://www.coltecsecurity.com/lector-de-matriculas-lpr/

Conure. (2022, abril 6). Is LPR different from ALPR and ANPR. *Conure*. https://www.conurets.com/is-lpr-different-from-alpr-and-anpr/

DigiKey. (2023). *ESP32-S3-WROOM-1-N8*. Digi-Key Electronics. https://www.digikey.es/es/products/detail/espressif-systems/ESP32-S3-WROOM-1-N8/15200089

Everen, D. V. (2023, febrero 28). *The History of OCR*. Veryfi. https://www.veryfi.com/ocr-api-platform/history-of-ocr/

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (2003). *Feature Detectors—Sobel Edge Detector*. https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

Gimenez-Palomares, Monsoriu, Alemany-Martinez, E. (2016). Aplicación de la convolucón de matrices al filtrado de imagenes. *Ingeniería del agua*, *18*(1), ix. https://doi.org/10.4995/ia.2014.3293

historyphotography. (2014, abril 7). Bayer Color Filter. *Historyphotography*. https://historyphotography.wordpress.com/2014/04/07/bayer-color-filter/

Huitema, E., & French, I. (2022, enero 22). *How E Ink Developed Full-Color e-Paper—IEEE Spectrum*. https://spectrum.ieee.org/how-e-ink-developed-full-color-epaper

jaintarum. (2022, marzo 16). Python Tensorflow—Tf.keras.layers.Conv2D() Function. *GeeksforGeeks*. https://www.geeksforgeeks.org/python-tensorflow-tf-keras-layers-conv2d-function/

Mora, R. C. (2020, noviembre 26). Instalación de TensorFlow y entorno de desarrollo Python en Mac. *Adictos al trabajo*. https://www.adictosaltrabajo.com/2020/11/26/instalacion-de-tensorflow-y-entorno-de-desarrollo-python-en-mac/

NVIDIA. (2021). *Convolution*. NVIDIA Developer. https://developer.nvidia.com/discover/convolution

Pardo Herrera, C. J. (2019, abril 17). *Deep Learning + Detección de bordes (HED) / Holistically-Nested Edge Detection*. https://carlosjuliopardoblog.wordpress.com/2019/04/17/deep-learning-deteccion-de-bordes-hed-holistically-nested-edge-detection/

published, L. S. (2022, noviembre 11). *Photography cheat sheet: What is Field of View (FoV)?* Digitalcameraworld. https://www.digitalcameraworld.com/tutorials/photography-cheat-sheet-what-is-field-of-view-fov

Redondo Sanz, D. (2020). *CONTROL DE UN PUENTE GRÚA DIDÁCTICO MEDIANTE VISUALIZACIÓN DE SU CARGA*.

RFC Puertas Automaticas. (2013). *RFC Puertas automaticas en Madrid, Instalación, reparación y mantenimiento de puertas de garaje en arroyomolinos—Normativa puertas*. https://www.puertasautomaticasonline.com/normativa-puertas/

Rodríguez, F. (2023, mayo 2). *(1) 7 Consejos para Trabajar con Redes Neuronales en Aprendizaje Automático | LinkedIn*. https://www.linkedin.com/pulse/7-consejos-para-trabajar-con-redes-neuronales-en-rodr%C3%ADguez-mgs/?originalSubdomain=es

Rosalie. (2016, agosto 24). *LED and OLED Technology in Overview*. HiFi & Friends. https://www.hifi-im-hinterhof.de/blog/en/led-and-oled-technology-in-overview/

Rouizi, Y. (2022, noviembre 7). *How to Remove Background with OpenCV | Don't Repeat Yourself*. https://dontrepeatyourself.org/post/how-to-remove-background-with-opencv/?utm_content=cmp-true

Sánchez-Agustino, F. J. N. (2016). *Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional*.

Satria, G. (2021). *What are ANPR, ALPR, LPR Camera? - WSH PEER*. https://www.wsh-peer.com/en/what-are-anpr-alpr-lpr-camera/

Segurdoma ACS®. (2016). *Barreras para Parking ELKA*. Segurdoma ACS. https://segurdoma.es/control-de-accesos/barreras-parking-elka

Survision, S. (2020). *ANPR or LPR: What is the difference?* Survision | License Plate Recognition Cameras. https://survisiongroup.com/post-anpr-or-lpr:-what-is-the-difference

Techmake, E. (2020, mayo 12). *Empezando con Arduino - 3C: Sensor Ultrasónico HC-SR04*. Techmake Solutions. https://techmake.com/blogs/tutoriales/empezando-con-arduino-3c-sensor-ultrasonico-hc-sr04

Traco Power. (2023). *Tmlm.pdf*.

Walton. (2020). *Liquid crystal display (LCD) | Britannica*. https://www.britannica.com/technology/liquid-crystal-display

Wikipedia. (2023). PlatformIO. En *Viquipèdia, l'enciclopèdia lliure*. https://ca.wikipedia.org/w/index.php?title=PlatformIO&oldid=32152928

Relación de documentos

(X) Memoria 94 páginas
(_) Anexos 10 páginas

La Almunia, a 22 de 11 de 2023

Firmado: Octavio Marín Carnicer