Trabajo de Fin de Grado
Ingeniería en Tecnologías Industriales

Análisis de algoritmos de detección de características de OpenCV en Raspberry Pi

Autor: Javier Jesús de Vicente Sugue
Tutores: Ignacio Alvarado Aldea
Mario Pereira Martín

Dpto. Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo de Fin de Grado
Ingeniería en Tecnologías Industriales

Análisis de algoritmos de detección de características de OpenCV en Raspberry Pi

Autor:
Javier Jesús de Vicente Sugue

Tutor:
Ignacio Alvarado Aldea
Profesor titular:
Mario Pereira

Dpto. de Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo de Fin de Grado: Análisis de algoritmos de detección de características de OpenCV en Raspberry Pi

Autor: Javier Jesús de Vicente Sugue

Tutor: Ignacio Alvarado Aldea

Profesor Titular: Mario Pereira Martín

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia
A mis amigos

Agradecimientos

Gracias a todas las personas que han formado parte de mi vida de una u otra forma.
Gracias a mi familia.

Resumen

En el proyecto que se presenta a continuación, se va a realizar el estudio de algunos métodos de detección de características de la librería de visión artifical “OpenCV” para evaluar su eficiencia a la hora de implementar algoritmos de visión artifical en sistemas embebidos, concretamente el dispositivo que se utiliza en este proyecto es la Raspberry Pi.

La Raspberry Pi es considerada en la actualidad como un sistema con mucho potencial ya que pese a sus proporciones es capaz de presentar un mejor rendimiento que muchos ordenadores de sobremesa [1]. En este proyecto se va a explicar por qué es una gran opción para implementar en vehiculos autónomos.

Los algoritmos que se van a tratar son SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features) y ORB (Oriented Fast and Rotated Brief) y van a ser evaluados en función de unos parámetros específicos. Estos parámetros han sido específicamente elegidos debido a que tienen mucho peso a la hora de ejecutar programas en sistemas embebidos y son: el número de puntos clave detectados en las distintas imagenes utilizadas y el tiempo de ejecución del algortimo elegido en cada caso.

Los resultados serán obtenidos a través de simulaciones realizadas con imagenes ya que, en definitiva, una secuencia de video es simplemente un conjunto de fotogramas reproducidos uno tras otro y es suficiente para obtener resultados válidos para el fin de este proyecto.

La idea de la realización de este proyecto surje de el deseo de crear sistemas de visión artificial portátiles que puedan ser utilizados en pequeños vehiculos motorizados o en robots, por ello deben ser fisicamente lo mas ligeros y pequeños posibles, intentando sacar el máximo rendimiento a estos dispositivos para obtener un sistema robusto y que pueda ser ejecutado en tiempo real.

Con los resultados obtenidos de este proyecto se pretende crear un sistema de visión artificial para vehículos autónomos con distintas aplicaciónes de visión artifical tales como detección de objetos, mapeado de entornos y otras aplicaciónes útiles.

Abstract

In the project presented below, the study of some detection methods of the characteristics of the artificial vision library "OpenCV" will be carried out to evaluate its efficiency when implementing artificial vision algorithms in embedded systems, specifically the device used in this project is the Raspberry Pi.

The Raspberry Pi is currently considered as a system with a lot of potential since, despite its proportions, it is capable of presenting a better performance than many desktop computers [1]. This project will explain why it is a great option to implement in autonomous vehicles.

The algorithms to be treated are SIFT (Scale Invariant Feature Transform), SURF (Speeded Up Robust Features) y ORB (Oriented Fast and Rotated Brief) and will be evaluated based on specific parameters. These parameters have been specifically chosen because they have a lot of weight when executing programs on embedded systems and they are: the number of key points detected in the different images used and the execution time of the algorithm chosen in each case.

The results will be obtained through simulations made with images since, in short, a video sequence is simply a set of frames reproduced one after another and it is enough to obtain valid results for the end of this project.

The idea of carrying out this project arises from the desire to create portable artificial vision systems that can be used in small motorized vehicles or robots, therefore they must be physically as light and small as possible, trying to get the most out of them to obtain a robust system that can be executed in real time.

The results obtained from this project are intended to create an artificial vision system for autonomous vehicles with different applications of artificial vision such as object detection, environment mapping and other useful applications.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 ESTADO DEL ARTE

Las cosas no se hacen siguiendo caminos distintos para que no sean iguales, sino para que sean mejores.
- Elon Musk -

La visión artificial es una disciplina científica que incluye métodos para capturar, procesar y analizar imágenes reales para generar información que pueda ser procesada por una máquina.
El principal objetivo de la visión artificial es proporcionar "ojos" a la máquina para ver lo que está sucediendo en el mundo real y poder tomar decisiones involucradas en la automatización de cualquier proceso.

Una forma sencilla de entender este sistema es traduciéndolo a nuestros propios sentidos. Los seres humanos utilizamos nuestros ojos para comprender el mundo que tenemos a nuestro alrededor y la visión artificial trata de reproducir ese mismo efecto en las máquinas de tal forma que podrán recibir una imagen o una secuencia de varias imágenes y elegir la forma de actuar que lleverán a cabo en situaciones determinadas.
Esta "comprensión" en las máquinas se consigue descomponiendo la imagen en pequeñas partes (píxeles) y realizando un estudio de sus características.

En la actualidad, la tecnología y los sistemas automatizados implementados en la industria son el fruto de años de evolución científica continua tanto en investigación como en innovación de los sistemas de visión artificial y sus aplicaciones.
La evolución de estos sistemas en la industria ha estado estrechamente relacionada con el desarrollo de las cámaras fotográficas y la extracción de imágenes para realizar su análisis con fines científicos.

El origen de los sistemas de visión artificial se remonta a la década de 1960 con la creación de un prototipo automatizado basado en cámaras y sistemas de procesamiento de imágenes. Se buscaba tener acceso a determinadas estructuras con la finalidad de analizar su contenido.
En la década de 1980 gracias a el desarrollo de la ingeniería informática y la creación de procesadores más potentes se consiguieron microprocesadores que podían captar, procesar y reproducir imágenes de una cámara conectada de manera remota.
En la actualidad, ha avanzado tanto el estudio de estos sistemas que podemos encontrarlos implementados en nuestra vida diaria.

En la industria es común encontrar sistemas de visión artificial implementados en fábricas para realizar el control de calidad de los productos ya que hacerlo de manera manual sería una tarea demasiado tediosa para un operario, así las empresas consiguen gran precisión y un menor "lead time" de su producto. Como ejemplo de esto, encontramos clasificadores de frutas que utilizan la visión artificial para analizar piezas de fruta en una línea de embalaje descartando automáticamente las piezas que no son aptas para consumo [2].

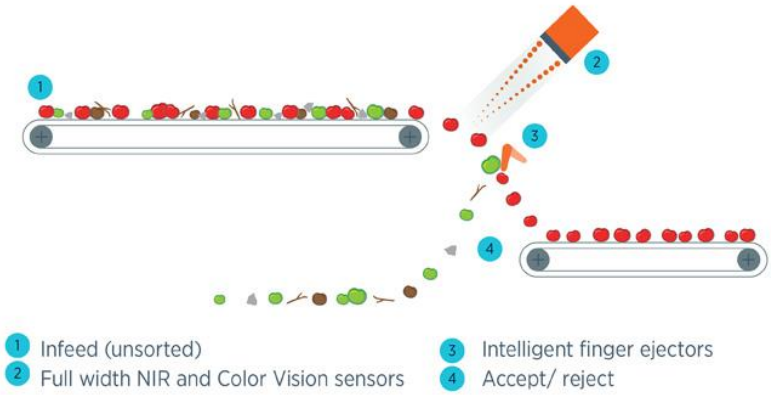


Figura 1. Clasificadora de campo NFM de la marca TOMRA.

Al avanzar los procesadores de nuestros dispositivos móviles hemos conseguido disponer entre otras muchas funciones de de reconocimiento facial, llegando a, en algunos casos, poder utilizar este software para funciones como la fotografía. Con nuestros propios dispositivos móviles podemos detectar rostros de manera automática antes de realizar una fotografía para realizar así un mejor enfoque incluso pudiendo detectar cuando una persona sonríe de tal forma que su sonrisa sea el disparador que haga que la cámara tome una fotografía. Esta tecnología la utilizan móviles con el sistema operativo Android permitiendo a sus usuarios el desbloqueo del dispositivo mediante detección facial [3].

Figura 2. Sistema de detección facial utilizado en cámaras de móviles.

Otro de los sectores que se está beneficiando mucho de estos avances es el sector automovilístico. La empresa Tesla utiliza un sistema de visión artificial que junto con inteligencia artificial (IA) permite a sus vehículos navegar de forma autónoma utilizando métodos como redes neuronales, mecanismos de autonomía, …
La empresa todavía no ha conseguido la conducción autónoma completa (en Tesla es denominado Autopilot) pero esperan conseguirlo durante el año 2020. Calculan que su sistema será capaz de procesar hasta 2.5 gigapíxeles por segundo [4].

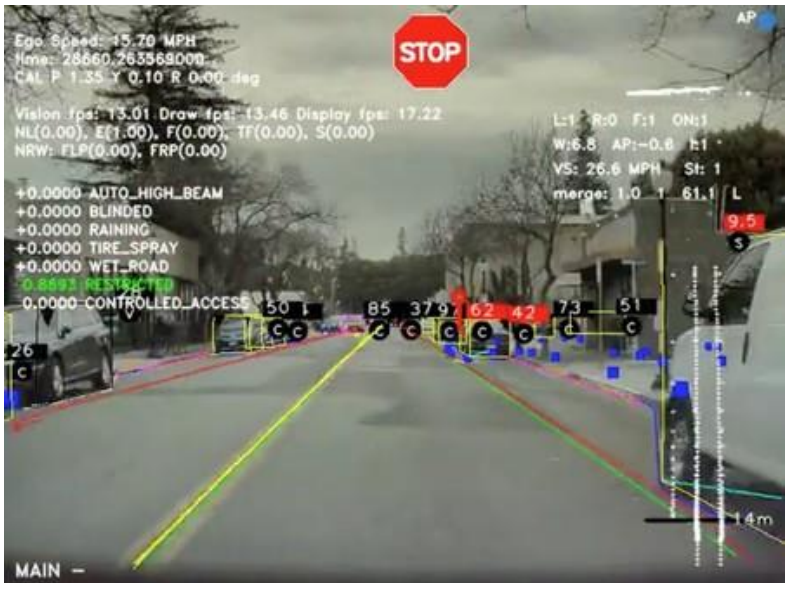


Figura 3.Procesamiento de señales captados por Tesla Autopilot.

2 HARDWARE

Hardware: las partes de un ordenador que pueden ser pateadas.
- Jeff Pesis-

2.1 Raspberry Pi

La Raspberry Pi 3 es la tercera generación de ordenadores de placa reducida y de bajo coste, el modelo que se va a usar en este trabajo es una renovación de la placa lanzada en febrero de 2016 (Raspberry Pi model B).

La RPI model B+ fue lanzada en marzo de 2018 y entre sus características podemos destacar que incluye un chip integrado Broadcom BCM2837, que contiene un procesador ARMv8 de 4 núcleos que tiene una frecuencia de funcionamiento de 1,4 GHz, un procesador gráfico VideoCore IV con la posibilidad de reproducir vídeo en 1080p y hasta gráficos 3D. Presenta además una salida de vídeo y audio a través de un conector HDMI, que hacen posible la conexión tanto a pantallas de televisión como a monitores que cuenten con dicha conexión. En cuanto a vídeo, cuenta con una salida de video compuesto y una salida de audio con conexión Jack de 3.5 mm. Incorpora una interfaz para su cámara oficial (CSI) y otra interfaz para display (DSI) Como se puede observar en la figura 4.

Posee una conexión Etherner 10/100/1000 Mbps vía hub USB limitado a 300 Mbit/s. Como diferencia respecto a su modelo anterios incorpora Wifi 802.11ac de doble banda y Bluetooth 4.2. La placa incluye además 4 puertos USB con limitación de corriente. Además, cuenta con 40 pines GPIO. En su parte inferior tiene un lector de tarjetas microSD, de esta forma se minimiza el espacio necesario para tener un ordenador en un volumen minimo de espacio, dándonos la capacidad de poder instalar este sistema en pequeños vehículos, robots, etc…

Figura 4.Placa Raspberry Pi 3 model B+

2.1.1 Especificaciones técnicas de modelos de Raspberry Pi anteriores

	SoC	CPU	GPU	RAM	USB	V/A	Boot	Red	Alimentación	Tamaño	Fecha	Precio
 Model A	Broadcom BCM2835	700MHz ARM1176JZF-S	VideoCore IV	256MB	1	RCA Jack HDMI	SD	No	300mA 1,5w / 5v MicroUSB GPIO	85,6 x 53,98 mm	04/12	25$
 Model A+	Broadcom BCM2835	700MHz ARM1176JZF-S	VideoCore IV	256MB	1	Jack HDMI	uSD	No	400mA 2w / 5v MicroUSB GPIO	65 x 56 mm	11/14	20$
 3 Model A+	Broadcom BCM2837B0	1,4GHz QUAD ARM Cortex-A53	VideoCore IV	512MB	1	Jack HDMI	uSD	Dual-band WiFi, BT	2,5A 12,5w / 5v MicroUSB GPIO	65 x 56 mm	11/18	20$
 Model B	Broadcom BCM2835	700MHz ARM1176JZF-S	VideoCore IV	512MB	2	RCA Jack HDMI	SD	ETH 10/100	700mA 3,5w / 5v MicroUSB GPIO	85,6 x 53,98 mm	04/12	35$
 Model B+	Broadcom BCM2835	700MHz ARM1176JZF-S	VideoCore IV	512MB	4	Jack HDMI	uSD	ETH 10/100	500mA 2,5w / 5v MicroUSB GPIO	85 x 56 mm	07/14	35$
 2 Model B	Broadcom BCM2836	900MHz QUAD ARM Cortex-A7	VideoCore IV	1GB	4	Jack HDMI	uSD	ETH 10/100	800mA 4w / 5v MicroUSB GPIO	85 x 56 mm	02/15	35$
 3 Model B	Broadcom BCM2837	1,2GHz QUAD ARM Cortex-A53	VideoCore IV	1GB	4	Jack HDMI	uSD	ETH 10/100 WiFi, BT	2,5A 12,5w / 5v MicroUSB GPIO	85 x 56 mm	02/16	35$
 3 Model B+	Broadcom BCM2837B0	1,4GHz QUAD ARM Cortex-A53	VideoCore IV	1GB	4	Jack HDMI	uSD	ETH 10/100/300 (USB) Dual-band WiFi BT	2,5A 12,5w / 5v MicroUSB GPIO PoE (HAT)	85 x 56 mm	03/18	35$
 4 Model B	Broadcom BCM2711	1,5GHz QUAD ARM Cortex-A72	VideoCore IV	1, 2 o 4GB	2 (2.0) 2 (3.0)	Jack 2 micro HDMI	uSD	ETH 1000 Dual-band WiFi BT	2,5A 12,5w / 5v USB-C GPIO PoE (HAT)	85 x 56 mm	06/19	35$
 Zero	Broadcom BCM2835	1GHz ARM1176JZF-S	VideoCore IV	512MB	1 Micro	Mini HDMI	uSD	No	160mA 0,8w / 5v MicroUSB GPIO	65 x 30 mm	11/15	5$
 Zero W	Broadcom BCM2835	1GHz ARM1176JZF-S	VideoCore IV	512MB	1 Micro	Mini HDMI	uSD	Wifi, BT	160mA 0,8w / 5v MicroUSB GPIO	65 x 30 mm	02/17	10$

Figura 5.Comparativa de modelos de Raspberry Pi.

3.2 Cámara

Como cámara para la realización se ha utilizado la Raspberry Pi Camera Module V2.
Se ha elegido esta cámara debido a que es la oficial distribuida por la propia empresa que fabrica la Raspberry Pi. Entre sus cualidades hay que destacar que presenta buena compatibilidad ya que aprovecha el puerto CSI de la placa. Es muy compacta, mide aproximadamente 25 mm x 24 mm x 9 mm y pesa 3 g, lo que la hace perfecta para implementarla en proyectos para aplicaciones móviles o en pequeños vehículos donde el peso es un factor muy importante para tener en cuenta. La cámara reemplazó al anterior modelo de esta denominado Raspberry Pi Camera Module V1 en abril de 2016 y está equipada con un sensor Sony Exmor IMX219 de 8 megapíxeles que permite tomar vídeos en alta definición, así como fotografías lo cual significa una gran mejora con su respecto a la versión anterior.

Se puede acceder a la cámara a través de las API MMAL y V4L, y hay numerosas bibliotecas de terceros (third-party libraries) creadas para ello, entre ellas se encuentra la librería de Pyhon "Picamera".

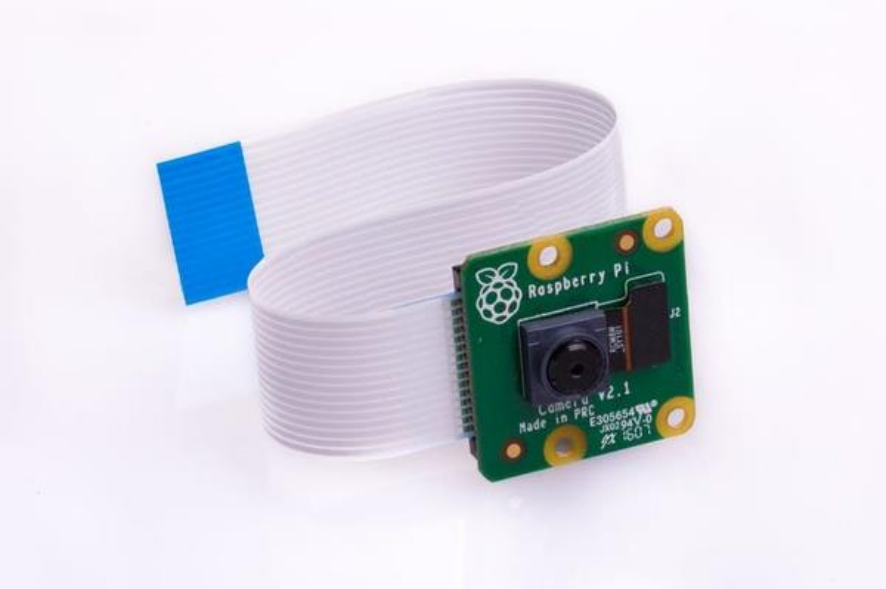


Figura 6. Cámara Raspberry Pi Camera Module V2.

2.3 Tarjeta microSD

Las tarjetas microSD, T-flash o transflash hacen referencia al formato más pequeño de tarjeta de memoria, llegando a superar a las tarjetas miniSD.

Fueron desarrolladas por SanDisk, y en el año 2005 fueron incorporadas por la Asociación de Tarjetas SD con el nombre de microSD.

Sus dimensiones son de 15 mm x 11 mm x 1 mm y posee un área de 165 mm^2. Estas dimensiones la convierten en la tarjeta de menor tamaño con tres veces y media menores dimensiones que la miniSD y hasta un décimo de la parte del volumen que una tarjeta SD convencional.

Su uso se concentra en aplicaciones donde el tamaño es un factor muy importante como es nuestro caso. La tarjeta microSD elegida para el desarrollo de este proyecto ha sido la SanDisk microSDHC UHS-I de 32 Gb clase 10, que permite una velocidad de lectura de hasta 80 Mb/s. Esta velocidad ha sido el principal factor por el que se ha decidido utilizar esta tarjeta, ya que para el proyecto que queremos desarrollar necesitamos que nuestra placa sea capaz de acceder a la memoria de una forma ágil y rápida y esta tarjeta reúne las condiciones necesarias para ello.

Figura 7. Tarjeta de memoria microSD.

2.4 Alimentación

Para la alimentación de nuestra placa se van a utilizar dos opciones. La primera consiste en una fuente de alimentación clásica conectada a la corriente. Las características principales de esta fuente es que proporciona 5 V a una intensidad de 3 A.

Figura 8. Fuente de alimentación.

Esta es la opción que se ha elegido para poder trabajar mientras se programa la Raspberry Pi ya que es la más cómoda. A la hora de una posible implementación en un vehículo, un robot, etc, … se utilizará la segunda opción, se trata de una batería externa con las mismas características que la fuente de alimentación. Esta opción nos permitirá implantar nuestro sistema en un equipo que esté destinado a moverse por lo que poder prescindir de los cables será un punto muy importante.

2.5 Conexiones

2.5.1 CSI

Se trata de un estándar más conocido con el nombre de Interfaz Serie para Cámaras (Camera Serial Interface).

Permite comunicar la Raspberry con la cámara de manera simple, utilizando la conexión y el cable que se muestra en la siguiente figura:

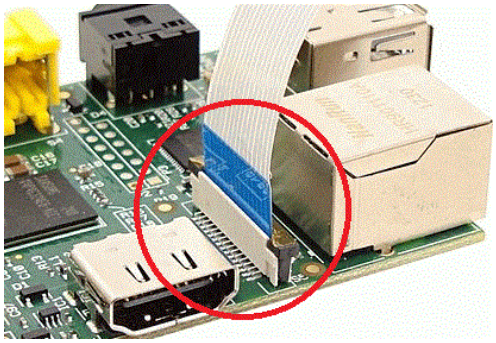


Figura 9. CSI de la Raspberry Pi.

2.5.2 SSH

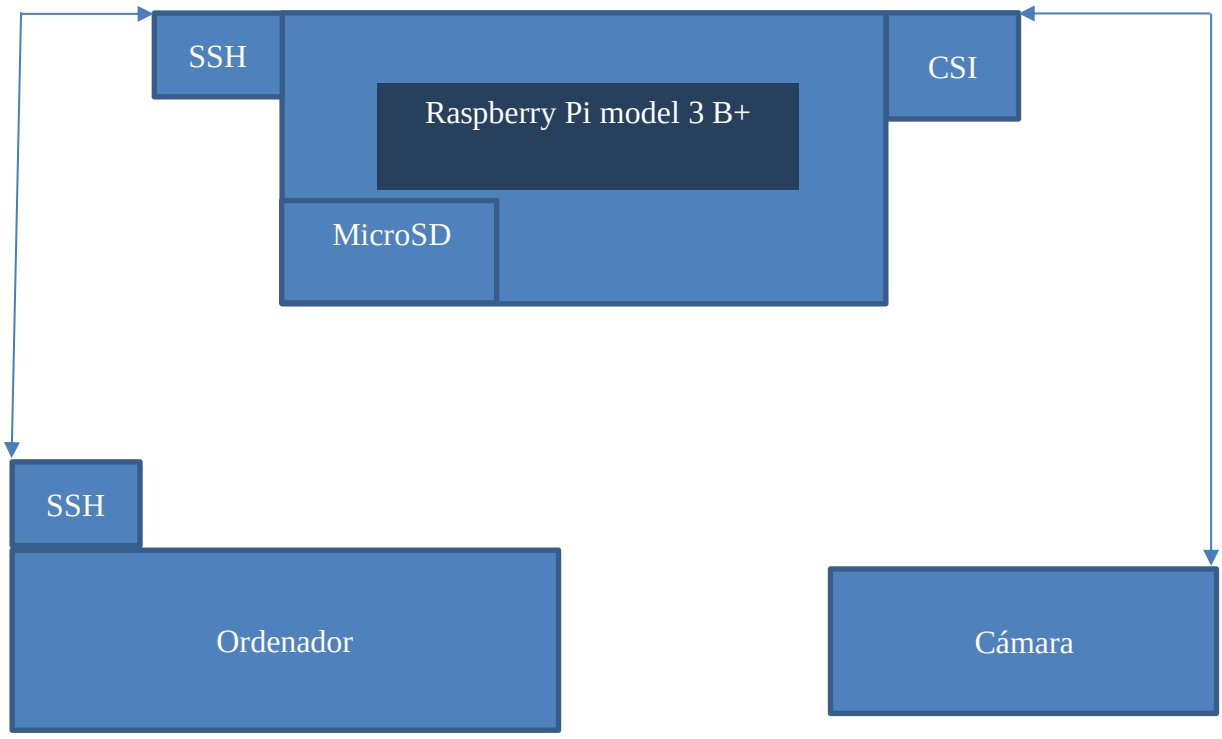
Existn diversos procedimientos para conectarse de manera remota a una Raspberry Pi, en este proyecto trabajaremos con SSH. SSH son las siglas de Secure Shell que es el nombre tanto del protocolo como del programa que lo implementa y cuya función principal es dar acceso remoto a

un servidor por medio de un canal seguro en el que toda la información se encuentra cifrada. Aparte de la conexión también permite transmitir datos de forma segura.

El transporte se realiza mediante TCP (Transsmision Control Protocol) es uno de los protocolos fundamentales de Internet. Dicho protocolo establece la conexión mediante la negociación en tres pasos. Se considera que este protocolo es muy fiable y robusto ya que existen una serie de mecanismos (uso de números de secuencia para ordenar segmentos, detectar paquetes duplicados, …) que actúan como protección.

Aunque por defecto la Raspberry Pi lleva instalada un servicio SSH que como hemos mencionado previamente, se utiliza para establecer conexiones remotas encriptadas. Por razones de seguridad el servicio no se encuentra activo por defecto. En el apartado 5.1.4 de este proyecto se describe el método que hay que seguir para realizar la activación de este servicio.

2.6 Esquema de conexión



En el esquema anterior se puede observar un esquema de las conexiones internas de la propia placa y los métodos que utiliza la Raspberry Pi para conectarse con el resto de los dispositivos que se conectan de forma externa.

3 SOFTWARE

4.1 Sistema Operativo

3.1.1 RASPBIAN

Como Sistema operativo se va a usar Raspbian. Raspbian es una distribución del sistema operativo GNU/Linux basado en Debian, y por lo tanto libre y gratuito para la Raspberry Pi, orientado a la enseñanza de la informática. Además, es el aconsejado por la empresa que fabrica las placas debido a que está optimizado para el hardware de la Raspberry Pi.

La versión actual del software es conocida con el nombre de Raspbian Buster, ha sido publicada el 26 de septiembre de 2019 y cuenta con tres versiones distintas:

- RASPBIAN BUSTER with desktop and recommended software
- RASPBIAN BUSTER with desktop
- RESPBIAN BUSTER lite

Como el propio nombre de las versiones indica, la primera versión incluye el escritorio (interfaz gráfica) y las aplicaciones recomendadas ya preinstaladas. La segunda versión incluye simplemente el escritorio y la tercera es una versión especial que no contiene ningún programa preinstalado y no dispone de interfaz gráfica por lo que todas las operaciones se realizan a través de la línea de comandos. Esta última versión se asemeja a la consola destinada a ejecutar comandos que podemos encontrar en cualquier terminal. En la tabla 1 se encuentra más información acerca de las versiones mencionadas anteriormente:

Tabla 1. Comparación de las versiones de Raspbian

	Aplicaciones recomendadas	Escritorio (Interfaz gráfica)	Tamaño de memoria que ocupa
RASPBIAN BUSTER with desktop and recommended software	SI	SI	2541 Mb
RASPBIAN BUSTER with desktop	NO	SI	1123 Mb
RASPBIAN BUSTER lite	NO	NO	435 Mb

Es notable la diferencia de tamaño entre las distintas opciones de instalación del sistema operativo. En este proyecto en principio se iba a utilizar la versión lite principalmente por la diferencia de tamaño con respecto al resto de versiones. Esta diferencia permite a nuestra Raspberry Pi disponer de más memoria y poder ser usada de una forma óptima. Pero durante el desarrollo de este proyecto ha habido que cambiar la versión y usar la versión más grande (RASPBIAN BUSTER with desktop and recommended software) ya que debido a un error salió a la luz que la mayoría de los

programas gtk requieren una versión completa con escritorio para poder ser ejecutados. Esto fue un punto de inflexión importante en el proyecto.

3.2 Lenguaje de programación

3.2.1 Python

Python es un lenguaje de programación de alto nivel orientado a objetos con semántica dinámica integradas principalmente para la web y el desarrollo de aplicaciones. Respecto al nivel de dificultad, nos encontramos ante un lenguaje relativamente simple y fácil de aprender ya que requiere una sintaxis que se centra en la legibilidad.
El lenguaje de programación Pyhton comenzó su implementación en 1989 cuando Guido Van Rossum que trabaja en el CWI (centro de investigación holandés de carácter oficial) comenzó a trabajar en el proyecto como un pasatiempo, dándole continuidad al lenguaje de programación ABC del que había formado parte del equipo de desarrollo.
Los desarrolladores pueden leer y traducir código Python de una forma mucho más sencilla que otros lenguajes. Esto resulta bastante ventajoso ya que se reduce el coste de mantenimiento y desarrollo del programa porque permite que los equipos trabajen en colaboración sin un lenguaje con diferencias significativas y sin barreras de experiencia haciendo que cualquiera sea capaz de entenderlo independientemente de su nivel como programador.
Otro de los beneficios de este lenguaje de programación es que tanto la biblioteca estándar como el intérprete se encuentran disponibles de forma gratuita, tanto en binario como en código fuente, por tanto, es una opción atractiva para los desarrolladores que no quieren preocuparse por pagar altos costos de desarrollo.
Además de todo esto, Python admite el uso de módulos y paquetes, lo que significa que los programas pueden ser diseñados en un estilo modular y el código puede reutilizarse en otros proyectos ya que es muy sencillo importar y exportar dichos módulos entre proyectos.

Figura 10.Logotipo de Pyhton.

3.2.2 Ventajas de utilizar Python

Comaprado a otros lenguajes de programacion como C o C++, Python es más lento, pero una de las características más importantes de este lenguaje y de gran influencia en este proyecto es que Python puede ser fácilmente extendido con C/C++.
Esta característica nos ayuda a escribir códigos computacionales intensivos en C/C++ y crear un contenedor de Python (wrapper) como módulos de Python. Esto nos aporta principalmente dos ventajas:
La primera, nuestro código es tan rápido como el código C/C++ original ya que realmente es código C/C++ auténtico que funciona en segundo plano y segundo, es muy fácil desarrollar código en Python. Así es como funciona una de las principales librerías de este trabajo (OpenCV) dónde actúa como contenedor de Python en torno a la implementación original de C/C++.

3.3 Librerías

Python contiene varias librerías que permiten hacer transformaciones y procesamiento de imágenes. En el presente las imágenes se consideran un recurso esencial del que extraer datos ya que se puede obtener información muy útil de ellas. A continuación se presentan las distintas librerías que han sido utilizadas para la ejecución de este proyecto.

3.3.1 OpenCV

OpenCV es una biblioteca de visión artificial desarrollada por la empresa Intel en un principio. Se centra en procesamiento de imagen en tiempo real. Es de código abierto y multiplataforma ya que puede usarse en Mac OSX, Windows y Linux conteniendo más de 500 funciones que abarcan una gran gama de áreas de proceso de visión: reconocimiento facial, calibración de cámaras, visión robótica, …. Su primera aparición en versión alfa tuvo lugar en 1999 y ha tenido una gran evolución usándose actualmente en una gran cantidad de aplicaciones.

Desde aplicaciones que incluyen sistemas de seguridad con detección de movimiento, sistemas de mapeado hasta aplicaciones en las que se realiza detección de objetos. En general engloba prácticamente todos los campos de la visión artificial y ha podido usarse en tantas aplicaciones debido a que se publicó bajo licencia BSD (licencia de software otorgada para los sistemas (Berkely Software Distribution), concretamente de tipo “modificada en tres cláusulas”, que permite que sea usada libremente con propósitos comerciales y de investigación.

Un ejemplo de programa de OpenCV se puede ver a continuación:
El problema al que se intenta dar solución es encontrar características similares de una imagen en otra. Esto permite encontrar objetos que podemos tener en dicha imagen mediante reconocimiento de puntos similares (denominados puntos clave).

En la siguiente figura se muestra un ejemplo del desarrollo de este código. Se puede ver como a partir de la imagen de la izquierda (botella de agua) puede detectarse en la imagen de la derecha un objeto similar marcando incluso la zona de coincidencia mediante un marco blanco e indicando los puntos de correspondencia entre las dos imágenes:

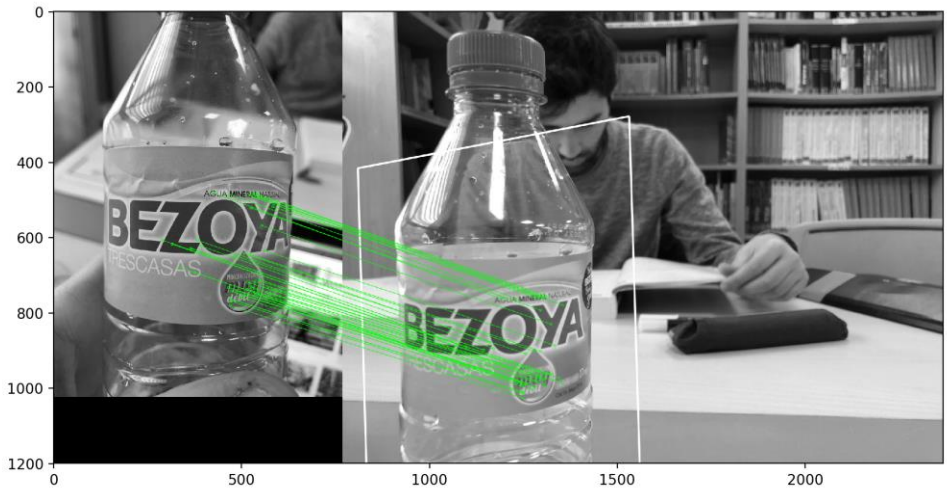


Figura 11. Programa de reconocimiento entre imágenes.

3.3.2 OpenCV – Python API

OpenCV – Python es una librería de enlaces diseñada para resolver problemas de visión por computadora. Se trata de un contenedor de Python para la implementación original de OpenCV C++. Como se ha comentado anteriormente, esto nos aporta dos ventajas: primero, el código es igual de rápido que el código C/C++ original y segundo, es más fácil escribir código en Python que en C/C++ debido a su simplicidad.

OpenCV – Python hace uso de una serie de dependencias que son librería en las que se apoya para poder realizar distintas funciones. En los siguientes apartados veremos en qué consisten estas dependencias.

3.3.3 Dependencias de OpenCV

3.3.3.1 Picamera

Picamera es una libreria que nos permite utilizar la cámara de la Raspberry Pi (Raspberry Pi Camera Module V2) a través de una interfaz de Python. Esta librería nos permitirá acceder a dicho módulo. Si el proyecto fuese realizado mediante una cámara USB común, no es necesario utilizar este módulo. Procesar imágenes está conformado por operaciones como rotaciones, giros, transformaciones y demás

3.3.3.2 NumPy

NumPy es una libreria open source altamente optimizada para operaciones numéricas, se trata de una extensión de Python la cual agrega un gran soporte para vectores y matrices. Aporta una sintaxis de estilo MatLab dónde todas las estructuras de matriz de OpenCV que se obtienen de las imágenes se convierten en matrices de NumPy que son las que se utilizan para realizar las operaciones numéricas de una forma eficiente.

Independientemente de las operaciones que se puedan realizar en NumPy, pueden ser combinadas con la librería OpenCV de ta forma que conseguimos un aumento en el número de herramientas disponibles a la hora de codificar. Fue creada por Travis Oliphant en 2005 incorporando características de Numarray con algunas modificaciones.
Los principales puntos por los que se ha decidido utilizar esta librería son los siguientes:

- Posee una potente estructura de datos
- Implementa matrices y matrices multidimensionales
- Las estructuras garantizan cálculos eficientes con matrices

Veamos un ejemplo de la eficiencia de NumPy:
Vamos a comparar la lista de Python con la librería Numpy, primero evaluaremos la memoria asignada mediante el siguiente programa:

```
import sys
S= range(1000)
print(sys.getsizeof(5)*len(S))

D= np.arange(1000)
print(D.size*D.itemsize)
```

Se asigna memoria a dos matrices de igual tamaño y se imprime el tamaño (la memoria) que ocupa cada una. Comparando los resultados vemos que la lista de Python ha asignado una memoria de 28000 mientras que NumPy ha asignado 8000, podemos concluir que es preferente utilizar las matrices creadas con NumPy con respecto a la economía de la memoria.
Ahora evaluamos la rapidez mediante el siguiente código:

```
import time

SIZE = 1000000

L1= range(SIZE)
L2= range(SIZE)
A1= np.arange(SIZE)
A2=np.arange(SIZE)

start= time.time()
result=[(x,y) for x,y in zip(L1,L2)]
print((time.time()-start)*1000)

start=time.time()
result= A1+A2
print((time.time()-start)*1000)
```

Obtenemos que con las listas de Python la suma tarda en ejecutarse 207 ms mientras que la operación realizada por Numpy toma solamente 51 ms en ejecutarse. Además de esto, podemos notar que para el primer caso tuvimos que utilizar un bucle “for” mientras que con NumPy podemos simplemente sumar, esto deja en evidencia la simplicidad que nos aporta esta librería.

3.3.3.3 SciPy

SciPy es una librería open source y grauita utilizada para computación científica y técnica que se compone de herramientas y algoritmos matemáticos. Al igual que NumPy fue creada a partir de la colección original de módulos de Python de Travis Oliphant.
Contiene módulos para optimización, algebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de imágenes y señales (parte que nos interesa), ODE solvers, y otras tareas comunes para ciencia e igeniería.
SciPy utiliza los vectores multidimensionales proporcionados por Numpy como estructura de datos básica.

3.3.3.4 Matplotlib

Matplotlib es una librería open source y gratuita diseñada para el trazado de gráficas utilizando datos pertenecientes a listas o arrays en lenguaje Python y su extensión matemáca NumPy. Proporciona una API diseñada para recordar a MatLab (de ahí su nombre).

Esta librería es de gran utilidad y tiene muchas aplicaciones en el campo de vision artificial ya que se usa por ejemplo a la hora de realizar reconocimiento facial para crear los marcos que indican dónde se encuentran los rostros en las imágenes.

Figura 12. Ejemplo de detección facial.

3.4 Herramientas auxiliaries

3.4.1 PIP

Pip es un administrador de paquetes, un sistema de gestión utilizado para administrar e instalar paquetes y módulos escritos en Python.

Un gran punto a favor de Pip es la sencillez de su interfaz de la línea de comandos que nos permite instalar y desinstalar paquetes y módulos de software Python con una sencilla orden:

Para instalar un paquete:

```
pip install nombre-del-paquete
```

Para desinstalar paquetes:

```
pip uninstall nombre-del-paquete
```

3.4.2 CMake

CMake es un sistema de código abierto extensible que gestiona el proceso de compilación en un sistema operativo y de manera independiente del compilador. A diferencia de muchos sistemas multiplataforma, CMake está diseñado para usarse junto con el entorno de construcción nativo. Los archivos de configuración simples ubicados en cada directorio de origen (llamados archivos CMakeLists.txt) se usan para generar archivos de compilación estándar (por ejemplo, archivos MAKE en Unix y proyectos / espacios de trabajo en Windows MSVC) que se usan de la manera habitual. CMake puede generar un entorno de compilación nativo que compilará el código fuente, creará bibliotecas, generará envoltorios y creará ejecutables en combinaciones arbitrarias.

4 INSTALACIÓN

El software es un gas: se expande hasta llenar su ordenador.
- Nathan Myhrvold -

5.1 Instalación y preparación del Sistema Operativo

4.1.1 Instalación de Raspbian

Podemos encontrar el software en la página oficial de Raspberry Pi, en la sección de descargas. Allí tendremos acceso a todas las versiones del sistema operativo Raspbian que han existido por si queremos trabajar con una versión más antigua por conveniencia debido a ventajas como la compatibilidad.
Una vez hemos accedido a la sección de descargas, nos encontramos con dos opciones de instalación del sistema operativo. Podemos instalar el SO utilizando "NOOBS" o bien, instalar Raspbian manualmente mediante imagen.

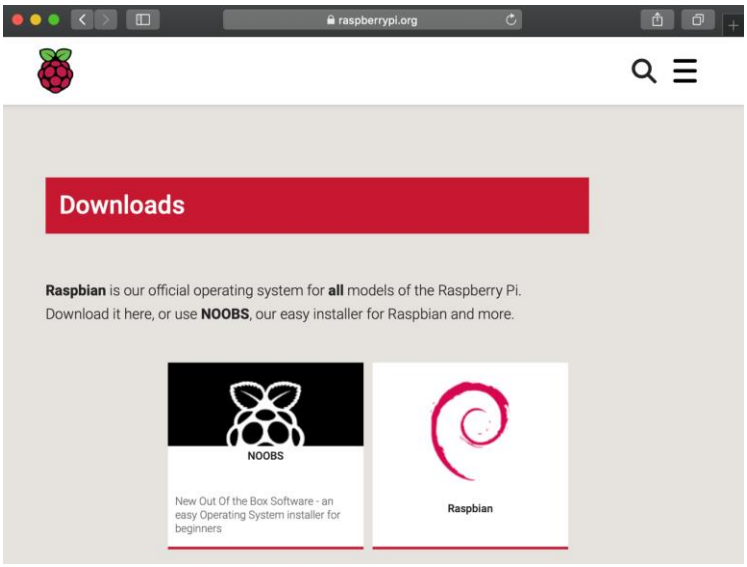


Figura 13. Opciones de descarga del SO Raspbian.

NOOBS son las siglas para "New Out Of the Box Software". Se trata de un instalador de sistemas operativos simplificado, es una herramienta facilitada por Raspberry para una fácil instalación del SO Raspbian en la placa. Contiene Raspbian y LibreELEC, además contiene también una selección de sistemas operativos que serán descargados de internet por el propio NOOBS y posteriormente se instalarán.
El método de instalación que utilizaremos será el segundo, instalaremos directamente el SO mediante la imagen de Raspbian que descargaremos de la página de descargas de Raspberry. Una

vez descargada grabaremos esta imagen en la tarjeta microSD utilizando un programa cualquiera como Etcher.
Una vez grabada la imagen en la tarjeta microSD, insertamos la tarjeta en la ranura microSD de la Raspberry Pi y la conectamos a la corriente, veremos como empieza el proceso de instalación del SO y una vez terminado podremos ingresar en el sistema con las siguientes credenciales:

Usuario: pi
Contraseña: raspberry

4.1.2 Balena Etcher

Etcher es un software de descarga gratuita que permite grabar imágenes de SO en distintos tipos de dispositivos de almacenamiento (como tarjetas microSD, USB, …). Tiene una interfaz muy simple en la que elegimos la imagen que queremos grabar y pulsamos "Flash!" (así se realizará la grabación de la imagen directamente en el dispositivo elegido).

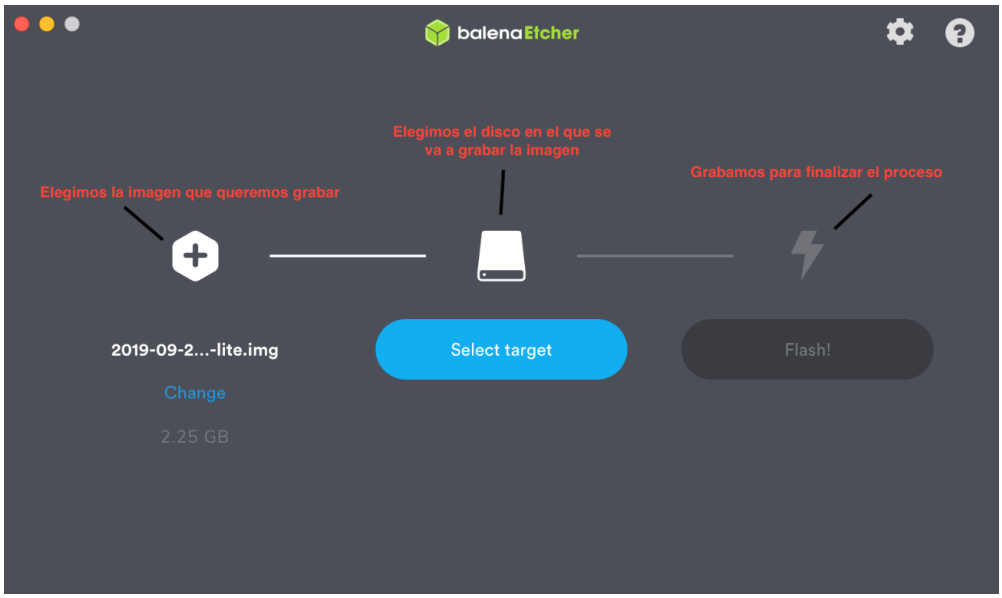


Figura 14. Interfaz del software BalenaEtcher.

4.1.3 Conexión de la Raspberry Pi a la red Wifi

Para conectar la Raspberry Pi a la red Wifi necesitaremos conocer tanto el nombre como la contraseña de la red. Conociendo estos datos, vamos a editar el fichero /etc/wpa_supplicant/wpa_supplicant.conf empleando el siguiente comando:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

En el fichero añadimos lo siguiente cambiando los datos por los correspondientes a nuestra red:

```
network={
        ssid="nombre-red-wifi"
        psk="password-red-wifi"
        key_mgmt=WPA-PSK
}
```

Una vez realizado este paso, guardamos el fichero y reiniciamos la Raspberry Pi. En cuanto se inicie el sistema dispondremos de acceso a internet.

4.1.4 Conexión vía SSH

Aunque por defecto la Raspberry Pi lleva instalada un servicio SSH que como hemos mencionado previamente se utiliza para establecer conexiones remotas encriptadas. Por razones de seguridad el servicio no se encuentra activado por defecto.
El primer paso, por tanto, será conectar la Raspberry Pi a internet como se ha realizado en el apartado anterior. Tras esto y por motivos de seguridad, cambiaremos la contraseña que por defecto es "raspberry" para evitar que un equipo externo pueda acceder al servicio SSH de nuestra Raspberry Pi.
Tras esto habilitaremos por terminal el servicio SSH, para ello necesitamos conectar la Raspberry a una pantalla y un teclado. Una vez realizada la activación, no será necesario el uso de estos periféricos.
Al acceder al sistema de la Raspberry entraremos directamente a la línea de comandos donde escribiremos lo siguiente:

```
sudo raspi-config
```

Se abrirá el menú de la herramienta de configuración donde seleccionaremos "Opciones Avanzadas" y después "SSH":

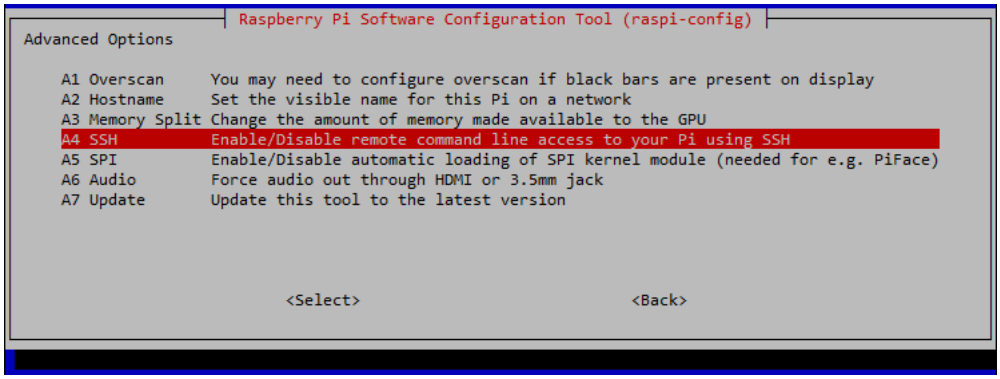


Figura 15. Menú de opciones avanzadas de la Raspberry Pi.

Tras activarlo ("Enable") habrá que confirmarlo antes de cerrar los ajustes.
Ya podemos proceder a la conexión en sí. MacOSX y Ubuntu soportan automáticamente la conexión con el protocolo SSH gracias a la implementación estándar del software libre OpenSSH. Simplemente con tener el equipo y la Raspberry Pi conectadas a la misma red Wifi es suficiente. Lo único que necesitamos conocer es la dirección IP para poder realizar la conexión.
Para poder acceder a esta información tecleamos en la línea de comandos:

```
Hostname -I
```

Ahora desde el equipo con el que queremos conectarnos a la Raspberry Pi escribimos la siguiente orden:

```
ssh pi@direccion-IP
```

Una vez realizados todos estos pasos ya dispondremos de acceso a la Raspberry Pi de forma remota facilitando mucho el trabajo ya que no necesitaremos disponer de periféricos conectados a la

Raspberry Pi, simplemente con conectar la alimentación y estar conectada a la misma red que el equipo con el que trabajaremos será suficiente.

4.1.5 Instalación y configuración de la cámara

Para poder utilizar la cámara hay que hacer los siguiente:
Conectar la cámara al puerto CSI de la Raspberry Pi tal como se muestra en la imagen:

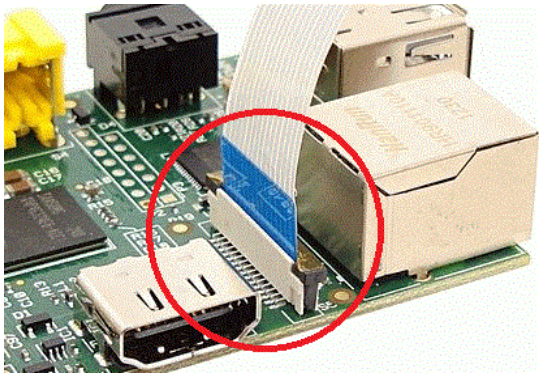


Figura 16. Conexión de la cámara mediante puerto CSI.

Una vez conectada, lo siguiente que debemos hacer es acceder al menú de la herramienta de configuración donde podremos habilitar la cámara.
Ejecutando el siguiente comando accedemos al menú:

```
sudo raspi-config
```

Aquí activaremos la cámara (“Enable”).

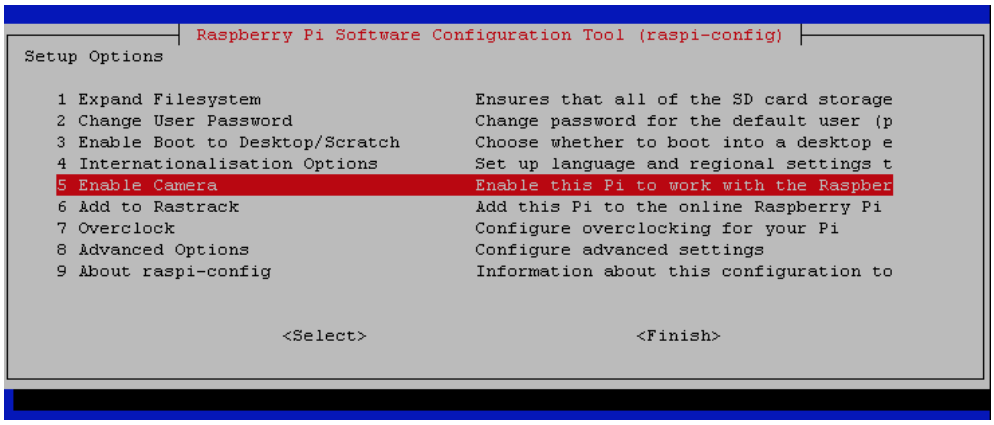


Figura 17. Menú para la activación de la cámara mediante puerto CSI.

Tras esto, reiniciamos la Raspberry Pi y una vez se haya vuelto a encender ya disponemos de acceso a la cámara.

4.1.6 Configurar hora en la Raspberry Pi

Primero debemos ejecutar el comando para acceder al menú:

```
sudo raspi-config
```

Ahora seleccionamos la opción “Internationalisation Options” donde cambiaremos el área geográfica (Europa) y la ciudad o región (Madrid) reiniciando la Raspberry tras realizar estos pasos.

Una vez realizado esto, ejecutando el siguiente comando podremos cambiar la fecha del sistema de la Raspberry Pi manualmente:

```
sudo date MMDDhhmm
```

- MM corresponde al mes
- DD corresponde al día
- hh corresponde a la hora
- mm corresponde a los minutos

4.2 Instalación de OpenCV

Una vez instalado el sistema operativo, con la Raspberry instalada y conexión a internet, procedemos a la instalación de la principal librería OpenCV.

Primero iniciaremos la Raspberry Pi, accediendo con nuestras credenciales y accederemos a ella mediante un equipo remoto vía SSH y accedemos al menú de configuración tal como hemos hecho con anterioridad mediante el comando:

```
sudo raspi-config
```

Seleccionamos la opción de “Advanced Options” donde procederemos a expandir el sistema de archivos mediante la opción “Expand Filesystem” como se muestra en la figura:

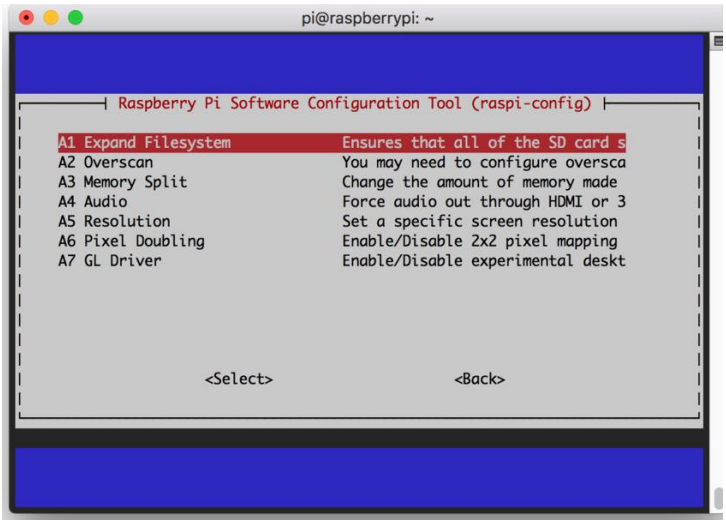


Figura 18. Menú de opciones avanzadas para expandir el sistema de archivos.

Esta acción nos permite acceder a todo el espacio libre de la memoria de nuestra tarjeta microSD ya que de serie la cantidad de memoria del sistema Raspberry Pi a la que podemos acceder está limitada. Tras la activación es necesario reiniciar el sistema mediante el comando:

```
sudo reboot
```

Ahora actualizamos el sistema:

```
sudo apt-get update && sudo apt-get upgrade
```

Una vez actualizado, instalamos una serie de herramientas de desarrollador, incluyendo CMake:

```
sudo apt-get install build-essential cmake unzip pkg-config
```

Instalamos una selección de librerías de video e imagen que son indispensables para poder trabajar con archivos de imagen y video:

```
sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-
     dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
```

Instalamos nuestro GUI (Interfaz gráfica de usuario) que tiene el nombre de GTK:

```
sudo apt-get install libgtk-3-dev
sudo apt-get install libcanberra-gtk*
```

Unos paquetes que contienen optimizaciones numéricas para OpenCV:

```
sudo apt-get install libatlas-base-dev gfortran
```

Y por ultimo las cabeceras de desarrollo de Python 3:

```
sudo apt-get install python3-dev
```

Estos son todos los requisitos que necesitamos para la instalación. Procedemos a la descarga de OpenCV. Primero navegamos a nuestra carpeta raíz:

```
cd ~
```

Y allí descargaremos opencv y opencv_contrib. La primera contiene los módulos de la librería OpenCV en sí mientras que la segunda contiene módulos y funciones extra que pueden sernos de utilidad ya que son utilizados con bastante frecuencia:

```
wget -O opencv.zip
     https://github.com/opencv/opencv/archive/4.0.0.zip
wget -O opencv_contrib.zip
     https://github.com/opencv/opencv_contrib/archive/4.0.0.zip
```

Extraemos los archivos:

```
unzip opencv.zip
unzip opencv_contrib.zip
```

Y les cambiamos el nombre para que nos sea más sencillo trabajar con ellos:

```
mv opencv-4.0.0 opencv
```

```
mv opencv_contrib-4.0.0 opencv_contrib
```

Instalamos PIP, un gestor de paquetes de Python:

```
wget https://bootstrap.pypa.io/get-pip.py
sudo python3 get-pip.py
```

Por medio de PIP instalamos el paquete de Python NumPy:

```
pip install numpy
```

Accedemos a nuestro entorno virtual, en el que instalaremos OpenCV. Ahora accedemos al repositoro en el que se encuentra la librería OpenCV:

```
cd ~/opencv
```

Aquí creamos y accedemos a una carpeta que se llame “build”:

```
mkdir build
cd build
```

Ejecutamos CMake para configurar la bild de OpenCV alterando los siguientes parámetros simplemente introduciendo las líneas de código que figuran a continuación:

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
-D ENABLE_NEON=ON \
-D ENABLE_VFPV3=ON \
-D BUILD_TESTS=OFF \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D BUILD_EXAMPLES=OFF ..
```

Tras esto, estamos listos para compilar OpenCV:

```
make -j4
```

“-j4” indica que vamos a utilizar los 4 núcleos de los que dispone el procesador de la Raspberry Pi para realizar la compilación puesto que es un proceso bastante costoso. Una vez hayamos comprobado que se ha compilado sin errores podemos proceder finalmente a la instalación de OpenCV en sí:

```
sudo make install
sudo ldconfig
```

Un paso crítico que hay que realizar una vez finalizada la instalación es crear un vinculo simbólico desde la instalación de OpenCV en el sistema “site-packages” al entorno virtual con el que trabajaremos llamado “tfg”:

```
cd ~/.virtualenvs/tfg/lib/python3.5/site-packages/
```

```
ln -s /usr/local/python/cv2/python-3.5/cv2.cpython-35m-arm-
linux-gnueabihf.so cv2.so
cd ~
```

Si no realizamos el paso anterior no seremos capaces de importar la librería OpenCV a nuestros scripts.
Una vez realizado todo este proceso, probaremos a entrar en el entorno virtual y comprobaremos que la versión de OpenCV que se encuentra instalada es la correcta:

```
cd ~
workon tfg
Python
>>>> import cv2
>>>> cv2.__version__
'4.1.1'
>>>>
```

6 INTRODUCCIÓN A LA LIBRERÍA DE VISIÓN OPENCV

En este apartado se van a desarrollar las funciones necesarias consideradas como básicas para realizar los programas que se van a desarrollar en el trabajo. Aunque no sea el objetivo principal, he encontrado que las siguientes funciones me han servido de gran utilidad a la hora de realizar procesamiento de imágenes.

Hay que diferenciar entre los casos en los que se utiliza una cámara genérica conectada mediante USB y casos en los que se utiliza el Raspberry Pi Camera module ya que en función de esto hay que realizar ligeras modificaciones en el código.

6.1 Librería OpenCV

6.1.1 Lectura de una imagen

Utilizamos la función cv2.imread() para realizar lectura de imágenes que preciamente se han tomado de la cámara. Estas imágenes deberán estar contenidas en el directorio de trabajo que se esté utilizando o bien habrá que pasarle a la función la ruta en la que se encuentra la imagen con la que se va a trabajar.

```
import numpy as np
import cv2
```

```
img = cv2.imread('imagen.jpg')
```

Esta función puede recibir un segundo parámetro que nos permite declarar la forma en la que la imagen será leída.

`Cv2.IMREAD_COLOR:` carga la imagen a color, es el flag por defecto.

`Cv2.IMREAD_GRAYSCALE:` carga la imagen en escala de grises.

`Cv2.IMREAD_UNCHANGED:` carga la imagen tal cual.

En vez de escribir estos flags al completo, podemos simplemente añadir ‘1’, ‘0’ o ‘-1’ respectivamente al final de la función.

Figura 19. Comparación entre una imagen a color y la misma imagen en escala de grises modificada mediante OpenCV.

6.1.2 Mostrar una imagen

Para mostrar una imagen usaremos la función cv2.imshow() que abre una ventana con la imagen que se desea mostrar, dicha ventana será del mismo tamaño que la imagen.

```
Cv2.imshow('imagen', img)
```

El primer argumento ('imagen') es una cadena de caracteres que contiene el nombre de la ventana que se va a abrir, el segundo argumento (img) es la variable que contiene la imagen.

En conjunto con la función anterior se usan las siguientes funciones:

```
Cv2.waitKey(0)
```

```
Cv2.destroyAllWindows()
```

La primera de ellas es una función de unión con el teclado. Se le pasa el tiempo en milisegundos que espera ante cualquier evento de teclado. Si se presiona una tecla durante dicho tiempo el programa continúa ejecutándose. Si se le pasa '0' espera indefinidamente a que se pulse una tecla.

La segunda función destruye las ventanas que han sido creadas una vez hayamos terminado de mostrar la imagen.

6.1.3 Guardar una imagen

Para realizar el guardado de una imagen utilizamos la función cv2.imwrite():

```
Cv2.imwrite('NuevaImagen.png', img)
```

El primer argumento ('NuevaImagen.png') que le pasamos a la función es el nombre de guardado del archivo que se va a crear con la imagen, el segundo (img) es la variable que contiene la imagen.

En el ejmplo anterior, se guarda la imagen en formato ‘png’, el formato de guardado se puede definir en el mismo nombre tal y como se ve en dicho ejemplo, o bien se puede definir mediante un flag al final de la función (como tercer parámetro).

6.1.4 Ejemplo

En las siguientes lineas de código se muestra un ejemplo práctico que resume lo descrito en los apartados anteriores, se trata de un programa que carga una imagen en escala de grises, la muestra por pantalla y la guarda si se pulsa la tecla ‘s’ terminando la ejecución o bien, simplemente termina sin guardar la imagen si se pulsa la tecla ‘ESC’:

```
import numpy as np
import cv2

img = cv2.imread('imagen', 0)

cv2.imshow('imagen', img)
k = cv2.waitKey(0)

if k == 27: #espera a la tecla ESC
     cv2.destroyAllWindows()

Elif k == ord('s'): #espera a la tecla s
     Cv2.imwrite('NuevaImagen.png', img)
     cv2.destroyAllWindows()
```

6.1.5 Capturar video desde la cámara

Tenemos que diferenciar entre dos casos, si utilizamos una cámara USB, o bien si usamos el Raspberry Pi Camera Module.

6.1.5.1 Utilizando una cámara USB genérica

Primero necesitamos crear un objeto “VideoCapture”. El argumento que le pasamos puede ser el índice del dispositivo (de la cámara en este caso), o bien el nombre del archivo de video que queramos usar. Normalmente si hay una única cámara conectada a la Raspberry Pi le pasamos un ‘0’ como se muestra a continuación:

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)
```

Tras esto, capturamos “frame a frame” y al final liberamos la captura tal y como se muestra:

```
while(True)
     #Capturar frame a frame
     Ret, frame = cap.read()

     #En este punto se realizan las operaciones con los frames
```

```
     #Mostramos el frame resultante
     Cv2.imshow('frame', img)
     if cv2.waitKey(1) & 0xFF == ord('q'):
          break
#Cuando se haya terminado el proceso, liberamos la captura
cap.release()
cv2.destroyAllWindows()
```

6.1.5.2 Utilizando la Raspberry Pi Camera Module V2

Si en el proyecto requerimos el uso de la Raspberry Pi Camera Module, necesitamos recurrir a la librería PiCamera como se ha descrito anteriormente en el trabajo en el apartado de librerías. Mediante esta, podemos acceder al módulo de la cámara tal y como se muestra en el siguiente código:

```
#Primero importamos los paquetes necesarios
from picamer.array import RGBArray
from picamera import PiCamera
import time
import cv2

#inicializar la cámara
camera = PiCamera()


#Podemos retocar ciertos parámetros como la resolución, el
#framerate,…

camera.resolution = (640,480)
camera.framerate = 32
rawCapture = piRGBArray(camera, size = (640,480))

#Permitimos a la cámara hacer un "warmup"
time.sleep(0.1)

#capturar los frames de la cámara
for frame in camera.capture_continuous(rawCapture, format =
"bgr", use_video_port = True):
image = frame.array

#Ahora realizamos las operaciones que queramos con el frame

#Mostramos el frame ya procesado
Cv2.imshow('frame', image)

#Despejamos el stream para preparar el siguiente frame
rawCapture.truncate(0)

#Si se pulsa la tecla 's', salimos del bucle
if key == ord('s'):
     break
```

7 MÉTODOS DE DETECCIÓN DE CARACTERÍSTICAS

7.1 Introducción a la detección de características

Para poder solucionar problemas en visión artificial, la máquina necesita entender las distintas características de las imágenes que reciben (conocidas en inglés como “features”).

Las características son representaciones matemáticas de áreas clave en una imagen, básicamente son las representaciones vectoriales del contenido visual de una imagen que utilizamos para poder realizar operaciones matemáticas con el fin de transformar dicha imagen y poder manipularla a nuestro antojo, este procedimiento es conocido con el nombre de “tratamiento de imágenes”.

Para comprender un poco más acerca de este tema, veamos el siguiente ejemplo:
Teniendo un rectángulo coloreado como el de la figura consideremos tres regiones distintas, cada una se corresponde con uno de los rectángulos que se pueden observar.

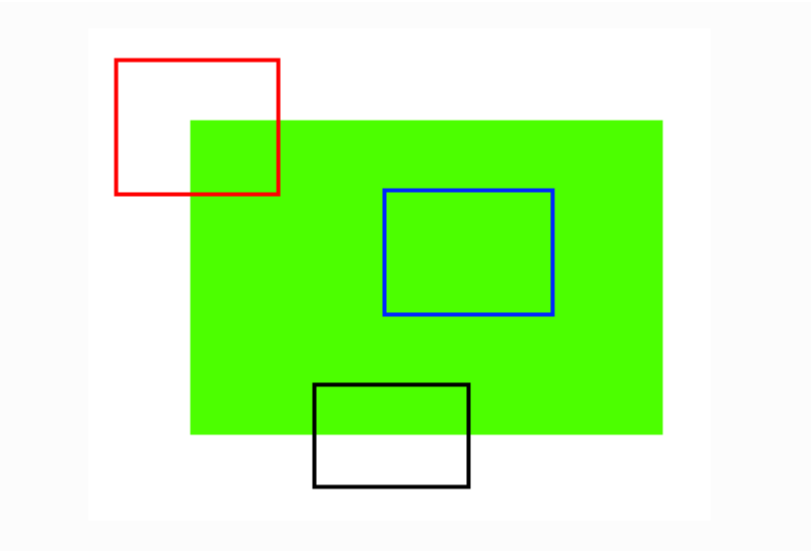


Figura 20. Ejemplo de las distintas regiones de un rectángulo.

El rectángulo azul es un área plana sin cambios lo que hace que sea difícil de buscar ya que hay muchas posiciones de la imagen en la que ese rectángulo se vería exactamente igual.

El rectángulo negro contiene un borde. Si lo desplazamos en dirección vertical veremos que el contenido del recángulo varía, mientras que si lo movemos en dirección horizontal se ve exactamente igual.

En el caso del contenido del rectángulo rojo pasa algo completamente distinto a los casos anteriores, se trata de una esquina. Dondequiera que se desplace este rectángulo el contenido que se encuentra siempre será distinto al de la posición en la que se encuentra en la figura por lo que consideramos que esta posición es única. Podemos considerar por lo tanto que las esquinas son buenas características (“Good Features”) de una imagen.
Ahora que conocemos cual es el tipo de características que buscamos en una imagen surge la pregunta de cómo encontramos dichas características, lo que haremos será buscar regiones en las

imágenes que presenten una variación máxima cuando se mueven una pequeña cantidad en todas las regiones a su alrededor. A este proceso de búsqueda de características se le conoce comúnmente con el nombre de detección de características ("Feature Detection").

Una vez se han detectado dichas características, tomamos una región alrededor de las mismas que permite buscar dichas regiones en otras imágenes para establecer correspondencias entre ellas. A este proceso se le conoce como descripción de características ("Feature Description")

En este trabajo nos centraremos en la detección de características y en los métodos que se utilizan para llevar a cabo este proceso. Se empezará explicando métodos que son precedentes directos a los que vamos a comparar y que no entran en dicha comparación ya que no son tan completos ni robustos como los que son objeto de este trabajo. Los principales métodos objeto de comparación se utilizan de manera indistinta en aplicaciones de procesamiento de imágenes en la actualidad.

7.2 Detector de esquinas de Harris

Teniendo en consideración que las esquinas son regiones de una imagen con una gran variación de intensidad en todas las direcciones, Chris Harris y Mike Stephens propusieron una forma de encontrar estas esquinas en su artículo "A Combined Corner and Edge Detection [5] en 1988, por lo que se denominó a este método "Detector de esquinas de Harris" o como se conoce en inglés, "Harris Corner Detector".

Pasaron esta simple idea a una expresión matemática que encuenta la diferencia de intensidad de desplazamiento de (u, v) en todas las direcciones, esto se expresa de la siguiente manera:

$$E(u,v) = \sum_{x,y} w(x,y)\,[I(x+u,y+v) - I(x,y)]^2$$

El término $w(x,y)$ se denomina "window function", se trata de una ventana rectangular o gaussiana que da peso a los píxeles debajo de ella.

Para realizar la detección de esquinas tenemos que maximizar la función $E(u,v)$, concretamente el segundo término. Aplicando Taylor a la ecuación anterior y utilizando algunos pasos matemáticos obtenemos la ecuación final:

$$E(u,v) \approx [u\ v]\ M \begin{bmatrix} u \\ v \end{bmatrix}$$

Dónde:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

I_x e I_y son las derivadas parciales en x e y respectivamente. Estos valores se encuentran gracias a la función de OpenCV: `cv.Sobel()`

Ahora se evalua la siguiente expresión:

$$R = \det(M) - k(trace(M))^2$$

Dónde:

- $det\ (M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- λ_1 y λ_2 son los autovalores de M

Y se utiliza un sistema de puntuación para evaluar si la ventana puede o no contener una esquina mediante el siguiente criterio:

$$\begin{cases} Cuando\ |R|\ es\ pequeño, lo\ que\ sucede\ cuando\ \lambda_1 \text{ y } \lambda_2\ son\ pequeños, es\ una\ región\ plana. \\ Cuando\ R < 0, que\ ocurre\ cuando\ \lambda_1 \gg \lambda_2\ o\ viceversa,\ \ es\ un\ borde. \\ Cuando\ R\ es\ grande, que\ sucede\ cuando\ \lambda_1 \text{ y } \lambda_2\ son\ grandes\ y\ \lambda_1 \sim \lambda_2, la\ región\ es\ una\ esquina. \end{cases}$$

Lo descrito anteriormente da como resultado la siguiente distribución:

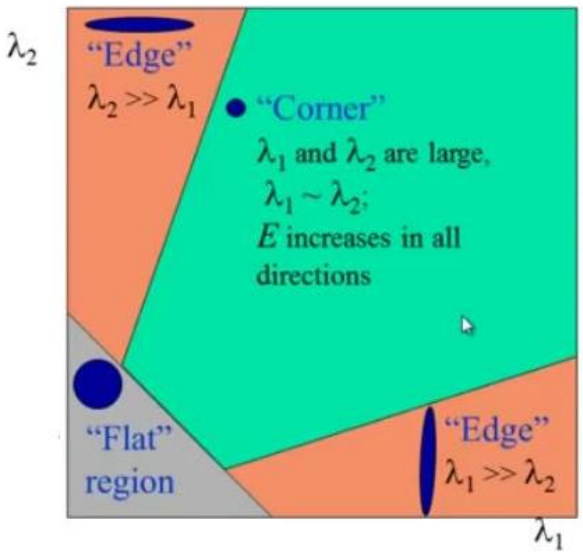


Figura 21. Gráfica para la evaluación de características mediante el método de detección de esquinas de Harris.

7.3 Método de Shi-Tomashi para la detección de esquinas

En el año 1994 J. Shi y C. Tomashi realizaron una pequeña modificación en su papel "Good features to track" [6].

La función de puntuación para evaluar el tipo de característica es el siguiente:

$$R = \min\ (\lambda_1, \lambda_2)$$

Si es mayor que el valor umbral, se considera como una esquina. Si lo trazamos en el espacio $\lambda_1 - \lambda_2$ obtenemos la siguiente distribución:

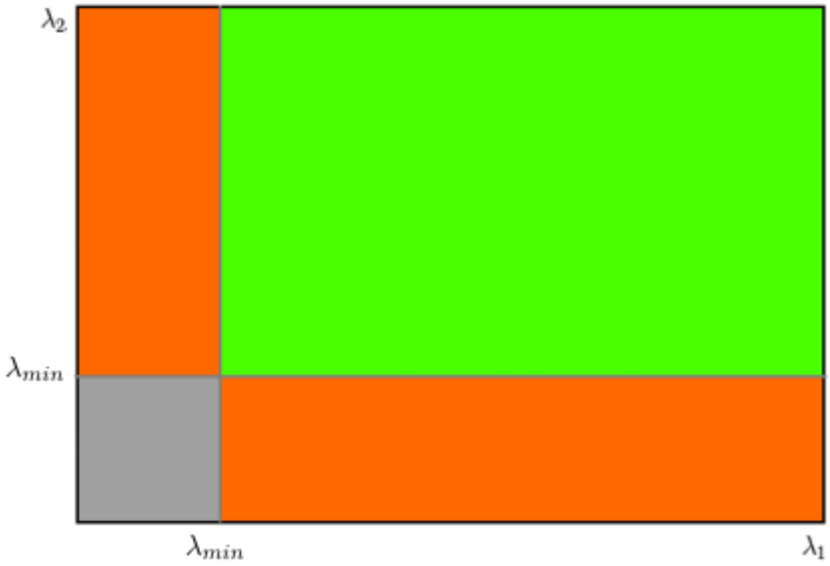


Figura 22. Distribución del método de Shi-Tomashi para la detección de esquinas.

En la figura anterior, la región verde muestra los valores para de λ_1 y λ_2 para los que se considera una esquina.

7.4 SIFT

Los métodos de detección de esquinas descritos hasta ahora son invariantes a la rotación, lo que significa que incluso si la imagen gira podemos seguir localizandolas en la imagen, pero estos métodos no contemplan cambios de escala en las imágenes.

Por ello en 2004, D. Lowe de la Universidad de British Columbia ideó un nuevo algoritmo denominado SIFT (Scale Invariant Feature Transform) en su artículo "Distinctive Image Features from Scale Invariant Keypoints" [7], que extrae puntos clave y sus descriptores.

El método consiste en lo siguiente:
Primero debemos tener en cuenta la escala de la imagen, por lo que se utilizan tres parámetros (x, y, σ), 'x' e 'y' representan un punto en el espacio donde puede encontrarse un punto clave potencial y 'σ' la escala a la que se encuentra. Estos valores se buscan mediante diferencia de gaussianas. Una vez encontrados estos puntos clave deben ser refinados para obtener resultados mas precisos. Utilizando la expansión del espacio de escala de la serie de Taylor para obtener una ubicación más precisa de los extremos eliminando los puntos clave de bajo contraste y los puntos clave pertenecientes a bordes quedando únicamente los puntos clave en los que estamos interesados.

Ahora se asigna una orientación a cada punto clave para lograr la invarianza a la rotación en la imagen. Se toma un vecindario alrededor del punto clave dependiendo de la escala, la magnitud y la dirección del gradiente.

Por último, se crea un el descriptor del punto clave. Se toma un vecindario de 16 x 16 alrededor del punto dividiéndose en varios sub-bloques dando al final como resultado 128 valores bin. Se representa como un vector y se toman medidas adicionales para conseguir robustez contra cambios de iluminación, rotación, etc…

7.5 SURF

En 2006 Bay, H., Tuytelaars, T. y Van Gool, L. publicaron el artículo: "SURF: Speeded Up Robust Features" [8] que introdujo un nuevo algoritmo denominado SURF.

SURF aproxima a Laplaciano de Gauss mediante filtro de caja. La siguiente figura muestra una demostración de tal aproximación. Una gran ventaja de la aproximación es que la convolución con filtro de caja se puede calcular fácilmente con la ayuda de imágenes integrales y se puede hacer en paralelo para diferentes escalas.

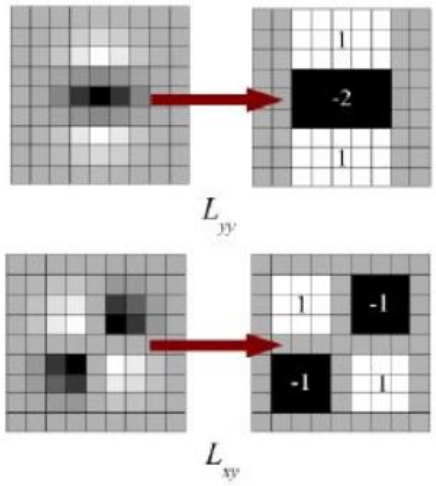


Figura 23. Aproximación mediante filtro de cajas.

Para la descripción de características SURF utiliza respuesta de ondas en dirección horizontal y vertical. Se toma una vecindad de tamaño 20s x 20s alrededor del punto clave donde 's' es la longitud. Se divide en subregiones y se forma un vector del siguiente tipo:

$$v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$$

Lo que nos proporciona un descriptor de características SURF de dimensión 64 (la mitad que SIFT). Cuanto menor sea la dimensión, mayor será la velocidad de computación.

Otro cambio de este algoritmo frente a SIFT es el uso del signo del Laplaciano. El signo del laplaciano distingue las manchas brillantes sobre fondos oscuros. En la etapa de correspondencia, sólo comparamos las características si tienen el mismo tipo de contraste (como se muestra en la figura). Este uso de información mínima permite una coincidencia más rápida sin reducir el rendimiento del descriptor.

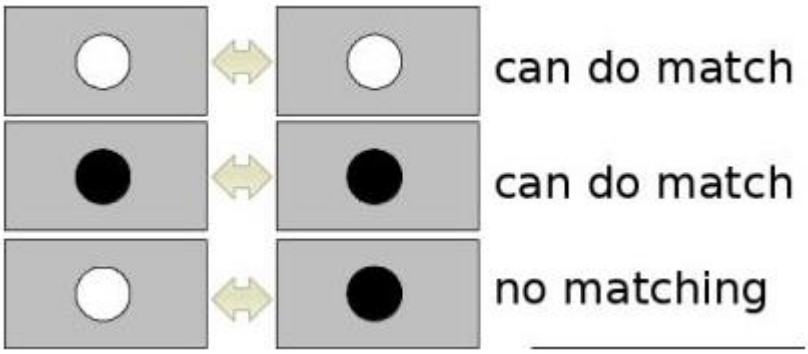


Figura 24. Comparación del tipo de contraste en el método SURF.

7.6 FAST

Hasta ahora hemos visto varios detectores de características, pero cuando se mira desde el punto de vista de una aplicación de tiempo real, no son lo suficientemente rápidos. Un buen ejemplo sería el uso de SLAM (Simultaneous Localization And Mapping) en dispositivos con recursos computancionales limitados como nuestra Raspberry Pi.

Como solución, el algoritmo FAST (Features from Accelerated Segment Test) fue propuesto por Edward Rosten y Tom Drummond en su artículo "Machine Learning for high – speed corner detection" [9] en 2006. El algoritmo consiste en lo siguiente:

Primero se selecciona un píxel 'p' en la imagen que se identificará como un punto de interés siendo su intensidad 'I_p'. Se selecciona un valor umbral 't' apropiado. Se toma un círculo de 16 pixeles alrededor del píxel que se va a comprobar como se ve en la siguiente figura:

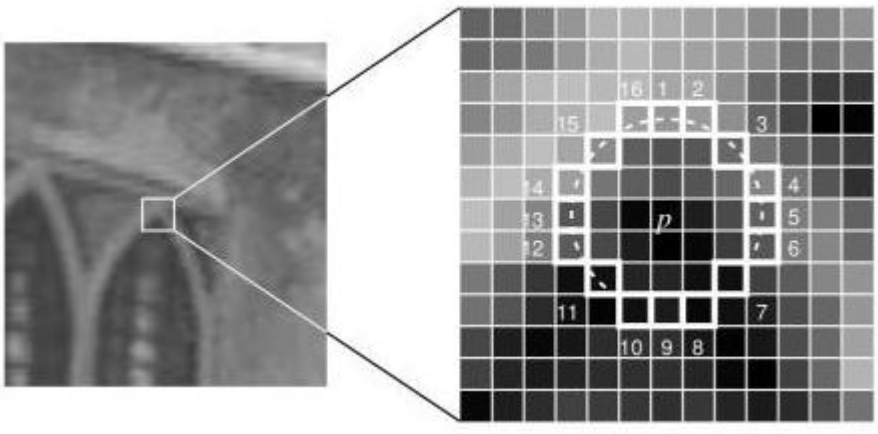


Figura 25. Región tomada en el método FAST para la detección de características.

Se comprueba si el píxel es una esquina si existe un conjunto de 'n' pixeles contiguos en el círculo seleccionado que son o más brillantes que 'I_p + t', o más oscuros que 'I_p - t'.
Se realiza una prueba de alta velocidad para excluir una gran cantidad de "no esquinas". Esta prueba examenina solamente los píxeles en las posiciones 1,9,5 y 13. Si 'p' es una esquina debe verificarse que tres de estos pixeles deben ser o mas brillantes que 'I_p + t', o más oscuros que 'I_p - t'. Si no se cumple entonces 'p' no puede ser una esquina.

7.7 BRIEF

Este algortimo proporciona un acceso directo para encontrar las cadenas binarias directamente sin encontrar descriptores. Toma un parche de imagen suavizado y selecciona un conjunto de pared de ubicaciones 'nd' (x, y) de una manera única. Luego se realizan algunas comparaciones de intensidad entre pixeles en las ubicaciones 'x' e 'y' para obtener una cadena de bits nd-dimensional.

Breve es un descriptor de características, no proporciona ningún método para encontrar las características, por lo tanto, hay que usar cualquier otro detector de características como SIFT, SURF, …

7.8 ORB

ORB proviene de "OpenCV Labs". Este algoritmo fue presentado por Ethan Rublee, Vincent Rabaud, Kurt Konolige y Gary R.

ORB es básicamente una fusión del detector de puntos clave FAST y el descriptor BRIEF con una serie de modificaciones que mejoran el rendimiento. Primero se utiliza FAST para encontrar puntos clave, luego aplica el método de detección de Harris para encontrar los 'N' puntos principales entre ellos.
Tras esto se calcula el centroide ponderado de intensidad del parche con la esquina ubicada en el centro. La dirección del vector desde este punto de esquina al centroide nos da la orientación. Para mejorar la invarianza de la rotación, los momentos se calculan con 'x' e 'y' que deben estar en una región circular de radio 'r', donde 'r' es el tamaño del parche.

Ahora para los descriptores, ORB utiliza descriptores BRIEF. ORB dirige BRIEF de acuerdo con la orientación de los puntos clave. Para cualquier conjunto de características de 'n' pruebas binarias en la ubicación a evaluar (x_i, y_i) se define una matriz de tamaño 2 x 'n' llamada 'S' que contiene las coordenadas de esos píxeles. Después utilizando la orientación del parche, θ, se encuentra su matriz de rotación y gira la 'S' para obtener una la versión girada 'S_θ'.

ORB discretiza el ángulo a incrementos de 2π / 30 (12 grados), y construye una tabla de búsqueda de patrones BRIEF calculados previamente. Mientras la orientación del punto clave θ sea coherente en todas las vistas, se utilizará el conjunto correcto de puntos S_θ para calcular su descriptor.

8 ANÁLISIS DE LOS MÉTODOS DE DETECCIÓN DE CARACTERÍSTICAS

En este apartado del proyecto, se va a realizar el análisis de los métodos de detección de características más complejos que se vieron en el apartado anterior. La finalidad de esto es poder seleccionar un buen algoritmo que usar en nuestra Raspberry Pi para utilizar programas de correspondencia entre imágenes y conseguir un rendimiento adecuado.

Los métodos que se van a evaluar son los siguientes:

- SIFT
- SURF
- ORB

Este análisis se va a llevar a cabo en la placa Raspberry Pi propuesta y utilizando el mismo hardware y software para todos los códigos que se van a utilizar para asegurar que los datos de las comparaciones que se van a realizar son tomados en igualdad de condiciones.

A continuación, se presentan los puntos que vamos a tener en cuenta a la hora de realizar esta comparación:

8.1 Keypoints (puntos clave)

Los Keypoints o puntos clave se puede definir como puntos de interés. Son ubicaciones espaciales o puntos en la imagen que definen lo que es interesante o lo que se destaca en la imagen.

La detección de puntos de interés es en realidad una parte de la detección de manchas (“blobs”), cuyo objetivo es encontrar regiones interesantes o áreas especiales en una imagen.

La razón por la cual los puntos clave son especiales es porque no importa cómo cambie la imagen, incluso si la imagen se gira, contrae, expande, transforma o está sujeta a distorsión (transformación u homografía proyectiva), se deberían poder encontrar los mismos puntos clave en esta imagen modificada cuando se compara con la imagen original.

Figura 26. Ejemplo de extracción de puntos clave de una imagen.

Teniendo en cuenta que estos puntos clave deben mantenerse, los utilizaremos para comprobar el rendimiento, aplicando distintos efectos a las imágenes y comprobando cómo se comportan los distintos métodos de detección de características.

8.2 Tiempo de ejecución

Cuando hablamos del tiempo de ejecución nos referimos al intervalo de tiempo que tarda un programa (o algoritmo en nuestro caso) en ejecutarse en un determinado sistema operativo.

Se trata de un parámetro muy importante ya que queremos que este tiempo sea el menor posible. Queremos ser capaces de ejecutar este algoritmo en tiempo real. Un sistema de tiempo real (STR) interacciona con el mundo real respondiendo al paso del tiempo o a sucesos externos. No sólo es importante el resultado lógico de la computación sino también el tiempo en el que se producen los resultados.

Buscamos por tanto que el tiempo que tardan en ejecutarse los algoritmos sean lo menor posibles.

8.3 Resultados de las simulaciones

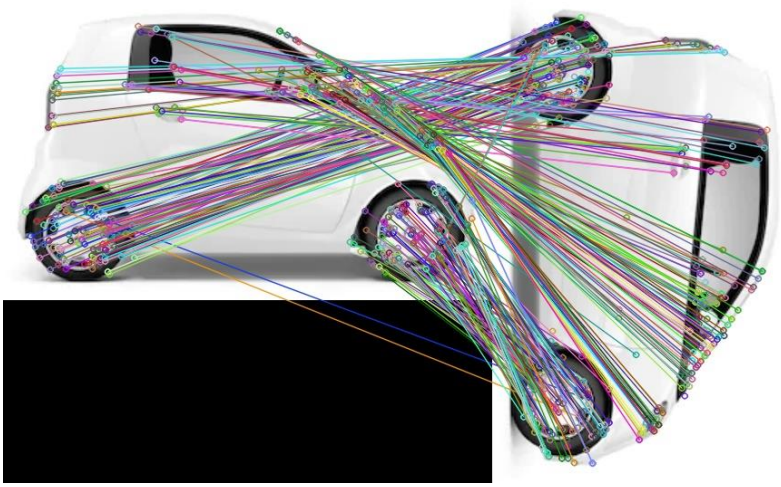
En este apartado se muestran los resultados de las simulaciones realizadas en la Raspberry Pi para los distintos métodos de detección de características. Para estas simulaciones se han utilizado imágenes con transformaciones con respecto a la imagen a la original para ver como se comportan los detectores.

8.3.1 Rotación

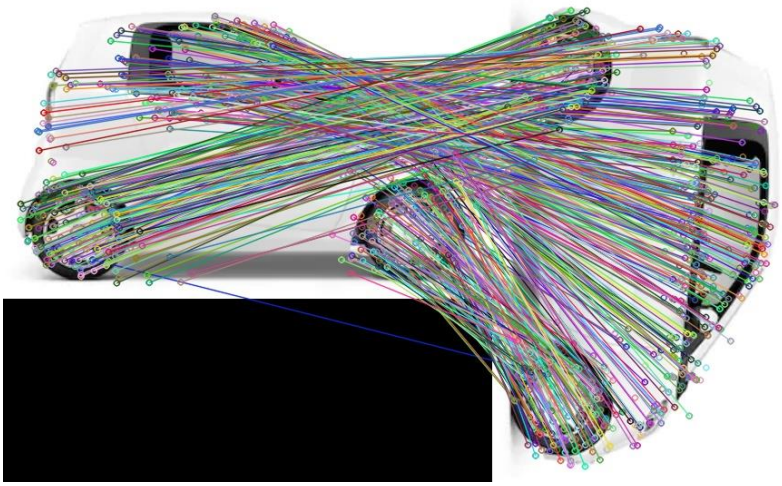
A continuación, se presentan los resultados para imágenes rotadas. Se han considerado rotaciones de 90 y 180 grados. Los resultados se dan en las tablas 2,3 y en la figura 26.

Rotación 90 grados					
	TIME	Keypoints1	Keypoints2	Matches	Match rating
SIFT	0,89569	387	389	360	92,54%
SURF	0,43512	648	643	595	91,82%
ORB	0,09047	475	475	475	100%

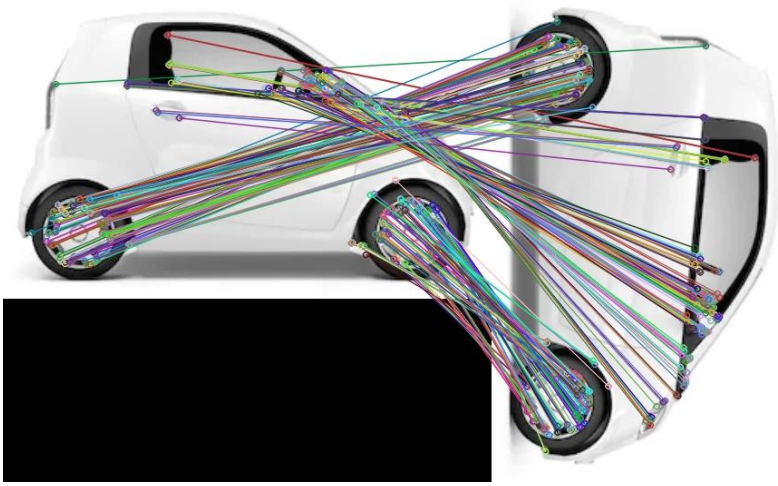
Tabla 2. Resultados de la simulación para imagen rotada 90 grados.



(a)



(b)

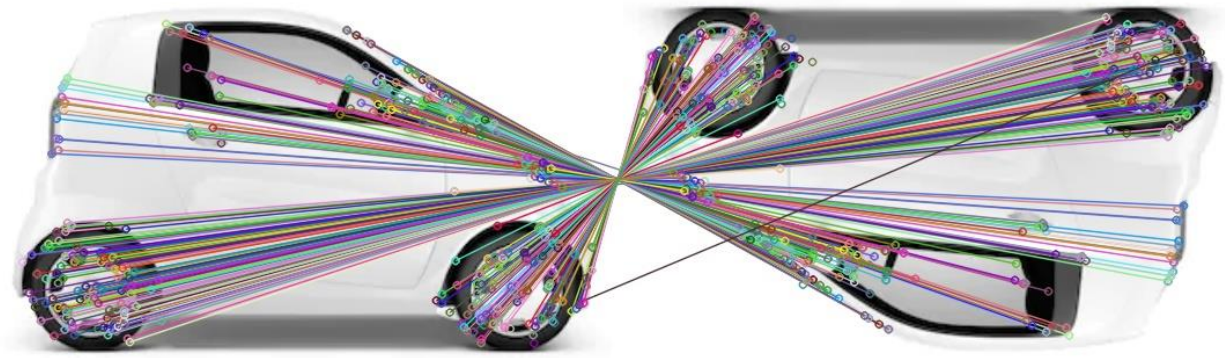


(c)

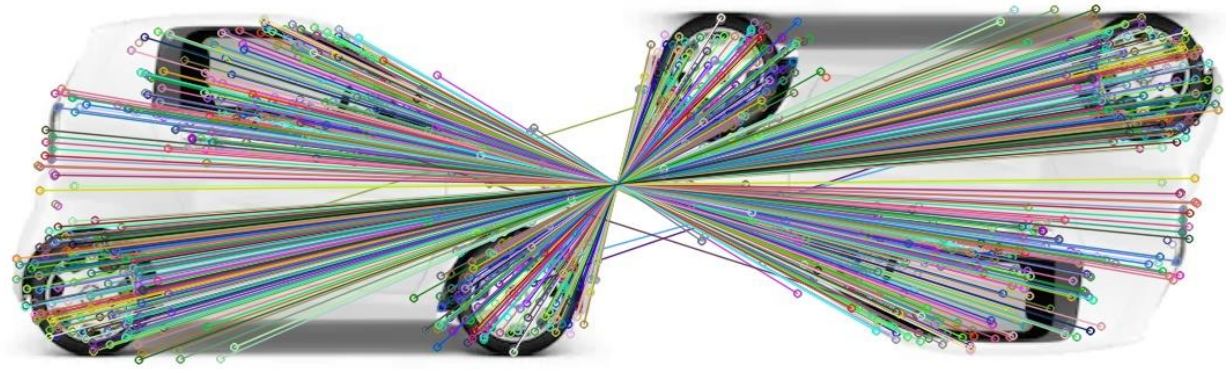
Figura 27. Matching de las imágenes con rotación de 90 grados utilizando (a) SIFT (b) SURF y (c) ORB

Rotación 180 grados					
	TIME	Keypoints1	Keypoints2	Matches	Match rating
SIFT	0,91222	387	391	349	89,25%
SURF	0,43092	648	634	589	90,89%
ORB	0,09047	475	475	475	100%

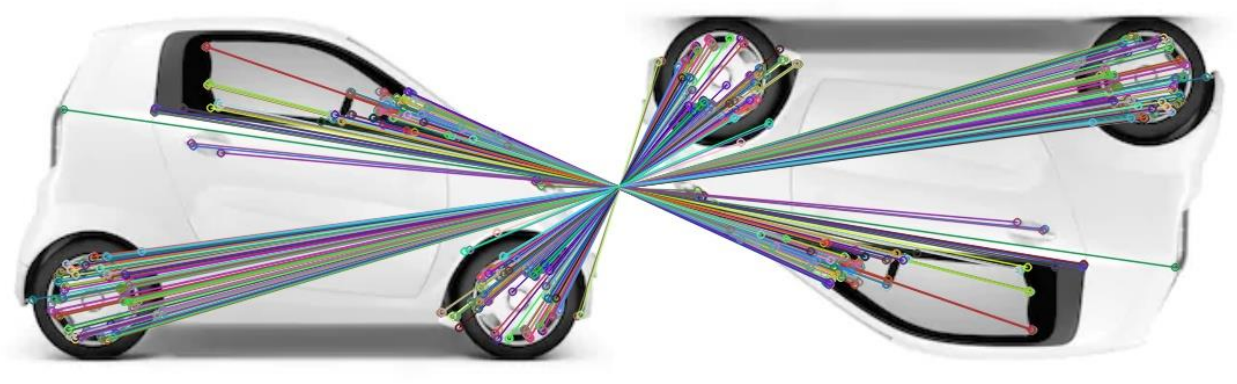
Tabla 3. Resultados de la simulación para imagen rotada 180 grados.



(a)



(b)



(c)

Figura 28. Matching de las imágenes con rotación de 180 grados utilizando (a) SIFT (b) SURF y (c) ORB

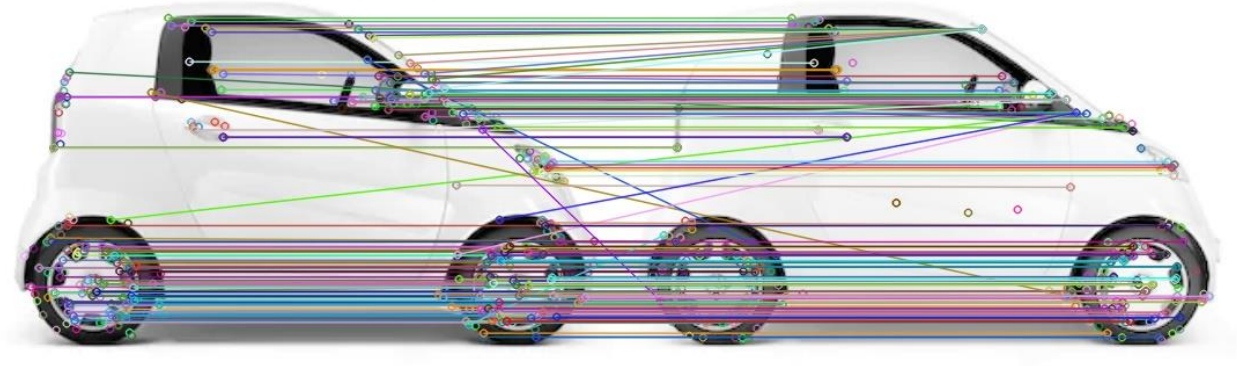
De estos resultados, se observa que el método con un mayor matching rate es ORB para ambos casos. Además de esto también es el que menor tiempo de ejecución tiene siendo un décimo de el tiempo que tarda el algoritmo SIFT.

8.3.2 Variación de intensidad

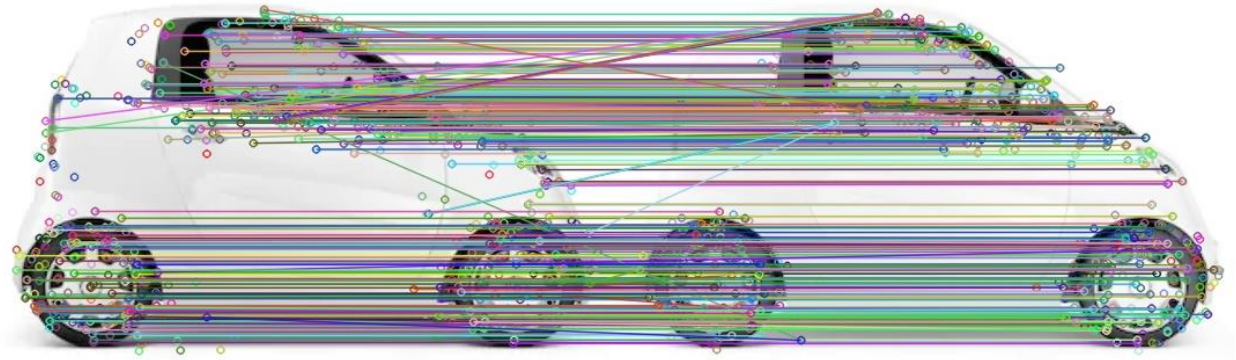
A continuación, se presentan los resultados de la comparación para imágenes con variación en la intensidad. Los resultados se presentan en la tabla 4 y en la figura 29.

Variación de Intensidad (claridad)					
	TIME	Keypoints1	Keypoints2	Matches	Match rating
SIFT	0,9011	387	308	195	50,38%
SURF	0,4171	648	605	332	51,23%
ORB	0,089	475	473	229	48,21%

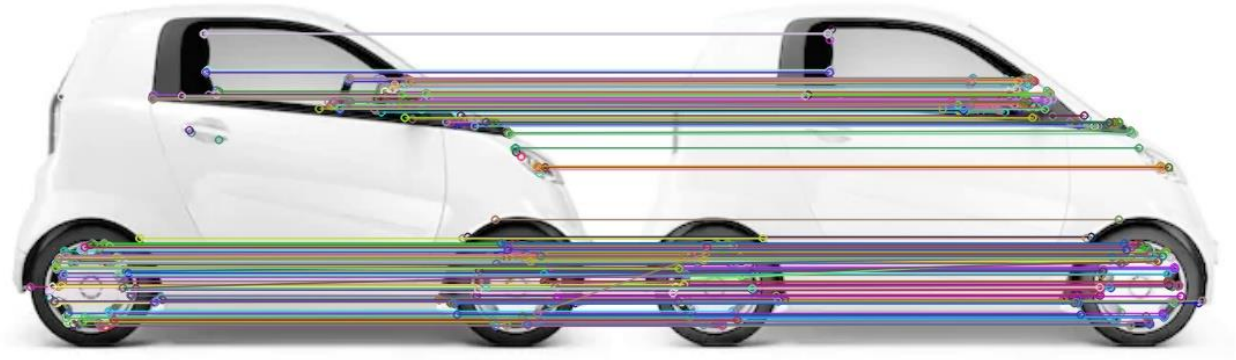
Tabla 4. Resultados de la simulación para imágenes con variaciones en la intensidad.



(a)



(b)



(c)

Figura 29. Matching de las imágenes con variación en la intensidad utilizando (a) SIFT (b) SURF y (c) ORB

El método con mayor match rating para variación en la intensidad es SURF, el menor tiempo de ejecución lo mantiene ORB.

8.3.3 Transformación afín

A continuación, se presentan los resultados de la comparación para imágenes con transformación afín. Los resultados se presentan en la tabla 5 y en la figura 30.

Transformación afín					
	TIME	Keypoints1	Keypoints2	Matches	Match rating
SIFT	0,8936	387	220	114	29,45%
SURF	0,43032	648	529	95	14,66%
ORB	0,08583	475	345	34	7,15%

Tabla 5. Resultados de la simulación para imágenes con transformación afín.

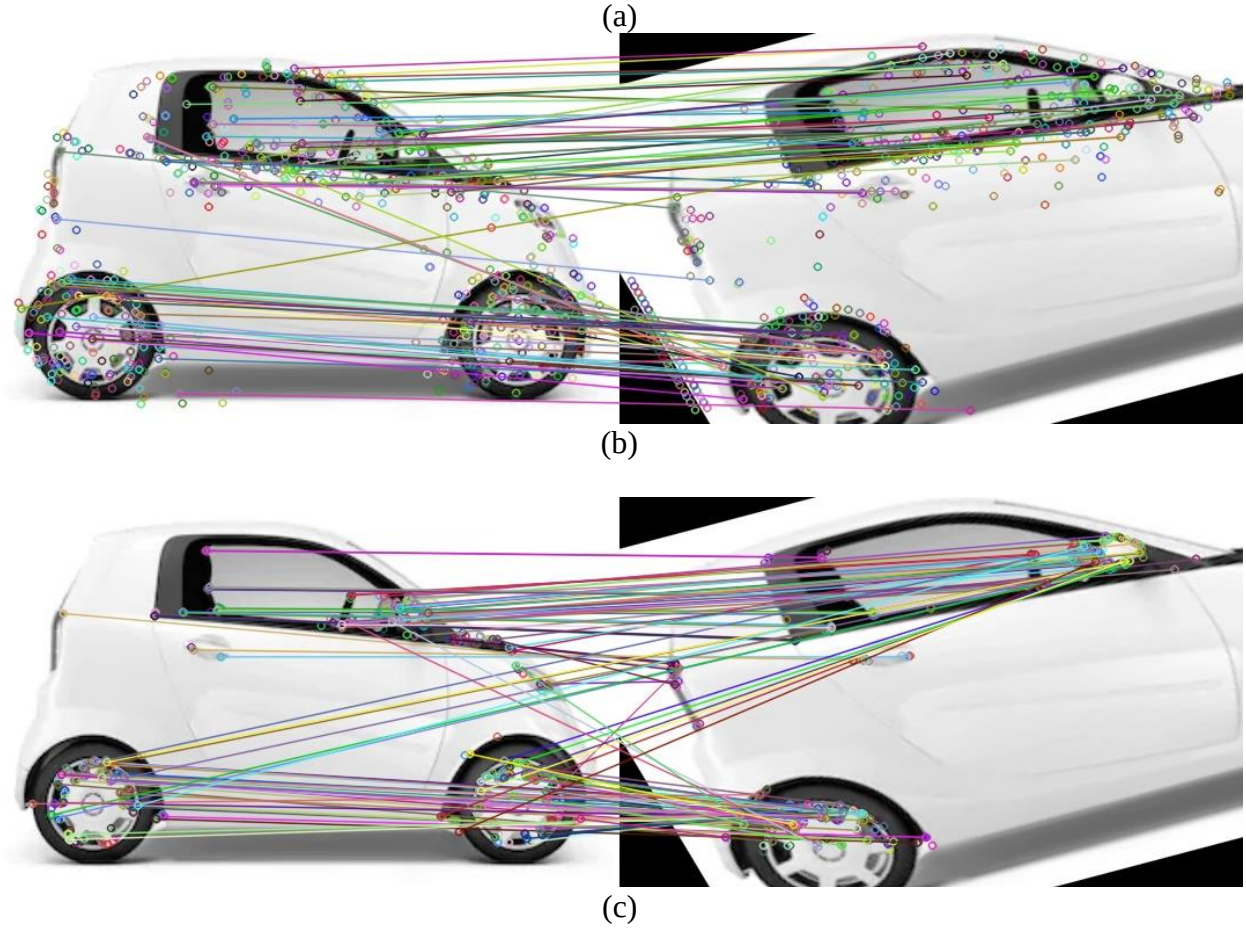


Figura 30. Matching de las imágenes con transformación afín utilizando (a) SIFT (b) SURF y (c) ORB

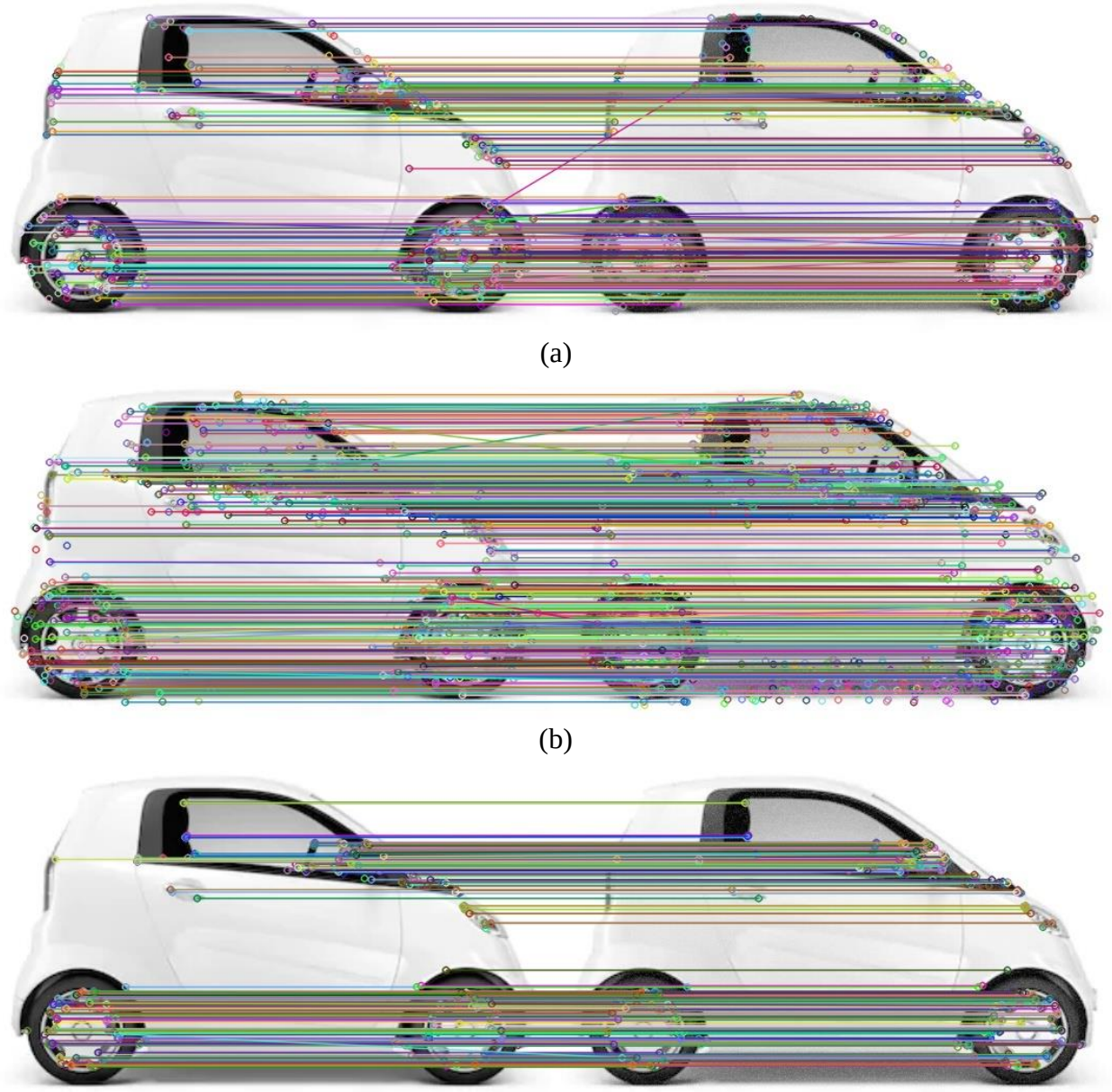
El método con mayor match rating para variación en la intensidad es SIFT, el menor tiempo de ejecución lo mantiene ORB.

8.3.4 Ruido

A continuación, se presentan los resultados de la comparación para imágenes con ruido. Los resultados se presentan en la tabla 6 y en la figura 31.

Inclusión de ruido					
	TIME	Keypoints1	Keypoints2	Matches	Match rating
SIFT	0,91529	387	424	262	61,79%
SURF	0,47284	648	920	469	50,97%
ORB	0,08918	475	474	365	76,84%

Tabla 6. Resultados de la simulación para imágenes con ruido.



(a)

(b)

(c)

Figura 31. Matching de las imágenes con ruido utilizando (a) SIFT (b) SURF y (c) ORB

El mayor matching rate lo obtiene el algoritmo ORB, así como el menor tiempo de ejecución.

CONCLUSIÓN

En vista a los resultados provistos en el apartado anterior, el método de detección de características más apropiado para utilizar en nuestro sistema es ORB.

ORB ha demostrado ser un método muy eficiente ya que tiene un tiempo de ejecución muy pequeño comparado con el resto de los métodos, siendo una décima parte del método SIFT y una sexta parte del método SURF. Hay que tener en cuenta que el tiempo mostrado es solamente el tiempo de ejecución del detector de características por lo que cuando se implementen más operaciones en el código, el tiempo de ejecución aumentará por lo que ORB es un buen método para utilizar en nuestra Raspberry Pi.

Pese a tener un tiempo de ejecución tan bajo, ha demostrado presentar unos resultados bastante satisfactorios a la hora de evaluar el "matching rate" es decir, los puntos clave detectados en las imágenes originales y en las imágenes retocadas tienen un alto porcentaje de coincidencia en la mayoría de los casos (normalmente superior a SIFT y SURF) por lo que podemos afirmar que nos encontramos ante un método robusto.

Otro factor para tener en cuenta es la patente, SIFT y SURF son métodos patentados y para poder utilizarlos en aplicaciones comerciales hay que pagar, mientras que ORB al provenir de "OpenCV Labs" es de uso gratuito.

Con todo esto, se concluye que el detector de características más eficiente para utilizar en proyectos que involucren un sistema con Raspberry Pi es ORB.

Implementando este algoritmo, podemos crear un programa que identifique objetos en imágenes y conseguir resultados como se muestra a continuación:

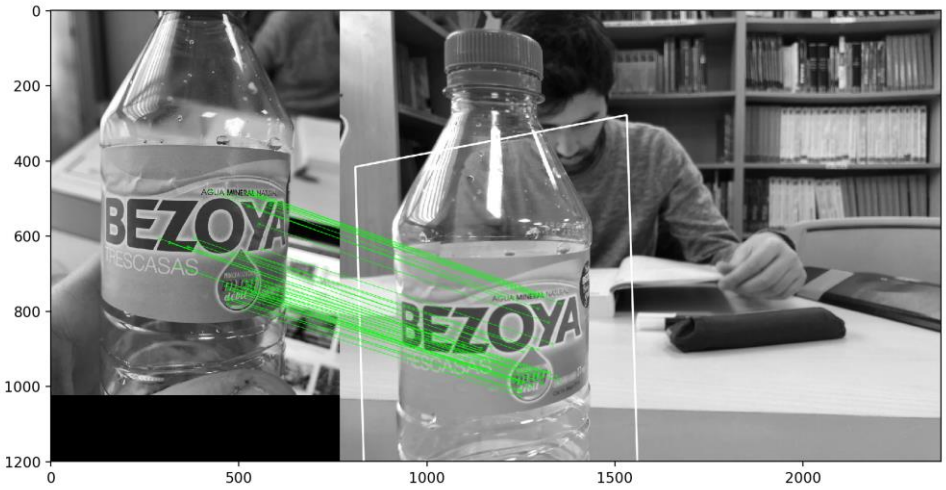


Figura 32. Ejemplo de reconocimiento de objetos en imágenes mediante ORB.

ANEXO A: CÓDIGO

A continuación, se presentan los códigos utilizados en este trabajo para la evaluación de los métodos de detección de características:

SIFT:

```
import numpy as np
import cv2

# Leer imágenes
img1 = cv2.imread('imagen1.jpeg')
img2 = cv2.imread('imagen2.jpeg')

# Iniciar el detector SIFT
sift = cv2.xfeatures2d.SIFT_create()

# Conversión a escala de grises
img1= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

e1 = cv2.getTickCount()

# Encontrar los keypoints y descriptores
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)

e2 = cv2.getTickCount()

# Número de Keypoints
print(len(des1))
print(len(des2))

# BFMatcher
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Crear imágenes con Keypoints
img1=cv2.drawKeypoints(img1,kp1,img1)
cv2.imwrite('2sift_keypoints.jpg',img1)

img2=cv2.drawKeypoints(img2,kp2,img2)
cv2.imwrite('3sift_keypoints.jpg',img2)

# Aplicar ratio test
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])

print("matches ")
```

```
print(len(good))

#Imprimir tiempo de ejecución
time = (e2 - e1)/ cv2.getTickFrequency()
print(time)

# Crear y guardar imagen de correspondencia
img3 = cv2.drawMatches(img1, kp1, img2, kp2,
     good,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
     )
cv2.imwrite('SIFTCorresp.jpg',img3)
```

SURF:

```
import numpy as np
import cv2

# Leer imágenes
img1 = cv2.imread('imagen1.jpeg')
img2 = cv2.imread('imagen2.jpeg')

# Iniciar el detector SURF
surf = cv2.xfeatures2d.SURF_create()

# Conversión a escala de grises
img1= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

e1 = cv2.getTickCount()

# Encontrar los keypoints y descriptores
kp1, des1 = surf.detectAndCompute(img1,None)
kp2, des2 = surf.detectAndCompute(img2,None)

e2 = cv2.getTickCount()

# Número de keypoints
print(len(des1))
print(len(des2))

# BFMatcher
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)

# Crear imágenes con Keypoints
img1=cv2.drawKeypoints(img1,kp1,img1)
cv2.imwrite('2surf_keypoints.jpg',img1)

img2=cv2.drawKeypoints(img2,kp2,img2)
cv2.imwrite('3surf_keypoints.jpg',img2)

# Aplicar test ratio
```

```
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])

print("matches ")
print(len(good))

#Imprimir tiempo de ejecución
time = (e2 - e1)/ cv2.getTickFrequency()
print(time)

# Crear y guardar imagen de correspondencia
img3 = cv2.drawMatches(img1, kp1, img2, kp2,
     good,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
     )
cv2.imwrite('SURFCorresp.jpg',img3)
```

ORB:

```
import numpy as np
import cv2

# Leer imágenes
img1 = cv2.imread('imagen1.jpeg')
img2 = cv2.imread('imagen2.jpeg')

# Iniciar detectorr ORB
orb = cv2.ORB_create()

# Conversión a escala de grises
img1= cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
img2= cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)

e1 = cv2.getTickCount()

# Encontrar los keypoints y descriptores
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

e2 = cv2.getTickCount()

# Número de Keypoints
print(len(des1))
print(len(des2))

# Crear imágenes con Keypoints
img1=cv2.drawKeypoints(img1,kp1,img1)
cv2.imwrite('2orb_keypoints.jpg',img1)

img2=cv2.drawKeypoints(img2,kp2,img2)
cv2.imwrite('3orb_keypoints.jpg',img2)
```

```
# BFmatcher
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1,des2)
matches = sorted(matches, key = lambda x:x.distance)

# Aplicar ratio test
good = []
for m,n in matches:
    if m.distance < 0.75*n.distance:
        good.append([m])

print("matches ")
print(len(good))

#Imprimir tiempo de ejecución
time = (e2 - e1)/ cv2.getTickFrequency()
print(time)

# Crear y guardar imagen de correspondencia
img3 = cv2.drawMatches(img1, kp1, img2, kp2,
     matches[:100],None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SING
     LE_POINTS)
cv2.imwrite('ORBCorresp.jpg',img3)
```

Crear imagen con ruido:

```
import cv2
import numpy as np


img = cv2.imread('imagen.png')
im = np.zeros(img.shape, np.uint8) # Usamos una imagen auxiliar
     para no borrar la original
mean = 0
sigma = 200
cv2.randn(im,mean,sigma) # Crear la distribución aleatoria
imagenruido = cv2.add(img, im) # Añadir ruido a la imagen
cv2.imwrite('imruido.jpeg',imagenruido) # Guardar la imagen
```

Crear imagen con transformación afín:

```
import numpy as np
import cv2 as cv

img = cv.imread('imagen.png')
rows,cols,ch = img.shape
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv.getAffineTransform(pts1,pts2) # Tomar transformación en
     base a los puntos seleccionados
imagenafin = cv.warpAffine(img,M,(cols,rows)) # Creación de
     imagen afin
```

```
cv.imwrite('pruebaafin.jpeg',imagenafin) # Guardar imagen
```

Crear imagen girada:

```
import numpy as np
import cv2

img = cv2.imread('iamgen.jpg',0)
rows,cols = img.shape
M = cv2.getRotationMatrix2D(((cols-1)/2.0,(rows-1)/2.0),90,1) #
     Girar imagen con matriz de rotación
imagenrotada = cv2.warpAffine(img,M,(cols,rows))

cv2.imwrite('imagenrot.jpeg',imagenrotada) # Guardar imagen
```

REFERENCIAS Y BIBLIOGRAFÍA

Referencias:
[1] https://www.xataka.com/ordenadores/raspberry-pi-3-model-b-analisis-mas-potencia-y-mejor-wifi-para-un-minipc-que-sigue-asombrando

[2] https://www.tomra.com/es-es/sorting/food/sorting-equipment/sentinel

[3] https://www.androidauthority.com/facial-recognition-technology-explained-800421/

[4] https://www.techiexpert.com/how-tesla-is-using-artificial-intelligence-and-big-data/

[5] Harris, C. G., & Stephens, M. (1988, August). A combined corner and edge detector. In *Alvey vision conference* (Vol. 15, No. 50, pp. 10-5244).

[6] Shi, J. (1994, June). Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition* (pp. 593-600). IEEE.

[7] D. G. Lowe, "Object recognition from local scale-invariant features," Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 1150-1157 vol.2, doi: 10.1109/ICCV.1999.790410.

[8] Bay, H., Tuytelaars, T., & Van Gool, L. (2006, May). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404-417). Springer, Berlin, Heidelberg.

[9] Rosten, E., & Drummond, T. (2006, May). Machine learning for high-speed corner detection. In *European conference on computer vision* (pp. 430-443). Springer, Berlin, Heidelberg.

Bibliografía:
https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

https://docs.python.org/3/

https://numpy.org/doc/1.18/

https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/

https://picamera.readthedocs.io/en/release-1.13/

https://www.scipy.org/

https://matplotlib.org/contents.html

https://www.raspberrypi.org/downloads/raspbian/

https://docs.opencv.org/3.4/d7/d66/tutorial_feature_detection.html