

Memoria:

PROYECTO_VISUALMOV

Sergio Santín de los Heros
CIFO – La Violeta
curso: [IFCT80-1] Creación de prototipos de IOT con Raspberry
profesor: Joan Masdemont

Índice:

Yo mismo	3
PRESENTACIÓN DEL PROYECTO_VISUALMOV	
(o PVM, Proyecto de Visualización Móvil)	4
ESTRUCTURA DEL PROYECTO Y PREPARACIÓN DEL ENTORNO	5
Diseño de una idea	5
Hardware (elementos del circuito)	5
El circuito	6
Instalación de la cámara	7
Una prueba con webcam (USB)	7
Instalación Az-Delivery cámara (CSI)	8
Software (Python)	10
Instalación de librerías	10
Código de programación (Proyecto-VisualMov.py)	11
Importar librerías	11
Configuración GPIO de periféricos	12
Recuperar información del S.O.	12
Generar carpetas y nombre de archivo	12
Borrar una carpeta	13
Criterios para eliminar una carpeta	13
¡Algo se mueve!	14
Shoot again!	15
Un viaje por la nube	15
Google Drive, la API	17
Un paseo por el taller de la creatividad. La caja (sin palabras) ...	25
Un ejemplo de ejecución de código desde PuTTY	25
CONCLUSIONES	26
FUENTES UTILIZADAS	27

Antes de iniciar este documento¹ en su aspecto más técnico y formal[...], quisiera prestar mi más sentido agradecimiento a las personas que me han permitido progresar en el desarrollo y evolución de este proyecto: a mi compañero y colaborador Xavier Espinosa que por visualizar una idea similar –lo cual nos elimina de todo concurso que premie la originalidad– el destino nos ha unido creando una sociedad bilateral facilitando documentación suficiente, y no menos eficiente, para convertir sendos ejercicios en una realidad; y a nuestro “destino” y docente, Joan Masdemont, que *“halla la luz en la oscuridad y limpia de impurezas el camino, hasta alcanzar nuestro cometido final”*, la aportación de nuevas ideas siempre bienvenidas para su estudio y futuras actualizaciones, puesto que, como le gusta recordar, *“tempus fugit”*, aunque en este caso lo asociaremos al tiempo eficiente destinado para el proyecto.

Me he permitido cierta licencia en la presentación del proyecto, como un pequeño preámbulo en la redacción de esta memoria, con la única intención de hacer de esta una lectura amena y no tan soporífera como acostumbran a ser los manuales técnicos. ¿Por qué no hacer un escrito para el “no iniciado” y dejarse llevar por la creatividad literato-científica?

¹ **ADVERTENCIA:** He optado por omitir algunos pasos preparativos en la configuración de la Raspberry (instalación del sistema, configuración wifi, instalación de paquetes, etc.) dando por concluido su aprendizaje y así ceñirme únicamente en el proyecto y sus dependencias.

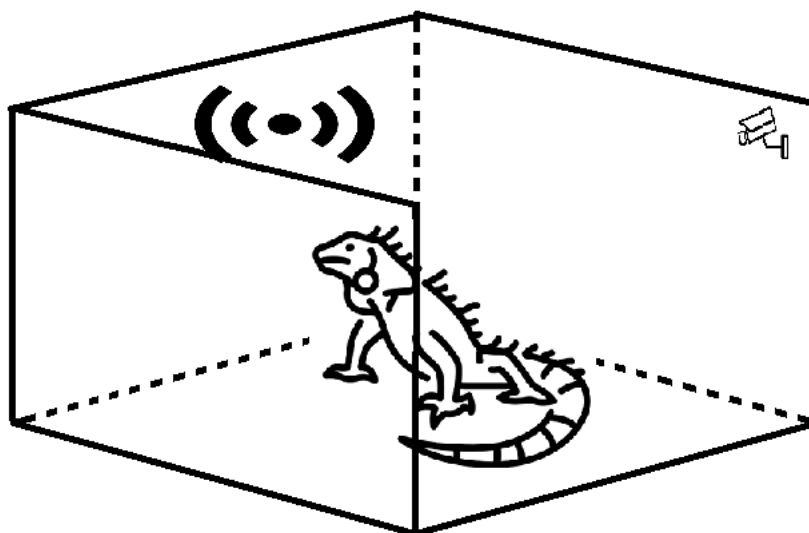
PRESENTACIÓN DEL PROYECTO_VISUALMOV (o PVM, Proyecto de Visualización Móvil)

Preguntarse “cómo” nace la idea de este proyecto es preguntarse entre todas las posibilidades disponibles en el kit de ELEGOO, su complejidad y, sobre todo, su temporalidad, la idea desde su más puro elemento formal hasta su materialización y funcionalidad final.

Cierto es que, entre todos los dispositivos disponibles, si había uno que atraía mi atención, ese era el sensor de movimiento (HC-SR501) el cual me llevó a la siguiente pregunta ¿por qué no activar un dispositivo ‘X’ cuando el sensor detecte “algo”? La clave o duda de mi encrucijada se hallaba en ese dispositivo ‘X’. Podía encender un led cuando detectase movimiento, lo tendré en cuenta; también una alarma, o mejor un motor que inicie su movimiento, entrando así en un bucle natural infinito –o círculo vicioso– entre el sensor de movimiento y motor que se mueve...

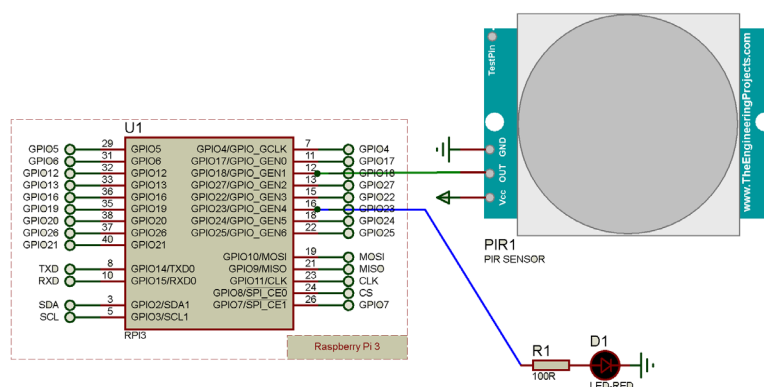
Llegado a este punto, nuevas interpelaciones alimentaron mi vacilación, ¿por qué no hacer algo funcional, práctico, que garantice su utilidad?, ¿qué dispositivo es el más apropiado, óptimo para el sensor de movimiento? La respuesta era obvia: una cámara o webcam: si algo entra, sale, ... se mueve, simplemente quiero verlo si no estoy presente. La cámara se activa, captura una imagen o graba un vídeo en el momento que el sensor detecta movimiento en el espacio que cubre. ¡Ahora sí!, la lista de posibilidades era finita con utilidades infinitas (cámara de seguridad o vigilancia que NO grabe 24h. seguidas; una casa para pájaros que por algún afortunado desenlace del destino anide algún pequeño gorrión o la errante golondrina hasta emprender su largo camino a cálidas regiones).

Para hacer más sofisticado el proyecto se analizaron los posibles soportes donde almacenar las imágenes/vídeos que fuera realizando la cámara. La primera opción era utilizar la misma memoria de la Raspberry: guardar los archivos en una carpeta y recuperarlos en el momento que el usuario final lo considere más oportuno. Pero la memoria del microprocesador es limitada y había que encontrar una solución alternativa. Y, ¿por qué no subirlo a la nube?, ¿y a la memoria del sistema creando una estructura de imágenes/día guardadas en carpetas generadas automáticamente el mismo día y que, transcurrido un mes, elimine automáticamente la carpeta con todo su contenido sin que el usuario final se preocupe del espacio disponible y a su vez, los archivos creados suban a una carpeta específica del Google Drive? ... Una idea que *a priori*, ambiciona más código que la suma de componentes físicos del proyecto. Ahora sí, parafraseando al cardenal protodiácono: “*Habemus Project*”.



ESTRUCTURA DEL PROYECTO Y PREPARACIÓN DEL ENTORNO

Diseño de una idea²



Por la falta de las librerías pertinentes de Proteus 8 Prof. en el esquema se muestra un sensor PIR de temperatura, adaptado para el diseño del circuito como sensor de movimiento HC-SR501.

En la imagen superior se observa el circuito diseñado con la demo **Proteus 8 Pro**. A su izquierda muestra el microprocesador Raspberry Pi 3 con 26 pines, cada uno de ellos enumerado y con una nomenclatura específica conocida técnica y comúnmente como GPIO junto a su número de identificación³, destinados para la recepción de datos o señales de control y la transmisión de órdenes de ejecución.

A la derecha de la placa, en la parte superior, se encuentra el dispositivo PIR (sensor de movimiento) con tres pines –GND, OUT, Vcc. Vcc es el pin de alimentación que está conectado a un potenciómetro; OUT, el pin que irá conectado directamente con el microprocesador (GPIO18) enviándole una señal de activación del sensor de control; GND es el pin de tierra –o masa–, que permitirá que se cierre el circuito.

En la parte inferior-derecha del circuito hay dos elementos más: Como elemento principal, el dispositivo LED. En segundo, una resistencia, que cumple un importante cometido sujeto a la ley de Ohm⁴, para evitar que el LED pueda quemarse al disponer de una resistencia directa baja. La resistencia está conectada con la RaspBerry mediante el GPIO23 y transmitirá la orden o instrucción emitida por la placa directamente a través de ella al LED, “on/off”. El LED, al igual que el sensor PIR, debe estar conectado a GND, 0 voltaje.

Hardware (elementos del circuito)

Idea en mente, para que el proyecto vea la luz y sea efectivo en su propósito, primero se realizará un pequeño inventario con todos los componentes e instrumentos necesarios para la confección del circuito.

NOTA: Verificar que todos y cada uno de los elementos necesarios para la elaboración del proyecto funcionan independientemente antes de iniciar el circuito. Pueden estar defectuosos de fábrica o por un mal uso, siendo la causa primera manifiesta del mal funcionamiento del circuito.

En el siguiente cuadro se detallan los dispositivos con sus especificaciones técnicas utilizados para la construcción del circuito IOT:

² No se hará referencia al código de Python en el circuito ni su funcionamiento. Estos serán explicados y demostrados en el apartado de Software.

³ Aunque el proyecto diseñado con Proteus no va a necesitar más de 2 pines, a partir de ahora se hará referencia a estos como GPIO para su identificación *a posteriori* con el código de Python (BCM).

⁴ Para una tensión dada, una baja resistencia permite una mayor corriente que una resistencia elevada.



Raspberry Pi 3 B+:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz.
- Wifi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11. b/g/n/ac, Bluetooth 4.2, BLE.
- RAM: 1GB LPDDR2 SDRAM.
- Pines GPIO (40); USB(x4); puerto HDMI; conector pantalla DSI; conector cámara CSI; micro USB (energía); Jack audio/vídeo 3,5mm; puerto ethernet RJ45.
- Dimensiones: 85,6mm x 56mm x 21mm.



Sensor de movimiento PIR HC-SR501:

- Dimensiones: 3,2cm x 2,4cm x 1,8cm.
- Circuitos de control incluidos en el módulo.
- Tiempo ajustable (min. 3s, máx. 5m)
- Distancia ajustable (min. 3mts, máx. 7mts)
- Voltaje: 4.5 – 20 V.
- Corriente: <60uA.
- Salida: Pulso lógico 3.3V.



Az-Delivery cámara para Raspberry Pi:

- Cámara de 5 megapíxeles de alta resolución.
- Filtros infrarrojos (filtro IR).
- Cable de cinta *Camera Serial Interface* (CSI).



Diodo LED:

- Patilla corta (cátodo) negativo (o masa).
- Patilla larga (ánodo) positivo (o paso de corriente) y emita luz.

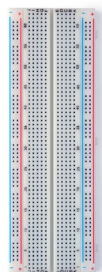


Resistencia 120 PCS de 220 – 300 Ω.



Breadboard Jumper Wire 65 PCS:

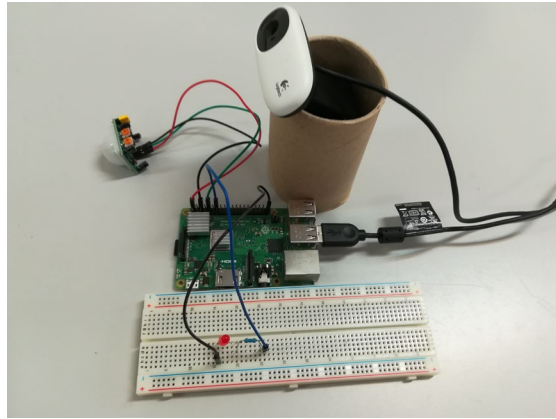
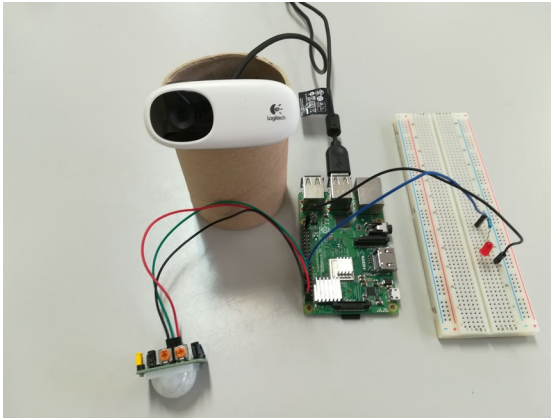
- macho/macho → para LED Data, GPIO23 (pin16), azul.
- macho/macho → para LED Ground, pin39, negro.
- macho/hembra → para sensor Data, GPIO18 (pin12), verde
- macho/hembra → para sensor PWR, pin2, rojo
- macho/hembra → para sensor Ground, pin6, negro.



Protoboard 830 Tie-Points Breadboard 1PC:

- Tapón transparente de 830 puntos de amarre en placa de pruebas sin soldadura ABS transparente.
- Plástico con leyenda de color.
- 1 IC / Área de circuito, 630 puntos de conexión.
- 2 tiras de distribución, 200 puntos de amarre.
- Tamaño: 6.5 x 2.2 x 0.3 pulgadas (165.1 mm x 54.6 mm x 8.5 mm).

El circuito



Ejemplo del prototipo del circuito propuesto. En este ejemplo se muestra la instalación de una webcam en el puerto USB de la Raspberry, optimizado en el proyecto final con una cámara con conexión CSI.

Como se puede ver en la imagen superior, el circuito responde al diseño creado anteriormente con Proteus y una webcam conectada al puerto USB de la Raspberry. En la protoboard se conecta el dispositivo lumínico junto a la resistencia que está conectada (cable azul) en el GPIO23 del procesador. Este pin alimentará de corriente y transmitirá la orden de *on/off* al LED. Para pasarle la masa –voltaje 0– y así cerrar el circuito, se conecta un cable (negro) a la patilla cátodo (-) del LED y en el pin 39 (Ground) de la Raspberry.

El sensor HC-SR501 se conectará directamente al microprocesador con los tres pines⁵ que dispone. El pin de GND del dispositivo se conectará al pin 6 (Ground-masa) de la placa; el pin Vcc (cable rojo) se conectará al pin 2 (5V PWR) para alimentar el sensor de corriente; finalmente, el pin OUTPUT irá conectado al pin 12 (GPIO18) de la Raspberry (cable verde).

Los reguladores de distancia, sensibilidad y tiempo del sensor de movimiento se han ajustado entre 2 y 5 minutos aproximadamente para su activación entre un ciclo y otro y la distancia se ha optado por la máxima posible (7m) aunque siempre dependerá del espacio disponible para su instalación final.



⁵ Ground, Output y VCC.

Instalación de la cámara

Una prueba con webcam (USB)

Para realizar las primeras pruebas con una cámara, antes de disponer de ella, se utilizó una webcam conectada a uno de los puertos USB disponibles en la RaspBerry. Al no estar conectada al puerto de cámara CSI (*Camera Serial Interface*) no se define la interface entre la webcam y el microprocesador anfitrión, sin poder configurarla desde **raspi-config** del sistema. Pero existen herramientas, o módulos de Debian (Linux), que se deben instalar para el uso y buen funcionamiento de estos dispositivos.

Antes de la instalación del programa se debe comprobar que el procesador reconoce la webcam conectada en el puerto. Para ello se ejecutará desde *pi@raspberrypi* la siguiente instrucción que mostrará los dispositivos conectados en los diferentes puertos USB existentes:

```
~$ lsusb
```

Una vez se ha comprobado la detección del dispositivo, el programa propuesto para el funcionamiento de la webcam es **fswebcam**⁶. Se instalará desde el sistema una vez actualizado. En *pi@raspberrypi* se escribirá el comando de instalación:

```
~$ sudo apt install fswebcam
```

Una vez que se ha instalado correctamente, se puede hacer una prueba de captura de imagen utilizando el siguiente comando⁷:

```
~$ fswebcam -p YUYV -d /dev/video0 -r 1280X720 IMAGEN/imagen.jpg
```

El atributo **'-p'** permite seleccionar el tipo o formato de archivo (*YUYV*); **'-d'** establece la fuente de uso, es decir el dispositivo conectado al USB (*/dev/video0*); **'-r'**, la resolución de la imagen (**1280X720**, dependerá de la webcam poder usar una u otra resolución); finalmente se indica el *path* –o ruta– de destino donde se guardará el archivo y, el nombre de éste con su formato-extensión **'IMAGEN/imagen.jpg'**.

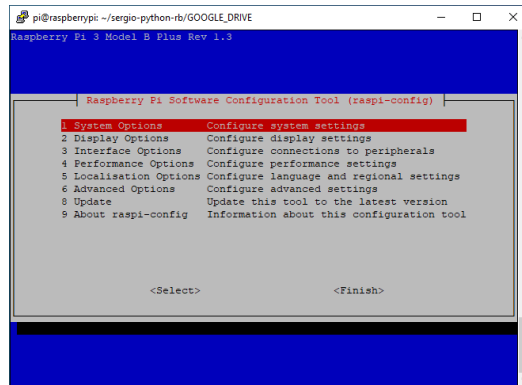
⁶ Existen otros programas en Linux que permiten la utilización de las webcams con la RaspBerry, uno de ellos es **Luvview**.

⁷ En la primera prueba que se realizó se utilizó el comando `$ fswebcam IMAGEN/imagen.jpg`, pero no funcionó sin especificar ninguno de los atributos disponibles, sobre todo aquel que permite definir el formato del archivo **'-p YUYV'**. Al no disponer de pruebas y por no reproducir nuevamente el error, no se dispone de documentación.

Instalación Az-Delivery cámara (CSI)

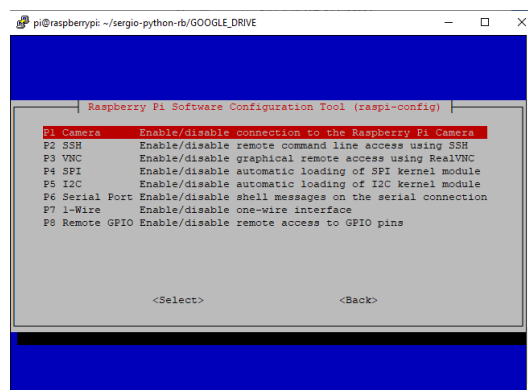
En este proyecto, la cámara utiliza la conexión con cable plano CSI (ver imagen). Una vez que se ha instalado correctamente el dispositivo se ha de configurar desde **raspi-config**. Para acceder a ésta interfaz de configuración del microprocesador, desde *pi@raspberrypi* se escribe:

```
~$ sudo raspi-config
```

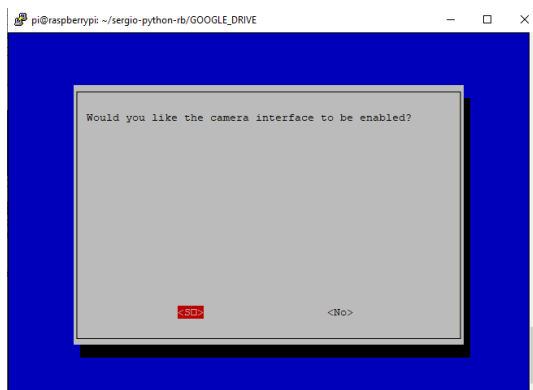


Una vez se ha accedido, seleccionar la primera opción, **1 System Options – Configure system settings**, que permitirá establecer la configuración del sistema.

Una vez dentro, mostrará el siguiente menú dónde se encuentra la cámara y otras herramientas de la RaspBerry dispuestos para su configuración, o detección. Seleccionar la opción **P1 Camera – Enable/disable connection to the Raspberry Pi Camera**.



La cámara se debe **Habilitar** para poder trabajar o disponer de ella en el microprocesador y trabajar con el proyecto. Seleccionar **<SO>** a la pregunta y de esta manera quedará habilitada la interfaz de la cámara.



Una vez finalizada la configuración o la habilitación de la cámara, para asegurar que los cambios se realizan satisfactoriamente se debe reiniciar la Raspberry. Desde el sistema ejecutamos el siguiente comando para desconectar el microprocesador,

```
~$ sudo init 0
```

o, simplemente se reinicia con el comando:

```
~$ sudo reboot
```

Una vez se ha reiniciado la Raspberry, ahora ya es posible probar si la interfaz **-raspistill-**, disponible para la grabación de vídeos o la captura de imágenes funciona correctamente. Para ello se ejecutará el siguiente comando:

```
~$ raspistill -o IMAGEN/newImage.jpg
```

Este programa dispone de atributos que permitirán la configuración de ciertos parámetros para ajustar las imágenes o los vídeos grabados. Entre todos ellos (ver `~$ raspistill -help`) merece hacer una mención especial a los atributos **'-vf'** y **'-hf'** que ajustará la orientación de la imagen captada.

Finalizada la instalación de la cámara y una vez montado el circuito, llega el momento de desarrollar el centro neurálgico del proyecto, el "cerebro" de toda acción: el software que se utilizará para que el procesador pueda recibir/transmitir ordenes desde -o a respectivamente-, los dispositivos utilizados.



Software (Python)⁸

Antes de entrar a explicar el código de Python *per se* y la importancia de su utilidad para el funcionamiento y control del circuito, se examinarán e instalarán las librerías necesarias para que el programa pueda cumplir con todos sus objetivos.

Instalación de librerías

picamera:

Éste módulo, nativo de Python, permitirá controlar la cámara instalada en el puerto CSI de la RaspBerry mediante código de programación sin necesidad de utilizar los subprocesos -o comandos- de otras aplicaciones (tipo Fsw webcam, Luvview, incluso la interfaz destinada a dicho propósito, Raspstill) controlables mediante la terminal del procesador -como sucede con las webcams conectadas a USB-, incluso desde el mismo código de Python.

NOTA: Como siempre, es recomendable y necesario actualizar la RaspBerry y así podrá encontrar la librería en los repositorios⁹.

Para la instalación de **picamera**, desde *pi@raspberrypi* se ejecutará el comando siguiente¹⁰:

```
~$ sudo apt-get install python3-picamera
```

También se puede realizar la instalación mediante el comando '**pip**' o '**pip3**' de python:

```
~$ pip3 install picamera
```

pyDrive:

Es una biblioteca que contiene *google-api-python-client* que simplifica las tareas comunes de la API de Google Drive, envolviendo la API en clases para cada recurso y hacer que el programa esté más orientado a objetos. También ayuda a operaciones comunes como crear una carpeta, subir o descargar un archivo, incluso la obtención de contenido y su posterior paginación.

La instalación se realizará, una vez más, desde la terminal del microprocesador:

```
~$ pip3 install PyDrive
```

Para trabajar con la API de Google Drive también es necesario instalar la versión de desarrollo actual de GitHub. Al igual que la instalación anterior de PyDrive, se ejecutará desde *pi@raspberrypi* el comando:

```
~$ pip3 install git + https://github.com/googledrive/PyDrive.git#egg=PyDrive
```

⁸ **NOTA IMPORTANTE:** Al no disponer de imágenes con orden cronológico que permitan visualizar el desarrollo y evolución del programa "Proyecto-VisualMov.py", el código se explicará de forma estructural –línea a línea– y por la implicación entre sus funciones.

⁹ Los comandos para actualizar el sistema de la RaspBerry Pi son: **\$ sudo apt-get update** y **\$ sudo apt-get upgrade**.

¹⁰ También es importante tener presente la versión de Python con la que se está trabajando. Recordar que, si se utiliza la versión 2, el comando será "*python*" mientras que con la última versión del lenguaje es "*python3*".

Código de programación (Proyecto-VisualMov.py)

Montado el circuito, conectada e instalada la cámara, más las dependencias o módulos de Python, falta una interfaz, un programa que permita al microprocesador (RaspBerry) recibir información del sensor PIR de movimiento (HC-SR501) cuando este se active; enviar dos órdenes: una al LED, que permanecerá encendido (1) mientras se ejecuta la segunda orden y el sensor no reciba señal; la segunda, activará la cámara conectada al puerto, disparará una secuencia de diez capturas que guardará en la memoria del sistema y, en caso que el microprocesador esté conectado a una red, ya sea por ethernet o wifi, subirá las imágenes a una carpeta seleccionada del Google Drive¹¹.

El código ha sido desarrollado con un editor que emula la codificación de muchos diferentes lenguajes de programación, y otros formatos (*NotePad++*), aunque no permite la ejecución. Dado que es una interfaz para trabajar con una RaspBerry, las pruebas han sido ejecutadas desde el cliente SSH (*PuTTY*) conectado al microprocesador.

Importar librerías

```
1 import RPi.GPIO as GPIO
2 import time
3 import picamera
4 import os
5 from datetime import datetime
6 from pydrive.auth import GoogleAuth
7 from pydrive.drive import GoogleDrive
```

RPi.GPIO: configuración de los pines de la RaspBerry.

picamera: control de la cámara.

pydrive (GoogleAuth): para trabajar con cliente de *OAuth* en un proyecto de *Google Cloud Platform*.

pydrive (GoogleDrive): para trabajar con las *APIS* de *Google Drive* y subir archivos a la nube.

datetime: se usará información del sistema para crear las diferentes carpetas y archivos

os: se utilizarán comandos de *Linux* para la creación y eliminación de carpetas y archivos (imágenes) guardados en la memoria del sistema.

time: para ralentizar las tareas del IOT.

¹¹ La cuenta dónde el prototipo envía las imágenes pertenece a mi correo ssantinhpython@gmail.com.

Configuración GPIO de periféricos

```
10 pinPir = 18
11 pinLed = 23
12
13
14 def config_periferico():
15     GPIO.setmode(GPIO.BCM)
16     GPIO.setwarnings(False)
17     GPIO.setup(pinPir, GPIO.IN)
18     GPIO.setup(pinLed, GPIO.OUT)
19
```

Los pines¹² del microprocesador serán reconocidos por la aplicación por el modo **BCM**. Este proyecto utilizará únicamente dos GPIO: **GPIO18** (variable **pinPir**) asumirá la detección de movimiento a través del sensor instalado en el circuito. Como es un dispositivo que envía una señal a la RaspBerry se debe configurar como **GPIO.IN**. El segundo pin que se ha de configurar es **GPIO23** (variable **pinLed**), este recibirá una orden del procesador para que se “encienda”, por ello se debe configura como **GPIO.OUT**.

Recuperar información del S.O.

```
21 def recupera_datetime():
22     ahora = datetime.now()
23     dia = ahora.day
24     mes = ahora.month
25     ano = ahora.year
26     hora = ahora.hour
27     minuto = ahora.minute
28     segundo = ahora.second
29
30     return dia, mes, ano, hora, minuto, segundo
31
```

Esta función¹³ utiliza la librería **datetime**. Almacena datos en variables independientes que serán retornadas en el momento que se llame a la subrutina en el mismo código.

Generar carpetas y nombres de archivo

La creación de carpetas y su tratamiento ha sido, quizás, el análisis y controversia más destacado en el planteamiento final del proyecto. La idea inicial era la implementación de una carpeta creada a mano (IMÁGENES) por el mismo usuario en la unidad donde está el programa *.py. Aunque no era una idea en balde, en la memoria del microprocesador habría una carpeta con tantas imágenes como actividad recursiva tuviera el programa y al recuperarlas el usuario encontraría cierto desconcierto de archivos.

¿Cómo construir las carpetas y los nombres de archivo?... La idea de coger datos del sistema (día, mes, año, hora, minutos y segundos) para construir una cadena como nombre de los archivos y las carpetas iba tomando forma, y así crear una estructura de carpetas (una carpeta cada día) y de ficheros en cada momento (hora, minuto y segundo) que la cámara realizara una captura, ordenando todo su trabajo.

```
73 def generar_carpeta_archivo():
74     d, me, a, h, mi, s = recupera_datetime()
75     n_c = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d, me, a)
76     r_a = "{}/PVMImg{:02d}{:02d}{:02d}{:02d}{:02d}.jpg".format(n_c, d, me, a, h, mi, s)
77     genera_carpeta = "mkdir {}".format(n_c)
78
79     if os.path.isdir(n_c):
80         pass
81     else:
82         os.system(genera_carpeta)
83         print("... La carpeta '{}' se ha creado satisfactoriamente.".format(n_c))
84
85     return r_a
86
```

¹² A partir de éste momento y durante la lectura del código de Python, se hará alusión a los pines o GPIO con el nombre de la variable: **pinPir** >> GPIO18, **pinLed** >> GPIO23.

¹³ Su importante propósito se explicará más adelante con la creación y borrado de carpetas.

En el código, la variable `'n_c'` contiene la concatenación de la cadena `"PVM_IMAGES"` junto a las variables `'d'` (día), `'me'` (mes), `'a'` (año)¹⁴ formando el nombre de la carpeta.

La variable `'r_a'`, contiene el path, el nombre del archivo y su extensión (la imagen capturada). El path, o la ruta, es el nombre de la carpeta más el signo `"/"` para indicar dónde se guardará el archivo. El nombre del fichero se formará con la concatenación de `"PVMImg"`, las variables `'d'`, `'me'`, `'a'`, `'h'` (hora), `'mi'` (minuto), `'s'` (segundo), el signo `"."` y el formato del archivo¹⁵.

La condición `os.path.isdir(n_c)` indicará si la carpeta que se va a crear existe. Si la respuesta es `'False'` creará la carpeta con el comando `'mkdir'` del módulo importado `os`.

Por último, la función devolverá el contenido de la ruta y el fichero cada vez que se la llame.

Borrar una carpeta

¿Cuál es la necesidad de borrar una carpeta y su contenido? Ante la capacidad finita que dispone la memoria de la Raspberry, había que ingeniar una manera de eliminar carpeta y ficheros, dando margen al usuario para que recuperase los datos¹⁶.

```
33 def borrar_carpeta(carpeta):
34     borrar_carpeta_OS = "rm -r {}".format(carpeta)
35
36     if os.path.isdir(carpeta):
37         os.system(borrar_carpeta_OS)
38         print("... La carpeta '{}' ha sido eliminada satisfactoriamente.".format(carpeta))
39         time.sleep(2)
40     else:
41         print("... No se ha encontrado la carpeta '{}' para su eliminación.".format(carpeta))
42         time.sleep(2)
43
```

Borrar una carpeta con un argumento lógico como la necesidad de “eliminar para liberar espacio”, no significa realmente un problema en cuanto al desarrollo del código. La función recibe el nombre de la carpeta que eliminará como parámetro; si ésta existe, `os.path.isdir(carpeta)`, ejecutará el comando `'rm -r'` de `os` y eliminará la carpeta y todo su contenido.

Criterios para eliminar una carpeta

Quizás la pregunta exacta, o más apropiada, no era ¿por qué borrar una carpeta? Más bien, el tener que borrar una carpeta es una premisa, pues es obvio que, llegado el momento, por la falta de espacio la aplicación se detendría. La verdadera pregunta que debía encontrar una respuesta lógica es ¿Qué carpeta se debe eliminar?

Los criterios adoptados para eliminar una carpeta, dando margen al usuario que pueda recuperar sus imágenes, están pensados para conservar los archivos durante un mes, día más-día menos, es decir, la carpeta creada el 1 de enero del 2021 (`PVM_IMAGES01-01-2021`) se eliminará el 1 de febrero del 2021. Y aquí empieza un pequeño elenco de criterios.

¹⁴ Los datos de las variables (d, me, a, h, mi, s) se obtendrán del retorno de los valores obtenidos de la función `recuperar_datetime` que se actualiza cada vez que se la llama. Ver “Recuperar información del S.O.”

¹⁵ Las imágenes guardadas tendrán formato `jpg`.

¹⁶ La idea de incorporar Google Drive nunca fue la primera opción para almacenar las imágenes. En caso que el microprocesador no estuviera conectado a la red no podría subir los momentos captados por la cámara.

```

45 def eliminar_carpeta():
46     d, me, a, h, mi, s = recupera_datetime()
47
48     if me == 1:
49         m_anterior = 12
50         a_anterior = a - 1
51         b_carpeta = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d, m_anterior, a_anterior)
52         borrar_carpeta(b_carpeta)
53     elif (me == 3 and d == 2):
54         d_enero = [29, 30, 31]
55         for d_e in d_enero:
56             b_carpeta_e = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d_e, 1, a)
57             borrar_carpeta(b_carpeta_e)
58         b_carpeta_f = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d, 2, a)
59         borrar_carpeta(b_carpeta_f)
60     elif ((me == 4 or me == 6 or me == 9 or me == 11) and (d == 30)):
61         m_anterior = me - 1
62         b_carpeta1 = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d, m_anterior, a)
63         b_carpeta2 = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(31, m_anterior, a)
64         print("... Se borrarán las carpetas del día 30 y 31 del mes anterior.")
65         borrar_carpeta(b_carpeta1)
66         borrar_carpeta(b_carpeta2)
67     else:
68         m_anterior = me - 1
69         b_carpeta = "PVM_IMAGES{:02d}-{:02d}-{:02d}".format(d, m_anterior, a)
70         borrar_carpeta(b_carpeta)
71

```

La función *eliminar_carpeta* se ejecutará cada día que se lance la aplicación. Decidirá si una carpeta dada según el día, mes y año debe ser eliminada aplicando todos los criterios condensados en la estructura *if-elif-else*, en caso que ésta exista.

El código no busca una carpeta concreta. Una vez más se llama a la función *eliminar_carpeta* para construir el nombre de la carpeta que se quiere eliminar contenido en la variable '**b_carpeta**'.

1º criterio: si el mes actual es enero, se deben borrar las carpetas del mes de diciembre (**m_anterior = 12**) del año anterior (**a_anterior = a - 1**). Estos datos modificados permitirán construir el nombre de las carpetas que se debe borrar pertenecientes al mes de diciembre del año pasado.

2º criterio: ¿cómo borro las carpetas de enero de los días 29, 30 y 31 si febrero solo tiene 28 días? La fecha por defecto '2 de marzo' se eliminarán todas estas carpetas de enero además de la carpeta creada el 2 de febrero. Para evitar la reiteración del mismo código, una lista con tres índices contendrá el valor de estos días. Un bucle *for* recorrerá la lista '**d_e**' para eliminar las carpetas.

3º criterio: éste tercer supuesto implica a todos aquellos meses que tienen 30 días. ¿Cuándo borrar la carpeta del día 31 del mes anterior? La respuesta, el mismo día 30.

Estos tres criterios mentados se pueden considerar en un apartado de "supuestos especiales". El **4º criterio** simplemente eliminará la carpeta creada el día presente del mes anterior, como se indica en el segundo párrafo de este apartado.

¡Algo se mueve!

```

122 def comprobar_movimiento():
123
124     if GPIO.input(pinPir):
125         print("... Sensor HC-SR501: Movimiento detectado.")
126         GPIO.output(pinLed, GPIO.HIGH)
127         disparo_camara()
128         GPIO.output(pinLed, GPIO.LOW)
129     else:
130         print("... Sensor HC-SR501: No se detecta movimiento.")
131

```

Esta función, de sencillo código y comprensión, es la que recupera la señal emitida por el sensor PIR. La condición *if GPIO.input(pinPir)*, en caso que detecte un pico de radiación mediante los infrarojos, enviará una orden al LED, *GPIO.output(pinLed, GPIO.HIGH)*, para que se encienda mientras siga activo. La otra orden es la captura de la imagen de la cámara, **disparo_camara**. Finalmente, cuando no es detectado ningún movimiento, el led volverá a su estado de reposo **0**.

Shoot again!

```
99 def disparo_camara():
100     with picamera.PiCamera() as mi_imagen:
101         mi_imagen.start_preview()
102         cont_imagen = 0
103         while cont_imagen < 10:
104             nombre_archivo = generar_carpeta_archivo()
105             print("... Capturando imagen en -> {}".format(nombre_archivo))
106             mi_imagen.capture(nombre_archivo)
107             cont_imagen += 1
108             time.sleep(0.5)
109             try:
110                 subir_googleDrive(nombre_archivo)
111                 print("... Archivo subido a Google Drive.")
112             except Exception as e:
113                 print("... No se ha completado la subida del archivo.")
114                 print("... ERROR: {}".format(e))
115
116     mi_imagen.stop_preview()
117     mi_imagen.close()
118
```

Esta función **disparo_camara**, junto a la anterior **comprobar_movimiento**, son la esencia del proyecto VisualMov, la idea original contenedora de todo un ejercicio de reflexión, desarrollo y evolución. En este caso, y como mayor innovación hasta ahora con Python, permite enviar ordenes a la cámara instalada en la Raspberry, bien por USB¹⁷, o como en el proyecto, por medio de la conexión CSI.

Para poder capturar/guardar la imagen, se recupera el *path* y el nombre del archivo, `nombre_archivo = generar_carpeta_archivo`.

Creada la instancia con el módulo `picamera`, `with picamera.PiCamera() as mi_imagen`, antes de por utilizar el dispositivo, la cámara debe ser iniciada, `mi_imagen.start_preview()`. La variable '`cont_imagen`' es un contador de disparos realizados por la cámara. Su cometido como variable de control en el bucle (`while`) consiste en contabilizar las imágenes y, superadas las 10 fotografías se detendrá la función. Para capturar una imagen con `pycamera`, Python utiliza el método `mi_imagen.capture(nombre_archivo)`.

Cuando la función llega a su fin, la cámara no puede quedar activa y se debe apagar. Los métodos utilizados simultáneamente `mi_image.stop_preview()` y `mi_imagen.close()` permitirán que el programa detenga y cierre la cámara del microprocesador respectivamente.

Un viaje por la nube

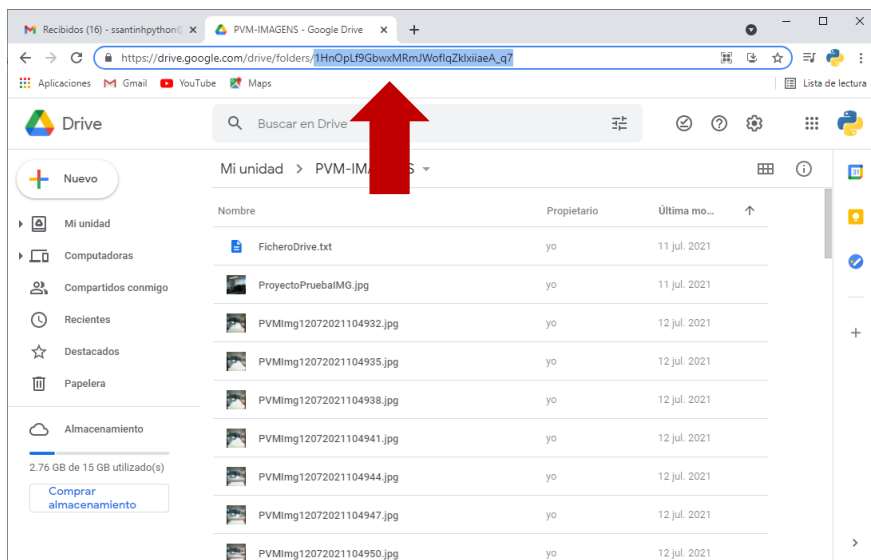
```
88 def subir_googleDrive(ruta_archivo):
89     id_carpetaDrive = "1HnOpLf9GbwXMRmJWofIqZklxiiAeA_q7"
90     autenticacion = GoogleAuth()
91     autenticacion.LocalWebserverAuth()
92     drive = GoogleDrive(autenticacion)
93     archivoDrive = drive.CreateFile({'parents': [{'kind': 'drive#fileLink', 'id': id_carpetaDrive}]})
94     archivoDrive['title'] = ruta_archivo.split('/')[-1]
95     archivoDrive.SetContentFile(ruta_archivo)
96     archivoDrive.Upload()
97
```

A partir de éste momento la idea inicial de este proyecto se puede decir que ha llegado a su fin. Para que el proyecto tuviera ciertas pinceladas de mayor sofisticación se pensó la posibilidad de ver la actividad de la Raspberry *in situ*. La inclusión del Google Drive al proyecto permite que éste responda a las proposiciones de IOT, el "internet de las cosas" y hacer del ejercicio un instrumento más atractivo de cara a una posible producción en masa.

La función `subir_googleDrive` recibe un parámetro con los datos de la ruta y el nombre del archivo (captura de imagen) que va a ser publicado a una carpeta específica de Google Drive. La variable '`id_carpetaDrive`'

¹⁷ Recordar que el módulo `picamera` de python solo es compatible con cámaras conectadas a CSI. Para las webcams conectadas por USB se debe hacer uso de aplicaciones como `Fswebcam` y sus comandos de control.

contiene el **id** de la carpeta. Este id es el que muestra la *url* de la carpeta de Google Drive una vez se accede a su contenido.



Siguiendo con el código, cabe destacar dos partes bien diferenciadas de esta función: una, se crea una instancia de **GoogleAuth** que permita que el programa acceda a las dependencias de la API de Google Drive con las credenciales generadas en el proyecto de **Google Cloud Platform**¹⁸. Para recuperar dichas credenciales o 'token' de la librería de **pyDrive**, Python utiliza el método *autenticacion.LocalWebServerAuth*, que se pasará a la instancia 'drive' creada con las librerías **pyDrive**, *drive = GoogleDrive(autenticacion)*.

La segunda parte concierne a la creación del archivo o imagen que se subirá a la carpeta creada expresamente en Google Drive para visualizar los resultados del proyecto. La línea de código *archivoDrive = drive.CreateFile({'parents': [{'kind': 'drive#fileLink', 'id': id_carpetaDrive}]})*, contiene la creación del fichero. Se le pasará como parámetros el tipo o clase de archivo, '*drive#fileLink*' y el identificador de la carpeta de destino del Drive que se encuentra en la variable '*id_carpetaDrive*'.

Para indicar la ruta donde está el archivo que se desea publicar en internet, *archivoDrive['title'] = ruta_archivo.split('/')[-1]*, del dato recuperado como parámetro '*ruta_archivo*', la función *split* se encarga de 'separar' dicho *path* de origen y el archivo de imagen pendiente de subir a la nube (ej.: *[path]/[nombre_archivo].[ext]*). Se indica dónde está el fichero preparado para su publicación con el método *archivoDrive.SetContentFile(ruta_archivo)* dentro de la del sistema y, finalmente se sube carga el archivo para su 'paseo por la nube', *archivoDrive.Upload()*.

¹⁸ Ver la creación de un nuevo proyecto en Google Cloud Platform, indispensable para trabajar con la API de Drive.

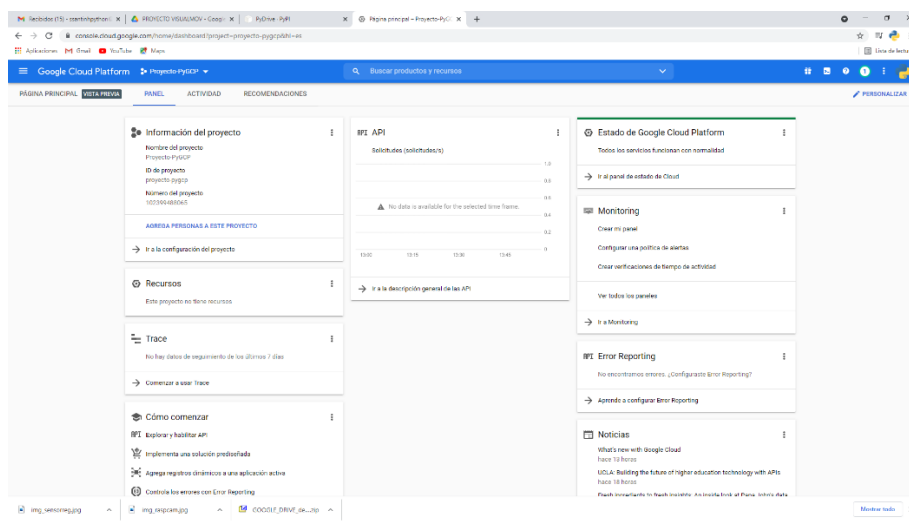
Google Drive, la API

Como se ha ido avanzando a lo largo de este documento, trabajar con las API's de otras aplicaciones no solamente implica el hecho de importar sus librerías junto a todas sus dependencias. Es importante crear un proyecto vinculado o autenticado con el programa de Python, o cualquier otro lenguaje de programación con el que se desee desarrollar una app.

En este apartado se verán aquellos códigos adicionales, como **Quickstart.py** y **settings.yaml**, y como se generan los ficheros con extensión ***.json**¹⁹, contenedores del id de cliente del proyecto de Google Cloud, el secreto de cliente, *token*, etc.

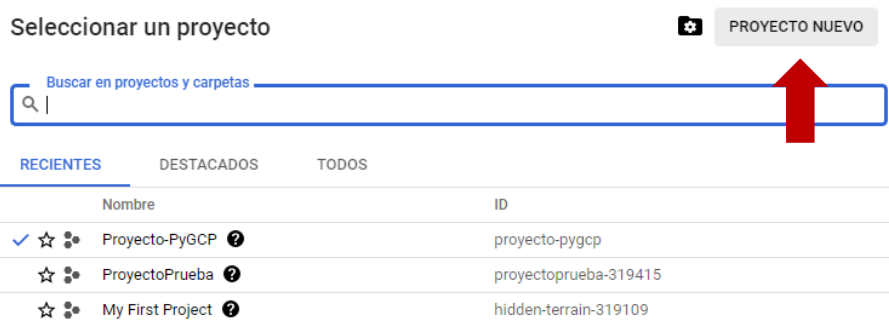
Proyecto Google Cloud Platform²⁰

Cuando se accede a la consola de desarrollador de GCP²¹ aparece la pantalla 'Panel' con un resumen de las actividades recientes con las que se ha ido trabajando.



En '**Seleccionar un proyecto**' aparece una lista con todos los proyectos creados hasta el momento. En el caso que compete, se ha de crear un nuevo ejercicio y se presionará '**Proyecto Nuevo**'.

Seleccionar un proyecto



¹⁹ **client_secrets.json** y **credentials.json**

²⁰ <https://cloud.google.com/gcp/?...>

²¹ Se hará alusión a Google Cloud Platform con el acrónimo de GCP.

En la ventana de **'Proyecto nuevo'** se introducirá el nombre del proyecto como dato obligatorio. Como es un trabajo personal que no pertenece a red, organización ni empresa, la opción de **'Ubicación'** se puede omitir y se presiona **'CREAR'**.

Proyecto nuevo

Tienes 4 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. [Más información](#)

[MANAGE QUOTAS](#)

Nombre del proyecto *

MiProyecto

ID de proyecto: miproyecto-319812. No se podrá cambiar más tarde. [EDITAR](#)

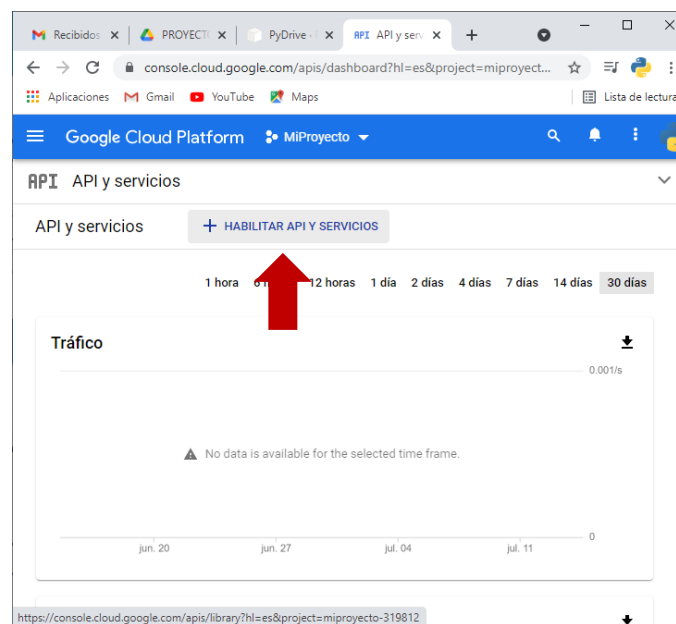
Ubicación *

Sin organización [EXPLORAR](#)

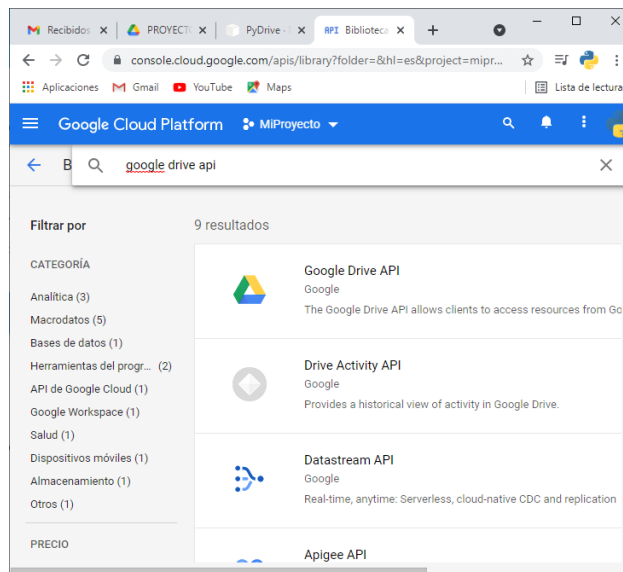
Organización o carpeta superior

[CREAR](#) [CANCELAR](#)

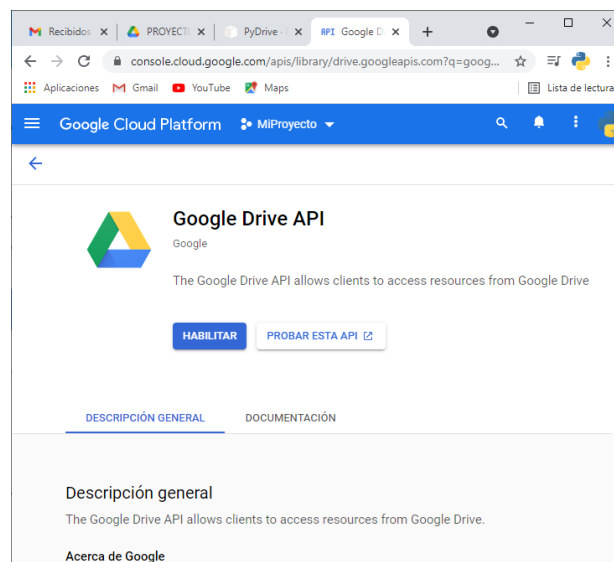
Creado el nuevo proyecto, el siguiente paso será habilitar las APIs de Google Drive. En el menú principal seleccionar la opción de **'API y Servicios'**. Presionado el botón de **'+ HABILITAR API Y SERVICIOS'** mostrará una ventana con la relación de todas las API existentes.



Como se ha realizado una búsqueda exacta, del resultado obtenido seleccionar **‘Google Drive API’**.

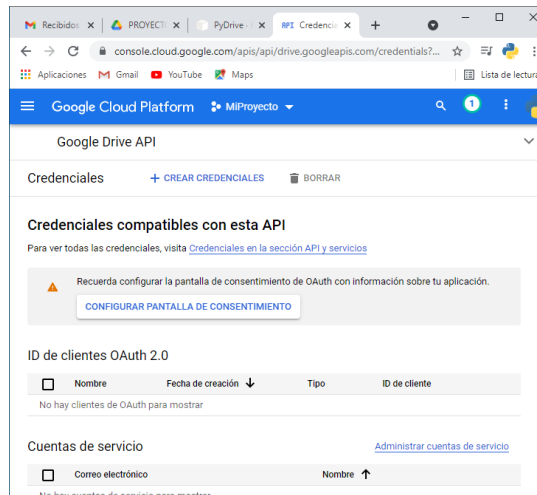


En la pantalla de la API de Google Drive²², presionar **‘HABILITAR’**.

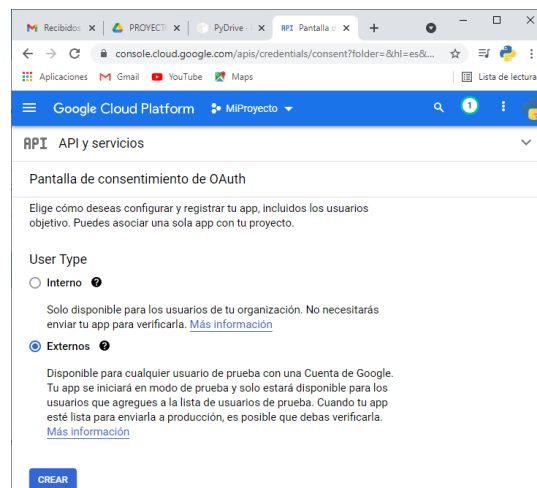


²² Para obtener mayor documentación sobre esta API y sus posibilidades, presionar sobre **‘DOCUMENTACIÓN’**.

El siguiente paso en la creación de un nuevo proyecto *Cloud* es la creación de las credenciales. Pero antes Google Cloud Platform muestra una ventana de aviso con la obligatoriedad de **‘CONFIGURAR PANTALLA DE CONSENTIMIENTO’**.



En la primera ventana **‘Pantalla de consentimiento de OAuth’** se debe indicar que tipo de usuario es, *Interno* (pertenecer a una organización) o *Externo* (agente libre). En este ejemplo de prueba se selecciona **‘Externo’** y finalmente presionar **‘CREAR’**.



La siguiente ventana es **‘Editar el registro de la app’** donde se almacenan cierta **‘Información de la aplicación’** y **‘Dominio de la app’**. En Información de la aplicación los datos que se introducirán son **‘Nombre de la aplicación’**, **‘Correo electrónico de asistencia del usuario’**. Por ahora, y mientras no se patente el proyecto, se prescindirá del **‘logotipo de la app’**.

Google Cloud Platform

RPI API y servicios

Editar el registro de la app

1 Pantalla de consentimiento de OAuth — 2 Permisos — 3 Usuarios de prueba — 4 Resumen

Información de la aplicación

Esta información aparece en la pantalla de consentimiento y permite que los usuarios finales sepan quién eres y cómo comunicarse contigo

Nombre de la aplicación *

MiAplicacion

El nombre de la aplicación que solicita el consentimiento

Correo electrónico de asistencia del usuario *

ssantinipython@gmail.com

Para que los usuarios se comuniquen contigo si tienen preguntas sobre su consentimiento

Logotipo de la app

EXPLORAR

Sube una imagen con un tamaño máximo de 1 MB en la pantalla de consentimiento que ayudará a los usuarios a reconocer tu app. Los formatos de imagen permitidos son JPG, PNG y BMP. Para obtener los mejores resultados, los logotipos deben ser cuadrados y de 120 x 120 px.

En Dominio de la app, aunque no es obligatorio, se introducirán los datos requeridos: **‘Página principal de la aplicación’**, **‘Vínculo a la Política de Privacidad de la aplicación’** y **‘Vínculo a las Condiciones del Servicio de la aplicación’**. En todas ellas se ha copiado la url de la consola de usuario de GCP. Finalmente, en la edición del registro de la app, en **‘Dominios autorizados’** se indicará la página de **‘google.com’**.

Google Cloud Platform

RPI API y servicios

Editar el registro de la app

Dominio de la app

Para protegerlos a ti y a tus usuarios, Google solo permite que las apps que usan OAuth puedan emplear los dominios autorizados. Se mostrará la siguiente información a los usuarios en la pantalla de consentimiento.

Página principal de la aplicación

<https://console.cloud.google.com/apis/credentials/consent/editNewAppInternalUser>

Proporciona a los usuarios un vínculo a tu página principal

Vínculo a la Política de Privacidad de la aplicación

<https://console.cloud.google.com/apis/credentials/consent/editNewAppInternalUser>

Proporciona a los usuarios un vínculo a tu página pública de Política de Privacidad

Vínculo a las Condiciones del Servicio de la aplicación

<https://console.cloud.google.com/apis/credentials/consent/editNewAppInternalUser>

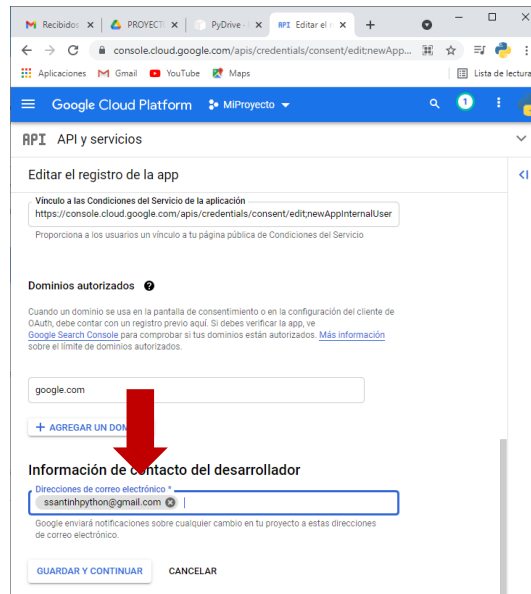
Proporciona a los usuarios un vínculo a tu página pública de Condiciones del Servicio

Dominios autorizados

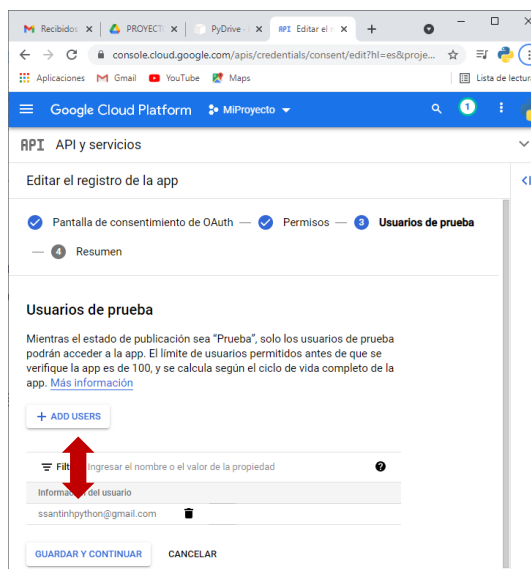
Cuando un dominio se usa en la pantalla de consentimiento o en la configuración del cliente de OAuth, debe contar con un registro previo aquí. Si debes verificar la app, ve [Google Search Console](#) para comprobar si tus dominios están autorizados. [Más información](#) sobre el límite de dominios autorizados.

google.com

Continuando en la misma página, en la sección de **‘Información de contacto del desarrollador’** se debe introducir la dirección de correo electrónico de la persona/as que han diseñado el proyecto. Una vez completados estos datos se presiona el botón de **‘GUARDAR Y CONTINUAR’**.

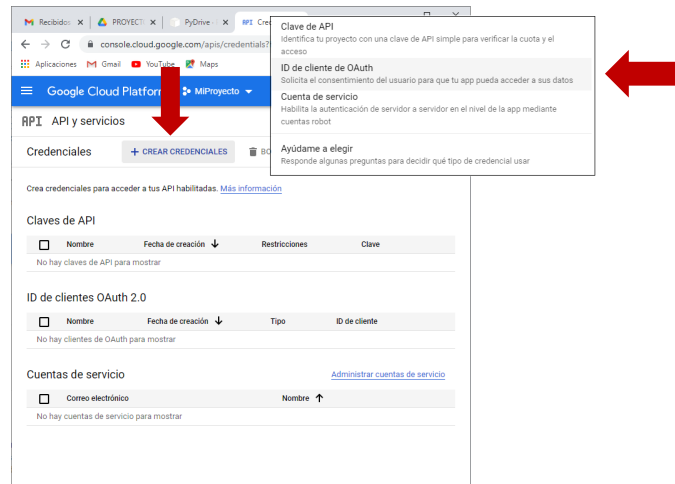


El paso siguiente (2) será indicar los permisos. Por ahora, para este proyecto iniciado, no se precisa la concesión de permisos adicionales, así que serán omitidas las pantallas de los permisos dando paso a los **‘Usuarios de prueba’**. Como el único usuario es el que suscribe, únicamente se indicará la dirección de correo electrónico del programador. Para ello, presionar **‘+ ADD USERS’** e introducir el correo electrónico del desarrollador. Una vez introducido, sin incorporar ningún usuario más, presionar **‘GUARDAR Y CONTINUAR’**

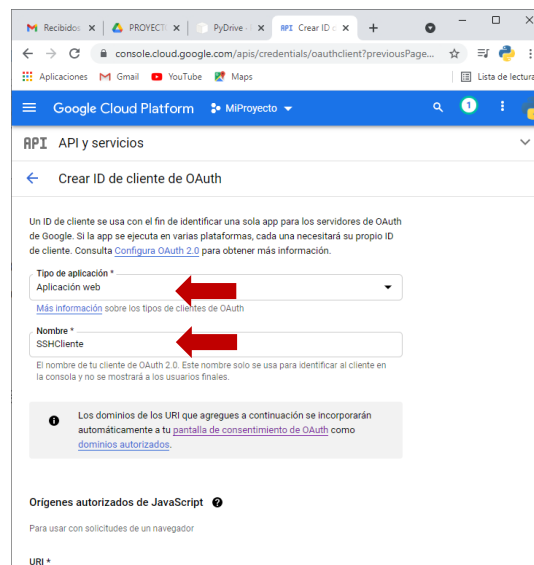


Finalmente, se mostrará un resumen con toda la información detallada de la pantalla de consentimiento de OAuth. En caso de necesidad de realizar algún tipo de modificación, se podrá acceder más adelante y editar dichos datos. Presionar **‘VOLVER AL PANEL’**, ahora es el momento de crear las credenciales.

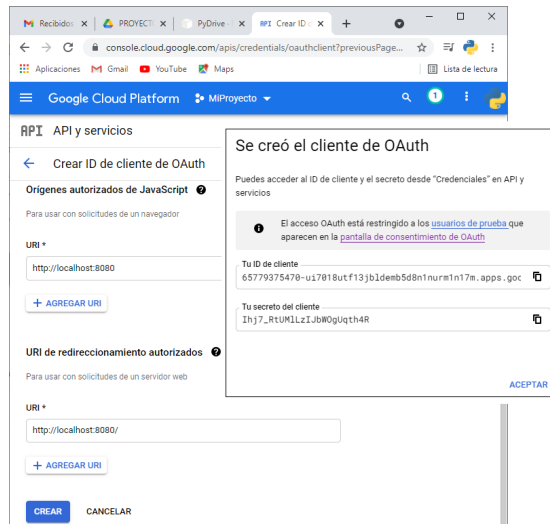
Las credenciales son autorizaciones, identificaciones que el código de la aplicación necesita para acceder y trabajar con el Google Drive. Al tratarse de un nuevo proyecto, obviamente no aparecerá ninguna. Presionar sobre **'+ CREAR CREDENCIALES'** y seleccionar **'ID de cliente de OAuth'**.



Al crear un nuevo cliente OAuth, se debe cumplimentar debidamente los datos que solicitan las credenciales, son campos obligatorios. En **'Tipo de aplicación'**, seleccionar del menú desplegable **'Aplicación web'**. En **'Nombre'** se indicará el nombre de cliente OAuth que seleccione el programador/usuario.



Siguiendo la página de creación de las credenciales hacia abajo, el siguiente dato que solicita es **'Orígenes autorizados de JavaScript'**, en la URL escribir **'http://localhost:8080'**. En **'URL de redireccionamiento autorizados'** se indicará la misma URL anteriormente citada, con barra final, **'http://localhost:8080/'**. Finalmente, se presiona **'CREAR'** y ya se dispone del cliente OAuth.



Creadas las credenciales, estas se pueden descargar en un archivo que será renombrado a 'client_secrets.json'. Pero falta crear el archivo de 'credentials.json'. Este fichero que contiene las autorizaciones pertinentes para que el programa desarrollado pueda conectar con el Drive de Google se genera automáticamente tras la ejecución del archivo 'quickstart.py' que se mostrará a continuación.

Quickstart.py

```
1 from pydrive.auth import GoogleAuth
2 from pydrive.drive import GoogleDrive
3 gauth = GoogleAuth()
4 gauth.LocalWebserverAuth()
5 gauth.LoadCredentialsFile(gauth)
6
```

La ejecución de este código permite realizar la petición al cliente de OAuth y a la unidad de Google Drive cargando el archivo de credenciales. Devuelve una URL que donde se ha de re-direccionar al proyecto de GCP en un entorno gráfico. Después de introducir la dirección obtenida, se indicará la cuenta de correo del usuario y 'continuar', de esta manera se la autorización estará completada.

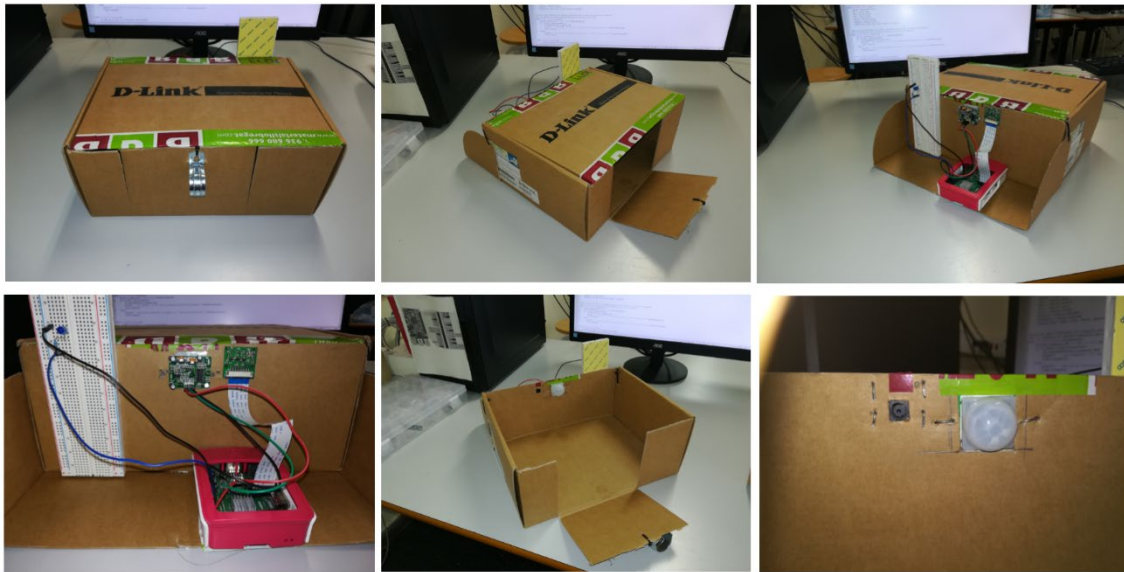
Settings.yaml

```
1 client_config_backend: settings
2 client_config:
3   client_id: 102399488065-08nhheatr0n153n33m2u8dq60k6naab9.apps.googleusercontent.com
4   client_secret: ccEf00b99pXCg71QDlDXrvDl
5
6 save_credentials: True
7 save_credentials_backend: file
8 save_credentials_file: credentials.json
9
10 get_refresh_token: True
11
12 oauth_scope:
13   - https://www.googleapis.com/auth/drive
14
```

Este archivo es de configuración que graba las credenciales del archivo generado tras la ejecución de 'quickstart.py' permitiendo que nunca caduquen. Únicamente se debe cambiar los datos de 'client_id' y 'client_secret' (cliente OAuth) por los obtenidos en la creación de credenciales de CGP o en el archivo descargado anteriormente 'client_secrets.json' y, volver a ejecutar 'quickstart.py' siguiendo los mismos pasos anteriores tras su ejecución.

NOTA IMPORTANTE: todos estos archivos deben estar en la misma carpeta donde se encuentra el programa Proyecto-VisualMov.py para que en su ejecución no se produzcan errores de autorización.

Un paseo por el taller de la creatividad. La caja (sin palabras) ...



Un ejemplo de ejecución del código desde PuTTY

```
pi@raspberrypi: ~/sergio-python-rb/GOOGLE_DRIVE
pi@raspberrypi:~/sergio-python-rb/GOOGLE_DRIVE $ LS
-bash: LS: orden no encontrada
pi@raspberrypi:~/sergio-python-rb/GOOGLE_DRIVE $ ls
build          Proyecto-VisualMov.py      PVM_IMAGES12-07-2021
client_secrets.json  Proyecto-VisualMov.spec   __pycache__
credentials.json    PVM_IMAGES09-07-2021     quickstart.py
dist             PVM_IMAGES12-06-2021     settings.yaml
pi@raspberrypi:~/sergio-python-rb/GOOGLE_DRIVE $ python3 Proyecto-VisualMov.py
... La carpeta 'PVM_IMAGES12-06-2021' ha sido eliminada satisfactoriamente.
... Sensor HC-SR501: Esperando resolución.
... Sensor HC-SR501: Movimiento detectado.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140230.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140230.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140233.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140235.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140237.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140239.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140242.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140244.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140246.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140249.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140251.jpg.
... Sensor HC-SR501: Esperando resolución.
... Sensor HC-SR501: Movimiento detectado.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140255.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140255.jpg.
... Capturando imagen en -> 'PVM_IMAGES12-07-2021/PVMImg12072021140258.jpg.
^C... Fin de la ejecución.
```

CONCLUSIONES

Desde el momento que se gestó la idea de este proyecto, siempre se ha planteado la posibilidad de hacer un “dispositivo de dispositivos”, es decir, la inclusión de cuantos más mecanismos mejor, como por ejemplo un motor “step by step” que permita hacer fotografías panorámicas - ¡Gràcies Joan!. Pero, entre montar el circuito, comprobar previamente que todo funciona bien, desarrollar el código, probarlo, corregir errores, optimizarlo y, finalmente, encontrar los recursos necesarios para montar un espacio, una estructura, una maqueta, una plataforma o base, y finalmente, “la caja”, para embellecerlo, el tiempo transcurría, siendo consciente que todo puede cambiar a mejor, evolucionar... pero mi cabeza me decía: ¡no le des la espalda a Murphy²³!

En líneas generales el proyecto VisualMov funciona como es debido, incluso los casos de posibles errores de la aplicación, como por ejemplo la subida de un archivo a internet sin conexión a red, están controlados impidiendo que se interrumpa la ejecución del programa.

Entre los dispositivos del circuito, el sensor HC-SR501, aunque permita regular la sensibilidad y la distancia, es muy sensible ante cualquier indicio de radiación. Las primeras pruebas que se realizaron con el sensor PIR estaba constantemente activo, ¿por qué? Gracias al poder de la observación advertí que el aire acondicionado estaba en funcionamiento y la burbuja de aire frío creada por él, era captada por los infrarrojos del sensor. En el momento que reduje el espacio del sensor, aislándolo del aula con la caja de cartón, éste empezó a funcionar cuando debía.

Hablar de potenciar el circuito es utilizar dispositivos mucho más sofisticados, pero también sujetos a las leyes del capital. Pero acepto las posibles actualizaciones como la mencionada anteriormente de capturar imágenes panorámicas, o subir las imágenes –o vídeos– a otras “nubes” como Dropbox o Telegram.

En lo que concierne a esta versión: ¡Funciona!

²³ Ley de Murphy, “si algo puede pasar, pasará”.

FUENTES UTILIZADAS

<https://www.programoergosum.es/tutoriales/webcam-con-fswebcam-en-raspberry-pi/>

<https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspistill.md>

<https://geekytheory.com/tutorial-raspberry-pi-uso-de-picamera-con-python>

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

<https://pypi.org/project/picamera/>

<https://pypi.org/project/PyDrive/>

<https://pythonhosted.org/PyDrive/quickstart.html>

<https://github.com/googleworkspace/PyDrive>

<https://cloud.google.com/>