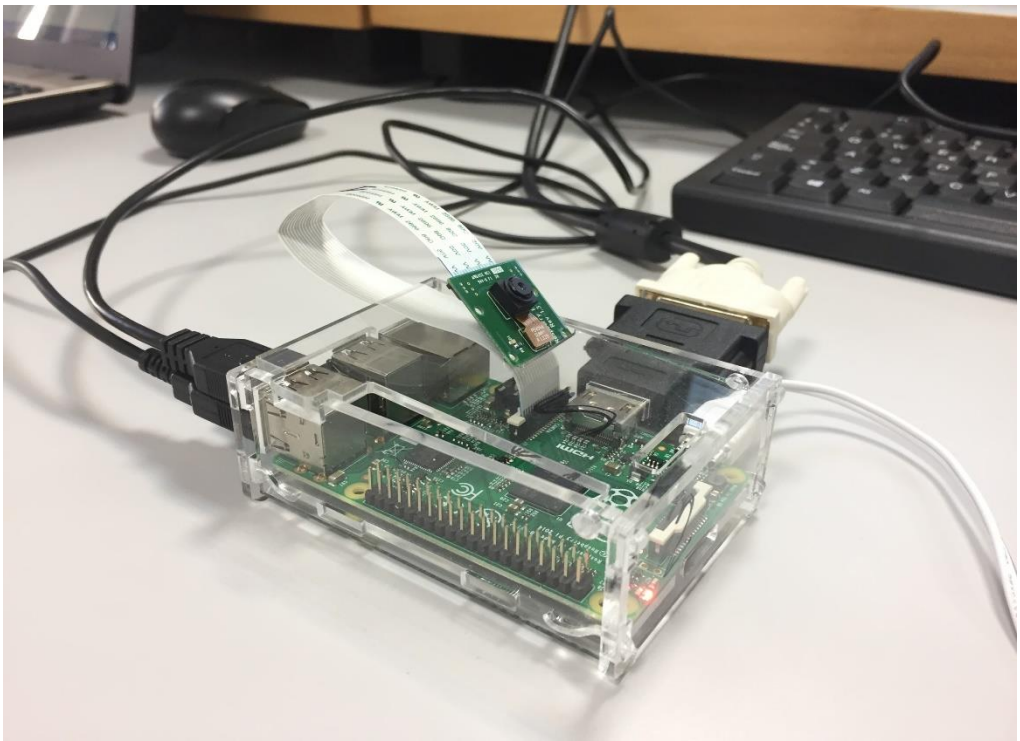
DETECTOR DE VELOCIDAD

Detección del objeto y cálculo de la velocidad



JOSE ANTONIO SIERRA IBÁÑEZ
ANTONIO LÓPEZ HERNÁNDEZ

Contenido

1. Planteamiento inicial

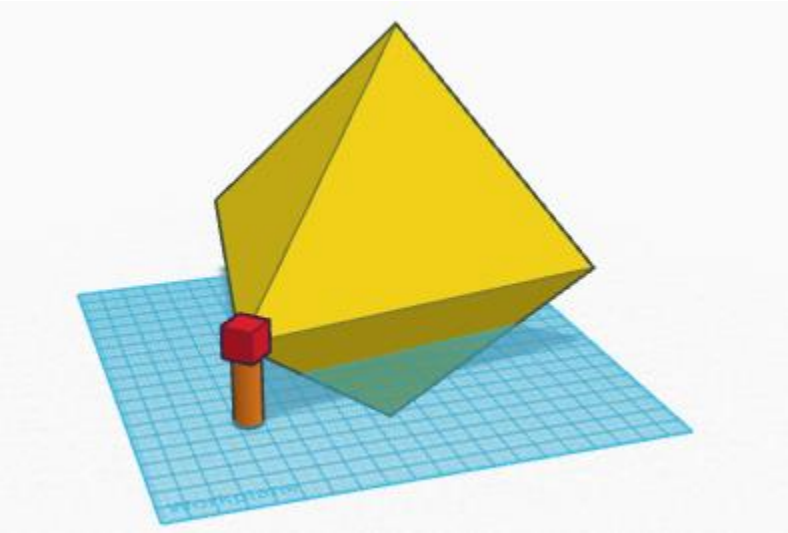
Nuestro objetivo era crear un radar con los dispositivos con los que contabamos que eran una Raspberry 3 y una cámara.

El pensamiento es el siguiente, tenemos una cámara enfocando una zona, por ella pasará un objeto y la aplicación debe de ser capaz de determinar la velocidad. Para ello seguimos la fórmula de la velocidad:

$$velocidad = \frac{espacio}{tiempo}$$

Gracias a OpenCV vamos a usar una serie de características que detectarán el movimiento en el foco de la cámara, entonces iniciará un temporizador, el objeto saldrá del foco de la cámara y se activará otro temporizador, así sabremos el tiempo que ha tardado, y diciéndole a la cámara a la distancia que ha pasado el objeto, puede calcular el espacio que ha recorrido.

Esto es necesario por el siguiente motivo:



Imaginemos el que cuadrado rojo es la cámara y la zona amarilla es su ángulo de visión, esto implica que cuando un objeto pase muy cercano a la cámara recorre un espacio diferente que si pasa por la zona más alejada de la cámara. Esto es muy importante dado que según a qué distancia pase de la cámara recorre un espacio u otro. Sin embargo con una sola cámara no podemos calcular esto, asique para nuestro desarrollo especificamos en el código a qué distancia se encuentra el objeto respecto de la cámara, y usaremos unas fórmulas ofrecidas por piCamera para estimular la distancia que ha recorrido este objeto.

2. Explicación del código

El código viene documentado, aun así hay ciertos aspectos que queremos resaltar.

```
# Calculo de la velocidad
def get_speed(pixels, ftperpixel, secs):
    if secs > 0.0:
        return ((pixels * ftperpixel)/ secs) * 3.6
    else:
        return 0.0
```

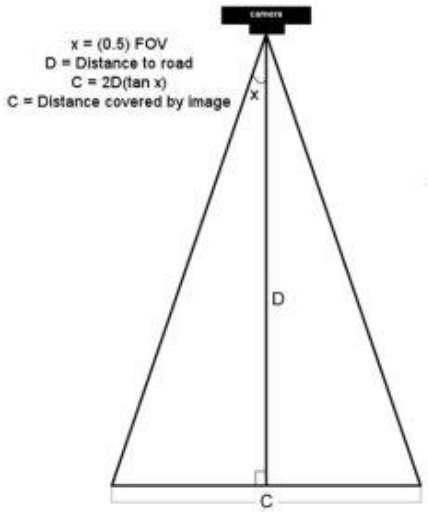
Esta es la función principal de todo, el cálculo de la velocidad, velocidad es espacio partido tiempo, y para este cálculo nuestro espacio son los pixeles recorridos, y el tiempo es el cálculo de segundos, usamos 3.6 para hacer un paso rápido a km/h.

```
# Distancia desde la cámara al objeto
DISTANCE = 2
# El numero de pixeles que hace falta que cambien para tenerlo en cuenta
THRESHOLD = 50
# El tamaño minimo del objeto para tenerlo en cuenta
MIN_AREA = 175
```

Las tres variables esenciales, la distancia a la que se encuentra el objeto respecto de la cámara, el número de píxeles que tienen que cambiar para considerarlo objeto y el área que tienen que ocupar esos pixeles, estos son los valores que tenemos que cambiar para que nuestro radar funcione en otros entornos.

```
FOV = 53.5
frame_width_ft = 2*(math.tan(math.radians(FOV*0.5))*DISTANCE)
ftperpixel = frame_width_ft / float(IMAGEWIDTH)
```

Este cálculo lo encontramos en uno de los enlaces de la biografía, como no tenemos dos cámaras para definir la distancia a la que pasan los objetos, se hace un cálculo con los ángulos respecto a la distancia que pasa el objeto.



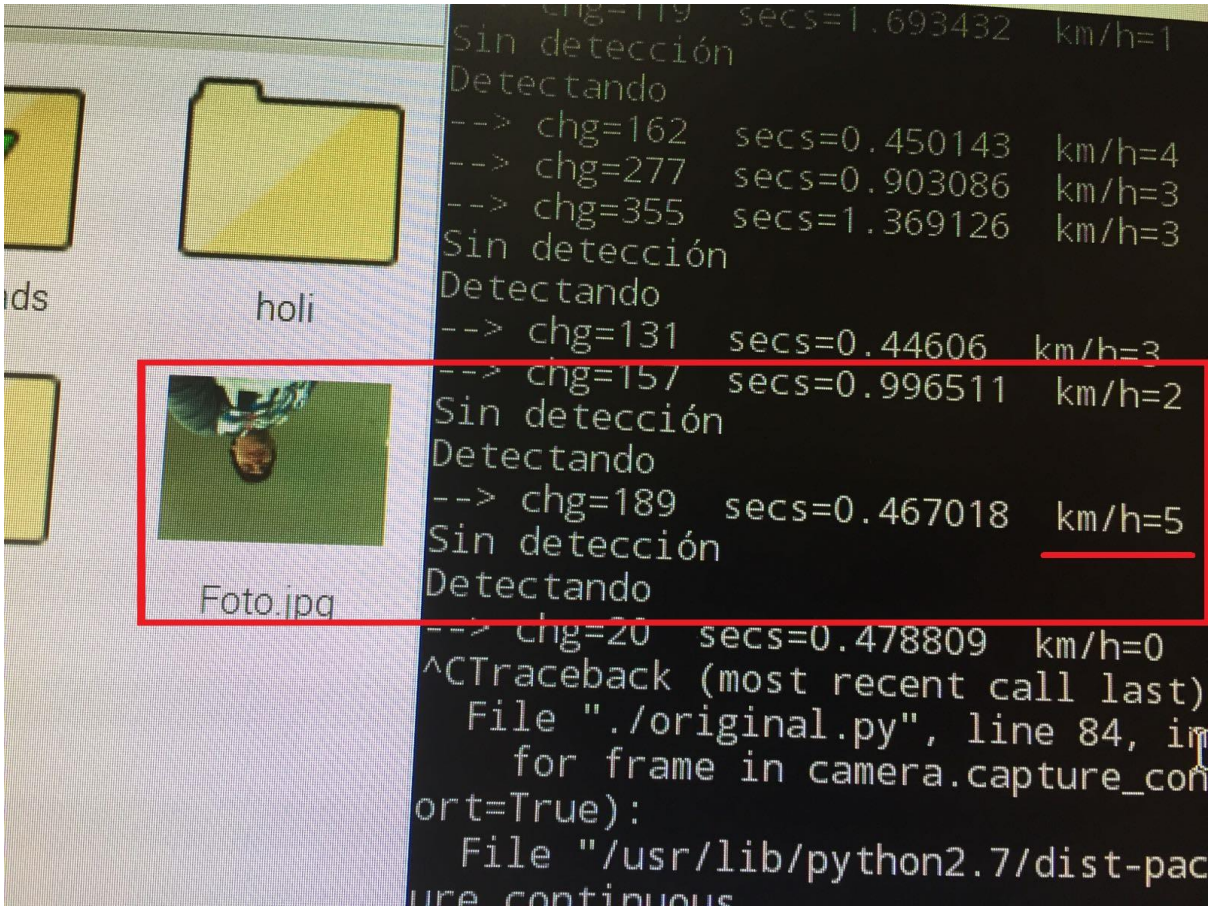
Se basa en este cálculo para saber el espacio que ha recorrido el objeto.

```
#Si esta en espera se cambia a detección
if state == WAITING:
    # Inicializamos la detección
    state = TRACKING
# Obtenemos la posición y el tiempo
    initial_x = x
    last_x = x
    initial_time = timestamp
    last_kph = 0
    text_on_image = 'Detectando'
    print(text_on_image)
else:
    #Si está en modo detección obtenemos la dirección del objeto y calculamos los cambios
    # y los mostramos por consola
    if state == TRACKING:
        #Si va de izquierda a derecha
        if x >= last_x:
            direction = LEFT_TO_RIGHT
            abs_chg = x + w - initial_x
        #Si va de derecha a izquierda
        else:
            direction = RIGHT_TO_LEFT
            abs_chg = initial_x - x
        # Calculamos la velocidad a partir de los cambios en la imagen
        secs = secs_diff(timestamp,initial_time)
        kph = get_speed(abs_chg,ftperpixel,secs)
        #Imprimimos por consola
        print("--> chg={}  secs={}  km/h={}".format(abs_chg,secs,"%.0f" % kph))
        # Una vez el objeto ha salido del área, guardamos
        if ((x <= 2) and (direction == RIGHT_TO_LEFT)) \
                or ((x+w >= 440 - 2) \
                and (direction == LEFT_TO_RIGHT)):
            # Guardamos la foto obtenida que más cambios tenga
            cv2.imwrite("Foto.jpg",image)
            state = SAVING
        # Una vez salga el objeto, obtener la última velocidad y última posición
        last_kph = kph
        last_x = x
```

Este es el núcleo de funcionamiento dentro de los bucles, cuando detectan un frame que ha cambiado el estado cambia a “grabando”, tiene en cuenta si el objeto está pasando de derecha a izquierda o izquierda a derecha, e imprime por pantalla el cambio que ha ocurrido y la velocidad a la que viajaba el objeto.

El cálculo consiste en que dada una distancia que nosotros le hemos dado diciéndole a qué distancia va a pasar el objeto, hace un cálculo para saber la anchura en píxeles que va a recorrer el objeto, con eso tenemos la distancia, después gracias a openCV detectamos movimiento y contornos, al detectarlo sabemos cuántos cambios hay desde el frame actual al frame base, ya tenemos el espacio recorrido por el objeto y su tamaño, utilizamos un contador desde que empezamos a detectar movimiento hasta que no lo detectamos y así puede hacer una estimación de la velocidad.

A la hora de obtener una velocidad de las varias que salen por consola cogemos la de la línea que tenga mayores cambios.



Para que la imagen se guarde debe superar un número de cambios, esto lo hacemos para que no guarde 20 fotos, solo 1 que sea justo cuando el objeto se encuentra en el centro de la imagen.

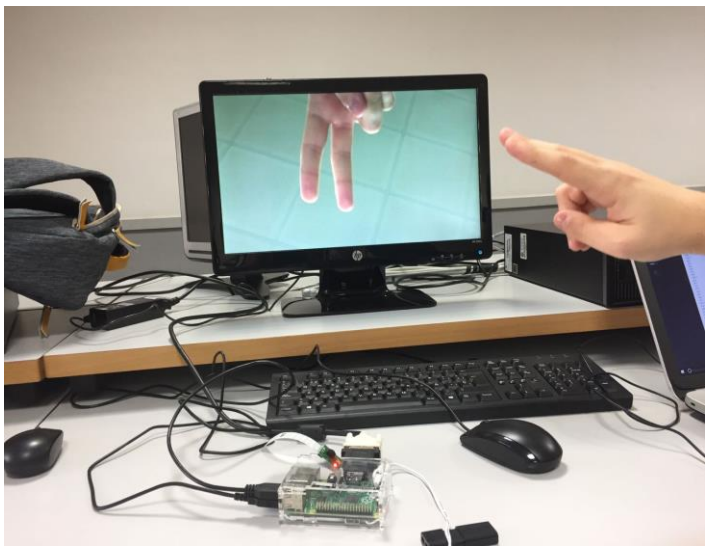
```
# Si el objeto está parado, no lo contamos como movimiento e impedimos
# que la cámara este detectándolo
if (state == WAITING) or (loop_count > 20):

    state=WAITING;
    loop_count = 0

# Limpiamos el stream
rawCapture.truncate(0)
loop_count = loop_count + 1
```

Este pequeño bucle lo utilizamos para que la cámara no se quede colgada ante un cambio permanente.

3. Pruebas

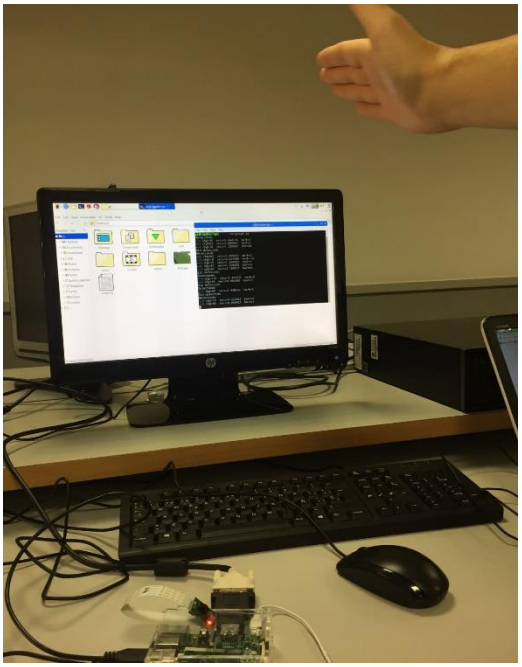
3.1 Comprobación de la cámara

Ejecutamos el programa video.py para comprobar que la cámara funciona correctamente y mostrar la zona por la que va a pasar el objeto.



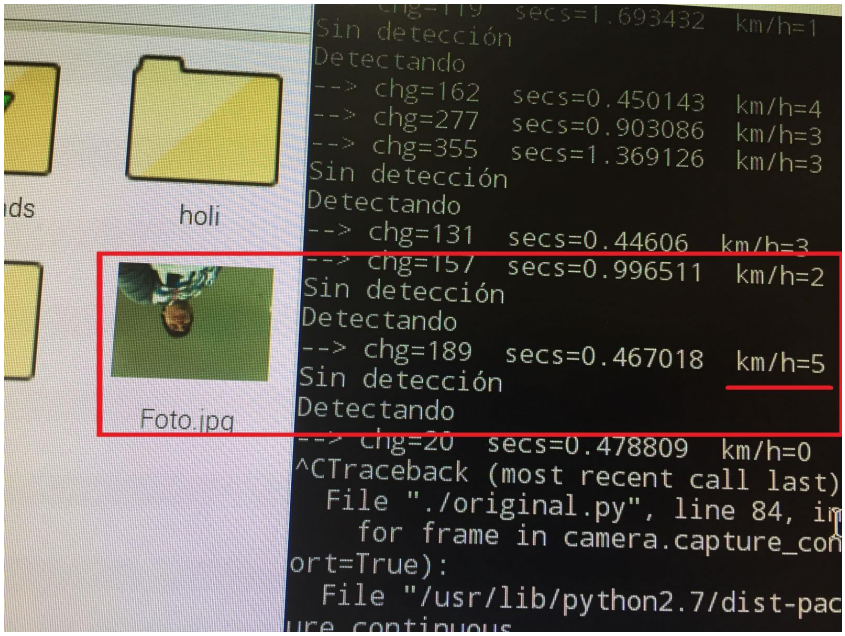
3.2 Ejecutamos radar.py

Desde la consola de comandos dándole permisos al archivo radar.py podremos ejecutar el script que ejecuta el código, una vez detecte movimiento, se mostrará por consola los datos correspondientes, los cambios de píxeles, los segundos que transcurre desde que aparece hasta que desaparece el objeto y la velocidad que lleva. Guardará una foto que será la que supere un número de cambios de esa traza de imágenes que está guardando. Como podemos ver en la imagen de a continuación podemos dejar el escritorio como se ve para poder ver la foto que ha guardado junto con las medidas que ha tomado.



3.3 Interpretación de los resultados

Una vez que hemos ejecutado el programa podemos ver que cuando pasamos la mano o un objeto nos marca las distintas velocidades y cogerá la foto que tenga mayores cambios, y solo si esos cambios superan nuestro tamaño mínimo.



4. Consideraciones y Sugerencias

Podemos encontrarnos con que se quede fija en un bucle infinito, esto es debido a que la cámara puede detectar un cambio como la luz ambiente y que lo considere movimiento, simplemente cerramos el proceso y volvemos a abrirlo.

El programa está pensado para que el objeto pase a una distancia de 2 metros, por defecto. Si comenzamos a pasar la mano a menos de esa distancia, el cálculo de la velocidad será erróneo, además debemos de tener la cámara enfocando correctamente y no tumbada, dado que está pensado para un paso de izquierda a derecha y viceversa y no para un paso de arriba hacia abajo.

Con raspberry pi 2, el programa funciona, pero bastante lento y mal dado que tiene que procesar demasiada información y después de echar una foto, si pasamos otra vez el objeto puede que no lo detecte o que no realice la foto dado que el hecho de ejecutar el programa ya pone al procesado al 80% de su potencia. Con raspberry pi 3 no tenemos ningún problema de potencia.

5. Posibles mejoras

- Cámara en tiempo real, que a la vez que realice el trazado nos muestre lo que está viendo la cámara. No es algo difícil de implementar pero el programa no arrancaría en Raspberrypi 2 por la cantidad de recursos que usaría.
- Guardar la imagen con la velocidad buena, es decir como hemos visto guarda una imagen pero muestra por consola bastantes líneas aunque solo nos tenemos que fijar en una, lo ideal es guardar la imagen e imprimir sobre ella la velocidad, lo hemos intentado pero daba muchos problemas.

6. Bibliografía

Para la realización de este proyecto nos hemos apoyado en diferentes fuentes que nos han ayudado tanto a comprender OpenCV2 como a ayudarnos a la realización del proyecto.

- https://programarfacil.com/blog/vision-artificial/deteccion-de-movimiento-con-opencv-python/
- http://robologs.net/2014/04/25/instalar-opencv-en-raspberry-pi-2/
- https://lignux.com/crear-y-ejecutar-scripts-python/
- https://geekytheory.com/tutorial-raspberry-pi-uso-de-picamera-con-python/
- https://www.raspberrypi.org/downloads/noobs/
- http://computerhoy.com/noticias/hardware/que-es-raspberry-pi-donde-comprarla-como-usarla-8614
- https://gregtinkers.wordpress.com/2016/03/25/car-speed-detector/