Detector de Movimiento

Presentado por: *Fernando González Ramos.*
Titulación: *Ingeniería Telemática.*

Contenido

Introducción

Se trata de un sensor de movimiento basado en imagen; ya que, a diferencia de la gran mayoría de sensores de movimiento que hacen uso de sensores de ultrasonido, infrarrojos, o láser, éste requiere como sensor, una cámara.

Esta cámara puede ser de cualquier tipo. Bien sea una webcam de toda índole, tanto usb como integrada (como puede ser el caso de un ordenador portátil) o, como se menciona más adelante, una cámara **PiCam**.

La cámara que recibe el nombre *PiCam* se trata de una cámara de tamaño muy reducido diseñada específicamente para su uso con la serie de “mini ordenadores” Raspberry Pi o similar.

Más adelante, se detallan las ventajas de usar este tipo de sistema, así como sus inconvenientes. Pero, en primer lugar, nos centraremos en los detalles de su infraestructura.

Metodología

En primer lugar, es necesario explicar que el software de este sistema está compuesto íntegramente por código escrito en lenguaje Python y cuya compatibilidad con cualquier sistema operativo GNU/LINUX derivado de Debian (así como Debian, Ubuntu, Lubuntu, Xubuntu, Linux Mint o Raspbian) está garantizado, siempre que se cumplan las siguientes dependencias requeridas:

- Python 2.7 o superior.
- Librería de Python *matplotlib*
- Librería de Python *numpy*
- Librería de Python *scipy*
- Librería de Python *Image*
- Librería de Python *Tkinter*
- Librería de Python *cv2* (OpenCV)

El comportamiento del sistema de detección de movimiento consta de dos partes claramente diferenciadas: el tratamiento de la imágenes y la estimación. Esto es así debido a que la metodología llevada a cabo por el software es secuencial; realizando, en primer lugar, una fase de “entrenamiento” y; en segundo lugar, una fase de detección. Comenzaré explicando esta última.

1.- Tratamiento de Imagen

Su implementación se encuentra en la librería ***imgs_ops.py*** adjunta.

Esta librería contiene una única clase llamada *Images* con un método público denominado *update*, es decir al que se le puede llamar desde otro programa haciendo uso de un objeto, previamente creado, de la clase *Images* (se explica posteriormente).

Este método está diseñado para ser invocado **iterativamente**. A continuación, explicaré, *grosso modo*, su funcionamiento:

Para comenzar, se obtiene un "frame" de la cámara (haciendo uso de la librería *cv2*), se cambia su espacio de color de BGR a GRAY (escala de grises). Posteriormente, se le aplica un filtro paso alto para así detectar los bordes presentes en la imagen, a la que se le aplica también una dilatación para así realzar dichos bordes.

Una vez hecho esto con el frame actual y habiendo hecho la misma operación con el frame capturado anteriormente, se obtiene la diferencia entre ambos (haciendo uso del método *absdiff* presente en la librería *cv2*).

Por último, sobre la imagen "diferencia" obtenida, se lleva al cabo una operación a la que he denominado "dep", debido al parecido que guarda con la operación de cálculo de la *densidad espectral de potencia* de una señal. Este cálculo se trata de sumar cada uno de los valores de cada píxel de la imagen.

El valor que arroja "dep" será el que determine si hubo movimiento en frente de la cámara o no.

*__NOTA:__ Todo el código explicado en esta sección es adjuntado y puede visualizarse si se desea.

Pero, **¿qué valor de *dep* es el óptimo para determinar el coeficiente de verosimilitud necesario para discernir el movimiento de la quietud?**

Intuitivamente, se puede pensar en obtener este valor mediante la experimentación.

Sin embargo, lo deseable es que el detector de movimiento sea eficaz en cualquier entorno en el que se encuentre, no únicamente en el entorno bajo cuyas condiciones se ha obtenido dicho valor umbral de manera empírica.

Por este motivo, el sistema, al iniciarse, pone en marcha una fase previa a la que he denominado "de entrenamiento", que explico a continuación.

2.- Estimación

La fase de entrenamiento trata de un problema de estimación donde es necesario calcular cuál es el valor umbral óptimo para decidir si existe movimiento frente a la cámara o no (coeficiente de verosimilitud).

El código que realiza esta tarea se encuentra en la librería ***data_train.py*** adjunta.

Lo que se lleva a cabo en esta fase son exactamente 200 mediciones (para una mayor optimización deberían ser más pero actualmente se utiliza este valor para no alargar el proceso) de los valores de "dep", que han de obtenerse bajo las condiciones en las que el sistema va a operar para así garantizar su buen funcionamiento, habiendo movimiento delante de la cámara y sin haber movimiento.

Una vez obtenidos los 200 valores para cada caso, 400 en total, se realiza una media de ambos por separado y, dado que el número de muestras es el mismo, es decir, se asume que ambos sucesos ocurren con la misma frecuencia (son equipobables), se aplica un decisor ML, que determina que el valor umbral buscado es la media de las medias de los valores obtenidos con y sin movimiento.

Una vez que este valor ha sido obtenido, se utiliza en la segunda fase, explicada anteriormente, de detección, en la que se establece que si del valor de "dep" supera el umbral, ha habido movimiento; si no, no se ha producido movimiento.

3.- Programa Principal

El programa principal, denominado ***movement_detector.py*** y que se adjunta, comienza creando dos objetos de sendas clases correspondientes para ejecutar el código contenido en ambas librerías explicadas anteriormente.

En primer lugar, comienza el entrenamiento y, cuando éste finaliza se ejecuta el método *update* de la librería *imgs_ops.py* en bucle a una velocidad de 20 Hertzios.

```
#Reactive system

  rate = Timer(20)  #20 Hz
  while(1):

    img.update()
    print("Update\n")
    rate.sleep()
```

Para mantener estos 20 hertzios, hago uso de una clase a la que he llamado *Timer,* la cual recibe como parámetro la frecuencia, en hertzios, deseada; que se encarga de "dormir" el programa el tiempo necesario para mantener dicha frecuencia. En este caso, he elegido 20Hz porque es la tasa de fps aproximada de la cámara que he utilizado para todos mis ensayos. **Esta es una buena manera de no desperdiciar CPU,** sobre todo, pensando que este sistema está pensado para su integración en mini ordenadores o computadoras de bajo coste, como mencioné en la introducción.

Conclusión

Se han realizado pruebas en diferentes entornos, todos ellos de interiores, y sus resultados han sido notablemente satisfactorios y han dejado en evidencia las ventajas e inconvenientes que este sistema presenta frente a otros sistemas de detección de movimiento ampliamente utilizados.

De entre sus **ventajas**, destacan, la sencillez en la puesta en marcha de este sistema en cualquier computadora y su fácil integración en cualquier entorno pasando prácticamente desapercibido (buena herramienta para vigilancia), poco coste a nivel energético, computacional y económico, ya que puede instalarse en un mini ordenador de bajo coste como la Raspberry Pi 3 que se muestra en la fotografía

Otra notoria ventaja reside en el hecho de que la cámara abarca un amplio sector, mientras que si se quisiera detectar movimiento en ese mismo sector utilizando otro sistema, sería necesario poner varios sensores. De este modo, con una sola cámara, puede detectarse movimiento en cualquier punto del área de visión que abarca la misma.

Como **inconveniente** principal: las condiciones del entorno en el que está llevándose a cabo la detección no pueden variar de forma brusca pues, si así fuera, el valor umbral ya no sería válido y el detector no funcionaría con precisión.