1-11-2022

2° Examen de Visión Computacional

Descriptores

DIEGO DOMINGO CHACON RIVERA - 307382
UASLP – FACULTAD DE INGENIERÍA – VISIÓN COMPUTACIONAL

Planteamiento del problema

Implementar un programa mediante una señal de video o que, al presentarle una nueva fotografía del objeto, sea capaz de identificar los objetos elegidos en el paso anterior, utilizando funciones de descriptores que contiene la biblioteca de OpenCV. Se deben de seleccionar al menos 3 objetos diferentes que sean de preferencia coloridos y/o con textura.

El programa deberá usar los siguientes descriptores:

- ORB
- SIFT
- HOG

Descripción de la solución

Implementación ORB

Para llevar a cabo el desarrollo de ORB se implementó una clase, la cual tiene como atributos los siguientes datos:

- **orb**: Instancia de la biblioteca OpenCV para crear el detector ORB. Dentro de los parámetros **se recomienda conservar los valores default,** aunque el valor de **nfratures** se puede cambiar para dar un número máximo de entidades a conservar.
- **imgTrain**: Lista donde se almacenan las imágenes para entrenar el algoritmo.
- **imgClassNames**: Lista donde se almacenan los nombres de las imágenes de entrenamiento.
- **desList**: Lista donde se almacena el descriptor por imagen.
- **Id**: Posición de coincidencia entre la imagen a comparar (query) y la imagen de entrenamiento (train).
  Anexos\ORB y SIFT\atributos.png

Se definieron una serie de métodos para el manejo del descriptor:

- **Constructor (__init__)**: recibe un **path**, con el cual a través de un ciclo for se rellena la lista de imágenes de entrenamiento (imgTrain) y la lista de nombre de las imágenes (imgClassNames).
  Anexos\ORB y SIFT\constructor.png
- **findQueryDes**: Recorre la lista de las imágenes de entrenamiento (imgTrain). Por cada imagen encuentra los puntos clave y calcula sus descriptores mediante la función **detectAndCompute**() la cual debe de recibir una imagen y la declaración **NONE**. Estos descriptores los almacena en la lista **desList.**
  Anexos\ORB y SIFT\findQueryDes.png
- **findID**: Recibe una imagen a analizar (query) la cual debe de ser almacenada, después, obtiene los puntos clave y calcula los descriptores de la imagen usando **detectAndCompute**(). Ahora bien, para encontrar los puntos de coincidencia entre la imagen y las imágenes de entrenamiento se aplica el uso de BFMatcher, ya que es un comparador de fuerza bruta. Para utilizarlo simplemente se usó de la función **BFMatcher( )** incluida en OpenCV, la cual no tiene parámetros.

Se define una **matchList** dentro de la cual vamos a almacenar todos los puntos de coincidencia con las imágenes de entrenamiento. Para realizar esto, utilizamos un ciclo for que recorra la lista de descriptores de las imágenes de entrenamiento (desList) y por cada descriptor recorrido lo comparamos con el descriptor de la imagen query, haciendo uso de la función **knnMatches(),** la cual tiene como parámetros:

- Descriptor de entrenamiento (qd)
- Descriptor de la imagen query (des)
- Numero k de pasadas

knnMatches() es la k-clasificación de vecino más cercano. El algoritmo kNN encuentra los k registros más cercanos a los nuevos datos del conjunto de entrenamiento y luego determina la categoría de los nuevos datos según su clasificación principal.

Una vez terminado el ciclo, se debe verifica que la lista de coincidencias no este vacía y se obtiene el id del valor con más coincidencias, esto se hace usando la función **max(Lista de valores)**.
Anexos\ORB y SIFT\findID.png

- **nameImg:** Dentro de esta función, se extrae el nombre de la imagen con más coincidencias y se le coloca a la imagen query. Esto se hace aplicando la función **putText()**, la cual recibe los siguientes parámetros:
  - Imagen query en la cual se agregará el texto.
  - Cadena de texto a agregar (para nuestro caso se utiliza la comprendida dentro de la lista de nombres de imágenes train (imgClassName) y se accede por medio de índice el cual es el id previamente obtenido).
  - Esquena superior izquierda por coordenadas X y Y (donde iniciara el texto).
  - Tipo de la fuente.
  - Tamaño de la fuente
  - Color de la fuente
  - Grosor de la fuente.

  Anexos\ORB y SIFT\nameImg.png
- **showImg:** Redimensiona la imagen y la muestra en una ventana.
  Anexos\ORB y SIFT\showImg.png

Implementación SIFT

Para llevar a cabo el desarrollo de SIFT se implementó una clase, pero esta clase cuenta con los mismos métodos que la clase del descriptor ORB, ya que también se implemento el uso de **BFMatcher**, por otra parte, los atributos de la clase son muy similares ya que también se implemento una serie de listas para el entrenamiento del algoritmo, la lista de descriptores de las imágenes de entrenamiento y el id de coincidencia. Sin embargo, para poder implementar el algoritmo se requirió de una variable única perteneciente a la clase, la cual es:

sift = cv2.xfeatures2d.SIFT_create()

Este atributo utiliza la función **xfeatures2d.SIFT_create()**, incorporada en OpenCV para poder hacer instancia del algoritmo.

- **Xfeatures2d** es una clase para extraer características robustas aceleradas de una imagen e incorpora funciones o métodos para el manejo de los algoritmos SIFT y SURF.
- **SIFT.create()** es un método que nos ayuda a crear una instancia para extraer puntos clave y calcular descriptores utilizando el algoritmo Scale Invariant Feature Transform.

Anexos\ORB y SIFT\atributos_sift.png

Implementación de HOG

Siguiendo la misma forma de los 2 anteriores descriptores, para HOG se implementó una clase, que tiene como atributos:

- **hog**: Implementa el descriptor de histograma de gradientes orientados HOG. Esto lo hace llamando a la función **HOGDescriptor()**, la cual **es recomendable dejar los parámetros por default**.
- **bounding_boxes**: Lista de cuadros delimitadores para identificar a las personas.
- **Img**: Imagen sobre la cual se aplicará el descriptor.

  Anexos\HOG\atributos.png

De igual manera, aplica una serie de métodos los cuales nos ayudaran a manejar el comportamiento del algoritmo.

- **Constructor (__init__)**: recibe como parámetro una imagen (img), la cual se almacena en la clase. Posteriormente se debe de escalar la imagen, ya que esto nos ayuda a garantizar que se deben de evaluar menos ventanas deslizantes en la pirámide de imágenes. Ahora bien, tenemos que llamar a **setSVMDetector()** para configurar la máquina de soporte vectorial para que sea un detector de personas preentrenado, cargado a través de la función **cv2.HOGDescriptor_getDefaultPeopleDetector()**.

  ```
  hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
  ```

  Anexos\HOG\constructor.png
- **findPeople**: Este método implementa la detección de personas en la imagen. Esto los hace mediante un llamado al método **detectMultiScale()** del descriptor HOG. El método **detectMultiScale()** tiene como parametros:
  - **winStride:** Es un tamaño de paso de ventana deslizante de (4 ,4) píxeles tanto en la dirección X como en la dirección Y.
  - **padding:** Nos ofrece un relleno de la imagen tanto en la dirección X como en la dirección Y.
  - **scale: C**onstruye una pirámide de imágenes con una escala = 1.05.

  La función **detectMultiScale()** genera 2 listas:

  - **bounding_boxes**: Coordenadas (x, y) del cuadro delimitador de cada persona en la imagen.
  - **weights**: El valor de confianza devuelto por el SVM para cada detección.

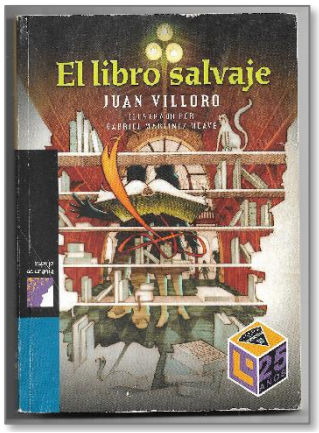
  Anexos\HOG\findPeople.png

- **drawPeople**: Se encarga de generar un rectángulo identificador para las personas detectadas. Esto lo hace recorriendo la lista de **bounding_boxes** y nos devuelve una serie de coordenadas en cuadrante para poder pintar el cuadro.
  En ocasiones la función **detectMultiScale()** detecta falsamente un cuadro delimitador (junto con el cuadro delimitador correcto), para corregir esto necesitamos de un valor grande del parámetro **overlapThresh** en la función **non_maxima_suppression.**
  Anexos\HOG\drawPeople.png
- **showImg:** Redimensiona la imagen y la muestra en una ventana.
  Anexos\HOG\showImg.png

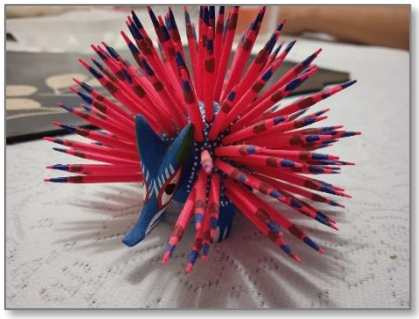
Descripción de los resultados

Imágenes de entrenamiento para ORB y SIFT

Libro Infantil



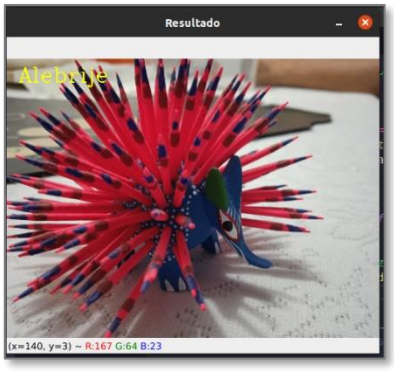
Libro Salvaje

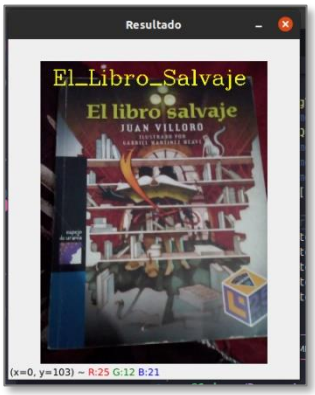


Alebrije

Resultados de ORB

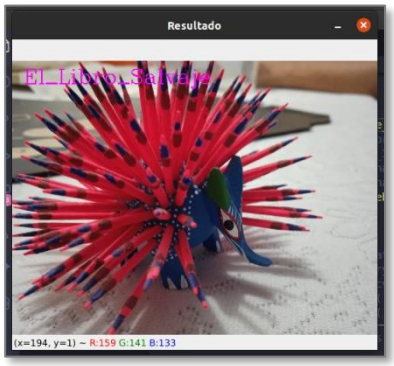
Los resultados fueron óptimos para todos los casos analizados. Para los 3 objetos evaluados, se identifico a la perfección cada uno, como podemos ver con el nombre en amarillo.





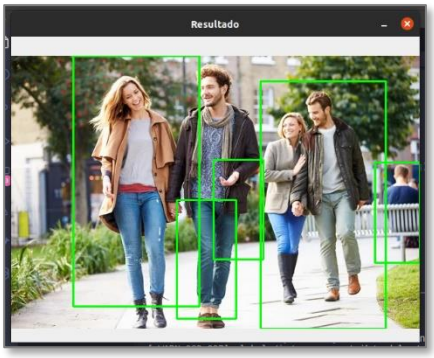
Resultados de SIFT

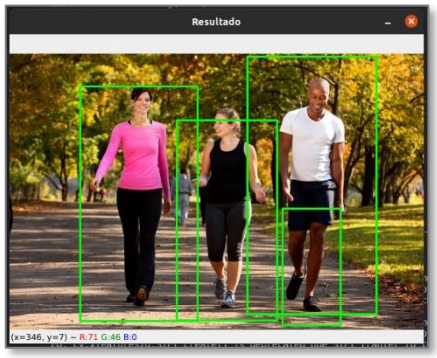
Los resultados no fueron los mas óptimos, ya que solo detecto correctamente los 2 libros. Para el alebrije fallo y en si lugar le colocaba el nombre de “El libro salvaje”.

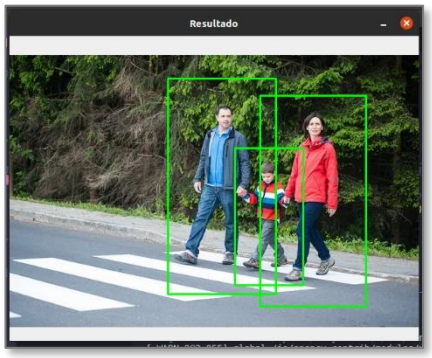


Resultados de HOG

Se presento un resultado optimo al momento de detectar a las personas, pero no en todos los casos se detecto completamente a las personas o bien solo partes de ellas.







Discusión

Ya que ORB y SIFT se utilizaron para detectar objetos, se obtuvo que ORB tuvo un mejor resultado ya que logro identificar cada uno de los objetos de manera correcta. El algoritmo describió puntos clave óptimos para cada figura presentada, por lo cual la inconsistencia de las características obtenidas por el descriptor fue menor. Por su parte SIFT, describió a la perfección los 2 libros, ofreciendo una gran cantidad de puntos clave en cada uno de ellos. Pero esta cantidad fue lo que lo hizo fallar, ya que con una imagen como lo es “El Alebrije” que tiene muchos colores y que a su vez muchos los comparte con la imagen de “El Libro Salvaje” se enfocó de manera errónea en los puntos clave del descriptor, lo cual le genero inconsistencia y fallo.

Para el descriptor HOG se apreció que en ocasiones solo detecta parte de las personas (piernas, brezos, cabeza, etc.) lo cual no es malo, pero no es lo mas optimo. En los resultados presentados, se ve la diferencia de la detección si los analizamos de izquierda a derecha, mucho depende la separación y posición de las personas, color de la imagen, cantidad de objetos en la imagen, entre otros aspectos.

Conclusión

El proyecto fue un buen trabajo académico y muy atractivo, ya que el uso de los descriptores para detectar objetos y el pensar en sus implementaciones futuras o a proyectos grandes, te das cuenta del poder que tienen los descriptores. Siento que para obtener resultados óptimos se debe de entrenar cada descriptor con miles o millones de imágenes. Además de que OpenCV no deja de sorprenderme, es una biblioteca bastante extensa que cuenta con cientos de funciones para el manejo de imágenes y lo hace de una manera muy comprensible y fácil de aplicar.