MIGUEL CORTECERO TORRES

[Dirección de correo electrónico]

Descripción breve

[Dibujar su lector con un resumen de la participación. Normalmente es un breve resumen del documento.   
Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

[Título del documento]

[Subtítulo del documento]

# Parte 1

## Algoritmo implementado

Para la parte 1 de la práctica se ha hecho uso del paradigma de clases y la estructura resultante consta de:

Práctica1.py: que es la clase principal desde donde serán llamados los métodos de las diferentes clases

KeyPoint: clase que se basa en la estrucuta de un keyPoint de openCV pero añade un campo más que es distanceToCenter. Este atributo contendrá, el módulo, el vector, el ángulo y el pto del centro de la imagen del vector de votación asociado al keyPoint

Operations: clase que contiene métodos de apoyo, en este caso únicamente posee la función del cálculo al centro.

Training: contiene los métodos para leer las imágenes de entrenamiento y entrenar nuestro sistema con ellas.

Processing: contiene los métodos que nos permite obtener la ubicación de un coche a partir del detector entrenado anteriormente.

### Training

Para la etapa de entrenamiento leeremos todas las imágenes que tengamos en la carpeta “training”. De cada imagen hallaremos sus puntos de interés usando un dectector ORB (orb = cv2.ORB(nfeatures=kpNum, nlevels=4, scaleFactor=1.3)). En dicho detector le indicaremos cúantos puntos de interés queremos que nos obtenga de cada imagen, que en nuestro caso serán 150, el número de niveles en los que queremos que detecte y con qué factor de escala. Esto es importante para poder luego distinguir imágenes con distintos tamaños de coches.

Una vez obtenidos los ptos de interés, hallaremos el vector que los une con el centro y los incluiremos en un objeto de tipo KeyPoint. Además, de cada imagen nos guardaremos un un array global, en su posición correspondiente, tanto los ptos de intereés como los descriptores, para su posterior uso en la etapa de procesamiento/testing.

Además, de cada imagen crearemos un índice de búsqueda (flannBasedMatcher) que contendrá los descriptores asociados a la imagen correspondiente, y ese índice se guardará en un array global que contendrá todos los índices de todas las imágenes.

### Processing

Una vez entrenado el detector procederemos a leer las imágenes de prueba para buscar la posición del coche en la imagen.

En esta parte tendremos una máscara de acumulación donde iremos almacenando votos encontrados para la imagen actual.

Para ello, de cada imagen de prueba habrá que sacar los ptos de interés y los descriptores usando un ORB de la misma manera que en entrenamiento. Una vez hallados, crearemos una máscara intermedia en la que se añadirán los votos que provenientes de las imágenes de entrenamiento.

Para llegar hasta ello primero debemos buscar los k-vecinos más cercanos a cada pto de interés encontrado en la imagen de testing usando la función flann.knnMatch(descProcessingArray, self.globalDescArray[index], k=5). Una vez obtenidos los vecinos tendremos que añadir los votos a la máscara intermedia. Para hacer esto deberemos comparar cada pto de interés de la imagen de prueba con cada pto de interés de las imágenes de entrenamiento, teniendo en cuenta su ángulo y su distancia al centro. Realizando las operaciones pertinentes obtendremos el vector de votación, que nos indicará el pto donde se produce un voto. Es aquí entonces donde tendremos que anotarlo en la máscara intermedia, descartando aquellos votos que se produzcan fuera de la imagen.

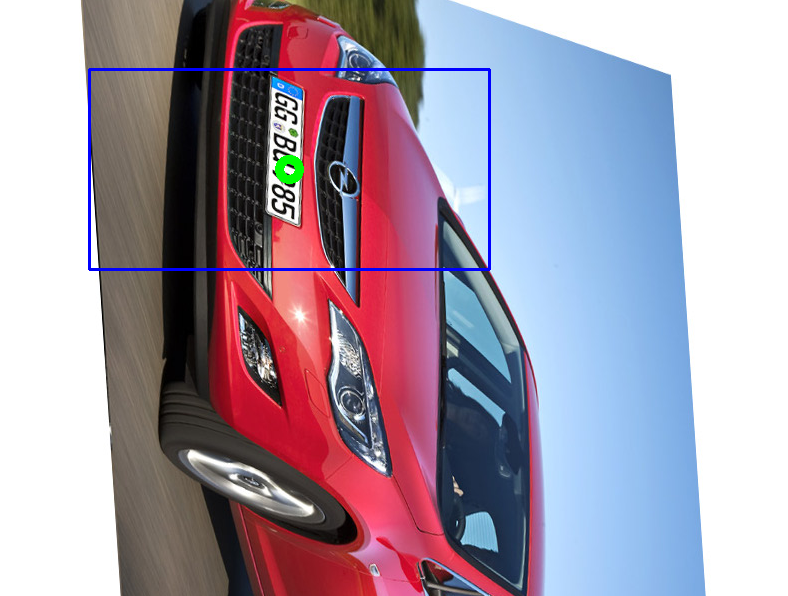
Este proceso se repite comparando cada imagen de prueba con todos los datos obtenidos en entrenamiento, integrando los votos de cada máscara intermedia en la máscara de acumulación final. Por lo tanto, al final del proceso obtendremos una máscara de acumulación que será una matriz de valores, donde el valor máximo de esa matriz representará donde se encuentra el coche. (cv2.minMaxLoc)

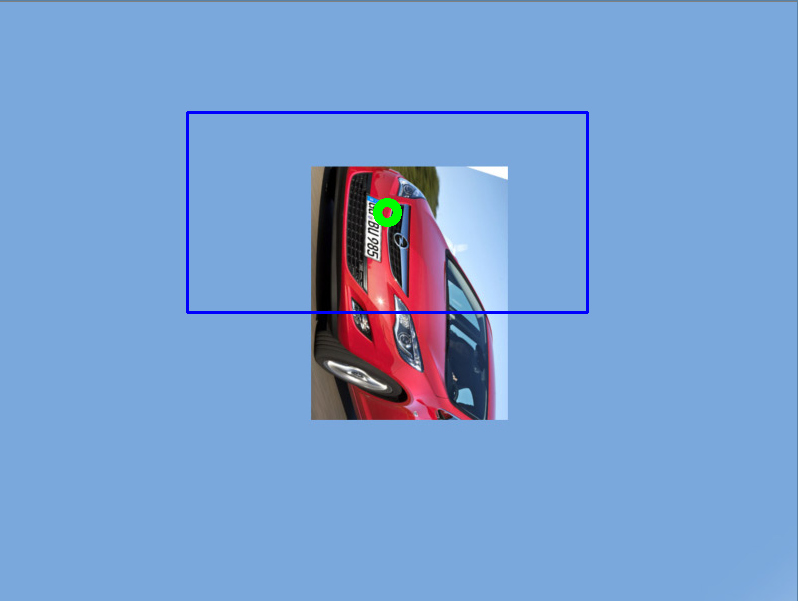
Finalmente dibujamos el pto y el área donde se ecuentra el coche con los siguientes métodos: cv2.circle(processingImage, maxLoc, 10, (0,255,0), thickness=7, lineType=8, shift=0)  
cv2.rectangle(processingImage, pt1, pt2, (255,0,0), thickness=2, lineType=8, shift=0)

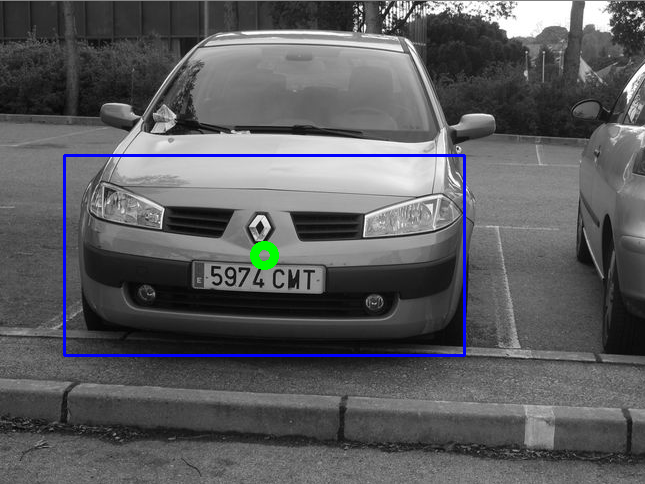
NOTA: en nuestro caso no hemos considerado necesaria hacer la rotación de los vectores de votación encontrados pues estamos usando un detector ORB (que está basado en FAST y BRIEF), que es invariante a los cambios de rotación. http://docs.opencv.org/3.0-beta/doc/py\_tutorials/py\_feature2d/py\_orb/py\_orb.html

### Ejemplos en ejecución

Para la prueba de funcionamiento se han probado imágenes tanto las imágenes suministradas como otras a las que les hemos aplicado alguna transformación (cizallado, rotación, disminución de tamaño).

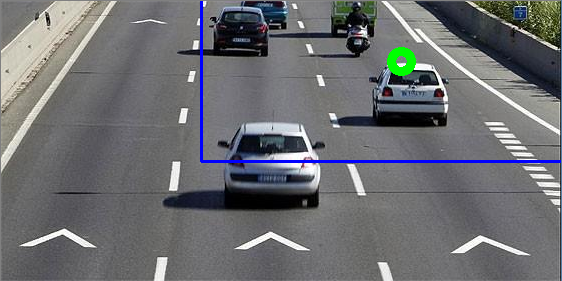








En imágenes con varios coches sólo es capaz de mostrar la ubicación de uno de ellos.



Cuando la imagen es borrosa suele fallar al encontrar el coche correctamente.



### Estadísticas

Imágenes de entrenamiento: 33

Imágenes de prueba: 33

Número de vecinos buscado: 5

Número de puntos de interés buscado: 150

De 33 imágenes de prueba se han obtenido 27 en las que el coche se ubica al menos un coche correctamente centrado, con un ligero margen de error. Es decir, aproximadamente, un 82% de acierto.

# Parte 2

# Parte 3