Image Matching

Matching de imágenes

- Encontrar las coincidencias (matches) entre dos imágenes.
- Un comparador (matcher) extrae dos conjuntos de keypoints, los compara y genera una lista de coincidencias.
- Dos formas de matching en OpenCV:

a) Fuerza bruta.

b) FLANN (Fast Library for Approximate Nearest Neighbors).

Matching por fuerza bruta

- Cada descriptor de keypoint en el primer conjunto se compara con cada descriptor de keypoint en el segundo conjunto.
- Al final se elige la mejor coincidencia en base a la distancia mínima.
- En OpenCV, la clase `BFMatcher` tiene el método `match` para encontrar la mejor coincidencia y el método `knnMatch` para encontrar las k mejores coincidencias.

Matching simple en OpenCV

- Procedimiento:

1. Encontrar los keypoints y los descriptores usando ORB o SIFT.
2. Para cada keypoint encontrar la mejor coincidencia usando el método `match` de la clase `BFMatcher`.
3. Ordenar las coincidencias por distancia.
4. Dibujar las n mejores coincidencias usando el método `drawMatches`.

Objeto DMatch

- El método match regresa un objeto DMatch que tiene 4 campos:
- distance – distancia entre los descriptores; cuanto más baja, es mejor
- trainIdx – índice del descriptor en los descriptores de la imagen en donde se busca
- queryIdx – índice del descriptor en los descriptores de la imagen que se busca
- imgIdx – índice de la imagen de entrenamiento

Ejemplo

- Buscar el logotipo de la NASA en la imagen de la derecha.

Ejemplo

- Disponible en image_features/orb_match.py.

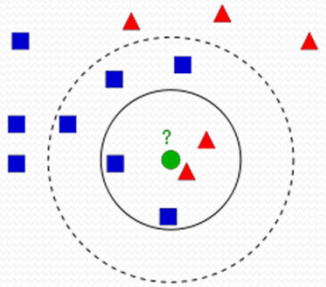
Filtrado de concidencias

- El objetivo es eliminar las coincidencias falsas.
- Se hace en dos pasos:

1. k vecinos más cercanos con $k = 2$.
2. Se aplica la prueba de proporción (ratio test).

k vecinos más cercanos

- Es un algoritmo para clasificar un dato nuevo.
- Ejemplo usando la distancia euclidiana en 2D:



Fuente: https://docs.opencv.org/4.7.0/d5/d26/tutorial_py_knn_understanding.html

- El círculo sólido tiene los $k = 3$ vecinos más cercanos al punto verde (los 2 cuadrados azules son equidistantes y se toma uno solo de ellos).
- El círculo punteado tiene los $k = 7$ vecinos más cercanos.

Ratio test

1. Para cada característica en la imagen I, encontrar los 2 vecinos más cercanos en J cuyas distancias son d_1 y d_2.
2. Calcular la relación $r = d_1/d_2$. Si $r \leq 0.8$, la puntuación de la característica es $\exp(-a \times d_1)$ donde a es un número real positivo arbitrario. Si $r > 0.8$, la puntuación es 0.
3. Repetir 1 y 2 para todas las características en I y tomar el promedio de todos los puntajes. Este será su puntaje de coincidencia general. Los valores más altos indicarán mejores coincidencias de imagen.

Ratio test

- La principal suposición es que la imagen que se busca aparece solo una vez en la imagen de entrenamiento.
- Esto implica que cada keypoint en la imagen que se busca tiene, a lo más, una coincidencia correcta en la imagen de entrenamiento y que la segunda coincidencia siempre es incorrecta.
- Si la primera coincidencia es correcta, su distancia debe ser mucho menor que la segunda coincidencia.

Match k vecinos en OpenCV

- Procedimiento:

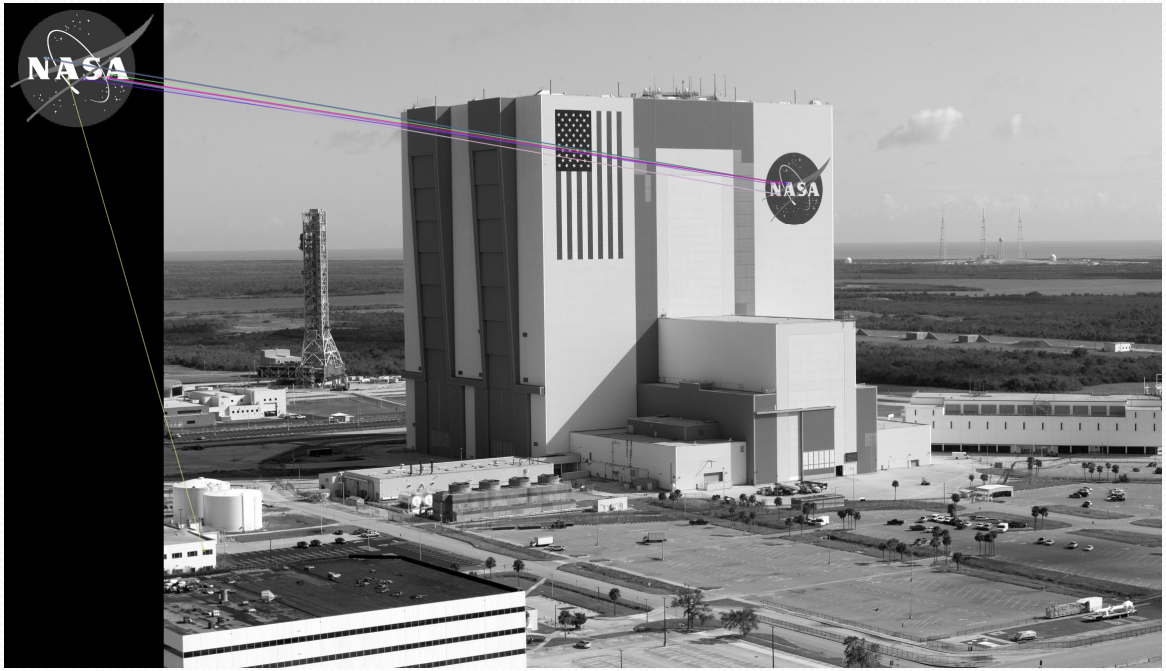
1. Encontrar los descriptores usando ORB o SIFT.
2. Para cada keypoint encontrar las 2 mejores coincidencias usando el método knnMatch de la clase BFMatcher (knnMatch regresa las dos coincidencias en orden creciente de distancia).
3. Ordenar las parejas de coincidencias por distancia.
4. Aplicar el ratio test.
5. Dibujar las n mejores coincidencias usando el método drawMatches.

Ejemplo

- Buscar el logotipo de la NASA en la imagen de la derecha.

Ejemplo

- Disponible en image_features/orb_knn.py.



Comparación match vs knnMatch / ratio

Matching con FLANN

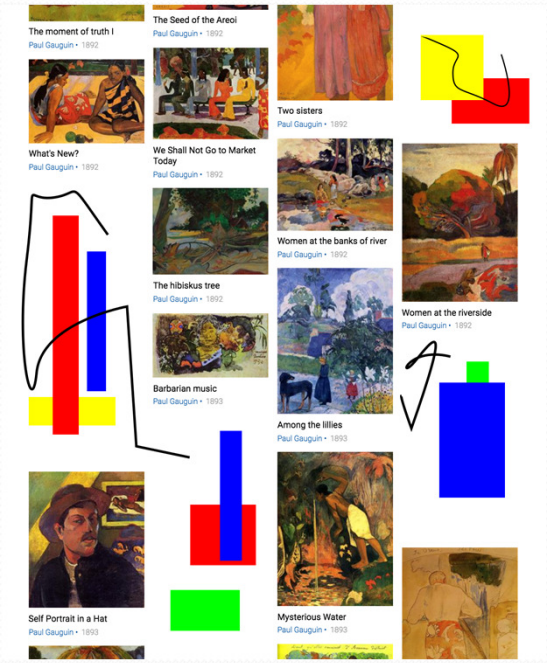
- Fast Library for Approximate Nearest Neighbors.
- Librería con varios algoritmos para encontrar vecinos más cercanos de manera aproximada.
- Para datasets grandes, es más rápido que BFMatcher.
- https://github.com/flann-lib/flann

FLANN con OpenCV

1. Detectar los keypoints y encontrar las descripciones usando ORB o SIFT.
2. Construir un objeto `FlannBasedMatcher`.
3. Obtener las coincidencias invocando el método `knnMatch`.
4. Filtrar las coincidencias usando el ratio test.
5. Dibujar las coincidencias.

Ejemplo

- Encontrar la imagen de la izquierda en la imagen de la derecha.



Ejemplo

- Disponible en image_feature/flann.py.