

Detectores y Descriptores

Araguás, Gastón Redolfi, Javier

5 de Junio del 2019

Detectores/Descriptores

- Detectar
- Describir

Detectores/Descriptores más comunes

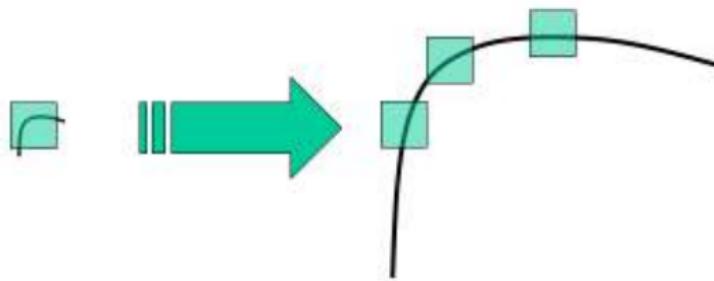
- SIFT
- SURF
- FAST
- BRIEF
- ORB

Descriptores según dominio de entrada

- Dominio Real
 - SIFT
 - SURF
- Dominio de los 0s y 1s (binarios)
 - LBP
 - BRIEF
 - ORB

SIFT

- Scale Invariant Feature Transform
- Esta compuesto por dos partes, un detector y un descriptor
- A diferencia de Harris es invariante ante escala



- Además es invariante ante rotaciones

SIFT - Invarianza ante la rotación

- Permite que el punto de interés sea invariante ante la rotación.
- Se realiza un histograma de orientaciones con los pixeles que rodean al punto de interés.
- El histograma contiene 36 bins que codifican los 360° .
- El pico más alto del histograma será considerado la dirección del punto.
- Si hay otros picos también se consideran (mayores al 80 % del máximo).

Ejemplo de uso de SIFT - Código

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('img2.ppm')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

sift = cv2.xfeatures2d.SIFT_create()
kp, dscr = sift.detectAndCompute(gray, None)

cv2.drawKeypoints(img, kp, img)

cv2.imwrite('sift_keypoints.png', img)

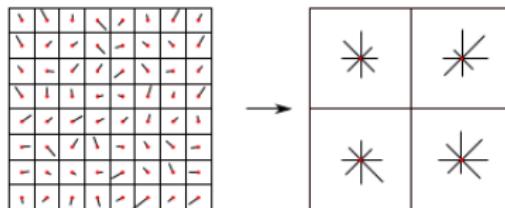
cv2.imshow('sift_keypoints', img)
cv2.waitKey()
```

Ejemplo de uso de SIFT - Puntos clave



Descriptor SIFT

- Una vez localizado el punto de interés, tenemos la ubicación, su escala y su orientación.
- El próximo caso es computar un descriptor que sea invariante ante esos parámetros.
 - Se calcula magnitud y orientación del gradiente.
 - Los gradientes se orientan teniendo en cuenta la orientación del punto de interés.
 - La región del punto de interés se divide en una grilla de por ejemplo 4x4.
 - En cada celda de la grilla se calcula un histograma de orientaciones con 8 bins.
 - Nos da un vector de 128 dimensiones, 4x4 regiones por 8 bins.

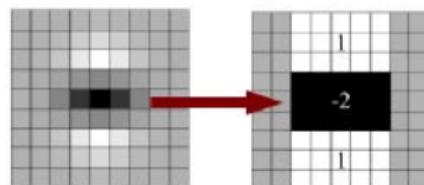


Descriptor SIFT

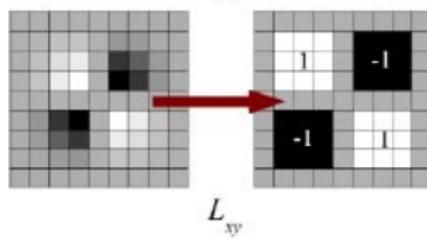
```
[ 19.    3.    0.    0.   10.   72.   66.    2.    2.    0.    0.  
  0.   71.  134.    9.    0.    0.    0.    0.    0.   13.   26.    0.  
  0.    0.    0.    0.    0.    0.    0.    0.   59.   79.   36.    9.  
  8.   11.   30.   44.  134.   80.   20.    0.    0.   69.  134.   45.  
 19.    2.    0.    0.   65.  134.   17.    0.    0.    0.    0.    0.  
  0.    1.    7.    1.   44.   15.   16.   25.   17.    6.   95.   70.  
134.   99.    6.    0.    0.    5.   35.   61.   47.   97.   38.    2.  
  1.   21.   53.   10.    8.   19.    9.    0.    0.    1.    7.    2.  
  2.    0.    0.    0.    5.   75.  134.   92.   45.   38.    1.    0.  
16.  109.   77.   23.   24.  115.   26.    0.    2.    4.    1.    2.  
46.   66.    7.    0.    0.    0.    0.    0.]
```

SURF

- Speeded Up Robust Features
- Versión rápida de SIFT
- Alrededor de 3 veces más rápido
- Se aproxima el LoG con un Box Filter (filtro de caja)
 - Puede ser calculado con imágenes integrales
 - Es fácilmente paralelizable (escalas)



$$L_{yy}$$



$$L_{xy}$$

Ejemplo de uso de SURF - Código

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('img2.ppm')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

surf = cv2.xfeatures2d.SURF_create()
kp, dscr = surf.detectAndCompute(gray, None)

cv2.drawKeypoints(img, kp, img)

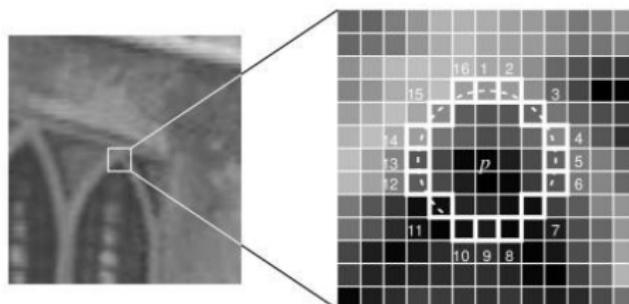
cv2.imwrite('surf_keypoints.png', img)

cv2.imshow('surf_keypoints', img)
cv2.waitKey()
```

Ejemplo de uso de SURF - Puntos clave



- Features from Accelerated Segment Test
- SURF sigue siendo lento para aplicaciones en tiempo real
- Está basado en Aprendizaje de Máquina (Machine Learning)
- Es solamente un detector, no descriptor
- No es invariante ante rotaciones ni escalas
 1. Se selecciona un pixel p con intensidad I_p
 2. Se determina un valor de umbral t
 3. Se considera un círculo de 16 pixeles alrededor de p
 4. Si existen $n = 12$ pixeles contiguos del círculo más brillosos que $I_p + t$ o menos brillosos que $I_p - t$

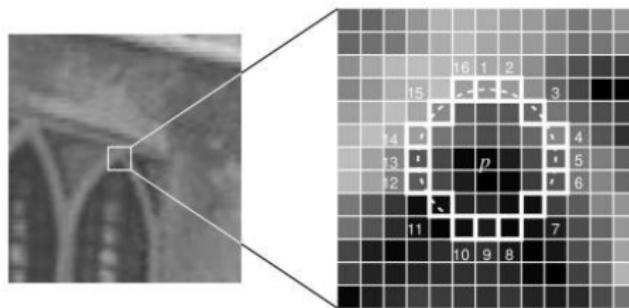


Algoritmo para el cómputo de FAST

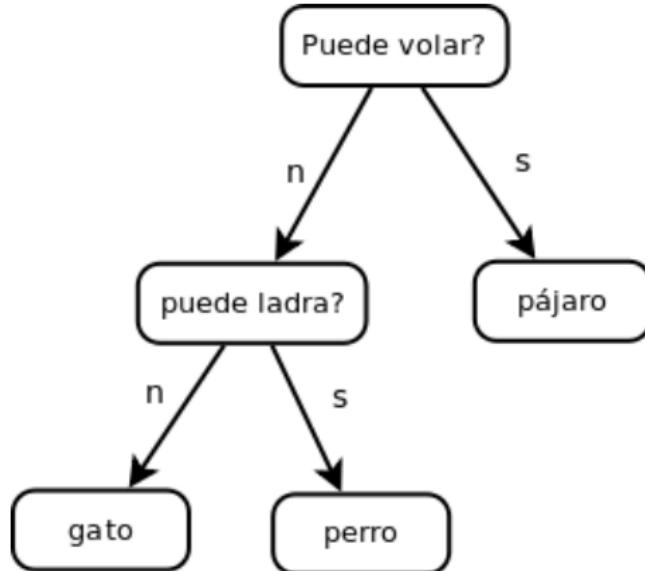
- El algoritmo contiene 3 pasos
 - Primero se hace una prueba rápida
 - Luego se aplica un detector basado en Machine Learning
 - Por último se aplica un método de supresión de no máximo

Algoritmo para el cómputo de FAST

- Prueba rápida
 - Comparando si los pixeles 1 y 9 están fuera del umbral
 - Luego comparando los pixeles 5 y 13

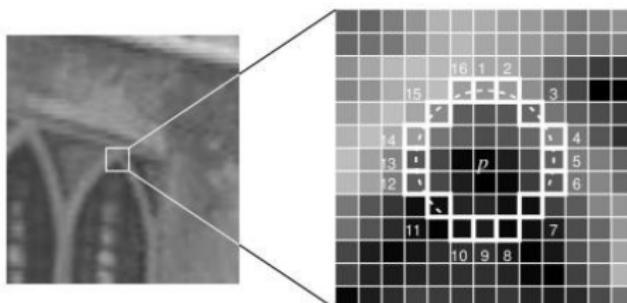


Clasificador de tipo Árbol



Algoritmo para el cómputo de FAST

- Detector basado en Machine Learning
 - Se selecciona un conjunto de imágenes
 - Se ejecuta FAST en forma bruta
 - Para cada punto se guardan los 16 píxeles usando 3 estados
 - Más brilloso
 - Similar
 - Menos brilloso
 - Y se guarda una variable booleana que indica si es corner o no
 - Con este conjunto se entrena un clasificador de tipo árbol que se aplica a los puntos que pasaron la prueba rápida



Algoritmo para el cómputo de FAST

- Supresión de no máximo
 - Se computa una función de score para todos los puntos
 - La función de score es la suma de los valores absolutos de la resta entre el pixel p y los pixeles continuos
 - Se eliminan los puntos cuya función de score es menor que la de sus vecinos

Ejemplo de uso de FAST - Código

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('img2.ppm')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

fast = cv2.FastFeatureDetector_create(threshold=100)

kp = fast.detect(gray, None)
cv2.drawKeypoints(img, kp, img, color=(255, 0, 0))

cv2.imwrite('fast_keypoints.png', img)

cv2.imshow('fast_keypoints', img)
cv2.waitKey()
```

Ejemplo de uso de FAST - Puntos clave



Esquema de Matching con SIFT



Esquema de Matching con SIFT

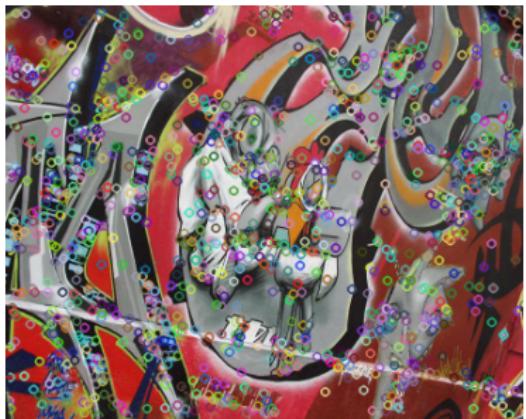
- Detección de extremos en espacio de escala
- Localización del punto clave
- Asignación de una orientación
- Cómputo del descriptor en cada punto clave
- Matching de descriptores entre imágenes
 - Se computan los puntos clave en las imágenes
 - Se computan los descriptores de los puntos claves
 - Los puntos claves de una imagen se matchean con los de la otra imagen buscando su vecino más cercano
 - Se calcula el cociente entre las distancias a los 2 vecinos más cercanos de cada descriptor y si es mayor que 0.8 se descarta el match (conocido como criterio de Lowe)

Práctica 0 - Matching usando SIFT

Alineación de imágenes usando SIFT

1. Capturar dos imágenes con diferentes vistas del mismo objeto
2. Computar puntos de interés y descriptores en ambas imágenes
3. Establecer matches entre ambos conjuntos de descriptores
4. Eliminar matches usando criterio de Lowe
5. Computar una homografía entre un conjunto de puntos y el otro
6. Aplicar la homografía sobre una de las imágenes y guardarla en otra (mezclarla con un alpha de 50 %)

Práctica 0 - Puntos claves en ambas imágenes



Práctica 0 - Matches entre ambas imágenes



Práctica 0 - Mezcla de ambas imágenes



Práctica 0 - Código de Ayuda

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import numpy as np
import cv2

MIN_MATCH_COUNT = 10

img1 = # Leemos la imagen 1
img2 = # Leemos la imagen 2

dscr = # Inicializamos el detector y el descriptor

kp1, des1 = # Encontramos los puntos clave y los descriptores con SIFT en la imagen 1
kp2, des2 = # Encontramos los puntos clave y los descriptores con SIFT en la imagen 2

matcher = cv2.BFMatcher(cv2.NORM_L2)

matches = matcher.knnMatch(des1, des2, k=2)

# Guardamos los buenos matches usando el test de razón de Lowe
good = []
for m, n in matches:
    if m.distance < 0.7*n.distance:
        good.append(m)

if (len(good) > MIN_MATCH_COUNT):
    src_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1, 1, 2)

    H, mask = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC, 5.0) # Computamos la homografía con RANSAC

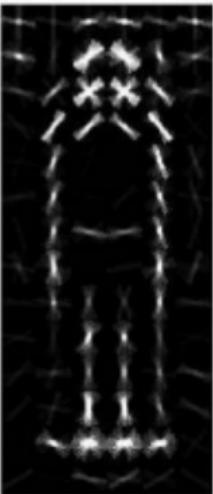
    wimg2 = # Aplicamos la transformación perspectiva H a la imagen 2

    # Mezclamos ambas imágenes
    alpha = 0.5
    blend = np.array(wimg2 * alpha + img1 * (1 - alpha), dtype=np.uint8)
```

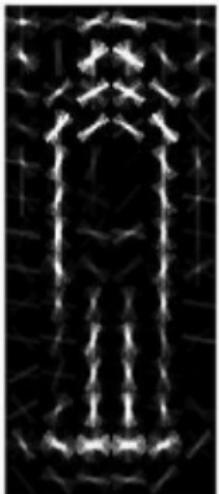
Detección de personas usando HOG - HOG sobre una persona



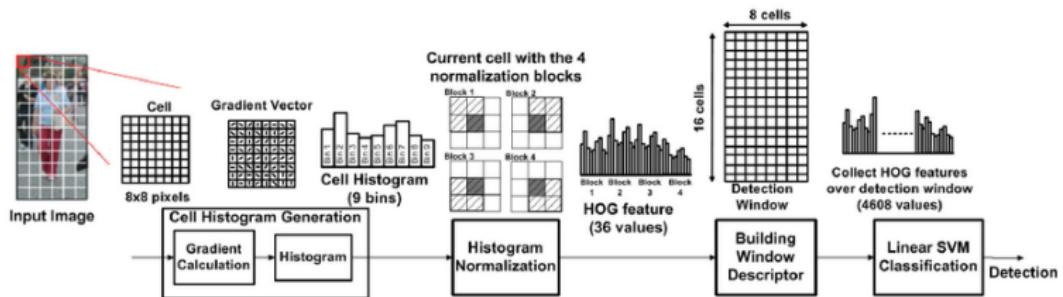
(a)



(b)



Detección de personas usando HOG - Algoritmo



Detección de personas usando HOG - Código

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('people.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

hog = cv2.HOGDescriptor()
hog.setSVMClassifier(cv2.HOGDescriptor_getDefaultPeopleDetector())

detections, weights = hog.detectMultiScale(gray)

detections_rectangles = detections.tolist()

for x, y, w, h in detections_rectangles:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)

cv2.imwrite('detected_people.png', img)

cv2.imshow('detections', img)
cv2.waitKey()
```

Resultado de la detección

