

EXTRACCIÓN CARACTERÍSTICAS

Módulo features2D (e7.py)

- Puntos Característicos: SIFT/SURF/ORB
- Descriptores: SIFT/SURF/ORB

https://docs.opencv.org/4.10.0/d5/d51/group_features2d_main.html

https://docs.opencv.org/4.10.0/d3/df6/namespacecv_1_1xfeatures2d.html

https://docs.opencv.org/4.10.0/db/d27/tutorial_py_table_of_contents_feature2d.html

Extracción Características: módulo features2D

- Ejemplo (**e7.py**): partiremos del código **e1.py**
- Detectores:
 - Lista de puntos: tuplas/listas de objetos **cv.KeyPoint**
 - **pt**: tuple (x,y), **size**: float, **angle**: float, **response**: float, **octave**:int, **class_id**:int
 - **cv.KeyPoint()** → <KeyPoint object>
 - **cv.KeyPoint(x, y, size[, angle[, response[, octave[, class_id]]]])** → <KeyPoint object>
 - **SIFT**, **ORB**, AKAZE, KAZE, FAST, MSER, GFTT [, HARRIS], AGAST, BRISK, SimpleBlob
 - **SURF**, STAR, MSD, TBMR, HARRISLP ([modulo xfeatures2D – opencv-contrib](#))
- Descriptores: Vectores de características (por filas): **ndarray(np,nc)**
 - **SIFT**, **ORB**, AKAZE(MLDB), KAZE(MSURF), BRISK

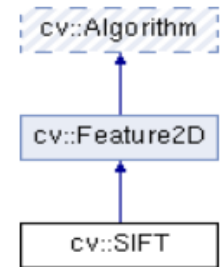
([modulo xfeatures2D](#))

 - **SURF**, FREAK, VGG, BRIEF, DAISY
LUCID, LATCH, BOOST, BEBLID

```
Capturing images.
Hit q/Q to exit.
Detected: 500 points
Descriptor Dimension: 32
[168, 114, 58, 109, 96, 14, 115, 120, 121, 10, 118, 88, 95, 27, 11, 48, 180, 17,
70, 106, 224, 180, 49, 15, 231, 217, 32, 1, 66, 128, 106, 1631_
```



Extracción Características: módulo features2D



- Ejemplo (**e7.py**): extracción de características
- Clase: **SIFT**: **cv.SIFT**

- `cv.SIFT.create([, nfeatures=0 [, nOctaveLayers=3 [, contrastThreshold=0.04 [, edgeThreshold=10 [, sigma=1.6]]]])` → detector (cv.SIFT)

- Clase base: **cv.Feature2D**

Detectar Puntos:

- `cv.Feature2D.detect(image [, mask])` → keypoints (tupla de cv.KeyPoint)
- `cv.Feature2D.detect(images [, masks])` → keypoints
 - mask: 8-bit binary matrix with non-zero values in the region of interest

Calcular Descriptores:

- `cv.Feature2D.compute(image, keypoints[, descriptors])` → keypoints, descriptors
- `cv.Feature2D.compute(images, keypoints[, descriptors])` → keypoints, descriptors

Si no existe el descriptor para un punto se elimina el keypoint.

También se pueden añadir nuevos (múltiples orientaciones dominantes)

descriptors: ndarray(np,ds)

- `cv.Feature2D.detectAndCompute(image, mask[, descriptors[, useProvidedKeypoints=False]])`
→ keypoints, descriptors
- `cv.Feature2D.descriptorSize()` → retval
- `cv.Feature2D.descriptorType()` → retval

Extracción Características: módulo features2D

- Ejemplo (**e7.py**): **SIFT** partiremos del código **e1.py**

```
FEATURE_TYPE = 'SIFT'          # Default Feature Detector sift/surf/orb

# check command line parameters
parser = argparse.ArgumentParser(description='OpenCV example: Feature Points Detector')
parser.add_argument('-c', dest='cameraID', type=int, default=CAMERA_ID, metavar='id', help='camera id')
parser.add_argument('-f', dest='feature', type=str, default=FEATURE_TYPE, metavar='feature',
                    help='sift | surf | orb')

CAMERA_ID = parser.parse_args().cameraID
FEATURE_TYPE = parser.parse_args().feature.lower()          # feature (lowercase)
```

```
# Initialization section
if FEATURE_TYPE == 'sift':
    detector = cv.SIFT.create(100)

# Detector information
print(f"{FEATURE_TYPE=}")
print(f"{detector.descriptorSize()}=")
```

```
# processing section (image capture loop)
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)

keypoints = detector.detect(gray_image)
keypoints, descriptors = detector.compute(gray_image, keypoints)
```

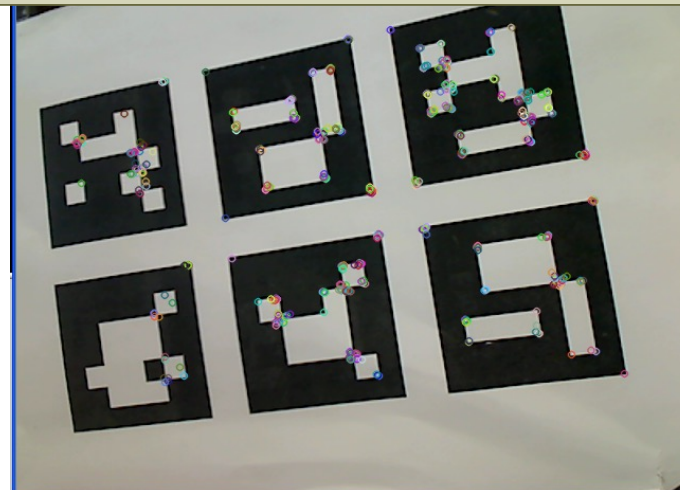
Extracción Características: módulo features2D

- Ejemplo (**e7.py**): Dibujar puntos
 - `cv.drawKeypoints(image, keypoints, outImage[, color=-1 [, flags]])` → `outImage`
 - **outImage**: *None* (DEFAULT flags), Imagen de fondo (DRAW_OVER_OUTIMG)
 - **color**: tupla, (-1 random)
 - **flags**: `cv.DRAW_MATCHES_FLAGS_DEFAULT`, (crea *outimage* con *image* como fondo)
`cv.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG`,
`cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS`,
`cv.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS`

```
# Draw keyPoints
```

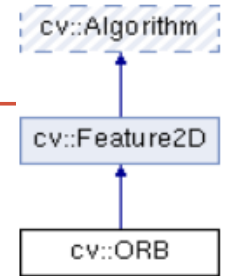
```
dispimage = cv.drawKeypoints( capture, keypoints, outImage=None,  
                             flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS )
```

```
cv.imshow(WINDOW_CAMERA1, dispimage) # Display the resulting frame
```



Extracción Características: módulo features2D

- Ejemplo (**e7.py**): extracción de características
- Clase: **ORB**: **cv.ORB**



- **cv.ORB.create**([, nfeatures=500 [, scaleFactor=1.2 [, nlevels=8 [, edgeThreshold=31 [, firstLevel=0 [, WTA_K=2[, scoreType=cv.ORB_HARRIS_SCORE [, patchSize=31 [, fastThreshold=20]]]]]]]]) → detector (cv.ORB)
- Métodos:
 - **cv.ORB.setMaxFeatures**(maxFeatures)

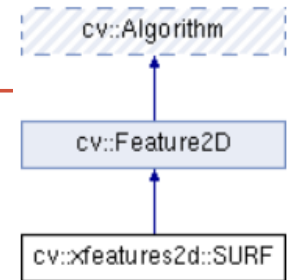
```
# Initialization section
elif FEATURE_TYPE == 'orb':
    detector = cv.ORB.create(nfeatures=100)
```

```
# processing section (image capture loop)
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)

keypoints = detector.detect(gray_image)
keypoints, descriptors = detector.compute(gray_image, keypoints)
```

Extracción Características: módulo features2D

- Ejemplo (**e7.py**): extracción de características
- Clase: **SURF**: **cv.xfeatures2d.SURF**
(modulo xfeatures2D – opencv-contrib) (patentado)



SURF: Precisa compilacion con **OPENCV_ENABLE_NONFREE** flag

- `cv.xfeatures2d.SURF.create([, hessianThreshold=100 [, nOctaves=4 [, nOctaveLayers=3 [, extended=False [, upright=False]]]])` → detector (cv. xfeatures2D_SURF)
- Métodos:
 - `cv.xfeatures2d.SURF.setExtended(extended)` Tamaño descriptor → True: 128, **False**: 64
 - `cv.xfeatures2d.SURF.setHessianThreshold(hessianThreshold)`

```

# Initialization section
elif FEATURE_TYPE == 'surf':
    detector = cv.xfeatures2d.SURF.create(hessianThreshold=400)
  
```

```

# processing section (image capture loop)
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)

keypoints = detector.detect(gray_image)
keypoints, descriptors = detector.compute(gray_image, keypoints)
  
```

Extracción Características: módulo features2D

- Resumen Detectores y Descriptores

- **Detectores:**

- **cv.SIFT.create()**
- **cv.ORB.create()**
- **cv.AKAZE.create()**
- **cv.KAZE.create()**
- **cv.BRISK.create()**
- **cv.FastFeatureDetector.create()**
- **cv.MSER.create()**
- **cv.GFTTDetector.create()**
- **cv.AgastFeatureDetector.create()**
- **cv.SimpleBlobDetector.create()**

- Paquete opencv-contrib-python:

- **cv.xfeatures2d.SURF.create()**
- **cv.xfeatures2d.StarDetector.create()**
- **cv.xfeatures2d.MSDDetector.create()**
- **cv.xfeatures2d.TBMR.create()**

- **Descriptores:**

- **cv.SIFT.create()**
- **cv.ORB.create()**
- **cv.AKAZE.create()**
- **cv.KAZE.create()**
- **cv.BRISK.create()**

- Paquete opencv-contrib-python:

- **cv.xfeatures2d.SURF.create()**
- **cv.xfeatures2d.BEBLIB.create()**
- **cv.xfeatures2d.BoostDesc.create()**
- **cv.xfeatures2d.DAISY.create()**
- **cv.xfeatures2d.FREAK.create()**
- **cv.xfeatures2d.LATCH.create()**
- **cv.xfeatures2d.LUCID.create()**
- **cv.xfeatures2d.VGG.create()**

CORRESPONDENCIA CARACTERÍSTICAS

Módulo features2D (e7b.py)

DescriptorMatcher

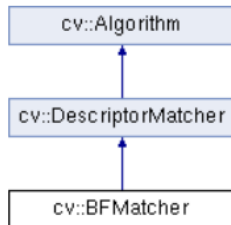
https://docs.opencv.org/4.10.0/d8/d9b/group_features2d_match.html

https://docs.opencv.org/4.10.0/dc/dc3/tutorial_py_matcher.html

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Matchers: cv.**BFMatcher** (Brute Force) / cv.**FlannBasedMatcher**

- `cv.BFMatcher.create([, normType=cv.NORM_L2[, crossCheck=False]])`
→ <cv.BFMatcher object>

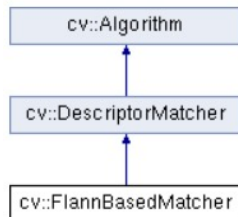


normType: parámetro medida de distancia

- cv.NORM_L1, cv.NORM_L2 -> SIFT, SURF
- cv.NORM_HAMMING, cv.NORM_HAMMING2 -> ORB, BRIEF, BRISK

crossCheck : (bool) correspondencia valida si coinciden en los dos sentidos

- `cv.FlannBasedMatcher.create()` → <cv.FlannBasedMatcher object>



Por defecto precisa que el descriptor sea de tipo **CV_32F** (**np.float32**)

Para Descriptores **binarios** es preciso configurar los parámetros de la clase con el constructor por defecto:

- `cv.FlannBasedMatcher([, indexParams[, searchParams]])` → <cv.FlannBasedMatcher object>

`index_params = dict(algorithm = 6, table_number = 6, key_size = 12, multi_probe_level = 3)`

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Inicialización:

```
MATCHER_TYPE = 'bf'          # Default Feature Matcher bf/flann

# command line parameters
parser.add_argument('-m', dest= 'matcher', type=str, default= MATCHER_TYPE, metavar= 'matcher',
                    help= 'bf (BruteForce) | flann')
MATCHER_TYPE = parser.parse_args().matcher.lower()      # feature (lowercase)
```

```
# Init Feature Matcher
if MATCHER_TYPE == 'flann':
    if FEATURE_TYPE == 'orb': # Binary descriptor -> NORM_HAMMING
        index_params = dict(algorithm = 6, table_number = 6, key_size = 12, multi_probe_level = 3)
        matcher = cv.FlannBasedMatcher(index_params)
    else:
        matcher = cv.FlannBasedMatcher.create()

else: # Brute Force Matcher
    if FEATURE_TYPE == 'orb': # Binary descriptor -> NORM_HAMMING
        matcher = cv.BFMatcher.create(normType=cv.NORM_HAMMING, crossCheck=True)
    else:
        # Scalar descriptor -> NORM_L2
        matcher = cv.BFMatcher.create(normType=cv.NORM_L2, crossCheck=True)

print(f"{MATCHER_TYPE=}")
new_reference = True      # Take new reference image
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Procesamiento:

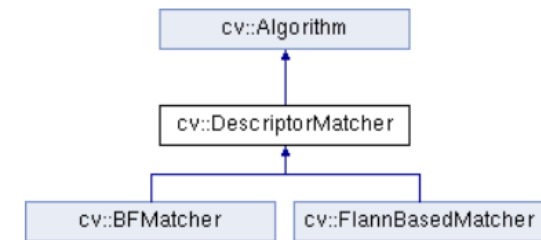
```
# processing section (image capture loop)
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)

keypoints = detector.detect(gray_image)
keypoints, descriptors = detector.compute(gray_image, keypoints)
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Matchers: cv.**BFMatcher** (Brute Force) / cv.**FlannBasedMatcher**

- Componentes clase **cv.DMatch**:
 - **queryIdx**: índice vector de descriptores 1 (cámara)
 - **trainIdx**: índice vector de descriptores 2 (referencia)
 - **distance**: distancia calculada entre vectores de descripción



- Métodos clase **cv.DescriptorMatcher**:
 - **cv.DescriptorMatcher.match**(queryDescriptors, trainDescriptors [, matches [, mask]])
→ matches (list/tuple of cv.DMatch)
 - **cv.DescriptorMatcher.knnMatch**(queryDescriptors, trainDescriptors, k [, mask[, compactResult]])
→ matches (list/tuple of k-tuple of cv.DMatch)
busca k vecinos más cercanos. Devuelve una lista/tuple, cada elemento: tupla de k cv.DMatch
 - **cv.DescriptorMatcher.radiusMatch**(queryDescriptors, trainDescriptors, maxDistance [, mask [, compactResult]]) → matches (list/tuple of k-tuple of cv.DMatch)
busca vecinos a distancia maxDistance. Devuelve lista/tupla, cada elemento: tupla de k cv.DMatch

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Bucle principal:

```
.....

# Store reference on first frame or Reset
if new_reference:
    keypoints_ref = keypoints           # Tuples are immutable so reference copy is valid
    descriptors_ref = descriptors.copy() # Clone ndarray (avoid reference copy)
    image_ref = capture.copy()         # Clone ndarray (avoid reference copy)
    new_reference = False

# Match descriptors.
matches = matcher.match(descriptors, descriptors_ref) # list of matches

# Sort matches in the order of their distance.
matches = sorted(matches, key=lambda x: x.distance)

.....
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Visualización Correspondencias:
 - `cv.drawMatches(img1, keypoints1, img2, keypoints2, matches1to2, outImg [, matchColor [, singlePointColor[, matchesMask[, flags]]]])` → outImg
 - `cv.drawMatchesKnn(img1, keypoints1, img2, keypoints2, matches1to2, outImg [, matchColor[, singlePointColor[, matchesMask[, flags]]]])` → outImg
 - **flags:** `cv.DRAW_MATCHES_FLAGS_DEFAULT`, (crea outimage con image como fondo)
`cv.DRAW_MATCHES_FLAGS_DRAW_OVER_OUTIMG`,
`cv.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS`

```
# Draw first 20 matches.
dispimage = cv.drawMatches(capture, keypoints, image_ref, keypoints_ref,
                           matches[:min(20,len(matches))], outImg=None,
                           flags=cv.DRAW_MATCHES_FLAGS_DEFAULT)

cv.imshow(WINDOW_CAMERA1, dispimage) # Display matches

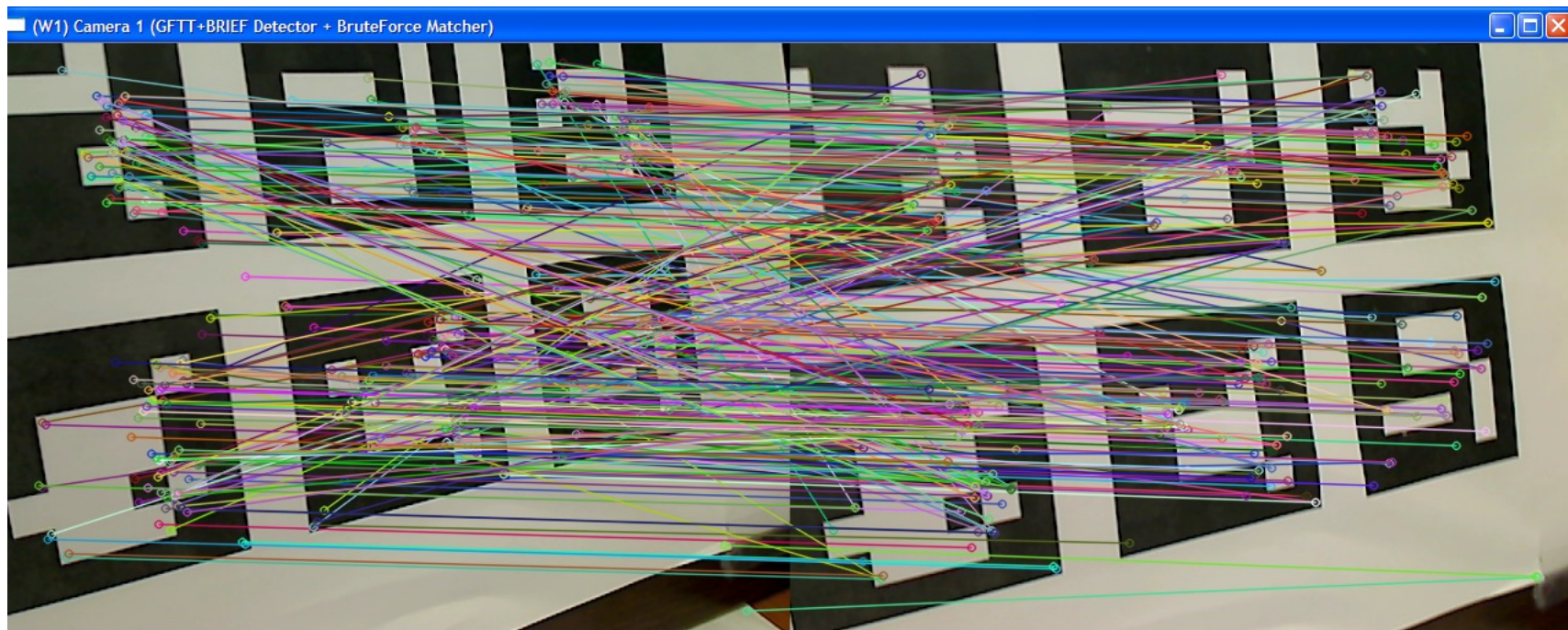
# check keystroke to exit (image window must be on focus)
key = cv.pollKey()
if key == ord('q') or key == ord('Q') or key == 27:
    break
elif key == ord('r') or key == ord('R') or key == ord(' '):
    new_reference = True # Reset Reference image
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7b.py**): extracción de características /matching
- Mostrar correspondencias:

```
# show matches on console
print(f"Left image points: {len(keypoints)} - Right image points: {len(keypoints_ref)}")
print(f"Matched points: {len(matches)}")

for i in range(min(10,len(matches))):
    print(f"-- Match [{i}] Keypoint Cam: {matches[i].queryIdx}", end="")
    print(f" -- Keypoint Ref: {matches[i].trainIdx} -- dist: {matches[i].distance}")
```



CORRESPONDENCIA CARACTERÍSTICAS

Módulo `features2D`. (`e7c.py`)

- Correspondencia entre dos cámaras
- `knnMatch()` obtener k correspondencias
- Radio test: compara las dos mejores

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia entre dos cámaras

```
CAMERA_ID = [0, 1]           # camera ids

# check command line parameters
parser.add_argument( '-c', dest='cameraID', type=int, default=CAMERA_ID[0], metavar='id',
                    help='camera id')

CAMERA_ID[0] = parser.parse_args().cameraID
CAMERA_ID[1] = parser.parse_args().cameraID + 1
```

```
# Init cameras
cameras = []
for id in CAMERA_ID:
    camera = cv.VideoCapture(id)
    if not camera.isOpened():
        print("you need to connect a camera, sorry.")
        exit()
    cameras.append(camera)
```

```
# free camera resources
for camera in cameras:
    if camera.isOpened(): camera.release()
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia de dos cámaras
- Procesamiento para cada cámara:

```
while True:
    # Capture frame-by-frame
    captures = []
    for camera in cameras:
        ret, capture = camera.read()
        if ret: captures.append(capture)

        ....

    gray_images = []
    for capture in captures:
        gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY)
        gray_images.append(gray_image)

    keypointsList = []
    descriptorsList = []
    for gray_image in gray_images:
        keypoints = detector.detect(gray_image)           # Tuple of keypoints
        keypoints, descriptors = detector.compute(gray_image, keypoints)

        keypointsList.append( keypoints )
        descriptorsList.append( descriptors )

    .....
```

Correspondencia Características: módulo features2D

- Ejemplo (**e7c.py**): correspondencia de dos cámaras
- Visualización correspondencias:

```
# knnMatch descriptors. list/tuple of k-tuple of matches
matches = matcher.knnMatch(descriptorsList[0], descriptorsList[1], k=2)

# Apply ratio test to the two best matches
goodMatches = []
for m1, m2 in matches: # tuple with two elements k==2
    if m1.distance/m2.distance < 0.75:
        goodMatches.append(m1)

# Sort matches in the order of their distance.
matches = sorted(goodMatches, key=lambda x: x.distance)

# Draw first 20 matches.
dispimage = cv.drawMatches(captures[0], keypointsList[0], captures[1], keypointsList[1],
    matches[:min(20,len(matches))], outImg=None, flags=cv.DRAW_MATCHES_FLAGS_DEFAULT)
```

