

EXTRACCIÓN DE CONTORNOS

Módulo imgproc (e4)

Listas de puntos enlazados

https://docs.opencv.org/4.10.0/d3/dc0/group_imgproc_shape.html

https://docs.opencv.org/4.10.0/d3/d05/tutorial_py_table_of_contents_contours.html

Extracción de Contornos: módulo imgproc

- Ejemplo (**e4.py**): (partiremos del código de e1.py)
- Extraer contornos de una imagen binaria (**Canny**):
 - Métodos:
 - `cv.findContours(image, int mode, int method [, contours [, hierarchy [, offset]])`
→ `contours, hierarchy`

Modes: `cv.RETR_EXTERNAL`, `cv.RETR_LIST`, `cv.RETR_CCOMP`, `cv.RETR_TREE`,

Methods: `cv.CHAIN_APPROX_NONE`, `cv.CHAIN_APPROX_SIMPLE` ,
`cv.CHAIN_APPROX_TC89_L1`, `cv.CHAIN_APPROX_TC89_KCOS`

contours : tupla de `ndarrays(np,1, 2)`, cada punto una fila, 3^{er} eje (**x,y**)

hierarchy: `ndarray(1,np,4)` `hierarchy[0][i][0]`→next , `[1]`→previous, `[2]`→child, `[3]`→parent
0-based indices in contours

```
gray_image = cv.cvtColor( capture, cv.COLOR_BGR2GRAY )    # transforms to gray level
edge_image = cv.Canny( gray_image, threshold1=80, threshold2=150 )    # Canny detector

# find contours
contours, hierarchy = cv.findContours ( edge_image, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE )

# Draw contours
for idx in range(len(contours)):
    color = np.random.randint(low=0, high=256, size=3).tolist()
    cv.drawContours(capture, contours, idx, color, thickness=1, lineType=cv.LINE_8)
```

EJERCICIO 4b

Interpolación y filtrado de contornos (e4b.py)

- Aproximación poligonal
- Propiedades de contornos
- Filtrado de contornos
- Persistencia: almacenamiento

Extracción de Contornos: módulo imgproc

- Ejercicio (**e4b.py**): extracción de contornos (listas de puntos enlazados)
- Interpolar (aproximación poligonal) y filtrar contornos:
 - Métodos:
 - **cv.approxPolyDP(curve, epsilon, closed [, approxCurve])** → **approxCurve**
 - epsilon: (float) approximation accuracy
 - closed: (bool). If True its first and last vertices are connected
 - **cv.arcLength(curve, closed)** → **retval**
 - **cv.contourArea(contour[, oriented=False])** → **retval**
 - **cv.isContourConvex(contour)** → **retval**

```
# Filter out non rectangular contours
contours_filtered = list() # empty list (remember tuples are immutable)

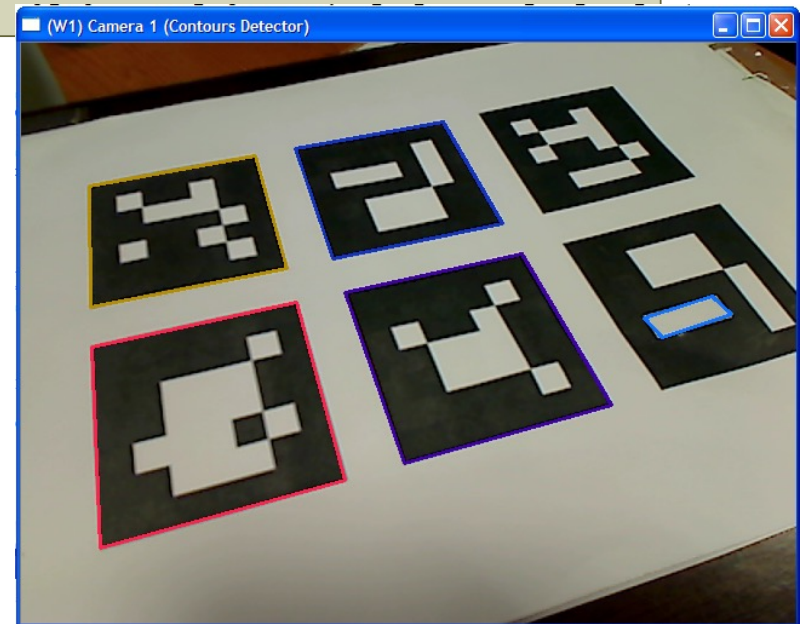
for cnt in contours:
    # approximate to a polygon (epsilon 5% of perimeter)
    approxCurve = cv.approxPolyDP(cnt, cv.arcLength(cnt, True)*0.05, True)

    # check that the polygon has 4 points, and is convex and big enough
    if ( approxCurve.shape[0] == 4 and cv.isContourConvex(approxCurve)
        and cv.contourArea(approxCurve) > 200 ):
        contours_filtered.append(approxCurve)
```

Extracción de Contornos: módulo imgproc

- Ejercicio (**e4b.py**): extracción de contornos (listas de puntos enlazados)
- Mostrar contornos:
 - `cv.drawContours(image, contours, contourIdx, color [, thickness=1, [lineType=cv.LINE_8 [, hierarchy [, maxLevel=INT_MAX, offset)`
 - `contourIdx` (-1) todos los contornos

```
# Draw contours
for idx in range(len(contours_filtered)):
    color = np.random.randint(low=0, high=256, size=3).tolist()
    cv.drawContours(capture, contours_filtered, idx, color, thickness=2, lineType=cv.LINE_AA)
```



EJERCICIO 4b

Persistencia Librería Python JSON (e4b.py)

- Almacenar contornos detectados
- Leer fichero de datos

```
{  
  "size": 3,  
  "contours": [[[[357, 262]], [[355, 288]], [[384, 290]], [[386, 263]]], [[120, 317]],  
               [[124, 439]], [[371, 443]], [[328, 261]]], [[119, 253]], [[120, 281]],  
               [[149, 281]], [[149, 254]]]]  
}
```

<https://docs.python.org/3/library/json.html>

Persistencia: JSON, python library

- Lectura/escritura de datos en ficheros /JSON:
 - Permite almacenar/leer los tipos base y listas/tuplas/diccionarios/conjuntos de forma transparente
 - Otras estructuras: ndarray → convertirla en **lista** con el método **.tolist()**

```
contours_json = list() # empty list
for cnt in contours_filtered:
    contours_json.append( cnt.tolist() )
```

- Importar librería:

```
import json
```

- Escribir fichero JSON:

- `json.dump(obj, fp)`
 - **obj**: pasaremos un diccionario { 'name': object }

```
# Writing contours to the file
with open('contours.json', 'w') as file:
    json.dump( { 'size': len(contours_json), 'contours': contours_json }, file )
```

Persistencia: JSON, python library

- Lectura/escritura de datos en ficheros /JSON:
- Leer fichero JSON:
 - `json.load(file)` → obj
 - obj: es un diccionario con los datos decodificados → { 'clave': valor }

```
# Reading contours from file
try:
    with open('contours.json') as file:
        data = json.load( file )
        print("Num. Contours:", data['size'])
        print(data['contours'])
except:
    print("File not valid")
```

EJERCICIO 4c (módulo imgproc)

Detección de rectas y círculos mediante transformada de Hough (e4c.py)

https://docs.opencv.org/4.10.0/dd/d1a/group_imgproc_feature.html

https://docs.opencv.org/4.10.0/d6/d10/tutorial_py_houghlines.html

https://docs.opencv.org/4.10.0/da/d53/tutorial_py_houghcircles.html

Extracción de Contornos: módulo imgproc

- Ejemplo (**e4c.py**): (partiremos del código de e1.py)
- Extraer líneas rectas de una imagen binaria (**Canny**):
 - `cv.HoughLines(image, rho, theta, threshold
[, lines [, srn [, stn [, min_theta [, max_theta]]]])` → lines

image: Binary image

lines: `ndarray(nl,1,2)` una línea por fila, una columna con una tupla de 2 elementos (ρ, θ)
 ρ distance from the coordinate origin (0,0) (top-left corner of the image).
 θ is the line rotation angle in radians ($0 \sim$ vertical line, $\pi/2 \sim$ horizontal line).

rho: Distance resolution of the accumulator in pixels.

theta: Angle resolution of the accumulator in radians.

threshold: Only those lines are returned that get enough votes ($>$ threshold).

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

```
gray_image = cv.cvtColor( capture, cv.COLOR_BGR2GRAY ) # transforms to gray level
edge_image = cv.Canny( gray_image, threshold1=80, threshold2=150 ) # Canny detector

# find straight lines (Hough Transform)
lines = cv.HoughLines(edge_image, rho=1, theta=np.pi/180, threshold=200)
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**e4c.py**):
- Visualización Líneas:

```
# Draw the lines
if lines is not None:
    for line in lines:
        rho, theta = line[0]    # rho = x*cos(theta)+ y*sin(theta)
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho           # central point (rho is orthogonal to the line)
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b)) # two far away points of the line
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        # cv.line do automatic clipping to the image size

        color = np.random.randint(low=0, high=256, size=3).tolist()
        cv.line(capture, (x1, y1), (x2, y2), color, thickness=2)
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**e4c.py**):
- Transformada Hough Círculos:
 - `cv.HoughCircles(image, rmethod, dp, minDist
[, circles [, param1 [, param2 [, minRadius[, maxRadius]]]])` → circles

image: grayscale image

circles: ndarray(1, nc, 3) 1 fila, un círculo por columna con una tupla de 3 elementos (xc,yc,ρ)
(xc,yc); centro del círculo, ρ radio

rmethod: cv.HOUGH_GRADIENT, cv.HOUGH_GRADIENT_ALT.

dp: Inverse ratio of the accumulator resolution to the image resolution.

minDist: Minimum distance between the centers of the detected circles.

param1: higher threshold of Canny edge detector (the lower one is twice smaller).

param2: the accumulator threshold for the circle centers

$$r^2 = (x - x_c)^2 + (y - y_c)^2$$

```
# find circles (Hough Transform)
```

```
circles = cv.HoughCircles(gray_image, cv.HOUGH_GRADIENT, dp=1, minDist=20,  
param1=150, param2=30, minRadius=20, maxRadius=100)
```

Extracción de Contornos: módulo imgproc

- Ejemplo (**e4c.py**):
- Visualización Círculos:

```
# Draw detected circles
if circles is not None:
    circles = np.around(circles).astype(np.uint16)    # integer approximation
    for circle in circles[0, :]:
        x, y, r = circle
        color = np.random.randint(low=0, high=256, size=3).tolist()

        cv.circle(capture, (x, y), r, color, thickness=2)    # draw the outer circle
        cv.circle(capture, (x, y), 2, color, thickness=2)    # draw the center of the circle
```

EJEMPLO 5

Segmentación de regiones por Color

- Conversión de color
- Extracción de canales de una imagen
- Umbralización
- Filtrado morfológico
- Extracción de contornos
- Cálculo de momentos

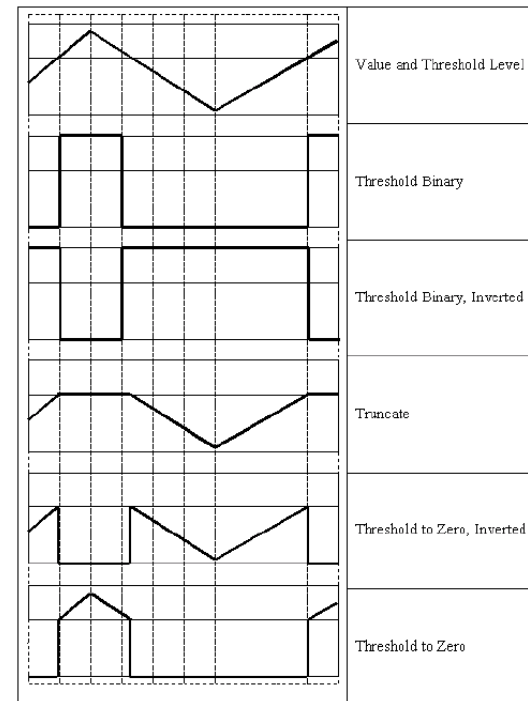
Ejercicio: segmentación de regiones por color

- Programa base: **e5.py** (partiremos del código de e3b.py)
- Módulos `imgproc / core`
- Funciones: Conversión de Color:
 - `cv.cvtColor(src, code [,dst, [, dstCn=0]])` → dst
 - Códigos:
 - `cv.COLOR_BGR2GRAY`, `cv.COLOR_GRAY2BGR`
 - `cv.COLOR_BGR2HSV`, `cv.COLOR_HSV2BGR`, `cv.COLOR_BGR2HSV_FULL`, `cv.COLOR_HSV2BGR_FULL`
 - `cv.COLOR_BGR2HLS`, `cv.COLOR_HLS2BGR`, `cv.COLOR_BGR2HLS_FULL`, `cv.COLOR_HLS2BGR_FULL`
 - `cv.COLOR_BGR2XYZ`, `cv.COLOR_XYZ2BGR`
 - `cv.COLOR_BGR2Lab`, `cv.COLOR_Lab2BGR` (escalado 0-1.0 float32 cv.CV_32F)
 - `cv.COLOR_BGR2Luv`, `cv.COLOR_Luv2BGR` (escalado 0-1.0 float32 cv.CV_32F)
 -
 - `cv.normalize(src, dst [, alpha [, beta [, norm_type [, dtype [, mask]]]])` → dst. / Numpy arithmetic
 - `cv.split(m [, mv])` → mv Tupla (a, b, c) / Numpy: `b = img[:, :, 0]`, `g = img[:, :, 1]`, `r = img[:, :, 2]`
 - `cv.merge(mv [, dst])` → dst / Numpy: `img = np.dstack((b,g,r))`



Ejercicio: segmentación de regiones por color

- Módulos imgproc /core
 - Umbralización:
 - `cv.threshold(src, thresh, maxval, type [, dst])` → `retval, dst`
 - **type:**
 - `cv.THRESH_BINARY`,
 - `cv.THRESH_BINARY_INV`
 - `cv.THRESH_TRUNC`
 - `cv.THRESH_TOZERO`
 - `cv.THRESH_TOZERO_INV`
 - **maxval:** valor alto de salida (BINARY)
 - **Cálculo umbral automático:**
 - (+ `cv.THRESH_OTSU`)



- `cv.inRange (src, lowerb, upperb [, dst])` → `dst`
 - **lowerb, upperb:** pueden ser imágenes, escalares o vectores

$$\text{dst}(I) = \text{lowerb}(I)_0 \leq \text{src}(I)_0 \leq \text{upperb}(I)_0$$

Ejercicio: segmentación de regiones por color

- Módulo imgproc

- Funciones:

- Morfología:

- `cv.erode(src, kernel [, dst [, anchor=(-1,-1) [, iterations=1 [, borderType [, borderValue]]]])`
→ dst
- `cv.dilate(src, kernel [, dst [, anchor [, iterations [, borderType [, borderValue]]]])` → dst
- `cv.morphologyEx(src, op, kernel [, dst [, anchor [, iterations [, borderType [, borderValue]]]])`
→ dst
 - **op:** `cv.MORPH_OPEN`, `cv.MORPH_CLOSE`, `cv.MORPH_GRADIENT`,
`cv.MORPH_TOPHAT`, `cv.MORPH_BLACKHAT`

kernel: `cv.getStructuringElement(shape, ksize [, anchor= (-1,-1)])`. → retval

shape: `cv.MORPH_CROSS`, `cv.MORPH_RECT`, `cv.MORPH_ELLIPSE`

ksize: (w, h)

- Operaciones lógicas:

- `cv.bitwise_and(src1, src2 [, dst [, mask=None]])` → dst
- `cv.bitwise_or(src1, src2 [, dst [, mask=None]])` → dst
- `cv.bitwise_xor(src1, src2 [, dst [, mask=None]])` → dst
- `cv.bitwise_not(src1, dst [, mask=None]])` → dst

Ejercicio: segmentación de regiones por color

- Módulo `imgproc`

- Momentos:

- `cv.moments (array [, binaryImage =False])` → moments

- **array**: np.int32 or np.float32
 - **binaryImage**: bool if True all non-zero image pixels are treated as 1's
 - **moments**: Diccionario {'m00': float,...}
 - spatial moments -> m00, m10, m01, m20, m11, m02, m30, m21, m12, m03
 - central moments -> mu20, mu11, mu02, mu30, mu21, mu12, mu03;
 - central normalized moments -> nu20, nu11, nu02, nu30, nu21, nu12, nu03

- `cv.HuMoments(moments [, hu])` → hu

$$m_{ji} = \sum_{x,y} (\text{array}(x, y) \cdot x^j \cdot y^i)$$

$$\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}}$$

$$\mu_{ji} = \sum_{x,y} (\text{array}(x, y) \cdot (x - \bar{x})^j \cdot (y - \bar{y})^i)$$

$$\nu_{ji} = \frac{\mu_{ji}}{m_{00}^{(i+j)/2+1}}$$

$$hu[0] = \eta_{20} + \eta_{02}$$

$$hu[1] = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$

$$hu[2] = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$

$$hu[3] = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$

$$hu[4] = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$hu[5] = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$hu[6] = (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

Ejercicio: segmentación de regiones por color

- Programa base: **e5.py** (partiremos del código de e3b.py)

```
AREA_MIN = 1000      # min area
HSV_THRESHOLD_LOW = [100, 30, 30] # Low and High HSV threshold
HSV_THRESHOLD_HIGH = [120, 255, 255] # Low and High HSV threshold

hsv = cv.cvtColor( capture, cv.COLOR_BGR2HSV_FULL ) # transforms to HSV

hue, sat, intensity = cv.split(hsv)

res = cv.inRange(hsv, tuple(HSV_THRESHOLD_LOW), tuple(HSV_THRESHOLD_HIGH))
```

```
# morphological filter(opening / closing)
kernel = cv.getStructuringElement(cv.MORPH_CROSS, (3, 3)) # [[0,1,0], [1,1,1], [0,1,0]]
res = cv.morphologyEx(res, cv.MORPH_OPEN, kernel, iterations=2)
res = cv.morphologyEx(res, cv.MORPH_CLOSE, kernel, iterations=2)

# find contours
edge_image = cv.Canny(res, threshold1=50, threshold2=200)

# find contours
contours, hierarchy = cv.findContours(edge_image, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
```

Ejercicio: segmentación de regiones por color

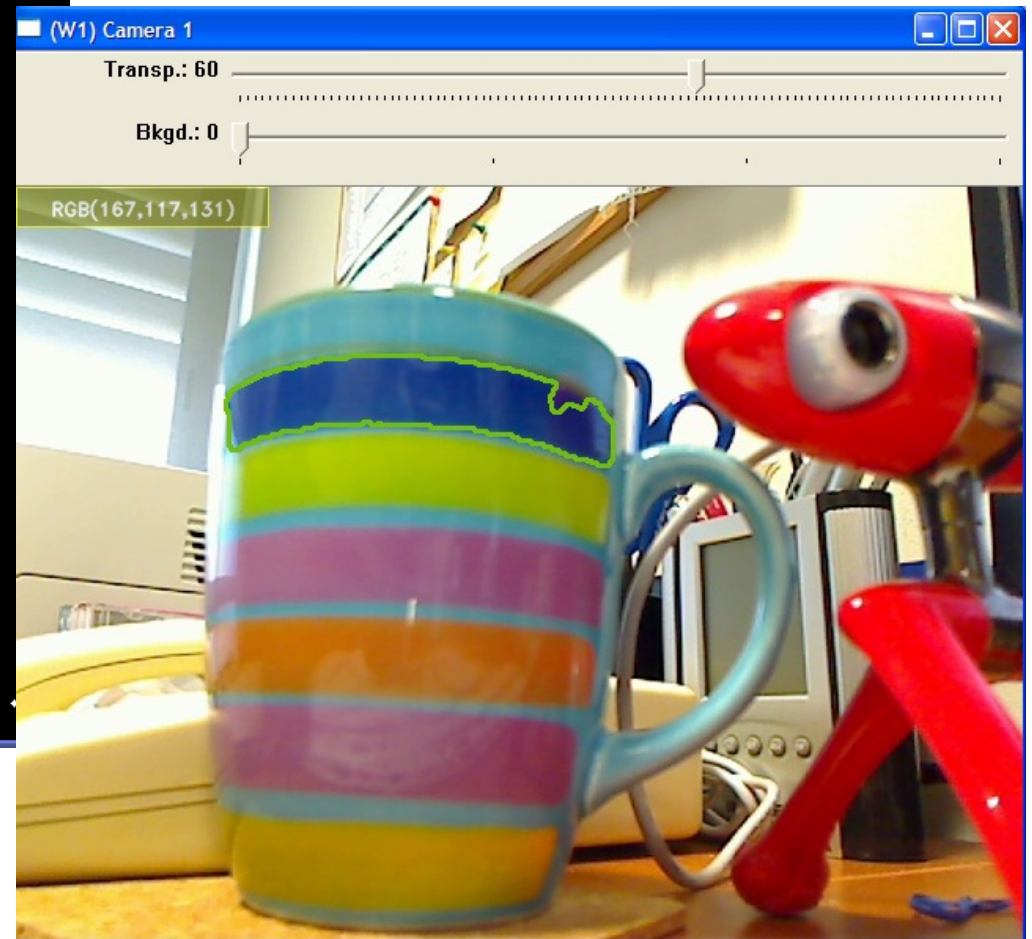
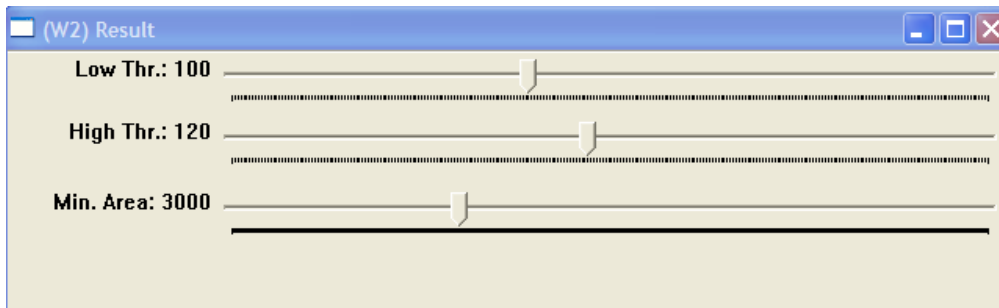
- Programa base: **e5.py** (partiremos del código de e3b.py)

```
# search for the biggest area contour
id_max = 0; area_max = 0.0
contours_filtered = list()
for idx, cnt in enumerate(contours):
    area = cv.contourArea(cnt)
    if area > AREA_MIN:
        contours_filtered.append(cnt)
        if area > area_max:
            id_max = idx; area_max = area
```

```
# Calculates Hu-moments for bigger contour (if any)
if len(contours_filtered) > 0:
    m = cv.moments(contours[id_max])
    hu = cv.HuMoments(m)
    # print(f"Hu-moments:{hu.transpose()}")
```

```
# Draw contours
for idx in range(len(contours_filtered)):
    color = np.random.randint(low=0, high=256, size=3).tolist()
    cv.drawContours(capture, contours_filtered, idx, color, thickness=2, lineType=cv.LINE_AA)
```

Ejercicio: segmentación de regiones por color



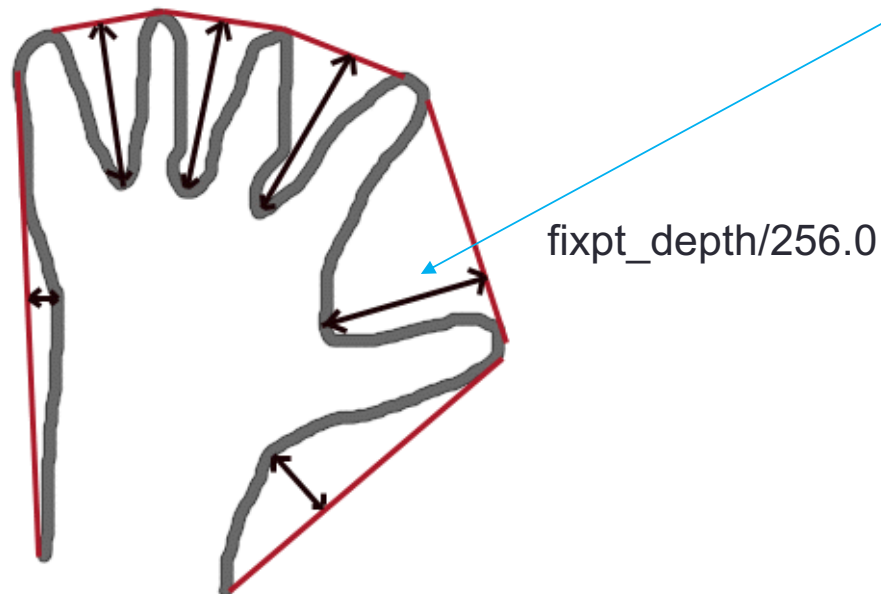
EJERCICIO 5b

Segmentación de regiones por Color

- Incorporar al ejemplo previo el procesamiento del convexHull para la detección de formas

Ejercicio: segmentación de regiones por color

- **Convex Hull**: envolvente convexa
 - `cv.convexHull (points [, hull [, clockwise =False [, returnPoints=True]])` → hull
 - **points**: `ndarray(npc,1,2)` (un solo contorno)
 - **hull**: (2 opciones):
 - **returnPoints=True**: Contorno (puntos): `ndarray(nph,1,2)`
 - **returnPoints=False**: Índices de los puntos del contorno: **lista de enteros**. `ndarray(nph,1)`
 - `cv::convexityDefects (contour, convexhul [, convexityDefects])` → `convexityDefects`
 - **convexhull** : Índices del contorno: **lista de enteros**
 - **convexityDefects** : `ndarray(nph,1,4)` : (start_index, end_index, farthest_pt_index, **fixpt_depth**)



Ejercicio: segmentación de regiones por color

- Programa base: **e5b.py** (partiremos del código de e5.py)
 - LUT:
 - `cv.LUT(src, lut [, dst]) → dst` $dst(I) \leftarrow lut(src(I))$
 - `cv.merge(mv [, dst]) → dst` combina una tupla de imágenes en una imagen en color
mv: input vector of matrices to be merged; all of the same size and the same depth.

```
# Normalize HUE channel (selected hue value in the middle(128))
HUE_CENTER = 10        # Hue valued to be centered in HUE normalization

lookUpTable = np.zeros(256, dtype=np.uint8)
for i in range(len(lookUpTable)):
    lookUpTable[i] = (i+128-HUE_CENTER) % 256

hue = cv.LUT(hue, lookUpTable)

hsv = cv.merge((hue, sat, intensity)) # Hue shifted HSV image for inRange segmentation

res = cv.inRange(hsv, tuple(HSV_THRESHOLD_LOW), tuple(HSV_THRESHOLD_HIGH))
```

Ejercicio: segmentación de regiones por color

- Programa base: **e5b.py** (partiremos del código de e5.py)
- Calcula Convex Hull, visualiza contorno y Convex-Hull:

```
# Calculates Convex Hull of the biggest contour (if any)
if len(contours_filtered)>0:
    # Convex Hull indexes clockwise
    convexhull = cv.convexHull(contours[id_max], clockwise=True, returnPoints=False)
    convexitydefects = cv.convexityDefects(contours[id_max], convexhull)

    # calculates mean of convexity defects
    convexity = 0.0 # convexity shape descriptor (mean of convexitydefects)
    for item in convexitydefects:
        convexity += item[0,3]/256.0

    convexity /= len(convexitydefects)
    print(f"Convexity Defect: {convexity}")
```

```
# Bold Draw bigger contour and convexhull
if len(contours_filtered)>0:
    cv.drawContours(displmage, contours, id_max, (0, 255, 0), thickness=1, lineType=cv.LINE_AA)
    # Draws convex hull for the biggest contour
    for id in convexhull[:-1,0]:
        cv.line(displmage, tuple(contours[id_max][id,0,:]), tuple(contours[id_max][id+1,0,:]),
                (0, 0, 255), thickness=2, lineType=cv.LINE_AA)
```