UNIVERSIDAD TECNOLÓGICA NACIONAL

VISIÓN POR COMPUTADORA

TRABAJO PRÁCTICO 9 - GRUPO 6

Medición de objetos

Amaya Matias
Lamas Matias
Navarro Facundo
Nobile Jonathan
Orellana Cristian

25 de junio de 2020

1. Introducción

Una forma de medir objetos de una imagen consiste en partir desde el conocimiento de las medidas de uno de los mismos, considerado como objeto patrón. Teniendo una referencia de medición, lo siguiente es detectar los objetos de la imagen. Para ello se emplean detectores de bordes.

En una imagen, si se encuentra una predefinida diferencia entre la intensidad de pixeles vecinos, se encuentra un borde. Este es el concepto sobre el que trabajan los detectores, aplicando derivadas para encontrar esas variaciones de intensidad.

Los detectores se pueden clasificar en detectores de primera derivada y detectores de segunda derivada. En la primera derivada de la intensidad, el borde esta representado por un máximo de la función f'. En la segunda derivada de la intensidad, el borde esta representado por un cero de la función f''. Ahora bien, ruido en la imagen puede cumplimentar las condiciones matemáticas mencionadas anteriormente. Lo cual implica potencial necesidad de implementar filtros para eliminar ese ruido y mejorar la performance de los detectores.

Una vez detectados los objetos, se aplica la matemática correspondiente para manifestar las medidas a partir de la medida patrón.

2. Desarrollo

Trabajaremos sobre la siguiente imagen, sabiendo que el papel glacé mide 10x10 cms.

Figura 1: Imagen fuente

Lo primero es poner en perspectiva a la imagen a partir del papel glacé.

```
def perspective(image, src_pts, dst_pts):
    (h, w) = image.shape[:2]
    M = cv2.getPerspectiveTransform(src_pts, dst_pts)
    rectified = cv2.warpPerspective(img, M, (w, h))
    return rectified
img = cv2.imread('prueba2.JPG')
dst_pts = np.float32([[53, 105], [253, 105], [253, 305], [53, 305]])
src_pts = np.float32([[55, 105], [248, 136], [246, 326], [28, 310]])
img = perspective(img, src_pts, dst_pts)
```

Como resultado, obtenemos:

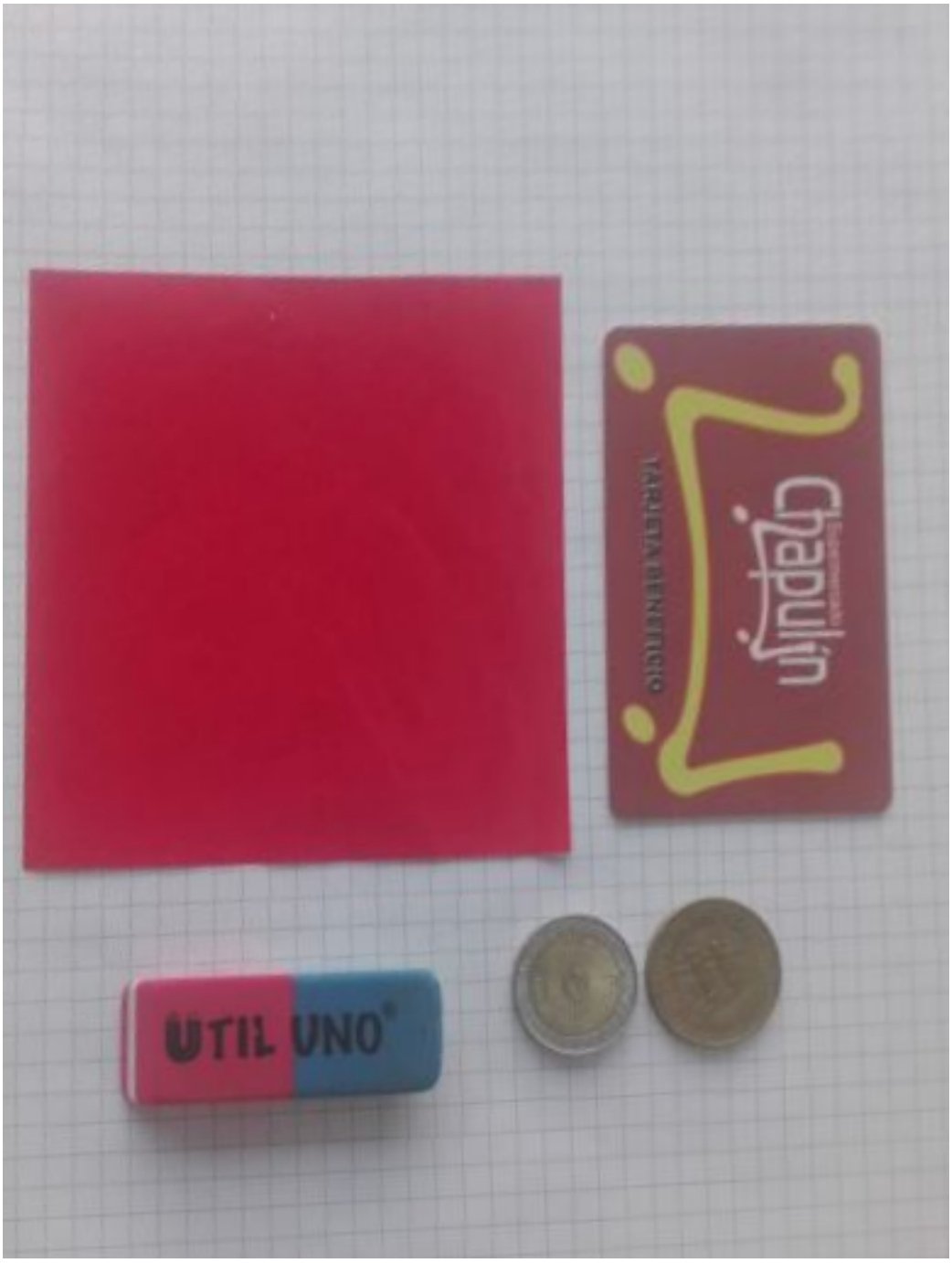

Figura 2: Imagen rectificada

Lo siguiente es binarizar la imagen, filtrarla y aplicarle los detectores de bordes.

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
gray_img = cv2.GaussianBlur(gray_img, (5, 5),11)
edges = cv2.Laplacian(gray_img, cv2.CV_8U, ksize=5)
edges = cv2.Canny(edges, 50, 300)
```

Como se ve en el fragmento de código, se implementa un filtro gaussiano y dos detectores de bordes, de primera derivada (Canny) y segunda derivada (Laplacian). Se utilizan dos para mejores resultados pero no es necesario. Con los correctos parámetros en el filtro y el detector, se puede cumplir el objetivo.

La función cv2.GaussianBlur está recibiendo tres argumentos. El primero es la imagen sobre la cual trabaja, el segundo es el tamaño del kernel (ancho,alto) y el tercero es la desviación estándar en direccion de x. Devuelve la imagen filtrada.

La función cv2.Laplacian recibe la imagen a operar, el tipo de dato de los pixeles de la imagen de salida y el tamaño del kernel del Sobel interno.

La función cv2.Canny recibe la imagen a operar y dos umbrales para ayudar a la función a definir que es un borde. Si la diferencia de intensidad entre pixeles vecinos es mayor al umbral superior, tenemos un borde. Si la diferencia es menor al umbral superior pero mayor al umbral inferior, tenemos un borde sólo si esos puntos de la imagen están conectados a puntos que sean bordes en cualquier otra parte de la imagen. Por último, si la variación de intensidad es menor al umbral inferior, nunca es un borde.

Por último, seleccionamos los contornos a trabajar e implementamos los algoritmos necesarios para medir y expresar los resultados.

```
contours, hierachy = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
patron=0.04926108374
for c in contours:
    if cv2.contourArea(c) <300: continue
    x, y, w, h = cv2.boundingRect(c)
    cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 2)
    base=round (w*patron ,1)
    altura=round (h*patron , 1)
    if base <4:
            radio = base/2
            cv2.putText(img, Rad: :.2f ".format(radio), (x-1, y+70), cv2.FONT_HERSHEY_COMPLEX,0.29,(0, 0,
255), 1)
    else:
            cv2.putText(img, ":.1f x :.1f cm ".format(base,altura), (x+9, y-10 ), cv2.FONT_HERSHEY_COMPLEX,0.4,(0,
0, 255), 1)
cv2.imshow('resultado', horizontal_concat)
```

La funcion cv2.findCountours almacena en contours los contornos encontrados a partir de los bordes que le pasamos como primer argumento provenientes del detector Canny. En el segundo argumento indicamos que solo nos interesan contornos externos. En el tercer argumento le indicamos a la función que no guarde todos los puntos del contorno, sino solo aquellos que se encuentran en los cambios de dirección del mismo.

Barremos contours con un ciclo for y trabajamos sobre cada contorno individual. Eliminamos todos los contornos que tengan un área menor a 300 (los cuadrados de la hoja cuadriculada). La función cv2.boundingRect devuelve la información necesaria para dibujar un rectángulo que envuelva al contorno en cuestión. Con ayuda de la variable patron, calculamos las medidas precisadas. Dicha variable es la relacion entre las medidas en cms y pixeles del papel glacé (cms/pixeles). Finalmente se escriben estos resultados en la imagen.

A continuación se muestra el resultado final.

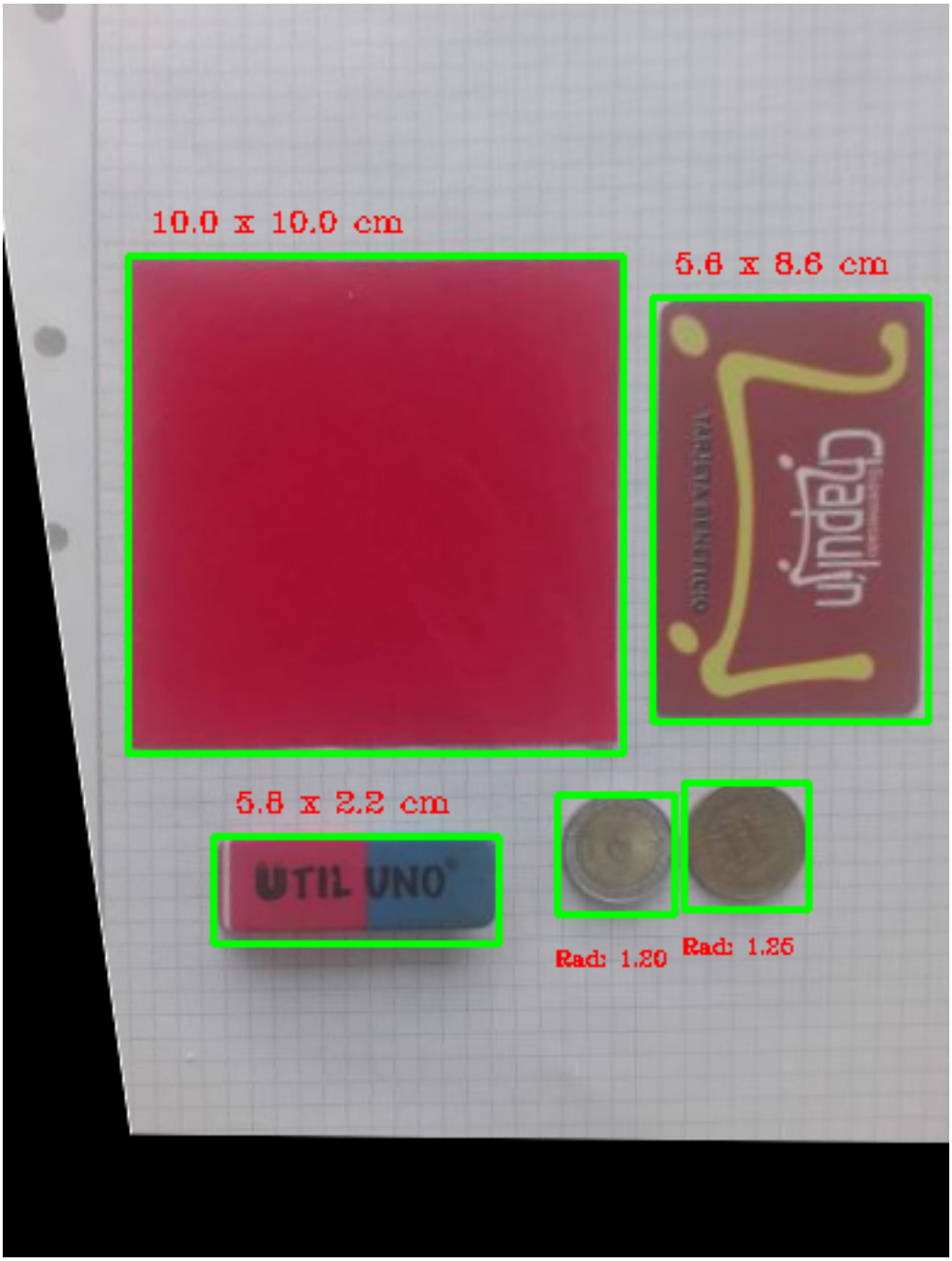

Figura 3: Resultado final

3. Conclusión

El método utilizado para resolver el problema está muy limitado a la imagen sobre la que trabaja. Implementarlo en otra imagen supone conocer medidas de la misma, manipular nuevamente parámetros de filtrado y umbralizado para la detección de los bordes, como así tambien nuevos algoritmos para eliminar contornos u objetos que no sean de interés.