Trabajo Práctico N° 2
Procesamiento de Imágenes - UNR

Monedas y Dados - Patentes

Integrantes:

Julian Ignacio Göttig - Legajo: **G-5922/6**

Fecha: **01/12/2025**

Detección y Clasificación de Monedas y Dados

Desarrollamos un algoritmo capaz de **detectar monedas y dados**, clasificarlos correctamente y, en el caso de los dados, **determinar cuántos puntos tienen en la cara superior**, todo a partir de una única imagen con iluminación no uniforme.

Qué hicimos

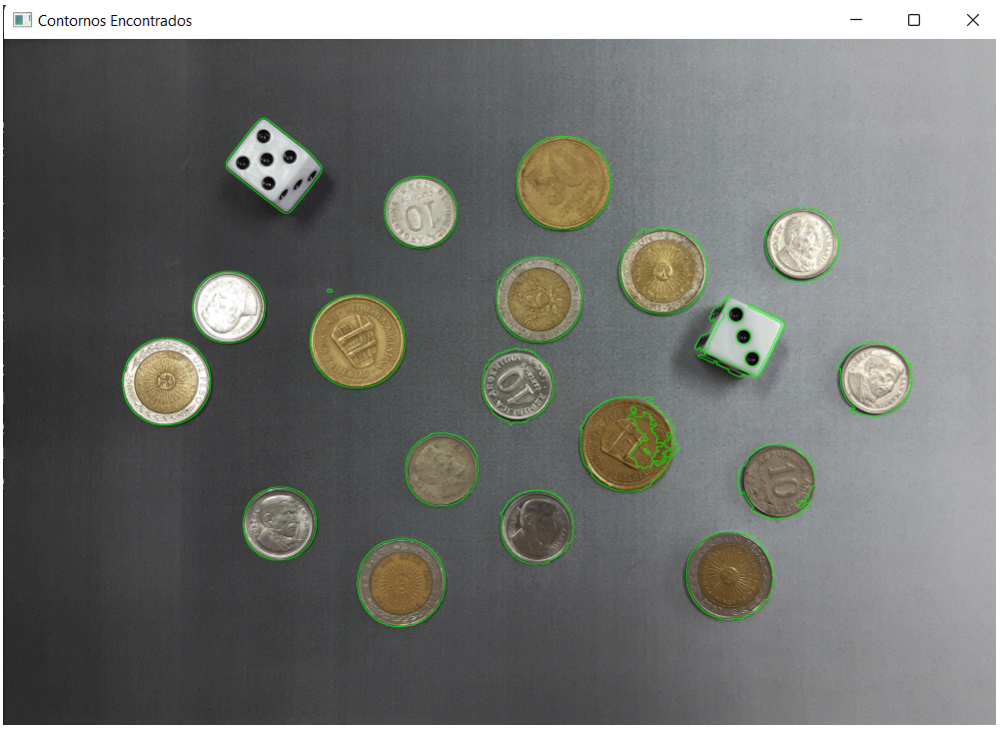
Primero cargamos la imagen y la convertimos a escala de grises para simplificar el procesamiento. Aplicamos un **suavizado Gaussiano** y el detector de bordes **Canny**, lo que permitió resaltar los límites de monedas y dados.
Después realizamos dos operaciones morfológicas importantes: una **dilatación**, para conectar bordes cercanos, y un **cierre** con un kernel grande, cuyo propósito fue rellenar huecos y dejar cada objeto completamente cerrado.

Con esta imagen ya “limpia”, detectamos contornos usando únicamente los contornos externos (RETR_EXTERNAL) y luego los filtramos por área.
Posteriormente clasificamos cada contorno en **moneda o dado**, analizando su **circularidad** y la **relación de aspecto**. Para las monedas, además clasificamos su valor aproximado según el **ancho del bounding box**, que actúa como una estimación del diámetro.

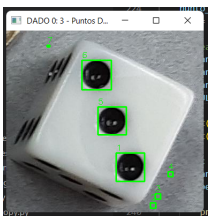
Finalmente, para cada dado obtuvimos un ROI y realizamos un proceso de segmentación específico para detectar sus puntos. Luego de filtrar contornos por **área** y **circularidad**, contamos los puntos detectados y los mostramos sobre la imagen.

Qué problemas tuvimos

Uno de los primeros problemas fue que los bordes de muchas monedas eran irregulares debido a la iluminación y a la textura del fondo, lo que hacía que los contornos quedarán abiertos o incompletos. Esto generaba errores tanto en la detección como en la clasificación. Como se ve en la siguiente imagen, en las **Primeras Etapas,** los bordes de las monedas no eran correctos



Otro problema fue distinguir correctamente los puntos de los dados: algunos se detectaban como manchas y otros se perdían por falta de contraste o por pequeñas rupturas en la segmentación. Además, cada dado ocupa un área diferente, por lo que el tamaño de los puntos varía según la región recortada.



También surgió un inconveniente inicial al intentar clasificar monedas pequeñas (10 centavos) que tenían contornos bastante deformados; su circularidad no alcanzaba para clasificarlas correctamente.

Qué técnicas aplicamos para resolverlo

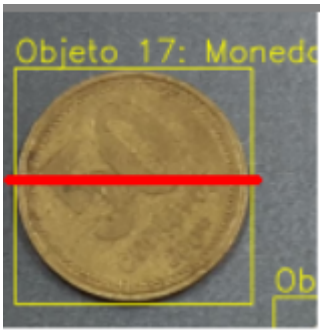
Para corregir la fragmentación de los contornos, aplicamos una **dilatación** y posteriormente una **clausura morfológica** con un kernel grande, lo que aseguró que cada moneda y cada dado quedaran como regiones sólidas y separadas del fondo.

Ademas, utilizamos la función de findContourns de RETR_EXTERNAL para localizar los bordes externos sin tomar los internos

Para distinguir monedas de dados utilizamos dos medidas geométricas:

- **Circularidad**: útil para detectar objetos casi circulares (monedas).
- **Relación de aspecto** del bounding box: los dados tienden a tener aspecto más cuadrado y circularidad más baja.

Para clasificar monedas por valor aplicamos una técnica simple pero efectiva: usar el **ancho del bounding box** como aproximación al diámetro real, estableciendo umbrales entre los distintos tipos.



En cuanto a los puntos de los dados, utilizamos:

- **Umbralización Otsu inversa** para separar los puntos oscuros.
- **Apertura morfológica** para eliminar manchas pequeñas que generaban ruido.
- **Clausura** para cerrar los círculos de los puntos y obtener contornos más definidos.
- **Filtrado por área y circularidad**, asegurando que solo se detecten puntos reales y no bordes/puntos laterales o manchas internas.

Estas técnicas permitieron lograr una detección consistente tanto de monedas como de puntos en los dados, incluso con diferencias de iluminación y tamaños.

Segmentación de Patentes y Caracteres

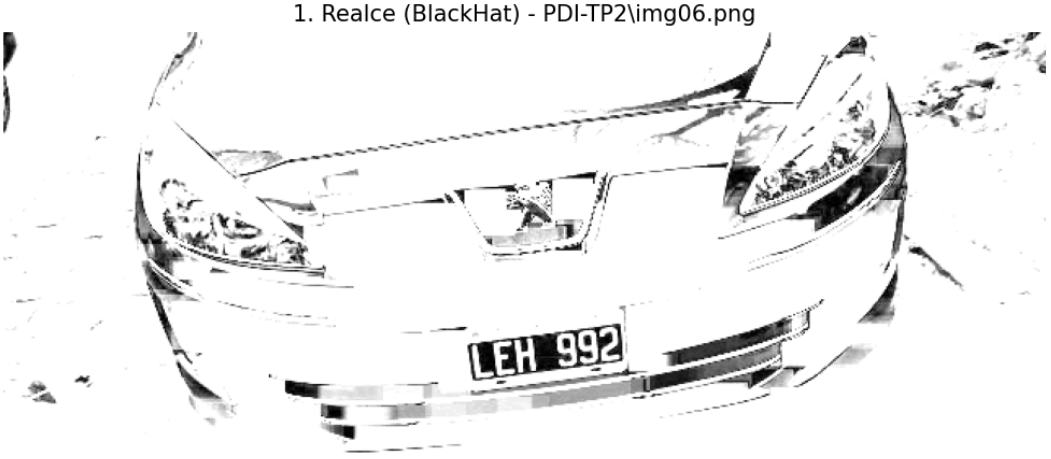
Breve introducción: A diferencia del código de monedas y dados, en este ejercicio tuvimos varios inconvenientes por la gran variedad de patentes, iluminaciones, tamaños y calidades de imagen.
Intentamos ser lo más abarcativos posible, a pesar de que entendemos que las primeras patentes no llegan a segmentarse de forma correcta, luego vemos que las últimas patentes tienen una gran mejoría pudiendo tomar los caracteres de buena forma.

Qué hicimos

El objetivo del código fue detectar automáticamente la patente en cada imagen, localizarla, segmentar los caracteres y cortarlos de forma ordenada.
Para esto, armamos una función procesar_imagen que se aplica en forma automática a todas las imágenes img*.png de la carpeta, usando glob.

Carga y escala de grises
Se lee cada imagen y se la pasa a escala de grises para simplificar los cálculos.

Realce de la placa con BlackHat
Aplicamos un filtro morfológico BlackHat con un kernel rectangular grande, y luego un ajuste fuerte de contraste (convertScaleAbs con alpha y beta) para resaltar las zonas donde la placa se diferencia del fondo. La idea es que la zona de la patente quede casi “dibujada” en la imagen resultante.



Umbralización y bordes
Sobre esa imagen realzada usamos Otsu (con inversión) para binarizar, y después Canny para extraer los bordes principales.

Dilatación y cierre para unificar la placa
Con un kernel rectangular se dilatan los bordes y luego se aplica un closing para unir fragmentos y rellenar cortes. Esto genera contornos más completos de posibles patentes.

Selección de la placa
Se buscan contornos externos y, para cada uno, se calculan relación de aspecto, área y densidad de píxeles blancos dentro del bounding box.
Solo se aceptan como candidatos los contornos con forma y tamaño similares a una placa.
De esos candidatos, se elige el de mayor área como la placa final y se recorta esa ROI.

Una vez tenemos la placa candidata, la mostramos con su bounding-box en la imagen a color.

Preprocesamiento de caracteres dentro de la placa
Sobre la ROI en gris se aplica un suavizado Gaussiano y una umbralización Otsu NORMAL (caracteres blancos, fondo negro).
Luego usamos operaciones morfológicas (closing vertical y un thinning suave mediante erosión y resta) para limpiar la placa y resaltar el “esqueleto” de los caracteres.

Detección, filtrado y orden de caracteres
Se extraen los contornos en la imagen de caracteres y se los filtra con criterios relativos al tamaño de la ROI:

altura mínima y máxima,

relación de aspecto,

área mínima y máxima.

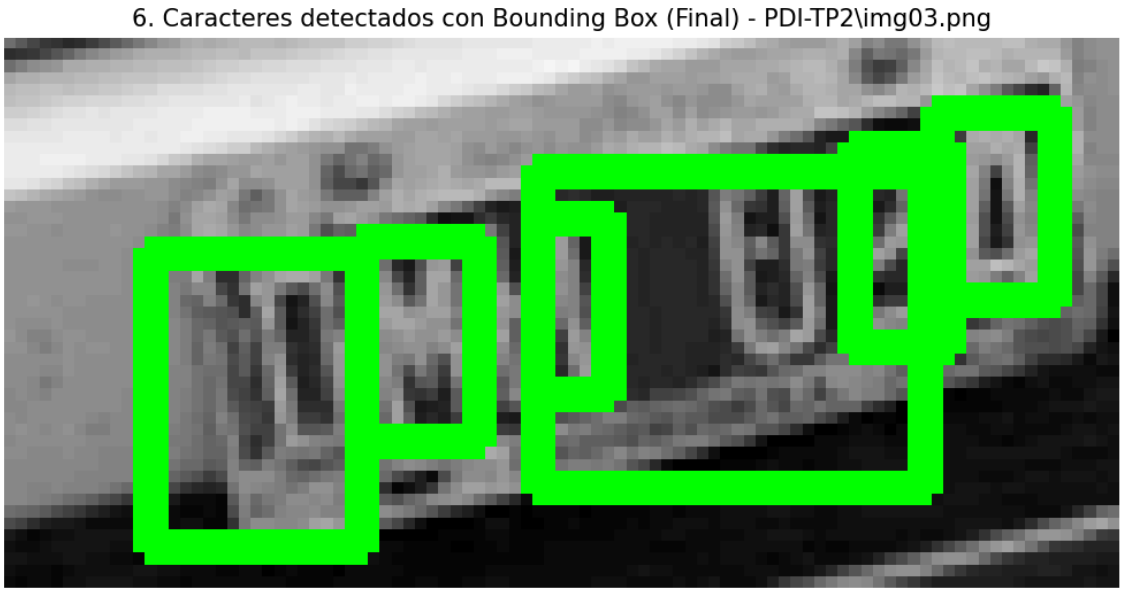
Los candidatos que cumplen todo se ordenan de izquierda a derecha y se dibujan sobre la ROI. Finalmente se recorta cada carácter individual y se muestra en subplots.

Qué problemas tuvimos

Gran variabilidad entre imágenes: algunas patentes están muy contrastadas y otras muy lavadas; algunas están de frente y otras algo inclinadas o desplazadas; incluso cambia el formato (patentes viejas/nuevas).

Sensibilidad a la iluminación: en varias imágenes el fondo tiene reflejos o sombras que generaban bordes “falsos” similares a una patente, haciendo que el codigo eligiera primero contornos incorrectos.

Tamaño y calidad de los caracteres: en ciertas imágenes las letras/números salen muy finos o parcialmente pegados, lo que hacía que el contorno del carácter quedara roto o se uniera con otro.



Parámetros frágiles: los umbrales de área, relación de aspecto y densidad de bordes no funcionaban igual para todas las imágenes. Ajustar algo para una patente rompía otra, por lo que hubo que iterar bastante hasta encontrar valores razonablemente generales.

(Respecto a esto, creo que sobre el final encontramos cual era una buena práctica para ajustar estos parámetros con menor cantidad de iteraciones)

Qué técnicas aplicamos para resolverlo

Para ir atacando estos problemas fuimos incorporando y ajustando varias técnicas:

Realce específico de la placa con Blackhat + contraste
Esto ayuda a resaltar la patente aun con fondos complejos o iluminación desigual, porque realza variaciones de brillo en estructuras rectangulares.

Umbralización Otsu (invertida y normal)
Usamos Otsu invertido para resaltar la placa en la etapa de detección global, y Otsu normal dentro de la ROI para obtener caracteres blancos bien definidos sobre fondo negro.

- Operaciones morfológicas (dilatación, cierre, thinning)
- Dilatación + cierre sobre bordes para unificar el contorno de la placa.
- Closing vertical y erosión con resta para afinar caracteres y separar mejor unas letras de otras.
- Filtros geométricos y de densidad para la placa

En la etapa de detección se filtran contornos por:

relación de aspecto (placa más larga que alta), revisando las medidas legales para estas (aunque mucho no sirvió, dado que tuvimos que ampliar esos rangos)

Área mínima y máxima,

densidad de píxeles blancos (descartando regiones con demasiados bordes dispersos).

Criterios relativos al tamaño de la ROI para los caracteres
Los umbrales de altura, área y relación de aspecto de los caracteres se definen en función de la altura y el área de la placa, y no como valores fijos. Eso hace que el filtro se adapte mejor a patentes más grandes o más chicas.

En resumen, aunque el problema resultó bastante más complejo de lo esperado por la diversidad de imágenes, el código terminó combinando varias técnicas de procesamiento morfológico, análisis de contornos y filtrado geométrico para lograr una detección y segmentación de caracteres razonablemente robusta en un conjunto variado de patentes.

Agregado Extra (Rescates):

Como en varias imágenes no estábamos llegando a detectar la cantidad de patentes necesarias (muchas veces candidatos_placa quedaba vacío) y, cuando detectábamos algo, el análisis resultaba poco confiable (por umbralado/contraste variable, ruido y bordes incompletos), decidimos agregar dos capas de “rescate”. La idea fue mantener el flujo principal (Otsu + contornos) como “camino ideal”, pero si ese camino falla, habilitar alternativas más tolerantes para no perder placas.

El Rescate 1 se activa cuando Otsu no detecta nada. En vez de abandonar, probamos una iteración de umbralados manuales (t_val entre 50 y 100) sobre scale_abs. Para cada umbral, binarizamos, invertimos, sacamos bordes con Canny, reforzamos con dilatación, y buscamos contornos candidatos. Luego aplicamos los mismos filtros de forma: relación de aspecto de placa y rango de área. Además, agregamos una validación extra de “calidad” midiendo densidad de blancos en la ROI, para evitar falsos positivos. Si en algún umbral aparecen candidatos, se guarda ese resultado y se dibuja en rojo como (“RESCATE”).

El Rescate 2 es más “agresivo” y cambia el enfoque: si no logramos placa por contorno, intentamos detectarla indirectamente por sus caracteres. Acá aplicamos un adaptiveThreshold sobre la imagen en gris, detectamos contornos que puedan ser caracteres individuales usando filtros de proporción y altura relativa (porcentaje del alto de la imagen), y después hacemos un agrupamiento por alineación: que estén alineados verticalmente, con alturas similares y distancias horizontales coherentes (como una secuencia de letras/números). Si encontramos un grupo suficientemente consistente (idealmente 6 o más), reconstruimos la ROI de la patente tomando el bounding que engloba esos caracteres, le agregamos margen (padding) y lo usamos como candidatos_placa. Extra: en este rescate también dejamos armados candidatos_caracter relativos a la ROI, para reutilizarlos después sin recalcular.

En resumen, el Rescate 1 intenta “salvar” la detección de placa ajustando el umbral cuando Otsu falla por condiciones de iluminación/contraste. Y el Rescate 2 directamente asume que la placa puede no aparecer como rectángulo claro, pero sus caracteres sí dejan evidencia, entonces reconstruimos la placa desde ese patrón. Con estas dos capas, logramos mejorar la tasa de detección y evitar que imágenes válidas se descarten.