UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Trabajo Práctico N°2

PROCESAMIENTO DIGITAL DE IMÁGENES I

INFORME

Detección y clasificación de Monedas y Dados
Detección de patentes

Carrera: Tecnicatura Universitaria en Inteligencia Artificial

Alumnos: *Demarré, Lucas*

Donnarumma, Cesar Julian

Menescaldi, Brisa

Mussi, Miguel

Ciclo: *2023*

TABLA DE CONTENIDOS

Parte I: Introducción

Problema 1 – Detección y clasificación de Monedas y Dados

La imagen monedas.jpg, adquirida con un smartphone, consiste de monedas de distinto valor y tamaño, y de dados sobre un fondo de intensidad no uniforme *(ver Figura 1).*

- **a)** Procesar la imagen de manera de segmentar las monedas y los dados de manera automática.
- **b)** Clasificar los distintos tipos de monedas y realizar un conteo, de manera automática.
- **c)** Determinar el número que presenta cada dado mediante procesamiento automático.

Figura 1 – Imagen con monedas y dados.

Problema 2 – Detección de patentes

La carpeta Patentes contiene imágenes de la vista anterior o posterior de diversos vehículos donde se visualizan las correspondientes patentes. En Figura 2 puede verse una de las imágenes.

a) Implementar un algoritmo de procesamiento de las imágenes que detecte automáticamente las patentes y segmente las mismas. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.
b) Implementar un algoritmo de procesamiento que segmente los caracteres de la patente detectada en el punto anterior. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa

Figura 2 – Imagen de la patente.

Problema 1

Pasos a seguir

- Cargamos la imagen.
- La convertimos a escala de grises.
- Aplicamos un filtro pasabajos.
- Segmentamos la imagen con Canny.
- Dilatamos la imagen para engrosar los contornos.
- Rellenamos esos contornos.
- Erosionamos para darle más forma a los círculos.
- Buscamos componentes conectadas.
- Detectamos cuáles son monedas y cuáles no.
- Contamos la cantidad de monedas totales y de cada tipo.
- Detectamos los dados.
- Contamos la cantidad de dados.
- Hacemos Cropping de la imagen de Canny de los Bounding Box de los dados.
- Rellenamos los contornos y dilatamos la imagen.
- Se contó la cantidad de elementos circulares dentro de los Bounding Box de los dados.
- Mostramos sobre la imagen original las monedas y dados junto a las cantidades extraídas de cada uno.

Problemas y resolución

En primer lugar, cargamos la imagen a color para su posterior manipulación.

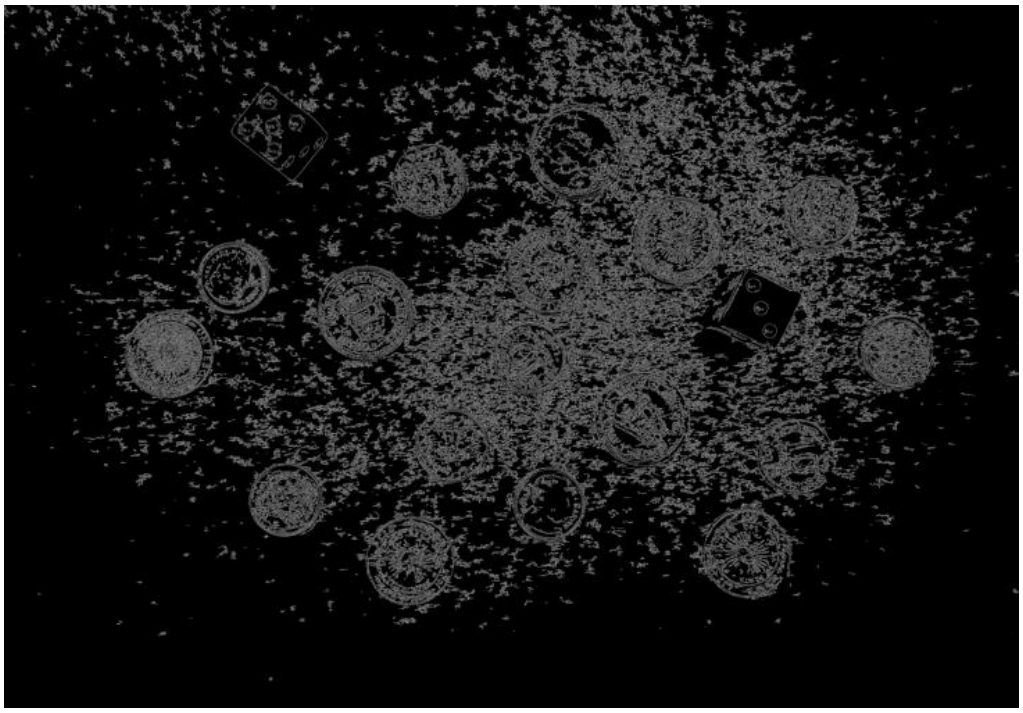
```
# Cargar la imagen a colores
img = cv2.imread('Ejercicio 1/monedas.jpg', cv2.IMREAD_COLOR)
```

Luego, la convertimos a escala de grises.

```
# Convertir imagen a escala de grises
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

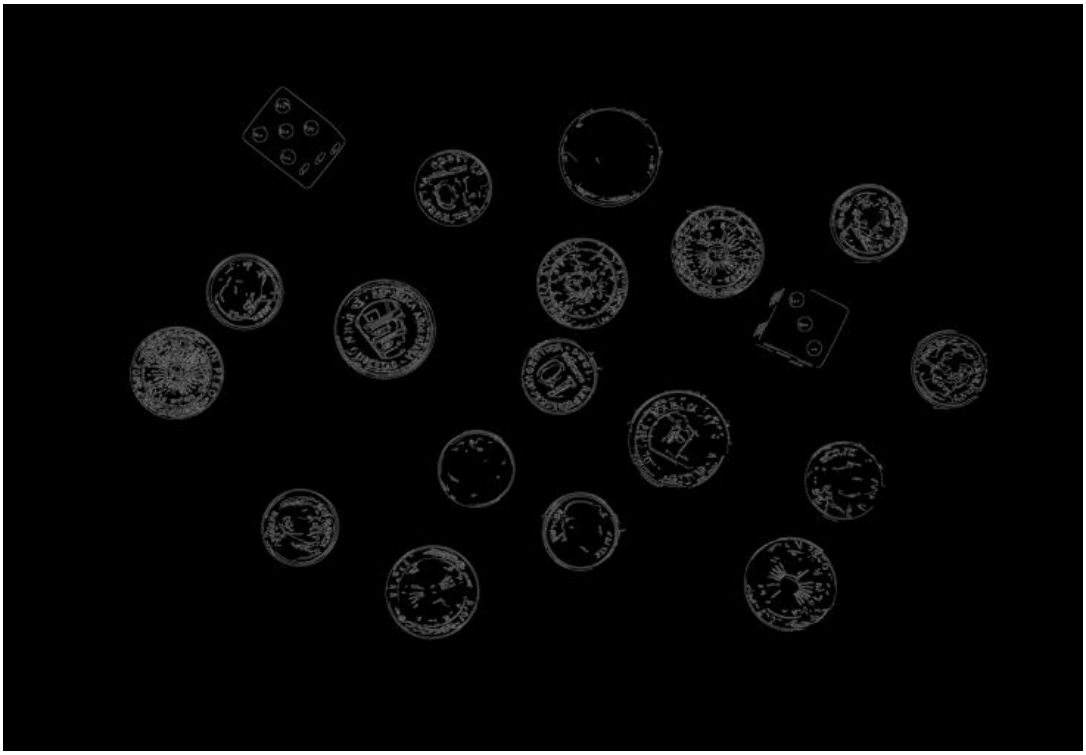
Lo siguiente es aplicar un *filtro pasabajos* previo a segmentar con *Canny* para suavizar la imagen y eliminar ruido. Si no hiciéramos esto, la segmentación nos quedaría con un resultado no favorable, tal que así:

```
# Aplicar Canny para segmentar
img_canny = cv2.Canny(img_gris, threshold1=20, threshold2=150)
```

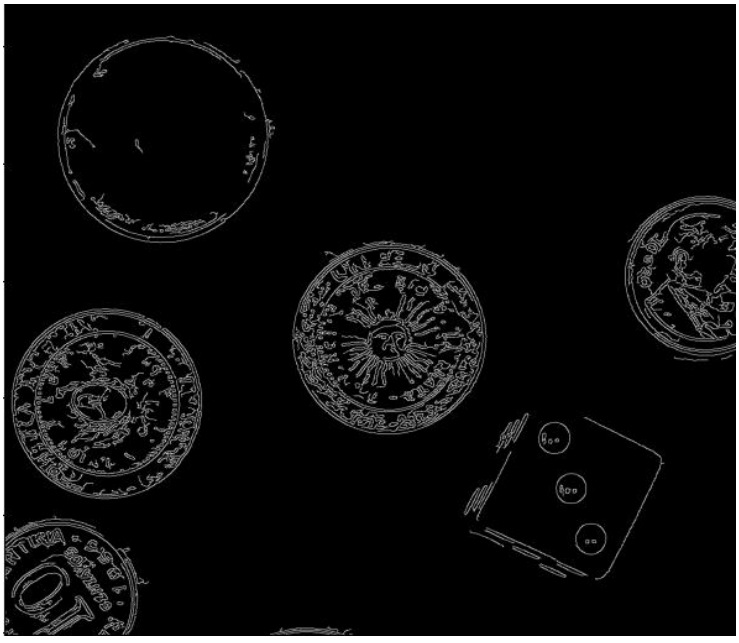


Para evitar, justamente lo que se mostró arriba, es necesario aplicar el filtro, dando así un resultado más correcto tras aplicar *Canny* posterior al *GaussianBlur (filtro pasabajos)*.

```
# Aplicar filtro pasabajos para suavizado:
img_blur = cv2.GaussianBlur(img_gris, ksize=(5, 5), sigmaX=1.5)

# Aplicar Canny para segmentar
img_canny = cv2.Canny(img_blur, threshold1=20, threshold2=150)
```

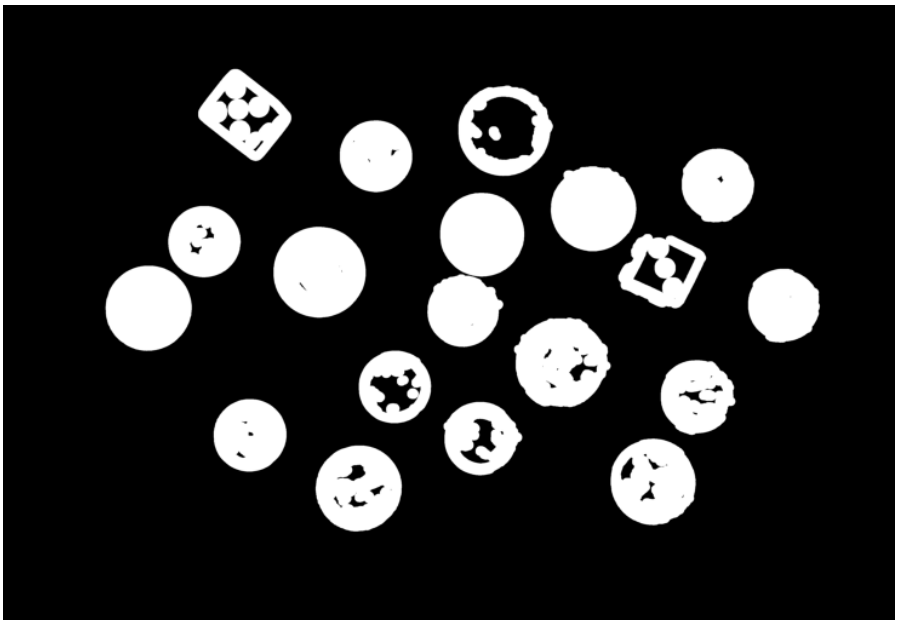


Si ampliamos la imagen, podemos notar que los contornos exteriores de nuestros objetos no están cerrados. Por lo que, para solucionarlo, intentamos engrosar un poco los bordes dilatando la imagen para cerrar los contornos y luego poder rellenarlos



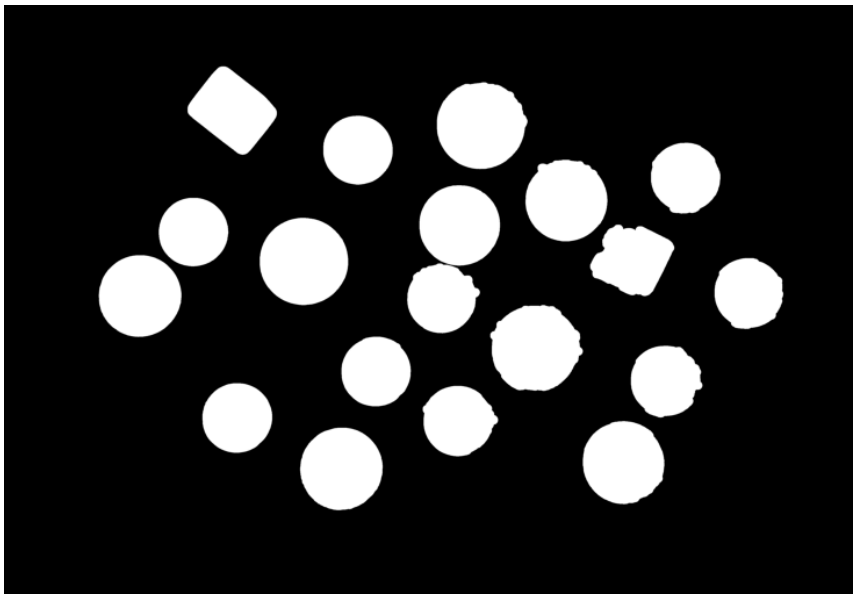
El resultado de hacer esto fue el siguiente.

```
# Dilatar para engrosar un poco los bordes y después poder rellenarlos
elemento_estructural_1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (40, 40))

img_dilatada = cv2.dilate(img_canny, elemento_estructural_1, iterations=1)
```



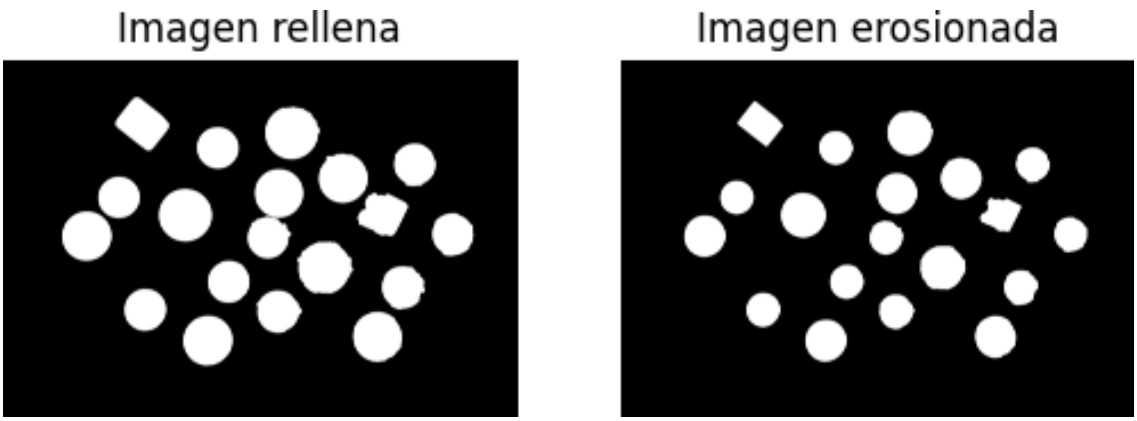
Ahora el siguiente paso es rellenar dichos contornos.

```
# Encuentra los contornos:
contours, _ = cv2.findContours(img_dilatada, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Crea una imagen en blanco para dibujar los contornos y rellenar
imagen_rellena = np.zeros_like(img_dilatada)

# Dibuja y rellena los contornos en la imagen en blanco
imagen_rellena = cv2.drawContours(imagen_rellena, contours, -1, 255,
thickness=cv2.FILLED)
```



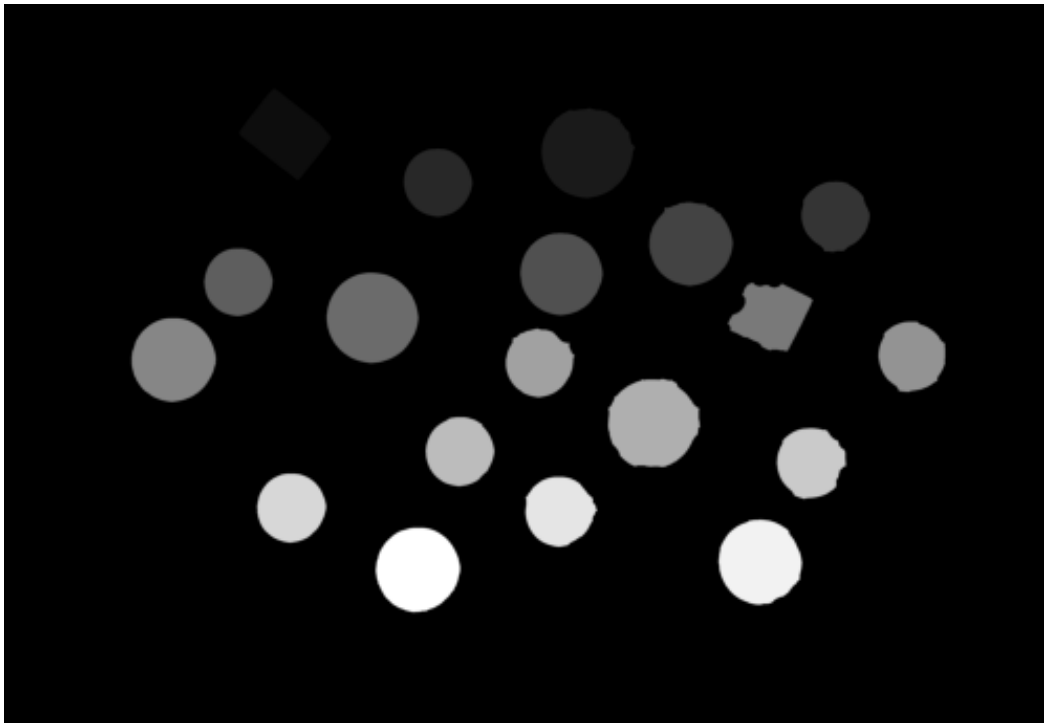
Las formas más o menos están, salvo un dado que está bastante deformado. Lo que ocurrió es que en un primer intento al intentar trabajar con el factor de forma de cada componente conectado encontrado en la imagen ocurría que ese dado era más semejante a un círculo *(según el ff)* que algunos de los círculos propiamente dicho *(que también tienen cierta deformidad)*. Para solucionar esto se decidió erosionar un poco la imagen para afilar un poco las formas de los círculos y que este problema no ocurra.

```
# Erosión para dar más forma a los círculos
elemento_estructural_2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (20, 20))

img_erosionada = cv2.erode(imagen_rellena, elemento_estructural_2, iterations=3)
```



El resultado lejos está de ser perfecto, pero a los fines de poder trabajar con el factor de forma y resolver el problema resultó. Ahora si ya podemos trabajar con los componentes conectados de la imagen.

```
# Buscar componentes conectadas
num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(img_erosionada)
```



Se crea una copia de la imagen convertida a RGB donde se hará la clasificación y se escribirán las estadísticas, adicional a esto, creamos contadores para cada objeto que podemos encontrar en la imagen.

```
# Copia de la imagen sobre la que vamos a dibujar rectángulos
img_clasificada = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Contador de objetos
cantidad_monedas = 0
cantidad_dados = 0
monedas_1= 0
monedas_50= 0
monedas_10= 0
```

La idea es ir analizando componente a componente desde el índice 1 *(es decir sin el fondo)* en adelante, buscando el contorno externo y calculando el factor de forma de cada elemento. En caso de que sea mayor de cierto umbral, determinado de manera experimental observando los distintos factores y ubicando a partir de que numero comienzan a ser círculos, es una moneda.

```
# Vamos analizando elemento a elemento recorriendo desde el primer elemento, que
no es el fondo, en adelante
for i in range(1, num_labels):
    # Generamos una matriz con todos 0 salvo donde está el elemento que se está
analizando
    obj = (labels == i).astype(np.uint8)
    # Obtenemos el contorno del elemento
    contour, _ = cv2.findContours(obj, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Calculamos el factor de forma del elemento
    area = cv2.contourArea(contour[0])
    perimetro = cv2.arcLength(contour[0], True)
    ff = area / perimetro**2

    # Si el factor de forma es mayor que un umbral establecido experimentalmente,
es una moneda
    if ff > 0.065:
        cantidad_monedas+=1
```

Luego, vamos trabajando en cada moneda estableciendo umbrales, pero, a partir de la observación de las áreas para clasificar en monedas de 10cvos, 50cvos o de 1 peso y dibujando un rectángulo de color según corresponda que es el Bounding Box de los componentes conectados de nuestra imagen erosionada.

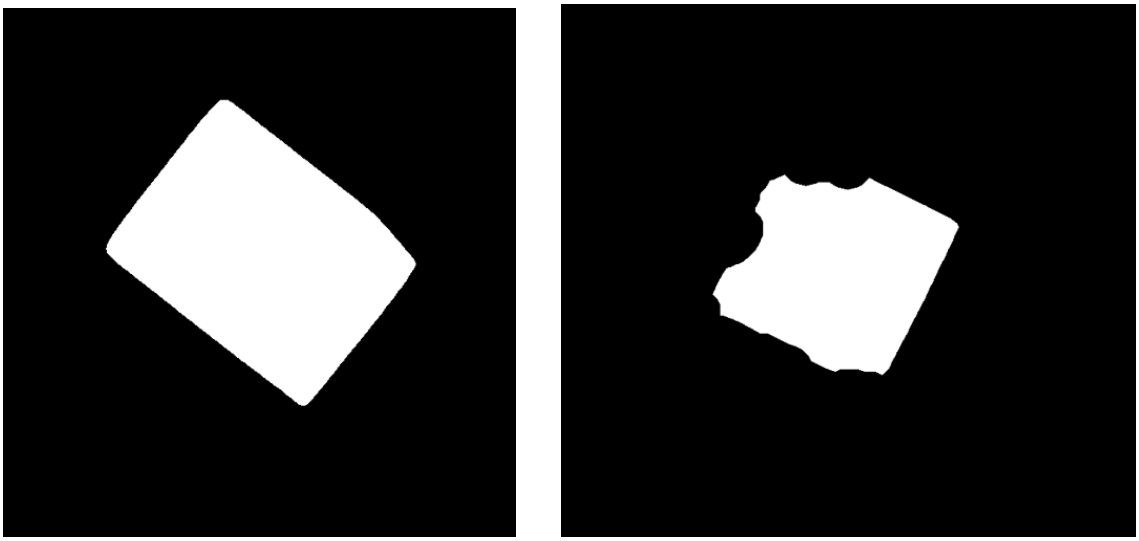
```
        if 53000 > stats[i, cv2.CC_STAT_AREA] > 48000:
            # Dibujamos rectángulo sobre la imagen
            img_clasificada = cv2.rectangle(img_clasificada, (stats[i,
cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP]), (stats[i,
cv2.CC_STAT_LEFT]+stats[i, cv2.CC_STAT_WIDTH] , stats[i, cv2.CC_STAT_TOP]+stats[i,
cv2.CC_STAT_HEIGHT]), (255, 0, 0), 4)
            monedas_10+=1
        elif stats[i, cv2.CC_STAT_AREA] > 72000 and stats[i, cv2.CC_STAT_AREA] <
77000:
            # Dibujamos rectángulo sobre la imagen
            img_clasificada = cv2.rectangle(img_clasificada, (stats[i,
cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP]), (stats[i,
```

```
cv2.CC_STAT_LEFT]+stats[i, cv2.CC_STAT_WIDTH] , stats[i, cv2.CC_STAT_TOP]+stats[i,
cv2.CC_STAT_HEIGHT]), (0, 255, 0), 4)
            monedas_1+=1
        else:
            # Dibujamos rectángulo sobre la imagen
            img_clasificada = cv2.rectangle(img_clasificada, (stats[i,
cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP]), (stats[i,
cv2.CC_STAT_LEFT]+stats[i, cv2.CC_STAT_WIDTH] , stats[i, cv2.CC_STAT_TOP]+stats[i,
cv2.CC_STAT_HEIGHT]), (255, 255, 255), 4)
            monedas_50+=1
```

Si el factor de forma es menor a ese umbral estamos hablando de que no es moneda por lo tanto es dado y también se dibuja el bounding box sobre la imagen a color.

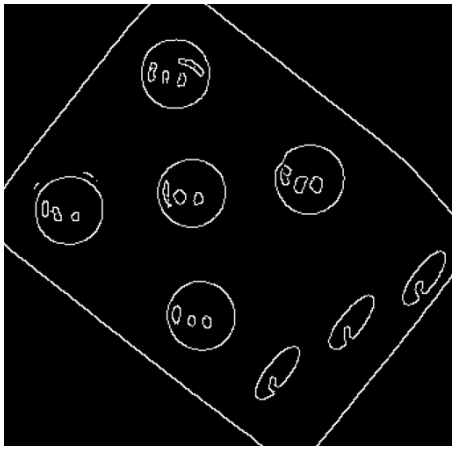
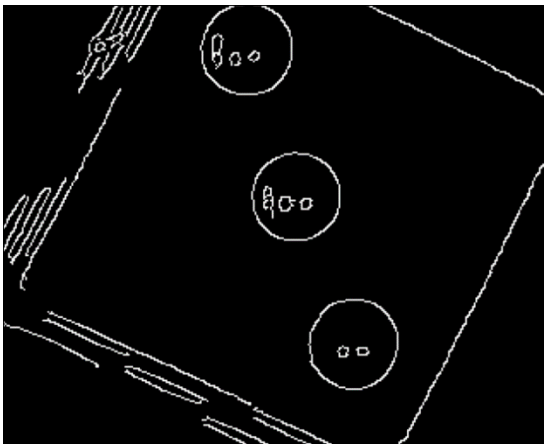
```
# Si no, es un dado
    else:
        cantidad_dados+=1
```

Ahora lo que resta en cada dado es determinar el número de la cara superior. Hay que recordar lo que nosotros teníamos que eran dos contornos rellenos, dado que, la información sobre los puntos que representan los números de la cara superior se había perdido.



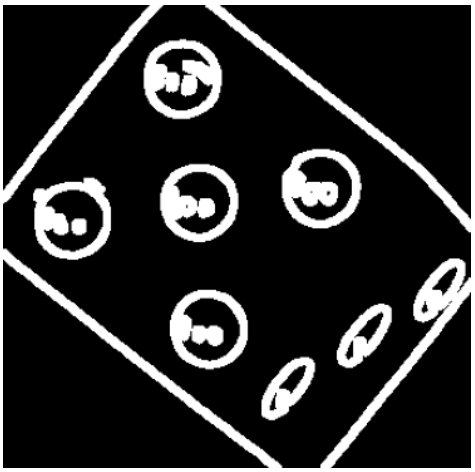
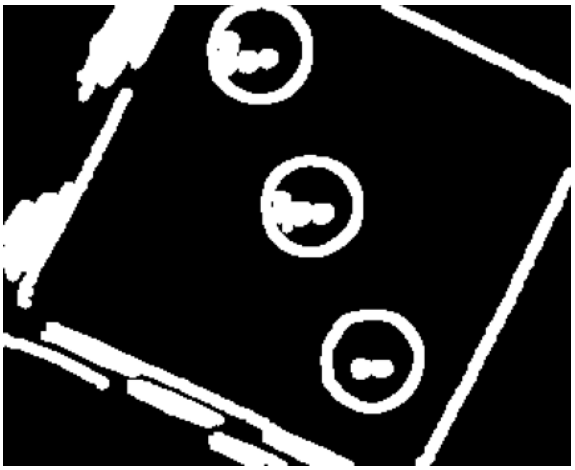
Para solucionar este problema lo que hicimos fue, utilizando la información de los bounding box de los dados, cropping de la imagen resultado de segmentar con *Canny* para obtener los dados.

```
    # Dibujamos un rectángulo sobre la imagen
        img_clasificada = cv2.rectangle(img_clasificada, (stats[i,
cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP]), (stats[i,
cv2.CC_STAT_LEFT]+stats[i, cv2.CC_STAT_WIDTH] , stats[i, cv2.CC_STAT_TOP]+stats[i,
cv2.CC_STAT_HEIGHT]), (0, 0, 255), 4)

        # Cropping de la imagen de canny de los bounding box de los dados.
        dado_canny= img_canny[ stats[i, cv2.CC_STAT_TOP] : stats[i,
cv2.CC_STAT_TOP]+stats[i, cv2.CC_STAT_HEIGHT], stats[i, cv2.CC_STAT_LEFT]:stats[i,
cv2.CC_STAT_LEFT]+stats[i, cv2.CC_STAT_WIDTH ]]
```

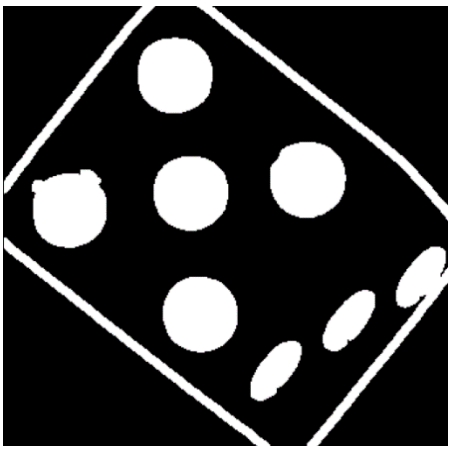
Viendo las imágenes se resolvió nuevamente rellenar los contornos exteriores, buscar los componentes conectados y mirar nuevamente las áreas, con la idea de contar ciertas áreas dentro de determinado rango que corresponderían a los círculos que forman los números.

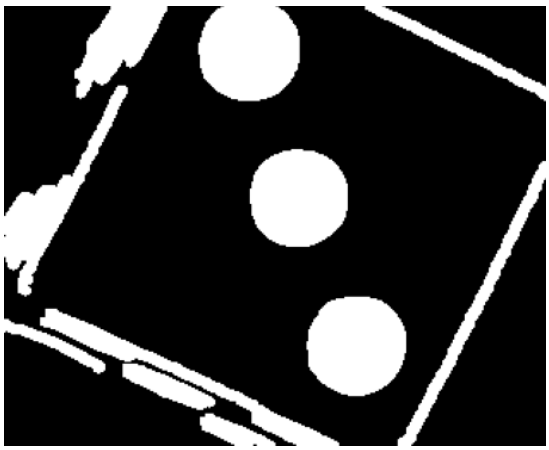
```
        # Dilatamos para ensanchar los bordes un poco para cerrar los contornos de
los círculos
        elemento_estructural_3 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (6,
6))
        dado_dilatado = cv2.dilate(dado_canny, elemento_estructural_3,
iterations=1)
```

Como un paso previo a esto habría que ensanchar un poco los bordes ya que en algún caso los círculos no están cerrados del todo. Se procedió nuevamente a dilatar logrando el objetivo y luego a rellenar los contornos.

```
        # Rellenamos los contornos para rellenar los circulitos y dilatamos la
imagen
        contours, _ = cv2.findContours(dado_dilatado, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        imagen_rellena = np.zeros_like(dado_dilatado)
        dado_dilatado = cv2.drawContours(dado_dilatado, contours, -1, 255,
thickness=cv2.FILLED)
```





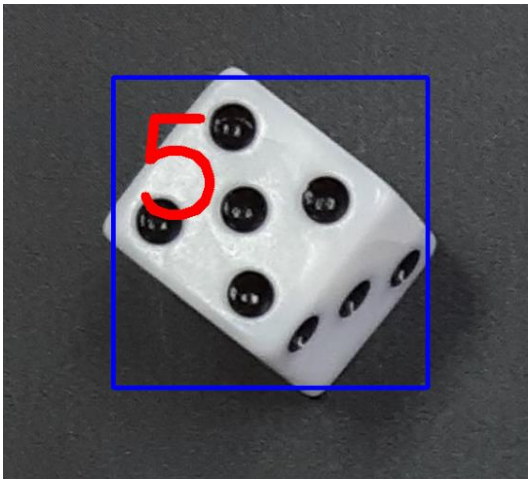
Ahora si ya podemos buscar los componentes conectados. Observando las estadísticas *(áreas)* de cada imagen se pudo ver que es posible establecer determinado rango de valores entre los cuales estarán las áreas de los círculos.

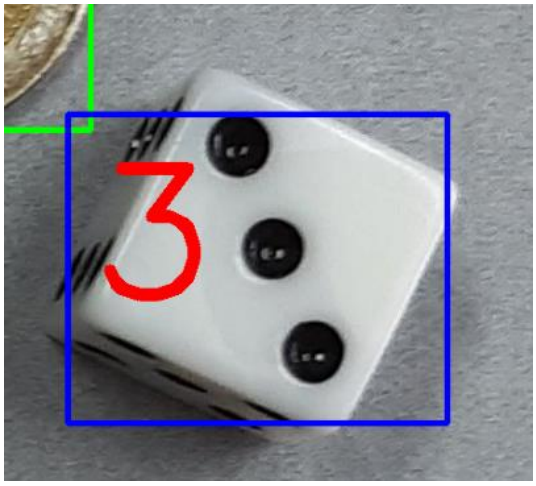
```
[[   0    0  343  339 93597]
 [   0    0  117  146  1304]
 [ 148    0  195  168  1780]
 [ 103   26   57   58  2682]
 [ 204  106   58   58  2688]
 [ 115  117   57   57  2611]
 [  22  127   57   60  2751]
 [   0  179  204  160  1783]
 [ 232  186  111  153  2286]
 [ 122  209   57   58  2642]
 [ 244  219   40   47  1088]
 [ 189  258   39   46  1065]]
```

Entonces simplemente se contó la cantidad de elementos que están en ese rango y se escribió el mismo en la imagen de salida dentro del bounding box del dado.

```
        # Contamos los números de los dados:
        # La idea es que los círculos van a tener áreas similares, establecemos un
umbral de modo experimental
        num_labels_dados, labels_dados, stats_dados, centroids_dados =
cv2.connectedComponentsWithStats(dado_dilatado)
        contador_numeros=0
        for j in range(1, num_labels_dados):
            if stats_dados[j, cv2.CC_STAT_AREA] > 2600 and stats_dados[j,
cv2.CC_STAT_AREA] < 2800:
```

```
                contador_numeros += 1
            # Escribimos el número en la imagen
            img_clasificada = cv2.putText(img_clasificada, str(contador_numeros),
(stats[i, cv2.CC_STAT_LEFT]+20, stats[i, cv2.CC_STAT_TOP]+150),
cv2.FONT_HERSHEY_SIMPLEX, 5, (255, 0, 0), 10)
```

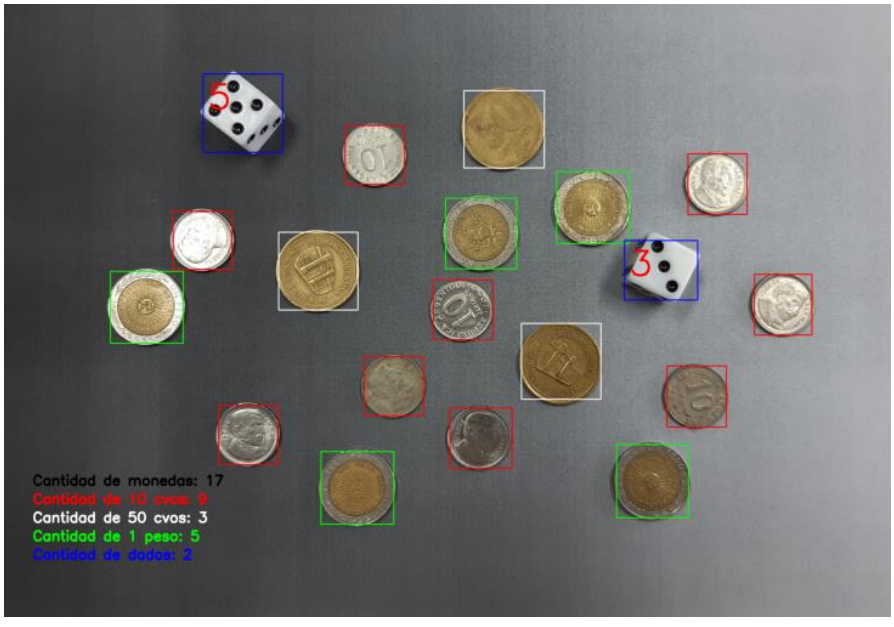




En última instancia se escribió también en la imagen de salida un texto con los resúmenes de cantidad de monedas totales, cantidad de cada tipo de moneda y cantidad dados.

```
img_clasificada = cv2.putText(img_clasificada, f'Cantidad de monedas:
{cantidad_monedas}', (140,2100), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 0), 8)
img_clasificada = cv2.putText(img_clasificada, f'Cantidad de 10 cvos:
{monedas_10}', (140,2180), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 8)
img_clasificada = cv2.putText(img_clasificada, f'Cantidad de 50 cvos:
{monedas_50}', (140,2260), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 8)
img_clasificada = cv2.putText(img_clasificada, f'Cantidad de 1 peso: {monedas_1}',
(140,2340), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 8)
img_clasificada = cv2.putText(img_clasificada, f'Cantidad de dados:
{cantidad_dados}', (140,2420), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 0, 255), 8)
```

El resultado final es el siguiente:



Problema 2

Pasos a seguir

- Cargamos las imágenes.
- Las convertimos a escala de grises.
- Las umbralamos.
- Buscamos las componentes conectadas.
- Filtramos por área esas componentes.
- Filtramos nuevamente, pero esta vez por relación ancho/alto.
- Filtramos por última vez por distancia euclídea.
- Dibujamos las Bounding Box de cada componente.
- Dibujamos un rectángulo alrededor de la patente.

Problemas y resolución

Como primera instancia, cargamos la imagen.

```
# Cargar la imagen a colores
img = cv2.imread(f'Ejercicio2/Patentes/{imagen}', cv2.IMREAD_COLOR)
```

Luego, la convertimos a escala de grises.

```
# Convertir imagen a escala de grises
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Posterior a eso, la umbralamos.

```
# Umbralado
img_umbralada = cv2.threshold(img_gris, 121, 255, cv2.THRESH_BINARY)[1]
```

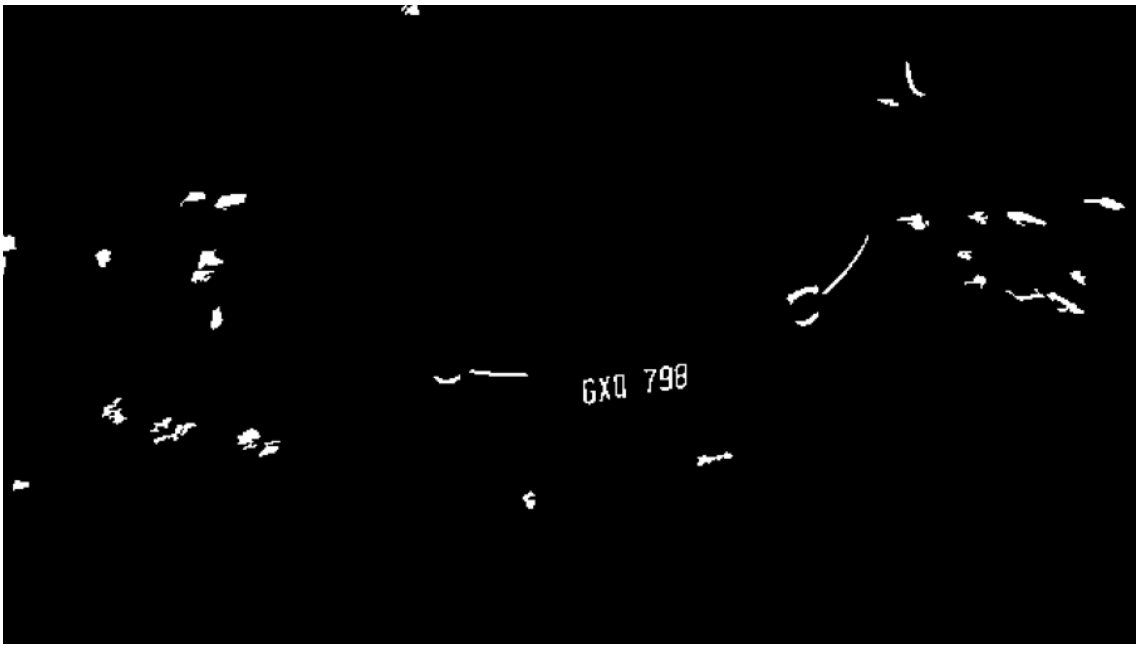
El valor de umbral se eligió experimentalmente, probando valores para los cuales los caracteres de la patente nos queden separados de los bordes y entre sí. También se tuvo que tener en cuenta que no se rompan.

El siguiente paso fue buscar los componentes conectados de la imagen umbralada para hacer un primer filtro de los elementos con áreas muy grandes y también con áreas muy chicas.

```
# Componentes conectadas
num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(img_umbralada)
```

Nuevamente los valores que componen el intervalo de áreas deseables se eligieron experimentalmente.

```
# Filtramos elementos con áreas muy pequeñas y muy chicas
labels_copia = labels.copy()

# Recorrer cada componente conectado sin incluir el fondo
for i in range(num_labels):

# Los que tengan un area superior e inferior a determinados umbrales
    if stats[i, -1] < 26 or stats[i, -1] > 98:

    # Los eliminamos
    labels_copia[labels_copia == i] = 0
```

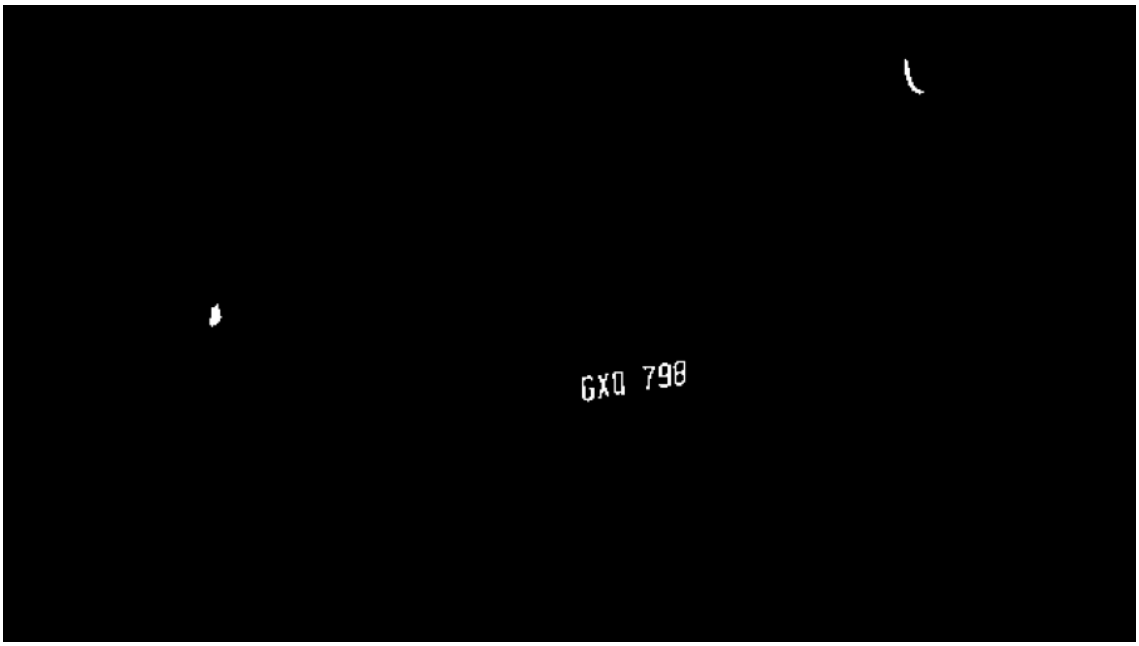


En algunos casos la imagen quedo más limpia en otros menos, pero en general quedaron elementos que no son lo que buscamos.

Sobre este resultado se resolvió hacer un nuevo filtro, pero esta vez por relación entre ancho/alto de los bounding box de los caracteres. Es decir, los elementos cuya relación ancho/alto no estén dentro de determinado intervalo de valores (buscado experimentalmente) se eliminaron.

```
# En las imágenes resultantes además de las letras quedan otros elementos
casi_letras = cv2.threshold(labels_copia.astype(np.uint8), 0, 255,
cv2.THRESH_BINARY)[1]
num_labels_2, labels_2, stats_2, centroids_2 =
cv2.connectedComponentsWithStats(casi_letras)

labels_copia_2 = labels_2.copy()
# Recorremos todo menos el fondo
for i in range(num_labels_2):
    # Eliminamos los elementos cuya relación ancho/alto no esté dentro del
intervalo especificado
    if stats_2[i, 3]/stats_2[i, 2] < 1.5 or stats_2[i, 3]/stats_2[i, 2] > 4:
        labels_copia_2[labels_copia_2 == i] = 0
```

Pasada esta instancia en algunos casos nuevamente quedaron elementos que no son lo que buscamos pero que volvieron a pasar el filtro.



Entonces se necesitó un tercer filtro basado en la distancia euclídea. Cada una de las letras de la patente debería estar cerca de al menos un elemento *(otra letra).* Utilizando el centroide de cada uno de los componentes conectados se calculó la distancia euclídea de cada centroide al resto de los demás, si había al menos un elemento cerca probablemente sea una letra, sino se descarta el objeto.

```
casi_letras_2 = cv2.threshold(labels_copia_2.astype(np.uint8), 0, 255,
cv2.THRESH_BINARY)[1]
num_labels_3, labels_3, stats_3, centroids_3 =
cv2.connectedComponentsWithStats(casi_letras_2)

labels_copia_3 = labels_3.copy()
# Recorremos todos los elementos
for i in range(num_labels_3):
    # Inicializamos un contador
    contador=0
    # Comparamos el elemento i con todo el resto de los elementos
    for j in range(num_labels_3):
        # Calculamos las distancias euclídeas de todos con todos
        distancia_euclidea = np.sqrt(np.sum((centroids_3[i] - centroids_3[j]) **
2))
        # Si la distancia está dentro de determinado umbral
        if distancia_euclidea > 3 and distancia_euclidea < 17:
            # Esos dos elementos están cerca
            contador+=1
    # Si el contador es 0 no hay ningún elemento cerca (no es una letra)
    if contador==0:
        # Lo borramos
        labels_copia_3[labels_copia_3 == i] = 0

# Volvemos a actualizar los componentes conectados que quedaron
casi_letras_3 = cv2.threshold(labels_copia_3.astype(np.uint8), 0, 255,
cv2.THRESH_BINARY)[1]
```

```
num_labels_4, labels_4, stats_4, centroids_4 =
cv2.connectedComponentsWithStats(casi_letras_3)
```

Así se llegó al siguiente resultado.

Ahora que tenemos identificadas las letras sigue dibujar los bounding box de cada uno de los componentes conectados anterior en la imagen original.

```
# Copia de la imagen donde se mostrarán los resultados
resultado_final = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Dibujamos los rectángulos en los caracteres
for i in range(num_labels_4):
    resultado_final = cv2.rectangle(resultado_final, (stats_4[i,
cv2.CC_STAT_LEFT], stats_4[i, cv2.CC_STAT_TOP]),
                  (stats_4[i, cv2.CC_STAT_LEFT]+stats_4[i, cv2.CC_STAT_WIDTH] ,
                   stats_4[i, cv2.CC_STAT_TOP]+stats_4[i, cv2.CC_STAT_HEIGHT]),
(0, 0, 255), 1)
```

Y a partir de las estadísticas de los caracteres buscar: el valor de x más a la izquierda, calcular el valor de x más a la derecha, el de y más arriba y el de y más abajo. Con esto podríamos dibujar un rectángulo que estaría bastante ajustado a lo que son los caracteres, pero la patente en si es más grande que eso, entonces solo resta alejarlos un poco cada uno para su lado y dibujarlo.

```
if len(stats_4) > 1:
    # En x el valor más a la izquierda
    izquierda = stats_4[1:, 0].min()
    # En y el valor más arriba
    arriba = stats_4[1:, 1].min()
    # En x el valor más a la derecha
    derecha = (stats_4[1:, 0] + stats_4[1:, 2]).max()
    # En y el valor más abajo
    abajo = (stats_4[1:, 1] + stats_4[1:, 3]).max()

    # Dibujamos el rectángulo de la patente
    # Los números de arriba se calcularon teniendo en cuenta los caracteres, pero
la patente entera esta
```

```
    # un poco más lejos en cada una de las direcciones por eso los alejamos un
poco.
    resultado_final = cv2.rectangle(resultado_final, (int(izquierda-
izquierda*0.03), int(arriba-arriba*0.03)), (int(derecha+derecha*0.03),
int(abajo+abajo*0.03)), (255, 0, 0), 1)
```

El resultado final es el siguiente:

Repositorio

GitHub

Todos los archivos correspondientes a la resolución de este Trabajo Práctico se encuentran alojados en el siguiente repositorio público de GitHub: Repositorio