Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Detección y Segmentación

TUIA - IA 4.4 Procesamiento de Imágenes

Trabajo Práctico Nro. 3

***Fecha*:** 5/12/2023

Docentes:

- *Gonzalo Sad*
- *Facundo Reyes*
- *Julián Álvarez*

Grupo 9:

- *Revello Simon*
- *Giampaoli Fabio*
- *Ferrucci Constantino*
- *Arevalo Ezequiel*

Presentación

Se presentan dos problemas recurrentes en el área de procesamiento de imágenes que requieren resolución utilizando como fuente el contenido teórico/práctico de la asignatura *Procesamiento de Imágenes Digitales 1*.

En este informe se pretende reflejar el estado inicial de los problemas, el procedimiento conceptual de resolución de cada uno por separado, y su presentación final junto con las conclusiones obtenidas.

Problema 1

Este problema contiene 4 vídeos donde se observa una tirada de 5 dados en cada uno. A continuación se observa un ejemplo de estado final de la tirada de dados de uno de los videos.

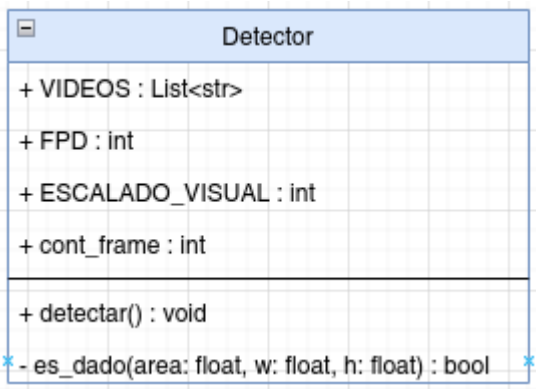
El objetivo es desarrollar un algoritmo para detectar automáticamente cuando se detienen los dados y leer el número obtenido en cada uno. También informar todos los pasos del procesamiento que se realizó para su detección.

Además se pide generar videos (uno para cada archivo) donde los dados, mientras estén en reposo, aparezcan resaltados con un bounding box de color azul y además, agregar sobre los mismos el número reconocido.

Resolución - Problema 1

Para resolver este ejercicio decidimos organizar la solución el dos clases:

- **Detector**



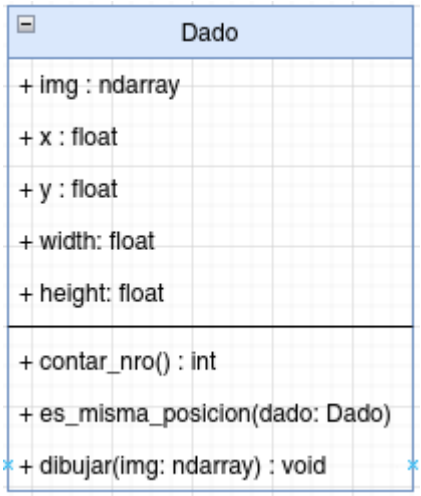
Propiedades:

- **VIDEOS:** Es una lista de strings que contiene la ruta relativa a los videos a analizar.
- **FPD:** Fotogramas por Distancia. Determina cuántos fotogramas deben pasar para evaluar la distancia entre los dados.
- **ESCALADO_VISUAL:** Entero que es utilizado para ajustar el frame una vez que haya sido procesado.
- **cont_frame:** Entero que se utiliza para contar cuántos fotogramas se han analizado. Si es igual a *FPD* se toma la distancia entre los dados y se reinicia a cero.

Métodos:

- **detectar:** Itera sobre los elementos de la constante *VIDEOS* y realiza el análisis sobre cada uno.
- **es_dado:** Dada el área, el ancho y alto de un componente determina si es un dado o no.

- **Dado**



Propiedades:

- **img:** Arreglo numpy de la sección de la imagen original en donde está el dado.
- **x:** Posición en eje x.
- **y:** Posición en eje y.
- **width:** Ancho del dado.
- **height:** Alto del dado.

Métodos:

- **contar_nro**: Cuenta los números de la cara del dado.
- **es_misma_posicion:** Recibe como parámetro otro dado, calcula la distancia euclidea y retorna *True* si la misma es menor a 80.
- **dibujar:** Dibuja un rectángulo y el número que tiene su cara en la imagen que se pasa como parámetro.

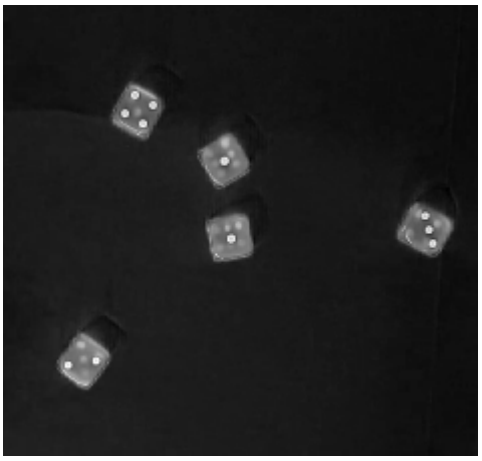
Método “detectar” en profundidad

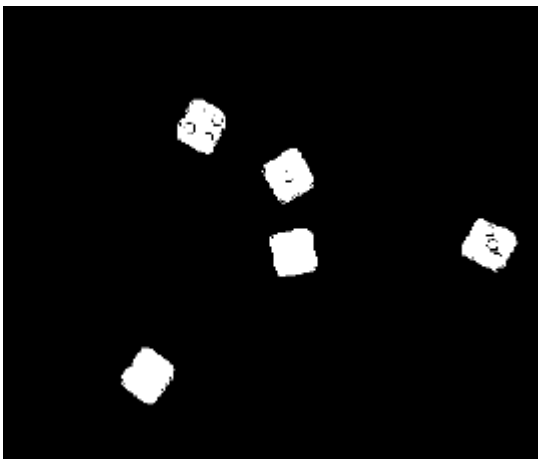
```
    def detectar(self):
        """
        Itera sobre cada imágen y detecta los dados con sus números
        """
        for i, v in enumerate(self.VIDEOS):
            prev_dados : List[Dado] = []
            cap = cv2.VideoCapture(v)
            if not cap.isOpened():
                raise Exception("Error al abrir video")
            fps = int(cap.get(cv2.CAP_PROP_FPS))
            width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
            height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

            if not os.path.exists("out"):
                os.mkdir("out")

            video_out = cv2.VideoWriter(f"out/Video_{i + 1}.mp4", cv2.VideoWriter.fourcc(*"mp4v"), fps, (width //
self.ESCALADO_VISUAL, height // self.ESCALADO_VISUAL))
```

Como primer paso se itera sobre cada elemento de la constante *VIDEOS*. Para cada uno, se instancia una variable *prev_dados* la cual es una lista que contendrá los dados que pertenecen a un fotograma que está a una distancia *FPD* del fotograma actual. Luego, se intenta abrir el video, en caso de no tener éxito se tira una excepción.
Se procede a obtener los fps del video, su ancho y alto. Después, se crea el directorio *out* en caso de que no exista. En esta carpeta se guardarán los videos procesados. Inmediatamente después, se instancia un *VideoWriter* el cual es el objeto que se encargará de generar los videos con los recuadros en los cubos y sus respectivos números.

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    org_frame = frame.copy()

    frame[:,:,0] = 0
    frame[:,:,1] = 0
    t, frame_binario = cv2.threshold(frame[:,:,2], 80, 255,cv2.THRESH_BINARY)
    n_labels, _, stats, _ = cv2.connectedComponentsWithStats(frame_binario, connectivity=8)
```

Mientras el video esté abierto, se obtiene el frame correspondiente. En el caso de haber llegado al final del video, se realiza un break del while loop. Guardamos en la variable *org_frame* una copia del original para que no sufra modificaciones. Luego, a la variable *frame* le tomamos su canal rojo para aplicar el umbral y anulamos los canales verde y azúl. Decidimos hacerlo de esa manera, dado que los dados que se lanzan en el video son de ese color por lo tanto será más sencillo identificarlos.





A la izquierda podemos ver cómo se ve la imagen sólo con su canal rojo y a la derecha vemos cómo se ve luego de aplicar el umbral. Ahora, utilizando la función *connectedComponentsWithStats* será muy fácil identificar los dados.

```
dados: List[Dado] = []
for j in range(1, n_labels):
    x1 = stats[j, cv2.CC_STAT_LEFT]
    y1 = stats[j, cv2.CC_STAT_TOP]
    w = stats[j, cv2.CC_STAT_WIDTH]
    h = stats[j, cv2.CC_STAT_HEIGHT]
    area = stats[j, cv2.CC_STAT_AREA]
    if self.__es_dado(area, w, h):
        dado = Dado(org_frame[y1:y1+h, x1:x1+w], x1, y1, w, h)
        dados.append(dado)
```

Creamos una variable *dados*, la cual es una lista de dados. Luego, iteramos por cada componente encontrada y determinamos si es un dado con la función *es_dado*. Si se pasa la

condición entonces instanciamos la clase *Dado* y la guardamos en la lista declarada anteriormente.

```
def __es_dado(self, area: float, w: float, h: float):
    aspect_ratio = h / w
    return area > 3700 and area < 5500 and aspect_ratio > 0.7 and aspect_ratio < 1.2
```

La relación de aspecto se define como la altura dividido el ancho. Si consideramos que las caras de los cubos son cuadrados, entonces la relación de aspecto debe ser uno ya que en los cuadrados el ancho es igual al alto. Sin embargo, nunca encontraremos un cuadrado perfecto (y por lo tanto la relación de aspecto no será exactamente igual a uno) ya que eso sólo se lograría si los cubos hubieran sido grabados con una cámara exactamente a noventa grados por lo tanto decidimos tomar un intervalo válido cercano a uno: (0.7 ; 1.2). Luego, el área debe estar en el intervalo (3700 ; 5500).

```
for d in dados:
    for pd in prev_dados:
        if d.es_misma_posicion(pd):
            org_frame = d.dibujar(org_frame)
```

Una vez rellenado el arreglo de dados, iteramos sobre el mismo y sobre el arreglo de dados que corresponden a un fotograma anterior. Evaluamos su posición con la función *es_misma_posicion* perteneciente a la clase *Dados*. Si el resultado es *True* entonces dibujamos el recuadro en *org_frame* con la función *dibujar*.

```
def es_misma_posicion(self, dado : Dado):
    """
    Determina si el dado está en la misma posición calculando la distancia euclidea
    """
    distancia = math.sqrt(
        math.pow(self.x - dado.x, 2)
        +
        math.pow(self.y - dado.y, 2)
    )
    return distancia < 80
```

La función *es_misma_posicion* calcula la distancia euclidea entre dos dados y retorna *True* si la distancia es menor a 80. En un primer momento se podría pensar que si un dado está quieto, entonces la distancia previa con la actual debería ser igual a cero. Sin embargo, hay que tener en cuenta que la cámara no está fija sino que está grabada con una mano la cual posee un pulso el cual genera leves movimientos, esto hace que la posición del cuadrado en el plano se vea modificada muy levemente.

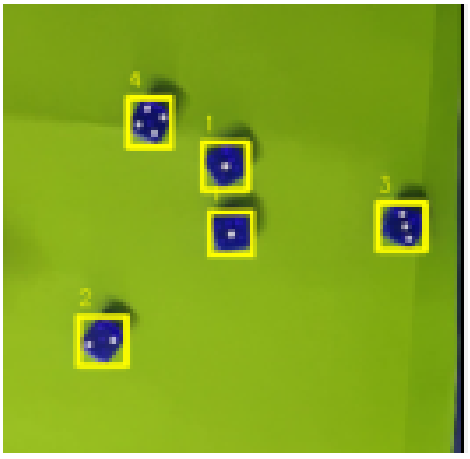
```
def dibujar(self, img: np.ndarray):
    """
    Dibuja un rectángulo con el nro de la cara en la imagen dada
    """
    img2 = cv2.rectangle(img, (self.x, self.y), (self.x + self.width, self.y + self.height), (255, 255, 0), thickness=10)
    return cv2.putText(img2, str(self.contar_nro()), (self.x, self.y - 20), cv2.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 0), 2)
```

La función *dibujar* utiliza el método *rectangle* de OpenCV y el método *putText* para escribir la

cantidad de números que tiene la cara del dado.

```
def contar_nro(self):
    """
    Cuenta los números de la cara de un dado
    """
    cont = 0
    gray = cv2.cvtColor(self.img, cv2.COLOR_RGB2GRAY)
    _, d_bin = cv2.threshold(gray, 120, 255, cv2.THRESH_BINARY)
    d_nro, _, d_stats, _ = cv2.connectedComponentsWithStats(d_bin, connectivity=4)
    for j in range(1, d_nro):
        darea = d_stats[j, cv2.CC_STAT_AREA]
        dwidth = d_stats[j, cv2.CC_STAT_WIDTH]
        dheight = d_stats[j, cv2.CC_STAT_HEIGHT]
        aspect_ratio = dheight / dwidth
        if darea > 60 and darea < 160 and aspect_ratio > 0.6 and aspect_ratio < 1.2:
            cont += 1
    return cont
```

La función *contar_nro* pasa a blanco y negro la imágen del dado, luego aplica un umbral y obtiene los componentes con *connectedComponentsWithStats* y en base al radio y relación de aspecto va aumentando la variable *cont*. Esta es una muestra de cómo se dibujan estos objetos sobre los dados detectados:



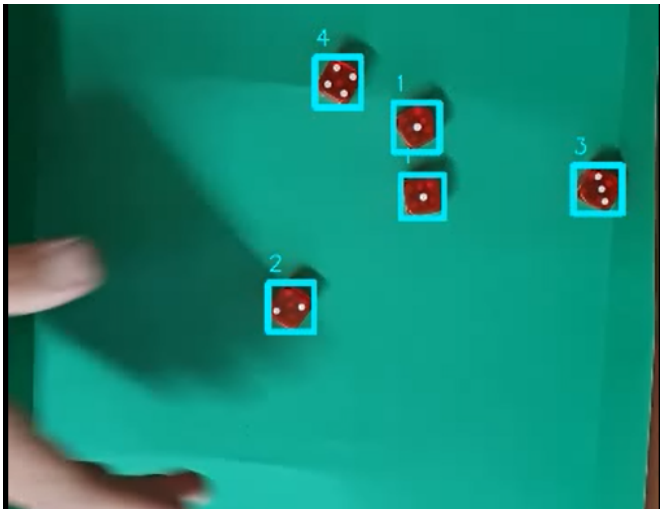

```
        if self.cont_frame == self.FPD:
            prev_dados = dados
            self.cont_frame = 0
        self.cont_frame += 1

        resized = cv2.resize(org_frame, (width // self.ESCALADO_VISUAL, height // self.ESCALADO_VISUAL))
        video_out.write(resized)

        if cv2.waitKey(81) & 0xFF == ord('q'):
            break
video_out.release()
```

Volviendo a la función *detectar*, luego de graficar el dado se verifica si la variable *cont_frame* es igual a *FPD*. Si es así, *prev_datos* será igual a *dados*, el contador será reseteado y volverá a contarse y el ciclo se repite.

Finalmente, se ajusta el tamaño del video y se escribe el frame editado en *video_out*. Una vez finalizado el video se llama al método *release* para generar el archivo de salida.

Los resultados son videos similares a los originales pero con menor resolución, con los agregados además de marcar sobre ciertos fotogramas los dibujos de los rectángulos y números que representan cada dado.

Ejemplo de la mitad del primer video:



Conclusiones Generales

El trabajo práctico fue sumamente útil para adquirir conocimientos sobre herramientas fundamentales como OpenCV, Python como lenguaje de programación y conocimientos teóricos de procesamiento de imágenes para desarrollar un algoritmo efectivo de detección de dados en videos.

La implementación de las clases *Dado* y *Detector* permitió una estructura limpia, organizada y modular para el desarrollo del algoritmo. La clase *Dado* facilitó el manejo de las propiedades y métodos asociados a cada dado detectado, mientras que la clase *Detector* se encargó de iterar sobre los videos, aplicar técnicas de procesamiento de imágenes y generar videos de salida con los resultados.

El uso de técnicas tales como la umbralización, análisis de componentes conectados y la comparación de posiciones entre fotogramas fueron fundamentales para lograr una detección confiable de los dados, incluso considerando pequeños movimientos de la cámara.

Todos los videos han sido procesados exitosamente, pudiendo detectar correctamente los dados cuando dejan de moverse, y contar los números que representan en su cara superior cuando esto ocurre.