Procesamiento de imágenes

Jesus Rodolfo Izurieta Veliz

21 de julio de 2020

1. Introducción

El reconocimiento de objetos es la tarea de encontrar e identificar automáticamente objetos en una imagen. Los humanos detectamos y reconocemos objetos en imágenes con extrema facilidad, inclusive si los objetos sufren variaciones de forma, tamaño, localización, color, textura, brillo o están parcialmente obstruidos.

2. Objetivos

2.1. Objetivos generales

Conteo de monedas en una fotografía (?)

2.2. Objetivos específicos

3. Fundameentos teóricos

Representamos una imagen digital como una secuencia

$$I = (I(x, y), w, h) \quad \forall x \in [0, w], y \in [0, h] \tag{1}$$

Donde $I(x, y)$ es una función de $\mathbb{N}^2$ en el intérvalo $[0, 1] \in \mathbb{R}$.

$$I : \mathbb{N}^2 \to [0, 1]$$

Donde el rango representa el nivel de luminosidad para los pixeles en coordenadas x e y en el caso de imágenes en escala de grises, para imágenes a color, tendremos tres funciones, una para

cada uno de los colores rojo, verde y azul. En el caso de una imagen digital, estos valores varían en el rango $[0, 255] \in \mathbb{N}$.

4. Desarrollo

4.1. Estructura del programa

El proyecto no contará con una interfaz gráfica, sino que constará de un notebook interactivo de jupyter en el que se podrá apreciar tanto el código fuente como los resultados de cada paso del proceso de manipulado de las imágenes. Para esto, se desarrollan las siguientes bibliotecas:

4.1.1. Clase Image `image.py`

La clase imagen es una abstracción básica que nos permitirá acceder fácilmente a una imagen que puede ser cargada desde un archivo, o puede ser creada desde un array y posteriormente mostrada en un notebook de jupyter.

Método	Descripción
`load_file(ruta)`	Carga la imagen que se encuentra en la ruta
`load_array(array)`	Convierte el array en una imagen
`show()`	Retorna un objeto PIL.Image.Image
`I(x, y)`	Retorna los píxeles en la posición x, y
`I_normal(x, y)`	Igual que I() pero en el rango [0, 1]
`I_m(x, y, color)`	Retorna un único pixel
`I_mnormal(x, y, color)`	Igual que I_m() pero en el rango [0, 1]
`iterator()`	Retorna un iterador de tuplas x, y
`map_over(func)`	Permite sobreescribir píxeles mediante una función

La función `map_over(func)` nos permitirá recorrer la totalidad de la imagen aplicando en cada pixel la función `func`, por ejemplo el código `img2.map_over(lambda x, y, z: (x, x, x))` ejecutado sobre una imagen img2, permitirá cambiar el valor de cada canal por su valor rojo, lo que convertirá a la imagen a blanco y negro.

4.1.2. Utilización

Ruta de la imagen como argumento, salida de texto (o imagen modificada).

4.2. Preprocesamiento

4.2.1. Reducción de ruido

4.2.2. Corrección de iluminación

4.3. Segmentación

4.4. Datos de comparación

4.5. Procesamiento

4.5.1. Momentos de Hu

Para obtener un valor propio de la forma geométrica de un objeto en una imagen, usaremos momentos de Hu o momento sinvariantes, un algoritmo que nos permite obtener un conjunto de valores asignados a una matriz según la disposición que esta tenga, sin variar en cuanto a transformaciones como escalado o rotación.

4.5.2. Extracción de rasgos

4.5.3. Comparación cy clasificación

5. Pruebas

6. Conclusiones

7. Referencias

8. Anexos

8.1. Código fuente

8.1.1. Clase Imagen

8.1.2. Filtrado ADF

8.1.3. Filtrado Top-Hat

8.1.4. Filtrado Otsu

8.1.5. Momentos de Hu

Obtención de momentos geométricos de orden $p + q$, $m_{pq} = \sum_x \sum_y x^p y^q I(x, y)$.

```
def m_pq():
```

8.1.6. KNN

8.2. Resultados