Respecto a la captura de imágenes:

* En la segunda toma de imágenes para poder alimentar la base de conocimientos se observó y analizó la recaptura por motivos de distancia, luces y sombras que complicaron el procesamiento de éstas.
* Observación sobre el uso de distintos tipos de filtro (dobel, etc) y fondo para minimizar el ruido en la imagen (se denota el uso de fondo oscuro), así mismo se resalta el uso para obtención de bordes con *Canny* sin transformación a escala de grises.
* Tras el uso de una imagen con fondo oscuro se observan importantes diferencias respecto al resaltado de contornos mostrados en la Figura 1 y Figura 2, ambas con transformación a escala de grises con anterioridad al aplicado de *Canny* (suaviza\_resalta\_contornos.py).

Imagen que contiene Gráfico

Descripción generada automáticamente

Figura 1. Imagen con fondo claro

Imagen que contiene Gráfico

Descripción generada automáticamente

Figura 2. Imagen con fondo oscuro

deteccionbordes.py

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Figura 3. Comparación de imágenes con fondos oscuro y claro

La ventaja observada sobre el fondo oscuro es que evita la necesidad de reducir el ruido que pueden llegar a implicar las sombras en el tratamiento principal de las imágenes.

Esta conclusión podemos denotarla gracias al uso de reduccionruido.py, como se puede observar en las imágenes a continuación.

Gráfico

Descripción generada automáticamente Interfaz de usuario gráfica

Descripción generada automáticamente

Figura 4. Reducción de ruido en imágenes con fondo claro vs fondo oscuro

**COMPARACIÓN**

Reducción\_deteccion

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Deteccionbordes

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Suaviza\_resalta\_contornos (mismo resultado contornos.py)

Imagen que contiene Gráfico

Descripción generada automáticamente

Filtro Sobel | Ecualización

Interfaz de usuario gráfica

Descripción generada automáticamente

Filtro Sobel | Ecualización adaptiva

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Filtro Sobel | Ecualización adptativa gran escala

\*\*La aplicación de los demás filtros es similar

Imagen que contiene Calendario

Descripción generada automáticamente

Filtro Roberts | Ecualización adaptativa

Imagen que contiene Gráfico

Descripción generada automáticamente

Filtro Prewitt | Ecualización adaptativa

Un dibujo de una persona

Descripción generada automáticamente con confianza baja

\*\*Se buscaba el rellenado de la imagen, eliminando espacios blancos, lo cual fue posible en la imagen base (B) a través del uso de la DILATACIÓN.

10 Iteraciones

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

5 iteraciones

Captura de pantalla de un celular

Descripción generada automáticamente

Como es notable, con 5 iteraciones aún hay espacios en negro, por lo que se procede a utilizar las 10 iteraciones.

Sin embargo, en la prueba de la imagen de la seña para A, el número de iteraciones tuvo que subir a 15.

Interfaz de usuario gráfica

Descripción generada automáticamente

Al usar este número de iteraciones con la seña para B, no se denota alguna alteración que refleje algún cambio relevante en el tratamiento de nuestra imagen. Por lo tanto, se considerará este número de iteraciones como base.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Duda:

¿debe verse llena?

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

\*Descriptores convenitentes para cada una de las señas, y valores

\*Excel de valores de descriptores por seña

*# Librerias*

import cv2 as cv

import numpy as np

from skimage import io

from skimage import filters

*# Pre-procesamiento de imagen*

**def** algoritmo\_region\_bordes(*imagen*):

*# Eliminación de ruido - Preprocesamiento (2)*

*im\_ruido* = cv.pyrMeanShiftFiltering(*imagen*,*sp*=30,*sr*=50)

    tamanio\_imagen('Imagen sin ruido')

    cv.imshow('Imagen sin ruido', *im\_ruido*)

*# Escala de grises - Preprocesamiento (3)*

*im\_gris* = cv.cvtColor(*im\_ruido*, cv.*COLOR\_BGR2GRAY*)

    tamanio\_imagen('Imagen a escala de grises')

    cv.imshow('Imagen a escala de grises', *im\_gris*)

*# Binarización | Binarización de Otsu - Segmentación (4)*

*ret*, *im\_binaria* = cv.threshold(*im\_gris*, 0, 255, cv.*THRESH\_BINARY* | cv.*THRESH\_OTSU*)

    tamanio\_imagen('Imagen binarizada')

    cv.imshow('Imagen binarizada', *im\_binaria*)

*# Aplicación de filtros - Segmentación (5)*

*img\_filtro1* = filters.sobel(*im\_binaria*)

    tamanio\_imagen('Imagen con filtro Sobel')

    cv.imshow('Imagen con filtro Sobel', *img\_filtro1*)

*# Operaciones morfologicas - Técnicas y funciones de Python (6)*

*kernel* = np.ones((5,5),np.*uint8*)

*apertura* = cv.morphologyEx(*im\_binaria*, cv.*MORPH\_OPEN*, *kernel*=*kernel*, *iterations*=2)

*dilatacion* = cv.dilate(*apertura*, *kernel*, *iterations*=6)

    tamanio\_imagen('Imagen dilatada')

    cv.imshow("Imagen dilatada", *dilatacion*)

*#Binarización trunca - Segmentación (7)*

*ret*,*bin\_trunca* = cv.threshold(*im\_gris*,127,255,cv.*THRESH\_TRUNC*)

*#Combinación de imágenes - Técnicas y funciones de Python (8)*

*im\_combinada* = *dilatacion* | *bin\_trunca*

    tamanio\_imagen('Imagen combinada rellena')

    cv.imshow("Imagen combinada rellena", *im\_combinada*)

*#Proceso de etiquetado de la imagen - Técnicas y funciones de Python (9)*

*# Binarización de otsu*

*ret*, *im\_binaria2* = cv.threshold(*im\_combinada*, 0, 255, cv.*THRESH\_BINARY* | cv.*THRESH\_OTSU*)

    tamanio\_imagen('Imagen binaria rellena')

    cv.imshow('Imagen binaria rellena', *im\_binaria2*)

*distancia* = cv.distanceTransform(*im\_binaria2*, cv.*DIST\_L2*, 3)

*distancia\_salida* = cv.normalize(*distancia*, 0, 1.0, cv.*NORM\_MINMAX*)

*# cv.imshow('distance-', dist\_out \* 50)--0.6,255*

*ret*, *superficie* = cv.threshold(*distancia\_salida*, *distancia\_salida*.max() \* 0.3, 255, cv.*THRESH\_BINARY*)

*# Marcadores de superficie a 8 bits*

*superficie\_8* = np.*uint8*(*superficie*)

*superficie\_desconocida* = cv.subtract(*dilatacion*, *superficie\_8*)      *#dilatacion*

*# Etiquetado*

*ret*, *etiquetado* = cv.connectedComponents(*superficie\_8*)

    print("Total de regiones:",*ret*)

*# Agregamos 1*

*etiquetado* = *etiquetado* + 1

*# Región desconocida como 0*

*etiquetado*[*superficie\_desconocida* == 255] = 0

*etiquetado* = cv.watershed(*imagen*, *markers*=*etiquetado*)

*imagen*[*etiquetado* == -1] = [0, 0, 255]

    tamanio\_imagen('Imagen etiquetada')

    cv.imshow('Imagen etiquetada', *imagen*)

*#Fin etiquetado ---*

*# Tamaño de imagen*

**def** tamanio\_imagen(*nombre\_imagen*):

    cv.namedWindow(*nombre\_imagen*, cv.*WINDOW\_NORMAL*)

    cv.resizeWindow(*nombre\_imagen*, 500, 550)

*# Adquisición de imagen (1)*

*imagen* = cv.imread('Images/B\_dark.jpg')

tamanio\_imagen('Imagen original')

cv.imshow('Imagen original', *imagen*)

*# Procesamiento de imagen*

algoritmo\_region\_bordes(*imagen*)

cv.waitKey(0)

cv.destroyAllWindows()

|  |  |
| --- | --- |
| **Nombre** | **Descripción** |
| **cv2**.cvtColor(imagen*, código de conversión*) | Transforma la imagen a escala de grises*.*  El parámetro *imagen* hace referencia al nombre de la imagen a procesar.  El código de conversión hace referencia al nuevo plano de colores al que la imagen será convertida, en este caso, a través de la función *COLOR\_BGR2GRAY* |
| **cv2**.threshold(imagen, umbral, máximo valor, técnica) | Convierte la imagen a formato binario, para facilitar la manipulación y tratamiento de ésta. Asignando pixeles en relación con el umbral definido (*treshold*).  El parámetro *imagen* hace referencia al nombre de la imagen a procesar.  El *umbral* se refiere al límite de valores con los que se comparará cada valor de los píxeles determinados.  El valor máximo, define un límite de valores que puede ser asignado a los píxeles determinados.  La técnica se refiere al algoritmo responsable de la binarización, para este caso, a través de la función *THRESH\_BINARY.* |
| **cv2**.dilate(imagen, kernel, iteraciones) | El parámetro *imagen* hace referencia al nombre de la imagen a procesar.  *Kernel* es usualmente un número que define la naturaleza de la operación, refiere a una matriz con la que la imagen se convolucionará.  Las *iteraciones* definen el número de veces que se realizará el procesamiento, siendo un parámetro opcional. |
| **np**.ones(*condición, tipo, orden*) | Regresa un arreglo nuevo del tipo y condición (forma) especificados, donde el valor de los elementos se establece en 1.  La *condición* es un número entero que define el tamaño del arreglo, el tamaño indicado en este parámetro definirá el tamaño de salida.  El *tipo* es un parámetro opcional con un número flotante por default. Específica el tipo de datos a procesar.  El *orden* define la manera en que la información saliente será guardada dentro de un arreglo multidimensional. |
| shape[:n] | Encuentra las dimensiones de un arreglo.  El parámetro *n*, indica el índice que se busca. |
| floodfill() | A partir de un punto determinado, el algoritmo se mueve en todas direcciones marcando un trazado de píxeles que comparten el mismo color con el punto inicial. |
| bitwise\_not() | Permite la manipulación de bits individuales a un nivel máximo de granularidad. |
| distanceTransform(imagen, dst, distancia, mascara) | El nivel de grises se intensifica dependiendo de los puntos dentro del área definida, estos cambian respecto a la distancia respectiva respecto al punto 0 o inicial.  El parámetro *imagen* representa el objeto a ser manipulado.  Para representar la imagen u objeto de salida, se utiliza el parámetro *dst*.  *Distancia* se refiere a un entero que representa el tipo de transformación de distancia que será aplicado.  Mientras que, *máscara* define un entero que representa el tamaño de máscara que se usará. |
| normalize() | Normaliza la información a través de algoritmos entrenados. |
| subtract() | Resta |
| connectedComponents() |  |
| watershed() |  |
| findContours() |  |
| drawContours () |  |
| median() |  |
| percentile() |  |
| .rescale\_intensity() |  |
| equalize\_hist() |  |
| equalize\_adapthist() |  |
| filters.sobel |  |
| filters.roberts |  |
| filters.prewitt |  |