Main flow

start

↓

user input: path

↓

call UDF load_img function with input path

↓

call UDF rgb_to_grayscale with input image

↓

call UDF gaussian_filter with inputs gray_array and hard-setting sigma as 1

↓

call UDF sodef_filter with input blurred_img

↓

show blurred image, apply cmap to image

↓

turn of axes, show final gray blurred image.

↓

end

load_img

load_img(path)
input: path
output: lin_array
purpose: loads user selected image as RGB array

↓

open image selected by user

↓

put image into an array img_array and normalize

↓

call linearize UDF with img_array

↓

is image array 4 dimensions?

— TRUE → convert to three

— FALSE → return img_array

linearize

linearize(img_array)
input: img_array
output: img_array
purpose:uses nested for loop to get row and column indexing

↓

grayscale image?

TRUE / FALSE

TRUE → Loop through each pixel and apply linearization formula

FALSE → Loop through each pixel in R, G, B channels and apply linearization formula

→ return img_array

rgb_to_grayscale

rgb_to_grayscale
input: lin_array
output: gray_array
purpose: conver RGB to grayscale

→ make RGB into grayscale with given constants

↓

return gray_array

gaussian_filter

gaussian_filter
inputs: gray_array, sigma (=1)
output: blurred (8-bit)
purpose: blur grayscale image

↓

ensure sigma > 0

↓

find kernel size using given formula

↓

use numpy linspace from -k to k by kernel_size interval for x and y values

↓

create meshgrid for relative x and y values for each position in kernel

↓

calculate gaussian with given function

↓

normalize kernel

↓

make array same dimensions as gray array but with all values as 0

↓

pad edges of gray_array with black space

→ is x in range gray_array shape [0]

TRUE → is y in range gray_array shape [1]

FALSE ...

is y in range gray_array shape [1]
TRUE → find region around the pixel and blur the value by averaging nearby pixels
REPEAT

FALSE →

return blurred (8-bit)

REPEAT

sobel_filter

sobel_filter
input: image_array
output: data_array
putpose: exxagerates lines/borders in black and white

↓

pad edges with zeros (black) and conver to float

↓

make a copy of image as array

↓

define sobel operator kernels for x and y gradients

↓

is row in range (1, len(data_array)-1)

REPEAT

TRUE → is column in range (1, len(data_array[0])-1)

TRUE → extract 3x3 space around given kernel

↓

calculate the gradient in x and y directions

→ calculate gradient magnitude and clip to be 0-255

→ remove the padding added earlier

↑

is row in range of length of data array
TRUE

FALSE

is column in range of length of data_array[0]
TRUE

FALSE

→ apply threshold to highlight edges

↓

return data_array