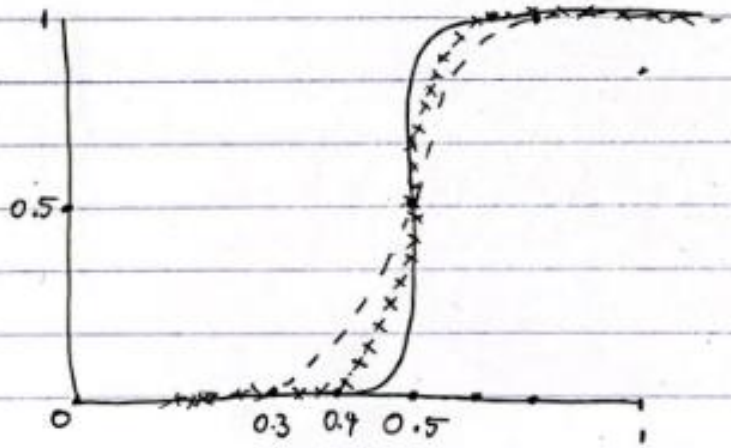
Homework 2

problem 1

a) $r < m$, slope increases
$r > m$, slope decreases
$r = m$, slope infinite norm.s $= \frac{1}{2}$

$s = \frac{1}{1 + \left(\frac{m}{r}\right)^E}$ E-control function slope during transition from High to Low

b)



E=10 ---
E=20 xxx
E=50 —

c) $r < m$, $s = 0$
$r > m$, $s = 255$
$r = m$, $s = 128$

$m = 128$

$\frac{1}{1+\left(\frac{m}{m-1}\right)^E} < \frac{c}{2}$

$1 < \frac{c}{2}\left(1+\left(\frac{m}{m-1}\right)^E\right)$

$\frac{2}{c} < 1 + \left(\frac{m}{m-1}\right)^E$

$\left(\frac{2}{c} - 1\right) < \left(\frac{m}{m-1}\right)^E$

$\log_{\left(\frac{m}{m-1}\right)}\left(\frac{2}{c} - 1\right) < E$

$E > \log_{\frac{128}{127}}\left(\frac{2}{10^{-308}} - 1\right)$

Introduction

Implement a simple Intensity transform function that can perform transformations for gamma, log, and negative intensity functions and is scaled from 0-1 as a 32bit floating point.

Approach

While encourage to use MatLab my implementation was done with python and OpenCV
A package that is useful for image and data processing. I used an Object Oriented approach and created a class for Intensity transformations. In the future, I hope to add more algorithms from homework and projects related to Intensity transformations in the file. I hope this will pay off in the future when it comes to studying for tests. Currently when the class is called the user must input an image, a mode representing the type of intensity transform, and an optional gamma value for the gamma intensity. After the transformed image is created it is stored in the class object and retrieved using an accessor method. The functions can be called directly to perform transforms on the image. The user would need to set the image using the class. They would then call the intensity transform function to traverse the image and perform the selected function. The function is selected by providing an intensity transform function. the user can call the transform functions in the class such as (log_func) and they can also create their own function and pass it through as a parameter.

Automated usage:

```
path = “my/image/path”
img = cv2.imread(path)
mode = “gamma”
ixf = intXform4e(img, mode, gamma_value)
new_img = ixf.get_img()
```

Manuel usage:

```
path = “my/image/path”
img = cv2.imread(path)

ixf = intXform4e(img)
ixf.intensity_transform(ixf.gamma_func, gamma)
new_img = ixf.get_img()

def custom_func(local_pixels):
    mid_x = (1,local_pixel.shape[1]-1)
    mid_y = (1,local_pixel.shape[0]-1)
    return custom_function(local_pixels[mid_y,mid_x])
ixf.intensity_transform(custom_func)
new_img = ixf.get_img()
```

image width - w
image height - h
number of images - i

Time complexity
T(w,h,i) = O(w*h*i)
Space Complexity
S(w,h,i) = O(w*h)

The Transform functions:
negative_pixel_intensity = L - 1 - current_pixel

C = 1 or max_intesity/log(1+local_max_intesity)
log_pixel_intensity = C * log(1+current_pixel)

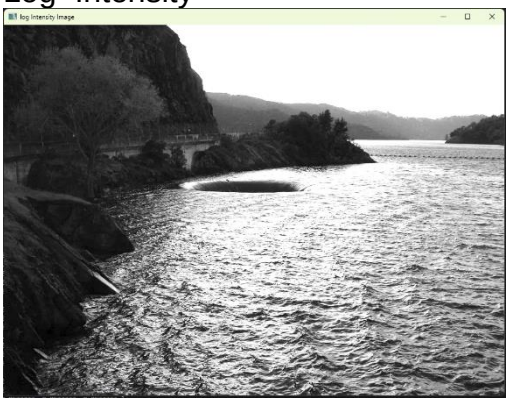
gamma_pixel_intensity = max_intensity(current_pixel/max_intensity)^gamma

Experimental Results

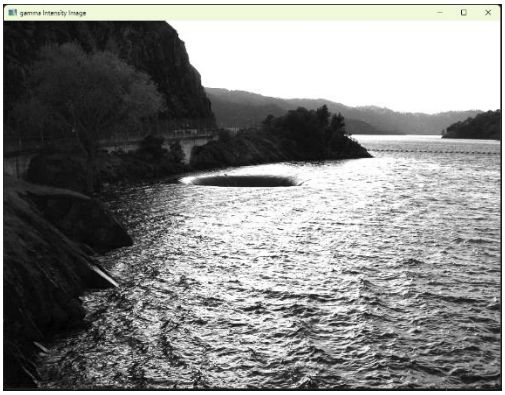
The results of implementing my class were successful but had unexpected consequences.
I applied it using two different methods one was valuing a module in an OpenCV to do the absolute difference calculation. The other approach was doing the calculation using pythons minus operator and absolute function. What surprised me at first is that the implementation by raw The calculation was very noisy with lots of bright pickles where I would have expected. This may be due to noise and light changes in between the images. The OpenCV approach was much cleaner and hinted at extra filtering behind the scenes to reduce noise or better identify the background being subtracted.

Negative Intensity

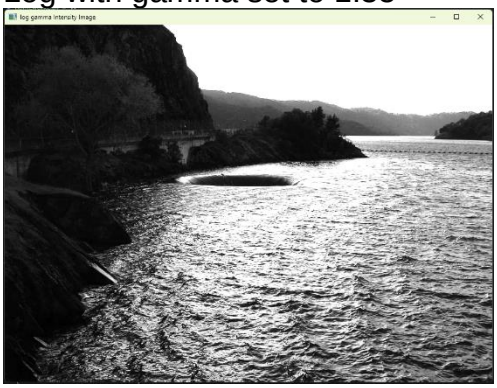
Log Intensity



Gamma = 1



Log with gamma set to 1.35



Conclusion

The algorithm is simple but also very powerful for image intensity transformation. It is similar to the algorithm for affine transformation but at each pixel, we use a function to update its value rather than its location.