Name: Batuhan

Surname: Yalçın

ID: 64274

COMP527/ELEC519 – Programming Assignment 3: Flash/No Flash Photography

All the parts completed. The bonus part exceeds my computer and colabs performance. They are quite well (except bonus 3 part 2 since I was able to run in small iteration) but they can be improved with increasing number of iteration in good computer. For bonus 3 part 1 the image quite well you can say it's okey however, I should say the colors are not correct additional white balance algorithm and gamma correction can lead perfect result.

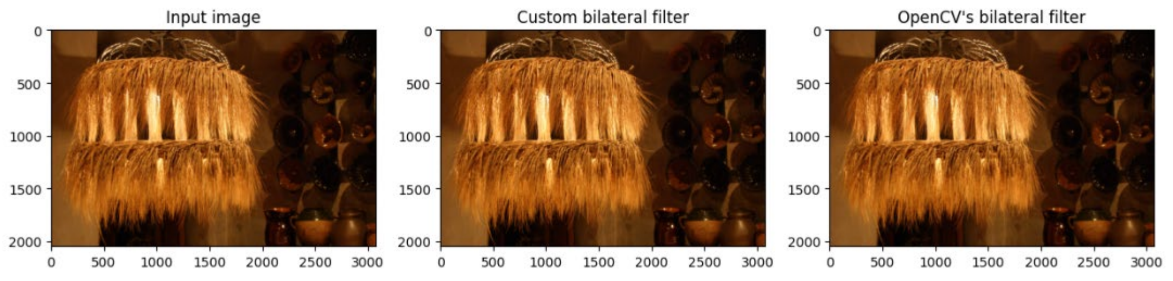
1-) Bilateral filtering:

Implementing bilateral filtering:

To implement Bilateral Filtering, First I calculate number of required intensity segment. To do that I divide the difference between the maximum value and the minimum value in the image to standard deviation. Then I compute the Gaussian filter kernel and apply it to the input image for each intensity segment. Then using numpy's interpolate function I interpolate the filtered values for each color channel separately.

To support my result, I also get mean squared error with cv2 built in bilateral filter and I got less than 1e-10 error so my implementation was correct.

An this was results for kernel_size = 5, sigma_s = 0.05 sigma_r = 40   :

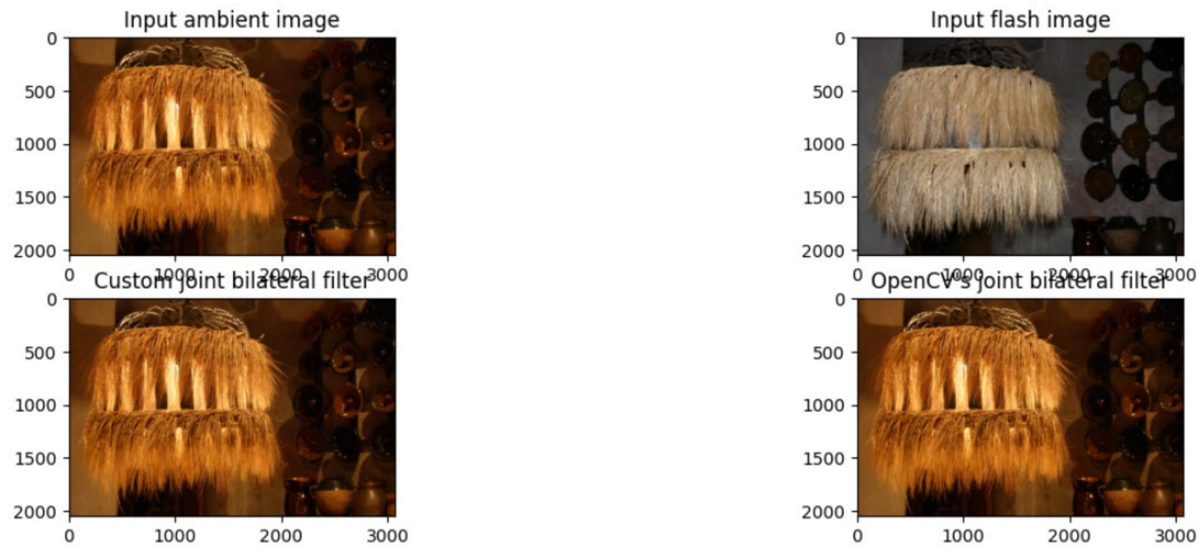


Implementing joint-bilateral filtering:

The algorithm very similar with the bilateral algorithm, this time for intensity segments and instensity weights flash image used, the resultant intensity segments and weights applied to ambient image. Again, it applied all three channels.

To support my result, I also get mean squared error with cv2 built in joint bilateral filter and I got less than 1e-10 error so my implementation was correct.

An this was results for kernel_size = 5, sigma_s = 0.05 sigma_r = 40:


Input ambient image
Input flash image
Custom joint bilateral filter
OpenCV's joint bilateral filter

Implementing detail transfer:

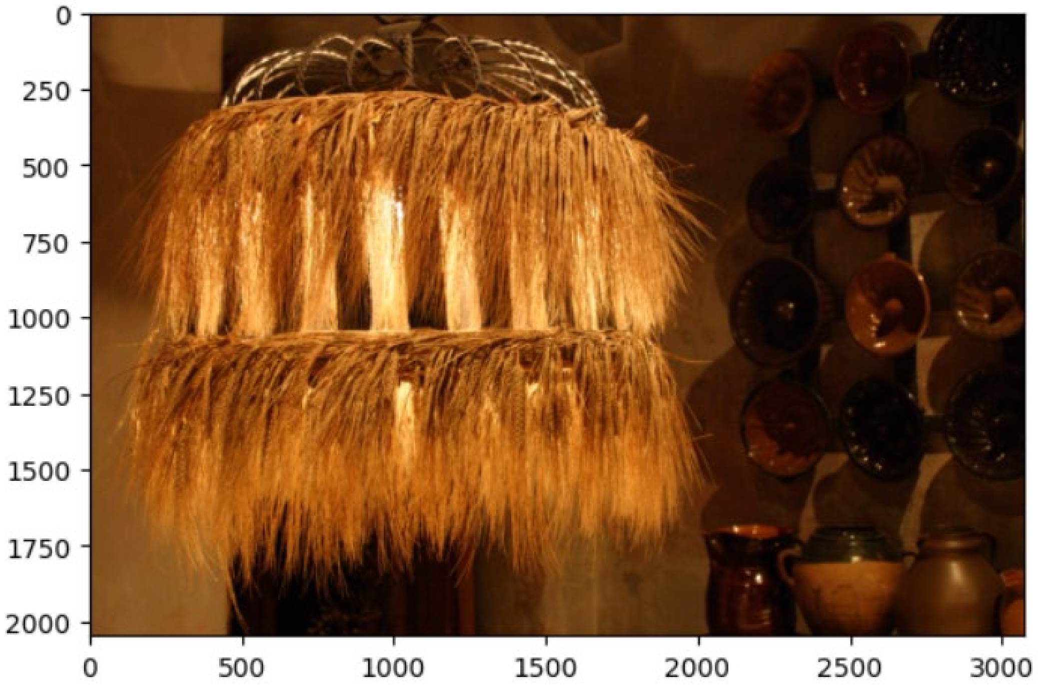
For detail transfer given function used, for fbase as described bilateral filtered filtering applied to the flash image, for Anr joint bilateral function called. For epsilon 0.001 used.

An this was results for kernel_size = 5, sigma_s = 0.05 sigma_r = 40, epsilon 0.001:

Implementing shadow and specularity masking:

To implement Shadow and specularity masking first the tiff image should be linearized for better result for masking 0.0404482 threshold used as described in hints. For masking it was little bit complicated and I get help from ethernet, to apply suggested morphological filtering operations which are (dilation, erosion, opening, and closing). Then using Abase as ambient base, Adetail form detail transfer and masking I got final image.

An this was results for kernel_size = 5, sigma_s = 0.05 sigma_r = 40, epsilon 0.001:



Implementation details:

Selecting Best Parameter for Sigma_s:

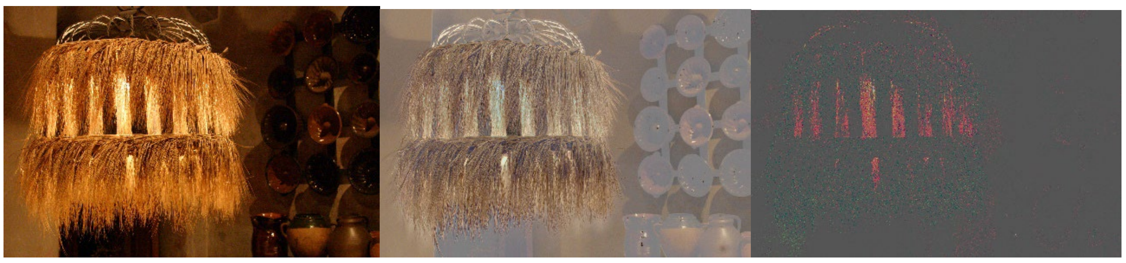
sigma_s is responsible from the standard deviation of the spatial of the Gaussian kernel .

When sigma_s increasing noises are decrasing however big sigma_s leading losing details. I found best values are between 5 and 10 however there are no significant between them so I stop at sigma_s = 5

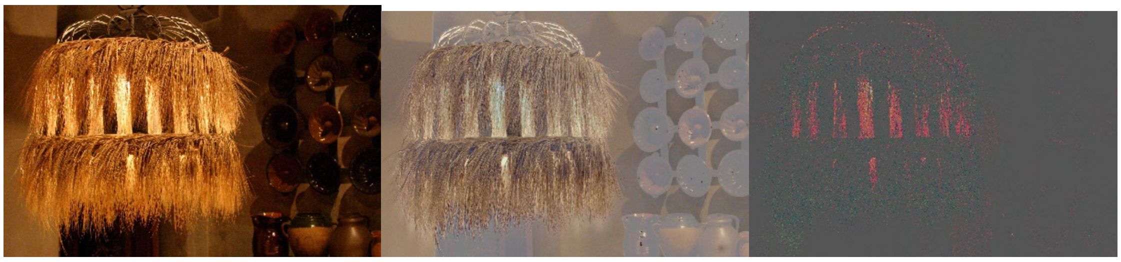
Followings are results:

Pictures are left result middle differences with flash image, right differences with ambient image.

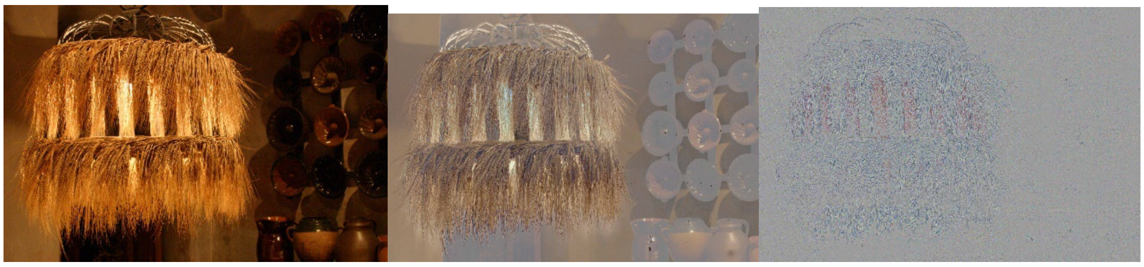
Sigma_s = 0.05:



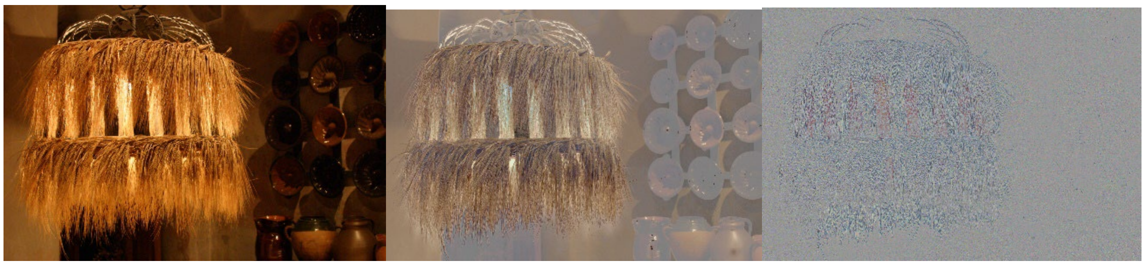
Sigma_s = 0.1:



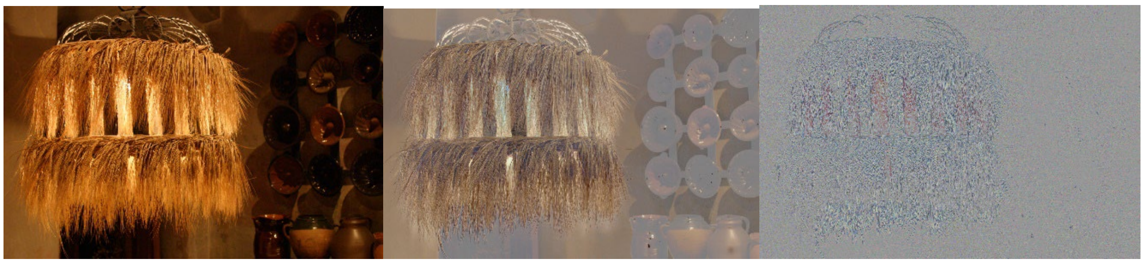
Sigma_s = 1:



Sigma_s = 5:



Sigma_s = 10:



Selecting Best Parameter for Sigma_r:

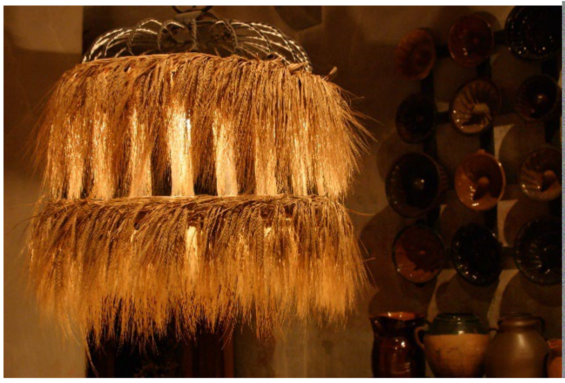
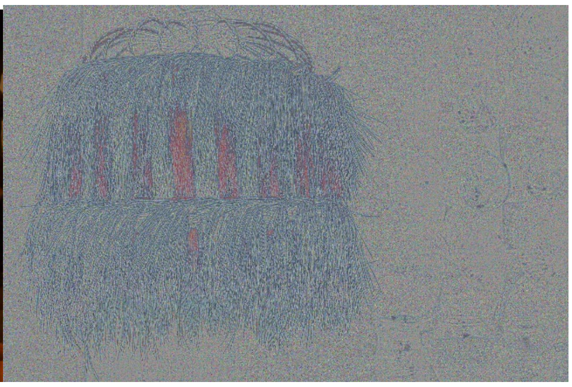
sigma_r is responsible from the standard deviation of the intensity of the Gaussian kernel

When we keep the Sigma_r small the edges get become better since the small nearby pixels gets more than the far pixels and number of intensity segment increases. For the sigma_r 0.05 or 0.08 results are satisfying

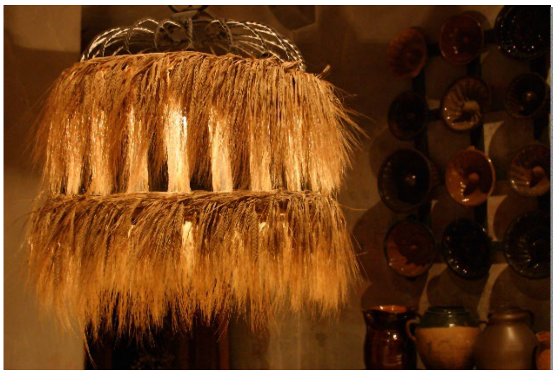
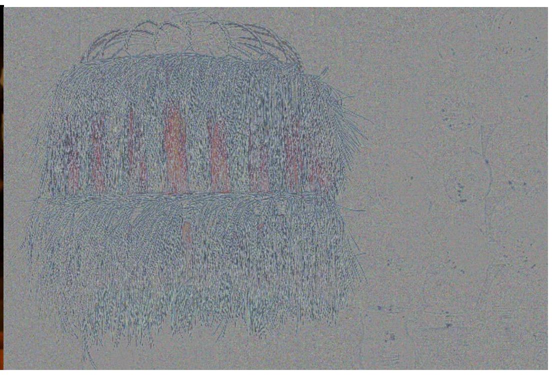
Results are given in below:

Pictures are left result middle differences with flash image, right differences with ambient image.

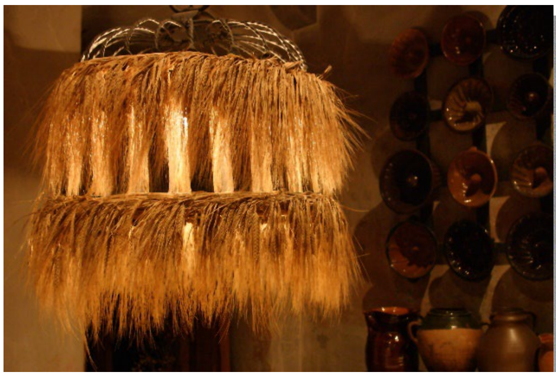
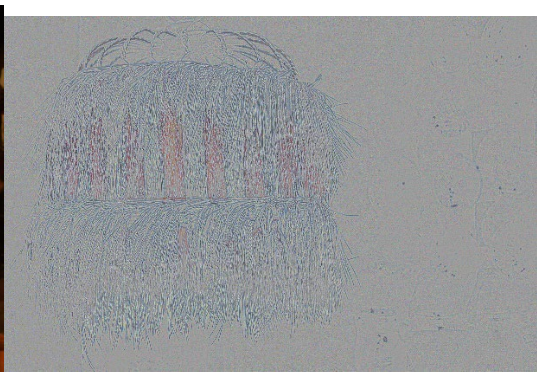
Sigma_r =0.05:

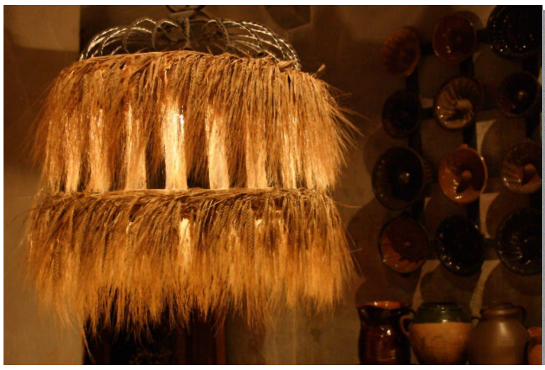
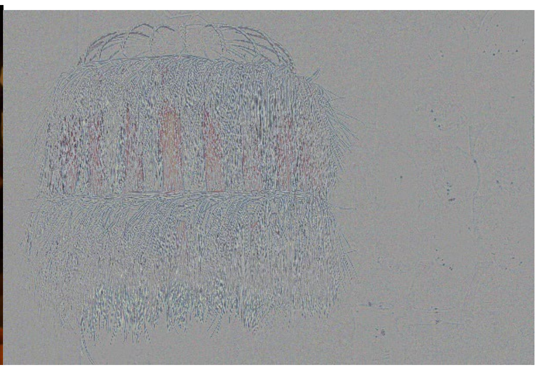
Sigma_r =0.1:

Sigma_r=0.15:

Sigma_r=0.2:

Selecting Best Parameter for kernel_size:

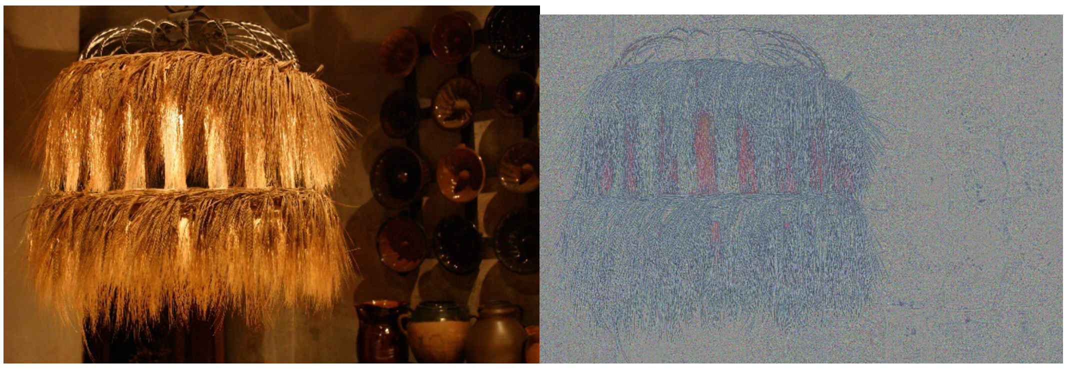
Kernel size responsible from the kernel size of the gaussian filter

When Kernel size increase the image get more smoother however we are losing the details. So optimum Level is I choose 11

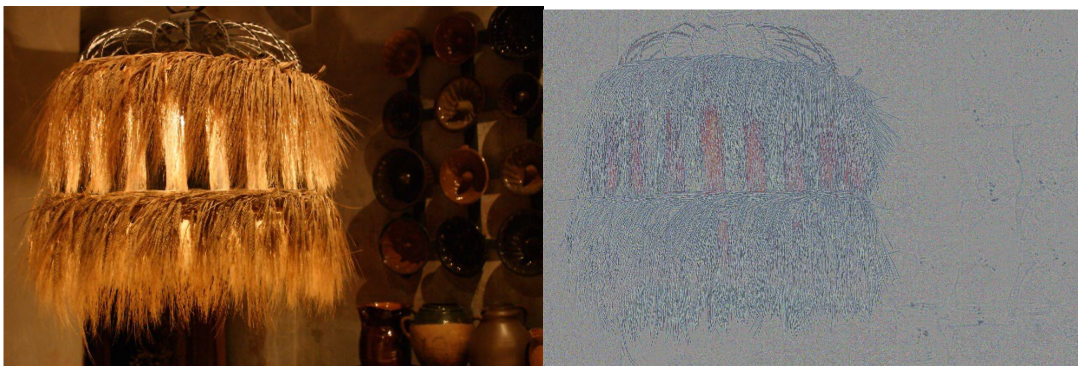
Results are given in below:

Pictures are left result middle differences with original image.

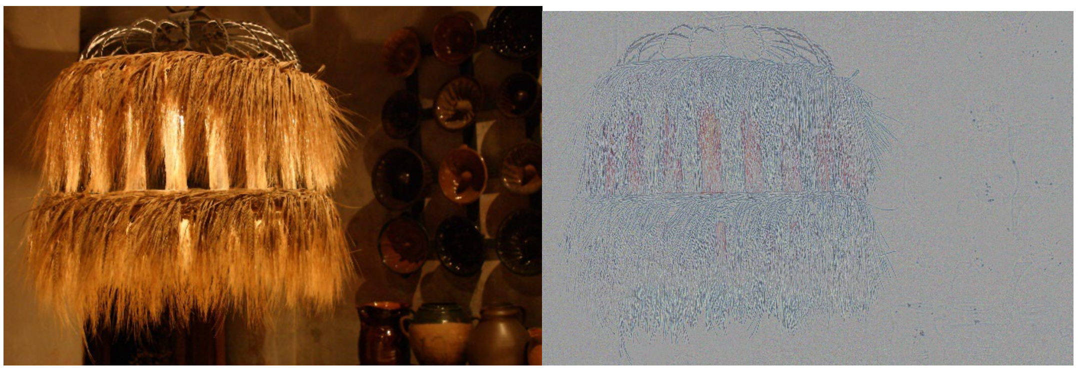
Kernel Size 1:



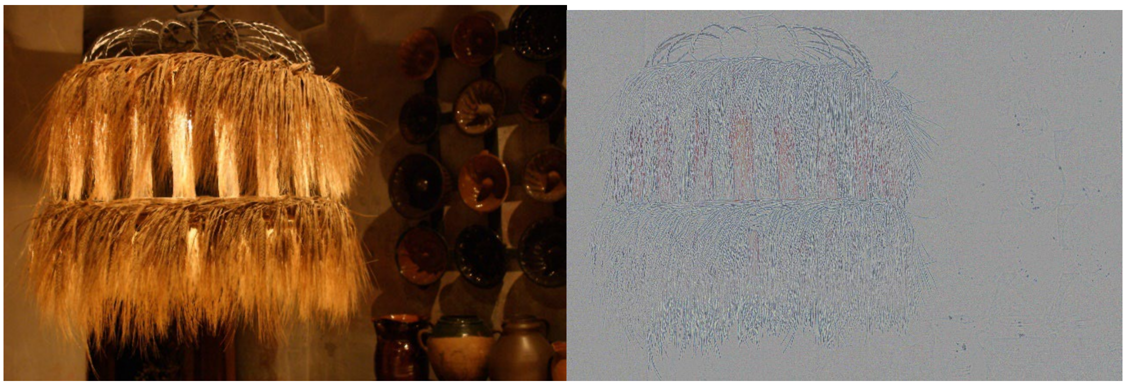
Kernel Size 5:



Kernel Size 11:



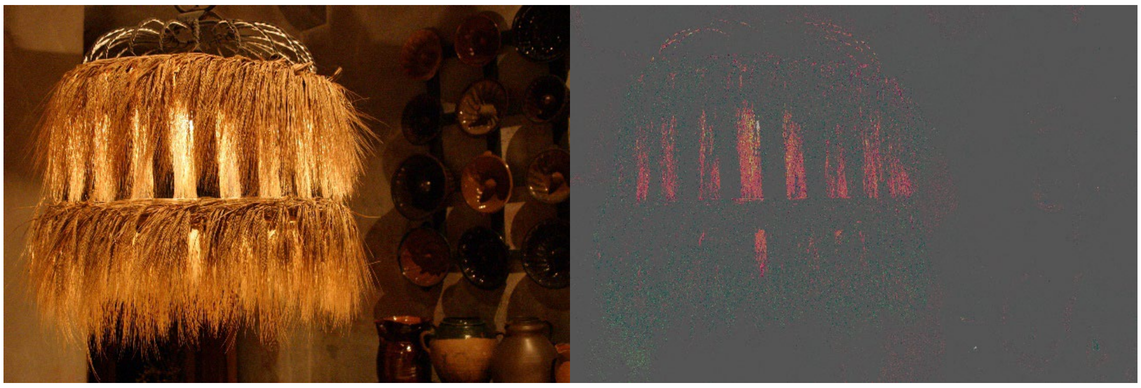
Kernel Size 17:



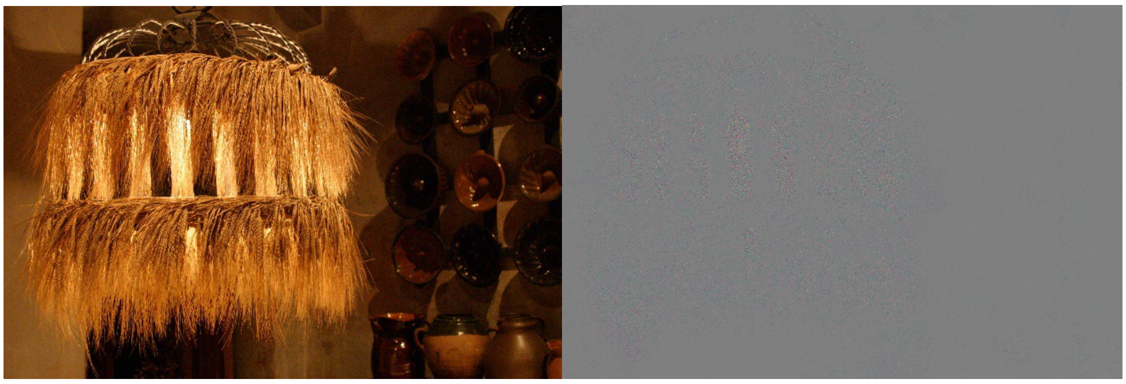
Selecting Best Method:

Each method has some advantages and disadvantages, for example piecewise bilateral good in removing noise in dark areas but it also loose more details than the other. Joint Bilateral is better for details however shadows and specularities are not removed. In detail transfer, details are protected while removing noises but not good in specularities and shadow also. The best method was **Shadow and specularity mask**   Because it give best result however as a disadvantages the mask threshold changes picture to picture.

Bilateral Filtering:

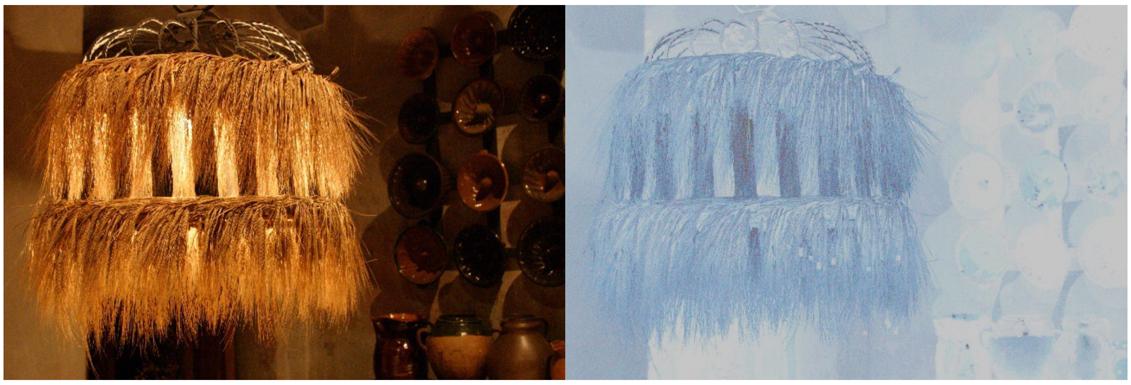


Joint Bilateral Filtering:



Detail Transform:

shadow and specularity masking:



2-) Gradient-domain processing:

To Gradient Domain processing pictures were in the png Format so I use the mpimg function to read files because it automatically normalizes the picture.

Differentiate and then re-integrate an image:

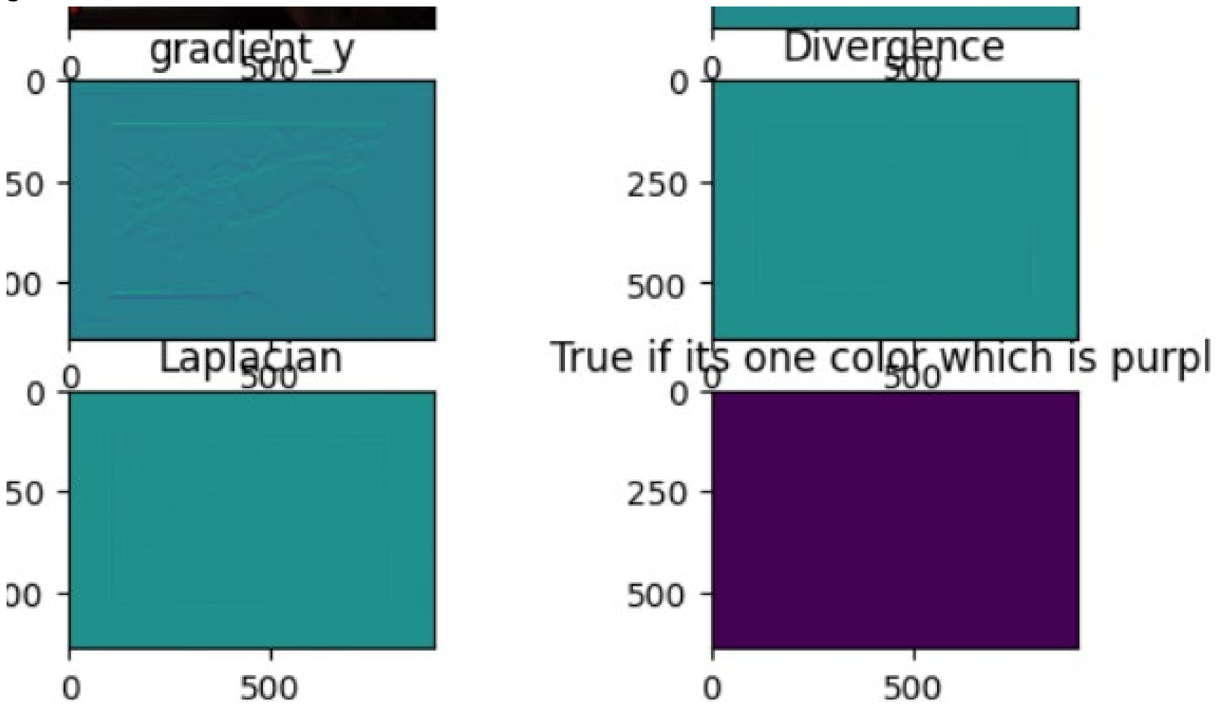
This was the straightforward described function in papers and description file, first to calculate the gradient I difference the image by along it rows and columns. Note that this was leading to losing shape due to difference. To protect the shape the image should be prepended to zeros.

Then to calculate divergence I re-difference to Gradients it was also leading to loss dimension to protect dimensions we should append the divergence.

For Laplacian first using convolve2d building function with filter, mode='same', boundary='fill', fillvalue=0 parameters I get Laplacian for all channels separately however later I need to do channelwise for using in for loop for conjugate gradient descend.

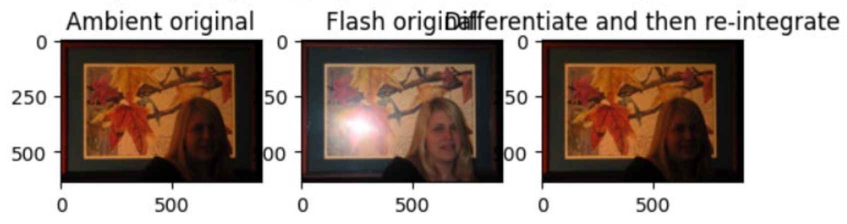
Then for passion solver I first describe the boundaries as given in description. Then apply the conjugate gradient descent I use the exactly given algorithm 1 in page 4.

To check code I compare to Laplacian and Gradient of the image it should give the close value and it give close value to me. These was the result:



Then applying all the channels, I got these result for 1000 iteration and 0.001 epsilon:

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
```



Create the fused gradient field:

This was the most complicated part for me first I create the GOC map, new gradient and pixel-wise saturation functions exactly same with as described in description. Then as described in documentation I create the fused gradient field.

After several try, I found the best parameters are following:

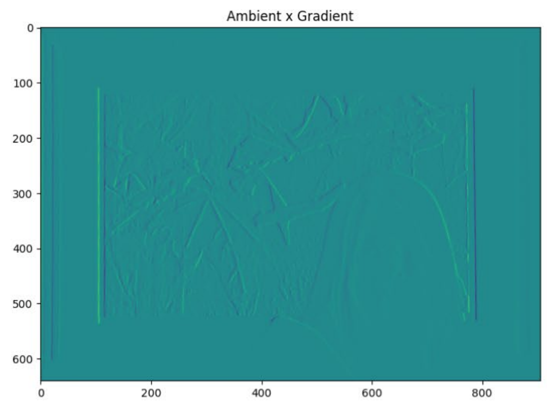
Sigma = 40

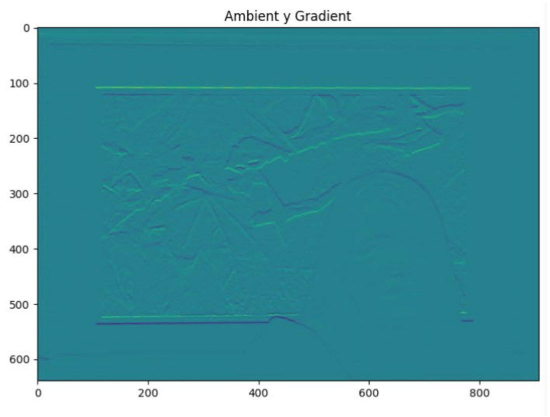
Tau = 0.7

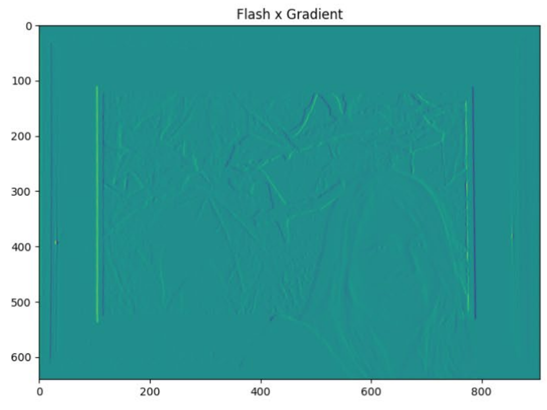
Max_iteration = 5000

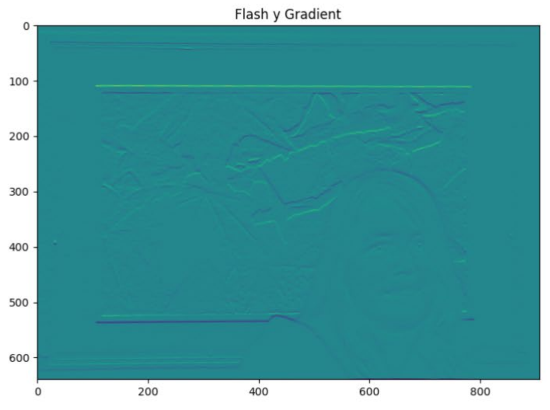
Epsilon = 0.001

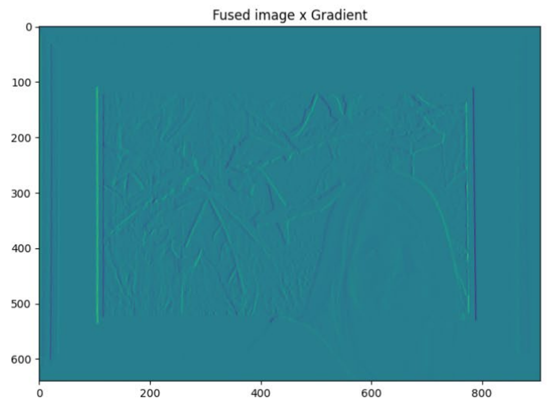
The Gradient Results:

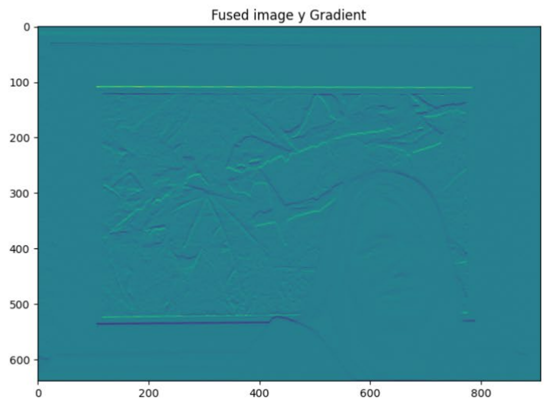










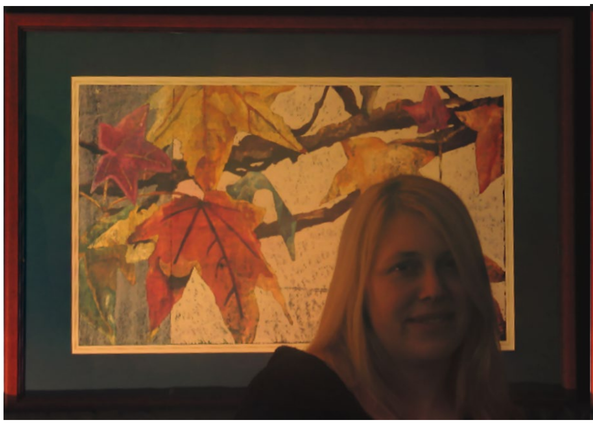


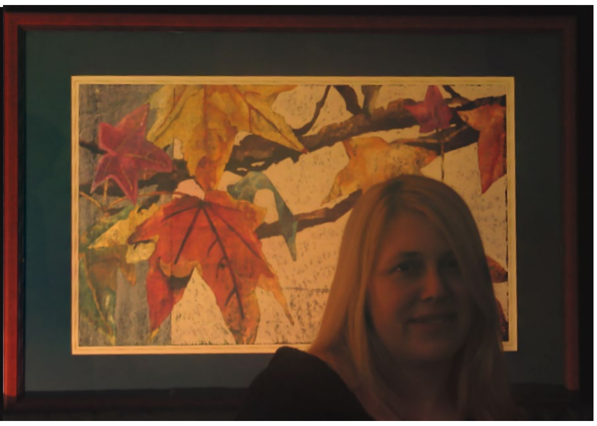
Initialization and boundary changes:

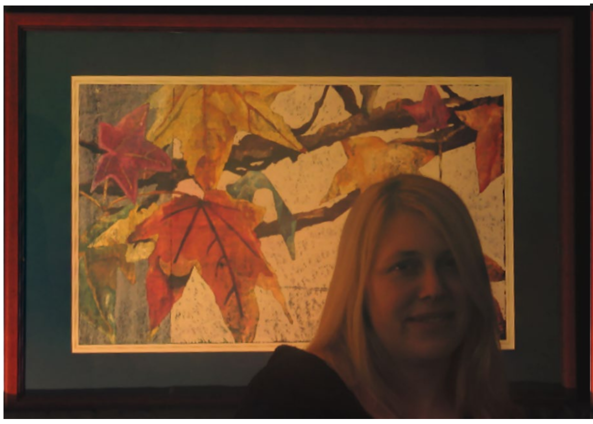
The Poisson solver is a robust algorithm that can produce accurate results from any initial starting point. However, using a better initialization that is closer to the desired output can significantly reduce computation time. When working with boundary conditions, we need to consider the differences between the regions. In our case, the boundary between the flash and ambient images is slightly brighter than the ambient boundary, and the average value is in between these two regions. Therefore, it is crucial for the Poisson solver to account for these differences to produce accurate results.

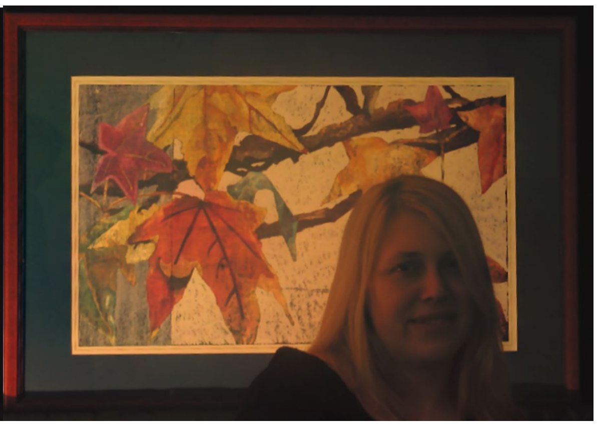
Below are results from the different 16 combination:

İnput Ambient:

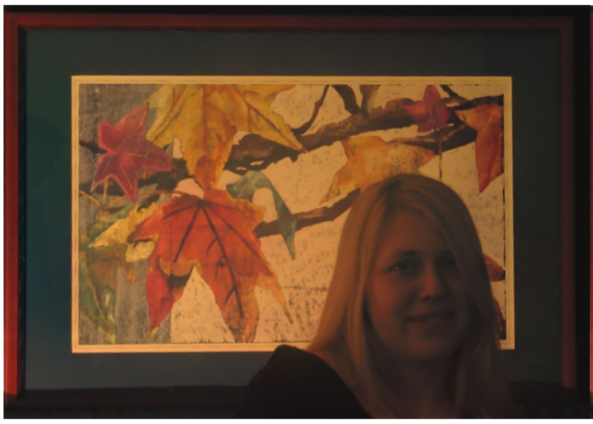

Ambient boundary

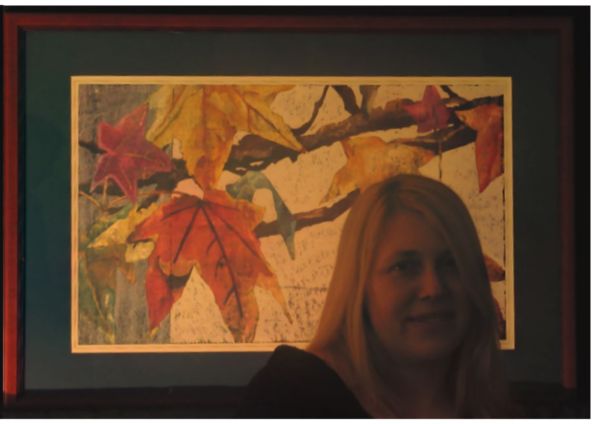

Average boundary


Flash boundary


Zeros boundary

İnput Average:


Ambient boundary


Average boundary

Flash boundary

Zeros boundary

İnput Flash:

Ambient boundary

Average boundary

Flash boundary

Zeros boundary

İnput zeros:

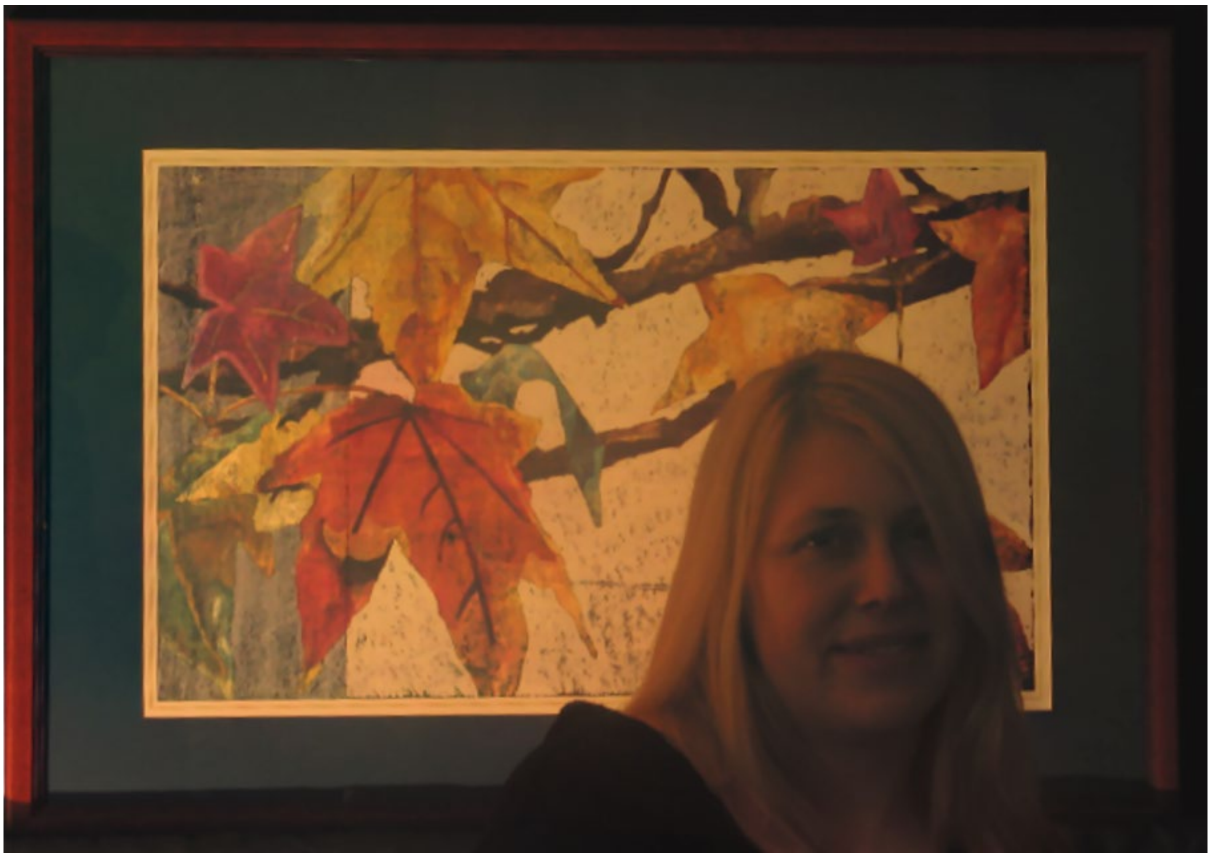
Ambient boundary

Average boundary

Flash boundary

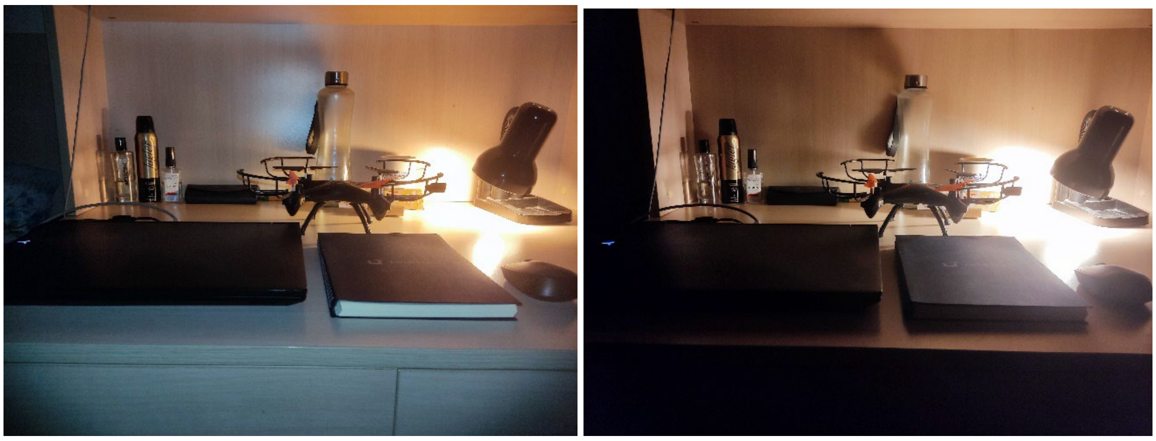
Zeros boundary

Result:



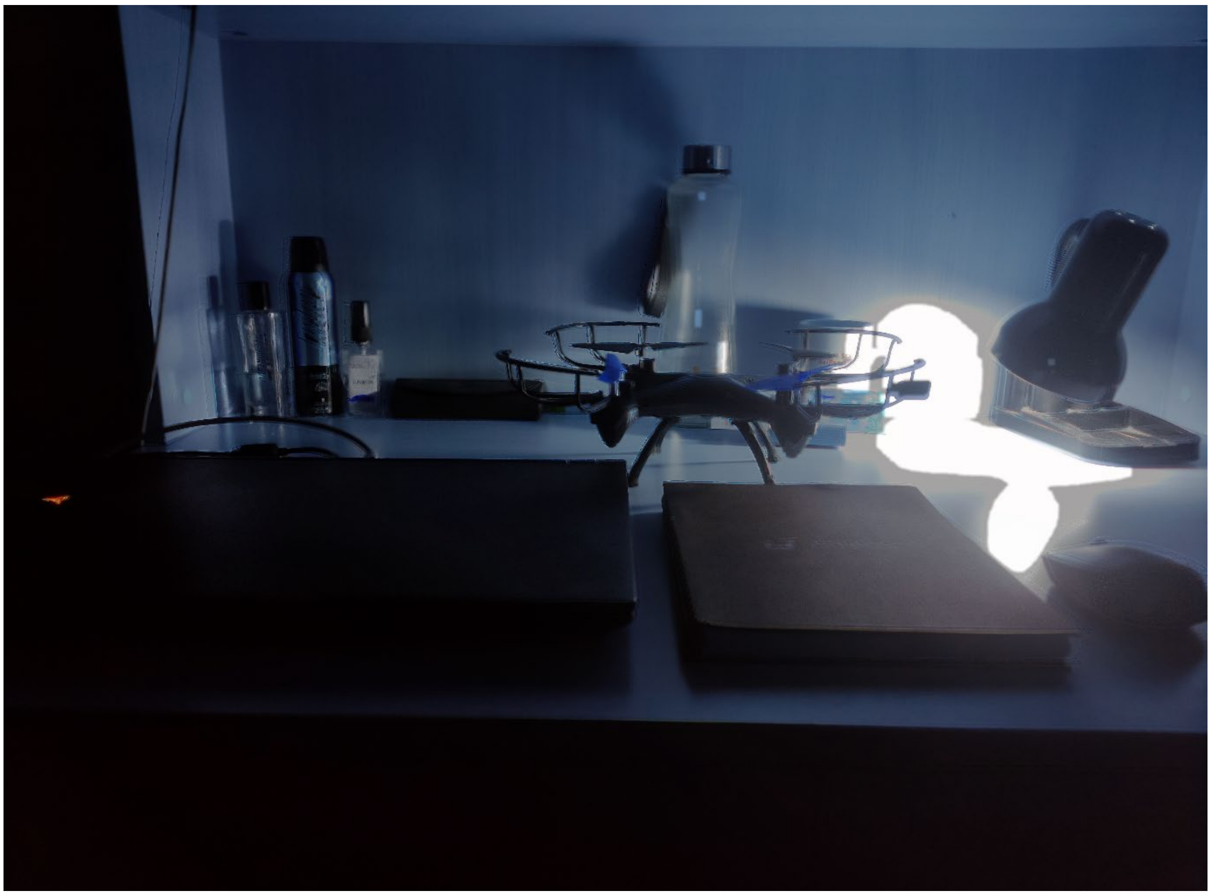
3-) (Bonus) Capture your own flash/no-flash pairs:

My initial Pairs to be implemented bilateral filter:



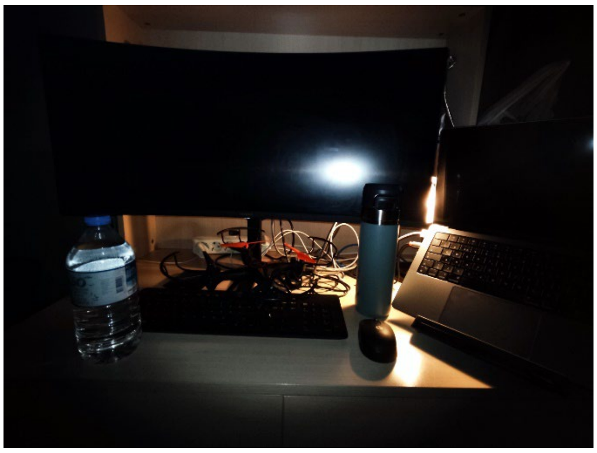
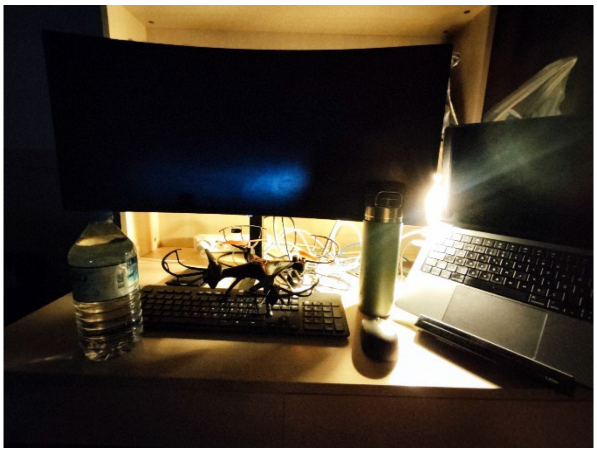
Left one with flash right one ambient.

Result:



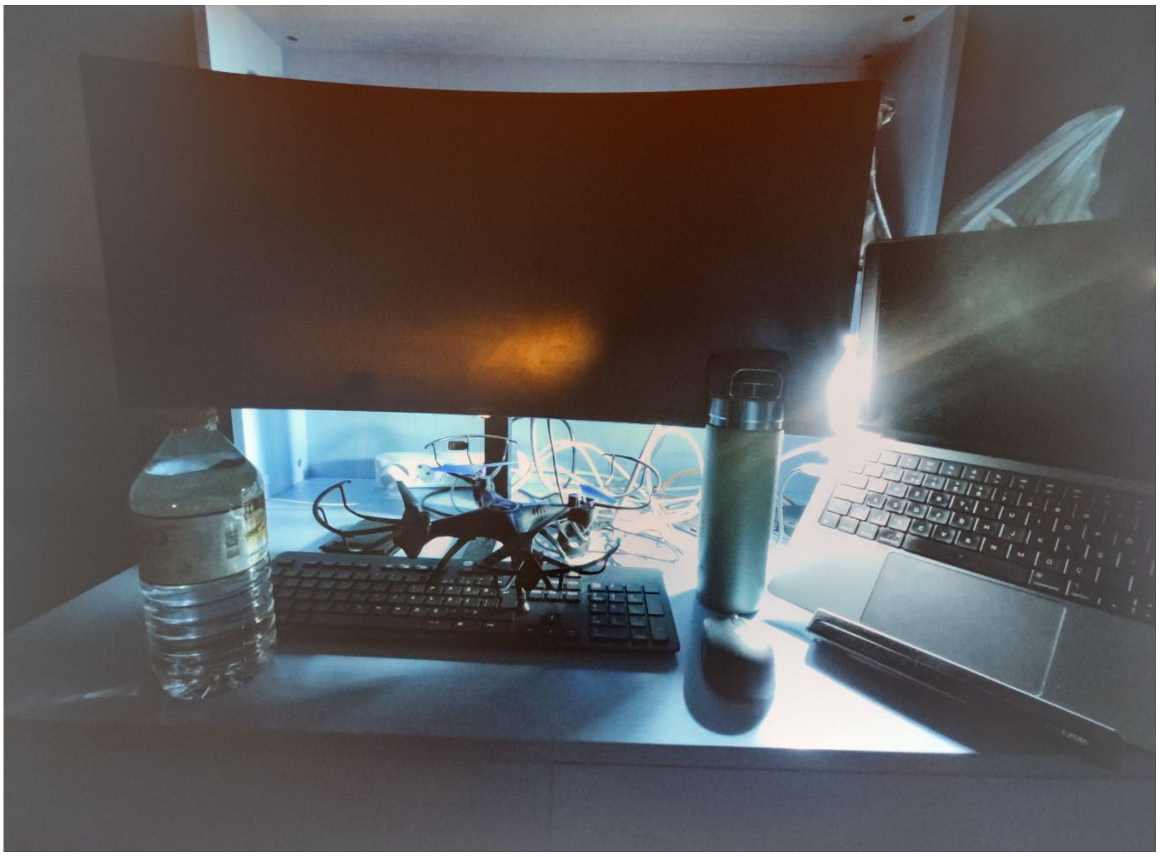
Actually it needs white balance algorithm however, I couldn't find enough time to implement our previous white balance algorithm after that it will be perfect

My initial Pairs to be implemented fused gradient:

 :

Left one with flash right one ambient.

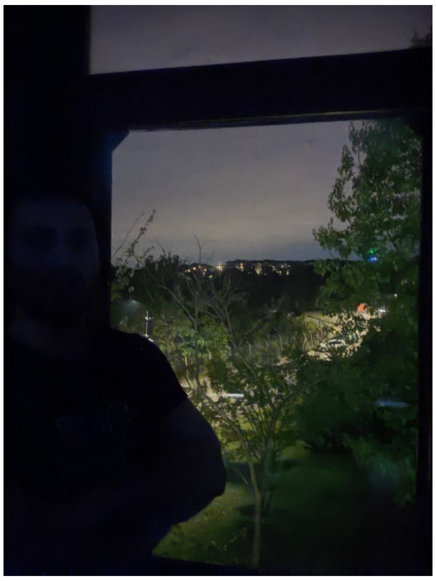
Result:



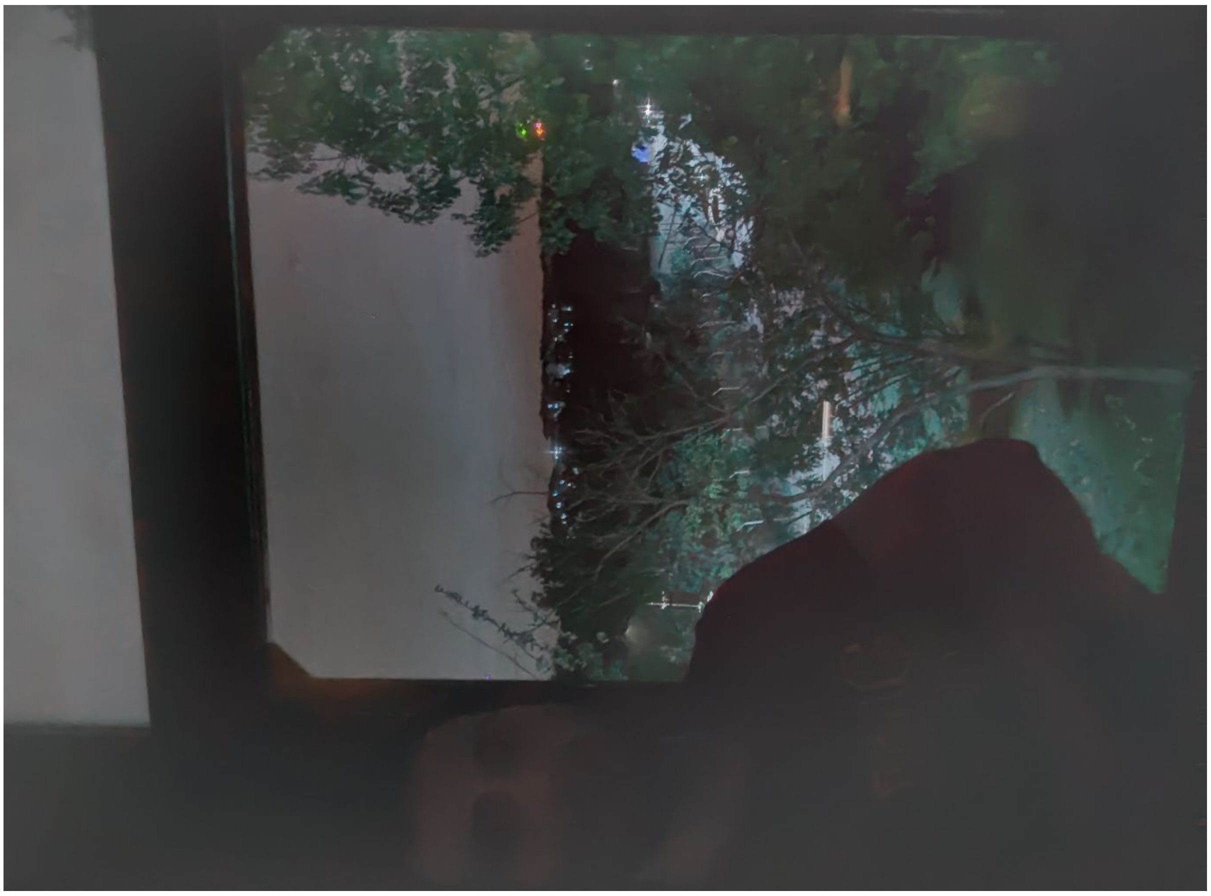
This was terrible result. During implementation 2 or 3 three time my computer collapsed. Colabl also collapsed several times. Since I couldn't find enough time, I stop at max 1337 iteration for solving in gradient field. Increasing the iteration to 10000 may give very good result if it does not converge earlier than this.

4-) (Bonus) Reflection removal in flash/no-flash photography

My initial phots:



Results:



Increasing maximum iteration for gradient field integration with conjugate gradient descent can give the better result. However, since my notebook not so good I am working on google colab. In even this image take around 7 hours 40 minute so increasing max iteration will lead to much more time requirement. Also additional masking and filtering operation can give much better result but since the demanded algorithm not uses additional filter despite the part 2 I did not implement additional filtering.