

CS 4732

MACHINE VISION

PROJECT 2

Image Enhancement

#### INSTRUCTOR

**Dr. Mahmut KARAKAYA**

**Michael Rizig**

**001008703**

**1. ABSTRACT**

In this project, we are given 3 tasks to complete, image transformations, histogram equalization, and noise reductio. For transformations, we use log and power transformations to expand a narrow contrast range, and experiment with the gamma values 1, 1.5 and 2.2 on each power transformation to see how they affect the result. We found that different combinations of log constants and gamma values gave varying results on image clarity and balance in the contrast. For histogram equalization, we create a function based on the image’s histogram distribution and attempt to normalize the histogram from a smaller spikelike shape to a even distribution. We found that transforming the data via division and manipulating the b values allowed us to find nearly perfect formulas to more equally distribute the histograms, leading to a much more balanced image. Finally for noise reduction, we try both average and median filtering in 3x3, 5x5, and 7x7 filter sizes and compare the results from each run to determine which size best fits our input image. We found that in general, the larger the filter size, the greater the blur effect. We also found that median filtering works better to counter salt and pepper noise without gaining too much blur in the process.

To view all edits, changes, and see step by step revision history, view this project on my GitHub:

<https://github.com/michaelrzg/CS4732-Projects>

**2. Test RESULTS**

**2.1 Test Results for log transformation.**

**(Only a few selected images are used here to highlight the effect. All output images can be found in output>log folder in the zip submission)**

**For the log transformation, we used the formula found in the slides:**

**T(x) = y = log (1+x) \* c**

**in these runs we experimented with the c values: {10,20,30,50,70,90,110,120,150 } and found varying results.**

**Image 1a:** Original Image ‘univeristy.png. **This is the input image.**

**Image 1b**: y = log(1+x) \* 20. Constant =20; we can see that the image actually gets a bit darker since our constant is too small.

**İmage 1c:** y = log(1+x) \* 30. Constant =30; bumping up our c, we see slightly more details under the garbanzo and in shadier parts of our image.

**İmage 1d**: y = log(1+x) \* 50. Constant =50; at this constant, we can see a lot more detail as compared to our input image.

**Image 1e**: y = log(1+x) \* 70. Constant =70; this seems to be the best constant for our image, as it allows us to brighten the darker areas without completely washing out the whites.

**Image 1f:** = y = log(1+x) \* 90. Constant =90; this constant seems too bright, giving the image a washed out look. Each run above this constant lead to even more washed out results.

|  |  |
| --- | --- |
| **A group of trees in a park at night  Description automatically generated**  **(a)** | A dark night in a park  Description automatically generated with medium confidence  **(b)** |
| **A dark night in a park  Description automatically generated with medium confidence**  **(c)** | **A group of trees in a park  Description automatically generated**  **(d)** |
| **A group of trees in a park  Description automatically generated**  **(e)** | **A group of trees in a parking lot  Description automatically generated**  **(f)** |

**Figure 1:** (a) Original image (input/university.png), (b) y = log(1+x) \* 20(output/log/logConstant-20.png), (c) y = log(1+x) \* 30(output/log/logConstant-30.png) ,(d) y = log(1+x) \* 50(output/log/logConstant-50.png), (e) y = log(1+x) \* 70(output/log/logConstant-70.png), (f) y = log(1+x) \* 90(output/log/logConstant-90.png)

**2.2 Test Results for power transformation.**

**(Only a few selected images are used here to highlight the effect. All output images can be found in output>log folder in the zip submission)**

**For the power transformation, we used the formula found in the slides:**

**T(x) = y = c \* xy = c \* x(y/gamma)**

**For the power transformation, we found that values close to or above 1 darkened the image and values below 1 brightened the image, so we opted to continuously decrease the y value. Since our ry term returns a value between (0,1), we used the constant 255 to scale it to our greyscale image.**

**We utilize the y values {.9,.8,.7,.6,.5,.4) for each pass.**

**For gamma we divide our power by each gamma value within the range, {1,1.5,2.2}.**

**Image 1a:** We begin with the original university.png. **This is the input image.**

**Image 1b:** y=.9 gamma = 1. We see once again that our image actually gets darker, meaning we need to lower our y value.

**İmage 1c:** y=.8, gamma =1. This image looks pretty similar to our input, no meaningful improvement.

**İmage 1d:** y=.6, gamma =1. This is where we start to see considerable improvement in our contrast. Our darkest points stay dark while comparatively lighter points scale up.

**Image 1e:** y=.4, gamma=1. At this point, the whites begin to get washed out, and the darker points lose their darkness, we know that at this gamma we’ve gone too low with our y value.

**Image 1f:** y=.9 gamma =1.5. Starting over and moving on to our next gamma, at .9 we see that our contrast is already dramatically changed for the better compared to gamma =1 at y=.9.

**Image 1g:** y=.8 gamma =1.5. At the next step, we see that the contrasts improve still, once again making a larger jump than our previous .8 at 1 gamma. This image looks the best in my opinion.

**Image 1h:** y=.6 gamma = 1.5; this step bumps the contrast up again, with the whites beginning to be overpowering.

**Image 1i:** y=.4 gamma = 1.5; At this point the contrast is way too bright, and the blacks are completely washed out.

**Image 1j:** y=.9 gamma = 2.2. We move on to the final gamma value 2.2, and see the same trend as before, with the image increasing in brightness dramatically with each step.

**Image 1k:** y=.8 gamma = 2.2; at .6 the image is as bright as .4 with the 1.5 gamma, meaning the curve for this graph would have a higher slope.

**Image 1l:** y=.6 gamma = 2.2; at this y we can clearly see the image is completely washed out, meaning we have gone way too far with our y value with this gamma.

**Image 1m**: y=.4 gamma = 2.2; once again we are completely washed out, and the image is unusable

|  |  |  |
| --- | --- | --- |
| **A group of trees in a park at night  Description automatically generated**  **(a)** | **A dark night with trees and buildings  Description automatically generated with medium confidence**  **(b)** | **A dark night in a park  Description automatically generated with medium confidence**  **(c)** |
| **A tree in a park at night  Description automatically generated**  **(d)** | A group of trees in a park  Description automatically generated  **(e)** | A tree in a park at night  Description automatically generated  **(f)** |
| A group of trees in a park  Description automatically generated  **(g)** | **A group of trees in a park  Description automatically generated**  **(h)** | A group of trees in a park  Description automatically generated  **(i)** |
| **A group of trees in a park  Description automatically generated**  **(i)** | **(j)** | **(k)** |
| **(l)** | **(m)** | - |

**Figure 2: (a)** original image (input/university.png), **(b)** y=.9 gamma=1 (output/power/gamma-1/yValue-0.9.png), **(c)** y=.8 gamma=1 (output/power/gamma-1/yValue-0.8.png), **(d)** y=.6 gamma=1 (output/power/gamma-1/yValue-0.6.png), **(e)** y=.4 gamma=1 (output/power/gamma-1/yValue-0.4.png), **(f)** y=.9 gamma=1.5 (output/power/gamma-1.5/yValue-0.9.png), (g) y=.8 gamma=1.5 (output/power/gamma-1.5/yValue-0.8.png), **(h)** y=.6 gamma=1.5 (output/power/gamma-1.5/yValue-0.6.png), **(i)** y=.4 gamma=1.5 (output/power/gamma-1.5/yValue-0.4.png), **(j)** y=.9 gamma=2.2 (output/power/gamma-2.2/yValue-0.9.png), **(k)** y=.8 gamma=2.2 (output/power/gamma-2.2/yValue-0.8.png), **(l)** y=.6 gamma=2.2 (output/power/gamma-2.2/yValue-0.6.png), **(m)** y=.4 gamma=2.2 (output/power/gamma-2.2/yValue-0.4.png)

**2.2 Test Results for Image Quantization (Greyscale)**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**3. CODES**

**3.1 Code for Image Negative**

% Name: MAHMUT KARAKAYA

% Number: 123456

% Project 1

close all;

clear;

clc;

% read the input image as input image

inimage = imread(flower.jpg');

% Show input image

figure,imshow(inimage,[]);

% Get the size of input image

[row,col,chan] = size(inimage);

% Predefine the output image

outimage = zeros(row, col,chan);

% Compute the effect pixel by pixel

for y = 1:1:row

for x = 1:1:col

for z=1:chan

% Get the negative of the pixel value

outimage(y,x,z) = 255 - inimage(y,x,z);

end

end

end

% Change the image format to uint8 before saving the result.

outimage = uint8(outimage);

% Show output image

figure(2),imshow(outimage,[]);

% Save the output image as image file.

imwrite(outimage,'output.jpg','jpeg');

**3.2 Code for Image Upside-Down**

% Name: MAHMUT KARAKAYA

% Number: 123456

% Project 1

close all;

clear;

clc;

% read the input image as input image

inimage = imread('lena.jpg');

% Show input image

figure,imshow(inimage,[]);

% Get the size of input image

[row,col,chan] = size(inimage);

% Predefine the output image

outimage = zeros(row, col,chan);

% Compute the effect pixel by pixel

for y = 1:1:row

for x = 1:1:col

for z=1:chan

% Get the upside down image

outimage(y,x,z) = inimage(row-y+1,x,z);

end

end

end

% Change the image format to uint8 before saving the result.

outimage = uint8(outimage);

% Show output image

figure(2),imshow(outimage,[]);

% Save the output image as image file.

imwrite(outimage,'output.jpg','jpeg');