1. Displaying High Dynamic Range Images

Part A:
The camera captures the scene with a contrast ratio of
\[
\frac{I_{\text{max}}^{\text{scene}}}{I_{\text{min}}^{\text{scene}}} = \frac{1000}{1}.\]

At the output of the $\gamma$-predistortion circuit with $\gamma = 3.0$, the ratio of max to min voltages is
\[
\frac{V_{\text{max}}}{V_{\text{min}}} = \left(\frac{I_{\text{max}}^{\text{scene}}}{I_{\text{min}}^{\text{scene}}}\right)^{1/3} = \frac{1000^{1/3}}{1^{1/3}}.\]

Then, at the output of the image display with $\gamma = 2.0$, the contrast ratio is
\[
\frac{I_{\text{max}}^{\text{output}}}{I_{\text{min}}^{\text{output}}} = \left(\frac{V_{\text{max}}}{V_{\text{min}}}\right)^2 = \frac{1000^{2/3}}{1^{2/3}} = \frac{100}{1}.\]

We see that the required contrast ratio for the display to represent the full dynamic range of the scene is 100:1.

Part B:
The grayscale HDR images (without any $\gamma$-nonlinearity mapping) look like the following.
Many details in the dark portions of these two images (e.g., the ceiling in memorial and the walls in atrium) are difficult to see, because the computer display has a limited contrast ratio.

Part C:
Applying a γ-nonlinearity mapping with γ = 2.8 for memorial and γ = 3.0 for atrium yields the following images. After applying the γ-nonlinearity mapping, almost all the details in the images are clearly visible. You may have chosen a slightly different value for γ for each image depending on your computer display and viewing conditions.

Part D:
Applying a γ-nonlinearity mapping with γ = 2.8 for memorial and γ = 3.0 for atrium to each of the RGB components, we get the following images.
By adjusting the value of $\gamma$ differently for the different color components, we can change the color appearance of the scene. For example, here is the result with $\gamma_R = 2.8$, $\gamma_G = 3.6$, and $\gamma_B = 2.4$ for memorial and with $\gamma_R = 2.0$, $\gamma_G = 3.4$, and $\gamma_B = 1.8$ for atrium. Using different values of $\gamma$ for the RGB components can distort the color balance in an objectionable, brightness-dependent manner. Using the same $\gamma$ value for the RGB components preserves the general color appearance.

MATLAB Code:

```matlab
% EE368/CS232
% Homework 1
% Problem: Displaying High Dynamic Range Images
% Script by David Chen, Huizhong Chen
clc; clear all;

imageFiles = {'memorial.hdr', 'atrium.hdr'};
gammaGray = [2.8 3.0];
gammaColorSame = [2.8 3.0];
gammaColorDiff = [2.8 3.6 2.4; 2.0 3.4 1.8];
warning off;
for nImage = 1:length(imageFiles)
    % Part B
    imgHDR = hdrread(imageFiles{nImage});
    imgHDRGray = rgb2gray(imgHDR);
    figure(1); clf;
    imshow(imgHDRGray, [0 1]);
    title('Grayscale HDR Image, No \gamma-Nonlinearity');

    % Part C
    imgHDRGrayGamma = (imgHDRGray).^(1/gammaGray(nImage));
    figure(2); clf;
    imshow(imgHDRGrayGamma, [0 1]);
    gammaStr = ['\gamma = ' num2str(gammaGray(nImage), '%.2f')];
    title(['Grayscale HDR Image, \gamma-Nonlinearity with ' gammaStr]);

    % Part D: same gamma
    imgHDRGammaSame = imgHDR.^(1/gammaColorSame(nImage));
    figure(3); clf;
```
imshow(imgHDRGammaSame, [0 1]);
gammaStr = ['\gamma = ' num2str(gammaColorSame(nImage), '%.2f')];
title({'Color HDR Image, \gamma-Nonlinearity with ' gammaStr});
figure(4); clf;
channelStr = 'RGB';
bins = linspace(0,10,200);
for nChannel = 1:3
    subplot(3,1,nChannel);
    imgChannel = imgHDRGammaSame(:,:,nChannel);
    counts = hist(imgChannel(:), bins);
    bar(bins, counts);
    axis([0 1 0 2e5]);
    xlabel('Value'); ylabel('Count');
    title([channelStr(nChannel) ' Component']);
end % nChannel

% Part D: different gamma
imgHDRGammaDiff = zeros(size(imgHDR));
for nChannel = 1:3
    imgHDRGammaDiff(:,:,nChannel) = ...
    imgHDR(:,:,nChannel).^1/gammaColorDiff(nImage,nChannel));
end % nChannel
figure(5); clf;
imshow(imgHDRGammaDiff, [0 1]);
gammaStr = ['\gamma_R = ' num2str(gammaColorDiff(nImage,1), '%.2f') ', ' ...
'\gamma_G = ' num2str(gammaColorDiff(nImage,2), '%.2f') ', ' ...
'\gamma_B = ' num2str(gammaColorDiff(nImage,3), '%.2f') ];
title({'Color HDR Image, \gamma-Nonlinearity with ' gammaStr});
figure(6); clf;
for nChannel = 1:3
    subplot(3,1,nChannel);
    imgChannel = imgHDRGammaDiff(:,:,nChannel);
    counts = hist(imgChannel(:), bins);
    bar(bins, counts);
    axis([0 1 0 2e5]);
    xlabel('Value'); ylabel('Count');
    title([channelStr(nChannel) ' Component']);
end % nChannel

if nImage == 1
    pause
end

end % nImage
warn
2. Denoising for Astrophotography

We show the original frame at $t = 30$ from the video, the denoised background frame at $t = 30$ without alignment, and the denoised background frame at $t = 30$ with alignment. Both denoised frames show much smaller noise levels. However, the denoised frame without alignment suffers from blurring of sharp features, e.g., the stars and the moon are severely blurred. With proper alignment, the frame can be denoised just as effectively, while better preserving the sharp features in the image.
Original Frame (t = 30)

Denoised Frame without Alignment (t = 30)
MATLAB Code:

```matlab
% EE368/CS232
% Homework 1
% Problem: Denoising for Astrophotography
% Script by David Chen, Huizhong Chen
clc; clear all;
labels = {'hw1_sky_1', 'hw1_sky_2'};
for nLabel = 1:length(labels)
    label = labels{nLabel};
    vrObj = VideoReader([label '.avi']);
    dxRange = -1 : 0.5 : 1;
    dyRange = -1 : 0.5 : 1;
    frameBackgroundUnaligned = im2double(read(vrObj, 1));
```
frameFirst = frameBackgroundUnaligned;
frameBackgroundAligned = frameBackgroundUnaligned;
[height, width, channels] = size(frameBackgroundUnaligned);
figure(1); clf; figure(2); clf; figure(3); clf;
numFrames = 30;
SNRs = zeros(1, numFrames);
for nFrame = 2:numFrames
    fprintf('Frame %d
', nFrame);
    % Read new frame
    frame = im2double(read(vrObj, nFrame));
    % Perform unaligned averaging
    alpha = (nFrame-1)/nFrame;
    frameBackgroundUnaligned = alpha*frameBackgroundUnaligned + (1-alpha)*frame;
    % Perform aligned averaging
    [frameAligned, dx, dy] = find_best_distorted_version(frame, ...
        frameBackgroundAligned, dxRange, dyRange);
    frameBackgroundAligned = alpha*frameBackgroundAligned + (1-alpha)*frameAligned;
    fprintf('dx = %f, dy = %f
', dx, dy);
    dxRange = dx - 1 : 0.5 : dx + 1;
    dyRange = dy - 1 : 0.5 : dy + 1;
    % Show frames
    if nFrame == numFrames
        figure(1);
        imshow(frame); title(sprintf('Frame %d', nFrame));
        imwrite(frame, ...
            sprintf('Results/%s_frame_original_%d.png', label, nFrame));
        figure(2);
        imshow(frameBackgroundUnaligned); title('Unaligned Background Frame');
        imwrite(frameBackgroundUnaligned, ...
            sprintf('Results/%s_frame_unaligned_%d.png', label, nFrame));
        figure(3);
        imshow(frameBackgroundAligned); title('Aligned Background Frame');
        imwrite(frameBackgroundAligned, ...
            sprintf('Results/%s_frame_aligned_%d.png', label, nFrame));
    end
end % nFrame
if nLabel < length(labels)
    pause;
end
end % nLabel

function [frameAligned, bestDx, bestDy] = ...
    find_best_distorted_version(frame, frameRef, dxRange, dyRange)
[height, width, channels] = size(frame);
frameGray = rgb2gray(frame);
frameRefGray = rgb2gray(frameRef);
minSSE = inf;
for dx = dxRange
    for dy = dyRange
        A = [1 0 dx; 0 1 dy; 0 0 1];
        tform = maketform('affine', A.);
        frameGrayTform = imtransform(frameGray, tform, 'bicubic', ...
            'XData', [1 width], 'YData', [1 height], 'FillValues', 0, ...
            'size', [height, width]);
        border = 5;
        frameSSE = sum(sum((...
            frameGrayTform(border:end-border, border:end-border) - ...
            frameRefGray(border:end-border, border:end-border) ...)
        ).^2));
        if frameSSE < minSSE
            minSSE = frameSSE;
            bestDx = dx;
            bestDy = dy;
        end
    end
end % for dy
end % for dx
end % function
end % dy
end % dx

A = [1 0 bestDx; ...
    0 1 bestDy; ...
    0 0 1];
tform = maketform('affine', A.);
frameAligned = imtransform(frame, tform, 'bicubic', ...
    'XData', [1 width], 'YData', [1 height], 'FillValues', [0;0;0], ...
    'size', [height,width]);
end
3. Image Subtraction for Tampering Detection

To detect the tampered regions in each painting, we perform image subtraction between the reference image and the tampered image. Below, we show the image subtraction results, without alignment first and with alignment second. The alignment searches over a range of horizontal and vertical shifts in the range $[-3,3]^2$. As can be observed, proper alignment is very important for accurate detection of the tampered local regions.

![Image Difference without Alignment for Irises](image1)

![Image Difference with Alignment for Irises](image2)
Image Difference without Alignment for *Starry Night*

Image Difference with Alignment for *Starry Night*
Next, we threshold the image difference with alignment. If the absolute difference is greater than $t = 0.1$, then we set the pixel to white. Otherwise, we set the pixel to black. Below, we show the results of thresholding, where we can observe each tampered image has three local alterations.
MATLAB Code:

```matlab
% EE368/CS232
% Homework 1
% Problem: Image Tampering Detection
% Script by David Chen, Huizhong Chen

clc; clear all;

% Process tampered images
tamperedImages = {'hw1_painting_1_tampered.jpg', ...
    'hw1_painting_2_tampered.jpg'};
referenceImages = {'hw1_painting_1_reference.jpg', ...
    'hw1_painting_2_reference.jpg'};
for nTest = 1:length(tamperedImages)
    % Load tampered image
    imgTampered = im2double(imread(tamperedImages{nTest}));
    [height, width, channels] = size(imgTampered);
    % Load reference image
    imgReference = im2double(imread(referenceImages{nTest}));
    % Subtract without alignment
    imgDiffUnalign = abs(rgb2gray(imgTampered) - rgb2gray(imgReference));
    border = 3;
    imgDiffUnalign([1:border height-border+1:height,:]) = 0;
    imgDiffUnalign(:,[1:border width-border+1:width]) = 0;
    figure(1); clf;
    imshow(imgDiffUnalign,
           'XData', [1 width], ...
           'YData', [1 height], ...
           'FillValues', [0,0,0].');
    % Perform alignment
    minSSE = inf;
    for dx = -3 : 3
        for dy = -3 : 3
            A = [1 0 dx; 0 1 dy; 0 0 1];
            tform = maketform('affine', A.);
            imgTform = imtransform(imgTampered, tform, 'bilinear', ...
                                     'XData', [1 width], ...
                                     'YData', [1 height], ...) ...
            imgSSE = sum(sum(sum((imgTform - imgReference).^2)));
            if imgSSE < minSSE
                minSSE = imgSSE;
                imgTamperedAlign = imgTform;
                bestDx = dx;
                bestDy = dy;
            end
        end
    end
    fprintf('Best dx = %.2f, dy = %.2f\n', bestDx, bestDy);
    % Subtract with alignment
    threshold = 0.1;
    imgDiffAlign = abs(rgb2gray(imgTamperedAlign) - rgb2gray(imgReference));
    imgDiffAlign([1:border height-border+1:height,:]) = 0;
    imgDiffAlign(:,[1:border width-border+1:width]) = 0;
    figure(2); clf;
    imshow(imgDiffAlign,[]); colorbar;
    figure(3); clf;
    imshow(imgDiffAlign > threshold);
    if nTest < length(tamperedImages)
        pause;
    end
end
```

14
4. Nighttime Road Contrast Enhancement

Part A:
The original images and their histograms of grayscale values look like the following.
For all three images, the large peak at very low grayscale values in the histogram corresponds to the large dark areas in these images, and the narrow range of nonzero counts in the rest of the histogram, which explains the low contrast of each image.

Part B:
The global histogram equalized images and their histograms of grayscale values look like the following.
By applying global histogram equalization, the small grayscale values (corresponding to dark regions) are spread out over the entire range of values from 0 to 255. Although contrast is improved, several negative side effects are observed after processing:

- The dark regions in each image (e.g., the sky) have a visually unpleasant noisy appearance.
- Effects due to nonuniform illumination in the scene are amplified, e.g., the light emanating from the streetlamp in the third image now appears unnaturally bright.
- Without considering local contrast, the visibility of certain objects is sometimes reduced. For example, the text on the yield sign in the first image is no longer clearly visible and some of the lane markings on the roads are more difficult to see.

These side effects can unfortunately distract the driver, so we should avoid generating these side effects as part of our enhancement algorithm as much as possible.

Part C:
The locally adaptive histogram equalized images and their histograms of grayscale values look like the following.
Now, the lane markings on the roads and edges/corners on the building facades, which were previously difficult to see, have much improved contrast and greater visibility. At the same time, the problems of amplification of noise and amplification of nonuniform illumination are mostly avoided. Since illumination can be very nonuniform in a scene during nighttime, locally adaptive histogram equalization is better suited to contrast enhancement of nighttime road images than global histogram equalization.

MATLAB Code:

```matlab
% EE368/CS232
% Homework 1
% Problem: Histogram Equalization
% Script by David Chen, Huizhong Chen
clc; clear all;

imageFiles = {'hw1_dark_road_1.jpg', ...
              'hw1_dark_road_2.jpg', ...
              'hw1_dark_road_3.jpg'};
for nImage = 1:length(imageFiles)
    % Read image
    img = imread(imageFiles(nImage));
```
% Calculate histogram
figure(1); clf; set(gcf, 'Position', [50 50 800 300]); subplot(1,2,1); imshow(img);
set(gca, 'FontSize', 12);
title('Original Image');
counts = imhist(img);
subplot(1,2,2); bar(0:255, counts/sum(counts));
set(gca, 'FontSize', 12);
xlabel('Graylevel'); ylabel('Probability'); title('Empirical PMF'); axis([0 255 0 0.25]);

% Perform global histogram equalization
imgGlobHistEq = histeq(img);
figure(2); clf; set(gcf, 'Position', [100 100 800 300]); subplot(1,2,1); imshow(imgGlobHistEq);
set(gca, 'FontSize', 12);
title('After Global Histogram Equalization');
counts = imhist(imgGlobHistEq);
subplot(1,2,2); bar(0:255, counts/sum(counts));
set(gca, 'FontSize', 12);
xlabel('Graylevel'); ylabel('Probability'); title('Empirical PMF'); axis([0 255 0 0.25]);

% Perform locally adaptive histogram equalization
clipLimit = 0.02;
numTiles = [16 16];
imgLocHistEq = adapthisteq(img, 'ClipLimit', clipLimit, 'NumTiles', numTiles);
figure(3); clf; set(gcf, 'Position', [150 150 800 300]); subplot(1,2,1); imshow(imgLocHistEq);
set(gca, 'FontSize', 12);
title('After Adaptive Histogram Equalization');
counts = imhist(imgLocHistEq);
subplot(1,2,2); bar(0:255, counts/sum(counts));
set(gca, 'FontSize', 12);
xlabel('Graylevel'); ylabel('Probability'); title('Empirical PMF'); axis([0 255 0 0.25]);

if nImage < length(imageFiles)
    pause
end
end % nImage