Solution to Problem Set 1

Assigned: Feb 9, 2006
Due: Feb 23, 2006

Note

The solution provided here is merely a reference. It might not be complete. Should you find any errors or parts difficult to understand, please write to TA.

Problem 1 *Pinhole Camera*

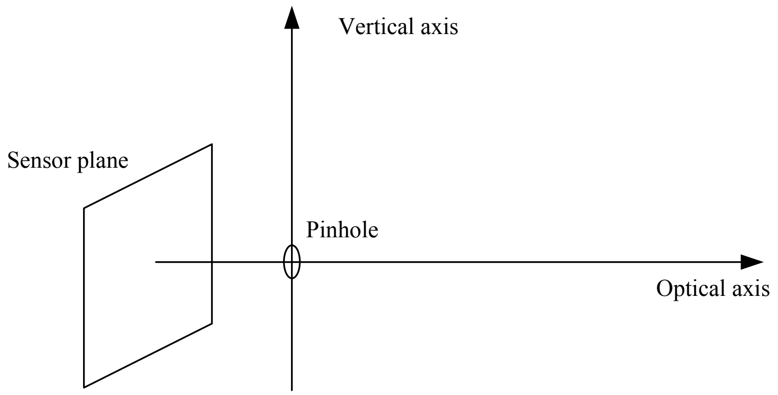


Figure 1: Pinhole camera schematic, defining the vertical and optical axes.

(a) With a pinhole camera, if you photograph a brick wall, with the sensor plane parallel to the brick wall, will the lines of the bricks be parallel or converging in the resulting photograph?

(b) Suppose you move the camera during an exposure. Under each of the following types of camera motion, will the amount that an object is blurred depend on its distance away from the camera (and why)

  (i) rotation about a vertical axis through the pinhole (vertical axis assumed to be parallel to the sensor plane).

   (ii) rotation about the optical axis.
   (iii) translation in a direction parallel to the sensor plane.
   (iv) translation perpendicular to the sensor plane.

(c) For a pinhole camera, how will the image change from one obtained using a small circular aperture (ignoring diffraction effects):
   (i) if you double the size of the aperture?
   (ii) if you change the aperture to a thin vertical slit?

Solution.

(a) *Parallel*. The key is that the projection of a line parallel to the sensor plane is also parallel to that line.

(b) We are asking whether the blur will depend on the distance. In other words, if disparity can be created from camera motion then the blur will depend on the distance. If there is no disparity then the blur is independent of the distance. Obviously when the camera rotates around vertical or optical axes there is no disparity, whereas when camera moves forward/backward or left/right disparity will be generated. So *no* for (i) and (ii). *Yes* for (iii) and (iv).

(c) (i) The image becomes *blurry* and *brighter*. (ii) The image is *vertically blurred*. □

Problem 2 *Warm-up Matlab Assignment*

Unzip file `ps1.zip`. Load image `street.jpg` into Matlab. This is the output data taken by a CCD color camera before tonescale adjustment. Apply a "gamma" tonescale correction of $I_{out} = I_{in}^{\gamma}$ where $\gamma = 0.5$. Display the two images side-by-side in your answer document.

Solution. Run the following MATLAB code

```
im=imread('street.jpg');
im=im2double(im);
imshow(im);
figure;imshow(im.^0.5);
```

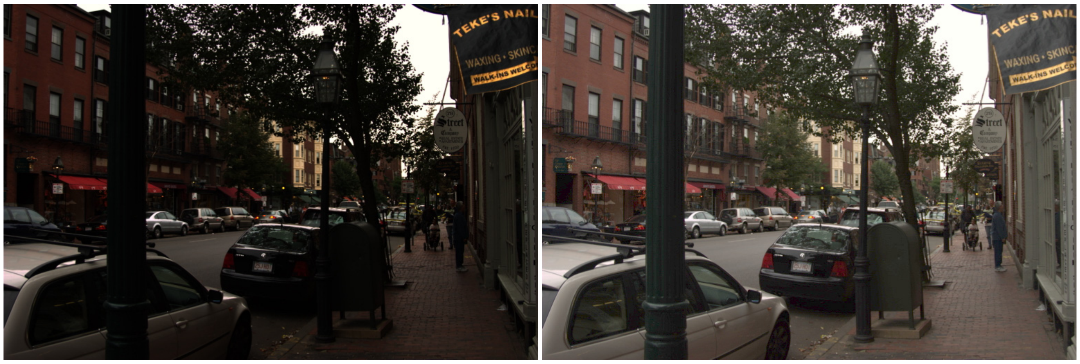

(a) (b)

Figure 2: Original image (a) looks more natural after gamma correction (b).

See Figure 2 for the image before and after gamma correction. □

Problem 3 *Color*

Load image `desk-orange.jpg` into MATLAB. The image is from web: `http://www.scifun.ed.ac.uk/card/images/left/desk-orange.jpg`. This is a photograph taken under Tungsten illumination. (Usually color correction is done before the tonescale adjustment, but for convenience in viewing the images, we will apply it after tonescale adjustment here.)

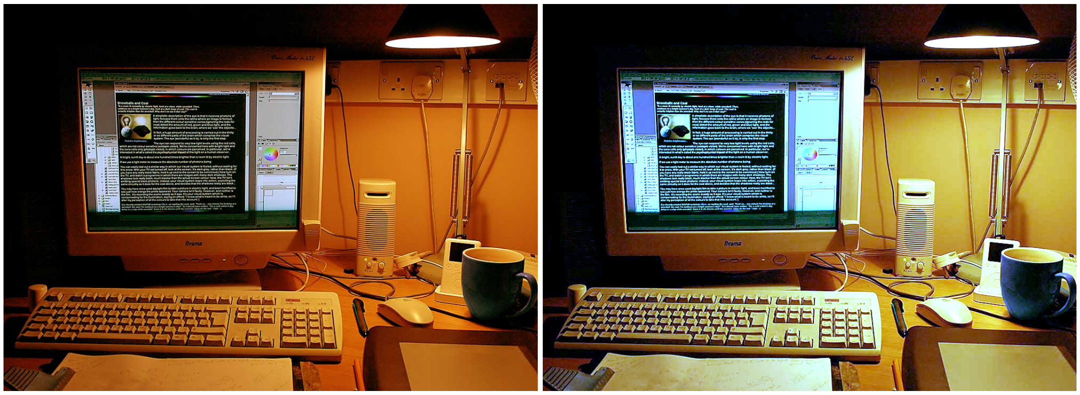
"Von kries adaptation" refers to restricting the 3x3 color transformation matrix to have zeros on all off-diagonal terms. What 3 entries for "von kries adaptation" color scaling is needed to provide proper color balance to this image, under the assumption of

(a) "gray world"–the average pixel intensity for each color channel is the same for all 3 color channels. To minimize the overall brightness change, assume that the average pixel value of the color balanced image is $[c, c, c]$ where c is the maximum of the mean value per each uncorrected RGB channel. Show, side-by-side, the uncorrected and color corrected images.

(b) the brightest pixel value in each color channel in the image is white? Show, side-by-side, the uncorrected and color corrected images.

Solution. Run the following MATLAB code

```
im=imread('desk-orange.jpg');
```



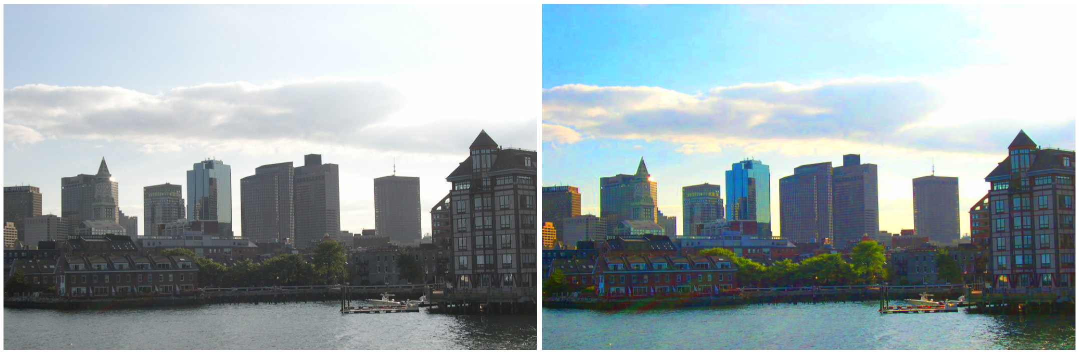
(a) (b)

Figure 3: The original image (a) is dim under the Tungsten illumination. It becomes less reddish after color balancing (b).

```
im=im2double(im);
[height,width,n]=size(im);
figure;imshow(im);

% convert image to a [npixels x 3] matrix
X=reshape(im,[height*width,3]);

% compute mean
H=mean(X,1);

disp(['mean of the image (R G B)  ' num2str(H)]);

% adjust the mean to be identical, matching the maximum
Y=X.*repmat(max(H)./H,[height*width,1]);

figure;imshow(reshape(Y,[height,width,3]));
```

See the images before and after color balance in Figure 3. For part (b), because there exist a white pixel in the original image, so the pixel values will not change after the proposed transform. □

(c) (*6.882 only*) For a generic, color-balanced image (not specific to the image above), what 3x3 matrix, applied to every pixel, will increase the color saturation of the image? Your answer should be a formula for the matrix, parameterized by the amount of color saturation desired. Load image `harbor.bmp` into Matlab, apply the transform, and display the two images



(a) (b)

Figure 4: The original image (a) looks grayish. It becomes more vivid after color saturation (b).

side by side.

Solution. The direction (unitary vector) $b_1 = \frac{1}{\sqrt{3}}[1,\ 1,\ 1]^T$ in RGB space may only change the brightness. There exist other two orthogonal unitary vectors b_2 and b_3 perpendicular to b_1. Magnify the RGB value in the space spanned by b_2 and b_3 should increase the saturation. So the transform matrix should be

$$\mathbf{A} = \begin{bmatrix} | & | & | \\ b_1 & b_2 & b_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} 1 & & \\ & \rho & \\ & & \rho \end{bmatrix} \begin{bmatrix} - & b_1^T & - \\ - & b_2^T & - \\ - & b_3^T & - \end{bmatrix} \tag{1}$$

where ρ is the ratio of saturation. The bases b_1, b_2 and b_3 can be found by the eigenvector of matrix $\mathbf{1}$ (all elements are 1). And why?

Here is the Matlab code. Note that in the code RGB is reshaped to be row vector. So `A` is indeed the transpose of $\mathbf{A}$ in Equation (1).

```
im=imread('harbor.jpg');
im=im2double(im);
[height,width,n]=size(im);
figure;imshow(im);

% convert image to a [npixels x 3] matrix
X=reshape(im,[height*width,3]);

% the amount to increase saturation
SatRatio=5;
```

```
[V,D]=eig(ones(3));

% forming the transform matrix
A=V*diag([SatRatio,SatRatio,1])*V';

% apply the transform matrix to the RGB samples
Y=X*A;

figure;imshow(reshape(Y,[height,width,3]));
```

See Figure 4 for the images before and after saturating. □

Problem 4 *Demosaicing*

Given image `watchtower.jpg`:

(a) synthesize the image observed through YGC striped sampling pattern.

(b) form the RGB demosaiced image, using a linear interpolation algorithm for demosaicing.

(c) form the RGB demosaiced image, using the median filter interpolation algorithm.

Assume

$$\begin{bmatrix} Y \\ G \\ C \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Solution. There are two approaches to do linear interpolation here. (1) Interpolate YGC color for each pixel in the YGC pattern, and then convert YGC to RGB. (2) First interpolate G for all the pixels, and then get R for Y patterns and B for C patterns. Interpolate R and B for all the pixels. Either of the two approaches is acceptable. The sample code here follows the second approach. For part (c) do median filtering on R-G and B-G and add them back. See the following MATLAB code.

```
im=imread('watchtower.jpg');
im=im2double(im);
[height,width,n]=size(im);

%----------------------------------
% forming YGC pattern
Y=im(:,1:3:end,:);
```

```
Y(:,:,3)=0;
foo=(Y(:,:,1)+Y(:,:,2))/2;
Y(:,:,1)=foo;
Y(:,:,2)=foo;

G=im(:,2:3:end,:);
G(:,:,1:2:3)=0;

C=im(:,3:3:end,:);
C(:,:,1)=0;
foo=(C(:,:,2)+C(:,:,3))/2;
C(:,:,2)=foo;
C(:,:,3)=foo;

Im=im;
Im(:,1:3:end,:)=Y;
Im(:,2:3:end,:)=G;
Im(:,3:3:end,:)=C;

figure;imshow(Im);

%---------------------------------
% demosaicing by interpolation
im_d=zeros(size(im));
[xx,yy]=meshgrid(1:width,1:height);

% interpolate G channel
xxg=xx(:,2:3:end);xxg=[xxg(:,1)-3,xxg,xxg(:,end)+3];
yyg=yy(:,2:3:end);yyg=[yyg(:,1) yyg yyg(:,end)];
G=squeeze(G(:,:,2));G=[G(:,1) G G(:,end)];
im_d(:,:,2)=interp2(xxg,yyg,G,xx,yy,'linear');

% interpolate R channel
xxr=xx(:,1:3:end);xxr=[xxr,xxr(:,end)+3];
yyr=yy(:,1:3:end);yyr=[yyr,yyr(:,end)];
R=Y(:,:,1)*2-im_d(:,1:3:end,2);R=[R R(:,end)];
im_d(:,:,1)=interp2(xxr,yyr,R,xx,yy,'linear');

% interpolate B channel
xxb=xx(:,3:3:end);xxb=[xxb(:,1)-3,xxb,xxb(:,end)+3];
yyb=yy(:,3:3:end);yyb=[yyb(:,1) yyb yyb(:,end)];
B=C(:,:,3)*2-im_d(:,3:3:end,2);B=[B(:,1) B B(:,end)];
im_d(:,:,3)=interp2(xxb,yyb,B,xx,yy,'linear');

figure;imshow(im_d);
```

```
%---------------------------------
% demosaicing by median filtering
R_G=im_d(:,:,1)-im_d(:,:,2);
R_G=medfilt2(R_G,[5,5]);
im_d(:,:,1)=im_d(:,:,2)+R_G;

B_G=im_d(:,:,3)-im_d(:,:,2);
B_G=medfilt2(B_G,[5,5]);
im_d(:,:,3)=im_d(:,:,2)+B_G;

figure;imshow(im_d);
```

See Figure 5 for results. ☐

(d) (*extra credit*) Put yourself in the "shoes" of an image demosaicing algorithm. Guess (by any means–eyeballing it, by plotting color values, by using a color interpolation algorithm) what the true values are for the missing color samples in `waterfall.bmp`. Assume that an initial color interpolation has been performed for the surrounding pixels and that we only have to infer the missing color values for the 4 central pixels of the image patch, which lies at the center of the image, or `im(64:65,64:65,:)` in MATLAB notation.

Your answer to this problem should be 8 numbers: the 2 missing colors in each of the 4 pixels that are missing color samples in the image patch. (The color sampling was made using a Bayer sampling pattern).

Scoring for this extra credit problem will be a function of the squared error distance between your estimated values and the *actual* pixel values that were present in the original image.

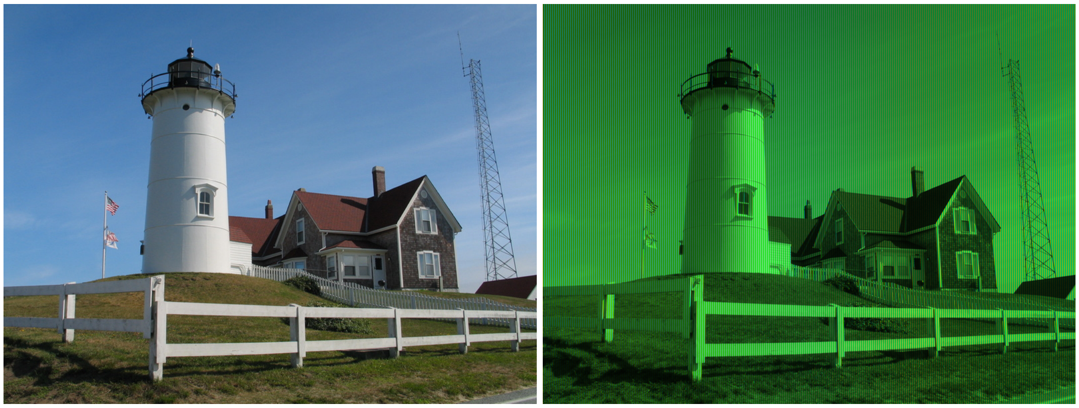
Solution. The RGB values for the four pixels are

$$\begin{matrix} [73\ 92\ 62] & [59\ 75\ 46] \\ [82\ 97\ 74] & [77\ 93\ 67] \end{matrix}$$

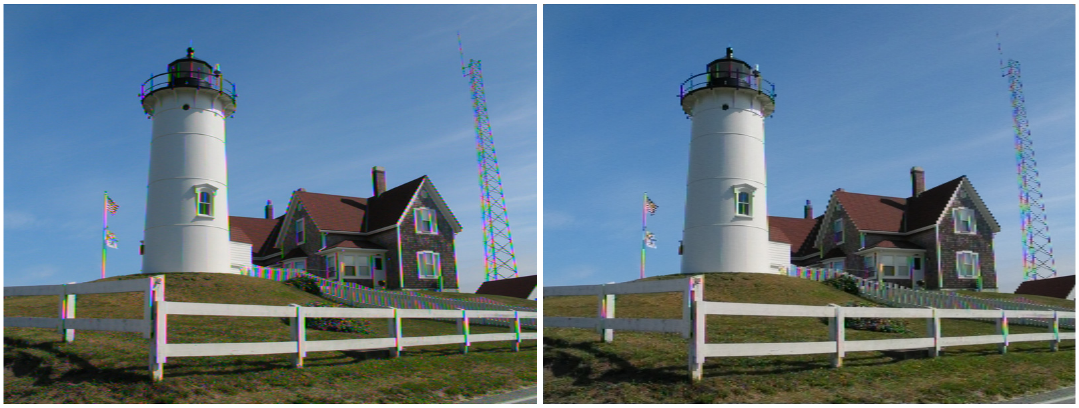
Or in range [0,1]

$$\begin{matrix} [0.2863\ 0.3608\ 0.2431] & [0.2314\ 0.2941\ 0.1804] \\ [0.3216\ 0.3804\ 0.2902] & [0.3020\ 0.3647\ 0.2627] \end{matrix}$$

For this problem you may use gradient information from G channel for interpolation, or guess from surrounding pixels (similar to example-based



(a) Original RGB image (b) YGC striped sampling patter



(c) Demosaicing by linear interpolation (d) Demosaicing by median filtering

Figure 5: Demosaicing by median filtering (d) has less color aliasing effect than linear interpolation (c).

texture synthesis). $\square$

Problem 5 *Color Metamerism (Not required; Extra Credit)*

The Matlab file `CIE.mat` contains the spectra which will be needed for this problem. The vectors c_x, c_y, and c_z give the CIE color matching functions for the X, Y, and Z coordinates, respectively. These functions are sampled at 5nm intervals for wavelengths from 400 through 700 nm.

Suppose you have a test color specified as a linear combination of three spectral primaries at 420, 520, and 620 nm, given by $(a, b, c)^T$, respectively. In terms of a, b, and c, what linear combination of light sources at 460, 510, and 590 nm is needed to give a perceptual match to the test color?

Solution. The equations can be established at that the coefficients of the two lights are the same, namely $X = \int_\Lambda f_x(\lambda)S_1(\lambda)d\lambda = \int_\Lambda f_x(\lambda)S_2(\lambda)d\lambda$, and this holds for both Y and Z. Let the combination coefficient of the light $[460, 510, 590]$nm be $[a', b', c']^T$. We have

$$
\begin{aligned}
X &= f_x(420)a + f_x(520)b + f_x(620)c = f_x(460)a' + f_x(510)b' + f_x(590)c' \\
Y &= f_y(420)a + f_y(520)b + f_y(620)c = f_y(460)a' + f_y(510)b' + f_y(590)c' \\
Z &= f_z(420)a + f_z(520)b + f_z(620)c = f_z(460)a' + f_z(510)b' + f_z(590)c'
\end{aligned}
$$

Since the matching functions are sampled at 5nm intervals for wavelength from 400nm to 700nm, the indices of 420, 520, 620, 460, 510 and 590 are 5, 25, 45, 13, 23, and 39, respectively. We have the new coordinate

$$
\begin{aligned}
\begin{bmatrix} a' \\ b' \\ c' \end{bmatrix} &= \begin{bmatrix} cx(13) & cx(23) & cx(39) \\ cy(13) & cy(23) & cy(39) \\ cz(13) & cz(23) & cz(39) \end{bmatrix}^{-1} \begin{bmatrix} cx(11) & cx(31) & cx(51) \\ cy(11) & cy(31) & cy(51) \\ cz(11) & cz(31) & cz(51) \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\
&= \begin{bmatrix} 0.3933 & -0.0776 & 0.0458 \\ -0.0693 & 1.3128 & -0.4880 \\ 0.0201 & 0.0718 & 0.8239 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix}
\end{aligned}
$$