TSBB09 Computer Exercise A
The Digital Camera

Developed by Per-Erik Forssén and Maria Magnusson 2008.

Computer Vision Laboratory, Linköping University, Sweden

Updated by: Maria Magnusson, Klas Nordberg and Marcus Wallenberg.
Last update: November 2021

Contents

1 Preliminaries

☞ Before attending the computer exercise it is necessary to read through the course material [1] as well as this guide to the exercise. The guide contains a few home exercises to be answered before the session. They are all clearly marked with a pointing finger.

Extra Implementation exercises marked with an "Extra" box may be temporarily skipped and completed when the other exercises are finished.

Examples of MATLAB commands frequently used are `imread`, `imshow`, `size`, `zeros`, and `reshape`. If you are unfamiliar with any of these commands, you should use the MATLAB `help` function to find out how what they do.

1.1 Displaying images in Matlab

There are two commands for displaying images in Matlab, `imagesc` and `imshow`. The second one automatically scales the image so that the pixels become rectangular, removes the axes and sets the colortable to grayscale. Consequently

```
imagesc(Im);                  or                imshow(Im, []);
axis image;
axis off;
colormap gray;
```

give the same result. In the commands above, `Im` contains the image and `[]` automatically finds the minimum and maximum of the image and displays it with a linear scale in between. Note that `[]` is not needed in `imagesc`. There exists also other colortables, i.e. `jet`, or it is possible to design your own colortable.
If another range of pixel values is desired, you can write

```
imagesc(Im, [min max]);         or          imshow(Im, [min max]);
```

giving that the image is displayed with a linear scale between the values `min` and `max`.
The command

```
colorbar;
```

is useful in connection with displaying images. It shows how the colors in the image correspond to pixel values.
The command

```
imshow(Im);
```

display the image with a linear scale between 0 and 1. Consequently, there is *no automatical* scaling due to pixel values. When true color images (with three colorplanes R ,G, and B) are going to be displayed, the pixel values should be scaled to the interval between 0 and 1.
One exception to what have been said above is 8-bit images of class uint8. For such images just

```
imshow(Im);
```

works fine.

2 Tasks

Today digital cameras are ubiquitous. Almost all consumer still image cameras use digital sensors, and digital cameras can also be found in most mobile phones and laptops. In this assignment we will look at the output from a sensor that uses a Bayer pattern colour filter array (CFA) to capture colour images. This is the most common CFA in use today, and it can be found in both still-image cameras and camcorders. We will also have a look at interlaced images, which are common on analogue video camcorders. These images have been digitized with a framegrabber.

The different tasks of this exercise are relatively independent of each other and, therefore, they can be solved in arbitrary order. Some useful files are located in a common directory. Start by adding a path to these directories in the MATLAB window:

```
addpath /courses/TSBB09/DigitalCamera/
addpath /courses/TSBB09/DigitalCamera/shadingcorr
addpath /courses/TSBB09/DigitalCamera/bayer
```

2.1 Shading correction

Shading correction is described in the appendix. Our goal in this assignment is to correct for uneven illumination. Note that uneven illumination is only one example what can be cured by shading correction. *Shading correction can also be applied to correct errors inside the camera caused by the sensor elements and the optics. A main application in consumer cameras is to correct for vignetting. This procedure is rather called vignetting correction instead of shading correction.*

☞ **Home exercise** Assume that you have two reference images, one dark and one bright, given by $b_A(x, y)$ and $b_B(x, y)$, respectively. Also assume that you have an original image $b(x, y)$ taken with the camera. A corrected image will be computed as $\hat{f}(x, y) = c(x, y)\,(b(x, y) + d(x, y))$. The values in $b_A(x, y)$ should be corrected to $\hat{f}_A(x, y)$ and the values in $b_B(x, y)$ should be corrected to $\hat{f}_B(x, y)$. Note that $\hat{f}_A(x, y)$ and $\hat{f}_B(x, y)$ are constant values independent of (x, y).

QUESTION 1: Express c and d as functions of b_A, b_B, $\hat{f}_A$ and $\hat{f}_B$.

Next we will perform shading correction in practice. The following procedure was performed in advance:

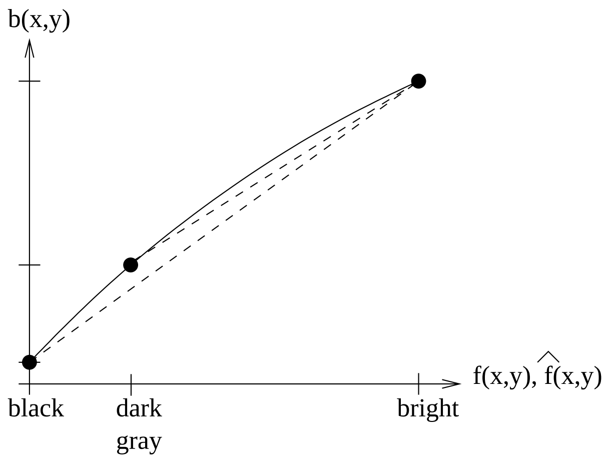
- A black cloth was selected as background. Uneven illumination was introduced by tilting the lamp. The introduced uneven illumination *was not* changed during the rest of the experiment. A network camera was arranged to "look at" the scene.
- Then, the background was covered with a white cloth to serve as the bright reference image. The bright reference image $b_B(x, y)$ was taken and named *whiteimage*.
- Then, the white cloth was removed so that the black cloth became visible again. The first dark reference image $b_B(x, y)$ was taken and named *darkimage*. This dark reference image is dark gray rather than black, because the scene was highly illuminated by the lamp.
- Then, the lens was blocked and a second dark reference image $b_B(x, y)$ was taken and named *blackimage*. This image is almost black.
- Then, the blocking of the lens was removed and a scene with different objects was arranged on the black cloth background. The original image $b(x, y)$ was taken and named *origimage*.

Copy the MATLAB program `shadcorr.m` to your home directory and execute it. The four images mentioned above should show up. We now have three reference images. The second dark reference image produced by blocking of the lens is the most common type of dark reference image.

QUESTION 2: In our case, however, it may be a little better to instead use the dark gray image as the dark reference image – why? Use the figure below to answer the question.

The shading corrected image should not have intensity values that deviate too much from the original image. Therefore, the reference levels should be chosen wisely. The value $\hat{f}_B$ can, for example, be selected to the average of the bright image and the value $\hat{f}_A$ can be selected to the average of the dark reference image: `mean(mean(darkimage))`

QUESTION 3: Which values do you select for $\hat{f}_A$ and $\hat{f}_B$?



Write MATLAB code for shading correction and be prepared to show it later to the teacher. It can be helpful to use the jet colormap during development to see better, i.e. `colormap(jet)`.

QUESTION 4: Did you succeed in correcting for the uneven illumination?

QUESTION 5: What is the result if you try the black image as a dark reference image? Which values do you select for $\hat{f}_A$ and $\hat{f}_B$?

QUESTION 6: Check if your corrected image is noisier than the original image! Try to explain why!

QUESTION 7: In practice, the bright and dark reference images are often obtained by averaging a number of bright and dark images, respectively. What could be the reason for this?

2.2 Bayer pattern interpolation

Home exercise An excision of a *Bayer image* shown below. A part of the a corresponding *Bayer pattern* is indicated. Your task is to derive *Rimage*, the resulting red (R) color plane for the image excision.

0	0	0	0	0
0	0	1	0	0
0	1	2	1	0
1	2	3	2	1
2	3	3	3	2

Bayer image

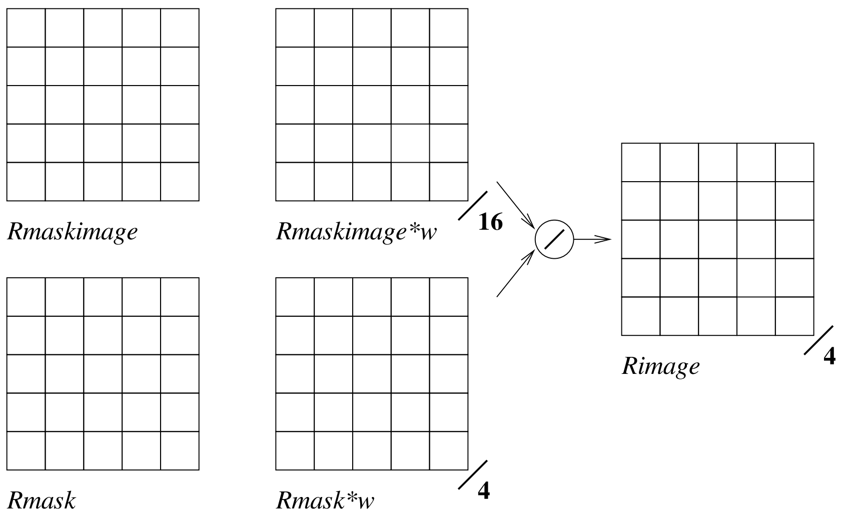
G	R			
B	G			

Bayer pattern

Rimage ?

QUESTION 8:
Fill in the correct pixel values in the figure below. Proceed as follows:
• *Rmask* is an image with ones at the R positions of the Bayer pattern.
• *Rmaskimage* is *Rmask* multiplied with the *Bayer image.*
• Convolution is performed with the averaging filter w shown below (center is marked in boldface). For simplicity, assume that *Rmask* is repetitive and that pixels outside *Rmaskimage* are zero.
• The two convolved images are divided with eachother. This operation (2 convolutions with w and one division) is denoted *normalized averaging.*

$$w = \frac{1}{4}\begin{array}{|c|}\hline 1 \\ \hline \mathbf{2} \\ \hline 1 \\ \hline \end{array} * \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & \mathbf{2} & 1 \\ \hline \end{array} = \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & \mathbf{4} & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$



Start MATLAB and add paths as described in the beginning of the Task section. Load one of the images from the folder `bayer/` into the variable `im`. Also make sure you convert the image to `double` format using the function `im2double`. (This function also scale the image to the interval between 0 and 1.)

The images in the folder `bayer/` are raw sensor images from a camera with a Bayer colour filter array. This means that each pixel receives light transmitted through an *optical* filter placed over the photo detector. The images are all captured under constant illumination, and there is no motion in the scene. This is important as we will later use them for estimation of the noise characteristics for the used camera.

Bayer patterns come in four basic flavours, shown in table 1. Display the image using `imshow`, and use the zoom tool to examine details of the image.

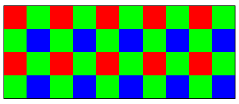
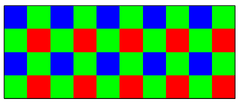
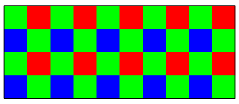
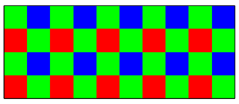
			
RGGB	BGGR	GRBG	GBRG

Table 1: Bayer patterns

Home exercise We will now create masks for the three different colours. Start by creating three masks for the GBRG colour array: commands:

```
mask1=zeros(rows,cols);
mask1(2:2:end,1:2:end)=1;
mask2=zeros(rows,cols);
mask2(...
mask2(...
mask3=zeros(rows,cols);
mask3(...
```

QUESTION 9: What were the missing indexing commands:

Using these masks, we can colour the pixels in our input image `im` according to:
`im_rgb=reshape([im.*mask1 im.*mask2 im.*mask3],[size(im) 3]);`

Try the four different colour arrays, and look at the result using `imshow`.

QUESTION 10: Which of the four colour filter arrays in table 1 is the correct one? (Hint: The dust bin on the lower left is blue)

The colour image we generated above was incorrect in the sense that in each pixel, only one of R,G, or B had a non-zero value. To obtain values for the missing colour bands in each pixel, we will now employ a technique known as *normalized averaging*, or *non-linear mean filtering*. First we define an interpolation kernel:

```
f=[1 2 1]/4;
```

We will apply this filter both horizontally and vertically to the red pixels, using the `conv2` command:

```
img=conv2(f,f,im.*mask1,'same');
```

QUESTION 11: Compare the means of `img` and `im`. Why are they different?

Now instead normalise the result with the filtered mask:

```
img=conv2(f,f,im.*mask1,'same')./conv2(f,f,mask1,'same');
```

QUESTION 12: Again compare the means. Are they more similar now? Briefly explain how normalized averaging works.

Now apply the same filtering on the green and blue channels, and fuse the result using `reshape`. Look at the result using `imshow`. Zoom in on the details, and try to find artifacts.

QUESTION 13: What artifacts can you find?

The interpolation scheme we have just implemented is the most basic one, and is used in many still-image cameras. To avoid the artifacts, professional photographers save images in RAW (i.e. Bayer) format, and use e.g. Adobe Photoshop or iPhoto to obtain colour images.

WRITE MATLAB CODE: Write a function `raw2rgb.m` that takes as input a raw array in `uint8` format, and converts it to an RGB image using the interpolation scheme you just used.

```
function rgb=raw2rgb(raw)
...
...
...
```

Note that this function will be used in the next section!

Extra

The raw image `bayer_image.png` can also be converted into a rgb-image.

QUESTION 14: Which colour filter array is the correct one for this image?

2.3 Noise measurements

Now, read all the images in the noise measurement dataset to memory:

```
fpath='bayer/';

im=cell(100,1);
for k=1:100,
    fname=sprintf('raw%04d.png',k);
    im{k}=imread([fpath fname]);
end
```

Use your new Matlab function `raw2rgb` to verify that one image, e.g. No. 100, looks okay. Then show (a section of) all images in a time sequence to verify that they indeed contain different instances of noise:

```
figure(1)
imshow(raw2rgb(im{100}));
figure(2)
for k=1:100,
   tmp = raw2rgb(im{k});
   tmp = tmp(401:480, 871:980, :);
   imagesc(tmp);
   drawnow;
end
```

Next compute the average image **of the raw images** in the dataset, using a `for` loop over `im{k}`. Call the average image `imm`. Make sure you convert the images to `double` before averaging by using `im2double`. (This function also scales the image to the interval between 0 and 1.) Use `raw2rgb` to convert your raw average image to an rgb average image: `immrgb = raw2rgb(imm);`

Now look at the average image and one of the individual images side by side, using `subplot`. Zoom in on the details.

QUESTION 15: Can you notice noise reduction in the average image?

We also need a variance image for the dataset. This is an image with pixel values corresponding to the variance of the pixel across the sequence. This can be obtained by averaging `im{k}.∧2` across the sequence, and then subtracting `imm.*imm`. Call your variance image `imv`. Use `raw2rgb` to convert your raw variance image to an rgb variance image: `imvrgb = raw2rgb(imv);`

QUESTION 16: Look at your variance image with `imshow`. You will need to scale the image since (r,g,b)-valus above (1,1,1) are shown white and (r,g,b)-valus below (0,0,0) are show black. Scale the image so that the floor looks pink and blue. Which scale factor did you use?

QUESTION 17: Where does the noise have the highest *absolute* variance?

The ratio between the signal power S and the noise power N is an important quality measure of the image. It comes in different variants, like $\text{SNR}_{\text{DB}} = 10\log_{10}(S/N)$ and $\text{SNR} = S/N$.

QUESTION 18: N is proportional to the noise variance, but how can you compute S? Compute an image showing $S/(N + \text{eps})$. Scale the image so that it looks approximately as the original image, but more yellowish. Which scale factor did you use?

QUESTION 19: Where is SNR largest, in bright areas or in dark areas of the image?

Finally, we are going to collect and merge measurements from the same intensity, and plot noise variance curves as a function of intensity for each colour. Type in the following script, and run it.

```
indim=uint8(imm*255);
rhist=zeros(256,1);
ghist=zeros(256,1);
bhist=zeros(256,1);

for k=0:255,
    k
    redk=find(mask1);
    indk=redk(find(indim(redk)==k));
    rhist(k+1)=sum(imv(indk))/(length(indk)+eps);

    greenk=find(mask2);
    indk=greenk(find(indim(greenk)==k));
    ghist(k+1)=sum(imv(indk))/(length(indk)+eps);

    bluek=find(mask3);
    indk=bluek(find(indim(bluek)==k));
    bhist(k+1)=sum(imv(indk))/(length(indk)+eps);
end
```

Now, plot all three curves in one graph:

```
plot(0:255,rhist*255^2,'r',0:255,ghist*255^2,'g',0:255,bhist*255^2,'b');
axis([0 255 0 10])
grid on
```

QUESTION 20: The noise we get when measuring light photons has approximately a Poisson distribution. It can be shown that such noise variance depends linearly on the image intensity. What about your curves?

QUESTION 21: Remember that your S/N image looked yellowish compared to the original image. Can you explain that by looking at the three curves?

QUESTION 22: Why do the curves look the way they do near 255? Could you see this effect in the variance image also?

QUESTION 23: For blue, there were actually no pixel values at 0, 1 and 2 and for green and red there were no pixel values at 0 and 1. Therefore, the curves should be extrapolated towards 0 giving a small offset. Which type of noise in the measurement process may this offset correspond to?

2.4 Deinterlacing

Analogue video comes in two main formats: *interlaced* and *progressive scan*. In progressive scan each image is taken at one point in time, alternatively the lines are taken sequentially one at the time from top to bottom. For interlaced scans, however, the odd scan lines come from one point in time, and the even scan lines come from the next following time point. These two sets of scan lines are called the *fields* of the interlaced image. Interlaced mode allows double the temporal frequency at the same bandwidth as progressive mode, which is an advantage in the old systems for broadcast television. The disadvantage is that still images from an interlaced sequence typically contain even and odd lines from two different time points, making such an image appear distorted if there is much motion in the image.

When analogue video is digitized, using a *framegrabber*, the images obtained also have interlaced scan lines. We will now look at such an image.

Start Matlab and add paths as described in the beginning of the Task section. Use `imread` to load the image `interlaced_image.png`. Load the image into the variable `im`, and make sure you convert it to `double` format using the function `im2double`. (This function also scales the image to the interval between 0 and 1.) Now look at the image using `imshow`. Use the zoom tool to examine details.

QUESTION 24: What is moving, and what is stationary in this image? What is the direction of the motion?

Next, generate two masks, one for each of the two interlaced fields:

```
[rows,cols,ndim]=size(im);
mask1=zeros(rows,cols);
mask1(1:2:end,:)=1;
mask2=zeros(rows,cols);
mask2(2:2:end,:)=1;
```

We can use these to mask out the individual fields in the image as follows:

```
imshow(im.*repmat(mask1,[1 1 3]));
imshow(im.*repmat(mask2,[1 1 3]));
```

QUESTION 25: Can you see two different motion positions in the images?

The two images we just created have every second line set to zero. You should now fill in the missing values using interpolation. You may use the convolution command `conv2` and the interpolation kernel: `f=[1 2 1]'/2;`.

QUESTION 26: Why is 2 (and not 4) a good normalization factor here?

Use the following code to create the two images `im1` and `im2`. Two of the lines are incomplete, and you are expected to add the proper calls to `conv2`.

```
im1 = zeros(size(im));
im2 = zeros(size(im));
for k=1:3,
    bk=im(:,:,k);
    im1(:,:,k)= ... insert your command here...
    im2(:,:,k)= ... insert your command here...
end
```

QUESTION 27: What are the missing commands?

QUESTION 28: Look at the result with `imshow`. Can you see any artifacts at the top and in the bottom of the image? Why are they there?

---

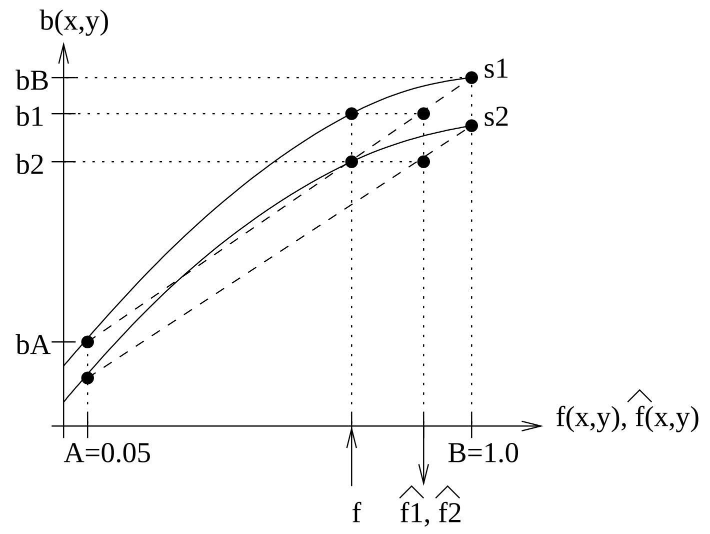
3 Appendix: Shading correction

The digital sensor image consists of a set (e.g. 512×512) parallel information channels, one for each pixel. In the radiometric chain: lighting $\Rightarrow$ optical system $\Rightarrow$ photo-detectors, there are individual differences for each channel. The differences can for example be due to vignetting or non-linearity in the photo-detectors. The sum (or rather the product) of all these effects gives the relationship

$$b(x,y) = s_{xy}[f(x,y)]$$

where $b(x,y)$ is the preliminary image value in pixel (x,y) and $f(x,y)$ is the signal value of the "real" image (brightness/reflectivity/transparency) in the pixel (x,y) and s_{xy} is the slightly nonlinear characteristics of the camera system in the pixel (x,y).

s_{xy} is called the *shading function* and the problem/phenomenon is called *shading.* Methods to estimate $f(x,y)$ from $b(x,y)$ are called *shading correction.* As seen from the above, each pixel has its own function s_{xy}. The figure below shows two curves s_1 and s_2 that may be representative for the shading s_{xy} in two different pixels in an image sensor.



In practice we can assume that s_{xy} is strictly monotonic and, consequently, it is in principle possible to determine the inverse function s_{xy}^{-1} and calculate f(x,y)=$s_{xy}^{-1}[b(x,y)]$. To simplify matters, we linearize s_{xy} (and s_{xy}^{-1}) such that

$$\hat{f}(x,y) = c(x,y)[b(x,y) + d(x,y)] \approx f(x,y) \qquad (1)$$

and the inverse mapping s_{xy}^{-1} is then determined once c, d are known. Notice that s_{xy} and therefore also c, d depend on (x,y). These parameters, in turn, can be determined as soon as the functions b and f are known for two different points on the curve s_{xy}. This is done independently for each pixel located at (x, y). This linearization is the first simplification.

Sometimes a second simplification is used based on the assumption that s_{xy} varies slowly over the image coordinates x and y (which is true for the optical variation but more doubtful for the other shading phenomena). Then we are using the same pair of coefficients (c, d) for a large number of closely located pixels. As an example, 64×64 coefficient pairs may be used for a 512×512 image, i.e. s is constant within an 8×8 pixel region.

Let us now see how a simple calibration can be done in terms of an example. For two different constant lighting conditions $f_A(x, y)$ and $f_B(x, y)$, we measure the response $b(x, y)$ in the sensor pixels 1 and 2. Suppose that the shading functions s_1 and s_2, look as in the figure. We now decide that the proper, corrected values for these two f-values are, for example, $\hat{f}_A = 0.05$ and $\hat{f}_B = 1.0$. This gives us a system of equations for each pixel (in the figure, only b_A and b_B for sensor 1 is marked).

$$\begin{cases} 0.05 & = c[b_A + d], \\ 1 & = c[b_B + d], \end{cases}$$

which gives

$$c = \frac{19/20}{b_B - b_A}, \quad d = \frac{b_B - 20b_A}{19}.$$

From the figure we can see that we have (very roughly) approximated the two shading functions with two straight lines. Now suppose that the same light intensity f hits the two sensor elements giving the different image values b_1 and b_2. Correction according to equation (1) means that we interpret this as two values $\hat{f}_1$ and $\hat{f}_2$, which are highly divergent from the correct value f because the shading functions are strongly nonlinear. However, we can tolerate this as long as the difference between $\hat{f}_1$ and $\hat{f}_2$ is small. (In the figure they actually coincide.) Therefore, the false contrast between b_1 and b_2, also called *fixed pattern noise*, is eliminated in the shading corrected image. That we succeed so well in this case is because the two functions s_1 and s_2 are uniform and equal to each other by means of an additive and a multiplicative factor. Fortunately, this often seems to be the case in the real image sensors.

References

[1] Abbas El Gamal and Helmy Eltoukhy. CMOS image sensors. *IEEE Circuits and Devices Magazine*, May/June 2005.