Questions may continue on the back. Type your text. You may write math formulas by hand. What we cannot read, we will not grade.

You may do this assignment alone or in pairs. If you work with one other person, hand in a single solution and put both names on it. No other collaboration or discussion is allowed for this assignment.

Hand in your solution as a single, stapled paper document at the beginning of class on the due date.

1. The images 1.jpg, 4.jpg, 5.jpg, 6.jpg are reproduced at a reduced scale in figure 1, and are available at full resolution with this assignment. They were taken under the same lighting conditions, and with four different combinations of camera gain and aperture: the two apertures were $f/22$ and $f/2.8$, and the two gains were ISO 1600 (high gain) and ISO 200 (low gain). For each picture, the camera automatically chose an exposure time that would expose the image well.

Hand in a table with the picture number and the corresponding gain and aperture. Do not hand in the pictures themselves. Explain your answer. [Hint: this question is hard to answer from the reproductions below. Look at the original images instead.]

2. The images 2.jpg and 7.jpg are reproduced in figure 2 at a reduced scale, and are available at full resolution with this assignment. They were taken with the same aperture setting $f/8$ under the same lighting conditions, and are approximately equally bright. Which one was taken with the shortest exposure time? Explain your reasoning.

3. To focus an object 1 meter away with a certain thin lens, the camera sensor must be placed 40 mm behind the lens. How far from the same lens must the sensor be to focus an object 50 cm away?

4. Image a.jpg in figure 3 was taken with a lens that has a much shorter focal length than was used for b.jpg.

(a) Explain qualitatively why proportions in one of the two images look more distorted than in the other. In other words, why do I look like a cave man in image a.jpg? [Hint 1: “You look like a cave man in both” is not an acceptable answer. Hint 2: Both
lenses used for these pictures have negligible distortion. Hint 3: Think about distances from the lens, and keep in mind that an object that is twice the distance from the camera looks half the size.]

(b) Which of the two lenses above has the wider field of view? Explain your reasoning in words and/or with a picture.

5. In a Bayer-mosaic sensor, half of the pixels are sensitive to green, a quarter to red, and a quarter to blue. The green pixels are arranged like the black cells on a checkerboard. The red pixels are in the white cells and in every other row and every other column. The blue pixels are in the remaining white cells. See http://en.wikipedia.org/wiki/Bayer_filter or the class notes on image formation.

If sensors had infinite extent, there would be only one Bayer mosaic pattern. Since sensors have boundaries, four types of patterns are possible, depending on how one starts in the top left corner of the sensor. These patterns are shown and named in figure 4.

For example, the pattern in figure 5 is an 8 x 8 Bayer mosaic of type ‘rggb’, as determined by the top left square in the black box.

Referring to the sample mosaic in figure 5, note that a blue pixel has four green neighbors arranged in a ‘+’ and four red neighbors arranged in an ‘×’. Similarly, a red pixel has four green neighbors arranged in a ‘+’ and four blue neighbors arranged in an ‘×’. Green pixels are different: Each green pixel has two red neighbors and two blue ones. In half of the rows (or columns), the red neighbors are horizontally adjacent (‘−’) and the blue ones are vertically adjacent (‘|’). In the remaining rows, the situation is reversed. So there are four types of neighborhoods: ‘×’, ‘+’, ‘−’, and ‘|’.

Green pixels are blind to red and blue, and so forth, so a Bayer mosaic image has only one third the information contained in a full RGB image of the same size, which has one value of R, G, B for each pixel. The missing values are determined by algorithms called demosaicing, the simplest of which is to average available neighboring pixel values of the required colors.

The MATLAB Image Processing Toolbox has a function demosaic that converts a Bayer mosaic into an RGB image. This function implements a more sophisticated algorithm by H. S. Malvar, L. He, and R. Cutler, “High quality linear interpolation for demosaicing of
Figure 4 The four types of Bayer pattern are named by the arrangement of pixels in the top left $2 \times 2$ square in the image.

Figure 5 An ‘rggb’ Bayer mosaic.

Bayer-patterned color images.” ICASSP, 2004. This function is implemented as a mex file in MATLAB, that is, it is written in a different programming language, compiled, and called from MATLAB.

The reason for this foreign-language implementation is most likely that the algorithm is implemented as a for loop over the output image: For each pixel, use the pixel coordinates to determine which two color values need to be reconstructed at that point, and which of its neighbors are where, and then do the reconstruction. Large for loops are hopelessly inefficient in MATLAB, hence the mex solution.

In this homework problem, we will implement demosaicing by averaging neighboring values and without looping explicitly over all the pixels in the image, using matrix operations instead. The following subproblems take you through the steps of writing a MATLAB function with the following header

```matlab
function C = bayer2rgb(I, sensorAlignment)
```

where $I$ is the input $m \times n$ image with the Bayer mosaic, $C$ is the corresponding $m \times n \times 3$ color image, and `sensorAlignment` is a four-character string out of the set {‘rggb’, ‘bggr’, ‘gbrg’, ‘grbg’} that specifies the type of Bayer pattern stored in $I$.

Our solution will have lower quality than demosaic because of the simpler algorithm and will perform more operations than strictly necessary, but will be reasonably efficient in native MATLAB because it uses no explicit for loops over the image.

Think of this problem both as a way to understand Bayer sensors and as an exercise in MATLAB-style programming for images.

(a) Keep it Simple Part I. Neighborhoods of pixels on image boundaries are different from those for interior pixels. For instance, in an ‘rggb’ Bayer mosaic image $I$, the top left pixel is red and has only two green neighbors instead of four and one blue neighbor instead of four, and so forth.

Handling all these cases separately and optimally would complicate the code. Instead, we pad the input image $I$ with a one-pixel wide rim of values all around so that all the pixels we care about in the output image have the same pattern of neighbors. If the size of $img$ before padding is $m \times n$, its size afterwards is $(m + 2) \times (n + 2)$.

If the pixel values used for the padding are chosen well, the effects of padding on the resulting color image $C$ will be visually negligible. Padding with zeros is not a good idea, because all pixels will become darker at image boundaries. What is a good and simple way to pad the image $I$, so as to avoid jarring visual artifacts?

Explain in a sentence or two what you do and why, and hand in the snippet of your code that performs the padding. Do not use any explicit for loops (or any equivalent constructs such as while, goto, and so forth).

WARNING: The input image $I$ is padded, but the output image $C$ is not.

(b) Keep it Simple Part II. As explained earlier, the type of Bayer pattern is encoded by the subarray $I(1:2, 1:2)$ of the input image, that is, by the top-left two-by-two set of pixels in $I$. After padding, the type of Bayer mosaic is encoded by the subarray $I(2:3, 2:3)$ instead.
Reasoning separately for each of the four types of Bayer patterns in figure 4 would complicate the code. Instead, write a MATLAB switch statement that examines the sensorAlignment argument and adds the appropriate number of rows and/or column to the top and/or left of the padded version of I in order to transform the given Bayer pattern into one of type ‘rggb’.

In this way, when you write the demosaicing code you can assume the ‘rggb’ pattern, and both your reasoning and your code become simpler. Of course, we need to remember to remove from the result any rows or columns we add to the input. Because we remove the additional entries at the end, it does not matter what values we use in the additional row and/or column at this stage. Just pad with zeros.

Hand in your code for the switch statement, including an error message in the otherwise clause of the statement. Add comments such as “% Add a row” to each case of the switch. If you do not know how switch works, consult the MATLAB help.

(c) Lest you forget, it is safest to also write right away the code that removes from the output image C any rows and/or columns corresponding to those you added to I. Hand in that snippet of code, with the same type of comments as in the previous snippet.

(d) As we saw earlier, there are four types of neighborhood patterns: ‘×’, ‘+’, ‘−’, and ‘|’. Each of them entails averaging different neighbors. In order to use matrix operations easily, we just compute all four averages everywhere, regardless of pixel color. At the end, we’ll pick the averages we need. This is wasteful when counting scalar arithmetic operations, but ends up being faster than doing the minimal number of operations, because we can use efficient matrix operations.

Hand in a code snippet that produces four arrays of size \(m \times n\) named \(X, P, H,\) and \(V\) that implements averaging for the four neighborhood patterns ‘×’, ‘+’, ‘−’, and ‘|’, respectively. The input is the \((m + 2) \times (n + 2)\) padded image \(I\) with the Bayer mosaic. Again, do not use any explicit for loops. As a hint, the first one is done for you:

\[
X = \left( \frac{I(1:(end-2), 1:(end-2)) + I(3:end, 1:(end-2)) + I(1:(end-2), 3:end) + I(3:end, 3:end)}{4} \right);
\]

If you do not see what is going on, work out a small example. You may be able to save operations if you compute \(P\) after \(H\) and \(V\) rather than before.

(e) We have done more work than necessary because we have computed \(4mn\) pixel values, while we only needed \(2mn\) (the missing colors). However, everything was very fast because of simple matrix operations. Now we need to pick the values we need and place them where they belong in the color output image \(C\), which has size \(m \times n \times 3\). We can do this as follows:

- Create an \(m \times n \times 3\) color output image \(C\) where each color band is a copy of \(I\):

  \[
  C = \text{cat}(3, I, I, I);
  \]

- Overwrite the appropriate pixels in each band with values from the interpolated images \(X, P, H,\) and \(V\).

Hand in the complete code of a MATLAB function \(\text{bayer2rgb}\) that converts the input Bayer-mosaic image \(I\) to an output color image \(C\) with the same number of rows and columns as \(I\) and with pixels of the same data type. Your function should be based on the following skeleton and the code you wrote in answer to the previous questions. Again, do not use any explicit for loops.

```matlab
function C = bayer2rgb(I, sensorAlignment)
if ~ismatrix(I)
    error('Input must be a 2D image');
end

% Remember input image pixel data type, and convert to double
type = class(I);
I = double(I);

%%% Your code here %%%

% Make the output image pixels have the same data type as in the input image
C = cast(C, type);
```

The argument `sensorAlignment` is one of ‘gbrg’, ‘grbg’, ‘bggr’, ‘rggb’ as described earlier and also in the MATLAB help page for demosaic. See your MATLAB help page or the following URL:

To help you with this part of the problem, here is how you build the red band. The left part of figure 6 shows the pixels of the red band that are already in their proper place for an input Bayer mosaic of type ’rggb’. The empty spaces need to be filled in from the average images you created earlier. Empty spaces that have an I pixel to their left and one to their right come from the H image, and so forth. The right part of figure 6 shows what image each of the missing pixels is supposed to come from.

![Figure 6](image)

Thanks to the switch statement and padding you developed earlier, you can safely assume that you are working with a Bayer mosaic of type ’rggb’. So the code for constructing the red band is as follows (recall that the pixels from I are already in place):

```matlab
C(1:2:end, 2:2:end, 1) = H(1:2:end, 2:2:end);
C(2:2:end, 1:2:end, 1) = V(2:2:end, 1:2:end);
C(2:2:end, 2:2:end, 1) = X(2:2:end, 2:2:end);
```

WARNING: It is easy to make mistakes in this exercise, so work through small examples with diagrams similar to the ones in figure 6 (which you need not turn in).

Hand in the complete code for your function bayer2rgb. Add brief comments to state what each block of instructions does.

(f) The function testBayer provided with this assignment creates (rather uninteresting) images of type double, both in Bayer mosaic form and as full RGB color images. The color bands are affine transformations of the row and column coordinates, so averaging should yield exact results. You can use images produced by this function to test your code.

WARNING: If you want to display these images, use imshow(uint8(bayer)) for pixel values up to 255 and imshow(uint16(bayer)) for images with higher values. Displaying the double images with imshow will give nonsense.

If you run, say

```matlab
pattern = 'bggr';
[bayer, rgb] = testBayer(4, 5, pattern);
color = bayer2rgb(bayer, pattern);
```

then the central 2 × 3 portion of rgb should be identical to that of color. That is, the result of the following instructions should be zero (or a most of the order of 10^{-13} if there are numerical inaccuracies):

```matlab
r = rgb(2:(end-1), 2:(end-1), :);
c = color(2:(end-1), 2:(end-1), :);
norm(r(:) - c(:))
```

The two images will differ somewhat on the margins, depending on exactly how you pad the input image to handle boundaries. If you have different results in the central part, there is a problem somewhere in your code. Iterate until you get it right.

Hand in the three 4 × 5 arrays resulting from the following commands:

```matlab
pattern = 'bggr';
[bayer, rgb] = testBayer(4, 5, pattern);
color = bayer2rgb(bayer, pattern);
round(color)
```

Also state the value of the norm commuted as shown above.

(g) Try your code on the Bayer mosaic mandi.tif that comes with the MATLAB Image Processing toolbox. To retrieve it, just say

```matlab
I = imread('mandi.tif');
```
In case you have trouble retrieving this image, it is also provided with the homework assignment. Do not hand in the image, just state if you see reasonable colors.

In addition, hand in the result of running the following code, which runs \texttt{bayer2rgb} ten times on this $2014 \times 3039$ mosaic to measure how long each run takes:

\begin{verbatim}
I = imread('mandi.tif');
tic; for k = 1:10, C = bayer2rgb(I, 'bggr'); end; time = toc/10
\end{verbatim}

Also state what CPU and operating system you ran your code on.
Sample Exam Questions

Answers to the questions below are not due as part of this homework assignment. They will not be graded, and no sample answers will be provided (you will find answers by reading the textbook and notes). They are here only as a way for you to test your own knowledge in computer vision, and to give you an idea of the type of questions that may come up in the exam.

- What is the main advantage of a small pinhole in a pinhole camera?
- What is the main drawback of a small pinhole in a pinhole camera?
- What is the focal distance $z_I$ for an object that is $z_O = 200$ mm away from a thin lens with a focal length of $f = 20$ mm? You may leave your answer as a formula of the form $z_I = \ldots$.
  WARNING: If you use equation (1.5) in the textbook, please note that the minus sign in that equation is balanced by a minus sign in $-Z$ in figure 1.8. The simplest way to avoid sign issues is to replace both minus signs with plus signs.
- Define focal length in terms of focal distance.
- Do you close or open the lens aperture to increase depth of field?
- How much more light does an $f/1.4$ lens capture compared to an $f/2$ lens?
- If a scene is well exposed with an aperture $f/8$ and a $1/120$ second exposure, what exposure do you need—everything else being equal—with an aperture $f/4$?
- What is a drawback of increasing camera gain?
- Explain briefly why a large pixel fill factor is good.
- Does a pixel count the number or the wavelength of incoming photons, or both?
- What fraction of the pixels in a Bayer pattern are green?
- What probability distribution best describes the statistics of photon counts in a pixel exposed to low light levels?
- Do you achieve lower distortion by using a lens on a smaller or bigger sensor than the lens was designed for?
- Which ISO setting gives you grainier (noisier) pictures: ISO 200 or ISO 1600?