Homework 3 Writeup

Bayer image to RGB image using bicubic interpolation

Processing Steps

Below describes how I processed given Bayer image to an RGB image.

1. From the given 1-channel Bayer image, make three 1-channel (Not interpolated) images each corresponding to R, G, B. Pixel corresponding to the Bayer channel has the same value of the Bayer image, and the other pixels has zero value.
2. For bicubic interpolation, R channel image and B channel image, apply `rb_kernel` , and to G channel image, apply `g_kernel`. Two kernels are described in the following paragraph.
3. Concat three single channel images to make a 3-channel RGB image.

Bicubic interpolation using kernel

(Extra credit)

I implemented interpolation using kernels consist of coefficients that has to be multiplied to neighbor pixels to get the target pixel value.

Consider a 7×7 region in the bayer image, denoted as a matrix below, and we are estimating value of p_{44}, which is the pixel at the center of the region.

$$\begin{bmatrix} p_{11} & \cdots & p_{17} \\ \vdots & \ddots & \vdots \\ p_{71} & \cdots & p_{77} \end{bmatrix}$$

1. Estimation of R channel.
   (a) Postion of p_{44} is R: Just have the value from the Bayer image.
   (b) Position of p_{44} is Gr: Used 1D cubic interpolation, which requires 4 neighbor points information. Estimate from p_{41}, p_{43}, p_{45}, p_{47}, which are the four R positions in the region and in the same horizontal line.
   (c) Position of p_{44} is Gb: Used 1D cubic interpolation. Estimate from p_{14}, p_{34}, p_{54}, p_{74}, which are the four R positions in the region and in the same vertical line.
   (d) Position of p_{44} is B: Used bicubic interpolation, which requires 8 neighbor points information. Estimate from $p_{11}, p_{13}, p_{15}, p_{17}, p_{31}, p_{33}, p_{35}, p_{37}, p_{51}, p_{53}, p_{55}, p_{57}, p_{71}, p_{73}, p_{75}, p_{77}$, which are the eight R positions in the region.
2. Estimation of G channel.
   (a) Postion of p_{44} is R: Used bicubic interpolation. Estimate from $p_{14}, p_{23}, p_{25}, p_{32}, p_{34}, p_{36}, p_{41}, p_{43}, p_{45}, p_{47}, p_{52}, p_{54}, p_{56}, p_{63}, p_{65}, p_{74}$, which are the eight G positions in the diamond region inscribed in the 7×7 region.

(b) Postion of p_{44} is G: Just have the value from the Bayer image.

(c) Postion of p_{44} is B: Same as when the position of p_{44} is R.

3. Estimation of B channel.

   Similar rules of estimation of R channel is applied.

Consider 1-2 case as example of 1D cubic interpolation case.
The cubic function can be expressed as below:

$$f(x) = \sum_{i=0}^{3} a_i x^i$$

$p_{41}, p_{43}, p_{45}, p_{47}$ correspond to $f(-1), f(0), f(1), f(2)$, and the target p_{44} corresponds to $f(1/2)$.
Using the below difference approximation,

$$f'(x) = (f(x+1) - f(x-1))/2$$

$p_{41}, p_{43}, p_{45}, p_{47}$ can be expressed by $f(0), f(1), f'(0), f(1)$.

And using the below relationship obtained by assigning 0, 1 to $f(x)$ and $f'(x)$,

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & 2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f(0) \\ f(1) \\ f'(0) \\ f'(1) \end{bmatrix}$$

a_0, a_1, a_2, a_3 can be expressed by $p_{41}, p_{43}, p_{45}, p_{47}$.
This gives the relationship

$$f(1/2) = \sum_{i=0}^{3} a_i (1/2)^i = (-16/256)p_{41} + (81/256)p_{43} + (81/256)p_{45} + (-16/256)p_{47}$$

Similarly, consider 1-d as example of bicubic interpolation case.
The bicubic function can be expressed as below:

$$f(x) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$$

$p_{11}, p_{13}, ...p_{77}$ correspond to $f(-1,-1), f(-1,0), ...f(2,2)$, and the target p_{44} corresponds to $f(1/2, 1/2)$.
Using the below difference approximations,

$$f_x(x,y) = (f(x+1,y) - f(x-1,y))/2$$

$$f_y(x,y) = (f(x,y+1) - f(x,y-1))/2$$

$$f_x y(x,y) = (f(x+1,y+1) - f(x-1,y-1) + f(x+1,y-1) - f(x-1,y+1))/4$$

$p_{11}, p_{13}, ...p_{77}$ can be expressed by $f(0,0)$, $f(1,0)$, $f(0,1)$, $f(1,1)$, $f_x(0,0)$, ..., $f_x(1,1)$, $f_y(0,0)$, ..., $f_y(1,1)$, $f_xy(0,0)$, ..., $f_xy(1,1)$.

And using the below relationship obtained by assigning 0, 1 to $f(x,y)$, $f_x(x,y)$, $f_y(x,y)$, $f_xy(x,y)$,

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & 2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & 2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

$a_{00}, ..., a_{33}$ can be expressed by $p_{11}, p_{13}, ...p_{77}$.
This gives the relationship

$$f(1/2, 1/2) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} (1/2)^(i + j)$$

$$= \begin{bmatrix} 1/256 & -9/256 & -9/256 & 1/256 \\ -9/256 & 81/256 & 81/256 & -9/256 \\ -9/256 & 81/256 & 81/256 & -9/256 \\ 1/256 & -9/256 & -9/256 & 1/256 \end{bmatrix} \circ \begin{bmatrix} p_{11} & p_{13} & p_{15} & p_{17} \\ p_{31} & p_{33} & p_{35} & p_{37} \\ p_{51} & p_{53} & p_{55} & p_{57} \\ p_{71} & p_{73} & p_{75} & p_{77} \end{bmatrix}$$

By obtaining the estimated pixel value using p_{11}, ..., p_{77} applying the rules described above as shown in two examples, kernels can be defined like below. Note that for the `g_kernel`, the weight values are rotated 45 degrees, since for this case p_{43} corresponds to $f(0,0)$, p_{34} corresponds to $f(1,0)$ and so on.

- `rb_kernel`: Used for R channel and B channel.

$$(1/256) \begin{bmatrix} 1 & 0 & -9 & -16 & -9 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ -16 & 0 & 144 & 256 & 144 & 0 & -16 \\ -9 & 0 & 81 & 144 & 81 & 0 & -9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -9 & -16 & -9 & 0 & 1 \end{bmatrix}$$

- `g_kernel`: Used for G channel.

$$(1/256) \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -9 & 0 & -9 & 0 & 0 \\ 0 & -9 & 0 & 81 & 0 & -9 & 0 \\ 1 & 0 & 81 & 256 & 81 & 0 & 1 \\ 0 & -9 & 0 & 81 & 0 & -9 & 0 \\ 0 & 0 & -9 & 0 & -9 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Calculation of disparity map

Cost aggregation approach 1. Box filtering
After calculating the initial cost volume, my first approach for cost aggregation was box filtering. However, the result was not really successful, especially for the parts where disparity is discontinuous.

Cost aggregation approach 2. Gaussian filtering
My next approach was to apply Gaussian filter to the cost volume. I set the stddev of Gaussian function as half of the window size. Since the weight of the Gaussian filter is the largest at a given pixel position, accuracy for disparity discontinuous region was improved.

Cost aggregation approach 3. Gaussian filtering and Various size window
(Extra credit)
This is the approach I used for in the submitted code.
[NOTE] I changed the function `calculate_disparity_map`'s template - I erased the `window_size` parameter, since in my approach I used various sizes of windows to calculate disparity map.

In the previous approaches, I tried various sizes of windows.
When I tried small size window, accuracy for the disparity discontinuous regions was good, but in the continuous regions the result was very vulnerable to noise.
When I tried a larger size windows, it wasn't vulnerable to noise, but it was bad at accurately calculating the disparity in the discontinuous regions.

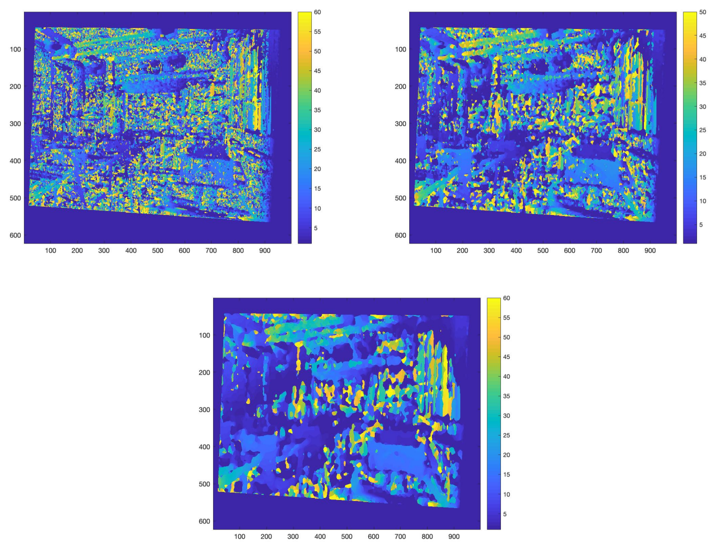

Figure 1: [1] 3×3 window, stddev 1.5 Gaussian filtered. [2] 5×5 window, stddev 2.5 Gaussian filtered. [3] 9×9 window, stddev 4.5 Gaussian filtered.

In order to resolve this trade-off, I calculated cost volumes varying window sizes, apply each cost volumes with Gaussian filter corresponding to each window sizes, and then accumulated them in a single cost volume giving various weights. After some trials, I decided that using 3×3 window and 9×9 window with weight $(0.5, 0.5)$ gives good result compared to the caculation time it requires. The result was less vulnerable to noise in the continuous regions, and shows the discontinuous regions more accurately.

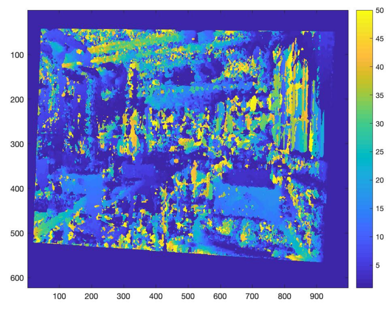

Figure 2: Combination of 3, 9 size windows.

Since this approach requires calculations of three cost volumes, it took quite a long time. I used below approach to reduce the required time as much as possible.
To calculate a cost volume of given `img` with `window_size`, I first constructed two matrices - `dev_img` of size `(img_h, img_w, window_size*window_size)`, and `dev_norm` of size `(img_h, img_w)`. Then, I slided the average window on the `img`, pre-calculated the deviation for the window-covered area, and stored the deviation vector to the corresponding position of `dev_img`, and stored the norm of the deviation vector in the corresponding position of `dev_norm`.
Below code shows the process.

```
function [dev_img, dev_norms] = calculate_dev(img,
   window_size)
    [img_h, img_w] = size(img);
    img_c = window_size * window_size;

    dev_img = zeros(img_h, img_w, img_c);
    dev_norms = zeros(img_h, img_w);

    avg_filter = fspecial('average', window_size);
    mean_img = imfilter(img, avg_filter, 'symmetric');

    window_rad = (window_size - 1) / 2;
    padded_img = padarray(...
        img, [window_rad, window_rad], 'both',...
        'symmetric');

    for i=1:img_h
        for j=1:img_w
```

```
            dev_vec =...
                reshape(padded_img(i:i+window_size-1,...
                j:j+window_size-1), [1, img_c])...
                - mean_img(i, j);
            dev_img(i, j, :) = dev_vec;
            dev_norms(i, j) = norm(dev_vec);
        end
    end
end
```

After doing the pre-calculation like below,

```
[dev_img_l, dev_norms_l] =...
    calculate_dev(img_left, window_size);
[dev_img_r, dev_norms_r] =...
    calculate_dev(img_right, window_size);
```

the cost volume can be earned by just calculating like below.

```
for k = 1 : max_disparity
    for i = 1 : img_h
        for j = 1 : img_w - k
            cost_vol(i, j, k) = ...
                dot(dev_img_l(i, j+k, :), ...
                    dev_img_r(i, j, :))...
                / (dev_norms_l(i, j+k) * ...
                    dev_norms_r(i, j));
            end
        end
    end
end
```

Additional notes

- As mentioned above, I changed the function template (erased a parameter) of `calculate_disparity_map()`.
- In functions `rectify_stereo_images()` and `calculate_disparity_map()`, I defined local function(s), so there are some codes(function definitions) added outside of the block commented as `Your code here`.
- For the positions corresponding to black margin region of the rectified images, I set the cost volume value of disparity layer 1 to a value larger then any value that can be caculated as a result of NCC, so that the region will result in disparity 1 (`cost_volume(:,:,1)`) in the disparity map. This was to avoid unexpected effect at the black margin region that can be caused by large kernels.