Berkay Mengunogul 150119934

CSE 4084 Multimedia Systems Homework 2 Report

Problem 1

Problem 1.2:

Load the frame with the file name “frame1.jpg” into a 288 x 352 MATLAB array using function “imread”, and then convert the array type from 8-bit integer to real number using function “double” or “cast” (note that the range of intensity values after conversion is between 0 and 255.) Denote by I_1 the converted MATLAB array. Repeat this step for the frame with the file name “frame2.jpg” and denote the resulting MATLAB array by I_2. In this problem, I_2 corresponds to the current frame, and I_1 corresponds to the previous frame (i.e. the reference frame).

Solution 1.2:

```
% Problem 1.2
i1 = double(imread("HW2/frame1.jpg"))
i2 = double(imread("HW2/frame2.jpg"))
```

Code 1: Images are read.

i1 = 288×352

0	28	0	16	12	...
104	173	167	180	178	
124	228	237	240	235	
96	208	215	208	202	
94	202	202	200	202	
82	189	190	197	210	
74	187	183	184	201	
81	198	186	172	186	
67	195	208	203	196	
73	195	203	201	203	
⋮					

Figure 1: Array representation of i1

i2 = 288×352

0	22	1	17	6	...
74	161	174	185	169	
76	212	241	240	230	
55	194	206	198	205	
62	190	191	187	205	
47	177	187	192	207	
40	171	187	188	195	
56	180	188	179	181	
57	189	202	195	190	
57	181	192	197	209	
⋮					

Figure 2: Array representation of i2

Images are read and converted into a double as it is given in the problem description. Input images are read with imread function and converted into double with double function as shown in Code 1.

Problem 1.3:

Consider the 32 x 32 target block in I_2 that has its upper-left corner at (65,81) and lower-right corner at (96,112). Note this is MATLAB coordinate convention, i.e., the first number between the parenthesis is the row index extending from 1 to 288 and the second number is the column index extending from 1 to 352. The target block is therefore a 32 x 32 sub-array of I_2.

Solution 1.3:

```
% Problem 1.3
target_block = i2(65:96, 81:112)
```

Code 2: Bounded area is selected.

In this part, the searched area is selected and annotated as target_block.

Problem 1.4:

Denote the target block by B_{target}. Motion estimation via block matching searches for the 32 x 32 sub-array of I_1 that is "most similar" to B_{target}. Recall that in the lectures we have introduced various forms of matching criteria e.g. correlation coefficient, mean-squared error (MSE), mean-absolute-error (MAE), etc.

In this problem, we use MAE as the matching criterion. Given two blocks B_1 and B_2 both of size M x N, the MAE is defined as

$$MAE(B_1, B_2) = \frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} |B_1(i,j) - B_2(i,j)|$$

To find the block in I_1 that is most similar to B_{target} in the MAE sense, you will need tos can through all the 32 x 32 blocks in I_1, compute the MAE between each of these blocks and B_{target}, and find the one that yields the smallest of MAE.

Note that in practice motion search is only performed over a region of the reference frame, but for the sake of simplicity, we perform motion search over the entire reference frame I_1, in this part. Determine the coordinates of the upper-left corner of the matched block in MATLAB convention. What is the backward motion vector for the center pixel of B_{target}?

Solution 1.4:

```
% Problem 1.4 results
[x , y, mae_val] = mae(target_block, i1)
oflow = opticalFlow(i1(65+16, 81+16), i2(65+16, 81+16))
plot(oflow)
```

Code 3: Finding the best matching area

In this section, I have implemented a Mean-Absolute-Error (MAE) function which can be seen in Code 4. that returns the starting coordinates and minimum MAE value of the best matching 32x32 area. This function gets the searched area as b_target, a frame that is going to be used for searching the area as b1 for its parameters. With two nested for loops, it goes through every pixel of the 2D image. Since the target block is 32x32 pixels, 31 is subtracted from both height and width. The reason is not going outside of the frames while executing the search. With the given formula MAE is calculated and it is kept at result variable. The result variable is declared as -1 for the initial case. At the first iteration, the MAE value is kept at result. It is checked with if condition. While executing the rest of the iteration, MAE values are going to be compared. The current value is saved at temp_mae. If temp_mae is has a smaller value comparing the result, result will be changed as the value and coordinates which are represented as i and j will be recorded as x and y.

Problem 1.4 MAE function

```
function [x, y, result] = mae(b_target, b2)
    [b2_heigth,i1_width] = size(b2);
    result = -1;
    for i = 1:b2_heigth-31
        for j = 1:i1_width-31
            temp_mae = sum(abs(b_target - b2(i:i+31, j:j+31)), "all")/(32*32);
            if result == -1
                result = temp_mae;
                x = i;
                y = j;
            elseif temp_mae < result
                result = temp_mae;
                x = i;
                y = j;
            end
        end

    end
end
```

Code 4: Find the Best Matching Area with Mean-Absolute-Value Function

For finding the backward motion vector of center pixel of target block, opticalFlow function is used. Since only center pixel needed to be calculated, 16 is added to the upper left corner of the target block and matched area. It is results are shown in Figure 3, and it is plotted in Figure 4.

```
oflow =
  opticalFlow with properties:

             Vx: 24
             Vy: 16
    Orientation: 0.5880
      Magnitude: 28.8444
```

Figure 3: Results of Backward Motion Vector

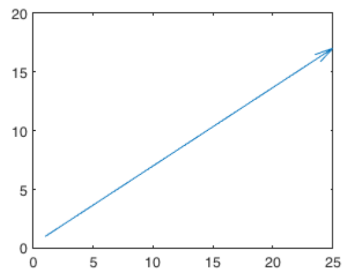

Figure 4: Plotting of Backward Motion Vector

Problem 1.5:

What is the corresponding MAE value (up to two decimal points)?

Solution 1.5:

With the function shown in Code 4, the founded result is “22,99”.

Problem 1.6:

Find and show the frame difference (DFD = $I_2 - I_1$). Estimate the backward motion vector of each block in frame I_2 by using 16 x 16 blocks and searching for the best matching block in I_1. Use a search region of size 64x64. Plot the motion vectors on frame I_2 using the MATLAB function **quiver**().

Predict frame 2, i.e., $\hat{I}_2$ by motion compensating the blocks in frame I_1. Find and show the dispaced frame difference (DFD = $I_2 - \hat{I}_2$) (similar to the images in slide 57 of motion estimation lecture notes). Compute the PSNR value between the original image I_2 (converted to type 'double') and the predicted image $\hat{I}_2$.

Solution 1.6:

```
% Problem 1.6
DFD = (im2double(imread("frame2.jpg")) - im2double(imread("frame1.jpg")))
imshow(DFD)
```

Code 5:Application of DFD

First, normalized versions of the frame1 and frame2 are read with the im2double function. Then frame1 is subtracted from frame2 as shown in Code 5 to find the DFD. Then the result is displayed at Figure 5.

Figure 5: Result of DFD

Problem 1.7:

Repeat step (6) using logarithmic search and compare your results. What are the advantages/disadvantages of using logarithmic search?

Solution 1.7:

The biggest advantage of the logarithmic search, it does not need computation power as much as the exhaustive search. Since it uses less computation power it is responding faster. It may not be obvious in this problem, but effects will be greatly change in higher resolutions.

Problem 1.8:

Discuss the advantages and disadvantages of the block-matching based motion estimation method.

Solution 1.8:

Block-matching based motion estimation is pretty useful when the environment has stable conditions. It is performed very easily but it has shortcomings such objects need to be rigid, lightning conditions must remain the same during the process. It cannot handle complex scenarios. Therefore, it is not useful in a dynamic environment.

Problem 2

Problem 2.1:

Download the noisy image. Load the noisy image into a MATLAB array and convert the type of the array from 8-bit integer 'uint8' to real number 'double'. Visualize the noisy image using the built-in MATLAB function "imshow". The function "imshow" takes as its argument either [0,255] for an 8-bit integer array (i.e. of type 'uint8'), or [0-1] for a normalized real-valued array (i.e. of type 'double'). To provide imshow with the correct argument, you would need either "cast" your real-valued array into "uint8", or normalize it by 255.

Solution 2.1:

```
noisy_img = im2double(imread("noisy.jpg"))
imshow("noisy.jpg")
```

Code 6: Loading the Noisy Image

```
noisy_img = 240x320
    0.1176         0    0.1059    0.1176  ...
    0.9608    0.1294    0.0588    0.0353
    0.1020    0.1059    0.1373    0.0863
    0.0745    0.0549    0.0431    0.1176
    0.0824    0.0863    0.0706    0.1333
    0.0549    0.0863    0.0235    0.0431
    0.0863    0.1176    0.0980    0.0510
    0.0549    0.0549    0.0863    0.0745
    0.0784    0.0627    0.1098    0.0902
    0.1412    0.0980    0.0627    0.0314
      ⋮
```

Figure 6: Array Representation of Noisy Image

Figure 7: Noisy Image

As it is shown in Code 6, the image is read and normalized with the im2double function. The result is shown in Figure 6 and the image is displayed in Figure 7.

Problem 2.2:

Perform 3x3 median filtering using the built-in MATLAB function "medfilt2". For this problem, the only argument you need to provide "medfilt2" with is the array you have created in step (1). Visualize the filtered image using "imshow". Remember to either cast the result to 'uint8' or normalize it before feeding it to "imshow".

Solution 2.2:

```
first_pass = medfilt2(noisy_img)
imshow(im2uint8(med_filtered_img))
```

Code 7: Applying Median Filter.

```
first_pass = 240×320
       0    0.0588    0.0353    0.0588  ...
  0.1020    0.1059    0.1059    0.1059
  0.0745    0.1020    0.0863    0.0667
  0.0745    0.0824    0.0863    0.0706
  0.0549    0.0706    0.0706    0.0510
  0.0824    0.0863    0.0863    0.0706
  0.0549    0.0863    0.0745    0.0863
  0.0549    0.0863    0.0863    0.0902
  0.0549    0.0784    0.0745    0.0863
  0.0627    0.0784    0.0627    0.0627
     ⋮
```

Figure 8: Array Representation of the Resulting Image

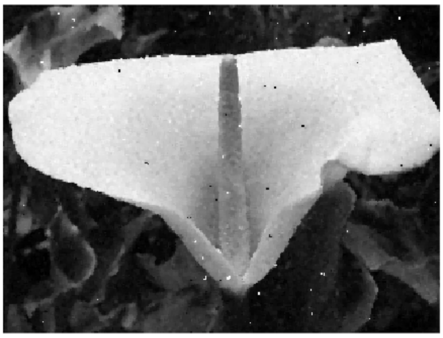

Figure 9: Resulting Image after First-Pass

With the given directions in the problem statement, the median filter is applied with medfilt2 function to the noisy image as shown in Code 7. Results are saved to first_pass variable and it is array form is presented in Figure 8. It is displayed with the imshow function in Figure 9.

Problem 2.3:

Perform a second-pass median filtering on the filtered image that you have obtained from step (2). Visualize the two-pass filtered image. Compare it with the noisy input image and the 1-pass filtered image.

Solution 2.3:

```
second_pass = medfilt2(med_filtered_img)
imshow(im2uint8(second_pass))
```

Code 8: Applying Median Filter for second time

```
second_pass = 240×320
       0    0.0353    0.0588    0.0353  ...
  0.0588    0.0863    0.0863    0.0863
  0.0745    0.0863    0.0863    0.0902
  0.0706    0.0745    0.0706    0.0863
  0.0706    0.0824    0.0706    0.0863
  0.0549    0.0745    0.0745    0.0745
  0.0549    0.0863    0.0863    0.0863
  0.0549    0.0745    0.0863    0.0745
  0.0549    0.0745    0.0784    0.0745
  0.0549    0.0627    0.0706    0.0627
     ⋮
```

Figure 10: Array Representation of the Resulting Image after Second-Pass

Figure 11: Resulting Image after Second-Pass

the median filter is applied with medfilt2 function to the first_pass as shown in Code 8. Results are saved to the second_pass variable and it is array form is presented in Figure 10. It is displayed with the imshow function in Figure 11.

Problem 2.4:

Download the noise-free image. Compute the PSNR value between the noise-free image and the noisy input image (up to two decimal points).

Solution 2.4:

```
noise_free = im2double(imread("original.jpg"))
original_psnr = num2str(psnr(noisy_img, noise_free), '%05.2f')
```

Code 9: PSNR calculation of noisy image

With the given directions, the noise-free image is read and normalized. PSNR value of the noisy image calculated with respect to the noise-free image as shown in Code 9. It is resulted as “11.33”.

Problem 2.5:

Calculate the PSNR value between the noise-free image and the 1-pass filtering output (up to two decimal points).

Solution 2.5:

```
first_pass_psnr = num2str(psnr(first_pass, noise_free), '%05.2f')
```

Code 10: PSNR calculation of First-Pass

PSNR value of the noisy image calculated with respect to the results of the first-pass as shown in Code 10. It is resulted as “27.38”.

Problem 2.6:

Calculate the PSNR value between the noise-free image and the 2-pass filtering output (up to two decimal points).

Solution 2.6:

```
second_pass_psnr = num2str(psnr(second_pass, noise_free), '%05.2f')
```

Code 11: PSNR calculation of Second-Pass

PSNR value of the noisy image calculated with respect to the results of the second-pass as shown in Code 11. It is resulted as “29.65”.