

CMPSCI 370 Homework 01

Dongwon Park

TOTAL POINTS

74 / 103

QUESTION 1

Matrix manipulation 8 pts

1.1 zero array 1 / 1

✓ - 0 pts Correct

1.2 uniformly random array 1 / 1

✓ - 0 pts Correct

1.3 norm 2 / 2

✓ - 0 pts Correct

1.4 dot product, angle and distance 3 / 3

✓ - 0 pts Correct

1.5 reshape 1 / 1

✓ - 0 pts Correct

QUESTION 2

Image formation 10 pts

2.1 pinhole camera 4 / 4

✓ - 0 pts Correct

2.2 size of the object 2 / 2

✓ - 0 pts Correct

2.3 distance 2 / 2

✓ - 0 pts Correct

2.4 time to capture same amount of light 2 /

2

✓ - 0 pts Correct

QUESTION 3

Aligning Prokudin-Gorskii images 25 pts

3.1 outputs of evalAlignment on the toy images 3 / 3

✓ - 0 pts Correct

3.2 outputs of real examples 16 / 18

✓ - 2 pts One Incorrect Shift

3.3 code for alignChannels 4 / 4

✓ - 0 pts Correct

QUESTION 4

Demosaicing 35 pts

4.1 code for mosaicImage 8 / 10

✓ - 2 pts The order of RGB is wrong

4.2 output and code for demosaic images

20 / 20

✓ - 0 pts Correct

4.3 plots 5 / 5

✓ - 0 pts Correct

QUESTION 5

5 Extra credit 0 / 25

✓ + 0 pts no extra credit

+ 10 pts coarse-to-fine-alignment

+ 5 pts handling boundaries

+ 0 pts cropping boundaries is mentioned but the results show the boundary artifacts

+ 8 pts linear interpolation but no results showed

+ 10 pts bilinear interpolation

+ 5 pts I can see that you handled the boundary for alignchannel in the results and the code. However, make sure you mention that you did it and how you

did it in the report for the future homework to get extra credit.

Homework 1 solution template

CMPSCI 370 Spring 2019, UMass Amherst

Name: DongWon Park

Here is a template that your solutions should roughly follow. Include outputs as figures, and code should be included in the end.

1 Matrix manipulation

1. Create a $m \times n$ array of all zeros.

```
import numpy as np  
array = np.zeros([m,n])
```

2. Create a random $m \times n$ array.

```
array = np.random.uniform(0, 1, [m, n])
```

3. Code to compute its length (or norm).

```
length = np.sqrt(np.sum(np.array(v)**2))
```

4. Given variables u and v representing arrays of size $n \times 1$, write down code to compute their
 - (a) dot product

```
dot = np.sum(np.multiply(u, v))
```

(b) angle

```
angle = np.degrees( np.arccos( np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v)) ) )
```

(c) distance

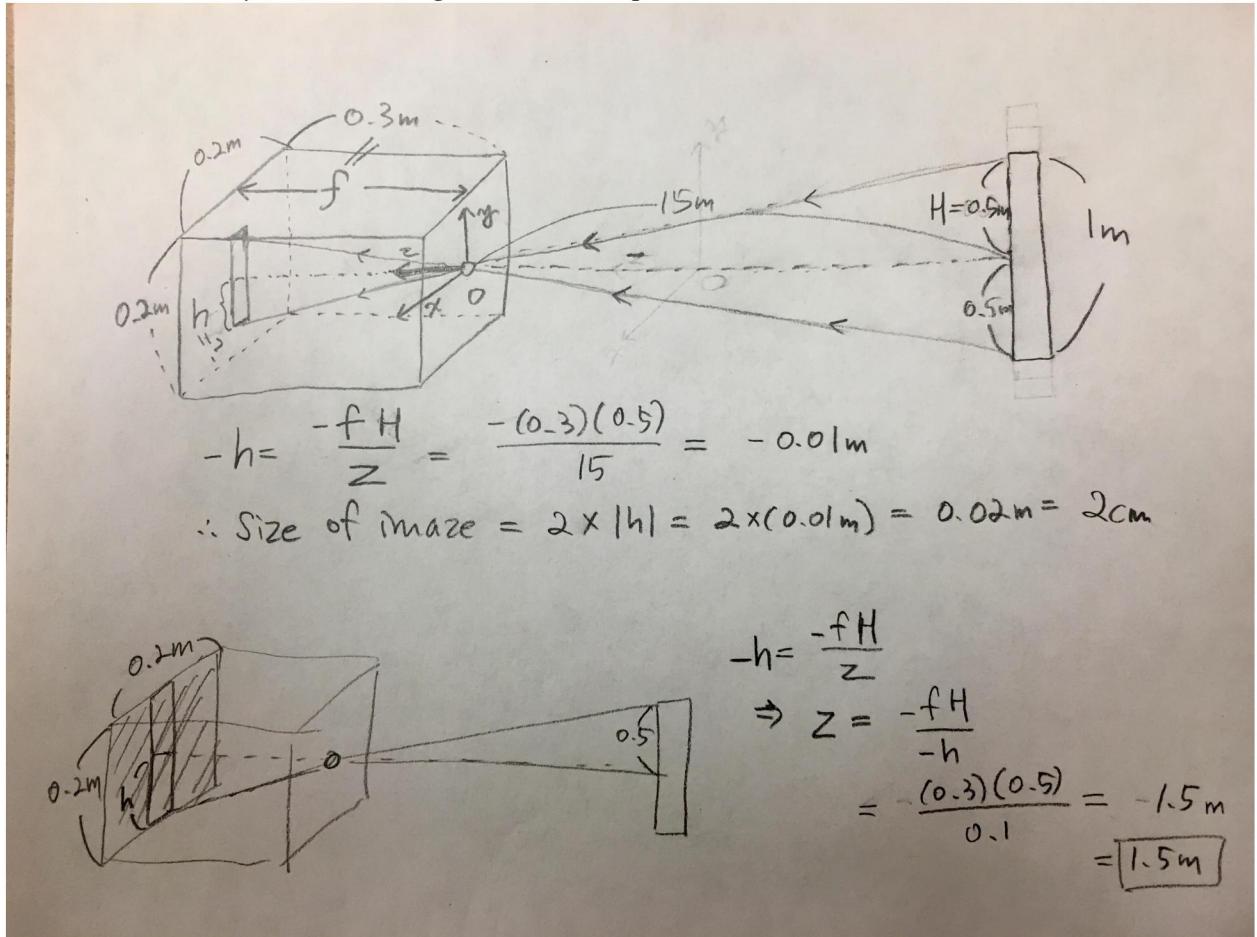
```
distance = np.sqrt( np.sum( (np.array(u) - np.array(v))**2 ) )
```

5. Given an array $a \in \mathbb{R}^{m \times n}$ write code to reshape it to a vector of size $nm \times 1$.

```
reshaped = np.ndarray.flatten(a, 'C')
```

2 Image formation

1. Illustration of the object and the image formed in the pinhole camera.



2. Calculations for the size of the object.
2cm. (Explanation in the picture)

3. Calculations for the distance.

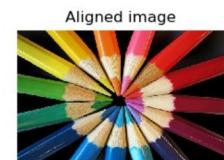
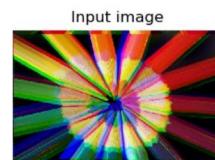
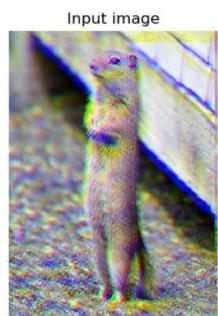
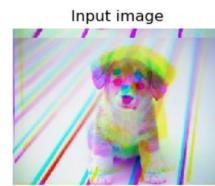
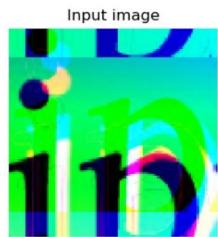
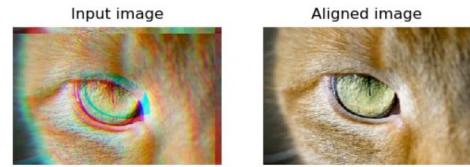
1.5m (Explanation in the picture)

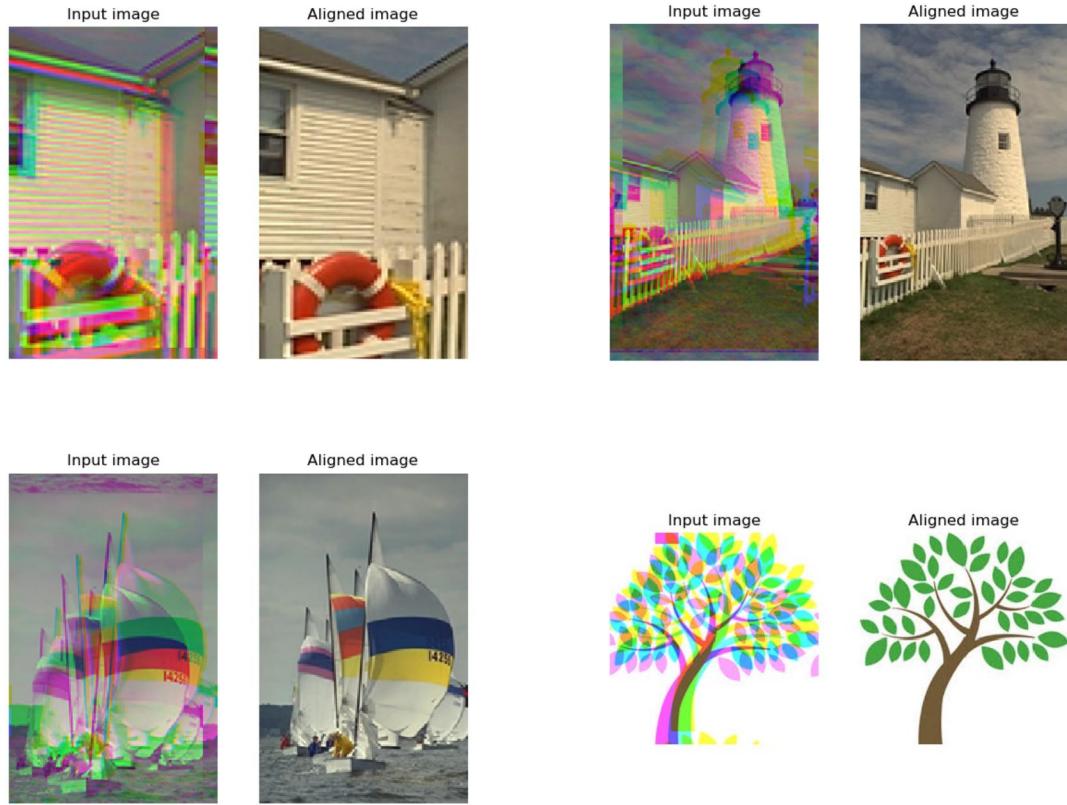
4. Time taken for the camera with a lens.

0.1ms Because when the radius gets bigger by 10 times, because the whole area of the circle will be squared, which is 100 times, so it can absorb the same amount of light 100 times faster. So 10 ms become 0.1ms

3 Aligning Prokudin-Gorskii images

1. Outputs of `evalAlignment` on the toy images.





2. Output of `alignChannels.py` that shows the computed shifts as seen below.

```
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/win10_workspace/CS370/Hw01/code$ python
Evaluating alignment...
balloon.jpeg
  gt shift: (7, 11) (-8, -8)
  pred shift: (-7, -11) (8, 8)
cat.jpg
  gt shift: (10, -9) (11, -14)
  pred shift: (-10, 9) (-11, 14)
ip.jpg
  gt shift: (14, -4) (-12, 0)
  pred shift: (-14, 4) (12, 0)
puppy.jpg
  gt shift: (15, 2) (-4, 14)
  pred shift: (-15, -2) (4, -14)
squirrel.jpg
  gt shift: (-2, -3) (11, 1)
  pred shift: (2, 3) (-11, -1)
pencils.jpg
  gt shift: (12, 3) (-10, -13)
  pred shift: (-12, -3) (10, 13)
house.png
  gt shift: (-8, -9) (0, -13)
  pred shift: (8, 9) (0, 13)
```

```
light.png
gt shift: (-6, 9) (-8, -10)
pred shift: (6, -9) (8, 10)
sails.png
gt shift: (14, -10) (0, 2)
pred shift: (-14, 10) (0, -2)
tree.jpeg
gt shift: (12, -14) (-14, 3)
pred shift: (-12, 14) (14, -3)
```

```
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/w
y
Evaluating alignment...
balloon.jpeg
    gt shift: (7, 11) (-8, -8)
    pred shift: (-7, -11) (8, 8)
cat.jpg
    gt shift: (10, -9) (11, -14)
    pred shift: (-10, 9) (-11, 14)
ip.jpg
    gt shift: (14, -4) (-12, 0)
    pred shift: (-14, 4) (12, 0)
puppy.jpg
    gt shift: (15, 2) (-4, 14)
    pred shift: (-15, -2) (4, -14)
squirrel.jpg
    gt shift: (-2, -3) (11, 1)
    pred shift: (2, 3) (-11, -1)
pencils.jpg
    gt shift: (12, 3) (-10, -13)
    pred shift: (-12, -3) (10, 13)
house.png
    gt shift: (-8, -9) (0, -13)
    pred shift: (8, 9) (0, 13)
light.png
    gt shift: (-6, 9) (-8, -10)
    pred shift: (6, -9) (8, 10)
sails.png
    gt shift: (14, -10) (0, 2)
    pred shift: (-14, 10) (0, -2)
tree.jpeg
    gt shift: (12, -14) (-14, 3)
    pred shift: (-12, 14) (14, -3)
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/w
```

3. A figure that shows all the aligned color images. Only include the images from the Prokudin-Gorskii dataset in the original resolution. For example, do not take low-resolution screenshots of the outputs;

Instead save them using appropriate commands in Matlab and Python.



Figure 1: Aligned color images.

Output of `alignProkudin.py`

```
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/win10_workspace/CS370/Hw01/code$ python alignProkudin.py
Directory ../output/prokudin-gorskii already exists.
00125v.jpg
shift: G (-4, 1)  B (-9, -1)
00153v.jpg
shift: G (-7, -2)  B (-14, 0)
00398v.jpg
shift: G (-6, -1)  B (-11, -4)
00149v.jpg
shift: G (-5, 0)  B (-9, -2)
00351v.jpg
shift: G (-9, -1)  B (-13, -1)
01112v.jpg
shift: G (-5, -1)  B (-5, -1)
```

4 Color image demosaicing

1. Errors of the nearest neighbor interpolation algorithm.

```
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/win10_workspace/CS370/Hw01/code$ python
Directory ../output/demosaic already exists.
-----
#   image    baseline   nn
-----
0  balloon.jpeg  0.225312  0.018164
1  cat.jpg      0.143735  0.021648
2  ip.jpg       0.279488  0.025186
3  puppy.jpg    0.102673  0.013631
4  squirrel.jpg 0.129523  0.038034
5  pencils.jpg  0.222792  0.019367
6  house.png    0.150520  0.026477
7  light.png    0.116505  0.026639
8  sails.png   0.087650  0.020434
9  tree.jpeg    0.180790  0.023829
-----
average      0.163899  0.023341
-----
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/win10_work
.bk
Directory ../output/demosaic already exists.
-----
#           image          baseline        nn
-----
0         balloon.jpeg  0.225312  0.018164
1         cat.jpg      0.143735  0.021648
2         ip.jpg       0.279488  0.025186
3         puppy.jpg    0.102673  0.013631
4         squirrel.jpg 0.129523  0.038034
5         pencils.jpg  0.222792  0.019367
6         house.png    0.150520  0.026477
7         light.png    0.116505  0.026639
8         sails.png   0.087650  0.020434
9         tree.jpeg    0.180790  0.023829
-----
average      0.163899  0.023341
-----
(py2) dpark@dpark-15Z960-A-AA52U1:~/workspace/win10_work
```

2. A figure (e.g., Figure 2) that shows the images obtained after interpolation.

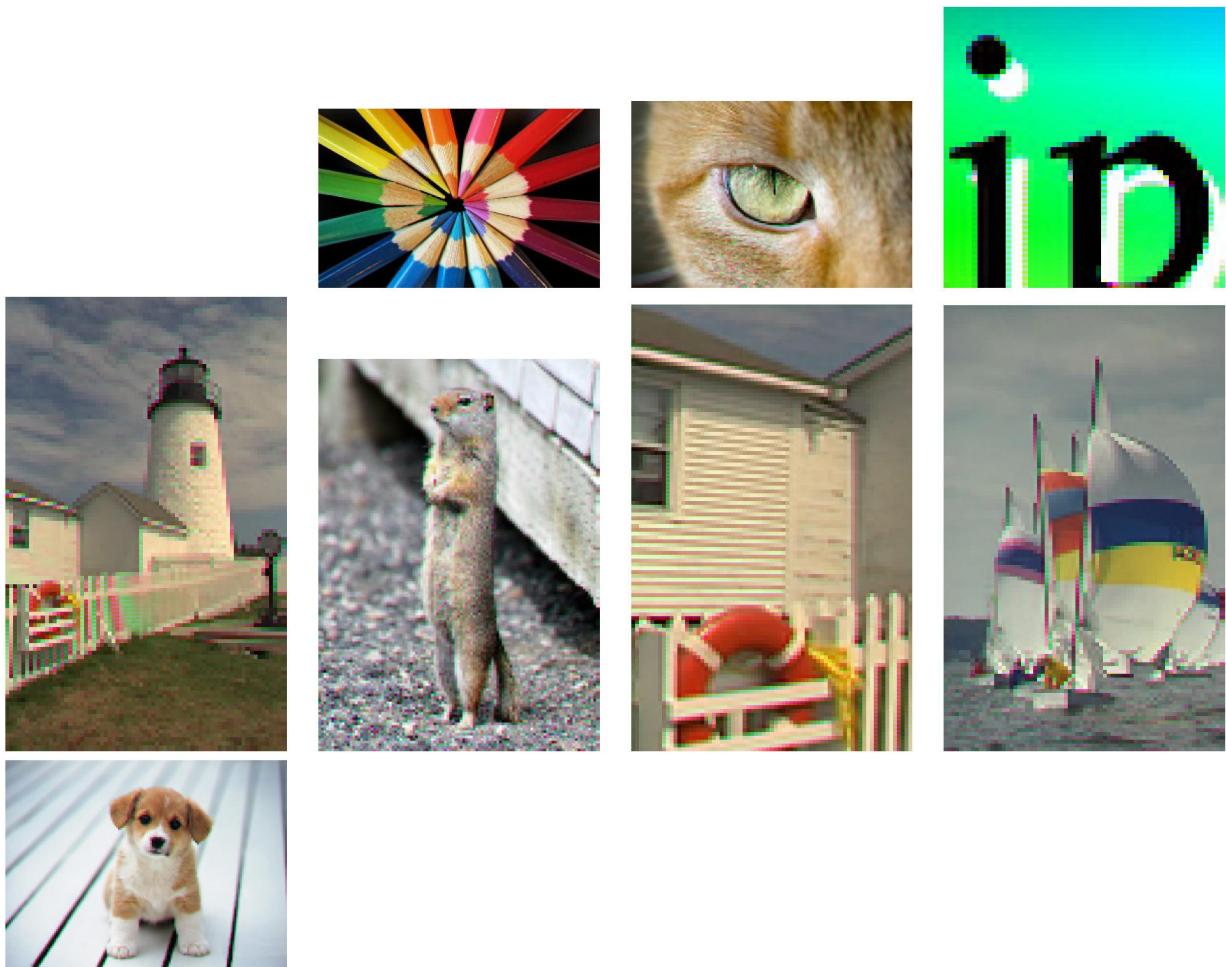


Figure 2: Demosaiced images using nearest neighbor interpolation.

I don't know why but nn-dmsc images for Balloon and Tree are not showing on Latex. So I included screenshot of them here.



3. The three plots for the `puppy.jpg` image.

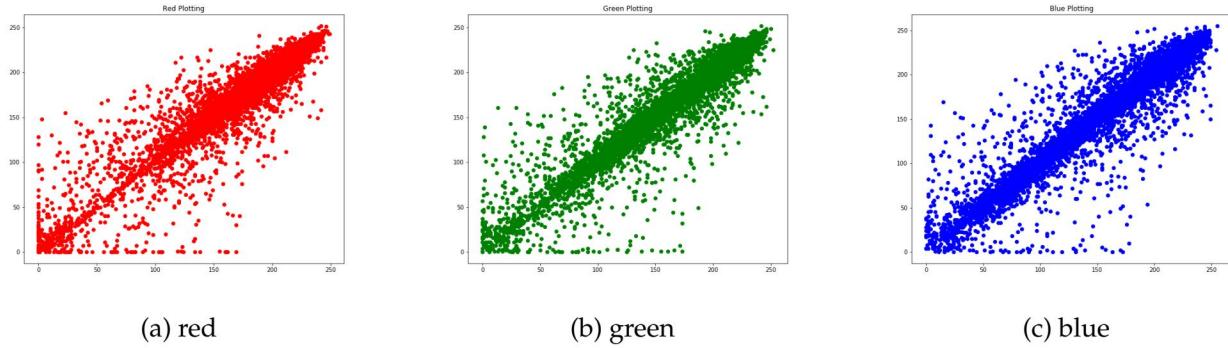


Figure 3: The value of pixels against the values of their neighbor.

What I observe: If the plotted graph showed very high correlation then our demosaic algorithm will work perfectly but there are some outliers on each channel so that we can not avoid losing some information by that errors.

5 Solution code

Include the source code for your solutions as seen below (only the files you implemented are necessary). In latex the command `\verb+input{alignChannels.m}` allows you to include the code verbatim as seen below. Regardless of how you do this the main requirement is that the included code is readable (use proper formatting, variable names, etc.) A screenshot of your code works too provided you include a link to source files.

5.1 alignChannels.py

```
# This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2018  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Homework 1  
  
import numpy as np  
  
def alignChannels(img, max_shift):  
  
    # Check if image is big enough to do cropping for the better alignment  
    if img.shape[0] > (max_shift[0]*20) and img.shape[1] > (max_shift[1]*20):  
        # print("Image is big enough to do Cropping!!! for better alignments")  
        img_aligned = img[max_shift[0]*2:-max_shift[0]*2, max_shift[1]*2:-max_shift[1]*2]  
    else:  
        # image is small, so no cropping.  
        img_aligned = img[:, :, :]  
  
    # Get a reference channel to compare when align  
    red = img_aligned[:, :, 0]  
    # convert it to a 1D vector  
    r_flat = red.flatten()  
  
    # storage to record shifting numbers  
    pred_shift = np.array([[0, 0], [0, 0]])  
  
    # Compare: RED -> BLUE  
    # search for all possible combination of shifts for one color channel  
    max_dot_b = 0  
    for i in range(-max_shift[0], max_shift[0]+1):  
        for j in range(-max_shift[1], max_shift[1]+1):  
            # get separate 2D matrix for a color  
            blue = np.roll(img_aligned[:, :, 1], [i, j], axis=[0, 1])  
  
            # convert the obtained 2D matrix into a flatten matrix so it becomes a 1D vector  
            b_flat = blue.flatten()  
  
            # get a dot product(inner product) between two color channel vectors to check s  
            new_dot = np.dot(r_flat, b_flat)  
  
            # if found a better shift(smaller scalar value from dot product)  
            if new_dot > max_dot_b:  
                max_dot_b = new_dot  
                pred_shift[0] = np.array([i, j])  
  
    # Compare: RED -> GREEN  
    # search for all possible combination of shifts for one color channel  
    max_dot_g = 0  
    for i in range(-max_shift[0], max_shift[0]+1):  
        for j in range(-max_shift[1], max_shift[1]+1):
```

```

# get separate 2D matrix for a color
green = np.roll(img_aligned[:, :, 2], [i, j], axis=[0, 1])

# convert the obtained 2D matrix into a flatten matrix so it becomes a 1D vector
g_flat = green.flatten()

# get a dot product(inner product) between two color channel vectors to check similarity
new_dot = np.dot(r_flat, g_flat)

# if found a better shift(smaller scalar value from dot product)
if new_dot > max_dot_g:
    max_dot_g = new_dot
    pred_shift[1] = np.array([i, j])

# align the image with predicted shifting numbers
img_aligned[:, :, 1] = np.roll(img_aligned[:, :, 1], [pred_shift[0][0], pred_shift[0][1]], axis=[0, 1])
img_aligned[:, :, 2] = np.roll(img_aligned[:, :, 2], [pred_shift[1][0], pred_shift[1][1]], axis=[0, 1])

return (img_aligned, pred_shift)

```

5.2 mosaicImage.py

```
# This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2018  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Homework 1  
  
import numpy as np  
  
def mosaicImage(img):  
    ''' Computes the mosaic of an image.  
  
    mosaicImage computes the response of the image under a Bayer filter.  
  
    Args:  
        img: NxMx3 numpy array (image).  
  
    Returns:  
        NxM image where R, G, B channels are sampled according to RGRG in the  
        top left.  
    '''  
  
    image_height, image_width, num_channels = img.shape  
    assert(num_channels == 3) #Checks if it is a color image  
  
    # make an empty matrix  
    result = np.empty((image_height, image_width))  
  
    # take red channel  
    result[::2, ::2] = img[::2, ::2, 0]  
  
    # take blue channel  
    result[1::2, 1::2] = img[1::2, 1::2, 1]  
  
    # take green channel  
    result[1::2, 0::2] = img[1::2, 0::2, 2]  
    result[0::2, 1::2] = img[0::2, 1::2, 2]  
  
    return result
```

5.3 demosaicImage.py

```
# This code is part of:  
#  
# CMPSCI 370: Computer Vision, Spring 2018  
# University of Massachusetts, Amherst  
# Instructor: Subhransu Maji  
#  
# Homework 1  
  
import numpy as np  
  
def demosaicImage(image, method):  
    """ Demosaics image.  
  
    Args:  
        img: np.array of size NxM.  
        method: demosaicing method (baseline or nn).  
  
    Returns:  
        Color image of size NxMx3 computed using method.  
    """  
  
    if method.lower() == "baseline":  
        return demosaicBaseline(image.copy())  
    elif method.lower() == 'nn':  
        return demosaicNN(image.copy()) # Implement this  
    else:  
        raise ValueError("method {} unkown.".format(method))  
  
def demosaicBaseline(img):  
    """Baseline demosaicing.  
  
    Replaces missing values with the mean of each color channel.  
  
    Args:  
        img: np.array of size NxM.  
  
    Returns:  
        Color image of sieze NxMx3 demosaiced using the baseline  
        algorithm.  
    """  
    mos_img = np.tile(img[:, :, np.newaxis], [1, 1, 3])  
    image_height, image_width = img.shape  
  
    red_values = img[0:image_height:2, 0:image_width:2]  
    mean_value = red_values.mean()  
    mos_img[:, :, 0] = mean_value  
    mos_img[0:image_height:2, 0:image_width:2, 0] = img[0:image_height:2, 0:image_width:2]  
  
    blue_values = img[1:image_height:2, 1:image_width:2]  
    mean_value = blue_values.mean()  
    mos_img[:, :, 2] = mean_value
```

```

mos_img[1:image_height:2, 1:image_width:2, 2] = img[1:image_height:2, 1:image_width:2]

mask = np.ones((image_height, image_width))
mask[0:image_height:2, 0:image_width:2] = -1
mask[1:image_height:2, 1:image_width:2] = -1
green_values = mos_img[mask > 0]
mean_value = green_values.mean()

green_channel = img
green_channel[mask < 0] = mean_value
mos_img[:, :, 1] = green_channel

return mos_img

def demosaicNN(img):
    '''Nearest neighbor demosaicing.

    Args:
        img: np.array of size NxM.

    Returns:
        Color image of size NxMx3 demosaiced using the nearest neighbor
        algorithm.
    '''
    # Check if image has odd height or width
    image_height, image_width = img.shape
    is_odd_height = 0 if image_height % 2 == 0 else 1
    is_odd_width = 0 if image_width % 2 == 0 else 1

    # new height and width
    nH = image_height - 1 if is_odd_height == 1 else image_height
    nW = image_width - 1 if is_odd_width == 1 else image_width

    # empty array for return
    result = np.empty((image_height, image_width, 3))

    #### RED ####

    # 1. retrieve original R value
    # Position: X O
    #           O O
    result[:nH:2, :nW:2, 0] = img[:nH:2, :nW:2]

    # 2. Filling in missing values
    result[:nH:2, 1:nW:2, 0] = img[:nH:2, :nW:2] # upper right - copy from upper left
    result[1:nH:2, 1:nW:2, 0] = img[:nH:2, :nW:2] # lower right - copy from upper left
    result[1:nH:2, :nW:2, 0] = img[:nH:2, :nW:2] # lower left - copy from upper left
    # 3. Handle Edge Cases
    if is_odd_height:
        result[-1, ::, 0] = result[-2, ::, 0]
    if is_odd_width:

```

```

    result[:, :, -1, 0] = result[:, :, -2, 0]

    ##### BLUE #####
    # 1. retrieve original B value
    # Position: O O
    #           O X
    result[1:nH:2, 1:nW:2, 1] = img[1:nH:2, 1:nW:2]

    # 2. filling in missing values
    result[1:nH:2, 0:nW:2, 1] = img[1:nH:2, 1:nW:2] # lower left - copy from lower right
    result[:nH:2, :nW:2, 1] = img[1:nH:2, 1:nW:2] # upper left - copy from lower right
    result[:nH:2, 1:nW:2, 1] = img[1:nH:2, 1:nW:2] # upper right - copy from lower right
    # 3. Handle Edge Cases
    if is_odd_height:
        result[-1, :, 1] = result[-2, :, 1]
    if is_odd_width:
        result[:, -1, 1] = result[:, -2, 1]

    ##### GREEN #####
    # 1. retrieve original B value
    # Position: O O
    #           X O
    result[1:nH:2, 0:nW:2, 2] = img[1:nH:2, 0:nW:2]
    # Position: O X
    #           O O
    result[0:nH:2, 1:nW:2, 2] = img[0:nH:2, 1:nW:2]
    # 2. filling in missing values
    result[:nH:2, :nW:2, 2] = img[:nH:2, 1:nW:2] # upper left - copy from upper right
    result[1:nH:2, 1:nW:2, 2] = img[1:nH:2, :nW:2] # lower right - copy from lower left
    # 3. Handle edge cases
    if is_odd_height:
        result[-1, :, 2] = result[-2, :, 2]
    if is_odd_width:
        result[:, -1, 2] = result[:, -2, 2]

return result

```