Homework 01

0856030, 林正偉, aunbdaunbd@gmail.com

Github: https://github.com/lincw6666/Digital_Image_Processing

October 29, 2019

1 Introduction

In homework 01, we are asked to apply some image processing techniques to produce nice images. There are 6 images supplied (5 regular photos, and one CT). We can do intensity transformation, color adjustment and noise reduction. But only apply noise reduction to the CT image.

2 Methods Review

2.1 Intensity Transformation

2.1.1 Gamma Correction

The following is the equation of gamma correction:

$$s = cr^{\gamma}$$

where c is a scaler which usually is one, and r is the origin intensity of the image. Gamma correction will enhance the intensity in some ranges and decrease the others' intensity. The degree of enhancement/decrease depends on the slope of the equation. Remind that we need to normalize the intensity before we apply it.

When the image is too bright or too dark, and we don't want change the intensity linearly, we can apply gamma correction.

2.1.2 Histogram Equalization

Sometimes the intensity isn't distributed evenly on the whole range. This makes the image too dark or too bright. But gamma correction doesn't work since it won't change the distribution significantly. The image might getting darker or brighter after gamma correction, but they have low contract. To deal with this problem, histogram equalization flatten the distribution of the intensity to make it evenly distribute to the whole range.

2.2 Color Adjustment

2.2.1 HSV Adjustment

Map the image from RGB to HSV. Then we can modify its hue, saturation and value, which respond to the pure-color component, how pure the color is, and the brightness. It can make the image be more colorful or enhance the contrast of the color.

2.3 Noise Reduction

2.3.1 Spatial Filters

Two kinds of spatial filters I've used: box filter and Gaussian filter. Box filter gives the same weight for the target pixel and its neighbors. Gaussian filter gives higher weight for the target pixel and decrease the weight by distance for its neighbors.

These two filters are somewhat different. The result of applying box filter seems to be more blur. For Gaussian filter, the result preserves more details of the origin image, it just likes 'smoothing'.

2.3.2 Order-Statistics Filters

Four kind of order-statistics filters I've used: median filter, min/max filter and midpoint filter. They're all non-linear filters. The median filter chooses the median value from it and its neighbors. Likewise, min filter chooses the min value, max filter chooses the max value. Midpoint filter pluses the min and max value, then divides it by two.

2.3.3 Adaptive Filters

The only adaptive filter I've used is bilateral filter. It's somewhat similar to Gaussian filter. But it preserves the gradient of the edge. The edge won't be blurred a lots as Gaussian filter.

2.4 Sharpen Filter

It highlights the edges in the image.

3 Experiments

3.1 GUI

I thought that it is hard to adjust the parameters of each filter without GUI. Therefore, I built a GUI which allows user to:

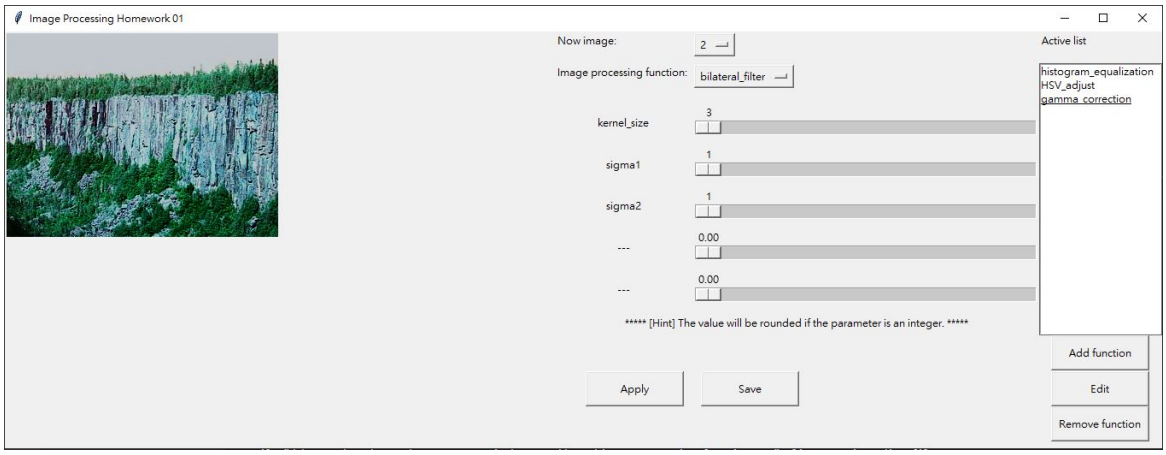
1. Select image.
2. Choose functions to use.
3. Modify the parameters of a function which we've applied.
4. Remove the function we've applied.
5. Save the parameters and the output images.

The GUI system is shown in Figure 1.

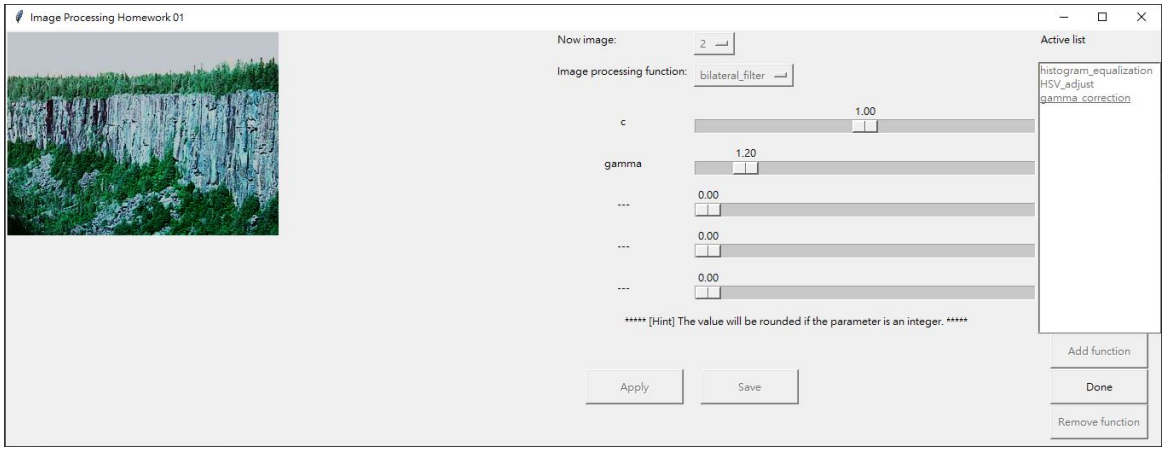
3.2 Image 01

First, the image looked dark, I thought that I should apply intensity transformation. In the beginning, I only used gamma correction. The image got brighter, however it seemed to have low contract. Therefore, I also applied histogram equalization. I flattened it to 0 255. But some parts were too bright. Then I flattened it to 0 245, the result was better.

Second, the image was blurred, I thought that I may sharpen it. In the beginning, I only used sharpen filter, with its center value as nine. There were many vertical and horizontal lines highlighted. Therefore, I used an 5×5 Gaussian filter to smooth it. But it seemed that the Gaussian filter blurred what sharpen filter had sharpened. So I used an 5×5 Gaussian filter instead. Now, the result looked better.



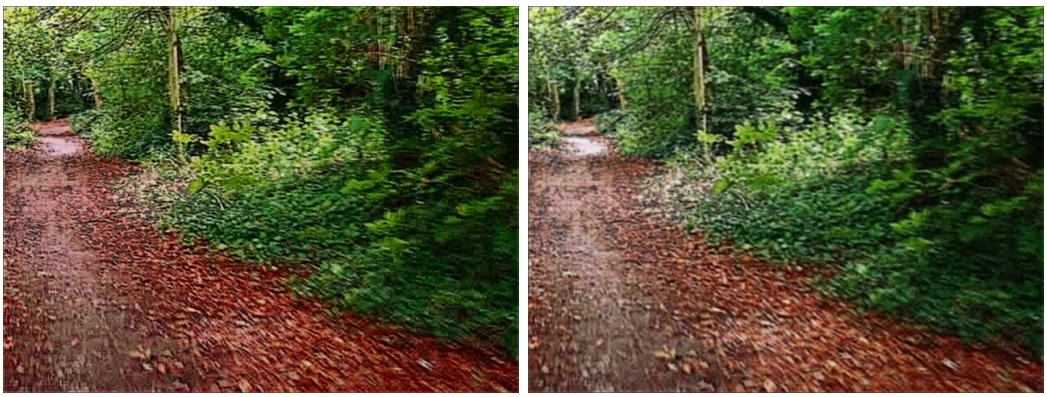
(a) How the GUI looks like.



(b) The GUI when we edit the applied function.

Figure 1: These are my GUI system. (a) shows the how it looks like. (b) shows that when we modify the parameters from the applied function, only scale bar and the "Done" button are active. After modifying the parameters, click the "Done" button then you can go back to the GUI in (a).

Last, I increased the saturation to make it look better "for me". I thought that different people like different result. I show two of my results in Figure 2.



(a) The result I think that is better. (b) The other result I think that is good.

Figure 2: Results of experiments on image 01.

3.3 Image 02

First, the image was too bright, so I applied histogram equalization. The result was perfect.

Next, I thought that the result looked great. The remaining things I could do is adjust the color. Therefore, I applied HSV adjustment. Moreover, I found that gamma correction can make the image look better.

Last, I tried to sharpen the image. But the result was unacceptable.

3.4 Image 03

I think that it's the hardest task. Though I've tried several methods, the result looks like a shit.

First, the image was too dark, so I applied histogram equalization. However, there were plenty of noises. I tried to use Gaussian filter to smooth it. The result looked worse than the origin image. Then, I tried to use gamma correction. It seemed to be better, but it was still worse.

Second, I tried to sharpen the image. This also sharpen the noises. Then I came up my mind that I could use bilateral filter to blur the noise but preserve the sharpened edge. The noises in the sky were fewer than origin. I thought that it's acceptable, but far from perfect.

Last, the remaining things I did are try and error. I applied color adjustment and extra intensity transformation. The final version enhances the saturation, lowers the value, and apply an extra gamma correction with gamma = 0.8.

3.5 Image 04

First, the image seemed a little bit dark, so I applied histogram equalization. The result was good.

Second, I sharpened the image. Here came the problem. There occurred a lot of noises. I tried to apply Gaussian filter and bilateral filter. But they didn't work well. After several tests, I found that the noises gone if I removed histogram equalization. Instead, I enhanced the value, which increased the intensity linearly. This made a good result. One thing I have to mention is that I decreased the range sigma in the bilateral filter to put more effort on preserving the edge. It's really helpful.

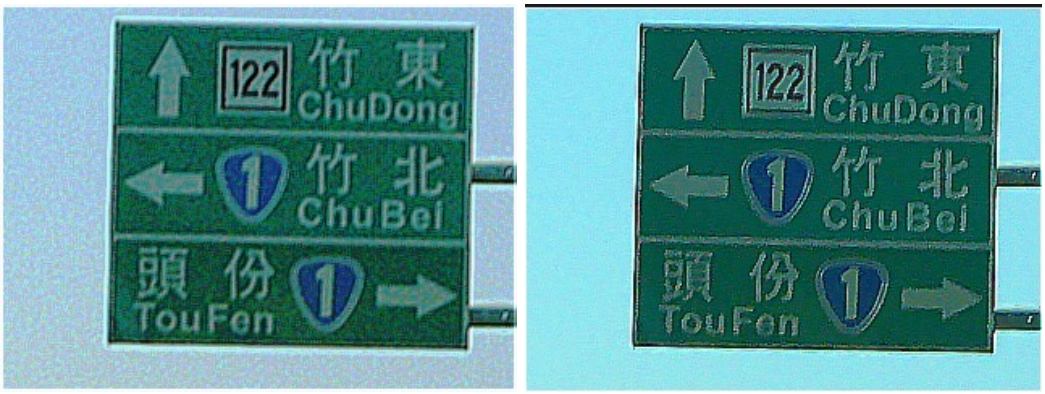
Last, the same as I'd done for the previous images, I adjusted the saturation and hue, and applied gamma correction to get a nice image. These were done by try and error, which meant that I just knew that I might get a better image by adjusting saturation and intensity, but I didn't know how much should I adjust. All done by experiments.

Figure 3 shows that the results with and without histogram equalization, with Gaussian filter v.s. with bilateral filter.

3.6 Image 05

First, the image looked grey, it was very similar to the image before applying histogram equalization on the lecture slide. Therefore, I applied histogram equalization. The result was great.

Next, I tried to sharpen the image. This made a lot of noises, but I thought that it was acceptable. Then, I tried to use bilateral filter to reduce the noises. As previous examples, I used smaller range sigma to preserve the edge. Something interested was that if the kernel size was greater or equal to seven, the result just liked oil painting. Therefore, I set the kernel size to five. The differences are shown in Figure 4b and 4c.



(a) With histogram equalization and applied Gaussian filter. We can see that there are lots of noises.

(b) Without histogram equalization and applied bilateral filter. We can see that the noises are removed.

Figure 3: Results of experiments on image 04.

Last, nothing new, as previous images, I applied HSV adjustment and extra gamma correction.

(a) Applied histogram equalization and sharpen filter.

(b) Applied bilateral filter with kernel size as seven.

(c) Applied bilateral filter with kernel size as five.

Figure 4: Results of experiments on image 05. You can see that (b) is somewhat similar to an oil painting.

3.7 Image 06

It's a CT image. The only process allowed to apply on it is noise reduction. The CT image on the lecture slide applies bilateral filter. Therefore, I applied bilateral filter with different size, also with the smaller range sigma to preserve the edge. The results got stable when the kernel size was larger than eleven.

4 Discussions

4.1 Observation

First, I'm going to talk about the function of order-statistics filter. In the lecture slides, they seem to be powerful on noise reduction. However, they work poor on homework 01. In my opinion, those filters are good at reducing noises on images, which intensity distribution are separated. This means that the intensity is sparse on the whole range. For images with dense intensity, such as homework 01, they work poor.

Next, I find that when we sharpen an image, we also sharpen the noises. Therefore, we need to combine sharpen filter with noise reduction, such as Gaussian filter, box filter and bilateral filter. Comparing these three filters, box filter seems useless. It does nothing but blur the image and destroy the structure. Gaussian filter as least preserves some structure. I think that bilateral filter is the most useful. It preserves the structure and reduces the noises powerfully. And it performs fast. Maybe one draw back is that it is non-linear, so we can't change the order related to other functions.

Last thing I want to talk about is the order of the filters. For linear filters, it doesn't matter. But for non-linear filters, it's a lots matters. Sometimes the result is better when we first apply noise reduction then sharpen the image, vice versa. I still couldn't find the relationship between the order and the quality of the result.

4.2 Functions and Parameters Summary

Table 1 shows the functions I used and the parameters I set.

4.3 Interpretations of Results

For image 01, it is somewhat blurred. I'd applied unmasking method but it didn't work. However, I think that it's already a pretty nice image.

For image 02, I think there's nothing can improve.

For image 03, frankly speaking, it's a shit. Lots of noises and blurred image. But I can't get better result.

For image 04, if we can make the white lines and the edges of the words clearer, it'll be a perfect image. Though, I think it's great now.

For image 05, there are still lots of background noises. And the bilateral filter makes the tree below look much more dense. This is caused by the spatial smoothing of bilateral filter. However, I think the result is acceptable.

For image 06, there are still some noises. I'd used the built in bilateral filter from OpenCV. When we applied it with kernel size as fifteen, the result was a state-of-art. I'm trying to reach that image, but I failed. Though, I think that my result is pretty nice ether.

Table 1: All parameters for applied functions for each image.

Image 01	**Image 02**	**Image 03**
Sharpen Filter 1. center value: 9	Histogram Equalization 1. range: 0 - 255	Gamma Correction 1. c: 1 2. gamma: 0.5
Histogram Equalization 1. range: 0 - 245	HSV Adjustment 1. hue: +10 2. saturation: +10 3. value: +10	Sharpen Filter 1. center value: 9
Gaussian Filter 1. kernel size: 3 2. sigma: 1	Gamma Correction 1. c: 1 2. gamma: 1.2	Bilateral Filter 1. kernel size: 7 2. space sigma: 35 3. range sigma: 35
Gamma Correction 1. c: 1 2. gamma: 1.79		HSV Adjustment 1. hue: +0 2. saturation: +40 3. value: -20
HSV Adjustment 1. hue: -6 2. saturation: +35 3. value: +0		Gamma Correction 1. c: 1 2. gamma: 0.8
Image 04	**Image 05**	**Image 06**
Bilateral Filter 1. kernel size: 5 2. space sigma: 25 3. range sigma: 5	Histogram Equalization 1. range: 30 - 255	Bilateral Filter 1. kernel size:13 2. space sigma: 65 3. range sigma: 11
Sharpen Filter 1. center value: 9	Bilateral Filter 1. kernel size: 5 2. space sigma: 25 3. range sigma: 20	
HSV Adjustment 1. hue: +0 2. saturation: +10 3. value: +40	Sharpen Filter 1. center value: 5	
Gamma Correction 1. c: 1 2. gamma: 1.2	Gamma Correction 1. c: 1 2. gamma: 0.8	
	HSV Adjustment 1. hue: +0 2. saturation: +10 3. value: -10	

5 Code

5.1 main.py

The start point of my APP.

```
import image_processing_data_structure as img_ds
import image_processing_functions as img_f
import image_processing_GUI as img_GUI

# Put images under 'image_dir'. Then load images.
img_ds.load_image(dir='./images/')

img_GUI.start_GUI()
```

5.2 image_processing_functions.py

The following codes implement all image processing functions I'd used.

```
import numpy as np
from cv2 import cv2
import sys


def __is_image_type_valid(img):
    assert type(img)!=type(np.ndarray), '[gamma_correction] Error!!
        Invalid '+\
        'image type!! : '+str(type(img))


def __image_reshape(img, shape):
    __is_image_type_valid(img)
    assert img.shape[2] == 3, '[__image_reshape] Error!! Only
        accept 3 '+\
        'channels image!!'
    h, w, c = shape
    assert c == 3, '[__image_reshape] Error!! Only reshape to 3
        channels '+\
        'image!!'
    assert img.shape[0] == h*w, '[__image_reshape] Error!! Unmatch
        image '+\
        'and reshape image size!!'
    return np.array([img[w*i:w*(i+1), 0, :] for i in range(h)])


# We apply all the function through this wrapper.
def base_func_wrapper(func, img, **kwargs):
    __is_image_type_valid(img)
    if not callable(func):
        print('[base_func_wrapper] Error!! Bad function!!')
        sys.exit(0)
    else:
        #try:
        return func(img, **kwargs)
        #except:
        #    sys.exit(0)
    return None


def __normalize(img):
    return np.uint8(img / 255.0)


def __array_to_vector(img, kernel_size):
    k_rows, k_cols = kernel_size
    padx, pady = k_rows//2, k_cols//2
```

```
    rows, cols, channels = img.shape
    ret = np.zeros((rows+2*padx, cols+2*pady, channels))
    ret[padx:rows+padx, pady:cols+pady, :] = img
    row_len = cols + 2*pady
    row_channel_len = row_len * channels
    start_idx = np.array([
        [j*channels+(row_channel_len)*i + k\
            for i in range(rows-k_rows+1+2*padx)\
                for j in range(cols-k_cols+1+2*pady)]\
                    for k in range(channels)])
    grid = np.array(
        np.tile([j*channels+(row_channel_len)*i\
            for i in range(k_rows) for j in range(k_cols)], (
               channels, 1)))
    to_take = start_idx[:, :, None] + grid[:, None, :]
    return [ret.take(to_take[i]) for i in range(channels)]


def __convolution(img, kernel):
    # Check the kernel's properties.
    assert type(kernel)!=type(np.ndarray), '[convolution] Error!! '+\
       Invalid '+\
        'kernel type!!'

    # Main task.
    vectorized_img = __array_to_vector(img, kernel.shape)
    kernel = kernel.reshape(-1, 1)
    return __image_reshape(
        np.stack(
            [np.matmul(vec_img, kernel) for vec_img in
               vectorized_img], axis=-1
        ),
        img.shape)


def gamma_correction(img, c=1.0, gamma=1.0):
    # # Check arguements.
    # try:
    #     c = float(c)
    #     gamma = float(gamma)
    # except:
    #     print('[gamma_correction] Error!! Invalid arguements\'
       type!!')
    #     sys.exit(0)

    # Main task.
    return np.uint8(c * pow(img/255.0, gamma) * 255.0)


def __check_spacial_filter_kernel_size(kernel_size, err_msg):
```

```
    try:
        kernel_size = int(kernel_size)
        assert kernel_size%2 != 0
        return kernel_size
    except:
        print('['+err_msg+'] Error!! Invalid kernel size!! :',
            kernel_size)
        sys.exit(0)
    return None


def box_filter(img, kernel_size=3):
    kernel_size = __check_spacial_filter_kernel_size(kernel_size, '
        box_filter')
    kernel = np.ones((kernel_size, kernel_size)) / (kernel_size**2)
    return __convolution(img, kernel).astype(np.uint8)


def gaussian_filter(img, kernel_size=3, sigma=1):
    kernel_size = __check_spacial_filter_kernel_size(
        kernel_size, 'gaussian_filter')
    half_size = kernel_size // 2
    x, y = np.mgrid[-half_size:half_size+1, -half_size:half_size+1
        ]
    kernel = np.exp(-(x**2+y**2)/(2.0*sigma**2))
    kernel = kernel / kernel.sum()
    return __convolution(img, kernel).astype(np.uint8)


def bilateral_filter(img, kernel_size=3, sigma1=1, sigma2=0.1):
    kernel_size = __check_spacial_filter_kernel_size(
        kernel_size, 'bilateral_filter')
    # Build a kernel which is similar to Gaussian filter.
    half_size = kernel_size // 2
    x, y = np.mgrid[-half_size:half_size+1, -half_size:half_size+1
        ]
    kernel = -(x**2+y**2)/(2.0*sigma1**2)
    kernel = kernel.reshape(1, -1)
    # Apply additional weight.
    vectorized_img = __array_to_vector(img, (kernel_size,
        kernel_size))
    weight = np.array([
        x - np.repeat(
            [x[:, half_size]], kernel_size**2, axis=0
        ).transpose() for x in vectorized_img])
    kernel = np.exp(kernel - (weight**2)/(2.0*sigma2**2))
    kernel_sum = np.array(
        [np.sum(k, axis=-1).transpose()[:, None] for k in kernel]
    )
    kernel_sum = np.where(kernel_sum > 0, kernel_sum, 1.0)
```

```
    kernel = kernel / kernel_sum
    return __image_reshape(
        np.stack(
            [x for x in np.sum(vectorized_img*kernel, axis=-1)],
               axis=-1
        )[:, None, :],
        img.shape).astype(np.uint8)



def __order_filter(img, kernel_size, order_func, func_name=''):
    kernel_size = __check_spacial_filter_kernel_size(
        kernel_size, func_name)
    return __image_reshape(
        np.stack(
            order_func(
                __array_to_vector(img, (kernel_size, kernel_size)),
                    axis=2
            ), axis=-1
        )[:, None, :],
        img.shape).astype(np.uint8)


def median_filter(img, kernel_size=3):
    return __order_filter(img, kernel_size, np.median, '
       median_filter')


def max_filter(img, kernel_size=3):
    return __order_filter(img, kernel_size, np.max, 'max_filter')


def min_filter(img, kernel_size=3):
    return __order_filter(img, kernel_size, np.min, 'min_filter')


def midpoint_filter(img, kernel_size=3):
    kernel_size = __check_spacial_filter_kernel_size(
        kernel_size, 'midpoint_filter')
    lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)
    min_l = np.min(
        __array_to_vector(
            lab_img[:, :, 0][:, :, None], (kernel_size, kernel_size
               )), axis=2)
    max_l = np.max(
        __array_to_vector(
            lab_img[:, :, 0][:, :, None], (kernel_size, kernel_size
               )), axis=2)
    img_l = (min_l+max_l) / 2.0
    lab_img[:, :, 0] = np.array(
```

```
            [img_l[0, img.shape[1]*i:img.shape[1]*(i+1)]
            for i in range(img.shape[0])])
    return cv2.cvtColor(lab_img, cv2.COLOR_Lab2BGR).astype(np.uint8
        )


def sharpen_filter(img, center_value=5):
    center_value = int(center_value)
    assert (center_value==5) or (center_value==9),\
        '[sharpen_filter] Error!! Invalid center value!! : '+\
            str(center_value)

    if center_value == 5:
        kernel = np.array(
            [[0, -1, 0],
             [-1, 5, -1],
             [0, -1, 0]])
    else:
        kernel = np.array(
            [[-1, -1, -1],
             [-1, 9, -1,],
             [-1, -1, -1]])
    ret = __convolution(img, kernel)
    return np.clip(ret, 0, 255).astype(np.uint8)


def histogram_equalization(img, min_val, max_val):
    hist = np.zeros(256)
    for pixel in img.flatten():
        hist[pixel] += 1
    accumulate = [hist[0]]
    for x in hist:
        accumulate.append(accumulate[-1] + x)
    accumulate = np.array(accumulate)
    new_val = (accumulate-accumulate.min()) * (max_val-min_val) +
        min_val
    accumulate = (
        new_val / (accumulate.max()-accumulate.min())
    ).astype(np.uint8)
    return accumulate[img]


def HSV_adjust(img, H=0, S=0, V=0):
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    hsv_img = np.clip(hsv_img+np.array([H, S, V]), 0, 255)
    return cv2.cvtColor(hsv_img.astype(np.uint8), cv2.COLOR_HSV2BGR
        )


# List all available functions.
```

```
func_list = [
    gamma_correction,
    histogram_equalization,
    box_filter,
    gaussian_filter,
    bilateral_filter,
    sharpen_filter,
    median_filter,
    max_filter,
    min_filter,
    midpoint_filter,
    HSV_adjust,
]
func_name_list = [func.__name__ for func in func_list]
# The name of parameters for each function.
func_param_name = [
    ['c', 'gamma'],      # gamma_correction
    ['min_val', 'max_val'],                    #
       histogram_equalization
    ['kernel_size'],     # box_filter
    ['kernel_size', 'sigma'],     # gaussian_filter
    ['kernel_size', 'sigma1', 'sigma2'],     # bilateral_filter
    ['center_value'],    # sharpen_filter
    ['kernel_size'],     # median_filter
    ['kernel_size'],     # max_filter
    ['kernel_size'],     # min_filter
    ['kernel_size'],     # midpoint_filter
    ['H', 'S', 'V'],     # HSV_adjust
]
param_scale = {
    'c': {'from_': 0, 'to': 2, 'resolution': 0.01},
    'gamma': {'from_': 0, 'to': 10, 'resolution': 0.01},
    'min_val': {'from_': 0, 'to': 100, 'resolution': 1.0},
    'max_val': {'from_': 155, 'to': 255, 'resolution': 1.0},
    'kernel_size': {'from_': 3, 'to': 15, 'resolution': 1.0},
    'sigma': {'from_': 1, 'to': 5, 'resolution': 1.0},
    'sigma1': {'from_': 1, 'to': 100, 'resolution': 1.0},
    'sigma2': {'from_': 1, 'to': 100, 'resolution': 1.0},
    'center_value': {'from_': 5, 'to': 9, 'resolution': 1.0},
    'H': {'from_': -100, 'to': 100, 'resolution': 1.0},
    'S': {'from_': -100, 'to': 100, 'resolution': 1.0},
    'V': {'from_': -100, 'to': 100, 'resolution': 1.0},
}

def func_name_to_id(func_name):
    return func_name_list.index(func_name)
```

5.3 image_processing_data_structure.py

The following codes implement how I store the images and their parameters.

```
import numpy as np
from cv2 import cv2
import image_processing_functions as img_f
import pandas as pd
import sys

image = None


# It stores the image itself and all the parameters for the image
    processing
# functions.
class My_Image:
    # @origin_img: Store the np.ndarray of input image.
    # @img: Store the modified image.
    # @is_func_valid: Show which image processing function is
        applying now.
    #       It's a bitmap.
    # @func_param: Store the parameters for each function.
    def __init__(self, path):
        self.origin_img = None
        self.img = None
        self.img_name = None
        self.func = []
        self.func_param = []
        self.load_image(path)

    # @path: It should start with './'. EX: './images/'.
    def load_image(self, path):
        assert type(path)!=type(str), '[Image:load_image] \'path\'
            need to be'+\
            ' a string!!'
        image = cv2.imread(path)
        # Check whether the image exists.
        if type(image) == type(None):
            # The root directory for the vscode environment locates
                at
            # 'D:/Course/Graduate_1/'. This try block makes it
                runnable on
            # vscode.
            path = path[0] + '/Image_Processing/Homework/HW01' +
                path[1:]
            image = cv2.imread(path)
            if type(image) == type(None):
                print('[Image:load_image] Error!! File not found!!
                    :', path)
                sys.exit(0)
```

```
        self.origin_img = image
        self.img_name = path

    def write_image(self):
        path = self.img_name.split('/')
        path[-1] = path[-1].replace('p', 'P', 1)
        path[-1] = path[-1][:5] + '_0856030' + path[-1][5:]
        path[-2] = 'results'
        path = '/'.join(path)
        try:
            if type(self.img) != type(None):
                cv2.imwrite(path, self.img)
            else:
                cv2.imwrite(path, self.origin_img)
        except:
            print('[Image:write_image]',\
                'Error!! Can\'t save image to file!! :', path)


    # Save the parameters for each function to a file. One line
       contains
    # parameters for exactly one function, with comma seperates
       them.
    #
    # Example:
    # The following shows that (1, 2, 3) for func_1, (4, 5, 6) for
       func_2.
    #     1,2,3
    #     4,5,6
    #
    def save_param(self):
        path = self.img_name.split('/')
        path[-1] = path[-1].replace('bmp', 'csv')
        path[-2] = 'parameters'
        path = '/'.join(path)

        try:
            df = pd.DataFrame({
                'func_name': self.func,
                'func_param': self.func_param
                })
            df.to_csv(path, index=False)
        except:
            print('[Image:save_param]',\
                'Error!! Can\'t save parameters to file!! :', path)

    # Apply all the valid image processing functions on the image.
    def apply_processing_function(self):
        self.img = self.origin_img
        for i in range(len(self.func)):
```

```
                func_id = img_f.func_name_to_id(self.func[i])
                self.img = img_f.base_func_wrapper(
                    img_f.func_list[func_id], self.img,
                    **dict(
                        zip(
                            img_f.func_param_name[func_id], self.
                                func_param[i]
                        )
                    )
                )


def load_image(dir='./images/'):
    global image
    image_name = [dir+'p1im'+str(i)+'.bmp' for i in range(1, 7)]
    image = [My_Image(img_name) for img_name in image_name]
```

5.4 image_processing_GUI.py

The following codes implement the GUI system.

```
import tkinter as tk
from PIL import Image, ImageTk
import time
import image_processing_data_structure as img_ds
import image_processing_functions as img_f

tk_frame = None
app = None


class Application(tk.Frame):
    def __init__(self, master):
        tk.Frame.__init__(self, master)
        self.grid()
        self.create_widgets()


    # Create all the component in the GUI.
    def create_widgets(self):
        # [Canvas] Show the image.
        height, width = img_ds.image[0].origin_img.shape[:2]
        self.canvas_photo =\
            ImageTk.PhotoImage(
                image=Image.fromarray(img_ds.image[0].origin_img))
        self.canvas = tk.Canvas(self, width=width, height=height)
        self.canvas.grid(row=0, column=0, rowspan=16, sticky=tk.NW)
        self.image_on_canvas =\
            self.canvas.create_image(
                0, 0, image=self.canvas_photo, anchor=tk.NW)

        # [Label]
        self.label = [
            # Labels put in front of option menus.
            tk.Label(self, text='Now image: '),
            tk.Label(self, text='Image processing function: '),
            # Labels put in front of scale bars.
            tk.Label(self, text='Parameter 01'),
            tk.Label(self, text='Parameter 02'),
            tk.Label(self, text='Parameter 03'),
            tk.Label(self, text='Parameter 04'),
            tk.Label(self, text='Parameter 05'),
            tk.Label(self, text='***** [Hint] The value will be
                rounded if '+\
                 'the parameter is an integer. *****'),
            tk.Label(self, text='Active list')
        ]
        self.label[0].grid(row=0, column=2, sticky=tk.NW)
```

```
        self.label[1].grid(row=1, column=2, sticky=tk.NW)
        for i in range(2, 7):
            self.label[i].grid(row=i+1, column=2)
        self.label[7].grid(row=8, column=2, columnspan=2)
        self.label[8].grid(row=0, column=4, sticky=tk.NW)

        # [Option Menu]
        # Variables start with 'option_...[]' are related to one of
            the option
        # menu. The ith member means:
        #   0: Choose image which we are going to process.
        #   1: Which image processing function we are using now.
        self.option_list = [
            [i for i in range(1,7)],
            [func.__name__ for func in img_f.func_list]
        ]
        self.option_var = [ tk.IntVar(), tk.StringVar() ]
        self.option_menu = [
            tk.OptionMenu(
                self, self.option_var[0], *self.option_list[0],
                command=self.change_image),
            tk.OptionMenu(
                self, self.option_var[1], *self.option_list[1],
                command=self.change_function)
        ]
        # [Option Menu] Choose image.
        self.option_var[0].set("<-- Choose image -->")
        self.option_menu[0].grid(row=0, column=3, sticky=tk.NW)
        # [Option Menu] Choose image processing function.
        self.option_var[1].set("<-- Choose function -->")
        self.option_menu[1].grid(row=1, column=3, sticky=tk.NW)

        # [Scale] Value of parameters of each function.
        self.scale_var = [tk.DoubleVar() for _ in range(5)]
        self.scale = [
            tk.Scale(
                self, from_=0, to=15, length=400, variable=self.
                    scale_var[i],
                resolution=0.01, orient=tk.HORIZONTAL)
            for i in range(5)
        ]
        for i in range(len(self.scale)):
            self.scale[i].grid(row=3+i, column=3, sticky=tk.NW)

        # [Button]
        self.button = [
            tk.Button(
                self, height=2, width=15, text='Add function',
                command=self.add_function),
            tk.Button(
```

```
                self, height=2, width=15, text='Remove function',
                command=self.del_function),
            tk.Button(
                self, height=2, width=15, text='Apply',
                command=self.apply_function),
            tk.Button(
                self, height=2, width=15, text='Save',
                command=self.save_parameters),
            tk.Button(
                self, height=2, width=15, text='Edit',
                command=self.edit_function)
        ]
        self.button[0].grid(row=9, column=4)
        self.button[1].grid(row=11, column=4)
        self.button[2].grid(row=10, column=2, padx=10, sticky=tk.NE
            )
        self.button[3].grid(row=10, column=3, padx=10, sticky=tk.NW
            )
        self.button[4].grid(row=10, column=4)

        # [List Box] Show the active list.
        self.listbox = tk.Listbox(self, height=20)
        self.listbox.grid(row=1, column=4, rowspan=8, sticky=tk.NW)


    def __get_image_id(self):
        try:
            return self.option_var[0].get() - 1
        except:
            return -1


    def __get_func_id(self, func_name):
        if func_name in img_f.func_name_list:
            return img_f.func_name_to_id(func_name)
        else:
            return -1


    # When we change the function, we need to update the scale in
        the GUI.
    #
    # @id: The id of the selected function in img_ds.func.
    def __update_scale(self, id):
        img_id = self.__get_image_id()
        assert 0 <= id <= img_ds.image[img_id].func, '[
            __update_scale] ' +\
            'Error!! Invalid function id!!'
        if img_id != -1:
            param = img_ds.image[img_id].func_param[id]
```

```
            for i in range(len(param)):
                self.scale[i].set(param[i])
            for i in range(len(param), len(self.scale)):
                self.scale[i].set(0.0)


    # Clear the scale when we change the image or function.
    def __clear_scale(self):
        for i in range(len(self.scale)):
            self.scale[i].set(0.0)


    # Activate while option menu select an image.
    def change_image(self, img_id):
        img_id -= 1
        img = img_ds.image[img_id].img\
            if type(img_ds.image[img_id].img) != type(None)\
                else img_ds.image[img_id].origin_img
        self.canvas_photo = ImageTk.PhotoImage(image=Image.
           fromarray(img))
        self.canvas.itemconfig(self.image_on_canvas, image=self.
           canvas_photo)
        self.__clear_scale()
        self.listbox.delete(0, 'end')
        self.listbox.insert(0, *img_ds.image[img_id].func)


    # Activate while option menu select a function.
    def change_function(self, func_name):
        func_id = self.__get_func_id(func_name)
        func_param_len = len(img_f.func_param_name[func_id])
        param_len = len(self.label) - 2
        for i in range(func_param_len):
            param_name = img_f.func_param_name[func_id][i]
            self.label[i+2]['text'] = param_name
            self.scale[i].config(**img_f.param_scale[param_name])
        if param_len-1 > func_param_len:
            for i in range(func_param_len, param_len-2):
                self.label[i+2]['text'] = '---'
                self.scale[i].config(from_=0, to=15, resolution
                   =0.01)
        self.__clear_scale()


    # When we add a function to the active list, update its
       parameters.
    #
    # @img_id: The [img_id]the image.
    # @func_id: The [func_id]th function in img_f.func_list
    def __add_param(self, img_id, func_id):
```

```
        img_ds.image[img_id].func_param.append(
            [self.scale_var[i].get()
            for i in range(len(img_f.func_param_name[func_id]))]
        )


    # Add function to the active list.
    def add_function(self):
        img_id = self.__get_image_id()
        func_name = self.option_var[1].get()
        func_id = self.__get_func_id(func_name)
        if (img_id!=-1) and (func_id!=-1):
            img_ds.image[img_id].func.append(func_name)
            self.__add_param(img_id, func_id)
            self.listbox.insert('end', func_name)


    # Get the position of the selected line in the listbox.
    def __get_listbox_pos(self):
        ret = self.listbox.curselection()
        if ret != tuple():
            return ret[0]
        else:
            return -1


    # Remove the parameters when we want to remove a function from
       the active
    # list.
    #
    # @img_id: The [img_id]th image.
    # @id: The [id]th function in the list.
    def __del_param(self, img_id, id):
        del img_ds.image[img_id].func[id]
        del img_ds.image[img_id].func_param[id]


    # Remove the function from the active list.
    def del_function(self):
        pos = self.__get_listbox_pos()
        if pos != -1:
            # Update func and func_param in the image.
            img_id = self.__get_image_id()
            self.__del_param(img_id, pos)
            # Update GUI.
            self.listbox.delete(pos)
            self.__clear_scale()


    # Edit the function from the active list.
```

```
    def edit_function(self):
        pos = self.__get_listbox_pos()
        if pos != -1:
            # Disable option menus, buttons and listbox.
            for i in range(len(self.option_menu)):
                self.option_menu[i].config(state=tk.DISABLED)
            for i in range(len(self.button)-1):
                self.button[i].config(state=tk.DISABLED)
            self.listbox.config(state=tk.DISABLED)
            # Update GUI.
            img_id = self.__get_image_id()
            func_id = img_f.func_name_to_id(img_ds.image[img_id].
                func[pos])
            param_len = len(img_ds.image[img_id].func_param[pos])
            for i in range(param_len):
                param_name = img_f.func_param_name[func_id][i]
                self.label[i+2]['text'] = param_name
                self.scale[i].config(**img_f.param_scale[param_name
                    ])
                self.scale[i].set(img_ds.image[img_id].func_param[
                    pos][i])
            for i in range(param_len, len(self.label)-4):
                self.label[i+2]['text'] = '---'
                self.scale[i].config(from_=0, to=15, resolution
                    =0.01)
                self.scale[i].set(0.0)
            self.button[-1]['text'] = 'Done'
            # Change button command.
            self.button[-1].config(command=self.done_edit_function)


    # Finish editting the function in the active list.
    def done_edit_function(self):
        pos = self.__get_listbox_pos()
        if pos != -1:
            # Save the parameters.
            img_id = self.__get_image_id()
            param_len = len(img_ds.image[img_id].func_param[pos])
            img_ds.image[img_id].func_param[pos] = [
                self.scale_var[i].get() for i in range(param_len)
            ]
            # Enable option menus, buttons and listbox.
            for i in range(len(self.option_menu)):
                self.option_menu[i].config(state=tk.NORMAL)
            for i in range(len(self.button)-1):
                self.button[i].config(state=tk.NORMAL)
                self.listbox.config(state=tk.NORMAL)
            # Update GUI.
            self.__clear_scale()
            func_id = img_f.func_name_to_id(self.option_var[1].get
```

```
                ())
            param_len = len(img_f.func_param_name[func_id])
            for i in range(param_len):
                param_name = img_f.func_param_name[func_id][i]
                self.label[i+2]['text'] = param_name
                self.scale[i].config(**img_f.param_scale[param_name
                    ])
            for i in range(param_len, len(self.label)-4):
                self.label[i+2]['text'] = '---'
                self.scale[i].config(from_=0, to=15, resolution
                    =0.01)
            self.button[-1]['text'] = 'Edit'
            # Change button command.
            self.button[-1].config(command=self.edit_function)


    # Apply all image processing function to the image.
    def apply_function(self):
        img_id = self.__get_image_id()
        if img_id != -1:
            img_ds.image[img_id].apply_processing_function()
            self.canvas_photo =\
                ImageTk.PhotoImage(
                    image=Image.fromarray(img_ds.image[img_id].img)
                        )
            self.canvas.itemconfig(
                self.image_on_canvas, image=self.canvas_photo)


    # When the result is good, you can save the parameters and
        output the
    # image.
    def save_parameters(self):
        img_id = self.__get_image_id()
        if img_id != -1:
            img_ds.image[img_id].save_param()
            img_ds.image[img_id].write_image()


# Start point of the GUI app.
def start_GUI():
    global tk_frame, app
    tk_frame = tk.Tk()
    tk_frame.title('Image Processing Homework 01')
    app = Application(tk_frame)
    tk_frame.mainloop()
```