МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики  
Кафедра технической кибернетики

**Отчет по лабораторной работе №5**

Дисциплина: «Системы обработки изображений»

Тема: **«Фильтрация и восстановление изображений»**

Выполнил: Пшенин В.И.

Группа: 6133-010402D

Самара 2020

**Задание на лабораторную работу**

1. Выбрать цветное изображение, по которому будет построен КИХ-фильтр.
2. Выбрать 2 цветных тестовых изображения, одно из которых соответствует по формату изображения из пункта 1 (например, оба изображения представляют собой черно-белую фотографию лиц известных людей), второе – нет.
3. Разделить каждое из 3-х изображений по каналам R, G, B. Каждый канал – это полутоновое изображение, над которым нужно будет проводить дальнейшие пункты.
4. Решить уравнение Винера-Хопфа для двумерного случая (url: <http://sernam.ru/book_kir.php?id=20>). Для этого необходимо тензорное уравнение привести к системе линейных уравнений и решить систему.
   1. Расписать уравнение Винера-Хопфа в двумерном случае, задав множество точек , принадлежащих скользящему окну.
   2. Переписать множество , через последовательность точек .
   3. Переписать систему уравнений на основе полученной последовательности, чтобы получилась явная зависимость каждого уравнения от . Получим  уравнений и  неизвестных. Найденные неизвестные представляют собой значения искомой маски.
   4. Переписать систему в компактном виде. Функции корреляции можно вычислять при помощи кода, представленного в приложении Б.
   5. Создать скрипт для вычисления значений матрицы и вектора для решения системы линейных уравнений .
5. Свернуть найденный вектор обратно в матрицу через ту же зависимость, которая была использована для перехода от множества к последовательности в пункте 4.2. Таким образом, размер маски будет составлять .
6. Протестировать фильтрацию на изображении из пункта 1, осуществив КИХ-фильтрацию с использованием маски, вычисленной на основе методики, описанной в пунктах 3-5.
7. Повторить пункт 6 для того же изображения, предварительно зашумив его с вероятностью искажения *p*.
8. Осуществить зашумление тестовых изображений из пункта 2 с вероятностью искажения пикселя . Осуществить КИХ фильтрацию изображений.
9. Вычислить ошибку восстановления сигнала для всех трёх изображений.

|  |  |
| --- | --- |
| **№ варианта** | **Размер окна *N* и величина искажения *p*** |
| **8** |  |

**Выполнение задания**

Было выбрано 3 изображения, а также осуществлено зашумление с вероятностью искажения пикселя p=0.14. Тестовое изображения и их искаженные варианты представлены в таблице 1.

Таблица 1 – Тестовые изображения

|  |  |  |
| --- | --- | --- |
| **№** | Исходное изображение | Искаженное изображение  (шум p=0.14) |
| **1** | C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (6).png | C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия.png |
| **2** | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (2).png** |  |
| **3** | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (5).png** | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (7).png** |

Каждое изображение было разделено на 3 канала (R,G,B). Задача синтеза квазиоптимального восстанавливающего КИХ-фильтра заключается в определении значений в пределах окна обработки , обеспечивающих минимум среднеквадратичной ошибки восстановления. Они определяются из системы уравнений Винера-Хопфа. Был написан алгоритм поиска КИХ-фильтра, код представлен в приложении А. Ниже представлена таблица 2 с найденными фильтрами.

Таблица 2 – КИХ-фильтры для тестовых изображений

|  |  |  |
| --- | --- | --- |
| Изображение, № | Канал | Фильтр |
| 1 | R | [0.10273098 0.10622496 0.0961732 ]  [0.08021493 0.12617828 0.1272875 ]  [0.11370848 0.15896468 0.15807259] |
| G | [0.16763391 0.1243778 0.13688061]  [0.08434347 0.05904277 0.12036502]  [0.09737352 0.10941505 0.17129695] |
| B | [0.0895824 0.10338878 0.13316335]  [0.10544355 0.12099637 0.13087663]  [0.08492682 0.1168701 0.17805123] |
| 2 | R | [ 0.00175164 -0.06380801 0.02574139]  [ 0.07155224 0.10410308 0.24637985]  [ 0.18551823 0.22793572 0.31104587] |
| G | [0.05505864 0.05611664 0.12054447]  [0.11162004 0.10755889 0.12904714]  [0.17476304 0.16117957 0.1905232 ] |
| B | [0.04397363 0.0622591 0.09563341]  [0.11086584 0.12863807 0.1467238 ]  [0.13542683 0.18192418 0.19707336] |
| 3 | R | [0.11909454 0.12590085 0.12796302]  [0.13221183 0.13359963 0.1220333 ]  [0.10632568 0.10266997 0.10662886] |
| G | [0.09529919 0.15169364 0.13654776]  [0.09643296 0.11313507 0.11767397]  [0.1019935 0.12652691 0.12907299] |
| B | [0.16476226 0.1195617 0.11129174]  [0.12629852 0.11012827 0.10917783]  [0.11837978 0.10577284 0.11195834] |

Осуществим КИХ-фильтрацию с использованием маски, вычисленной ранее. Результат представлен в таблице 3.

Таблица 3 – Результат применения КИХ-фильтров

|  |  |  |
| --- | --- | --- |
| № | Искаженное изображение (шум p=0.14) | Отфильтрованное изображение |
| **1** | C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия.png | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (1).png** |
| **2** |  |  |
| **3** | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (7).png** | **C:\Users\otori\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Без названия (8).png** |

Вычислим ошибку восстановления сигнала.

Таблица 4 – Ошибка восстановление сигнала

|  |  |  |
| --- | --- | --- |
| Изображение, № | Ошибка до фильтрации | Ошибка после фильтрации |
| 1 | 0.212 | 0.093 |
| 2 | 0.217 | 0.102 |
| 3 | 0.215 | 0.093 |

**ПРИЛОЖЕНИЕ А**

**Код программы**

import cv2

import random

from PIL import Image

import numpy as np

from IPython.display import display

# In[2]:

def show(img):

img2 = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

return Image.fromarray(img2)

# In[3]:

def myFilter(img,mask):

newImage = np.copy(img)

ijStart = mask.shape[0] // 2

img = cv2.copyMakeBorder(img, ijStart, ijStart, ijStart, ijStart, cv2.BORDER\_REPLICATE)

for i in range(0,newImage.shape[0]):

for j in range(0,newImage.shape[1]):

frame = img[i:i+ijStart\*2+1,j:j+ijStart\*2+1]

br = int(round(np.sum(mask\*frame)))

if br > 250: br=250

elif br < 0: br=0

newImage[i,j] = br

return newImage

# In[4]:

def myNoisePer(img,per):

newImage = np.copy(img)

countOfPixels = int(img.shape[0]\*img.shape[1]/100\*per)

for c in range(countOfPixels):

i = random.randint(0,img.shape[0]-1)

j = random.randint(0,img.shape[1]-1)

newImage[i,j] = [random.choice([0,255]),random.choice([0,255]),random.choice([0,255])]

return newImage

# In[5]:

def B(A, B, k, l):

rows = A.shape[0]

cols = A.shape[1]

if k < 0 and l < 0:

return np.sum(np.sum(A[0:rows - abs(k), 0:cols - abs(l)] \* B[abs(k):rows, abs(l):cols])) / (

(rows - 1) \* (cols - 1))

elif k < 0 and l >= 0:

return np.sum(np.sum(A[0:rows - abs(k), l:cols] \* B[abs(k):rows, 0:cols - l])) / ((rows - 1) \* (cols - 1))

elif k >= 0 and l < 0:

return np.sum(np.sum(A[k:rows, 0:cols - abs(l)] \* B[0:rows - k, abs(l):cols])) / ((rows - 1) \* (cols - 1))

return np.sum(np.sum(A[k:rows, l:cols] \* B[0:rows - k, 0:cols - l])) / ((rows - 1) \* (cols - 1))

# In[6]:

def WienerHFilter(img,imgNoise,D):

bCrossCor = []

bAutoCor = []

for n in D:

for m in D:

bCrossCor.append(B(img, imgNoise, n, m))

for k in D:

for l in D:

bAutoCor.append(B(imgNoise, imgNoise, n - k, m - l))

bAutoCor = np.array(bAutoCor).reshape(len(D)\*\*2, len(D)\*\*2)

mask = np.linalg.solve(bAutoCor, bCrossCor)

return np.array(mask).reshape(len(D), len(D))

# In[7]:

def getError(img1, img2):

img1 = img1/255

img2 = img2/255

error = (np.sum((np.array(img1.flatten()) - np.array(img2.flatten())) \*\* 2) / (img1.shape[0] \* img2.shape[1] \* 3)) \*\* 0.5

return error

# In[8]:

def doLab(src):

img = cv2.imread(src)

D = [1, 2, 3]

imgNoisy = myNoisePer(img, 14)

b, g, r = cv2.split(img)

bNoise, gNoise, rNoise = cv2.split(imgNoisy)

rF = WienerHFilter(r,rNoise,D)

print(rF)

gF = WienerHFilter(g,gNoise,D)

print(gF)

bF = WienerHFilter(b,bNoise,D)

print(bF)

rOut = myFilter(rNoise, rF)

gOut = myFilter(gNoise, gF)

bOut = myFilter(bNoise, bF)

newImg = cv2.merge((bOut, gOut, rOut))

cv2.imwrite("out"+src, newImg)

display(show(img),show(imgNoisy),show(newImg))

print("Error (noise image): ", getError(img, imgNoisy))

print("Error (after filter): ", getError(img, newImg))