

CodeListingEx4.png

Санкт-Петербургский Национально Исследовательский
Университет
информационных технологий, механики и оптики
Кафедра систем управления и информатики

Основы технического зрения

Отчет по лабораторной работе №1
Фильтрация и выделения контуров

Работу
выполнили:
Суздалев Олег
Караваев А.А
Группа: Р3335
Преподаватель:
Шаветов С.В

Санкт-Петербург
2019

1 Цель работы.

Освоение основных способов фильтрации изображений от шумов и выделения контуров.

2 Теоретическое обоснование применяемых методов и функций фильтрации изображений

В практике почти всегда цифровые изображения будут содержать шумы, обусловленными помехами. В дальнейшем, при обработке изображений различными методами, шумы очень сильно затрудняют их работу, поэтому целесообразно будет использовать прежде всего фильтры.

2.1 Низкочастотная фильтрация.

Низкочастотная фильтрация убирает области с сильным изменением интенсивностей и позволяет исследовать низкочастотную содержащую цифрового изображения.

Особенности:

- Неотрицательные коэф. маски.
- Сумма всех коэф. равна единице.

В данной работе мы будем использовать контргармонический фильтр и фильтр Гаусса.

2.1.1 Контргармонический фильтр.

$$I_{new}(x, y) = \frac{\sum_{i=0}^m \sum_{j=0}^n I(i, j)^{Q+1}}{m \cdot n \cdot \sum_{i=0}^m \sum_{j=0}^n I(i, j)^Q} \quad (1)$$

где Q — порядок фильтра. Контргармонический фильтр является обобщением усредняющих фильтров и при $Q > 0$ подавляет шумы типа «перец», а при $Q < 0$ — шумы типа «соль», однако одновременное удаление белых и черных точек невозможно. При $Q = 0$ фильтр превращается в арифметический, а при $Q = 1$ — в гармонический.

2.1.2 Фильтр Гаусса.

При фильтрации изображений используется двумерный фильтр Гаусса:

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2)$$

Чем больше параметр σ , тем сильнее размывается изображение. Как правило, радиус фильтра $r = 3\sigma$

2.2 Нелинейная фильтрация.

Низкочастотная фильтрация будет работать плохо в любом случае, кроме случая, когда помехи распределены нормально на изображении, поэтому вводится понятие нелинейной фильтрации, которая отлично справляется с импульсными помехами.

2.2.1 Медианная фильтрация.

В классическом медианном фильтре используется маска с единичными коэффициентами. Произвольная форма окна может задаваться при помощи нулевых коэффициентов. Значения интенсивностей пикселей в окне представляются в виде вектора-столбца и сортируются по возрастанию. Отфильтрованному пикселью присваивается медианное (среднее) в ряду значение интенсивности

2.2.2 Взвешенная медианная фильтрация.

Та же самая медианная фильтрация, только теперь маске используются весовые коэффициенты.

2.2.3 Ранговая фильтрация.

Обобщением медианной фильтрации является ранговый фильтр порядка r , выбирающий из полученного вектора-столбца элементов маски пиксель с номером $r \in [1, N]$, который и будет являться результатом фильтрации.

2.2.4 Винеровская фильтрация.

Использует пиксельно-адаптивный метод Винера, основанный на статистических данных, оцененных из локальной окрестности каждого пикселя.

$$\mu = \frac{1}{n \cdot m} \quad (3)$$

$$\sigma^2 = \frac{1}{n \cdot m} \quad (4)$$

$$I_{new}(x, y) = \mu + \frac{\sigma^2 - v^2}{\sigma^2} (I(x, y) - \mu) \quad (5)$$

2.2.5 Адаптивная медианная фильтрация.

В данной модификации фильтра скользящее окно размера $s \times s$ адаптивно увеличивается в зависимости от результата фильтрации. Основной идеей является увеличение размера окна до тех пор, пока алгоритм не найдет медианное значение, не являющееся импульсным шумом, или пока не достигнет максимального размера окна.

2.3 Высокочастотная фильтрация.

Данная фильтрация используется для выделения перепадов интенсивностей и определения границ на изображениях.

2.3.1 Фильтры Собела, Робертса и Превитта.

Используют общий подход с использование масок, которые получают оценки по градиентам.

2.3.2 Фильтр Лапласа.

В отличие от предыдущих фильтров использует производную второго порядка.

2.3.3 Алгоритм Кэнни.

Пожалуй самый распространенный и чаще всего встречающийся на практике алгоритм выделения контуров на изображении. Алгоритм использует несколько шагов, включая фильтрацию исходного изображения использование оператора Собела и последующего анализа значений модуля градиента.

2.4 Исходное изображение.



Рис. 1: Исходное изображение.

2.5 Листинг.

Код написанной библиотеки.

```
1 import cv2
2 import numpy as np
3
4 from scipy.signal import wiener
5 from scipy.ndimage import rank_filter as _rank_filter
6 from typing import Tuple
7
8 def filter(img: np.ndarray, filter: np.ndarray) -> np.ndarray:
9     """ Generic function, that applies filter-mask to each pixel of image
10
11     Arguments:
12         img {np.ndarray} -- input image
13         filter {np.ndarray} -- filter array
```

```

14
15     Returns:
16         np.ndarray -- filtered image
17     """
18     h = w = img.shape[0]
19     f_h, f_w = filter.shape[0], filter.shape[1]
20
21     flat_filter = filter.ravel()
22     return np.correlate(img, flat_filter, 'same')
23
24 def contraharmonic_filter(img: np.ndarray, Q: int, mask_size: tuple) -> np.
25     ndarray:
26     """ Apply contraharmonic filter to input image
27
28     Arguments:
29         img {np.ndarray} -- image
30         Q {int} -- Q parameter
31         mask_size {tuple} -- convolution mask size
32
33     Returns:
34         np.ndarray -- filtered image
35     """
36
37     data = np.array(img, dtype=np.float64)
38     data = data.ravel()
39     num = np.power(data, Q + 1)
40     den = np.power(data, Q)
41     kernel = np.full(mask_size, 1.0)
42
43     res = filter(num, kernel)/filter(den, kernel)
44     return res.reshape(img.shape).astype(np.uint8)
45
46 def wiener_filter(img: np.ndarray, mask_size: tuple) -> np.ndarray:
47     """ Apply wiener filtering to image
48
49     Arguments:
50         img {np.ndarray} -- input image
51         mask_size {tuple} -- convolution mask size
52
53     Returns:
54         np.ndarray -- filtered image
55     """
56
57     data = np.array(img, dtype=np.float64)
58     data = data.ravel()
59     res = wiener(data, mask_size[0])
60     return res.reshape(img.shape).astype(np.uint8)
61
62 def rank_filter(img: np.ndarray, rank: int, mask_size: tuple) -> np.ndarray:
63     """ Apply rank filtering to image
64
65     Arguments:
66         img {np.ndarray} -- input image
67         rank {int} -- rank parameter
68         mask_size{tuple} -- footprint size
69     Returns:
70         np.ndarray -- filtered image
71     """
72
73     data = np.array(img, dtype=np.float64)
74     data = data.ravel()
75     res = _rank_filter(input=data, rank=rank, footprint=mask_size)
76     return res.reshape(img.shape).astype(np.uint8)
77
78 def create_window(x: int, y: int, window: np.ndarray,

```

```

76             img: np.ndarray) -> Tuple[np.ndarray, np.ndarray, int]:
77     """ Creates window, sorts it and finds median
78
79     Arguments:
80         x {int} -- x start idx
81         y {int} -- y start idx
82         win {np.ndarray} -- window array
83         img {np.ndarray} -- img array
84
85     Returns:
86         np.ndarray -- filled window
87         np.ndarray -- target vector
88         int          -- length of vector
89     """
90
91     ax = x
92     ay = y
93
94     W = 2 * window + 1
95     vlength = W * W
96
97     """Creating the window"""
98     filter_window = np.array(np.zeros((W, W)))
99     target_vector = np.array(np.zeros(vlength))
100
101    img = np.pad(img, window, mode='constant')
102
103    """populate window, sort, find median"""
104    filter_window = img[ay:ay+(window*2)+1, ax:ax+(window*2)+1]
105    target_vector = np.reshape(filter_window, ((vlength),))
106
107    return filter_window, target_vector, vlength
108
109 def calc_median(target_array, array_length):
110     sorted_array = np.sort(target_array)
111     Zmed = sorted_array[int(array_length/2)]
112     Zmin = sorted_array[0]
113     Zmax = sorted_array[int(array_length-1)]
114     return Zmin, Zmed, Zmax
115
116 def adaptive_median_filter(img: np.ndarray, mask_size: tuple, window: int,
117                            thresh: float) -> np.ndarray:
118     # TODO for now naive implementation with pure python, can be optimized via
119     # numpy
120     """ Apply adaptive median filter to image
121
122     Arguments:
123         img {np.ndarray} -- input image
124         mask_size {tuple} -- mask size
125         window {int} -- window size
126         thresh {float} -- threshold value
127
128     Returns:
129         np.ndarray -- filtered image
130     """
131
132     min_window, max_window = 1,4
133     xlength, ylength = img.shape[0], img.shape[1]
134
135     img_array = np.reshape(np.array(img), (ylength, xlength))
136     pixel_count = 0

```

```

137     for y in range(0, ylength):
138         for x in range(0, xlength):
139             window = min_window
140             while (window <= max_window):
141                 """Creating and populating window"""
142                 filter_window, target_vector, vlength = create_window(
143                     x, y, window, img_array)
144
145                 """calculating the median for the window"""
146                 Zmin, Zmed, Zmax = calc_median(target_vector, vlength)
147                 A1 = int(Zmed) - int(Zmin)
148                 A2 = int(Zmed) - int(Zmax)
149                 if (A1 > 0 and A2 < 0):
150                     B1 = int(img_array[y, x]) - int(Zmin)
151                     B2 = int(img_array[y, x]) - int(Zmax)
152                     if not(B1 > 0 and B2 < 0):
153                         img_array[y, x] = Zmed
154                         pixel_count += 1
155                         break
156                     else:
157                         break
158                 else:
159                     window += 1
160
161     return np.reshape(img_array, (xlength*ylength,)).astype(np.uint8)

```

Код исполняемого файла лабораторной работы.

```

1 import cv2
2 import numpy as np
3 import matplotlib
4 from PIL import Image
5 from matplotlib import pyplot as plt
6 from skimage.util import random_noise
7 from skimage import img_as_ubyte
8 from skimage.filters import roberts, sobel, prewitt, laplace
9 from closedcv.filters import *
10
11 matplotlib.rcParams["backend"] = "TkAgg"
12 np.set_printoptions(threshold=np.inf)
13
14
15 if __name__=="__main__":
16     img = cv2.imread('./input/img.jpeg', 0)
17
18     # Apply noise to input image
19     gauss = random_noise(img, mode='gaussian', seed=None, clip=True)
20     gauss = img_as_ubyte(gauss)
21     dir_ = './res/'
22     cv2.imwrite(dir_ + 'gauss.jpeg', gauss)
23
24     Q = [-1, 0, 1]
25     for q in Q:
26         cntrharmonic = contraharmonic_filter(gauss, q,(3,3))
27         cv2.imwrite(dir_ + 'contraharmonic_Q_' + str(q) + '.jpg', cntrharmonic)
28
29     for q in Q:
30         gauss.blur = cv2.GaussianBlur(gauss, (3,3), q)
31         cv2.imwrite(dir_ + 'gaussian_Q_' + str(q) + '.jpg', gauss.blur)
32
33     # Median, weighted median, rang and Winner filtering
34     cv2.imwrite(dir_ + 'median.jpg', cv2.medianBlur(gauss, 1))
35
36     cv2.imwrite(dir_ + 'weighted_median.jpg', cv2.medianBlur(gauss, 3))

```

```

37     cv2.imwrite(dir_ + 'wiener.jpg', wiener_filter(gauss, (3,3)))
38
39     cv2.imwrite(dir_ + 'rank.jpg', rank_filter(gauss, -1, (3,3)))
40
41 #cv2.imwrite(dir_ + 'adaptive_median.jpg', adaptive_median_filter(gauss,
42 (3,3), 5, 5))
43
44 # Edge detectors
45 #roberts_ = img_as_ubyte(roberts(img))
46 cv2.imwrite(dir_ + 'roberts.jpg', img_as_ubyte(roberts(img)))
47 cv2.imwrite(dir_ + 'sobel.jpg', img_as_ubyte(sobel(img)))
48 cv2.imwrite(dir_ + 'prewitt.jpg', img_as_ubyte(prewitt(img)))
49 cv2.imwrite(dir_ + 'laplace.jpg', img_as_ubyte(laplace(img)))
50 cv2.imwrite(dir_ + 'canny.jpg', cv2.Canny(img, 100, 200))

```

2.6 Результирующие изображения.



Рис. 2: Зашумленное изображение.



Рис. 3: Контр-гармонический фильтр с $Q=-1,0,1$.



Рис. 4: Фильтр Гаусса с $Q=-1,0,1$.

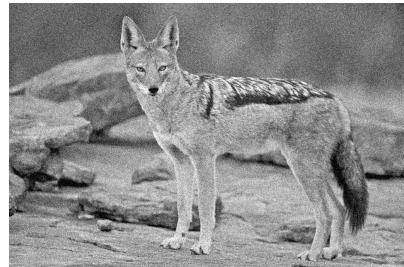


Рис. 5: Медианный фильтр.



Рис. 6: Взвешенный медианный фильтр.



Рис. 7: Винеровская фильтрация.



Рис. 8: Ранговая фильтрация.

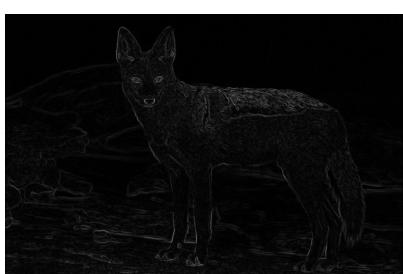


Рис. 9: Фильтр Собела, Робертса и Превитта

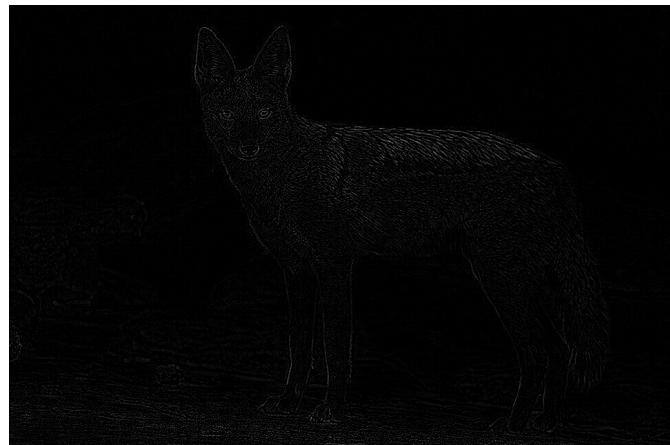


Рис. 10: Фильтр Лапласа.

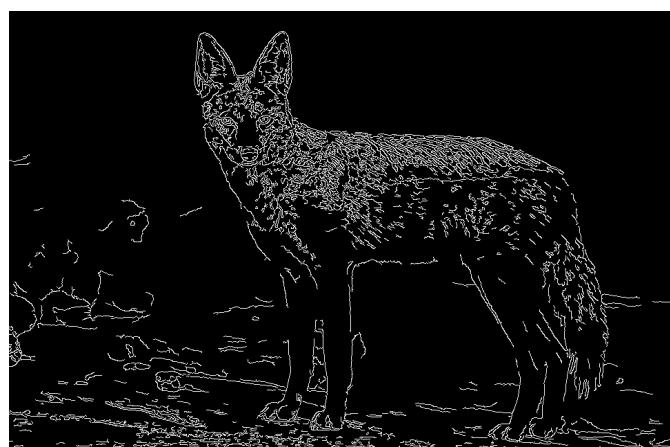


Рис. 11: Алгоритм Кэнни.

3 Вывод.

В данной работе были изучены низкочастотные, нелинейные и высокочастотные фильтры. Универсального фильтра на каждый случай жизни не существует, но было установлено что при низких частотах лучше работает - Гаусса, а при высоких - многоступенчатый алгоритм Кенни.