КУРЕЦ Любовь ИСУ

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

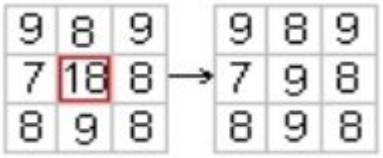
Устранение шумов на полутоновом изображении (усредняющий и медианный фильтр)

Усредняющий фильтр

Работа усредняющего фильтра заключается в замене яркости в данном пикселе на среднюю яркость, вычисленную в его 8-окрестности, включая и сам пиксель:

$$a_9 = \frac{1}{9}\sum_i \sum_j a_{ij}$$

Достоинством усредняющего фильтра является его простота. К недостаткам можно отнести сглаживание ступенчатых и пилообразных функций. Кроме того, пиксели, имеющие существенное отличие в значении яркости и являющиеся шумовыми, будут вносить значительный вклад в результат обработки.



Медианный фильтр

Яркость a9 заменяется медианой пикселей, попадающих в окно матрицы (маски).

Медианой дискретной последовательности n элементов при нечетном n называется элемент, для которого существует (n-1)/2 элементов меньших или равных ему по величине и (n-1)/2 элементов больших или равных ему по величине.

При обработке пикселей изображения, находящихся на границах экрана, возникают проблемы. Для их решения можно виртуально нарастить изображение его зеркальным отображением на границе.
Фильтр также вызывает уплощение вершины треугольной функции.

ВХОДНОЕ ИЗОБРАЖЕНИЕ

РЕЗУЛЬТАТ

Усредняющий фильтр(значени 2)

Медианный фильтр(значение 3)

Листинг программы

```
def apply_median_filter(image_matrix, matrix_height, matrix_width,
shift):
  image_matrix_with_filter = [[0] * matrix_width for _ in
range(matrix_height)]
  expand_matrix, new_matrix_height, new_matrix_width =
create_expand_matrix(image_matrix=image_matrix,
                                                    matrix_height=matrix_height,
                                                    matrix_width=matrix_width,
                                                    shift=shift)
  for i in range(shift, new_matrix_height - shift):
    for j in range(shift, new_matrix_width - shift):
      array = []
      for pos_i in range(i - shift, i + shift):
        for pos_j in range(j - shift, j + shift):
          array.append(expand_matrix[pos_i][pos_j])
      array.sort()
      image_matrix_with_filter[i - shift][j - shift] = array[len(array) // 2]
  return image_matrix_with_filter
def create_average_operator(matrix_order, default_value=1):
  matrix = [[default_value] * matrix_order for _ in range(matrix_order)]
  matrix_sum = 0
  median_element = matrix_order // 2
  for i in range(matrix_order):
    for j in range(matrix_order):
      if i <= median_element and j <= median_element:
        matrix[i][j] **= i + j
      elif i <= median_element:
        matrix[i][j] = matrix[i][matrix_order - 1 - j]
      elif j <= median_element:
        matrix[i][j] = matrix[matrix_order - 1 - i][j]
      else:
        matrix[i][j] = matrix[matrix_order - 1 - i][matrix_order - 1 - j]
      matrix_sum += matrix[i][j]
```

```
  return matrix, matrix_sum
def apply_averaging_filter(image_matrix, matrix_height, matrix_width,
shift):
  average_operator_matrix, matrix_sum = create_average_operator(2 *
shift + 1)
  image_matrix_with_filter = [[0] * matrix_width for _ in
range(matrix_height)]
  expand_matrix, new_matrix_height, new_matrix_width =
create_expand_matrix(image_matrix=image_matrix,
                                                  matrix_height=matrix_height,
                                                  matrix_width=matrix_width,
                                                  shift=shift)

  for i in range(shift, new_matrix_height - shift):
    for j in range(shift, new_matrix_width - shift):
      sub_matrix = [el[j - shift: j + shift + 1] for el in expand_matrix[i -
shift: i + shift + 1]]
      image_matrix_with_filter[i - shift][j - shift] = round(
        compute_matrix_value(sub_matrix, average_operator_matrix, 3)
/ matrix_sum)
  return image_matrix_with_filter
```