Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э.
Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет ИУ – «Информатика и управление»

Кафедра ИУ-3 – «Информационные системы и телекоммуникации»

# Отчёт по лабораторной работе 1 по дисциплине
# «Цифровая обработка изображений»

Выполнил: Цветков А. А.
Группа: ИУ3-48М
Проверил: Алфимцев А.Н.

Москва, 2017

## Разработка программы Юзабилити-Советчика

По заданию лабораторной работы 1 необходимо: разработать программу по загрузке цифрового изображения, подключить функцию обращения к пикселем цифрового изображения, запрограммировать формулу перевода цифрового изображения в полутоновой формат, провести информационный поиск по способам применения оператора выделения границ Кенни, запрограммировать алгоритм анализа Гештальт-принципов для результата обработки изображения эскиза алгоритмом выделения границ, запрограммировать визуализацию результата юзабилити-анализа.

Исходное изображение и его копия в полутоновом формате изображены на рисунках 1-2.

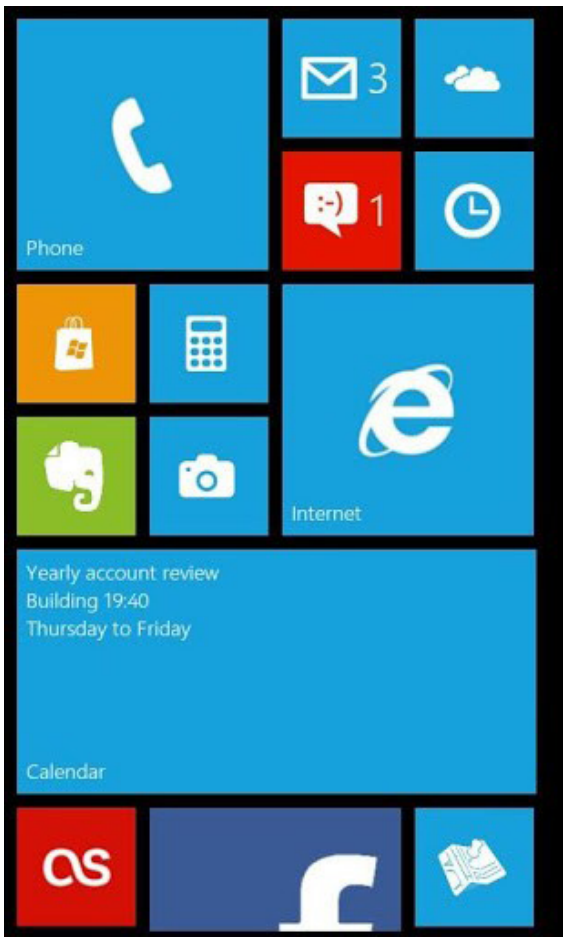


Рисунок 1 – Исходное изображение

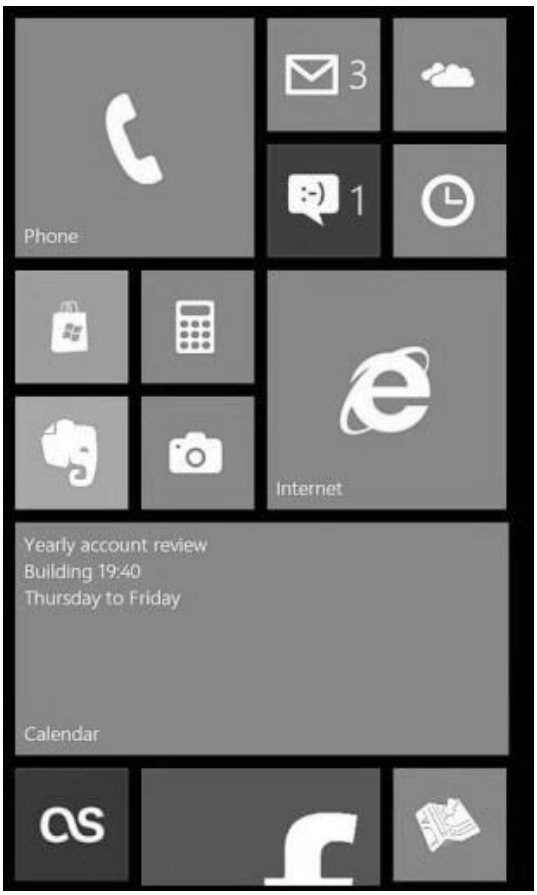


Рисунок 2 – Исходное изображение в полутоновом формате

Для выделения границ объектов используется детектор границ Кенни.

Края (границы) — это такие кривые на изображении, вдоль которых происходит резкое изменение яркости или других видов неоднородностей.

Причины возникновения краёв:

- изменение освещенности;
- изменение цвета;
- изменение глубины сцены (ориентации поверхности).

Получается, что края отражают важные особенности изображения, и поэтому целями преобразования изображения в набор кривых являются:

- выделение существенных характеристик изображения;
- сокращение объема информации для последующего анализа.

Самым популярным методом выделения границ является детектор границ Кенни. Хотя работа Кенни была проведена на заре компьютерного

зрения (1986), детектор границ Кенни до сих пор является одним из лучших детекторов.

Шаги детектора:

- Убрать шум и лишние детали из изображения;
- Рассчитать градиент изображения;
- Сделать края тонкими (edge thinning);
- Связать края в контура (edge linking).

Детектор использует фильтр на основе первой производной от гауссианы. Так как он восприимчив к шумам, лучше не применять данный метод на необработанных изображениях. Сначала, исходные изображения нужно свернуть с гауссовым фильтром. Границы на изображении могут находиться в различных направлениях, поэтому алгоритм Кенни использует четыре фильтра для выявления горизонтальных, вертикальных и диагональных границ.

Воспользовавшись оператором обнаружения границ (например, оператором Собеля, который вычисляет градиент яркости изображения в каждой точке), у нас получается значение для первой производной в горизонтальном направлении (Gy) и вертикальном направлении (Gx). Из этого градиента можно получить угол направления границы:

$$Q = \arctan(Gx/Gy) \quad (1)$$

Угол направления границы округляется до одной из четырех углов, представляющих вертикаль, горизонталь и две диагонали (например, 0, 45, 90 и 135 градусов). Затем идет проверка того, достигает ли величина градиента локального максимума в соответствующем направлении.

Например, для сетки 3x3:

- если угол направления градиента равен нулю, точка будет считаться границей, если её интенсивность больше чем у точки выше и ниже рассматриваемой точки;
- если угол направления градиента равен 90 градусам, точка будет считаться границей, если её интенсивность больше чем у точки слева и справа рассматриваемой точки;
- если угол направления градиента равен 135 градусам, точка будет считаться границей, если её интенсивность больше чем у точек

находящихся в верхнем левом и нижнем правом углу от рассматриваемой точки;

- если угол направления градиента равен 45 градусам, точка будет считаться границей, если её интенсивность больше чем у точек находящихся в верхнем правом и нижнем левом углу от рассматриваемой точки.

Таким образом, получается двоичное изображение, содержащее границы (тонкие края) (рисунок 3)

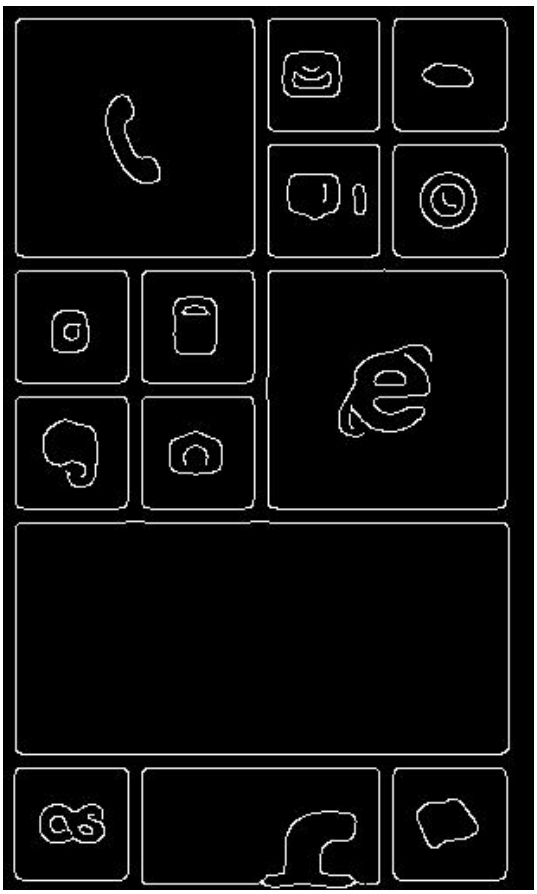

Рисунок 3 – Обнаруженные детектором Кенни границы объектов на изображении

После применения детектора границ Кенни воспользуемся заполним найденные контуры при помощи морфологических преобразований (рисунок 4).

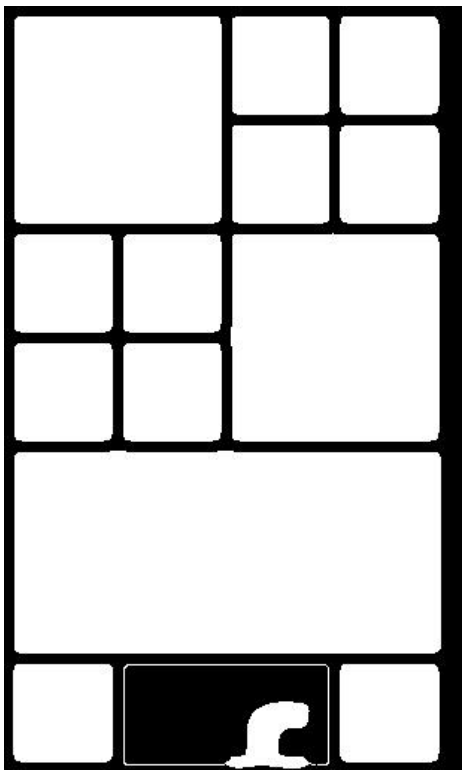

Рисунок 4 – Применение морфологических преобразований

Далее воспользуемся функцией маркировки изображения для поиска цельных объектов. Объекты, площадь которых больше указанной, выделим прямоугольниками, найдем и пронумеруем центральные точки данных объектов (рисунок 5).

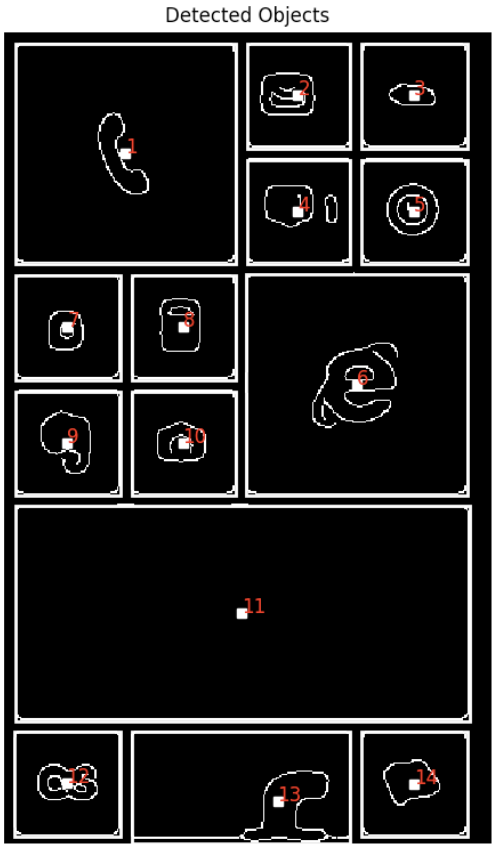


Рисунок 5 – Выделенные объекты и их центральные точки

Для проведения анализа Гештальт-принципов отсортируем отдельно координаты x и координаты y найденных центральных точек. Далее отберем группы точек, разница координат у которых по одной оси не превышает заданного порога (данные точки будут приблизительно лежать вдоль одной прямой).

## Результат работы программы

**Найденные центральные точки:**

[1] x0: 75.9788049956 y0: 75.9784940664
[2] x0: 184.99251812 y0: 39.4774374562
[3] x0: 257.971513218 y0: 39.4569279854
[4] x0: 185.0 y0: 112.439691445
[5] x0: 257.970145852 y0: 112.456927985
[6] x0: 221.906826186 y0: 221.963849182
[7] x0: 39.4561605907 y0: 185.455237656
[8] x0: 112.426558891 y0: 185.456351039
[9] x0: 39.4563510393 y0: 258.426558891
[10] x0: 112.441312384 y0: 258.426756007
[11] x0: 149.463976796 y0: 365.954342984
[12] x0: 39.0291173794 y0: 473.428571429
[13] x0: 172.140123804 y0: 484.902644907
[14] x0: 257.956492027 y0: 473.456947608

**Результат анализа Гештальт-принципов:**

Следующие точки по оси y лежат примерно на одной прямой:

[3] 39.4569279854

[2] 39.4774374562

Следующие точки по оси y лежат примерно на одной прямой:

[4] 112.439691445

[5] 112.456927985

Следующие точки по оси y лежат примерно на одной прямой:

[7] 185.455237656

[8] 185.456351039

Следующие точки по оси y лежат примерно на одной прямой:

[9] 258.426558891

[10] 258.426756007

Следующие точки по оси y лежат примерно на одной прямой:

[12]  473.428571429

[14]  473.456947608

Следующие точки по оси x лежат примерно на одной прямой:

[12]  39.0291173794

[7]  39.4561605907

[9]  39.4563510393

Следующие точки по оси x лежат примерно на одной прямой:

[8]  112.426558891

[10]  112.441312384

Следующие точки по оси x лежат примерно на одной прямой:

[2]  184.99251812

[4]  185.0

Следующие точки по оси x лежат примерно на одной прямой:

[14]  257.956492027

[5]  257.970145852

# Листинг программы

```
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt
from scipy import ndimage as ndi
from skimage import io, color, feature, img_as_uint
from skimage.measure import label, regionprops
import os

basedir = os.path.abspath(os.path.dirname(__file__))

INPUT_DIR = os.path.join(basedir, 'input')
OUTPUT_DIR = os.path.join(basedir, 'output')


def find_row(dict, name, th):
    # sort the input seq
    sorted  y = sorted(dict.items(), key=lambda x: x[1])
    input = []
    for k, v in sorted  y:
        input.append(v)

    res = []
    for i in range(0, len(input) - 1):

        # if two points coord difference is less than the threshold
        if abs(input[i] - input[i + 1]) < th:
            if input[i] not in res:
                res.append(input[i])
            res.append(input[i + 1])
            if i == (len(input) - 2):
                if len(res) > 1:
                    print("Следующие точки по оси " + str(name) + " лежат примерно на одной прямой:")
                    for el in res:
                        for k in dict.keys():
                            if dict[k] == el:
                                print("[" + str(k) + "] ", el)
        else:
            if len(res) > 1:
                print("Следующие точки по оси " + str(name) + " лежат примерно на одной прямой:")
                for el in res:
                    for k in dict.keys():
                        if dict[k] == el:
                            print("[" + str(k) + "] ", el)
            res = []


if __name__ == '__main__':
    # input image as array of bytes
    input_image = io.imread(os.path.join(INPUT_DIR, 'phone.jpg'))

    # convert image from rgb to gray
    image_gray = color.rgb2gray(input_image)

    # save the gray copy of image into a file
    io.imsave(os.path.join(OUTPUT_DIR, 'rgb2gray.jpg'), image_gray)

    # find edges using Canny algorithm
    edges_canny = feature.canny(image_gray, sigma=2.9)

    # filled edges using mathematical morphology
    edges2 = ndi.binary_fill_holes(edges_canny)

    # saving iamge
    io.imsave(os.path.join(OUTPUT_DIR, 'canny.jpg'), img_as_uint(edges_canny))
    io.imsave(os.path.join(OUTPUT  DIR, 'detected.jpg'), img  as  uint(edges2))

    io.imshow(edges  canny)
    io.show()
```

```
io.imshow(edges2)
io.show()

# segment an image with image labelling
label_image = label(edges2)

# create plot for result
fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(edges  canny, cmap=plt.cm.gray)
ax.set_title('Detected Objects')

coord_y = {}
coord_x = {}

i = 1

# find regions and center of each region after image labelling
for region in regionprops(label_image):

    # take regions with large enough areas
    if region.area >= 150:
        # draw rectangle around segmented objects
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='white', linewidth=2)

        ax.add_patch(rect)

        # find center points in segmented objects
        y0, x0 = region.centroid
        coord_y[i] = y0
        coord  x[i] = x0

        ax.plot(x0, y0, 'ws', markersize=6)
        ax.text(x0, y0, str(i), fontsize=12, color="red")

        print("[%s] x0: %s y0: %s" % (i, x0, y0,))
        i += 1

# interface analysis
find_row(coord_y, "y", 5)
find_row(coord_x, "x", 5)

# set axis off
ax.set_axis_off()
plt.tight_layout()
plt.show()
```