



федеральное государственное бюджетное образовательное  
учреждение высшего образования  
**«Национальный исследовательский университет «МЭИ»**

---

Институт информационных и вычислительных технологий  
Кафедра управления и интеллектуальных технологий

Отчёт по лабораторной работе №4  
По дисциплине «Методы и алгоритмы обработки данных и изображений»

Выполнили студенты: Михайловский М. Ю., Озеров С. Д.

Группа: А-02м-25

Бригада: 2

Проверил: Бородкин А. А.

## Содержание

<b>1</b>	<b>Выполнение работы</b>	<b>3</b>
1.1	Предобработка изображения . . . . .	3
1.2	Реализация контурного анализа . . . . .	4
1.3	Классификация символов автомобильных номеров . . . . .	4
<b>2</b>	<b>Выводы</b>	<b>5</b>
<b>A</b>	<b>Листинги</b>	<b>6</b>

## 1 ВЫПОЛНЕНИЕ РАБОТЫ

### 1.1 Предобработка изображения

Предобработка представляет собой сглаживание, бинаризацию и выделение контуров на изображении. Реализованные функции предобработки изображения представлены в листинге 2. Для получения эталонных контуров использовалось изображение, представленное на рис. 1.1. Результат предобработки до выделения контуров представлен на рис. 1.2. Выделенные контуры представлены на рис. 1.3.

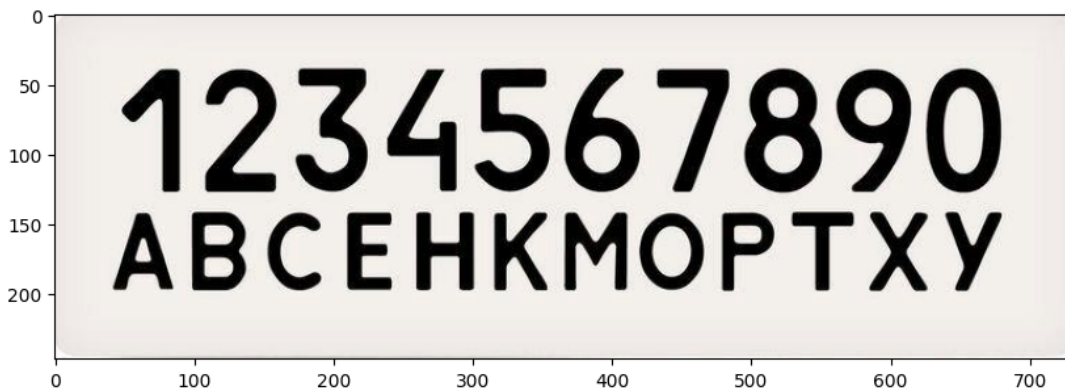


Рис. 1.1. Изображение с эталонными символами



Рис. 1.2. Предобработанное изображение с эталонными символами



Рис. 1.3. Выделенные контуры на эталонном изображении

Из набора выделенных эталонных контуров была собрана библиотека эталонных контуров для каждого возможного символа. Именно они будут использоваться для сравнения с выделенными контурами на реальных изображениях.

При получении контуров реального изображения производится исключение контуров со слишком малой площадью, большой длиной. А также контуры упорядочиваются по координате  $x$  на изображении.

## 1.2 Реализация контурного анализа

Для применения аппарата теории контурного анализа был реализован класс `Contour`, описанный в листинге 3. В нём реализованы основные операции над контурами. Приведём основную информацию.

Контур описывается в виде вектора комплексных чисел:

$$\Gamma = [\gamma_1 \ \gamma_2 \ \dots \ \gamma_n]^T, \ \gamma_i \in \mathbb{C} \quad (1)$$

Здесь  $\gamma_i$  – элементарный вектор (ЭВ), описывающий переход от  $i - 1$  точки контура, к  $i$  точке (для контура с  $n + 1$  точками, начинающимися с 0-го номера).  $\Gamma$  – вектор-контур, описывающий целый контур.

Для вектор-контуров определено скалярное произведение, как сумма скалярных произведений их элементов:

$$(\Gamma, N) = \sum_{n=0}^{k-1} (\gamma_n, \nu_n) \quad (2)$$

За длину вектора принимается сумма длин элементарных векторов:

$$|\Gamma| = \|\Gamma\|_2 = \sqrt{(\Gamma, \bar{\Gamma})} \quad (3)$$

Здесь  $\bar{\Gamma}$  – комплексно-сопряжённое значению  $\Gamma$ .

Для сравнения подобия двух контуров используется взимная корреляционная функция (ВКФ):

$$\tau(m) = (\Gamma, N^{(m)}) \quad (4)$$

Где  $N^{(m)}$  – вектор, полученный из  $N$  циклическим сдвигом его элементов налево на  $m$  позиций.

В качестве меры схожести двух контуров использовалась следующая метрика:

$$\tau_{\max} = \frac{1}{|\Gamma| \cdot |N|} \max \tau(m) \quad (5)$$

Для выравнивания длин вектор-контуров, требуемая определением скалярного произведения, была реализована функция эквализации вектор-контура.

Она позволяет получить схожий вектор-контур по своим характеристикам, но с заданным количеством элементарных векторов в нём.

## 1.3 Классификация символов автомобильных номеров

Для классификации символов на автомобильных номерах был реализован алгоритм, представленный в листинге 5. Приведём пример работы алгоритма на следующем изображении

(рис. 1.4).

Алгоритм определяет схожесть каждого контура с каждым эталонным в соответствии с (5). Если наиболее схожий контур имеет значение  $\tau_{\max} > 0,5$ , то ему назначается метка, такая же как у эталонного контура.

Сами контуры упорядочены по  $x$  координате, первые 6 классифицированных контуров составляет в предсказание номера. В данном примере номер составлен довольно похоже, но с ошибкой. Такой подход к классификации контуров может путать букву "в" с "8".

Также можно обратить внимание, что алгоритм с большой степенью уверенности классифицировал "6" из кода региона как "9". Это связано с тем, что контурный анализ позволяет определить похожие контуры инвариантно к повороту.

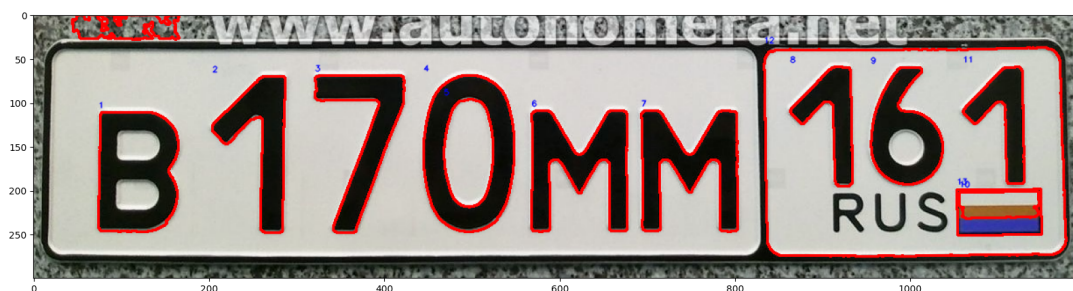


Рис. 1.4. Пример классификации на реальном номере

Листинг 1. Вывод алгоритма классификации контуров

1	Контур	1 наиболее похож на	8 со схожестью	0.72
2	Контур	2 наиболее похож на	1 со схожестью	0.71
3	Контур	3 наиболее похож на	7 со схожестью	0.68
4	Контур	4 наиболее похож на	0 со схожестью	0.73
5	Контур	6 наиболее похож на	М со схожестью	0.61
6	Контур	7 наиболее похож на	М со схожестью	0.61
7	Контур	8 наиболее похож на	1 со схожестью	0.83
8	Контур	9 наиболее похож на	9 со схожестью	0.84
9	Контур	10 наиболее похож на	2 со схожестью	0.54
10	Контур	11 наиболее похож на	1 со схожестью	0.83
11	Контур	13 наиболее похож на	8 со схожестью	0.81
12				
13	Получили номер 8170ММ			
14	Чтение и предобработка изображения: 196мс			
15	Отсев лишних контуров и сортировка по координатам: 11мс			
16	Отрисовка контуров на изображении: 194мс			
17	Классификация контуров: 1699мс			

## 2 ВЫВОДЫ

В этой лабораторной работе были рассмотрены и реализованы функции для применения инструментов контурного анализа изображений. На основе этого метода была реализована классификация символов на автомобильных номерах.

Выделенные контуры получилось классифицировать в соответствии с возможными символами, однако не для всех символов такой подход хорошо работает. Например, цифры 6 и 9 путаются алгоритмом из-за инвариантности к повороту.

## Приложение А. Листинги

### Листинг 2. Загрузка, предобработка и выделение контуров

---

```

1  import os
2  import cv2 as cv
3  import numpy as np
4  import matplotlib.pyplot as plt
5  def imshow_rgb(img, figsize = None):
6      if figsize is None:
7          figsize = (10, 6)
8
9      plt.figure(figsize=figsize)
10     plt.imshow(img)
11     plt.show()
12
13  def imshow_hsv(img):
14      rgb_img = cv.cvtColor(img, cv.COLOR_HSV2RGB)
15      imshow_rgb(rgb_img)
16
17  def imshow_grayscale(img):
18      plt.figure(figsize=(10, 6))
19      plt.imshow(img, cmap='gray')
20      plt.show()
21
22  # Проверяем существование файла
23  picture_files_path = './jpg/'
24  file1 = picture_files_path + 'dict_.jpg'
25
26  def read_binarize(file, show_additional_steps=True):
27      dict_img = cv.imread(file)
28
29      if dict_img is None:
30          print("Ошибка загрузки изображения")
31      else:
32          print(f"Изображение загружено успешно. Размер: {dict_img.shape}")
33          imshow_rgb(dict_img)
34
35      if show_additional_steps:
36          blurred = cv.GaussianBlur(dict_img, (5, 5), 0)
37          cv.GaussianBlur(dict_img, (5,5), 0)
38          imshow_rgb(blurred)
39
40
41      dict_hsv = cv.blur(dict_img, (2, 2))
42      dict_hsv = cv.cvtColor(dict_hsv, cv.COLOR_BGR2HSV)
43      dict_mask = cv.inRange(dict_hsv, (0, 0, 0), (255, 255, 141))
44
45      imshow_grayscale(dict_mask)
46
47      contours, hierarchy = cv.findContours(dict_mask, mode=cv.RETR_TREE,
48      ↪  method=cv.CHAIN_APPROX_NONE)
49      return dict_img, dict_mask, contours, hierarchy

```

---

```

50 dict_img, dict_mask, contours, hierarchy = read_binarize(file1)
51
52
53 def show_img_with_contours(dict_img, contours):
54     img = dict_img.copy()
55     cv.drawContours(img, contours, -1, 255, thickness=2)
56     # 11. Добавление информации о контурах
57     for i, cnt in enumerate(contours):
58         area = cv.contourArea(cnt)
59         x, y, w, h = cv.boundingRect(cnt)
60
61         if area > 100: # Фильтрация по площади
62             cv.putText(img, f'{i}', (x, y-5),
63                        cv.FONT_HERSHEY_SIMPLEX, 0.3, (0, 0, 255), 1, cv.LINE_AA)
64
65     imshow_rgb(img, figsize=(20, 12))
66
67 show_img_with_contours(dict_img, contours)
68
69 sample_contour_id = {
70     'A': 30,
71     'B': 27,
72     'C': 35,
73     'E': 26,
74     'H': 25,
75     'K': 24,
76     'M': 23,
77     'O': 33,
78     'P': 21,
79     'T': 20,
80     'Y': 19,
81     'X': 18,
82     '0': 16,
83     '1': 15,
84     '2': 14,
85     '3': 13,
86     '4': 4,
87     '5': 3,
88     '6': 11,
89     '7': 2,
90     '8': 8,
91     '9': 6
92 }

```

---

Листинг 3. Класс комплексного контура

---

```

1 class Contour:
2     def __init__(self, contour, convertFromDecart = True):
3         if convertFromDecart:
4             complex_contour = self.decartToComplex(contour)
5         else:
6             complex_contour = contour
7

```

```

8         self._set_countour(complex_contour)
9
10    def _set_countour(self, contour):
11        self.contour = contour
12        self._contour_length = self._get_contour_length()
13
14    @staticmethod
15    def tuple_to_complex(point: tuple) -> complex:
16        return complex(point[0][0], point[0][1])
17
18    @staticmethod
19    def multiplyCVectors(a: np.array, b: np.array) -> np.array:
20        return a.T @ b.conjugate()
21
22    def multiplyTo(self, other: "Contour") -> np.array:
23        return Contour.multiplyCVectors(self.contour, other)
24
25    @staticmethod
26    def calc_contour_length(c: np.array):
27        return np.sum(np.sqrt(c * c.conjugate()))
28
29    def _get_contour_length(self):
30        return Contour.calc_contour_length(self.contour)
31
32    def contour_length(self):
33        return self._contour_length
34
35    @staticmethod
36    def VKF(contour1: "Contour", contour2: "Contour"):
37        c1: np.array = contour1.contour
38        c2: np.array = contour2.contour
39
40        res = []
41        for k in range(c1.shape[0]):
42            c2_roll = np.roll(c2, -k)
43            res.append(Contour.multiplyCVectors(c1, c2_roll))
44
45        return np.array(res)
46
47    @staticmethod
48    def contours_similarity(cont1: "Contour", cont2: "Contour"):
49        vkf = Contour.VKF(cont1, cont2)
50        similarity = np.max(np.abs(vkf)) / np.sqrt(cont1.contour_length().real *
51        ↪ cont2.contour_length().real)
52
53        return similarity
54
55    def decartToComplex(self, contour):
56        compl_contour = [ ]
57        last_point = Contour.tuple_to_complex(contour[0])
58        for i, point_tuple in enumerate(contour):
59            if i == 0:
60                continue
61
62            point = Contour.tuple_to_complex(point_tuple)

```

```

62         compl_contour.append(point - last_point)
63         last_point = point
64
65     return np.array(compl_contour)
66
67
68     def equalize(self, p, log=False):
69         eqvec = [complex(0,0)] * p
70         eq_len = self.contour_length() / p
71         if log:
72             print(eq_len)
73         pind = 0
74
75         def calc_norm(vec):
76             if not isinstance(vec, np.ndarray):
77                 vec = np.array(vec)
78                 return np.sqrt(vec * vec.conjugate())
79
80             return np.sqrt(vec.T @ vec.conjugate())
81
82         vec = self.contour
83         vec_ost = vec[0]
84         vec_isp = complex(0,0)
85         j = 0
86         while pind < p - 1:
87             vlen = calc_norm(vec_ost)
88             if vlen > eq_len:
89                 eqvec[pind] = vec_ost * eq_len / vlen
90                 vec_ost = vec_ost - eqvec[pind]
91                 if log:
92                     print('f: ' + str(pind) + ' ' + str(eqvec[pind]))
93                 pind += 1
94             else:
95                 s = calc_norm(vec_ost)
96
97                 for t in range(j + 1, len(vec)):
98                     s0 = s
99
100                     s = s + calc_norm(vec[t])
101
102
103                 if (s > eq_len):
104                     vec_isp_len = eq_len - s0
105                     vec_isp = vec[t] * vec_isp_len /
106                         ↪ calc_norm(vec[t])
107
108                     vecs = complex(0,0)
109                     for tt in range(j+1, t - 1):
110                         vecs = vecs + vec[tt]
111
112                     eqvec[pind] = vec_ost + vec_isp + vecs
113                     vec_ost = vec[t] - vec_isp
114
115                 if log:

```

---

```

116         print(str(pind) + ' ' +
               ↪ str(eqvec[pind]))
117
118         pind += 1
119         j = t
120         break
121
122     if pind >= p - 1:
123
124
125         eqvec[pind] = -sum(eqvec)
126
127         if log:
128             print(str(pind) + ' ' + str(eqvec[pind]))
129         pind += 1
130         break
131     if pind >= p :
132         break
133
134     return Contour(np.array(eqvec), convertFromDecart=False)

```

---

Листинг 4. Формирование набора эталонных контуров, определение функции схожести

---

```

1 sample_contours = dict()
2 for character, id in sample_contour_id.items():
3     sample_contours[character] = Contour(contours[id])
4
5 def calc_similaritites(contour: np.array, equalize_to: int=None):
6     cont = Contour(contour)
7
8     similarities = dict()
9     if equalize_to is None:
10         equalize_to = cont.contour.shape[0]
11     else:
12         cont = cont.equalize(equalize_to)
13
14     for character, sample_contour in sample_contours.items():
15         equalized_sample_contour = sample_contour.equalize(equalize_to)
16
17         similarity = Contour.contours_similarity(cont, equalized_sample_contour)
18         similarities[character] = similarity
19
20     return similarities

```

---

Листинг 5. Классификация контуров по буквам

---

```

1 from time import perf_counter
2
3 to_open = picture_files_path + 'n11.jpg'
4
5 start_time = perf_counter()
6 dict_img, dict_mask, contours, hierarchy = read_binarize(to_open, show_additional_steps=False)

```

---

---

```

7  predobr_end_time = perf_counter()
8
9  filtered_contours = []
10 min_area = 1500
11 max_perimeter = 2000
12 for i, contour in enumerate(contours):
13     area = cv.contourArea(contour)
14     contour_length = Contour(contour).contour_length()
15     if area > min_area and contour_length < max_perimeter:
16         filtered_contours.append(contour)
17
18 def first_point_x(item):
19     return item[0][0][0]
20
21 filtered_contours = list(sorted(filtered_contours, key=first_point_x))
22
23 contours_predobr_end_time = perf_counter()
24 show_img_with_contours(dict_img, filtered_contours)
25 contours_draw_end_time = perf_counter()
26
27 number = ''
28 for i, contour in enumerate(filtered_contours):
29     similarities = calc_similaritites(contour)
30     character, max_similarity = max(similarities.items(), key=lambda x: x[1])
31     if max_similarity < 0.5:
32         continue
33
34     if len(number) < 6:
35         number += character
36     print(f'Контур {i:>3} наиболее похож на {character:>3} со схожестью
37         ↪ {max_similarity:>5.2f}')
38
39 classify_end_time = perf_counter()
40
41 print(f'\nПолучили номер {number}')
42 print(f'Чтение и предобработка изображения: {(predobr_end_time - start_time)*1000:.0f}мс')
43 print(f'Отсев лишних контуров и сортировка по координатам: {(contours_predobr_end_time -
44     ↪ predobr_end_time)*1000:.0f}мс')
45 print(f'Отрисовка контуров на изображении: {(contours_draw_end_time -
46     ↪ contours_predobr_end_time)*1000:.0f}мс')
47 print(f'Классификация контуров: {(classify_end_time - contours_draw_end_time)*1000:.0f}мс')

```

---