



НИУ ИТМО

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«Преобразование Хафа»

Выполнили:

Александр Иванов, Ф ТЕХ.ЗРЕНИЕ 1.1

Ани Аракелян, ТЕХ.ЗРЕНИЕ 1.1

Никита Братушка, ТЕХ.ЗРЕНИЕ 1.3

Преподаватель:

Шаветов С. В.

Санкт-Петербург, 2024

Содержание

1. Поиск прямых	3
1.1. Исходные изображения	3
1.2. Программа на языке Python	4
1.3. Результаты преобразования	6
2. Поиск окружностей	12
2.1. Исходные изображения	12
2.2. Программа на языке Python	13
2.3. Результаты преобразования	15
3. Выводы	19
4. Ответы на вопросы	19

1. Поиск прямых

1.1. Исходные изображения

Выберем изображения, которые содержат прямые линии. Первым изображением будет логотип нашего любимого университета:



Рис. 1: Логотип университета ИТМО

Также возьмём рисунок, составленный из прямых линий:

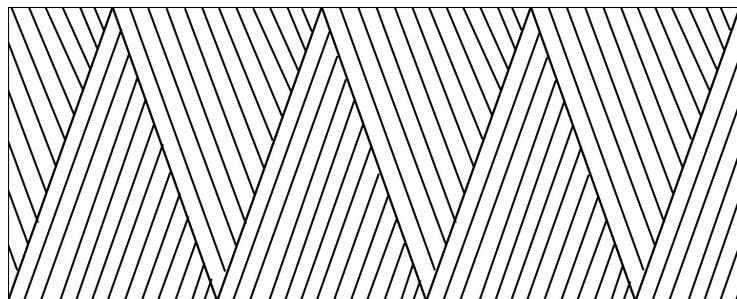


Рис. 2: Паркет ёлочкой

И обратимся к творчеству абстракциониста Пита Мондриана:

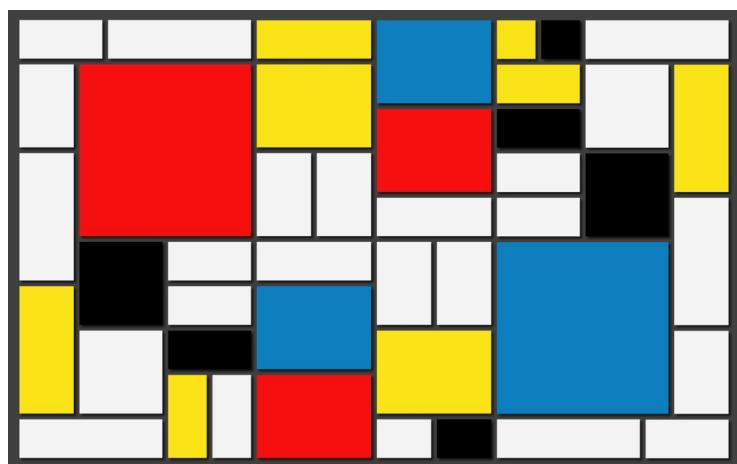


Рис. 3: Цифровая репродукция картины Пита Мондриана

1.2. Программа на языке Python

```
def line_detection(option, image):
    # Grayscale image
    # edged = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    # Canny edges
    edged = cv.Canny(image, 250, 270, None, 3)
    match option:
        case 0:
            # output of the original image
            cv.imshow('Source Image', image)
            cv.waitKey(0)
        case 1:
            # Hough transform
            # creating image copies
            result = image.copy()
            result_p = image.copy()
            # computing lines via classic Hough transform
            lines = cv.HoughLines(edged, 1, np.pi / 180, 150)
            # computing lines via probabilistic Hough transform
            lines_p = cv.HoughLinesP(edged, 1, np.pi / 180, 50, None,
30, 4)
            #
            angles = np.linspace(-np.pi / 2, np.pi / 2, 360,
endpoint=False)
            # Accumulator computing
            ah, _, _ = skimage.transform.hough_line(edged, theta=angles)
            mx = 0
            mn = 10000
            for i in range(len(lines_p)):
                # overlaying lines found by using probabilistic
                # transform and their edges
                line = lines_p[i][0]
                pt1 = (line[0], line[1])
                pt2 = (line[2], line[3])
                cv.line(result_p, pt1, pt2, (0, 255, 255), 5,
cv.LINE_AA)
                cv.circle(result_p, pt1, 5, (0, 0, 255), -1)
                cv.circle(result_p, pt2, 5, (0, 0, 255), -1)
                # computing the shortest and longest lines
                norm = np.linalg.norm(np.array(pt2) - np.array(pt1))
                if norm < mn:
                    mn = round(norm, 2)
                if norm > mx:
                    mx = round(norm, 2)
            for j in range(len(lines)):
                # overlaying lines found by using classic transform and
                # their edges
                rho = lines[j][0][0]
                theta = lines[j][0][1]
                a, b = math.cos(theta), math.sin(theta)
                x0, y0 = a * rho, b * rho
```

```

        pt1 = (np.int32(x0 + 1000 * (-b)), np.int32(y0 + 1000 *
a))
        pt2 = (np.int32(x0 - 1000 * (-b)), np.int32(y0 - 1000 *
a))
        cv.line(result, pt1, pt2, (0, 255, 255), 5, cv.LINE_AA)
        cv.circle(result, pt1, 5, (255, 0, 0), -1)
        cv.circle(result, pt2, 5, (255, 0, 0), -1)
        # cv.imwrite('results/lines/Klinom_p.jpg', result_p)
        # transferring received images and data to function of
        creating a comparative image
        image_displaying(image, edged, result_p, result, ah, mn,
mx, len(lines_p))
    case _:
        print('Wrong option! Enter the right number')

```

Листинг 1: Исходный код функции для нахождения и отображения прямых линий с помощью преобразования Хафа

```

def image_displaying(source, canny, finale_p, finale, parametric, mn,
mx, nmbr):
    # creating figure with mosaic layout
    fig, axs = plt.subplot_mosaic([['src', 'can'], ['fin', 'finp'],
    ['p', 'info']], layout='tight', figsize=(9.2, 7.8))
    # plotting images
    axs['src'].imshow(cv.cvtColor(source, cv.COLOR_BGR2RGB))
    axs['can'].imshow(canny, cmap=matplotlib.cm.gray)
    axs['fin'].imshow(cv.cvtColor(finale, cv.COLOR_BGR2RGB))
    axs['finp'].imshow(cv.cvtColor(finale_p, cv.COLOR_BGR2RGB))
    axs['p'].imshow(cv.resize(parametric.astype(np.float32) /
    np.max(parametric),
                           (parametric.shape[1], 300)),
                  cmap=matplotlib.cm.gray)
    # turning off axes
    axs['src'].axis('off')
    axs['can'].axis('off')
    axs['fin'].axis('off')
    axs['finp'].axis('off')
    axs['p'].axis('off')
    axs['info'].axis('off')
    axs['src'].set_title('Image')
    axs['can'].set_title('Canny edges')
    axs['fin'].set_title('Hough')
    axs['finp'].set_title('Probabilistic Hough')
    axs['p'].set_title('Parameter space')
    # plotting number of lines found, minimum and maximum length
    axs['info'].text(0.5, 0.5, f'Number of lines: {nmbr}\nShortest
line: {mn}\nLongest line: {mx}', size=25,
                     ha='center',
                     va='center',
                     bbox=dict(boxstyle="square",
                               ec=(1., 0.5, 0.5),
                               fc=(1., 0.8, 0.8)
                               )

```

```

        )
# displaying and saving the plot
plt.show()
fig.savefig('results/lines/ITMO.jpg')
plt.close()

```

Листинг 2: Исходный код функции для создания сравнительного изображения с результатами преобразования

1.3. Результаты преобразования

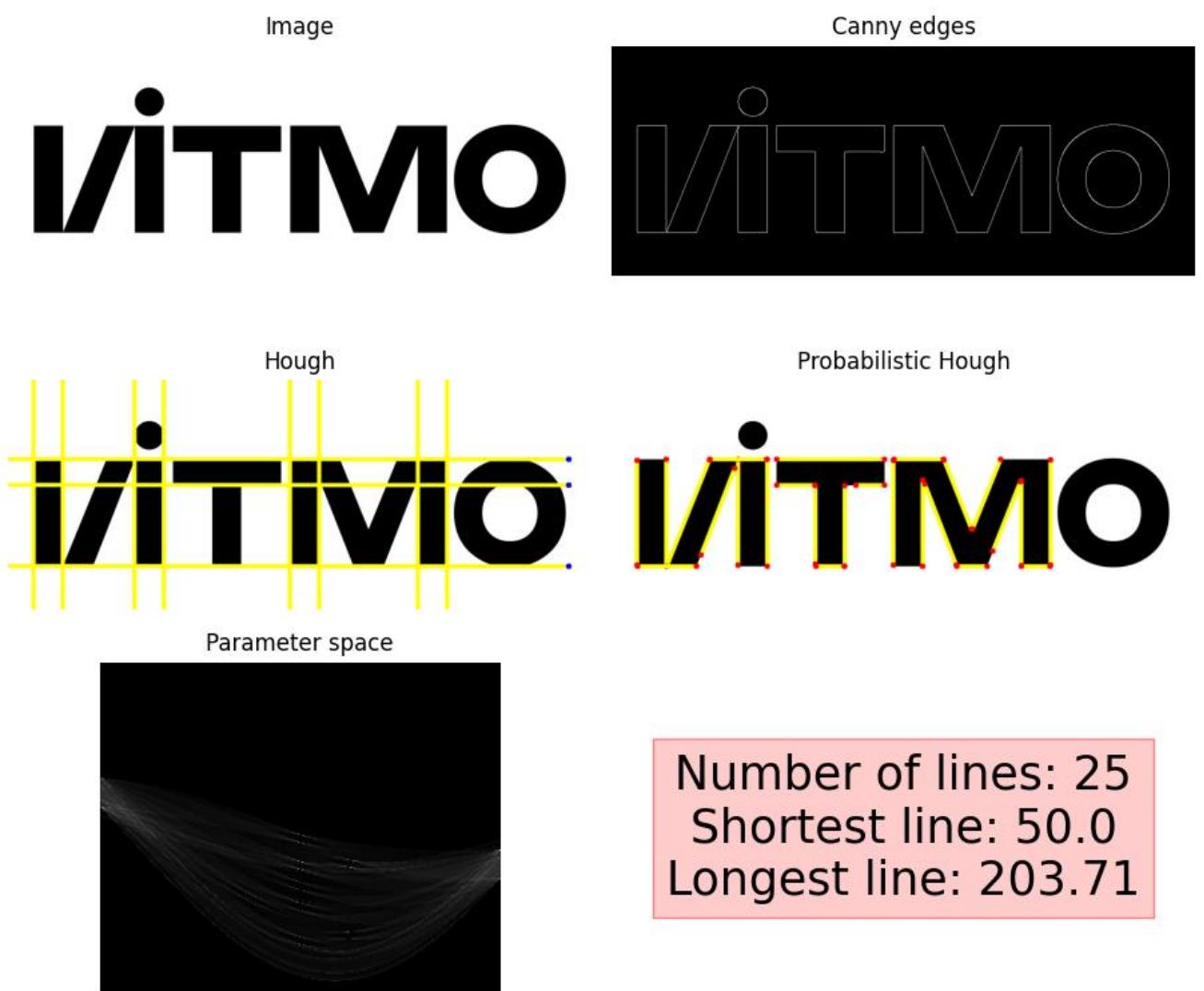


Рис. 4: Исходное изображение 1; контуры изображения, полученные алгоритмом Кэнни; результаты классического и вероятностного преобразованием Хафа; пространство параметров; подсчёт прямых

Вероятностное преобразование Хафа в результате работы выдаёт значения крайних точек распознанных линий, поэтому результат его работы позволяет довольно точно выделить их. Классическое преобразование, в свою очередь, возвращает угол наклона и длину радиуса вектора прямой, которой принадлежит исходная линия. Наши выводы подтверждаются рисунками 4, 2 и 6.

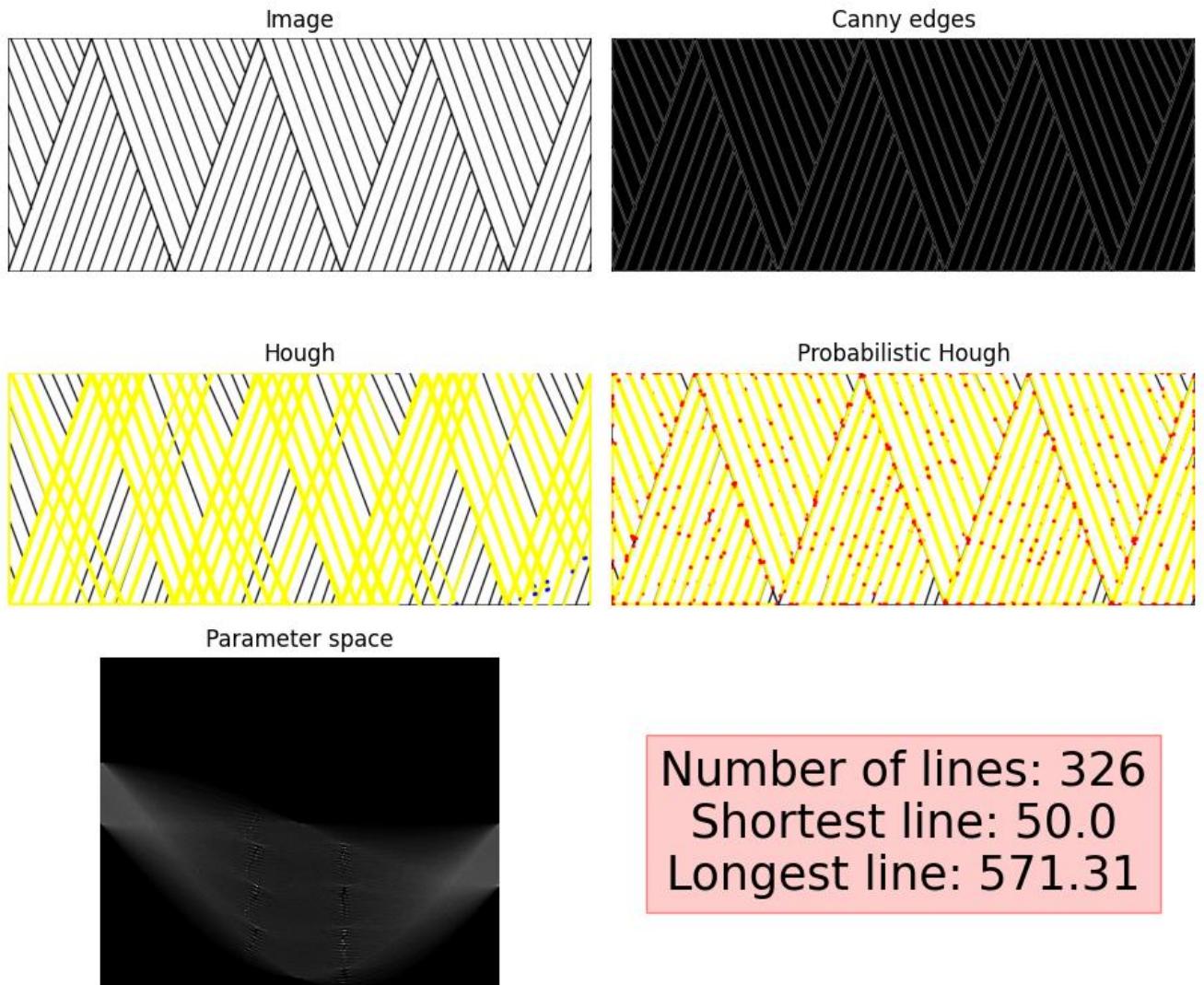


Рис. 5: Исходное изображение 2; контуры изображения, полученные алгоритмом Кэнни; результаты классического и вероятностного преобразованием Хафа; пространство параметров;подсчёт прямых

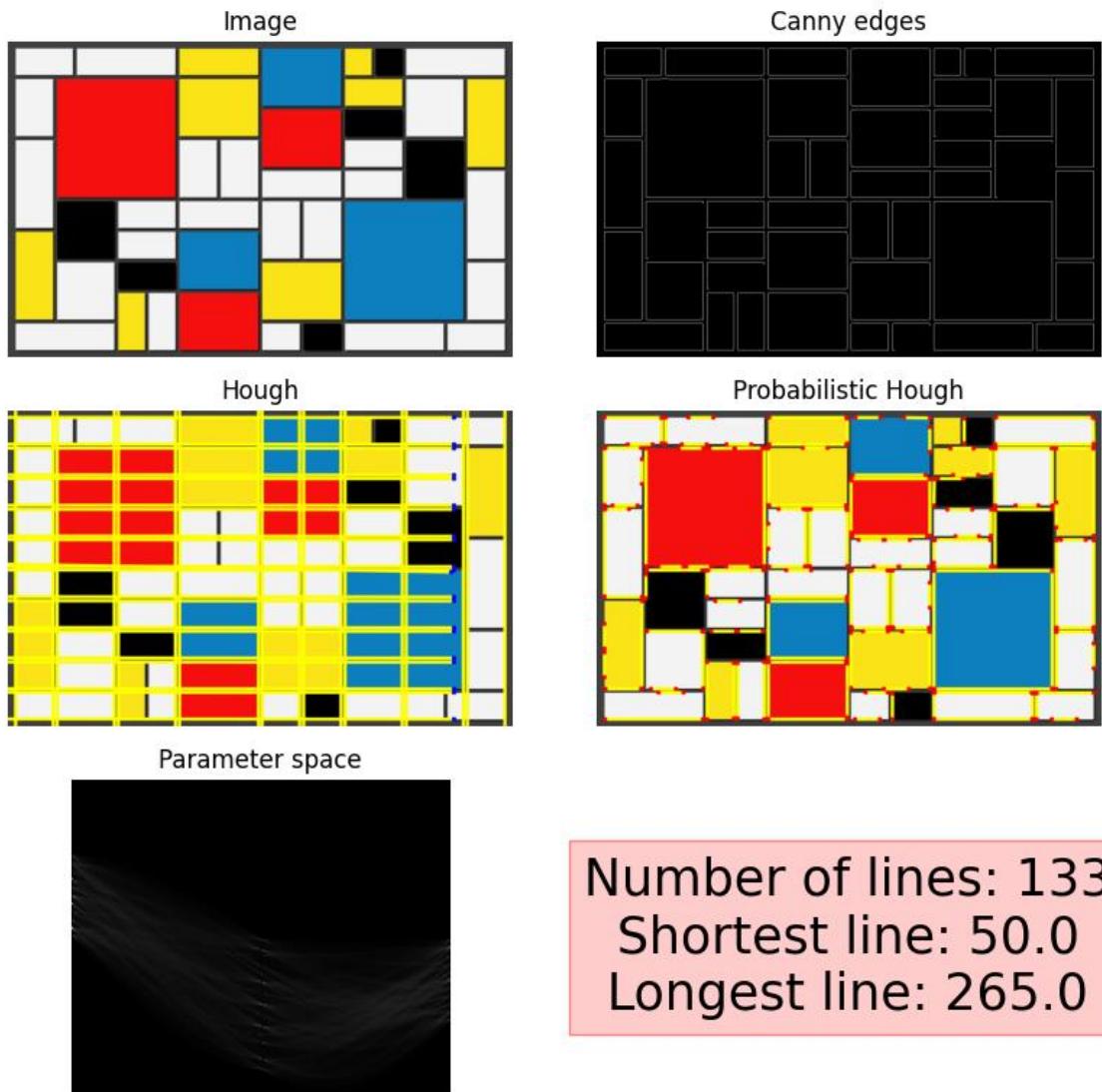


Рис. 6: Исходное изображение 3; контуры изображения, полученные алгоритмом Кэнни; результаты классического и вероятностного преобразованием Хафа; пространство параметров; подсчёт прямых

Теперь попробуем применить преобразование к полутонаовым изображениям:



Рис. 7: Исходное изображение 1; полутонаовое изображение; результаты классического и вероятностного преобразованием Хафа; пространство параметров; подсчёт прямых

Можно констатировать, что в данном случае преобразование Хафа не справляется с поставленной задачей. Это связано с тем, что алгоритм воспринимает в качестве контуров всю область изображения вокруг чёрных участков. Поэтому на изображении с логотипом прямыми линиями были выделены вся область вокруг букв (см. рисунок 7), исключение составили участки, ширина которых составляла менее 30 пикселей. Аналогичный результат получился в случае обработки репродукции Пита Мондриана (см. рисунок 9).

Из-за тонких чёрных линий 2 изображения (рисунок 2) линии полностью покрыли его, в результате чего мы получили жёлтый прямоугольник (см. рисунок 8).

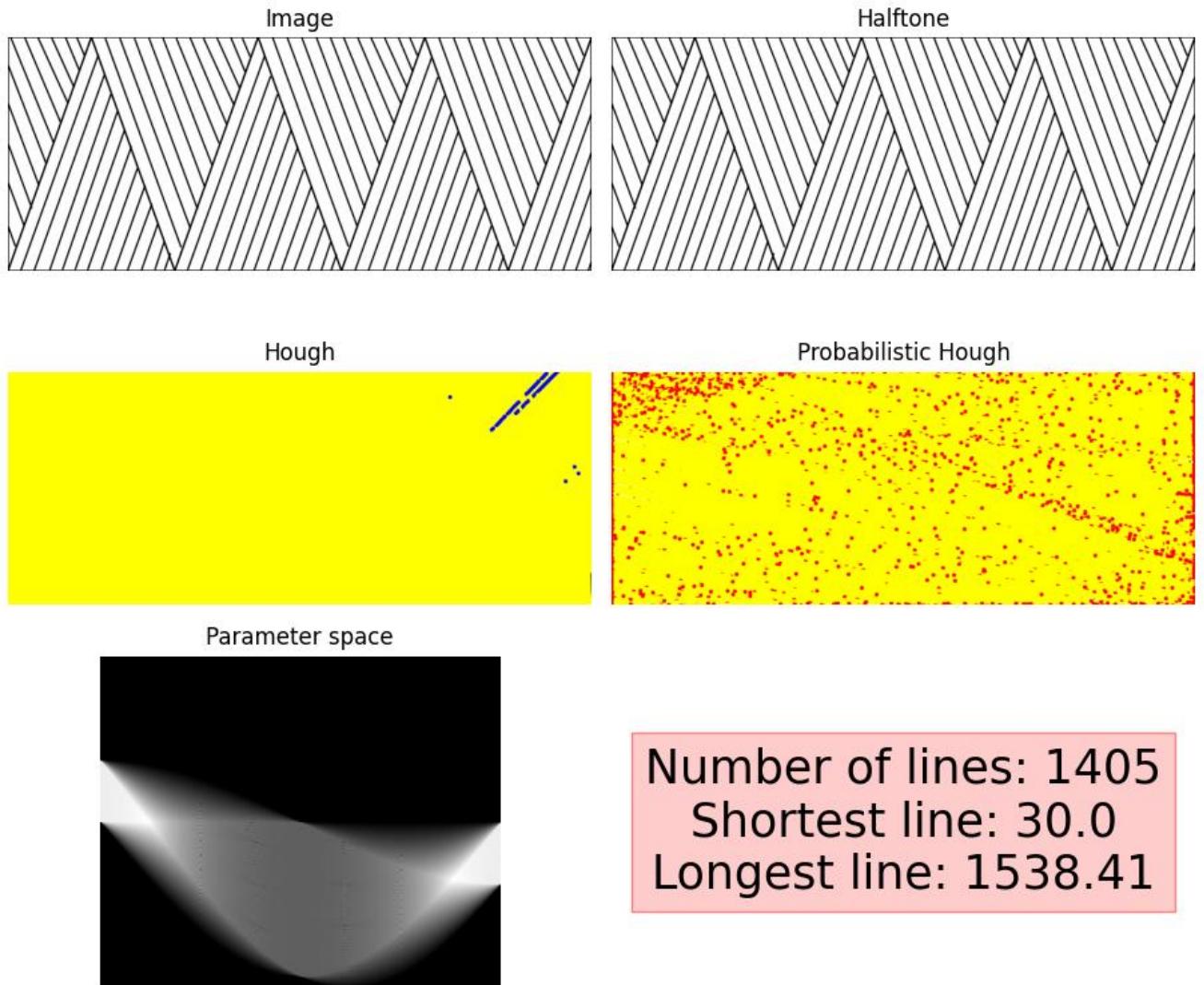


Рис. 8: Исходное изображение 2; полутонаовое изображение; результаты классического и вероятностного преобразованием Хафа; пространство параметров; подсчёт прямых

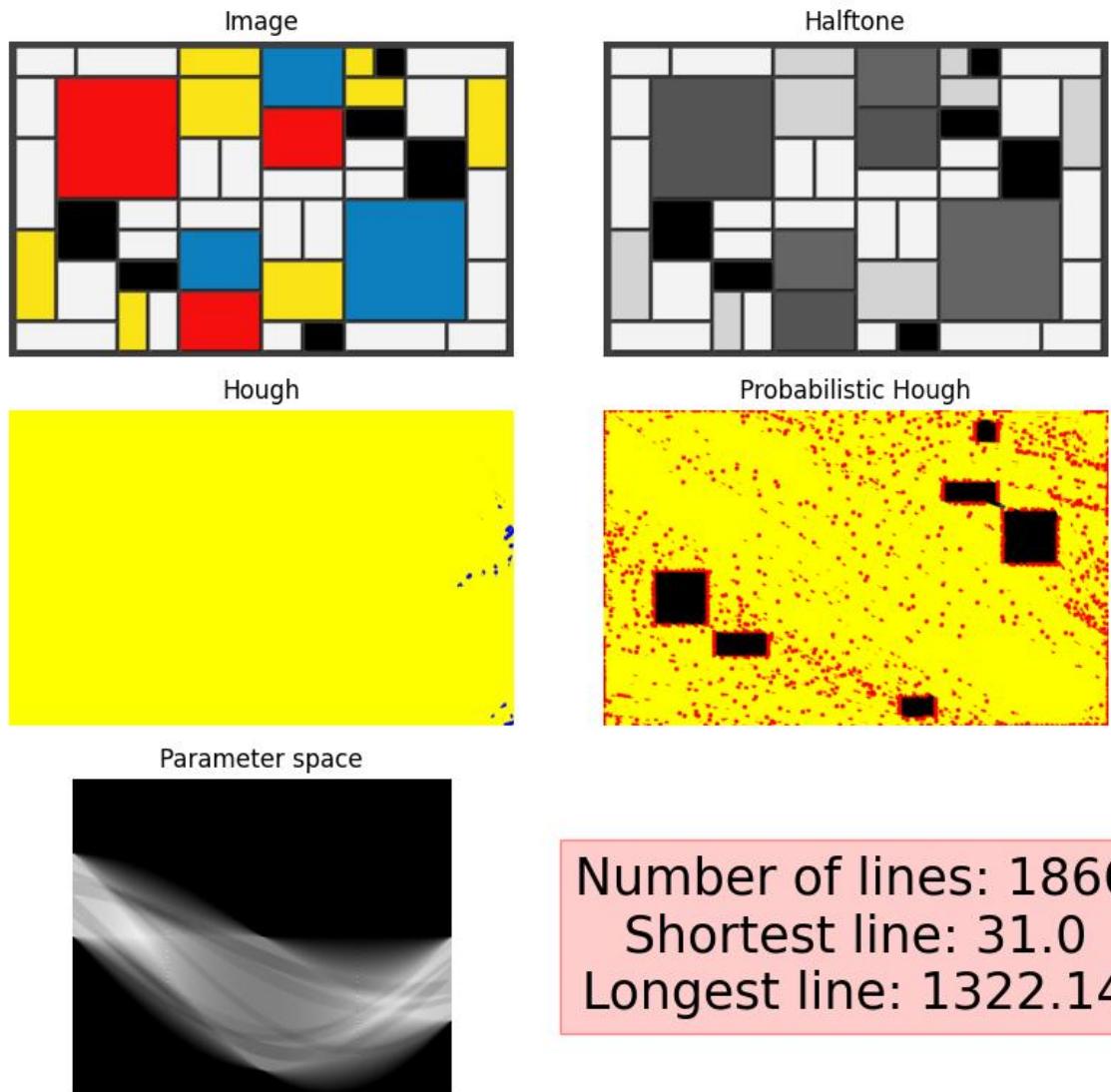


Рис. 9: Исходное изображение 3; полутонаовое изображение; результаты классического и вероятностного преобразованием Хафа; пространство параметров; подсчёт прямых

2. Поиск окружностей

2.1. Исходные изображения

Настало время подобрать изображения, содержащие окружности. Мы, авторы этого отчёта — господа спортивные, поэтому для этого задания мы использовали самый известный символ спортивного движения:

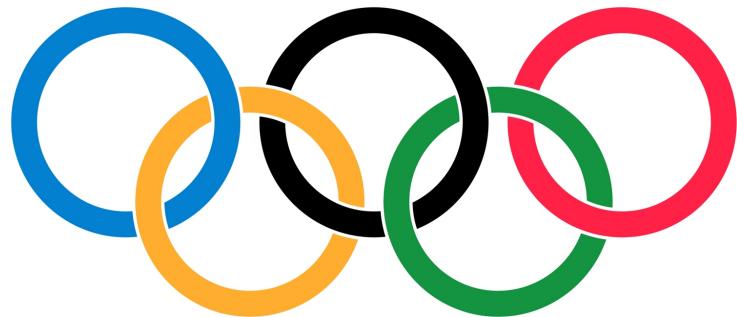


Рис. 10: Олимпийский флаг

Мы не имели никакого права не обратиться к произведениям основоположника абстракционизма и нашего соотечественника — Василия Васильевича Кандинского:



Рис. 11: В.В. Кандинский – Несколько кругов

Последним изображением стала фотография Mercedes-Benz 300 SEL AMG «Красная свинья», ставший ключевым элементом в первом для новой компании AMG триумфе в автогонках:



Рис. 12: Mercedes-Benz 300 SEL AMG «Красная свинья»

2.2. Программа на языке Python

```
def circle_detection(option, image):
    # creating image copies
    result = image.copy()
    result_r = image.copy()
    edged = image
    match option:
        case 0:
            # output of the original image
            cv.imshow('Source Image', image)
            cv.waitKey(0)
        case 1:
            # Hough transform
            # Specified radius
            hough_radius = np.arange(180, 181)
            # Getting Hough accumulator
            hough_res = skimage.transform.hough_circle(edged,
hough_radius)
            # Getting coordinates for centers of circles
            ha, cx, cy, radii =
skimage.transform.hough_circle_peaks(hough_res, hough_radius,
total_num_peaks=1)
            for center_y, center_x, radius in zip(cy, cx, radii):
                # overlaying circles of specified radius
                cv.circle(result, (center_x, center_y), int(radius),
(255, 255, 0), 10, cv.LINE_AA)
                # Radius range
                hough_radius_r = np.arange(175, 239)
```

```

        # Getting Hough accumulator
        hough_res_r = skimage.transform.hough_circle(edged,
hough_radius_r)
            # Getting coordinates for centers of circles
            ha, cx, cy, radii =
skimage.transform.hough_circle_peaks(hough_res_r, hough_radius_r,
total_num_peaks=4)
                for center_y, center_x, radius in zip(cy, cx, radii):
                    # overlaying circles
                    cv.circle(result_r, (center_x, center_y), int(radius),
(255, 255, 0), 10, cv.LINE_AA)
            # transferring received images and data to function of
creating a comparative image
            image_displaying(image, edged, result, result_r,
hough_radius[0], hough_radius_r[0], hough_radius_r[-1])
            # cv.imwrite('results/circles/Mercedes_SP.jpg', result)
            # cv.imwrite('results/circles/Mercedes_R.jpg', result_r)
        case _:
            print('Wrong option! Enter the right number')

```

Листинг 3: Исходный код функции для нахождения и отображения окружностей с помощью преобразования Хафа

```

def image_displaying(source, canny, finale, finale_r, sp, mn, mx):
    # creating figure with mosaic layout
    fig, axs = plt.subplot_mosaic([['src', 'can'], ['fin', 'finr'],
['info', 'infor']], layout='tight',
                                    figsize=(9.2, 7.8))
    # plotting images
    axs['src'].imshow(cv.cvtColor(source, cv.COLOR_BGR2RGB))
    axs['can'].imshow(canny, cmap=matplotlib.cm.gray)
    axs['fin'].imshow(cv.cvtColor(finale, cv.COLOR_BGR2RGB))
    axs['finr'].imshow(cv.cvtColor(finale_r, cv.COLOR_BGR2RGB))
    # turning off axes
    axs['src'].axis('off')
    axs['can'].axis('off')
    axs['fin'].axis('off')
    axs['finr'].axis('off')
    axs['info'].axis('off')
    axs['infor'].axis('off')
    axs['src'].set_title('Image')
    axs['can'].set_title('Canny edges')
    axs['fin'].set_title('Hough (Specified radius)')
    axs['finr'].set_title('Hough (Range)')
    # plotting the specified radius size
    axs['info'].text(0.5, 0.5, f'Radius length:{sp}', size=25,
                    ha='center',
                    va='center',
                    bbox=dict(boxstyle="square",
                              ec=(1., 0.5, 0.5),
                              fc=(1., 0.8, 0.8)
                            )
)

```

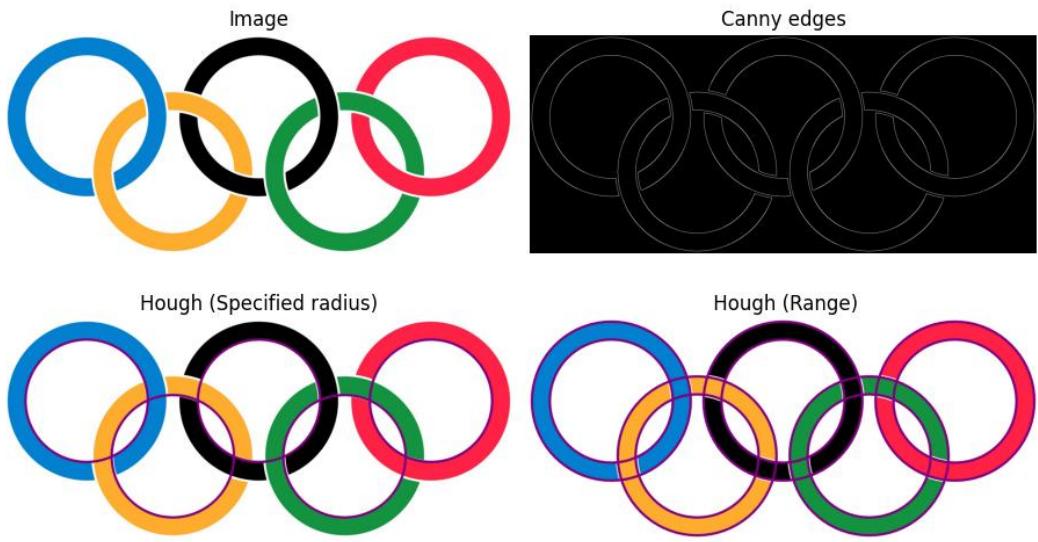
```

# plotting minimum and maximum radius sizes
axs['infor'].text(0.5, 0.5, f'Min radius: {mn}\nMax line: {mx}', size=25,
                  ha='center', va='center',
                  bbox=dict(boxstyle="square",
                            ec=(1., 0.5, 0.5),
                            fc=(1., 0.8, 0.8)
                          )
                )
# displaying and saving the plot
plt.show()
fig.savefig('results/circles/olympic.jpg')
plt.close()

```

Листинг 4: Исходный код функции для создания сравнительного изображения с результатами преобразования

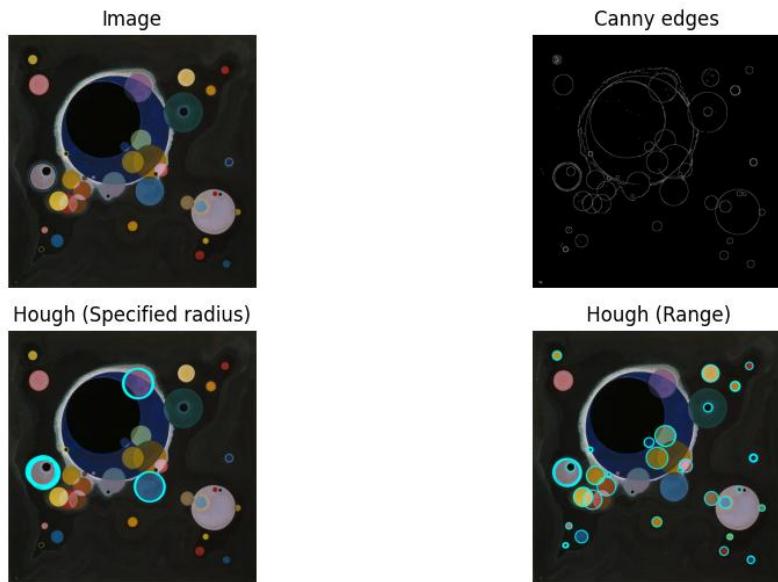
2.3. Результаты преобразования



Radius length:180

Min radius: 175
Max line: 239

Рис. 13: Исходное изображение 1; контуры изображения, полученные алгоритмом Кэнни; результаты преобразования Хафа для определенного радиуса и диапазона радиусов



Radius length:75

Min radius: 10
Max line: 74

Рис. 14: Исходное изображение 2; контуры изображения, полученные алгоритмом Кэнни; результаты преобразования Хафа для определенного радиуса и диапазона радиусов



(a) Результат преобразования Хафа для определенного радиуса окружностей



(b) Результат преобразования Хафа для диапазона радиусов

Рис. 15: Результаты преобразования Хафа

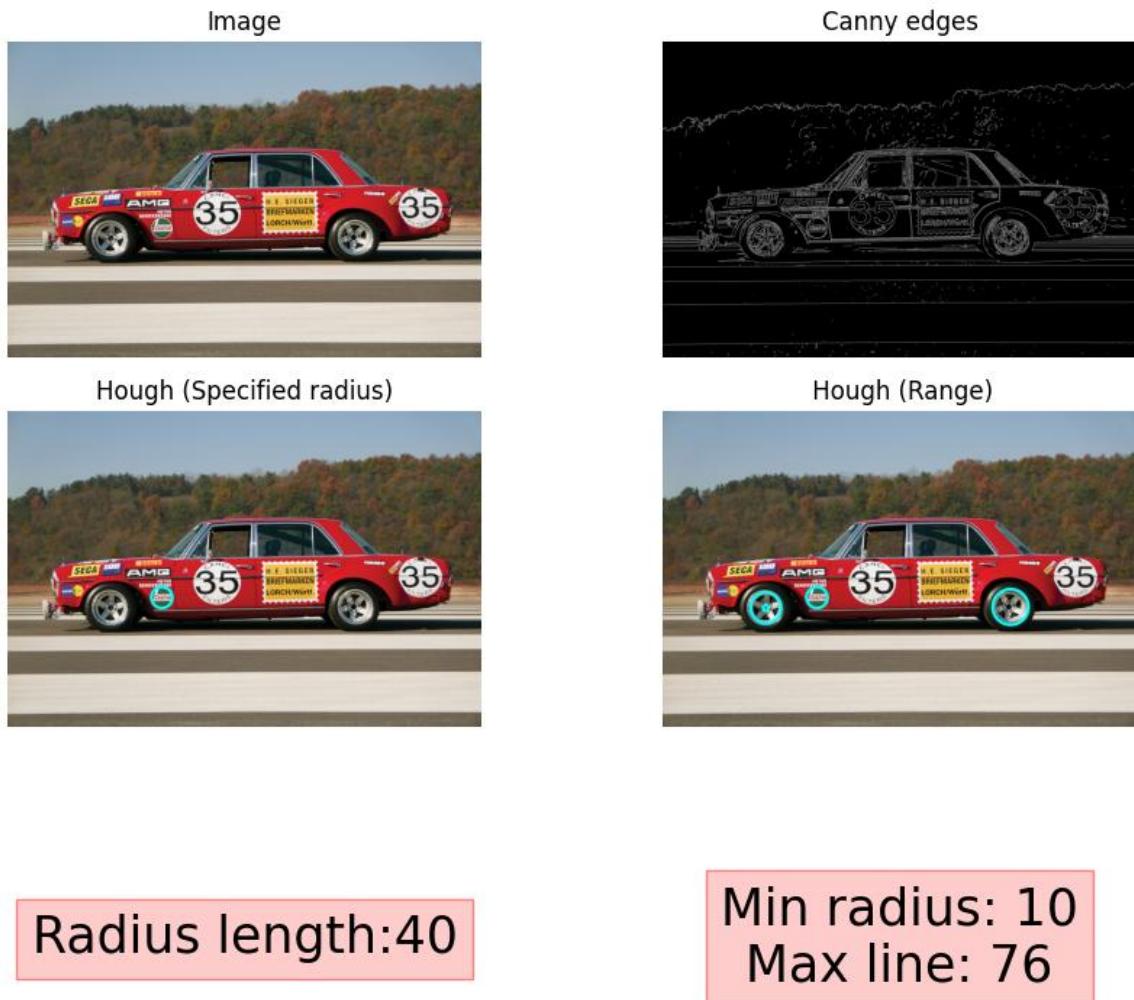


Рис. 16: Исходное изображение 3; контуры изображения, полученные алгоритмом Кэнни; результаты преобразования Хафа для определенного радиуса и диапазона радиусов



(a) Результат преобразования Хафа для определенного радиуса окружностей



(b) Результат преобразования Хафа для диапазона радиусов

Рис. 17: Результаты преобразования Хафа

3. Выводы

В результате выполнения работы мы познакомились с преобразованием Хафа для поиска геометрических примитивов.

Стоит отметить, что важную роль в успешном определении прямых линий и окружностей играет оператор Кэнни, который обнаруживает контуры изображения. Нам удалось убедиться в этом при использовании полутонового изображения для преобразования Хафа: результаты при обнаружении прямых были удручающими (см. рисунки 7, 8 и 9), а при определении окружностей программа не смогла ничего вывести.

Полный код программы и изображения можно найти на [GitHub](#).

4. Ответы на вопросы

Q1. Какая идея лежит в основе преобразования Хафа?

A1. В основе данного преобразования лежит идея поиска общего геометрического места точек (ГМТ) с помощью метода «голосования» точек. В классическом преобразовании также используется идея пространства параметров (θ, ρ) уравнения прямой $y = kx + b \Leftrightarrow x\cos(\theta) + y\sin(\theta) = \rho$

Q2. Можно ли использовать преобразование Хафа для поиска произвольных контуров, которые невозможно описать аналитически?

A2. Решение данной задачи возможно при использовании **обобщенного преобразования Хафа**.

Q3. Что такое рекуррентное и обобщенное преобразования Хафа?

A3. Особенность рекуррентного преобразования Хафа заключается в том, что мы применяем преобразование в скользящем окне, определяя аккумулятор преобразования для каждой области, после чего будем составлять общий аккумуляторный массив. В результате каждая точка общего аккумулятора будет характеризоваться параметрами наиболее достоверного отрезка прямой, проходящего через него.

Обобщенное преобразование Хафа применяется для случая произвольных контуров, не описываемых аналитически. В данном случае функция расстояния от пикселя границы

до центра является функцией $R(\phi)$ от угла ϕ радиус-вектора, направленного от точки контура к центру (точка локализации). При этом ϕ может являться не только углом абсолютного направления на центр, но и относительным углом между направлением градиента и направлением радиуса-вектора.

Q4. Какие бывают способы параметризации в преобразовании Хафа?

A4. Бывают следующие способы параметризации: точки периметра (n, m) сетки изображения, точка периметра и угол (α, n) , Наклон и смещение (α, d) , основание нормали.