

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный Исследовательский Университет ИТМО»

**ЛАБОРАТОРНАЯ РАБОТА №5
ПО ПРЕДМЕТУ «ТЕХНИЧЕСКОЕ ЗРЕНИЕ»
ПО ТЕМЕ «ПРЕОБРАЗОВАНИЕ ХАФА»**

Преподаватель:
Шаветов С. В.

Выполнили:
Румянцев А. А.
Чебаненко Д. А.
Овчинников П. А.

Поток: ТЕХ. ЗРЕНИЕ 2.1
Факультет: СУиР
Группа: R3241

Санкт-Петербург
2024

Содержание

1 Цель работы	2
2 Теоретическое обоснование применяемого преобразования для поиска геометрических примитивов	2
3 Ход выполнения работы	4
3.1 Поиск прямых	4
3.2 Поиск окружностей	6
3.3 Листинги программных реализаций	8
3.4 Дополнительные сведения	11
4 Ответы на вопросы	12
4.1 Какая идея лежит в основе преобразования Хафа?	12
4.2 Можно ли использовать преобразование Хафа для поиска произвольных контуров, которые невозможно описать аналитически?	12
4.3 Что такое рекуррентное и обобщенное преобразования Хафа?	12
4.4 Какие бывают способы параметризации в преобразовании Хафа?	12
5 Выводы о проделанной работе	13

1 Цель работы

Освоение преобразования для поиска геометрических примитивов.

2 Теоретическое обоснование применяемого преобразования для поиска геометрических примитивов

Идея преобразования Хафа заключается в поиске общих геометрических точек (ГМТ).

Данный подход может использоваться для построения треугольника по трем заданным сторонам. Сначала откладывается одна сторона треугольника, после этого концы отрезка рассматриваются как центры окружностей радиусами равными длинам второго и третьего отрезков. Место пересечения двух окружностей является общим ГМТ, откуда проводятся отрезки до концов первого отрезка. Другими словами, было проведено голосование двух точек в пользу вероятного расположения третьей вершины треугольника. В результате «голосования» «победила» точка, набравшая два голоса. Ниже приведен соответствующий рисунок 1.

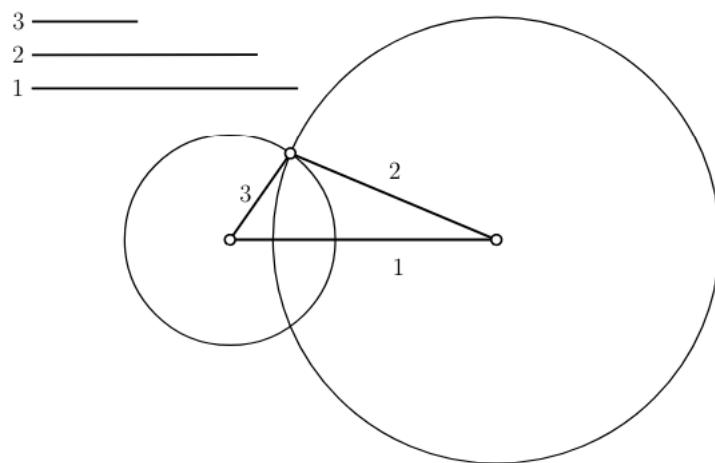


Рис. 1: Построение треугольника по трем заданным сторонам.

Для работы с реальными данными обобщим данную идею, чтобы работать с большим количеством характеристических точек на изображении, участвующих в голосовании. Допустим, необходимо найти в бинарном точечном множестве окружность известного радиуса R , причем в данном множестве могут присутствовать и ложные точки, не лежащие на искомой окружности. Набор центров возможных окружностей искомого радиуса вокруг каждой характеристической точки образует окружность радиуса R . Тогда, точка, соответствующая максимальному пересечению числа окружностей, и будет являться центром окружности искомого радиуса. Ниже приведен соответствующий рисунок 2.

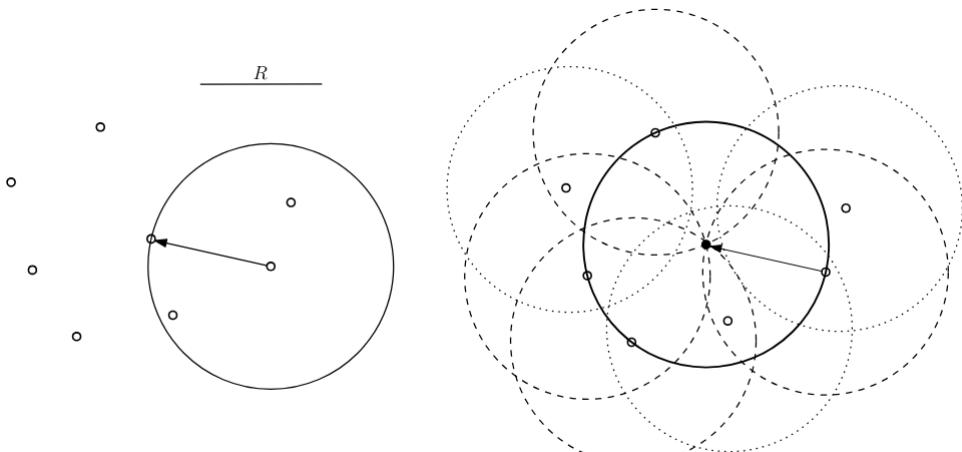


Рис. 2: Обнаружение окружности известного радиуса в точечном множестве.

В данной лабораторной работе используется классическое преобразование Хафа, базирующееся на рассмотренной идее голосования точек. Преобразование изначально было предназначено для выделения прямых на бинарных изображениях. В преобразовании Хафа для поиска геометрических примитивов используется пространство параметров. Самое распространенное параметрическое уравнение прямых приведено ниже.

$$y = kx + b, \quad (1)$$

$$x \cos \theta + y \sin \theta = \rho, \quad (2)$$

где ρ – радиус-вектор, проведенный из начала координат до прямой; θ – угол наклона радиус-вектора.

Пусть в декартовой системе координат прямая задана уравнением (1), из которого легко вычислить радиус-вектор ρ и угол θ (2). Тогда в пространстве параметров Хафа прямая будет представлена точкой с координатами (ρ_0, θ_0) . Ниже представлен соответствующий рисунок 3.

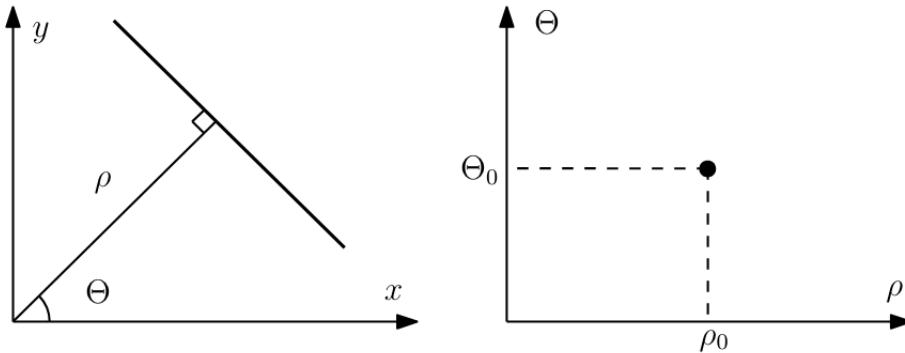


Рис. 3: Представление прямой в пространстве Хафа.

Подход преобразования Хафа заключается в суммировании для каждой точки пространства количества поданных за нее голосов. Вследствие этого в дискретном виде пространство Хафа называется *аккумулятором* и представляет собой некую матрицу $A(\rho, \theta)$, хранящую информацию о голосовании. В декартовой системе координат через каждую точку можно провести бесконечное число прямых. Их совокупность породит в пространстве параметров синусоидальную функцию отклика. Таким образом, любые две синусоидальные функции отклика в пространстве параметров пересекутся в точке (ρ, θ) , только если порождающие их точки в исходном пространстве лежат на прямой. Выходит, что для поиска прямых в исходном пространстве необходимо найти все локальные максимумы аккумулятора. Ниже представлен соответствующий рисунок 4.

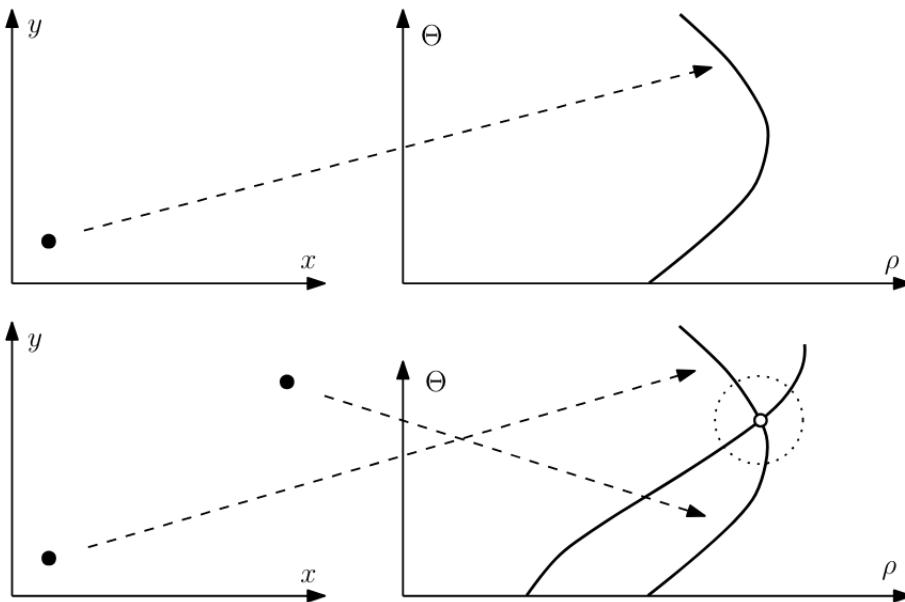


Рис. 4: Процедура голосования.

Рассмотренный алгоритм является универсальным. Его можно использовать для поиска любой другой кривой, описываемой в пространстве некоторой функцией с определенным числом параметров $F = (a_1, a_2, \dots, a_n, x, y)$, что влияет только на размерность пространства параметров.

Воспользуемся преобразованием Хафа для поиска окружностей заданного радиуса R . Окружность на плоскости описывается формулой $(x - x_0)^2 + (y - y_0)^2 = R^2$. Набор центров всех возможных окружностей радиуса R , проходящих через характеристическую точку, образует окружность радиуса R вокруг этой точки. Вследствие этого функция отклика в преобразовании Хафа для поиска окружностей представляет окружность такого же размера с центром в голосующей точке. Таким образом, аналогично предыдущему случаю необходимо найти локальные максимумы аккумуляторной функции $A(x, y)$ в пространстве параметров (x, y) , которые и будут являться центрами искомых окружностей.

Преобразование Хафа позволяет обнаруживать линии инвариантно не только к аффинным преобразованиям плоскости, но и к группе проективных преобразований в пространстве, так как преобразование инвариантно к сдвигу, масштабированию и повороту, при этом при проективных преобразованиях трехмерного пространства прямые линии всегда переходят только в прямые линии (в вырожденном случае – в точки).

3 Ход выполнения работы

3.1 Поиск прямых

Выберем три произвольных изображения, содержащих прямые.



(a) Штрихкод



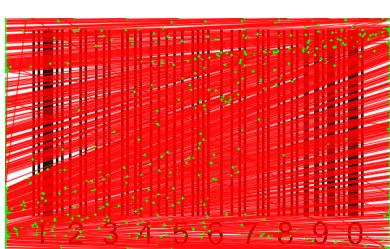
(b) Архитектура



(c) Интерьер

Рис. 5: Произвольные изображения, содержащие прямые.

Попробуем сразу применить преобразование Хафа без использования каких-либо дифференциальных операторов. Сначала преобразуем изображение к оттенкам серого, далее применим Хафа и нарисуем линии на оригинальном изображении.



(a) Штрихкод



(b) Архитектура



(c) Интерьер

Рис. 6: Преобразование Хафа на выбранных изображениях.

Исходя из результатов сделан вывод, что рассматривать преобразование Хафа без какой-либо подготовки изображения не нужно, так как линии вышли хаотичными и не соответствуют ожиданиям. Их исследование не внесет пользы в общий результат выполнения работы. Предположительно, шум, не достаточно четкие границы объектов и небинарное представление картинки привели к соответствующим изображениям.

Обрабатываем изображения алгоритмом Кэнни – оператором обнаружения границ изображения. Основными этапами алгоритма являются сглаживание изображения (для удаления шума), поиск градиентов (границы отмечаются там, где градиент изображения приобретает максимальное значение), подавление немаксимумов (только локальные максимумы отмечаются как границы), двойная пороговая фильтрация (потенциальные границы определяются порогами) и трассировка области неоднозначности (итоговые границы определяются путём подавления всех краёв, не связанных с определенными (сильными) границами). Перед обработкой изображение было преобразовано в оттенки серого, чтобы уменьшить вычислительные затраты.



(а) Штрихкод



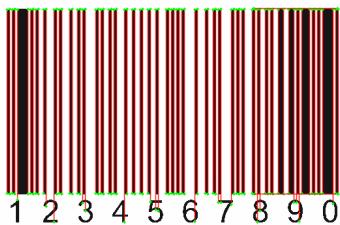
(б) Архитектура



(с) Интерьер

Рис. 7: Алгоритм Кэнни на выбранных изображениях.

Преобразуем изображения алгоритмом Хафа и получим результат, представленный на рисунке 8. Найденные линии отрисованы красным цветом, точки начала и конца каждой линии отмечены зеленым цветом. Ниже каждой картинки написаны длина в пикселях самой длинной и самой короткой линий, а также количество отрисованных линий.



(а) Штрихкод

Максимальная длина линии: 325р
Минимальная длина линии: 129р
Всего линий: 88



(б) Архитектура

Максимальная длина линии: 438р
Минимальная длина линии: 20р
Всего линий: 160



(с) Интерьер

Максимальная длина линии: 793р
Минимальная длина линии: 110р
Всего линий: 43

Рис. 8: Алгоритм Хафа на выбранных обработанных изображениях.

Для штрихкода линии были найдены почти идеально – все вертикальные линии отображены, однако есть погрешности в некоторых из них – линии залезают на числа ниже черных полос. Также появились лишние горизонтальные линии. Для архитектуры линии были найдены неплохо, были определены верхние контуры зданий, а также контуры окон, балконов и некоторых полос на самих зданиях. Кроме того выделены контуры фар, автомобильных окон, номерных знаков и бамперов. Для интерьера алгоритм Хафа справился хорошо – большинство линий в точности повторяют прямоугольные контуры окон в дверях, какие то выбросы по типу линий «за окнами» или «на окнах» отсутствуют.

На рисунке 9 для каждого исходного изображения отображено пространство признаков преобразования Хафа. На оси OX находятся углы в градусах, на OY расстояние в пикселях. Исходя из результатов можно сделать вывод, что для штрихкода большинство линий сосредоточено в диапазонах $[-100^\circ, -60^\circ]$ и $[60^\circ, 100^\circ]$. Для архитектуры линии хорошо просматриваются при примерно $-100^\circ, -80^\circ, 80^\circ, 100^\circ$. Для интерьера больше всего линий сконцентрировано в диапазоне $[60^\circ, 100^\circ]$.

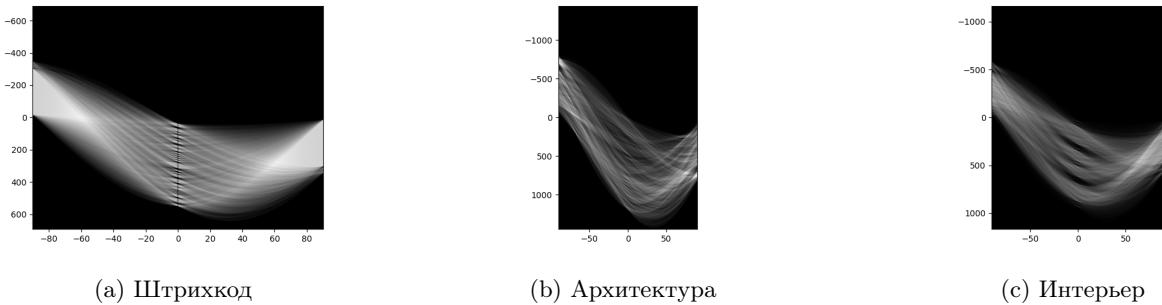


Рис. 9: Пространство параметров для каждого исходного изображения.

3.2 Поиск окружностей

Выберем три произвольных изображения, содержащих окружности.



Рис. 10: Произвольные изображения, содержащие окружности.

Попробуем сразу применить преобразование Хафа без использования каких-либо дифференциальных операторов. Сначала преобразуем изображение к оттенкам серого, далее применим Хафа и нарисуем окружности на оригинальном изображении.

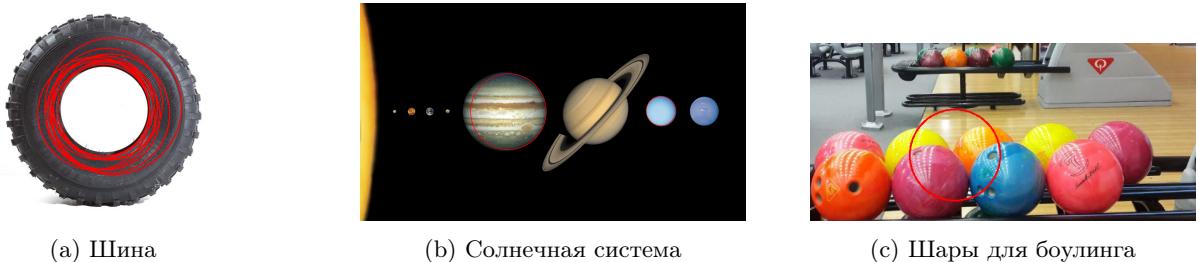
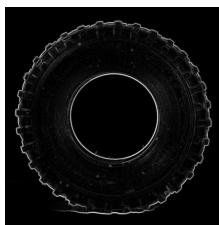


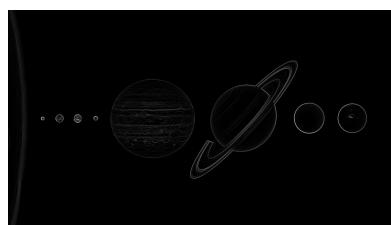
Рис. 11: Преобразование Хафа на выбранных изображениях.

Результаты здесь немного лучше, чем результаты поиска линий. Для шины были найдены окружности, которые походят на правду, но ничего конкретного не выделяют, хорошо получился только контур отверстия шины, хоть и приближенно. В солнечной системе удалось выделить две окружности – контур Юпитера и Урана. Однако стоит помнить, что планеты в большинстве своем представляются эллипсоидами, поэтому получить точный контур того же Юпитера не получится. Для шаров для боулинга результат плохой – не выделен ни один шар, есть какая-то окружность левее середины, не выделяющая никакой контур. Предположительно, неплохие результаты для первых двух изображений получились вследствие заметного контраста между объектами, которые нам хотелось бы выделить, и фоном, что не сказать про третью картинку.

Обработаем изображения алгоритмом Собеля – дискретным дифференциальным оператором, вычисляющим приближённое значение градиента яркости изображения. Оператор основан на свёртке изображения небольшими сепарабельными целочисленными фильтрами в вертикальном и горизонтальном направлениях, поэтому его относительно легко вычислять, однако используемая им аппроксимация градиента достаточно грубая. В общем результат обработки очень похож на результат применения алгоритма Кэнни и представлен на рисунке 12.



(a) Шина



(b) Солнечная система



(c) Шары для боулинга

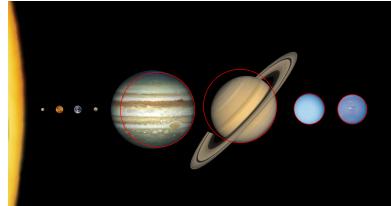
Рис. 12: Алгоритм Собеля на выбранных изображениях.

Преобразуем изображения алгоритмом Хафа и получим результат, представленный на рисунке 13. Найденные окружности отрисованы красным цветом. Ниже каждой картинки написаны радиус в пикселях самой большой и самой маленькой окружностей, а также количество отрисованных окружностей.



(a) Шина

Макс. радиус окружности: 231р
Мин. радиус окружности: 107р
Всего окружностей: 2



(b) Солнечная система

Макс. радиус окружности: 185р
Мин. радиус окружности: 73р
Всего окружностей: 4



(c) Шары для боулинга

Макс. радиус окружности: 66р
Мин. радиус окружности: 34р
Всего окружностей: 6

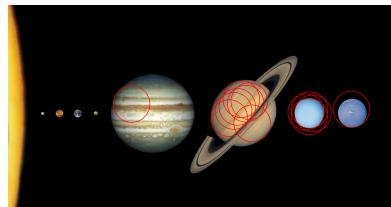
Рис. 13: Алгоритм Хафа на выбранных обработанных изображениях.

Для шины были найдены две окружности, повторяющие контур отверстия и внешней части, никаких лишних выбросов не обнаружено. В солнечной системе стало отрисовано на две планеты больше – были захвачены Сатурн и Нептун. Для шаров для боулинга отрисовались контуры четырех шаров, однако есть две окружности, которые можно назвать выбросами. Причина их появления заключается в не очень хорошей отрисовке контуров после применения алгоритма Собеля к исходному изображению – на рисунке 12 видно, что на третьем изображении намного лучше выделено окружение шаров, чем они сами. Тем не менее алгоритм Хафа достаточно хорошо справился с поиском окружностей.

Поиск окружностей радиуса 90 пикселей алгоритмом Хафа для исходных изображений без дифференциальных преобразований представлен на рисунке 14.



(a) Шина



(b) Солнечная система



(c) Шары для боулинга

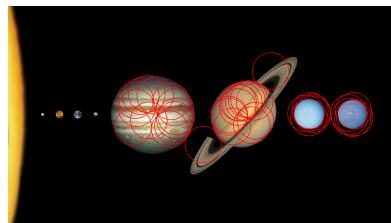
Рис. 14: Поиск окружностей радиуса 90 пикселей алгоритмом Хафа на исходных изображениях.

Как видим, алгоритм отработал верно – все окружности выглядят одинаково, а значит одного радиуса.

Поиск окружностей радиуса 90 пикселей алгоритмом Хафа для исходных изображений с алгоритмом Собеля представлен на рисунке 15. Алгоритм отработал аналогично, однако окружностей стало больше, за исключением случая с шиной.



(a) Шина



(b) Солнечная система



(c) Шары для боулинга

Рис. 15: Поиск окружностей радиуса 90 пикселей алгоритмом Хафа на обработанных изображениях.

3.3 Листинги программных реализаций

Условимся, что кодом ниже мы считываем изображение и преобразуем его к серым тонам всегда при использовании алгоритма Хафа.

```

1 render_to = 'renders'
2 src_dir = 'img'
3 curr_img = 'ci3.png'
4 filename = f'{src_dir}/{curr_img}'
5 clr_src = cv2.imread(cv2.samples.findFile(filename), cv2.IMREAD_COLOR)
6 src = cv2.cvtColor(clr_src, cv2.COLOR_BGR2GRAY)

```

Листинг 1: Считывание изображения и преобразование к серым тонам.

Для начала рассмотрим реализацию алгоритма Хафа для линий. Для поиска линий используется метод `detect_lines`, в котором вызывается `HoughLinesP` – вероятностное преобразование Хафа – с передаваемыми параметрами:

1. **image** – Изображение (в бинарном представлении или хотя бы в оттенках серого).
2. **rho** – Разрешение параметра r в пикселях.
3. **theta** – Разрешение параметра θ в радианах.
4. **threshold** – Минимальное количество пересечений для «обнаружения» линии.
5. **minLineLength** – Минимальное количество точек, которые могут образовывать линию.
6. **maxLineGap** – Максимальный разрыв между двумя точками, которые считаются на одной линии.

Метод `draw_lines` в цикле бегает по линиям и отрисовывает их на переданном в метод изображении по координатам начала и конца линии. Также метод отрисовывает точки.

```

1 def detect_lines(src, rho, theta, threshold, minLineLength, maxLineGap):
2     lines = cv2.HoughLinesP(
3         image=src,
4         rho=rho,
5         theta=theta,
6         threshold=threshold,
7         lines=None,
8         minLineLength=minLineLength,
9         maxLineGap=maxLineGap
10    )
11    return lines
12
13 def draw_lines(src, lines, thickness=1, color=(0, 0, 255),
14                 marker_radius=2, dot_color1=(0, 255, 0), dot_color2=(0, 255, 0)):
15     if lines is not None:
16         for i in range(0, len(lines)):
17             l = lines[i][0]
18             x_1, y_1, x_2, y_2 = l[0], l[1], l[2], l[3]
19             cv2.line(src, (x_1, y_1), (x_2, y_2), color, thickness, cv2.LINE_AA)
20             cv2.circle(src, (x_1, y_1), marker_radius, dot_color1, -1)
21             cv2.circle(src, (x_2, y_2), marker_radius, dot_color2, -1)
22
23 return src

```

Листинг 2: Алгоритм Хафа для поиска линий и отрисовки их на изображении.

Для отрисовывания пространства параметров использовался код, представленный на листинге 3.

```

1 def parameters_space_lines(src, arr_ang_step=0.1, brightness=3):
2     dots = int(round(360 / arr_ang_step))
3     angles = np.linspace(-np.pi / 2, np.pi / 2, dots, endpoint=False)
4     H, theta, rho = skimage.transform.hough_line(src, theta=angles)
5
6     ang_step = 0.5 * np.diff(theta).mean()
7     dis_step = 0.5 * np.diff(rho).mean()
8
9     bounds = [np.rad2deg(theta[0] - ang_step),
10             np.rad2deg(theta[-1] + ang_step),
11             rho[-1] + dis_step, rho[0] - dis_step]
12     parameters_space = cv2.cvtColor(np.float32(brightness * H / np.max(H)),
13                                     cv2.COLOR_GRAY2RGB)
14
15     return parameters_space, bounds

```

Листинг 3: Программа для отрисовки пространства параметров для линий.

Для поиска максимальной и минимальной длины линии был написан код представленный на листинге 4.

```

1 def min_max_len_lines(lines):
2     min_len = float('inf')
3     max_len = 0
4     min_len_line = None
5     max_len_line = None
6
7     if lines is not None:
8         for line in lines:
9             x1, y1, x2, y2 = line[0]
10
11         line_len = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
12         if line_len > max_len:
13             max_len = line_len
14             max_len_line = line
15         if line_len < min_len:
16             min_len = line_len
17             min_len_line = line
18
19     return min_len, min_len_line, max_len, max_len_line

```

Листинг 4: Поиск максимальной и минимальной длины линии.

Пример использования кода с предыдущих листингов расположен на листинге 5.

```

1 rho = 1
2 theta = np.pi / 180
3 threshold = 130
4 threshold1 = 350
5 threshold2 = 230
6 apertureSize = 3
7 minLineLength = 20
8 maxLineGap = 5
9
10 edge_src = cv2.Canny(src, threshold1, threshold2, None, apertureSize)
11 lines_edge_src = detect_lines(edge_src, rho, theta,
12                                 threshold, minLineLength, maxLineGap)
13 output_edge_src = draw_lines(clr_src.copy(), lines_edge_src)
14 ps_pace, bds = parameters_space_lines(edge_src)
15
16 edge_min_len, edge_min_len_1, edge_max_len, edge_max_len_1 = min_max_len_lines(
17     lines_edge_src)
18 edge_lines_count = len(lines_edge_src)
19
20 cv2.imwrite(f'{render_to}/canny_{curr_img}', edge_src)
21 plt.imshow(ps_pace, extent=bds, aspect=0.1)
22 plt.savefig(f'{render_to}/canny_par_space_{curr_img}')
23 cv2.imwrite(f'{render_to}/canny_hl_{curr_img}', output_edge_src)
24 print(f'canny_max_len_line={edge_max_len}p\n'
25       f'canny_min_len_line={edge_min_len}p\n'
26       f'canny_lines_count={edge_lines_count}')

```

Листинг 5: Пример использования программы для поиска линий и построения пространства параметров.

Для окружностей реализация почти такая же, но пропадают и добавляются некоторые параметры:

1. **method** – Метод преобразования Хафа «Градиент», который используется для обнаружения окружностей.

2. **dp** – Параметр, который определяет коэффициент разреженности аккумуляторной матрицы преобразования Хафа.
3. **minDist** – Минимальное расстояние между центрами обнаруженных окружностей.
4. **param1** – Параметр для определения порогового значения для детекции границ окружностей на изображении.
5. **param2** – Параметр для определения минимального числа голосов, которые должна получить окружность, чтобы она была признана окружностью.
6. **minRadius** – Минимальный радиус ожидаемой окружности.
7. **maxRadius** – Максимальный радиус ожидаемой окружности.

Метод `draw_circles` бегает по окружностям и рисует их на передаваемом изображении с помощью координат центра окружности `i[0], i[1]` и радиуса окружности `i[2]`.

```

1  def detect_circles(src, dp, min_dis, par1, par2, min_r, max_r):
2      circles = cv2.HoughCircles(
3          image=src,
4          method=cv2.HOUGH_GRADIENT,
5          dp=dp,
6          minDist=min_dis,
7          param1=par1,
8          param2=par2,
9          minRadius=min_r,
10         maxRadius=max_r
11     )
12     return circles
13
14 def draw_circles(src, circs, color=(0, 0, 255), thickness=2):
15     if circs is not None:
16         circles = np.uint16(np.around(circs))
17         for i in circles[0, :]:
18             cv2.circle(src, (i[0], i[1]), i[2], color, thickness)
19     return src

```

Листинг 6: Алгоритм Хафа для поиска и отрисовки окружностей.

Также написан метод для поиска минимального и максимального радиусов найденных окружностей.

```

1  def min_max_radius_circles(circles):
2      min_radius = float('inf')
3      max_radius = 0
4      min_circle = None
5      max_circle = None
6
7      if circles is not None:
8          circles = np.uint16(np.around(circles))
9          for i in circles[0, :]:
10              if i[2] < min_radius:
11                  min_radius = i[2]
12                  min_circle = i
13              if i[2] > max_radius:
14                  max_radius = i[2]
15                  max_circle = i
16
17      return min_radius, min_circle, max_radius, max_circle

```

Листинг 7: Поиск минимального и максимального радиуса найденных окружностей.

Пример использования методов, описанных в предыдущих листингах, расположен на листинге 8.

```

1  sobelx = cv2.Sobel(src, cv2.CV_64F, 1, 0, ksize=5)
2  sobely = cv2.Sobel(src, cv2.CV_64F, 0, 1, ksize=5)
3  edge_src = np.hypot(sobelx, sobely)
4  edge_src = np.uint8(edge_src / np.max(edge_src) * 255)
5
6  dp = 1
7  min_dis = 10
8  par1 = 40
9  par2 = 10
10 min_r = 30
11 max_r = 95
12 single_r = 90
13
14 circles_edge_src = detect_circles(edge_src, dp, min_dis, par1, par2, min_r, max_r)
15 output_edge_src = draw_circles(clr_src.copy(), circles_edge_src)

```

```

16
17     edge_min_radius, edge_min_circle, edge_max_radius, edge_max_circle =
18         min_max_radius_circles(circles_edge_src)
19     edge_circs_count = len(circles_edge_src[0])
20
21     circles_edge_src2 = detect_circles(edge_src, dp, min_dis, par1, par2, single_r, single_r)
22     output_edge_src2 = draw_circles(clr_src.copy(), circles_edge_src2)
23
24     cv2.imwrite(f'{render_to}/canny_{curr_img}', edge_src)
25     cv2.imwrite(f'{render_to}/canny_hc_{curr_img}', output_edge_src)
26     cv2.imwrite(f'{render_to}/canny_hc_r={single_r}_{curr_img}', output_edge_src2)
27     print(f'min_radius={edge_min_radius}p\nmax_radius={edge_max_radius}p\ncircles_count={edge_circs_count}\n')

```

Листинг 8: Пример применения алгоритма Хафа для поиска окружностей.

3.4 Дополнительные сведения

Для штрихкода использовались следующие параметры:

1. **image=src**
2. **rho=1**
3. **theta=np.pi / 180**
4. **threshold=50**
5. **threshold1=50**
6. **threshold2=200**
7. **apertureSize=3**
8. **minLineLength=50**
9. **maxLineGap=10**

Для архитектуры использовались следующие параметры:

1. **image=src**
2. **rho=1**
3. **theta=np.pi / 180**
4. **threshold=130**
5. **threshold1=350**
6. **threshold2=230**
7. **apertureSize=3**
8. **minLineLength=20**
9. **maxLineGap=5**

Для интерьера использовались следующие параметры:

1. **image=src**
2. **rho=1**
3. **theta=np.pi / 180**
4. **threshold=150**
5. **threshold1=300**
6. **threshold2=110**
7. **apertureSize=3**
8. **minLineLength=110**
9. **maxLineGap=5**

Для шины использовались следующие параметры:

1. **dp=1**
2. **minDist=10**
3. **param1=100**
4. **param2=250**
5. **minRadius=30**
6. **maxRadius=250**

Для солнечной системы использовались следующие параметры:

1. $dp=1$
2. $minDist=180$
3. $param1=50$
4. $param2=100$
5. $minRadius=10$
6. $maxRadius=300$

Для шины использовались следующие параметры:

1. $dp=1$
2. $minDist=55$
3. $param1=100$
4. $param2=50$
5. $minRadius=30$
6. $maxRadius=95$

4 Ответы на вопросы

4.1 Какая идея лежит в основе преобразования Хафа?

Идея преобразования Хафа заключается в преобразовании точек изображения в параметрическое пространство, где каждая фигура представлена уникальным набором параметров. Путем подсчета голосов (аккумуляции) определяются пики в пространстве параметров, которые соответствуют возможным фигурам на изображении, позволяя обнаруживать линии, окружности и другие геометрические формы.

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

Преобразование Хафа может быть использовано для поиска произвольных контуров, которые невозможно описать аналитически. Это достигается путем создания соответствующего пространства параметров из угловых координат и/или радиусов, которые представляют форму необходимого контура. Затем применяется подсчет голосов для каждой возможной формы в этом пространстве, позволяя выявить пики, которые соответствуют контурам на изображении.

4.3 Что такое рекуррентное и обобщенное преобразования Хафа?

Рекуррентное преобразование Хафа – одификация преобразования Хафа, которая позволяет эффективно работать с более сложными формами, такими как кривые или произвольные формы. В отличие от классического преобразования Хафа, которое обрабатывает только фиксированные формы (например, прямые линии и окружности), рекуррентное преобразование Хафа использует итеративный процесс для поиска более сложных форм путем комбинирования простых форм.

Обобщенное преобразование Хафа – расширение преобразования Хафа, которое позволяет обнаруживать произвольные формы без заранее определенных параметрических уравнений. Вместо этого, ГНТ представляет каждую точку на изображении как объект, который может быть частью произвольной формы. Затем алгоритм просматривает все возможные комбинации точек, чтобы найти соответствующие формы. ГНТ более гибок и может использоваться для обнаружения различных форм, включая те, которые сложно описать с помощью аналитических уравнений.

4.4 Какие бывают способы параметризации в преобразовании Хафа?

В лабораторной работе были рассмотрены способы параметризации только для линий и окружностей. В общем бывают следующие виды параметризации:

Для прямых линий:

1. Угловая и дистанционная (угловая координата и расстояние до начала координат).

2. Угловая и смещение (угловая координата и смещение относительно оси координат).

Для окружностей:

1. Координаты центра и радиус окружности.

Для эллипсов:

1. Координаты центра, большая и малая полуоси, угол наклона эллипса.

Для общих кривых:

1. Параметризация Безье, используемая для описания кривых Безье.

2. Параметризация Б-сплайнов, которая позволяет описывать более сложные кривые.

Для произвольных форм:

1. В обобщенном преобразовании Хафа (GHT) каждая точка изображения рассматривается как параметр, и произвольные формы определяются с использованием комбинации этих точек.

5 Выводы о проделанной работе

В ходе данной лабораторной работы были рассмотрены преобразования Хафа для поиска прямых линий и окружностей на изображении. Было выяснено, что перед применением алгоритма Хафа необходимо подготовить изображение – как минимум сделать в оттенках серого и дифференциальным оператором выделить контуры. В ином случае нет гарантии, что алгоритм сработает как нужно – все зависит от конкретного изображения. Кроме того, вследствие высокой ресурсоемкости алгоритма изображение лучше сжать для дальнейшего исследования, иначе будет потрачено много времени, а результат может оказаться не подходящим.