Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Отчет по лабораторной работе №6
«Преобразование Хафа»
по дисциплине «Техническое зрение»

Выполнил: студент гр. R3338,
Кирбаба Д.Д.
Преподаватель: Шаветов С.В.,
канд. техн. наук, доцент ФСУ и Р

Санкт-Петербург, 2022

Цель работы

Исследование трансформации для нахождения геометрических примитивов.

Теоретическое обоснование применяемых методов

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

Например, данный подход используется при построении треугольника по трем заданным сторонам, когда 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 и проводятся отрезки до концов первого отрезка. Иными словами, можно сказать, что было проведено голосование двух точек в пользу вероятного расположения третьей вершины треугольника. В результате «голосования» «победила» точка, набравшая два «голоса» (точки на окружностях набрали по одному голосу, а вне их — по нулю).

Обобщим данную идею для работы с реальными данными, когда на изображении имеется большое количество особых характеристических точек, 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.

Таким образом, точка, соответствующая максимальному пересечению числа окружностей, и будет являться центром окружности искомого радиуса.

Классическое преобразование Хафа, базирующееся на рассмотренной идее голосования точек, изначально было предназначено для выделения прямых на бинарных изображениях. В преобразовании Хафа для поиска геометрических примитивов используется пространство параметров. Самым распространенным параметрическим уравнением прямых является:

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

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

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. Тогда в пространстве параметров Хафа прямая будет представлена точкой с координатами (ρ_0, θ_0).

Подход преобразования Хафа заключается в том, что для каждой точки пространства параметров суммируется количество голосов, поданных за нее, поэтому в дискретном виде пространство Хафа называется аккумулятором и представляет собой некоторую матрицу $A(\rho, \theta)$, хранящую

информацию о голосовании. Через каждую точку в декартовой системе координат можно провести бесконечное число прямых, совокупность которых породит в пространстве параметров синусоидальную функцию отклика.

Таким образом, любые две синусоидальные функции отклика в пространстве параметров пересекутся в точке (ρ, θ) только в том случае, если порождающие их точки в исходном пространстве лежат на прямой. Исходя из этого можно сделать вывод, что для того, чтобы найти прямые в исходном пространстве, необходимо найти все локальные максимумы аккумулятора.

Рассмотренный алгоритм поиска прямых может быть таким же образом использован 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 повлияет лишь на размерность пространства параметров.

Воспользуемся преобразованием Хафа для поиска окружностей заданного радиуса R. Известно, что 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, поэтому функция отклика в преобразовании Хафа для поиска окружностей представляет окружность такого же размера с центром в голосующей точке. Тогда аналогично предыдущему случаю необходимо найти локальные максимумы аккумуляторной функции $A(x, y)$ в пространстве парамаетров (x, y), которые и будут являться центрами искомых окружностей.

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

Ход выполнения работы

1. Поиск прямых

Исходные изображения:

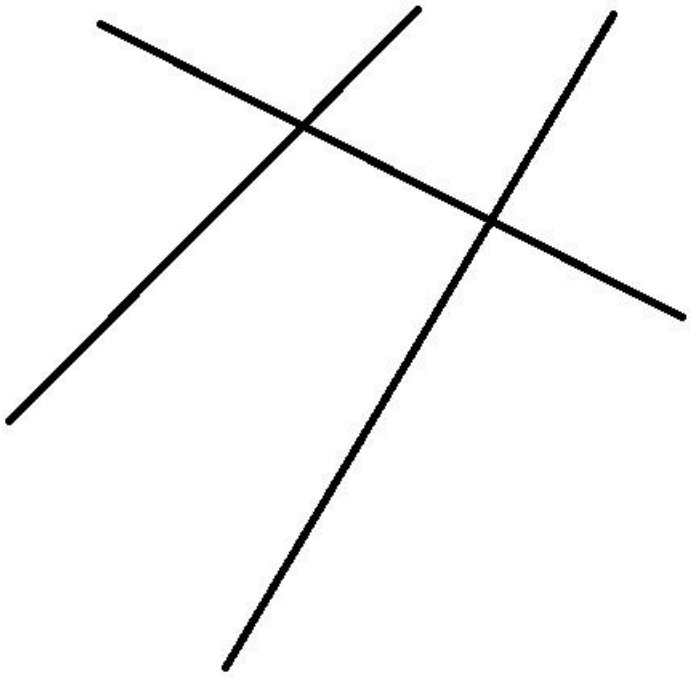

Figure 1: исходное изображение №1.

Figure 2: исходное изображение №2.

Figure 3: исходное изображение №3.

Найдем прямые линии на данных изображениях с помощью преобразования Хафа.

Listing 1. Поиск прямых линий с помощью вероятностного преобразования Хафа на Python.

```
# Apply progressive probabilistic Hough transform for finding lines
lines_prob = cv.HoughLinesP(image=img, rho=1, theta=np.pi / 180, threshold=75,
minLineLength=75, maxLineGap=5)
```

Функция *HoughLinesP* – производит робастный поиск прямых линий используя прогрессивное вероятностное преобразования Хафа. Алгоритм описан в следующей публикации Jiri Matas, Charles Galambos, and Josef Kittler. Robust detection of lines using the progressive probabilistic hough transform. Computer Vision and Image Understanding, 78(1):119–137, 2000.

Параметры функции:

- *image* – 8-битовое, с одним каналом, бинарное изображение. Может быть изменено функцией.
- *lines* – возвращаемый массив линий. Каждая линия представлена 4-мерным вектором (x_1, y_1, x_2, y_2), где (x_1, y_1) и (x_2, y_2) конечные точки детектированного отрезка.
- *rho* – разрешение дистанции аккумулятора в пикселях.
- *theta* – разрешение угла аккумулятора в радианах.

- *threshold* – пороговое значение аккумулятора, возвращаются только те линии, количество голосов у которых превышает заданный порог.
- *minLineLength* – минимальное значение отрезка, отрезки с меньшей длиной не возвращаются.
- *maxLineGap* – максимально разрешимый зазор между точками на одной линии для их рассмотрения как линии.

При применении данной функции к изображению будем производить поиск оптимальных параметров (вручную), дающих лучший результат

Очевидно, что при подаче на вход функции исходного изображения в градациях серого, результат будет неверным, так как алгоритм должен работать с бинарными изображениями (это и написано в документации).

Покажем этот неверный результат на примере изображения №2.

Listing 2. Вывод линий, найденных с помощью преобразования Хафа на Python.

```
# Display lines
img_out_prob = cv.cvtColor(img, cv.COLOR_GRAY2BGR).copy()
if lines_prob is not None:
    for i in range(0, len(lines_prob)):
        l = lines_prob[i][0]
        cv.line(img_out_prob, (l[0], l[1]), (l[2], l[3]), (0, 255, 255), 2,
cv.LINE_AA)
fig = plt.figure(figsize=(13, 9))
plt.imshow(cv.cvtColor(img_out_prob, cv.COLOR_BGR2RGB))
plt.axis('off')
```

Figure 4: найденные линии при применении преобразования Хафа к исходному изображению №2.

Как видим, действительно линии не имеют смысла.

Таким образом, необходимо сделать бинарные изображения содержащие контуры из наших исходных. Сделаем это с помощью применения алгоритма Кэнни.

Listing 3. Применение алгоритма Кэнни на Python.

```
# Apply Canny algorithm
img_edge = cv.Canny(img, 200, 250, None, 3, L2gradient=True)
```

Естественно, что для каждого изображения необходимо подобрать такие параметры функции *Canny()*, чтобы получить лучший результат.

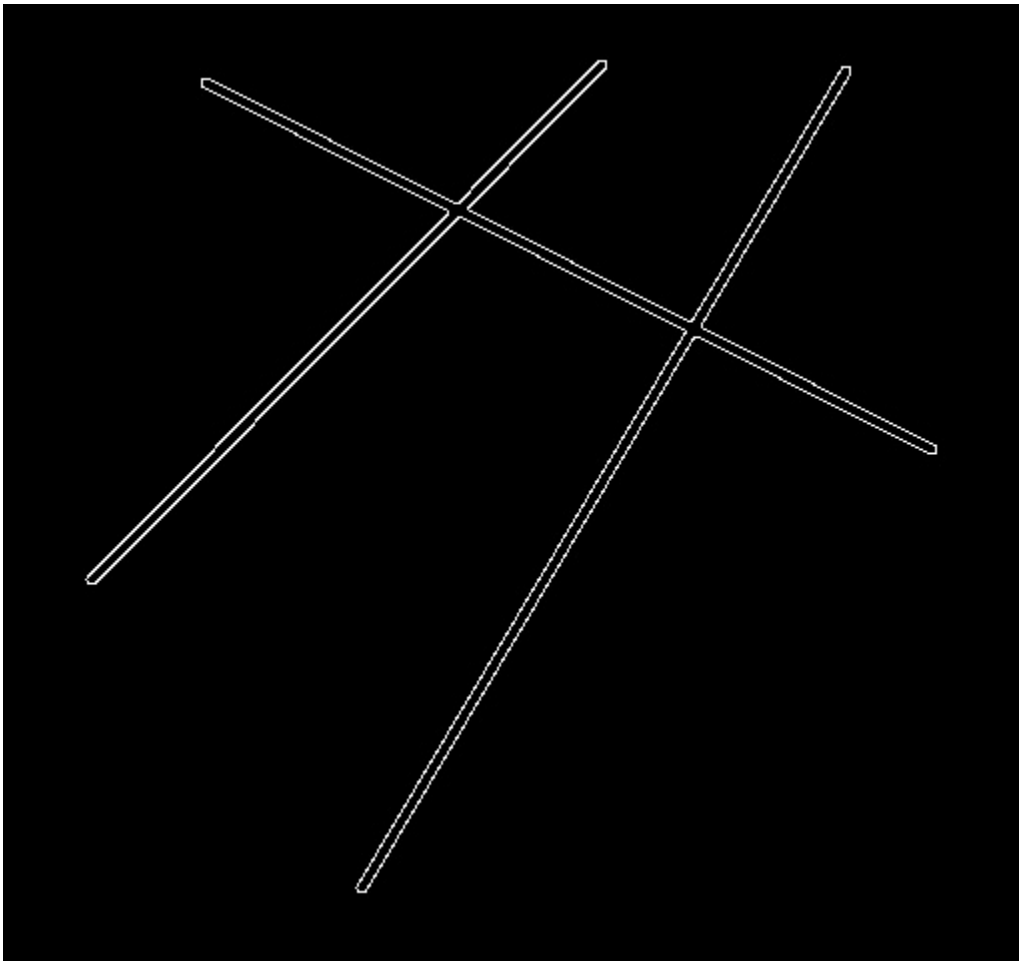

Figure 5: края изображения №1.

Параметры:

- *threshold1 = 200*
- *threshold2 = 250*
- *apertureSize = 3*
- *L2gradient = True*

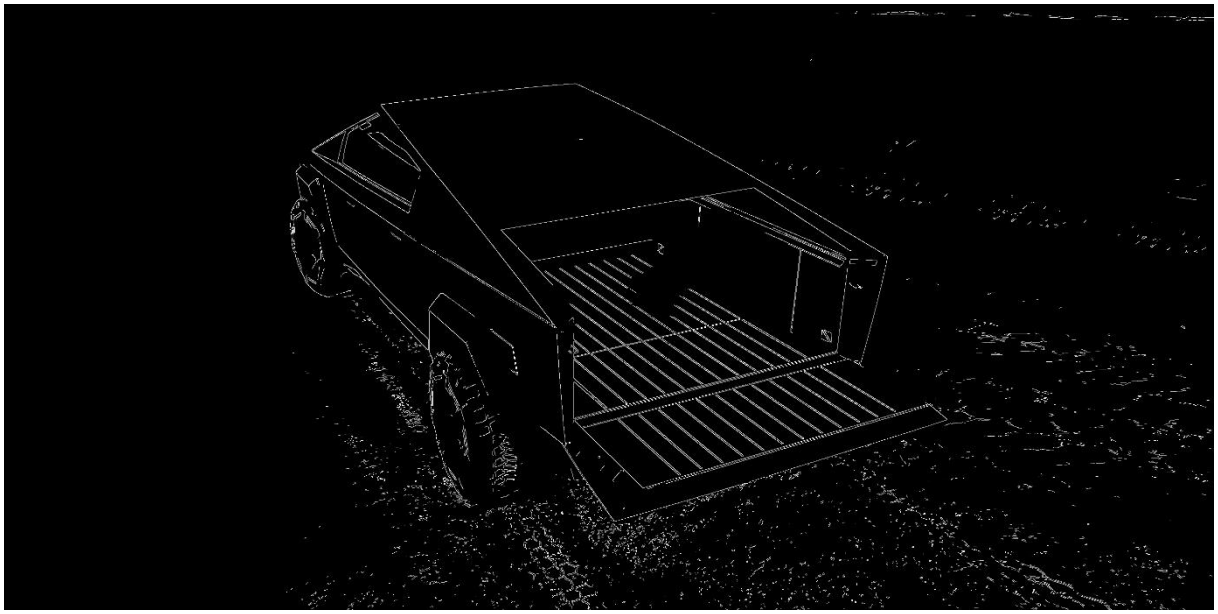

Figure 6: края изображения №2.

Параметры:

- *threshold1 = 150*
- *threshold2 = 200*
- *apertureSize = 3*
- *L2gradient = True*

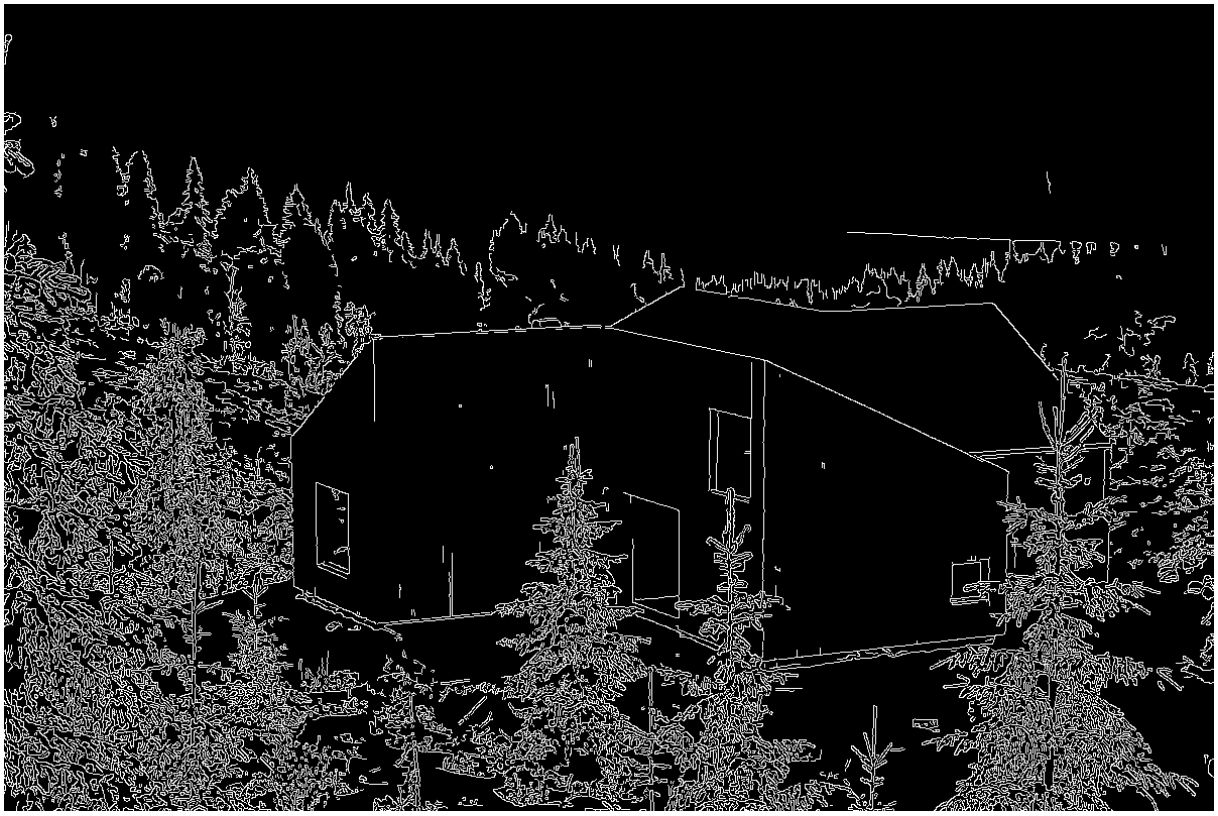

Figure 7: края изображения №3.

Параметры:

- *threshold1 = 100*
- *threshold2 = 200*
- *apertureSize = 3*
- *L2gradient = True*

Теперь, имея бинарные изображения краев исходных изображений можем применить преобразование Хафа для поиска прямых линий.

Для начала отобразим пространство признаков преобразования Хафа для исходных изображений.

Будем использовать коэффициент усиления = 3 для более отчетливого пространства.

Listing 3. Вывод пространства признаков преобразования Хафа.

```
# Displaying the Hough transform parameter space
angle_step_factor = 0.1
angles = np.linspace(-np.pi / 2, np.pi / 2, int(round(360 / angle_step_factor)),
endpoint=False)
img_h, theta, rho = skimage.transform.hough_line(img_edge, theta=angles)
# Visualize
fig = plt.figure(figsize=(13, 9), dpi=80)
angle_step = 0.5 * np.diff(theta).mean()
dist_step = 0.5 * np.diff(rho).mean()
bounds = [np.rad2deg(theta[0] - angle_step), np.rad2deg(theta[-1] + angle_step),
rho[-1] + dist_step, rho[0] - dist_step]
```

```
plt.imshow(cv.cvtColor(np.float32(3.0 * img_h / np.max(img_h)), cv.COLOR_GRAY2RGB),
extent=bounds, aspect=0.1)
plt.title('Hough parameter space')
plt.xlabel('Angles (degrees)')
plt.ylabel('Distance (pixels)')
plt.show()
fig.savefig(img_path + img_name.rpartition('.')[0] + "_param_space.jpg")
```

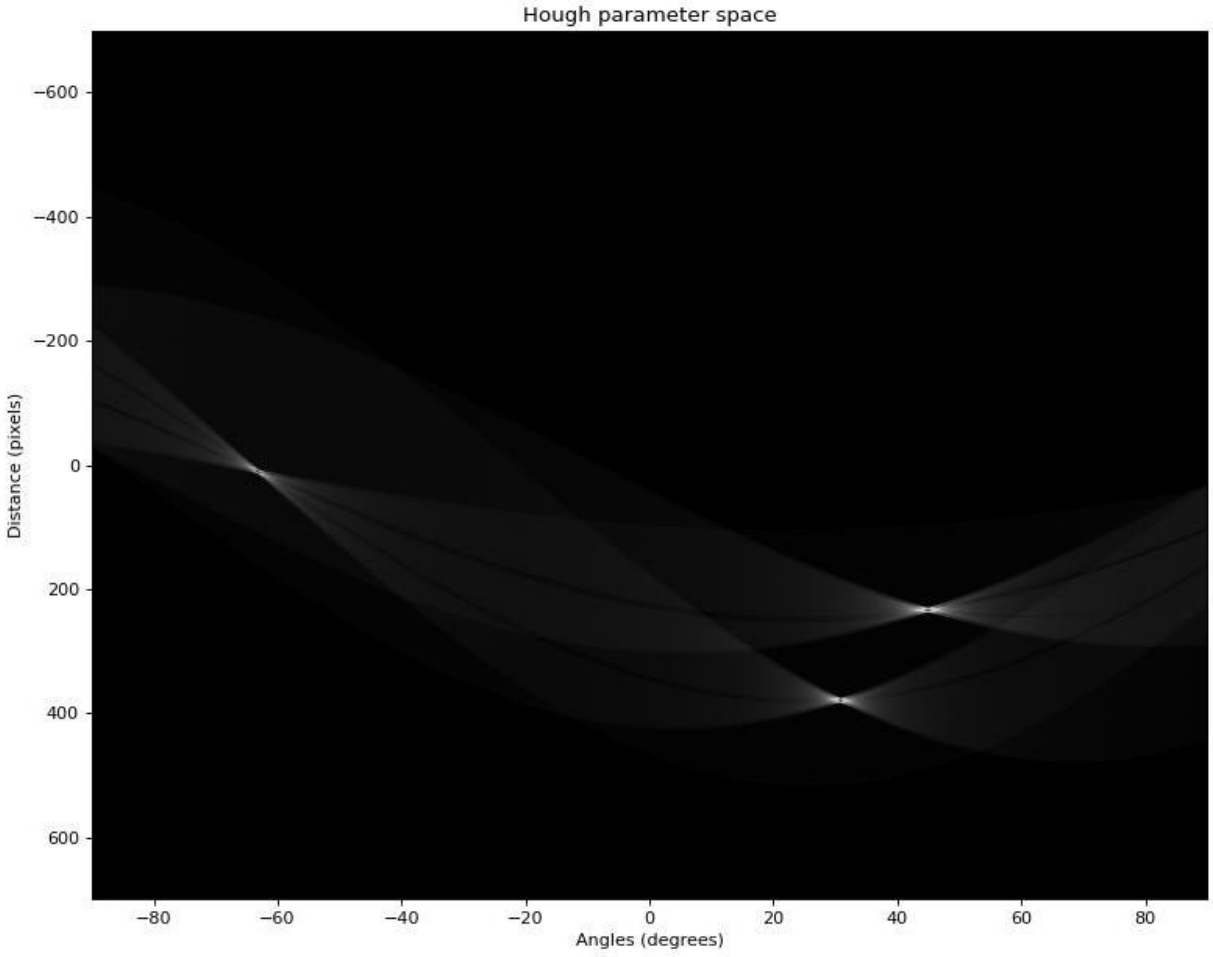

Figure 8: пространство параметров изображения №1.

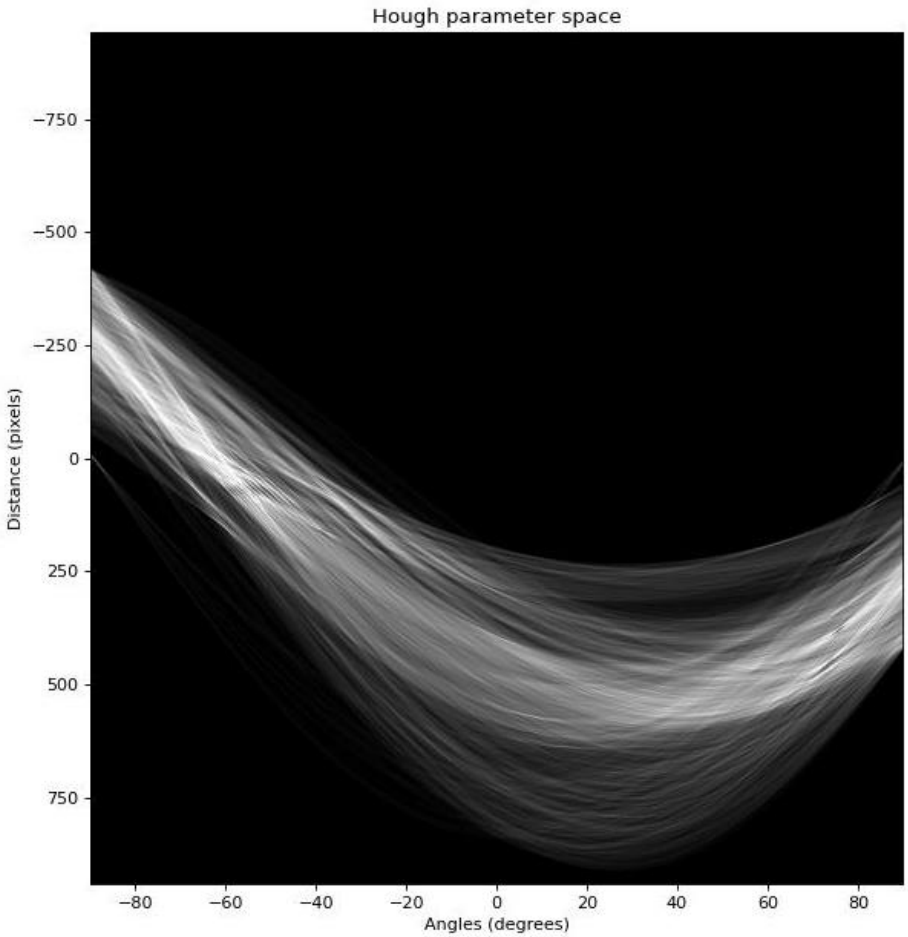

Figure 9: пространство параметров изображения №2.

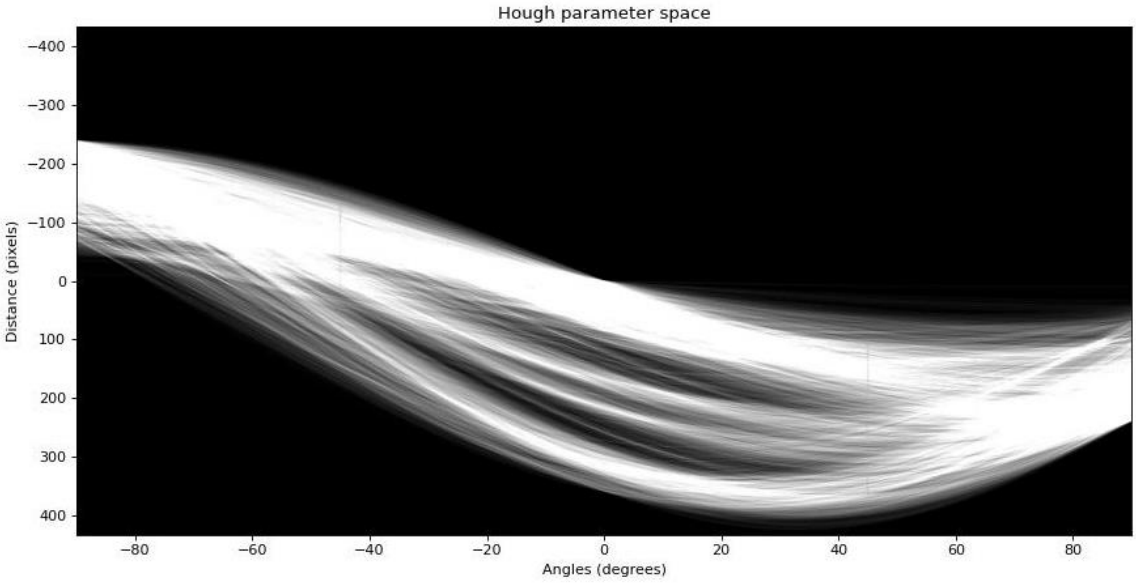

Figure 10: пространство параметров изображения №3.

Из изображения 1 очевидно наличие 3-х линий. Можно даже сказать, что углы их наклона примерно равны $\{-60°, 30°, 50°\}$.

Из изображения пространства Хафа 2-го изображения можно сказать, что преобладают линии с коэффициентом наклона $[-80, -60]$ и $[60, 80]$. Также данные кластеры находятся на достаточно большом расстоянии между собой. Кластер прямых с наклоном $[-80, -60]$ представляет собой линии, образованные ребристостью в грузовом отсеке пикапа.

А вот исходя из анализа пространства Хафа изображения №3 сложно что-то сказать. Необходимо пропускать линии через некоторое пороговое значения аккумулятора для определения наиболее значимых направлений.

Теперь отобразим линии на исходных изображениях.

Listing 4. Отображение линий на исходных изображениях.

```
# Display lines
img_out_prob = cv.cvtColor(img, cv.COLOR_GRAY2BGR).copy()
# img_out_prob = np.zeros_like(cv.cvtColor(img, cv.COLOR_GRAY2BGR))
shortest = -1
longest = -1
if lines_prob is not None:
    for i in range(0, len(lines_prob)):
        l = lines_prob[i][0]
        cv.line(img_out_prob, (l[0], l[1]), (l[2], l[3]), (255, 0, 0), 2,
cv.LINE_AA)
        distance = np.linalg.norm(np.array([l[0] - l[2], l[1] - l[3]]))
        if distance > longest or longest == -1:
            longest = distance
        if distance < shortest or shortest == -1:
            shortest = distance
    for i in range(0, len(lines_prob)):
        l = lines_prob[i][0]
        cv.circle(img_out_prob, (l[0], l[1]), radius=3, color=(0, 255, 255),
thickness=-1)
        cv.circle(img_out_prob, (l[2], l[3]), radius=3, color=(0, 255, 255),
thickness=-1)
fig = plt.figure(figsize=(13, 9))
plt.imshow(cv.cvtColor(img_out_prob, cv.COLOR_BGR2RGB))
plt.axis('off')
```

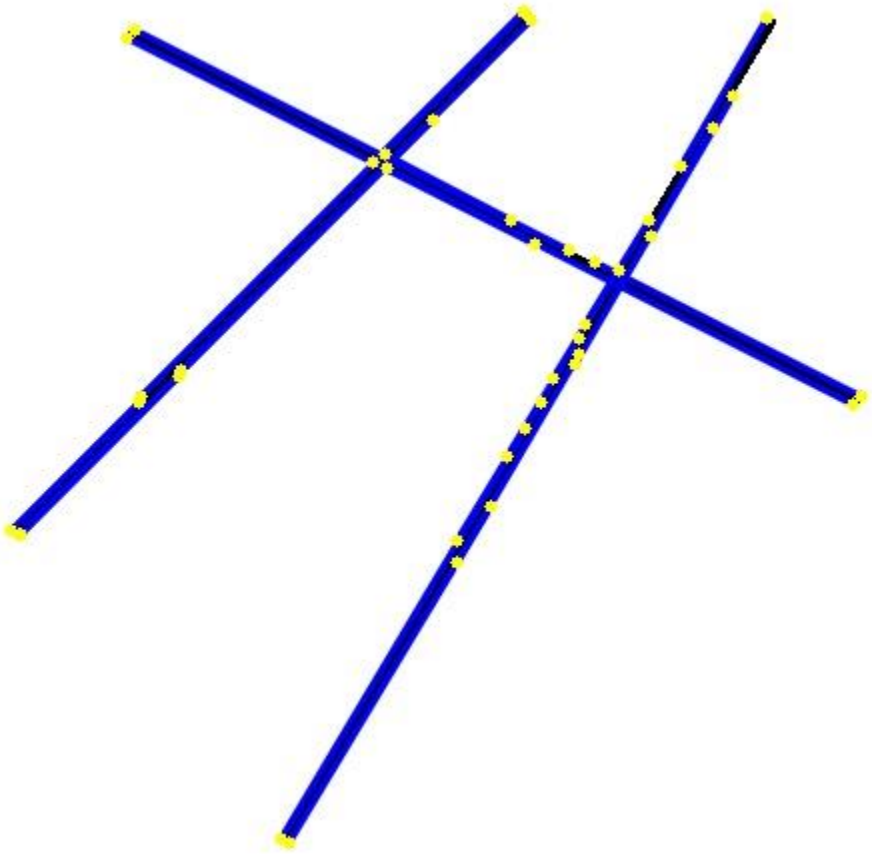

Figure 11: найденные отрезки на изображении №1.

Использованные параметры функции $cv.HoughLinesP$:

- $threshold = 5$
- $minLineLength = 50$
- $maxLineGap = 20$

Результаты:

- $Number\ of\ lines = 21$
- $Max\ length = 298\ pix.$
- $Min\ length = 122\ pix.$

Результаты выделения линий на таком простом изображении хорошие.

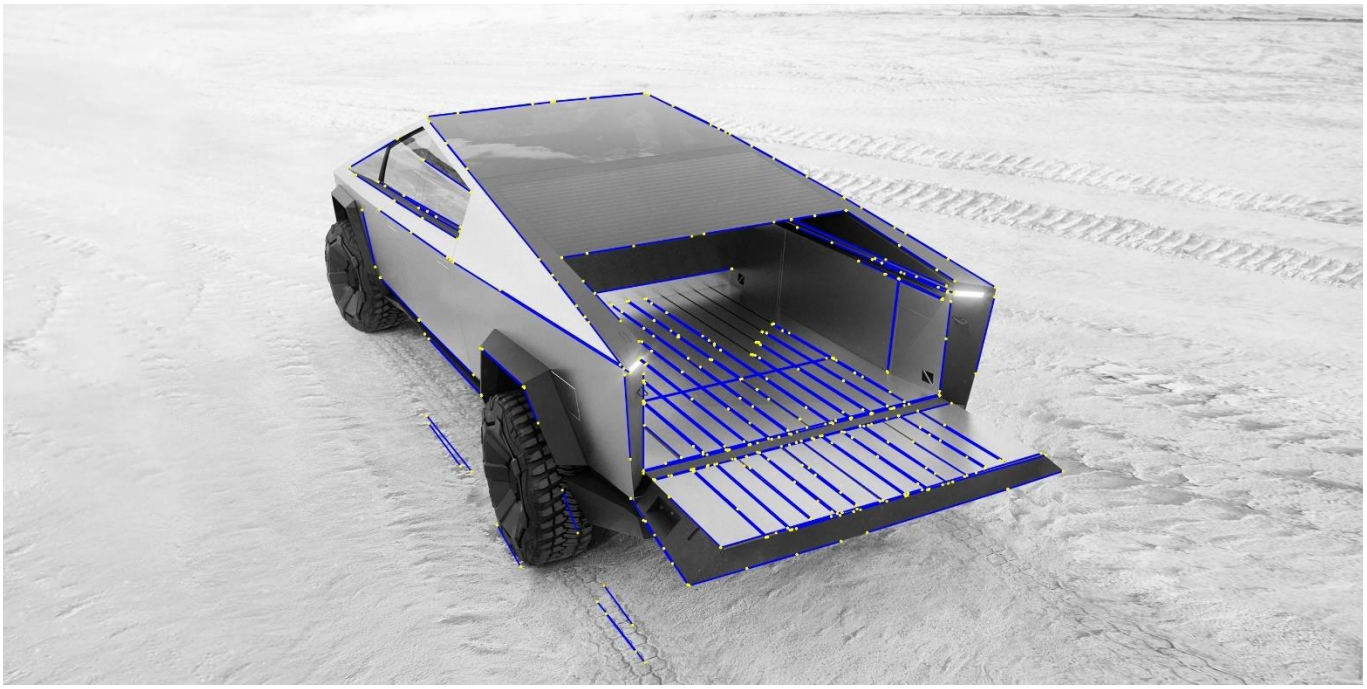

Figure 12: результат поиска отрезков на изображении №2.

Использованные параметры функции $cv.HoughLinesP$:

- $threshold = 30$
- $minLineLength = 75$
- $maxLineGap = 10$

Результаты:

- $Number\ of\ lines = 196$
- $Max\ length = 440\ pix.$
- $Min\ length = 50\ pix.$

Результаты поиска линий хорошие, естественно при условии подбора параметров. Удалось выделить контуры автомобиля.

Figure 13: результат выделения отрезков на изображении №3.

Использованные параметры функции $cv.HoughLinesP$:

- $threshold = 50$
- $minLineLength = 75$
- $maxLineGap = 5$

Результаты:

- $Number\ of\ lines = 7$
- $Max\ length = 75\ pix.$
- $Min\ length = 189\ pix.$

Данный результат является лучшим после долгого перебора параметров преобразования. В данном случае удалось хоть как-то выделить некоторые линии дома, захватив при этом довольно много линий, связанных с деревьями.

В итоге выделить линии на изображении №3 не удалось с помощью преобразования Хафа, для решения данной задачи нужно использовать более продвинутые и сложные алгоритмы.

2. Поиск окружностей

Исходные изображения:

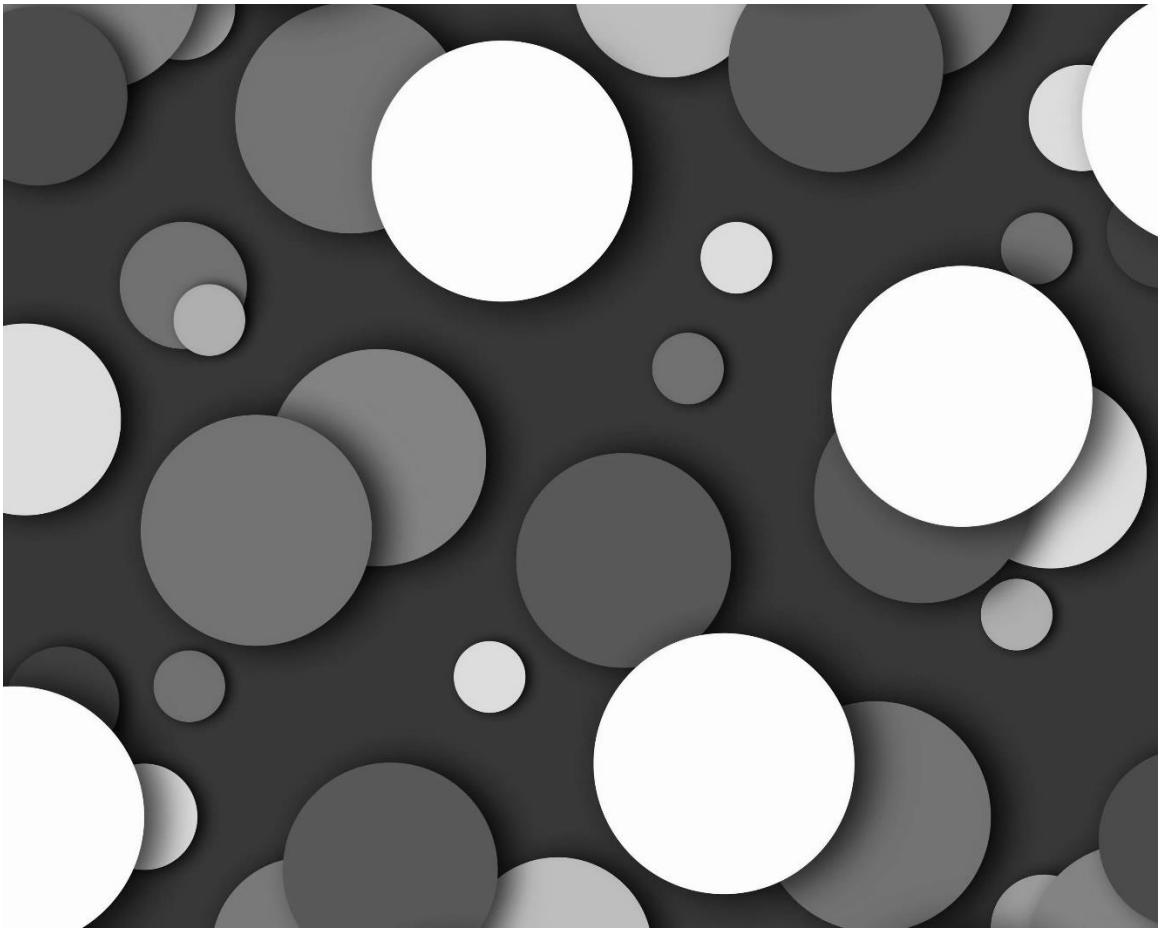

Figure 14: изображение №1.

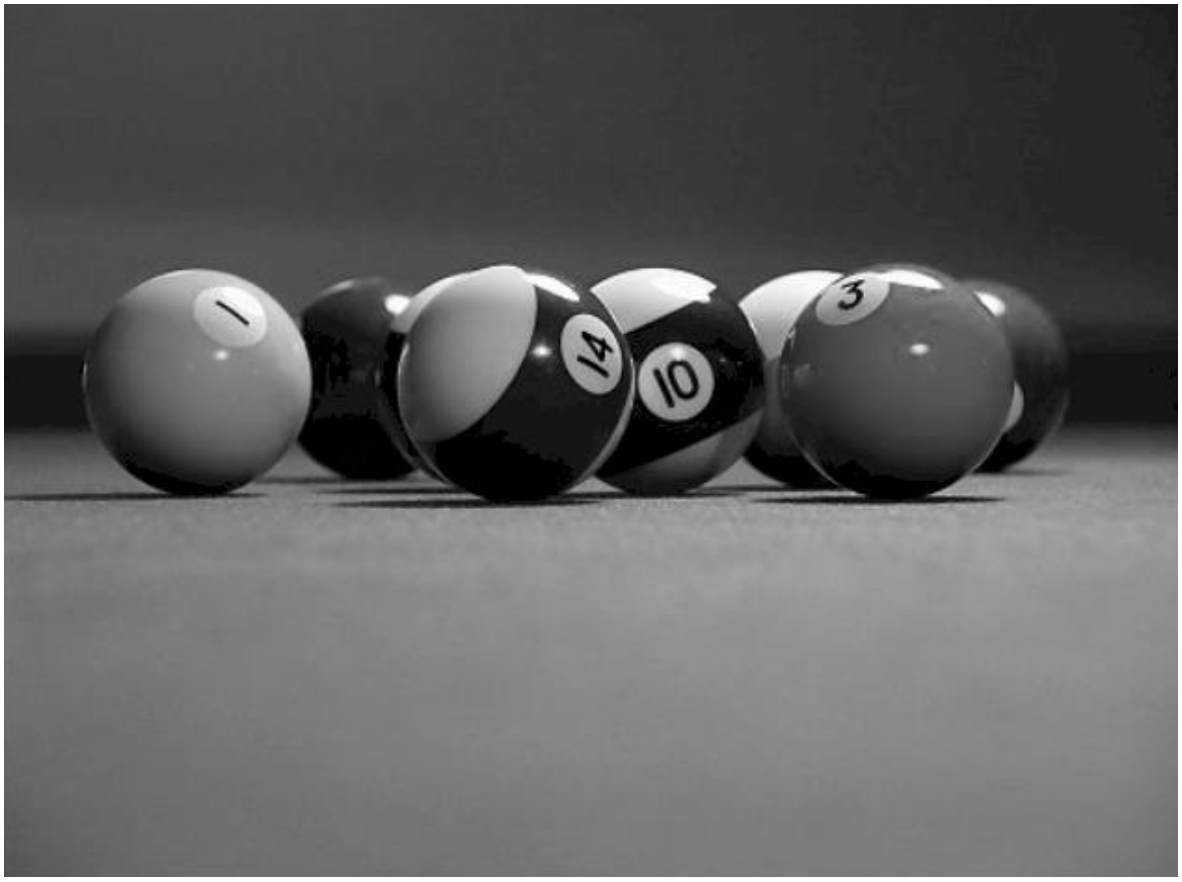

Figure 15: изображение №2.

Figure 16: изображение №3.

Применим функцию $cv.HoughCircles()$ для поиска окружностей на изображении. Данная функция имеет следующие параметры:

- $image$ – 8-битное, одноканальное изображение в градациях серого. То есть функция будет работать с небинарными изображениями, однако все же лучше найти края с помощью алгоритма Кэнни и подавать бинарное изображение.
- $circles$ – итоговый массив, содержащий найденные окружности. Каждый элемент массива является вектором с плавающей точкой $(x, y, radius, votes)$.
- $method$ – метод определения окружностей. Доступны следующие методы: HOUGH_GRADIENT и HOUGH_GRADIENT_ALT.
- dp – обратное отношение разрешения аккумулятора к разрешению изображения. Если $dp = 2$, аккумулятор имеет половину ширины и высоты. Для HOUGH_GRADIENT_ALT рекомендуемое значение $dp = 1.5$, если только не нужно обнаруживать очень маленькие круги.
- $minDist$ – минимальное расстояние между двумя детектируемыми окружностями.
- $param1$ – значение более высокого порог из двух, переданных детектору границ Кэнни (нижний в два раза меньше). HOUGH_GRADIENT_ALT использует алгоритм Щарра для вычисления производных изображения, поэтому пороговое значение обычно выше.
- $param2$ – в случае HOUGH_GRADIENT это порог аккумулятора для центров кругов на этапе обнаружения. Чем он меньше, тем больше ложных кругов может быть обнаружено. Окружности, соответствующие большим значениям аккумулятора, будут возвращены

первыми. В случае алгоритма HOUGH_GRADIENT_ALT это мера «совершенства» окружности. Чем он ближе к 1, тем «лучше» алгоритм выбирает круг.

- $minRadius$ – минимальный радиус окружностей.
- $maxRadius$ – Максимальный радиус окружности. Если <= 0, используется максимальный размер изображения. Если < 0, HOUGH_GRADIENT возвращает центры без нахождения радиуса. HOUGH_GRADIENT_ALT всегда вычисляет радиусы окружности.

Применим данный метод для исходных изображений.

Также стоит отметить, что перед преобразованием Хафа рекомендуется пропустить изображение через сглаживающий фильтр, например через фильтр Гаусса. Это будет выполнено всегда в данной работе.

Listing 5. Применение преобразования Хафа для нахождения окружностей любого радиуса на изображении на Python.

```
# Apply filtering
img = cv.GaussianBlur(img, (7, 7), sigmaX=1.5, borderType=cv.BORDER_REFLECT)
# Apply standard Hough transform for finding circles
circles = cv.HoughCircles(image=img, method=cv.HOUGH_GRADIENT, dp=1, minDist=10)
```

Listing 6. Отображение найденных окружностей и их центров на исходном изображении на Python.

```
# Display lines
img_out = cv.cvtColor(img, cv.COLOR_GRAY2BGR).copy()
if circles is not None:
    for i in range(0, len(circles[0])):
        c = (circles[0][i]).astype(np.int32)
        cv.circle(img_out, (c[0], c[1]), radius=c[2], color=(0, 255, 255),
thickness=2, lineType=cv.LINE_AA)
        cv.circle(img_out, (c[0], c[1]), radius=2, color=(0, 0, 255), thickness=2,
lineType=cv.LINE_AA)
fig = plt.figure(figsize=(13, 9))
plt.imshow(cv.cvtColor(img_out, cv.COLOR_BGR2RGB))
plt.axis('off')
```

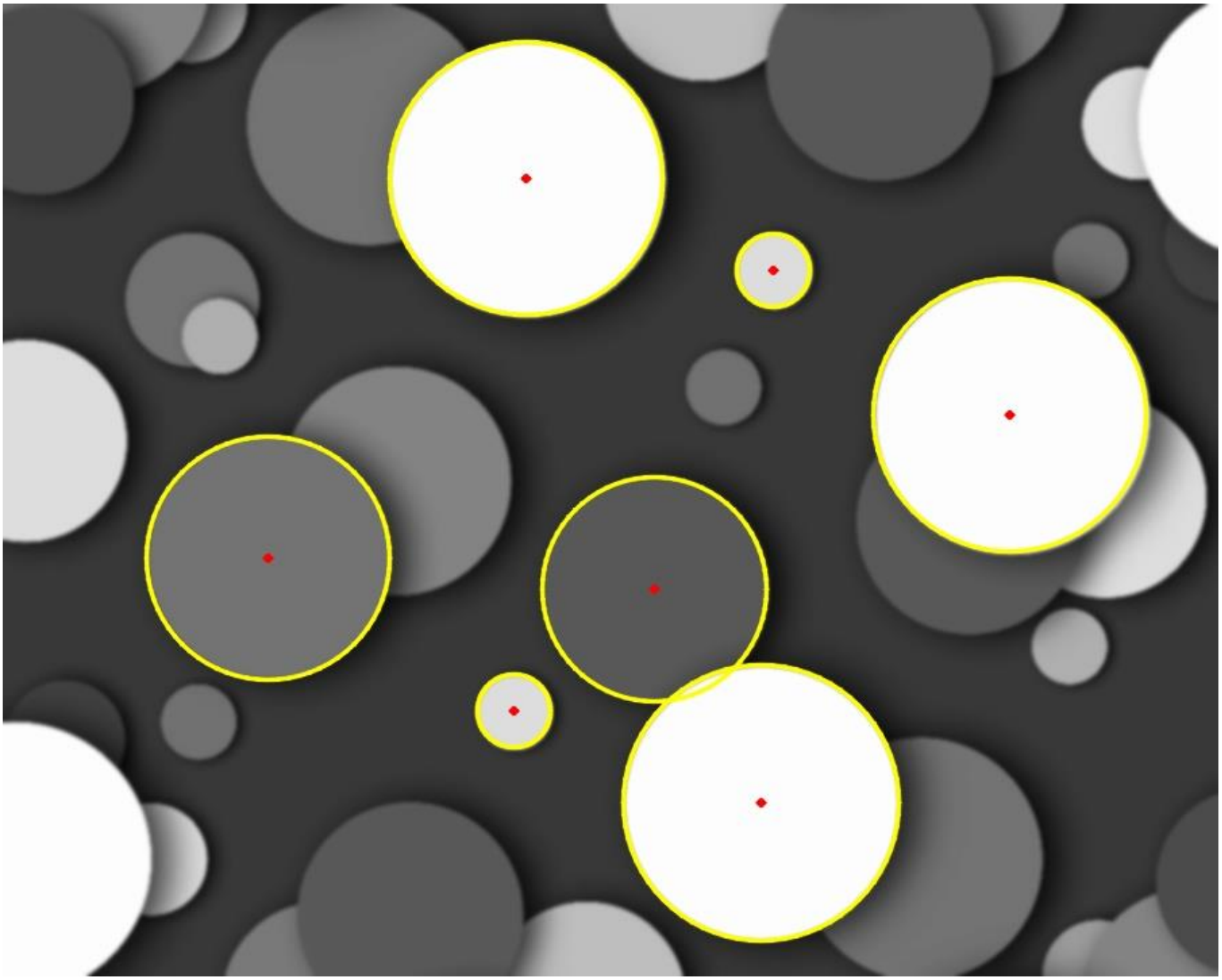

Figure 17: поиск окружностей для исходных изображений в градациях серого (желтыми линиями обозначены окружности, красные точки – их центры).

Результаты поиска окружностей:

$$maxRadius = 102 \qquad minRadius = 27 \qquad circlesNumber = 7$$

Алгоритм нашел малую часть окружностей присутствующих на изображении.

При применении алгоритма к изображению №2 алгоритм не нашел ни одной окружности.

Применим преобразование Хафа для изображения №3.

Figure 18: найденные окружности и их центры изображения №3.

Результаты поиска окружностей:

$$maxRadius = 125 \qquad minRadius = 40 \qquad circlesNumber = 11$$

Теперь будем использовать метод поиска $cv.HOUGH\_GRADIENT\_ALT$ и параметры $param1, param2$ для дополнительного применения краевых детекторов к изображению, а также возможности поиска «неидеальных окружностей».

Listing 7. Поиск окружностей любого радиуса методом $cv.HOUGH\_GRADIENT\_ALT$ с применением краевого детектора Щарра и коэффициентом «идеальности» окружностей на Python.

```
cv.imwrite(img_path + img_name.rpartition('.')[0] + '_canny.jpg', img_edge)
#%%
# Apply filtering
img = cv.GaussianBlur(img, (7, 7), sigmaX=1.5, borderType=cv.BORDER_REFLECT)
# Apply standard Hough transform for finding circles
circles = cv.HoughCircles(image=img, method=cv.HOUGH_GRADIENT_ALT, dp=1.5,
minDist=10, param1=300, param2=0.7, minRadius=0, maxRadius=0)
```

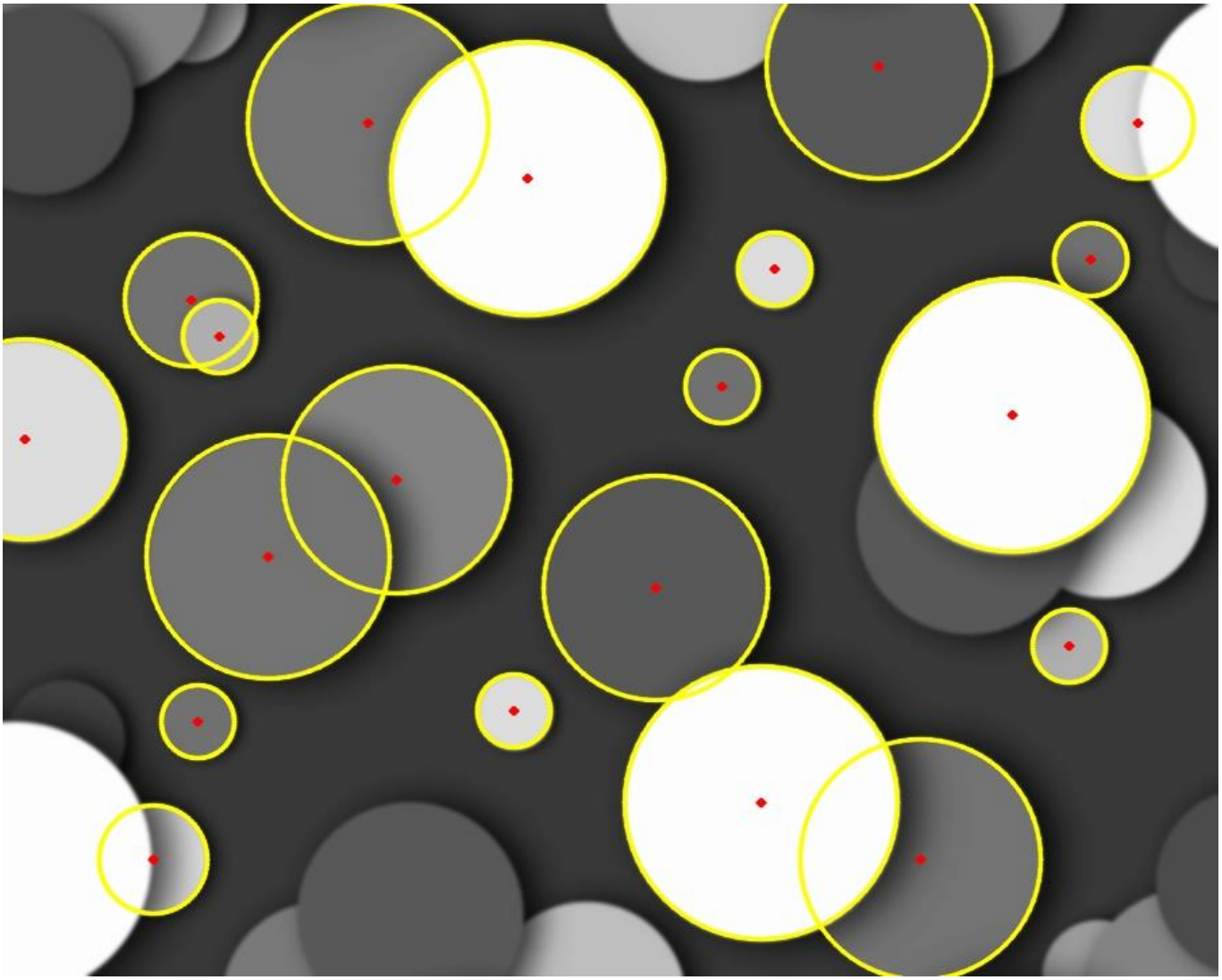

Figure 19: окружности и их центры, найденные с помощью преобразования Хаффа.

Результаты поиска окружностей:

$$maxRadius = 102 \qquad minRadius = 28 \qquad circlesNumber = 20$$

Результат работы алгоритма существенно лучше, чем при применении к изображению в градациях серого. Удалось найти на 9 окружностей больше.

Применим алгоритм к изображению №2.

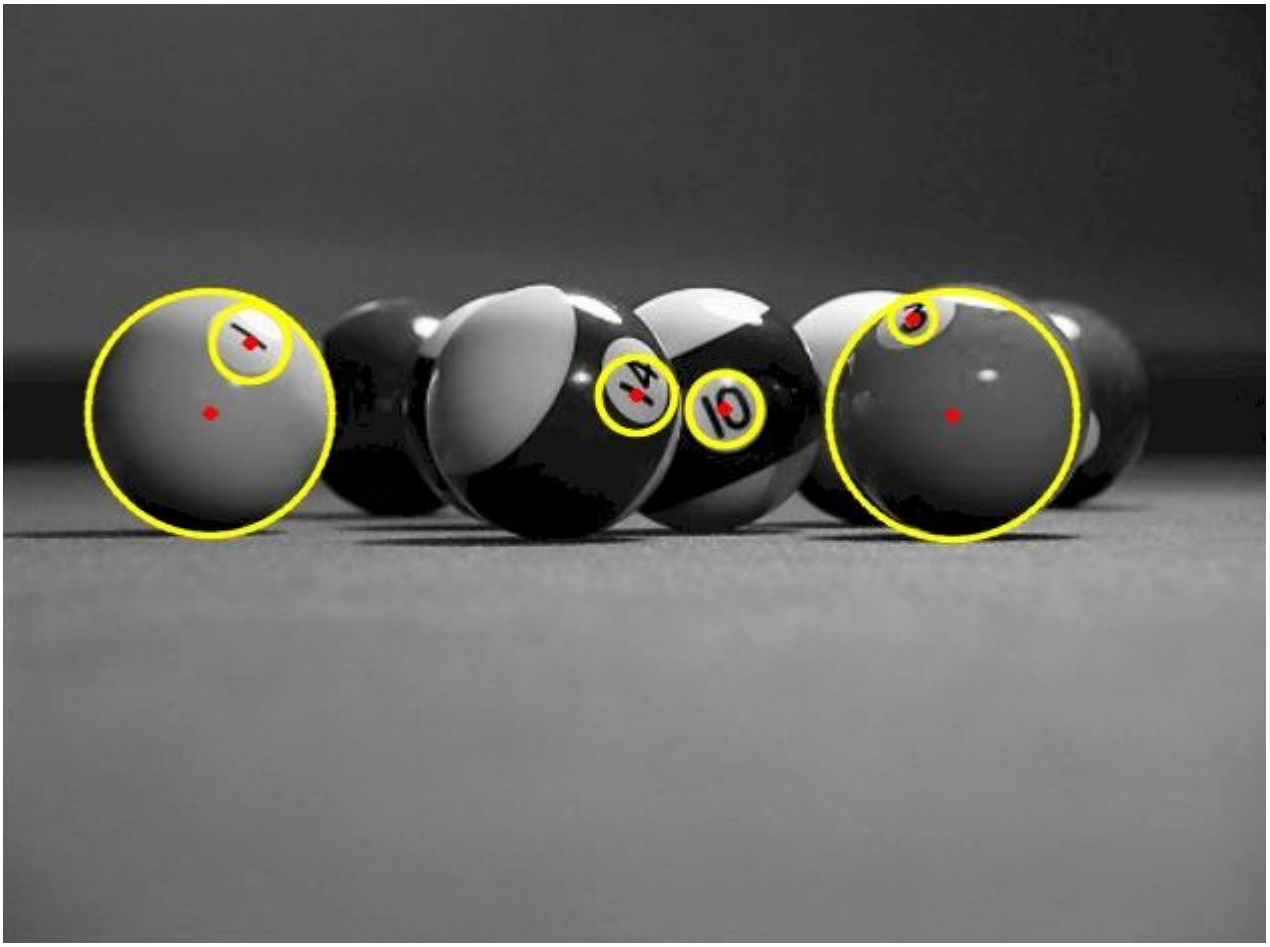

Figure 20: окружности и их центры, найденные с помощью преобразования Хафа.

Результаты поиска окружностей:

$maxRadius = 63$ $minRadius = 13$ $circlesNumber = 6$

Как видно, результат применения алгоритма с краевым детектором намного лучше (без краевых фильтров алгоритм вообще не нашел ни одной окружности).

Применим алгоритм к изображению №3.

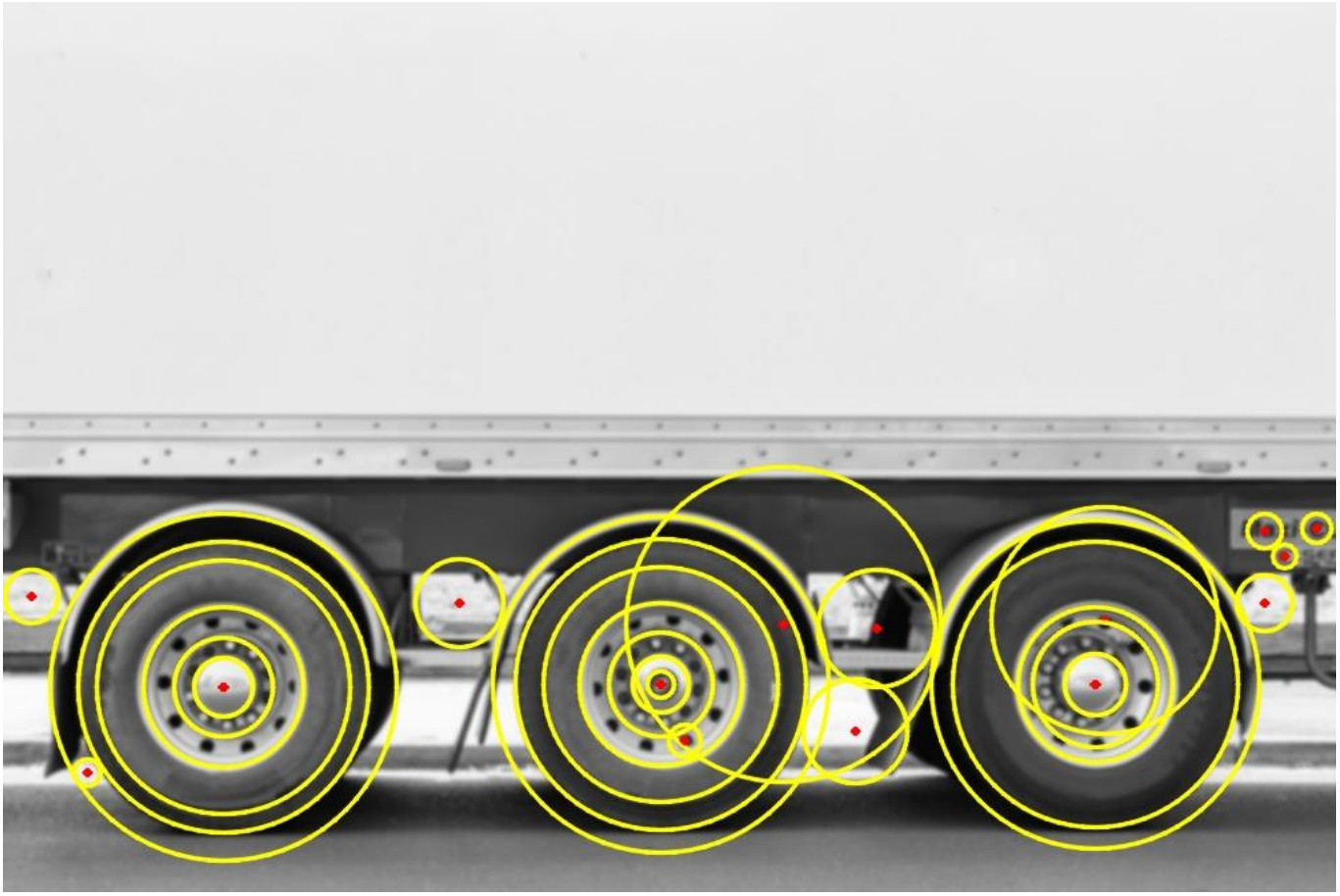

Figure 21: найденные окружности и их центры.

Результаты поиска окружностей:

$$maxRadius = 130 \qquad minRadius = 10 \qquad circlesNumber = 30$$

Были найдены все окружности колес разных радиусов, однако присутствуют лишние окружности.

Теперь найдем окружности определенного радиуса на изображении №3. Таким образом, нам удастся выделить именно те объекты, которые необходимо.

Выделим только окружности, которые описывают диски колес.

Путем последовательного уменьшения интервала допустимых радиусов окружностей был найден диапазон $[60, 65]\ pix$ в котором находятся окружности описывающие диски.

Результат выделения:

Figure 22: результат выделения окружностей заданного радиуса.

Как видим, удалось выделить все 3 диска.

3. Реализация классического преобразования Хафа и сравнение с встроенными функциями *OpenCV*.

Listing 8. Реализация поиска линий на изображении с помощью классического преобразования Хафа на Python.

```
# Read an image
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)
img_edge = cv.Canny(img, 150, 200, None, 3)
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)


# Do Hough transform
angle_step_factor = 0.1
angles = np.linspace(-np.pi / 2, np.pi / 2, int(round(360 / angle_step_factor)),
endpoint=False)
img_h, theta, rho = skimage.transform.hough_line(img_edge, theta=angles)


# Output result
fig = plt.figure(figsize=(16, 9))
```

```
plt.imshow(cv.cvtColor(img_gray, cv.COLOR_GRAY2RGB))
plt.xticks([]),plt.yticks([])

for _, angle, dist in zip(*skimage.transform.hough_line_peaks(img_h, theta, rho)):
    (x0, y0) = dist * np.array([np.cos(angle), np.sin(angle)])
    plt.axline((x0, y0), slope=np.tan(angle + np.pi/2))

plt.savefig(img_path + img_name.rpartition('.')[0] + '_classic_hough.jpg')
```

Применим данный метод к изображению №2 с линиями и сравним с методом с помощью функции из OpenCV:

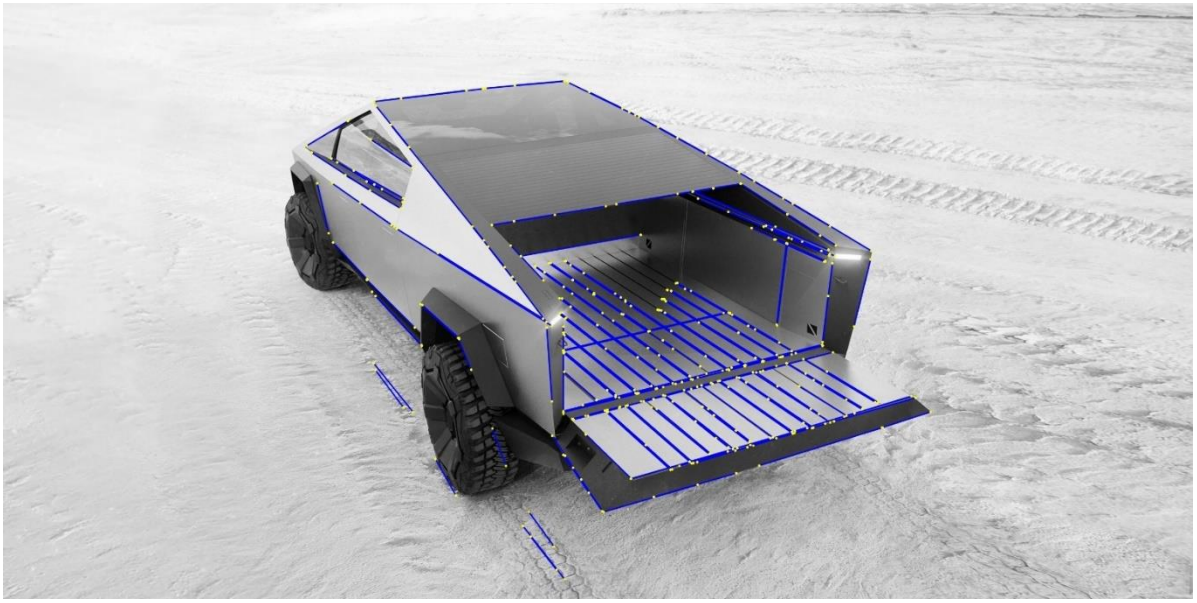

Figure 23: поиск линий с помощью OpenCV.

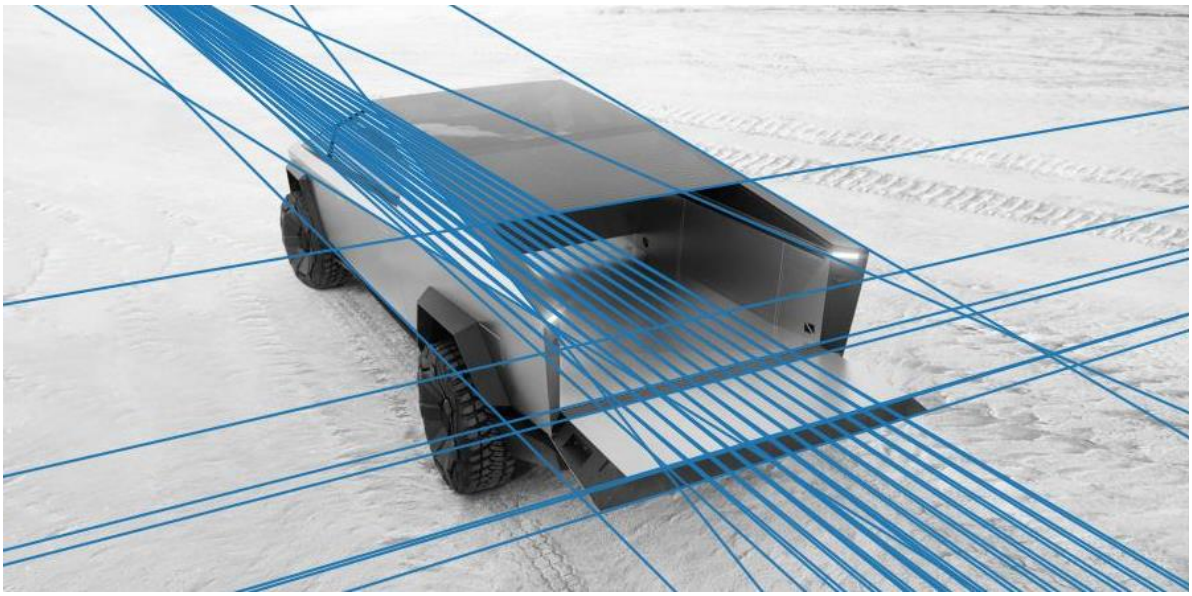

Figure 24: поиск линий классическим методом Хафа.

Очевидно, что при использовании классического метода нам выдаются линии, а не отрезки, как при использовании функции OpenCV.

Также заметны сильные различия во времени выполнения программ (алгоритмы OpenCV работают в разы быстрее).

В итоге выполнение классического метода полезно только в образовательных целях, тогда как функция библиотеки OpenCV имеет некоторые настраиваемые параметры, благодаря которым можно получать требуемые результаты.

Реализуем также поиск окружностей с помощью классического метода преобразования Хафа.

Listing 9. Реализация поиска окружностей на изображении с помощью классического преобразования Хафа на Python.

```
# Read an image
img = cv.imread(img_path + img_name, cv.IMREAD_COLOR)
img_edge = cv.Canny(img, 150, 200, None, 3)
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)


# Do Hough transform
radius = np.arange(20, 200, 2)
circles = skimage.transform.hough_circle(img_edge, radius)
accums, cx, cy, radii = skimage.transform.hough_circle_peaks(circles, radius,
threshold = 0.65 * np.max(circles))


img_out = cv.cvtColor(img_gray, cv.COLOR_GRAY2RGB)
for center_y, center_x, rad in zip(cy, cx, radii):
    circy, circx = skimage.draw.circle_perimeter(center_y, center_x,
int(round(rad)), shape=img_out.shape)
    img_out[circy, circx] = (0, 0, 255)
plt.figure(figsize=(32, 16), dpi=80)
plt.imshow(img_out)
plt.xticks([]),plt.yticks([])

plt.savefig(img_path + img_name.rpartition('.')[0] + '_classic_hough.jpg')
```

Применим данный метод к изображению №1 с окружностями и сравним результаты с функцией OpenCV:

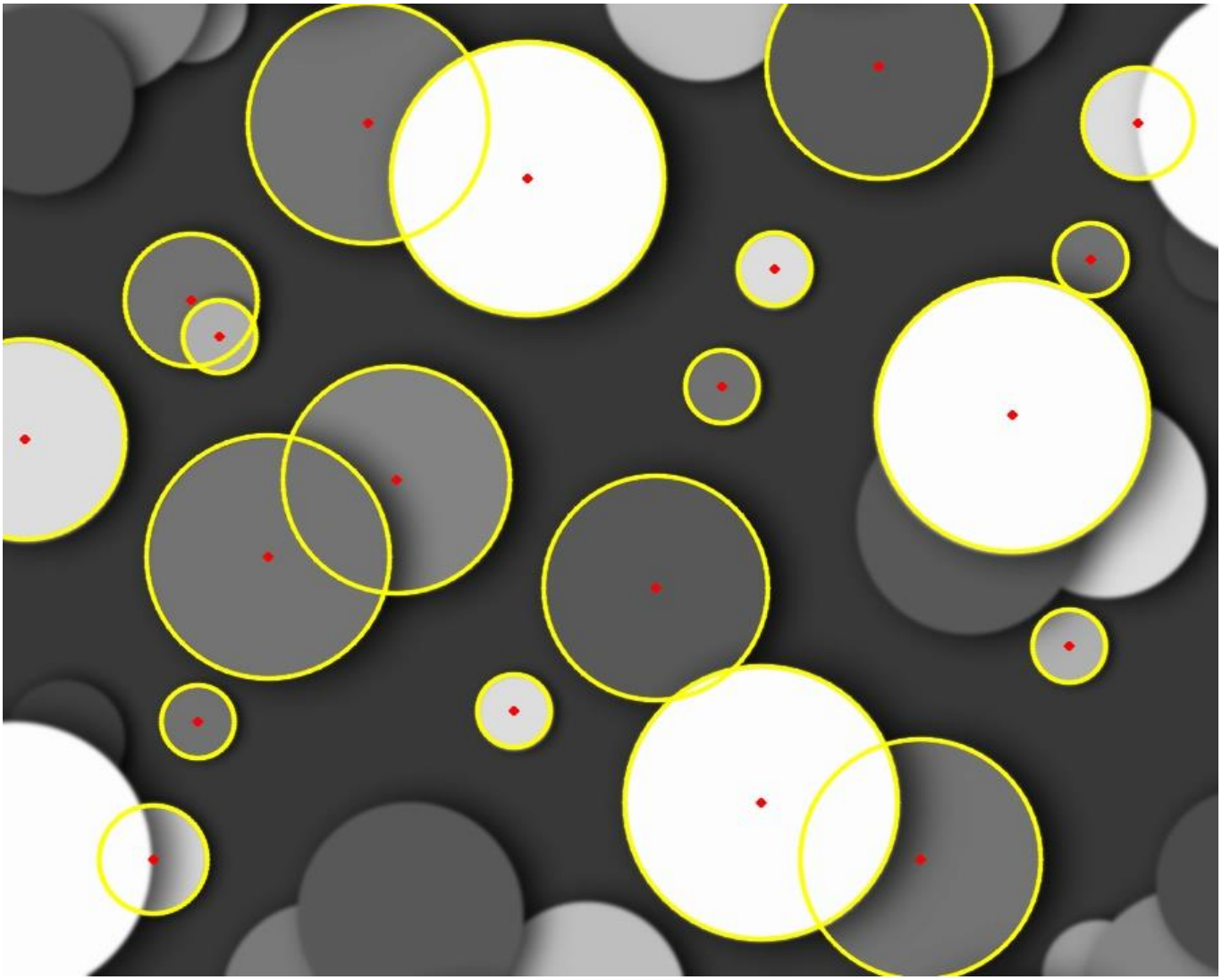

Figure 25: поиск окружностей с помощью OpenCV.

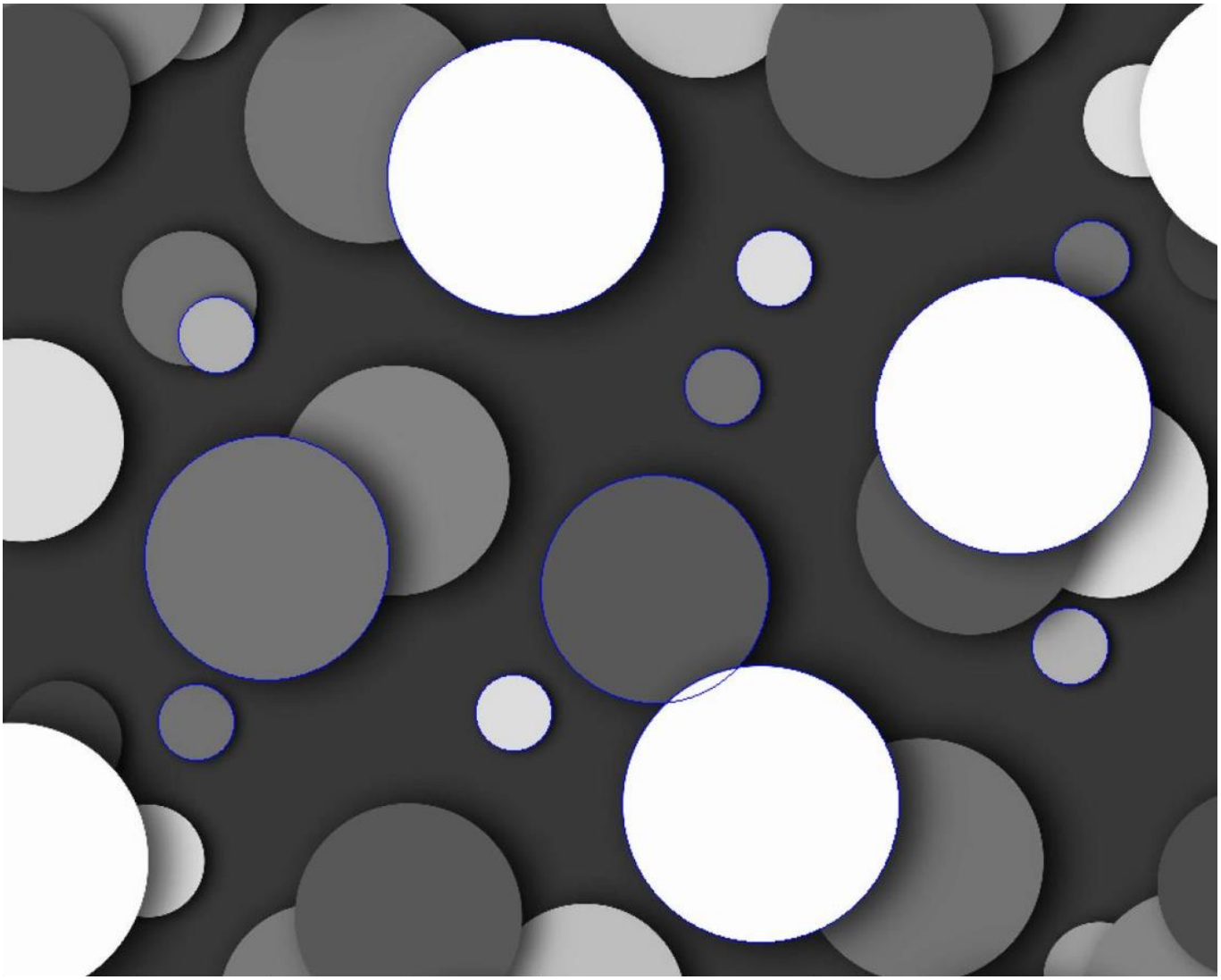

Figure 26: поиск окружностей классическим методом преобразования Хафа.

Количество найденных окружностей $= 12$. Что сильно меньше, чем то, которое нашел алгоритм OpenCV. Также такой классический алгоритм выполняется в разы дольше, чем алгоритм OpenCV.

Итого, использование методов преобразования Хафа лучше производить с помощью библиотеки OpenCV для более быстрого вычисления, лучших результатов, гибкой настройки параметров.

Выводы

В данной лабораторной работе были выполнены поиски прямых линий и окружностей с помощью преобразования Хафа.

Применять такие алгоритмы необходимо только на изображениях, состоящих из контуров, либо заранее применять дифференциальные операторы и сглаживание фильтрами. Иначе будет найдено слишком много фигур или вообще не найдено ни одной, а пространство Хафа будет непригодно для анализа.

Поиск линий или окружностей с помощью преобразования Хафа достаточно ресурсоемкое как по памяти, так и по вычислительным возможностям, однако этого можно избежать, если реализовывать алгоритмы самостоятельно для конкретной задачи на $C++$ (таким образом можно даже использовать алгоритмы в системах реального времени). Реализация преобразования Хафа в библиотеке $OpenCV$ достаточно хорошо оптимизирована и использует векторные вычисления, поэтому очень удобно и эффективно использовать встроенные функции для анализа изображений.

Стоит отметить, что существует еще обобщённое преобразование Хафа, с помощью которого можно искать контуры заданной формы.