Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет по лабораторной работе №5

«Преобразование Хафа»

Авторы:

Юманов М.А.

Алексеева В.В.

Преподаватель:

Шаветов С.В.

Санкт-Петербург, 2024

Цель работы

Освоение преобразования для поиска геометрических примитивов.

Теоретическое обоснование применяемых методов и функций

Выбор мы аналогично предыдущим работам остановили на использовании библиотеки OpenCV в сочетании с языком программирования C++.

Преобразование Хафа в идейной составляющей подразумевает под собой поиск общих геометрических мест точек (ГМТ). Например, данный подход используется при построении треугольника по трем заданным сторонам, когда в начале 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, было проведено голосование двух точек в пользу вероятного расположения третьей вершины треугольника. В результате «победила» точка, набравшая два «голоса» (точки на окружностях набрали по одному голосу, а вне их — по нулю).

При большом количестве характеристических точек на изображении идею можно обобщить.

В классическом преобразовании Хафа, основанном на идее «голосования» точек, используется пространство параметров. Сам подход заключается в том, что для каждой точки пространства параметров суммируется количество голосов, поданных за нее.

Наша работа с преобразованием Хафа включает в себя поиск прямых и окружностей для изображений двух видов, а также выполнение дополнительных задач. Подробнее рассмотрим далее…

Порядок выполнения работы

Поиск прямых. Давайте выберем три произвольных картинки, которые бы содержали прямые. Мы произведем поиск прямых с помощью алгоритма Хафа. Далее отразим найденные прямые на исходном изображении. Кроме того,

отметим точки начала и окончания линий. Определим длины самой длинной и короткой. В конце мы вычислим количество найденных нами прямых.

Подключение необходимых библиотек:

```
#include <iostream>
#include <string>
#include <vector>

#include <opencv2/core.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
```

Код программы:

```
// drawing hough lines
cv::Mat hough_lines(cv::Mat image_input, int threshold, int j, bool use_canny = 0, int
thresh_canny_1 = 0, int thresh_canny_2 = 0){
    cv::Mat image;
    image_input.copyTo(image);
    cv::Mat image_gray;
    cv::cvtColor(image, image_gray, cv::COLOR_BGR2GRAY);

    if (use_canny){
        cv::Mat canny;
        cv::Canny(image_gray, canny, thresh_canny_1, thresh_canny_2);
        canny.copyTo(image_gray);
        cv::imwrite("canny_" + std::to_string(j) + ".jpg", canny);
    }

    std::vector<cv::Vec4i> linesP;
    HoughLinesP(image_gray, linesP, 1, CV_PI/180, threshold, 130, 50);

    std::vector<int> dist;
    for(size_t i = 0; i < linesP.size(); i++ ){
        cv::Vec4i l = linesP[i];
        cv::Point pt1 = cv::Point(l[0], l[1]);
        cv::Point pt2 = cv::Point(l[2], l[3]);
        line(image, pt1, pt2, cv::Scalar(0,0,255), 3, cv::LINE_AA);
        dist.push_back(sqrt(pow(pt1.x - pt2.x, 2) + pow(pt1.y - pt2.y, 2)));
        cv::circle(image, pt1, 10, cv::Scalar(0, 255, 0), cv::FILLED);
        cv::circle(image, pt2, 10, cv::Scalar(0, 255, 0), cv::FILLED);
    }

    int max = 0;
    int min = INT_MAX;
    int cnt = 0;
    for (int i = 0; i < dist.size(); ++i){
```

```
        max = std::max(max, dist[i]);
        min = std::min(min, dist[i]);
        cnt++;
    }

    if (use_canny){


        std::cout << "Длина самого длинного отрезка " << j << "-й картинки с
использованием алгоритма Кэнни: " << max << std::endl;
        std::cout << "Длина самого короткого отрезка " << j << "-й картинки с
использованием алгоритма Кэнни: " << min << std::endl;
        std::cout << "Количество найденных прямых на " << j << "-й картинке с
использованием алгоритма Кэнни: " << cnt << std::endl;
    } else {
        std::cout << "Длина самого длинного отрезка " << j << "-й картинки: " << max <<
std::endl;
        std::cout << "Длина самого короткого отрезка " << j << "-й картинки: " << min <<
std::endl;
        std::cout << "Количество найденный прямых на " << j << "-й картинке: " << cnt <<
std::endl;
    }
    std::cout << std::endl;

    return image;

}



int main(){
    std::string filename = "original_";
    std::string filename_ext = ".jpg";

    std::vector<int> threshold {160, 20, 70};
    std::vector<int> thresh_canny_1{250, 250, 200};
    std::vector<int> thresh_canny_2 {255, 255, 255};

    // saving images
    for (int i = 1; i < 4; ++i){
        cv::Mat image = cv::imread(filename + std::to_string(i) + filename_ext);
        // drawing lines
        cv::imwrite("lines_" + std::to_string(i) + ".jpg", hough_lines(image,
threshold[i-1], i));
        // drawing lines with differential operator
        cv::imwrite("lines_canny_" + std::to_string(i) + ".jpg",
            hough_lines(image, threshold[i-1], i, true, thresh_canny_1[i-1],
thresh_canny_2[i-1]));
    }

    return 0;
}
```

Исходные данные и выход программы:

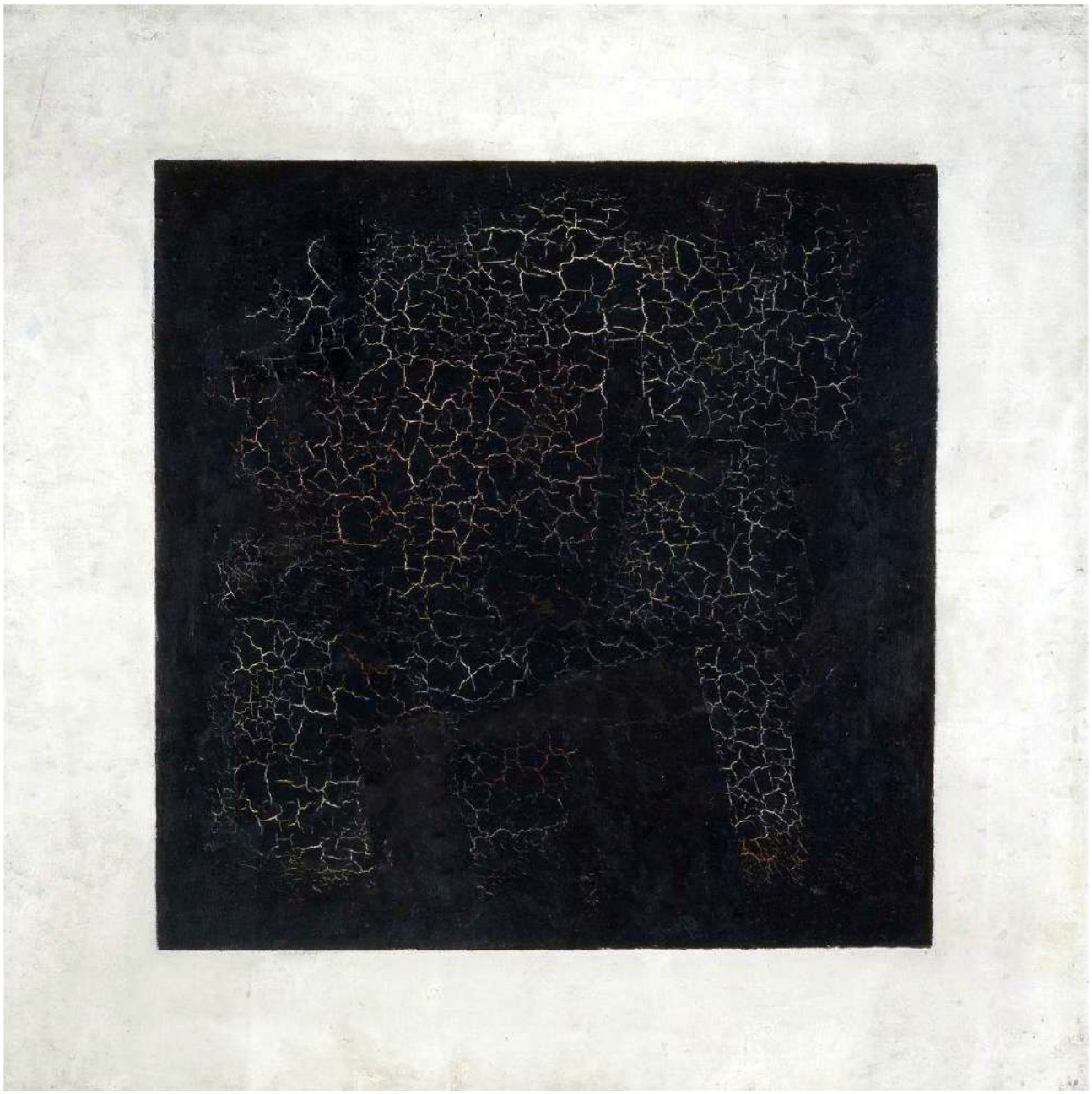

Рис. 1. Первое исходное изображение

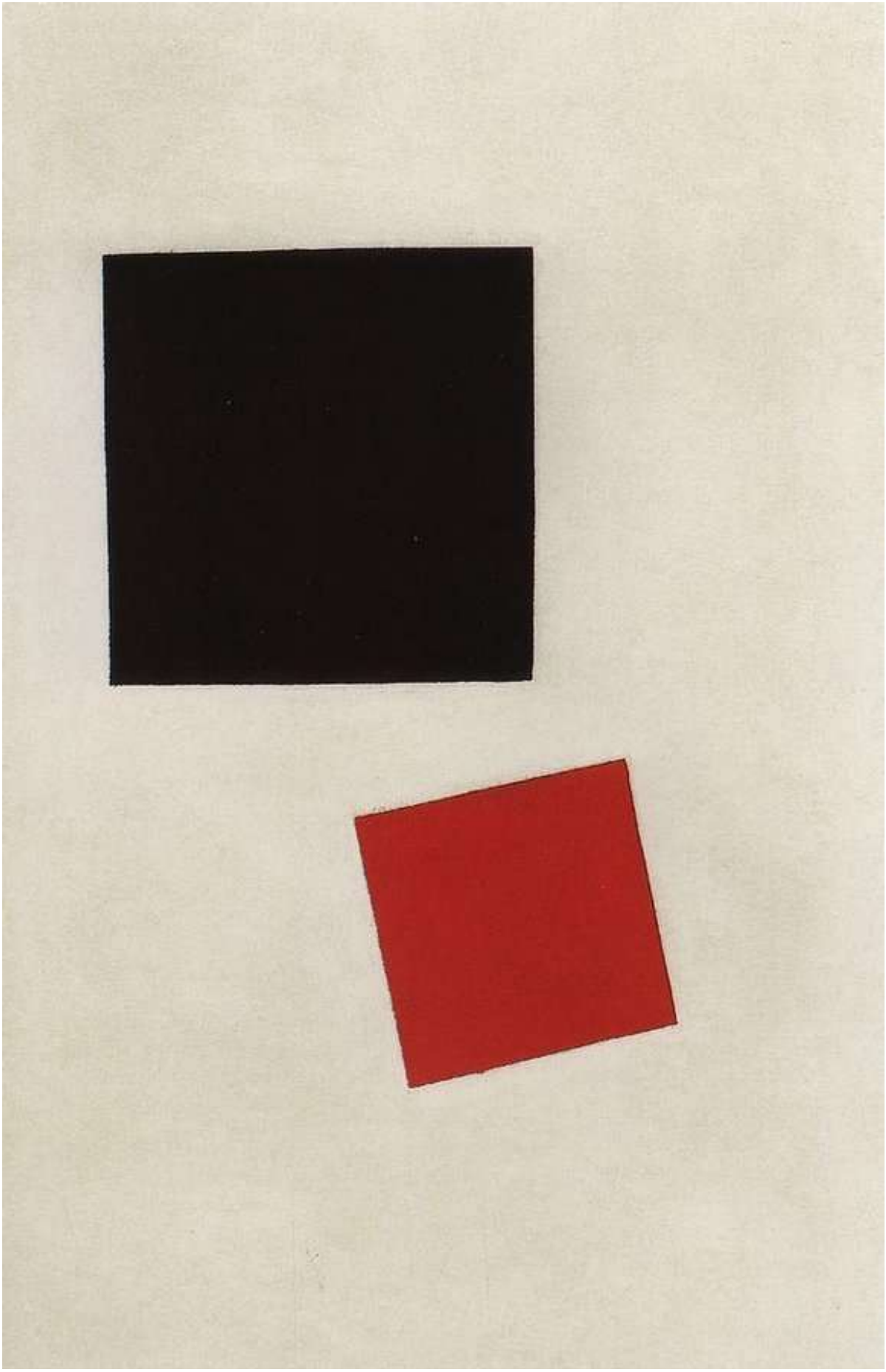

Рис. 2. Второе исходное изображение

Рис. 3. Третье исходное изображение

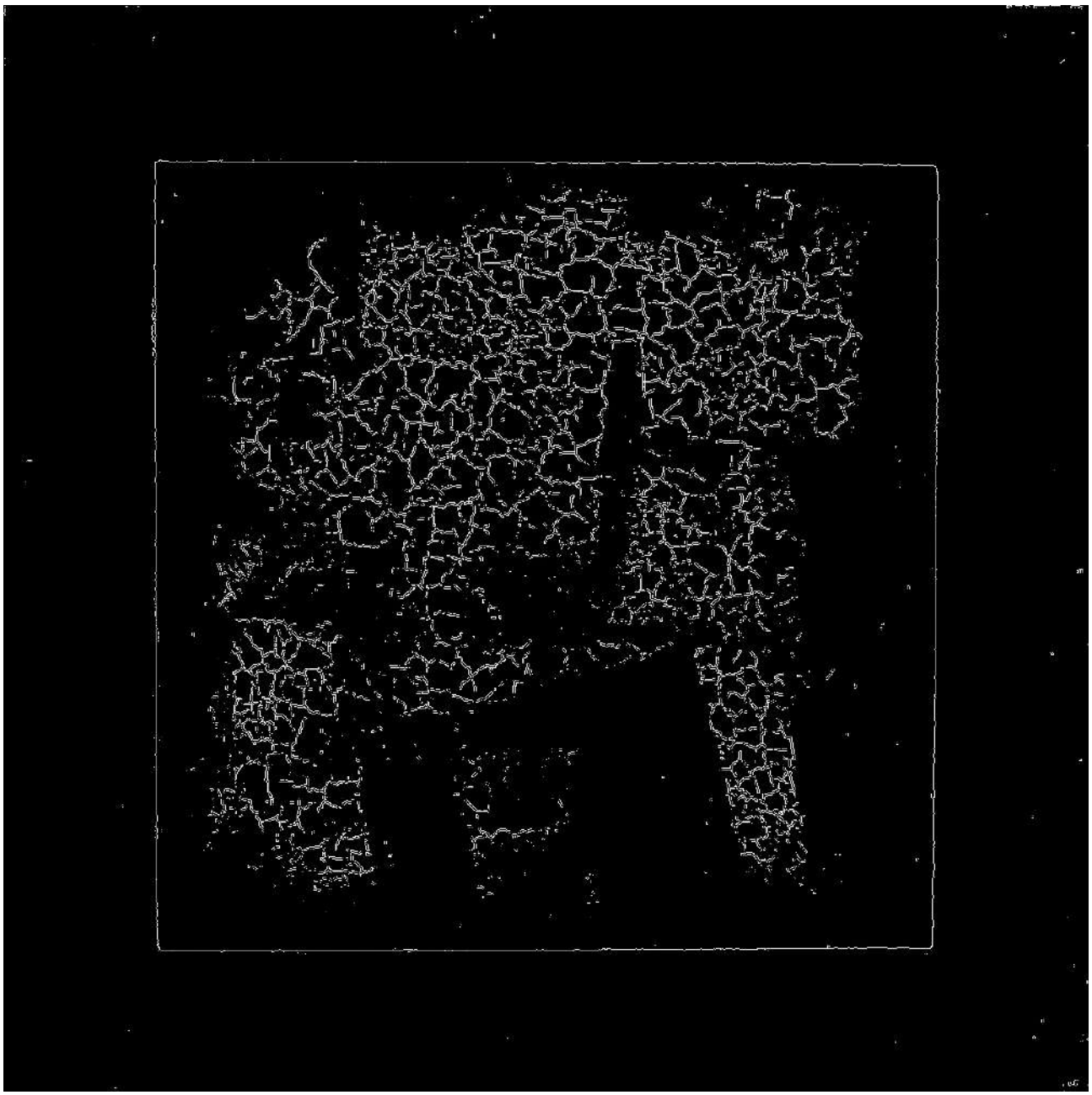

Рис. 4. Первое изображение, обработанное алгоритмом Кэнни

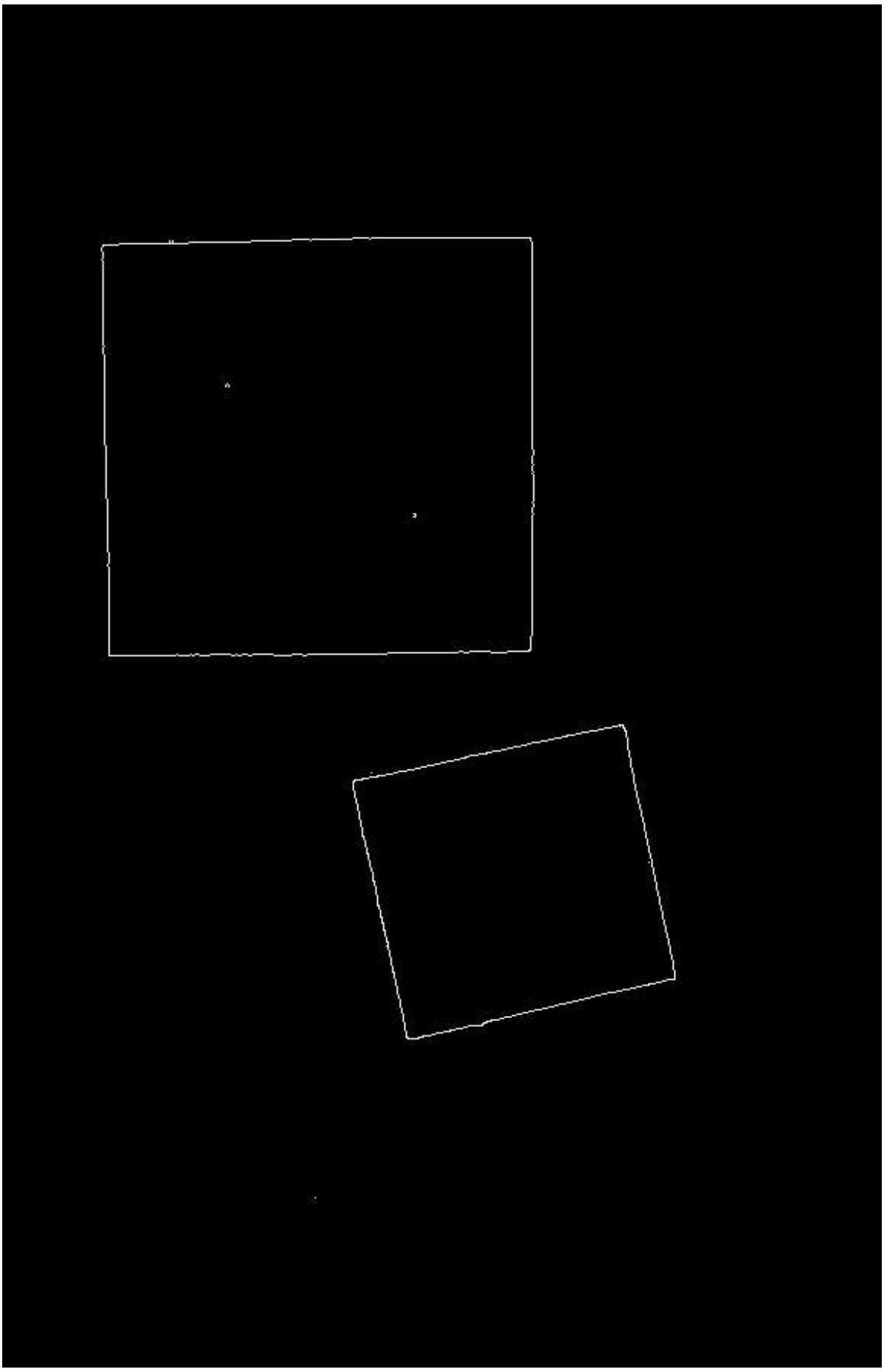

Рис. 5. Второе изображение, обработанное алгоритмом Кэнни

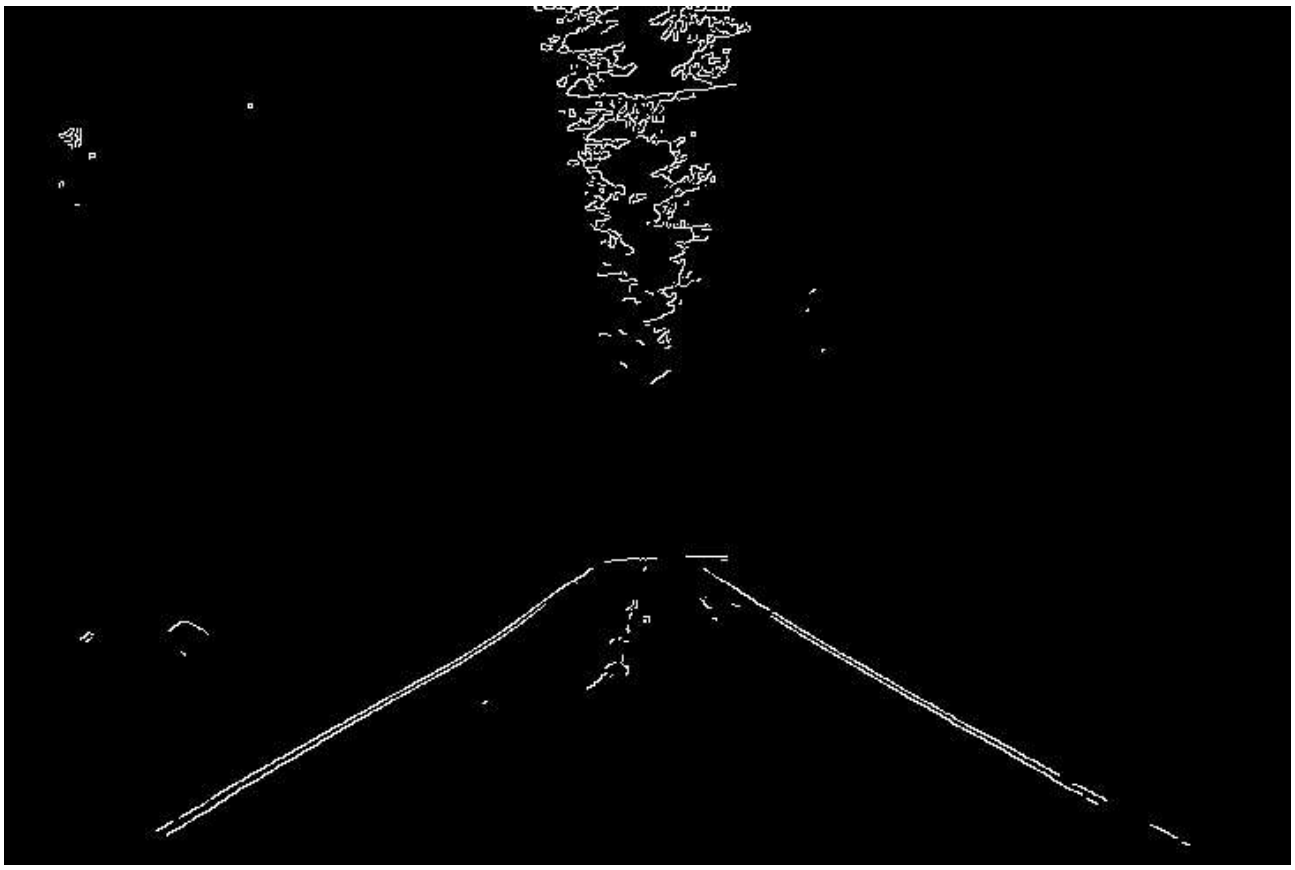

Рис. 6. Третье изображение, обработанное алгоритмом Кэнни

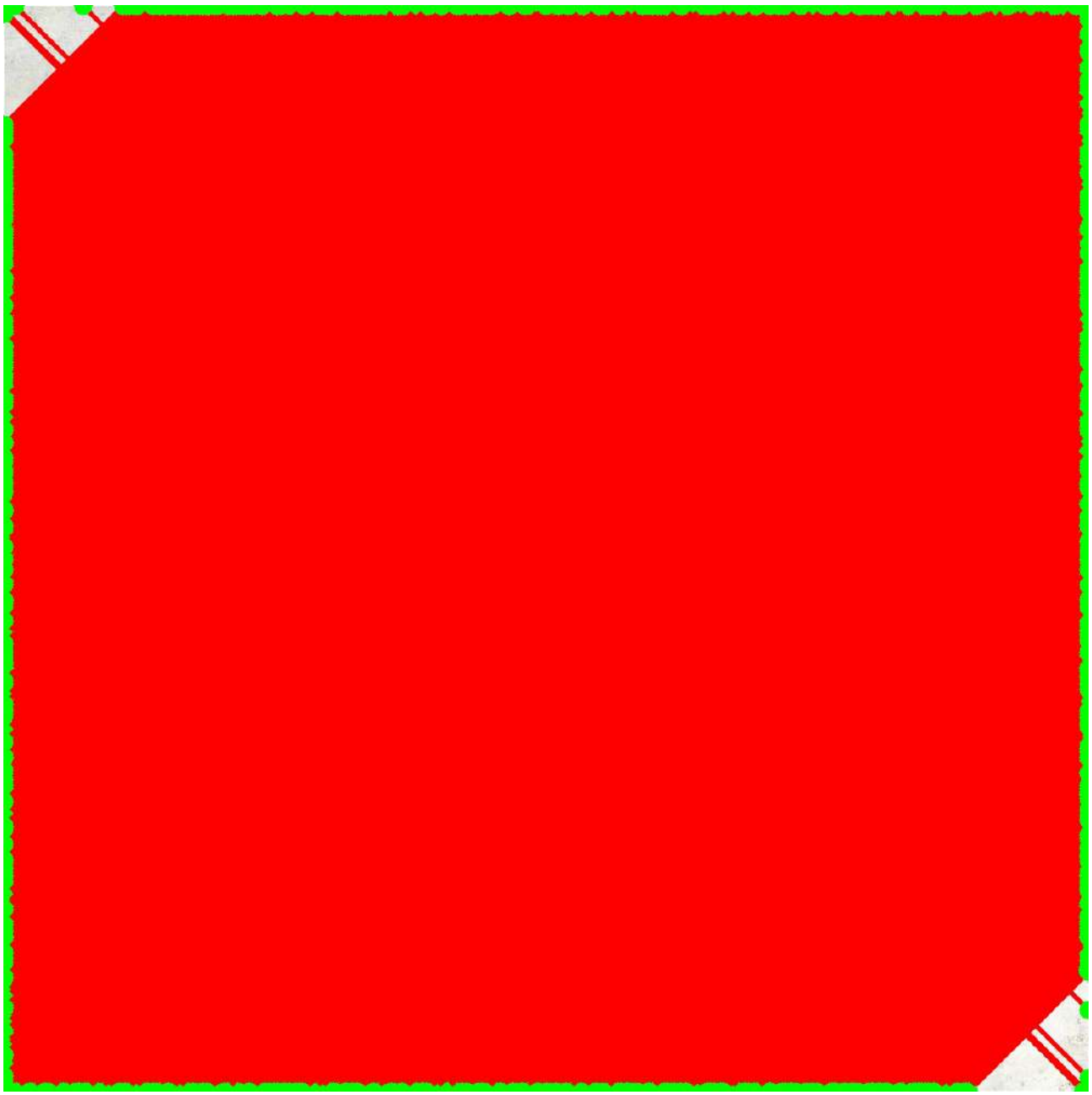

Рис. 7. Поиск прямых и точек на первом изображении

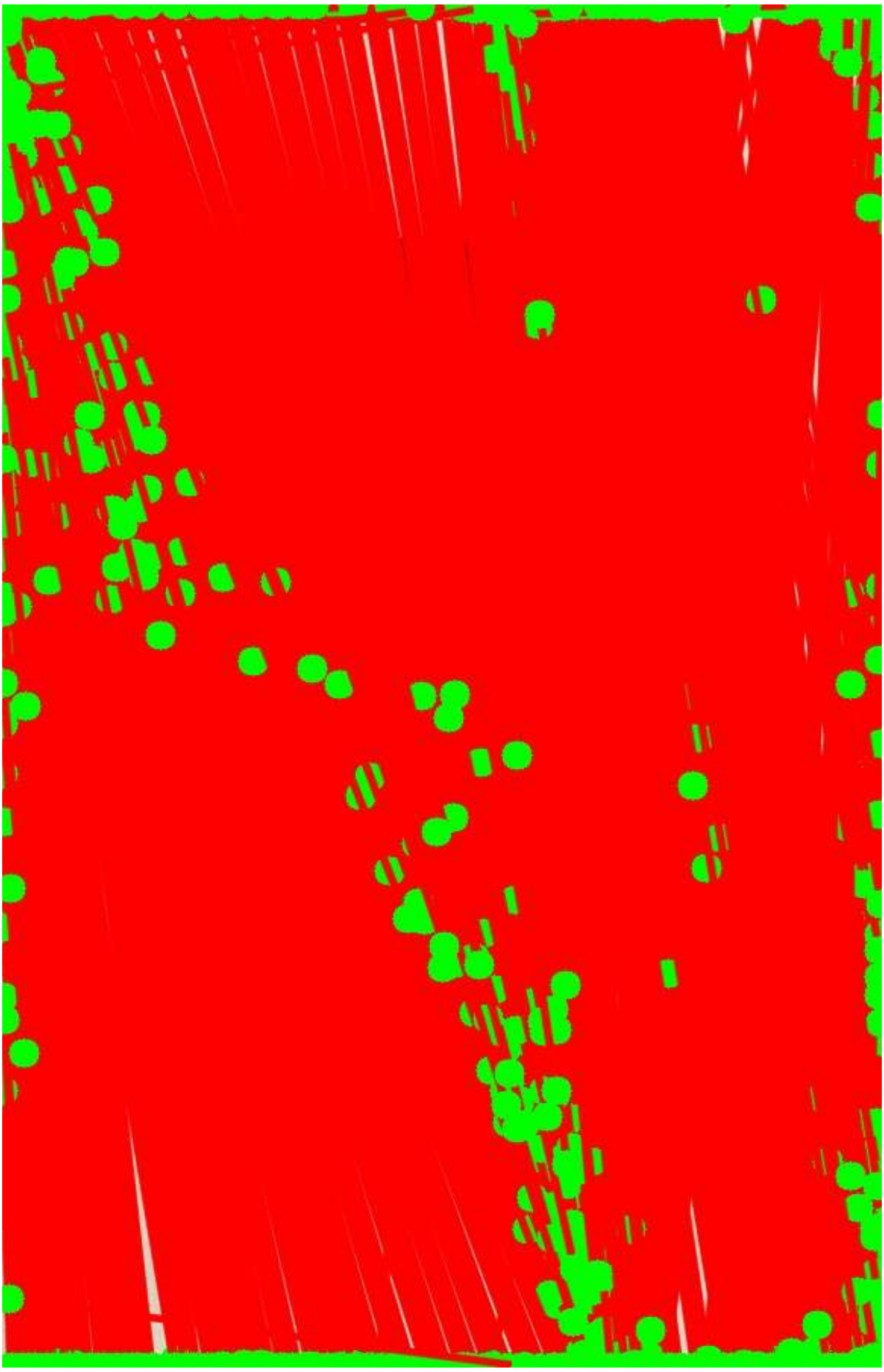

Рис. 8. Поиск прямых и точек на втором изображении

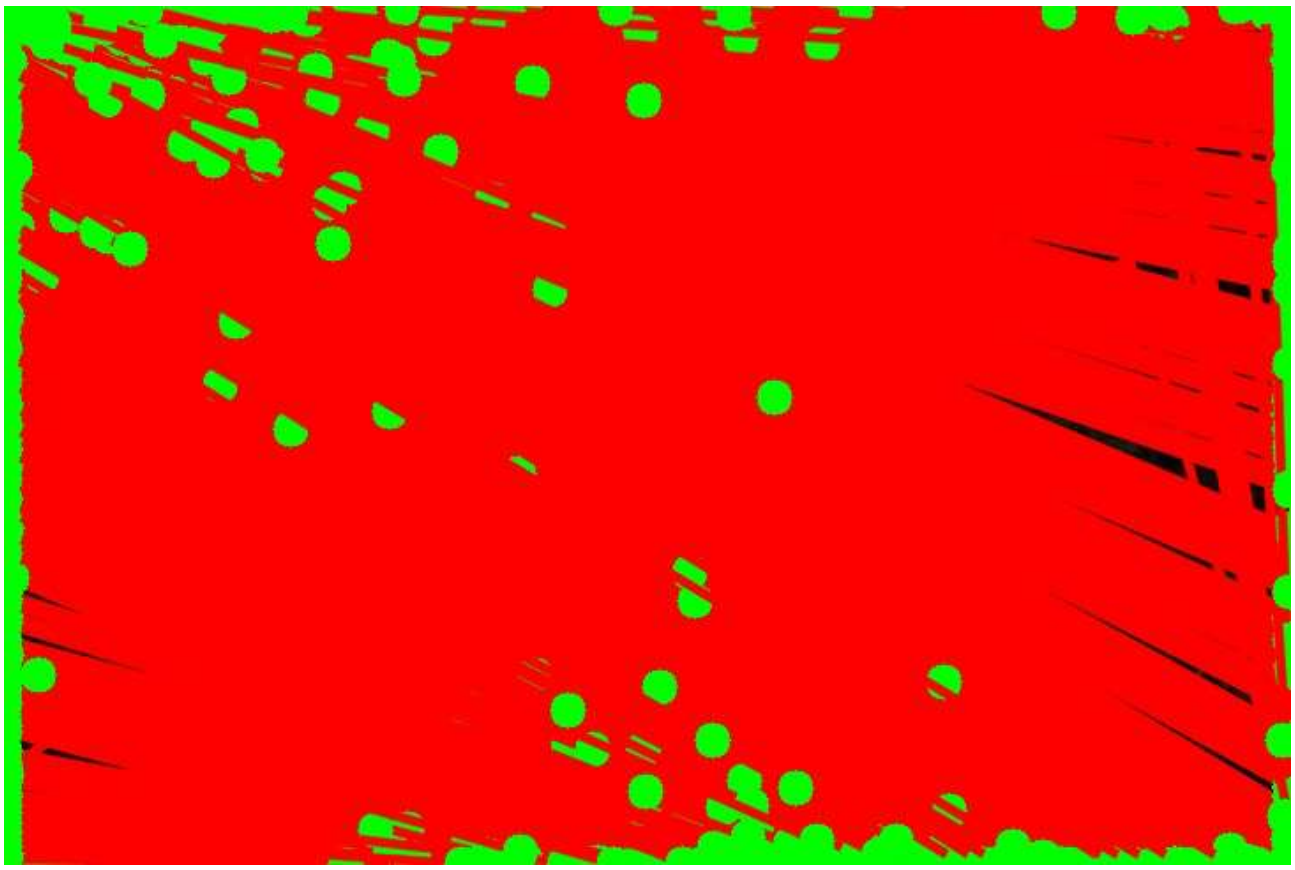

Рис. 9. Поиск прямых и точек на третьем изображении

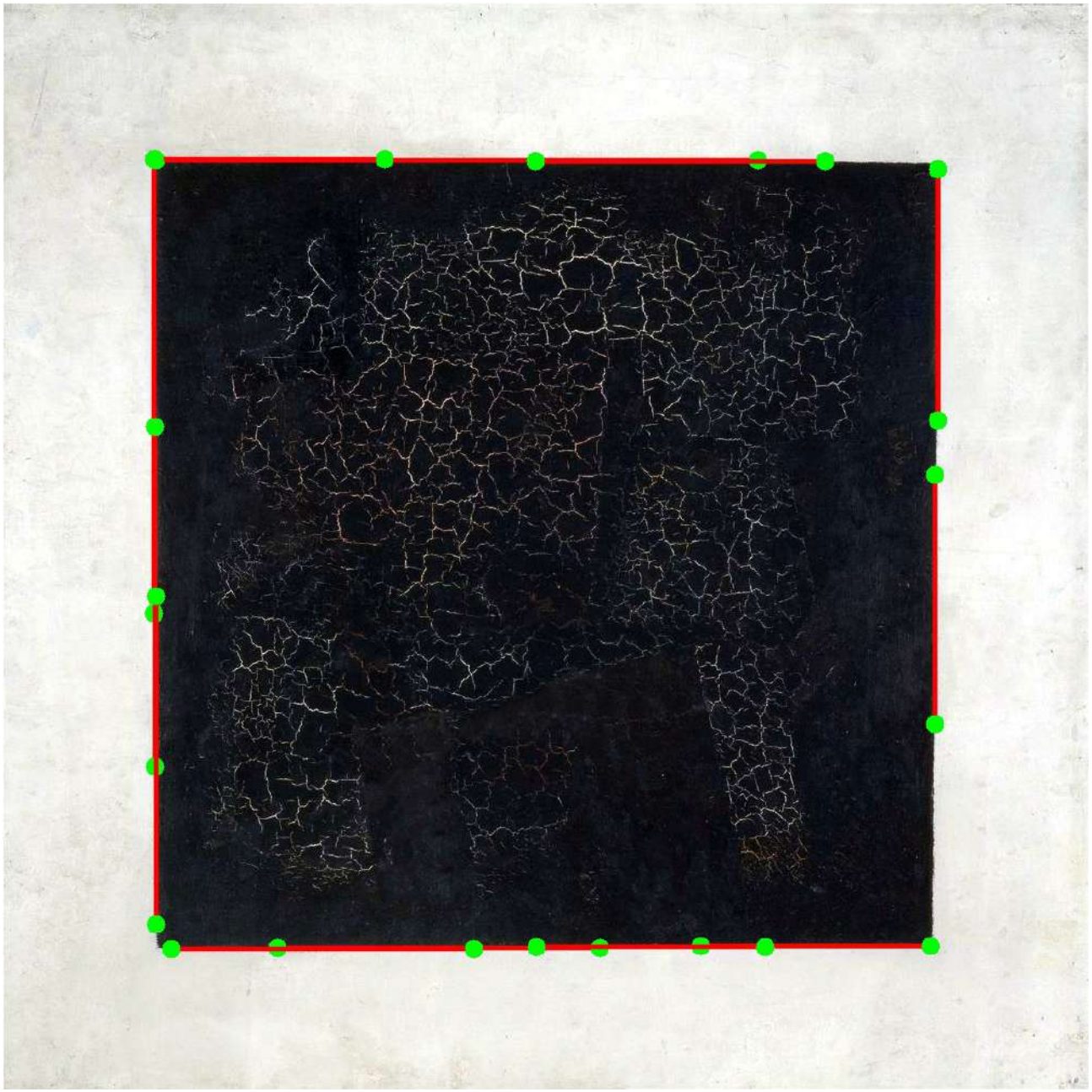

Рис. 10. Выделенные линии и точки на первом изображении с применением алгоритма Кэнни

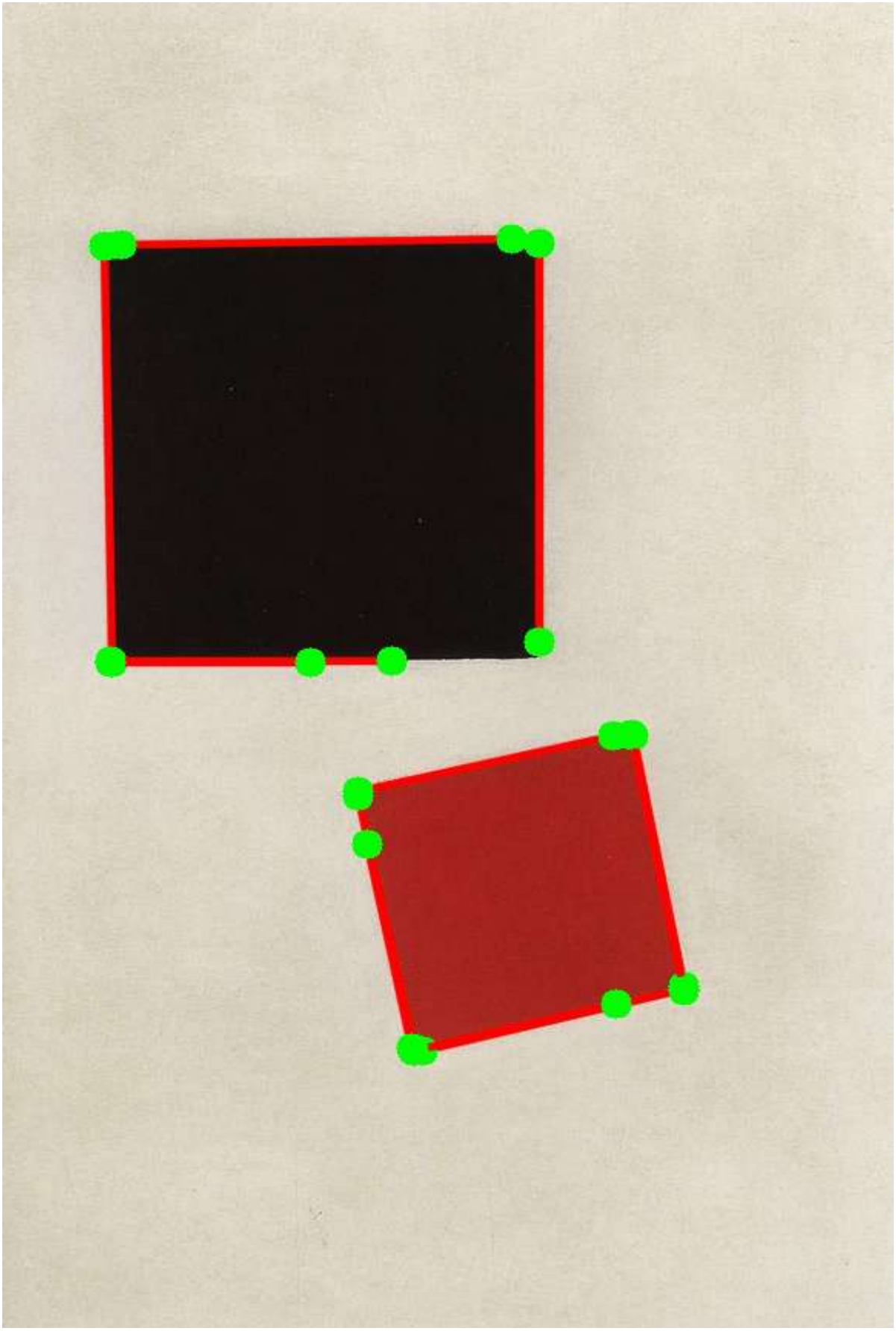

Рис. 11. Выделенные линии и точки на втором изображении с применением алгоритма Кэнни

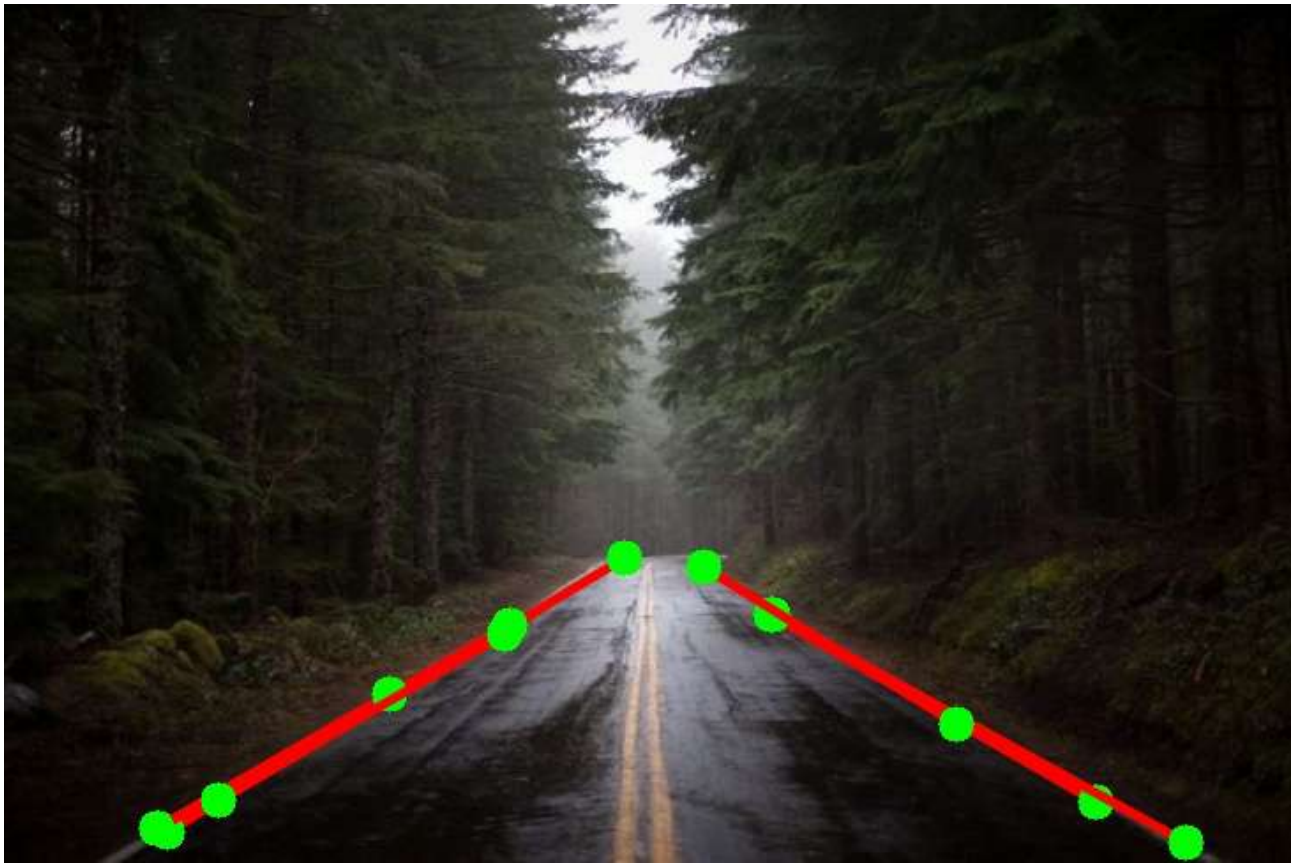

Рис. 12. Выделенные линии и точки на третьем изображении
с применением алгоритма Кэнни

Теперь наш робот станет точнее ориентироваться в пространстве и лучше понимать реальность. Кроме того, по полученным результатам мы поняли, что использование «обычной» картинки без дифференциального оператора просто бессмысленно.

Результаты измерений:

Длина самого длинного отрезка 1-й картинки: 1689

Длина самого короткого отрезка 1-й картинки: 185

Количество найденный прямых на 1-й картинке: 1483

Длина самого длинного отрезка 1-й картинки с использованием алгоритма Кэнни: 663

Длина самого короткого отрезка 1-й картинки с использованием алгоритма Кэнни: 252
Количество найденных прямых на 1-й картинке с использованием алгоритма Кэнни: 12

Длина самого длинного отрезка 2-й картинки: 1009
Длина самого короткого отрезка 2-й картинки: 155
Количество найденный прямых на 2-й картинке: 654

Длина самого длинного отрезка 2-й картинки с использованием алгоритма Кэнни: 284
Длина самого короткого отрезка 2-й картинки с использованием алгоритма Кэнни: 130
Количество найденных прямых на 2-й картинке с использованием алгоритма Кэнни: 12

Длина самого длинного отрезка 3-й картинки: 849
Длина самого короткого отрезка 3-й картинки: 156
Количество найденный прямых на 3-й картинке: 566

Длина самого длинного отрезка 3-й картинки с использованием алгоритма Кэнни: 301
Длина самого короткого отрезка 3-й картинки с использованием алгоритма Кэнни: 152
Количество найденных прямых на 3-й картинке с использованием алгоритма Кэнни: 7

Поиск окружностей. Давайте выберем три картинки с окружностями. Мы осуществим поиск окружностей как определенного радиуса, так и из

заданного диапазона с помощью алгоритма Хафа. Далее мы отразим найденные окружности на исходном изображении.

Код программы:

```
// drawing hough circles
cv::Mat hough_circles(cv::Mat image_input, int threshold, int j, int R1, int R2, bool
use_canny = 0, int thresh_canny_1 = 0, int thresh_canny_2 = 0){
    cv::Mat image;
    image_input.copyTo(image);
    cv::Mat image_gray;
    cv::cvtColor(image, image_gray, cv::COLOR_BGR2GRAY);

    if (use_canny){
        cv::Mat canny;
        cv::Canny(image_gray, canny, thresh_canny_1, thresh_canny_2);
        canny.copyTo(image_gray);
        cv::imwrite("canny_" + std::to_string(j) + ".jpg", canny);
    }

    std::vector<cv::Vec3f> circles;
    cv::HoughCircles(image_gray, circles, cv::HOUGH_GRADIENT, 1, 20, 100, 30, R1, R2);


    for( size_t i = 0; i < circles.size(); ++i){
        cv::Vec3i c = circles[i];
        cv::Point center = cv::Point(c[0], c[1]);
        circle(image, center, 1, cv::Scalar(0,100,100), 3, cv::LINE_AA);
        int radius = c[2];
        circle(image, center, radius, cv::Scalar(255,0,255), 3, cv::LINE_AA);
    }

    return image;

}


int main(){
    std::string filename = "original_";
    std::string filename_ext = ".jpg";

    std::vector<int> threshold {400, 2, 20};
    std::vector<int> thresh_canny_1{250, 250, 230};
    std::vector<int> thresh_canny_2 {255, 255, 255};

    // saving images
    for (int i = 1; i < 4; ++i){
        cv::Mat image = cv::imread(filename + std::to_string(i) + filename_ext);

        // drawing circles with R = 103
        cv::imwrite("circles_R=103_" + std::to_string(i) + ".jpg", hough_circles(image,
threshold[i-1], i, 100, 105));
```

```
        // drawing circles with R in range(100, 120)
        cv::imwrite("circles_R=100-120_" + std::to_string(i) + ".jpg",
hough_circles(image, threshold[i-1], i, 100, 120));


        // drawing circles with differential operator with R = 103
        cv::imwrite("circles_canny_R=103_" + std::to_string(i) + ".jpg",
            hough_circles(image, threshold[i-1], i, 100, 105, true, thresh_canny_1[i-1],
thresh_canny_2[i-1]));
        // drawing circles with differential operator with R in range (100, 120)
        cv::imwrite("circles_canny_R=100-120_" + std::to_string(i) + ".jpg",
            hough_circles(image, threshold[i-1], i, 100, 120, true, thresh_canny_1[i-1],
thresh_canny_2[i-1]));

    }

    return 0;
}
```

В этом случае, кажется, результат на «обычном» изображении выглядит лучше. Возможно, параметры для алгоритма Кэнни подобраны некорректным образом. Из-за этого определение границ не оправдало наших ожиданий.

Исходное изображение и выход программы:

Рис. 13. Первое исходное изображение

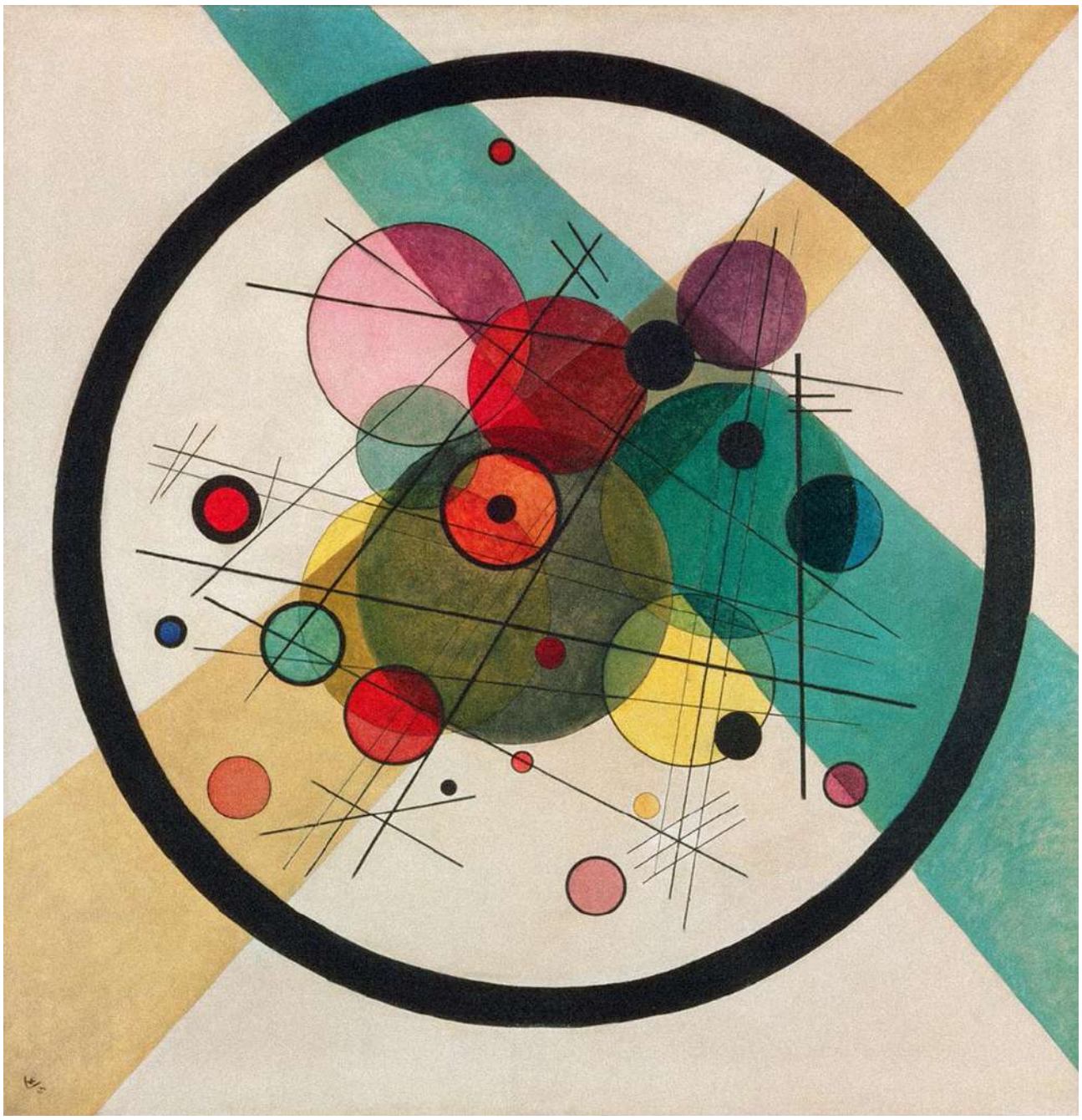

Рис. 14. Второе исходное изображение

Рис. 15. Третье исходное изображение

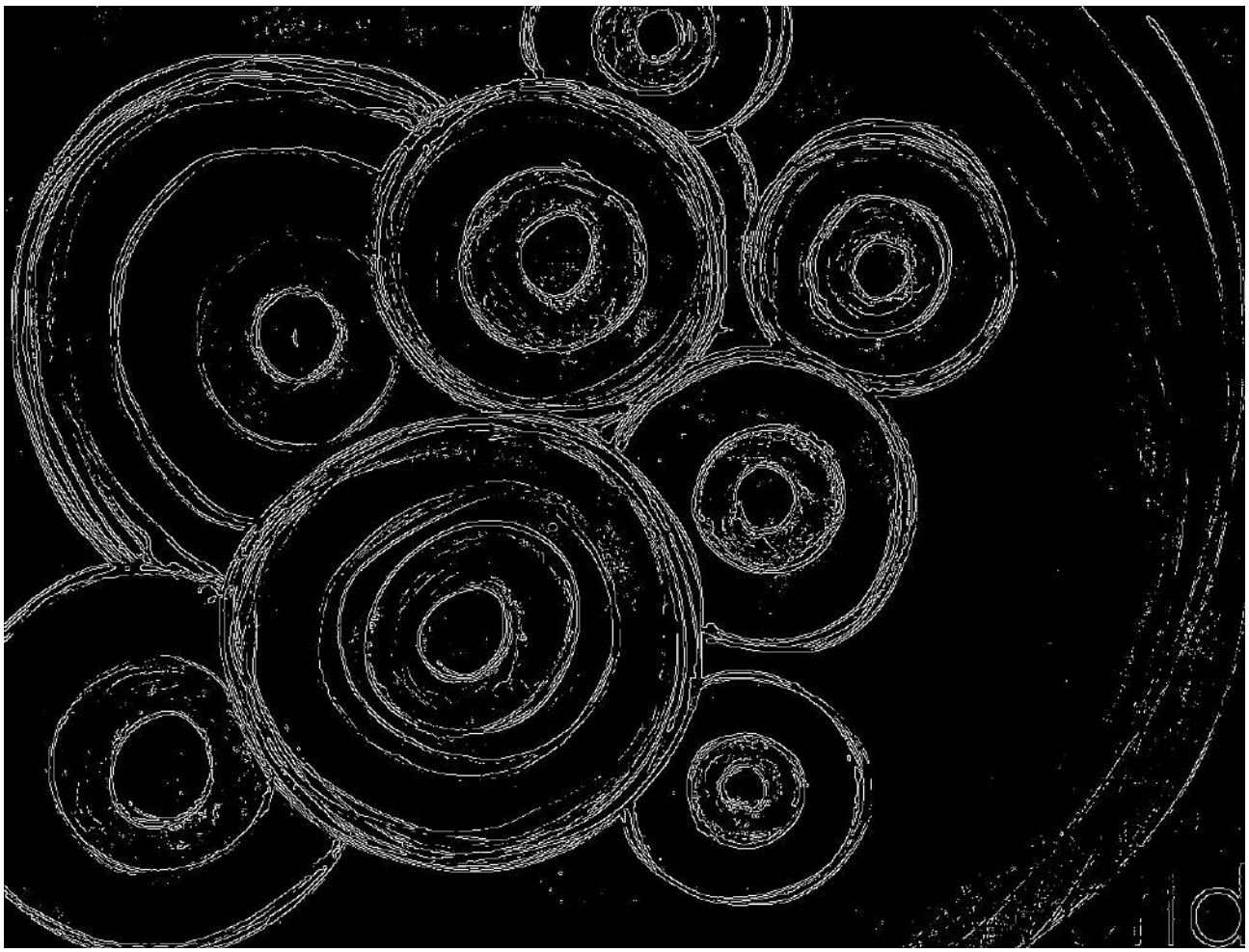

Рис. 16. Первое изображение, обработанное алгоритмом Кэнни

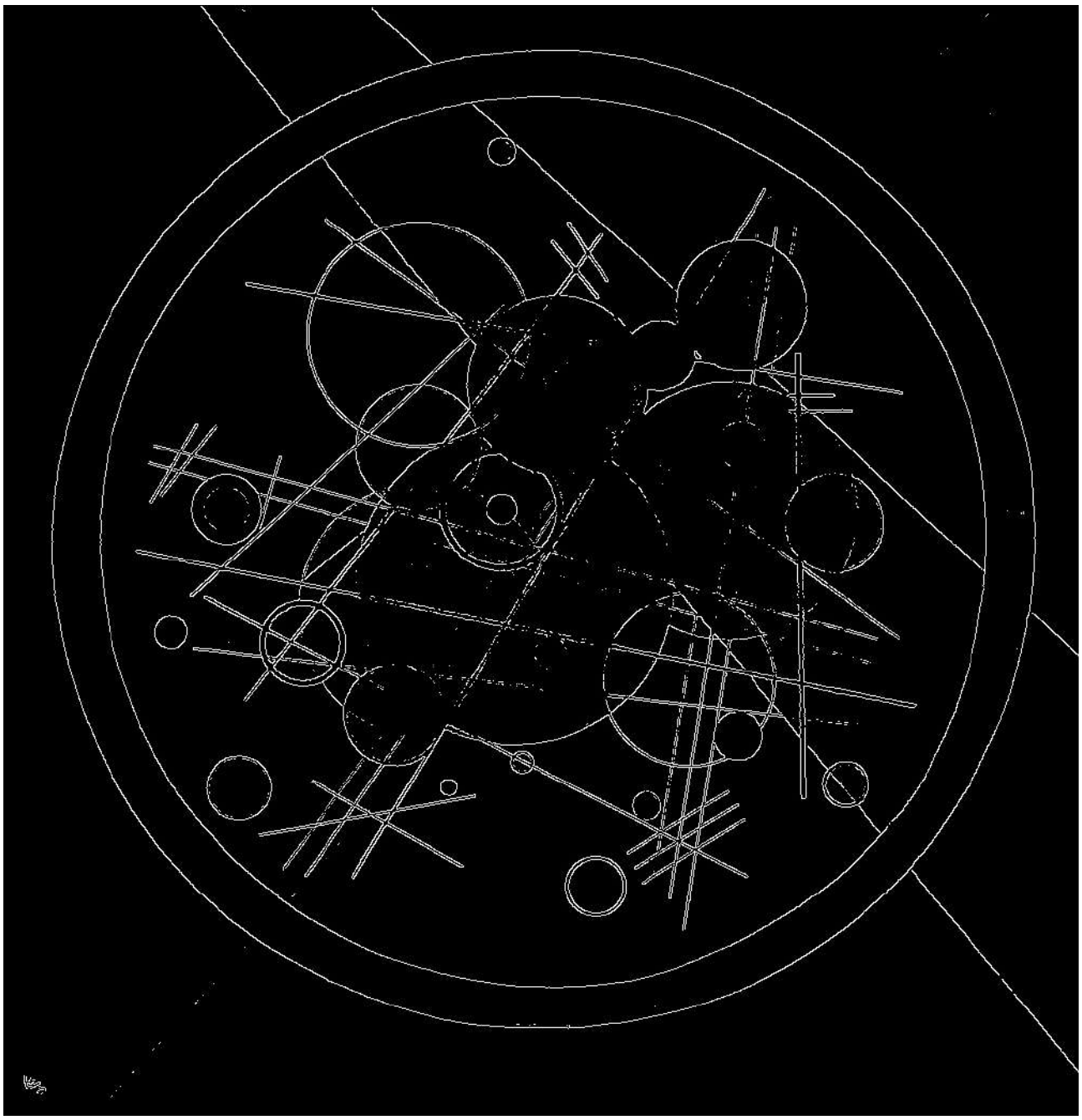

Рис. 17. Второе изображение, обработанное алгоритмом Кэнни

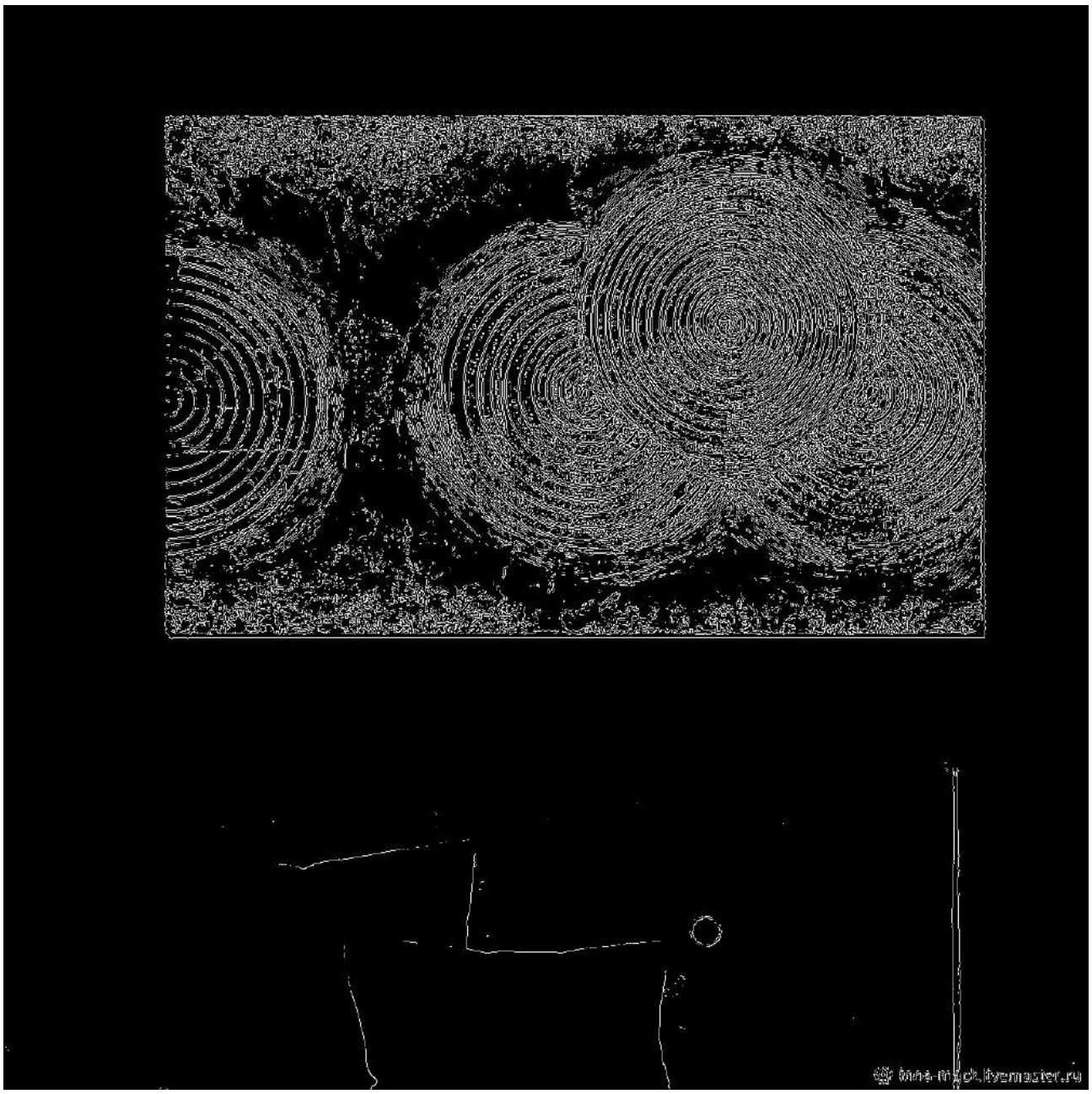

Рис. 18. Третье изображение, обработанное алгоритмом Кэнни

Рис. 19. Отраженные окружности $R = 103$ на первом изображении

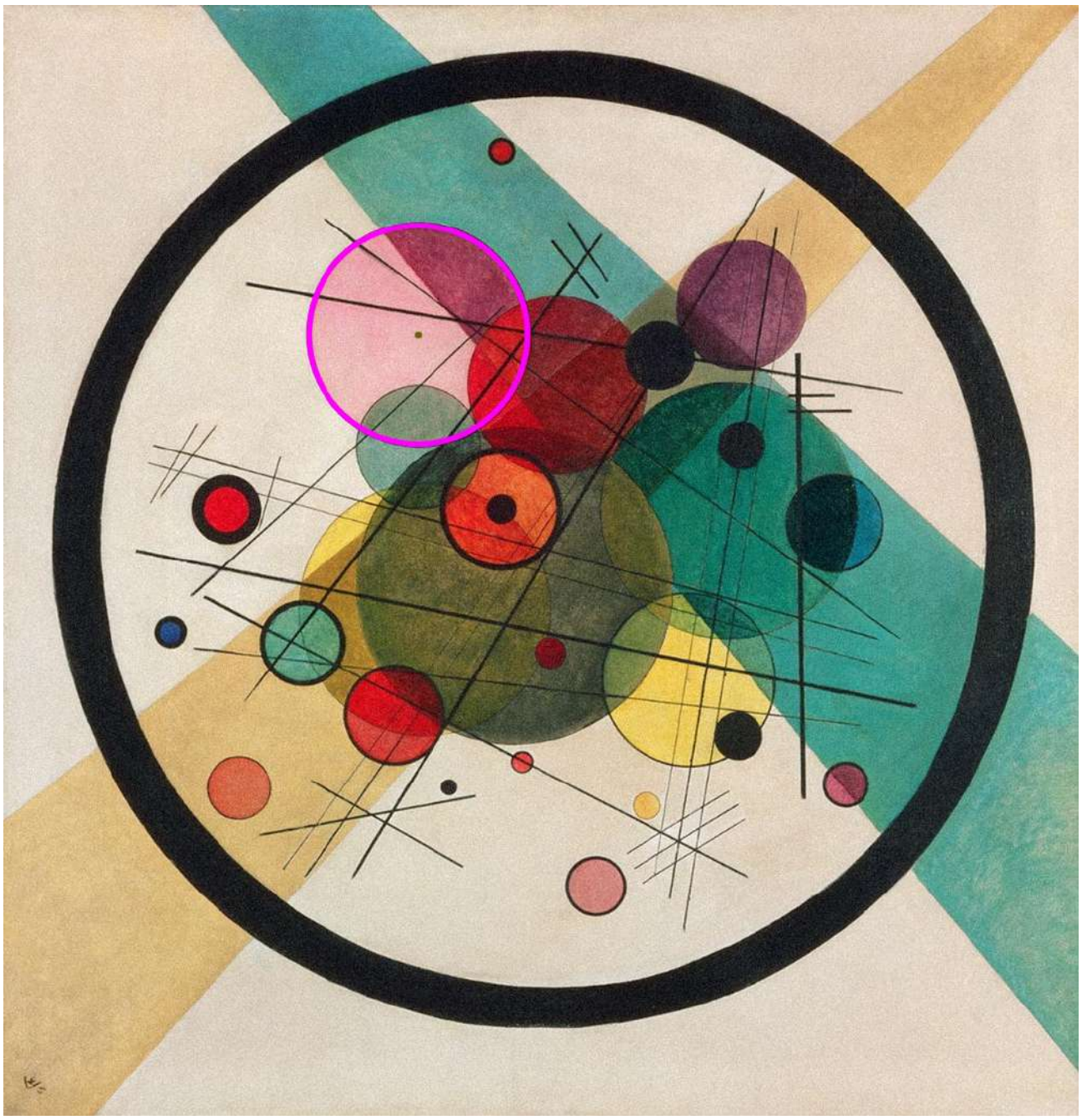

Рис. 20. Отраженные окружности $R = 103$ на втором изображении

Рис. 21. Отраженные окружности $R = 103$ на третьем изображении

Рис. 22. Отраженные окружности $R \in [100, 120]$ на первом изображении

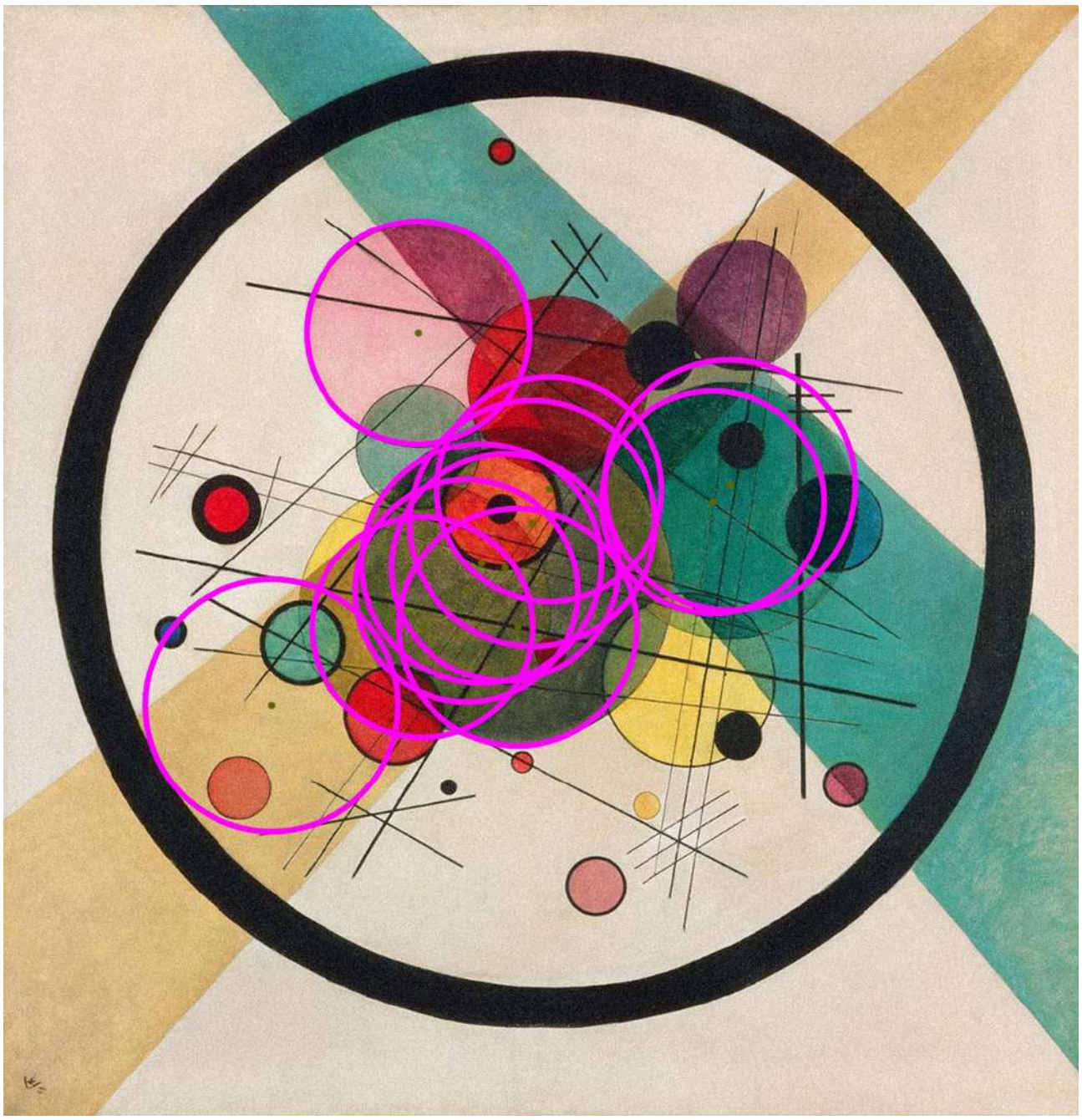

Рис. 23. Отраженные окружности $R \in [100, 120]$ на втором изображении

Рис. 24. Отраженные окружности $R \in [100, 120]$ на третьем изображении

Рис. 25. Отраженные окружности $R = 103$ на первом изображении с применением алгоритма Кэнни

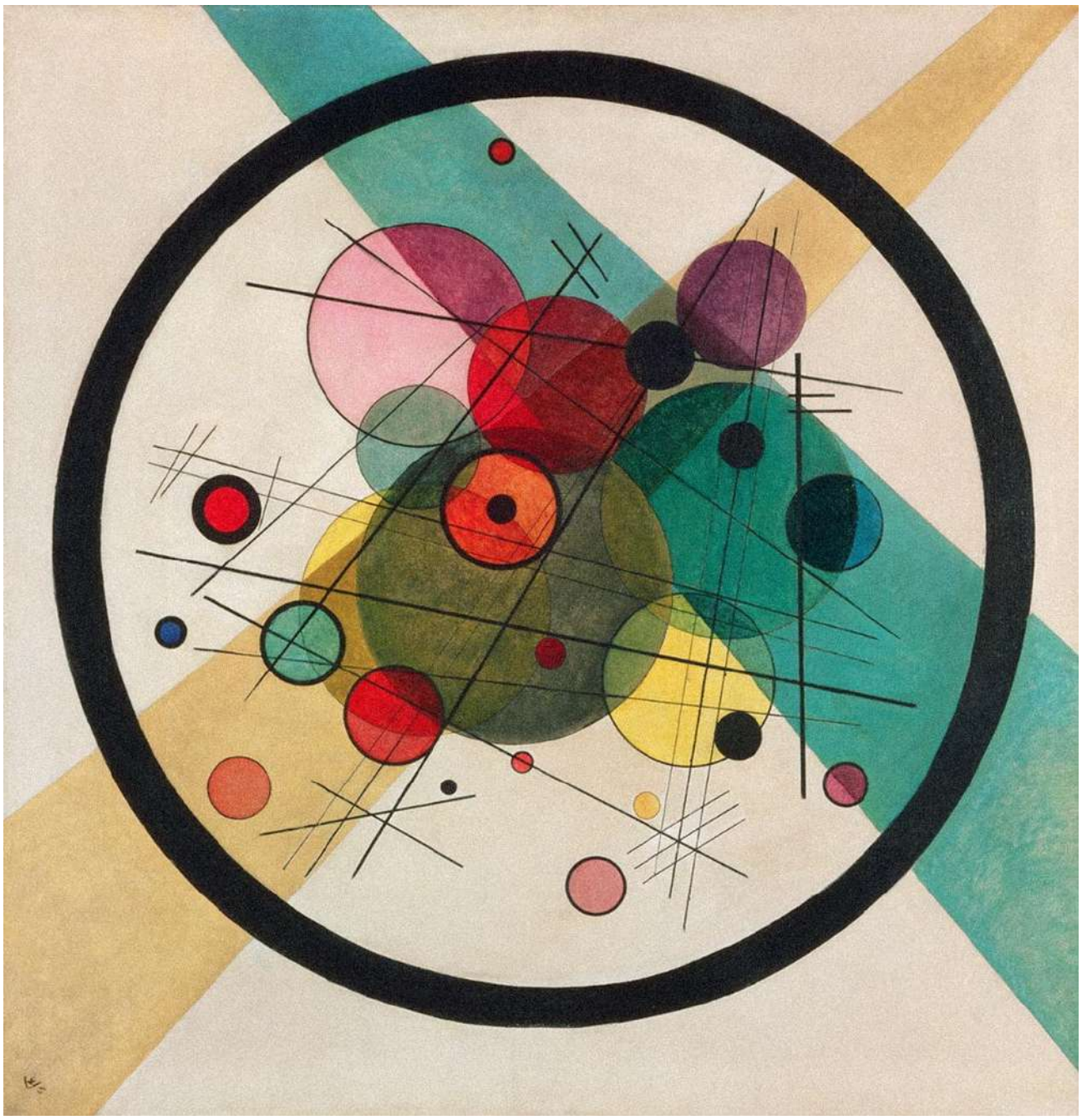

Рис. 26. Отраженные окружности $R = 103$ на втором изображении с применением алгоритма Кэнни

Рис. 27. Отраженные окружности $R = 103$ на третьем изображении с применением алгоритма Кэнни

Рис. 28. Отраженные окружности $R \in [100, 120]$ на первом изображении с применением алгоритма Кэнни

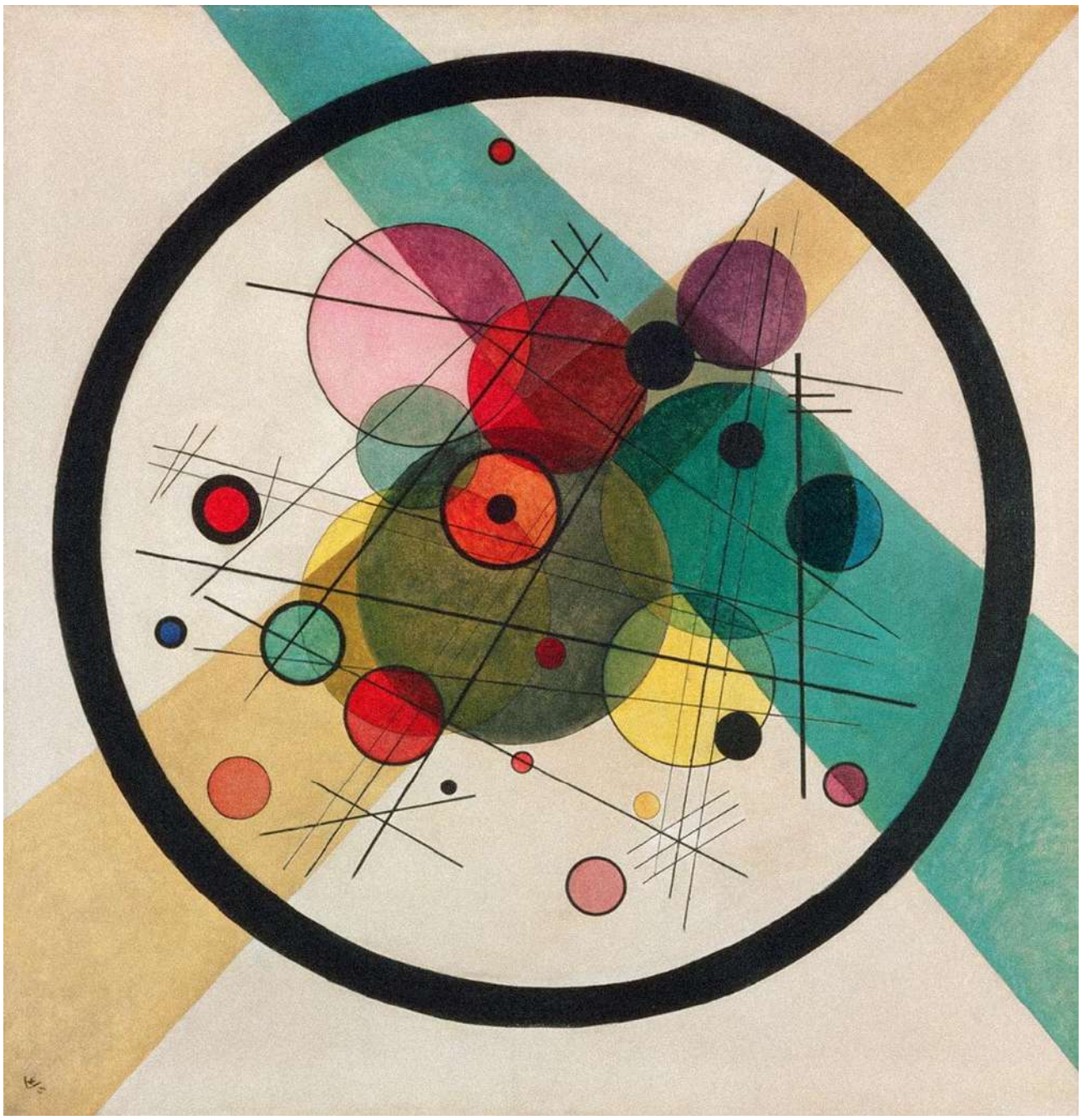

Рис. 29. Отраженные окружности $R \in [100, 120]$ на втором изображении с применением алгоритма Кэнни

Рис. 30. Отраженные окружности $R \in [100, 120]$ на третьем изображении с применением алгоритма Кэнни

Выводы о проделанной работе:

В ходе лабораторной работы все также была использована библиотека OpenCV и язык программирования C++.

Сегодня мы рассматривали преобразование Хафа, основанное на поиске общих геометрических мест точек (ГМТ).

Сначала мы брали три исходные картинки. В первом пункте требовалось отразить линии на исходном изображении и на изображении, полученном после использования дифференциального оператора. В нашем случае – алгоритма Кэнни. По полученным результатам мы поняли, что использование «обычной» картинки без дифференциального оператора оказалось бесплодно. Выделенные линий без алгоритма Кэнни не представляли никакого интереса из-за того, что были больше похожи на хаос. Упорядоченности результата не наблюдалось.

Кроме этого, мы посчитали длины отрезков и вывели количество прямых для двух видов изображений: «обычных» и картинок с заранее выполненным алгоритмом Кэнни.

Во второй же части работы мы пришли к противоположению: итоговые картинки выглядели правдоподобнее без алгоритма Кэнни. Мы предположили, что это было связано с неверным подбором параметров, из-за которых границы изображений вышли «кривые».

В целом, результаты лабораторной работы нас порадовали. Мы смогли добиться приемлемых результатов.

Вопросы к защите лабораторной работы

1. Какая идея лежит в основе преобразования Хафа?

Преобразование Хафа используется в компьютерном зрении и обработке изображений для нахождения параметров линий или форм на изображении. Так, основная идея заключается в том, что каждая точка в пространстве параметров (обычно параметры уравнения прямой в декартовой системе координат для поиска прямых линий) представляет собой прямую на изображении. Алгоритм

ищет точки на изображении, которые смогут быть частью прямой, и считает их в пространстве параметров. После этого проводится подсчет пересечений точек в пространстве параметров, чтобы определить параметры реальных линий или форм на изображении.

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

Да, преобразование Хафа можно использовать для поиска произвольных контуров. Например, если на изображении присутствуют формы или объекты, которые не могут быть описаны стандартными математическими уравнениями прямых линий или окружностей, преобразование Хафа все равно может быть применено для анализа их геометрических особенностей.

Рассмотренный нами алгоритм поиска прямых может быть использован для поиска любой кривой, описываемой в пространстве некоторой функцией с определенным числом параметров, что повлияет только на размерность этого пространства.

3. Что такое рекуррентное и обобщенное преобразования Хафа?

Рекуррентное преобразование Хафа представляет собой модификацию классического преобразования. Оно также использует алгоритм для обработки изображений и анализа контуров. Это преобразование требует меньше памяти и времени, что делает его более привлекательным при анализе изображений.

Обобщенное преобразование Хафа: идея еще более общая. Оно расширяет применение преобразования Хафа на различные типы форм, не ограничиваясь только поиском прямых или окружностей. Обобщенное преобразование может быть использовано для детекции и анализа форм сложной геометрии, таких как эллипсы, параболы, различные нестандартные контуры и даже для обнаружения объектов на изображениях с использованием хаотичных или неизвестных форм.

4. Какие бывают способы параметризации в преобразовании Хафа?

Способ параметризации в преобразовании Хафа выбирается в зависимости от конкретной задачи обнаружения объектов или форм на изображении.

Перечислим три основные из них:

1. Для поиска прямых используется параметризация в виде угла наклона и расстояния от начала координат до пересечения прямой с осью ординат. Так, параметризация прямой будет выражаться в виде двух параметров: угла и расстояния.
2. Для поиска окружностей: параметризация в зависимости от радиуса.
3. Для поиска иных форм: различные параметры, соответствующие геометрическим свойствам этих форм (эллипса, параболы и другой нестандартной или сложной геометрии).