



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: ТЕХНИЧЕСКОЕ ЗРЕНИЕ

Отчет по лабораторной работе №5

“Преобразование Хафа”

Выполнили:

Новиков Дмитрий, ТЕХ.ЗРЕНИЕ 1.1

Преподаватель:

Шаветов С. В.

Санкт-Петербург, 2024

Содержание

1. Поиск прямых	2
1.1. Изображение №1	5
1.1.1. Исходное изображение	6
1.1.2. Обработанное алгоритмом Кэнни	7
1.2. Изображение №2	8
1.2.1. Исходное изображение	9
1.2.2. Обработанное алгоритмом Кэнни	10
1.3. Изображение №3	11
1.3.1. Исходное изображение	12
1.3.2. Обработанное алгоритмом Кэнни	13
2. Поиск окружностей	13
2.1. Изображение №1	16
2.1.1. Исходное изображение	16
2.1.2. Обработанное алгоритмом Кэнни	17
2.2. Изображение №2	18
2.2.1. Исходное изображение	19
2.2.2. Обработанное алгоритмом Кэнни	21
2.3. Изображение №3	22
2.3.1. Исходное изображение	23
2.3.2. Обработанное алгоритмом Кэнни	24
3. Ответы на вопросы	24

[Исходный код на GitHub.](#)

Для удобства отладки и сборки используется средство автоматизации сборки ПО CMake:

```
cmake_minimum_required(VERSION 2.8)

project( CV_LW4 )
find_package( OpenCV REQUIRED )

include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( ${PROJECT_NAME} src/lab2.cpp )

target_link_libraries( ${PROJECT_NAME} ${OpenCV_LIBS} )
```

Листинг 1: CMakeLists.txt для сборки проекта

1. Поиск прямых

Для удобства работы и модульности Л/Р были созданы следующие функции и структура:

1. Структура func, которая предназначена для инкапсуляции результатов преобразования Хафа различными способами. Также данная структура помогает избежать использования boost::any для получения различных типов данных из функций, которые будут описаны ниже. Главный недостаток boost::any заключается в потери "типов" при преобразовании.
2. Функции "canny hough transform" и "ordinary hough transform" применяют преобразования Хафа к изображению, полученному после применения оператора Кэнни и обычной бинаризации соответственно.
3. Статическая функция func::comp используется как компаратор для сравнения длин найденных линий в функциях std::max element и std::min element.
4. Статическая функция func::length line возвращает длину линии.

```
struct func{
    func(std::vector<cv::Vec4i>& lines, cv::Mat& image): lines{lines},
    image{image} {};
    std::vector<cv::Vec4i> lines;
    cv::Mat image;
    static bool comp(cv::Vec4i&, cv::Vec4i&);
```

```

static uint lenght_line(cv::Vec4i&);

};

func canny_hough_tranform(cv::Mat& image){

    cv::Mat image_new, image_out = image.clone();
    cv::Canny(image, image_new, 50, 200);
    std::vector<cv::Vec4i> linesP;

    HoughLinesP(image_new, linesP, 1, CV_PI / 150, 80, 40, 4);

    for (int i = 0; i < linesP.size(); ++i){
        cv::Vec4i I = linesP[i];
        cv::line(image_out, cv::Point(I[0], I[1]),
                 cv::Point(I[2], I[3]),
                 cv::Scalar(0, 255, 0),
                 1, cv::LINE_AA);

        cv::circle(image_out, cv::Point(I[0], I[1]), 1,
                   (127, 0, 60), 5,
                   cv::LINE_AA);

        cv::circle(image_out, cv::Point(I[2], I[3]), 1,
                   (0, 0, 255), 5,
                   cv::LINE_AA);
    }

    return func(linesP, image_out);
}

func ordinary_hough_tranform(cv::Mat& image){

    cv::Mat image_new, image_out = image.clone();
    cv::cvtColor(image, image_new, cv::COLOR_BGR2GRAY);

    cv::threshold(image_new, image_new, 50, 255, cv::THRESH_BINARY);

    std::vector<cv::Vec4i> linesP;

    HoughLinesP(image_new, linesP, 1, CV_PI / 180, 50, 50, 15);

    for (int i = 0; i < linesP.size(); ++i){
        cv::Vec4i I = linesP[i];
        cv::line(image_out, cv::Point(I[0], I[1]),
                 cv::Point(I[2], I[3]),
                 cv::Scalar(0, 255, 0),
                 1, cv::LINE_AA);

        cv::circle(image_out, cv::Point(I[0], I[1]), 1,
                   (127, 0, 60), 5,
                   cv::LINE_AA);
    }
}

```

```

        cv::circle(image_out, cv::Point(I[2], I[3]), 1,
                   (0, 0, 255), 5,
                   cv::LINE_AA);
    }

    return func(linesP, image_out);
}

bool func::comp(cv::Vec4i& l1, cv::Vec4i& l2){
    return (std::sqrt(std::pow(abs(l1[2] - l1[0]), 2) + std::pow(abs(l1[3]
    - l1[1]), 2)) <
            std::sqrt(std::pow(abs(l2[2] - l2[0]), 2) + std::pow(abs(l2[3]
    - l2[1]), 2)));
}

uint func::length_line(cv::Vec4i& l){
    return std::sqrt(std::pow(abs(l[2] - l[0]), 2) +
                    std::pow(abs(l[3] - l[1]), 2));
}

```

Листинг 2: Вспомогательные функции и структура

```

// GRayscale

cv::Mat image;
image = cv::imread(path + "/source/test.png", 1);

func output = ordinary_hough_transform(image);

std::cout << "MAX LENGTH = " <<
    func::length_line(*std::max_element(output.lines.begin(),
    output.lines.end(), func::comp)) <<
        "\nMIN LENGTH = " <<
    func::length_line(*std::min_element(output.lines.begin(),
    output.lines.end(), func::comp)) <<
        "\nNUMBER OF LINES = " << output.lines.size();

cv::imwrite(path + "/outputs/image1_ordinary.png", output.image);
cv::imshow("image", output.image);
cv::waitKey();

// Canny

cv::Mat image;
image = cv::imread(path + "/source/test.png", 1);

func output = canny_hough_transform(image);

std::cout << "MAX LENGTH = " <<

```

```

func::length_line(*std::max_element(output.lines.begin(),
output.lines.end(), func::comp)) <<
    "\nMIN LENGTH = " <<
func::length_line(*std::min_element(output.lines.begin(),
output.lines.end(), func::comp)) <<
    "\nNUMBER OF LINES = " << output.lines.size();

cv::imwrite(path + "/outputs/image1_canny.png", output.image);
cv::imshow("image", output.image);
cv::waitKey();

```

Листинг 3: Общий код для поиска прямых с помощью преобразования Хафа

Синие точки означают конец линии, черные - начало.

1.1. Изображение №1

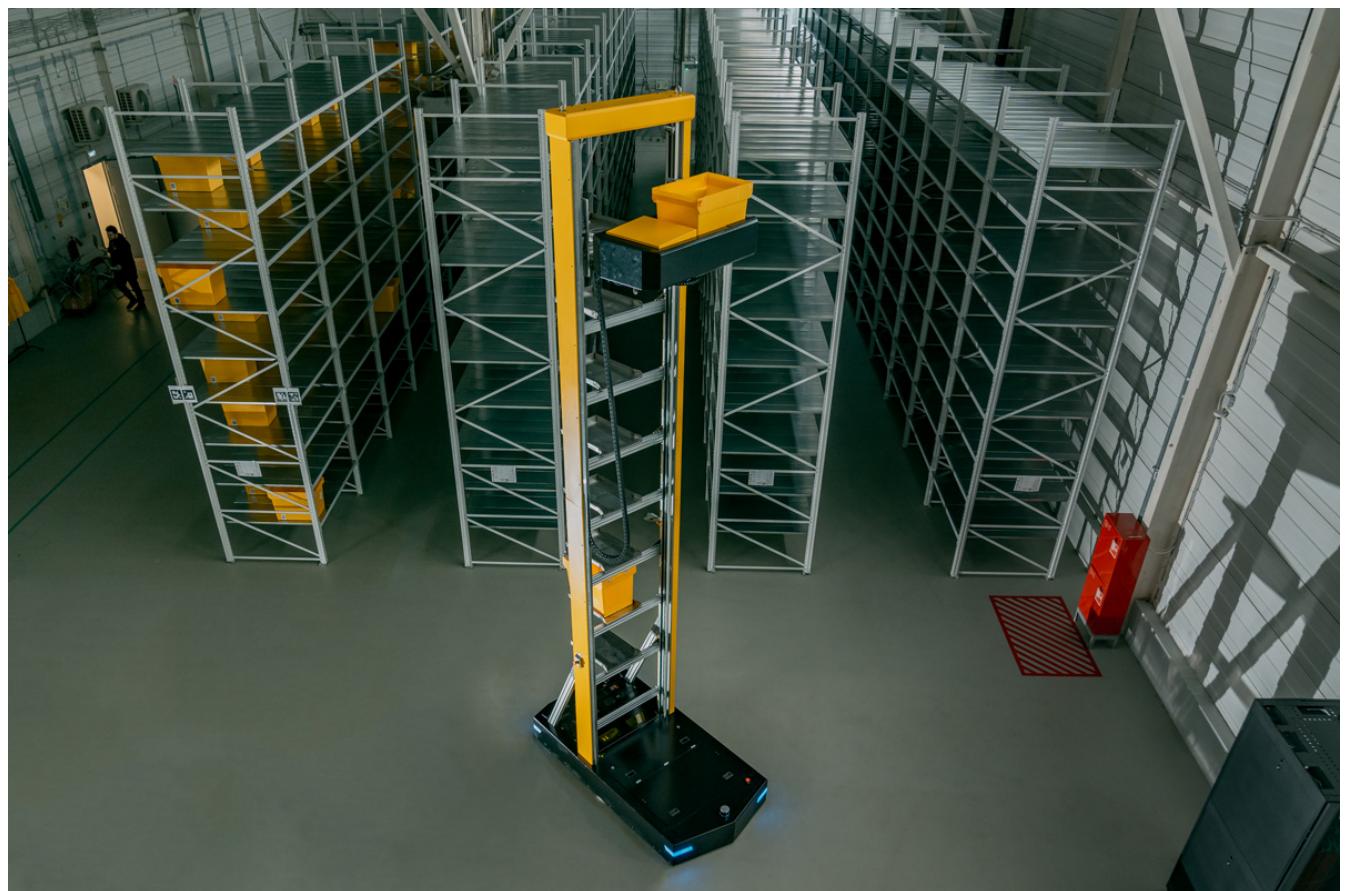


Рис. 1: Исходное изображение

1.1.1. Исходное изображение

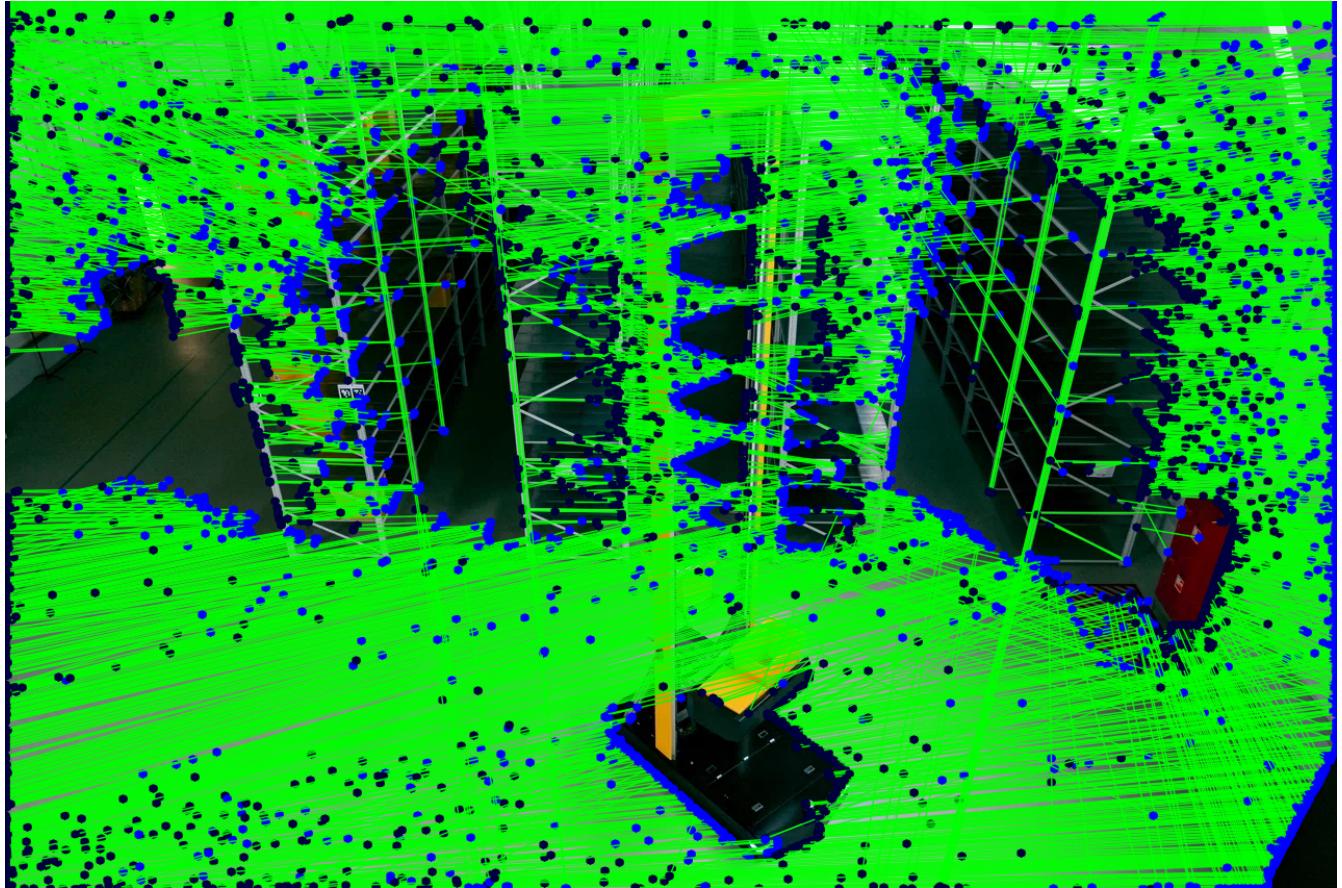


Рис. 2: Результат преобразования Хафа

MAX LENGTH = 1279

MIN LENGTH = 50

NUMBER OF LINES = 2946

Данное хаотичное поведение объясняется тем, что на вход преобразованию поступило простое бинаризированное по порогу изображение, на котором отсутствуют четкие границы (из-за ручного подбора порога), следовательно присутствует много точек, через которые можно провести линии.

1.1.2. Обработанное алгоритмом Кэнни



Рис. 3: Результат преобразования Хафа

MAX LENGTH = 329

MIN LENGTH = 40

NUMBER OF LINES = 314

1.2. Изображение №2



Рис. 4: Исходное изображение

1.2.1. Исходное изображение

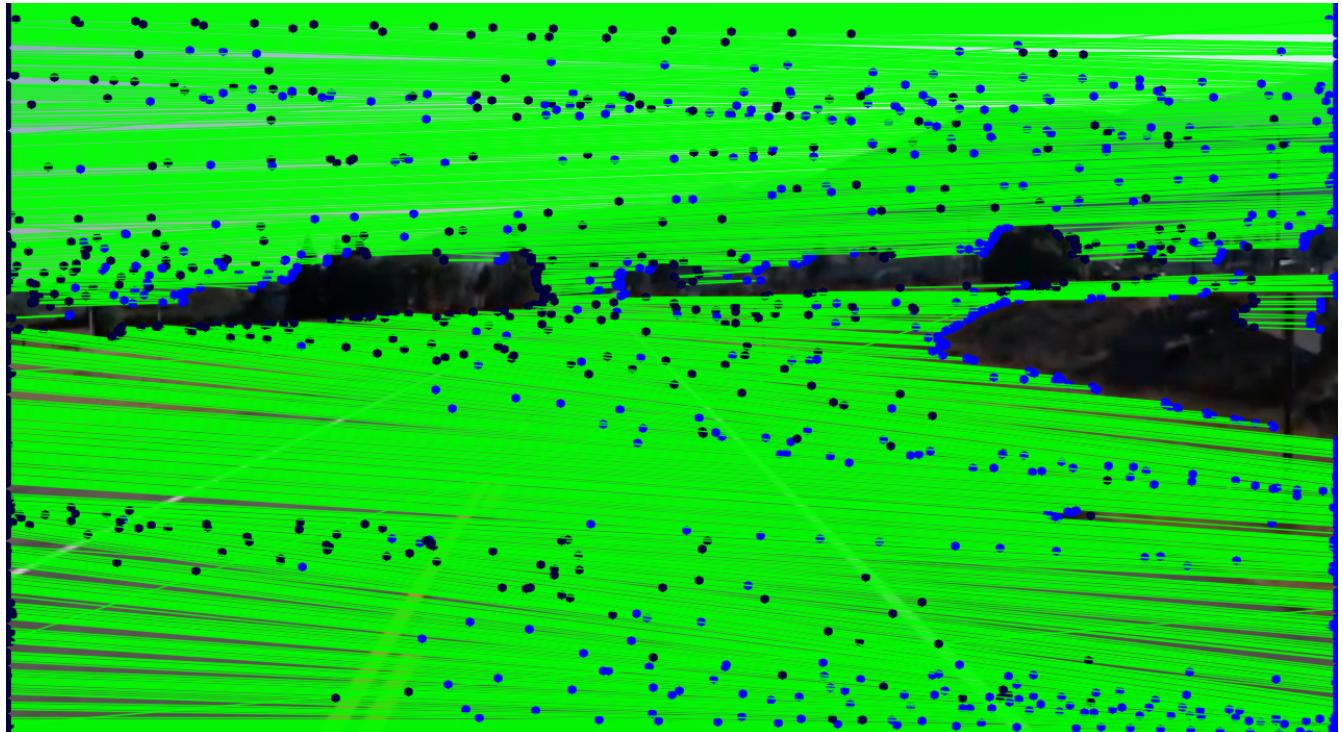


Рис. 5: Результат преобразования Хафа

MAX LENGTH = 1290

MIN LENGTH = 50

NUMBER OF LINES = 984

1.2.2. Обработанное алгоритмом Кэнни

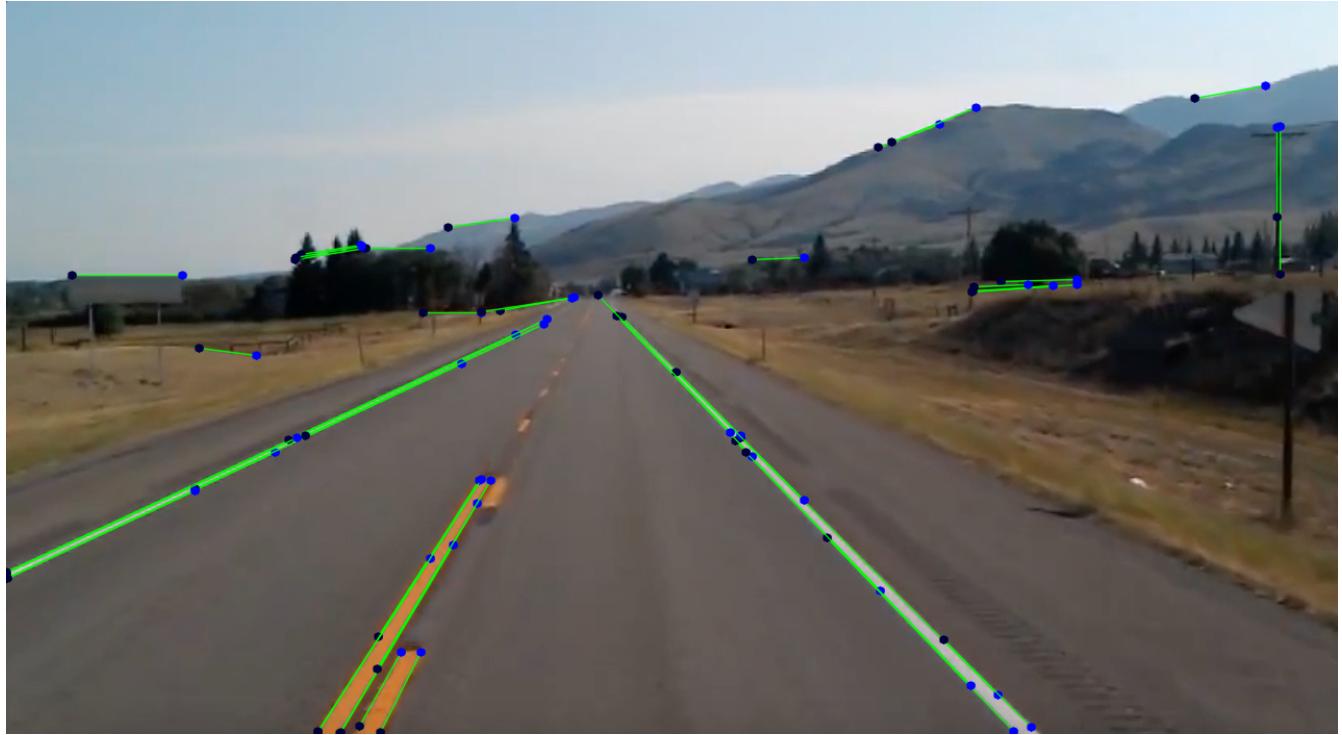


Рис. 6: Результат преобразования Хафа

MAX LENGTH = 5339

MIN LENGTH = 50

NUMBER OF LINES = 44

1.3. Изображение №3

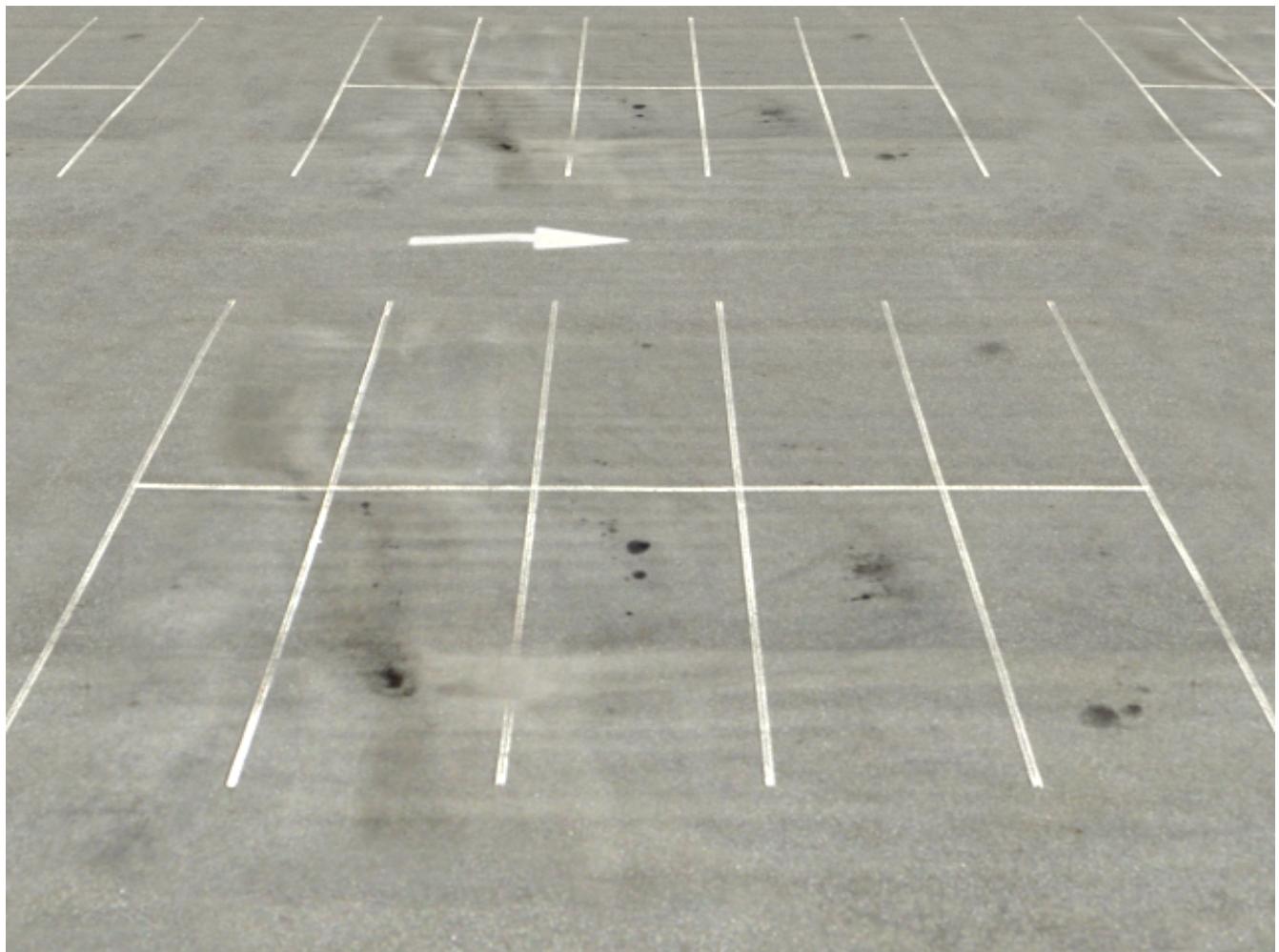


Рис. 7: Исходное изображение

1.3.1. Исходное изображение

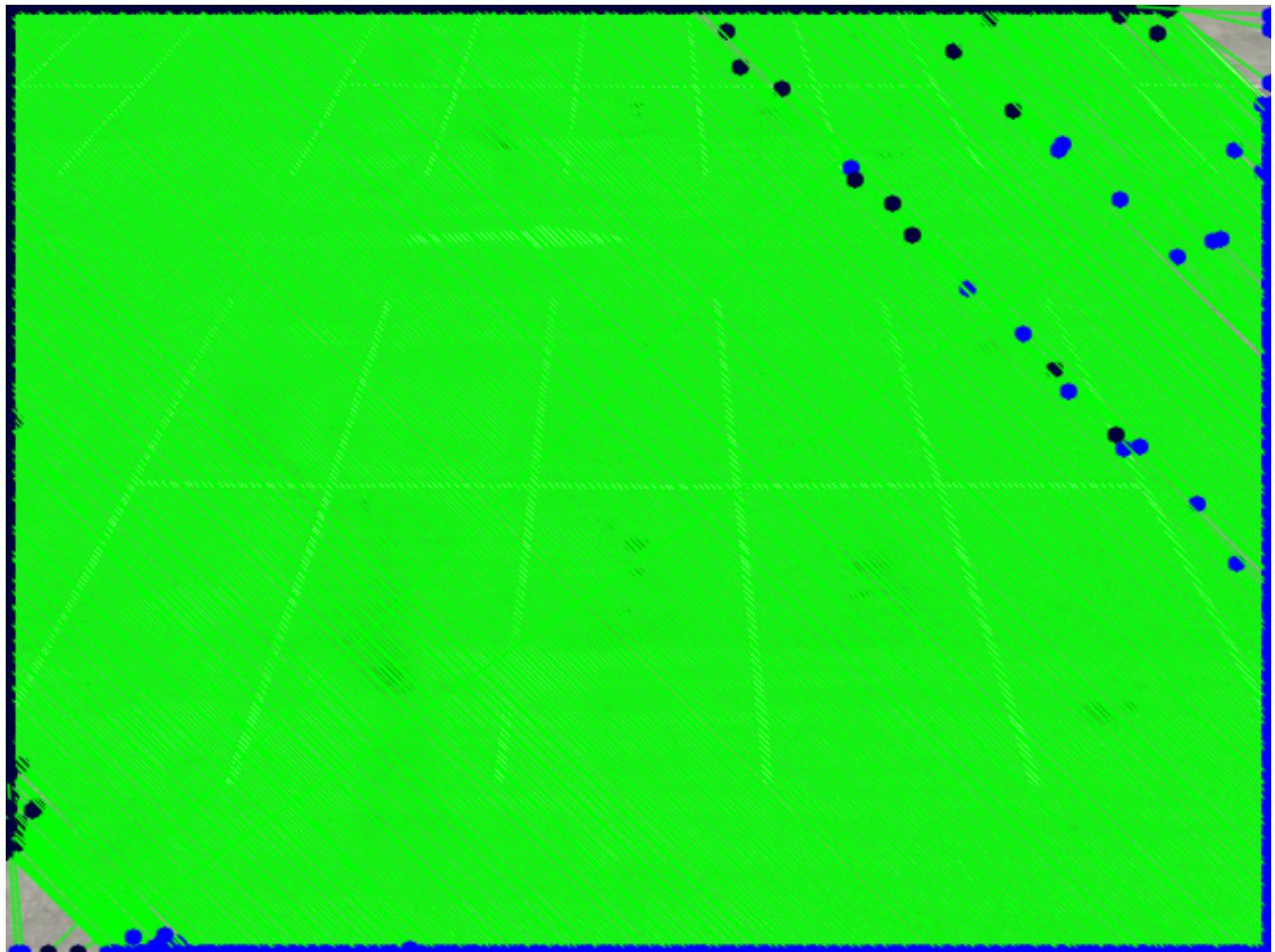


Рис. 8: Результат преобразования Хафа

MAX LENGTH = 735

MIN LENGTH = 53

NUMBER OF LINES = 738

1.3.2. Обработанное алгоритмом Кэнни

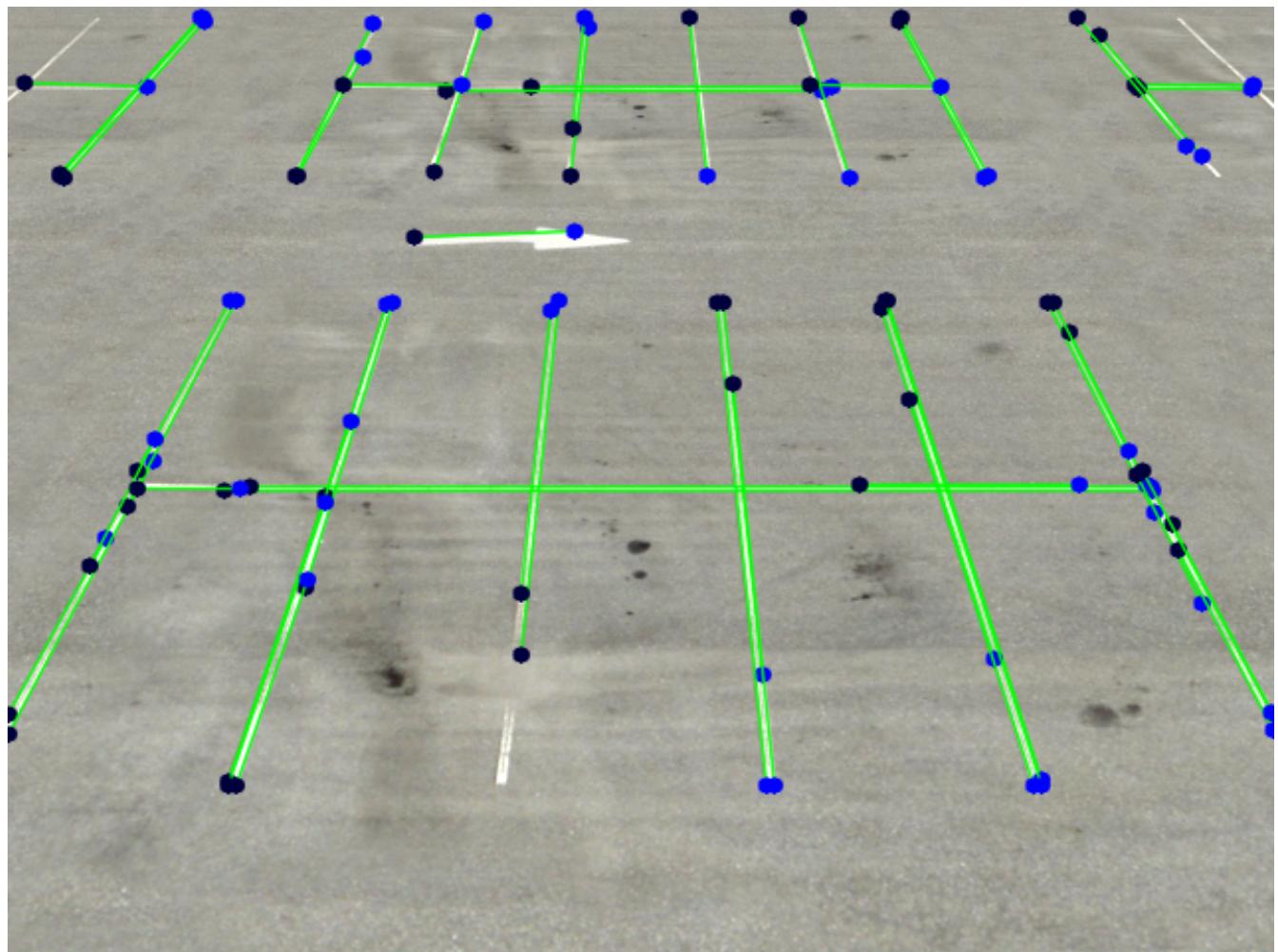


Рис. 9: Результат преобразования Хафа

MAX LENGTH = 469

MIN LENGTH = 52

NUMBER OF LINES = 52

2. Поиск окружностей

```
cv::Mat image, gray;
cv::cvtColor(image, gray, cv::COLOR_BGR2GRAY);
cv::medianBlur(gray, gray, 5);
```

```

std::vector<cv::Vec3f> circles;

cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, 1,
gray.rows/16, // change this value to detect circles with different
    distances to each other
100, 30, 50, 55 // change the last two parameters
// (min_radius & max_radius) to detect larger circles
);
for( size_t i = 0; i < circles.size(); i++ )
{
    cv::Vec3i c = circles[i];
    cv::Point center = cv::Point(c[0], c[1]);
    // circle center
    cv::circle(image, center, 1, cv::Scalar(255,0,0), 3, cv::LINE_AA);
    // circle outline
    int radius = c[2];
    cv::circle(image, center, radius, cv::Scalar(255,0,0), 3, cv::LINE_AA);
}
cv::imwrite(path + "/outputs/image6_ordinary_r50.png", image);
cv::imshow("detected circles", image);
cv::waitKey();

cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, 1,
gray.rows/16, // change this value to detect circles with different
    distances to each other
100, 30, 50, 65 // change the last two parameters
// (min_radius & max_radius) to detect larger circles
);
for( size_t i = 0; i < circles.size(); i++ )
{
    cv::Vec3i c = circles[i];
    cv::Point center = cv::Point(c[0], c[1]);
    // circle center
    cv::circle(image, center, 1, cv::Scalar(255,0,0), 3, cv::LINE_AA);
    // circle outline
    int radius = c[2];
    cv::circle(image, center, radius, cv::Scalar(255,0,0), 3, cv::LINE_AA);
}
cv::imwrite(path + "/outputs/image6_ordinary_r5065.png", image);
cv::imshow("detected circles", image);
cv::waitKey();

// CANNY

cv::cvtColor(image, gray, cv::COLOR_BGR2GRAY);
cv::Canny(gray, gray, 50, 200);

std::vector<cv::Vec3f> circles;

cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, 1,
gray.rows/16, // change this value to detect circles with different
    distances to each other

```

```

100, 30, 50, 55 // change the last two parameters
// (min_radius & max_radius) to detect larger circles
);
for( size_t i = 0; i < circles.size(); i++ )
{
    cv::Vec3i c = circles[i];
    cv::Point center = cv::Point(c[0], c[1]);
    // circle center
    cv::circle(image, center, 1, cv::Scalar(255,0,0), 3, cv::LINE_AA);
    // circle outline
    int radius = c[2];
    cv::circle(image, center, radius, cv::Scalar(255,0,0), 3, cv::LINE_AA);
}
cv::imwrite(path + "/outputs/image6_canny_r50.png", image);
cv::imshow("detected circles", image);
cv::waitKey();

cv::HoughCircles(gray, circles, cv::HOUGH_GRADIENT, 1,
gray.rows/16, // change this value to detect circles with different
    distances to each other
100, 30, 50, 65 // change the last two parameters
// (min_radius & max_radius) to detect larger circles
);
for( size_t i = 0; i < circles.size(); i++ )
{
    cv::Vec3i c = circles[i];
    cv::Point center = cv::Point(c[0], c[1]);
    // circle center
    cv::circle(image, center, 1, cv::Scalar(255,0,0), 3, cv::LINE_AA);
    // circle outline
    int radius = c[2];
    cv::circle(image, center, radius, cv::Scalar(255,0,0), 3, cv::LINE_AA);
}
cv::imwrite(path + "/outputs/image6_canny_r5065.png", image);
cv::imshow("detected circles", image);
cv::waitKey(); // image = cv::imread(path + "/source/6.png", 1);

```

Листинг 4: Общий код для поиска окружностей с помощью преобразования Хафа

2.1. Изображение №1



Рис. 10: Исходное изображение

2.1.1. Исходное изображение



Рис. 11: Результат преобразования Хафа для окружностей радиуса 50



Рис. 12: Результат преобразования Хафа для окружностей диапазона радиусов [50, 65]

2.1.2. Обработанное алгоритмом Кэнни



Рис. 13: Результат преобразования Хафа для окружностей радиуса 50



Рис. 14: Результат преобразования Хафа для окружностей диапазона радиусов [50, 65]

2.2. Изображение №2



Рис. 15: Исходное изображение

2.2.1. Исходное изображение



Рис. 16: Результат преобразования Хафа для окружностей радиуса 25



Рис. 17: Результат преобразования Хафа для окружностей диапазона радиусов [20, 30]

2.2.2. Обработанное алгоритмом Кэнни



Рис. 18: Результат преобразования Хафа для окружностей радиуса 35



Рис. 19: Результат преобразования Хафа для окружностей диапазона радиусов [35, 45]

2.3. Изображение №3



Рис. 20: Исходное изображение

2.3.1. Исходное изображение

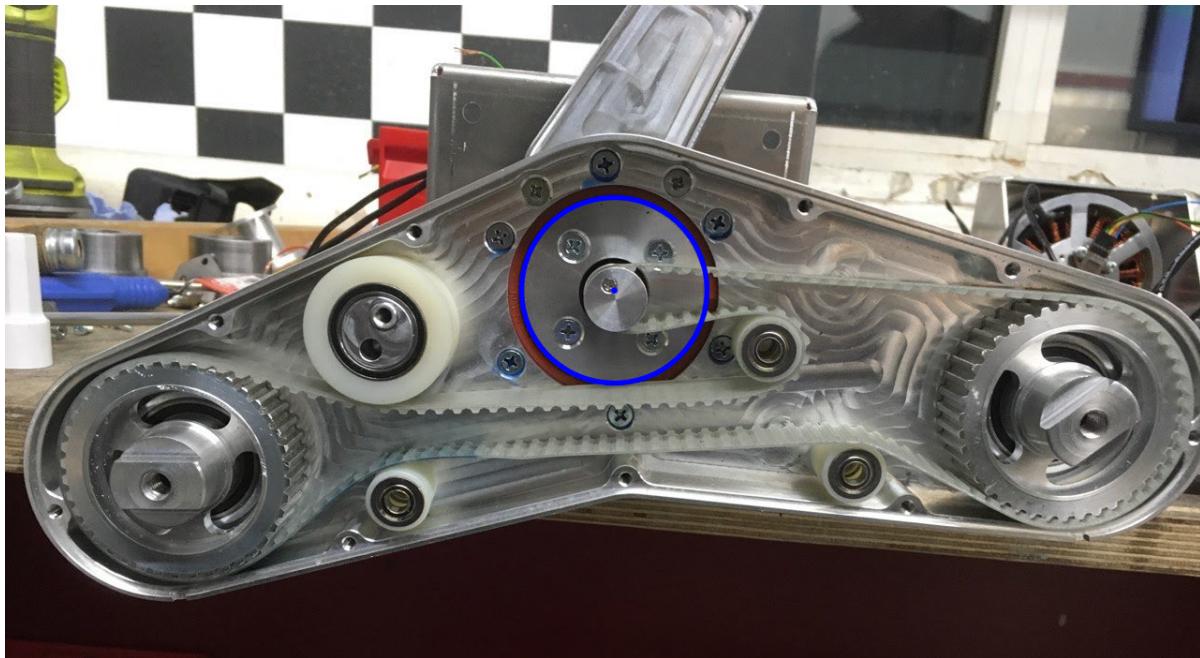


Рис. 21: Результат преобразования Хафа для окружностей радиуса 100

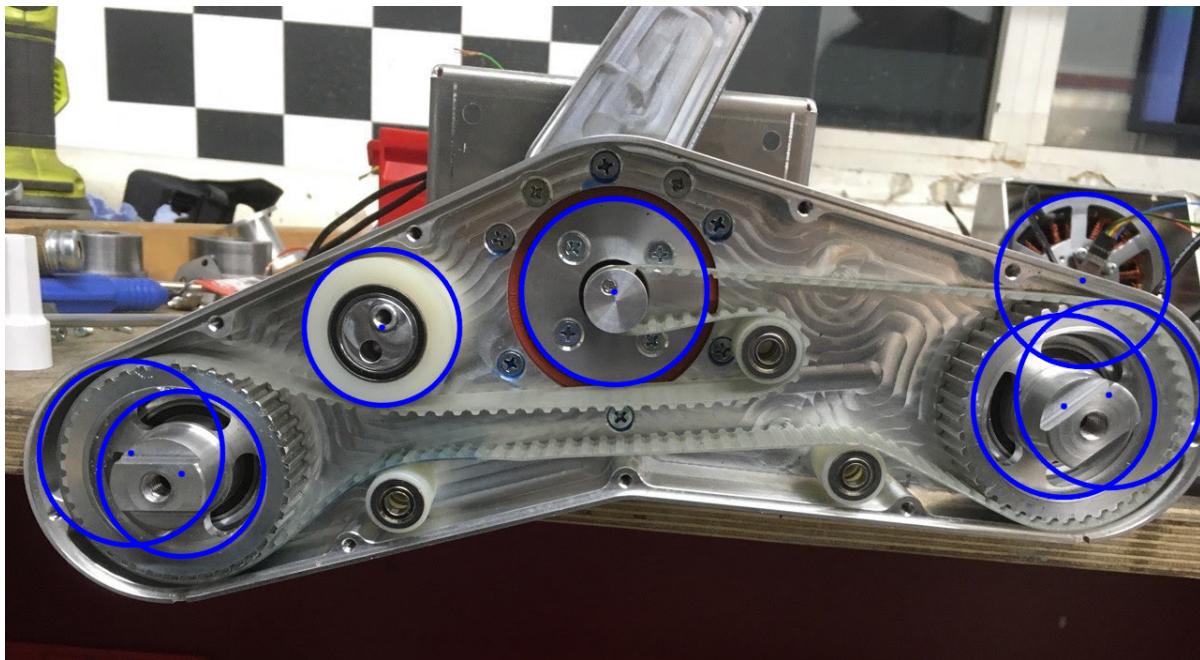


Рис. 22: Результат преобразования Хафа для окружностей диапазона радиусов [80, 100]

2.3.2. Обработанное алгоритмом Кэнни

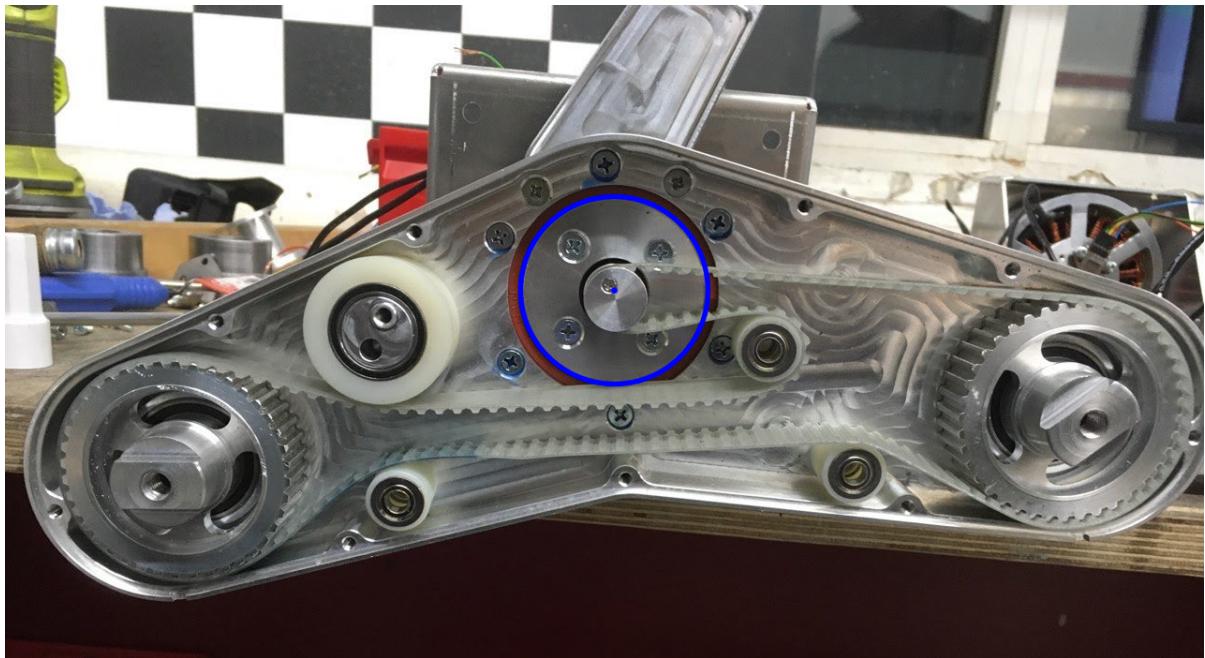


Рис. 23: Результат преобразования Хафа для окружностей радиуса 100

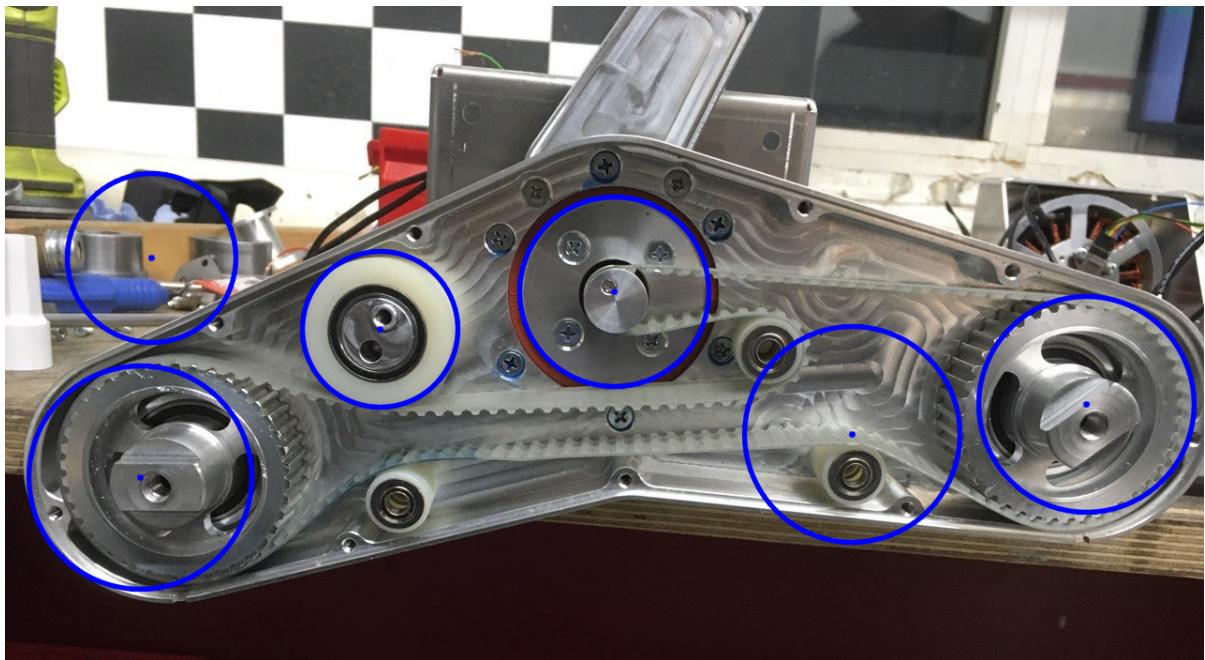


Рис. 24: Результат преобразования Хафа для окружностей диапазона радиусов [80, 100]

3. Ответы на вопросы

Q1. Какая идея лежит в основе преобразования Хафа?

A1. Преобразование Хафа - это метод анализа изображений, используемый для обнаружения форм геометрических объектов, особенно в компьютерном зрении и обработке изображений. Идея, лежащая в его основе, заключается в преобразовании координатных пространств, чтобы перейти от пространства параметров объектов (например, прямых или окружностей) к пространству параметров линий, сгруппировав их по общим свойствам.

В контексте обнаружения прямых на изображениях, преобразование Хафа переводит пиксели изображения в параметрическое пространство, где каждая точка представляет потенциальную прямую в исходном изображении. После этого используется аккумуляторный массив для определения областей, в которых сосредоточены прямые линии. Это позволяет выделять прямые линии, даже если они прерывисты или находятся под углом.

Идея заключается в том, чтобы перевести задачу поиска прямых на изображении в задачу поиска точек в параметрическом пространстве, что делает процесс более эффективным и менее чувствительным к шуму и другим артефактам на изображении.

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

A2. Да, преобразование Хафа можно применять не только для обнаружения прямых линий, но и для поиска произвольных контуров или форм, которые невозможно описать аналитически. Это достигается за счет того, что преобразование Хафа позволяет искать не только прямые линии, но и другие геометрические формы, такие как окружности или эллипсы, путем адаптации преобразования и выбора соответствующих параметров.

Например, для поиска произвольных контуров на изображении можно использовать преобразование Хафа с параметризацией, которая позволяет искать точки, соответствующие кривым или формам, таким как круги, эллипсы или даже неопределенные контуры. Затем можно применять дополнительные методы, такие как кластеризация или фильтрация, для идентификации и извлечения конкретных контуров или форм из набора найденных точек в преобразованном пространстве параметров.

Таким образом, преобразование Хафа является мощным инструментом для обнаружения различных геометрических форм на изображениях, включая произвольные контуры, которые могут быть сложно описать аналитически.

Q3. Что такое рекуррентное и обобщенное преобразования Хафа?

A3. Рекуррентное и обобщенное преобразования Хафа являются расширениями классического преобразования Хафа, которые были разработаны для более эффективного обнаружения объектов на изображениях, особенно в условиях повышенного шума или сложной геометрии.

Рекуррентное преобразование Хафа: Это модификация классического преобразования Хафа, которая включает процесс итеративного уточнения найденных параметров объектов путем повторного применения преобразования Хафа к изображению с последующим объединением результатов. Это позволяет повысить точность обнаружения объектов и снизить влияние шума.

Обобщенное преобразование Хафа: Это расширение, которое позволяет обнаруживать объекты с произвольными формами, а не только с фиксированными, такими как прямые или окружности. В обобщенном преобразовании Хафа используются более сложные модели объектов, и соответствующие преобразования применяются к изображению для поиска параметров этих моделей. Это делает метод более гибким и способным к обнаружению более широкого спектра объектов.

Q4. Какие бывают способы параметризации в преобразовании Хафа?

A4. Преобразование Хафа позволяет обнаруживать объекты на изображении, представляя их в параметрическом пространстве. Способы параметризации в преобразовании Хафа зависят от типа объекта, который вы хотите обнаружить. Некоторые из основных способов параметризации включают:

Для прямых линий:

Параметризация в полярных координатах:

Параметризация в декартовых координатах: Используются параметры, такие как полуоси

Для окружностей:

Параметризация в полярных координатах: r - радиус и координаты центра

Для эллипсов:

Параметризация в декартовых координатах: Используются параметры, такие как полуоси и координаты центра и угол поворота.

В случае обобщенного преобразования Хафа или адаптации для обнаружения произвольных форм, параметры могут варьироваться в зависимости от конкретной геометрической модели, используемой для представления объекта. Выбор определенного способа параметризации зависит от характеристик объекта и задачи обнаружения, а также может влиять на эффективность и точность процесса обнаружения.