**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе №4**

**по дисциплине «Интеллектуальные видеосистемы»**

Тема: СЕГМЕНТАЦИЯ ДВИЖУЩИХСЯ ОБЪЕКТОВ НА ОСНОВЕ ОЦЕНКИ ЭНЕРГИИ ДВИЖЕНИЯ

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8571 |  | Темный Д.А.  Мальцева О.Н. |
| Преподаватель |  | Поздеев А.А. |

Санкт-Петербург

2023

**Цель работы**

Целью лабораторной работы является знакомство с алгоритм сегментации объектов на основе энергии движения при отсутствии глобального движения. В соответствии с заданием необходимо разработать программу, реализующую алгоритм сегментации движущегося объекта с помощью оценки энергии движения с помощью библиотеки OpenCV.

Исходными данными являются два кадра видеопоследовательности со статичным фоном, на которых присутствует перемещающийся объект. В программе должны быть реализованы следующие процедуры, с выводом результата на экран:

* загрузка пары кадров,
* получение препарата межкадровой разности,
* получение контурного препарата с помощью одного из алгоритмов, указанных преподавателем: Собеля, разницы гауссиан, Кэнни, Робертса.
* объединение препаратов по логическому «И»
* обработка сформированного изображения с помощью морфологических фильтров (фильтрация и заливка),
* наложение строба на объект.

**Теоретические сведения**

Достаточно простой и эффективный метод сегментации объектов на основе признака движения основан на оценке «энергии движения». Под этим термином понимают межкадровую разность телевизионных сигналов. В случае двух цифровых изображений, представляющих соседние (или отстоящие друг от друга на некоторое время t кадры, оценка движения может быть найдена как поэлементная абсолютная разность двух кадров:



где, – значение яркости пикселя с координатами в кадре , – значение яркости пикселя с координатами в кадре .

Препарат межкадровой разности двух кадров представлен на рисунке 1.



Рис. 1 – Пара кадров (верхний ряд) и их разность (внизу)

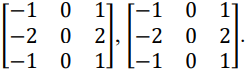
При анализе видеоряда и сегментации объектов глобальное движение следует предварительно компенсировать.

При отсутствии глобального движения алгоритм сегментации объектов на основе энергии движения состоит из следующих шагов:

1. получение препарата межкадровой разности;
2. формирование контурного препарата для текущего кадра ;
3. бинарное квантование контурного препарата и межкадровой разности;
4. объединение двух полученных препаратов с помощью операции логического «И» – получение изображения, содержащего оценку энергии движения, относящейся только к текущему кадру;
5. морфологическая фильтрация полученного изображения;
6. формирование строба объектов.

Второй и третий шаги необходимы потому, что изображение межкадровой разности отражает энергию движения, как предыдущего, так и текущего кадров. В результате центр области движения не совпадает с центром движения реального объекта. Определение контуров объектов в текущем кадре и объединение препаратов по «И» устраняет этот фактор.

Для выделения контуров в текущем кадре видеопоследовательности могут применяться различные алгоритмы (оператор Собеля, разницы гауссиан и так далее). Традиционным в данном случае является использование оператора Собеля. Алгоритм использует ядра (маски) размером 3 на 3 элемента для выделения горизонтальных и вертикальных контуров соответственно:



Для изображения (кадр ) осуществляют свертку с каждым из ядер для вычисления приближённых значений производных по горизонтали и по вертикали . Приближённое значение величины градиента можно поэлементно вычислить путём использования полученных приближенных значений производных по направлениям:



Пример контурного препарата, полученный для кадра t на рисунке 1 приведен на рисунке 2.

Контурное изображение затем бинарно квантуют так же, как и изображение межкадровой разности , что дает возможность получить объединенное изображение , выполнив поэлементно логическую операцию «И»:



В результате получают изображение (рис. 3), в котором выделенные движущиеся объекты занимают положение, соответствующее текущему кадру.

Полученный таким образом препарат может содержать множество нежелательных фрагментов, вызванных, например, шумами. Для снижения помех целесообразно провести предварительную фильтрацию исходного изображения (например, медианным фильтром), а затем изображения межкадровой разности.

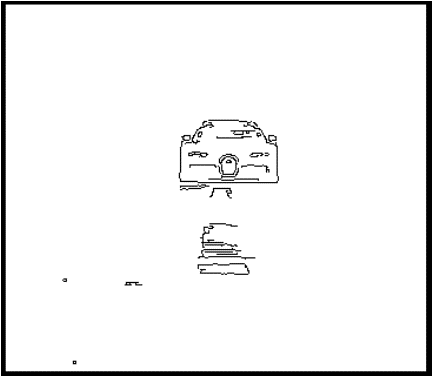


Рис. 2 – Контурный препарат

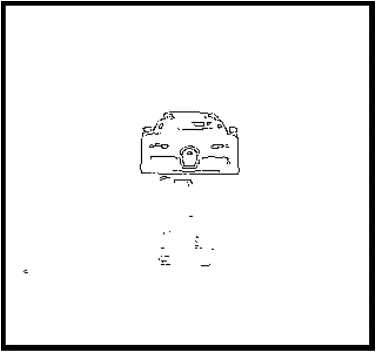


Рис. 3 – Выделенный слой движения

Изображения движущихся объектов на результирующем изображении могут иметь разрывы, лакуны и прочие дефекты, которые затруднят точную сегментацию. Для удаления этих дефектов используют так называемую морфологическую фильтрацию, основанную на логических операциях. Морфологическую фильтрацию применяют к цифровым изображениям как бинарно квантованным, так и полутоновым.

Логические операции совершают между маской и областью цифрового изображения, выделенной этой маской. Результат операции устанавливают в новую битовую матрицу на место, где находится фокус маски. Используют маски различной формы и размерности.

Морфологическая операция «открытие» (последовательное применение операций эрозии и наращивания) с маской малого размера (относительно предполагаемого размера объекта) позволяет подавить импульсный шум на изображении. Применение после этого операции «закрытие» (последовательное применение операций наращивания и эрозии) с увеличенным размером маски способствует удалению разрывов и заполнению лакун в изображениях объектов.

На заключительном шаге алгоритма организуют наложение строба на объект (рис. 4). Для формирования строба в методе сегментации объектов на основе «энергии движения» используют алгоритм проекций. Для результирующего изображения , прошедшего морфологическую фильтрацию, в каждом столбце и в каждой строке подсчитывают количество ненулевых («белых» в рассматриваемом примере) пикселей. По полученным данным строят соответствующие гистограммы, сопоставляющие номеру строки (столбца) – число ненулевых пикселей.

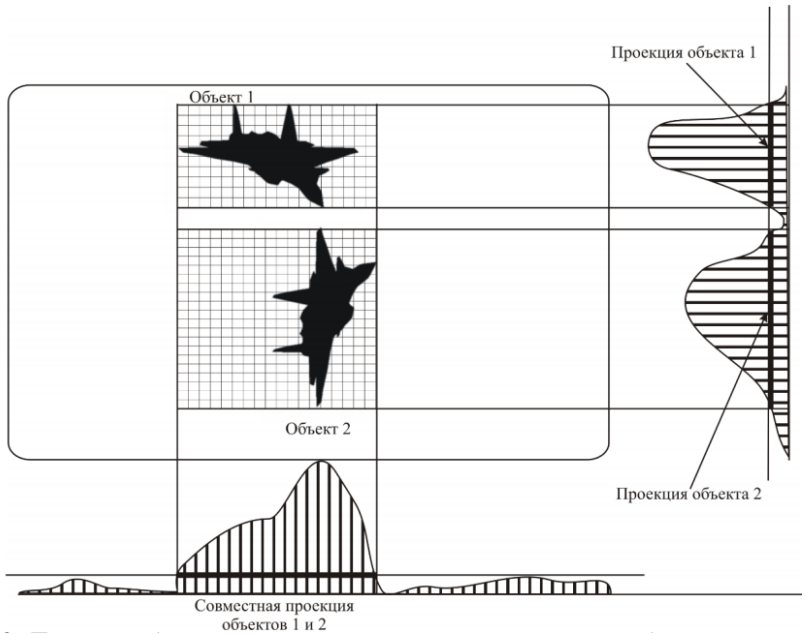


Рис. 4 – Стробирование объектов

Затем выделяют проекции – совокупность строк (столбцов), у которых число ненулевых пикселей превышает некий порог и которые образуют на гистограмме непрерывный отрезок. Проекции, длина которых меньше заданной величины (определяемой априорным знанием о предполагаемых размерах объектов), отбрасывают, чтобы исключить изолированные помехи. На базе оставшихся проекций формируют стробы.

Если в кадре движутся несколько объектов, то для установления соответствия между проекциями по горизонтали и вертикали используют правило: число пикселей в горизонтальной и вертикальной проекции одного объекта должны быть равны.

**Сегментация объектов**

Исходные кадры представлены на рис. 5 и 6.



Рис. 5 – Первый кадр



Рис. 6 – Второй кадр

На рис. 7 изображен результат вычитания исходных кадров друг из друга (получение препарата межкадровой разности).

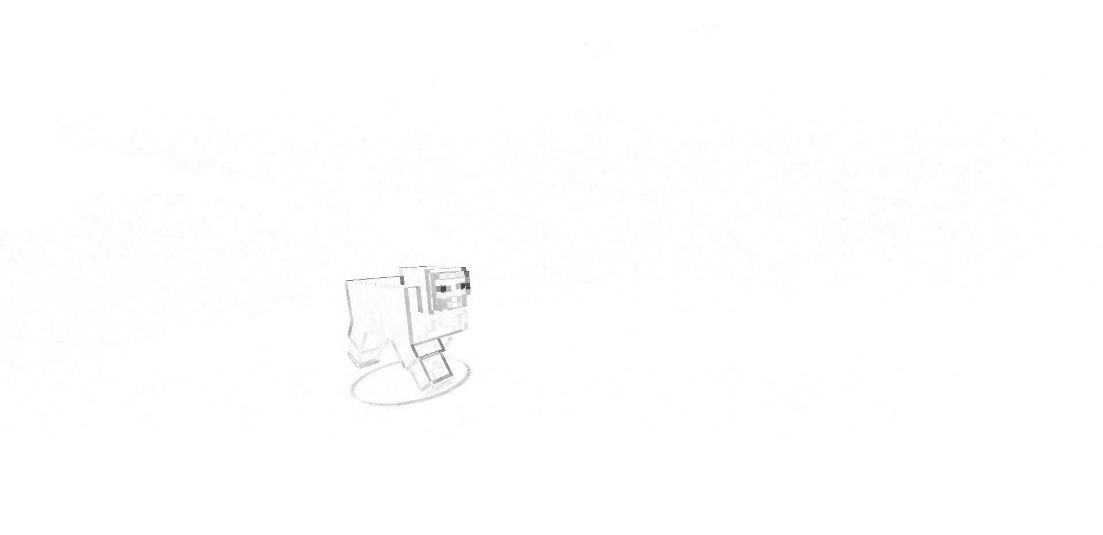


Рис. 7 – Препарат межкадровой разности (MAD)

Для получения контурного препарата использовался обобщенный оператор Собела (рис. 8).

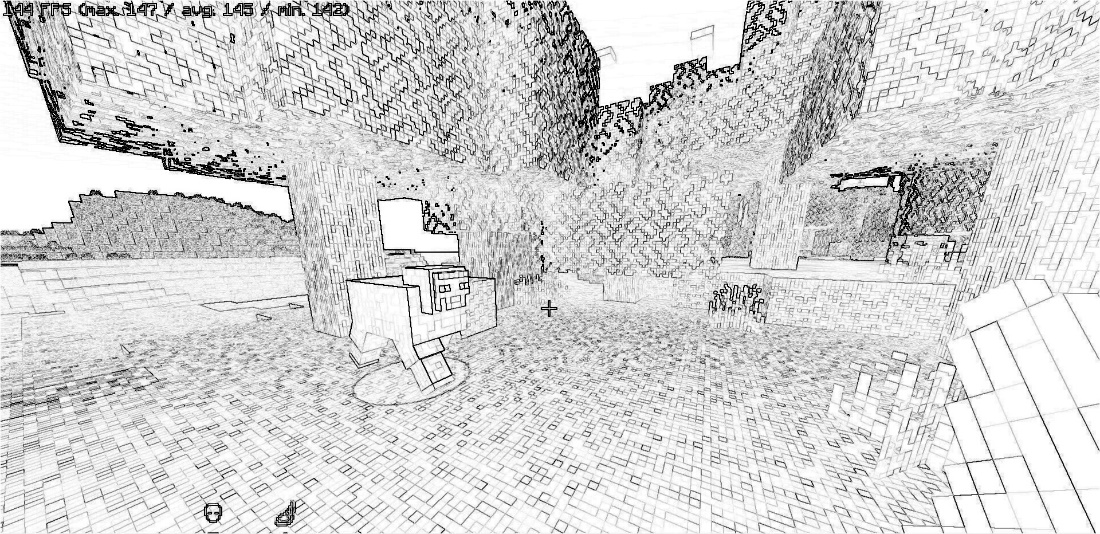


Рис. 8 – Полученный контурный препарат

Далее проведена бинаризация и попиксельная конъюнкция контурного препарата с препаратом межкадровой разности (рис. 9).

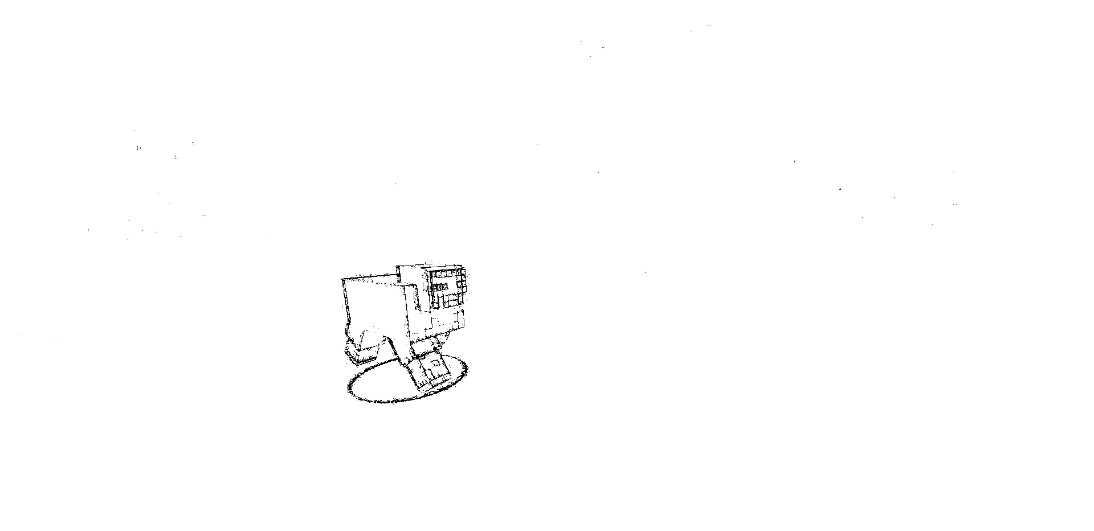


Рис. 9 – Выделенный контур движения (объединенное изображение)

К результирующему контуру движения были последовательно применены операция открытия со структурным элементом форму ромба и операция закрытия с квадратным структурным элементом размера 105x105 (рис. 10).

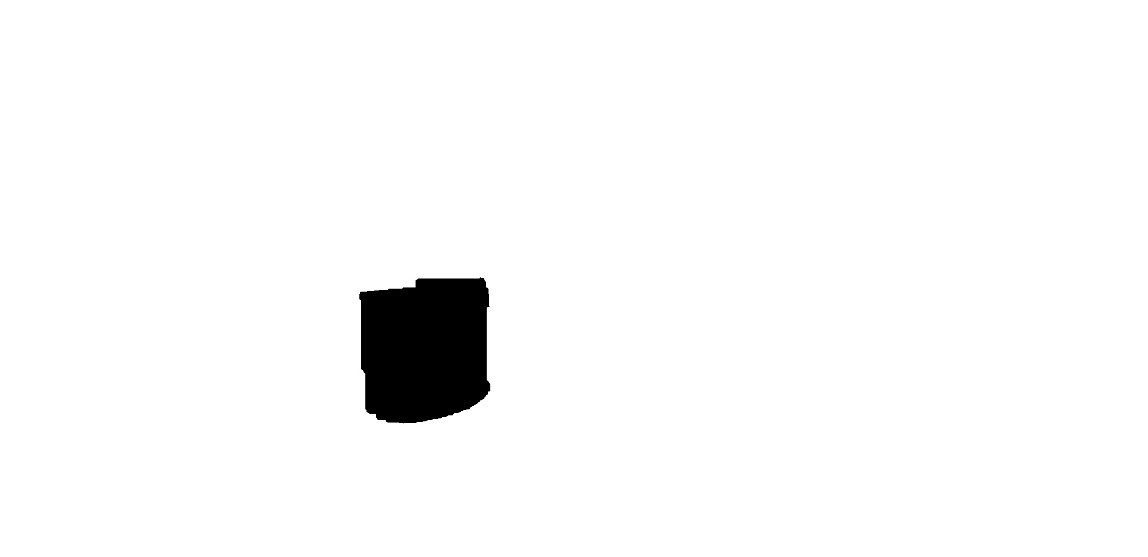


Рис. 10 – Результат морфологической фильтрации

На рис. 11 изображен выделенный контур объекта и его проекций с отображением порога в 50 пикселей для исключение изолированных помех.

С помощью непрерывных отрезков полученных гистограмм были определены стробы и отображены на втором исходном изображении (рис. 12). По итогу единственный движущийся в кадре объект был стробирован корректно.

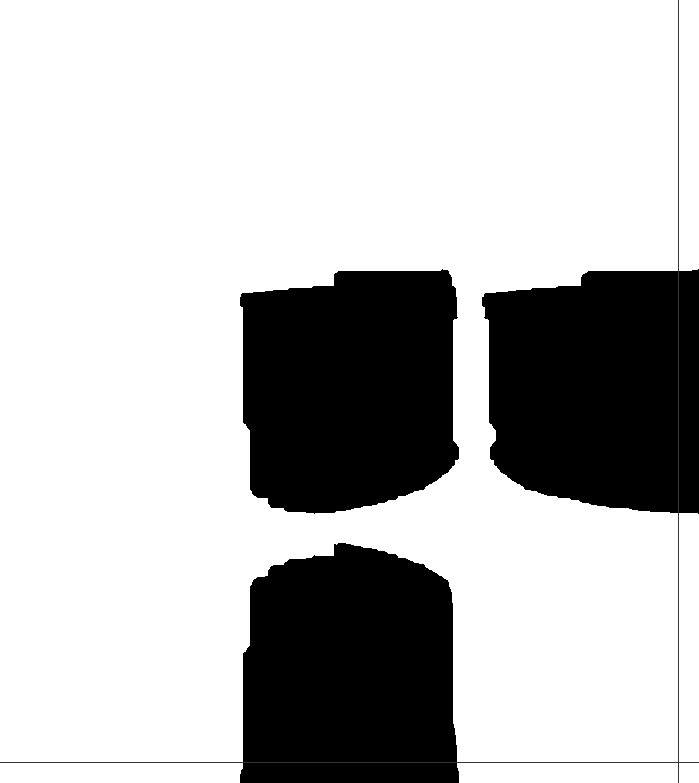


Рис. 11 – Проекции объекта (стробирование)



Рис. 12 – Выделенный объект на одном из исходных изображений (втором)

**Вывод**

В ходе работы был реализован алгоритм сегментации движущегося объекта в OpenCV: получен контур движения, по которому определены горизонтальная и вертикальная проекция (гистограммы), на основе которых были сформированы и наложены стробы на обнаруженный объект.

В результате объект, меняющий свое положение между кадрами, был корректно обнаружен и стробирован.

**Листинг Кода**

**main.cpp:**

#include <iostream>

#include <string>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

#include <lab2/include/img\_to\_binary.h>

#include <gradient\_and\_strobing.h>

int main(int argc, char\* argv[]) {

    cv::Mat src\_left;

    cv::Mat src\_right;

    cv::Mat src\_left\_colour;

    cv::Mat src\_right\_colour;

    if ( argc <=2 ) {

        src\_left\_colour = cv::imread("p3\_1.jpg");

        src\_right\_colour = cv::imread("p3\_2.jpg");

    } else if ( argc == 3 ) {

        src\_left\_colour = cv::imread(argv[1]);

        src\_right\_colour = cv::imread(argv[2]);

    }

    cv::cvtColor(src\_left\_colour, src\_left, cv::COLOR\_BGR2GRAY);

    cv::cvtColor(src\_right\_colour, src\_right, cv::COLOR\_BGR2GRAY);

    cv::Mat out\_MAD;

    cv::Mat out\_MAD\_binary;

    cv::Mat out\_gradientR;

    cv::Mat out\_gradientR\_binary;

    cv::Mat out\_concat;

    cv::Mat out\_concat\_binary;

    cv::Mat out\_filtered;

    cv::Mat out\_cropped;

    cv::Mat out\_histogram;

    cv::Mat out\_histogram\_crop;

    cv::Mat out\_masked;

    cv::absdiff(src\_right, src\_left, out\_MAD);

    get\_gradient(src\_right, out\_gradientR);

    get\_binary(out\_MAD, out\_MAD\_binary, 10);

    get\_binary(out\_gradientR, out\_gradientR\_binary, 20);

    cv::bitwise\_and(out\_MAD\_binary, out\_gradientR\_binary, out\_concat);

    int closing\_mask\_size = 105;

    filter(out\_concat, out\_filtered, closing\_mask\_size);

    crop\_border(out\_filtered, out\_cropped);

    int strob\_threshould = 20;

    strob(out\_cropped, out\_histogram\_crop, strob\_threshould);

    std::vector<std::vector<int>> weights = strob(out\_filtered, out\_histogram, strob\_threshould);

    mask\_strob\_object(src\_right\_colour, out\_masked, weights);

    std::string output\_folder = "output";

    std::string create\_folder = "mkdir " + output\_folder;

    system(create\_folder.c\_str());

    output\_folder += "/";

    cv::imwrite(output\_folder + "MAD.jpg", 255 - out\_MAD);

    cv::imwrite(output\_folder + "MAD\_binary.jpg", 255 - out\_MAD\_binary);

    cv::imwrite(output\_folder + "gradientR.jpg", 255 - out\_gradientR);

    cv::imwrite(output\_folder + "gradientR\_binary.jpg", 255 - out\_gradientR\_binary);

    cv::imwrite(output\_folder + "concatinated.jpg", 255 - out\_concat);

    cv::imwrite(output\_folder + "filtered.jpg", 255 - out\_filtered);

    cv::imwrite(output\_folder + "cropped.jpg", 255 - out\_cropped);

    cv::imwrite(output\_folder + "histogram.jpg", 255 - out\_histogram);

    cv::imwrite(output\_folder + "histogram\_crop.jpg", 255 - out\_histogram\_crop);

    cv::imwrite(output\_folder + "masked.jpg", out\_masked);

    int confirm = 0;

    std::cout << "\nShow all? (0/1) ";

    std::cin >> confirm;

    if (confirm) {

        show\_img("inputL", src\_left);

        show\_img("inputR", src\_right);

        show\_img("MAD", out\_MAD);

        show\_img("gradientR", out\_gradientR);

        show\_img("concatinated", out\_concat);

        show\_img("filtered", out\_filtered);

        show\_img("cropped", out\_cropped);

        show\_img("histogram", out\_histogram);

        show\_img("histogram\_crop", out\_histogram\_crop);

        show\_img("masked", out\_masked);

        cv::waitKey(0);

    }

    std::cout << "\n";

}

**gradient\_and\_strobing.h (функции обработки):**

#include <opencv2/opencv.hpp>

#include <opencv2/core.hpp>

#include <vector>

#include <random>

#include <cstdlib>

#include <lab1\_5/include/sobel\_outline.h>

#include <lab2/include/morphological\_operations.h>

void crop\_border(const cv::Mat &input\_img, cv::Mat &output\_img)

{

    cv::Mat bounds;

    cv::findNonZero(input\_img, bounds);

    output\_img = input\_img(cv::boundingRect(bounds));

}

void mask\_strob\_object(const cv::Mat &input\_img, cv::Mat &output\_img, std::vector<std::vector<int>> &hist) {

    output\_img = input\_img.clone();

    std::vector<int> color;

    for (int i = 0; i < 3; ++i)

    {

        color.push\_back(55 + std::rand()%200);

    }

    bool top\_left = false;

    std::pair<int, int> p1 = {0, 0};

    std::pair<int, int> p2 = {0, 0};

    int diagonal\_pos = 0;

    for (int i = 1; i < input\_img.cols; ++i)

    {

        for (int j = 1; j < input\_img.rows; ++j)

        {

            diagonal\_pos = input\_img.cols + j - i;

            if (hist[0][j] && hist[1][i])

            {

                if (!top\_left)

                {

                    p1 = {i, j};

                    top\_left = true;

                }

                if (input\_img.type() == CV\_8UC3)

                {

                    cv::Vec3b pixel\_val = input\_img.at<cv::Vec3b>(j, i);

                    for (int k = 0; k < pixel\_val.channels; ++k)

                        output\_img.at<cv::Vec3b>(j, i)[k] = pixel\_val[k] + color[k] < 255 ? pixel\_val[k] + color[k] : 255;

                }

                else

                {

                    uchar pixel\_val = input\_img.at<uchar>(j, i);

                    output\_img.at<uchar>(j, i) = pixel\_val + color[0] < 255 ? pixel\_val + color[0] : 255;

                }

                p2 = {i, j};

            }

            if (((hist[0][j-1] && !hist[0][j]) && (hist[1][i-1] && !hist[1][i])))

            {

                cv::rectangle(output\_img, cv::Point(p1.first, p1.second), cv::Point(p2.first, p2.second), cv::Scalar(color[0], color[1], color[2]), 2);

                for (int i = 0; i < 3; ++i)

                {

                    color[i] = 55 + std::rand()%200;

                }

                top\_left = false;

            }

        }

    }

}

void get\_gradient(const cv::Mat &input\_img, cv::Mat &output\_img)

{

    sobel\_general(input\_img, output\_img);

}

std::vector<std::vector<int>> strob(const cv::Mat &input\_img, cv::Mat &strob\_img, int threshold = 10) {

    std::cout << std::endl << "Strobing" << std::endl;

    std::vector<int> projectionH(input\_img.rows, 0);

    std::vector<int> projectionV(input\_img.cols, 0);

    int diagonal\_axis = input\_img.cols + input\_img.rows + 1;

    std::vector<int> projectionD(diagonal\_axis, 0);

    cv::copyMakeBorder(input\_img, strob\_img, input\_img.rows\*1.1, input\_img.rows\*1.1, input\_img.cols\*1.1, input\_img.cols\*1.1, cv::BORDER\_CONSTANT, cv::Scalar(0));

    int hist\_row = strob\_img.rows - 1;

    int hist\_col = strob\_img.cols - 1;

    for (int i = 0; i < input\_img.cols; ++i)

    {

        hist\_row = strob\_img.rows - 1;

        for (int j = 0; j < input\_img.rows; ++j)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            if (pix\_value)

            {

                strob\_img.at<uchar>(hist\_row--, i + strob\_img.cols/2 - input\_img.cols/2) = 255;

                hist\_col = strob\_img.cols - 1;

                while(strob\_img.at<uchar>(j + strob\_img.rows/2 - input\_img.rows/2, hist\_col))

                {

                    hist\_col--;

                }

                strob\_img.at<uchar>(j + strob\_img.rows/2 - input\_img.rows/2, hist\_col--) = 255;

                ++projectionH[j];

                ++projectionV[i];

                ++projectionD[input\_img.cols + j - i];

            }

        }

        if (( ( i + 1 ) % ( (input\_img.cols) / 10 ) ) == 0)

            progressbar(input\_img.cols, i);

    }

    for (int i = 0; i < strob\_img.cols; ++i)

    {

        for (int j = 0; j < strob\_img.rows; ++j)

        {

            strob\_img.at<uchar>(strob\_img.rows - 1 - threshold, i) = 200;

            strob\_img.at<uchar>(j, strob\_img.cols - 1 - threshold) = 200;

        }

    }

    for (int i = 0; i < projectionH.size(); ++i)

    {

        if (projectionH[i] < threshold)

            projectionH[i] = 0;

    }

    for (int i = 0; i < projectionV.size(); ++i)

    {

        if (projectionV[i] < threshold)

            projectionV[i] = 0;

    }

    for (int i = 1; i < projectionD.size() - 1; ++i)

    {

        if (projectionD[i] < threshold)

            projectionD[i] = 0;

    }

    std::cout << std::endl;

    return {projectionH, projectionV, projectionD};

}

void filter(const cv::Mat &input\_img, cv::Mat &out\_image, int closing\_mask\_size = 105)

{

    cv::Mat input\_border\_img;

    cv::Mat out\_open;

    cv::copyMakeBorder(input\_img, input\_border\_img, closing\_mask\_size, closing\_mask\_size, closing\_mask\_size, closing\_mask\_size, cv::BORDER\_CONSTANT, cv::Scalar(0));

    opening(input\_border\_img, out\_open, 3, CROSS);

    closing(out\_open, out\_image, closing\_mask\_size);

    out\_image = out\_image(cv::Range(closing\_mask\_size, out\_image.rows - closing\_mask\_size), cv::Range(closing\_mask\_size, out\_image.cols - closing\_mask\_size));

}