

Работа 6. Выделение границ проективно искаженных прямоугольников страниц

автор: Лоев В.А.

url: https://mysvn.ru/LOEV V A/loev v a/prj.labs/lab_6/

Задание

1. Реализовать выделение "границ и уголков" на фотографиях документа.
2. Реализовать детектирование наиболее правдоподобного четырехугольника изображения страницы документа на фото.
3. Используя эталонную геометрическую разметку из лабораторной 5 реализовать численную оценку качества выделения отдельных примитивов и финального результата для отдельных изображений и для набора.
4. Реализовать представление результатов выполнения лабораторной (иллюстрации, таблицы, графики и т.д.) и вставить в отчет.

Результаты

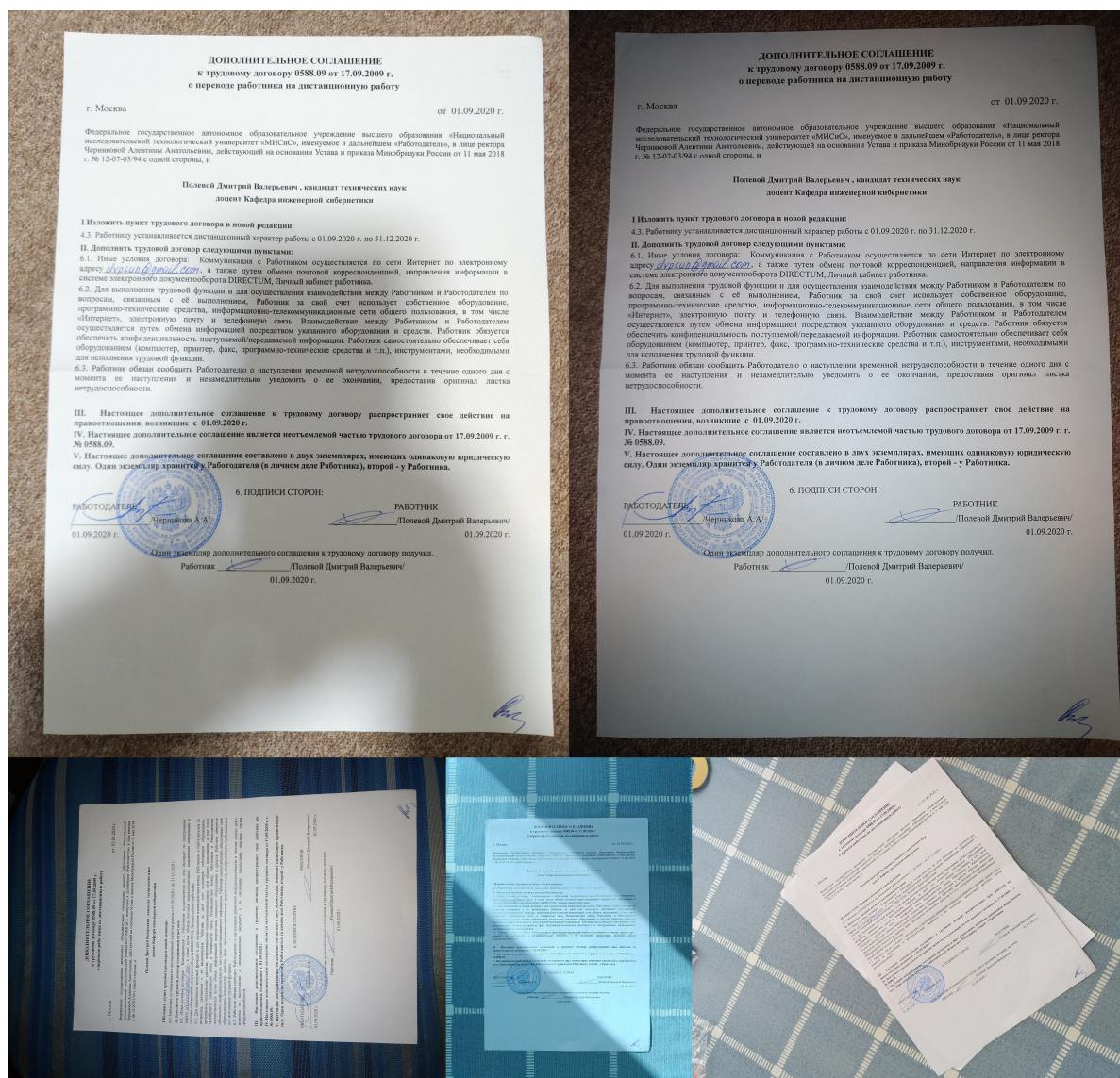


Рис. 1. Исходные изображения I_1, I_2, I_3, I_4, I_5

Процесс детектирования страницы документа

1. Медианный фильтр

Параметры:

- Окно = (55, 55)

2. Гауссов фильтр

Параметры:

- Окно = (71, 71)

3. Адаптивная бинаризация(с Гауссовской взвешенной суммой)

Параметры:

- Окно = (71, 71)

- С = 3

4. Медианный фильтр

Параметры:

- Окно = (25, 25)

5. Выделение компонент связности

Параметры:

- Общая площадь компоненты > 600 пикселей

6. Определение границ методом Кэнни

Параметры:

- Верхний порог = 200

- Нижний порог = 100

7. Определение линий методом Хафа

Параметры:

- Порог детекции = 200

- Минимальная длина линии = 200 пикселей

- Максимальное расстояние между точками одной линии = 300 пикселей

8. Фильтрация точек линий

Отбираются 4 точки наиболее близкие к углам изображения и по ним строится область документа

Визуализация результатов

На изображениях ниже красным четырехугольником показана обнаруженная область документа, зеленым -- эталонная область с разметки

**ДОПОЛНИТЕЛЬНОЕ СОГЛАШЕНИЕ
к трудовому договору 0588.09 от 17.09.2009 г.
о переводе работника на дистанционную работу**

г. Москва

от 01.09.2020 г.

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский технологический университет «МИСиС», именуемое в дальнейшем «Работодатель», в лице ректора Черниковой Алентины Анатольевны, действующей на основании Устава и приказа Минобрнауки России от 11 мая 2018 г. № 12-07-03/94 с одной стороны, и

Полевой Дмитрий Валерьевич , кандидат технических наук
доцент Кафедра инженерной кибернетики

I Изложить пункт трудового договора в новой редакции:

4.3. Работнику устанавливается дистанционный характер работы с 01.09.2020 г. по 31.12.2020 г.

II. Дополнить трудовой договор следующими пунктами:

6.1. Иные условия договора: Коммуникация с Работником осуществляется по сети Интернет по электронному адресу dipis@yandex.ru, а также путем обмена почтовой корреспонденцией, направления информации в системе электронного документооборота DIRECTUM. Лицом, избирающим представителя,

системы электронного документооборота DIRECTUM, Личный кабинет работника.

6.2. Для выполнения трудовой функции и для осуществления взаимодействия между Работником и Работодателем по вопросам, связанным с её выполнением, Работник за свой счет использует собственное оборудование, программно-технические средства, информационно-телекоммуникационные сети общего пользования, в том числе «Интернет», электронную почту и телефонную связь. Взаимодействие между Работником и Работодателем осуществляется путем обмена информацией посредством указанного оборудования и средств. Работник обязуется обеспечить конфиденциальность поступаемой/передаваемой информации. Работник самостоятельно обеспечивает себя оборудованием (компьютер, принтер, факс, программно-технические средства и т.п.), инструментами, необходимыми для исполнения трудовой функции.

5.3. Работник обязан сообщить Работодателю о наступлении временной нетрудоспособности в течение одного дня с момента ее наступления и незамедлительно уведомить о ее окончании, предоставив оригинал листка нетрудоспособности.

III. Настоящее дополнительное соглашение к трудовому договору распространяет свое действие на правоотношения, возникшие с 01.09.2020 г.

IV. Настоящее дополнительное соглашение является неотъемлемой частью трудового договора от 17.09.2009 г. г. № 0588.09.

V. Настоящее дополнительное соглашение составлено в двух экземплярах, имеющих одинаковую юридическую силу. Один экземпляр хранится у Работодателя (в личном деле Работника), второй - у Работника.

6. ПОДПИСИ СТОРОН:

РАБОТОДАТЕЛЬ
Черникова А.А.
01.09.2020 г.

РАБОТНИК
Полевой Дмитрий Валерьевич
01.09.2020 г.

Работник Полевоий Дмитрий Валерьевич/
01.09.2020 г.

Рис. 2. Найденная и эталонная область для I_1

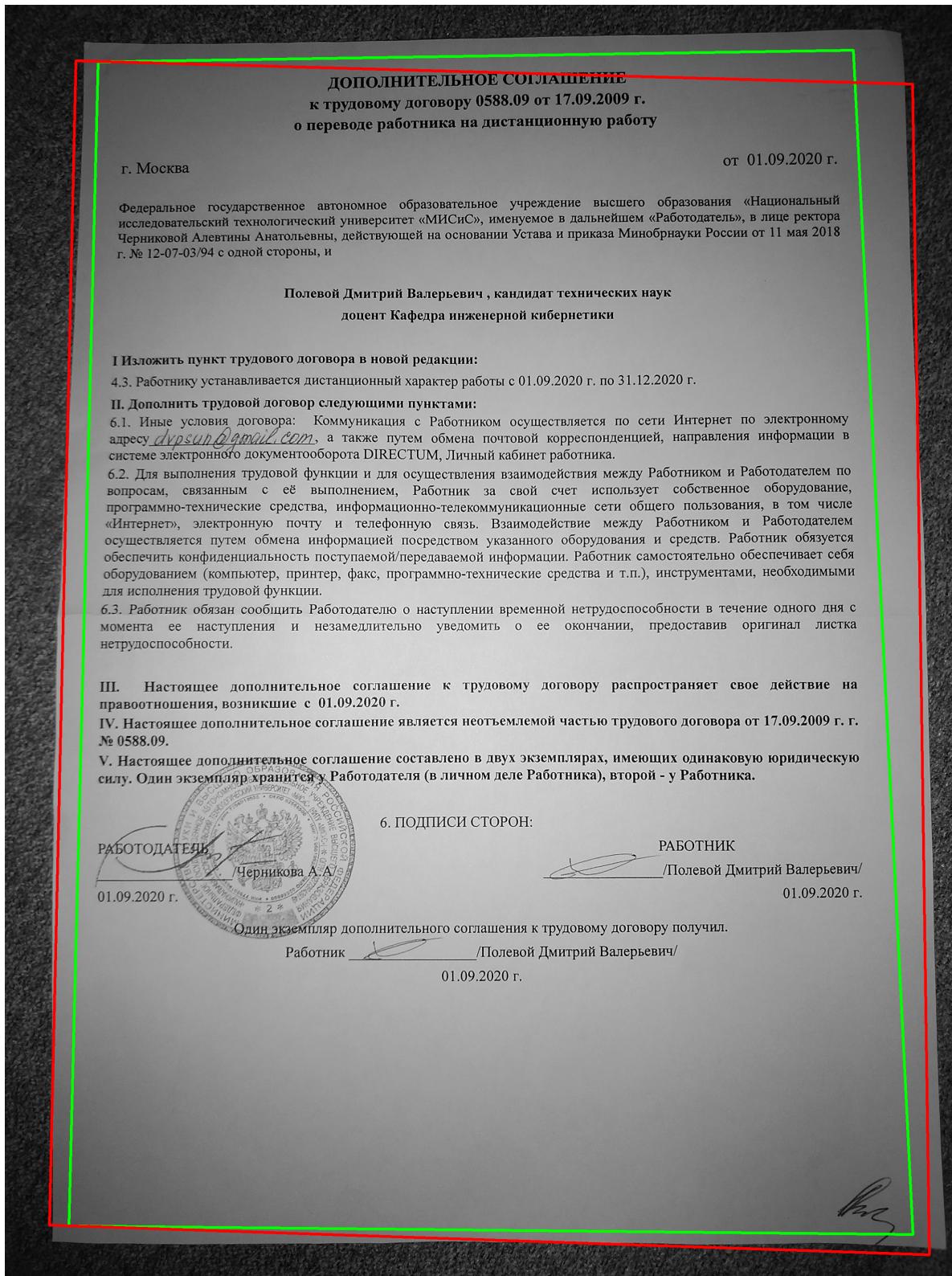


Рис. 3. Найденная и эталонная область для I_2

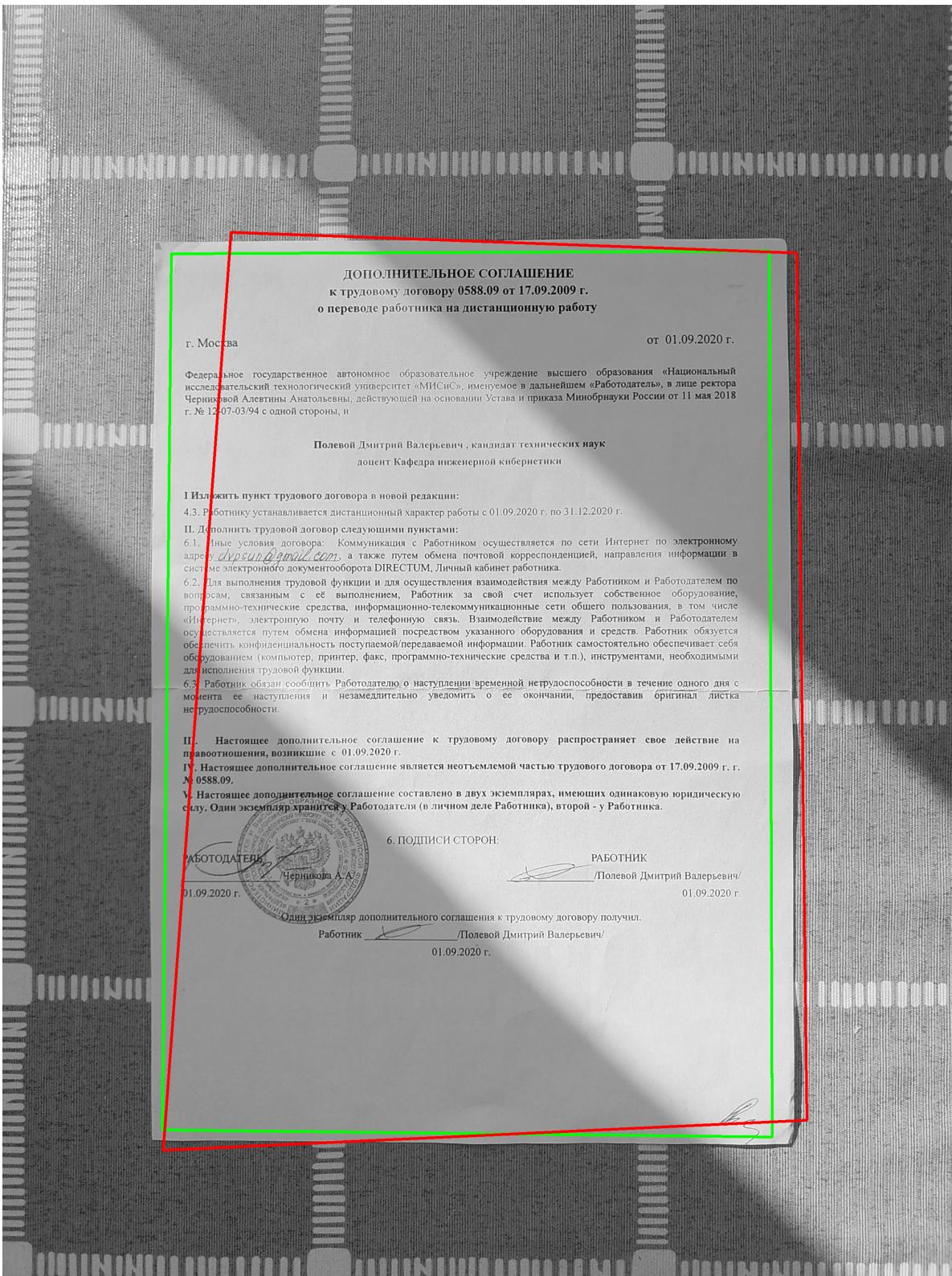


Рис. 4. Найденная и эталонная область для I_3

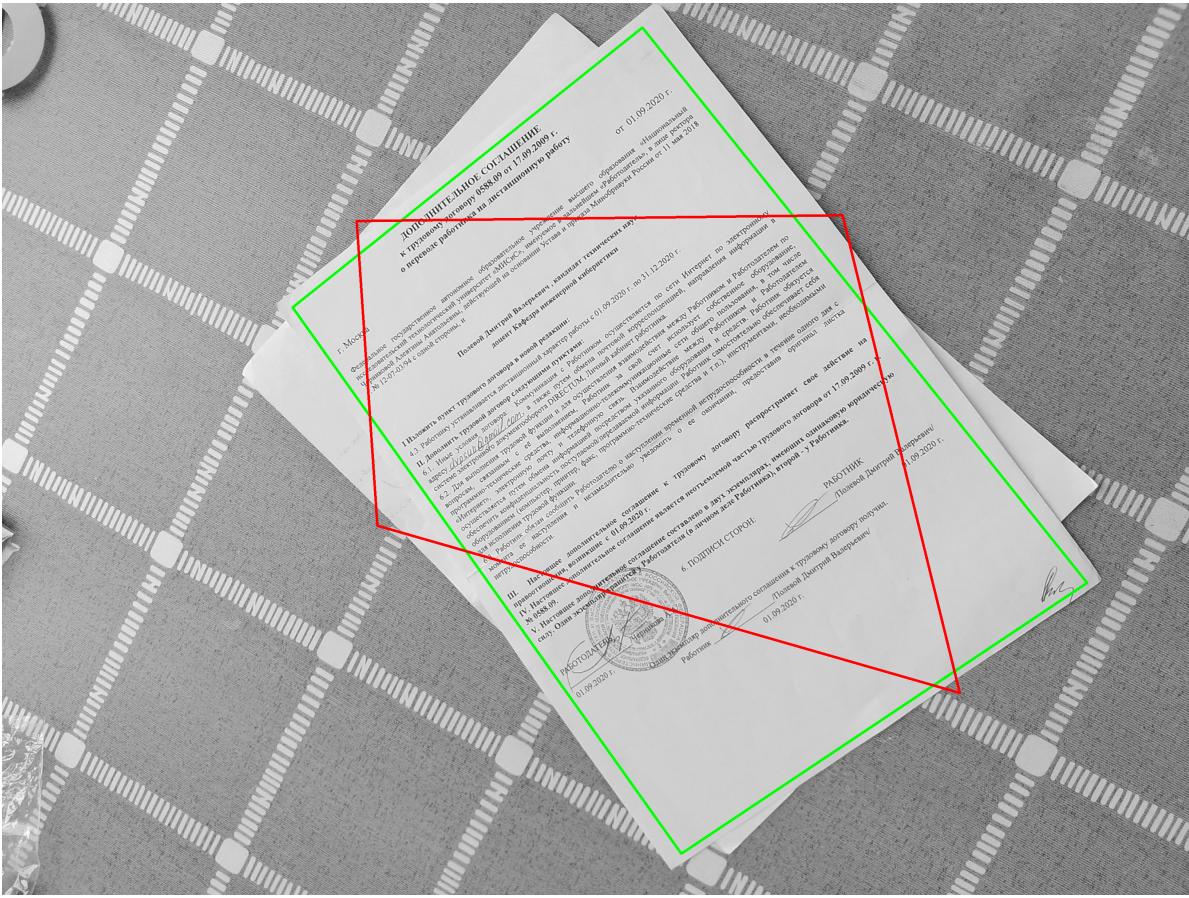


Рис. 5. Найденная и эталонная область для I_4

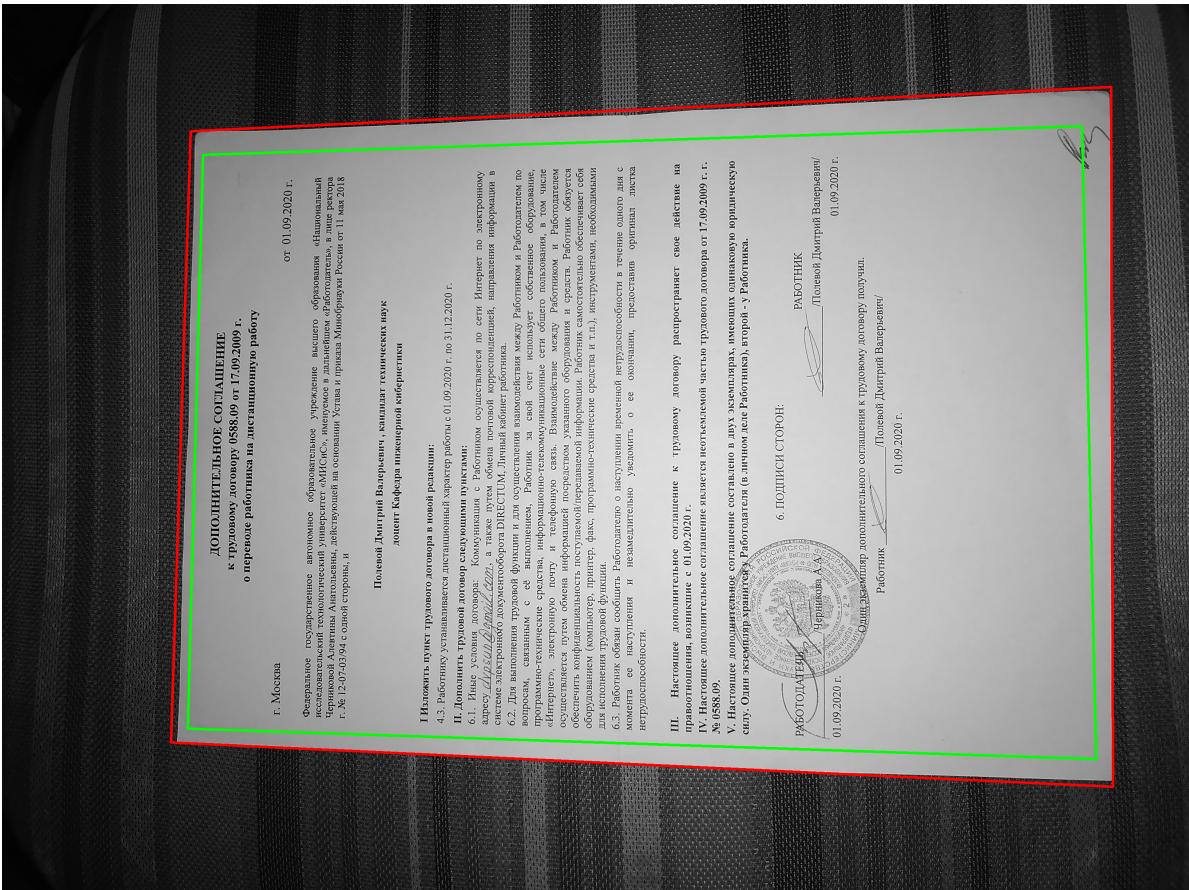


Рис. 6. Найденная и эталонная область для I_5

Численные оценки

RMSE (Root Mean Squared Error)

Для каждого угла найденной области в качестве ошибки оценивается Евклидово расстояние(e) до угла эталона по формуле:

$$RMSE = \sqrt{\frac{1}{n} * \sum e_i^2}, i = 1..n.$$

IoU (Intersection over Union)

$$IoU = \frac{Detected \cap Etalon}{Detected \cup Etalon}$$

Фото	RMSE	IoU
1	97.233	0.902966
2	101.122	0.913933
3	104.686	0.886758
4	566.053	0.590879
5	93.4037	0.883498
Среднее для набора	192.5	0.835607

Таблица 1. Оценки для детектирования

Текст программы

```
#include <opencv2/opencv.hpp>
#include <fstream>
#include "json.hpp"

using namespace std;
using namespace cv;

using json = nlohmann::json;

double euclidDist(Point p_1, Point p_2) {

    return sqrt((p_1.x - p_2.x) * (p_1.x - p_2.x) + (p_1.y - p_2.y) * (p_1.y - p_2.y));
}

Point getClosestPoint(Point p_source, vector<Point> examples) {

    Point closest_point(examples[0]);

    for (int i = 1; i < examples.size(); i++) {
        if (euclidDist(p_source, closest_point) > euclidDist(p_source,
examples[i])) {
            closest_point = examples[i];
        }
    }
}
```

```

    }

}

return closest_point;
}

double rootMeanCornersError(vector<Point> b_1, vector<Point> b_2) {

    double error = 0;
    for (int i = 0; i < b_1.size(); ++i) {

        error += pow(euclidDist(b_1[i], getClosestPoint(b_1[i], b_2)), 2);
    }
    return sqrt(error / b_1.size());
}

//double meanEclidDistErrorLines(vector<Point> b_1, vector<Point> b_2) {
//    double error = 0;
//    for (int i = 0; i < b_1.size()-1; ++i) {
//
//        error += (euclidDist(b_1[i], getClosestPoint(b_1[i], b_2))
//                  + euclidDist(b_1[i + 1], getClosestPoint(b_1[i + 1], b_2)));
//    }
//    error += (euclidDist(b_1[0], getClosestPoint(b_1[0], b_2))
//              + euclidDist(b_1[b_1.size() - 1], getClosestPoint(b_1[b_1.size() - 1],
// b_2)));
//
//    return error / b_1.size();
//}

double IOU(Mat src, vector<Point> detected, vector<Point> etalon) {

    //fill etalon polygon
    Mat src_etalon_borders = Mat::zeros(src.size(), CV_8U);
    fillPoly(src_etalon_borders, etalon, 255);

    //fill detected polygon
    Mat src_detected_borders = Mat::zeros(src.size(), CV_8U);
    fillPoly(src_detected_borders, detected, 255);

    //calculate Intersection over union
    double intersection_val = 0;
    double union_val = 0;

    for (int i = 0; i < src.cols; i++) {
        for (int j = 0; j < src.rows; j++) {
            if ((src_etalon_borders.at<uchar>(j, i) == 255) &
                (src_detected_borders.at<uchar>(j, i) == 255)) {
                intersection_val += 1;
            }
            if ((src_etalon_borders.at<uchar>(j, i) == 255) |
                (src_detected_borders.at<uchar>(j, i) == 255)) {
                union_val += 1;
            }
        }
    }
    return intersection_val / union_val;
}

```

```

        }

    }

    return intersection_val / union_val;
}

vector<Point> getEtalonBorders(json photo_borders) {

    //etalon borders
    string file_name_photo = photo_borders["_via_image_id_list"][0];

    vector<Point> etalon_points;
    for (int i = 0; i < photo_borders["_via_img_metadata"][file_name_photo]
    ["regions"].size(); i++) {
        for (int j = 0; j < photo_borders["_via_img_metadata"][file_name_photo]
        ["regions"][i]["shape_attributes"]["all_points_x"].size(); j++) {

            int x = photo_borders["_via_img_metadata"][file_name_photo]
            ["regions"][i]["shape_attributes"]["all_points_x"][j];
            int y = photo_borders["_via_img_metadata"][file_name_photo]
            ["regions"][i]["shape_attributes"]["all_points_y"][j];

            etalon_points.push_back(Point(x, y));
        }
    }

    return etalon_points;
}

Mat borderBlur(Mat photo) {

    Mat f_55;
    medianBlur(photo, f_55, 55);

    GaussianBlur(f_55, f_55, Size(71, 71), 0, 0);

    Mat bin;
    adaptiveThreshold(f_55, bin, 255, ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY,
    71, 3);

    Mat f_25;
    medianBlur(bin, f_25, 25);

    bitwise_not(f_25, f_25);
    Mat labelImage, statsImage, centroidsImage;
    int nLabels = connectedComponentsWithStats(f_25, labelImage, statsImage,
    centroidsImage, 8);
    cout << nLabels << endl;

    Mat mask = Mat::zeros(f_25.size(), CV_8UC1);
}

```

```

        for (int i = 1; i < statsImage.rows; ++i) {

            if (statsImage.at<int>(Point(CC_STAT_AREA, i)) > 600)
            {
                mask += (labelImage == i);
            }
        }

        Mat res = Mat::zeros(f_25.size(), CV_8UC1);
        res += mask;
        bitwise_not(res, res);

        return res;
    }

vector<Point> getBordersPoints(Mat src) {

    //blur all except list borders
    Mat blured;
    blured = borderBlur(src);

    //detect edges with Canny
    Mat can_img;
    Canny(blured, can_img, 100, 200);

    //Find lines on canny image
    vector<Vec4i> linesP;
    HoughLinesP(can_img, linesP, 2, CV_PI / 180, 250, 200, 250);

    //find list borders points
    Point left_top(0, 0);
    Point right_top(src.cols, 0);
    Point right_bot(src.cols, src.rows);
    Point left_bot(0, src.rows);

    vector<Point> line_points;

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

        Vec4i l = linesP[i];
        Point p_1 = Point(l[0], l[1]);
        Point p_2 = Point(l[2], l[3]);

        line_points.push_back(p_1);
        line_points.push_back(p_2);
    }

    Point left_up_border = getClosestPoint(left_top, line_points);
    Point right_up_border = getClosestPoint(right_top, line_points);
    Point right_bottom_border = getClosestPoint(right_bot, line_points);
    Point left_bottom_border = getClosestPoint(left_bot, line_points);
}

```

```

        vector<Point> res = { left_up_border , right_up_border, right_bottom_border,
left_bottom_border };

    return res;
}

Mat drawBorders(Mat src, vector<Point> borders, Scalar color) {

    Mat src_with_borders = src.clone();

    line(src_with_borders, borders[0], borders[1], color, 5);
    line(src_with_borders, borders[0], borders[3], color, 5);
    line(src_with_borders, borders[1], borders[2], color, 5);
    line(src_with_borders, borders[3], borders[2], color, 5);

    return src_with_borders;
}

int main() {

//load photos
Mat photo_1 = imread("P_20210309_064128_vHDR_Auto.jpg", IMREAD_GRAYSCALE);
Mat photo_2 = imread("P_20210309_064141_vHDR_Auto.jpg", IMREAD_GRAYSCALE);
Mat photo_3 = imread("P_20210410_170813_vHDR_On.jpg", IMREAD_GRAYSCALE);
Mat photo_4 = imread("P_20210410_171055_vHDR_On.jpg", IMREAD_GRAYSCALE);
Mat photo_5 = imread("P_20210410_171334_vHDR_On.jpg", IMREAD_GRAYSCALE);

//load borders
ifstream photo_1_borders_stream("P_20210309_064128_vHDR_Auto.json");
ifstream photo_2_borders_stream("P_20210309_064141_vHDR_Auto.json");
ifstream photo_3_borders_stream("P_20210410_170813_vHDR_On.json");
ifstream photo_4_borders_stream("P_20210410_171055_vHDR_On.json");
ifstream photo_5_borders_stream("P_20210410_171334_vHDR_On.json");

json photo_1_borders;
json photo_2_borders;
json photo_3_borders;
json photo_4_borders;
json photo_5_borders;

photo_1_borders_stream >> photo_1_borders;
photo_2_borders_stream >> photo_2_borders;
photo_3_borders_stream >> photo_3_borders;
photo_4_borders_stream >> photo_4_borders;
photo_5_borders_stream >> photo_5_borders;

//get etalon borders
vector<Point> etalon_borders_1 = getEtalonBorders(photo_1_borders);
vector<Point> etalon_borders_2 = getEtalonBorders(photo_2_borders);
vector<Point> etalon_borders_3 = getEtalonBorders(photo_3_borders);
vector<Point> etalon_borders_4 = getEtalonBorders(photo_4_borders);
vector<Point> etalon_borders_5 = getEtalonBorders(photo_5_borders);

Mat borders_photo_1 = photo_1.clone();

```

```

Mat borders_photo_2 = photo_2.clone();
Mat borders_photo_3 = photo_3.clone();
Mat borders_photo_4 = photo_4.clone();
Mat borders_photo_5 = photo_5.clone();

cvtColor(borders_photo_1, borders_photo_1, COLOR_GRAY2BGR);
cvtColor(borders_photo_2, borders_photo_2, COLOR_GRAY2BGR);
cvtColor(borders_photo_3, borders_photo_3, COLOR_GRAY2BGR);
cvtColor(borders_photo_4, borders_photo_4, COLOR_GRAY2BGR);
cvtColor(borders_photo_5, borders_photo_5, COLOR_GRAY2BGR);

//draw etalon borders
borders_photo_1 = drawBorders(borders_photo_1, etalon_borders_1, Scalar(0, 255, 0));
borders_photo_2 = drawBorders(borders_photo_2, etalon_borders_2, Scalar(0, 255, 0));
borders_photo_3 = drawBorders(borders_photo_3, etalon_borders_3, Scalar(0, 255, 0));
borders_photo_4 = drawBorders(borders_photo_4, etalon_borders_4, Scalar(0, 255, 0));
borders_photo_5 = drawBorders(borders_photo_5, etalon_borders_5, Scalar(0, 255, 0));

//Mat blured_1 = borderBlur(photo_1);
//Mat blured_2 = borderBlur(photo_2);
//Mat blured_3 = borderBlur(photo_3);
//Mat blured_4 = borderBlur(photo_4);
//Mat blured_5 = borderBlur(photo_5);

//imwrite("blured_1.png", blured_1);
//imwrite("blured_2.png", blured_2);
//imwrite("blured_3.png", blured_3);
//imwrite("blured_4.png", blured_4);
//imwrite("blured_5.png", blured_5);

//get detected borders
vector<Point> detected_borders_1 = getBordersPoints(photo_1);
vector<Point> detected_borders_2 = getBordersPoints(photo_2);
vector<Point> detected_borders_3 = getBordersPoints(photo_3);
vector<Point> detected_borders_4 = getBordersPoints(photo_4);
vector<Point> detected_borders_5 = getBordersPoints(photo_5);

//draw detected borders
borders_photo_1 = drawBorders(borders_photo_1, detected_borders_1, Scalar(0, 255));
borders_photo_2 = drawBorders(borders_photo_2, detected_borders_2, Scalar(0, 255));
borders_photo_3 = drawBorders(borders_photo_3, detected_borders_3, Scalar(0, 255));
borders_photo_4 = drawBorders(borders_photo_4, detected_borders_4, Scalar(0, 255));
borders_photo_5 = drawBorders(borders_photo_5, detected_borders_5, Scalar(0, 255));

//save result
imwrite("detected_borders_photo_1.png", borders_photo_1);
imwrite("detected_borders_photo_2.png", borders_photo_2);

```

```

imwrite("detected_borders_photo_3.png", borders_photo_3);
imwrite("detected_borders_photo_4.png", borders_photo_4);
imwrite("detected_borders_photo_5.png", borders_photo_5);

//calculate error for points
double corners_error_photo_1 = rootMeanCornersError(detected_borders_1,
etalon_borders_1);
double corners_error_photo_2 = rootMeanCornersError(detected_borders_2,
etalon_borders_2);
double corners_error_photo_3 = rootMeanCornersError(detected_borders_3,
etalon_borders_3);
double corners_error_photo_4 = rootMeanCornersError(detected_borders_4,
etalon_borders_4);
double corners_error_photo_5 = rootMeanCornersError(detected_borders_5,
etalon_borders_5);

double corners_mean_error = (corners_error_photo_1 + corners_error_photo_2 +
corners_error_photo_3
+ corners_error_photo_4 + corners_error_photo_5) / 5;

//display errors for photos
cout << "Corners error for photo_1: " << corners_error_photo_1 << endl;
cout << "Corners error for photo_2: " << corners_error_photo_2 << endl;
cout << "Corners error for photo_3: " << corners_error_photo_3 << endl;
cout << "Corners error for photo_4: " << corners_error_photo_4 << endl;
cout << "Corners error for photo_5: " << corners_error_photo_5 << endl;
cout << "Corners mean error: " << corners_mean_error << endl;

//calculate IOU
double IoU_photo_1 = IoU(photo_1, detected_borders_1, etalon_borders_1);
double IoU_photo_2 = IoU(photo_2, detected_borders_2, etalon_borders_2);
double IoU_photo_3 = IoU(photo_3, detected_borders_3, etalon_borders_3);
double IoU_photo_4 = IoU(photo_4, detected_borders_4, etalon_borders_4);
double IoU_photo_5 = IoU(photo_5, detected_borders_5, etalon_borders_5);

double IoU_mean = (IoU_photo_1 + IoU_photo_2 + IoU_photo_3 +
IoU_photo_4 + IoU_photo_5) / 5;

cout << "IoU for photo_1: " << IoU_photo_1 << endl;
cout << "IoU for photo_2: " << IoU_photo_2 << endl;
cout << "IoU for photo_3: " << IoU_photo_3 << endl;
cout << "IoU for photo_4: " << IoU_photo_4 << endl;
cout << "IoU for photo_5: " << IoU_photo_5 << endl;
cout << "IoU mean: " << IoU_mean << endl;

waitKey(0);
return 0;
}

```