

# Работа 4. Детектирование границ документов на кадрах видео

---

автор: Балаев А.А. дата: 2022-04-11T00:26:20

## Задание

0. текст, иллюстрации и подписи отчета придумываем самостоятельно
1. самостоятельно снимаем видео смартфоном
  - объект съемки - купюры (рубли разного номинала), расправленные и лежащие на поверхности (проективно искаженны прямоугольник)
  - количество роликов - от 5 шт.
  - длительность - 5-7 сек
  - условия съемки разные
2. извлекаем по 3 кадра из каждого ролика (делим кол-во кадров на 5 и берем каждый с индексом 2/5,3/5,4/5)
3. цветоредуцируем изображения
4. бинаризцем изображения
5. морфологически обрабатываем изображения
6. выделяем основную компоненту связности
7. руками изготавливаем маски (идеальная зона купюры)
8. оцениваем качество выделение зоны и анализируем ошибки

## Результаты

### Исходные кадры из каждого ролика



Рис. 1. Кадры со 100 рублевой купюры на темном фоне

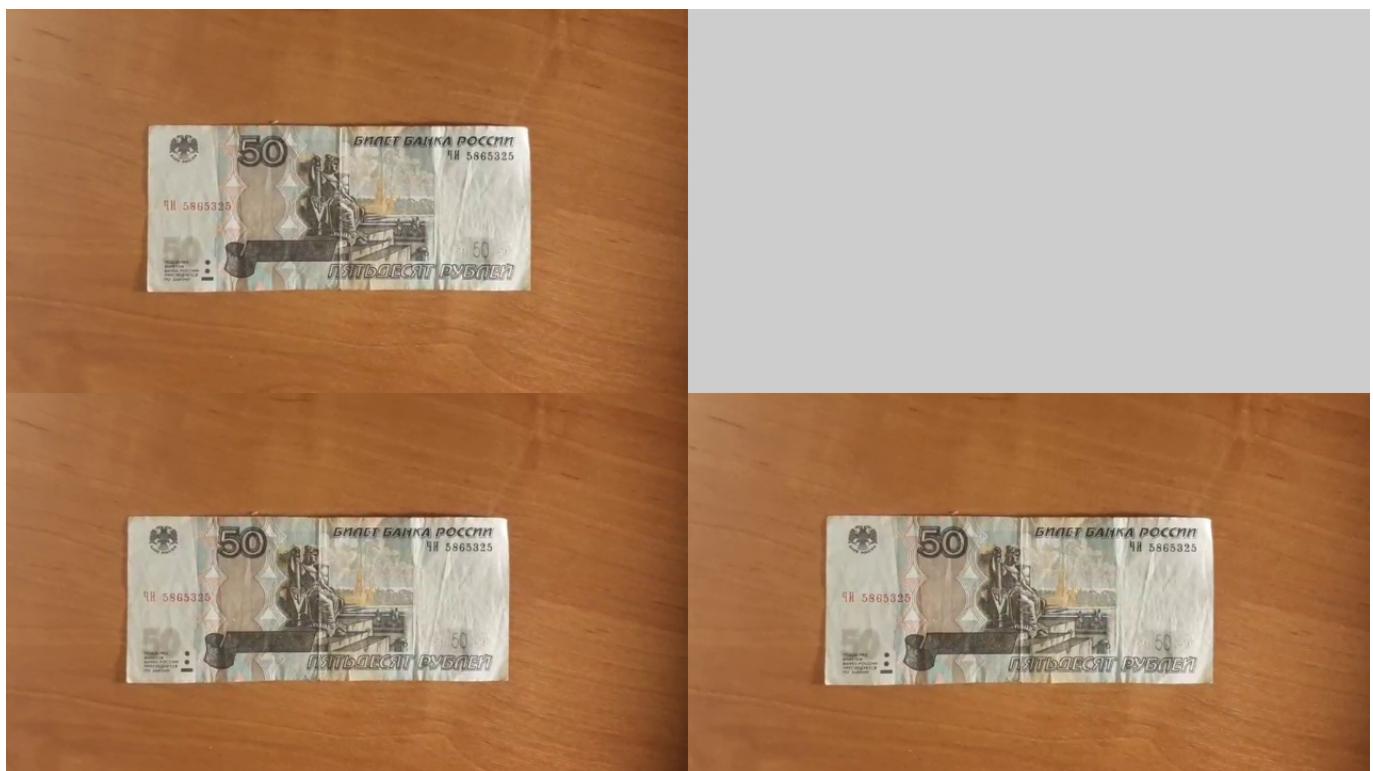


Рис. 2. Кадры со 50 рублевой купюрой на столе



Рис. 3. Кадры со 200 рублевой купюры на синем фоне

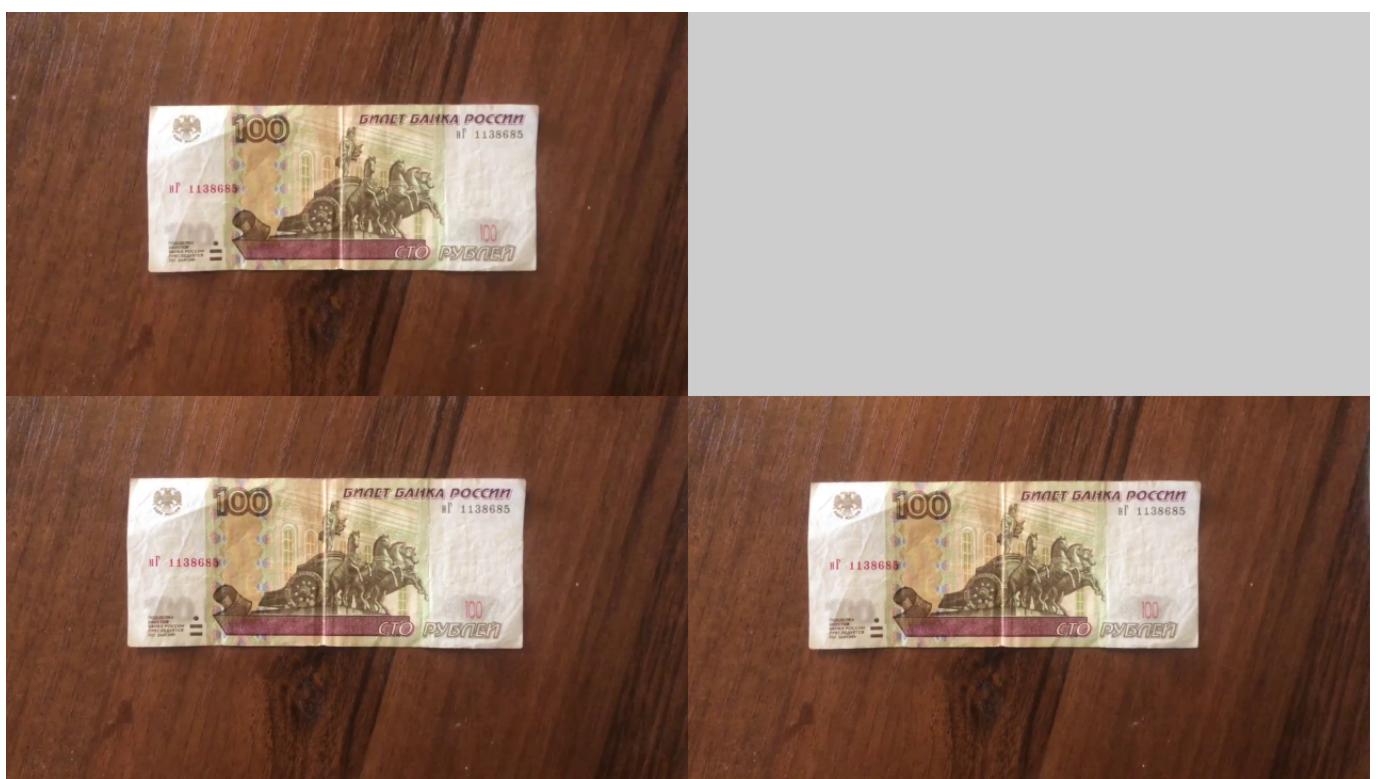


Рис. 4. Кадры со 100 рублевой купюрой на столе



Рис. 5. Кадры со 500 рублевой купюры на темном фоне

## Цветоредуцированные изображения



Рис. 1. Цветоредуцированные изображения 100 рублевой купюры на темном фоне

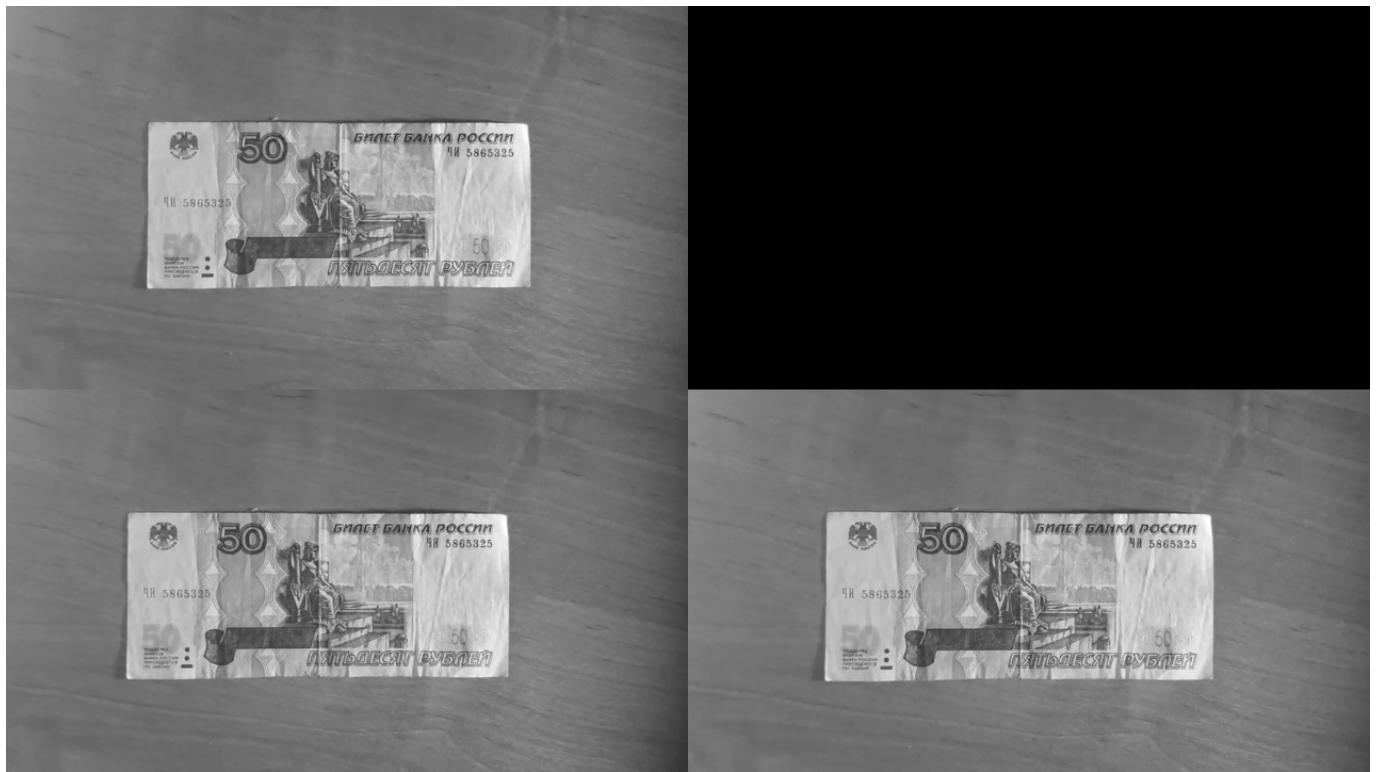


Рис. 2. Цветоредуцированные изображения 50 рублевой купюры на столе

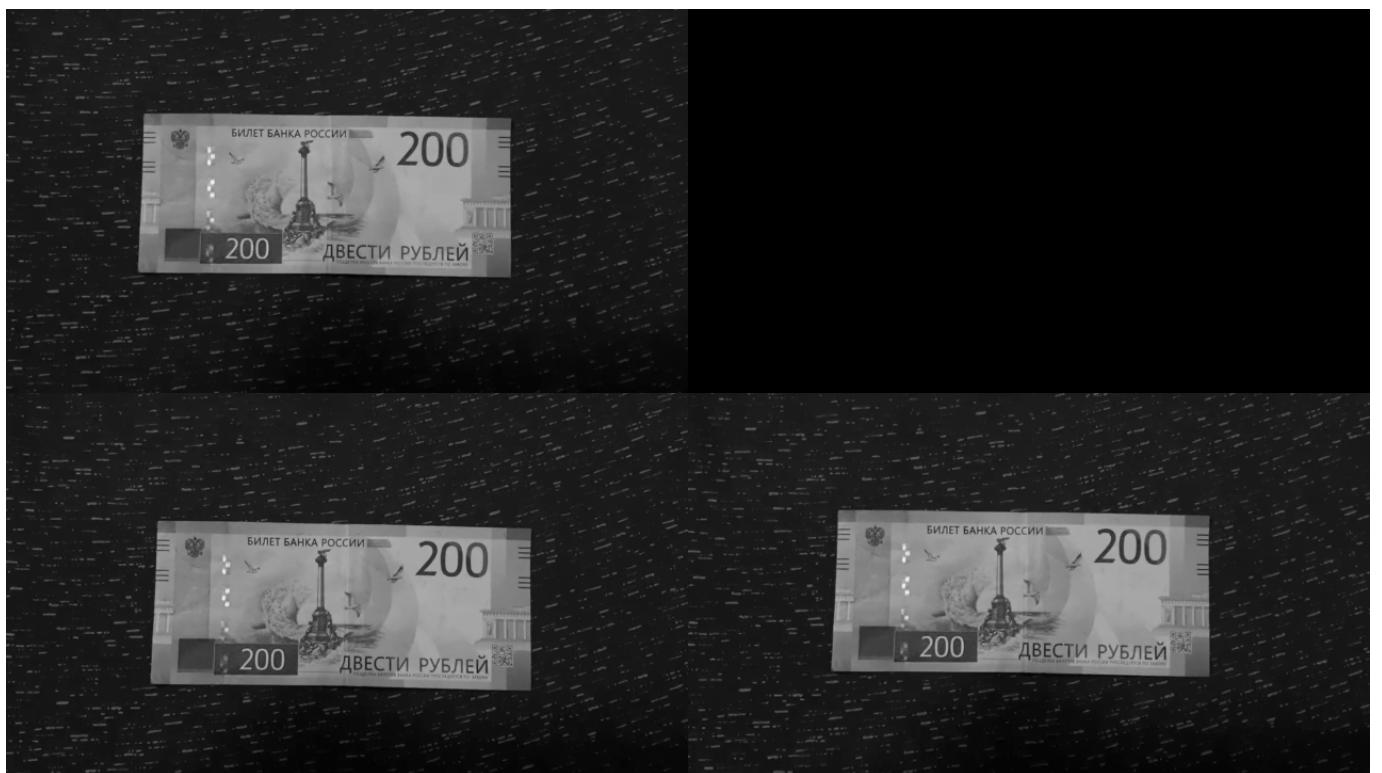


Рис. 3. Цветоредуцированные изображения 200 рублевой купюры на синем фоне

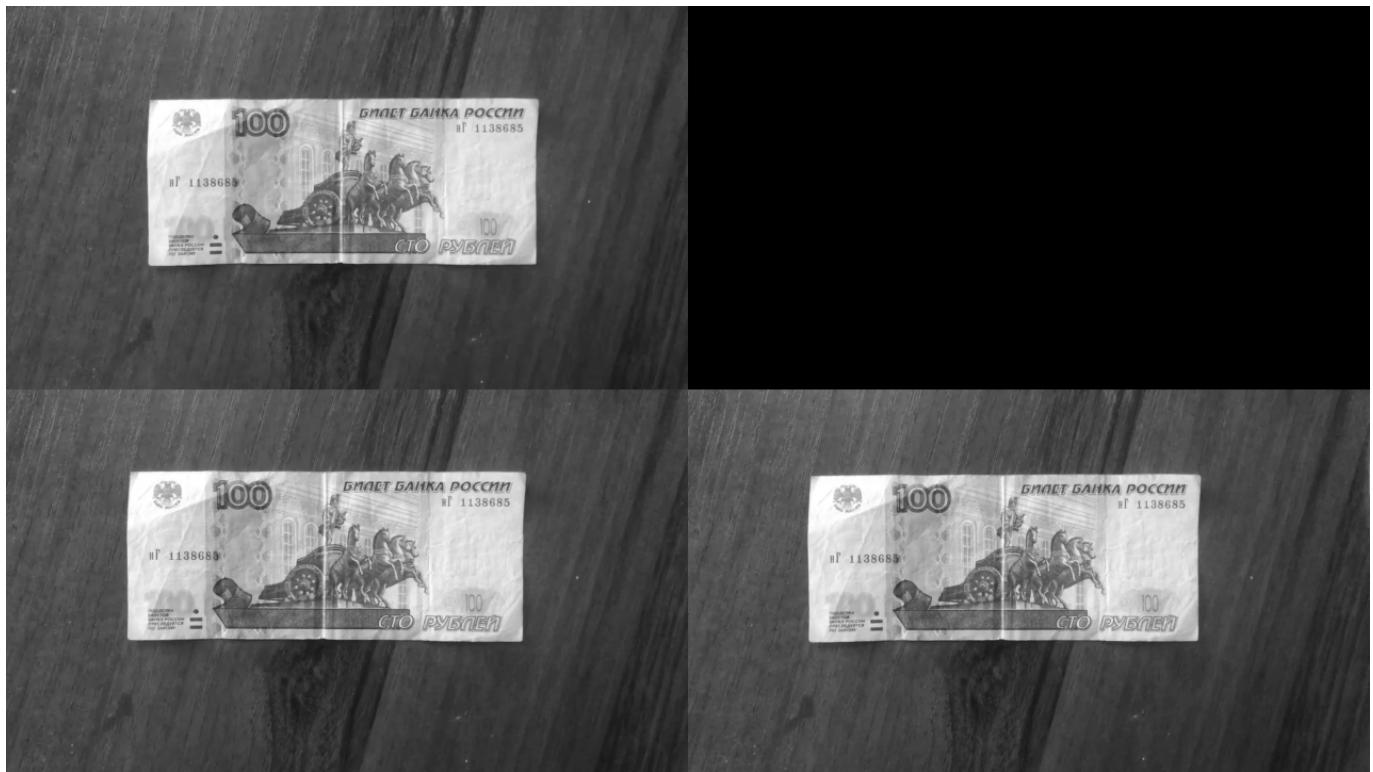


Рис. 4. Цветоредуцированные изображения 100 рублевой купюры на столе



Рис. 5. Цветоредуцированные изображения 500 рублевой купюры на темном фоне

## Бинаризованные изображения



Рис. 1. Бинаризированные изображения 100 рублевой купюры на темном фоне

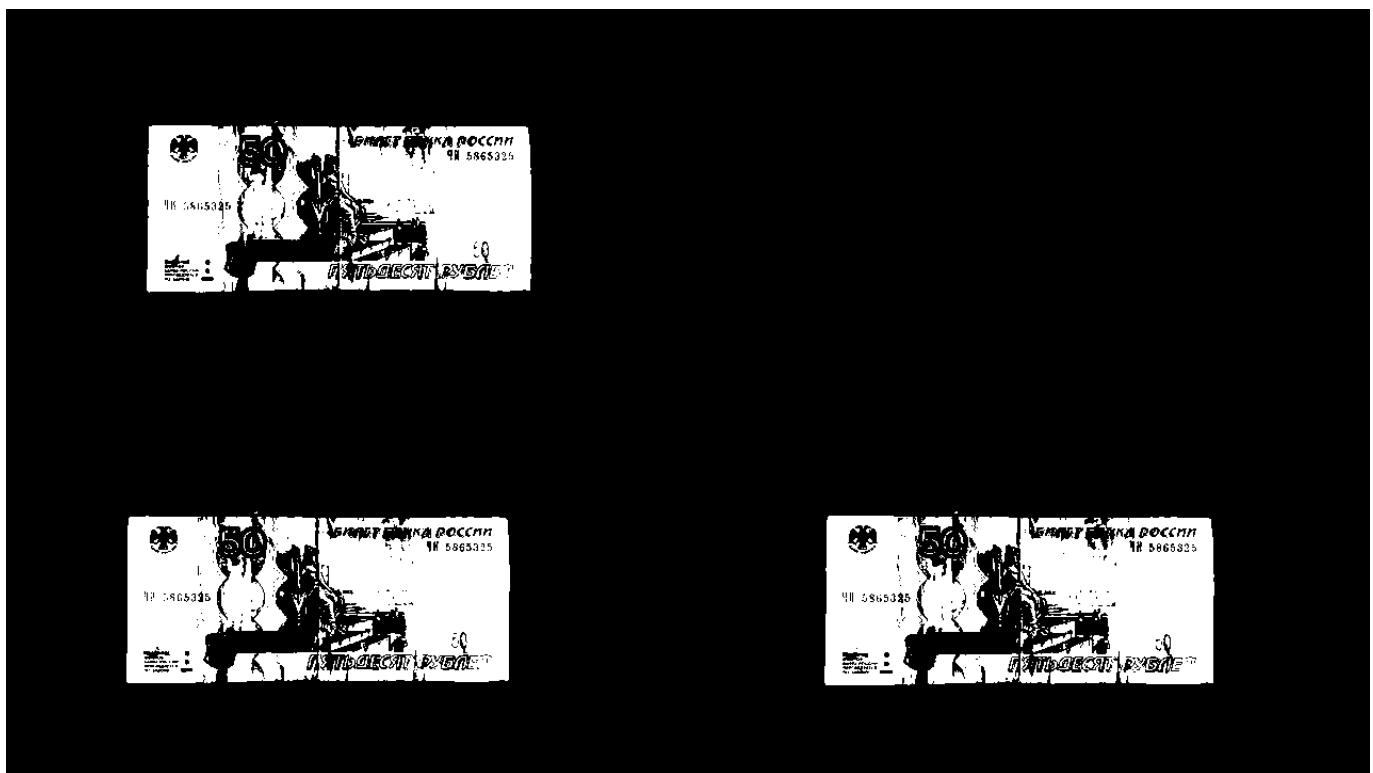


Рис. 2. Бинаризированные изображения 50 рублевой купюры на столе

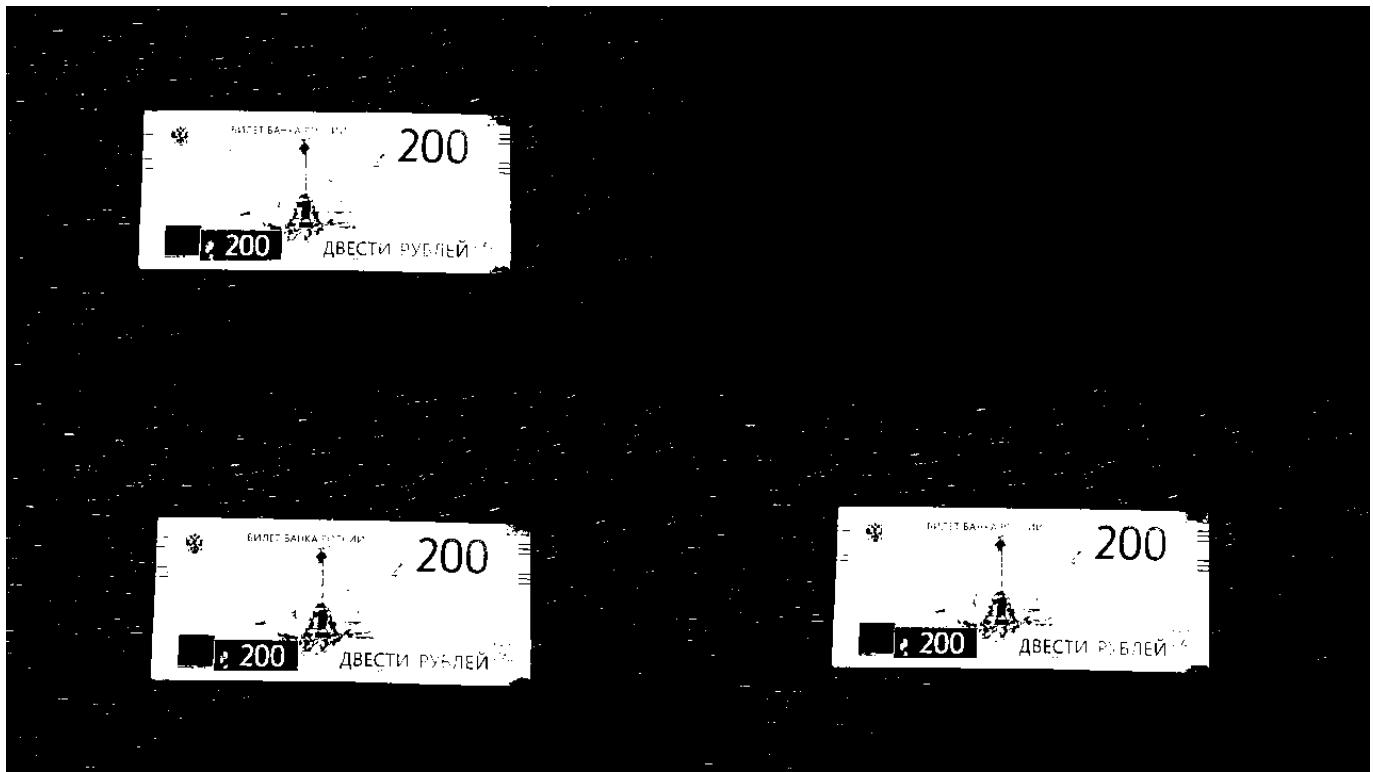


Рис. 3. Бинаризированные изображения 200 рублевой купюры на синем фоне

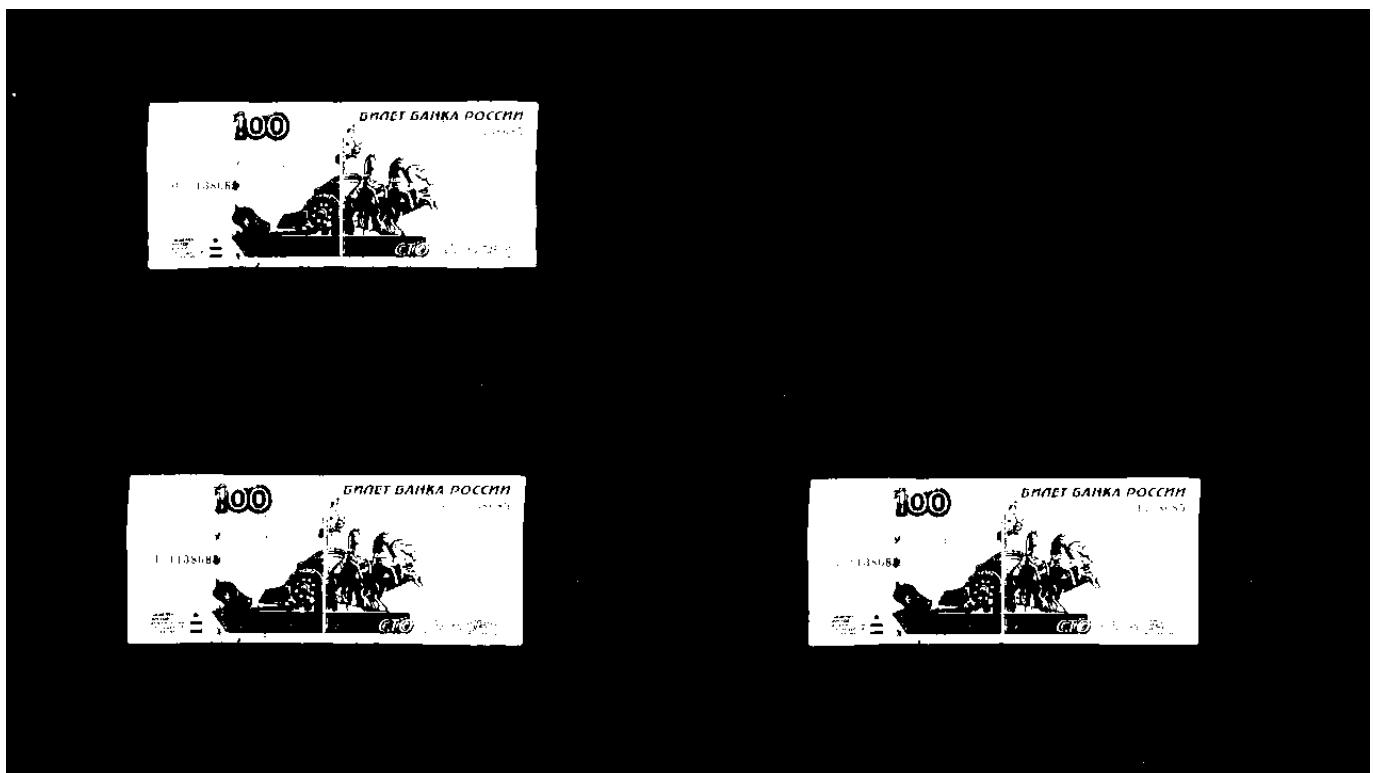


Рис. 4. Бинаризированные изображения 100 рублевой купюры на столе

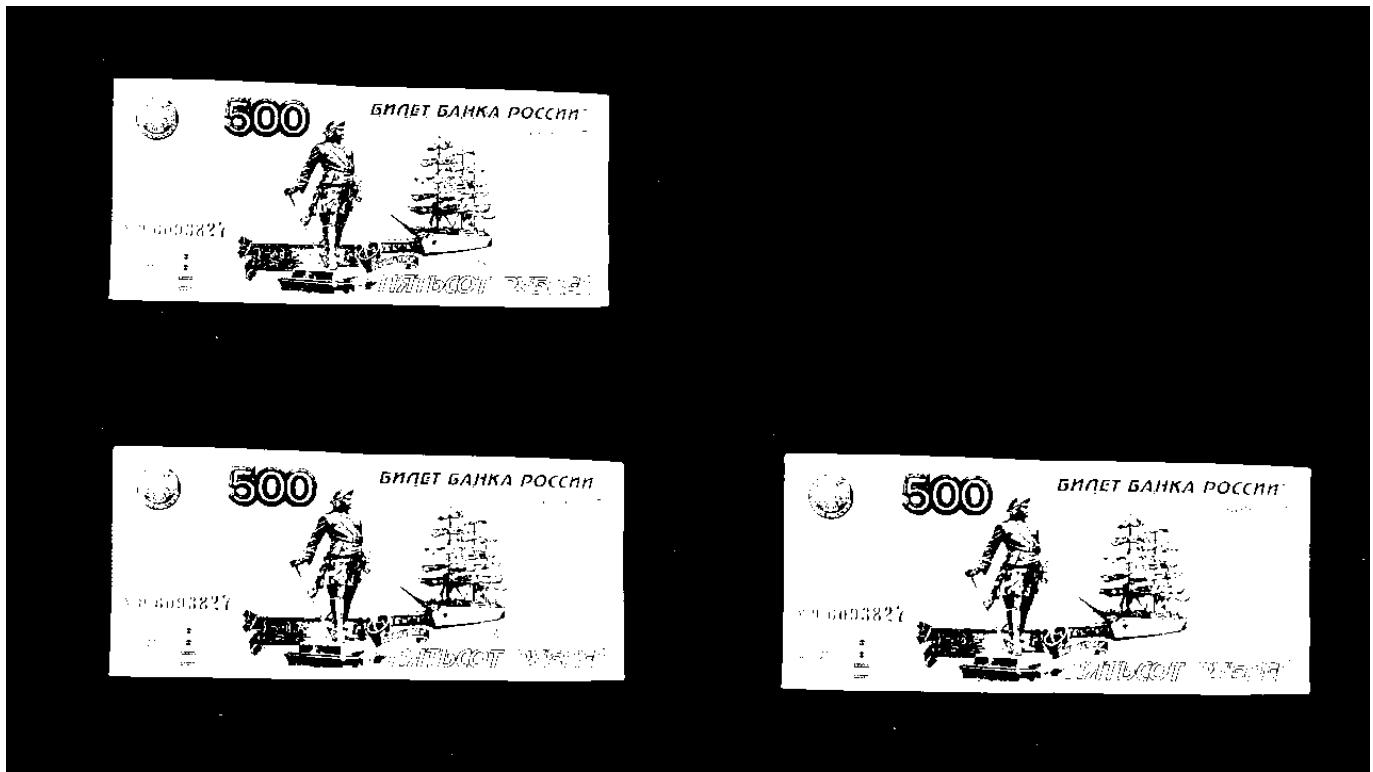


Рис. 5. Бинаризированные изображения 500 рублевой купюры на темном фоне

### Морфологически обработанные изображения

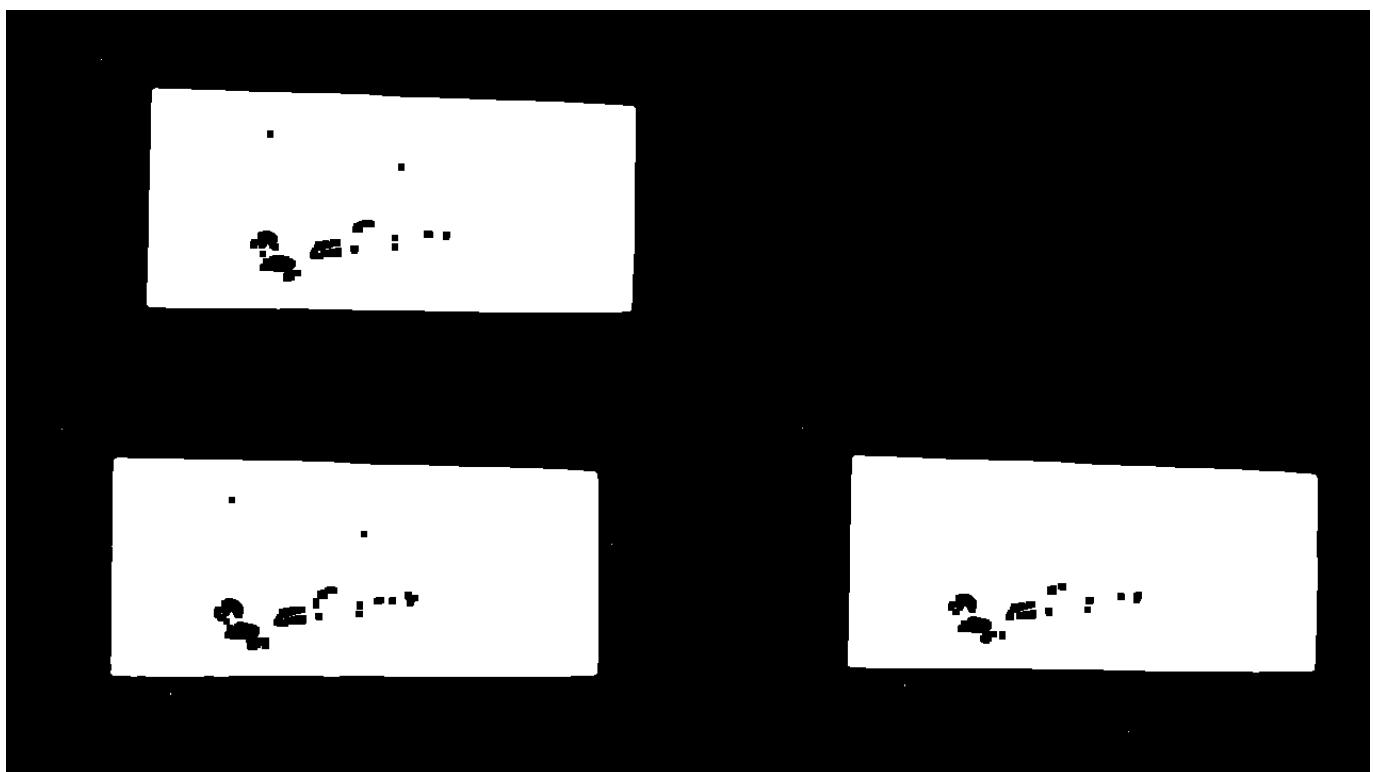


Рис. 1. Морфологически обработанные изображения 100 рублевой купюры на темном фоне

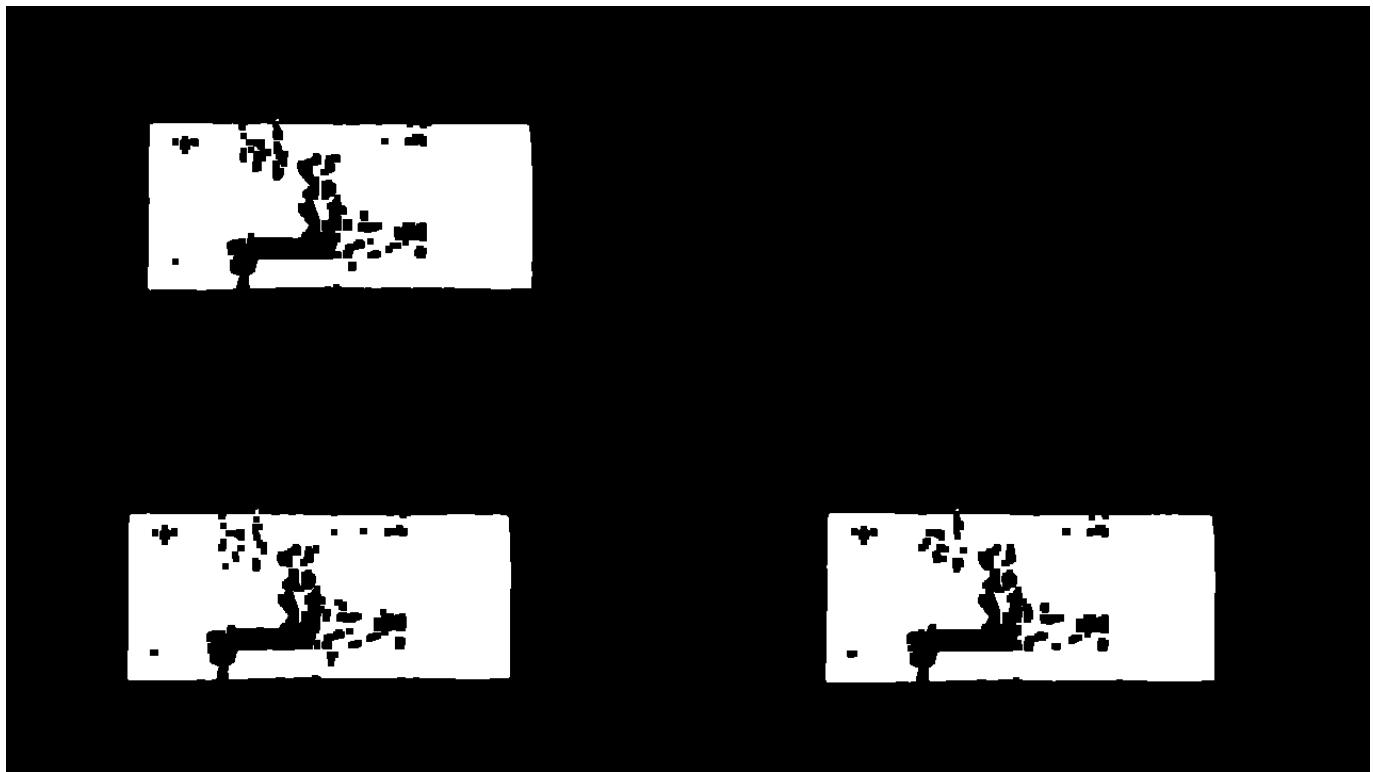


Рис. 2. Морфологически обработанные изображения 50 рублевой купюры на столе

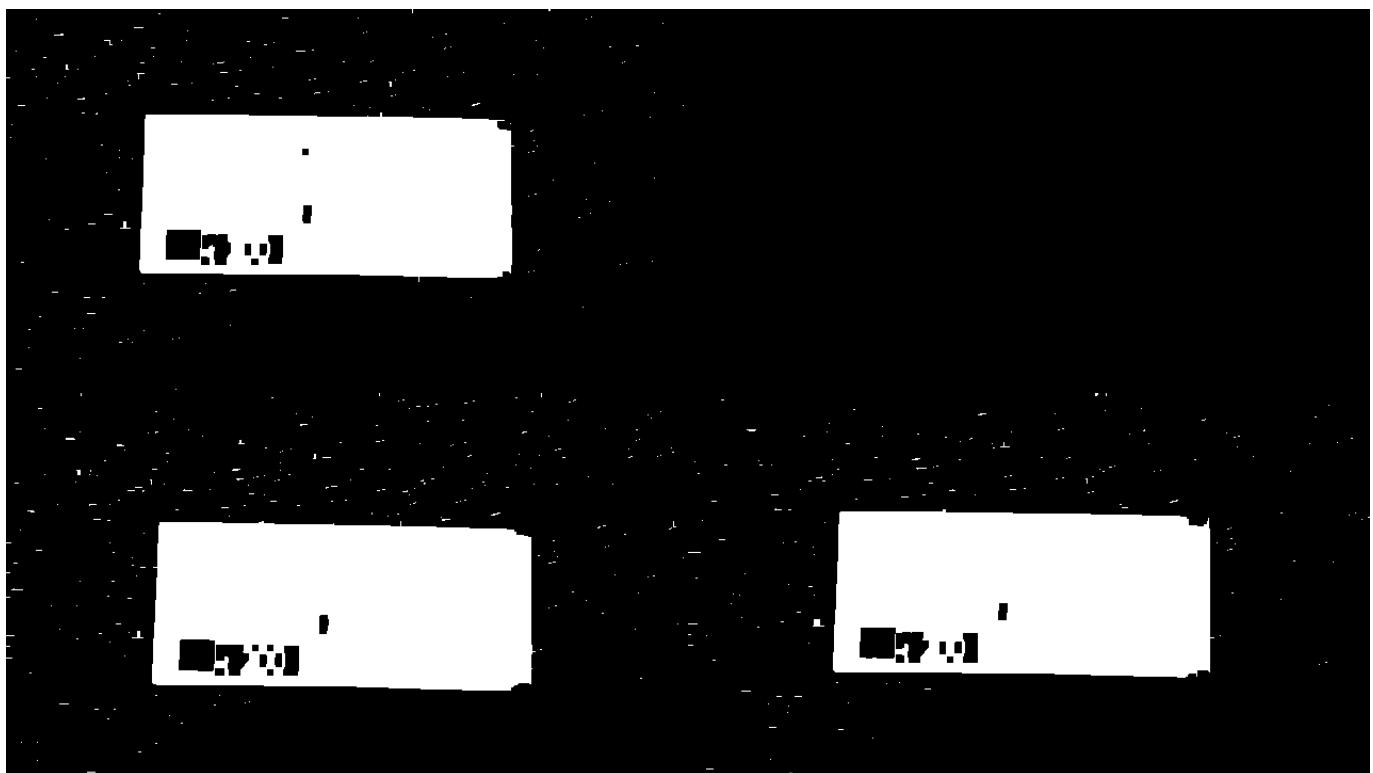


Рис. 3. Морфологически обработанные изображения 200 рублевой купюры на синем фоне

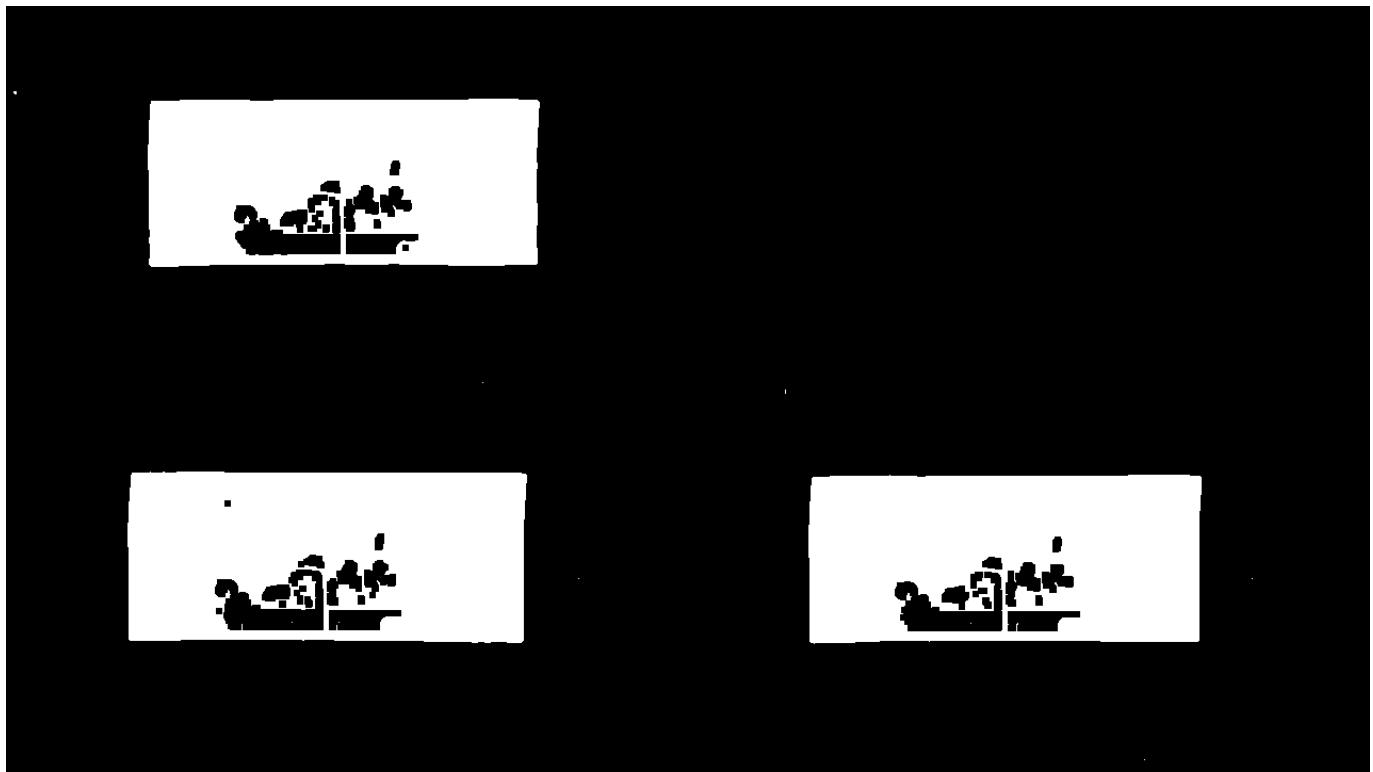


Рис. 4. Морфологически обработанные изображения 100 рублевой купюры на столе

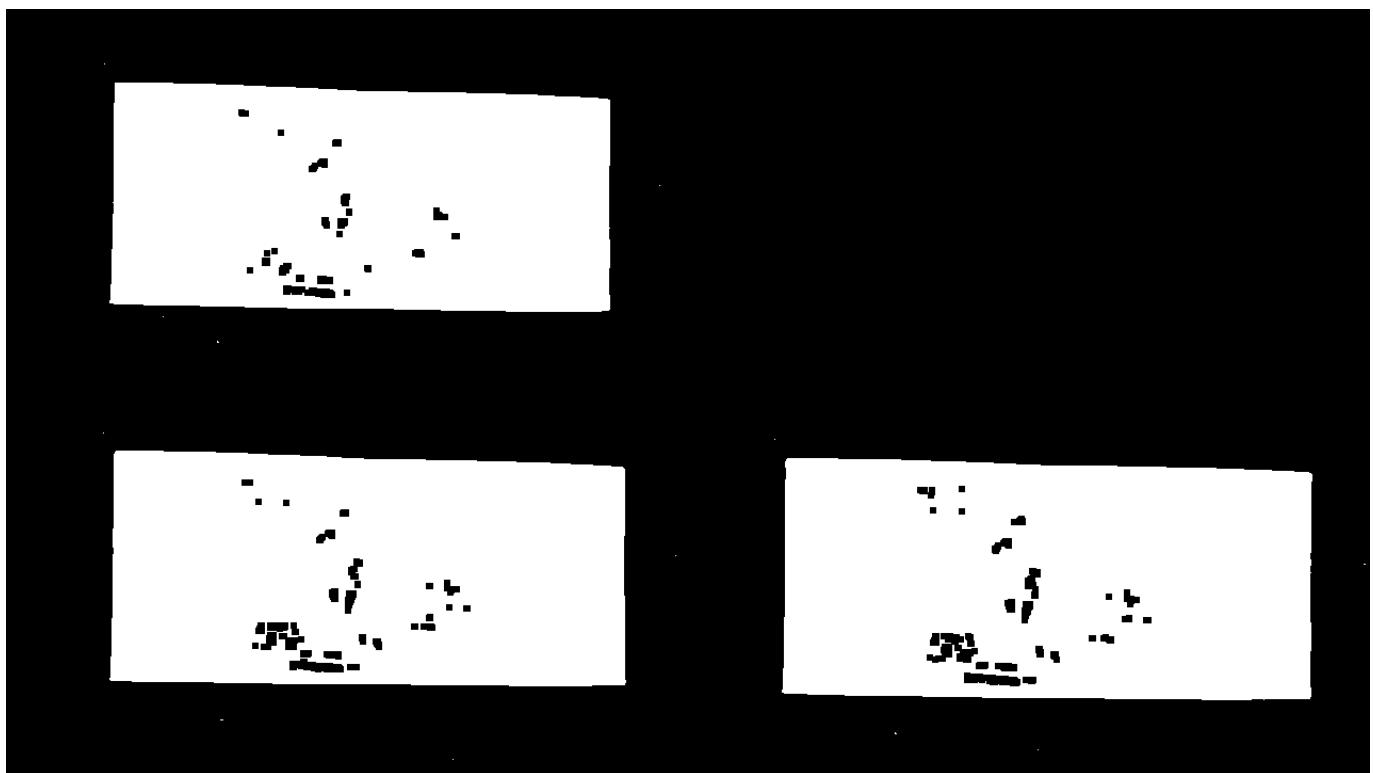


Рис. 5. Морфологически обработанные изображения 500 рублевой купюры на темном фоне

#### **Выделенные основные компоненты связности**

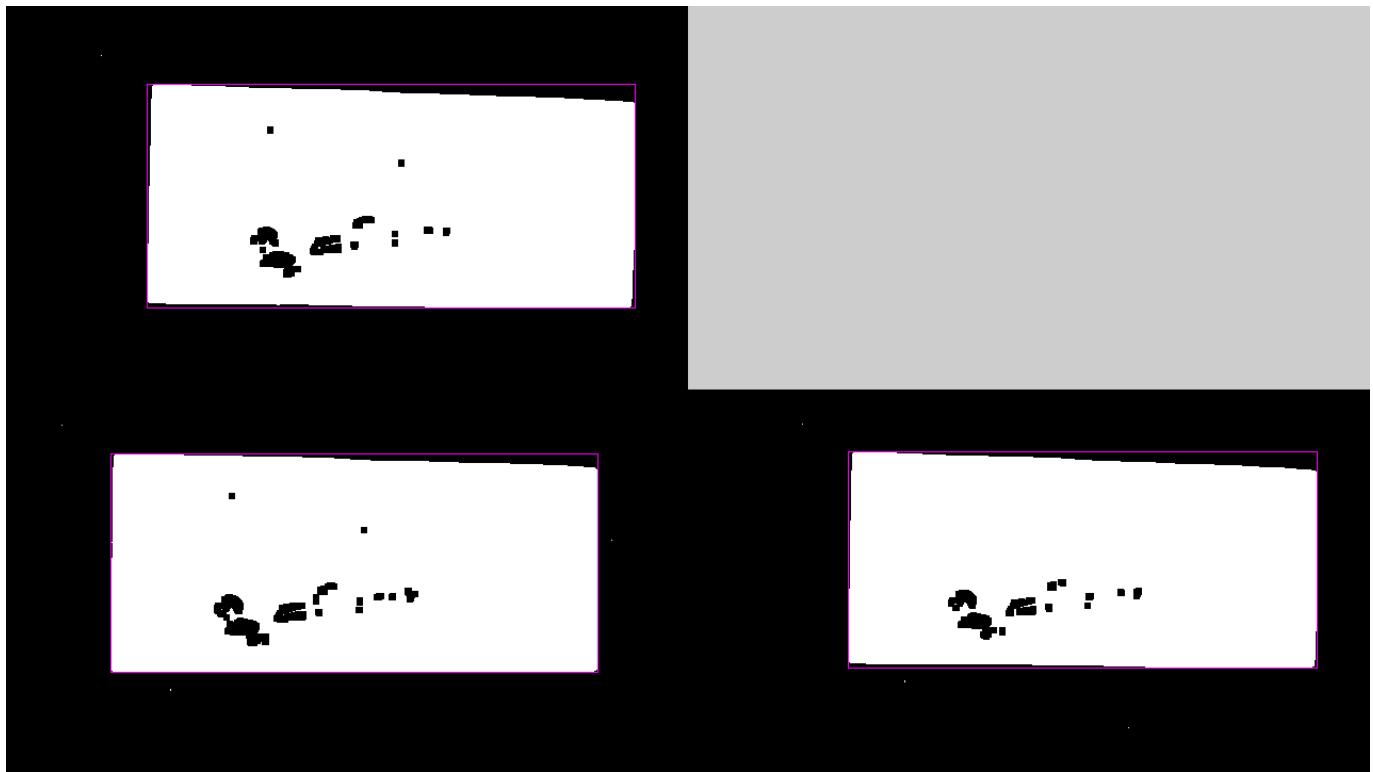


Рис. 1. Выделенные основные компоненты связности 100 рублей купюры на темном фоне

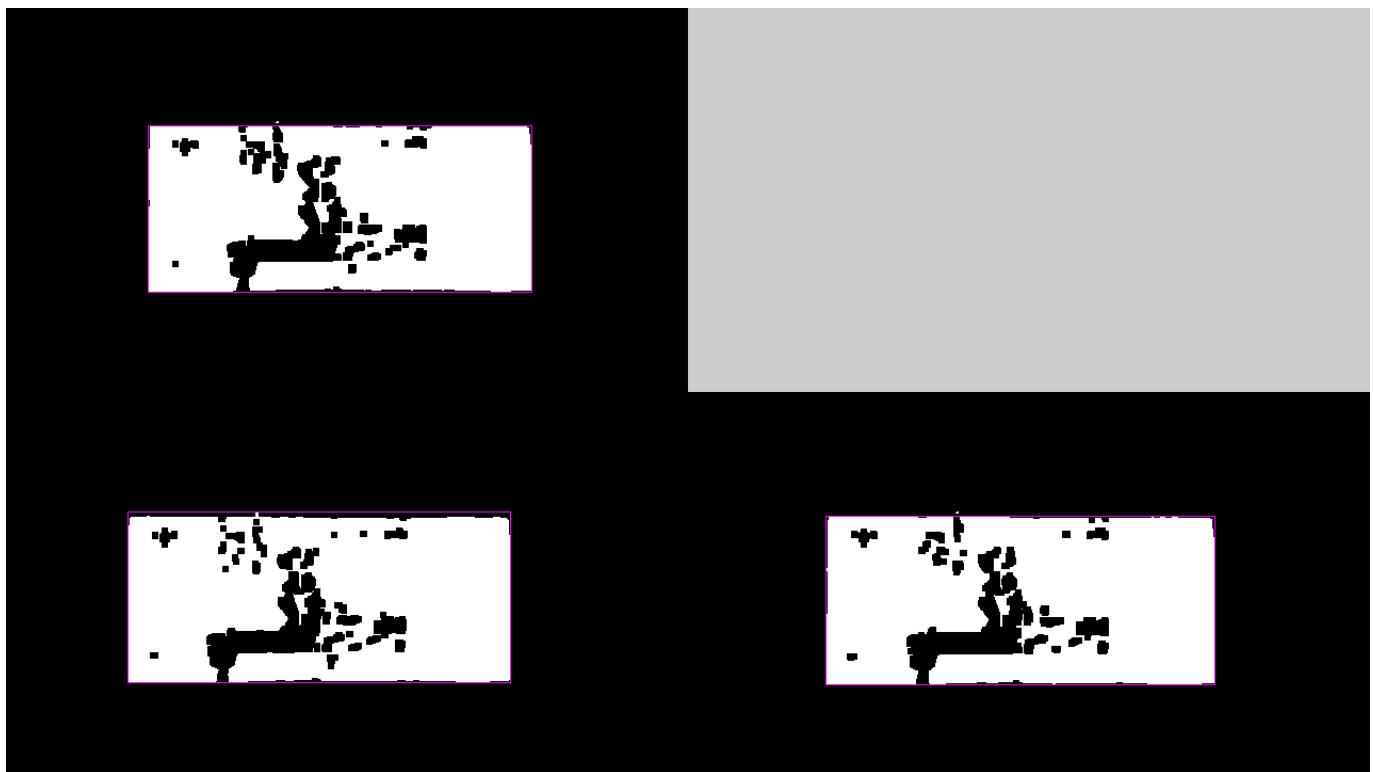


Рис. 2. Выделенные основные компоненты связности 50 рублей купюры на столе

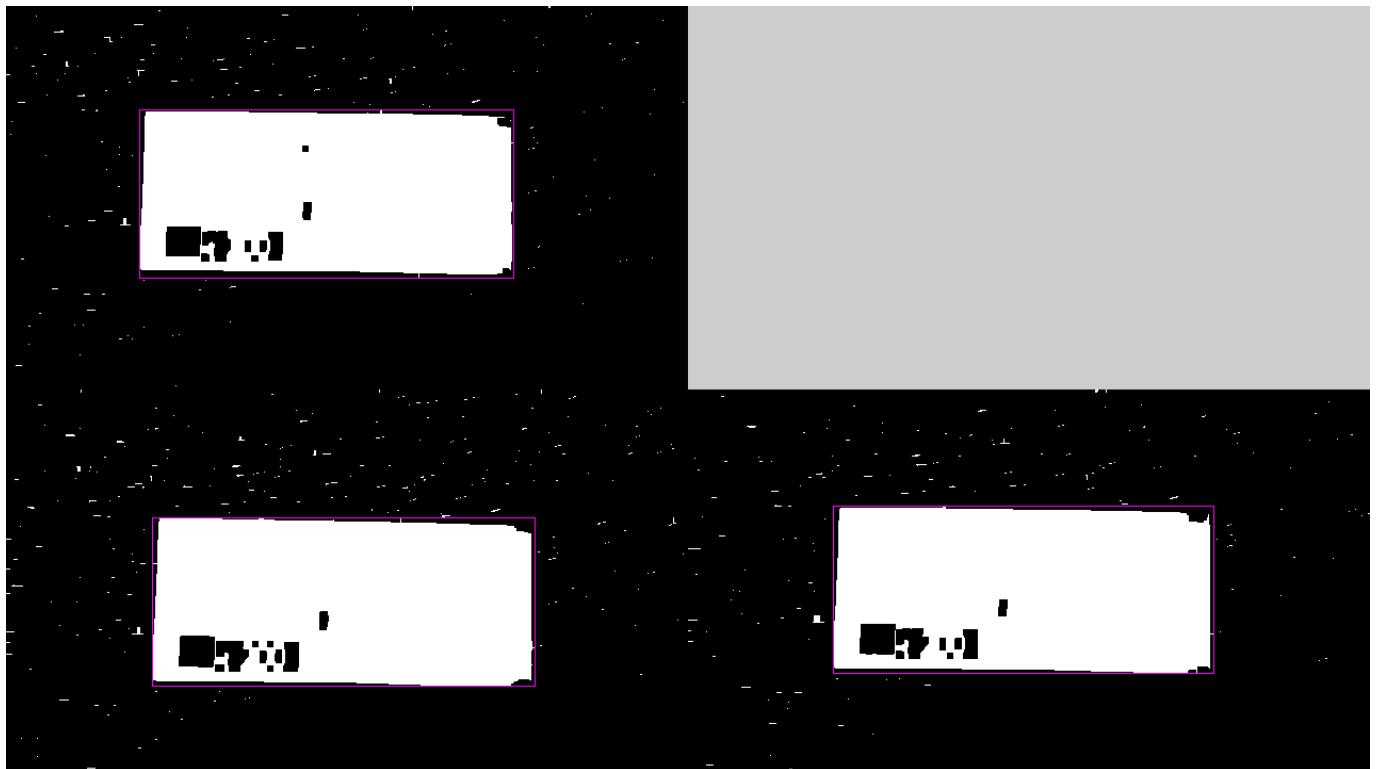


Рис. 3. Выделенные основные компоненты связности 200 рублевой купюры на синем фоне

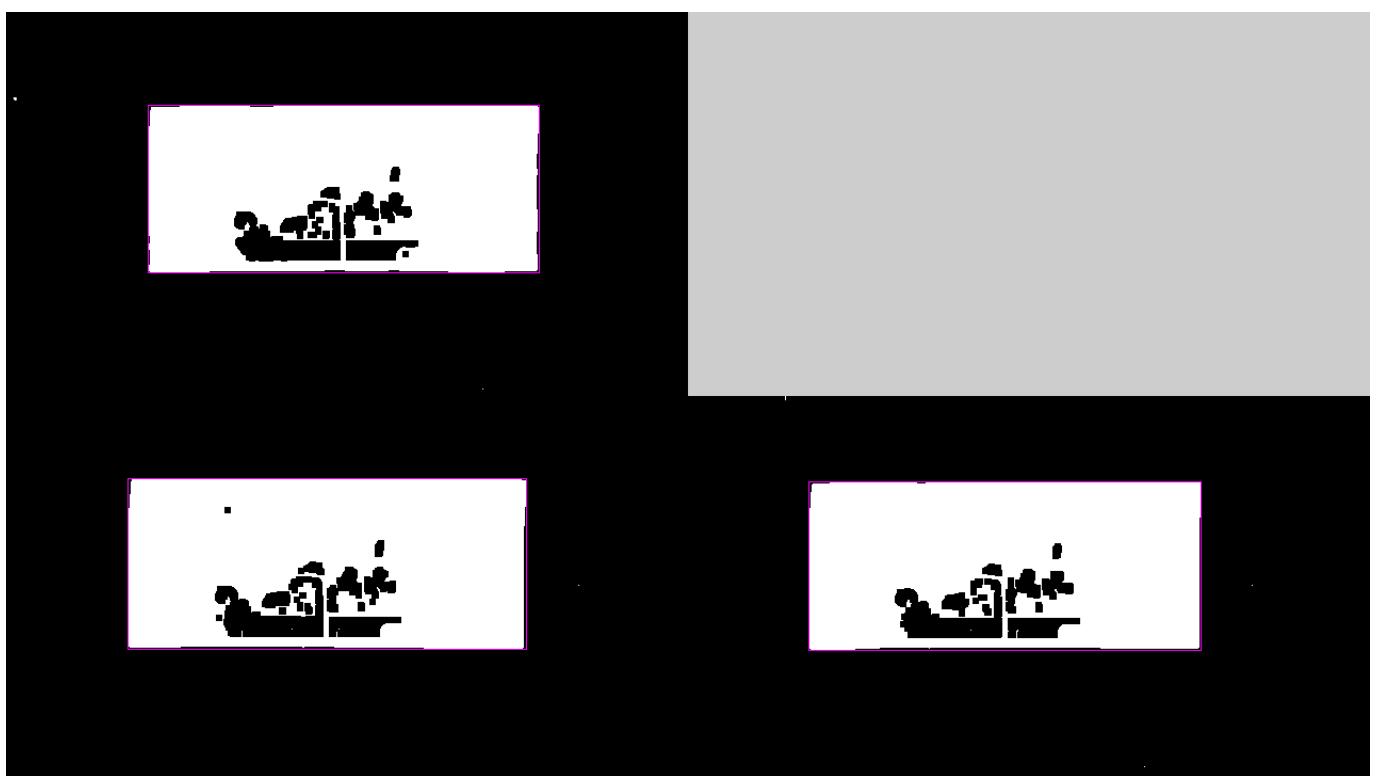


Рис. 4. Выделенные основные компоненты связности 100 рублевой купюры на столе

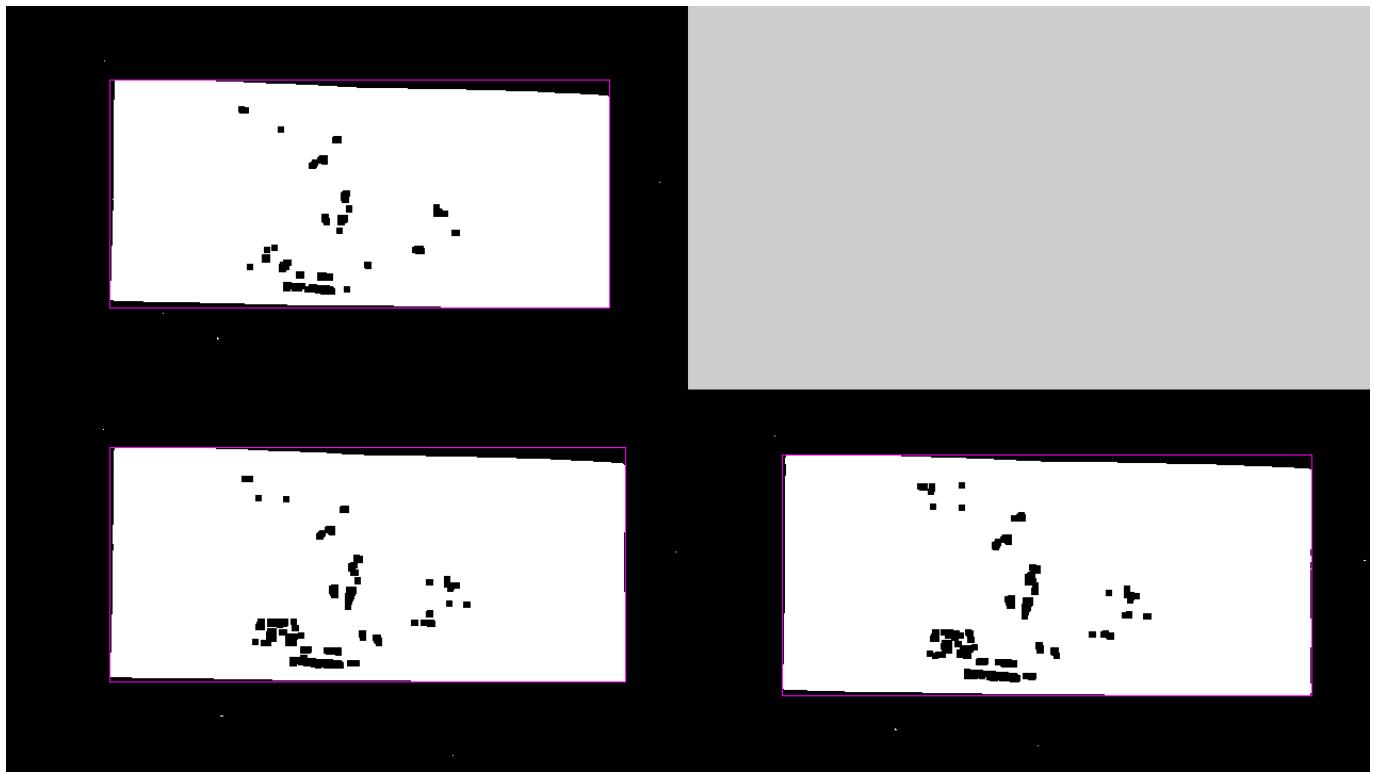


Рис. 5. Выделенные основные компоненты связности 500 рублевой купюры на темном фоне

### Текст программы

```

#include <opencv2/opencv.hpp>
#include <string>
#include <iostream>
#include <vector>

std::vector<cv::Mat> selectFramesFromVideo(std::string path, std::string name);
std::vector<cv::Mat> colorConversation(std::vector<cv::Mat> f, std::string name);
std::vector<cv::Mat> tresholdForEveryone(std::vector<cv::Mat> f, std::string name);
std::vector<cv::Mat> morph(std::vector<cv::Mat> f, std::string name);
std::vector<cv::Mat> connnectedComponents(std::vector<cv::Mat> f,
std::vector<std::vector<std::pair<int, int> >> p, std::string name);
void makeMosaic(std::vector<cv::Mat> src, std::string name, int type);
double sq(const std::vector<std::pair<int, int> & fig);

std::vector<cv::Mat> selectFramesFromVideo(std::string path, std::string name) {

    cv::VideoCapture cp(path);
    std::vector<cv::Mat> res(3);

    int amount_of_frames = cp.get(cv::CAP_PROP_FRAME_COUNT);
    int f1 = (amount_of_frames / 5) * 2,
        f2 = (amount_of_frames / 5) * 3,
        f3 = (amount_of_frames / 5) * 4;
    cv::Mat tmp1, tmp2, tmp3;
    cp.set(cv::CAP_PROP_POS_FRAMES, f1);
    cp >> res[0];
    cv::resize(res[0], res[0], cv::Size(640, 360), cv::INTER_LINEAR);
    cp.set(cv::CAP_PROP_POS_FRAMES, f2);
    cp >> res[1];
    cv::resize(res[1], res[1], cv::Size(640, 360), cv::INTER_LINEAR);
    cp.set(cv::CAP_PROP_POS_FRAMES, f3);
    cp >> res[2];
    cv::resize(res[2], res[2], cv::Size(640, 360), cv::INTER_LINEAR);

    cv::imwrite(name + "_img1.png", res[0]);
    cv::imwrite(name + "_img2.png", res[1]);
    cv::imwrite(name + "_img3.png", res[2]);
    return res;
}

std::vector<cv::Mat> colorConversation(std::vector<cv::Mat> f, std::string name) {
    std::vector<cv::Mat> vec;
    for (int i = 0; i < f.size(); ++i) {
        cv::Mat tmp(640, 360, CV_8UC3);
        cv::cvtColor(f[i], tmp, cv::COLOR_BGR2GRAY);
        cv::imwrite(name + "binary_" + std::to_string(i) + ".png", tmp);
        vec.push_back(tmp);
    }
    return vec;
}

```

```

std::vector<cv::Mat> thresholdForEveryone(std::vector<cv::Mat> f, std::string name) {
    int const max_binary_value = 255;
    std::vector<cv::Mat> vec;
    for (int i = 0; i < f.size(); ++i) {
        cv::Mat tmp(640, 360, CV_8UC3);
        cv::threshold(f[i], tmp, 0, max_binary_value, cv::THRESH_BINARY | cv::THRESH_OTSU);
        //cv::threshold(f[i], tmp1, 0, 255, cv::THRESH_BINARY + cv::THRESH_OTSU);
        //cv::GaussianBlur(f[i], blurred, cv::Size(5, 5), 0);
        //cv::threshold(blurred, tmp2, 0, 255, cv::THRESH_BINARY + cv::THRESH_OTSU);
        //cv::adaptiveThreshold(f[i], tmp, max_binary_value, 0, 2, 25, 10);
        cv::imwrite(name + "threshold_" + std::to_string(i) + ".png", tmp);
        vec.push_back(tmp);
    }
    return vec;
}

std::vector<cv::Mat> morph(std::vector<cv::Mat> f, std::string name) {
    std::vector<cv::Mat> vec;
    for (int i = 0; i < f.size(); ++i) {
        cv::Mat tmp(640, 360, CV_8UC3);
        cv::Mat k = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(6, 6),
cv::Point(-1, -1));
        cv::morphologyEx(f[i], tmp, cv::MORPH_OPEN, k);
        cv::morphologyEx(f[i], tmp, cv::MORPH_CLOSE, k);
        vec.push_back(tmp);
        cv::imwrite( name + "_morph_" + std::to_string(i) + ".png", tmp);
    }
    return vec;
}

std::vector<cv::Mat> connectedComponents(std::vector<cv::Mat> f,
std::vector<std::vector<std::pair<int, int>>> p, std::string name) {
    std::vector<cv::Mat> vec;
    for (int i = 0; i < f.size(); ++i) {
        int max_area = -1, x_max = -1, y_max = -1, w_max = -1, h_max = -1;
        cv::Mat labels;
        cv::Mat stats, centroids;
        int num_labels = cv::connectedComponentsWithStats(f[i], labels, stats,
centroids, 8, 4);

        std::vector<cv::Vec3b> colors(num_labels);

        for (int i = 0; i < num_labels; ++i) {
            colors[i] = cv::Vec3b(255, 255, 255);
        }

        cv::Mat dst = cv::Mat::zeros(f[i].size(), CV_8UC3);
        int w = f[i].cols;

```

```

int h = f[i].rows;

for (int row = 0; row < h; ++row) {
    for (int col = 0; col < w; ++col) {
        int label = labels.at<int>(row, col);
        if (label == 0) continue;
        dst.at<cv::Vec3b>(row, col) = colors[label];
    }
}

for (int j = 1; j < num_labels; ++j) {
    cv::Vec2d pt = centroids.at<cv::Vec2d>(j, 0);
    int x = stats.at<int>(j, cv::CC_STAT_LEFT);
    int y = stats.at<int>(j, cv::CC_STAT_TOP);
    int width = stats.at<int>(j, cv::CC_STAT_WIDTH);
    int height = stats.at<int>(j, cv::CC_STAT_HEIGHT);
    int area = stats.at<int>(j, cv::CC_STAT_AREA);
    if (area >= max_area) {
        max_area = area;
        x_max = x;
        y_max = y;
        w_max = width;
        h_max = height;
    }
    //cv::rectangle(dst, cv::Rect(x, y, width, height), cv::Scalar(255, 0,
255), 1, 8, 0);
}
//std::cout << "(x,y) = (" << x_max << ", " << y_max << ") and (width, height)
= (" << w_max << ", " << h_max << ")\n";
//std::cout << "area: " << max_area << "\n";
std::cout << name << "\n";
std::cout << "accuracy(square_orig / square_conn_comp): " << sq(p[i]) /
max_area << "\n\n";
cv::rectangle(dst, cv::Rect(x_max, y_max, w_max, h_max), cv::Scalar(255, 0,
255), 1, 8, 0);
std::cout << "\n";
vec.push_back(dst);
cv::imwrite(name + "con" + std::to_string(i) + ".png", dst);
}
return vec;
}

void makeMosaic(std::vector<cv::Mat> src, std::string name, int type) {
cv::Mat res_img(720, 1280, type);
src[0].copyTo(res_img(cv::Rect(0, 0, 640, 360)));
src[1].copyTo(res_img(cv::Rect(640, 360, 640, 360)));
src[2].copyTo(res_img(cv::Rect(0, 360, 640, 360)));

cv::imwrite("lab04_mosaic_" + name + ".png", res_img);

//return res_img;
}

```





