ПРИЛОЖЕНИЕ 1

Исходный код

П1.1 Модуль подключения к тепловизору

```
#include <ctime>
#include <iostream>
#include <algorithm>

// Объявление переменных для работы с CALLBACK функциями
bool opt = false, tep = false;
unsigned int tepclock = 0;
unsigned int optframe = 0;
unsigned int tepframe = 0;
CHeatmap CH1;

// CALLBACK Функция для получения максимальной
// и минимальной температуры
// вызывается 1 раз в секунду из API
void CALLBACK cbRadiometryAttachCB(LLONG lAttachHandle,
NET_RADIOMETRY_DATA* pBuf, int nBufLen,
LDWORD dwUser)
{
        static CString strTemperature = "";
        float* pTempForPixels = new
        float[pBuf->stMetaData.nWidth *
        pBuf->stMetaData.nHeight * sizeof(float)];
        if (VSIF_RadiometryDataParse(pBuf,
        NULL, pTempForPixels))
        {
                int x1, y1, x2, y2;
                float loTemp = 1000.0, hiTemp = -1000.0;
                for (int i = 0;
                i < pBuf->stMetaData.nHeight; i++)
                {
                        for (int j = 0;
                        j < pBuf->stMetaData.nWidth; j++)
                        {
                                float temp = pTempForPixels[i *
                                pBuf->stMetaData.nWidth + j];

                                if (temp < loTemp)
                                {
                                        loTemp = temp;
                                        x1 = j;
                                        y1 = i;
                                }

                                if (temp > hiTemp)
                                {
                                        hiTemp = temp;
                                        x2 = j;
                                        y2 = i;
                                }
                        }
                }
```

```
                // запись информации в файл
                FILE* fp;
                fp = fopen("saves\\temps.bin", "ab");
                fwrite(&(pBuf->stMetaData.stTime.dwSecond),
                sizeof(DWORD), 1, fp);
                fwrite(&loTemp, sizeof(float), 1, fp);
                fwrite(&hiTemp, sizeof(float), 1, fp);
                fclose(fp);
        }

        delete[] pTempForPixels;

}

// регистрация функции для вызова
void CHeatmap::OnStartfetch()
{
        NET_IN_RADIOMETRY_FETCH stInFetch =
        { sizeof(stInFetch), m_nHeatChannel };
        NET_OUT_RADIOMETRY_FETCH stOutFetch =
        { sizeof(stOutFetch) };
        VSIF_RadiometryFetch(m_lLoginID,
        &stInFetch, &stOutFetch, 1000);
}

// функция вызываемая при нажатии на кнопку начала
// записи
void CHeatmap::OnBnClickedStartfetch2()
{
        if (!m_isFetch)
        {
                if (VSIF_RadiometryStartFetch(m_lLoginID,
                m_nHeatChannel))
                {
                        m_isFetch = true;
                }
        }
        else
        {
                VSIF_RadiometryStopFetch(m_lLoginID,
                m_nHeatChannel);
                m_isFetch = false;
        }
}
// регистраций функции для получения максимальной
// и минимальной температуры
void CHeatmap::OnAttach()
{
        NET_IN_RADIOMETRY_ATTACH stIn = { sizeof(stIn) };
        stIn.nChannel = m_nHeatChannel;
        stIn.dwUser = (LDWORD)this;
        stIn.cbNotify = cbRadiometryAttachCB;
        NET_OUT_RADIOMETRY_ATTACH stOut = { sizeof(stOut) };
        m_lAttachhandle = VSIF_RadiometryAttach(m_lLoginID,
        &stIn, &stOut, 1000);
}
```

```
// Остановка вызова функции для получения максимальной и
// минимальной температуры
void CHeatmap::OnStop()
{
        VSIF_RadiometryDetach(m_lAttachhandle);
}

// Функция авторизации
void CRealPlayAndPTZControlDlg::OnBTLogin()
{
        //получение интерфейса ввода
        BOOL bValid = UpdateData(TRUE);
        if (bValid)
        {
                //переменная с возможным кодом ошибки.
                int err = 0;
                char* pchDVRIP;
                CString strDvrIP = GetDvrIP();
                pchDVRIP = (LPSTR)(LPCSTR)"192.168.25.199";
                WORD wDVRPort = (WORD)m_DvrPort;
                char* pchUserName = (LPSTR)(LPCSTR)"admin";
                char* pchPassword = (LPSTR)(LPCSTR)"susu";
                NET_DEVICEINFO_Ex deviceInfo = { 0 };
                //вызов аторизации из API
                LLONG lRet = VSIF_LoginEx2(pchDVRIP,
                wDVRPort, pchUserName,
                pchPassword, EM_LOGIN_SPEC_CAP_TCP, NULL,
                &deviceInfo, &err);
                // если регистрация успешна получение информации о
                // состоянии устройства
                if (0 != lRet)
                {
                        m_LoginID = lRet;
                        GetDlgItem(IDC_BT_Login)->EnableWindow(FALSE);
                        GetDlgItem(IDC_BT_Leave)->EnableWindow(TRUE);
                        GetDlgItem(IDC_BUTTON_Play)->EnableWindow(TRUE);
                        int nRetLen = 0;
                        NET_DEV_CHN_COUNT_INFO stuChn =
                        { sizeof(NET_DEV_CHN_COUNT_INFO) };
                        stuChn.stuVideoIn.dwSize =
                        sizeof(stuChn.stuVideoIn);
                        stuChn.stuVideoOut.dwSize =
                        sizeof(stuChn.stuVideoOut);
                        BOOL bRet = VSIF_QueryDevState(lRet,
                        VS_DEVSTATE_DEV_CHN_COUNT, (char*)&stuChn,
                        stuChn.dwSize, &nRetLen);
                        if (!bRet)
                        {
                                DWORD dwError =
                                VSIF_GetLastError() & 0x7fffffff;
                        }
                        m_nChannelCount = __max(deviceInfo.nChanNum,
                        stuChn.stuVideoIn.nMaxTotal);
                        int nIndex = 0;
                        m_comboChannel.ResetContent();
```

```
                        for (int i = 0; i < m_nChannelCount; i++)
                        {
                                CString str;
                                str.Format("%d", i + 1);
                                nIndex = m_comboChannel.AddString(str);
                                m_comboChannel.SetItemData(nIndex, i);
                        }
                        if (0 < m_comboChannel.GetCount())
                        {
                                nIndex = m_comboChannel.
                                AddString(
                                ConvertString("Multi_Preview"));
                                m_comboChannel.SetItemData(nIndex, -1);
                                m_comboChannel.SetCurSel(0);
                        }
                        CH1.m_lLoginID = m_LoginID;
                        CH1.OnAttach();
                        CH1.OnBnClickedStartfetch2();
                }
                else
                {
                        //Сообщение об ошибке
                        ShowLoginErrorReason(err);
                }
        }
        SetWindowText(
        ConvertString("RealPlayAndPTZControl"));

}

// CALLBACK функция вызываемая при получении данных
void CALLBACK DecodeCallback(LONG nPort,
FRAME_DECODE_INFO* pFrameDecodeInfo,
FRAME_INFO_EX* pFrameInfo, void* pUser)
{
        if (pUser == 0)
        {
                return;
        }
        CRealPlayAndPTZControlDlg* dlg =
        (CRealPlayAndPTZControlDlg*)pUser;
        dlg->ReceiveDecodeData(nPort,
        pFrameDecodeInfo, pFrameInfo);
}

//обработка полученных данных
void CRealPlayAndPTZControlDlg::ReceiveDecodeData(
LONG nPort,
FRAME_DECODE_INFO* pFrameDecodeInfo,
FRAME_INFO_EX* pFrameInfo)
{
        // проверка на ошибки
        if (0 == nPort ||
        pFrameDecodeInfo == NULL || pFrameInfo == NULL)
                return;
```

```
        // обработка оптических кадров
        if (pFrameInfo->nFrameType == 0)
        {
                FILE* fp;
                if (pFrameInfo->nHeight == 1080)
                {
                        fp = fopen("saves\\1920x1080.yuv", "ab");
                        opt = true;
                        if (optframe != tepframe)
                        {
                                fclose(fp);
                                return;
                        }
                        if (tep && opt)
                        {
                                optframe += 1;
                        }

                }
                // обработка тепловых кадров
                else
                {
                        fp = fopen("saves\\1280x1024.yuv", "ab");
                        tep = true;
                        if (optframe != tepframe + 1)
                        {
                                fclose(fp);
                                return;
                        }
                        if (tep && opt)
                        {
                                tepframe += 1;
                        }

                }
                // запись полученого кадра и контроль за динной кадра
                if (tep && opt)
                {
                        tepclock = clock();
                        fwrite(pFrameDecodeInfo->pVideoData[0],
                        1, pFrameDecodeInfo->nStride[0] *
                        pFrameDecodeInfo->nHeight[0], fp);
                        fwrite(pFrameDecodeInfo->pVideoData[1],
                        1, pFrameDecodeInfo->nStride[1] *
                        pFrameDecodeInfo->nHeight[1], fp);
                        fwrite(pFrameDecodeInfo->pVideoData[2],
                        1, pFrameDecodeInfo->nStride[2] *
                        pFrameDecodeInfo->nHeight[2], fp);
                        Sleep(max(0, 50 - (clock() - tepclock)));
                }
                fclose(fp);
        }
}
```

П1.2 Модуль сегментации факела выбросов

```
#define _CRT_SECURE_NO_WARNINGS
#include <set>
#include <chrono>
#include <regex>
#include <iostream>
#include <opencv2/core.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/features2d.hpp>

using namespace cv;
using namespace std;

//объявление констант
#define RGB 256
#define CHANNELS 3
#define W 1920
#define H 1080 * 3 / 2

//объявление буфферов
unsigned char buf[H][W];
bool loadedFlag = false;
unsigned char colorTransform[RGB][RGB][RGB];
/*********************************************************/
//функция загрузки преобразователя цветовой карты
void loadMap(string fileName)
{
        //загрузка файлов
        FILE* input = NULL;
        input = fopen(fileName.c_str(), "rb");
        cout << "Map loading started_____________\n";
        //если файла не существует подготовка преобразователя
        if (input == NULL)
        {
                //подгоовка обучающей выборки для knn
                vector<unsigned char> gray(RGB);
                for (int i = 0; i < RGB; i++) gray[i] = i;
                Mat grayValues(RGB, 1, CV_8U, gray.data());
                Mat colorValues;
                applyColorMap(grayValues, colorValues,
                COLORMAP_JET);
                colorValues.convertTo(colorValues, CV_32FC3);
                vector<Mat> colorValuesVec;
                for (int i = 0; i < colorValues.rows; ++i) {
                        colorValuesVec.push_back({ 1,
                                CHANNELS, CV_32F,
                                colorValues.at<Vec3f>(i, 0).val });
                }

                //создание и обучение модели
                FlannBasedMatcher fm = FlannBasedMatcher();
                fm.add(colorValuesVec); fm.train();
                cout << "model trained_____________\n";
```

```
                // классификация RGB
                for (int i = 0; i < RGB; i++)
                {
                        for (int j = 0; j < RGB; j++)
                        {
                                for (int k = 0; k < RGB; k++)
                                {
                                        unicData.push_back((float)i);
                                        unicData.push_back((float)j);
                                        unicData.push_back((float)k);
                                }
                        }
                }
                Mat unic(unicData.size() / CHANNELS, CHANNELS,
                CV_32F, unicData.data());
                vector<DMatch> matches; fm.match(unic, matches);
                for (int i = 0; i < unicData.size() / CHANNELS; i++)
                {
                        colorTransform[(int)unicData[i * CHANNELS]]
                        [(int)unicData[i * CHANNELS + 1]]
                        [(int)unicData[i * CHANNELS + 2]] =
                        matches[i].imgIdx;
                }

                //сохранение классификатора
                FILE* output;
                output = fopen(fileName.c_str(), "wb");
                fwrite(colorTransform, sizeof(unsigned char),
                RGB * RGB * RGB, output);
                fclose(output);
        }
        //если файл есть просто загружаем
        else
        {
                fread(colorTransform, sizeof(unsigned char),
                RGB * RGB * RGB, input);
                fclose(input);
        }
        cout << "Map loading finished__________\n";
}

//вспомогательная функция для
//получения размеров из названия файла
pair<int, int> get_size(string& s)
{
        string s1 = "", s2 = "";
        int i = 0;
        for (i = 0; i < s.size() &&
        (s[i] >= '0' && s[i] <= '9'); ++i)
                s1 += s[i];
        for (i++; i < s.size() &&
        (s[i] >= '0' && s[i] <= '9'); ++i)
                s2 += s[i];
        return { stoi(s1), stoi(s2) };
}
```

```
//функция сегментации WaterShed
Mat segment_test(Mat& grayImage)
{
        //получение маркеров
        Mat image, res;
        cvtColor(grayImage, image, COLOR_BGR2GRAY);
        threshold(image, res, 0, 255,
        THRESH_BINARY_INV + THRESH_OTSU);
        cv::Mat markers = cv::Mat::zeros(image.size(),
        CV_32SC1);
        unsigned char maxzn = -1;
        pair<unsigned char, unsigned char> ind;
        for (int i = 0; i < image.rows; i++) {
                for (int j = 0; j < image.cols; j++) {
                        if (image.at<unsigned char>(i, j) >= maxzn)
                        {
                                maxzn = image.at<unsigned char>(i, j);
                                ind = { i, j };
                        }
                        markers.at<int>(i, j) = 0;
                }
        }
        cv::Size ksize(7, 7);
        double sigmaX = 2.0;
        double sigmaY = 2.0;

        // применяем фильтр Гаусса на изображении
        cv::GaussianBlur(image, image, ksize,
        sigmaX, sigmaY);
        erode(res, res, Mat(9,9,CV_8U),
        Point(-1, -1), 2, 1, 1);
        for (int i = 0; i < image.rows; i++) {
                for (int j = 0; j < image.cols; j++) {
                        if (image.at<unsigned char>(i, j)
                        >= maxzn * 0.9)
                        {
                                markers.at<int>(i, j) = 255;
                        }
                        else if (res.at<unsigned char>(i, j)
                        == 255)
                        {
                                markers.at<int>(i, j) = 1;
                        }
                }
        }

        //применение алгоритма сегментации
        //водоразделами
        watershed(grayImage, markers);
        cvtColor(res, res, COLOR_GRAY2BGR);
        cvtColor(image, image, COLOR_GRAY2BGR);
        markers.convertTo(markers, CV_8UC1);
        return markers;
}
```

```
//простейший вариант сегментации
Mat segment_basic(Mat& grayImage)
{
        uchar pixelCoords;
        Mat segImage(grayImage);
        for (int i = 0; i < grayImage.rows; i++) {
                for (int j = 0; j < grayImage.cols; j++) {
                        pixelCoords =
                        grayImage.at<uchar>(i, j);
                        if (pixelCoords >= 145)
                        segImage.at<uchar>(i, j) = 1;
                        else
                        segImage.at<uchar>(i, j) = 0;

                }
        }
        return segImage;
}

//загрузка оптического видео
//преобразование к RGB и сохранение
void videoload_opt(string fileNamein, string& fileNameOut)
{
        fileNameOut = fileNamein + "tmp.avi";
        int frameNum = 0;
        FILE* yuvFile = fopen(fileNamein.c_str(), "rb");

        pair<int, int> sizes = get_size(fileNamein);

        int w = sizes.first;
        int h = sizes.second;

        int frame_width = static_cast<int>(w);
        int frame_height = static_cast<int>(h);
        Size frame_size(frame_width, frame_height);
        int fps = 20;
        VideoWriter output_real(fileNameOut,
        VideoWriter::fourcc('M', 'J', 'P', 'G'),
        fps, frame_size);
        int i = 0;
        //цикл по кадрам
        while (!feof(yuvFile)) {
                i++;
                cout << i << endl;
                size_t readData = fread(buf,
                sizeof(unsigned char),
                h * w * 3 / 2, yuvFile);
                Mat trueImage(h * 3 / 2,
                w, CV_8U, buf),
                segImage;
                cvtColor(trueImage,
                trueImage, COLOR_YUV2BGR_I420);
                output_real.write(trueImage);
        }
        output_real.release();
}
```

```
//загрузка теплового видео
//конвертация цветовой карты
//преобразование к RGB и сохранение
void videoload_tep(string fileNamein, string& fileNameOut)
{
        fileNameOut = fileNamein + "tmp";
        int frameNum = 0;
        FILE* yuvFile = fopen(fileNamein.c_str(), "rb");

        pair<int, int> sizes = get_size(fileNamein);

        int w = sizes.first;
        int h = sizes.second;

        int frame_width = static_cast<int>(w);
        int frame_height = static_cast<int>(h);
        Size frame_size(frame_width, frame_height);
        int fps = 20;

        VideoWriter output_gray("output_gray.avi",
        VideoWriter::fourcc('M', 'J', 'P', 'G'),
        fps, frame_size);

        int i = 0;
        //цикл по кадрам
        while (!feof(yuvFile)) {
                i++;
                cout << i << endl;
                size_t readData = fread(buf, sizeof(unsigned char),
                h * w * 3 / 2, yuvFile);

                Mat grayImage, trueImage(h * 3 / 2, w, CV_8U, buf);

                cvtColor(trueImage, trueImage, COLOR_YUV2BGR_I420);
                cvtColor(trueImage, grayImage, COLOR_BGR2GRAY);
                rectangle(trueImage, { 725, 50 }, { 1210, 85 },
                { 127, 0, 0 }, -1);

                //загрузка преобразователя
                if (!loadedFlag)
                {
                        loadMap("convertMap.bin");
                        loadedFlag = true;
                }

                Vec3b pixelCoords;

                for (int i = 0; i < trueImage.rows; i++) {
                        for (int j = 0; j < trueImage.cols; j++) {
                                pixelCoords = trueImage.at<Vec3b>(i, j);
                                grayImage.at<uchar>(i, j) =
                                colorTransform[pixelCoords.val[0]]
                                [pixelCoords.val[1]]
                                [pixelCoords.val[2]];
```

```
                }
            }
            cvtColor(grayImage, grayImage, COLOR_GRAY2BGR);
            output_gray.write(grayImage);
        }
        output_gray.release();
}


//алгоритм RanSaC для востановления
//прямоугольника
vector<float> RANSAC(vector<int>& cord,
vector<DMatch>& good_matches,
vector<KeyPoint>& k1,
vector<KeyPoint>& k2,
int tries)
{
        vector<float> bestcord(0);
        float bestdist;
        while (tries--)
        {
            int ind1 = rand() % good_matches.size();
            int ind2 = rand() % good_matches.size();
            while (ind2 == ind1)
            ind2 = rand() % good_matches.size();

            //поиск внутреннего прямоугольника на
            //первой картинке
            vector<float> coord1 =
            { min(k1[good_matches[ind1].queryIdx].pt.x,
                    k1[good_matches[ind2].queryIdx].pt.x),
                    max(k1[good_matches[ind1].queryIdx].pt.x,
                    k1[good_matches[ind2].queryIdx].pt.x),
                    min(k1[good_matches[ind1].queryIdx].pt.y,
                    k1[good_matches[ind2].queryIdx].pt.y),
                    max(k1[good_matches[ind1].queryIdx].pt.y,
                    k1[good_matches[ind2].queryIdx].pt.y) };
            //поиск внутреннего прямоугольника на
            //второй картинке
            vector<float> coord2 =
            { min(k2[good_matches[ind1].trainIdx].pt.x,
                    k2[good_matches[ind2].trainIdx].pt.x),
                    max(k2[good_matches[ind1].trainIdx].pt.x,
                    k2[good_matches[ind2].trainIdx].pt.x),
                    min(k2[good_matches[ind1].trainIdx].pt.y,
                    k2[good_matches[ind2].trainIdx].pt.y),
                    max(k2[good_matches[ind1].trainIdx].pt.y,
                    k2[good_matches[ind2].trainIdx].pt.y) };
            //разница между внутреним и внешним
            //прямоугольником на 1 картинке
            vector<float> delta1 =
            { abs(coord1[0] - cord[0]),
                    abs(coord1[1] - cord[1]),
                    abs(coord1[2] - cord[2]),
                    abs(coord1[3] - cord[3]) };
```

```
//размеры внутреннего прямоугольника
//в 1 и во 2 картинках
vector<float> size1 =
{ abs(coord1[0] - coord1[1]),
        abs(coord1[2] - coord1[3]) };
vector<float> size2 =
{ abs(coord2[0] - coord2[1]),
        abs(coord2[2] - coord2[3]) };
// разница между внутреним и внешним
//прямоугольником на 2 картинке
vector<float> delta2 =
{ delta1[0] * size2[0] / size1[0],
        delta1[1] * size2[0] / size1[0],
        delta1[2] * size2[1] / size1[1],
        delta1[3] * size2[1] / size1[1] };
// внешний прямоугольник на 2 картинке
vector<float> newcord =
{ coord2[0] - delta2[0],
        coord2[1] + delta2[1],
        coord2[2] - delta2[2],
        coord2[3] + delta2[3] };
// размеры внешнего прямоугольника
//на 2 кортинке
vector<float> sizenew =
{ newcord[1] - newcord[0],
        newcord[3] - newcord[2] };

float dist = 0;
for (int i = 0; i < good_matches.size(); i++)
{
        // относительные координаты
        //точки на 1 изображении
        vector<float> tcord1 =
        { k1[good_matches[i].queryIdx].pt.x - cord[0],
                k1[good_matches[i].queryIdx].pt.y - cord[2] };
        // относительные координаты
        //точки на 2 изображении
        vector<float> tcord2 =
        { tcord1[0] * size2[0] / size1[0],
                tcord1[1] * size2[1] / size1[1] };
        // абсолютные преобразованные
        //координаты на 2 картинке
        vector<float> absconvcord =
        { newcord[0] + tcord2[0],
                newcord[2] + tcord2[1] };
        // абсолютные координаты на
        //2 картинке
        vector<float> abscord =
        { k2[good_matches[i].trainIdx].pt.x,
                k2[good_matches[i].trainIdx].pt.y };
        // расстояние
        float tdist =
        sqrt(((absconvcord[0] - abscord[0]) *
        (absconvcord[0] - abscord[0]) +
        (absconvcord[1] - abscord[1]) *
        (absconvcord[1] - abscord[1])));
```

```
                        dist += tdist * good_matches[i].distance;
                }
                //обновление лучшего результата
                if (bestcord.size() == 0 || dist < bestdist)
                {
                        bestcord = newcord;
                        bestdist = dist;
                }
        }
        return bestcord;
}

//получение коэффициента точности DICE
double diceCoefficient(Mat& mask1, Mat& mask2)
{
        double intersectionArea = 0,
        mask1Area = 0,
        mask2Area = 0;

        //подсчет площадей
        for (int i = 0; i < mask1.rows; i += 1) {
                for (int j = 0; j < mask1.cols; j += 1) {
                        uchar& msk1 = mask1.at<uchar>(i, j);
                        uchar& msk2 = mask2.at<uchar>(i, j);
                        if (msk1 == 255 && msk2 == 255)
                        intersectionArea++;
                        if (msk1 == 255) mask1Area++;
                        if (msk2 == 255) mask2Area++;
                }
        }

        double dice =
        (2 * intersectionArea) /
        (mask1Area + mask2Area);
        return dice;
}

//расчет DICE для тестовой выборки
void podschet()
{
        double koef_gl = 0;
        for (int i = 0; i < 100; i++)
        {
                Mat msk =
                imread("mask_razmet/image (" +
                to_string(i) + ").jpg", 0);
                Mat pred =
                imread("mask/" +
                to_string(i + 1) + ".jpg", 0);
                double koef = diceCoefficient(msk, pred);
                cout << koef << ", ";
                koef_gl += koef;
        }
        koef_gl /= 100;
        cout << endl << "DICE = " << koef_gl << endl;
}
```

```
//получение разницы масок
void difMask(string name)
{
        Mat msk_r = imread("examples/mask_razmet/" +
        name + ".png", 0);
        Mat msk = imread("examples/mask/" +
        name + ".png", 0);
        Mat msk_dif(min(msk.rows, msk_r.rows),
        min(msk.cols, msk_r.cols), CV_8U, buf);

        for (int i = 0; i < min(msk.rows, msk_r.rows);
        ++i)
        {
                for (int j = 0; j < min(msk.cols, msk_r.cols);
                ++j)
                {
                        //проверка исключающим или
                        if ((msk_r.at<unsigned char>(i, j) >= 128) ^
                        (msk.at<unsigned char>(i, j) >= 128))
                        {
                                msk_dif.at<unsigned char>(i, j) = 255;
                        }
                        else
                        {
                                msk_dif.at<unsigned char>(i, j) = 0;
                        }
                }
        }
        cvtColor(msk_dif, msk_dif, COLOR_GRAY2BGR);
        imwrite("examples/mask_dif/" +
        name + ".png", msk_dif);
}

//алгоритм детекции трубы
vector<float> trubaDetector(Mat img1, Mat img2)
{
        //подготовка детектора
        Ptr<SiftFeatureDetector> detector =
        SiftFeatureDetector::create(0, 7, 0.04, 10.0, 0.8);

        //дескрипторы для образца и изображения на котором
        //детектируется труба
        std::vector<KeyPoint> keypoints1;
        Mat descriptors1;
        detector->detectAndCompute(img1,
        noArray(),
        keypoints1,
        descriptors1);
        std::vector<KeyPoint>keypoints2;
        Mat descriptors2;
        detector->detectAndCompute(img2,
        noArray(),
        keypoints2,
        descriptors2);
```

```
        //подготовка классификатора для сопоставления
        Ptr<DescriptorMatcher> matcher =
        DescriptorMatcher::
        create(DescriptorMatcher::FLANNBASED);
        std::vector<std::vector<DMatch>>
        knn_matches;
        matcher->knnMatch(descriptors1,
        descriptors2,
        knn_matches,
        2);

        //отсеивание менее значимых сопоставлений
        vector<int> cords =
        { 0,
                img1.cols - 1,
                0,
                img1.rows - 1 };
        const float ratio_thresh = 0.7f;
        std::vector<DMatch> good_matches;
        for (size_t i = 0; i < knn_matches.size(); i++)
        {
                if (knn_matches[i][0].distance <
                ratio_thresh * knn_matches[i][1].distance)
                {
                        auto p1 =
                        keypoints1[knn_matches[i][0].queryIdx].pt;
                        auto p2 =
                        keypoints2[knn_matches[i][0].trainIdx].pt;

                        if (p1.x >= cords[0] && p1.x <= cords[1] &&
                        p1.y >= cords[2] && p1.y <= cords[3]) {
                                good_matches.push_back(knn_matches[i][0]);
                        }

                        // RunSUCK
                        // Match rectangle by points
                }
        }
        cout << good_matches.size() << " ";

        //использование алгоритма RanSaC
        vector<float> rect2 = RANSAC(cords,
        good_matches,
        keypoints1,
        keypoints2,
        500);
}


int main()
{
        //таймер
        auto start = chrono::high_resolution_clock::now();

        // загрузка видео
        string opt, tep;
```

```
videoload_opt("1920x1080.avi", opt);
videoload_tep("1280x1024.avi", tep);
VideoCapture opt_cap(opt);
VideoCapture tep_cap(tep);
VideoWriter res("res.avi",
VideoWriter::fourcc('m', 'p', '4', 'v'),
opt_cap.get(CAP_PROP_FPS),
Size(737 * 2, 588));

Mat opt_img, tep_img;
int frameNum = 0;

//цикл по кадрам
while (true) {
        if (frameNum >= 300) break;
        frameNum++;
        cout << frameNum << endl;
        opt_cap >> opt_img;
        tep_cap >> tep_img;
        if (opt_img.empty() ||
        tep_img.empty()) break;

        resize(tep_img,
        tep_img,
        Size(737, 588),
        INTER_LINEAR);

        Vec3b pixel_tep;
        Mat test(588, 737 * 2, CV_8UC3, buf);
        Mat res_im(588, 737, CV_8UC3, buf);

        //сегментация
        Mat segment = segment_test(tep_img);
        rectangle(segment,
        { 442, 459 },
        { 527, 587 },
        { 0, 0, 0 }, -1);

        //подготовка итогового видео
        for (int i = 0; i < tep_img.rows; ++i)
        {
                for (int j = 0; j < tep_img.cols; ++j)
                {
                        pixel_tep = tep_img.at<Vec3b>(i, j);
                        Vec3b pixel_opt =
                        opt_img.at<Vec3b>(i + 252, j + 558);
                        Vec3b& test_1 =
                        test.at<Vec3b>(i, j);
                        Vec3b& test_2 =
                        test.at<Vec3b>(i, j + 737);

                        if (segment.at<unsigned char>(i, j) == 255)
                        {
                                test_1[0] =
                                (unsigned char)(pixel_opt.val[0] *
                                0.6);
```

```
                                        test_1[1] =
                                        (unsigned char)(pixel_opt.val[1] *
                                        0.6);
                                        test_1[2] =
                                        (unsigned char)(pixel_opt.val[2] *
                                        0.6 + 255 * 0.4);

                                        test_2[0] =
                                        (unsigned char)(pixel_tep.val[0] *
                                        0.6);
                                        test_2[1] =
                                        (unsigned char)(pixel_tep.val[1] *
                                        0.6);
                                        test_2[2] =
                                        (unsigned char)(pixel_tep.val[2] *
                                        0.6 + 255 * 0.4);
                                }
                                else
                                {
                                        test_1[0] =
                                        (unsigned char)(pixel_opt.val[0]);
                                        test_1[1] =
                                        (unsigned char)(pixel_opt.val[1]);
                                        test_1[2] =
                                        (unsigned char)(pixel_opt.val[2]);

                                        test_2[0] =
                                        (unsigned char)(pixel_tep.val[0]);
                                        test_2[1] =
                                        (unsigned char)(pixel_tep.val[1]);
                                        test_2[2] =
                                        (unsigned char)(pixel_tep.val[2]);
                                }
                        }
                }
                putText(test,
                to_string(frameNum),
                Point(10, 450),
                FONT_HERSHEY_SIMPLEX,
                3,
                CV_RGB(0, 255, 0),
                2);

                res.write(test);
        }

        //освобождение данных
        opt_cap.release();
        tep_cap.release();
        res.release();

        printf("\n\n%lf\n", clock() * 1e-3);
}
```