1 СОВРЕМЕННЫЕ МЕТОДЫ ОПРЕДЕЛЕНИЯ ГЕОМЕТРИЧЕСКИХ И ФИЗИЧЕСКИХ ПАРАМЕТРОВ ФАКЕЛА ВЫБРОСОВ

1.1 Методы контроля выбросов загрязняющих веществ в атмосферу

1.1.1 Инструментальный метод контроля выбросов

Инструментальный метод – метод контроля, который осуществляется с использованием газоаналитических средств, прошедших проверку и внесенных в Государственный реестр средств измерений [4]. Этот метод применим для работы с организованными источниками и обычно включает использование газоанализаторов. Контроль проводится в специальной санитарно-защитной зоне, которая представляет собой территорию вокруг промышленных предприятий и других объектов, выбрасывающих загрязняющие вещества в атмосферу, воду и почву. Эта зона служит защитным барьером между указанными объектами и жилой зоной [6].

Газоанализаторы – это измерительные приборы, которые обычно применяют для анализа состава и свойств газовых смесей в различных химических процессах. Газоанализаторы можно разделить на несколько основных групп, в зависимости от целей и задач [2]:

1) газоанализаторы сжигания для отладки и контроля котлов, печей и сжигающих топливо устройств;

2) газоанализаторы определяющие параметры и контролирующие воздух рабочей зоны (безопасность);

3) приборы по контролю выхлопных газов от разных двигателей внутреннего сгорания;

4) газоанализаторы для контроля выбросов в воздух и разнообразных технических процессов.

Газоанализаторы делятся по своим особенностям и исполнению. Деление происходит на типы:

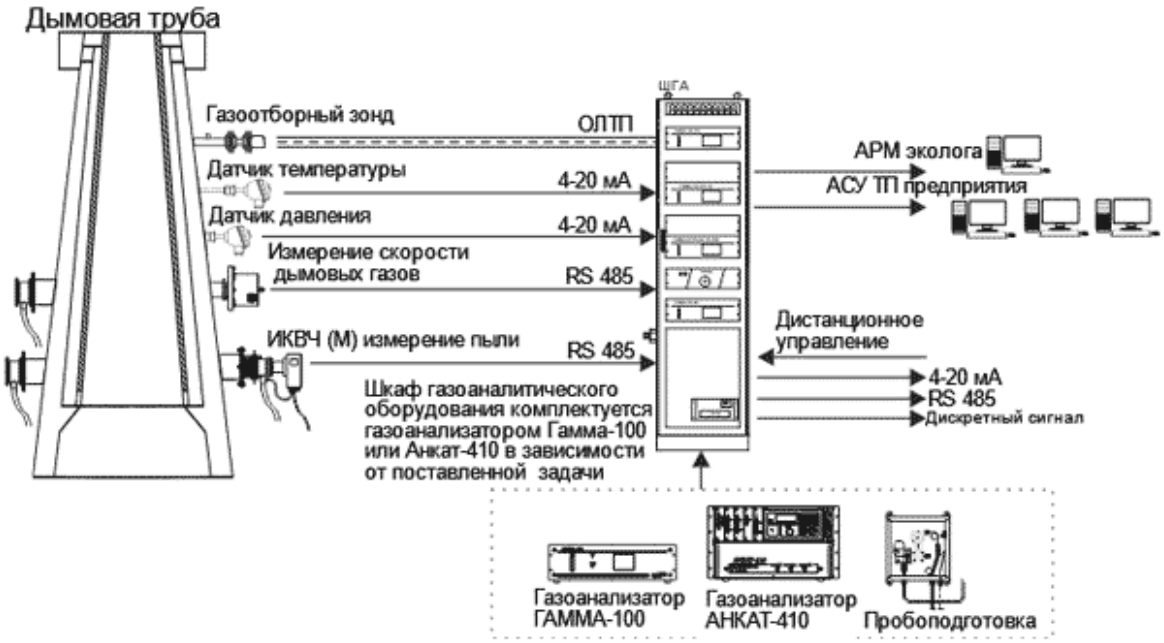

Рисунок 1.1 – Типовая структура газоаналитической системы

себя разные газоанализаторы, измерители и анализаторы, предназначеные для измерения состава газовой пробы. Это включает в себя измерение массовой концентрации различных газов, пыли.

Для транспортировки газовых проб, включая обогреваемые пробы, используется линия длиной до 150 метров. Обогреваемая линия транспортировки пробы предназначена для перемещения газовой пробы от газохода до газоанализаторов без необходимости пробоподготовки.

На основании вышеизложенной информации можно сделать вывод о дороговизне инструментального метода мониторинга промышленных выбросов. Также можно сказать о сложности внедрения и эксплуатации подобных систем.

1.1.2 Расчетный метод контроля выбросов

Этот метод применяется для оценки рассеивания выбросов, которые происходят от дымовых труб, вентиляционных шахт и также от источников организованного выброса загрязняющих веществ в атмосферу через

При расчете максимальных одноразовых концентраций ЗВ, учитывают возникающие в течение года комбинации значений M и V_1 при нормальных условиях эксплуатации предприятия. Данные значения используются для определения максимальной концентрации c_M ЗВ при времени осреднения в течение 20–30 минут. Методика расчета выбросов вредных или загрязняющих веществ в атмосферу от стационарных источников определяет способ зависимости мощности выброса M от скорости ветра.

При проектировании, реконструкции и введении в эксплуатацию объектов необходимо проводить расчеты для определения мощности выбросов M, высоты источников H, диаметра устьев D, температуры $T_{\text{г}}$ и расхода ГВС V_1 в технологической части проекта. Это относится как к новым объектам, так и к существующим производствам, которые подлежат инвентаризации стационарных источников выбросов вредных или загрязняющих веществ в атмосферный воздух.

При определении величины ΔT для предприятий, работающих по сезонному графику, допускается принимать значения расчетной температуры окружающего атмосферного воздуха $T_{\text{в}}$ равными средним месячным температурам воздуха за самый холодный месяц. Для остальных источников выбросов расчетная температура $T_{\text{в}}$ принимается равной средней максимальной температуре воздуха наиболее теплого месяца года.

Коэффициенты m и n определяются в зависимости от характеризующих свойства источника выброса параметров ν_M, ν'_M, f и f_e:

$$\nu_M = 0{,}65\sqrt[3]{\frac{V_1 \Delta T}{H}},$$

$$\nu'_M = 1{,}3\frac{\omega_0 D}{H},$$

$$f = 1000\frac{\omega_0^2 D}{H^2 \Delta T},$$

Различают несколько классификацийданных устройств. По принципу получения изображения тепловизоры делятся на:

1) тепловизоры с оптико-механическим сканированием. Основные элементы тепловизоров с оптико-механическим сканированием;

2) матричные тепловизоры.

Одним из главных компонентов тепловизоров с оптико-механическим сканированием, определяющим их тепловую чувствительность и максимальную дальность работы, является приемник инфракрасного излучения. В приемниках использованы чувствительные элементы, такие как фоторезисторы, которые изменяют свою проводимость под воздействием падающего излучения. Основным параметром приемников инфракрасного излучения является пороговая чувствительность – это минимальный поток излучения, который вызывает на выходе приемника сигнал, равный уровню шума или превышающий его в заданное количество раз.

С технической точки зрения, одним из преимуществ матричных тепловизоров является то, что они построены на основе матричного инфракрасного детектора. Это отличие от тепловизоров, которые используют сканирующие системы и которых сейчас ещё много на мировом рынке. За счет применения принципа накопления информационного сигнала, матричные тепловизоры обладают рядом преимуществ перед сканирующими системами, такими как высокая надежность, чувствительность, скорость работы и пространственное разрешение при равных условиях. На рисунке 1.2 [8] приведена типовая блок-схема матричных тепловизоров.

На сегодняшний день известно множество способов применения тепловизоров. Один из них – получение информации о состоянии материалов, степени их износа, примером является контроль за состоянием облицовки доменных печей. Полная замена облицовки доменных печей является весьма дорогой процедурой, так как влечет остановку производства на про-

2 СЕГМЕНТАЦИЯ ФАКЕЛА ВЫБРОСОВ С ПОМОЩЬЮ ТЕПЛОВЫХ СНИМКОВ

2.1 Постановка задачи сегментации факела выбросов

Необходимо провести классификацию пикселей последовательности изображении и соответствующих им элементов последовательности матриц температур, представляющих оптический и тепловой видео потоки, полученные с тепло-видео системы наблюдения. Выделить необходимо два класса – области соответствующие факелу вредных выбросов предприятий и не принадлежащие факелу. Если выражаться более формальным языком, определим целевую функцию (2.1), которая задает отображение множества X на множество Z.

$$f : X \to Z, \tag{2.1}$$

где X – множество последовательностей из пар вида x_i, y_i, где x_i – элемент множества изображений в пространстве RGB, представленных трехмерной матрицей, y_i – элемент множества двухмерных матриц, состоящих из чисел от 0 до 255. Z – множество последовательностей двухмерных матриц состоящих из вещественных чисел в диапазоне от 0 до 1, обозначающих вероятность принадлежности соответствующей пары пикселя и температуры к факелу вредных выбросов (маска).

Для того, чтобы восстановить целевую функцию (2.1) необходимо разработать алгоритм A, $A : X \to Z$. Этот алгоритм должен соответствовать следующим требованиям:

1) должен для некоторых элементов множества X, составляющих тестовую выборку находить маски максимально точно соответствующие заранее полученным элементам множества Z;

2) должен приближать целевую функцию для всех элементов из множества X;

3) должен допускать численную реализацию;

4) должен обеспечивать связность области дыма на маске.

2.2 Подготовка данных для задачи сегментации факела выбросов

2.2.1 Получение данных с тепловизора

Для решения поставленной задачи необходимо разработать алгоритм взаимодействия с тепловизором и научиться получать данные для последующей обработки. Для выполнения задачи был выбран тепловизор модели DS60xxFT-M (см. рисунок 2.1). Данное устройство предоставляет возможность получения оптических снимков в разрешении 1920 на 1080 пикселей, с частотой развертки в 25 Гц, а также тепловые снимки в разрешении 640 на 512 пикселей, с частотой в 25 Гц. Для данной модели тепловизора была разработана SDK (Software Development Kit – комплект для разработки программного обеспечения). Данный комплект инструментов представляет из себя набор готовых программ для подключения к тепловизору с персонального компьютера, а также взаимодействия с тепловизором. В перечень возможностей данного набора программ входит:

1) дистанционное упраление углом наклона и поворота тепловизора;

2) изменение уровня увеличения оптической камеры;

3) включение и выключение подсветки;

4) получение потока оптических снимков;

5) получение потока тепловых снимков;

6) получение матрицы температур;

7) сохранение оптических и тепловых снимков покадрово в память компьютера.

Все программы предоставляются в виде исходного кода с возможностью редактирования. Помимо SDK, предоставляется библиотека для работы с тепловизором и документация, используя которые можно само-

шить следующую подзадачу. Необходимо сопоставить каждому цвету из пространства RGB некоторую интенсивность пикселя, выполнив классификацию. Более формальным языком, Y – множество одномерных векторов длины 3 вида $[r_i, g_i, b_i]$, где r_i, g_i, b_i – целые числа от 0 до 255, X – множество целых чисел от 0 до 255. Необходимо восстановить целевую функцию (2.3) и реализовать для этого алгоритм отвечающий следующим требованиям:

1) для векторов полученых в результате преобразовании с помощью функции (2.2), должен находить значение, максимально приближенное к обратному преобразованию;

2) для векторов из Y, не являющихся результатом функции (2.2), должен также приближать целевую функцию;

3) должен допускать численную реализацию.

Для восстановления функции g было решено использовать модель машинного обучения «FlannBasedMatcher» [20], которая в общем случае позволяет задать отображение:

$$h : A \to B_i, \tag{2.4}$$

где A – некоторая трехмерная матрица фиксированных размеров, B – пространство трехмерных матриц такой же размерности. Введем оценку расстояния между трехмерными матрицами, которая будем вычислять по формуле:

$$D(A, B) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p}(A_{ijk} - B_{ijk})^2},$$

где n, m, p – размерности матриц. При этом для отображения (2.4) имеем:

$$D(A, B_i) = min(D(A, B_j)),$$

где $j \in [0, n-1]$, n – количество трехмерных матриц в пространстве B.

Данная модель основана на методе k ближайших соседей [13]. k бли-

ства RGB, что позволило существенно сократить время работы программы, по сути сведя функцию g к обращению к трехмерной матрице. Пример результата использования функции g представлен выше (см. рисунок 2.3).

2.2.3 Наложение карты абсолютных температур на оптические снимки

Заключительным этапом подготовки данных является преобразование изображения в оттенках серого в матрицу абсолютных температур и наложение полученной матрицы на оптический снимок. Изображение в оттенках серого является двумерной матрицей A, с высотой h и длинной w. Элементом массива является число от 0 до 255, которое является нормированным значением температуры. Перед восстановлением абсолютных температур необходимо провести изменение размера матрицы относительных температур, для корректного сопоставления. Для этого воспользуемся методом билинейной интерполяции. Данный метод представляет из себя обобщение линейной интерполяции одной переменной для функций двух переменных. Функция билинейной интерполяции интерполирует значения исходной функции двух переменных в произвольной подматрице по четырём её значениям в угловых элементах подматрицы и экстраполирует функцию на всю остальную поверхность. Данная функция имеет вид:

$$F(x,y) = b_1 + b_2x + b_3y + b_4xy, \tag{2.6}$$

где x и y – координаты элемента матрицы; b_1, b_2, b_3 и b_4 – некоторые неизвестные коэффициенты. Необходимо найти значение этих коэффициентов. Приведем один из способов вычисления этих коэффициентов. Для этого подставим в уравнение 2.6 координаты и значения угловых элементов подматрицы. Тогда необходимо решить систему из четырех уравнений (2.7):

В качестве алгоритма поиска ключевых точек был выбран алгоритм «SIFT». Метод SIFT (Scale-Invariant Feature Transform или Масштабно-инвариантное преобразование особенностей) – это один из наиболее популярных алгоритмов детектирования особых точек в изображениях [21]. Основная идея метода SIFT заключается в поиске особых точек, которые инвариантны к масштабу и повороту изображения, а также устойчивы к изменениям освещения и частичной закрытости. Алгоритм SIFT состоит из нескольких этапов:

1) построение пирамиды изображений: Изначальное изображение размывается с помощью гауссового фильтра с разными масштабами $G\left(x,y,k_i\sigma\right)$ в масштабе $k_i\sigma$, получаем изображение:

$$L\left(x,y,k_i\sigma\right) = G\left(k_i\sigma\right) * I\left(x,y\right),$$

где k_i – масштаб на некотором этапе, $I\left(x,y\right)$ – исходное изображение. Подробнее опишем применение фильтра гаусса. Для этого введем понятие операции свертки. Свертка – операция над матрицами $A_{n_x \times n_y}$ и $B_{m_x \times m_y}$, результатом которой является матрица $C_{(n_x - m_x + 1)\times(n_y - m_y + 1)} = A\ B$, где B – ядро или фильтр свертки. Каждый элемент результата вычисляется как скалярное произведение матрицы B и некоторой подматрицы A такого же размера (подматрица определяется положение элемента в результате). Формально элемент $C_{i,j}$ вычисляется по формуле:

$$C_{i,j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{m_y-1} A_{i+u,j+v} B_{u,v}.$$

Пример операции свертки показан на рисунке 2.8 [18].

В случае фильтра гаусса матрица B равна матрице $G(\sigma)$, элементы которой вычисляются по формуле:

$$G(\sigma)_{x,y} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-m_x/2)^2+(y-m_y/2)^2}{2\sigma^2}}, \tag{2.9}$$

значаются ключевой точке. Если же имеется несколько направлений, то создается дополнительная ключевая точка с точно таким же положением и масштабом, как у исходной точки, для каждого существующего дополнительного направления;

6) в начале процесса формируется набор гистограмм направлений, используя 4×4 соседних пикселя и разделение на 8 областей в каждой гистограмме. Подсчет данных гистограмм осуществляется на основе значений величины и ориентации элементов, взятых из 16×16 области вокруг ключевой точки. Значения гистограмм весятся с использованием функции Гаусса, где параметр σ равен половине ширины дескрипторного окна. Далее, дескриптор превращается в вектор, включающий все значения гистограмм. Общий размер вектора составляет 128 элементов. Для обеспечения инвариантности к аффинным изменениям в освещении, этот вектор нормализуется до единичной длины;

7) отбор особых точек – особые точки отбираются на основе надежности дескрипторов, исходя из порогового значения. Кроме того, особые точки могут быть отфильтрованы, если они находятся на границе изображения или находятся в областях с низким контрастом.

На этом алгоритм поиска ключевых точек завершается. В результате получаем набор ключевых точек и дескрипторов A для изображения образца и набор точек и дескрипторов B для входного изображения. Далее используем ранее описанный алгоритм «k ближайших соседей» для классификации элементов из набора A по n классам, где n – размер набора B. Для этого формируется обучающая выборка, представляющая набор дескрипторов из набора B. Далее для каждого элемента A_i получаем некоторый индекс t_i, соответствующий номеру класса, получаем вектор сопоставлений y_i, вида $[ap_i, ad_i, bp_{t_i}, bd_{t_i}, d_i]$, где ap_i и ad_i – точка и дескриптор из набора A соответственно, bp_{t_i} и bd_{t_i} – точка и дескриптор из набора B соответ-

8 **Основы тепловидения**: учебное пособие / В. В. Коротаев, Г. С. Мельников, С. В. Михеев, В. М. Самков, Ю. И. Солдатов. – Санкт-Петербург: Санкт-Петербургский национальный исследовательский университет информационных технологий, механники и оптики, 2012. – 123 с.

9 **Криксунов, Л. З.** Тепловизоры: справочник / Л. З. Криксунов, Г. А. Падалко – Киев: Техника, 1987. – 287 с.

10 **Об утверждении методов расчетов рассеивания выбросов вредных (загрязняющих) веществ в атмосферном воздухе**: Приказ министерства природных ресурсов и экологии Российской Федерации от 6 июня 2017 года № 273 // Министерство природных ресурсов и экологии Российской Федерации, 2017. – 1–10 с.

11 **Методическое пособие по аналитическому контролю выбросов загрязняющих веществ в атмосферу** / В. В. Цибульский, Л. И. Короленко, М. А. Яценко-Хмелевская, О. Р. Синицына, М. В. Боровкова. – Санкт-Петербург: ОАО «НИИ АТМОСФЕРА», 2012. – 25 с.

12 **Beucher, S** The watershed transformation applied to image segmentation / K. V. Lalitha, R. Amrutha, S. Michahial // Scanning Microscopy. – 1992. – V. 1992, № 6. – P. 299–314.

13 **Cunningham, P.** k-Nearest neighbour classifiers: a tutorial / P. Cunningham, S. J. Delany // ACM computing surveys. – 2021. – V. 54, № 6. – P. 1–25.

14 **Davison, J.** Gasoline and diesel passenger car emissions deterioration using on-road emission measurements and measured mileage / J. Davison, R. A. Rose, N. J. Farren // Atmospheric Environment: X. – 2022. – V. 14. – P. 100162.

15 Automation Technology: Early fire detection and condition monitoring: [сайт]. – 2020. URL: https://www.automationtechnology.de/cms/wp-content/uploads/2015/12/Monitoring-application-overview.pdf (дата обращения: 08.02.2023).

16 **Fischler, M. A.** Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography / M. A. Fischler, R. C. Bolles // Communications of the ACM. – 1981. – V. 24, № 6. – P. 381–395.

17 **Lee, J.** FLIR Technology: 'New' Technology Shines a Camera on Greenhouse Gas Emissions / J. Lee // Triple Pundit. – 2018. – № 12. – P. 1 URL: https://clck.ru/34N7zc (дата обращения: 02.01.2023).

18 **Mohamed, I. S.** Detection and tracking of pallets using a laser rangefinder and machine learning techniques / I. S. Mohamed. – Genova: European Master on Advanced Robotics+(EMARO+), 2017. – 77 p.

19 **Moreland, K. D.** Diverging Color Maps for Scientific Visualization (Expanded) / K. D. Moreland // Sandia National Laboratories. – 2009. – P. 3.

20 OpenCV: Feature Detection and Description: [сайт]. – 2018. URL: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html (дата обращения: 01.02.2023).

21 Scholarpedia: Scale Invariant Feature Transform: [сайт]. – 2016. URL: http://www.scholarpedia.org/article/Scale_Invariant_Transform (дата обращения: 05.04.2023).

22 **Ibraheem, N. A.** Understanding color models: a review / N. A. Ibraheem, M. M. Hasan, R. Z. Khan // Journal of science and technology. – 2009. – V. 2, № 3. – P. 265–275.

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);
}
```