Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
(БГУИР)

УДК 621.391.1; 378.147
№ госрегистрации 20140149
Инв. №

УТВЕРЖДАЮ
Проректор по научной работе
д-р техн. наук, проф.
_____А.П. Кузнецов
«02» февраля 2015 г.

ОТЧЕТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

«РАЗРАБОТАТЬ МЕТОДЫ, АЛГОРИТМЫ И ПРОГРАММНЫЕ СРЕДСТВА ОБНАРУЖЕНИЯ ОБЪЕКТОВ И ОПРЕДЕЛЕНИЯ ИХ КООРДИНАТ НА КАДРАХ ВИДЕОПОТОКА С БОРТА БЕСПИЛОТНОГО ЛЕТАТЕЛЬНОГО АППАРАТА»

(итоговый)

ХД №13-1188Б

Руководитель темы,
д-р техн. наук, проф. _____ В.К. Конопелько

Ответственный исполнитель,
канд. техн. наук, доц. _____ В.Ю. Цветков

Минск 2015

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы,
д-р техн. наук, проф. _____ В.К. Конопелько (введение, разделы 3, 4, заключение)

Отв. исполнитель темы,
канд. техн. наук, доц. _____ В.Ю. Цветков (разделы 1–4)

Исполнитель темы
канд. техн. наук _____ К.А. Волков (разделы 1–3)

Исполнитель темы _____ И.А. Борискевич (разделы 1–3)

Нормоконтролер _____ Л.А. Шичко

РЕФЕРАТ

Отчет 153 с.; 54 иллюстрации; 3 таблицы; 53 источника.

ОБНАРУЖЕНИЕ ОБЪЕКТА; СОПРОВОЖДЕНИЕ ОБЪЕКТА; СЛЕЖЕНИЕ ЗА НАЗЕМНЫМ ОБЪЕКТОМ С БОРТА БЕСПИЛОТНОГО ЛЕТАТЕЛЬНОГО АППАРАТА; ОПРЕДЕЛЕНИЕ КООРДИНАТ ОБЪЕКТА.

Цель работы: разработка методов, алгоритмов и программных средств сопровождения и определения координат объектов наблюдения на кадрах видеопотока от аналоговой видеокамеры в реальном масштабе времени.

Отчет по НИР включает следующие вопросы.

В разделе 1 представлены методы, алгоритмы и программные средства обнаружения, сопровождения и параметризация движения объекта.

В разделе 2 представлены методы, алгоритмы и программные средства улучшения качества и нормализация кадров видеопотока с борта БЛА, выделения объекта наблюдения, отмеченного оператором, на кадре видеопотока.

В разделе 3 представлены методы, алгоритмы и программные средства определения координат объекта наблюдения.

В разделе 4 представлены результаты анализа аналоговых и цифровых способов передачи видеоданных с борта БЛА.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Сопровождение цели (поиск соответствия между эталонным изображением цели и текущим кадром видеопоследовательности) является важным компонентом ряда приложений компьютерного зрения. Устойчивое сопровождение малоразмерных объектов с использованием нестационарной видеокамеры в реальном масштабе времени является нерешенной задачей.

Для решения данной задачи может быть использован метод сдвига среднего (Mean Shift) [1] и обучаемые классификаторы [2], В первом случае производится итеративный корреляционный поиск соответствия области текущего кадра с эталонным изображением цели, начиная с местоположения сопровождаемого объекта на предыдущем кадре. Во втором случае требуется предварительное обучение классификатора для разграничения пикселов объекта и фона. Данные методы обеспечивают эффективное сопровождение только в том случае, если цель незначительно смещается от кадра к кадру [3].

Альтернативой является использовании двухэтапных методов, в которых вначале осуществляется стабилизации текущего кадра, а далее производится обнаружения объекта посредством вычитания фона [4]. Однако высокая вычислительная сложность алгоритмов стабилизации не позволят применять данные методы в реальном масштабе времени [5].

Метод ковариационного сопровождение [3], учитывающий как яркостные, так и текстурные свойства цели, обеспечивает устойчивое сопровождение, однако из-за высокой сложности вычисления корреляционной матрицы данный метод также не позволяет обрабатывать видео в реальном масштабе времени.

Целью работы является разработка методов, алгоритмов и программных средств сопровождения и определения координат объектов наблюдения на кадрах видеопотока от аналоговой видеокамеры в реальном масштабе времени.

В задачи НИР входят исследование возможности, разработка методов, алгоритмов и программных средств сопровождения и определения координат объектов наблюдения на кадрах видеопотока от аналоговой видеокамеры в реальном масштабе времени двумя методами: по одному кадру видеопотока на основе координат носителя (GPS) и телеметрии (высота, скорость и ориентация носителя, ориентация и zoom камеры) в реальном масштабе времени; по двум кадрам видеопотока на основе координат носителя и телеметрии в реальном масштабе времени; исследование возможности и оценка эффективности использования криптографического и помехоустойчивого кодирования при передаче цифрового видеопотока с борта БЛА.

1 Обнаружение, сопровождение и параметризация движения объекта наблюдения

1.1 Методы, алгоритмы и программные средства обнаружения и сопровождения объекта наблюдения на кадрах видеопотока в реальном масштабе времени

Для быстрого поиска малоразмерных целей разработан пространственно-частотный ковариационный метод поиска, основанный на непрореженном дискретном лифтинг вейвлет-преобразовании Хаара [6, 7]. Сущность метода состоит в формировании признаковых изображений для перекрывающихся признаковых окон в пределах области поиска цели на текущем кадре видеопоследовательности, вычислении ковариационных матриц признаковых изображений и их сравнении с матрицей эталона. В отличие от метода пространственного корреляционного поиска цели предложенный метод использует субобласти аппроксимирующих и детализирующих коэффициентов непрореженного дискретного лифтинг вейвлет-преобразования Хаара в качества признаковых изображений для формирования ковариационного дескриптора (рисунок 1.1). Это обеспечивает лучшую пространственно-частотную локализацию цели и позволяет уменьшить время и вероятность ложного обнаружения цели, повысить устойчивость к шуму и изменению контраста видеопоследовательности. Возможны следующие комбинации коэффициентов первого уровня непрореженного дискретного лифтинг вейвлет-преобразования Хаара для вычисления ковариационного дескриптора: аппроксимирующие, вертикальные, горизонтальные и диагональные детализирующие (LL/LH/HL/HH), аппроксимирующие и суммарные детализирующие (LL/H), аппроксимирующие и диагональные детализирующие (LL/HH).

Рисунок 1.1 – Признаковые изображения, вычисленные в результате непрореженного дискретного лифтинг вейвлет-преобразования Хаара (LL/LH/HL/HH/H)

Алгоритм пространственно-частотного ковариационного поиска цели включает следующие шаги.

1) Инициализация начальных параметров алгоритма.

1.1) Формирование эталонного изображения E цели.

Производится считывание первого кадра $I(t) = \| i(t,x,y) \|_{(x=\overline{1,X}, y=\overline{1,Y})}$ видеопоследовательности $I = \| I(t) \|_{t=\overline{1,T}}$, где $X \times Y$ – размер кадра; $t = \overline{1,T}$ и T – номер и число кадров в видеопоследовательности соответственно. В качестве эталонного изображения выбирается прямоугольная область размером $M \times N$ на первом кадре видеопоследовательности, центральный пиксель которой является центром цели.

1.2) Формирование ковариационной матрицы и гистограммы эталонного изображения цели.

Для эталонного изображения E вычисляются ковариационная матрица C_E и гистограмма H_E.

1.3) Инициализация счетчиков кадров.

Устанавливаются начальные значения счетчиков кадров видеопоследовательности $t = 1$ и последовательных кадров без обнаруженной цели $T_a = 0$.

2) Начало цикла обработки кадров видеопоследовательности.

Переход к обработке очередного кадра видеопоследовательности. Значение счетчика кадров видеопоследовательности увеличивается на единицу: $t = t + 1$.

3) Определение области цели на t-м кадре видеопоследовательности.

В зависимости от условий область цели может быть образована пикселями наиболее вероятного положения цели или включать весь кадр.

4) Начало цикла обработки признаковых окон.

4.1) Инициализация счетчика признаковых окон: $l = 0$.

4.2) Вычисление элементов пространственно-частотной ковариационной матрицы $C_l(t) = \| c_l(i,j) \|_{i,j=\overline{1,D}}$ для l-го признакового окна

$$c_l(i,j) = \frac{1}{XY}(F(i) - \mu(i))(F(j) - \mu(j)), \qquad (1.1)$$

где $F_l = \| LL(x,y) \quad HL(x,y) \quad LH(x,y) \quad HH(x,y) \|_{x=\overline{1,X}, y=\overline{1,Y}}$ – признаковый образ, состоящий из аппроксимирующих и детализирующих вертикальных, горизонтальных и диагональных коэффициентов непрореженного дискретного лифтинг вейвлет-преобразования Хаара.

4.3) Вычисление весовой метрики сходства $\rho_l^2(t, C_E, C_l(t))$ ковариационных матриц l-го признакового окна $C_l(t)$ и эталона C_E с учетом близости к ожидаемому положению цели

$$\rho_l^2(C_E, C_l) = w_l tr[\log^2(C_E^{-1/2} C_l C_E^{-1/2}], \quad (1.2)$$

где $w_l = f(l, X, Y)$ – позиционный весовой коэффициент, учитывающий размеры цели и положение текущего признакового окна относительно ожидаемого положения цели. В зависимости от условий в качестве ожидаемого положения цели может использоваться положение цели на предыдущем кадре или прогнозируемое положение с учетом перемещения камеры и цели.

4.4) Значение счетчика признаковых окон увеличивается на единицу: $l = l + 1$.

4.5) Проверка условия окончания цикла обработки признаковых окон. Если $l < L$, то осуществляется переход на шаг 4.2, иначе – выход из цикла обработки признаковых окон.

5) Выбор лучшего признакового окна.

Номер лучшего признакового окна $W \in [1, L]$ вычисляется с помощью соотношения

$$W = \arg\max_{l \in \{1, L\}} (\rho_l^2(t, C_E, C_l(t))), \quad (1.3)$$

где $\arg\max_x(f(x))$ – функция максимизации, вычисляющая аргумент, соответствующий максимальному значению функции.

6) Сравнение метрики сходства с порогом.

Производится сравнение значения метрики сходства $\rho_W^2(t, C_E, C_l(t))$ для W-го признакового окна с заданным пороговым значением ρ_{Th}^2. Если условие $\rho_W^2(t, C_E, C_l(t)) > \rho_{Th}^2$ не выполняется, то принимается решение об отсутствии цели на t-м кадре видеопоследовательности. Значение счетчика T_a последовательных кадров без обнаруженной цели увеличивается на единицу $(T_a = T_a + 1)$ и осуществляется переход к шагу 7. Если условие $\rho_W^2(t, C_E, C_l(t)) > \rho_{Th}^2$ выполняется, то принимается решение об обнаружении цели, счетчик T_a обнуляется и производится переход к шагу 7.

7) Проверка условия окончания цикла обработки кадров видеопоследовательности.

Если обработка кадров не завершена $(t < T)$ осуществляется переход на шаг 2, иначе – выход из алгоритма.

В результате выполнения данного алгоритма для каждого кадра видеопоследовательности определяются координаты центра лучшего признакового окна W, в котором обнаружена цель или принимается решение об отсутствии цели в кадре.

На основе данного алгоритма разработаны программные средства поиска и сопровождения объекта наблюдения на кадрах видеопотока с БЛА. Описание программных средств поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА приведено в приложении А.

1.2 Методы, алгоритмы и программные средства вычисления параметров движения объекта наблюдения в реальном масштабе времени

Вычисление параметров движения (скорости, ускорения, направления) объекта наблюдения осуществляется на основе мгновенных значений его координат для каждого кадра видеопотока.

Возможны два основных подхода к определению параметров движения объекта наблюдения: по восстановленной на основе последовательности кадров видеопотока и телеметрии GPS-траектории объекта наблюдения; по смещению объекта наблюдения на кадрах видеопотока и телеметрии (без GPS-координат). Точность первого подхода в настоящее время ограничена точность определения GPS-координат. Точность второго подхода ограничена точностью фиксирования параметров телеметрии. Второй подход представляется более эффективным в силу более высокой точности.

В рамках НИР разработаны алгоритмы и программные средства определения параметров движения объекта наблюдения по смещению объекта наблюдения на кадрах видеопотока и телеметрии. Алгоритмы основаны на определении расстояния до цели по видеокадру и телеметрии. Описание программных средств определения параметров движения объекта наблюдения по его смещению на кадрах видеопотока и телеметрии приведено в приложении Б.

1.2.1 Алгоритм определения расстояния до цели по видеокадру и телеметрии

Для повышения точности и быстродействия воздушного сопровождения наземных объектов разработан алгоритм определения расстояния до цели по видеокадру и телеметрии. Сущность алгоритма заключается в использовании информации о параметрах поворота, тангажа, крена и сдвига видеокамеры, электронных карт рельефа местности и координат объекта на предыдущем (опорном) кадре $F(t-1)$.

Алгоритм предсказания области поиска сопровождаемого объекта по телеметрии включает следующие шаги.

1 Инициализация начальных параметров.

2 Компенсация крена видеокамеры и БЛА.

3 Определение расстояния до сопровождаемого объекта с компенсацией перепадов рельефа местности.

4 Компенсация поворота видеокамеры.

5 Компенсация сдвига видеокамеры.

6 Вычисления координат центрального пикселя цели.

1.2.2 Инициализация начальных параметров алгоритма

Модель исходного кадра $F(t-1)$ видеопоследовательности $F=\{F(t)\,|\,t=\overline{1,T}\}$ с данными телеметрии, соответствующими моменту времени $t-1$, можно представить в следующем виде:

$$F(t-1)=\{x(t-1),y(t-1),A(t-1),\beta_X(t-1),\beta_Y(t-1),\beta(t-1)\}\,, \qquad (1.4)$$

где $x(t-1)$ и $y(t-1)$ – значения координат центрального пиксела сопровождаемого объекта в момент времени $t-1$;

$H(t-1)$ – высота расположения сопровождаемого объекта, определяемая по электронной карте рельефа местности;

$\beta_X(t-1)$ – угол поворота видеокамеры в горизонтальной плоскости;

$\beta_Y(t-1)$ – угол поворота видеокамеры в вертикальной плоскости (тангажа);

$\beta(t-1)$ – угол крена видеокамеры.

1.2.3 Компенсация крена видеокамеры и БЛА

На данном шаге производится расстояний от центра фотоприемной матрицы до проекций объекта на горизонтальную и вертикальную оси с компенсацией крена видеокамеры и БЛА (рисунок 1.5).

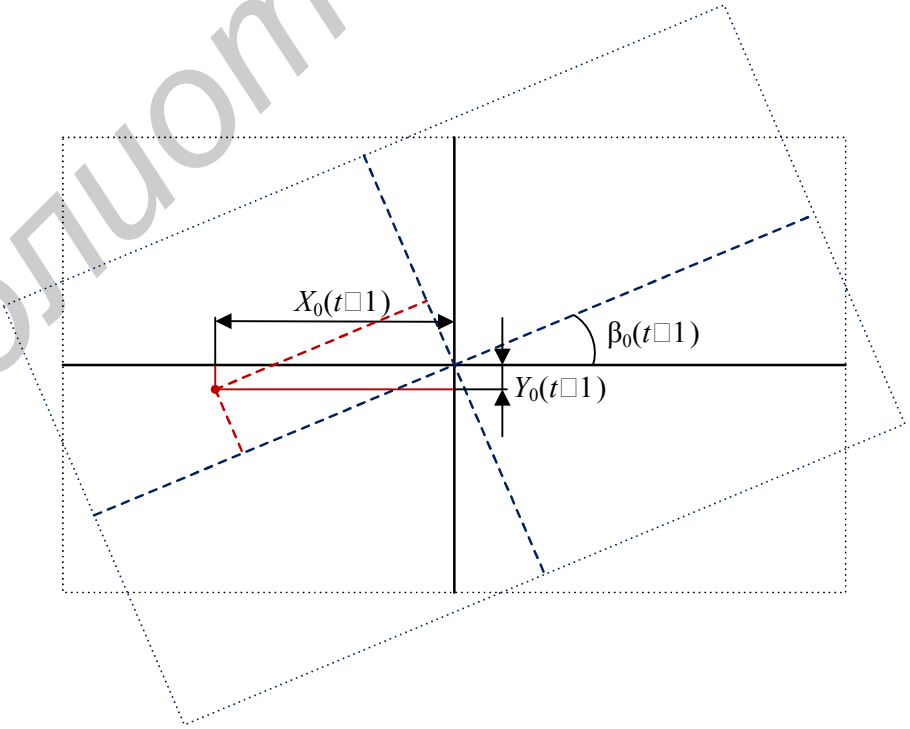


Рисунок 1.5 – Компенсация крена видеокамеры и БЛА
(вид сзади относительно фотоприемной матрицы)

Реальные физические расстояния от центра фотоприемной матрицы до проекций объекта на горизонтальную и вертикальную оси вычисляются с помощью следующих выражений:

$$X_0(t-1) = R \cdot \cos(\beta_0(t-1) - \text{arctg}(Y(t-1)/X(t-1)), \quad (1.5)$$

$$Y_0(t-1) = R \cdot \sin(\beta_0(t-1) - \text{arctg}(Y(t-1)/X(t-1)), \quad (1.6)$$

где $X(t-1)$ и $Y(t-1)$ – искаженные креном видеокамеры и БЛА расстояния от центра фотоприемной матрицы до центров проекций сопровождаемого объекта на горизонтальную и вертикальную оси;

$R = \sqrt{(X(t-1))^2 + (Y(t-1))^2}$ – радиус окружности с центром в середине неискаженного опорного кадра, по которой смещается проекция объекта;

$\beta_0(t-1) = \beta(t-1) + \beta_{UAV}(t-1)$ – реальный физический угол крена видеокамеры;

$\beta_{UAV}(t-1)$ – угол крена БЛА.

1.2.4 Определение расстояния до сопровождаемого объекта с компенсацией перепадов рельефа местности

На данном шаге производится расчет проекции на ось движения БЛА расстояния $d_Z(t-1)$ от камеры до сопровождаемого объекта (рисунок 1.6).

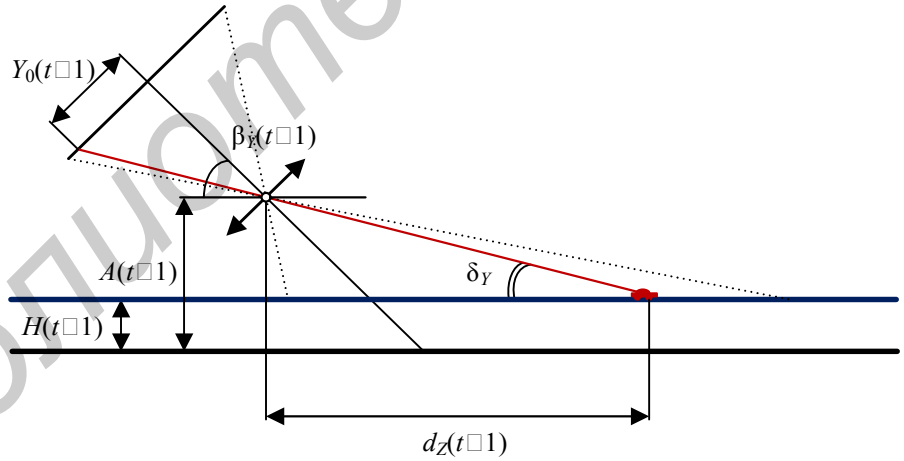


Рисунок 1.6 – Проекция на ось движения БЛА расстояния от камеры до сопровождаемого объекта (вид сбоку относительно фотоприемной матрицы)

Проекция расстояния вычисляется с помощью следующего выражения:

$$d_Z(t) = \frac{A(t-1) - H(t-1)}{\text{tg}(\delta_Y)}, \quad (1.7)$$

где $H(t-1)$ – высота расположения сопровождаемого объекта, определяемая по электронной карте рельефа местности в момент времени $t-1$;

$\delta_Y = 90 - \beta_Y(t-1) + \alpha_Y Y(t-1)/Y$;

α_Y – угол обзора видеокамеры в вертикальной плоскости;

Y – высота фотоприемной матрицы.

1.2.5 Компенсация поворота видеокамеры

На данном шаге производится компенсация поворота камеры в горизонтальной (рисунок 1.7) и вертикальной (рисунок 1.8) плоскостях, крена видеокамеры (рисунок 1.9).

1.2.5.1 Компенсация поворота видеокамеры в горизонтальной плоскости

Смещение области поиска сопровождаемого объекта на фотоприемной матрице при повороте видеокамеры в горизонтальной плоскости вычисляется с помощью следующего выражения:

$$s_X(t) = \frac{X\Delta\beta_X(t)}{\alpha_X}, \quad (1.8)$$

где X – ширина фотоприемной матрицы;

α_X – угол обзора видеокамеры в горизонтальной плоскости;

$\Delta\beta_X(t) = \beta_X(t-1) - \beta_X(t)$ – угол поворота видеокамеры в горизонтальной плоскости относительно ее положения в момент времени $t-1$.

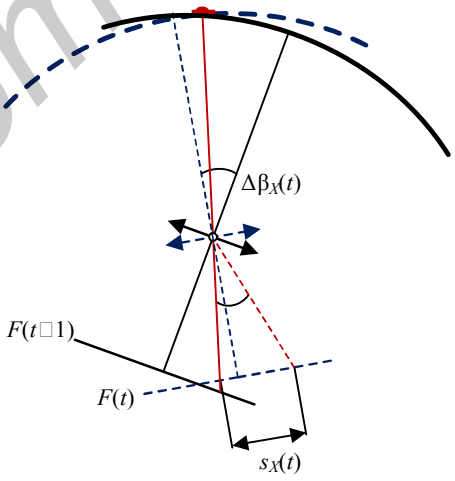


Рисунок 1.7 – Поворот видеокамеры в горизонтальной плоскости
(вид сверху относительно фотоприемной матрицы)

1.2.5.2 Компенсация поворота видеокамеры в вертикальной плоскости (компенсация тангажа видеокамеры)

Смещение области поиска сопровождаемого объекта на фотоприемной матрице при тангаже видеокамеры вычисляется с помощью следующего выражения:

$$s_Y(\mathrm{t}) = \frac{Y\Delta\beta_Y(t)}{\alpha_Y}, \tag{1.9}$$

где $\Delta\beta_Y(t) = \beta_Y(t-1) - \beta_Y(t)$ – угол тангажа видеокамеры относительно ее положения в момент времени $t-1$.

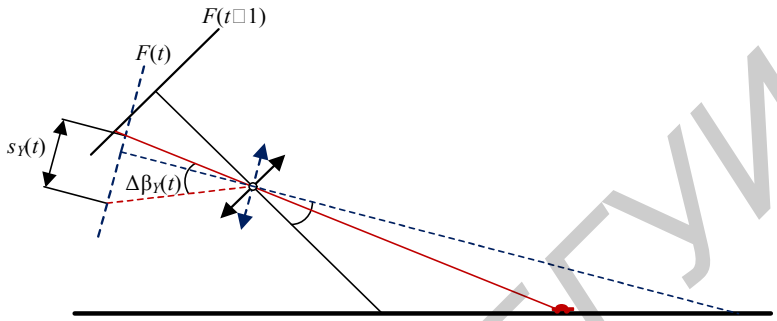


Рисунок 1.9 – Поворот видеокамеры в вертикальной плоскости
(вид сбоку относительно фотоприемной матрицы)

1.2.5.3 Компенсация крена видеокамеры

Смещение области поиска сопровождаемого объекта на фотоприемной матрице в горизонтальной и вертикальной плоскостях при крене видеокамеры вычисляется с помощью следующих выражений:

$$s_{RX}(t) = X(t-1) - R\cdot\cos(\beta(t) - \mathrm{arctg}(Y(t-1)/X(t-1)), \tag{1.10}$$

$$s_{RY}(t) = Y(t-1) - R\cdot\sin(\beta(t) - \mathrm{arctg}(Y(t-1)/X(t-1)), \tag{1.11}$$

Где $R = \sqrt{(X(t-1))^2 + (Y(t-1))^2}$ – радиус окружности с центром в середине опорного кадра, по которой смещается проекция объекта;

$\beta(t)$ – угол крена видеокамеры в момент времени t.

1.2.6 Компенсация сдвига видеокамеры

На данном шаге производится компенсация сдвига камеры в горизонтальной (рисунок 1.10) и вертикальной (рисунок 1.11) плоскостях, по оси движения БЛА (рисунки 1.12 и 1.13).

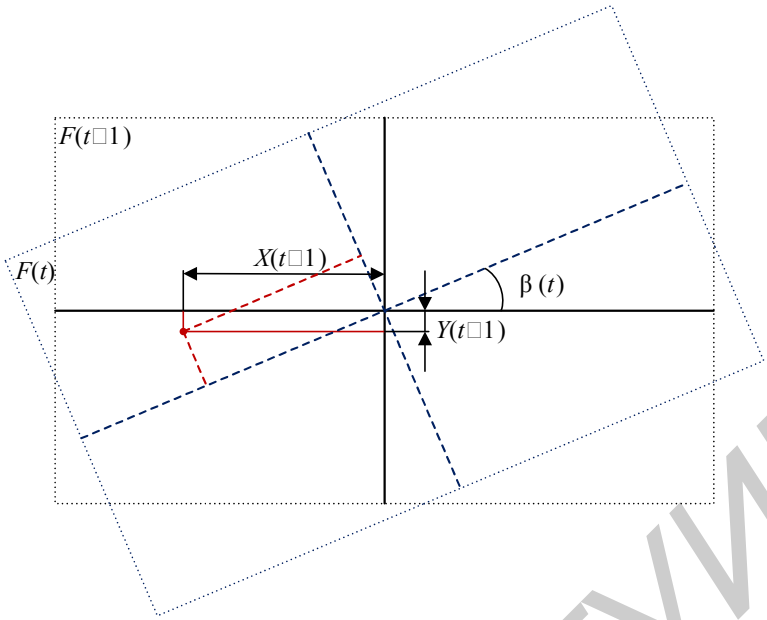


Рисунок 1.9 – Крен видеокамеры

(вид сзади относительно фотоприемной матрицы)

1.2.6.1 Компенсация сдвига видеокамеры в горизонтальной плоскости

Смещение области поиска сопровождаемого объекта на фотоприемной матрице при сдвиге видеокамеры в горизонтальной плоскости вычисляется с помощью следующего выражения:

$$s_{SY}(t) = \frac{X_0(t-1)S_{SX}(t)}{d_Z(t-1)\,\mathrm{tg}(\gamma_X(t-1))}, \tag{1.12}$$

где $S_{SX}(t)$ – сдвиг видеокамеры в горизонтальной плоскости относительно ее положения в момент времени $t-1$.

$\gamma_X(t-1) = \alpha_X X_0(t-1)/X$ – угол между главной оптической осью и осью проекции сопровождаемого объекта на фотоприемную матрицу.

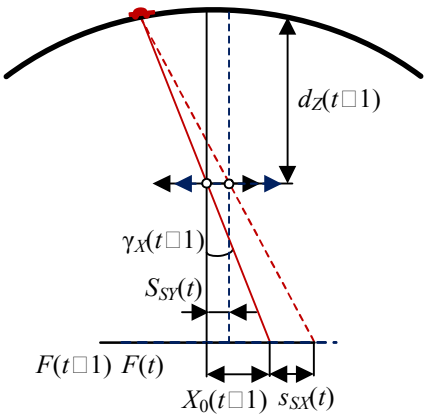


Рисунок 1.10 – Сдвиг видеокамеры в горизонтальной плоскости

(вид сверху относительно фотоприемной матрицы)

1.2.6.2 Компенсация сдвига видеокамеры в вертикальной плоскости

Смещение области поиска сопровождаемого объекта на фотоприемной матрице при сдвиге видеокамеры в вертикальной плоскости вычисляется с помощью следующего выражения

$$s_{SY}(t)=\frac{Y_0(t-1)S_{SY}(t)}{A(t-1)-H(t-1)-S_{SY}(t)}, \tag{1.13}$$

где $S_{SY}(t)$ – сдвиг видеокамеры в вертикальной плоскости относительно ее положения в момент времени $t-1$.

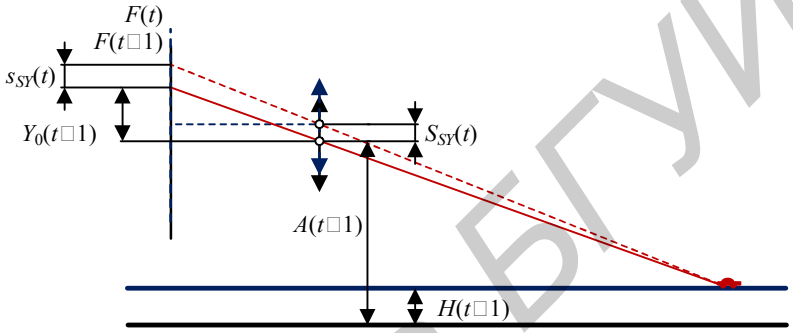


Рисунок 1.11– Сдвиг видеокамеры в вертикальной плоскости
(вид сбоку относительно фотоприемной матрицы)

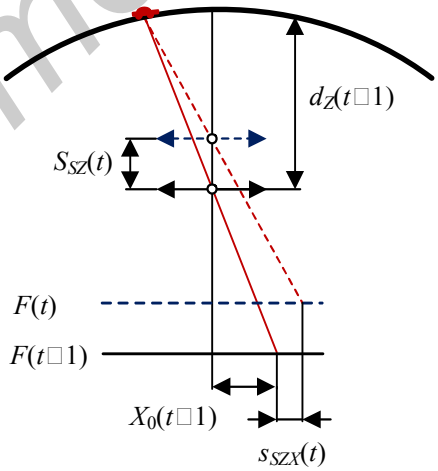


Рисунок 1.12 – Сдвиг видеокамеры по оси движения БПЛА
(вид сверху относительно фотоприемной матрицы)

1.2.6.3 Компенсация сдвига видеокамеры по оси движения БЛА

Горизонтальное смещение области поиска сопровождаемого объекта на фотоприемной матрице при сдвиге видеокамеры по оси движения БЛА вычисляется с помощью следующего выражения:

$$s_{SZX}(t) = X_0(t-1)S_{SZ}(t) / \left(d_Z(t-1) - S_{SZ}(t)\right), \quad (1.14)$$

где $S_{SZ}(t)$ – сдвиг видеокамеры по оси движения БЛА относительно ее положения в момент времени $t-1$.

Вертикальное смещение области поиска объекта на фотоприемной матрице при сдвиге видеокамеры по оси движения БЛА вычисляется с помощью следующего выражения:

$$s_{SZY}(t) = Y_0(t-1)S_{SZ}(t) / \left(d_Z(t-1) - S_{SZ}(t)\right), \quad (1.15)$$

где $S_{SZ}(t)$ – сдвиг видеокамеры по оси движения БПЛА относительно ее положения в момент времени $t-1$.

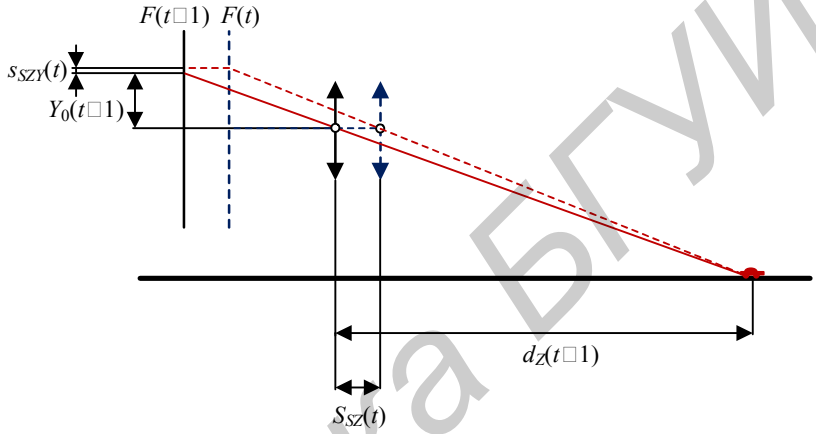


Рисунок 1.13 – Сдвиг видеокамеры по оси движения БЛА
(вид сбоку относительно фотоприемной матрицы)

1.2.7 Вычисление координат центрального пикселя цели

Значения координат центрального пикселя области поиска для нового кадра рассчитываются с помощью следующих выражений:

$$x(t) = x(t-1) + s_X(t) + s_{SZX}(t) + s_{SX}(t) + s_{RX}(t), \quad (1.16)$$

$$y(t) = y(t-1) + s_Y(t) + s_{SZY}(t) + s_{SY}(t) + s_{RY}(t). \quad (1.17)$$

1.3 Условия устойчивого обнаружения и сопровождения объектов по кадрам видеопотока с борта БЛА

Для оценки эффективности разработанного пространственно-частотного ковариационного метода и созданных программных средств использованы вероятность правильного обнаружения и время обработки кадра (обнаружения цели), оцениваемые в условиях изменения частоты, контраста и зашумления кадров.

Рассмотрены три комбинации коэффициентов первого уровня непрореженного дискретного лифтинг вейвлет-преобразования Хаара для вычисления ковариационного дескриптора: аппроксимирующие, вертикальные, горизонтальные и диагональные детализирующие (LL/LH/HL/HH), аппроксимирующие и суммарные детализирующие (LL/H), аппроксимирующие и диагональные детализирующие (LL/HH).

Установлено, что при понижении частоты кадров предложенный метод позволяет повысить вероятность правильного обнаружения цели на 15,5 % для ковариационного дескриптора на основе всех коэффициентов первого уровня разложения непрореженного дискретного лифтинг вейвлет-преобразования Хаара, на 13,9 % – для аппроксимирующих и суммарных детализирующих коэффициентов и на 15,1 % – для аппроксимирующих и диагональных детализирующих коэффициентов (рисунок 1.2).

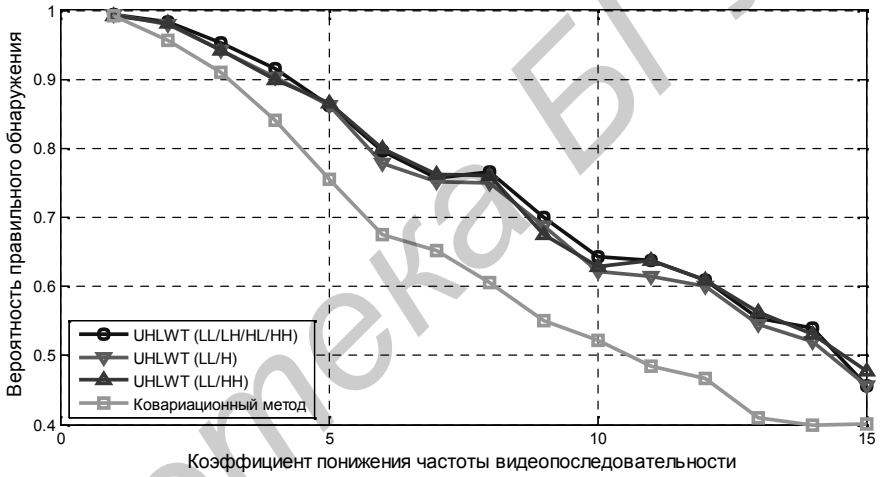


Рисунок 1.2 – Вероятность правильного обнаружения цели

Для оценки вероятности правильного обнаружения в условиях зашумления кадров использовался аддитивный гауссовый шум с нулевым средним и дисперсией σ_η^2. Установлено, что по сравнению с пространственным ковариационным методом предложенный метод в среднем на 10,7 % более устойчив к зашумлению при использовании всех коэффициентов первого уровня непрореженного дискретного лифтинг вейвлет-преобразования Хаара, обладает сопоставимой устойчивостью при использовании аппроксимирующих и суммарных детализирующих коэффициентов, обладает худшей устойчивостью к зашумлению при использовании аппроксимирующих и диагональных детализирующих коэффициентов (рисунок 1.3).

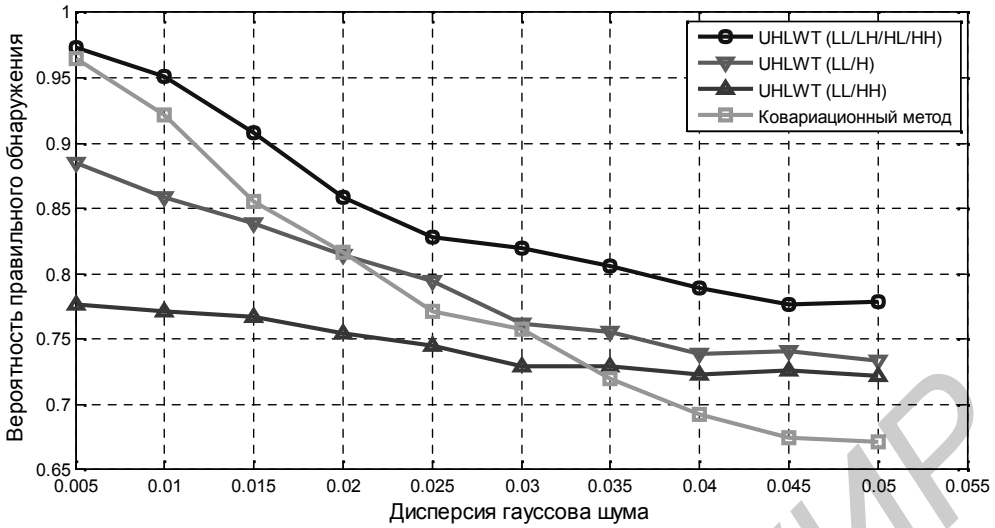


Рисунок 1.3 – Устойчивость методов обнаружения цели к шуму

Для оценки вероятности правильного обнаружения цели в условиях изменения контрастно-яркостных характеристик видеопоследовательности проводилось коррекция интенсивности пикселей в заданном диапазоне. Установлено, что предложенный метод обладает лучшей на 4,4 % устойчивостью к изменению контрастно-яркостных характеристик видеопоследовательности по сравнению с пространственным ковариационным методом (рисунок 1.4).

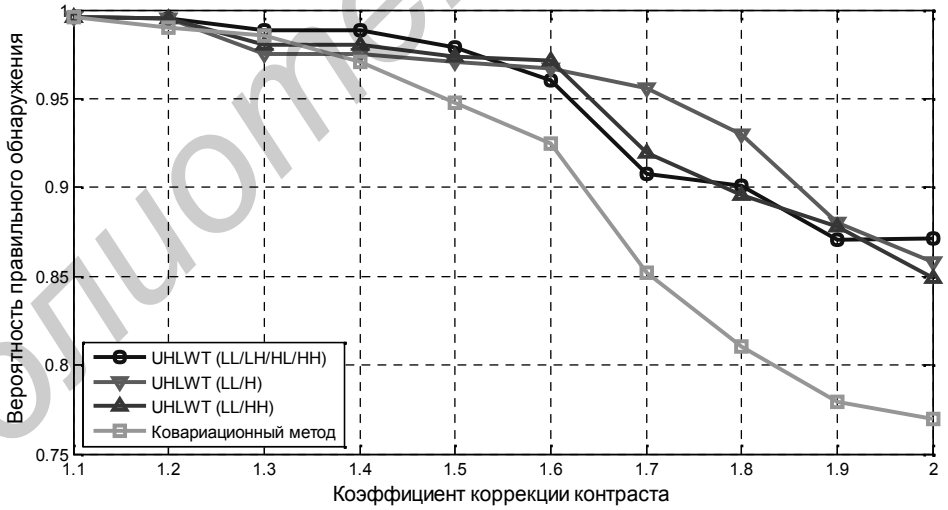


Рисунок 1.4 – Устойчивость методов обнаружения цели к изменению контрастно-яркостных характеристик видеопоследовательности

Время поиска цели на одном кадре видеопоследовательности оценивалось в среде программирования MATLAB R2013a на компьютере с процессором Intel Core i5 (2,6 ГГц) и ОЗУ 4 ГБ. Обработано 10 видеопоследовательностей (1500 видеокадров) размером 720×480 пикселей, полученных с беспилотного летательного аппарата. Площадь сопровождаемой

цели изменялась в диапазоне от 100 до 400 пикселей. Среднее время обработки одного кадра видеопоследовательности при использовании ковариационного метода составило 347,1 мс, при использовании модифицированного метода на основе всех коэффициентов первого уровня непрореженного дискретного лифтинг вейвлет-преобразования Хаара – 48,2 мс (выигрыш 7,2 раза), на основе аппроксимирующих и суммарных детализирующих коэффициентов – 41,7 мс (выигрыш 8,3 раза), на основе аппроксимирующих и диагональных детализирующих – 41,6 мс (выигрыш 8,3 раза).

Таким образом, для обеспечения лучшей устойчивости к зашумлению целесообразно формировать ковариационный дескриптор на основе субобласти аппроксимирующих и трех субобластей детализирующих коэффициентов первого уровня непрореженного дискретного лифтинг вейвлет-преобразования Хаара, для лучшего быстродействия – на основе субобласти аппроксимирующих и суммарной субобласти детализирующих коэффициентов.

1.4 Алгоритмы сопровождения малоразмерной цели с предсказанием по телеметрии

1.4.1 Структура автомата сопровождения малоразмерной цели с предсказанием по телеметрии

Разработана структура автомата сопровождения малоразмерной цели с предсказанием по телеметрии, состоящая из пяти блоков (рисунок 1.5):

- захвата, статической параметризации и определения местоположения цели;
- предсказания местоположения цели;
- поиска цели;
- уточнения статических параметров и местоположения цели;
- динамической параметризации цели.

Автомат сопровождения имеет 4 входа, на которые подаются координаты цели, отмеченные на кадре видеопотока оператором; кадр видеопотока с целью, отмеченной оператором; телеметрия; текущее время (с точностью, например, до 1 микросекунды).

Автомат сопровождения имеет интерфейс с электронной картой местности, через который определяется профиль местности на цель. Использование данного интерфейса и самой карты местности имеет смысл, если точность определения направления на цель достаточно высока и местность имеет сложный рельеф.

Автомат сопровождения имеет интерфейс с базой аберрационных корректирующих коэффициентов, через который определяется матрица аберрационных корректирующих

коэффициентов. Коррекция аберраций оптической системы камеры позволяет повысить точность предсказания местоположения цели, однако ее использование целесообразно в случае малой погрешности телеметрии относительно видеокадров.

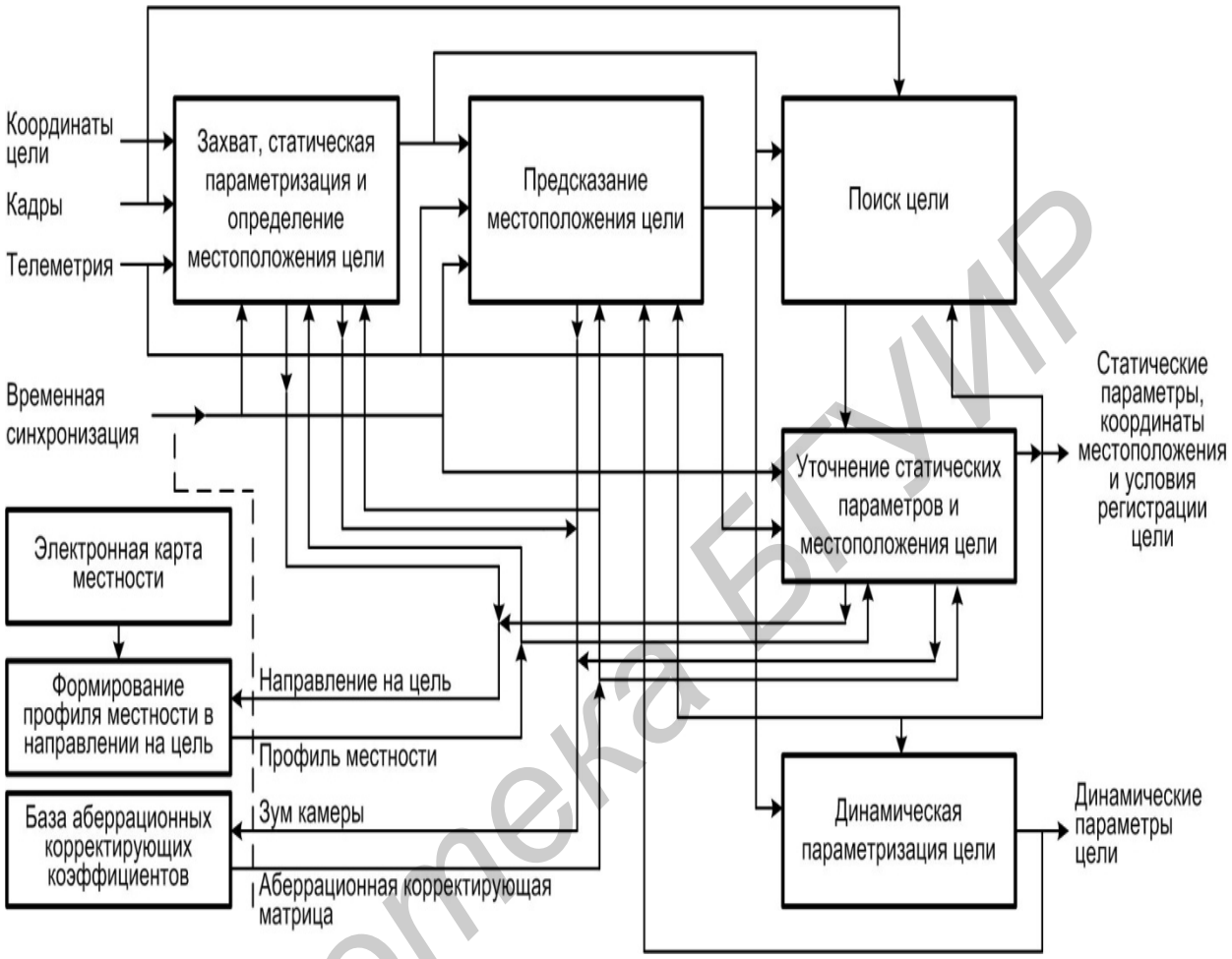


Рисунок 1.5 – Структура автомата сопровождения малоразмерной цели с предсказанием по телеметрии

Телеметрия включает следующие параметры:

- углы крена и тангажа БЛА (угол поворота БЛА, например, относительно направления на север, представляет интерес только для определения реальных координат цели);

- высота полета БЛА (над поверхностью земли);

- углы поворота, крена и тангажа видеокамеры относительно строительных осей БЛА, угловые скорости и ускорения видеокамеры;

- скорости и ускорения БЛА (вдоль строительных осей БЛА для расчета перемещения видеокамеры в пространстве);

- углы видимости видеокамеры (с учетом зума);

- GPS-координаты БЛА.

Данного набора достаточно для сопровождения цели в условиях равнинной местности. При сопровождении цели в условиях холмистой местности на блок захвата и статической параметризации цели и блок уточнения статических параметров цели необходимо подавать профиль местности для корректного определения расстояния от камеры до цели и повышения точности предсказания местоположения цели.

Автомат сопровождения имеет 2 выхода, на которых формируются следующие параметры:

- статические параметры, координаты местоположения, условия регистрации цели и другие параметры (например, расстояние до цели);

- динамические параметры цели.

К статическим параметрам цели относятся:

- цвет (текстурные характеристики) цели;

- площадь цели;

- форма (число и ориентация изломов контура) цели;

- размеры (наибольший и наименьший, которые могут совпадать) цели;

- ориентация цели (вдоль наибольшего размера или отсутствует, если цель круглая);

- дополнительные параметры, характеризующие условия регистрации цели (включают данные телеметрии и временной синхронизации).

К динамическим параметрам относятся скорость, ускорение, направление движения, угловая скорость цели.

1.4.2 Алгоритм захвата и сопровождения малоразмерной цели с предсказанием по телеметрии

Разработан алгоритм захвата и сопровождения малоразмерной цели с предсказанием по телеметрии, который включает две части (рисунок 1.6):

- инициализация, в которой происходит захват, определение координат центра, выделение контура и статическая параметризация цели;

- определение местоположения цели с предсказанием, в которой происходит предсказание и уточнение местоположения цели, определение координат центра, выделение контура, статическая и динамическая параметризация цели.

Вторая часть разделена на две части точкой выхода-входа. Алгоритм имеет два блока загрузки и три блока выгрузки данных.

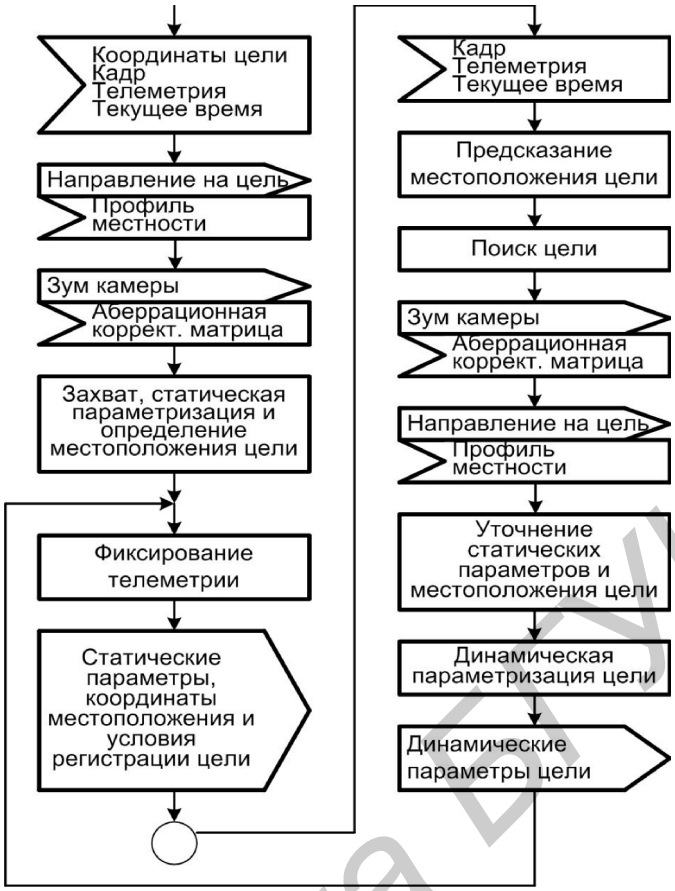


Рисунок 1.6 – Алгоритм сопровождения малоразмерной цели с предсказанием по телеметрии

1.4.3 Детализация структуры автомата сопровождения малоразмерной цели с предсказанием по телеметрии

1.4.3.1 Детализация блока захвата, статической параметризации и определения местоположения цели

Блок захвата, статической параметризации и определения местоположения цели состоит из следующих блоков (рисунок 1.7): нормализация и улучшение качества кадра; сегментация цели; параметризация цели.

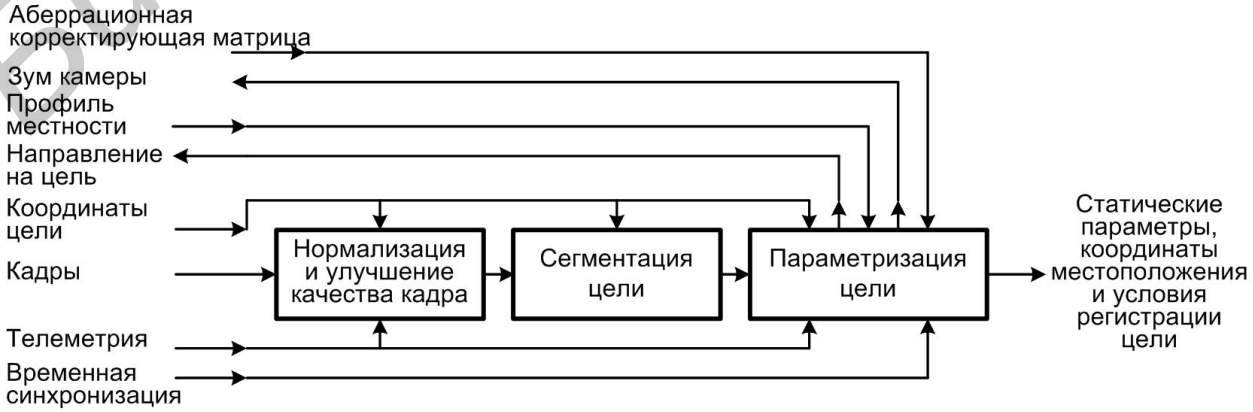


Рисунок 1.7 – Структура блока захвата и статической параметризации цели

В блоке нормализации и улучшения качества кадра (рисунок 1.8) с помощью квадратного окна с центром в пикселе (размер окна зависит от зума камеры), указанном оператором, формируется фрейм, состоящий из двух полуфреймов.

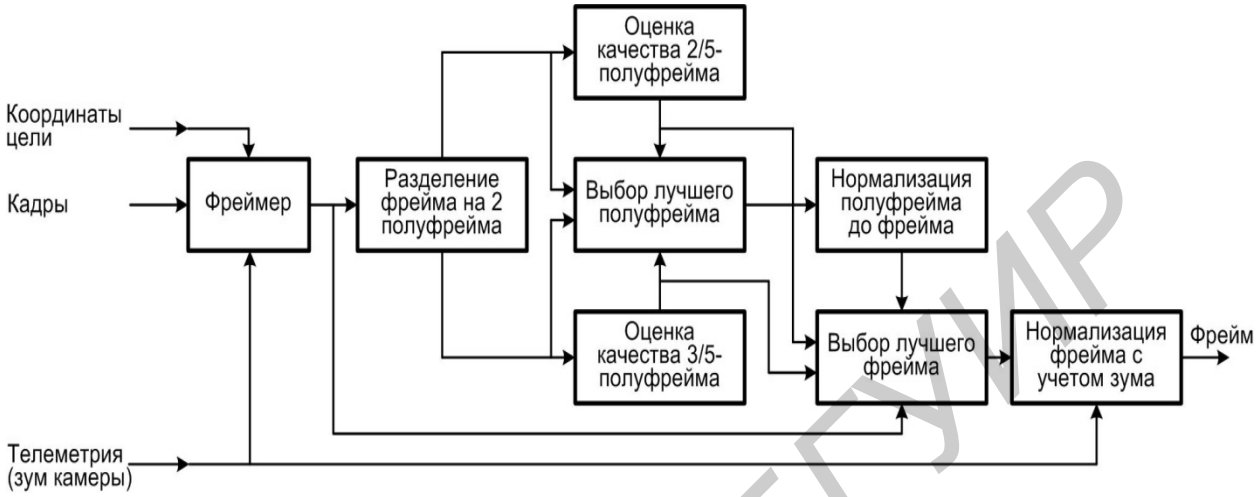


Рисунок 1.8 – Структура блока нормализации и улучшения качества кадра

Полуфреймы разделяются и в результате анализа их качества выбирается лучший полуфрейм, который нормализуется до размера фрейма. Если полуфреймы равны по качеству и синхронизированы, для дальнейшей обработки может использоваться исходный фрейм. Для ускорения данного блока оценка качества и нормализация могут не производится, а всегда может использоваться больший полуфрейм (ускорение примерно в 4-5 раз). В завершение фрейм масштабируется с учетом зума камеры.

В блоке сегментации цели (рисунок 1.9) определяется цвет пикселя, выделенного оператором и осуществляется пороговая обработка фрейма с выделением всех пикселей, цветовые компоненты которых отличаются от цветовых компонент выделенного пикселя на +/- 8 уровней. В результате пороговой обработки формируется бинарное изображение, единичные значения которого, указывают на пиксели, удовлетворяющие условию пороговой обработки. Затем бинарное изображение сегментируется с использованием в качестве точки затравки координат пикселя, выделенного оператором.

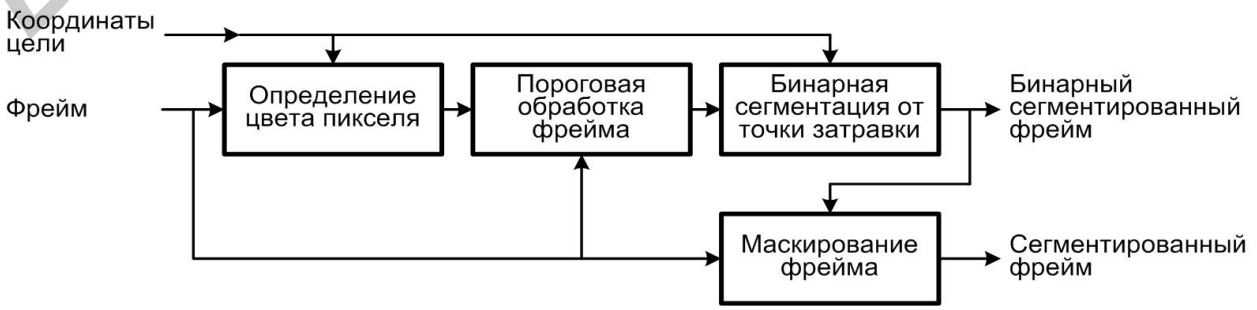


Рисунок 1.9 – Структура блока сегментации цели

В результате формируется бинарный сегментированный фрейм, который используется в качестве маски для формирования сегментированного фрейма. Цель на бинарном сегментированном фрейме представлена единичными пикселями, остальные пиксели равны нулю. Сегментированный фрейм содержит цветную цель на нулевом фоне. Для ускорения данного блока пороговая обработка может быть совмещена с сегментацией (ускорение в несколько раз в зависимости от соотношения площадей цели и фрейма).

В блоке параметризация цели (рисунок 1.10) определяются координаты центра цели, скорректированные координаты центра цели с учетом аберраций оптической системы камеры, статические параметры цели (площадь, размеры и ориентация, параметры характерных точек контура, математическое ожидание и дисперсия цвета по каждой цветовой компоненте), условия регистрации (высота, ориентация, скорости и ускорения по осям БЛА, GPS-координаты БЛА, ориентация, скорости и ускорения по осям, зум камеры), координаты GPS и расстояние до центра цели, время регистрации цели с точностью до микросекунды.

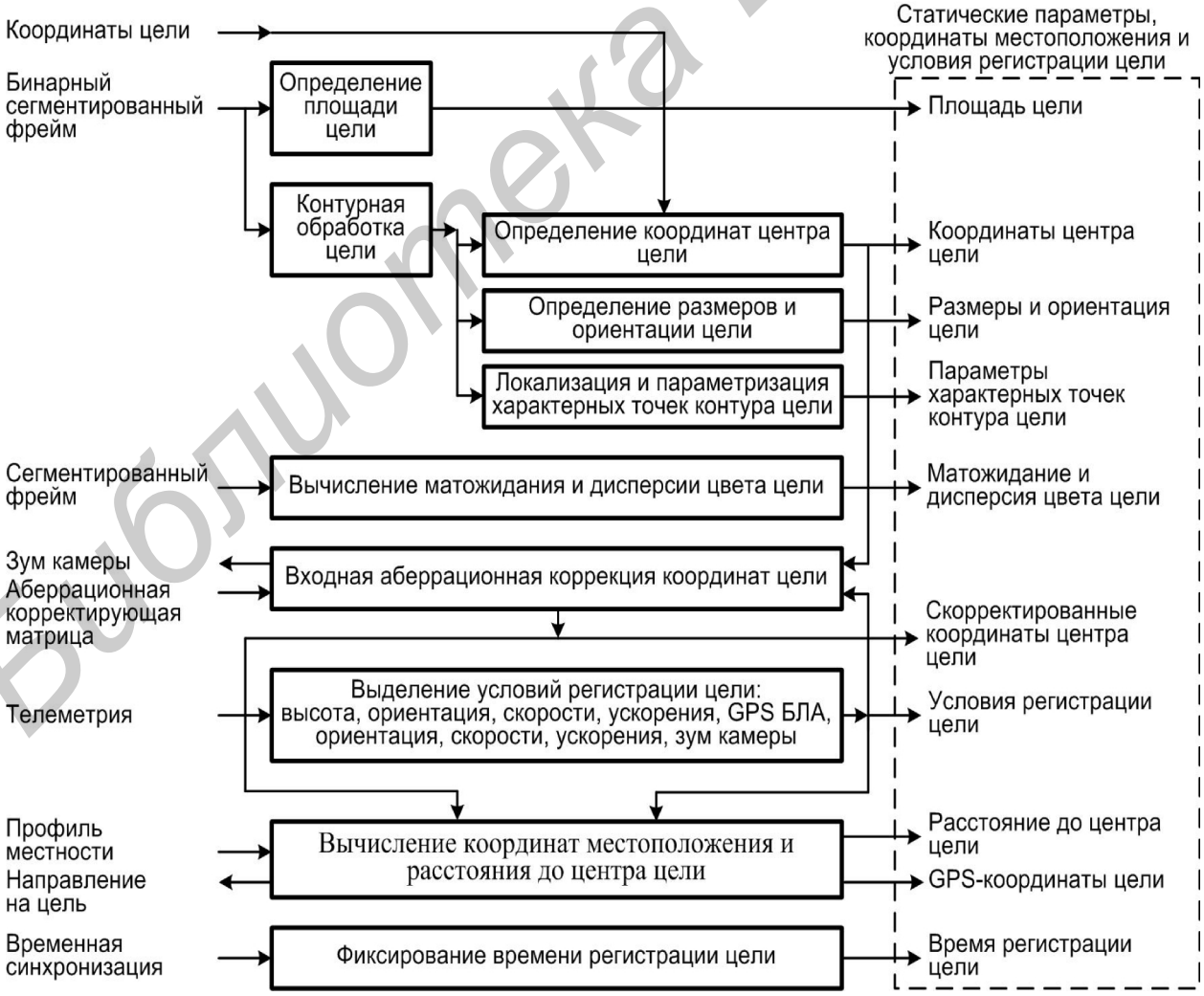


Рисунок 1.10 – Структура блока параметризация цели

Для коррекции аберраций оптической системы камеры используется значение зума (рисунок 1.11).

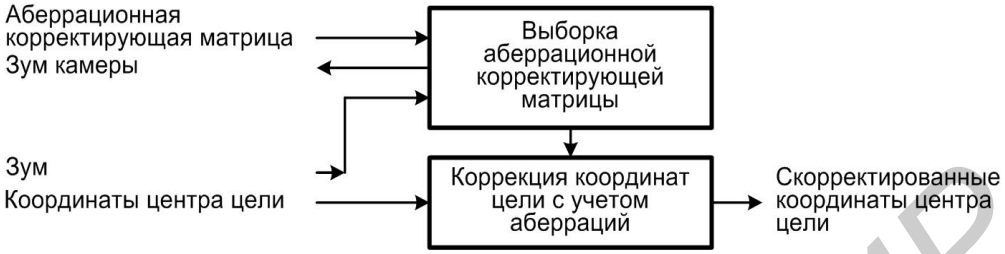


Рисунок 1.11 – Структура блока входной аберрационной коррекции координат цели

GPS-координаты и расстояние до центра цели вычисляются в результате уточнения профиля местности, для чего вычисляется направление на цель от местоположения БЛА на основе скорректированных координат центра цели (рисунок 1.12)

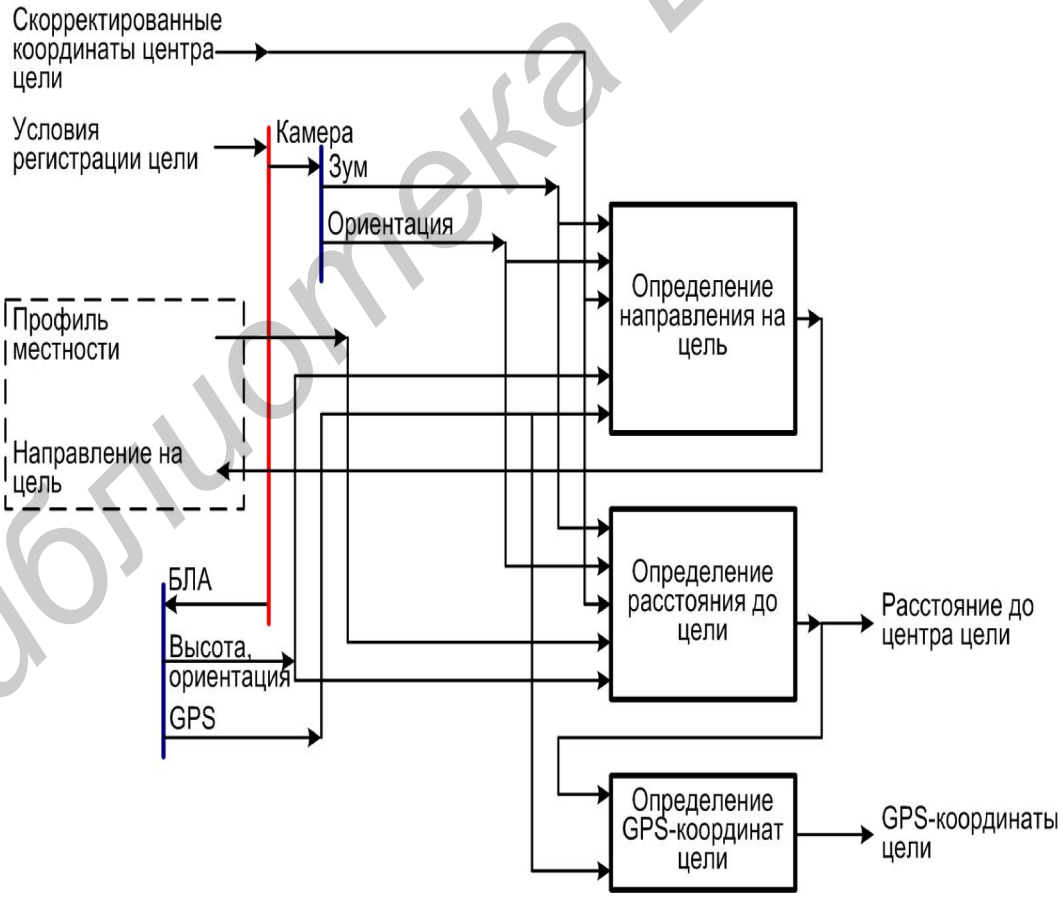


Рисунок 1.12 – Структура блока вычисления координат местоположения и расстояния до центра цели

1.4.3.2 Детализация блока предсказания местоположения цели

Блок предсказания местоположения цели вычисляет наиболее вероятные смещения по осям Y и X центра цели на основе телеметрии и состоит из следующих блоков (рисунок 1.13):

- вычисления интервала предсказания (использует метки времени текущую t и регистрации цели t-1);

- детектирования изменения телеметрии (сопоставляет GPS-координаты БЛА и выбирает исходные для вычисления смещения центра цели на изображении с учетом перемещения БЛА, камеры и самой цели на основе телеметрии в момент времени t-1 или t);

- предсказания углового перемещения камеры (использует параметры камеры в момент времени t-1 для прогнозирования положения камеры в момент времени t);

- предсказания перемещения и ориентации БЛА (использует параметры БЛА в момент времени t-1 для прогнозирования положения и ориентации БЛА в момент времени t);

- предсказания перемещения цели (использует динамические характеристики цели в момент времени t-1 для вычисления ее перемещения в момент времени t);

- вычисления смещения центра цели из-за углового перемещения камеры (использует реальные или прогнозные данные об ориентации камеры в момент времени t);

- вычисления смещения центра цели из-за перемещения и ориентации БЛА (использует скорректированные координаты и расстояние до цели в момент времени t-1, реальные или прогнозные данные о перемещении и ориентации БЛА в момент времени t);

- вычисления смещения центра цели из-за собственного перемещения (использует скорректированные координаты и расстояние до цели в момент времени t-1, данные о перемещении цели в момент времени t);

- суммирования перемещений (использует скорректированные координаты центра цели в момент времени t-1 и предсказанные значения перемещений цели в момент времени t для формирования координат возможного положения центра цели на изображении);

- масштабирования с учетом зума (использует значения зума в моменты времени t-1 или t для коррекции предсказанных координат центра цели);

- выходной аберрационной коррекции координат цели (использует аберрационную корректирующую матрицу для вычисления корректных координат положения центра цели с учетом аберраций оптической системы).

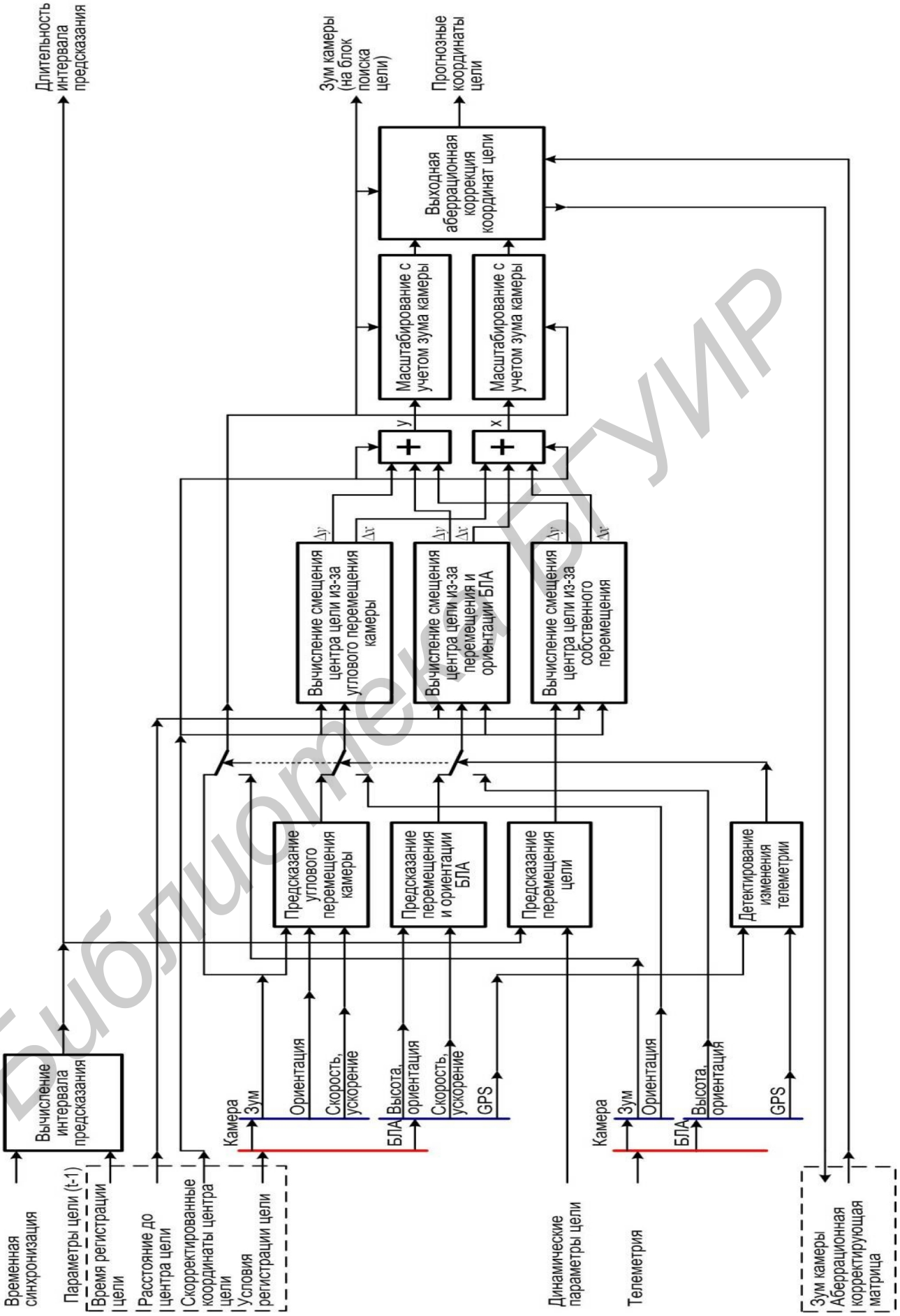


Рисунок 1.13 – Структура блока предсказания местоположения цели

1.4.3.3 Детализация блока поиска цели

Блок поиска цели состоит из следующих блоков (рисунок 1.14): нормализации и улучшения качества кадра; сегментации фрейма; параметризации сегментов; идентификации цели.

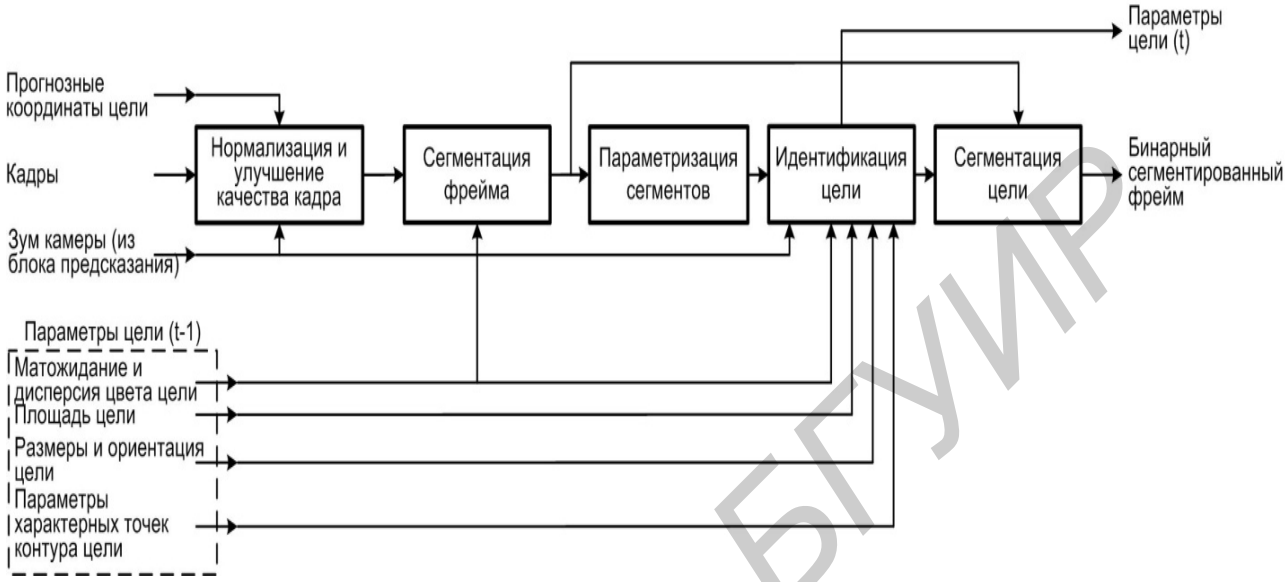


Рисунок 1.14 – Структура блока поиска цели

Блок нормализации и улучшения качества кадра полностью соответствует аналогичному блоку в блоке захвата и параметризации цели. Разница состоит в том, что на входы данного блока поступают прогнозные координаты цели и информация о зуме камеры с блока предсказания.

Блок сегментации фрейма отличается от аналогичного блока в блоке захвата и параметризации цели тем, что выделяет все области относительно заданного порога и присваивает им индексы, в результате чего на выходе формируется индексированный сегментированный фрейм (рисунок 1.15). Данный фрейм бинаризируется для формирования маски, с помощью которой формируется сегментированный фрейм, выделяющий потенциальные цели.

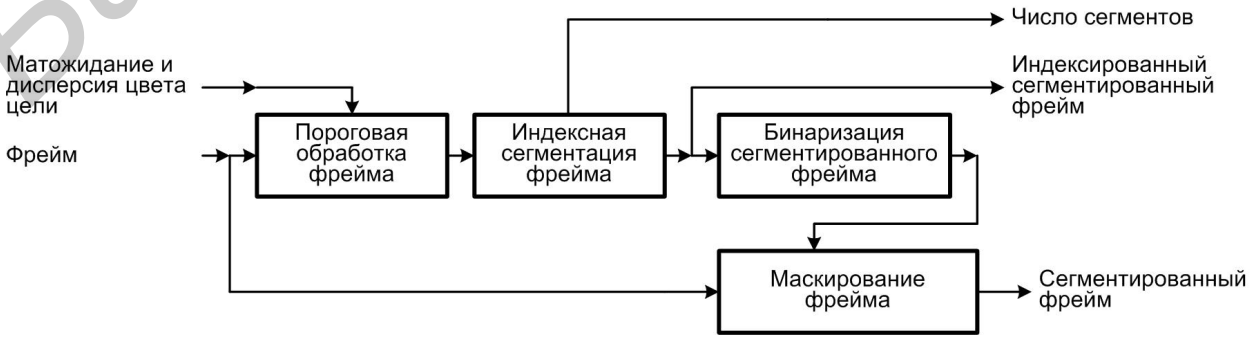


Рисунок 1.15 – Структура блока сегментации фрейма

В блоке параметризации сегментов формируются основные параметры (площадь, размеры и ориентация, параметры характерных точек контура) сегментов (рисунок 1.16).

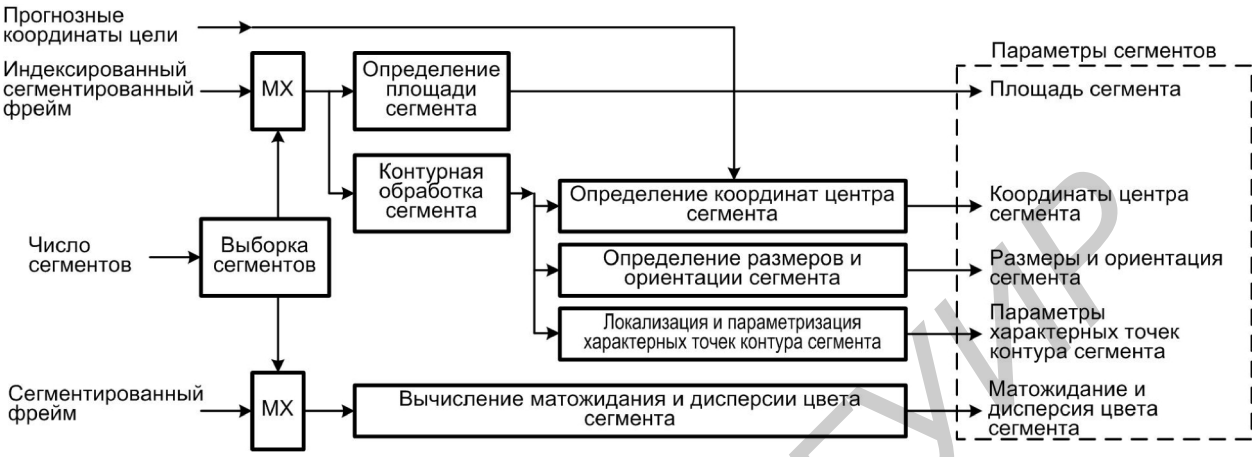


Рисунок 1.16 – Структура блока параметризации сегментов

В блоке идентификации цели осуществляется сопоставление статических параметров цели (площадь, размеры и ориентация, параметры характерных точек контура), зарегистрированных в момент времени t-1, и параметров сегментов (рисунок 1.17). В результате выделяется наиболее вероятный кандидат-сегмент (индекс сегмента), а также его параметры. На первом этапе идентификации приоритет при идентификации принадлежит площади и цвету. Затем используются остальные параметры, если на первом этапе обнаружено несколько сегментов-кандидатов. Последний критерий – удаленность от прогнозных координат центра цели (т.е. относительно условных нулевых координат).

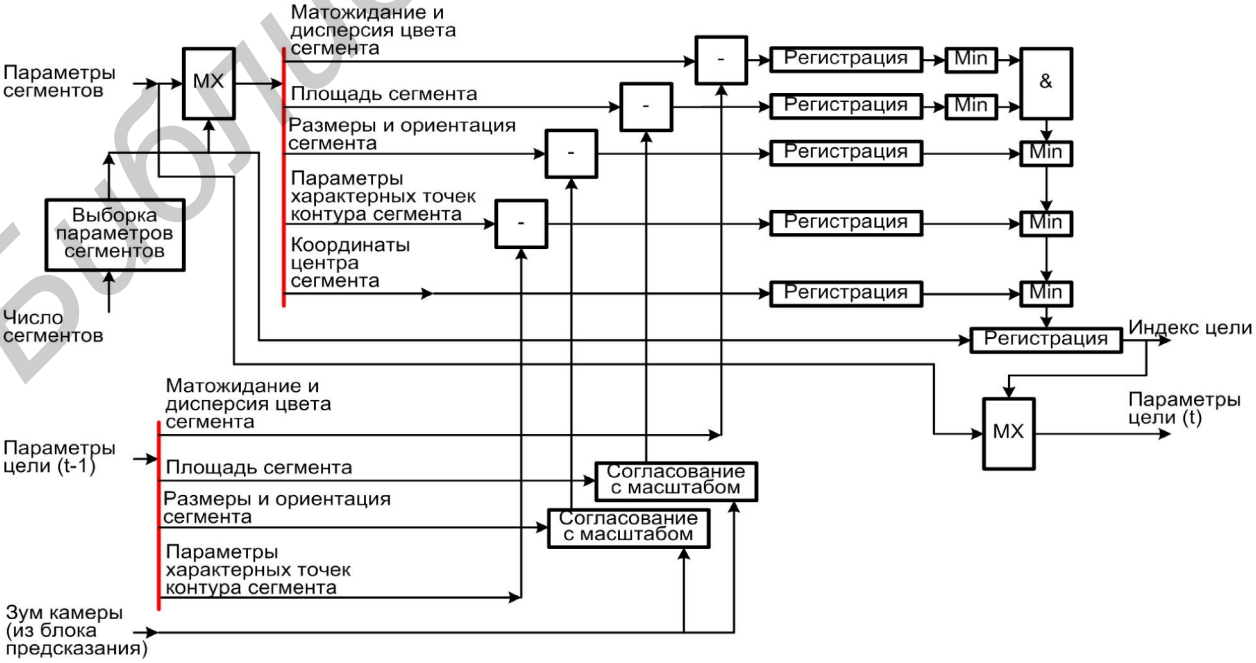


Рисунок 1.17 – Структура блока идентификации цели

В блоке сегментации цели формируется бинарный сегментированный фрейм (рисунок 1.18), который может использоваться для визуализации (подкрашивание цели, например).

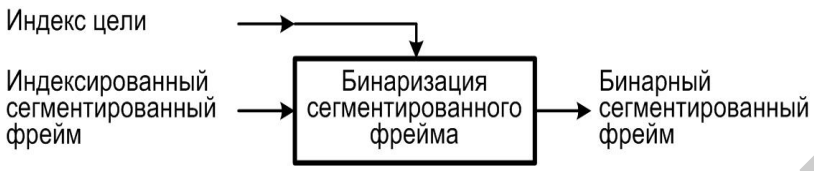


Рисунок 1.18 – Структура блока сегментации цели

Для ускорения поиска целесообразно производить пороговую обработку, параметризацию и идентификацию одновременно, начиная от центра фрейма (т.е. от координат прогнозного центра цели) и прекращать поиск при обнаружении первого совпадения (выигрыш в скорости обработки может составить 4-5 раз).

Для повышения вероятности и точности обнаружения при в качестве дополнительных параметра идентификации может быть использовано расстояние до центра сегмента, что потребует определения расстояний до центров всех сегментов и коррекции координат с учетом аберраций оптической системы камеры. Т.е. фактически полной параметризации всех сегментов. Это может быть реализовано только при наличии соответствующих вычислительных ресурсов.

1.4.3.4 Детализация блока уточнения статических параметров и местоположения цели

Блок уточнения статических параметров и местоположения цели полностью повторяет нижнюю часть блока и функционирует аналогично, вычисляя и сохраняя новые статические параметры, координаты местоположения и условия регистрации цели (рисунок 1.19).

Он использует статические параметры цели, вычисленные в блоке поиска цели. Если поиск цели заканчивается неудачно (цель не обнаружена), блок уточнения статических параметров и местоположения цели не отрабатывает и остаются в силе прежние параметры цели. Относительно них осуществляется предсказание, но уже с другим временным интервалом и, возможно, обновленной телеметрией.

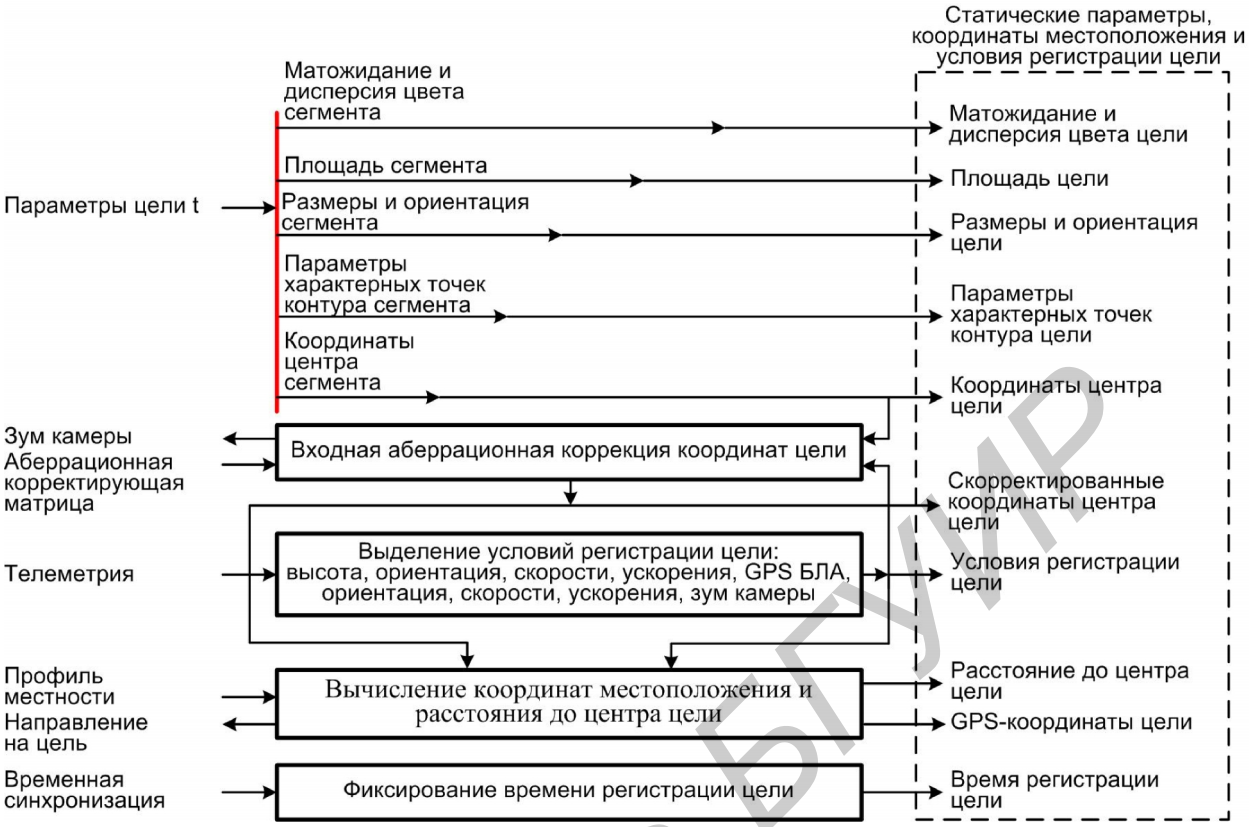


Рисунок 1.19 – Структура блока уточнения статических параметров и местоположения цели

1.3.5 Детализация блока динамической параметризации цели

В блоке динамической параметризации цели вычисляются направление движения цели, скорость и ускорение в направлении движения цели и угловая скорость, которые подаются на выход автомата сопровождения и на вход блока предсказания местоположения цели (рисунок 1.20). На входы блока динамической параметризации цели подаются статические параметры, координаты, время и условия регистрации цели на тактах t, t-1 и t-2.

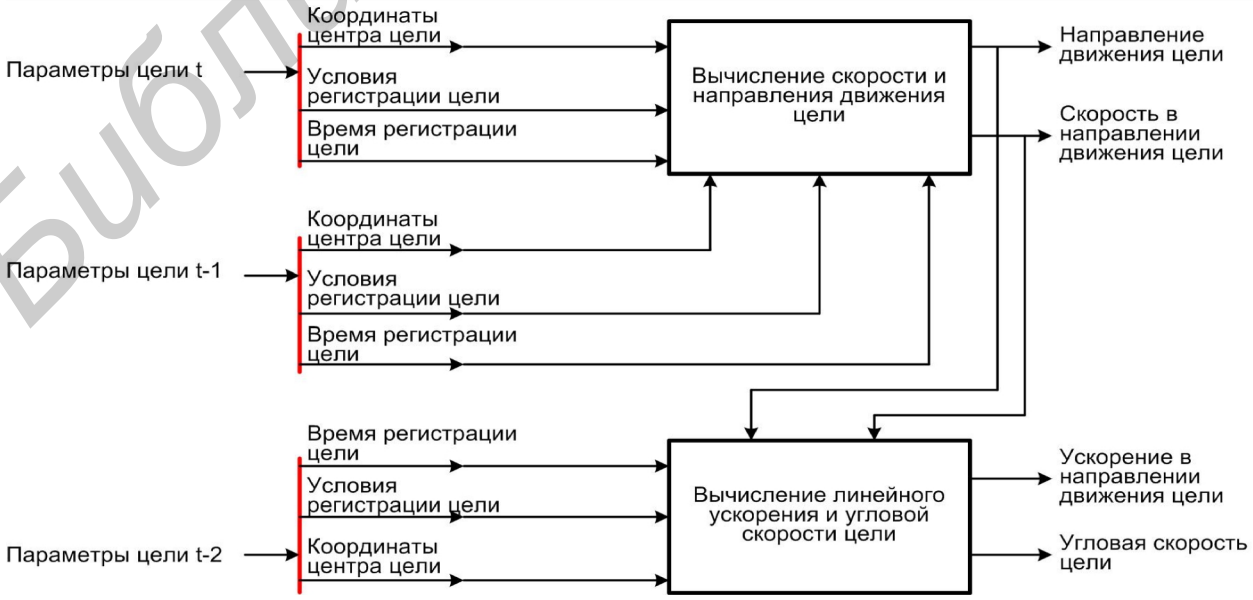


Рисунок 1.20 – Структура блока динамической параметризации цели

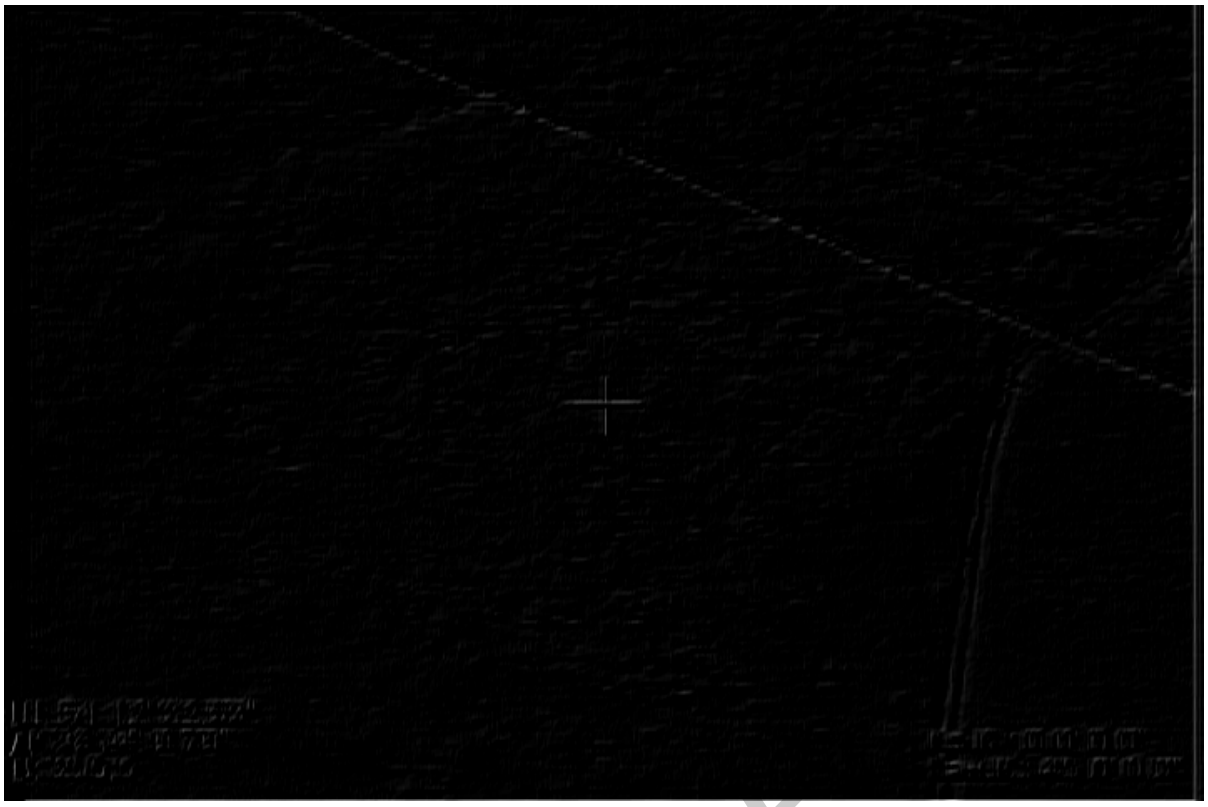


Таблица 2.1 – Оценка резкости полукадров

Полукадр	Тип фильтра			
	Простой	Собеля	Превитта	Робертса
А	1,5593	0,7639	0,7670	2,3659
Б	1,3365	0,7501	0,7525	2,0605

Как следует из таблицы 2.1, полукадр А более контрастный, чем полукадр Б.

2.2 Методы, алгоритмы и программные средства выделения объекта наблюдения, отмеченного оператором, на кадре видеопотока

2.2.1 Метод прогрессивной сегментации

Выделение цели основано на сегментации изображения. Сегментация широко используется для решения задач цифровой обработки изображений в картографировании, видеонаблюдении, распознавании и других областях. Известен ряд методов сегментации, основанных на разделении и слиянии областей, которые достаточно широко распространены благодаря относительной простоте реализации [8, 9]. Недостатком данных методов является возникновение ошибок сегментации сложных по структуре областей пикселей и отсутствие адаптации к ограничениям вычислительных ресурсов и времени вычислений.

Для устранения данных недостатков разработан метод прогрессивной сегментации изображений на основе реверсивной кластеризации (Reversible Cluster Progressive Segmentation – RCPS) [10]. Сущность метода состоит в древовидной кластеризации однородных по яркости областей пикселей и формировании множества кратномасштабных кластерных образов исходного изображения (прямая кластеризация); присвоении номеров кластеризованным однородным областям на всех уровнях кратномасштабного представления исходного изображения (обратная кластеризация); объединении соседних однородных по яркости кластеризованных областей на каждом уровне кратномасштабного представления исходного изображения (уточнение номеров сегментов).

В результате прямой кластеризации формируются иерархические множества матриц аппроксимации (определяют среднюю яркость кластера) и кластеризации (определяют однородность элементов кластера). Матрицы каждого иерархического уровня соответствуют некоторому кратномасштабному кластерному представлению исходного изображения. Это позволяет реализовать на следующем этапе прогрессивную сегментацию, выделяющую однородные области сначала для крупномасштабного представления изображения, а затем

постепенно уточняющую границы однородных областей на нижних по иерархии мелкомасштабных представлениях изображения. Коэффициент масштабирования по вертикали и горизонтали – два (выбирается исходя из размера кластера 2×2).

При обратной кластеризации формируется иерархическое множество матриц сегментации, элементы которых получают номера соответствующих им сегментов в результате формирования новых и наращивания существующих однородных по яркости областей. Кроме этого, выявляются соседние однородные области, имеющие одинаковую среднюю яркость, но различные номера сегментов, для их последующего слияния. Обратная кластеризация начинается с самого крупномасштабного представления изображения и распространяется на некоторое число уровней кратномасштабного представления изображения, определяемое допустимым временем обработки.

В результате уточнения номеров сегментов формируется результирующая матрица сегментации, размер которой совпадает с размером исходного изображения или кратен ему. Каждый элемент результирующей матрицы сегментации представляет некоторый пиксель изображения или его кратномасштабного представления и имеет в качестве значения номер соответствующего сегмента.

Метод позволяет устранить ошибки сегментации сложных по структуре областей за счет выявления соседних однородных областей, имеющих одинаковую среднюю яркость, но различные номера сегментов. Метод обеспечивает адаптацию к ограничениям вычислительных ресурсов и времени вычислений за счет сегментации изображения на каждом уровне его кратномасштабного представления.

2.2.2 Алгоритм прямой кластеризации

Алгоритм прямой кластеризации состоит из следующих шагов.

1) Формирование множества $\{A(l)\}_{(l=\overline{0,L})}$ матриц $A(l) = \left\| a^{(l)}(y,x) \right\|_{\left(y=\overline{0,Y/2^l-1},\, x=\overline{0,X/2^l-1}\right)}$ аппроксимации и инициализация элементов матрицы $A(0)$ аппроксимации 0-го уровня в соответствии с выражением $a^{(0)}(y,x) \Leftarrow p(y,x)$ при $y = \overline{0,Y-1}$, $x = \overline{0,X-1}$, где $\Leftarrow$ – операция присваивания; $p(y,x)$ – пиксель сегментируемого изображения $P = \left\| p(y,x) \right\|_{(y=\overline{0,Y-1},\, x=\overline{0,X-1})}$; $Y = 2^{f_Y}$, $X = 2^{f_X}$ – размеры сегментируемого изображения P; $f_Y > 0$, $f_X > 0$ – целые; $l = \overline{0,L}$ – номер итерации (уровня) сегментации; $L = \min(f_Y, f_X)$ – число итераций, определяемое минимальным из значений f_Y и f_X. В результате в качестве аппроксимированного образа $A(0)$ используется сегментируемое изображение P.

2) Формирование множества $\{C(l)\}_{(l=\overline{0,L})}$ матриц $C(l)=\left\|c^{(l)}(y,x)\right\|_{\left(y=\overline{0,Y/2^l-1},x=\overline{0,X/2^l-1}\right)}$ кластеризации и инициализация элементов матрицы $C(0)$ кластеризации 0-го уровня в соответствии с выражением $c^{(0)}(y,x)\Leftarrow 0$ при $y=\overline{0,Y-1}$, $x=\overline{0,X-1}$. В результате матрица $C(0)$ кластеризации нулевого уровня определяется нулевой.

3) Инициализация счетчика l циклов согласно выражению $l\Leftarrow 1$.

4) Начало цикла кластеризации. Формирование матрицы $C(l)$ кластеризации l-го уровня, элементы которой вычисляются с помощью выражений

$$\forall\left(j=\overline{0,1}\right)\forall\left(i=\overline{0,1}\right)\left(a^{(l-1)}(2y+j,2x+i)=a^{(l)}(y,x)\right)\wedge\left(c^{(l-1)}(2y+j,2x+i)=0\right)\rightarrow\left(c^{(l)}(y,x)\Leftarrow 0\right), \quad (2.1)$$

$$\exists\left(j=\overline{0,1}\right)\exists\left(i=\overline{0,1}\right)\left(a^{(l-1)}(2y+j,2x+i)\neq a^{(l)}(y,x)\right)\vee\left(c^{(l-1)}(2y+j,2x+i)=1\right)\rightarrow\left(c^{(l)}(y,x)\Leftarrow 1\right) \quad (2.2)$$

при $y=\overline{0,Y/2^l-1}$, $x=\overline{0,X/2^l-1}$,

где $a^{(l)}(y,x)=\frac{1}{4}\sum_{j=0}^{1}\sum_{i=0}^{1}a^{(l-1)}(2y+j,2x+i)$ – среднее арифметическое элементов кластера с координатами $(2y,2x)$ в матрице $A(l-1)$ аппроксимации нижнего $(l-1)$-го уровня.

В результате нулевым кластерам $\left\{c^{(l-1)}(2y+j,2x+i)\right\}_{(j=\overline{0,1},i=\overline{0,1})}$ матрицы $C(l-1)$ кластеризации $(l-1)$-го уровня и соответствующим им однородным по значениям кластерам $\left\{a^{(l-1)}(2y+j,2x+i)\right\}_{(j=\overline{0,1},i=\overline{0,1})}$ матрицы $A(l-1)$ аппроксимации $(l-1)$-го уровня ставятся в соответствие нулевые элементы $c^{(l)}(y,x)$ матрицы $C(l)$. Ненулевым кластерам $\left\{c^{(l-1)}(2y+j,2x+i)\right\}_{(j=\overline{0,1},i=\overline{0,1})}$ матрицы $C(l-1)$, а также нулевым кластерам $\left\{c^{(l-1)}(2y+j,2x+i)\right\}_{(j=\overline{0,1},i=\overline{0,1})}$ матрицы $C(l-1)$, имеющим неоднородные по значениям соответствующие кластеры $\left\{a^{(l-1)}(2y+j,2x+i)\right\}_{(j=\overline{0,1},i=\overline{0,1})}$ матрицы $A(l-1)$ аппроксимации, ставятся в соответствие единичные элементы $c^{(l)}(y,x)$ матрицы $C(l)$. Формируемые таким образом за L циклов L-уровневые нуль-деревья описывают расположение однородных кластерных областей во множестве $\{A(l)\}_{(l=\overline{0,L})}$.

5) Приращение счетчика циклов согласно выражению $l\Leftarrow l+1$.

6) Окончание цикла кластеризации. Проверка условия $l\le L$. Если оно выполняется – переход на шаг 4, иначе – выход из цикла и завершение алгоритма.

2.2.3 Алгоритм обратной кластеризации

Алгоритм обратной кластеризации состоит из следующих шагов.

1) Формирование множества $\{S(l)\}_{(l=\overline{0,L})}$ матриц $S(l)=\left\|s^{(l)}(y,x)\right\|_{\left(y=\overline{0,Y/2^l-1},\,x=\overline{0,X/2^l-1}\right)}$ сегментации и инициализация элементов матриц $S(l)$ сегментации уровней $\overline{0,L}$ в соответствии с выражением $s^{(l)}(y,x)\Leftarrow 0$ при $\overline{l=0,L}$, $y=\overline{0,Y/2^l-1}$, $x=\overline{0,X/2^l-1}$. В результате выполнения данного шага матрицы $S(l)$ сегментации уровней $\overline{0,L}$ определяются нулевыми.

2) Инициализация счетчика N_A однородных областей согласно выражению $N_A\Leftarrow 1$.

3) Инициализация матрицы $S(L)=\left\|s^{(L)}(y,x)\right\|_{(y=0,x=0)}$ сегментации L-го уровня, значение единственного элемента которой (вершины дерева сегментации) вычисляется с помощью выражения

$$\left(c^{(L)}(y,x)=0\right)\rightarrow\left(s^{(L)}(y,x)\Leftarrow N_A,\,N_A\Leftarrow N_A+1\right) \tag{2.3}$$

при $y=\overline{0,Y/2^L-1}$, $x=\overline{0,X/2^L-1}$.

Из выражения (2.3) следует, что для однородного изображения $s^{(L)}(y,x)=0$, а для неоднородного $s^{(L)}(y,x)=1$.

4) Инициализация счетчика l циклов согласно выражению $l=L$.

5) Начало цикла прогрессивной сегментации. Формирование значений элементов матрицы $S(l-1)$ сегментации $(l-1)$-го уровня с помощью выражения (масштабирование областей)

$$\left(c^{(l)}(y,x)=0\right)\rightarrow\left(s^{(l-1)}(2y+j,2x+i)\Leftarrow s^{(l)}(y,x)\right) \tag{2.4}$$

при $y=\overline{0,Y/2^l-1}$, $x=\overline{0,X/2^l-1}$, $j=\overline{0,1}$, $i=\overline{0,1}$.

В результате формируется $(l-1)$-й уровень для кластерных нуль-деревьев.

В общем случае возможны четыре комбинации соответствующих значений матрицы кластеризации на l–м и $(l-1)$–м уровнях. Для каждой из них предполагается следующая обработка на $(l-1)$–м уровне: (0,0) – масштабирование области (кластер $(l-1)$–го уровня наследует номер сегмента соответствующего элемента l–го уровня); (1,0) – формирование нового сегмента (элемент $(l-1)$–го уровня получает новый номер сегмента) или присоединение к существующему соседнему сегменту (элемент $(l-1)$–го уровня получает номер соседнего элемента $(l-1)$–го уровня); (1,1) – не обрабатывается; (0,1) – невозможная комбинация.

6) Формирование новых областей (разделение областей) согласно выражению

$$\left(c^{(l)}(y,x)=1\right)\wedge\left(c^{(l-1)}(2y+j,2x+i)=0\right)\wedge$$
$$\wedge\neg\exists(k\in[-1,1])\neg\exists(m\in[-1,1])\begin{pmatrix}a^{(l-1)}(2y+j,2x+i)=a^{(l-1)}(2y+j+k,2x+i+m)\wedge\\ \wedge s^{(l-1)}(2y+j+k,2x+i+m)\neq 0\end{pmatrix}\rightarrow$$
$$\rightarrow\left(s^{(l-1)}(2y+j,2x+i)\Leftarrow N_A, N_A\Leftarrow N_A+1\right) \quad (2.5)$$

при $y=\overline{0,Y/2^l-1}, x=\overline{0,X/2^l-1}$, $j=\overline{0,1}, i=\overline{0,1}, k=\overline{-1,1}, m=\overline{-1,1}$, $k+m\neq 0$.

7) Наращивание областей путем их присоединения к существующим однородным областям согласно выражению

$$\left(c^{(l)}(y,x)=1\right)\wedge\left(c^{(l-1)}(2y+j,2x+i)=0\right)\wedge\left(s^{(l-1)}(2y+j,2x+i)=0\right)\wedge$$
$$\wedge\exists(k\in[-1,1])\exists(l\in[-1,1])\begin{pmatrix}a^{(l-1)}(2y+j,2x+i)=a^{(l-1)}(2y+j+k,2x+i+m)\\ \wedge s^{(l-1)}(2y+j+k,2x+i+m)\neq 0\end{pmatrix}$$
$$\rightarrow\left(s^{(l-1)}(2y+j,2x+i)\Leftarrow s^{(l-1)}(2y+j+k,2x+i+m)\right) \quad (2.6)$$

при $y=\overline{0,Y/2^l-1}, x=\overline{0,X/2^l-1}$, $j=\overline{0,1}, i=\overline{0,1}, k=\overline{-1,1}, m=\overline{-1,1}$, $k+m\neq 0$.

8) Инициализация матриц $N_B=\|n_B(p,q)\|_{(p=\overline{0,N_A-1},q=\overline{0,M_A-1})}$ номеров и количества $V_B=\|v_B(p)\|_{(p=\overline{0,N_A-1})}$ смежных одинаковых областей согласно выражениям $n_B(p,q)\Leftarrow 0$, $v_B(p)\Leftarrow 0$ при $p=\overline{0,N_A-1}, q=\overline{0,M_A-1}$, где M_A – максимальное число смежных одинаковых областей.

9) Слияние однородных областей согласно выражению

$$\left(c^{(l)}(y,x)=1\right)\wedge\left(c^{(l-1)}(2y+j,2x+i)=0\right)\wedge\left(s^{(l-1)}(2y+j,2x+i)\neq 0\right)\wedge$$
$$\wedge\exists(k\in[-1,1])\exists(l\in[-1,1])\begin{pmatrix}a^{(l-1)}(2y+j,2x+i)=a^{(l-1)}(2y+j+k,2x+i+m)\wedge\\ \wedge s^{(l-1)}(2y+j+k,2x+i+m)\neq 0\wedge\\ \wedge s^{(l-1)}(2y+j+k,2x+i+m)\neq s^{(l-1)}(2y+j,2x+i)\wedge\\ \wedge\neg\exists(q\in[0,M_A])n_B\left(s^{(l-1)}(2y+j,2x+i),q\right)=\\ =s^{(l-1)}(2y+j+k,2x+i+m)\end{pmatrix}$$
$$\rightarrow\begin{pmatrix}n_B\left(s^{(l-1)}(2y+j,2x+i),v_B\left(s^{(l-1)}(2y+j,2x+i)\right)\right)\Leftarrow s^{(l-1)}(2y+j+k,2x+i+\\ v_B\left(s^{(l-1)}(2y+j,2x+i)\right)\Leftarrow v_B\left(s^{(l-1)}(2y+j,2x+i)\right)+1,\\ n_B\left(s^{(l-1)}(2y+j+k,2x+i+m),v_B\left(s^{(l-1)}(2y+j+k,2x+i+m)\right)\right)\Leftarrow s^{(l-1)}(2y+j\\ v_B\left(s^{(l-1)}(2y+j+k,2x+i+m)\right)\Leftarrow v_B\left(s^{(l-1)}(2y+j+k,2x+i+m)\right)+1\end{pmatrix} \quad (2.7)$$

при $y=\overline{0,Y/2^l-1}$, $x=\overline{0,X/2^l-1}$, $j=\overline{0,1}$, $i=\overline{0,1}, k=\overline{-1,1}$, $m=\overline{-1,1}$, $k+m\neq 0$.

10) Уменьшение счетчика циклов согласно выражению $l\Leftarrow l-1$.

11) Окончание цикла прогрессивной сегментации. Проверка условия $l > 0$. Если оно выполняется – переход на шаг 5, иначе – выход из цикла и завершение алгоритма.

2.2.4 Алгоритм уточнения номеров сегментов

Алгоритм уточнения номеров сегментов состоит из следующих шагов.

1) Формирование матриц $N_X = \|n_X(p,q)\|_{(p=\overline{0,N_A-1},q=\overline{0,M_A-1})}$ номеров, $N_C = \|n_C(p)\|_{(p=\overline{0,N_A-1})}$ замены номеров и $V_X = \|v_X(p)\|_{(p=\overline{0,N_A-1})}$ количества объединяемых областей.

2) Инициализация счетчика сегментов $N_S \Leftarrow 0$.

3) Формирование номеров изолированных однородных областей согласно выражению

$$\exists(p \in [0, N_A])(v_B(p) = 0) \to ((n_X(N_S, 0) \Leftarrow p), (n_C(p) \Leftarrow N_S), (N_S \Leftarrow N_S + 1)) \quad (2.8)$$

при $p = \overline{0, N_A - 1}$.

4) Инициализация счетчика циклов объединения областей $p \Leftarrow 0$.

5) Начало цикла объединения областей. Проверка условия $(v_B(p) = 0)$. Если условие выполняется, то переход на шаг 12.

6) Определение номера первой объединяемой области согласно выражению

$$(v_B(p) > 0) \to ((n_X(N_S, 0) \Leftarrow p), (v_X(N_S) \Leftarrow 1)). \quad (2.9)$$

7) Инициализация указателя стека связанных номеров областей $s \Leftarrow 0$.

8) Обработка стека согласно выражениям

$$\neg\exists(t \in [0, v_X(N_S) - 1])(n_X(N_S, t) = n_B(n_X(N_S, s), q)) \to$$
$$\to (n_X(N_S, v_X(N_S) + q) \Leftarrow n_B(n_X(N_S, s), q), (v_X(N_S) \Leftarrow v_X(N_S) + 1) \quad (2.10)$$

при $q = \overline{0, v_B(n_X(N_S, s)) - 1}$,

$$v_B(n_X(N_S, s)) \Leftarrow 0, \quad (2.11)$$

$$n_C(n_X(N_S, s)) \Leftarrow N_S. \quad (2.12)$$

9) Приращение указателя стека связанных номеров областей $s \Leftarrow s + 1$.

10) Проверка условия окончания обработки стека $s < v_X(N_S)$. Если условие выполняется – переход на шаг 8. Иначе – переход на следующий шаг.

11) Приращение счетчика сегментов $N_S \Leftarrow N_S + 1$.

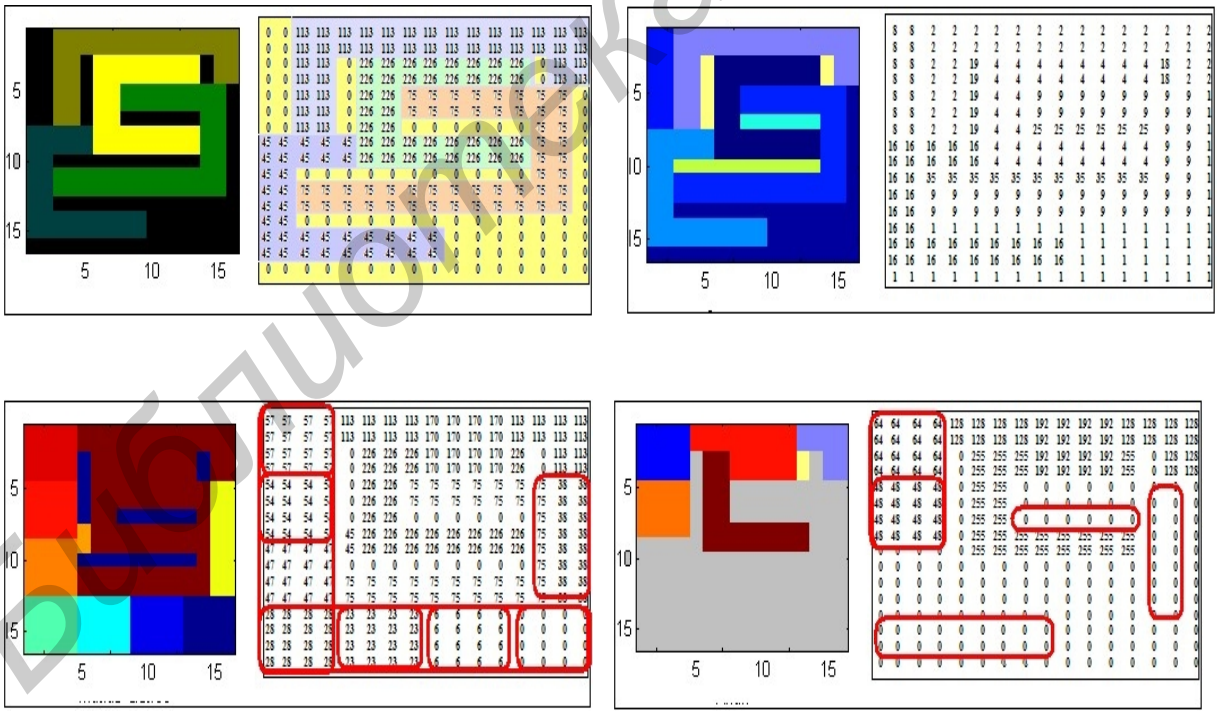
12) Приращение счетчика циклов объединения областей $p \Leftarrow p + 1$.

13) Проверка условия окончания цикла объединения областей $p < N_A$. Если условие выполняется – переход на шаг 5, иначе – переход на следующий шаг.

$$S_R = \left\| s_R(y,x) \right\|_{(y=\overline{0,Y-1}, x=\overline{0,X-1})}$$

$$s_R(y,x) \Leftarrow n_C\left(s^{(0)}(y,x)\right)$$

$y = \overline{0, Y-1} \quad x = \overline{0, X-1}$



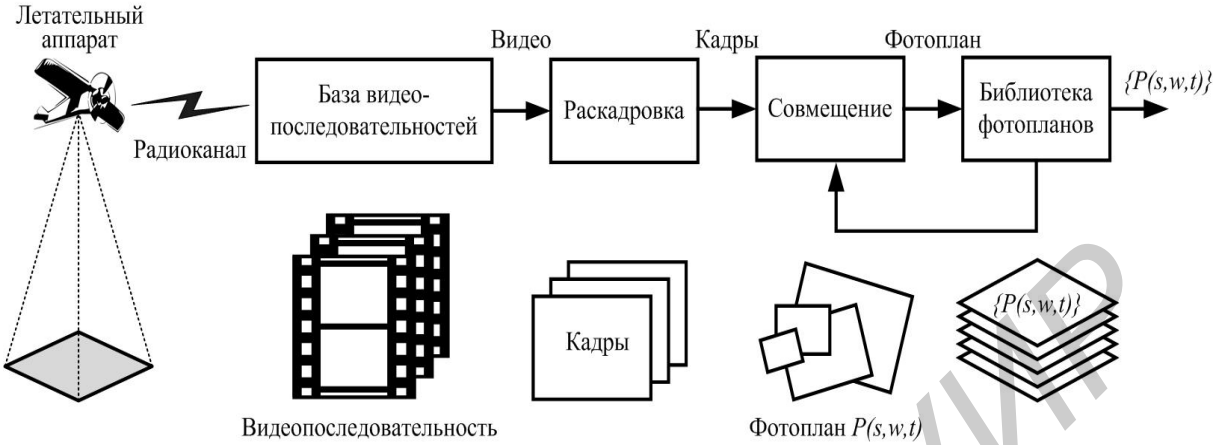
При выделении цели, отмеченной оператором, алгоритм сегментации существенно упрощается за счет определения начальной точки сегментации и обработки только одного сегмента, соответствующего выделенной цели. Это позволяет реализовать относительно простое программное средство выделения цели на основе упрощенной сегментации (приложение Г), функционирующее в реальном масштабе времени.

2.3 Возможности формирования и предварительной обработки фрагментов фотоплана, соответствующих маршруту полета БЛА. Алгоритмы и программные средства выделения фрагмента фотоплана по заданным координатам

Для совмещения изображений с борта БЛА с фотопланом предлагается метод кадровой компенсации движения видеокамеры [12]. Сущность метода состоит в поиске фрагмента фотоплана, соответствующего прогнозируемому кадру, и использовании координат этого фрагмента и коэффициентов гомографии для формирования прогнозного кадра, замещающего прогнозируемый кадр.

Алгоритм кадровой компенсации движения видеокамеры состоит из следующих шагов.

1) Формирование библиотеки фотопланов. Библиотека $\{P(s,w,t)\}_{(s=\overline{0,S-1},w=\overline{0,W-1},t=\overline{0,T-1})}$ фотопланов формируется на основе кадров видеопоследовательностей, полученных ранее с борта летательного аппарата, при положении оптической оси видеокамеры, примерно перпендикулярном вектору ее перемещения (рисунок 2.7), где $P(s,w,t)=\|p(y,x,s,w,t)\|_{(y=\overline{0,Y_P-1},x=\overline{0,X_P-1})}$ – фотоплан; Y_P, X_P – размеры фотоплана по вертикали и горизонтали; S, W, T – индексы, учитывающие сезонность фотоплана (по месяцам), погодные условия (ясно, облачно, осадки) и время суток (утро, день, вечер, ночь) соответственно. Библиотека фотопланов загружается в память кодера и декодера. Если траектория перемещения летательного аппарата нелинейная, библиотека фотопланов может быть сегментирована так, чтобы каждый сегмент соответствовал линейному участку траектории, включая точки ее изменения. Каждому сегменту могут быть поставлены в соответствие координаты местоположения летательного аппарата. Для формирования фотоплана могут использоваться известные методы поиска соответствия изображений, например RANSAC [13]. Для построения качественных панорам кадры видеопоследовательностей подвергаются нормализации по яркости и геометрической коррекции с учетом аберраций оптической системы видеокамеры.



$\{R_P(s,w,t)\}_{(s=\overline{0,S-1},w=\overline{0,W-1},t=\overline{0,T-1})}$

$\{P(s,w,t)\}_{(s=\overline{0,S-1},w=\overline{0,W-1},t=\overline{0,T-1})}$

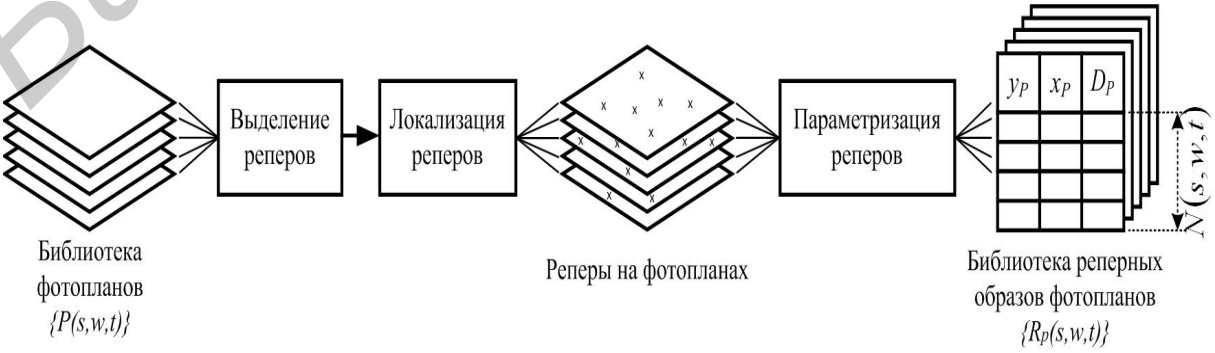
$$R_P(s,w,t)=\|r_P(n,s,w,t)\|_{(n=\overline{0,N(s.w.t)-1})}$$

$P(s,w,t)$ $N(s,w,t)$ $R_P(s,w,t)$

$r_P(n,s,w,t)=\{y_P(n,s,w,t),x_P(n,s,w,t),D_P(n,s,w,t)\}$

$y_P(n,s,w,t)$ $x_P(n,s,w,t)$ $P(s,w,t)$

$D_P(n,s,w,t)=\|d_P(l,n,s,w,t)\|_{(l=\overline{0,L_D-1})}$ L_D



$R_F(k) = \|r_F(n,k)\|_{(n=\overline{0,N_F(k)-1})}$ $F(k) = \|f(y,x,k)\|_{(y=\overline{0,Y-1},x=\overline{0,X-1})}$

$F(k)$

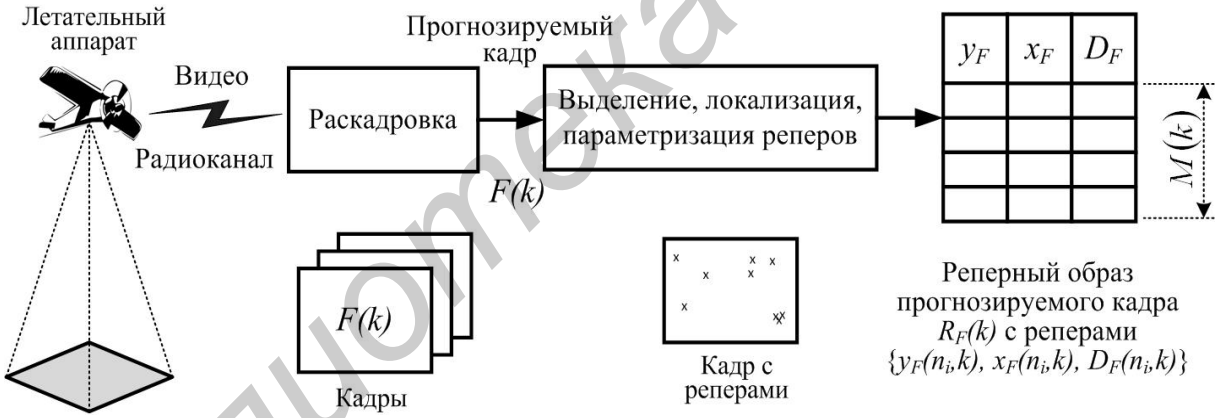
$N_F(k)$

$r_F(n,k) = \{y_F(n,k), x_F(n,k), D_F(n,k)\}$

$y_F(n,k)$ $x_F(n,k)$

$F(k)$ $D_F(n,k) = \|d_F(l,n,k)\|_{(l=\overline{0,L_D-1})}$

$\{R_P(s,w,t)\}_{(s=\overline{0,S-1},w=\overline{0,W-1},t=\overline{0,T-1})}$



$R_F(k)$

$\{R_P(s,w,t)\}_{(s=\overline{0,S-1},w=\overline{0,W-1},t=\overline{0,T-1})}$

времени суток (t). Вероятность ошибки позиционирования прогнозируемого кадра относительно фотоплана может быть уменьшена за счет учета положения видеокамеры и соответствующих геометрических предискажений прогнозируемого кадра. В результате выполнения данного шага выделяются 4 репера $\{r_F(n_i,k)\}_{(i=\overline{0,3})}$ на прогнозируемом кадре $F_I(k)$ и 4 соответствующих им репера $\{r_P(n_i,s,w,t)\}_{(i=\overline{0,3})}$ на фотоплане $P(s,w,t)$. Для идентификаторов соответствующих реперов выполняется условие

$$d_F(l,n_i,k) \approx d_P(l,n_i,s,w,t) \tag{2.14}$$

при $i=\overline{0,3}$, $l=\overline{0,L_D-1}$.

5) Формирование прогнозного кадра. На фотоплане $P(s,w,t)$ выделяется фрагмент $P_F(s,w,t)$, соответствующий четверке реперов $\{r_P(n_i,s,w,t)\}_{(i=\overline{0,3})}$. На основе выделенного фрагмента $P_F(s,w,t)$ формируется прогнозный кадр $F'(k)=\|f'(y,x,k)\|_{(y=\overline{0,Y-1},x=\overline{0,X-1})}$ в соответствии с выражением

$$F'(k)=\phi_H(P_F(s,w,t)), \tag{2.15}$$

где ϕ_H – геометрическое преобразование на основе матрицы гомографии $H=\begin{Vmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{Vmatrix}$ [16], коэффициенты которой вычисляются на основе координат реперов $\{y_F(n_i,k),x_F(n_i,k)\}_{(i=\overline{0,3})}$ и $\{y_P(n_i,s,w,t),x_P(n_i,s,w,t)\}_{(i=\overline{0,3})}$.

6) Яркостная коррекция прогнозного кадра. Вычисляются значения средней яркости $f_M(k)$ и $f'_M(k)$ прогнозируемого $F(k)$ и прогнозного $F'(k)$ кадров соответственно с помощью выражений

$$f_M(k)=\sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} f(y,x,k) \Big/ (Y\,X), \tag{2.16}$$

$$f'_M(k)=\sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} f'(y,x,k) \Big/ (Y\,X). \tag{2.17}$$

На основе соотношения значений средней яркости $f_M(k)$ и $f'_M(k)$ вычисляется корректирующий коэффициент $\delta_f(k)$ с помощью выражения

$$\delta_f(k)=f_M(k)/f'_M(k). \tag{2.18}$$

На основе прогнозного кадра $F'(k)$ и корректирующего коэффициента $\delta_f(k)$ формируется скорректированный по яркости прогнозный кадр $\hat{F}(k)=\|\hat{f}(y,x,k)\|_{(y=\overline{0,Y-1},x=\overline{0,X-1})}$, значения пикселей которого вычисляются с помощью выражения

$$\hat{f}\,(y,x,k)=\left[\delta_f\, f'(y,x,k)\right]$$

$y=\overline{0,Y-1}\quad x=\overline{0,X-1}$

$[\cdot]$

Сущность метода состоит в локализации и параметризации прямых линий на изображениях, формировании фазовых гистограмм прямых линий и их использовании для вычисления угла поворота одного изображения относительно другого. Метод обеспечивает ориентацию изображений, условия формирования которых существенно отличаются сезонностью, временем суток, метеорологической обстановкой и ракурсом, за счет использования прямых линий, более устойчивых по сравнению с реперными точками к изменению яркости, контраста, резкости, параллаксу. Метод может использоваться для совмещения изображения с борта БЛА с электронной картой местностью.

Алгоритм, реализующий данный метод, состоит из следующих шагов.

1) Локализация прямых линий на ориентируемых изображениях. В результате выполнения данного шага для ориентируемых изображений $F_1 = \|f_1(y,x)\|_{(y=\overline{0,Y_1-1},x=\overline{0,X_1-1})}$ и $F_2 = \|f_2(y,x)\|_{(y=\overline{0,Y_2-1},x=\overline{0,X_2-1})}$ формируются множества $\{h_A^{(1)}(n_R), h_D^{(1)}(n_R)\}_{(n_R=\overline{1,N_R^{(1)}})}$ и $\{h_A^{(2)}(n_R), h_D^{(2)}(n_R)\}_{(n_R=\overline{1,N_R^{(2)}})}$ параметров прямых контурных линий и матрицы $P_R^{(1)} = \|p_R^{(1)}(y,x)\|_{(y=\overline{0,Y_1-1},x=\overline{0,X_1-1})}$ и $P_R^{(2)} = \|p_R^{(2)}(y,x)\|_{(y=\overline{0,Y_2-1},x=\overline{0,X_2-1})}$ принадлежности контурных пикселей изображений F_1 и F_2 прямым контурным линиям, где Y_1, X_1 и Y_2, X_2 – размеры изображений F_1 и F_2; $N_R^{(1)}$, $N_R^{(2)}$ – количество прямых линий на изображениях F_1 и F_2; n_R – индекс, указывающий на прямую; $h_A^{(1)}(n_R)$, $h_A^{(2)}(n_R)$ – углы между осью абсцисс и перпендикулярами к прямым с индексами n_R на изображениях F_1 и F_2; $h_D^{(1)}(n_R)$, $h_D^{(2)}(n_R)$ – длины этих перпендикуляров. Ненулевые элементы $p_R^{(1)}(y,x)$ и $p_R^{(2)}(y,x)$ матриц $P_R^{(1)}$ и $P_R^{(2)}$ указывают на индексы n_R прямых контурных линий с параметрами $\{h_A^{(1)}(n_R), h_D^{(1)}(n_R)\}$ и $\{h_A^{(2)}(n_R), h_D^{(2)}(n_R)\}$ соответственно.

2) Построение фазовых гистограмм прямых линий ориентируемых изображений. В результате выполнения данного шага формируются фазовые матрицы $D_P^{(1)} = \|d_P^{(1)}(\varphi)\|_{(\varphi=\overline{0,N_\varphi-1})}$ и $D_P^{(2)} = \|d_P^{(2)}(\varphi)\|_{(\varphi=\overline{0,N_\varphi-1})}$ распределения вероятностей ориентаций прямых линий на изображениях F_1 и F_2, значения элементов $d_P^{(1)}(\varphi)$ и $d_P^{(2)}(\varphi)$ которых определяют количество прямых линий с угловыми ориентациями $\varphi(180/N_\varphi)$, где φ – индекс элемента фазовой матрицы; N_φ – число элементов в фазовых матрицах, определяющее точность учета угловой ориентации прямых линий.

Графическим отображением фазовых матриц $D_P^{(1)}$ и $D_P^{(2)}$ являются фазовые гистограммы, показывающие статистику ориентаций прямых линий на изображениях F_1 и F_2.

3) Идентификация фазовых гистограмм прямых линий ориентируемых изображений. На основе фазовой матрицы $D_P^{(2)}$ формируются циклически сдвинутые фазовые матрицы $\left\{\vec{D}_P^{(2)}(k)=\left\|\vec{d}_P^{(2)}(\varphi,k)\right\|_{(\varphi=\overline{0,N_\varphi-1})}\right\}_{(k=\overline{0,N_\varphi-1})}$, значения элементов $\vec{d}_P^{(2)}(\varphi,k)$ которых определяются с помощью выражения

$$\vec{d}_P^{(2)}(\varphi,k)=d_P^{(2)}\left(\mathrm{mod}_{N_\varphi}(\varphi-k)\right) \tag{2.20}$$

при $k=\overline{0,N_\varphi-1}$.

Формируется матрица $E=\left\|e(k)\right\|_{(k=\overline{0,N_\varphi-1})}$ среднеквадратических ошибок, значения элементов $e(k)$ которой вычисляются с помощью выражения

$$e(k)=\frac{\sum_{k=0}^{N_\varphi-1}\left(d_P^{(1)}(\varphi)-\vec{d}_P^{(2)}(\varphi,k)\right)^2}{N_\varphi} \tag{2.21}$$

при $k=\overline{1,N_\varphi-1}$.

Ищется минимальное значение $e(k)$ в матрице E среднеквадратических ошибок, для которого фиксируется значение k. Угол η, на который изображение F_2 повернуто относительно изображения F_1 находится с помощью выражения

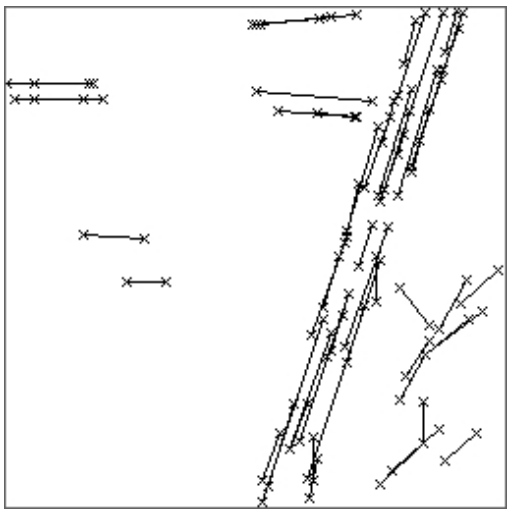
$$\eta=k\left(180/N_\varphi\right). \tag{2.22}$$

В результате выполнения данного алгоритма вычисляется угол η, на который необходимо повернуть одно изображение относительно другого для их правильной ориентации.

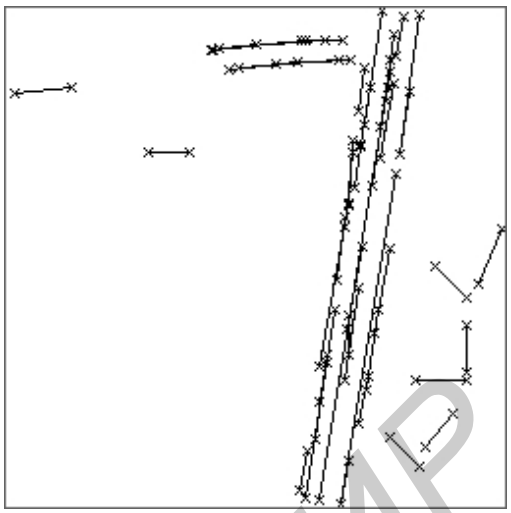
Для локализации прямых линий на изображениях широко используются методы, основанные на преобразовании Хафа [18], масочном поиске [19], вычислении градиента [20] и квантовании по ориентации [21]. Однако эти методы требуют значительных вычислительных ресурсов, имеют низкое быстродействие (преобразование Хафа, масочный поиск) и не обеспечивают качественную локализацию прямых, характеризуемую вероятностью ложной локализации и устойчивостью к изменению яркости, контраста и зашумлению изображения (вычисление градиента, квантование по ориентации). Устранение данных недостатков возможно за счет предварительного грубого квантования отрезков по ориентации с использованием малоразмерных масок, последующего уточнения ориентации этих отрезков и их объединения в прямые в фазовом пространстве пространственно-ориентированного

а

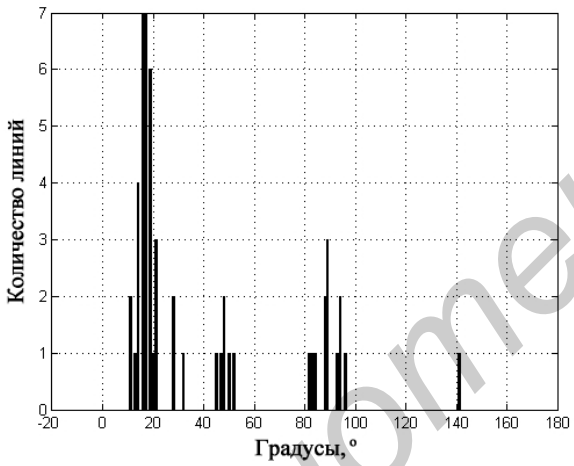
б



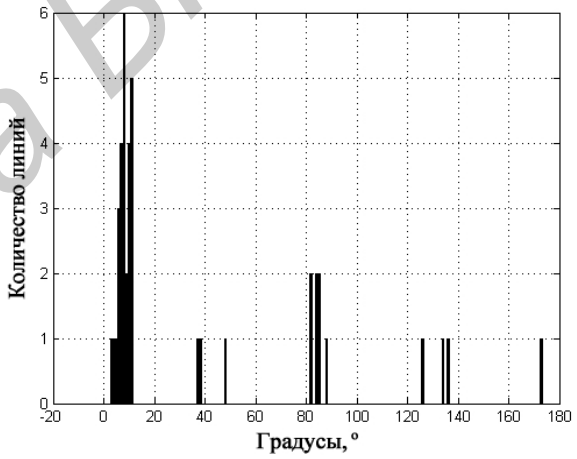
а



б



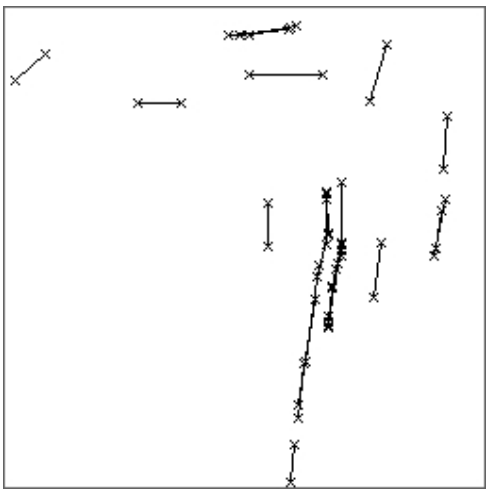
а



б

Методы позиционирования изображений	Предельные относительные значения изменения параметров второго изображения, при которых ошибка в ориентации не более 5 %			
	Яркость		Контраст	
	Уменьшение	Увеличение	Уменьшение	Увеличение
На основе фазовых гистограмм	100 %	100 %	80 %	80 %
SURF/RANSAC	40 %	100 %	40 %	60 %

а



б

3 Методы, алгоритмы и программные средства, оценка эффективности определения координат объекта наблюдения

3.1 Определение координат объекта наблюдения по одному кадру видеопотока на основе GPS-координат носителя и телеметрии в реальном масштабе времени

Разработан алгоритм расчета координат неподвижного объекта в кадре с учетом рельефа местности, радиальной дисторсии и положения главной точки.

3.1.1 Исходные данные

Телевизионная камера размещается в носовой части БЛА и управляется по крену и тангажу относительно его продольной строительной оси (БЛА «Бусел»). Объектив камеры в связанной с БЛА системе координат имеет координаты Xcam, Ycam, Zcam.

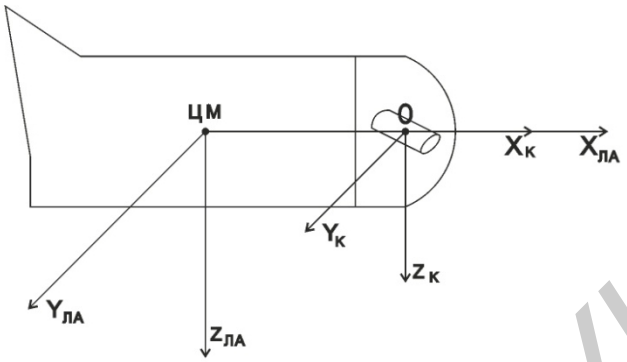
3.1.2 Обозначения и определения

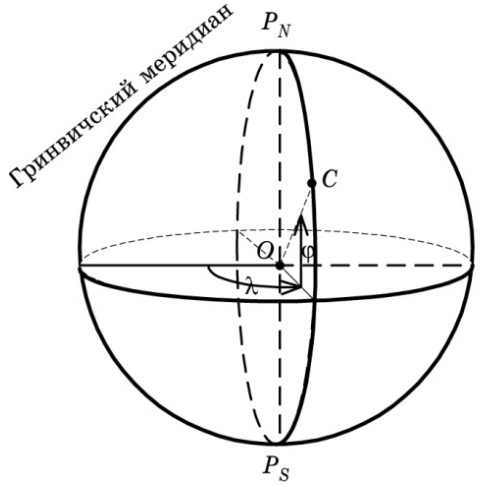
Связанная с БЛА система координат (по МС ИСО 1151, ч 1-5). Центр этой системы координат (СК) располагается в центре масс БЛА. Ось $OX_{ла}$ ориентирована вдоль строительной оси БЛА в направление его полета, ось $OY_{ла}$ - в сторону правого крыла, ось $OZ_{ла}$ направлена вниз к центру земли.

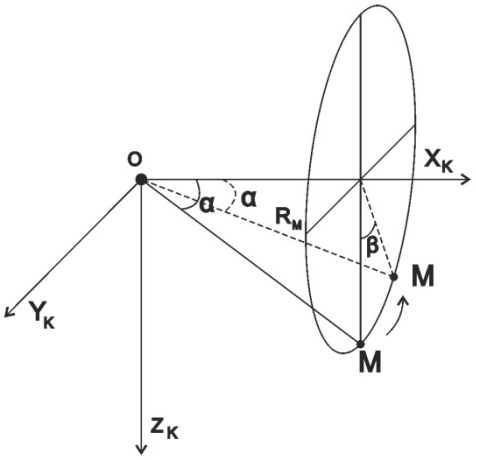
Связанная с камерой система координат (по МС ИСО 1151, ч 1-5). Центр этой системы координат располагается в точке выхода оптической оси объектива (ООО). Ось $OX_к$ ориентирована вдоль продольной строительной оси БЛА в направление его полета, ось $OY_к$ - в сторону правого крыла, ось $OZ_к$ направлена вниз к центру земли. Создается путем параллельного переноса системы координат, связанной с БЛА, в точку с координатами Xcam, Ycam, Zcam (точка О на рисунке 3.1).

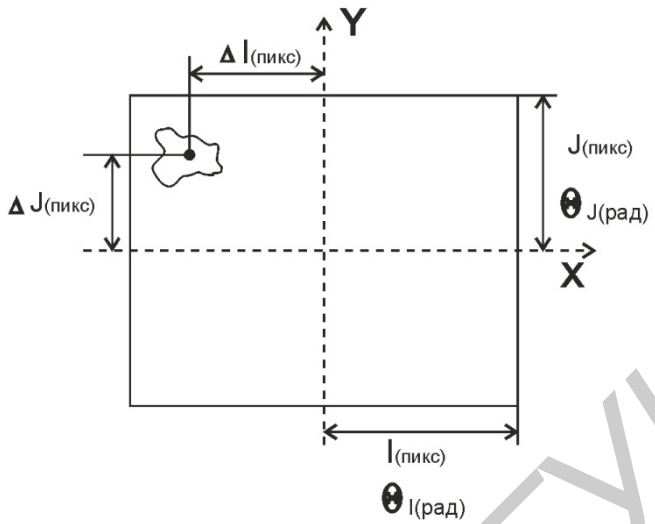
Нормальная система координат (по МС ИСО 1151, ч 1-5). Центр этой системы координат располагается в центре масс БЛА. Ось $OX_и$ ориентирована на север, ось $OY_и$ на восток, ось $OZ_и$ вниз к центру земли.

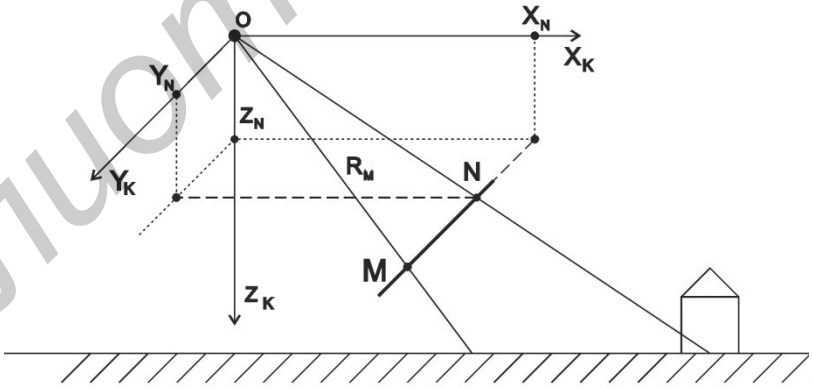
Геоцентрическая система координат. В геоцентрической СК в качестве модели Земли используется сфера, а не общеземной эллипсоид (типа WGS-84 или ПЗ-90 в географической), при этом за основную плоскость отсчета принимается плоскость экватора (рисунок 3.2). Положение точек на поверхности сферы определяется координатами: широтой φ (угол между радиус-вектором, соединяющего центр сферы с данной точкой С и плоскостью экватора) и долготой λ (двугранный угол между плоскостями гринвичского меридиана и меридиана данной точки). Для перехода от географической к геоцентрической системе







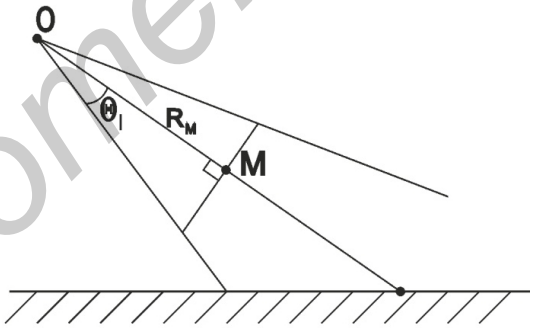




$$x_n = \frac{I - X}{I - X} \quad R_M \quad {}_{\mathrm{I}} - X \quad * \quad X$$

$$y_n = \frac{J - Y}{J - Y} \quad R_M \quad {}_{\mathrm{J}} - Y \quad * \quad Y$$

$$x \quad = x_n(\; + k \quad I - X \quad {}^2 + \quad J - Y \quad {}^2 + k \quad I - X \quad {}^2 + \quad J - Y \quad {}^{2\ 2}$$
$$+ k \quad I - X \quad {}^2 + \quad J - Y \quad {}^{2\ 3})$$
$$y \quad = y_n(\; + k \quad I - X \quad {}^2 + \quad J - Y \quad {}^2 + k \quad I - X \quad {}^2 + \quad J - Y \quad {}^{2\ 2}$$
$$+ k \quad I - X \quad {}^2 + \quad J - Y \quad {}^{2\ 3})$$



$$X_N^k = X_M + \quad X$$
$$Y_N^k = Y_M + \quad Y$$
$$Z_N^k = Z_M + \quad Z$$

$$X_M = R_M \quad (\propto)$$

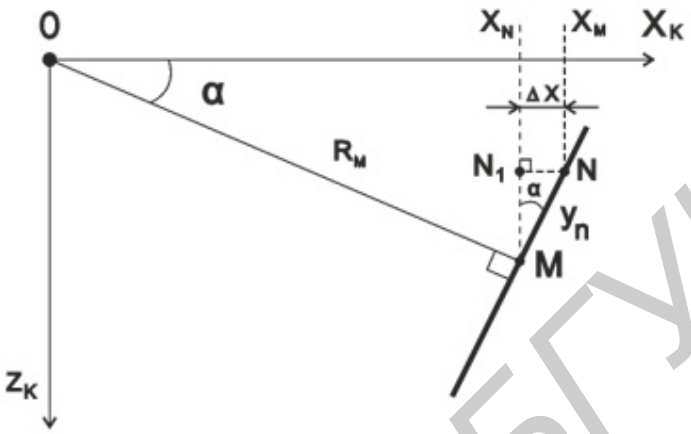
$$Y_M = (R_M \quad (\propto)) \quad (\;)$$

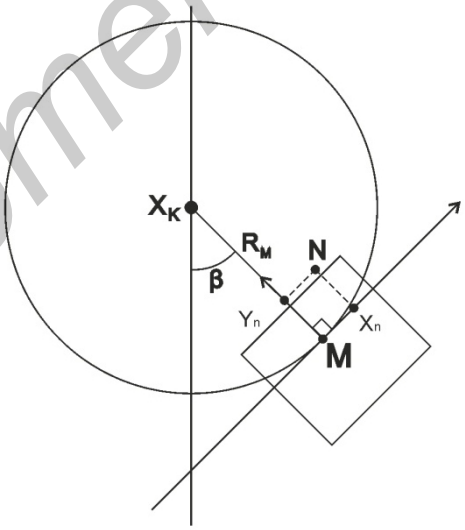
$$Z_M = (R_M \quad (\propto)) \qquad Sgn\,(\alpha)$$

$X = y_1 \qquad (\alpha)$

X

$X_k O Z_k$





$Y \quad Z$

$$Y = Cos\ \beta\ \ x \qquad - Sin\ \beta\, Sgn(\alpha)\ \ Cos\ \alpha\ y$$

$$Z = -(Sin\ \beta\, Sgn\ \alpha \quad x \qquad + Cos\ \beta\ \ Cos\ \alpha\ y \quad)$$

$$X_N^{\text{ла}} = X_N^k + \mathrm{X_{cam}}$$
$$Y_N^{\text{ла}} = Y_N^k + \mathrm{Y_{cam}}$$
$$Z_N^{\text{ла}} = Z_N^k + \mathrm{Z_{cam}}$$

5) Используя матрицу перехода F от связанной системы к нормальной

$$F = \begin{matrix} f1 & f2 & f3 \\ f4 & f5 & f6 \\ f7 & f8 & f9 \end{matrix},$$

где $f1 = Cos\ \psi\ Cos\ \theta$; $f2 = Sin\ \psi\ Cos\ \theta$; $f3 = -Sin\ \theta$; $f4 = Cos\ \psi\ Sin\ \theta\ Sin\ \varphi\ - Sin\ \psi\ Cos\ \varphi$; $f5 = Sin\ \psi\ Sin\ \theta\ Sin\ \varphi\ + Cos\ \psi\ Cos\ \varphi$; $f6 = Cos\ \theta\ Sin\ \varphi$; $f7 = Cos\ \psi\ Sin\ \theta\ Cos\ \varphi\ + Sin\ \psi\ Sin\ \varphi$; $f8 = Sin\ \psi\ Sin\ \theta\ Cos\ \varphi\ - Cos\ \psi\ Sin\ \varphi$; $f9 = Cos\ \theta\ Cos\ \varphi$; ψ – угол курса БЛА; θ – угол тангажа БЛА; φ – угол крена БЛА,

получим координаты точек N и O в нормальной системе координат:

$$X_N^{\text{и}} = X_N^k Cos\ \psi\ Cos\ \theta\ + Y_N^k Sin\ \psi\ Cos\ \theta\ - Z_N^k Sin\ \theta$$
$$Y_N^{\text{и}} = X_N^k(Cos\ \psi\ Sin\ \theta\ Sin\ \varphi\ - Sin\ \psi\ Cos\ \varphi\ ) + Y_N^k(Sin\ \psi\ Sin\ \theta\ Sin\ \varphi + Cos\ \psi\ Cos\ \varphi\ ) + Z_N^k Cos\ \theta\ Sin\ \varphi$$
$$Z_N^{\text{и}} = X_N^k(Cos\ \psi\ Sin\ \theta\ Cos\ \varphi\ + Sin\ \psi\ Sin\ \varphi\ ) + Y_N^k(Sin\ \psi\ Sin\ \theta\ Cos\ \varphi\ ) - Cos\ \psi\ Sin\ \varphi\ ) + Z_N^k Cos\ \theta\ Cos\ \varphi$$
$$X_{\text{O}}^{\text{и}} = X_{cam} Cos\ \psi\ Cos\ \theta\ + Y_{cam} Sin\ \psi\ Cos\ \theta\ - Z_{cam} Sin\ \theta$$
$$Y_O^{\text{и}} = X_{cam}(Cos\ \psi\ Sin\ \theta\ Sin\ \varphi\ - Sin\ \psi\ Cos\ \varphi\ ) + Y_{cam}(Sin\ \psi\ Sin\ \theta\ Sin\ \varphi + Cos\ \psi\ Cos\ \varphi\ ) + Z_{cam} Cos\ \theta\ Sin\ \varphi$$
$$Z_O^{\text{и}} = X_{cam}(Cos\ \psi\ Sin\ \theta\ Cos\ \varphi\ + Sin\ \psi\ Sin\ \varphi\ ) + Y_{cam}(Sin\ \psi\ Sin\ \theta\ Cos\ \varphi - Cos\ \psi\ Sin\ \varphi\ ) + Z_{cam} Cos\ \theta\ Cos\ \varphi$$

6) Строим параметрическое уравнение прямой в нормальной системе координат, проходящей через точки O и N:

$$\begin{matrix} X = & X_N^u - X_O^u\ t + X_O^u \\ Y = & Y_N^u - Y_O^u\ t + Y_O^u \\ Z = & Z_N^u - Z_O^u\ t + Z_O^u \end{matrix}$$, где t параметр.

Для того, чтобы найти координаты, подставляем в последнее уравнение вместо Z высоту БЛА над объектом ($h_{\text{ла}}$). Находим из него t:

$$t = \frac{h_{\text{ла}} - Z_O^{\text{и}}}{Z_N^u - Z_O^u}$$

и подставляем в два других уравнения. Пересечение оптической оси будет действительным, если параметр t > 0.

Таким образом у нас есть координаты объекта $\begin{matrix} X \\ Y \\ Z \end{matrix}$ в нормальной системе координат.

7) Коррекция полученных координат объекта в нормальной системе координат путем учета рельефа (матрица высот соответствует направлению вдоль прямой между точкой О и объектом, находится в нормальной системе координат):

7.1) В матрице высот определяем наибольшее и наименьшее значения высоты (H_{max} и H_{min}).

7.2) Для найденных наибольшего и наименьшего значений высоты определяем координаты точек $\begin{matrix} Xmax \\ Ymax \\ Zmax \end{matrix}$ и $\begin{matrix} Xmin \\ Ymin \\ Zmin \end{matrix}$. Для этого подставляем в уравнения из пункта 6 вместо $h_{ла}$ соответственно ($h_{ла}$ - H_{max}) и ($h_{ла}$ - H_{min}).

7.3) По теореме Пифагора рассчитываем расстояние S между точками $\begin{matrix} Xmax \\ Ymax \\ 0 \end{matrix}$ и $\begin{matrix} Xmin \\ Ymin \\ 0 \end{matrix}$.

7.4) Начиная с точки $\begin{matrix} Xmin \\ Ymin \\ Zmin \end{matrix}$, итерационно увеличиваем Zmin на величину $dh = \frac{Hmax-Hmin}{S} dS$, где dS – расстояние между двумя точками отсчета в матрице высот, и рассчитываем остальные координаты по уравнениям из пункта 6. На каждой итерации сравниваем значение Z с соответствующим значением H цифровой модели рельефа. Когда значение Z превысит соответствующее значение H (или сравняется с ним), рассчитываем высоту объекта $h_{об}$ с учетом рельефа по следующей формуле:

$$h_{об} = \frac{H_i Z_{i-1} - H_{i-1} Z_i}{H_i - H_{i-1} - dh},$$

где H_i и Z_i соответствуют итерации, когда значение Z впервые превысило соответствующее значение H (или сравнялось с ним), i-1 – предыдущая итерация.

Остальные координаты объекта рассчитываем по формулам пункта 6.

8) Чтобы получить координаты в геоцентрической системе вычислим:

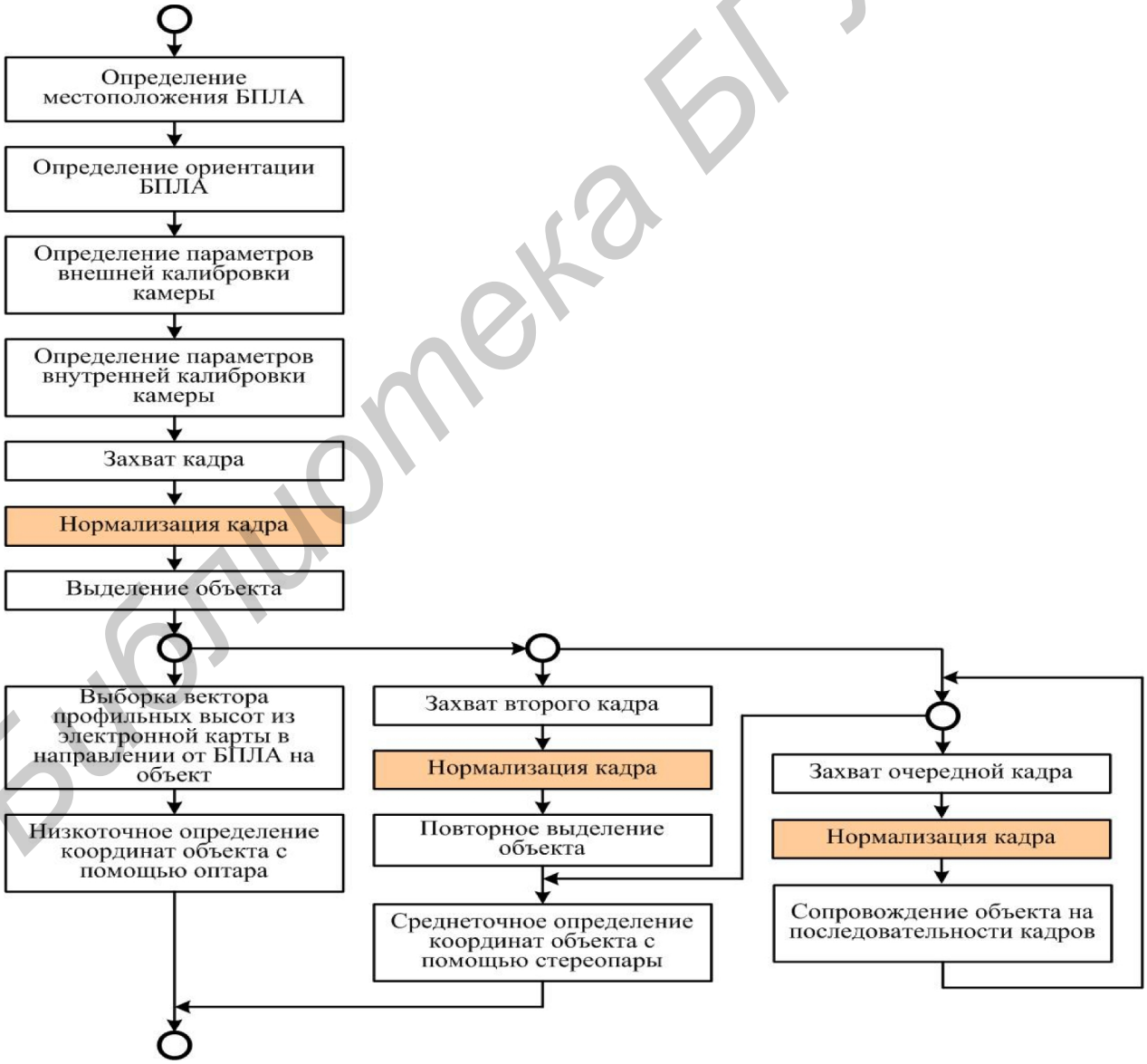
$\varphi_o = \varphi + \Delta\varphi$ –широта объекта;

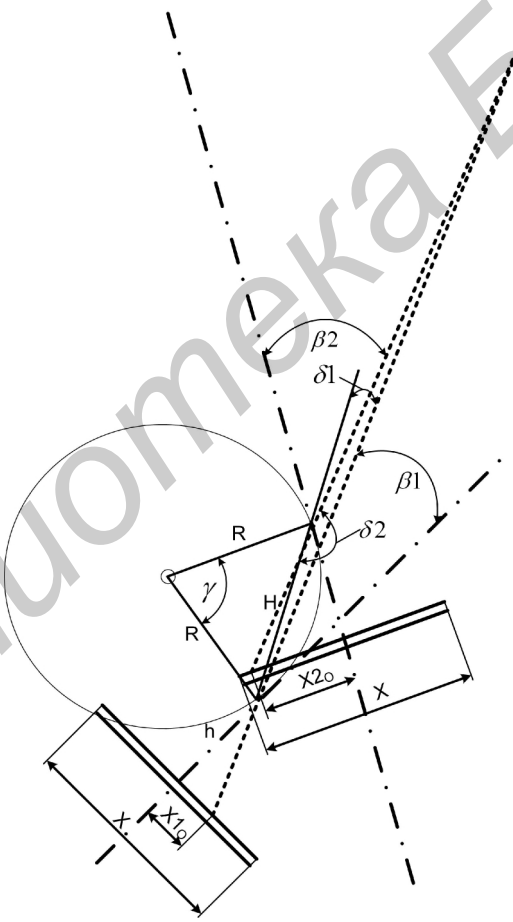
$\lambda_o = \lambda + \Delta\lambda$ – долгота объекта;

$h_o = h - \Delta h$ – высота объекта,

где $\Delta\varphi = X/R$ – приращение по широте; $\Delta\lambda = Y/(R * Cos\varphi)$– приращение по долготе; $\Delta h = Z$– приращение по высоте; φ, λ, h – широта, долгота и высота ЛА, соответственно.

Рассмотренный способ определения координат реализован программно для встраивания в НПУ (приложение Е).





$$H = \frac{R \cdot \sin\gamma}{\sin\left(\frac{180-\gamma}{2}\right)}$$

$$\delta 1 = 90 - \beta 1 - \frac{180 - \gamma}{2} \rightarrow \delta 1 = \frac{\gamma}{2} - \beta 1,$$

$$\delta 2 = 90 - \frac{180 - \gamma}{2} + 180 - \beta 2 \rightarrow \delta 2 = \frac{\gamma}{2} + 180 - \beta 2,$$

$$\frac{L2_O}{\sin \delta 2} = \frac{H}{\sin(180 - \delta 1 - \delta 2)} \rightarrow L2_O = \frac{R \cdot \sin \gamma \cdot \sin \delta 2}{\sin\left(\frac{180 - \gamma}{2}\right) \cdot \sin(180 - \delta 1 - \delta 2)}.$$

Пусть

$$X1_O = X2_O = \frac{X}{2}.$$

Тогда

$$\beta 1 = \beta 2 = \frac{\alpha}{2} \quad \delta 1 = \frac{\gamma}{2} - \frac{\alpha}{2} \quad \delta 2 = \frac{\gamma}{2} + 180 - \frac{\alpha}{2},$$

$$L2_O = \frac{R \cdot \sin \gamma}{2 \cdot \sin\left(\frac{180 - \gamma}{2}\right) \cdot \cos\left(\frac{\gamma}{2} - \frac{\alpha}{2}\right)}.$$

Достоинством данного способа определения координат цели является отсутствие необходимости в использовании электронной карты местности для определения высоты местоположения цели. В определенных условиях (при наличии дополнительной камеры, установленной в надир и точном определении высоты полета БЛА) возможно определение стереобазы не по скорости, а по проекции перемещения БЛА.

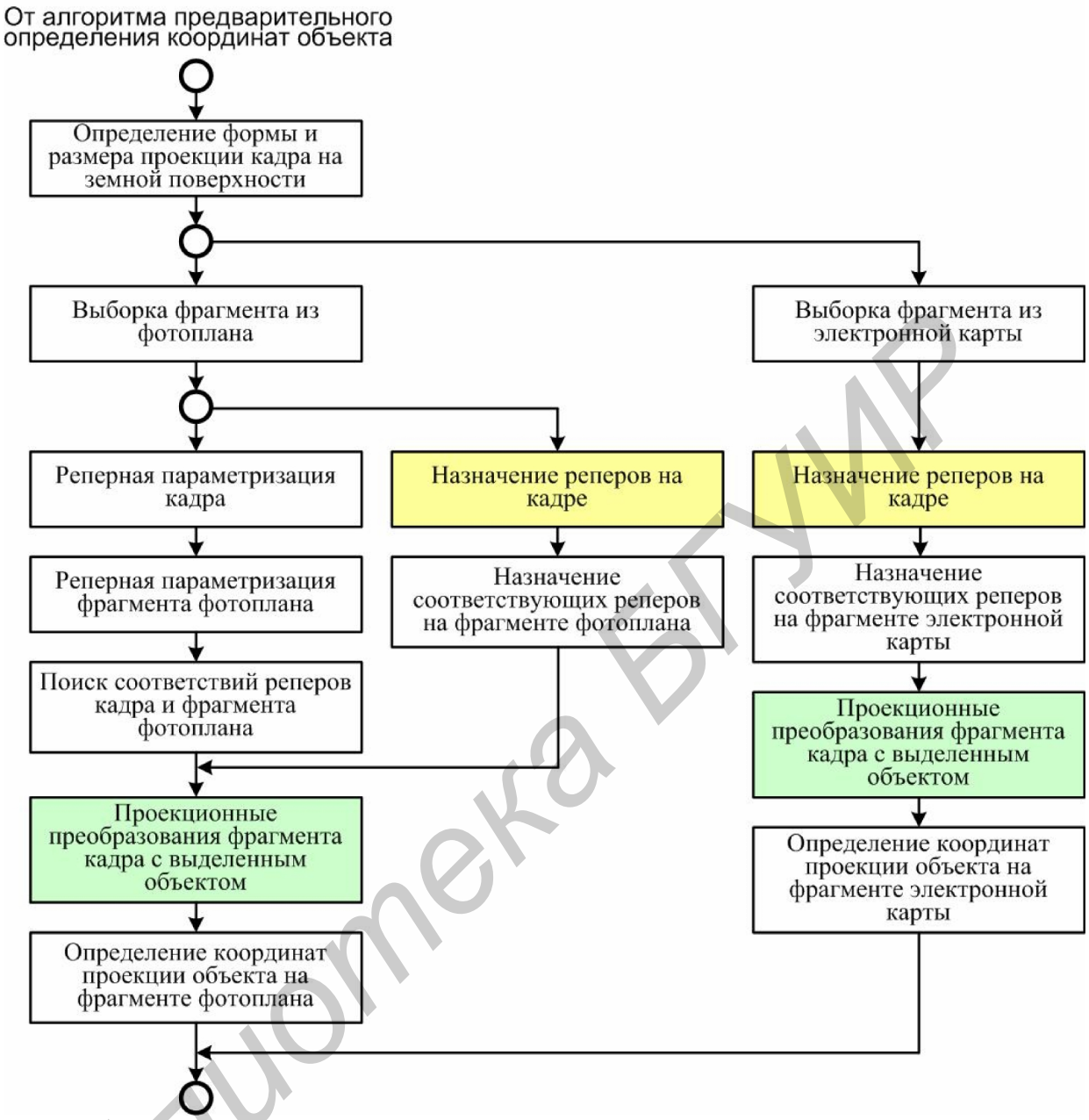
3.3 Определение координат объекта наблюдения по одному кадру видеопотока на основе координат носителя, телеметрии, фотоплана и электронной карты местности

На рисунке 3.11 приведен алгоритм, который реализует вариант высокоточного определения координат цели двумя способами не в реальном масштабе времени следующими способами:

1) с использованием одного кадра и фотоплана (по стоп-кадру / по кадру / с автоматической идентификацией / с ручным сопоставлением);

2) с использованием стоп-кадра и электронной карты (с ручным сопоставлением).

Данный алгоритм реализуется не в реальном масштабе времени. Он предполагает ручное или автоматическое совмещение стоп-кодра, выделенного оператором из видеопотока с порта БЛА, с фрагментом фотоплана или электронной карты местности.



использоваться для сопоставления видеокадра с фрагментом электронной карты местности в случае, когда градиентные методы поиска и идентификации не эффективны. Листинг программы выделения и локализации линий на изображениях, разработанной для среды Matlab, приведен в приложении Ж.

3.4 Возможности повышения точности сопровождения и определения координат объектов наблюдения с БЛА

3.4.1 Исследование возможности повышения точности сопровождения цели с БЛА

На точность сопровождения цели с БЛА оказывают влияние следующие факторы:

- качество видеоданных в целом (определяется качеством видеокамеры; погодными условиями; характером движения БЛА);

- качество выделения цели (определяется качеством методов и алгоритмов выделения цели; различимостью цели на фоне других объектов, в том числе, в условиях наличия объектов с такими же или схожими характеристиками).

Установлены следующие возможности повышения точности сопровождения цели с БЛА.

1) Использование цифровых способов формирования и передачи изображений, высокой кадровой частоты и высокого разрешения видеокамеры; большой битовой глубины видеокадра [26]. Цифровые способы формирования и передачи изображений обеспечивают меньшую ошибку по сравнению с аналоговыми способами при одинаковом отношении сигнал-шум. Повышение кадровой частоты обеспечивает большую дискретизацию траектории движения объекта, что уменьшает вероятность его потери (однако, это требует большего числа операций при обработке). Повышение разрешения видеокамеры и битовой глубины кадра позволяет улучшить различимость цели за счет повышения контраста и точности передачи характеристик цели (однако, так же приводит к усложнению обработки кадров).

2) Использование стабилизации видеокамеры. Необходимость и качество стабилизации видеокамеры определяются, прежде всего, характеристиками БЛА. Видеокамеры легких БЛА нуждаются в стабилизации более, чем тяжелых. Однако, именно для легких БЛА реализация стабилизации наиболее проблематична. Из-за ограничений на массогабаритные характеристики целевой нагрузки БЛА особенно проблематично построение эффективной механической системы стабилизации. Что касается электронной стабилизации, то ее использование при сопровождении цели требует наличия соответствующих вычислительных ресурсов и качества изображения. Качество изображения

определяет возможность выделения на нем признаков (реперов) для выявления смещения кадров [17, 24].

3) Повышение точности телеметрии. Данные телеметрии для БЛА и видеокамеры (высота, координаты, углы) позволяют предсказать смещение кадров, упростить задачу поиска цели и уменьшить вероятность ошибки обнаружения цели [27]. При этом телеметрия должна быть актуальной для каждого видеокадра (т.е. частота обновления телеметрии должна соответствовать частоте кадров и время формирования телеметрии не должно превышать периода следования кадров).

4) Повышение эффективности методов и алгоритмов поиска цели. Одна из возможностей заключается в использовании предсказания перемещения цели и камеры, основанного на статистике предшествующих перемещений. Экспериментально установлено, что наиболее устойчивыми методами поиска являются ковариационные [7].

3.4.2 Исследование возможности повышения точности определения координат цели с БЛА

Географические координаты цели могут быть определены в результате двух подходов: по GPS-координатам БЛА и по электронной карте местности.

Точность определения координат цели по GPS-координатам БЛА зависит от:

- точности определения координат цели относительно БЛА;
- точности определения GPS-координат БЛА.

Для повышения точности определения координат цели относительно БЛА необходимо повышать точность телеметрии, в том числе точность привязки телеметрии к видеокадрам [27]. Однако, повышение точности телеметрии в данном случае должно учитывать погрешности определения GPS-координат БЛА.

Второй подход представляется более точным и надежным, поскольку обеспечивает определение координат цели без GPS-координат БЛА. Его реализация в реальном масштабе времени требует, однако, существенных вычислительных затрат. В данном случае необходимо сопоставлять видеокадр с электронной картой местности. Для этого может быть произведено предварительное сопоставление видеокадра с фотопланом, заранее сопоставленным с электронной картой местности. Причем, даже для этого варианта использование классических алгоритмов поиска соответствия, основанных на вычислении градиента [14, 15], не эффективно. Поэтому актуальной является задача разработки новых методов выделения, параметризации и идентификации реперов на изображениях. Наиболее эффективные методы основаны на использовании прямых контурных линий [17, 22, 25].

4 Анализ аналоговых и цифровых способов передачи видеоданных с борта БЛА

4.1 Оценка эффективности использования аналоговой видеокамеры на борту БЛА

Достоинством варианта использования аналоговой камеры на борту БЛА является относительная простота реализации и как, следствие, относительно низкая стоимость, а также использование сравнительно малой полосы для передачи видеосигнала (используется телевизионный сигнал в формате PAL, требующий для передачи канал с полосой 6 МГц).

Недостатки данного подхода состоят в следующем.

1) Относительно высокая чувствительность к отношению сигнал-шум в канале передачи. Как следствие – сравнительно низкое (по отношению к цифровому способу формирования и передачи изображений) качество видеоизображения на приемном конце. Это усложняет последующую обработку информации при решении задач поиска и сопровождения целей на изображениях.

2) Отсутствие возможности обработки видеоинформации на канальном уровне (коррекция ошибок передачи, криптографическое кодирование). Это не позволяет восстанавливать потерянную видеоинформацию и защищать ее от несанкционированного доступа.

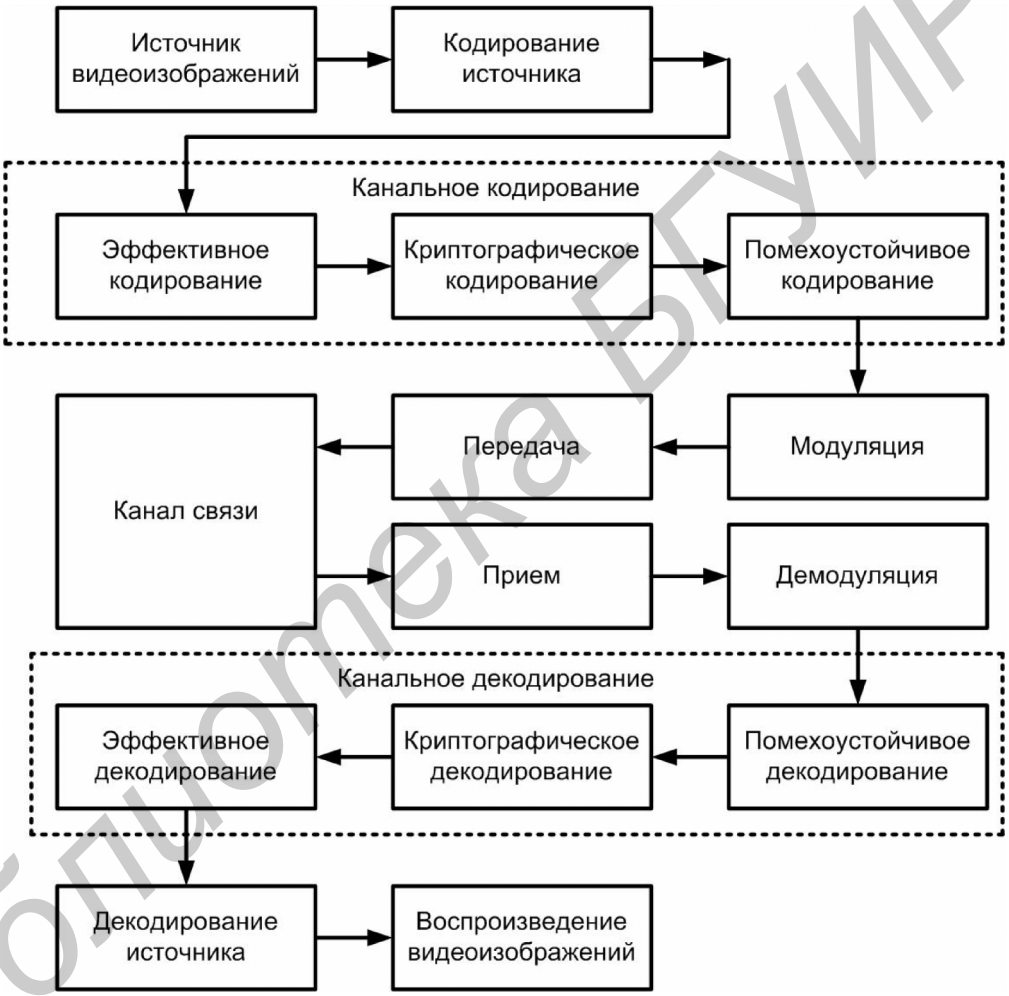
В связи с тем, что обработке целевой видеоинформации придается важное значение, целесообразно переходить к цифровым методам формирования и передачи.

4.2 Исследование методов кодирования и цифровой передачи видеоинформации

4.2.1 Модель канала передачи видеоизображений

На рисунке 4.1 представлена модель цифрового канала передачи видеоданных от цифровой видеокамеры, включающая блоки эффективного, криптографического и помехоустойчивого кодирования [28–30].

Эффективное кодирование обеспечивает компактное представление передаваемой информации. В узком смысле под эффективным кодированием понимается замена оцифрованных дискретов сигнала кодовыми словами различной длины согласно таблице подстановок, которая формируется на основе информации о вероятности значений дискретов. В результате достигается сокращение объема информации и требуется меньше места для ее хранения и меньше полосы канала для ее передачи. В более широком смысле под эффективным кодированием понимается сжатие информации, которое может быть с потерями или без потерь. При сжатии с потерями точность восстановления информации в



соответствии с алгоритмом помехоустойчивого кодирования. В более сложном случае осуществляется преобразование передаваемой информации. Различные методы помехоустойчивого кодирования обеспечивают различную корректирующую способность (различное число исправляемых ошибок на кодовый блок) и различную избыточность. В общем случае, избыточность увеличивается пропорционально корректирующей способности. Поэтому, при использовании помехоустойчивого кодирования необходимо предусмотреть соответствующее увеличение коэффициента сжатия при эффективном кодировании для того, чтобы вписаться в полосу канала передачи.

Криптографическое кодирование обеспечивает защиту информации от несанкционированного доступа. В общем случае для криптографического кодирования могут использоваться симметричные и несимметричные криптосистемы, а также их комбинация.

С точки зрения базовой эталонной модели OSI взаимодействия открытых систем эффективное кодирование (сжатие) выполняется на представительном уровне. Криптографическое кодирование информации также выполняется на представительском уровне, однако независимо может выполняться в транспортных сетях на сетевом уровне и в сетях беспроводного доступа на канальном уровне.

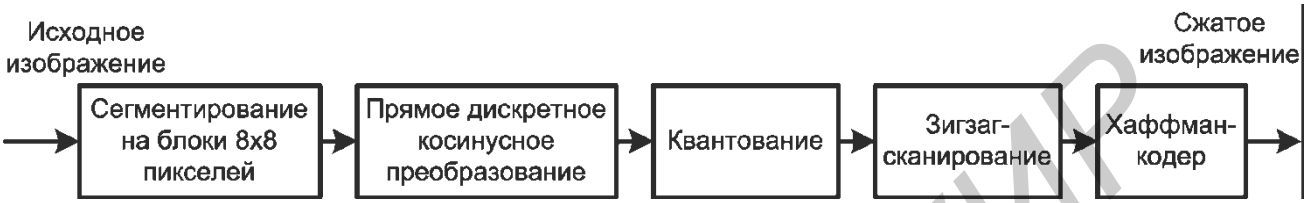
4.2.2 Эффективное кодирование для передачи видеоизображений

Используемые в настоящее время методы эффективного кодирования (сжатия) видеоизображений (подвижных изображений) делятся на три класса: на основе покадрового кодирования; на основе кодирования кадровой разности и на основе предсказания.

4.2.2.1 Методы сжатия видеоданных на основе покадрового кодирования

При покадровом кодировании каждый кадр исходного видеопотока сжимается независимо от других кадров как отдельное изображение. В данных методах учитывается только структурная избыточность, обусловленная локальной корреляцией значений соседних пикселей, и статистическая избыточность, обусловленная корреляцией значений пикселей на всем изображении. Наиболее распространенными методами, основанными на покадровом кодировании видеоизображений, являются Motion JPEG [29, 33] и Motion JPEG 2000 [29, 33], основанные на методах JPEG (Joint Photographic Experts Group) [34, 35] и JPEG 2000 [35] соответственно. Кодеки Motion JPEG и Motion JPEG 2000 имеют аналогичные структуры, которые отличаются только ядром, обеспечивающим кодирование кадра в формате JPEG и JPEG 2000 соответственно.

Метод JPEG является в настоящее время наиболее широко используемым стандартом сжатия неподвижных изображений [34, 35]. Основными операциями метода сжатия JPEG являются разделение изображения на блоки 8×8 пикселей, дискретно-косинусное



$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}\left(X_i - \widetilde{X}_i\right)^2$$

$$PSNR = 10\log_{10}\left(\frac{\left(2^{BD}-1\right)^2}{MSE}\right)$$

$$BR = BD/CR$$

$X_i, \widetilde{X}_i$

512×512

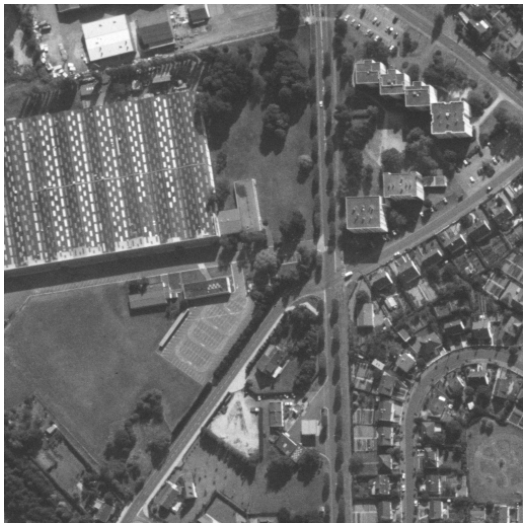


Рисунок 4.3 – Тестовое полутоновые изображение «Town»

Для тестового изображения «Town» на рисунке 4.4 представлены характеристики сжатия с потерями алгоритмов JPEG и JPEG 2000. Из рисунка 4.4 следует, что JPEG значительно уступает JPEG 2000. Более высокие характеристики сжатия JPEG 2000 обусловлены свойствами вейвлет-преобразования [36].

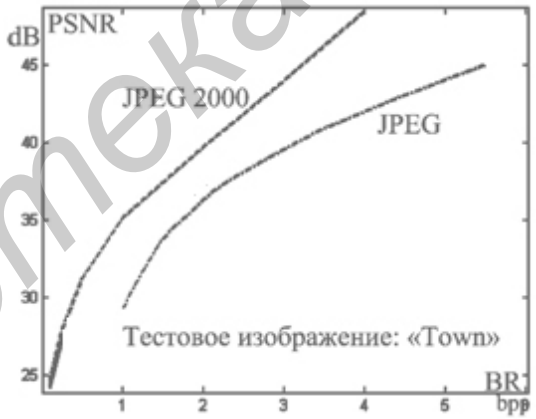
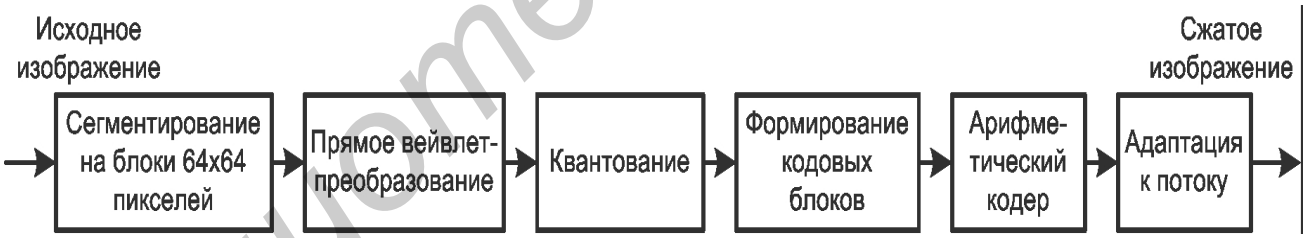


Рисунок 4.4 – Характеристики сжатия алгоритмов JPEG и JPEG 2000 для изображения «Town»

Базисные функции вейвлет-преобразования имеют высокую пространственную и достаточно высокую частотную локализацию, что позволяет использовать небольшое число коэффициентов для представления фрагментов изображений, как с плавными, так и резкими изменениями яркости. Концентрация информации в малом числе коэффициентов повышает степень сжатия. Кроме того, коэффициенты вейвлет-преобразования группируются в древовидную структуру с несколькими уровнями пространственно-частотного разрешения. Пространственная корреляция значений пикселей изображения в результате вейвлет-преобразования трансформируется во внутри и межуровневую корреляцию значений

64×64



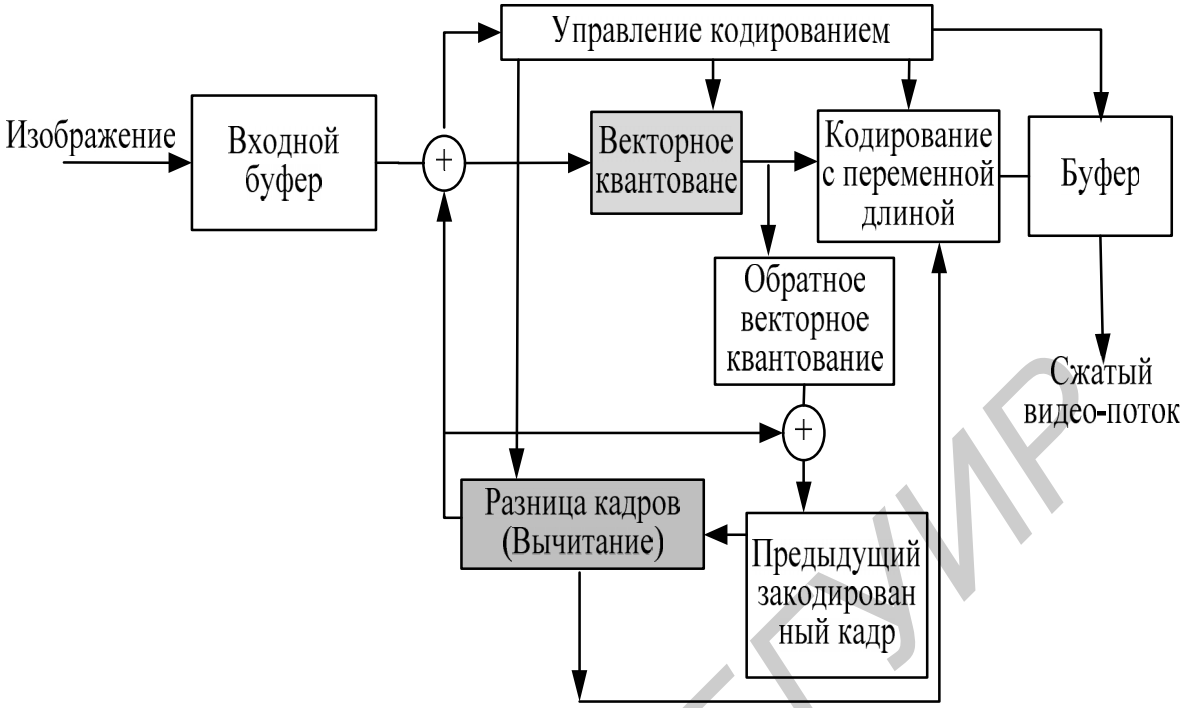


Рисунок 4.6 − Структура видеокодера Cinepak

4.2.2.3 Методы сжатия видеоданных на основе предсказания

Сущность методов сжатия видеоизображений на основе предсказания состоит в использовании опорного кадра для формирования следующего прогнозного кадра, предсказывающего изменения опорного кадра и близкого к действительному следующему кадру, вычислении разности между прогнозным и действительным кадрами, эффективном кодировании опорного кадра и полученной кадровой разности. В используемых в настоящее время методах сжатия видеоизображений предсказание основано на блочной компенсации движения. Сущность блочной компенсации движения заключается в равномерном разбиении каждого прогнозируемого кадра на блоки; поиске для каждого выделенного блока наиболее схожего с ним блока на опорном кадре в пределах ограниченной окрестности выделенного блока; замене каждого выделенного блока прогнозируемого кадра найденным прогнозным блоком. Сжатие видеоизображений с предсказанием на основе блочной компенсации движения реализовано в стандартах серии MPEG (Moving Picture Experts Group) и рекомендации H.264.

Стандарты MPEG и рекомендация H.264 предполагают гибридное эффективное видеокодирование, учитывающее различные виды избыточности видеоданных [33, 37, 38]. Они используют оценку движения и блочную компенсацию движения для устранения временной избыточности. На рисунке 4.7 представлена обобщенная схема гибридного видеокодирования с блочной компенсацией движения. Схема предполагает, что входной

кадр видеоданных разделяется на макроблоки и эффективное видеокодирование применяется к каждому макроблоку.

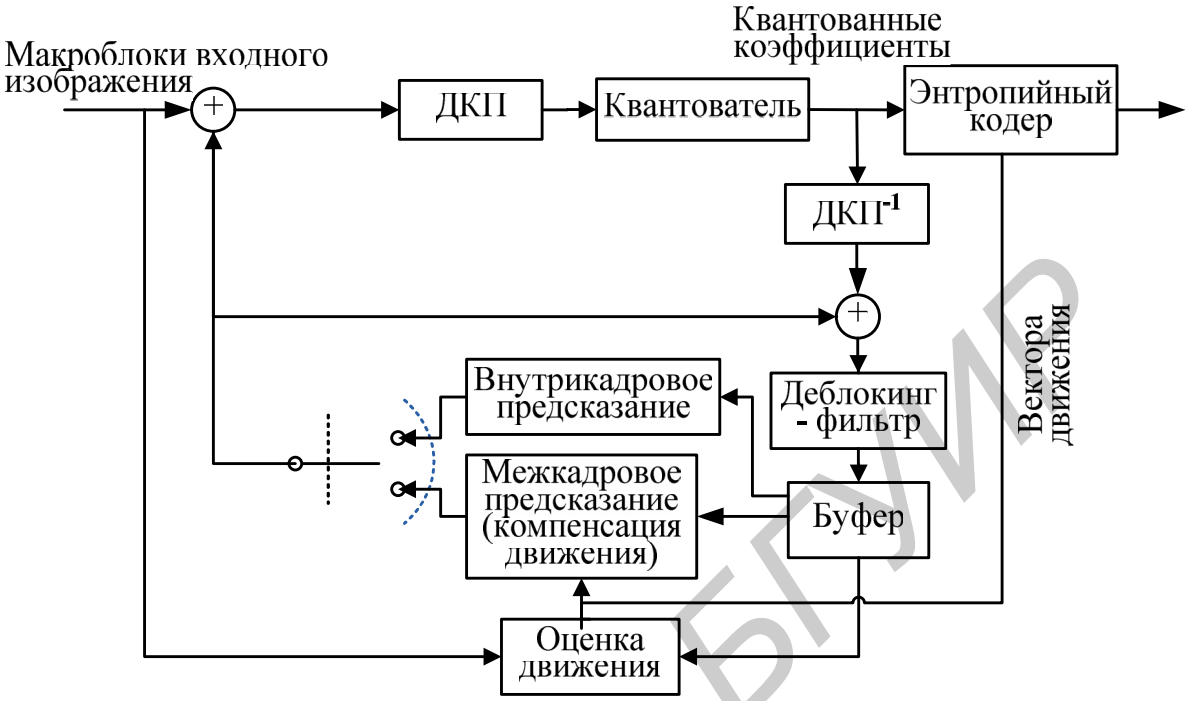
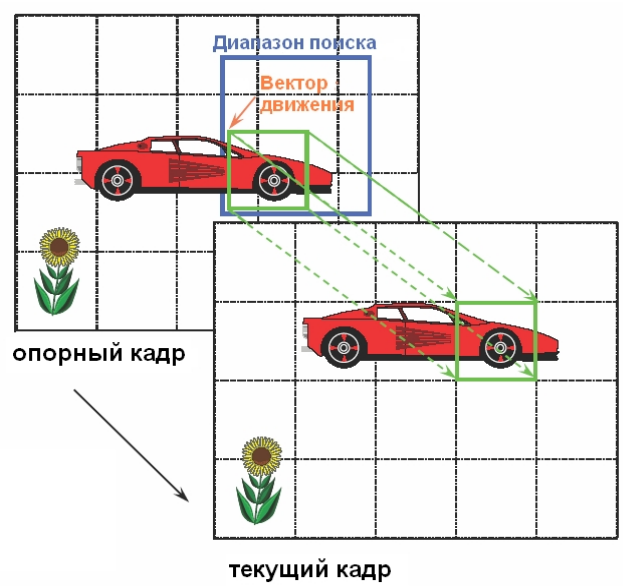


Рисунок 4.7 − Структура видеокодера с блочной компенсацией движения

Каждый макроблок состоит из трех компонентов Y (яркостной), Cr и Cb (цветоразностных). Поскольку зрительная система человека менее чувствительна к цветовым искажениям, чем к яркостным, цветоразносные компоненты формируются с коэффициентом децимации 2 по горизонтали и вертикали. Макроблок состоит из одного блока с матрицей 16 x16 пикселей для яркостной компоненты и двух блоков с матрицами 8x8 пикселей для цветоразностных компонент.

Макроблоки кодируются в Intra Frame или Inter Frame режимах. В Inter Frame режиме в качестве опорного используется предыдущий или последующий кадр видеопоследовательности для реализации блочной компенсации движения. В Intra Frame режиме для блочной компенсации движения используется текущий кадр видеопоследовательности. Для определения смещений макроблока в Intra Frame или Inter Frame режимах используется оценка движения (рисунок 4.8). Она основана на поиске наиболее схожей с макроблоком области в окрестности 16x16 пикселей. В качестве критерия наибольшей схожести используется минимум среднеквадратической ошибки. Сжатие видеоданных достигается за счет замены макроблока вектором движения и ошибкой предсказания, динамический диапазон которой существенно меньше динамического диапазона макроблока. Разрядность вектора движения определяется окрестностью поиска, а число векторов – количеством макроблоков на текущем кадре.



При использовании беспроводного доступа на базе WiMAX условия передачи существенно ухудшаются. С точки зрения информационной безопасности за счет сквозной защиты может быть обеспечен практически такой же уровень защищенности, как в стационарной сети. Однако, вероятность ошибки при информационном обмене между шлюзом и центром управления определяется вероятностью ошибки в радиоканале доступа 10^{-3} – 10^{-6} , что значительно выше, чем в стационарном варианте. Конкретное значение зависит от типа антенн, энергетического потенциала, вида модуляции, отношения сигнал-шум на входе приемника. При вероятности ошибки 10^{-6} искажениям подвергается каждый 6 – 7 кадр видеопоследовательности. Каким образом эти искажения будут проявляться при воспроизведении видеоизображений, зависит от используемого метода эффективного кодирования.

Наименее чувствительными к ошибкам являются методы сжатия видеоданных на основе покадрового кодирования. Они не размножают ошибки на другие кадры. При использовании метода Motion JPEG 2000 ошибка локализуется в пределах блока 64x64 пикселей. Ее заметность зависит от местоположения ошибочного символа в битовой последовательности, представляющей блок: чем дальше от начала последовательности находится ошибочный символ, тем меньше его вес и тем менее заметна ошибка.

При использовании методов сжатия на основе кодирования кадровой разности любая ошибка размножается на некоторую последовательность кадров, в результате чего становится заметнее оператору. Количество искаженных под действием одной ошибки кадров зависит от периодичности передачи опорных кадров (с учетом вероятности ошибки и коэффициента сжатия) и от местоположения ошибки в последовательности кадров. Чем дальше искаженный кадр от опорного, тем меньше проявляется ошибка (на меньшее число кадров она проецируется). Как и для методов сжатия на основе покадрового кодирования, ошибка имеет пространственную локализацию – ее проявление ограничивается кодовым блоком 8x8 или 64x64 пикселя в зависимости от используемого метода сжатия кадровой разности, например JPEG или JPEG 2000.

Методы сжатия на основе предсказания занимают промежуточное положение по заметности ошибок передачи после декодирования между методами сжатия на основе покадрового кодирования и кодирования кадровой разности. Это обеспечивается за счет повышения коэффициента сжатия и снижения вероятности искажения опорных кадров. В наименьшей степени эффект размножения ошибок проявляется в H.264, использующем блочную помехоустойчивую структуру потока. Следует также учесть, что видеоизображения имеют значительную естественную пространственную, временную и статистическую

избыточность, которую сжатие не может полностью устранить. Избыточность играет положительную роль в повышении устойчивости к ошибкам.

Если при вероятности ошибки в радиоканале 10^{-6} и более качество воспроизведения видеоизображений будет неудовлетворительным, то необходимо использовать дополнительное помехоустойчивое кодирование на представительном уровне базовой эталонной модели OSI. То же относиться и к криптографическому кодированию. Оценка необходимости использования дополнительного помехоустойчивого кодирования на представительном и выбор конкретного кода требует проведения практических исследований в реальных условиях использования системы.

4.3 Требования к полосе пропускания цифрового канала и вероятности ошибки передачи видеоданных при использовании цифровой видеокамеры на борту БЛА

Проведен анализ параметров цифровых радиоканалов, пригодных для передачи видеоизображений с борта БЛА [26].

Радиоканал имеет жесткие ограничения по скорости передачи и дальности, обусловленные массогабаритными и энергетическими характеристиками радиопередающей аппаратуры, частотным диапазоном и условиями распространения радиоволн. Единственным в своем роде стандартом, устанавливающим требования к радиоканалам передачи видеоданных с подвижных объектов, является STANAG 4609, разработанный НАТО в 2005 году и постоянно модифицируемый [39, 40]. Стандартом предусмотрена передача видеоданных на скоростях от 32 Кбит/с до 1485 Мбит/с. При этом возможно использовать стандартную аппаратуру радиопередачи.

В таблице 4.1 скорости передачи видеоданных по радиоканалам, установленные STANAG 4609, сопоставлены с нижними частотными диапазонами и полосами частот подвижных радиослужб [11].

Полоса радиоканала, необходимая для передачи видеоданных, определена в таблице 4.1 исходя из соотношения 1 Гц на 1 бит/с. Это является достаточно хорошим приближением. Для технологии Wi-Fi, например, скорости передачи от 1 Мбит/с до 54 Мбит/с соответствует полоса радиоканала 22 МГц при радиусе зоны уверенного приема 300 м на открытой местности [42]. Для технологии WiMAX скорости передачи 20 Мбит/с в движении на скорости до 120 км/ч (от 40 Мбит/с до 134,4 Мбит/с в стационарном варианте доступа) соответствует полоса радиоканала 25 – 28 МГц при радиусе зоны уверенного приема 3 км [42]. В обоих примерах используются ненаправленные антенны. При использовании направленных антенн расстояние между передатчиком и приемником определяется прямой видимостью.

Таблица 4.1 – Характеристики радиоканалов для передачи видеоданных

Скорость передачи, Мбит/с	Нижний частотный диапазон, МГц	Полоса частот, МГц	Диапазон / размер антенны	Мощность / габариты / вес передатчика	Условия связи
1485	(24–27,5)10^3	(25,5–27) 10^3	СВЧ (см) / 0,3 м	50 Вт / 30х25х10 см / 4 кг	Прямая видимость, узконаправленные антенны
360	5725–7075	6700–7075	УВЧ (дм) / 0,1 м	18 Вт / 20х15х4 см / 0,7 кг	
270	4500–5250	4500–4800			
80	1710–2025	1710–1930	УВЧ (дм) / 0,1 м	18 Вт / 20х15х4 см / 0,7 кг	Прямая видимость, направленные антенны
40	1215–1427	1350–1400			
25	174–273	235–267	ОВЧ (м) / 1 м	25 Вт / 10х10х20 см / 1,5 кг	Прямая видимость, направленные антенны
19,4	174–273	235–267			
15	174–273	235–267			
12	174–273	235–267			
10	174–273	235–267			
6	27,5–44	30,01–37,5	ОВЧ (м) / 1,5 м	100 Вт / 20х5х20 см / 3,5 кг	Близко к прямой видимости
5,5	27,5–44	30,01–37,5			
3	27,5–44	30,01–37,5			
1,5	27,5–44	30,01–37,5			
1	27,5–44	30,01–37,5			
0,512	7,1–9,995	7,45–8,1	ВЧ (декам) / 2 м	120 Вт / 20х10х20 см / 4 кг	Связь на большие расстояния
0,256	7,1–9,995	7,45–8,1			
0,128	2,3–2,85	2,3–2,498	СЧ (гктм) / 2 м	120 Вт / 40х10х30 см / 15 кг	Связь на большие расстояния
0,032	2,3–2,85	2,3–2,498			

В таблице 4.1 приведены также размеры антенн [43–48], энергетические и массогабаритные характеристики радиопередатчиков [49–51], условия передачи. Эти параметры ограничивают области использования объектов-носителей камер.

Для лёгких БЛА основные ограничения связаны с массогабаритными (1–2 кг) и энергетическими (питание от аккумулятора) характеристиками, а также размером антенн (0,1 м). Возможный диапазон частот – УВЧ (дм), в котором обеспечивается скорость от 40 до 360 Мбит/с и дальность передачи до нескольких десятков километров при использовании направленных антенн.

Для тяжелых летательных аппаратов (тяжелых БЛА, самолётов, больших аэростатов) доступен практически весь частотный диапазон вплоть до УВЧ, что открывает возможности выбора пропускной способности канала и обеспечения дальности передачи в широком диапазоне. Так как допустимо использовать большие антенны возможна передача на значительные расстояния, в том числе на скоростях до 0,5 Мбит/с при отсутствии прямой видимости.

Скорость формирования видеоинформации определяют три основных фактора: битовая глубина, частота и размер кадра. Чем они больше, тем выше скорость формирования видеоинформации. В принципе, повышение коэффициента сжатия видеоданных возможно за счет уменьшения этих параметров. Однако для кодеков MPEG-2, MPEG-4 и H.264 снижение частоты кадров возможно лишь в несколько раз из-за ограничений со стороны алгоритмов блочной компенсации движения. Стандартами НАТО максимальный коэффициент сжатия для кодека H.264 при частоте 12 кадров/с установлен равным 5200 раз [39]. Кроме того, частота кадров должна быть согласована со скоростью движения носителей и объектов видео-регистрации для обеспечения непрерывности воспроизведения движения (если этого требует последующая обработка или условия применения).

На качество воспроизведения существенное влияние оказывает скорость экспозиции кадров, ограничивающая максимальную кадровую частоту. Минимально допустимая скорость экспозиции кадров определяется допустимым смазом на изображении и зависит от скорости перемещения камеры, зависящей, в свою очередь, от скорости носителя и собственной скорости вращения камеры, а также от расстояния до объекта видео-регистрации. Наибольший смаз возникает в условиях, когда оптическая ось камеры перпендикулярна плоскости видео-регистрации (плоскости, в которой находится или перемещаются один или несколько объектов наблюдения).

Минимально допустимый размер кадра определяется задачей обработки многоракурсных изображений и зависит от угла обзора камеры и расстояния до объекта наблюдения. Минимальный размер изображения объекта наблюдения составляет: для задач обнаружения, сопровождения, определения местоположения – 4 пикселя; для задач идентификации, трехмерной реконструкции и панорамирования – около 70% размера кадра (в пикселях).

Поскольку для качественной передачи видеоданных требуется значительная пропускная способность, наиболее проблематична организация передачи многоракурсных изображений в носимых и возимых системах видеонаблюдения, работающих в СЧ и ВЧ диапазонах. Снижение частоты кадров в данном случае приводит к уменьшению эффективности кодеков MPEG-2, MPEG-4 и H.264. В таком случае более эффективным является использование кодеков MJPEG, MJPEG2000.

Для лёгких и тяжёлых БЛА возможно применение, как H.264/MPEG-4 при высокой кадровой частоте (с учетом скорости движения камеры) и стандартном разрешении изображений, так и MJPEG, MJPEG2000 при низкой кадровой частоте и высоком разрешении изображений.

Причем задача сжатия изображений остается актуальной даже при повышении пропускной способности каналов в системах на базе летательных аппаратов. В связи с их удалённостью от объектов наблюдения необходимо повышение разрешения изображений, а с учетом высокой скорости движения – повышение кадровой частоты, что увеличивает объем формируемых видеоданных от одной камеры. Задача передачи изображений усложняется тем, что на одном тяжелом летательном аппарате таких камер может быть установлено несколько десятков (как на современных американских самолетах-разведчиках и истребителях [39, 40]).

Отдельная проблема – обеспечение помехоустойчивости, т.к. вероятность ошибки в радиоканале составляет, как правило, 10^{-3} – 10^{-4}. Это приводит к тому, что практически ни один видеокадр стандартного и тем более высокого разрешения при небольшом коэффициенте сжатия не может быть передан без ошибки. Поэтому значительная доля полосы радиоканала (30–70 %) должна быть отведена для помехоустойчивого кодирования. В случае РЭБ полоса, доступная для передачи видео, сужается еще больше.

4.4 Возможности и оценка эффективности использования криптографического и помехоустойчивого кодирования при передаче цифрового видеопотока с борта БЛА

Необходимость защиты информации обусловлено увеличением стоимости потери конфиденциальной информации, нарушения ее целостности и доступности. Для выполнения большинства требований информационной безопасности разработан ряд эффективных алгоритмов симметричного блочного и поточного шифрования данных (ГОСТ 28147-89, DES, 3-DES, IDEA, AES, RC5, RC6 и RC4) и асимметричного блочного шифрования данных (RSA, ECC). Поиск новых технологий защиты обусловлен стремлением не зависеть от существующих стандартов и «нерешенных» математических проблем, которые могут перестать быть препятствием перед несанкционированным пользователем.

Генераторы псевдослучайных последовательностей (ПСП) являются важнейшими элементами любой системы защиты, надежность которой в значительной степени определяется свойствами используемых генераторов. Одно из требований, предъявляемое к современным генераторам: генерируемая ПСП должна быть статистически неотличима от абсолютно случайной.

Для повышения характеристик эффективности шифрования мультимедийного контента с учетом его перцептуальных особенностей разработаны методы генерации статистически безопасных квантованных и бинарных хаотических генераторов, основанные на учете нелинейных закономерностей влияния хаотических свойств (эргодичности, чувствительности к начальным условиям, динамической и структурной сложности) на криптографические свойства (перемешивание, рассеяние, детерминированную псевдослучайность, алгоритмическую сложность) посредством выбора множества хаотических функций с определенными начальными параметрами, установлении определенной взаимосвязи между его хаотическими функциями и обработки хаотических последовательностей, что позволяет формировать вещественные и бинарные хаотические последовательности и матрицы хаотических перестановок с улучшенными криптографическими свойствами (увеличение множества начальных ключевых параметров пропорционально количеству начальных параметров хаотических функций и степени случайности последовательностей пропорционально нормированной энтропии распределения векторов в пространстве) [52, 53].

Установлено, что лучшими статистическими свойствами обладают хаотические последовательности (ХП), формируемые параллельным бинарным генератором в понятиях методики трехуровневой оценки качества генерации. Показано, что генератор с кольцевой структурой обеспечивает приблизительно одинаковые значения аппроксимационной энтропии квантованной ХП вне зависимости от базовой хаотической функции и позволяет равномерно рассеивать выходную ХП на плоскости независимо от типа ХФ, используемой для его построения. Определено, что оптимальные значения статистических параметров кольцевого генератора достигается при числе раундов, равном 5. что согласуется с результатами успешного прохождения теста NIST.

Эффективным средством уменьшения уязвимости мультимедийной информации (ММИ) является переход от принципа независимого к принципу согласованного сжатия, шифрования и помехоустойчивого кодирования с учетом структуры данных источника ММИ. Одним из необходимых условий эффективного согласования процессов сжатия, шифрования и помехоустойчивого кодирования является формирование масштабируемой структуры сжатого битового потока по качеству и разрешению.

Возникающие при решении проблемы объединения шифрования с мультимедийными системами сжатия трудности методически обусловлены тем, что используемые традиционные алгоритмы кодирования и шифрования и принципы их независимого и последовательного применения не учитывают перцептуальные и корреляционно-статистические особенности мультимедийных данных и не удовлетворяют современным требованиям по обеспечению их передачи и обработки.

Для комплексной защиты контента изображений от несанкционированного доступа и случайных канальных помех с возможностью формирования версий изображения с различным качеством и пространственным разрешением с целью адаптации к условиям передачи, доступа и воспроизведения, увеличения количества секретных ключей (порядка 2^{256}), устойчивости к коалиционным атакам по секретному ключу, управления соотношением быстродействие/уровень защищенности и качественного восстановления изображения (PSNR>32 дБ, SSIM>0.9) при низком отношении сигнал/шум в канале 1…3 дБ и минимальной избыточности кодирования (<20) определены необходимые условия совместного согласованного использования алгоритмов сжатия, шифрования и помехоустойчивого кодирования: формирование масштабируемой структуры сжатого битового потока по качеству и разрешению с различной чувствительностью к ошибкам, иерархической структуры секретных ключей и перцептуальной иерархии пакетов битового потока для селективного шифрования и кодирования.

Выявлено, что прогрессивное и частичное шифрование битовых плоскостей, частичное шифрование поддиапазонов уровня разложения и их параметров (знаки, позиции или уточняющие биты вейвлет-коэффициентов) позволяют достичь высокой степени деградации изображения (PSNR<16 дБ и SSIM<0,18).

Установлено, что для модели канала связи с AWGN и модуляции BPSK использование селективного кодирования, основанного на схеме перцептуального разделения битового потока 3-3-3 и соответствующих ей LDPC-IRA (irregular repeat-accumulate, нерегулярный код с повторной аккумуляций) кодов с $R_{c1} = 1/2$, $R_{c2} = 8/9$ и $R_{c3} = 8/9$, при вносимой избыточности, равной 13% приводит к выигрышу 2,5…15,6 дБ в понятии *PSNR* и 0,07…0,58 при $E_b / N_0 \in \{0,\ldots 3,.55\}$ дБ в понятии *SSIM* по сравнению с равномерным кодированием с LDPC-IRA ($R_c = 8/9$). Выявлено, что при селективном помехоустойчивом кодировании биты высокой психовизуальной значимости будут восстанавливаться без ошибок при ОСШ 1 дБ в канале, биты средней значимости восстанавливаются без ошибок при уровне шума 2 дБ и биты низкой значимости восстанавливаются без ошибок при уровне шума 4 дБ.

ЗАКЛЮЧЕНИЕ

Основные результаты НИР заключаются в следующем.

1 Разработан пространственно-частотный ковариационный метод поиска малоразмерных целей на кадрах видеопоследовательности, основанный на вычислении непрореженного дискретного лифтинг вейвлет-преобразования Хаара. Показано, что предложенный метод по сравнению с пространственным ковариационным методом обеспечивает повышение скорости обработки кадров от 7,2 до 8,3 раз. Установлено, что по сравнению с пространственным ковариационным методом предложенный пространственно-частотный ковариационный метод позволяет увеличить вероятность правильного обнаружения цели от 13,9 % до 15,5 % при понижении кадровой частоты видеопоследовательности, повысить устойчивость к зашумлению в среднем на 10,7 % и повысить устойчивость к изменению контрастно-яркостных характеристик видеопоследовательности на 4,4 %.

2 Разработан метод, алгоритм и программные средства прогрессивной сегментации изображений на основе реверсивной кластеризации. Метод обеспечивает адаптацию к ограничениям вычислительных ресурсов и времени вычислений за счет сегментации изображения на каждом уровне его кратномасштабного представления. Показано, что метод позволяет устранить ошибки сегментации сложных по структуре областей за счет выявления соседних однородных областей, имеющих одинаковую среднюю яркость, но различные номера сегментов. Разработаны алгоритм и программное средство улучшения качества и нормализации кадров видеопотока с борта БЛА.

3 Для совмещения изображений с борта БЛА с фотопланом предложен метод кадровой компенсации движения видеокамеры. Сущность метода состоит в поиске фрагмента фотоплана, соответствующего прогнозируемому кадру, и использовании координат этого фрагмента и коэффициентов гомографии для формирования прогнозного кадра, замещающего прогнозируемый кадр. Установлено, что кадровая компенсация движения позволяет повысить качество предсказания в 5,3 раза по отношению к блочной компенсации движения за счет роста вычислительной сложности. Экспериментально установлено, что кадровая компенсация движения требует в 3,4 раза больше времени, чем блочная. При этом, однако, проигрыш в быстродействии кадровой компенсации движения по сравнению с блочной меньше выигрыша в качестве, что говорит, в целом, об эффективности предложенного метода.

4 Предложен метод ориентации двух перекрывающихся изображений на основе фазовых гистограмм прямых линий для определения перемещения видеокамеры. Сущность

метода состоит в локализации и параметризации прямых линий на изображениях, формировании фазовых гистограмм прямых линий и их использовании для вычисления угла поворота одного изображения относительно другого. Метод обеспечивает ориентацию изображений, условия формирования которых существенно отличаются сезонностью, временем суток, метеорологической обстановкой и ракурсом, за счет использования прямых линий, более устойчивых по сравнению с реперными точками к изменению яркости, контраста, резкости, параллаксу. Метод может использоваться для совмещения изображения с борта БЛА с электронной картой местностью. Экспериментально установлено, что затраты времени на ориентацию изображений с использованием фазовых гистограмм прямых линий в 1,7 раза меньше по сравнению с ориентацией по реперным точкам с использованием методов SURF/RANSAC. Показано, что предложенный метод обеспечивает корректную ориентацию изображений в более широком диапазоне изменения яркости и контраста по сравнению с методами SURF/RANSAC (диапазон изменения яркости и контраста, обеспечивающий корректную ориентацию изображений, расширяется на 60 %). Экспериментально установлено, что при использовании фильтра Гаусса с размером ядра до 60x60 пикселей ошибка ориентации тестовых изображений с помощью предложенного метода на основе фазовых гистограмм прямых линий не превышает 5 %. Для сравнения, методы SURF/RANSAC обеспечивают позиционирование после обработки второго тестового изображения фильтрами Гаусса с размером ядра не более 4x4 пикселей, что в 15 раз хуже по сравнению с предложенным методом на основе фазовых гистограмм.

5 Разработан алгоритм расчета координат неподвижного объекта в кадре с учетом рельефа местности, радиальной дисторсии и положения главной точки. Установлены следующие возможности повышения точности сопровождения цели с БЛА: использование цифровых способов формирования и передачи изображений, высокой кадровой частоты и высокого разрешения видеокамеры; большой битовой глубины видеокадра; использование стабилизации видеокамеры; повышение точности телеметрии; повышение эффективности методов и алгоритмов поиска цели.

6 Проведен анализ параметров цифровых радиоканалов, пригодных для передачи видеоизображений с борта БЛА. Показано, что для лёгких БЛА основные ограничения связаны с массогабаритными (1–2 кг) и энергетическими (питание от аккумулятора) характеристиками, а также размером антенн (0,1 м). Возможный диапазон частот – УВЧ (дм), в котором обеспечивается скорость от 40 до 360 Мбит/с и дальность передачи до нескольких десятков километров при использовании направленных антенн. Для тяжелых летательных аппаратов (тяжелых БЛА, самолётов, больших аэростатов) доступен практически весь частотный диапазон вплоть до УВЧ, что открывает возможности выбора

пропускной способности канала и обеспечения дальности передачи в широком диапазоне. Так как допустимо использовать большие антенны возможна передача на значительные расстояния, в том числе на скоростях до 0,5 Мбит/с при отсутствии прямой видимости.

7 Проведен анализ методов сжатия для передачи видеоданных с борта БЛА. Показано, что для лёгких и тяжёлых БЛА возможно применение, как H.264/MPEG-4 при высокой кадровой частоте (с учетом скорости движения камеры) и стандартном разрешении изображений, так и MJPEG, MJPEG2000 при низкой кадровой частоте и высоком разрешении изображений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] F. Porikli, O. Tuzel. Multi-kernel object tracking // Proceedings of IEEE Int'l. Conference on Multimedia and Expo. 2005. P. 1234–1237.

[2] S. Avidan. Ensemble tracking // Proc. IEEE Conf. on Computer Vision and Pattern Recognition. 2005. Vol. 2. P. 494–501.

[3] F. Porikli, O. Tuzel, P. Meer. Covariance tracking using model update based means on Riemannian manifolds // Proc. IEEE Conf. on Computer Vision and Pattern Recognition. 2006. Vol. 1. P. 728–735.

[4] S. Ribaric, G. Adrinek, S. Segvic, Real-time active visual tracking system // Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference. 2004. P. 231–234.

[5] Y. Wang, Z. Hou, K. Leman, R. Chang. Real-Time Video Stabilization for Unmanned Aerial Vehicles // IAPR Conference on Machine Vision Applications. 2011. P. 336–339.

[6] Борискевич, И.А. Оценка эффективности использования дискретного лифтинг вейвлет-преобразования Хаара для сопровождения малоразмерных целей / И.А. Борискевич, В.Ю. Цветков, Ф.А.-К.М. Аль-Хелли // Телекоммуникации: сети и технологии, алгебраическое кодирование и безопасность данных: материалы междунар. научно-технич. семинара. Минск, апрель–декабрь 2013 г. – Мн.: БГУИР, 2013. – С. 16-21.

[7] Борискевич, И.А. Сопровождение малоразмерных целей с нестационарной видеокамеры на основе ковариационных признаков и предсказания / И.А. Борискевич, В.Ю. Цветков // Доклады БГУИР. – № 3 (81), 2014. – С. 33 - 39.

[8] Dimiccoli, M. Hierarchal Region-Based Representation for Segmentation And Filtering With Depth in Single Image / M. Dimiccoli, Ph. Salembier // IEEE Trans., Barcelona, Spain, ICIP, 2009. – P.3533–3536.

[9] Гонсалес, Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс. – М.: Техносфера, 2005. – 1072 с.

[10] Цветков, В.Ю. Прогрессивная сегментация изображений на основе реверсивной кластеризации / В.Ю. Цветков, О.М. Альмияхи, Т.М. Аль-Джубури // Развитие информатизации и государственной системы научно-технической информации (РИНТИ-2013): доклады XIII Международной конференции (Минск, 20 ноября, 2014 г.). – Минск: ОИПИ НАН Беларуси, 2014. – С. 134-139.

[11] Interactive image segmentation by maximal similarity based region merging / J. Ning [et al.] // The Journal of Pattern Recognition Society. – 2010. – P. 445–456.

[12] Журавлёв, А.А. Кадровая компенсация движения видеокамеры на основе фотоплана / А.А. Журавлёв, В.Ю. Цветков // Доклады БГУИР. – № 1 (79), 2014. – С. 5 - 10.

[13] Hartley, R. Multiple view geometry in computer vision / R. Hartley, A. Zisserman. Cambridge University Press: 2nd edition, 2003. – 655 p.

[14] Lowe, D.G. Distinctive image features from scale invariant features / D.G. Lowe // International Journal of Computer Vision. – 2004. – Vol. 60, № 2. – P. 91–110.

[15] SURF: Speeded Up Robust Features / H. Bay [et al.] // Computer Vision and Image Understanding. – 2008. – Vol. 110, № 3. – P. 346–359.

[16] Cyganek, B. An introduction to 3D computer vision techniques and algorithms / B. Cyganek, J.P. Siebert. Wiley: 1st edition, 2009. – 502 p.

[17] Журавлёв, А.А. Ориентация аэрокосмических изображений на основе фазовых гистограмм прямых линий / А.А. Журавлев, В.Ю. Цветков, О.Дж. Аль-Фурайджи // Развитие информатизации и государственной системы научно-технической информации (РИНТИ-2013): доклады XII Международной конференции (Минск, 20 ноября, 2013 г.). – Минск: ОИПИ НАН Беларуси, 2013. – С. 134-139.

[18] Duda, R.O. Use of the Hough Transformation to Detect Lines and Curves in Pictures / R.O. Duda. – Communication of the ACM, 1972. – Vol. 15. – №1. – P. 229–246.

[19] Anver, M.M. Fuzzy edge detection using competition between multiple masks / M.M. Anver, R.J. Stonier // Proceedings of the 2nd International Conference on Computational Intelligence CIRAS 2003 (Singapore, 2003). – 2003. – P. 344–348.

[20] Rafael, G.G. LSD: A Fast Line Segment Detector with a False Detection Control / G.G. Rafael, J.M. Morel // IEEE Transactions on Pattern. Analysis and Machine Intelligence. – 2010. – Vol. 32. – №4. – P. 722–732.

[21] Raymond, K.K. Line detection algorithm using a simple tracing mechanism / K.K. Raymond, T.S. Chan // Proceeding of 91th Scandinavian Conference on Image Analysis, SCIA-1991 (Aalborg, Denmark, 1991). – 1991. – P. 1021–1028.

[22] Журавлев, А.А. Прогрессивная локализация прямых на изображениях с использованием ориентированного преобразования Хафа / А.А. Журавлев, В.Ю. Цветков // Телекоммуникации: сети и технологии, алгебраическое кодирование и безопасность данных : материалы междунар. научно-технич. семинара, Минск, декабрь 2011 г.) – Минск : БГУИР, 2011. – С. 74–81.

[23] Аль-Фурайджи, О.Дж. Секторная локализация, параметризация и идентификация реперов на основе угловых коэффициентов для совмещения перекрывающихся изображений / О.Дж. Аль-Фурайджи, В.К. Конопелько, В.Ю. Цветков // Доклады БГУИР, 2012. – № 6(68). – С. 122-128.

[24] Журавлев, А.А. Формирование устойчивых к изменению освещенности реперных структур аэрокосмических изображений на основе направляющих / А.А. Журавлев, В.Ю. Цветков // 5-я Междунар. науч. конфер. по военно-техническим проблемам, проблемам

обороны и безопасности, использованию технологий двойного применения : тез. докл., Минск, 25–26 мая 2011 г. – Минск : БелИСА, 2011. – С. 112–114.

[25] Журавлёв, А.А. Масочно-фазовый метод локализации прямых линий на изображении / А.А. Журавлёв, В.Ю. Цветков // Информатика. – 2014. – С. 85-96.

[26] Цветков, В.Ю. Предсказание, распознавание и формирование образов многоракурсных изображений с подвижных объектов / В.Ю. Цветков, В.К. Конопелько, В.А. Липницкий. – Минск: Изд. центр БГУ, 2014. – 223 с.

[27] Борискевич, И.А. Сопоставление матриц гомографии по телеметрии и видеоданным с борта БЛА / И.А. Борискевич, В.Ю. Цветков, С.В. Пручковский // Актуальные вопросы науки и техники в сфере развития авиации: Тезисы докладов IV Междунар. н.-т. конф. авиационного факультета УО «Военная академия Республики Беларусь» 15-16 мая 2014 г., Минск. – Минск: ВА РБ, 2014. – С. 186.

[28] Артюшенко, В.М. Цифровое сжатие видеоинформации и звука: учеб. пособ. / В.М. Артюшенко, О.И. Шелухин, М.Ю. Афонин; под ред. В.М. Артюшенко. – М.: Издательско-торговая корпорация «Дашков и Ко», 2003. – 426 с.

[29] Сэломон, Д. Сжатие данных, изображения и звука / Д. Сэломон. – М.: Техносфера. – 2004. – 360 с.

[30] Миано, Дж. Форматы и алгоритмы сжатия изображений в действии / Дж. Миано. – М.: Изд. Триумф, 2003 – 336 с.

[31] Блейхут, Р. Теория и практика кодов, контролирующих ошибки / Р. Блейхут. – М.: Мир, 1986. – 576 с.

[32] Золотарев, В.В. Помехоустойчивое кодирование. Методы и алгоритмы. Справочник / В.В. Золотарев, Г.В. Овечкин; под ред. Ю.Б. Зубарева. – М.: Телеком, 2004. – 126 с.

[33] Alois, M. Bock. Video Compression Systems / Bock Alois. Printed by Athenaeum Press Ltd, 2009. – 274 p.

[34] Pennebaker, W. B. JPEG Still Image Compression Standard / W. B. Pennebaker, J. L. Mitchell. – New York: Van Nostrand Reinhold, 1993. – 412 p.

[35] Joshi, R.L. Comparison of multiple compression cycle performance for JPEG and JPEG 2000 / R.L. Joshi, M. Rabbani, M. Lepley // Proc. of the SPIE, San Diego, CA, USA, July/August 2000. – Vol. 4115. P. 492-501.

[36] Уэлстид, С. Фракталы и вейвлеты для сжатия изображений в действии / С. Уэлстид. – М.: Издательство Триумф, 2003. – 230 с.

[37] Watkinson, J. The MPEG handbook: MPEG-1 MPEG-2 MPEG-4 / J. Watkinson // Focal Press: First edition, 2001. – 244 p.

[38] Ричардсон, Я. Видеокодирование H.264 и Mpeg-4 – стандарты нового поколения / Я. Ричардсон. – М: Техносфера, 2005. – 369 с.

[39] STANAG 4609 NATO Digital Motion Imagery Standard // NATO Standardization Agency, Brussels, Belgium: 1st edition, 2005. – 32 p.

[40] STANAG 4609 NATO Digital Motion Imagery Standard // NATO Standardization Agency, Brussels, Belgium: 3rd edition, 2009. – 73 p.

[41] Таблица распределения полос радиочастот между радиослужбами. – [Электронный ресурс] – Режим доступа: http://www.grfc.ru/grfc/sprav_info/ tools_1/005108. – Дата доступа: 13.03.2013.

[42] Пахомов, С. Мобильный WiMAX приходит в Россию / С. Пахомов // Компьютер-пресс, 2008. – № 2. – [Электронный ресурс] – Режим доступа: http://www.compress.ru/article.aspx?id=18639&iid=865. – Дата доступа: 13.03.2013.

[43] Антенна Diamond MD200. – [Электронный ресурс] – Режим доступа: http://www.mir-racii.ru/good_3625.html. – Дата доступа: 10.03.2013.

[44] Антенна Diamond MD4020. – [Электронный ресурс] – Режим доступа: http://www.mir-racii.ru/good_3626.html. – Дата доступа: 10.03.2013.

[45] Антенна Sirio TURBO 2000LB. – [Электронный ресурс] – Режим доступа: http://www.mir-racii.ru/good_1683.html. – Дата доступа: 10.03.2013.

[46] Дуров, А.А. Судовые УКВ-радиостанции: учебное пособие / Дуров А.А., Рябышкин В.Н. – Петропавловск-Камчатский: КамчатГТУ, 2002. – 94 с.

[47] Антенна ST-AF4-20. – [Электронный ресурс] – Режим доступа: http://www.synertec.ru/products.html. – Дата доступа: 10.03.2013.

[48] Антенна R&S HF907OM – [Электронный ресурс] – Режим доступа: http://www.rohde-schwarz.ru/products/radiomonitoring/antennas/HF907OM/Brief. – Дата доступа: 11.03.2013.

[49] Радиостанции YAESU – [Электронный ресурс] – Режим доступа: http://www.radio-admin.ru/category.php?curcat=1&makers=YAESU&parametrs= HF+%280.5-30+%CC%C3%F6%29]. – Дата доступа: 14.03.2013.

[50] Маршрутизатор Asus Mobile WiMAX/Wi-Fi Center. – [Электронный ресурс] – Режим доступа: http://catalog.yotatester.ru/router/Asus_Mobile_WiMAX_Wi-Fi_Center. – Дата доступа: 14.03.2013.

[51] Радиорелейные системы PDH. – [Электронный ресурс] – Режим доступа: http://digital-microwave-radio.at-communication.com/na/pdh-digital-microwave-radio.html. – Дата доступа: 14.03.2013.

[52] Конопелько, В.К. Многомерные технологии сжатия, защиты и коммутации изображений: монография / В.К. Конопелько, А.А. Борискевич, В.Ю. Цветков. – Минск: Белпринт, 2008. – 162 с

[53] Борискевич А. А. Голографическая защита информации / А. А. Борискевич, В. К. Ероховец, В. В. Ткаченко Объединенный институт проблем информатики Национальной академии наук Беларуси. – Минск : ОИПИ НАН Беларуси, 2012. – 279 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Описание программных средств поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА

Для поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА использованы следующие функции:

- void CalculateCovariance(int position_x, int position_y, int size_x, int size_y, int down_x, int down_y) – для вычисления элементов ковариационной матрицы. Входные параметры: пиксельные координаты и размеры объекта наблюдения
- void FindContourCooord(int window_height, int window_width, bool useTheshold, int &left, int &top, int &right, int &bottom) – для стабилизации рамки вокруг найденной позиции объекта наблюдения. Входные параметры: размеры области поиска и величина порога.
- void ProcessFrame(IplImage* sourceFrame, TrajectoryOffsetVector lastInterframeOffset) – для определения пиксельных координат объекта наблюдения на видеокадре с учетом пиксельной коррекции, вычисленной в блоке стабилизации видеопоследовательности.

Листинг файлов для поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА:

Заголовочный файл ObjectTracker.h:

```
#pragma once
#include "StdAfx.h"
#include "CommonData.h"

class ObjectTracker
{
private:
	IplImage* frame_GS;

	// object position & size
	int _defaultTargetLockOnFrameSize, _targetLockOnFrameSize;
	ScreenCoord _object_position, _object_position_previous /* ???? */,
_object_position_predicted;
	int _object_size_X, _object_size_Y;

	// features for template & candidates search area
	IplImage *template_32F, *candidate_32F; // 32-bit float images
	IplImage *template_dx, *template_dy, *template_dxx, *template_dyy, *template_value;
	IplImage *candidate_dx, *candidate_dy, *candidate_dxx, *candidate_dyy,
*candidate_value;
	IplImage *template_vector_value, *template_vector_dx, *template_vector_dy,
*template_vector_dxx, *template_vector_dyy;

	IplImage * _segmentation_image;
	CvMemStorage * _segmentation_memory;

	// pointers for reshaping into input for cvCalcCovarMatrix
	IplImage **template_input, **candidate_input;
```

```
        // covariance matrices & vectors of feature means for template & candidates
        CvMat *template_covariance_matrix;
        CvMat *candidate_covariance_matrix;

        CvMat *inv_candidate_covariance_matrix;
        CvMat *matrix_multiplication;
        CvMat *eigenvector;
        CvMat *eigenvalues;
        CvMat *eigenvalues_log;
        CvMat *eigenvalues_pow;

        TrajectoryRingBuffer * _objectFrameShiftBuffer;

        bool _targetLocked;

        void CalculateCovariance(int position_x, int position_y, int size_x, int size_y, int down_x,
int down_y);
        void FindContourCooord(int window_height, int window_width, bool useTheshold, int
&left, int &top, int &right, int &bottom);
public:
        ObjectTracker(void);
        ~ObjectTracker(void);
        void ProcessFrame(IplImage* sourceFrame, TrajectoryOffsetVector lastInterframeOffset);
        bool TargetLockOn(int x, int y);
        void DropTarget();
        void SetTargetRectSize(int size);
        int GetTargetRectSize();
        ScreenCoord GetObjectPosition();
};
```

Файл исходного кода ObjectTracker.cpp:

```
#include "StdAfx.h"
#include "CommonFunctions.h"
#include "ObjectTracker.h"
#include "PerformanceAnalyzer.h"

double const INFINITY2 = 100000;
int const MAX_FRAME_OBJECT_SHIFT_COUNT = 50;

// tracker settings
const double overlap_factor = 0.4; // set the part of non-overlapping area of candidate window
const int window_factor = 2; // set the number of candidate windows: 1 is equal to 9 candidate
windows, 2 to 25, 3 to 49 etc.
const double position_fixation_factor = 1.5; // range is 0-1: set the fixation weight to predicted
position
const double prediction_shift_factor = 0.5; // range is 0-1: set the maximum shift of predicted
position
```

```
const int NUMBER_OF_OBECT_FEATURES = 5; // number of object features (also check class for covarince calculation)

uchar* mcvPtr2D(const IplImage* img, int y, int x)
{
	uchar* ptr = 0;

	int pix_size = (img->depth & 255) >> 3;
	if (img->dataOrder == 0)
		pix_size *= img->nChannels;
	int width, height;
	ptr = (uchar*)img->imageData;

	if (img->roi)
	{
		width = img->roi->width;
		height = img->roi->height;
		ptr += img->roi->yOffset * img->widthStep + img->roi->xOffset * pix_size;
	}
	else
	{
		width = img->width;
		height = img->height;
	}

	ptr += y * img->widthStep + x * pix_size;

	return ptr;
}

void CopyElementValue(const IplImage* fromImg, int fromY, int fromX, const IplImage* toImg, int toY, int toX)
{
	uchar * ptr_element_value = mcvPtr2D(toImg, toY, toX); // destination data pointer
	uchar * ptr_value = mcvPtr2D(fromImg, fromY, fromX); // source data pointer
	((float*)ptr_element_value)[0] = ((float*)ptr_value)[0];
}

ObjectTracker::ObjectTracker(void)
{
	_targetLocked = false;
	frame_GS = 0;
	_objectFrameShiftBuffer = new TrajectoryRingBuffer(MAX_FRAME_OBJECT_SHIFT_COUNT);;
	_defaultTargetLockOnFrameSize = 30;
	_targetLockOnFrameSize = _defaultTargetLockOnFrameSize;
	template_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES, NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	candidate_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES, NUMBER_OF_OBECT_FEATURES, CV_32FC1);
```

```
	inv_candidate_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	matrix_multiplication = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	eigenvector = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	eigenvalues = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);
	eigenvalues_log = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);
	eigenvalues_pow = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);

	template_dx = 0;
	template_dy = 0;
	template_dxx = 0;
	template_dyy = 0;
	template_value = 0;
	template_32F = 0;

	candidate_dx = 0;
	candidate_dy = 0;
	candidate_dxx = 0;
	candidate_dyy = 0;
	candidate_value = 0;
	candidate_32F = 0;

	_segmentation_memory = cvCreateMemStorage(0);
	_segmentation_image = 0;
}

ObjectTracker::~ObjectTracker(void)
{
	delete _objectFrameShiftBuffer;
}

inline void InitOrCreateImage(IplImage ** image_ref, CvSize area_ROI_size, int depth, int
channels)
{
	if (image_ref == 0)
		CV_Error(CV_StsNullPtr, "");

	if (*image_ref == 0)
	{
		*image_ref = cvCreateImage(area_ROI_size, depth, channels);
	}
	else if ((area_ROI_size.width != (*image_ref)->width) || (area_ROI_size.height !=
(*image_ref)->height))
	{
		cvReleaseImage(image_ref);
		*image_ref = cvCreateImage(area_ROI_size, depth, channels);
	}
}

void InitMatrixes(IplImage * frameGS, int x, int y, int width, int height,
```

```
	IplImage ** matrix_32F, IplImage ** matrix_value, IplImage ** matrix_dx, IplImage **
matrix_dy, IplImage ** matrix_dxx, IplImage ** matrix_dyy)
{
	CvSize roiSize = cvSize(width, height);
	InitOrCreateImage(matrix_32F, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_dx, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_dy, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_dxx, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_dyy, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_value, roiSize, IPL_DEPTH_32F, 1);

	cvSetImageROI(frameGS, cvRect(x, y, width, height));
	cvConvertScale(frameGS, *matrix_32F);
	cvResetImageROI(frameGS);

	cvSobel(*matrix_32F, *matrix_dx, 1, 0, 3);
	cvSobel(*matrix_32F, *matrix_dy, 0, 1, 3);
	cvSobel(*matrix_32F, *matrix_dxx, 2, 0, 3);
	cvSobel(*matrix_32F, *matrix_dyy, 0, 2, 3);
	cvCopy(*matrix_32F, *matrix_value);
}

void CopyMatrixesElements(IplImage ** input_array, int size_x, int size_y,
	IplImage * matrix_value, IplImage * matrix_dx, IplImage * matrix_dy, IplImage *
matrix_dxx, IplImage * matrix_dyy)
{
#pragma omp parallel for num_threads(2)
	for (int y = 0; y < size_y; y++)
	{
		for (int x = 0; x < size_x; x++)
		{
			int i = x + y * size_x;
			CopyElementValue(matrix_value, y, x, input_array[i], 0, 0);
			CopyElementValue(matrix_dx, y, x, input_array[i], 0, 1);
			CopyElementValue(matrix_dy, y, x, input_array[i], 0, 2);
			CopyElementValue(matrix_dxx, y, x, input_array[i], 0, 3);
			CopyElementValue(matrix_dyy, y, x, input_array[i], 0, 4);
		} // for x
	} // for y
}

// calculate covariance matrix of candidates
void ObjectTracker::CalculateCovariance(int position_x, int position_y, int size_x, int size_y, int
down_x, int down_y)
{
	GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_CalculateCovariance);

	// translate frame coordinates to search area system
	int pos_x = position_x - down_x;
	int pos_y = position_y - down_y;

	CvRect roiRect = cvRect(pos_x, pos_y, size_x, size_y);
```

```
        cvSetImageROI(candidate_value, roiRect);
        cvSetImageROI(candidate_dx, roiRect);
        cvSetImageROI(candidate_dy, roiRect);
        cvSetImageROI(candidate_dxx, roiRect);
        cvSetImageROI(candidate_dyy, roiRect);

        // vectors of features for candidates
        // reshape feature images into vectors
        int input_height = size_x * size_y;

        CopyMatrixesElements(candidate_input, size_x, size_y, candidate_value, candidate_dx,
candidate_dy, candidate_dxx, candidate_dyy);

        // calculate covariance matrix
        cvCalcCovarMatrix((const void **)candidate_input, input_height,
candidate_covariance_matrix, 0, CV_COVAR_NORMAL | CV_COVAR_SCALE);

        // reset candidate ROI
        cvResetImageROI(candidate_dx);
        cvResetImageROI(candidate_dy);
        cvResetImageROI(candidate_dxx);
        cvResetImageROI(candidate_dyy);
        cvResetImageROI(candidate_value);

        GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_CalculateCovariance);
}

void ObjectTracker::DropTarget()
{
        _targetLocked = false;
}

void ObjectTracker::SetTargetRectSize(int size)
{
        _defaultTargetLockOnFrameSize = size;
}

int ObjectTracker::GetTargetRectSize()
{
        return _targetLockOnFrameSize;
}

void ObjectTracker::FindContourCooord(int window_width, int window_height, bool useTheshold,
int &left, int &top, int &right, int &bottom)
{
        InitOrCreateImage(&_segmentation_image, cvSize(window_width, window_height),
IPL_DEPTH_8U, 1);
        cvCanny(frame_GS, _segmentation_image, 2150, 4350, 5);
        if (useTheshold)
                cvAdaptiveThreshold(frame_GS, _segmentation_image, 255,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV, 21, 50);
```

```
	CvSeq *segmentation_contours;
	cvFindContours(_segmentation_image, _segmentation_memory, &segmentation_contours,
sizeof(CvContour), CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE, cvPoint(0, 0));

	CvPoint *contour_point = NULL;
	int min_x = (int)INFINITY2, min_y = (int)INFINITY2, max_x = 0, max_y = 0;

	for (; segmentation_contours != NULL; segmentation_contours = segmentation_contours-
>h_next)
	{
		for (int j = 0; j < segmentation_contours->total; j++)
		{
			contour_point = CV_GET_SEQ_ELEM(CvPoint, segmentation_contours, j);
			if (contour_point->x < min_x)
				min_x = contour_point->x;
			if (contour_point->y < min_y)
				min_y = contour_point->y;
			if (contour_point->x > max_x)
				max_x = contour_point->x;
			if (contour_point->y > max_y)
				max_y = contour_point->y;
		}
	}
	if (min_x < INFINITY2 && min_y < INFINITY2)
	{
		left = min_x - 2;
		top = min_y - 2;
		right = max_x + 2;
		bottom = max_y + 2;
	}
	else
	{
		top = _targetLockOnFrameSize / 2;
		left = _targetLockOnFrameSize / 2;
	}
}

bool ObjectTracker::TargetLockOn(int x, int y)
{
	GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_TargtLockOn);

	_targetLockOnFrameSize = _defaultTargetLockOnFrameSize;

	_object_position.x = x - _targetLockOnFrameSize / 2;
	_object_position.y = y - _targetLockOnFrameSize / 2;
	_object_size_X = _targetLockOnFrameSize;
	_object_size_Y = _targetLockOnFrameSize;

	if (_object_position.x < 0 || _object_position.y < 0 ||
		_object_position.x + _object_size_X > frame_GS->width || _object_position.y +
_object_size_Y > frame_GS->height)
		return false;
```

```
        // Find target contours
        cvSetImageROI(frame_GS, cvRect(_object_position.x, _object_position.y, _object_size_X,
_object_size_Y));
        int top_left_x = 0, top_left_y = 0, bottom_right_x = _object_size_X, bottom_right_y =
_object_size_Y;
        FindContourCooord(_object_size_X, _object_size_Y, false, top_left_x, top_left_y,
bottom_right_x, bottom_right_y);
        _object_position.x = _object_position.x + top_left_x;
        _object_position.y = _object_position.y + top_left_y;
        _object_size_X = bottom_right_x - top_left_x;
        _object_size_Y = bottom_right_y - top_left_y;
        _targetLockOnFrameSize = max(_object_size_X, _object_size_Y);

        cvDestroyWindow("LockedTarget");
        cvNamedWindow("LockedTarget", CV_WINDOW_AUTOSIZE);
        cvShowImage("LockedTarget", frame_GS);
        cvResetImageROI(frame_GS);


        InitMatrixes(frame_GS, _object_position.x, _object_position.y, _object_size_X,
_object_size_Y,
                &template_32F, &template_value, &template_dx, &template_dy, &template_dxx,
&template_dyy);

        // calculate the covariance matrix
        // vectors of features for template
        // reshape feature images into vectors
        int input_height = _object_size_X * _object_size_Y;
        CvSize input_width = cvSize(NUMBER_OF_OBECT_FEATURES, 1);

        // memory allocation for cvCalcCovarMatrix
        candidate_input = new IplImage*[input_height];
        template_input = new IplImage*[input_height];
        for (int i = 0; i < input_height; i++)
        {
                candidate_input[i] = cvCreateImage(input_width, IPL_DEPTH_32F, 1);
                template_input[i] = cvCreateImage(input_width, IPL_DEPTH_32F, 1);
        }

        CopyMatrixesElements(template_input, _object_size_X, _object_size_Y, template_value,
template_dx, template_dy, template_dxx, template_dyy);

        // calculate covariance matrix
        cvCalcCovarMatrix((const void **)template_input, input_height,
template_covariance_matrix, 0, CV_COVAR_NORMAL | CV_COVAR_SCALE);

        //??? Release images from template_input and candidate_input

        // set the previous & the predicted object position
        _object_position_previous.x = _object_position.x;
        _object_position_previous.y = _object_position.y;
```

```
	_object_position_predicted.x = _object_position.x;
	_object_position_predicted.y = _object_position.y;

	GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_TargtLockOn);
	_targetLocked = true;
	return true;
}

void ObjectTracker::ProcessFrame(IplImage* sourceFrame, TrajectoryOffsetVector
lastInterframeOffset)
{
	frame_GS = sourceFrame;

	if (!_targetLocked)
		return;

	GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_ProcessFrame);

	//!!! New code
	_object_position_predicted.x -= (int)lastInterframeOffset.x;
	_object_position_predicted.y -= (int)lastInterframeOffset.y;

	// determining the upper & the lower limits of search area
	int size_max = _object_size_X > _object_size_Y ? _object_size_X : _object_size_Y;
	int overlap_area = round2(double(size_max) * overlap_factor);
	// bottom X
	int down_limit_X = max(1, _object_position_predicted.x - window_factor * overlap_area);
	// bottom Y
	int down_limit_Y = max(1, _object_position_predicted.y - window_factor * overlap_area);
	// top X
	int up_limit_X = min(_object_position_predicted.x + window_factor * overlap_area,
frame_GS->width - _object_size_X);
	// top Y
	int up_limit_Y = min(_object_position_predicted.y + window_factor * overlap_area,
frame_GS->height - _object_size_Y);

	InitMatrixes(frame_GS, down_limit_X, down_limit_Y, up_limit_X + _object_size_X -
down_limit_X, up_limit_Y + _object_size_Y - down_limit_Y,
		&candidate_32F, &candidate_value, &candidate_dx, &candidate_dy,
&candidate_dxx, &candidate_dyy);

	int position_counter = -round2((1 + 2 * window_factor) ^ 2 / 2);

	double current_metric = INFINITY2; // initial value of minimized distance metric (=infinity)
	// search for all candidates inside the search area
	for (int candidate_position_Y = down_limit_Y; candidate_position_Y < up_limit_Y + 1;
candidate_position_Y += overlap_area)
	{
		for (int candidate_position_X = down_limit_X; candidate_position_X < up_limit_X
+ 1; candidate_position_X += overlap_area)
		{
			// more weight for predicted position
```

```
                        position_counter = position_counter + 1;
                        double position_weight = 1 + position_fixation_factor * position_counter / 10
/ ((1 + 2 * window_factor) ^ 2);

                        // calculate the covariance matrix
                        CalculateCovariance(candidate_position_X, candidate_position_Y,
_object_size_X, _object_size_Y, down_limit_X, down_limit_Y);

                        // distance metric
                        // find the eigenvalues
                        cvInvert(candidate_covariance_matrix, inv_candidate_covariance_matrix);
                        cvMatMul(template_covariance_matrix, inv_candidate_covariance_matrix,
matrix_multiplication);
                        cvEigenVV(matrix_multiplication, eigenvector, eigenvalues);

                        cvLog(eigenvalues, eigenvalues_log);
                        cvPow(eigenvalues_log, eigenvalues_pow, 2);
                        // metric calculation
                        CvScalar sum = cvSum(eigenvalues_pow);
                        double candidate_metric = cvSqrt(sum.val[0]) * position_weight;
                        // decision
                        if (candidate_metric < current_metric)
                        {
                                current_metric = candidate_metric;
                                _object_position.x = candidate_position_X;
                                _object_position.y = candidate_position_Y;
                        }
                } // for candidate_position_X
        } // for candidate_position_Y

        // frame stabilication by segmentation
        int stab_top, stab_left, stab_width, stab_height;

        if (_object_position.y - round2(_object_size_Y / 2) < 0)
                stab_top = 0;
        else
                stab_top = _object_position.y - round2(_object_size_Y / 2);
        if (_object_position.x - round2(_object_size_X / 2) < 0)
                stab_left = 0;
        else
                stab_left = _object_position.x - round2(_object_size_X / 2);
        if (_object_position.y - round2(_object_size_Y / 2) + 2 * _object_size_Y > frame_GS-
>height)
                stab_height = frame_GS->height - _object_position.y + round2(_object_size_Y / 2);
        else
                stab_height = 2 * _object_size_Y;
        if (_object_position.x - round2(_object_size_X / 2) + 2 * _object_size_X > frame_GS-
>width)
                stab_width = frame_GS->width - _object_position.x + round2(_object_size_X / 2);
        else
                stab_width = 2 * _object_size_X;
```

```
	if (stab_left > 0 && stab_top > 0 && stab_left + stab_width < frame_GS->width &&
stab_top + stab_height < frame_GS->height)
	{
		cvSetImageROI(frame_GS, cvRect(stab_left, stab_top, stab_width, stab_height));
		int segm_top = 0, segm_left = 0, segm_bottom = 0, segm_right = 0;
		FindContourCooord(stab_width, stab_height, true, segm_left, segm_top, segm_right,
segm_bottom);

		if ((segm_right - segm_left < _object_size_X + round2(_object_size_X) / 2 + 4) &&
(segm_bottom - segm_top < _object_size_Y + round2(_object_size_Y / 2) + 4))
		{
			_object_position.x = stab_left + segm_left;
			_object_position.y = stab_top + segm_top;
		}
	}
	//	cvShowImage("TEST", frame_GS);
	cvResetImageROI(frame_GS);

	// predict the object position on the next frame
	if ((abs(_object_position.x - _object_position_previous.x) < _object_size_X *
prediction_shift_factor) &&
		(abs(_object_position.y - _object_position_previous.y) < _object_size_Y *
prediction_shift_factor))
	{
		_object_position_predicted.x = _object_position.x + (_object_position.x -
_object_position_previous.x);
		_object_position_predicted.y = _object_position.y + (_object_position.y -
_object_position_previous.y);
	}
	else
	{
		_object_position_predicted.x = _object_position.x;
		_object_position_predicted.y = _object_position.y;
	}

	_object_position_previous.x = _object_position.x;
	_object_position_previous.y = _object_position.y;

	GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_ProcessFrame);
}

ScreenCoord ObjectTracker::GetObjectPosition()
{
	ScreenCoord screenCoord;
	if (_targetLocked)
		screenCoord.Init(_object_position.x, _object_position.y);
	else
		screenCoord.Init(-1, -1);
	return screenCoord;
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Описание программных средств определения параметров движения объекта наблюдения по его смещению на кадрах видеопотока и телеметрии

Для определения параметров движения объекта наблюдения по его смещению на кадрах видеопотока и телеметрии использованы следующие функции:

- static void CalculateCamCorrectionAngles(CamPreferences * camPreferences, ScreenCoord * targetCoord, double & angle_cor_X, double & angle_cor_Y) – для расчета углов смещения видеокамеры с целью наведения главной оптической оси на объект наблюдения Входные параметры: характеристики установленной на борту БЛА видеокамеры и пиксельные координаты объекта наблюдения.
- static void ConvertScreenCoord2WorldCoord(TelemetryData * telemetryData, CamPreferences * camPreferences, ScreenCoord * screenCoord, WorldGPSCoord * worldGPSCoord) – для расчета географических координат цели. Входные параметры: характеристики установленной на борту БЛА видеокамеры, данные телеметрии БЛА и пиксельные координат и пиксельные координаты объекта наблюдения.

Листинг файлов для определения параметров движения объекта наблюдения по его смещению на кадрах видеопотока и телеметрии:

Заголовочный файл CoordConverter.h:

```
#pragma once
#include "StdAfx.h"
#include "CommonData.h"
#include "CommonFunctions.h"

static void DecodeCoord(double coord, int& grad, int& min, double& sec)
{
	double absCoord = abs(coord);
	grad = int(absCoord);
	min = int(60.0 * (absCoord - grad));
	sec = (absCoord - grad - double(min) / 60.0) * 3600.0;
}

static double EncodeCoord(int grad, int min, double sec)
{
	double result = grad + double(min) / 60.0 + sec / 3600.0;
	return result;
}

static void GetGeoCoordAsStr(double coord, char minLetter, char maxLetter, char * str)
{
	int grad, min;
	double sec;
	DecodeCoord(coord, grad, min, sec);
	char site = coord <= 0 ? minLetter : maxLetter;
	sprintf(str, "%c %2d %2d' %5.2f\"", site, grad, min, sec);
}
```

```
static void CalculateCamCorrectionAngles(CamPreferences * camPreferences, ScreenCoord * targetCoord, double & angle_cor_X, double & angle_cor_Y)
{
	int frame_width = camPreferences->image_width_pix;
	int frame_height = camPreferences->image_height_pix;

	int object_shift_X = frame_width / 2 - targetCoord->x;
	int object_shift_Y = frame_height / 2 - targetCoord->y;
	angle_cor_X = object_shift_X * camPreferences->image_width_grad / frame_width;
	angle_cor_Y = object_shift_Y * camPreferences->image_height_grad / frame_width;
}

static void ConvertScreenCoord2WorldCoord(
	TelemetryData * telemetryData,
	CamPreferences * camPreferences,
	ScreenCoord * screenCoord,
	WorldGPSCoord * worldGPSCoord)
{
	double UAV_latitude = telemetryData->gps_lat;
	double UAV_longitude = telemetryData->gps_lon;
	double UAV_altitude = telemetryData->gps_hmsl;
	double UAV_pitch = telemetryData->pitch;
	double UAV_roll = telemetryData->roll;
	double UAV_yaw = telemetryData->yaw;
	double camera_pitch = telemetryData->cam_pitch;
	double camera_roll = telemetryData->cam_heading;

	double k_1 = camPreferences->radial_distortion_k_1;
	double k_2 = camPreferences->radial_distortion_k_2;
	double k_3 = camPreferences->radial_distortion_k_3;
	double deviation_X = camPreferences->center_deviation_X_pix;
	double deviation_Y = camPreferences->center_deviation_Y_pix;
	double X_cam = camPreferences->place_on_UAV_X_m;
	double Y_cam = camPreferences->place_on_UAV_Y_m;
	double Z_cam = camPreferences->place_on_UAV_Z_m;
	double half_frame_X_rad = camPreferences->image_width_rad() / 2;
	double half_frame_Y_rad = camPreferences->image_height_rad() / 2;
	double half_frame_X = camPreferences->image_width_pix / 2;
	double half_frame_Y = camPreferences->image_height_pix / 2;

	double R_m = UAV_altitude;
	double object_shift_X = half_frame_X - screenCoord->x;
	double object_shift_Y = half_frame_Y - screenCoord->y;

	double pixel_scale_X = R_m * tan(half_frame_X_rad) / half_frame_X;
	double pixel_scale_Y = R_m * tan(half_frame_Y_rad) / half_frame_Y;

	double x_n = (object_shift_X - deviation_X) / (half_frame_X - deviation_X) * (R_m * tan(half_frame_X_rad) - deviation_X * pixel_scale_X);
	double y_n = (object_shift_Y - deviation_Y) / (half_frame_Y - deviation_Y) * (R_m * tan(half_frame_Y_rad) - deviation_Y * pixel_scale_Y);
```

```
double x_corr = x_n * (1 + k_1 * (pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)) + k_2 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 2) + k_3 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 3));
double y_corr = y_n * (1 + k_1 * (pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)) + k_2 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 2) + k_3 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 3));

double sin_camera_pitch = sin(camera_pitch);
double cos_camera_pitch = cos(camera_pitch);
double sin_camera_roll = sin(camera_roll);
double cos_camera_roll = cos(camera_roll);
double sign_camera_pitch = sign(camera_pitch);
double cos_UAV_yaw = cos(UAV_yaw);
double sin_UAV_yaw = sin(UAV_yaw);
double cos_UAV_pitch = cos(UAV_pitch);
double sin_UAV_pitch = sin(UAV_pitch);
double sin_UAV_roll = sin(UAV_roll);
double cos_UAV_roll = cos(UAV_roll);


double delta_X = y_corr * sin_camera_pitch;
double delta_Y = cos_camera_roll * x_corr - sin(camera_roll * sign_camera_pitch) * cos_camera_pitch * y_corr;
double delta_Z = -(sin(camera_roll * sign_camera_pitch * x_corr + cos_camera_roll * cos_camera_pitch) * y_corr);

double X_m = R_m * cos_camera_pitch;
double Y_m = (R_m * sin_camera_pitch) * sin_camera_roll;
double Z_m = (R_m * sin_camera_pitch) * cos_camera_roll * sign_camera_pitch;

double X_kn = X_m + delta_X;
double Y_kn = Y_m + delta_Y;
double Z_kn = Z_m + delta_Z;

double X_in = X_kn * cos_UAV_yaw * cos_UAV_pitch + Y_kn * sin_UAV_yaw * cos_UAV_pitch - Z_kn * sin_UAV_pitch;
double Y_in = X_kn * (cos_UAV_yaw * sin_UAV_pitch * sin_UAV_roll - sin_UAV_yaw * cos_UAV_roll) + Y_kn * (sin_UAV_yaw * sin_UAV_pitch * sin_UAV_roll + cos_UAV_yaw * cos_UAV_roll) + Z_kn * cos_UAV_pitch * sin_UAV_roll;
double Z_in = X_kn * (cos_UAV_yaw * sin_UAV_pitch * cos_UAV_roll + sin_UAV_yaw * sin_UAV_roll) + Y_kn * (sin_UAV_yaw * sin_UAV_pitch * cos_UAV_roll - cos_UAV_yaw * sin_UAV_roll) + Z_kn * cos_UAV_pitch * cos_UAV_roll;

double X_io = X_cam * cos_UAV_yaw * cos_UAV_pitch + Y_cam * sin_UAV_yaw * cos_UAV_pitch - Z_cam * sin_UAV_pitch;
double Y_io = X_cam * (cos_UAV_yaw * sin_UAV_pitch * sin_UAV_roll - sin_UAV_yaw * cos_UAV_roll) + Y_cam * (sin_UAV_yaw * sin_UAV_pitch * sin_UAV_roll + cos_UAV_yaw * cos_UAV_roll) + Z_cam * cos_UAV_pitch * sin_UAV_roll;
```

```
	double Z_io = X_cam * (cos_UAV_yaw * sin_UAV_pitch * cos_UAV_roll + sin_UAV_yaw * sin_UAV_roll) + Y_cam * (sin_UAV_yaw * sin_UAV_pitch * cos_UAV_roll - cos_UAV_yaw * sin_UAV_roll) + Z_cam * cos_UAV_pitch * cos_UAV_roll;

	double t_m = (UAV_altitude - Z_io) / (Z_in - Z_io);

	double X_norm = (X_in - X_io) * t_m + X_io;
	double Y_norm = (Y_in - Y_io) * t_m + Y_io;
	double Z_norm = (Z_in - Z_io) * t_m + Z_io;

	double delta_longitude = Y_norm / (EARTH_RADIUS_M * cos_UAV_roll);
	double delta_latitude = X_norm / EARTH_RADIUS_M;

	worldGPSCoord->Init(UAV_latitude + delta_latitude, UAV_longitude + delta_longitude, 0); //??? hmsl is incorrect
}
```

Файл исходного кода TestTrackingMain.cpp:

```
#include "stdafx.h"

#include "ImageStabilizer.h"
#include "ObjectTracker.h"
#include "UAVImageProcessor.h"
#include "MapTileContainer.h"
#include "PerformanceAnalyzer.h"
#include "CoordConverter.h"

UAVImageProcessor* imageProcessor;

void onMouseEvent(int event, int x, int y, int flags, void* param)
{
	switch (event)
	{
	case CV_EVENT_LBUTTONDOWN:
		imageProcessor->TargetLockOn(x, y);
		break;
	case CV_EVENT_MOUSEMOVE:
		bool useMagnifier = (flags & CV_EVENT_FLAG_SHIFTKEY) == CV_EVENT_FLAG_SHIFTKEY;

		imageProcessor->ShowSight(x, y);
		break;
	}
}

int main(int argc, char *argv[])
{
	omp_set_num_threads(2);
	//int thread_count = omp_get_max_threads();
```

```
char* fileName;
if (argc > 1)
	fileName = argv[1];
else
{
	//fileName = "d:\\Tracking\\C++\\Test videos\\1_403-553_zoom.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\2_994-1044_opposite.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\3_710-860.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\4_1184-1234_shift.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\5_1336-1486.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\6_3820-3970.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\7_2322-2472.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\8_15142-15292.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\9_16200-16350.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\10_16435-16585.avi";
	//fileName = "d:\\Tracking\\C++\\Test videos\\11_tracking_950_1150.avi";
	//fileName = "d:\\video\\сопровождение_300_1000.avi";
	//fileName = "d:\\Video\\02.10.2014\\2014-10-01  18-27.avi";
	//fileName = "d:\\Video\\2014-09-30  12-14.avi";
	//fileName = "d:\\Projects\\Video\\1\\2014-10-06  15-11 10000-12000.avi";
	//fileName = "d:\\Projects\\ObjectTracking\\Sopr3.avi ";

	fileName = "d:\\Video\\сопровождение_1106_1646.avi";
	fileName = "d:\\Video\\2014-09-30  12-14.avi";
	fileName = "d:\\Video\\Видео с Борта\\session11.ts";
	fileName = "d:\\Video\\Видео с Борта\\session11_52400_52800.avi";
	//fileName = "d:\\video\\сопровождение_241_641.avi";
	//fileName = "d:\\Video\\02.10.2014\\2014-10-01  18-27.avi";
}

CvCapture* capture = 0;
capture = cvCaptureFromAVI(fileName);
if (!capture)
{
	fprintf(stderr, "Could not initialize capturing…\n");
	return -1;
}

cvNamedWindow("Transformed", CV_WINDOW_AUTOSIZE);
// mouse ivent for object selecting
cvSetMouseCallback("Transformed", onMouseEvent, 0);

CvFont font;
cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX_SMALL, 1.0, 1.0);

int frameWidth = (int)cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH);
int frameHeight = (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT);
int frameChannel = 3;
imageProcessor = new UAVImageProcessor(frameWidth, frameHeight, frameChannel);
```

```
MapTileContainer* mapTileContainer = new MapTileContainer("D:\\Tracking\\C++\\OpenCVTest1_20141030_1010\\MapTiles\\GoogleTiles.db");

IplImage* mapImage = cvCreateImage(cvSize(1200, 768), IPL_DEPTH_8U, frameChannel);
int mapScale = 17;

IplImage* transformedFrame = cvCreateImageHeader(cvSize(frameWidth, frameHeight), IPL_DEPTH_8U, frameChannel);
IplImage* currFrame = 0;

TelemetryData telemetryData;
telemetryData.gps_lon = 28.4;
telemetryData.gps_lat = 54.2;
telemetryData.gps_hmsl = 200;
telemetryData.pitch = 90;
telemetryData.yaw = 0;
telemetryData.yaw = 0;

WorldGPSCoord uavCoord, targetCoord;

int first_frame = 1;

//memset(&telemetryData, 0, sizeof(telemetryData));
imageProcessor->SetUseSightMagnifier(!imageProcessor->GetUseSightMagnifier());
for (;;)
{

	if (first_frame == 1)
	{
		currFrame = cvQueryFrame(capture);
		telemetryData.gps_lat += EncodeCoord(0, 0, 0.5);
		//telemetryData.gps_lat -= EncodeCoord(0, 0, 0.05);
		//telemetryData.gps_lon += EncodeCoord(0, 0, 0.04);
		uavCoord.Init(telemetryData.gps_lat, telemetryData.gps_lon, telemetryData.gps_hmsl);

		mapTileContainer->SetImageCenter(telemetryData.gps_lat, telemetryData.gps_lon);
		mapTileContainer->SetScale(mapScale);
		mapTileContainer->GetMapImage(mapImage);
		if (mapImage != 0)
			cvShowImage("MAP", mapImage);
		imageProcessor->ProcessFrame(currFrame->imageData, &telemetryData);
		transformedFrame->imageData = imageProcessor->GetProcessedFrame(true);

		if (transformedFrame->imageData != 0)
		{
			cvShowImage("Transformed", transformedFrame);
		}
```

```
                first_frame = 0;
                int key = cvWaitKey(0);
        }
        currFrame = cvQueryFrame(capture);
        if (!currFrame)
                break;
        telemetryData.gps_lat += EncodeCoord(0, 0, 0.5);
        //telemetryData.gps_lat -= EncodeCoord(0, 0, 0.05);
        //telemetryData.gps_lon += EncodeCoord(0, 0, 0.04);
        uavCoord.Init(telemetryData.gps_lat, telemetryData.gps_lon,
telemetryData.gps_hmsl);

        mapTileContainer->SetImageCenter(telemetryData.gps_lat, telemetryData.gps_lon);
        mapTileContainer->SetScale(mapScale);
        mapTileContainer->GetMapImage(mapImage);
        if (mapImage != 0)
                cvShowImage("MAP", mapImage);

        int64 tickCount = cvGetTickCount();

        imageProcessor->ProcessFrame(currFrame->imageData, &telemetryData);
        transformedFrame->imageData = imageProcessor->GetProcessedFrame(true);

        tickCount = cvGetTickCount() - tickCount;

        char time_ch[16];
        sprintf(time_ch, "%f", tickCount / (cvGetTickFrequency()*1000.));

        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////// ANGLES CORRECTION ///////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////
        double angle_cor_X, angle_cor_Y;
        imageProcessor->GetCamCorrectionAngles(angle_cor_X, angle_cor_Y);

        // show results on the frame
        char angle_corX_ch[16];
        char angle_corY_ch[16];
        sprintf(angle_corX_ch, "%f", angle_cor_X);
        sprintf(angle_corY_ch, "%f", angle_cor_Y);
        cvPutText(transformedFrame, angle_corX_ch, cvPoint(550, 420), &font,
CV_RGB(0, 255, 255));
        cvPutText(transformedFrame, angle_corY_ch, cvPoint(550, 440), &font,
CV_RGB(0, 255, 255));
        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////////

        //cvShowImage("Source", currFrame);
        if (transformedFrame->imageData != 0)
        {
```

```
                cvPutText(transformedFrame, time_ch, cvPoint(50, 50), &font, CV_RGB(0,
255, 255));
                cvShowImage("Transformed", transformedFrame);
        }

        char c = cvWaitKey(40);

        if (c == 27)
                break;
        switch (c)
        {
        case '1':
                imageProcessor->SetTargetRectSize(10);
                break;
        case '2':
                imageProcessor->SetTargetRectSize(15);
                break;
        case '3':
                imageProcessor->SetTargetRectSize(20);
                break;
        case '4':
                imageProcessor->SetTargetRectSize(30);
                break;
        case '5':
                imageProcessor->SetTargetRectSize(40);
                break;
        case 'r':
        case 'R':
                imageProcessor->DropTarget();
                break;

        case 's':
        case 'S':
                imageProcessor->SetUseSightMagnifier(!imageProcessor-
>GetUseSightMagnifier());
                break;

        case 'x':
        case 'X':
                imageProcessor->GetMaxStabilizationOffset() == 0 ? imageProcessor-
>SetMaxStabilizationOffset(200) : imageProcessor->SetMaxStabilizationOffset(0);
                break;

        case 'q':
        case 'Q':
                mapScale = mapScale > 11 ? mapScale - 1 : 17;
                break;
        case 'w':
        case 'W':
                mapScale = mapScale < 17 ? mapScale + 1 : 11;
                break;
        default:
```

```
                    break;
                }
            }

            cvReleaseImageHeader(&transformedFrame);
            cvReleaseCapture(&capture);
            cvDestroyAllWindows();

            GlobalPerformanceAnalyzer()->PrintStatistics();
            cvWaitKey(-10);


            delete imageProcessor;
            return 0;
}
```

ПРИЛОЖЕНИЕ В

(обязательное)

Описание программных средств улучшения качества и нормализации кадров видеопотока с борта БЛА

Для улучшения качества и нормализации кадров видеопотока с борта БЛА использована следующая функция:
- void ProcessFrame(IplImage* sourceFrame) – для определения величин пиксельной коррекции кадров видеопоследовательности. Входной параметр: текущий кадр видеопоследовательности.

Листинг файлов для поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА:

Заголовочный файл ImageStabilizer.h:

```
#pragma once

#define CV_NO_BACKWARD_COMPATIBILITY

#include "stdafx.h"
#include "CommonData.h"

const double STABILIZATION_CORRECTION_SPEED = 0.95;
const int STABILIZATION_CORRECTION_MAX_OFFSET = 200;
const unsigned int MAX_FRAME_OFFSET_COUNT = 50;

class ImageStabilizer
{
private:
	cv::Ptr<CvMat> _ERT_sA, _ERT_sB;

	IplImage* _currFrameGS;
	IplImage* _currFrameRGB;
	IplImage* _prevFrameGS;
	IplImage* _currFramePyramid;
	IplImage* _prevFramePyramid;
	IplImage* _transformedFrameRGB;
	CvSize _frameSize;

	unsigned __int64 _frameNumber;

	double _hMatrixStub[6];
	CvMat _hMatrix;

	double _stabilizationSpeed;
	int _maxStabilizationOffset;

	TrajectoryRingBuffer *_frameOffsetsBuffer;
```

```
        TrajectoryOffsetVector _lastFrameCorrectionShift;

        int EstimateRigidTransform();
public:
        double GetStabilizationSpeed();
        void SetStabilizationSpeed(double value);
        int GetMaxStabilizationOffset();
        void SetMaxStabilizationOffset(int value);

        ImageStabilizer(CvSize frameSize, int channels);
        ~ImageStabilizer(void);
        void ProcessFrame(IplImage* sourceFrame);
        IplImage* GetTransformedFrameRGB();
        IplImage* GetCurrentFrameGS();
        TrajectoryOffsetVector GetLastInterframeOffset();
        TrajectoryOffsetVector GetLastFrameCorrectionShift();
};
```

Файл исходного кода ObjectTracker.cpp:

```
#include "stdafx.h"
#include "ImageStabilizer.h"
#include "PerformanceAnalyzer.h"

void ShiftImage(IplImage* srcImage, IplImage* dstImage, int dx, int dy)
{
        memset(dstImage->imageData, 0, dstImage->imageSize);

        char * srcData = srcImage->imageData;
        char * dstData = dstImage->imageData;
        int lineWidth = srcImage->widthStep;
        int absDxPix = abs(dx) * srcImage->nChannels;
        int absDy = abs(dy);

        //Copy Part of Source Image
        int copyLength = lineWidth - absDxPix;
        int count = srcImage->height - absDy;
        int offsetSrc, offsetDst, baseSrc, baseDst;

        if (dx < 0)
        {
                offsetSrc = absDxPix;
                offsetDst = 0;
        }
        else
        {
                offsetSrc = 0;
                offsetDst = absDxPix;
        }

        if (dy < 0)
        {
```

```
        baseSrc = lineWidth * absDy + offsetSrc;
        baseDst = 0 + offsetDst;
}
else
{
        baseSrc = 0 + offsetSrc;
        baseDst = lineWidth * absDy + offsetDst;
}

srcData = srcData + baseSrc;
dstData = dstData + baseDst;
// #pragma omp parallel for
for (int i = 0; i < count; i++)
{
        memcpy(dstData, srcData, copyLength);
        srcData += lineWidth;
        dstData += lineWidth;
}

/*
//Clear borders
//top-bottom
if (dy != 0)
{
int fillOffset, fillCount;
if (dy < 0)
fillOffset = lineWidth * (dstImage->height - absDy);
else
fillOffset = 0;
fillCount = lineWidth * absDy;
memset(dstImage->imageData + fillOffset, 0, fillCount);
}

//left-right
if (dx != 0)
{
int fillOffset;
if (dx < 0)
fillOffset = lineWidth - absDxPix;
else
fillOffset = 0;

if (dy < 0)
fillOffset = 0 + fillOffset;
else
fillOffset = lineWidth * absDy + offsetDst;
char * fillData = dstImage->imageData + fillOffset;
for (int i = 0; i < count; i++)
{
memset(fillData, 0, absDxPix);
fillData  = fillData + lineWidth;
}
```

```
        }
        */
}

void mcvCopyImage(IplImage* srcImage, IplImage* dstImage)
{
        ShiftImage(srcImage, dstImage, 0, 0);
}

void GetRTMatrix(const CvPoint2D32f* a, const CvPoint2D32f* b, int count, CvMat* M)
{
        double sa[16], sb[4], m[4], *om = M->data.db;
        CvMat A = cvMat(4, 4, CV_64F, sa);
        CvMat B = cvMat(4, 1, CV_64F, sb);
        CvMat MM = cvMat(4, 1, CV_64F, m);

        int i;

        memset(sa, 0, sizeof(sa));
        memset(sb, 0, sizeof(sb));

        for (i = 0; i < count; i++)
        {
                sa[0] += a[i].x * a[i].x + a[i].y * a[i].y;
                sa[1] += 0;
                sa[2] += a[i].x;
                sa[3] += a[i].y;

                sa[4] += 0;
                sa[5] += a[i].x * a[i].x + a[i].y * a[i].y;
                sa[6] += -a[i].y;
                sa[7] += a[i].x;

                sa[8] += a[i].x;
                sa[9] += -a[i].y;
                sa[10] += 1;
                sa[11] += 0;

                sa[12] += a[i].y;
                sa[13] += a[i].x;
                sa[14] += 0;
                sa[15] += 1;

                sb[0] += a[i].x * b[i].x + a[i].y * b[i].y;
                sb[1] += a[i].x * b[i].y - a[i].y * b[i].x;
                sb[2] += b[i].x;
                sb[3] += b[i].y;
        }

        cvSolve(&A, &B, &MM, CV_SVD);

        om[0] = om[4] = m[0];
```

```
        om[1] = -m[1];
        om[3] = m[1];
        om[2] = m[2];
        om[5] = m[3];
}

//cvEstimateRigidTransform2(_currFrameGS, _prevFrameGS, &_hMatrix);
int ImageStabilizer::EstimateRigidTransform()
{
        GlobalPerformanceAnalyzer()->StartMeasure(Test1);

        const int TEST_FEATURE_POINT_COUNT_X = 5;
        const int TEST_FEATURE_POINT_COUNT_Y = 5;

        const int WIDTH = 160;
        const int HEIGHT = 120;

        const int RANSAC_MAX_ITERS = 500;
        const int RANSAC_SIZE0 = 3;
        const double RANSAC_GOOD_RATIO = 0.5;

        cv::Ptr<CvMat> sA, sB;
        cv::AutoBuffer<CvPoint2D32f> pA, pB;
        cv::AutoBuffer<int> good_idx;
        cv::AutoBuffer<char> status;
        //cv::Ptr<CvMat> gray;
        int featurePointCount = TEST_FEATURE_POINT_COUNT_X *
TEST_FEATURE_POINT_COUNT_Y;

        CvMat stubA, *A = cvGetMat(_currFrameGS, &stubA);
        CvMat stubB, *B = cvGetMat(_prevFrameGS, &stubB);
        CvSize sz0, sz1;
        int cn, equal_sizes;
        int i, j, k, k1;
        double scale = 1;
        CvRNG rng = cvRNG(-1);
        double m[6] = { 0 };
        CvMat M = cvMat(2, 3, CV_64F, m);
        int good_count = 0;
        CvRect brect;

        cn = CV_MAT_CN(A->type);
        sz0 = cvGetSize(A);
        sz1 = cvSize(WIDTH, HEIGHT);

        scale = MAX((double)sz1.width / sz0.width, (double)sz1.height / sz0.height);
        scale = MIN(scale, 1.);
        sz1.width = cvRound(sz0.width * scale);
        sz1.height = cvRound(sz0.height * scale);

        equal_sizes = sz1.width == sz0.width && sz1.height == sz0.height;
```

```
	sA = cvCreateMat(sz1.height, sz1.width, CV_8UC1);
	sB = cvCreateMat(sz1.height, sz1.width, CV_8UC1);
#pragma omp parallel sections
	{
#pragma omp section
		{
			cvResize(A, sA, CV_INTER_AREA);
		}
#pragma omp section
		{
		cvResize(B, sB, CV_INTER_AREA);
	}
	}
	A = sA;
	B = sB;

	pA.allocate(featurePointCount);
	pB.allocate(featurePointCount);
	status.allocate(featurePointCount);

	for (i = 0, k = 0; i < TEST_FEATURE_POINT_COUNT_Y; i++)
	for (j = 0; j < TEST_FEATURE_POINT_COUNT_X; j++, k++)
	{
		pA[k].x = (j + 0.5f) * sz1.width / TEST_FEATURE_POINT_COUNT_X;
		pA[k].y = (i + 0.5f) * sz1.height / TEST_FEATURE_POINT_COUNT_Y;
	}

	GlobalPerformanceAnalyzer()->StopMeasure(Test1);

	GlobalPerformanceAnalyzer()->StartMeasure(Test2);
	// find the corresponding points in B
	int opticalFlowFlag = 0;
	if (_currFramePyramid == 0)
	{
		_currFramePyramid = cvCreateImage(sz1, 8, 1);
		_prevFramePyramid = cvCreateImage(sz1, 8, 1);
		opticalFlowFlag |= CV_LKFLOW_PYR_A_READY;
	}

	cvCalcOpticalFlowPyrLK(A, B, _currFramePyramid, _prevFramePyramid, pA, pB,
featurePointCount, cvSize(10, 10), 3,
		status, 0, cvTermCriteria(CV_TERMCRIT_ITER, 40, 0.1), opticalFlowFlag);
	IplImage * swap_temp;
	CV_SWAP(_currFramePyramid, _prevFramePyramid, swap_temp);
	// repack the remained points
	for (i = 0, k = 0; i < featurePointCount; i++)
	if (status[i])
	{
		if (i > k)
		{
			pA[k] = pA[i];
			pB[k] = pB[i];
```

```
        }
        k++;
}

featurePointCount = k;

GlobalPerformanceAnalyzer()->StopMeasure(Test2);

//-------------------------------------------
GlobalPerformanceAnalyzer()->StartMeasure(Test3);

if (featurePointCount < RANSAC_SIZE0)
        return 0;

CvMat _pB = cvMat(1, featurePointCount, CV_32FC2, pB);
brect = cvBoundingRect(&_pB, 1);

// RANSAC stuff:
for (k = 0; k < RANSAC_MAX_ITERS; k++)
{
        int idx[RANSAC_SIZE0];
        CvPoint2D32f a[3];
        CvPoint2D32f b[3];

        memset(a, 0, sizeof(a));
        memset(b, 0, sizeof(b));

        // choose random 3 non-complanar points from A & B
        for (i = 0; i < RANSAC_SIZE0; i++)
        {
                for (k1 = 0; k1 < RANSAC_MAX_ITERS; k1++)
                {
                        idx[i] = cvRandInt(&rng) % featurePointCount;

                        for (j = 0; j < i; j++)
                        {
                                if (idx[j] == idx[i])
                                        break;
                                // check that the points are not very close one each other
                                if (fabs(pA[idx[i]].x - pA[idx[j]].x) +
                                        fabs(pA[idx[i]].y - pA[idx[j]].y) < FLT_EPSILON)
                                        break;
                                if (fabs(pB[idx[i]].x - pB[idx[j]].x) +
                                        fabs(pB[idx[i]].y - pB[idx[j]].y) < FLT_EPSILON)
                                        break;
                        }

                        if (j < i)
                                continue;

                        if (i + 1 == RANSAC_SIZE0)
                        {
```

```
                                        // additional check for non-complanar vectors
                                        a[0] = pA[idx[0]];
                                        a[1] = pA[idx[1]];
                                        a[2] = pA[idx[2]];

                                        b[0] = pB[idx[0]];
                                        b[1] = pB[idx[1]];
                                        b[2] = pB[idx[2]];

                                        double dax1 = a[1].x - a[0].x, day1 = a[1].y - a[0].y;
                                        double dax2 = a[2].x - a[0].x, day2 = a[2].y - a[0].y;
                                        double dbx1 = b[1].x - b[0].y, dby1 = b[1].y - b[0].y;
                                        double dbx2 = b[2].x - b[0].x, dby2 = b[2].y - b[0].y;
                                        const double eps = 0.01;

                                        if (fabs(dax1 * day2 - day1 * dax2) < eps * sqrt(dax1 * dax1 +
day1 * day1) * sqrt(dax2 * dax2 + day2 * day2) ||
                                                fabs(dbx1 * dby2 - dby1 * dbx2) < eps * sqrt(dbx1 *
dbx1 + dby1 * dby1) * sqrt(dbx2 * dbx2 + dby2 * dby2))
                                                continue;
                                        }
                                        break;
                                }

                                if (k1 >= RANSAC_MAX_ITERS)
                                        break;
                        }

                        if (i < RANSAC_SIZE0)
                                continue;

                        // estimate the transformation using 3 points
                        GetRTMatrix(a, b, 3, &M);

                        for (i = 0, good_count = 0; i < featurePointCount; i++)
                        {
                                if (fabs(m[0] * pA[i].x + m[1] * pA[i].y + m[2] - pB[i].x) +
                                        fabs(m[3] * pA[i].x + m[4] * pA[i].y + m[5] - pB[i].y) <
MAX(brect.width, brect.height) * 0.05)
                                        good_idx[good_count++] = i;
                        }

                        if (good_count >= featurePointCount * RANSAC_GOOD_RATIO)
                                break;
                }

                if (k >= RANSAC_MAX_ITERS)
                        return 0;

                if (good_count < featurePointCount)
                {
                        for (i = 0; i < good_count; i++)
```

```
                {
                        j = good_idx[i];
                        pA[i] = pA[j];
                        pB[i] = pB[j];
                }
        }

        GetRTMatrix(pA, pB, good_count, &M);
        m[2] /= scale;
        m[5] /= scale;
        cvConvert(&M, &_hMatrix);
        GlobalPerformanceAnalyzer()->StopMeasure(Test3);

        return 1;
}


TrajectoryOffsetVector MakeTrajectoryOffsetVector(double x, double y)
{
        TrajectoryOffsetVector to;
        to.x = x;
        to.y = y;
        return to;
}

ImageStabilizer::ImageStabilizer(CvSize frameSize, int channels)
{
        _frameSize = frameSize;
        _frameNumber = 0;

        _frameOffsetsBuffer = new TrajectoryRingBuffer(MAX_FRAME_OFFSET_COUNT);

        _currFrameRGB = 0;
        _transformedFrameRGB = cvCreateImage(_frameSize, 8, channels);
        _prevFrameGS = cvCreateImage(_frameSize, 8, 1);
        _currFrameGS = cvCreateImage(_frameSize, 8, 1);
        _currFramePyramid = 0;
        _prevFramePyramid = 0;

        _hMatrix = cvMat(2, 3, CV_64F, _hMatrixStub);
        _lastFrameCorrectionShift = MakeTrajectoryOffsetVector(0, 0);

        _stabilizationSpeed = STABILIZATION_CORRECTION_SPEED;
        _maxStabilizationOffset = STABILIZATION_CORRECTION_MAX_OFFSET;
}

ImageStabilizer::~ImageStabilizer(void)
{
        delete _frameOffsetsBuffer;
}

double ImageStabilizer::GetStabilizationSpeed()
```

```
{
	return _stabilizationSpeed;
}

void ImageStabilizer::SetStabilizationSpeed(double value)
{
	_stabilizationSpeed = value;
}

int ImageStabilizer::GetMaxStabilizationOffset()
{
	return _maxStabilizationOffset;
}

void ImageStabilizer::SetMaxStabilizationOffset(int value)
{
	_maxStabilizationOffset = value;
}

IplImage* ImageStabilizer::GetTransformedFrameRGB()
{
	return _transformedFrameRGB;
}

IplImage* ImageStabilizer::GetCurrentFrameGS()
{
	return _currFrameGS;
}

TrajectoryOffsetVector ImageStabilizer::GetLastInterframeOffset()
{
	return _frameOffsetsBuffer->GetLastOffsetVector();
}

TrajectoryOffsetVector ImageStabilizer::GetLastFrameCorrectionShift()
{
	return _lastFrameCorrectionShift;
}

void TestSaveFrame(int frameNumber, IplImage* frame)
{
	char fileName[1024];
	sprintf(fileName, "d:\\tmp\\2\\%d.jpg", frameNumber);
	cvSaveImage(fileName, frame);
}

void ImageStabilizer::ProcessFrame(IplImage* sourceFrame)
{
	GlobalPerformanceAnalyzer()->StartMeasure(ImageStabilizer_ProcessFrame);
	_currFrameRGB = sourceFrame;
	cvCvtColor(_currFrameRGB, _currFrameGS, CV_RGB2GRAY);
```

```
	if (_frameNumber > 0)
	{
		GlobalPerformanceAnalyzer()->StartMeasure(ImageStabilizer_EstimateRigidTransform);
		int resultFindHomography = EstimateRigidTransform();
		GlobalPerformanceAnalyzer()->StopMeasure(ImageStabilizer_EstimateRigidTransform);

		double globalX = _hMatrixStub[2];
		double globalY = _hMatrixStub[5];
		if (resultFindHomography == 0)
		{
			globalX = _lastFrameCorrectionShift.x;
			globalY = _lastFrameCorrectionShift.y;
		}

		//???
		/*
		if ((globalX < 5) && (globalX > -5))
		globalX = 0;
		if ((globalY < 5) && (globalY > -5))
		globalY = 0;
		*/
		TrajectoryOffsetVector avgTrajectoryOffsetVector = _frameOffsetsBuffer->GetAvgTrajectoryOffsetVector();

		if (resultFindHomography != 0)
			_frameOffsetsBuffer->PushTrajectoryOffsetVector(globalX, globalY);

		double dx = globalX - avgTrajectoryOffsetVector.x + (_lastFrameCorrectionShift.x * _stabilizationSpeed);
		double dy = globalY - avgTrajectoryOffsetVector.y + (_lastFrameCorrectionShift.y * _stabilizationSpeed);

		if (dx > _maxStabilizationOffset)
			dx = _maxStabilizationOffset;
		else if ((dx < -_maxStabilizationOffset))
			dx = -_maxStabilizationOffset;
		if (dy > _maxStabilizationOffset)
			dy = _maxStabilizationOffset;
		else if ((dy < -_maxStabilizationOffset))
			dy = -_maxStabilizationOffset;

		/*
		if (dx > 2 || dy > 2)
		{
			TestSaveFrame(_frameNumber - 1, _prevFrameGS);
			TestSaveFrame(_frameNumber, _currFrameGS);
		}
		*/

		GlobalPerformanceAnalyzer()->StartMeasure(ImageStabilizer_WarpAffine);
```

```
            ShiftImage(_currFrameRGB, _transformedFrameRGB, (int)dx, (int)dy);

            GlobalPerformanceAnalyzer()->StopMeasure(ImageStabilizer_WarpAffine);

            _lastFrameCorrectionShift.x = dx;
            _lastFrameCorrectionShift.y = dy;
        }
        else //first frame in sequence
        {
            mcvCopyImage(_currFrameRGB, _transformedFrameRGB);
        }
        mcvCopyImage(_currFrameGS, _prevFrameGS);
        _frameNumber++;
        GlobalPerformanceAnalyzer()->StopMeasure(ImageStabilizer_ProcessFrame);
}
```

ПРИЛОЖЕНИЕ Г

(обязательное)

Описание программных средств выделения объекта наблюдения, отмеченного оператором, на кадре видеопотока

Для выделения объекта наблюдения, отмеченного оператором, на кадре видеопотока использованы следующие функции:

- void SetTargetRectSize(int size) – для задания размера рамки захвата объекта наблюдения. Входной параметр: размер рамки.
- bool TargetLockOn(int x, int y) – для выделения эталонного изображения объекта наблюдения, его размеров и координат. Входные параметры: пиксельные координаты, отмеченные оператором на видеокадре

Листинг файлов для поиска и сопровождения объекта наблюдения по кадрам видеопотока с борта БЛА:

Заголовочный файл ObjectTracker.h:

```
#pragma once
#include "StdAfx.h"
#include "CommonData.h"

class ObjectTracker
{
private:
	IplImage* frame_GS;

	// object position & size
	int _defaultTargetLockOnFrameSize, _targetLockOnFrameSize;
	ScreenCoord _object_position, _object_position_previous /* ???? */,
_object_position_predicted;
	int _object_size_X, _object_size_Y;

	// features for template & candidates search area
	IplImage *template_32F, *candidate_32F; // 32-bit float images
	IplImage *template_dx, *template_dy, *template_dxx, *template_dyy, *template_value;
	IplImage *candidate_dx, *candidate_dy, *candidate_dxx, *candidate_dyy,
*candidate_value;
	IplImage *template_vector_value, *template_vector_dx, *template_vector_dy,
*template_vector_dxx, *template_vector_dyy;

	IplImage * _segmentation_image;
	CvMemStorage * _segmentation_memory;

	// pointers for reshaping into input for cvCalcCovarMatrix
	IplImage **template_input, **candidate_input;

	// covariance matrices & vectors of feature means for template & candidates
	CvMat *template_covariance_matrix;
	CvMat *candidate_covariance_matrix;
```

```
        CvMat *inv_candidate_covariance_matrix;
        CvMat *matrix_multiplication;
        CvMat *eigenvector;
        CvMat *eigenvalues;
        CvMat *eigenvalues_log;
        CvMat *eigenvalues_pow;

        TrajectoryRingBuffer * _objectFrameShiftBuffer;

        bool _targetLocked;

        void CalculateCovariance(int position_x, int position_y, int size_x, int size_y, int down_x,
int down_y);
        void FindContourCoord(int window_height, int window_width, bool useTheshold, int
&left, int &top, int &right, int &bottom);
public:
        ObjectTracker(void);
        ~ObjectTracker(void);
        void ProcessFrame(IplImage* sourceFrame, TrajectoryOffsetVector lastInterframeOffset);
        bool TargetLockOn(int x, int y);
        void DropTarget();
        void SetTargetRectSize(int size);
        int GetTargetRectSize();
        ScreenCoord GetObjectPosition();
};
```

Файл исходного кода ObjectTracker.cpp:

```
#include "StdAfx.h"
#include "CommonFunctions.h"
#include "ObjectTracker.h"
#include "PerformanceAnalyzer.h"

double const INFINITY2 = 100000;
int const MAX_FRAME_OBJECT_SHIFT_COUNT = 50;

// tracker settings
const double overlap_factor = 0.4; // set the part of non-overlapping area of candidate window
const int window_factor = 2; // set the number of candidate windows: 1 is equal to 9 candidate
windows, 2 to 25, 3 to 49 etc.
const double position_fixation_factor = 1.5; // range is 0-1: set the fixation weight to predicted
position
const double prediction_shift_factor = 0.5; // range is 0-1: set the maximum shift of predicted
position

const int NUMBER_OF_OBECT_FEATURES = 5; // number of object features (also check class
for covarince calculation)

uchar* mcvPtr2D(const IplImage* img, int y, int x)
{
```

```
	uchar* ptr = 0;

	int pix_size = (img->depth & 255) >> 3;
	if (img->dataOrder == 0)
		pix_size *= img->nChannels;
	int width, height;
	ptr = (uchar*)img->imageData;

	if (img->roi)
	{
		width = img->roi->width;
		height = img->roi->height;
		ptr += img->roi->yOffset * img->widthStep + img->roi->xOffset * pix_size;
	}
	else
	{
		width = img->width;
		height = img->height;
	}

	ptr += y * img->widthStep + x * pix_size;

	return ptr;
}

void CopyElementValue(const IplImage* fromImg, int fromY, int fromX, const IplImage* toImg,
int toY, int toX)
{
	uchar * ptr_element_value = mcvPtr2D(toImg, toY, toX); // destination data pointer
	uchar * ptr_value = mcvPtr2D(fromImg, fromY, fromX); // source data pointer
	((float*)ptr_element_value)[0] = ((float*)ptr_value)[0];
}

ObjectTracker::ObjectTracker(void)
{
	_targetLocked = false;
	frame_GS = 0;
	_objectFrameShiftBuffer = new
TrajectoryRingBuffer(MAX_FRAME_OBJECT_SHIFT_COUNT);;
	_defaultTargetLockOnFrameSize = 30;
	_targetLockOnFrameSize = _defaultTargetLockOnFrameSize;
	template_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	candidate_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);

	inv_candidate_covariance_matrix = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	matrix_multiplication = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
	eigenvector = cvCreateMat(NUMBER_OF_OBECT_FEATURES,
NUMBER_OF_OBECT_FEATURES, CV_32FC1);
```

```
	eigenvalues = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);
	eigenvalues_log = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);
	eigenvalues_pow = cvCreateMat(NUMBER_OF_OBECT_FEATURES, 1, CV_32FC1);

	template_dx = 0;
	template_dy = 0;
	template_dxx = 0;
	template_dyy = 0;
	template_value = 0;
	template_32F = 0;

	candidate_dx = 0;
	candidate_dy = 0;
	candidate_dxx = 0;
	candidate_dyy = 0;
	candidate_value = 0;
	candidate_32F = 0;

	_segmentation_memory = cvCreateMemStorage(0);
	_segmentation_image = 0;
}

ObjectTracker::~ObjectTracker(void)
{
	delete _objectFrameShiftBuffer;
}

inline void InitOrCreateImage(IplImage ** image_ref, CvSize area_ROI_size, int depth, int
channels)
{
	if (image_ref == 0)
		CV_Error(CV_StsNullPtr, "");

	if (*image_ref == 0)
	{
		*image_ref = cvCreateImage(area_ROI_size, depth, channels);
	}
	else if ((area_ROI_size.width != (*image_ref)->width) || (area_ROI_size.height !=
(*image_ref)->height))
	{
		cvReleaseImage(image_ref);
		*image_ref = cvCreateImage(area_ROI_size, depth, channels);
	}
}

void InitMatrixes(IplImage * frameGS, int x, int y, int width, int height,
	IplImage ** matrix_32F, IplImage ** matrix_value, IplImage ** matrix_dx, IplImage **
matrix_dy, IplImage ** matrix_dxx, IplImage ** matrix_dyy)
{
	CvSize roiSize = cvSize(width, height);
	InitOrCreateImage(matrix_32F, roiSize, IPL_DEPTH_32F, 1);
	InitOrCreateImage(matrix_dx, roiSize, IPL_DEPTH_32F, 1);
```

```
        InitOrCreateImage(matrix_dy, roiSize, IPL_DEPTH_32F, 1);
        InitOrCreateImage(matrix_dxx, roiSize, IPL_DEPTH_32F, 1);
        InitOrCreateImage(matrix_dyy, roiSize, IPL_DEPTH_32F, 1);
        InitOrCreateImage(matrix_value, roiSize, IPL_DEPTH_32F, 1);

        cvSetImageROI(frameGS, cvRect(x, y, width, height));
        cvConvertScale(frameGS, *matrix_32F);
        cvResetImageROI(frameGS);

        cvSobel(*matrix_32F, *matrix_dx, 1, 0, 3);
        cvSobel(*matrix_32F, *matrix_dy, 0, 1, 3);
        cvSobel(*matrix_32F, *matrix_dxx, 2, 0, 3);
        cvSobel(*matrix_32F, *matrix_dyy, 0, 2, 3);
        cvCopy(*matrix_32F, *matrix_value);
}

void CopyMatrixesElements(IplImage ** input_array, int size_x, int size_y,
        IplImage * matrix_value, IplImage * matrix_dx, IplImage * matrix_dy, IplImage *
matrix_dxx, IplImage * matrix_dyy)
{
#pragma omp parallel for num_threads(2)
        for (int y = 0; y < size_y; y++)
        {
                for (int x = 0; x < size_x; x++)
                {
                        int i = x + y * size_x;
                        CopyElementValue(matrix_value, y, x, input_array[i], 0, 0);
                        CopyElementValue(matrix_dx, y, x, input_array[i], 0, 1);
                        CopyElementValue(matrix_dy, y, x, input_array[i], 0, 2);
                        CopyElementValue(matrix_dxx, y, x, input_array[i], 0, 3);
                        CopyElementValue(matrix_dyy, y, x, input_array[i], 0, 4);
                } // for x
        } // for y
}

// calculate covariance matrix of candidates
void ObjectTracker::CalculateCovariance(int position_x, int position_y, int size_x, int size_y, int
down_x, int down_y)
{
        GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_CalculateCovariance);

        // translate frame coordinates to search area system
        int pos_x = position_x - down_x;
        int pos_y = position_y - down_y;

        CvRect roiRect = cvRect(pos_x, pos_y, size_x, size_y);

        cvSetImageROI(candidate_value, roiRect);
        cvSetImageROI(candidate_dx, roiRect);
        cvSetImageROI(candidate_dy, roiRect);
        cvSetImageROI(candidate_dxx, roiRect);
        cvSetImageROI(candidate_dyy, roiRect);
```

```
	// vectors of features for candidates
	// reshape feature images into vectors
	int input_height = size_x * size_y;

	CopyMatrixesElements(candidate_input, size_x, size_y, candidate_value, candidate_dx,
candidate_dy, candidate_dxx, candidate_dyy);

	// calculate covariance matrix
	cvCalcCovarMatrix((const void **)candidate_input, input_height,
candidate_covariance_matrix, 0, CV_COVAR_NORMAL | CV_COVAR_SCALE);

	// reset candidate ROI
	cvResetImageROI(candidate_dx);
	cvResetImageROI(candidate_dy);
	cvResetImageROI(candidate_dxx);
	cvResetImageROI(candidate_dyy);
	cvResetImageROI(candidate_value);

	GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_CalculateCovariance);
}

void ObjectTracker::DropTarget()
{
	_targetLocked = false;
}

void ObjectTracker::SetTargetRectSize(int size)
{
	_defaultTargetLockOnFrameSize = size;
}

int ObjectTracker::GetTargetRectSize()
{
	return _targetLockOnFrameSize;
}

void ObjectTracker::FindContourCooord(int window_width, int window_height, bool useTheshold,
int &left, int &top, int &right, int &bottom)
{
	InitOrCreateImage(&_segmentation_image, cvSize(window_width, window_height),
IPL_DEPTH_8U, 1);
	cvCanny(frame_GS, _segmentation_image, 2150, 4350, 5);
	if (useTheshold)
		cvAdaptiveThreshold(frame_GS, _segmentation_image, 255,
CV_ADAPTIVE_THRESH_GAUSSIAN_C, CV_THRESH_BINARY_INV, 21, 50);
	CvSeq *segmentation_contours;
	cvFindContours(_segmentation_image, _segmentation_memory, &segmentation_contours,
sizeof(CvContour), CV_RETR_CCOMP, CV_CHAIN_APPROX_NONE, cvPoint(0, 0));

	CvPoint *contour_point = NULL;
	int min_x = (int)INFINITY2, min_y = (int)INFINITY2, max_x = 0, max_y = 0;
```

```
	for (; segmentation_contours != NULL; segmentation_contours = segmentation_contours->h_next)
	{
		for (int j = 0; j < segmentation_contours->total; j++)
		{
			contour_point = CV_GET_SEQ_ELEM(CvPoint, segmentation_contours, j);
			if (contour_point->x < min_x)
				min_x = contour_point->x;
			if (contour_point->y < min_y)
				min_y = contour_point->y;
			if (contour_point->x > max_x)
				max_x = contour_point->x;
			if (contour_point->y > max_y)
				max_y = contour_point->y;
		}
	}
	if (min_x < INFINITY2 && min_y < INFINITY2)
	{
		left = min_x - 2;
		top = min_y - 2;
		right = max_x + 2;
		bottom = max_y + 2;
	}
	else
	{
		top = _targetLockOnFrameSize / 2;
		left = _targetLockOnFrameSize / 2;
	}
}

bool ObjectTracker::TargetLockOn(int x, int y)
{
	GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_TargtLockOn);

	_targetLockOnFrameSize = _defaultTargetLockOnFrameSize;

	_object_position.x = x - _targetLockOnFrameSize / 2;
	_object_position.y = y - _targetLockOnFrameSize / 2;
	_object_size_X = _targetLockOnFrameSize;
	_object_size_Y = _targetLockOnFrameSize;

	if (_object_position.x < 0 || _object_position.y < 0 ||
		_object_position.x + _object_size_X > frame_GS->width || _object_position.y + _object_size_Y > frame_GS->height)
		return false;

	// Find target contours
	cvSetImageROI(frame_GS, cvRect(_object_position.x, _object_position.y, _object_size_X, _object_size_Y));
	int top_left_x = 0, top_left_y = 0, bottom_right_x = _object_size_X, bottom_right_y = _object_size_Y;
```

```
	FindContourCooord(_object_size_X, _object_size_Y, false, top_left_x, top_left_y,
bottom_right_x, bottom_right_y);
	_object_position.x = _object_position.x + top_left_x;
	_object_position.y = _object_position.y + top_left_y;
	_object_size_X = bottom_right_x - top_left_x;
	_object_size_Y = bottom_right_y - top_left_y;
	_targetLockOnFrameSize = max(_object_size_X, _object_size_Y);

	cvDestroyWindow("LockedTarget");
	cvNamedWindow("LockedTarget", CV_WINDOW_AUTOSIZE);
	cvShowImage("LockedTarget", frame_GS);
	cvResetImageROI(frame_GS);


	InitMatrixes(frame_GS, _object_position.x, _object_position.y, _object_size_X,
_object_size_Y,
		&template_32F, &template_value, &template_dx, &template_dy, &template_dxx,
&template_dyy);

	// calculate the covariance matrix
	// vectors of features for template
	// reshape feature images into vectors
	int input_height = _object_size_X * _object_size_Y;
	CvSize input_width = cvSize(NUMBER_OF_OBECT_FEATURES, 1);

	// memory allocation for cvCalcCovarMatrix
	candidate_input = new IplImage*[input_height];
	template_input = new IplImage*[input_height];
	for (int i = 0; i < input_height; i++)
	{
		candidate_input[i] = cvCreateImage(input_width, IPL_DEPTH_32F, 1);
		template_input[i] = cvCreateImage(input_width, IPL_DEPTH_32F, 1);
	}

	CopyMatrixesElements(template_input, _object_size_X, _object_size_Y, template_value,
template_dx, template_dy, template_dxx, template_dyy);

	// calculate covariance matrix
	cvCalcCovarMatrix((const void **)template_input, input_height,
template_covariance_matrix, 0, CV_COVAR_NORMAL | CV_COVAR_SCALE);

	//??? Release images from template_input and candidate_input

	// set the previous & the predicted object position
	_object_position_previous.x = _object_position.x;
	_object_position_previous.y = _object_position.y;
	_object_position_predicted.x = _object_position.x;
	_object_position_predicted.y = _object_position.y;

	GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_TargtLockOn);
	_targetLocked = true;
	return true;
```

```
}

void ObjectTracker::ProcessFrame(IplImage* sourceFrame, TrajectoryOffsetVector
lastInterframeOffset)
{
	frame_GS = sourceFrame;

	if (!_targetLocked)
		return;

	GlobalPerformanceAnalyzer()->StartMeasure(ObjectTracker_ProcessFrame);

	//!!! New code
	_object_position_predicted.x -= (int)lastInterframeOffset.x;
	_object_position_predicted.y -= (int)lastInterframeOffset.y;

	// determining the upper & the lower limits of search area
	int size_max = _object_size_X > _object_size_Y ? _object_size_X : _object_size_Y;
	int overlap_area = round2(double(size_max) * overlap_factor);
	// bottom X
	int down_limit_X = max(1, _object_position_predicted.x - window_factor * overlap_area);
	// bottom Y
	int down_limit_Y = max(1, _object_position_predicted.y - window_factor * overlap_area);
	// top X
	int up_limit_X = min(_object_position_predicted.x + window_factor * overlap_area,
frame_GS->width - _object_size_X);
	// top Y
	int up_limit_Y = min(_object_position_predicted.y + window_factor * overlap_area,
frame_GS->height - _object_size_Y);

	InitMatrixes(frame_GS, down_limit_X, down_limit_Y, up_limit_X + _object_size_X -
down_limit_X, up_limit_Y + _object_size_Y - down_limit_Y,
		&candidate_32F, &candidate_value, &candidate_dx, &candidate_dy,
&candidate_dxx, &candidate_dyy);

	int position_counter = -round2((1 + 2 * window_factor) ^ 2 / 2);

	double current_metric = INFINITY2; // initial value of minimized distance metric (=infinity)
	// search for all candidates inside the search area
	for (int candidate_position_Y = down_limit_Y; candidate_position_Y < up_limit_Y + 1;
candidate_position_Y += overlap_area)
	{
		for (int candidate_position_X = down_limit_X; candidate_position_X < up_limit_X
+ 1; candidate_position_X += overlap_area)
		{
			// more weight for predicted position
			position_counter = position_counter + 1;
			double position_weight = 1 + position_fixation_factor * position_counter / 10
/ ((1 + 2 * window_factor) ^ 2);

			// calculate the covariance matrix
```

```
                        CalculateCovariance(candidate_position_X, candidate_position_Y,
_object_size_X, _object_size_Y, down_limit_X, down_limit_Y);

                        // distance metric
                        // find the eigenvalues
                        cvInvert(candidate_covariance_matrix, inv_candidate_covariance_matrix);
                        cvMatMul(template_covariance_matrix, inv_candidate_covariance_matrix,
matrix_multiplication);
                        cvEigenVV(matrix_multiplication, eigenvector, eigenvalues);

                        cvLog(eigenvalues, eigenvalues_log);
                        cvPow(eigenvalues_log, eigenvalues_pow, 2);
                        // metric calculation
                        CvScalar sum = cvSum(eigenvalues_pow);
                        double candidate_metric = cvSqrt(sum.val[0]) * position_weight;
                        // decision
                        if (candidate_metric < current_metric)
                        {
                                current_metric = candidate_metric;
                                _object_position.x = candidate_position_X;
                                _object_position.y = candidate_position_Y;
                        }
                } // for candidate_position_X
        } // for candidate_position_Y

        // frame stabilication by segmentation
        int stab_top, stab_left, stab_width, stab_height;

        if (_object_position.y - round2(_object_size_Y / 2) < 0)
                stab_top = 0;
        else
                stab_top = _object_position.y - round2(_object_size_Y / 2);
        if (_object_position.x - round2(_object_size_X / 2) < 0)
                stab_left = 0;
        else
                stab_left = _object_position.x - round2(_object_size_X / 2);
        if (_object_position.y - round2(_object_size_Y / 2) + 2 * _object_size_Y > frame_GS-
>height)
                stab_height = frame_GS->height - _object_position.y + round2(_object_size_Y / 2);
        else
                stab_height = 2 * _object_size_Y;
        if (_object_position.x - round2(_object_size_X / 2) + 2 * _object_size_X > frame_GS-
>width)
                stab_width = frame_GS->width - _object_position.x + round2(_object_size_X / 2);
        else
                stab_width = 2 * _object_size_X;

        if (stab_left > 0 && stab_top > 0 && stab_left + stab_width < frame_GS->width &&
stab_top + stab_height < frame_GS->height)
        {
                cvSetImageROI(frame_GS, cvRect(stab_left, stab_top, stab_width, stab_height));
                int segm_top = 0, segm_left = 0, segm_bottom = 0, segm_right = 0;
```

```
                FindContourCooord(stab_width, stab_height, true, segm_left, segm_top, segm_right,
segm_bottom);

                if ((segm_right - segm_left < _object_size_X + round2(_object_size_X) / 2 + 4) &&
(segm_bottom - segm_top < _object_size_Y + round2(_object_size_Y / 2) + 4))
                {
                        _object_position.x = stab_left + segm_left;
                        _object_position.y = stab_top + segm_top;
                }
        }
        //      cvShowImage("TEST", frame_GS);
        cvResetImageROI(frame_GS);

        // predict the object position on the next frame
        if ((abs(_object_position.x - _object_position_previous.x) < _object_size_X *
prediction_shift_factor) &&
                (abs(_object_position.y - _object_position_previous.y) < _object_size_Y *
prediction_shift_factor))
        {
                _object_position_predicted.x = _object_position.x + (_object_position.x -
_object_position_previous.x);
                _object_position_predicted.y = _object_position.y + (_object_position.y -
_object_position_previous.y);
        }
        else
        {
                _object_position_predicted.x = _object_position.x;
                _object_position_predicted.y = _object_position.y;
        }

        _object_position_previous.x = _object_position.x;
        _object_position_previous.y = _object_position.y;

        GlobalPerformanceAnalyzer()->StopMeasure(ObjectTracker_ProcessFrame);
}

ScreenCoord ObjectTracker::GetObjectPosition()
{
        ScreenCoord screenCoord;
        if (_targetLocked)
                screenCoord.Init(_object_position.x, _object_position.y);
        else
                screenCoord.Init(-1, -1);
        return screenCoord;
}
```

ПРИЛОЖЕНИЕ Д

(обязательное)

Описание программных средств выделения фрагмента фотоплана по заданным координатам

Для выделения фрагмента фотоплана по заданным координатам использованы следующие функции:

- void SetUAVCoord(WorldGPSCoord coord) – для задания координат объекта наблюдения на фотоплане. Входной параметр: географические координаты объекта наблюдения.
- void SetTargetCoord(WorldGPSCoord coord) – для задания координат БЛА на фотоплане. Входной параметр: географические координаты БЛА.

Листинг файлов для выделения фрагмента фотоплана по заданным координатам:

Заголовочный файл MapTileContainer.h

```
#pragma once
#include "StdAfx.h"
#include "CommonData.h"
#include "sqlite3.h"

// http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames
// http://habrahabr.ru/post/146107/
// http://habrahabr.ru/post/233809/
// http://www.foxbase.ru/Java/google-maps-preobrazovanie-koordinat.htm

struct MapTile
{
	int x;
	int y;
	int scale;
	int lastUsing;
	IplImage* image;
	MapTile()
	{
		x = -1;
		y = -1;
		scale = -1;
		lastUsing = -1;
		image = 0;
	}
	~MapTile()
	{
		cvReleaseImage(&image);
	}
};

class MapTileContainer
{
```

```
private:
	sqlite3 *db = 0;
	sqlite3_stmt *stmt;
	unsigned char *blobData = 0;

	static const int TILE_BUFFER_SIZE = 40;
	unsigned long _usingCounter;
	MapTile _tiles[TILE_BUFFER_SIZE];
	WorldGPSCoord _screenCenter;

	WorldGPSCoord _auvCoord;
	WorldGPSCoord _targetCoord;
	bool _showTarget;

	int _scale;
	IplImage* GetTileImage(int tileX, int tileY, int scale);
	void DrawTileAtMap(IplImage* tile, IplImage* map, int posX, int posY);
public:
	MapTileContainer(char * dbFilePath);
	~MapTileContainer();
	void SetUAVCoord(WorldGPSCoord coord);
	void SetTargetCoord(WorldGPSCoord coord);

	void SetImageCenter(double gps_lat, double gps_lon);
	void SetScale(int scale);
	void GetMapImage(IplImage* image);
};
```

Файл исходного кода MapTileContainer.cpp:

```
#include "StdAfx.h"
#include "MapTileContainer.h"
#include "CommonFunctions.h"

const int TILE_WIDTH = 256;
const int TILE_HEIGHT = 256;

MapTileContainer::MapTileContainer(char * dbFilePath)
{
	_usingCounter = 0;
	sqlite3_open(dbFilePath, &db);
	sqlite3_prepare_v2(db, "SELECT tile FROM MapTile WHERE x=? AND y=? AND
scale=?", -1, &stmt, 0);
	blobData = (unsigned char *)malloc(100000);
}

MapTileContainer::~MapTileContainer()
{
	sqlite3_finalize(stmt);
	sqlite3_close(db);
	free(blobData);
}
```

```
void MapTileContainer::SetUAVCoord(WorldGPSCoord coord)
{
	_auvCoord = coord;
}

void MapTileContainer::SetTargetCoord(WorldGPSCoord coord)
{
	_targetCoord = coord;
}

void MapTileContainer::SetImageCenter(double gps_lat, double gps_lon)
{
	_screenCenter.lat = gps_lat;
	_screenCenter.lon = gps_lon;
}

void MapTileContainer::SetScale(int scale)
{
	_scale = scale;
}

IplImage* MapTileContainer::GetTileImage(int tileX, int tileY, int scale)
{
	MapTile* tile = 0;
	MapTile* obsoleteTile = &_tiles[0];
	MapTile* resultTile = 0;
	for (int i = 0; i < TILE_BUFFER_SIZE; i++)
	{
		tile = &_tiles[i];
		if ((tile->scale == scale) && (tile->x == tileX) && (tile->y == tileY))
		{
			resultTile = tile;
		}
		else if (obsoleteTile->lastUsing > tile->lastUsing)
			obsoleteTile = tile;
	}

	if (resultTile == 0)
	{
		resultTile = obsoleteTile;
		resultTile->x = tileX;
		resultTile->y = tileY;
		resultTile->scale = scale;
		cvReleaseImage(&resultTile->image);

		sqlite3_bind_int(stmt, 1, tileX);
		sqlite3_bind_int(stmt, 2, tileY);
		sqlite3_bind_int(stmt, 3, scale);

		CvMat mat;
		mat.data.ptr = blobData;
```

```
                int stepRes = sqlite3_step(stmt);
                if (stepRes == SQLITE_ROW)
                {
                        int blobSize = sqlite3_column_bytes(stmt, 0);
                        memcpy(blobData, sqlite3_column_blob(stmt, 0), blobSize);
                        IplImage * decodedTile = cvDecodeImage(&mat,
CV_LOAD_IMAGE_COLOR);
                        resultTile->image = decodedTile;
                }
                sqlite3_reset(stmt);
        }

        resultTile->lastUsing = _usingCounter;
        return resultTile->image;
}

void MapTileContainer::DrawTileAtMap(IplImage* tile, IplImage* map, int posX, int posY)
{
        char * tileData = tile->imageData;
        char * mapData = map->imageData;
        int pixelSize = tile->nChannels;

        int mapHeight = map->height;
        int mapLineWidth = map->width;
        int mapLineWidthB = mapLineWidth * pixelSize;

        int tileHeight = tile->height;
        int tileLineWidth = tile->width;
        int tileLineWidthB = tileLineWidth * pixelSize;

        int tileLineCopyLengthB = tileLineWidthB;
        if (posX < 0)
                tileLineCopyLengthB = tileLineWidthB + posX * pixelSize;
        else if (posX > mapLineWidth - tileLineWidth)
                tileLineCopyLengthB = mapLineWidthB - posX * pixelSize;

        if (tileLineCopyLengthB <= 0)
                return;

        int tileLineOffsetB = 0;
        if (posX < 0)
                tileLineOffsetB = tileLineWidthB - tileLineCopyLengthB;

        int mapLineOffsetB = 0;
        if (posX > 0)
                mapLineOffsetB = posX * pixelSize;

        int tileBlockOffsetB = 0;
        if (posY < 0)
                tileBlockOffsetB = -posY * tileLineWidthB;

        int mapBlockOffsetB = 0;
```

```
        if (posY > 0)
                mapBlockOffsetB = posY * mapLineWidthB;

        int copyLineCount = tileHeight;
        if (posY < 0)
                copyLineCount = tileHeight + posY;
        else if (posY + tileHeight > mapHeight)
                copyLineCount = mapHeight - posY;//posY + tileHeight - mapHeight;

        tileData = tileData + tileBlockOffsetB + tileLineOffsetB;
        mapData = mapData + mapBlockOffsetB + mapLineOffsetB;

        for (int i = 0; i < copyLineCount; i++)
        {
                memcpy(mapData, tileData, tileLineCopyLengthB);
                tileData += tileLineWidthB;
                mapData += mapLineWidthB;
        }
}


void ConvertGPS2XY(WorldGPSCoord coord, int scale, double &x, double &y)
{
        double scaleK = pow((double)2, scale);
        x = ((coord.lon + 180) / 360) * scaleK;
        y = (1 - log(tan(deg2rad(coord.lat)) + 1 / cos(deg2rad(coord.lat))) / PI) / 2 * scaleK;
}

void MapTileContainer::GetMapImage(IplImage* image)
{
        int imageHeight = image->height;
        int imageWidth = image->width;

        /*
        int offset = 256 << (scale - 1);
        int x = round(offset + (offset * gps_lon / 180));
        int y = round(offset - offset / PI * log((1 + sin(gps_lat * PI / 180)) / (1 - sin(gps_lat * PI /
180))) / 2);
        */
        double centerX, centerY;
        ConvertGPS2XY(_screenCenter, _scale, centerX, centerY);

        int tileX = (int)floor(centerX);
        int tileY = (int)floor(centerY);
        int offsetX = (int)floor((centerX - tileX) * TILE_WIDTH);
        int offsetY = (int)floor((centerY - tileY) * TILE_HEIGHT);
        _usingCounter++;

        int fromTileX = (int)ceil((double)tileX - (imageWidth / 2 - offsetX) / TILE_WIDTH - 1);
        int toTileX = (int)ceil((double)tileX + (imageWidth / 2 + offsetX) / TILE_WIDTH + 0);
        int fromTileY = (int)ceil((double)tileY - (imageHeight / 2 - offsetY) / TILE_HEIGHT - 1);
        int toTileY = (int)ceil((double)tileY + (imageHeight / 2 + offsetY) / TILE_HEIGHT + 0);
```

```
	//Draw tiles
	for (int i = fromTileX; i <= toTileX; i++)
	for (int j = fromTileY; j <= toTileY; j++)
	{
		IplImage* tileImage = GetTileImage(i, j, _scale);
		if (tileImage != 0)
		{
			int posX = (i - tileX) * TILE_WIDTH + imageWidth / 2 - offsetX;
			int posY = (j - tileY) * TILE_HEIGHT + imageHeight / 2 - offsetY;
			DrawTileAtMap(tileImage, image, posX, posY);
			//DrawTileAtMap(tileImage, image, (i - tileX) * TILE_WIDTH - offsetX, (j -
tileY) * TILE_HEIGHT - offsetY);
		}
	}

	int objectScreenX, objectScreenY;
	// Draw UAV
	double uavX, uavY;
	ConvertGPS2XY(_auvCoord, _scale, uavX, uavY);
	objectScreenX = (int)((uavX - centerX) * TILE_WIDTH + imageWidth / 2);
	objectScreenY = (int)((uavY - centerY) * TILE_HEIGHT + imageHeight / 2);
	cvCircle(image, cvPoint(objectScreenX, objectScreenY), 10, CV_RGB(0, 255, 0), 2);

}
```

ПРИЛОЖЕНИЕ Е

(обязательное)

Описание программных средств определения координат объекта наблюдения по одному кадру видеопотока и телеметрии

Для определения координат объекта наблюдения по одному кадру видеопотока и телеметрии использована следующая функция:

- static void ConvertScreenCoord2WorldCoord(TelemetryData * telemetryData, CamPreferences * camPreferences, ScreenCoord * screenCoord, WorldGPSCoord * worldGPSCoord). Входные параметры: характеристики установленной на борту БЛА видеокамеры, данные телеметрии БЛА и пиксельные координат и пиксельные координаты объекта наблюдения.

Листинг файлов для определения координат объекта наблюдения по одному кадру видеопотока и телеметрии:

Заголовочный файл CoordConverter.h:

```
#pragma once
#include "StdAfx.h"
#include "CommonData.h"
#include "CommonFunctions.h"

static void DecodeCoord(double coord, int& grad, int& min, double& sec)
{
	double absCoord = abs(coord);
	grad = int(absCoord);
	min = int(60.0 * (absCoord - grad));
	sec = (absCoord - grad - double(min) / 60.0) * 3600.0;
}

static double EncodeCoord(int grad, int min, double sec)
{
	double result = grad + double(min) / 60.0 + sec / 3600.0;
	return result;
}

static void GetGeoCoordAsStr(double coord, char minLetter, char maxLetter, char * str)
{
	int grad, min;
	double sec;
	DecodeCoord(coord, grad, min, sec);
	char site = coord <= 0 ? minLetter : maxLetter;
	sprintf(str, "%c %2d %2d' %5.2f\"", site, grad, min, sec);
}

static void CalculateCamCorrectionAngles(CamPreferences * camPreferences, ScreenCoord * targetCoord, double & angle_cor_X, double & angle_cor_Y)
{
	int frame_width = camPreferences->image_width_pix;
```

```
	int frame_height = camPreferences->image_height_pix;

	int object_shift_X = frame_width / 2 - targetCoord->x;
	int object_shift_Y = frame_height / 2 - targetCoord->y;
	angle_cor_X = object_shift_X * camPreferences->image_width_grad / frame_width;
	angle_cor_Y = object_shift_Y * camPreferences->image_height_grad / frame_width;
}

static void ConvertScreenCoord2WorldCoord(
	TelemetryData * telemetryData,
	CamPreferences * camPreferences,
	ScreenCoord * screenCoord,
	WorldGPSCoord * worldGPSCoord)
{
	double UAV_latitude = telemetryData->gps_lat;
	double UAV_longitude = telemetryData->gps_lon;
	double UAV_altitude = telemetryData->gps_hmsl;
	double UAV_pitch = telemetryData->pitch;
	double UAV_roll = telemetryData->roll;
	double UAV_yaw = telemetryData->yaw;
	double camera_pitch = telemetryData->cam_pitch;
	double camera_roll = telemetryData->cam_heading;

	double k_1 = camPreferences->radial_distortion_k_1;
	double k_2 = camPreferences->radial_distortion_k_2;
	double k_3 = camPreferences->radial_distortion_k_3;
	double deviation_X = camPreferences->center_deviation_X_pix;
	double deviation_Y = camPreferences->center_deviation_Y_pix;
	double X_cam = camPreferences->place_on_UAV_X_m;
	double Y_cam = camPreferences->place_on_UAV_Y_m;
	double Z_cam = camPreferences->place_on_UAV_Z_m;
	double half_frame_X_rad = camPreferences->image_width_rad() / 2;
	double half_frame_Y_rad = camPreferences->image_height_rad() / 2;
	double half_frame_X = camPreferences->image_width_pix / 2;
	double half_frame_Y = camPreferences->image_height_pix / 2;

	double R_m = UAV_altitude;
	double object_shift_X = half_frame_X - screenCoord->x;
	double object_shift_Y = half_frame_Y - screenCoord->y;

	double pixel_scale_X = R_m * tan(half_frame_X_rad) / half_frame_X;
	double pixel_scale_Y = R_m * tan(half_frame_Y_rad) / half_frame_Y;

	double x_n = (object_shift_X - deviation_X) / (half_frame_X - deviation_X) * (R_m * tan(half_frame_X_rad) - deviation_X * pixel_scale_X);
	double y_n = (object_shift_Y - deviation_Y) / (half_frame_Y - deviation_Y) * (R_m * tan(half_frame_Y_rad) - deviation_Y * pixel_scale_Y);

	double x_corr = x_n * (1 + k_1 * (pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)) + k_2 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 2) + k_3 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 3));
```

```
        double y_corr = y_n * (1 + k_1 * (pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)) + k_2 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 2) + k_3 * pow((pow(object_shift_X - deviation_X, 2) + pow(object_shift_Y - deviation_Y, 2)), 3));

        double sin_camera_pitch = sin(camera_pitch);
        double cos_camera_pitch = cos(camera_pitch);
        double sin_camera_roll = sin(camera_roll);
        double cos_camera_roll = cos(camera_roll);
        double sign_camera_pitch = sign(camera_pitch);
        double cos_UAV_yaw = cos(UAV_yaw);
        double sin_UAV_yaw = sin(UAV_yaw);
        double cos_UAV_pitch = cos(UAV_pitch);
        double sin_UAV_pitch = sin(UAV_pitch);
        double sin_UAV_roll = sin(UAV_roll);
        double cos_UAV_roll = cos(UAV_roll);


        double delta_X = y_corr * sin_camera_pitch;
        double delta_Y = cos_camera_roll * x_corr - sin(camera_roll * sign_camera_pitch) * cos_camera_pitch * y_corr;
        double delta_Z = -(sin(camera_roll * sign_camera_pitch * x_corr + cos_camera_roll * cos_camera_pitch) * y_corr);

        double X_m = R_m * cos_camera_pitch;
        double Y_m = (R_m * sin_camera_pitch) * sin_camera_roll;
        double Z_m = (R_m * sin_camera_pitch) * cos_camera_roll * sign_camera_pitch;

        double X_kn = X_m + delta_X;
        double Y_kn = Y_m + delta_Y;
        double Z_kn = Z_m + delta_Z;

        double X_in = X_kn * cos_UAV_yaw * cos_UAV_pitch + Y_kn * sin_UAV_yaw * cos_UAV_pitch - Z_kn * sin_UAV_pitch;
        double Y_in = X_kn * (cos_UAV_yaw * sin_UAV_pitch * sin_UAV_roll - sin_UAV_yaw * cos_UAV_roll) + Y_kn * (sin_UAV_yaw * sin_UAV_pitch * sin_UAV_roll + cos_UAV_yaw * cos_UAV_roll) + Z_kn * cos_UAV_pitch * sin_UAV_roll;
        double Z_in = X_kn * (cos_UAV_yaw * sin_UAV_pitch * cos_UAV_roll + sin_UAV_yaw * sin_UAV_roll) + Y_kn * (sin_UAV_yaw * sin_UAV_pitch * cos_UAV_roll - cos_UAV_yaw * sin_UAV_roll) + Z_kn * cos_UAV_pitch * cos_UAV_roll;

        double X_io = X_cam * cos_UAV_yaw * cos_UAV_pitch + Y_cam * sin_UAV_yaw * cos_UAV_pitch - Z_cam * sin_UAV_pitch;
        double Y_io = X_cam * (cos_UAV_yaw * sin_UAV_pitch * sin_UAV_roll - sin_UAV_yaw * cos_UAV_roll) + Y_cam * (sin_UAV_yaw * sin_UAV_pitch * sin_UAV_roll + cos_UAV_yaw * cos_UAV_roll) + Z_cam * cos_UAV_pitch * sin_UAV_roll;
        double Z_io = X_cam * (cos_UAV_yaw * sin_UAV_pitch * cos_UAV_roll + sin_UAV_yaw * sin_UAV_roll) + Y_cam * (sin_UAV_yaw * sin_UAV_pitch * cos_UAV_roll - cos_UAV_yaw * sin_UAV_roll) + Z_cam * cos_UAV_pitch * cos_UAV_roll;

        double t_m = (UAV_altitude - Z_io) / (Z_in - Z_io);
```

```
	double X_norm = (X_in - X_io) * t_m + X_io;
	double Y_norm = (Y_in - Y_io) * t_m + Y_io;
	double Z_norm = (Z_in - Z_io) * t_m + Z_io;

	double delta_longitude = Y_norm / (EARTH_RADIUS_M * cos_UAV_roll);
	double delta_latitude = X_norm / EARTH_RADIUS_M;

	worldGPSCoord->Init(UAV_latitude + delta_latitude, UAV_longitude + delta_longitude, 0); //??? hmsl is incorrect
}
```

Файл исходного кода TestTrackingMain.cpp:

```
#include "stdafx.h"

#include "ImageStabilizer.h"
#include "ObjectTracker.h"
#include "UAVImageProcessor.h"
#include "MapTileContainer.h"
#include "PerformanceAnalyzer.h"
#include "CoordConverter.h"

UAVImageProcessor* imageProcessor;

void onMouseEvent(int event, int x, int y, int flags, void* param)
{
	switch (event)
	{
	case CV_EVENT_LBUTTONDOWN:
		imageProcessor->TargetLockOn(x, y);
		break;
	case CV_EVENT_MOUSEMOVE:
		bool useMagnifier = (flags & CV_EVENT_FLAG_SHIFTKEY) == CV_EVENT_FLAG_SHIFTKEY;

		imageProcessor->ShowSight(x, y);
		break;
	}
}

int main(int argc, char *argv[])
{
	omp_set_num_threads(2);
	//int thread_count = omp_get_max_threads();

	char* fileName;
	if (argc > 1)
		fileName = argv[1];
	else
	{
		//fileName = "d:\\Tracking\\C++\\Test videos\\1_403-553_zoom.avi";
```

```
		//fileName = "d:\\Tracking\\C++\\Test videos\\2_994-1044_opposite.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\3_710-860.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\4_1184-1234_shift.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\5_1336-1486.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\6_3820-3970.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\7_2322-2472.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\8_15142-15292.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\9_16200-16350.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\10_16435-16585.avi";
		//fileName = "d:\\Tracking\\C++\\Test videos\\11_tracking_950_1150.avi";
		//fileName = "d:\\video\\сопровождение_300_1000.avi";
		//fileName = "d:\\Video\\02.10.2014\\2014-10-01  18-27.avi";
		//fileName = "d:\\Video\\2014-09-30  12-14.avi";
		//fileName = "d:\\Projects\\Video\\1\\2014-10-06  15-11 10000-12000.avi";
		//fileName = "d:\\Projects\\ObjectTracking\\Sopr3.avi ";

		fileName = "d:\\Video\\сопровождение_1106_1646.avi";
		fileName = "d:\\Video\\2014-09-30  12-14.avi";
		fileName = "d:\\Video\\Видео с Борта\\session11.ts";
		fileName = "d:\\Video\\Видео с Борта\\session11_52400_52800.avi";
		//fileName = "d:\\video\\сопровождение_241_641.avi";
		//fileName = "d:\\Video\\02.10.2014\\2014-10-01  18-27.avi";
	}

	CvCapture* capture = 0;
	capture = cvCaptureFromAVI(fileName);
	if (!capture)
	{
		fprintf(stderr, "Could not initialize capturing…\n");
		return -1;
	}

	cvNamedWindow("Transformed", CV_WINDOW_AUTOSIZE);
	// mouse ivent for object selecting
	cvSetMouseCallback("Transformed", onMouseEvent, 0);

	CvFont font;
	cvInitFont(&font, CV_FONT_HERSHEY_COMPLEX_SMALL, 1.0, 1.0);

	int frameWidth = (int)cvGetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH);
	int frameHeight = (int)cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT);
	int frameChannel = 3;
	imageProcessor = new UAVImageProcessor(frameWidth, frameHeight, frameChannel);

	MapTileContainer* mapTileContainer = new
MapTileContainer("D:\\Tracking\\C++\\OpenCVTest1_20141030_1010\\MapTiles\\GoogleTiles.db
");

	IplImage* mapImage = cvCreateImage(cvSize(1200, 768), IPL_DEPTH_8U,
frameChannel);
	int mapScale = 17;
```

```
	IplImage* transformedFrame = cvCreateImageHeader(cvSize(frameWidth, frameHeight), IPL_DEPTH_8U, frameChannel);
	IplImage* currFrame = 0;

	TelemetryData telemetryData;
	telemetryData.gps_lon = 28.4;
	telemetryData.gps_lat = 54.2;
	telemetryData.gps_hmsl = 200;
	telemetryData.pitch = 90;
	telemetryData.yaw = 0;
	telemetryData.yaw = 0;

	WorldGPSCoord uavCoord, targetCoord;

	int first_frame = 1;

	//memset(&telemetryData, 0, sizeof(telemetryData));
	imageProcessor->SetUseSightMagnifier(!imageProcessor->GetUseSightMagnifier());
	for (;;)
	{

		if (first_frame == 1)
		{
			currFrame = cvQueryFrame(capture);
			telemetryData.gps_lat += EncodeCoord(0, 0, 0.5);
			//telemetryData.gps_lat -= EncodeCoord(0, 0, 0.05);
			//telemetryData.gps_lon += EncodeCoord(0, 0, 0.04);
			uavCoord.Init(telemetryData.gps_lat, telemetryData.gps_lon, telemetryData.gps_hmsl);

			mapTileContainer->SetImageCenter(telemetryData.gps_lat, telemetryData.gps_lon);
			mapTileContainer->SetScale(mapScale);
			mapTileContainer->GetMapImage(mapImage);
			if (mapImage != 0)
				cvShowImage("MAP", mapImage);
			imageProcessor->ProcessFrame(currFrame->imageData, &telemetryData);
			transformedFrame->imageData = imageProcessor->GetProcessedFrame(true);

			if (transformedFrame->imageData != 0)
			{
				cvShowImage("Transformed", transformedFrame);
			}
			first_frame = 0;
			int key = cvWaitKey(0);
		}
		currFrame = cvQueryFrame(capture);
		if (!currFrame)
			break;
		telemetryData.gps_lat += EncodeCoord(0, 0, 0.5);
```

```
		//telemetryData.gps_lat -= EncodeCoord(0, 0, 0.05);
		//telemetryData.gps_lon += EncodeCoord(0, 0, 0.04);
		uavCoord.Init(telemetryData.gps_lat, telemetryData.gps_lon,
telemetryData.gps_hmsl);

		mapTileContainer->SetImageCenter(telemetryData.gps_lat, telemetryData.gps_lon);
		mapTileContainer->SetScale(mapScale);
		mapTileContainer->GetMapImage(mapImage);
		if (mapImage != 0)
			cvShowImage("MAP", mapImage);

		int64 tickCount = cvGetTickCount();

		imageProcessor->ProcessFrame(currFrame->imageData, &telemetryData);
		transformedFrame->imageData = imageProcessor->GetProcessedFrame(true);

		tickCount = cvGetTickCount() - tickCount;

		char time_ch[16];
		sprintf(time_ch, "%f", tickCount / (cvGetTickFrequency()*1000.));

		////////////////////////////////////////////////////////////////////////////////////
		////////////////////////////////////////////////////////////////////////////////////
		////////////////////////////////// ANGLES CORRECTION ///////////////////////////////////////
		////////////////////////////////////////////////////////////////////////////////////
		////////////////////////////////////////////////////////////////////////////////////
		double angle_cor_X, angle_cor_Y;
		imageProcessor->GetCamCorrectionAngles(angle_cor_X, angle_cor_Y);

		// show results on the frame
		char angle_corX_ch[16];
		char angle_corY_ch[16];
		sprintf(angle_corX_ch, "%f", angle_cor_X);
		sprintf(angle_corY_ch, "%f", angle_cor_Y);
		cvPutText(transformedFrame, angle_corX_ch, cvPoint(550, 420), &font,
CV_RGB(0, 255, 255));
		cvPutText(transformedFrame, angle_corY_ch, cvPoint(550, 440), &font,
CV_RGB(0, 255, 255));
		////////////////////////////////////////////////////////////////////////////////////
		////////////////////////////////////////////////////////////////////////////////////
		////////////////////////////////////////////////////////////////////////////////////

		//cvShowImage("Source", currFrame);
		if (transformedFrame->imageData != 0)
		{
			cvPutText(transformedFrame, time_ch, cvPoint(50, 50), &font, CV_RGB(0,
255, 255));
			cvShowImage("Transformed", transformedFrame);
		}

		char c = cvWaitKey(40);
```

```
		if (c == 27)
			break;
		switch (c)
		{
		case '1':
			imageProcessor->SetTargetRectSize(10);
			break;
		case '2':
			imageProcessor->SetTargetRectSize(15);
			break;
		case '3':
			imageProcessor->SetTargetRectSize(20);
			break;
		case '4':
			imageProcessor->SetTargetRectSize(30);
			break;
		case '5':
			imageProcessor->SetTargetRectSize(40);
			break;
		case 'r':
		case 'R':
			imageProcessor->DropTarget();
			break;

		case 's':
		case 'S':
			imageProcessor->SetUseSightMagnifier(!imageProcessor-
>GetUseSightMagnifier());
			break;

		case 'x':
		case 'X':
			imageProcessor->GetMaxStabilizationOffset() == 0 ? imageProcessor-
>SetMaxStabilizationOffset(200) : imageProcessor->SetMaxStabilizationOffset(0);
			break;

		case 'q':
		case 'Q':
			mapScale = mapScale > 11 ? mapScale - 1 : 17;
			break;
		case 'w':
		case 'W':
			mapScale = mapScale < 17 ? mapScale + 1 : 11;
			break;
		default:
			break;
		}
	}

	cvReleaseImageHeader(&transformedFrame);
	cvReleaseCapture(&capture);
	cvDestroyAllWindows();
```

```
        GlobalPerformanceAnalyzer()->PrintStatistics();
        cvWaitKey(-10);


        delete imageProcessor;
        return 0;
}
```

ПРИЛОЖЕНИЕ Ж

(обязательное)

Листинг программы локализации прямых контурных линий для совмещения изображений

```
clc;
close all;
clear all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ЗАДАЕМ РАЗМЕРНОСТЬ БЛОКА
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ПОДГОТОВКА ИЗОБРАЖЕНИЯ ДЛЯ ОБРАБОТКИ
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rgb_img = imread('source.jpg');
q = .2989*rgb_img(:,:,1)...
 +.5870*rgb_img(:,:,2)...
 +.1140*rgb_img(:,:,3);

% hIdtc = video.ImageDataTypeConverter();
% hCsc = video.ColorSpaceConverter('Conversion','RGB to intensity');
% I3chan = step(hIdtc,imread('n20'));
% q = step(hCsc,I3chan);
% q = mat2gray(q);

[h,l] = size(q);
h = floor(h/r)*r;
l = floor(l/r)*r;

i = q(1:h, 1:l);
BW = edge(i, 'Canny');
figure(1), imshow(BW); title('Контурное изображение исходного изображения');
% imwrite(BW,['bw' '.bmp']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ПРОИЗВОДИМ ОБРАБОТКУ БЛОКОВ РАЗМЕРНОСТЬЮ 3x3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
IMAGE1 = zeros(h,l);
IMAGE2 = zeros(h,l);
IMAGE3 = zeros(h,l);
IMAGE4 = zeros(h,l);

Start = cputime;

for y = 2:1:(h-1)
   for x = 2:1:(l-1)
      if BW(y,x) == 1

         if BW((y-1),x) == 1 && BW((y+1),x) == 1
```

```
            IMAGE1((y-1):(y+1), (x-1):(x+1)) = BW((y-1):(y+1), (x-1):(x+1));
        elseif BW(y,(x-1)) == 1 && BW(y,(x+1)) == 1
            IMAGE2((y-1):(y+1), (x-1):(x+1)) = BW((y-1):(y+1), (x-1):(x+1));
        elseif BW((y-1),(x-1)) == 1 && BW((y+1),(x+1)) == 1
            IMAGE3((y-1):(y+1), (x-1):(x+1)) = BW((y-1):(y+1), (x-1):(x+1));
        elseif BW((y-1),(x+1)) == 1 && BW((y+1),(x-1)) == 1
            IMAGE4((y-1):(y+1), (x-1):(x+1)) = BW((y-1):(y+1), (x-1):(x+1));
        end;

      end;
   end;
end;

Elapsed = cputime - Start;

figure(2), imshow(IMAGE1); title('Контурное изображение 90 градусов');
figure(3), imshow(IMAGE2); title('Контурное изображение 0 градусов');
figure(4), imshow(IMAGE3); title('Контурное изображение -45 градусов');
figure(5), imshow(IMAGE4); title('Контурное изображение 45 градусов');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ПРЕОБРАЗОВАНИЕ ХАФА
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[H,theta,rho] = hough(IMAGE1,'RhoResolution',0.45,'Theta', [-25:0.5:25]);
P = houghpeaks(H,100,'threshold',0 );
x = theta(P(:,2));
y = rho(P(:,1));
lines1 = houghlines(IMAGE1,theta,rho,P,'FillGap',10,'MinLength',15);
figure(6), imshow(q), hold on
max_len = 0;
for k = 1:length(lines1)
   xy = [lines1(k).point1; lines1(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   len = norm(lines1(k).point1 - lines1(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
end;
title('Прямые линии под 90 градусов');

line1 = lines1;

[H,theta,rho] = hough(IMAGE2,'RhoResolution',0.6,'Theta', [-90:0.5:-65 , 65:0.5:89.5 ]);
P = houghpeaks(H,100,'threshold',0 );
x = theta(P(:,2));
y = rho(P(:,1));
lines1 = houghlines(IMAGE2,theta,rho,P,'FillGap',10,'MinLength',15);
figure(7), imshow(q), hold on
```

```
max_len = 0;
for k = 1:length(lines1)
   xy = [lines1(k).point1; lines1(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   len = norm(lines1(k).point1 - lines1(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
end;
title('Прямые линии под 0 градусов');

line2 = lines1;

[H,theta,rho] = hough(IMAGE3,'RhoResolution',0.5,'Theta', [-80:0.5:-10]);
P = houghpeaks(H,50,'threshold',0 );
x = theta(P(:,2));
y = rho(P(:,1));
lines1 = houghlines(IMAGE3,theta,rho,P,'FillGap',10,'MinLength',10);
figure(8), imshow(q), hold on
max_len = 0;
for k = 1:length(lines1)
   xy = [lines1(k).point1; lines1(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   len = norm(lines1(k).point1 - lines1(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
end;
title('Прямые линии под -45 градусов');

line3 = lines1;

[H,theta,rho] = hough(IMAGE4,'RhoResolution',0.5,'Theta', [10:0.5:80]);
P = houghpeaks(H,50,'threshold',0 );
x = theta(P(:,2));
y = rho(P(:,1));
lines1 = houghlines(IMAGE4,theta,rho,P,'FillGap',10,'MinLength',10);
figure(9), imshow(q), hold on
max_len = 0;
for k = 1:length(lines1)
   xy = [lines1(k).point1; lines1(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
```

```
   len = norm(lines1(k).point1 - lines1(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
end;
title('Прямые линии под 45 градусов');

line4 = lines1;

all_lines = cat(2, line1, line2, line3, line4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ПРЕОБРАЗОВАНИЕ ХАФА ИСХОДНОГО ИЗОБРАЖЕНИЯ
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 [H,theta,rho] = hough(BW,'RhoResolution',0.5,'ThetaResolution', 0.5);

P = houghpeaks(H,40,'threshold',0 );
x = theta(P(:,2));
y = rho(P(:,1));
lines1 = houghlines(BW,theta,rho,P,'FillGap',10,'MinLength',5);

figure(10), imshow(q), hold on
max_len = 0;
for k = 1:length(lines1)
   xy = [lines1(k).point1; lines1(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
   plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
   plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');

   len = norm(lines1(k).point1 - lines1(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
end;
title('Классическое преобразование Хафа');

q2 = zeros(600,600);

figure(69), imshow(q2), hold on
max_len = 0;
for k = 1:length(all_lines)
   xy = [all_lines(k).point1; all_lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','white');
   len = norm(all_lines(k).point1 - all_lines(k).point2);
   if ( len > max_len)
   max_len = len;
   xy_long = xy;
   end;
```

```
end;
title('Масочно-фазовое преобразование Хафа');

%%%%%%%%%%%%%%%%%%%%%%%%%%
% СБОР СТАТИСТИКИ
%%%%%%%%%%%%%%%%%%%%%%%%%%

TOTAL_POINTS = sum(sum(BW));
INT_POINTS = sum(sum(IMAGE1)) + sum(sum(IMAGE2)) + sum(sum(IMAGE3)) +
sum(sum(IMAGE4));
INT_POINTS_PERCENT =  INT_POINTS/TOTAL_POINTS;
```