|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Информационные системы и телекоммуникации»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент ИУ3-81 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  И. Д. Родионов **\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Лычков \_\_\_\_

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  С.А. Тренин\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

*2017 г.*

# АННОТАЦИЯ

Данный дипломный проект посвящен разработке метода оценки скорости объекта в видеопотоке, с движущейся камеры. В данной работе были рассмотрены методы и алгоритмы, предлагаемые для решения этой задачи. Разработана программа, реализующая выбранные методы и алгоритмы, проведено тестирование программы.

Актуальность данной темы заключается в замещении аппаратных решений программными в целях экономии средств выделяемых на дополнительное оборудование и его обслуживание.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 2](#_Toc483484477)

[ВВЕДЕНИЕ 4](#_Toc483484478)

[1 Исследовательская часть 5](#_Toc483484479)

[1.1 Обзор существующих методов стабилизации видеопотока 5](#_Toc483484480)

[1.1.1 Метод сопоставления блоков 5](#_Toc483484481)

[1.1.2 Метод стабилизации видеопотока с помощью оптимальных траекторий камеры 6](#_Toc483484482)

[1.1.3 Выбор алгоритма стабилизации видеопотока 7](#_Toc483484483)

[1.1.4 Обзор методов отслеживания объекта 10](#_Toc483484484)

[1.1.4.1 Метод Моравеца 11](#_Toc483484485)

[1.1.4.2 Метод Ростена и Драдмонда 11](#_Toc483484486)

[1.1.5 Выбор метода отслеживания объекта в видеопотоке 12](#_Toc483484487)

[2 Конструкторская часть 13](#_Toc483484488)

[2.1 Программная реализация метода стабилизации 13](#_Toc483484489)

[2.1.1 Корреляционный метод смещения 13](#_Toc483484490)

[2.1.1.1 Расчет корреляции 13](#_Toc483484491)

[2.1.1.2 Расчет смещения двух кадров 14](#_Toc483484492)

[2.1.1.3 Построение пирамид изображения 15](#_Toc483484493)

[2.1.1.4 Результат работы методов 2.1.1.1, 2.1.1.2, 2.1.1.3 в совокупности 16](#_Toc483484494)

[2.1.2 Функция поворота изображения 18](#_Toc483484495)

[2.1.2.1 Испытание работы функции поворота 18](#_Toc483484496)

[2.1.3 Функция сдвига изображения 20](#_Toc483484497)

[2.1.4 Стабилизация видеопотока 20](#_Toc483484498)

[2.2 Программная реализация метода отслеживания объекта в видеопотоке 22](#_Toc483484499)

[2.3 Разработка и программная реализация метода оценки скорости объекта в видеопотоке 24](#_Toc483484500)

[2.3.1 Оценка скорости движения объекта 25](#_Toc483484501)

[2.4 Оценка точности работы реализованных методов 28](#_Toc483484502)

[3 Технологическая часть 29](#_Toc483484503)

[ЗАКЛЮЧЕНИЕ 30](#_Toc483484504)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31](#_Toc483484505)

[ПРИЛОЖЕНИЕ А 32](#_Toc483484506)

# ВВЕДЕНИЕ

В наше время мы все чаще сталкиваемся с ситуациями, когда необходимо проследить за каким-либо объектом и измерить его скорость передвижения. При этом мы живём в век информационных технологий, которые нам позволяют минимизировать затраты на дополнительное оборудование, которое можно заменить программными методами решения поставленных задач.

Рассмотрим пример применения таких устройств в гражданской сфере: камеры фото и видео фиксации нарушений правил дорожного движения. Данное устройство состоит из двух компонентов: камера фото и видео фиксации, радар. Для того, чтобы определить скорость передвижения объекта, используется метод радиолокации, основанный на излучении радиоволн и регистрации их отражения от объектов. В нашем случае предлагается избежать покупки дополнительного оборудования и отслеживать скорость перемещения объектов в видеопотоке за счет обработки получаемой информации из видеопотока в реальном времени, которая будет обрабатываться программно.

Рассмотрим гипотетический пример из военной сферы. Предполагается отследить скорость перемещения вражеского объекта. Традиционно будет использоваться метод радиолокации. При использовании подобного метода, вражеский объект сможет засечь излучение радиоволн и сменить тактику ведения боя. Чтобы избежать обнаружения и незаметно измерить скорость передвижения вражеского объекта, нужно измерить скорость вражеского объекта без использования средств, которые могут нас обнаружить.

Рассмотрев вышеприведённые примеры можно сделать вывод, что разработка алгоритмов и методов оценки скорости движения объекта в видеопотоке актуальна.

В данной работе будет рассмотрен вариант реализации такого метода, с использованием среды разработки программного обеспечения и камеры для формирования видеопотока.

# 1 Исследовательская часть

# Обзор существующих методов стабилизации видеопотока

Существует два основных подхода для решения этой проблемы: механическая и цифровая обработка видеопотока.

Механический подход, или оптическая стабилизация, активизирует оптические системы для настройки датчиков движения камеры при вибрации, дрожании, например фиксация камеры или использование гироскопических стабилизаторов. Этот метод хорошо работает на практике, но не всегда используется из-за денежных расходов и ограничений в движении камеры.

Цифровая обработка видеопотока заключается в программной стабилизации видеопотока. Ниже мы рассмотрим методы программной стабилизации видеопотока.

# Метод сопоставления блоков

Суть подхода к решению стабилизации методом сопоставления блоков строится на предположении, что за время, которое пройдёт в момент смены двух последовательных кадров, местоположение объекта наблюдения изменится незначительно. Тогда в области любой точки кадра это изменение с достаточно высокой точностью можно приблизить переносом этой области на некоторый вектор[2].

Метод совмещения блоков состоит из следующих основных шагов:

* Кадр разбивается на множество непересекающихся блоков
* Для каждого блока следующего кадра ищется наиболее похожий блок предыдущего кадра
* Оценивается величина смещения положения блока в текущем кадре относительно предыдущего

# Метод стабилизации видеопотока с помощью оптимальных траекторий камеры

Данный алгоритм стабилизации видео используется на YouTube. Алгоритм[3] разработан в научно-исследовательском подразделении Google Research.

Алгоритм основан на структуре линейного программирования, чтобы минимизировать первую, вторую и третью производные результирующего пути камеры. Метод позволяет обеспечить стабилизацию видеопотока не только для высокочастотных колебаний, но и для колебаний низкой частоты.

Недостатки данного алгоритма:

* При недостаточном количестве жестко зафиксированных объектов на сцене или при резком движении, приближение к пути камеры недостаточно точное
* Использование обрезания кадра уменьшает количество информации

# Выбор алгоритма стабилизации видеопотока

Алгоритм стабилизации включает в себя следующие методы:

* Обнаружение смещения для текущего кадра
* Функция сдвига/поворота изображения

Обнаружение смещения текущего кадра

Основной подход по определению смещения, будет основываться на корреляционном методе, который можно описать следующим образом:

1. Выделяется центральная часть опорного изображения. Величина отступа определяется максимальным смещением, которое мы желаем определить. Центральная часть опорного изображения не должна быть слишком маленькой, иначе у функции, которая считает корреляцию, не будет хватать данных для стабильной работы.
2. На текущем кадре выбирается часть изображения такого же размера, смещенная относительно центра изображения.
3. Для каждого смещения рассчитывается расстояние, описывающее корреляцию центрально части опорного изображения и текущего изображения.
4. Смещение, корреляция для которого будет максимальной, будет искомым смещением.

Рассмотрим пример смещений на рисунке 1.

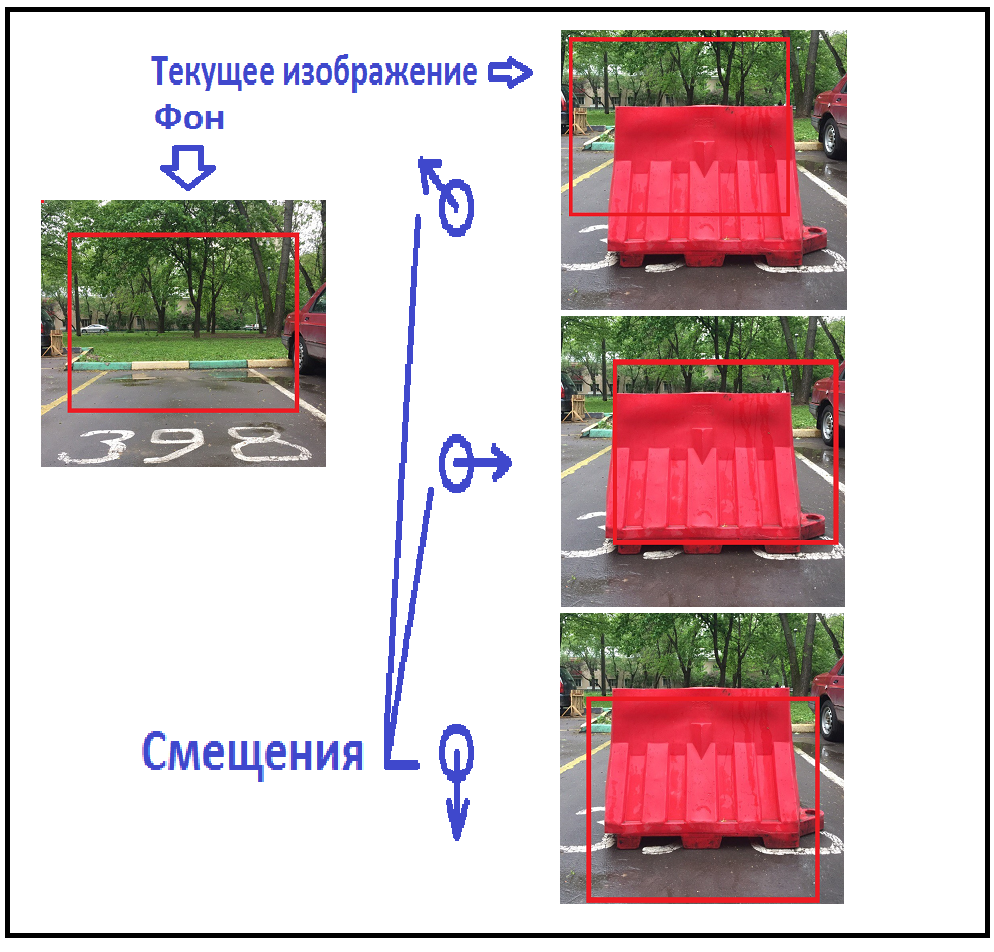


Рис.1 Смещение текущего кадра относительно фона

Если мы просто применим такой подход, без дополнительных методов, то скорость работы алгоритма будет катастрофически низкой, даже если скорость корреляционной функции будет очень высока. Так как нам нужно перебирать все варианты возможного смещения изображений относительно друг друга.

Решением это проблемы будет использование пирамидальной структуры[5] данных, что дает нам:

* Сокращение времени обработки изображения
* Определение более точных начальных приближений для обработки нижних уровней по результатам обработки верхних уровней

Локальный максимум корреляции будет искаться для максимального масштаба, затем на меньших масштабах его последовательно уточнять.

Пример пирамид изображений представлен на рисунке 2.



Рис. 2 Пирамидальная структура изображений

Использование функции сдвига(поворота) изображения

После того как мы реализовали метод расчета смещения изображения относительно текущего кадра, мы должны приступить к стабилизации.

Для стабилизации изображения нам необходим опорный кадр, относительно которого мы будем двигать/поворачивать каждый следующий кадр. Для сдвига/поворота изображения мы будем использовать функцию warpAffine, которая описана в библиотеке компьютерного зрения OpenCV.

OpenCV – библиотека алгоритмов компьютерного зрения и обработки изображений. Реализована на С/C++[6].

warpAffine – выполняет общее аффинное преобразование исходного изображения. В OpenCV аффинное преобразование представляет собой матрицу 2×3[7]. Первые два столбца этой матрицы кодируют вращение, масштаб и сдвиг, а последний столбец кодирует сдвиг:

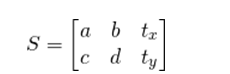


Рис. 3 Матрица аффинного преобразования

В случае если нам нужно поворачивать изображение мы инициализируем матрицу следующим образом:

(1)

где – угол поворота изображения,

tx и ty параметры сдвига по оси абсцисс и ординат соответственно.

Используя значения смещения из расчетов смещения изображения корреляционным методом, мы инициализируем наши параметры сдвига tx и ty.

После всех этапов обработки изображения мы получаем стабилизацию видеопотока относительно опорного кадра.

# Обзор методов отслеживания объекта

Чтобы выделить на изображении интерпретируемую информацию необходимо привязаться к локальным особенностям изображения. На изображениях выделяют особые точки. Особая точка – это точка на изображении окрестность, которой можно отличить от окрестности другой точки на изображении. Определяют особые точки с помощью детекторов или дескрипторов.

Детектор – извлекает особые точки из изображения. Этот метод обеспечивает постоянство нахождения одних и тех же особых точек относительно измененного изображения.

Дескриптор – идентифицирует особую точку, которая выделяется на фоне остальных особых точек.

# Метод Моравеца

Данный метод работает на основе изменения яркости квадратного окна относительно интересующей точки при сдвиге на 1 пиксель[1].

Работа данного алгоритма:

* Для каждого пикселя с координатами (x,y) вычисляется изменение интенсивности
* Определяется направление, где значение интенсивности наименьшее.
* Отсекаются пиксели, в которых значения оценочной функции меньше порогового значения.
* Удаляются повторяющиеся углы

Недостатком данного метода является отсутствие постоянства при поворотах изображения и возникновение ошибок детектирования при наличии большого количества диагональных ребер[].

# Метод Ростена и Драдмонда

Алгоритм называется Features from Accelerated Segment Test - особенности ускоренных испытаний сегмента[1].

Рассматривается окружность из 16 пикселей вокруг выбранной точки. Точка является угловой, если интенсивность смежных пикселей больше или меньше интенсивности выбранной точки.

Одним из недостатков является падение эффективности работы алгоритма, если вблизи окрестности выбранной точки, может появиться несколько особых точек.

# Выбор метода отслеживания объекта в видеопотоке

В качестве алгоритма отслеживания объекта в видеопотоке в данной работе был выбран алгоритм Лукаса – Канаде. [4]

Данный алгоритм является дифференциальным локальным методом вычисления оптического потока. Его смысл лежит в том, что отслеживается не какой-то фрагмент изображения, а конкретный пиксель (набор пикселей). При этом делается допущение, что значения пикселей переходят из одного кадра в другой без изменений. Такое допущение, на первый взгляд, маловероятно, так как от кадра к кадру глобальное освещение сопровождаемого объекта может сильно поменяться, но на практике получается, что этот алгоритм работает достаточно хорошо.

В данной работе этот алгоритм уместен, так как предполагается, что сопровождаемый объект находится в помещение и на его освещение ничего не влияет.

Так как данный проект является прототипом, начальные эксперименты будут проводиться в помещении с хорошим освещением.

# Конструкторская часть

В данной части будет рассмотрена программная реализация метода стабилизации, метода отслеживания движущегося объекта в видеопотоке. Будет представлена разработка и программная реализация метода оценки скорости движения, объекта в видеопотоке. Будет представлена оценка точности работы реализованных методов в видеопотоке.

# Программная реализация метода стабилизации

В данной части программы реализован корреляционный метод смещения с использованием метода пирамидальной структуризации изображения. Так же написана функция поворота изображения.

# Корреляционный метод смещения

# Расчет корреляции

Функция vectxcorr2 вычисляет ненормированную кросс-корреляцию между двумя двумерными полями скоростей I и J в узком диапазоне смещений относительно заданного начального приближения.

Код функции:

|  |
| --- |
| void vectxcorr2(  Mat& originDerivs, Mat& shiftedDerivs, Point prev\_offset, int tolerance,  Point& curr\_offset, int& max\_cc, Mat& cc  ) |

Параметры, которые функция vectxcorr2 принимает на вход:

* originDervis – производные исходного изображения;
* shiftedDervis – производные сдвинутого изображения;
* prev\_offset – начальное приближение в виде [x0,y0] для смещения, совмещающего изображение shifted с изображением jrigin;

x0 = -originPyr[0].cols … +originPyr[0].cols;

y0 = -originPyr[0].rows … +originPyr[0].rows;

* tolerance – радиус зоны перебора смещений (в пикселях);

Возвращаемые значения:

* cur\_offset – смещение, которое обеспечивает совмещение изображения shifted с origin (определяется по максимуму корреляции cc);
* max\_cc – значение максимума корреляции;
* cc – матрица значений кросс-корреляции ;

# Расчет смещения двух кадров

Функция corrtrack2d оценивает смещение следующего кадра относительно предыдущего по кросс-корреляции матриц градиентов яркости этих кадров в различных разрешениях.

Код функции:

|  |
| --- |
| void corrtrack2d(  vector<Mat>& originPyr, vector<Mat>& shiftedPyr, int tolerance, bool verbous,  Point& offset) |

Параметры функции corrtack2d, которые принимаются на вход:

* originPyr - пирамида исходного изображения и его производных;
* shiftedPyr - пирамида сдвинутого изображения и его производных;
* tolerance - радиус зоны перебора смещений на любом уровне пирамиды;
* verbous - указание распечатывать результаты промежуточных вычислений;

Возвращаемые значения:

* offset - смещение (x, y), которое надо применить к последующему кадру shifted, чтобы совместить его с предыдущим кадром origin;

Заголовок - значение корреляции, которое получилось для смещения offset.

# Построение пирамид изображения

ФункцияbuildOpticalFlowPyramid строит пирамиды изображений.

Код функции:

|  |
| --- |
| buildOpticalFlowPyramid(origin, originPyr, winSize, maxLevel-1, withDerivatives);  buildOpticalFlowPyramid(shifted, shiftedPyr, winSize, maxLevel-1, withDerivatives); |

Параметры функции buildOpticalFlowPyramid, которые принимаются на вход:

* origin – исходное изображение;
* originPyr – пирамида для исходного изображения;
* winSize – размер окна поиска на каждом уровне пирамиды;
* maxLevel – количество уровней в пирамиде;
* shifted – сдвинутое изображение;
* shiftedPyr – пирамида для сдвинутого изображения;

Возвращает количество уровней в пирамиде

# Результат работы методов 2.1.1.1, 2.1.1.2, 2.1.1.3 в совокупности

Рассчитаем смещение двух изображений: Origin рисунок 4, Shifted рисунок 5. Выведем в командной строке значения смещений и корреляции на каждом уровне пирамиды рисунок 6. Красная линия отмечает середину изображения.

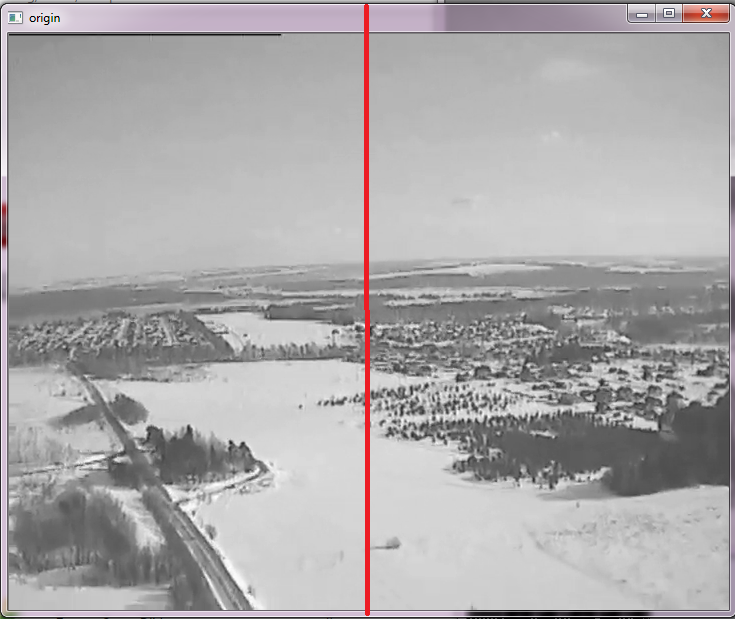


Рис. 4 Исходное изображение Origin

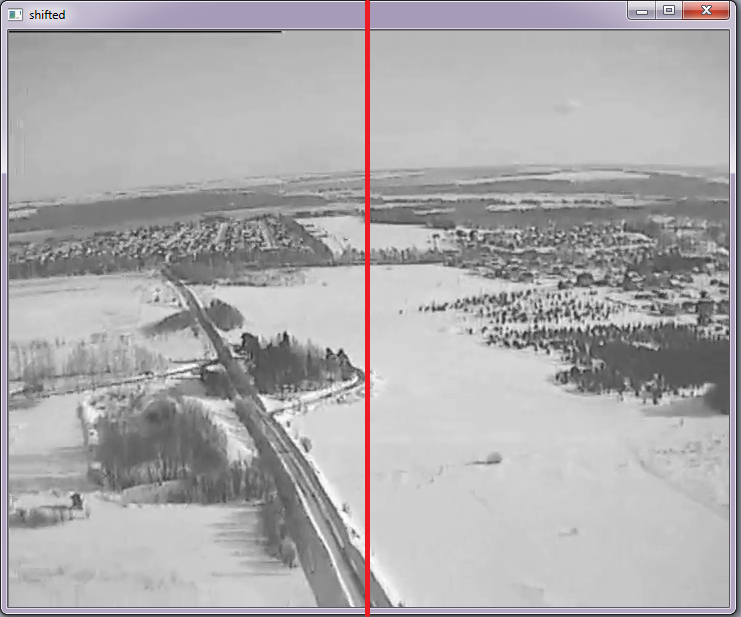


Рис 5. Сдвинутое изображение Shifted

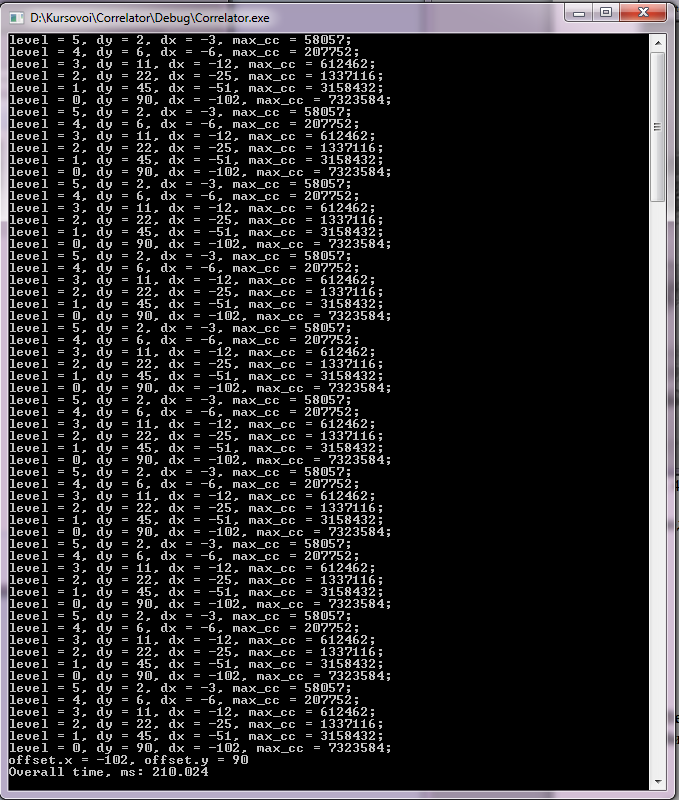


Рис. 6 Результат работы функций

где level – уровень пирамиды,

dy,dx – смещение,

max\_cc – значение корреляции,

offset – значение смещения сдвинутого изображения.

Мы получили значения смещения и корреляции на каждом уровне пирамиды. Используя значения смещений, в дальнейшем будем стабилизировать видеопоток, сдвигая текущие кадры на значения смещений относительно опорного кадра.

# Функция поворота изображения

Функция povorot() поворачивает изображение на заданный угол

Код функции:

|  |
| --- |
| Void povorot(Mat& origin, double f, Mat& rotateOrigin) |

Входные функции povorot(), которые подаются на вход:

* origin – исходное изображение;
* f – угол, на который нужно повернуть изображение;

Возвращаемое значение:

* rotateOrigin – повернутое изображение;

В функции povorot(), мы используем функцию общих аффинных преобразований warpAffine() из библиотеки OpenCV.

# Испытание работы функции поворота

Повернём изображение Origin (рисунок 7) на 45º градусов относительно центра. И выведем повернутое изображение (рисунок 8).



Рис. 7 Исходное изображение Origin

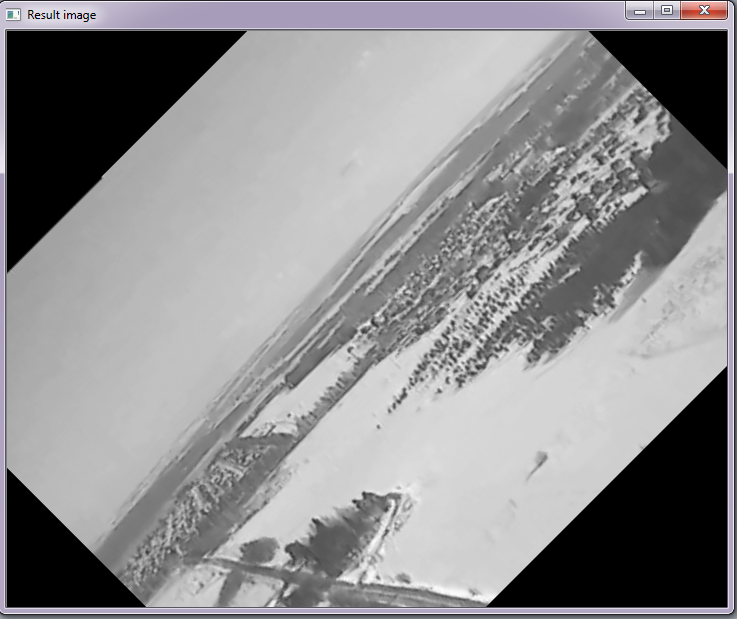


Рис. 8 Повернутое изображение

После аффинных преобразований функция поворот возвращает изображение, повернутое на 45º градусов.

Рассчитав значения смещения повернутых изображений, корреляционным методом функция поворота будет стабилизировать изображение относительно центра.

# Функция сдвига изображения

Функция move() двигает изображение по плоскости XOY.

Код функции:

|  |
| --- |
| void move(Mat& origin, int x, int y, int x1, int y1, Mat& rotateOrigin) |

Входные данные функции move:

* origin – исходное изображение;
* x – значение смещения offset по оси абсцисс ;
* y – значение смещения offset по оси ординат;
* x1 – значение положения точки после смещения по оси абсцисс;
* y1– значение положения точки после смещения по оси ординат;

Возвращаемые данные:

* rotateOrigin – смещенное изображение

Смещение offset мы получаем после использования функции corrtrack2d, которая нам возвращает значение offset.

# Стабилизация видеопотока

В методе стабилизации видеопотока мы используем совокупность функций buildOpticalFlowPyramid, corrtrack2d, move.

Код метода:

|  |
| --- |
| buildOpticalFlowPyramid(gray, shiftedPyr, winSize, maxLevel-1, withDerivatives);  corrtrack2d(originPyr, shiftedPyr, 3, false, offset);  int x = offset.x;  int y = offset.y;  if(make == 1)  {  move(image, x, y, x1, y1,rotateOrigin);  g++;  putText(rotateOrigin, myStr,org,FONT\_HERSHEY\_PLAIN,2.0,CV\_RGB(255,0,0),1,8);  imshow("LK Demo", rotateOrigin);  } else {  putText(image, myStr,org,FONT\_HERSHEY\_PLAIN,2.0,CV\_RGB(255,0,0),1,8);  imshow("LK Demo", image);  }  if(x > 15|| x < -15 || y > 15 || y < -15)  {  gray.copyTo(realImage);  buildOpticalFlowPyramid(realImage, originPyr, winSize, maxLevel-1, withDerivatives);  x1 += x;  y1 += y;  g = 0;  } |

Мы используем функцию пирамид изображения buildOpticalFlowPyramid и функцию расчета смещения corrtrack2d. Функция corrtack2d возвращает значение смещения offset. Мы фиксируем значения offset по оси абсцисс offset.x и оси ординат offset.y. Если мы включили режим стабилизации, значения offset передаются в нашу функцию сдвига изображения move. Если значения offset.x и offset.y превосходят границы нашего ограничения по пикселям (-15,15), мы сохраняем новый опорный кадр и относительно нового опорного кадра считаем следующее смещение.

# Программная реализация метода отслеживания объекта в видеопотоке

В данном разделе представлен программный блок, реализующий метод детектирования Лукаса Канаде

Код блока:

|  |
| --- |
| frame.copyTo(image);  cvtColor(image, gray, COLOR\_BGR2GRAY);  if( nightMode )  image = Scalar::all(0);    if( needToInit )  {  // automatic initialization  goodFeaturesToTrack(gray, points[1], MAX\_COUNT, 0.01, 10, Mat(), 3, 0, 0.04);  cornerSubPix(gray, points[1], subPixWinSize, Size(-1,-1), termcrit);  addRemovePt = false;  }  else if( !points[0].empty() )  {  vector<uchar> status;  vector<float> err;  if(prevGray.empty())  gray.copyTo(prevGray);  calcOpticalFlowPyrLK(prevGray, gray, points[0], points[1], status, err, winSize,  3, termcrit, 0, 0.001);  size\_t i, k;  for( i = k = 0; i < points[1].size(); i++ )  {  if( addRemovePt )  {  if( norm(point - points[1][i]) <= 5 )  {  addRemovePt = false;  continue;  }  }  if( !status[i] )  continue;  points[1][k++] = points[1][i];  circle( image, points[1][i], 3, Scalar(0,255,0), -1, 8);  }  points[1].resize(k);  }  if( addRemovePt && points[1].size() < (size\_t)MAX\_COUNT )  {  vector<Point2f> tmp;  tmp.push\_back(point);  cornerSubPix( gray, tmp, winSize, Size(-1,-1), termcrit);  points[1].push\_back(tmp[0]);  addRemovePt = false;  } |

В данном блоке используются 4 основные функции библиотеки OpenCV:

* cvtColor
* goodFeaturesToTrack
* cornerSubPix
* calcOpticalFlowPyrLK

где, cvtColor – функция, которая преобразует входное изображение из одного цветового пространства в другое. В случае преобразования в цветовое пространство RGB, порядок каналов должен быть определен явно (RGB или BGR);

goodFeaturesToTrack – функция, которая находит наиболее заметные углы на изображении или в указанной области;

cornerSubPix – функция, которая выполняет итерации для нахождения точного местоположения угла;

calcOpticalFlowPyrLK – функция, которая вычисляет оптический поток для набора признаков, используя итеративный метод Лукаса Канаде.

# Разработка и программная реализация метода оценки скорости объекта в видеопотоке

С помощью алгоритмов отслеживания объекта в видеопотоке производится получение информации из видеопотока. Получать ее необходимо в виде конечной точки на изображении, за которой и будет производиться слежка. При изменении положения объекта в пространстве, положение этой точки на изображении будет изменяться, то есть будут меняться ее координаты, измеряемые в пикселях на изображении.

Когда мы получили информацию в виде конечной точки P(x,y) на изображении, мы знаем её положение в пространстве. Когда происходит смещение объекта слежения, координаты точки P(x,y) будут меняться. Мы записываем координаты точки P на каждой итерации (каждый следующий кадр изображения). Когда выполняется n-я итерация, мы считаем смещение точки P(x,y) n-й итерации относительно точки P(x,y) 1-й итерации. Далее после n-й итерации, идёт k-я итерация, и мы считаем смещение S точки P(x,y) k-й итерации относительно точки P(x,y) итерации (n-k).

Учитывая частоту работы камеры – 30 кадров/сек, мы рассчитываем время t затраченное на съемку n-го количество кадров. Получив значение S смещения точки P(x,y) по оси абсцисс мы можем рассчитать скорость V перемещения точки по формуле:

(2)

Данный метод расчета скорости перемещения работает корректно при перемещении объекта на плоскости, относительно оси X и Y. Для полноценного расчета скорости перемещения объекта в видеопотоке, данный метод необходимо дорабатывать, включая в него расчет скорости в глубину по оси Z.

# Оценка скорости движения объекта

Метод фиксирует координаты положения объекта на каждом кадре. Далее рассчитывается смещение положения объекта за n-е количество кадров. Рассчитав значения смещений, производится расчет скорости перемещения объекта, с учетом времени, которое затрачено на перемещение.

Код метода:

|  |
| --- |
| for(;;)  {  kadr++;  kontT[kadr] = (double)getTickCount();  if(kadr == 1 || kadr == 0)  {  for(int i = 0; i < points[1].size(); i++)  {  kontX[kadr] = (int)points[1][i].x;  kontY[kadr] = (int)points[1][i].y;  }  } else if(kadr < 20)  {  for(int i = 0; i < points[1].size(); i++)  {  kontX[kadr] = (int)points[1][i].x;  kontY[kadr] = (int)points[1][i].y;  }  }  if(kadr == 20)  {  for(int i = 0; i < points[1].size(); i++)  {  kontX[kadr] = (int)points[1][i].x;  kontY[kadr] = (int)points[1][i].y;  }  cx = kontX[kadr] - kontX[0];  cy = kontY[kadr] - kontY[0];    flag = 1;  kadr = 0;    }  if(flag == 1)  {  for(int i = 0; i < points[1].size(); i++)  {  kontX[kadr] = (int)points[1][i].x;  kontY[kadr] = (int)points[1][i].y;  }  cx = kontX[kadr] - kontX[kadr+1];  cy = kontY[kadr] - kontY[kadr+1];  ct = (kontT[kadr] - kontT[kadr+1])/getTickFrequency();  }  cxM = (double)abs(cx)/20.15;  vxM = cxM/abs(ct);  sprintf(myStr, "v = %.2f px/sec", vxM);  .  .  } |

Переменная kadr используется для подсчета количества итераций, фиксируем номер обрабатываемого кадра.

В данном блоке программы используются три массива:

* kontT[]
* kontX[]
* kontY[]

где,

в kontT[] с помощью функции, getTickCount из библиотеки OpenCV,

записывается количество тиков затраченных на выполнение одной итерации, т.е. на обработку одного кадра;

в kontX[] и kontY[] записываются значения координат объекта перемещения.

Значения первых 20ти кадров записываются в массивы kontX[] и kontY[], далее, когда происходит обработка 20го кадра, мы считаем смещение за счет разности значений 20го и 1го кадра. После выставляем флаг, что выполнена операция обработки 20ти кадров и обнуляем значение переменной kadr.

Далее, значение координат на каждом следующем кадре мы записываем в начало массива и считаем разность текущего значение кадра(kadr) и значение кадра, который был записан 20 итераций назад

(kadr + 1). Тем самым мы считаем смещение шагом в 20 итераций. Далее мы считаем время, затраченное на обработку 20 кадров.

После чего считаем значение скорости движения объекта за счет отношения смещения ко времени, которое было затрачено на эту операцию. И выводим значение скорости на экран с помощью функции putText(). Далее должна производиться калибровка расчета скорости. С учетом разрешения камеры производится масштабирование за счет значения расстояния до объекта.

# 2.4 Оценка точности работы реализованных методов

В стадии оценки.

Результат работы реализованных методов представлен на рисунке 9.



Рис.9 Оценка скорости движения вагонетки

Используя данные экспериментальной установки, будут произведены расчеты оценки точности работы реализованных методов.

# 3 Технологическая часть

# ЗАКЛЮЧЕНИЕ

Все методы реализованы. Все методы программно реализованы. Проведены тестовые испытания.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методы детектирования [Электронный ресурс] - Режим доступа: https://habrahabr.ru/post/244541/ (дата обращения: 19.05.2017).

2. Обухова Н. А. – Информационно управляющие системы «Обнаружение и сопровождение движущихся объектов методом сопоставления блоков». – СПБГУ, 2004. – 7с.

3. Auto-Directed Video Stabilization [Электронный ресурс] –

Режим доступа:

https://static.googleusercontent.com/media/research.google.com/ru//pubs/archive/37041.pdf (дата обращения: 19.05.2017).

4. Применение метода Лукаса — Канаде для вычисления оптического потока [Электронный ресурс] - Режим доступа: <http://engjournal.ru/articles/1275/1275.pdf> (дата обращения: 19.05.2017).

5. Пирамиды изображений [Электронный ресурс] –

Режим доступа:

http://wiki.technicalvision.ru/index.php/Сопоставление\_с\_использованием\_пирамиды\_изображений (дата обращения: 19.05.2017).

6. OpenCV [Электронный ресурс] - Режим доступа: https://ru.wikipedia.org/wiki/OpenCV (дата обращения: 19.05.2017).

7. Learn OpenCV [Электронный ресурс] - Режим доступа:

https://www.learnopencv.com/warp-one-triangle-to-another-using-opencv-c-python/ (дата обращения: 19.05.2017).

# ПРИЛОЖЕНИЕ А

Графическая часть дипломного проекта

В графическую часть дипломного проекта входят:

1. Актуальность (рисунок А.1)
2. Постановка задачи (рисунок А.2)
3. Обзор аналогов (рисунок А.3)
4. Цели и задачи (рисунок А.4)
5. Результаты экспериментов (рисунок А.5)

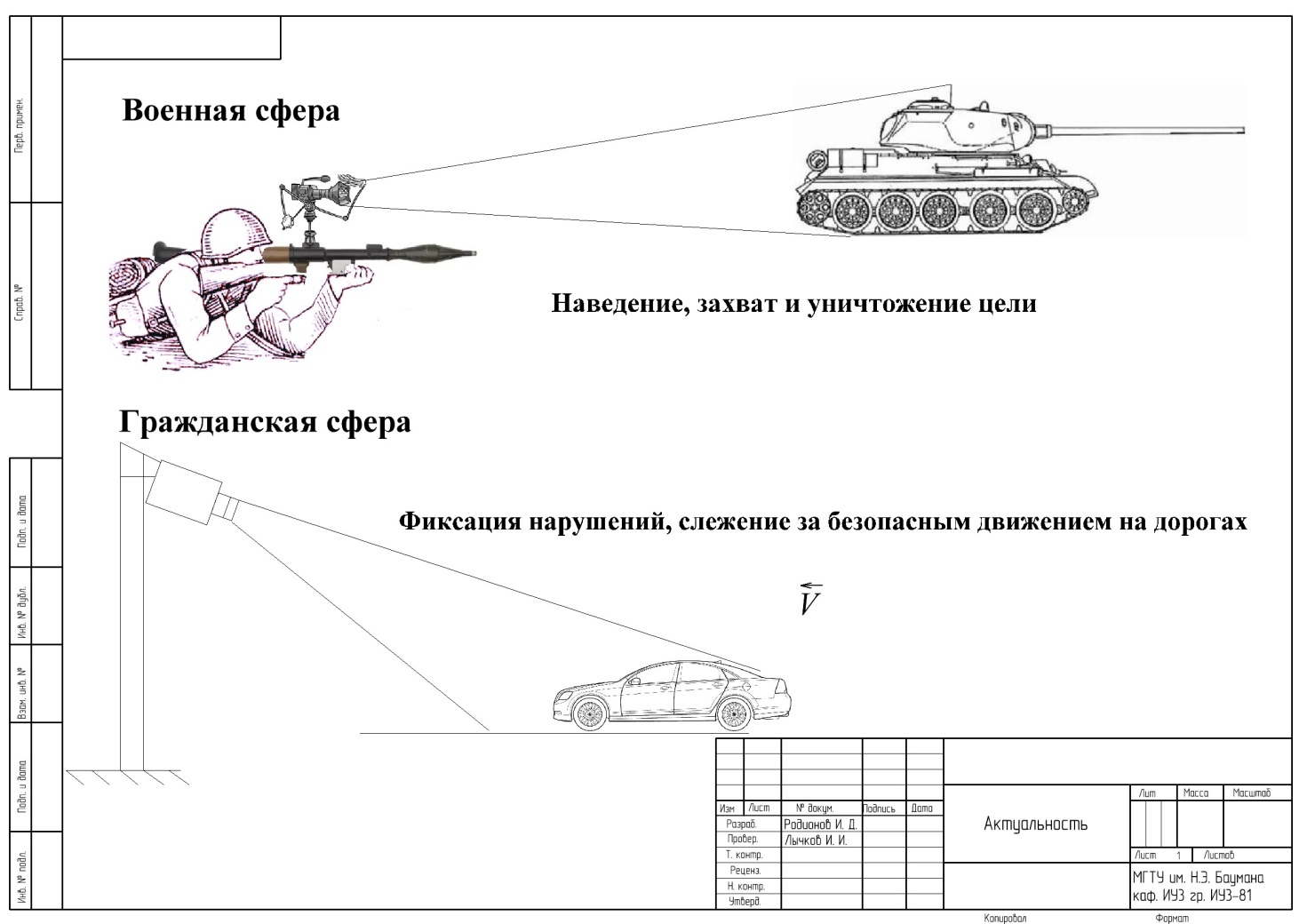


Рисунок А.1 Актуальность

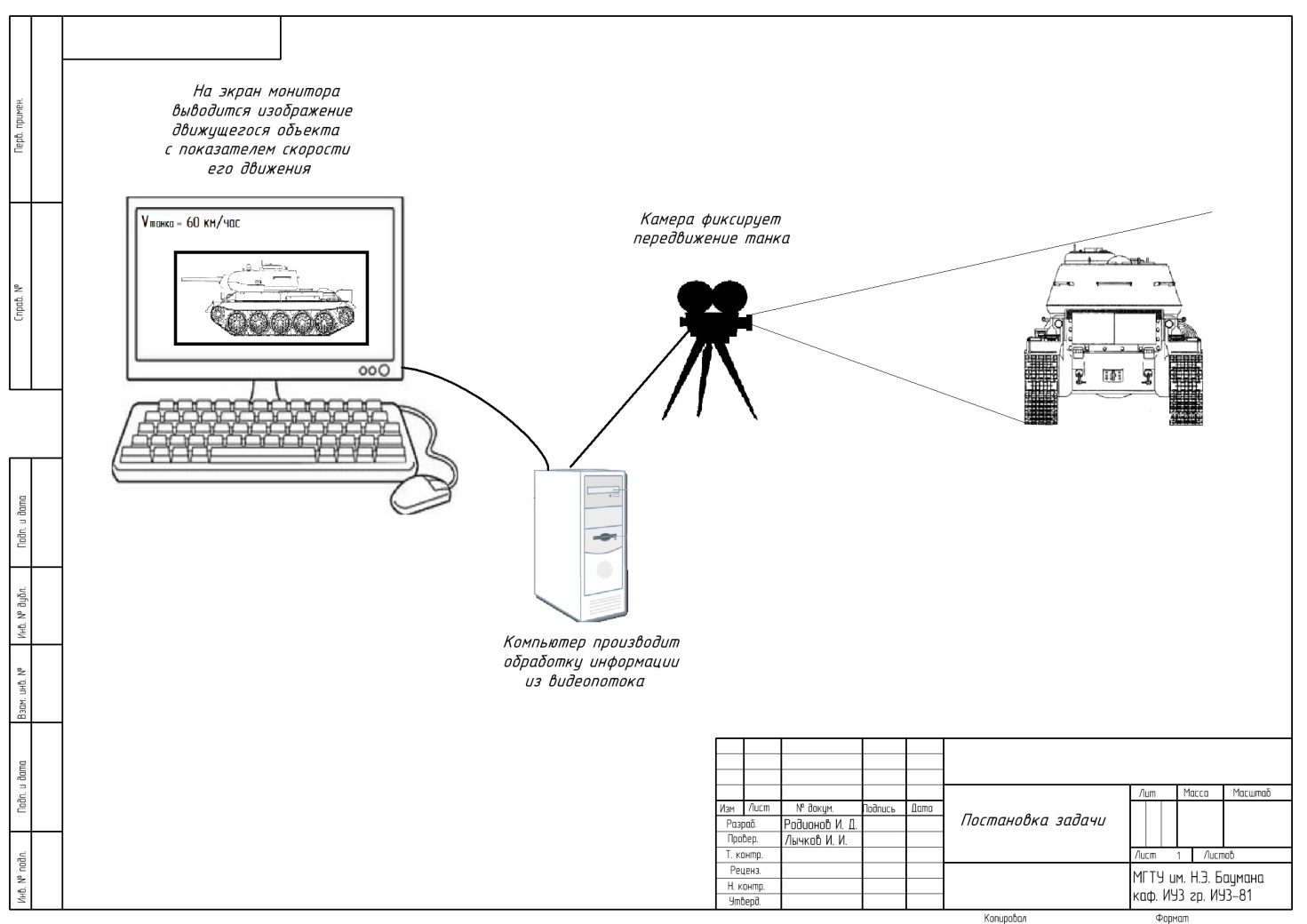


Рисунок А.2 Постановка задачи

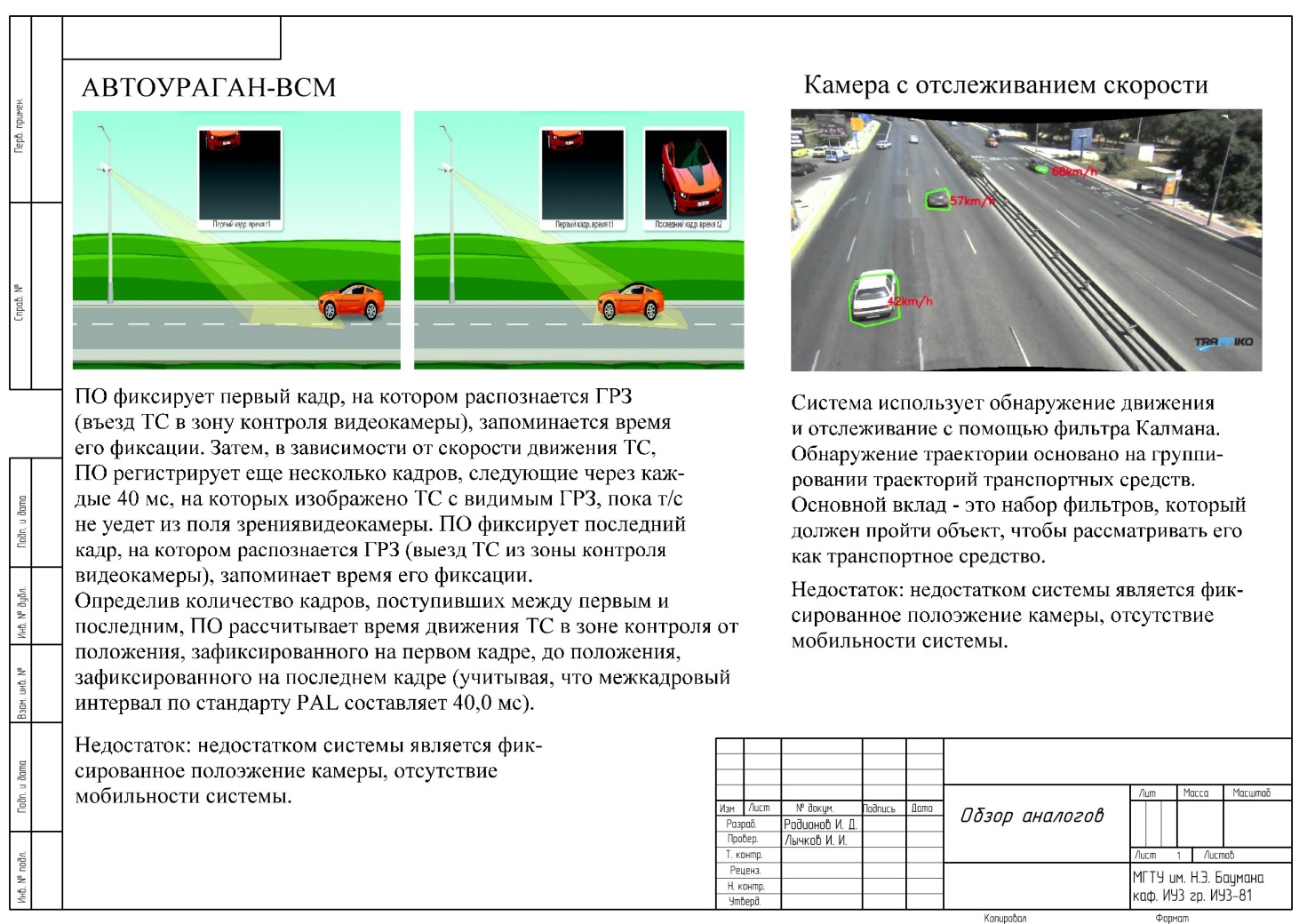


Рисунок А.3 Обзор аналогов

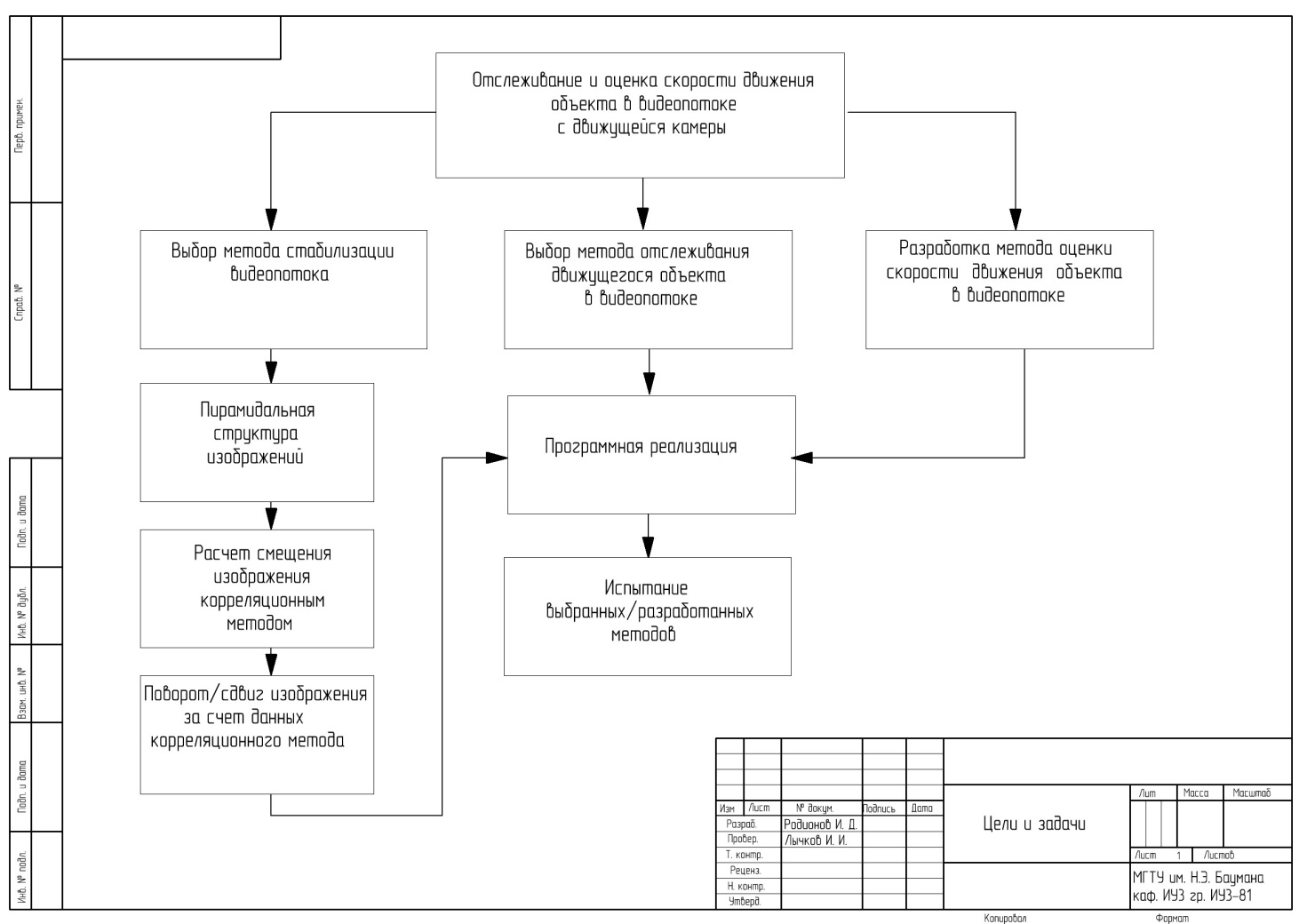


Рисунок А.4 Цели и задачи

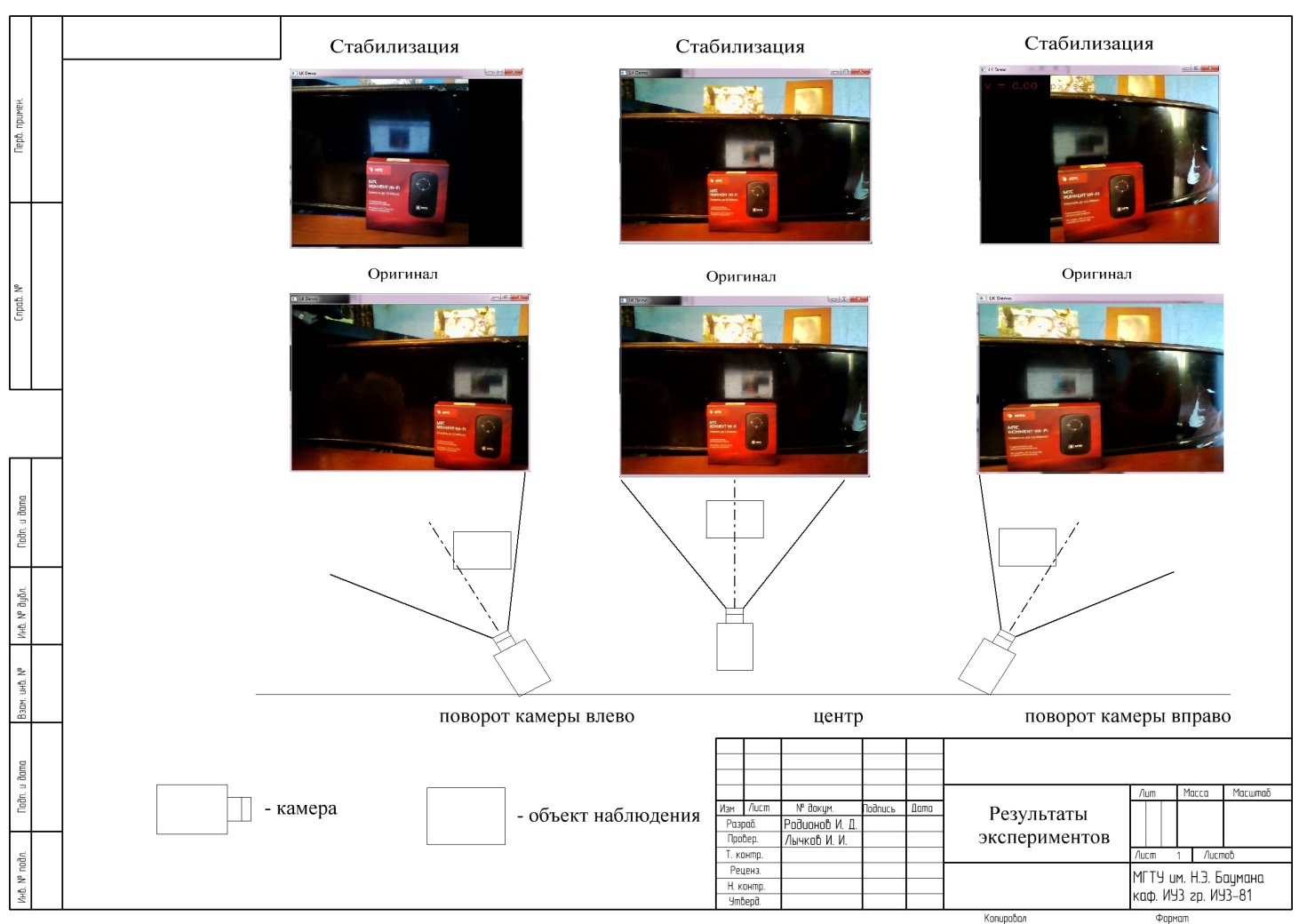


Рисунок А.5 Результаты экспериментов