РОССИЙСКАЯ ФЕДЕРАЦИЯ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ
ИМПЕРАТОРА НИКОЛАЯ II

Институт «Управления и информационных технологий» (ИУИТ)
Кафедра «Автоматизированные системы управления» (АСУ)

Допустить к защите
Заведующий кафедрой АСУ
_____
«_____» _____ 201__ г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

(направление подготовки 09.04.01 «Информатика и вычислительная техника»)

на тему: Система видеоаналитики панорамных изображений

Студент: _____ А.А. Багдасаров
(подпись) Фамилия И.О.

Руководитель магистерской работы: : _____ Е.Я. Соймина
(подпись) Фамилия И.О.

МОСКВА 2017

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ

Институт _____ Кафедра _____
Направление_____

УТВЕРЖДАЮ

Зав. Кафедрой АСУ_____
«____»_____20____

З а д а н и е

На магистерскую диссертацию (работу) студента

Багдасарова Александра Антоновича
(фамилия, имя, отчество)

1. Тема диссертации (работы) Система видеоаналитики панорамных изображений
   утверждена приказом по университету от «___»_____20____г. № _____
2. Срок сдачи студентом законченной работы_____19.06.2017 г._____
3. Исходные данные к работе Описание существующих системы видео аналитики 2) Требования к проектируемой системе видео аналитики 3) Описание существующих методов создания панорамных изображений
4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)
   Введение
   1) Описание системы видео аналитики 2) Разработка системы видео аналитики панорамных изображений 3) Оценка эффективности разработанного решения
   Заключение
5. Перечень графического материала (с точным указанием обязательных чертежей)_____
   а) Обзор предметной области
   б) Описание процесса видео анализа
   в) Описание недостатков существующей системы
   г) Описание разработанной системы видео аналитики панорамных изображений
   е) Демонстрация работы разработанной системы виде аналитики
   ж) Обоснование целесообразности внедрения разработанной системы видео аналитики панорамных изображений

6. Консультации по работе (с указанием относящихся к ним разделов проекта)

Раздел	Консультант	Подпись, дата	
		задние выдал	задание принял

7. Дата выдачи задания_____14.01.2017 г._____

Руководитель_____
(подпись)

Задание принял к исполнению_____
(подпись)

АННОТАЦИЯ

В данной магистерской диссертации рассматривается задача создания системы видеоаналитики панорамных изображений.

В первой главе представлен анализ существующих систем видеоаналитики и выявление их недостатков. Рассматривается современный алгоритм видео анализа и создания панорамных изображений из нескольких изображений. Выдвинуты требования к разрабатываемой системе.

Во второй главе описан сам процесс разработки информационной системы, описаны результаты работы алгоритма на языках Java и Python. Обосновано использование системы, написанной на языке Python.

В третьей главе осуществлен расчет необходимой оперативной памяти для обработки и хранения ключевых точек изображения и расчет необходимой общей зоны для создания панорамного изображения.

В приложении приведены: исходный код программы и результаты работы.

ANNOTATION

This Master's thesis overviews the task of the video analytics panoramic images creation.

The first chapter presents the existing video analytics systems analysis and reveals their lacks. The actual video analysis and the creation of panoramic views by several frames is researched in this part. The requirements to the developing systems are raised.

The second chapter describes the process of the informational system development, reports the results of algorithm operation in Java and Python languages. The system in Python language is proved.

The third chapter settles the necessary RAM for the processing and storage of the images key points and the necessary common area for the panoramic view creation.

The source code and the processing results are discovered in the application.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир охарактеризован стремительным развитием информационно-аналитических технологий. Одной из них является видеоаналитика.

Современные системы видео-аналитики автоматизируют четыре основных функции: обнаружение, слежение, распознавание, прогнозирование. Каждый год становится все больше компаний с их решениями в этой области. Существуют системы, работающие с архивами видео потоков и с видеопотоками, поступающими в реальном времени.

На данный момент у существующих систем есть недостатки при отождествлении одного объекта при детектировании его несколькими камерами одновременно. Поэтому в работе разрабатывается новый алгоритм распознавания образов для решения существующих недостатков в области видеоанализа для улучшения эффективности работы систем видео аналитики.

Решение недостатков в области панорамного распознавания образов очень актуально в области безопасности, маркетинга, дорожного движения. одним из ключевых звеньев в банковском деле. Замена интеллектуального труда человека машинным, обоснованное распределение функций между человеком и компьютером в процессе управления технологией приводит к повышению эффективности и качества принимаемых решений.

Целью магистерской диссертации является создание системы видеоанализа цилиндрических панорамных изображений, созданных из одновременных фреймов из видеопотоков, поступающих в систему с

нескольких камер слежения, для дальнейшего детектирования людей в этой итоговой цилиндрической видеозаписи.

Для достижения этой цели необходимо решить следующие задачи:

- оценить возможности известных алгоритмов анализа и обработки изображений

- разработать алгоритм сшивания цилиндрического изображения

- оптимизировать этот алгоритм для сокращения времени

- создание модулей определения движения

- разработать систему видео-аналитики панорамных изображений

- разрабатываемая система должна создавать панорамное изображение меньше чем за 80 мс при разрешении кадров 480x272

В первой главе представлен анализ систем видео аналитики (их назначение, функции). Обосновывается актуальность создания системы панорамного видеоанализа. Обозреваются существующие проблем и подходы к их решению. Рассматриваются алгоритмы сшивания панорамного изображения. В конце главы формулируется цель и задачи магистерской работы.

Во второй главе описывается непосредственно разрабатываемая система. Обосновывается выбор среды разработки и языка, на котором система будет написана.

В третей главе описываются результаты и производится анализ необходимой общей зоны на соседних кадрах для создания панорамного изображения. Делаются выводы и оценивается перспективность разработанной системы видео аналитики.

Научная новизна заключается в оценки быстродействия работы алгоритма создания панорамного изображения с помощью модулей использующих различные методы определения ключевых точек и написанных на разных языках программирования.

Практическая значимость результатов заключается в оценке ресурсоемкости ключевых точек в отношении оперативной памяти и необходимой общей зоны на соседних кадров при формировании панорамного изображения.

ГЛАВА 1. ОПИСАНИЕ СИСТЕМЫ ВИДЕОАНАЛИТИКИ

1.1. Характеристика систем видеоаналитики

1.1.1. Краткие сведения о системах видеоаналитики

Видеоаналитика - технология, которая использует различные методы компьютерного зрения для автоматизированного получения различных данных на основании анализа потока изображений, поступающих с видеокамер в режиме реального времени или из архивных записей. Видеоаналитика представляет собой программное обеспечение (ПО) для работы с видео контентом. В основе программного обеспечения лежит комплекс алгоритмов машинного зрения, позволяющих вести видео-мониторинг и производить анализ данных без прямого участия человека.

Наблюдение с помощью видеокамер зарождалось как замкнутая охранная система, предназначенная только для решения вопросов безопасности. Ограничения аналогового видеонаблюдения не позволяли использовать оборудование как-то иначе. Интеграция видеонаблюдения с цифровыми системами открыла возможность автоматизировано получать различные данные, анализируя последовательность изображений.

1.1.2. Функции систем видеоаналитики

Современные системы видеоаналитики могут выполнять следующие функции:

- Object detection (обнаружение объектов). Обнаружение объектов является основной функцией систем видеоанализа. Существующие системы способны

распознавать множество объектов в поле зрения камеры, в отличии систем, использующих инфракрасные датчики. Пользователь видео аналитики увидит картинку с выделенным объектом.

Так же возможно обнаружение объектов по заданным фильтрам, например, обнаружение только людей, машин или оставленных сумок в заданных областях.

- Object classification (классификация объектов). Функция, которая позволяет фильтровать поток сообщений о распознанных объектах. Используя различные классификаторы, например, каскад Хаара, объекты можно разделить на животных, одного или группу людей, различные транспортные средства. Так же возможно определение людей в касках или без. Возможно использование систем видео аналитики в области продаж для определения возраста или пола человека.
- Object Tracking (Определение пути объекта). Данная функция позволяет отслеживать движение объекта. Некоторые системы, благодаря PTZ-камерам, позволяют следить за объектами в пределах области возможного поворота камеры. Пользователь системы увидит выделенный объект и траекторию движения объекта. В зависимости от возможностей системы и алгоритмов, заложенных в нее, она может одновременно обрабатывать движение нескольких объектов. Использование данной функции используется для анализа движения покупателей в торговых центрах, определения скорости движения и направления движения объекта. Так же данная функция позволяет исключить ложные срабатывания на один и тот же объект.
- Object identification (Идентификация объекта). Функция идентификации объектов является наиболее сложным компонентом систем видео-аналитики.

  Современные системы видео аналитики используют базу данных сохранненых паттернов для дальнейшего сравнения с будущими

сообщениями. Возможно использования видеоанализа для определения человека по походке. Так же возможно соотношение номерного знака транспортного средства и водителя с лицом настоящего владельца. Самым сложным является пример интеграции системы видео аналитики с возможностью идентификации объекта с глобальной системой интернета вещей. Данная функция находит свое широкое применение в области безопасности, где необходимо распознавать лица.

- Обнаружение (распознавание) ситуаций. Современные системы видео аналитики реализуют функцию распознавания ситуаций с помощью определения запретных/контрольных зон или линий, а также случаи пересечения границ в одном или в двух направлениях. Это позволяет определять случаи пересечения объектом запретной территории, движения автомобиля не в том направлении, падения людей или предметов с платформы.
- Предсказание ситуаций или поведения объекта. Данная функция используется в мощных системах видеоаналитики, которые основываются на гибком подходе к распознаванию. Одним из примеров является предсказание образования пробки на перекрестке, если столкнулось два или более транспортных средства
- Интеллектуальное сжатие видео потока с учетом интереса пользователя (например, в системе сохраняются только видео фрагменты и сообщения, содержащие сообщения о нарушениях или тревожных ситуациях)
- Ранжирование событий видео-аналитики основываясь на приоритете.

В современных системах видео-аналитики все чаще находят свое применение pan-tilt-zoom-камеры (PTZ-камеры), которые способны поддерживать изменение направления фокусировки и зумирования, но их применение не раскрывает полного функционала камер в виду того, что системы видео-аналитики «теряются» при смене угла обзора камеры.

1.1.2. Обоснование необходимости создания системы видеоанализа панорамных изображений

Использование видеоаналитики дает возможность в автоматическом режиме, без участия человека, в процессе видеонаблюдения решать задачи, которые обычно под силу только человеческому зрению. Данная технология используется как для обеспечения безопасности, так и для повышения эффективности бизнеса в торговле, финансовом секторе и на транспорте. Важность трудно переоценить: в обычном случае после 12 минут непрерывного наблюдения оператор начинает пропускать до 45% событий. И до 95% потенциально тревожных событий будет пропущено уже после 22 минут непрерывного наблюдения (по результатам исследования IMS Research, 2002).

Но несмотря на широкое применение современных систем видеонаблюдения и видеоанализа, существуют ряд нерешенных проблем. Одной из проблем существующих систем видео-аналитики является статичный угол обзора камер. Это ведет к тому, что для определения объекта, частично попавшего в угол обзора камеры, могут возникнуть трудности. Одним из способов решения этой проблемы может быть использования нескольких камер для охвата большей зоны и, используя фреймы за одинаковые единицы времени, создавать панорамное изображение и использовать его в видео аналитике. Так же есть другой вариант, который частично решается с помощью PTZ – камер. Но современные системы видео-аналитики (например, IBM Intelligent Video Analytics) используют эти камеры со следующим алгоритмом – для камеры заранее создается несколько «домашних» позиций, в которые она перемещается через некоторые условные промежутки времени. Но при этом для распознавания определенной «домашней» позиции зачастую используется API

конкретной системы управления видео (VMS) так как визуальное распознавание находится на зачаточном уровне. Так же использование поворотных камер исключает возможность использования распознавания ситуаций. В современных системах видео-аналитики для распознаваний ситуаций пользователем системы задается одна или несколько линий, которые образуют границу, через которую объекты не имеют право переступать в одну или обе стороны. При возникновении случаев пересечения границы объектом, пользователю системы видео-аналитики приходит сообщения о нарушении. Визуализация представлена на рисунке 1.

Рис. 1. Визуализация распознавания ситуаций нарушения границы

Тонкость данного алгоритма заключается в том, что эти границы привязаны к пикселям, а не к реальным объектам, попавшим в поле зрения объектива камеры. Таким образом, при повороте камеры настройки границы не будут соответствовать реальным. Пример данного дефекта проиллюстрирован на рисунке 2.

Рис. 2. Визуализация дефекта анализа с использованием поворотной камеры после поворота угла обзора на несколько градусов вправо

Другой проблемой современных систем видео-аналитики является идентифицированные одного объекта в поле зрения разных камер как отдельные. Это осложняет определение трека движения объекта. Пример визуализации определения трека движения объектов представлен на рисунке 3.

Рис. 3. Трекинг объектов

Решить данную проблему можно используя PTZ - камеру, чтобы следить за определенным объектом, но в таком случае остальная область будет вне поле зрения и возрастает вероятность упустить другие объекты.

Другим способом является использование нескольких камер и создание на основе видеопотоков с них панорамного изображения.

1.1.3. Алгоритм анализа изображения

Современные системы видео аналитики работают с видео потоком в реальном времени или из архива. И в том, и в другом случае это последовательность кадров, которые передаются с некоторой частотой в секунду, который называется fps (frames per second). Каждый фрейм является статичной картинкой, которая несет информацию. В зависимости от формата (количеством бит на пиксель и их интерпретацией: какие биты за какую составляющую цвета отвечают) и разрешения каждый фрейм имеет определенный объем. Каждый фрейм, в зависимости от настроек системы, может быть цветным или черно-белым, сжатым или нет. Так же из-за необходимости хранения сообщений о нарушениях и сравнения кадров между собой системам видео аналитики необходим параметр, который был бы уникальным ключом в этой ситуации. Этим ключом является timestamp – уникальная временная метка каждого кадра.

Таким образом, видео поток всегда имеет следующие основные характеристики:

- Частота кадров в секунду
- Формат
- Разрешение
- Timestamp

Сам процесс видеоанализа состоит из нескольких этапов. На каждом из них кадры из видеопотока обрабатываются согласно правилам и передаются следующему звену алгоритма. Иногда, для улучшения итогового результата видеоанализа используются обратные связи между этапами. Схема представлена на рисунке 4.

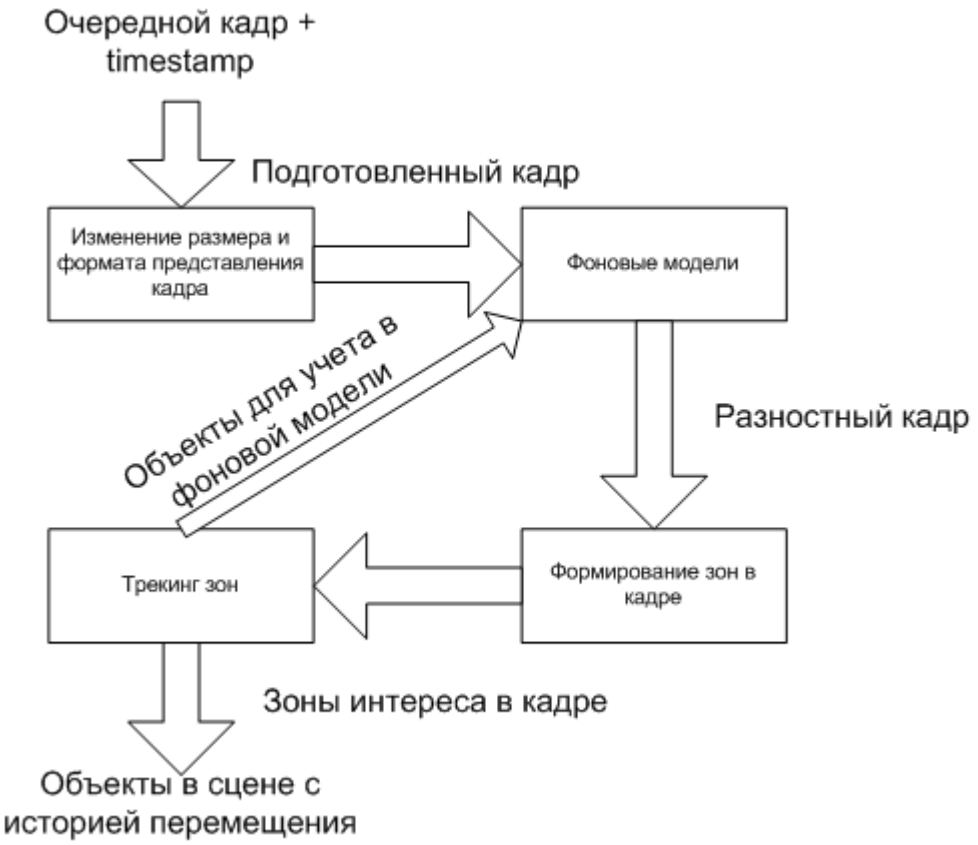

Рис. 4. Схема работы анализа видеопотока

1.1.3.1 Форматирование фрейма

Первый этап подготавливает каждый фрейм из видеопотока для дальнейшей обработки в алгоритме системы видеоанализа. Из предыдущей части было выяснено, что каждый фрейм обладает разрешением и форматом. Чем больше разрешение, тем больше пикселей, которые несут информацию и обладают определенным весом. Современные системы видео аналитики редко работают с одной камерой, так что для сокращения вычислительных мощностей и канало связи между камерами и сервером видеоанализа необходимо сократить размер каждого кадра в виду того, что в дальнейшем, каждый пиксель каждого кадра будет обрабатываться. При сокращении разрешения кадров теряется часть информации, но так как системы видеоанализа работают с крупными объектами, на выходе теряется лишняя информация и сокращается количество

различных артефактов, таких как «шум», которые связаны с освещением сцены, природными факторами и разрешающей способностью камеры.

В цветном изображении, обладающим форматом RGB24 каждый пиксель обладает весом в 3 байта. Сохранить информативность, при этом сократив объем можно с помощью перевода картинки из цветного в формат серого изображения Y8, в котором для каждого пикселя используется 1 байт информации. Таким образом объем видеопотока можно сократить в три раза. Пример преобразования кадра представлен на рисунке 5.

Так же можно сократить количество кадров в секунду в видеопотоке, что так же благополучно скажется на объеме обрабатываемой информации и скорости видеоанализа. Современные системы видео аналитики способны работать с видео, обладающим частотой 12 кадров в секунду [1].

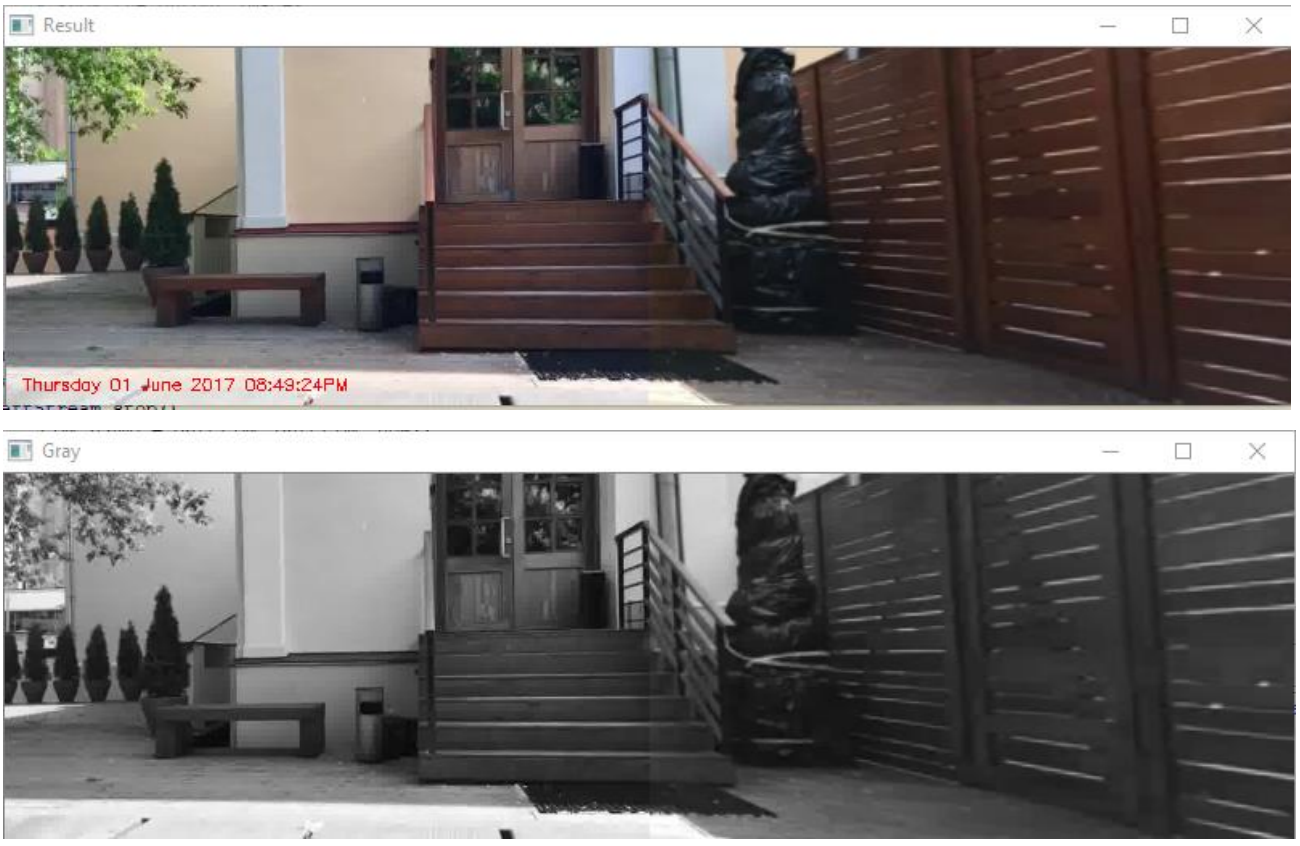

Рис. 5. Цветовое преобразование изображения

1.1.3.2 Создание фона

Для детектирования движения, распознавания объектов, создания трека движения необходимо создать фон, с помощью которого можно определить инварианты и меняющиеся пиксели в кадрах. Достигается это с помощью покадрового сравнения с шаблоном. С течением времени новые фреймы будут добавляться к текущей фоновой модели, но сперва ее нужно создать. В зависимости от алгоритма создания фона, мы будем иметь разный результат видеоанализа потока изображений. Самым простым примером является создание фоновой модели, построенной на пустой сцене, как на рисунке 6.

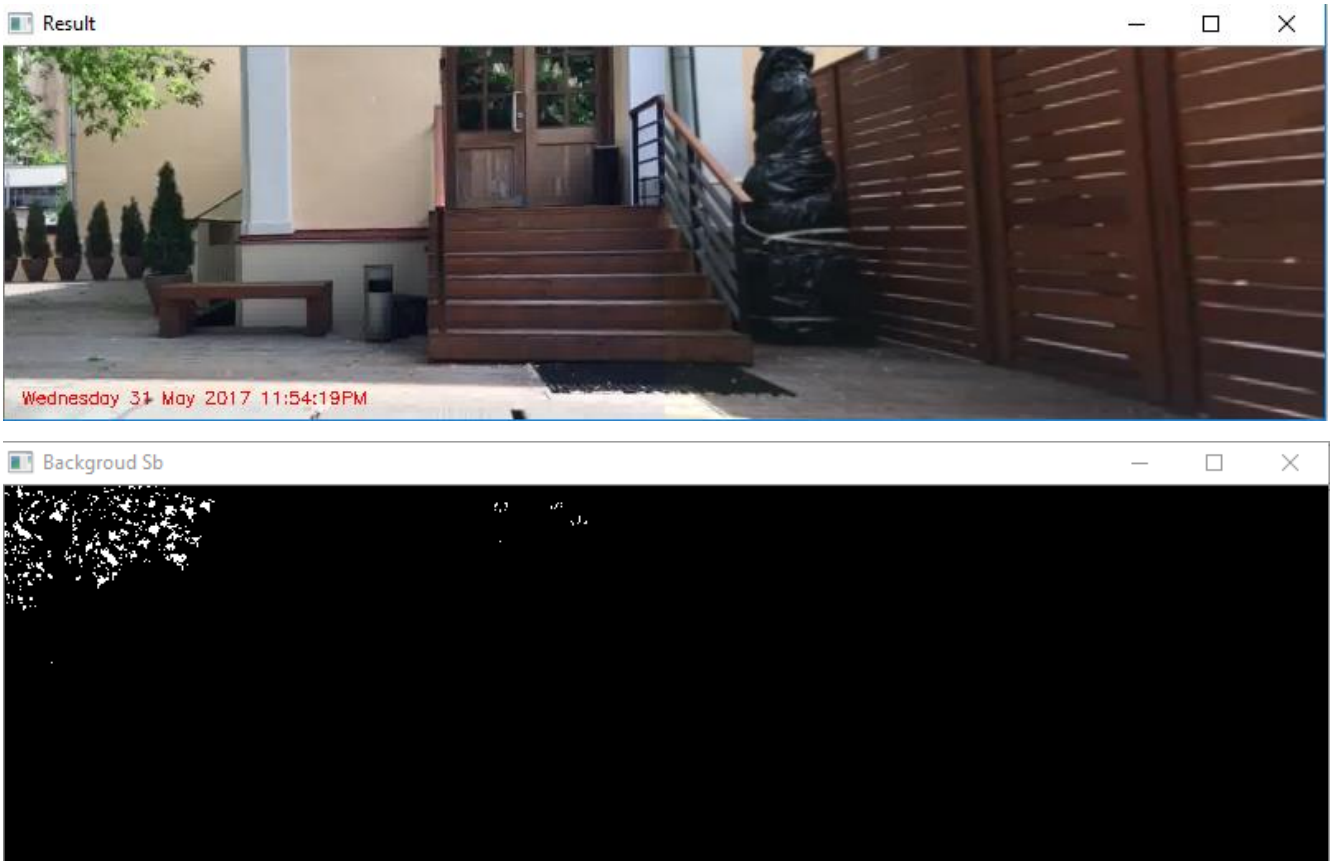

Рис. 6. Фоновая модель пустой сцены

Для сравнения, возьмем теперь следующие кадры из видео потока, где будет определен объект (рисунок 7). Используя разницу в кадрах, мы видим объект и в итоге, он выделяется рамкой для пользователя.

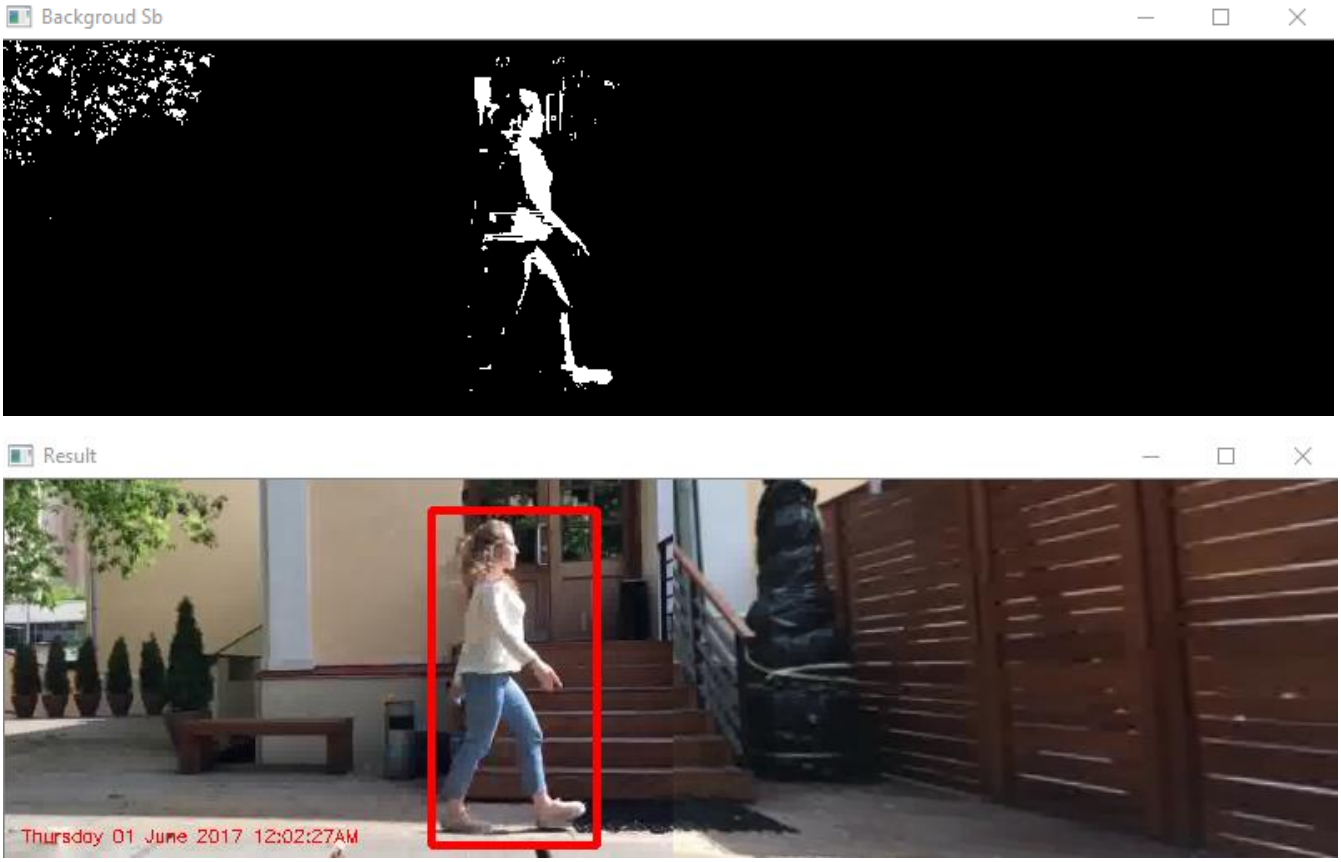

Рис. 7. Фоновая модель сцены с объектом

В виду того, что в системе идет сравнение с изображением, обладающим форматом Y8, то идет попиксельное сравнение с фоном фреймом в градациях серого. Для сокращения «шумов» и увеличения быстродействия процесса происходит бинаризация пикселей – все пиксели, большие 0 изменяют свое значение на 255, что в итоге ведет к образованию черного белого изображения, как на рисунке 8.

Рис. 8. Бинаризация разностного кадра

После этого шага имеется отделенный фон и объект с четкими границами, но в некоторых случаях, так же возможна «присоединение» к объекту его тени, а также видны результат зафиксированного движения веток дерева.

Данные артефакты негативно сказываются на результате работы системы видео аналитики. При сильном ветре, изменение освещения или бликах в отражающих поверхностях возможны появления ложных срабатываний системы, что в рамках сферы безопасности может негативно сказать на эффективности принятия решений. Таким образом на данном шаге необходимо классифицировать объекты, чтобы проигнорировать природные факторы, воздействие на камеру или шумы изображения: блики, попадание пыли на объектив камеры, артефакты, появляющиеся при сжатии кадров и т.д. С другой стороны, система должна уметь запоминать оставленные объекты и не добавлять их в фон с течением времени.

В зависимости от используемых мощностей, можно использовать различные алгоритмы. Многие из них используется в различных библиотеках и их можно комбинировать, для улучшения результата. Но следует использовать данные алгоритмы не во вред общей производительности системы. При использования видео потока с частотой кадров 20 фреймов в секунду у системы есть не больше 50 мс, на обработку кадра и применение алгоритма видео анализа.

1.1.3.3. Формирование зон

После того, как разностный кадр сформирован, на нем видны белые объекты на черном фоне, как например на рисунке 8. С помощью различных алгоритмов белые пиксели объединяются в зоны, которые потом будут видны пользователю системы, как прямоугольники или другие геометрические фигуры, которые выделяют объект. На данном этапе возможно разделение одного объекта на части или же наоборот – объединение двух разных объектов в один (такое возможно в случае, если один объект на момент появления в сцене перекрыл другой).

Избавиться от многих артефактов можно используя метаданные, которые несут в себе информацию о размерах и пути движении объекта.

1.1.3.4. Трекинг зон

На данном этапе происходит соотнесение результатов работы обработки каждого фрейма. Для соотнесения одного объекта между собой на разных кадрах возможно использование различных характеристик, хранящихся в метаданных текущего и предыдущего фреймов. При этом самой важной характеристикой является таймстамп кадра, с помощью которого можно формировать цикл. С помощью результатов на данном шаге возможно

построение аналитики скорости объекта и анализ истории движения в заданной области (рисунок 9).

Рис.9. Трекинг объекта

Как и на предыдущем шаге возможно возникновение ситуации, когда один объект перекрывает другой, но благодаря трекингу движения и истории этих движений, можно разделить эти объекты и воспринимать их как разные.

1.1.3.5. Объекты для учета в фоновой модели

В архитектуре могут присутствовать обратные связи, улучшающие работу предыдущих этапов. Пожалуй, самое главное решение — использовать информацию об объектах в сцене при формировании фона, например, так можно отличить отставленные предметы и не делать его частью фона. Или бороться с обратной ситуацией: если при создании фона на сцене был человек то, когда он уйдет, на его месте появится объект-“призрак”. Понимая, что в этом месте начинается траектория движения объекта, можно быстро убрать “призрака” в фон.

В результате мы получаем метаданные по объекту. Каждый из них характеризуется размерами, плотностью, скоростью, траекторией, направлением движения и другими параметрами.

Эти метаданные и используются при аналитике сцены. Можно определить пересечение объектом линии или движение в неправильном направлении. Можно подсчитать количество объектов в заданной зоне, праздное шатание, падение и множество других событий. будет список объектов в сцене.

1.1.4. Алгоритм формирования панорамного изображения

Самым главным моментом в планируемой системе является попарное сравнение одновременных фреймов с камер, для дальнейшего сшивания их панораму и построению на основе этих цилиндрических кадров видеоанализа потока изображений. В целом алгоритм состоит из следующих шагов.

1.1.4.1. Определение особых точек соседних изображений

На первом шаге происходит определение инвариантных характеристик соседних изображений. Это достигается с помощью выявления особых точек и их дескрипторов. Особая точка m – точка, которая является изображением окрестности, которой *o(m)* можно отличить от окрестности любой другой точки изображения *o(n)* в некоторой другой окрестности особой точки *o2(m)* [2]. Дескриптор является идентификатором особой точки, выделяющий её из множества особых точек.

Для обнаружения особых точек существует несколько алгоритмов, но в данной работе обратим внимание на два из них, с которыми в дальнейшем будем работать в разрабатываемой системе:

- Speeded Up Robust Features (SURF)
- Scale Invariant Feature Transform (SIFT)

Рассмотрим каждый из этих алгоритмов по отдельности.

1.1.4.1.1. SIFT

Алгоритм SIFT позволяет определять особые точки в виде капель (структуры, описываемые координатами центра, масштабом и направлением), так как они инвариантны ко всем преобразованиям (к аффинным преобразованиям, изменениям освещённости, положению камеры и к шуму, к аффинным преобразованиям,). Структуры такого вида являются самыми сложными, высокоуровневыми среди всех видов форм особых точек, и поэтому обеспечивают устойчивое их обнаружение. Алгоритм *SIFT* показывает лучшие результаты обнаружения особых точек по качеству чёткости и контрастности изображения, чем другие алгоритмы обнаружения капель, и подходит к предметной области, Одним недостатком *SIFT* является его вычислительная сложность, что ограничивает его применение в режиме постобработки.

Алгоритм *SIFT* состоит из 5 шагов:

- построение пирамиды гауссиан и их разностей. На этом шаге обеспечивается инвариантность к масштабированию;
- определение экстремумов;
- уточнение особых точек;
- определение ориентаций особых точек (обеспечивается инвариантность к повороту);
- построение дескрипторов (обеспечивается инвариантность к освещению, шуму, изменению положения камеры).

На первом шаге алгоритма *SIFT* [6, 7] строится масштабируемое пространство изображения – набор изображений, сглаженных фильтром Гаусса $G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$, где (x,y) – координаты точки, σ – радиус размытия.

По эти точкам строится разность гауссиан *D(x,y,σ)*, которое является попиксельным вычитанием изображений в одной октаве с разным коэффициентом размытия. Октава – изображение в одном масштабе, размытое фильтром Гаусса (4 изображения в одной октаве). На этом шаге обеспечивается инвариантность к масштабированию. Затем определяются экстремумы, которые заносятся в список потенциальных особых точек.

Далее происходит уточнение особых точек, которое состоит из двух составляющих:

1) исключаются точки с малой контрастностью с помощью вычисления экстремума разности гауссиан. Разность гауссиан раскладывается многочленом Тейлора второго порядка, взятого в точке вычисленного экстремума;

2) исключаются граничные точки (точки, имеющие большой локальный изгиб вдоль границы и малый в перпендикулярном направлении).

На конечном шаге алгоритма *SIFT* для окрестности особой точки вычисляются изменения яркостей точек, по которым строится дескриптор. Дескриптор – это вектор из 64 чисел, позволяет получить инвариантность относительно положения камеры. Затем дескриптор нормализуется, за счёт чего достигается инвариантность относительно изменения освещения. Пример нахождения общих точек с помощью дескриптора SIFT представлен на рисунке 10.

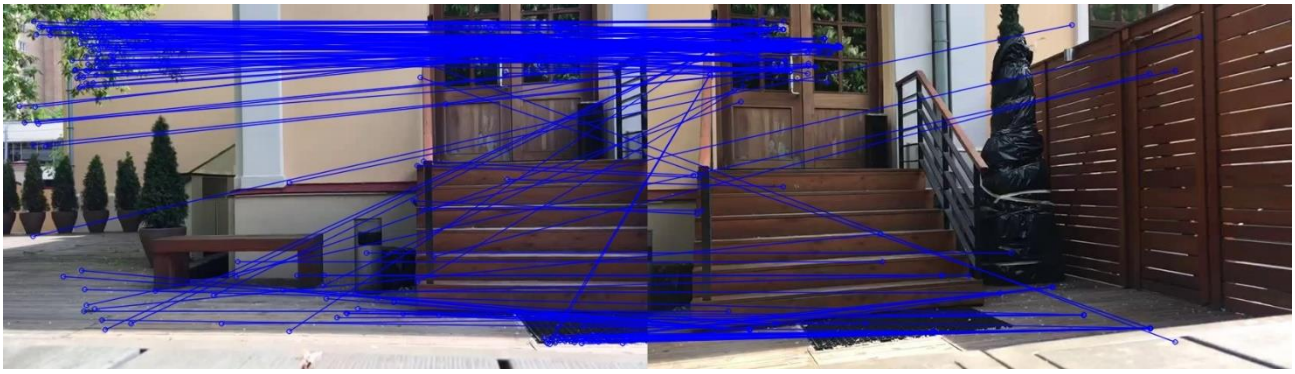

Рис. 10 Общие точки. Метод SIFT

1.1.4.1.2. SURF

Метод Speeded Up Robust Features (SURF) положительно зарекомендовал себя в задачах поиска объектов на изображениях, 3D реконструкции, при сравнении изображений. Рассмотрим применение этого метода в биометрических системах, осуществляющих аутентификацию по васкулярному рисунку руки.

Метод SURF решает две задачи – поиск особых точек изображения и создание их дескрипторов (описательного элемента, инвариантного к изменению масштаба и поворота). Кроме того, сам поиск ключевых точек тоже должен обладать инвариантностью, т.е. повернутый объект сцены должен обладать тем же набором ключевых точек, что и образец.

Данный метод ищет особые точки с помощью матрицы Гессе. Детерминант матрицы Гессе (т.н. гессиан) достигает экстремума в точках максимального изменения градиента яркости. Для двумерной функции ее детерминант определяется следующим образом:

$$H\big(f(x,y)\big) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial x \partial y} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$\det(H) = \frac{\partial^2 f}{\partial x^2}\frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2$$

Где H – гессиан, а f (x,y) — функция изменения градиента яркости.

Матрица Гессе инвариантна к повороту кадра, но не инвариантен к масштабу. В связи с этим в методе SURF используются разномасштабные филтры для нахождения гессианов. Для каждой ключевой точки считается

градиент и масштаб. Градиент в точке вычисляется с помощью фильтров Хаара. Размер фильтра берется равным 4s (где s – масштаб особой точки), а черные области имеют значение «-1», а белые «+1». Типы фильтров Хаара показаны на рисунке 11.

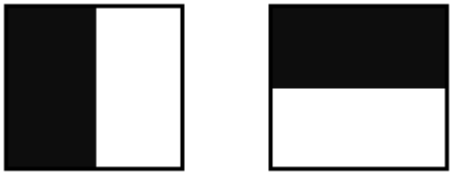

Рис. 11. Фильтры Хаара

Для эффективного вычисления фильтров Гессе и Хаара используется интегральное представление изображений. Это представление является матрицей, размерность которой совпадает с размерностью исходного изображения, а элементы вычисляются по формуле [3]:

$$II(x,y) = \sum_{i=0,j=0}^{i\le x\, j\le y} I(i,j)$$

После нахождения ключевых точек, метод SURF образует их дескрипторы. Дескриптор выглядит как набор из 64 (либо 128) чисел для каждой ключевой точки. Эти числа отображают флуктуации градиента вокруг ключевой точки. Поскольку ключевая точка представляет собой максимум гессиана, тем самым гарантируется, что в окрестности точки должны быть участки с разными градиентами. Таким образом, обеспечивается дисперсия (различие) дескрипторов для разных ключевых точек, за счет чего достигается инвариантность дескриптора относительно поворота. Размер области, на которой считается дескриптор, определяется масштабом матрицы Гессе, что обеспечивает инвариантность относительно масштаба. Результат использования дескриптора SURF представлен на рисунке 12.

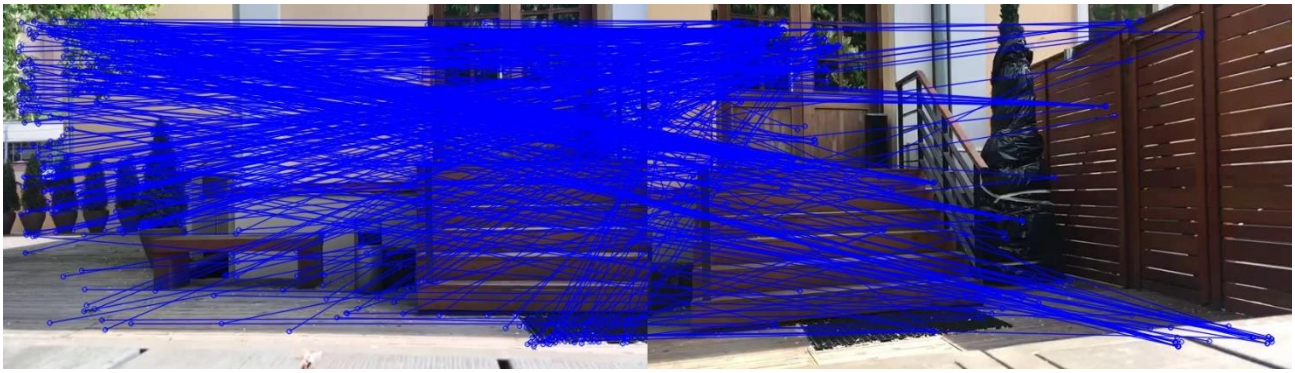

Рис. 12 Общие точки. Метод SURF

1.1.4.2. Нахождение соответствий

Следующий этап - формирование панорамного изображения между особыми точками, находятся соответствия на основе полученных инвариантных дескрипторов. Для этого строится *kd*-дерево, которое является структурой, предназначенной для хранения конечного множества точек *k*-мерного пространства [2]. В результате выделяется набор пар взаимосвязанных особых точек.

1.1.4.3. Вычисление матрицы преобразования

Итоговая последовательность всегда обладает ложными соответствиями. Для удаления ложных соответствий применяется метод согласования случайных выборок RANSAC (англ. *Random Sample Consensus*). Метод RANSAC - вероятностный метод, в котором определяется минимальная погрешность между парами точек, после чего вычисляются коэффициенты матрицы перспективного преобразования *H* размерности 3*x*3 (матрица гомографии) методом прямого линейного преобразования *DLT* (англ. *Direct Linear Transformation*). Решается система линейных алгебраических уравнений, в результате определяются 8 коэффициентов (параметров *DLT*), показывающих связь между системами координат плоскостей двух изображений [2].

1.1.4.4. Наложение изображений

Заключительный этап наложения изображений состоит из следующих частей:

1) определяется суммарный размер панорамного изображения;

2) первое изображение без изменения копируется в плоскость итогового;

3) второе изображение накладывается на плоскость первого с помощью полученной матрицы преобразования;

4) общая часть накладывается путём линейной интерполяции.

1.2. Выводы

Так как у существующих систем видео аналатики есть недостатки в областях слежения за объектом и распознавания ситуаций, а именно, проблемы с распознаванием объектов частично попавших в поле зрения камеры и соотношение образов одного объекта из видео-потоков с нескольких камер между собой было решено создать систему панорамного распознавания образов. В виду того, что использования PTZ – камер ведет к появлению новых деффектов и недостаточных потенциальных возможностей, было решено создать систему видео-аналитики панорамных изображений, созданных из одновременных фреймов из видеопотоков, поступающих в систему с нескольких камер слежения.

1.3. Цель и задачи магистерской диссертации

Целью магистерской диссертации является создание системы видеоанализа цилиндрических панорамных изображений, созданных из одновременных фреймов из видеопотоков, поступающих в систему с нескольких камер слежения.

Задачи магистерской диссертации:

1. Изучить алгоритм разбиения видеопотока, поступающего в систему видео-аналитики с камеры, на отдельные фреймы
2. Создание алгоритма получения панорамного изображения из фреймов с разных камер.
3. Разработать и применить наиболее быстрый алгоритм распознавания образов в видео-потоках.
4. Выбрать язык программирования, чтобы написанная на нем система видеоаналитики панорамных изображений могла работать с видео потоками с частотой кадров от 12fps и разрешением не меньше 480×272 пикселей.

ГЛАВА 2 РАЗРАБОТКА СИСТЕМЫ ВИДЕО АНАЛИТИКИ ПАНОРАМНЫХ ИЗОБРАЖЕНИЙ

2.1. Анализ исходных данных

Основным источником данных разрабатываемой системы будет видеопоток, с которым будет в дальнейшем работать система. Изначально будет отснято несколько видео фрагментов одновременно с нескольких видеокамер, которые будут соответствовать следующим условиям:

- Видео фрагменты запущены синхронно
- Видео фрагменты установлены так, что у соседних камер есть общая зона равная 10-30% от общей картинки, пример этого условия представлен на рисунке 13.
- Видео фрагменты синхронно заканчиваются

Рис. 13. Общая зона на двух видеопотоках

Данные видеопотоки будут в дальнейшем использовать для создания панорамного изображения и применения алгоритмов видеоанализа в реальном времени.

2.2. Архитектура разрабатываемой системы видеоаналитики панорамных изображений

Существующие системы видео аналитики в большинстве своем основываются на архитектуре «Клиент-Сервер», что позволяет хранить и обрабатывать данные в большом количестве, независимо от мощностей рабочих станций клиента. Рассмотрим более подробно виды архитектур систем видео аналитики.

2.2.1. Серверная видеоаналитика

Серверная видеоаналитика (server video analytics) предполагает централизованную обработку видеоданных на сервере. В данном архитектурном решении, сервер принимает и обрабатывает видеопотоки от множества камер или кодеров на центральном процессоре (CPU) или на графическом процессоре (GPU). Основным преимуществом серверной видео аналитики является возможность комбинирования алгоритмов видео аналитики на одной аппаратной платформе. Главный недостаток серверной видео аналитики – необходимость непрерывной передачи видео от источника видеоданных на сервер, что создает нагрузку на каналы связи. Примером серверной видео аналитики является платформа Kipod. Архитектура серверной видео аналитики представлена на рисунке 14.

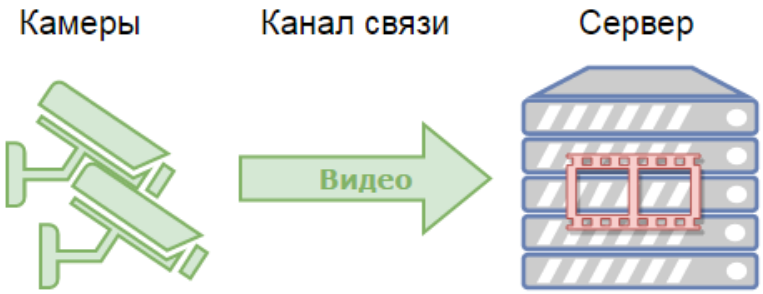

Рис. 14. Архитектура серверной видео аналитики

2.2.2. Распределенная видеоаналитика

Распределенная видеоаналитика (distributed video analytics) применяется, когда возможно использования «умных» видеокамер, которые сами способны производить элементарные действия распознавания, тем самым оставляю более сложные функции видео анализа на стороне сервера. Одним из примеров является обнаружение объектов в многокамерной системе на стороне камер, а сопоставление результатов уже происходит на стороне сервера (рисунок 15).

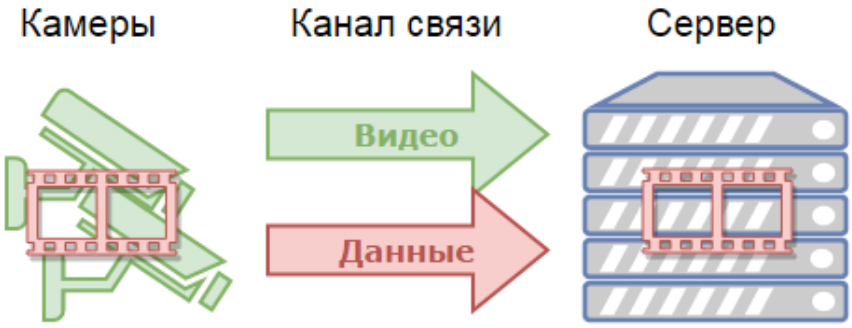

Рисунок 15. Архитектура распределенной видео аналитики

Поскольку система будет разрабатываться в рамках экономии финансовых средств, то все функции системы видео аналитики панорамных

изображений будут производиться на стороне сервера, которым будет являться рабочая станция.

2.3. Выбор среды разработки

Разрабатываемая информационная система будет основываться на технологии «клиент-сервер». В виду задачи кроссплатформенности приложения наше приложение будет web-приложением.

Достоинства веб-приложения:

- Web-приложение не требует установки дополнительного тяжеловесного программного обеспечения на рабочей станции клиента. Для полноценной работы достаточно иметь браузер и доступ в интернет.
- Web-приложение обеспечивает высокую мобильность за счет возможности пользоваться ими везде, где есть доступ в интернет.
- Web-приложение не требовательно к ресурсам и предъявляет минимальные требования к аппаратной платформе.
- Web-приложение не требует специальной настройки и администрирования (администраторами системы по сути являются разработчики приложения, что экономически выгоднее и эффективнее, чем содержать команду программистов и администраторов для установки и настройки приложений на рабочих станциях пользователей системы)

В виду того, что нами была выбрана архитектура «клиент-сервер» и веб-приложение, серверная часть будет написана на языке Java.

Java – объектно-ориентированный язык программирования. Приложения, которые были написаны на языке Java, транслируются (преобразуются) в специальный байт-код, который в дальнейшем выполняется с помощью Java Virtual Machine (JVM – виртуальная машина Java) – программой, которая

обрабатывает байтовый код и передает инструкции оборудованию, как интерпретатор [4].

У метода использования виртуальной машины Java для выполнения кода есть следующие преимущества:

- Полная независимость байт-кода от операционной системы и оборудования виртуального, железного сервера или рабочей станции, на котором выполняется код приложения, написанный на языке Java. За счет этого свойства достигается полная кроссплатформенность приложений и способность запускать исполнять приложения на любом устройстве, для которого есть виртуальная машина.
- Гибкая система безопасности, которая позволяет виртуальной машине полностью контролировать исполнение программы. Операции, которые превышают полномочия исполняемой программы, вызывают прерывание.
- Обратная совместимость с предыдущими версиями библиотек
- Наличие макроязыков. Существует возможность повторного использования кода, написанного на языках Groovy, Scala, NetRexx.

Самым главным достоинством Java, благодаря которому был выбран именно этот язык является возможность создания многопоточных приложений. А именно свойство много поточности является необходимым в рамках разрабатываемой системы видео аналитики панорамного изображения.

Но у приложений, написанных на языке Java, существуют так же и следующие недостатки:

- Медленное развитие языка. Этот недостаток ведет к тому, что возникают трудности с использованием современных необходимых библиотек. Так же, несмотря на то, что язык развивается медленно, он все же развивается, что ведет к тому, что существует большое

количество стандартных библиотек и средств, которые имеют одно и то же функциональное назначение.

- Низкое быстродействие компилятора Java
- Необходимость использования большого количества оперативной памяти из-за использования JIT-компиляции. Just-in-time compilation (JIT – компиляция) – технология динамической компиляции, которая позволяет увеличить производительность программных систем, использующих байт-код, с помощью компиляции байт-кода в машинный код непосредственно во время работы программы.

В виду недостатков в производительности и быстродействии было решено:

1. Ограничиться разработкой модулей серверной части системы, что позволит сэкономить нагрузку на аппаратную часть
2. Параллельно разработать систему на другом языке, для сравнения быстродействия алгоритмов.

Вторым языком программирования, позволяющим разрабатывать web-приложения на стороне сервера, был выбран Python.

Python – высокоуровневый язык общего назначения ориентированный на повышение производительности разрабатываемой системы и читаемости кода. Язык Python является языком более высокого уровня, чем Pascal, Java, C++ из-за того, что наличествуют в нем высокоуровневые структуры (списки, словари, тьюплы) [5].

Как и у любого языка, у Python есть свои достоинства и недостатки.

Рассмотрим достоинства языка Python:

- Кросс-платформенность интерпретатора. Интерпретатор Python реализован практически на всех платформах и операционных системах.
- Расширяемость языка.
- Огромное количество подключаемых модулей, которые позволяют использовать различные дополнительные функции (Numerical Python, Tkinter, OpenGL)

Рассмотрим особенности языка Python.

- Динамическая типизация. Благодаря отсутствию необходимого объявления типа переменных, появляется возможность автоматического обрабатывания обширного диапазона объектов
- Автоматическое управление памятью за счет распределения и освобождения памяти под объекты – «сборка мусора»
- Гибкие встроенные типы объектов – словари, списки и строки.
- Модульное программирование

2.4. Создание проектной модели

Данный пункт работы описывает проектную модель, основываясь на которую в будущем будет создаваться разрабатываемая система видео аналитики панорамных изображений. Модель проектируемой системы представлена на рисунке 16.

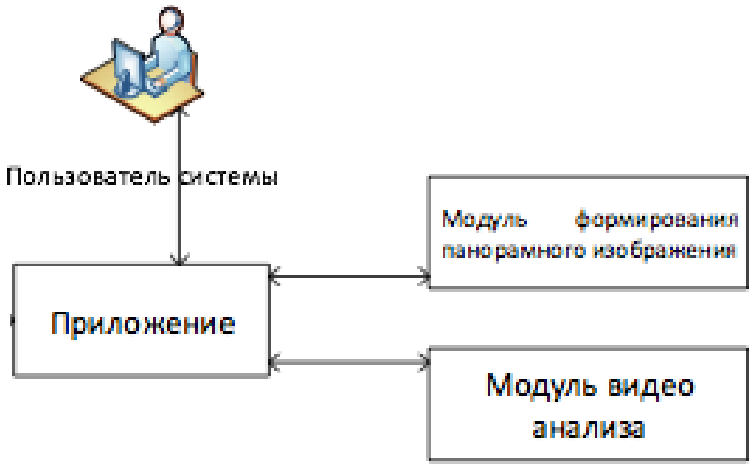

Рис. 16. Архитектура разрабатываемой системы

Разрабатываемая система видео аналитики в общем виде состоит из следующих компонентов:

- Пользователь системы. Пользователь системы взаимодействует с приложением. Выбирает устройства, которые нужно включить, для формирования панорамного изображения
- Приложение. Приложение включает в себя несколько модулей. Общими для систем, написанных на языке Java и Python, являются модули формирования панорамного изображения и видео анализа. Так же приложение записывает метаданные событий в базу данных.
- Модуль формирования панорамного изображения. Данный модуль позволяет совмещать одновременные кадры с нескольких видеокамер, формируя панорамное изображение.
- Модуль видео анализа. Данный модуль обрабатывает созданный видео-поток с панорамными изображениями и находит движение.

2.5. Разработка модели процессов

Данный пункт описывает создание модели работы системы видео аналитики панорамных изображений. Модель работы всех механизмов системы представлена на рисунке 17.

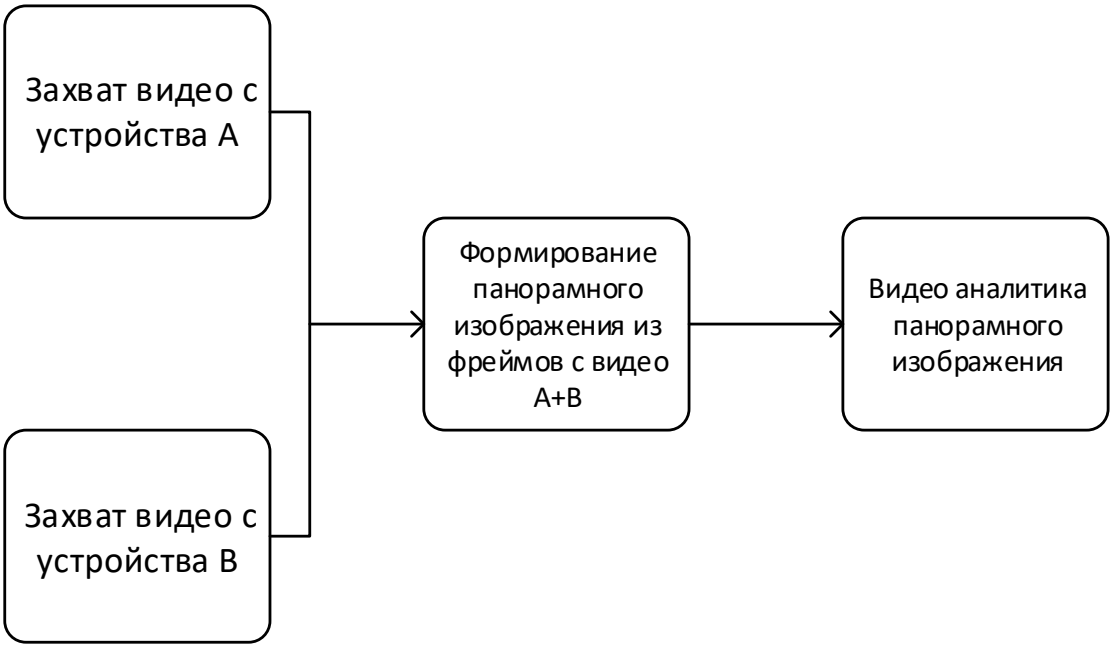

Рис. 17. Общий алгоритм работы системы видео аналитики панорамных изображений

При инициализации приложения, запускается два потока, которые захватывают видео с двух устройств. В тестовом примере используется два видеофрагмента.

Получаем два кортежа последовательности изображений $\langle a_1, a_2, \dots, a_n \rangle \in A$ и $\langle b_1, b_2, \dots, b_m \rangle \in B$, где a_i, b_i – фреймы видеопотоков, а $\langle a_1, a_2, \dots, a_n \rangle = \langle b_1, b_2, \dots, b_m \rangle$, что означает, что количество фреймов в двух видеопотоках должны быть равны.

Используя алгоритмы SURF и SIFT и определяя ключевые точки (дескрипторы) получаем новый кортеж изображений с той же длиной $C = A \times B = \langle a_1, a_2, \dots, a_n \rangle \times \langle b_1, b_2, \dots, b_n \rangle = \langle a_1 \times b_1, a_2 \times b_2, \dots, a_n \times b_m \rangle = \langle c_1, c_2, \dots, c_l \rangle$, где c_k – новый панорамный фрейм. Стоит заметить, что m=n=k, так что в дальнейшем можно использовать n, для обозначения длинны кортежа последовательности кадров обрабатываемых видеопотоков. Допустим, $t_a = t_c = t_b$, тогда $n = t_a * N$, где N – количество кадров в секунду

После создания нового кортежа кадров C_n, созданный видеопоток передается в обработку к модулю видео анализа. В разрабатываемой системе видео аналитики панорамных изображений базовой функцией распознавания образов будет являться детектор движений (motion detection).

Определение движения будет основываться на алгоритме разностных кадров. Алгоритм обрабатывая кортеж C_n сравнивает попарно фреймы, которые представляют из себя последовательность байт. Попиксельное вычисление межкадровой разностей расчитывается по формуле $d_{c_i}(x, y) = I_{c_i}(x, y) - I_{c_{i-1}}(x, y)$. После расчета разность сравнивается с заданным порогом, формируя двоичную маску:

$$m_{c_i}(x, y) = \begin{cases} 0, & d_{c_i}(x, y) < T \\ 1, & d_{c_i}(x, y) < T \end{cases}$$

где $m_{c_i}(x, y)$ – значение c_i элемента маски, а T – порог сравнения, который также называется порогом чувствительности.

Рассмотрим алгоритм более подробно.

На рисунке 18 представлены кадры c_{i-1} и c_i. На кадре c_i появился человек.

Рис. 18. Кадры c_{i-1} и c_i

Наши изображения c_{i-1} и c_i имеют разрешение 1100x480, что значит у нас для обработки имеется ~ 525000 пикселей и итераций для анализа. Для сокращения времени обработки и затрат ресурсов объединяем пиксели в блоки. Для усреднённых блоков размером 4x4 в одну минимальную зону уравнение будет выглядеть следующим образом [6].

$$I_{c_i}(x,y) = \frac{I_{c_i}(2x,2y) + I_{c_i}(2x,2y+1) + I_{c_i}(2x+1,2y) + I_{c_i}(2x+1,2y+1)}{4}$$

где I_{c_i} – яркость минимальной зоны из x-строки и y-столбца, другие – значения яркости пикселя. Благодаря объединения в блоки сокращается число итераций в 4 раза, соответственно, при увеличении размера блока, уменьшается число итераций.

После преобразования фреймов, сравниваем две матрицы цветов и на выходе получаем третью матрицу, содержащую разницу цветов по каждому блоку. Эта матрица и есть маска (рисунок 19)

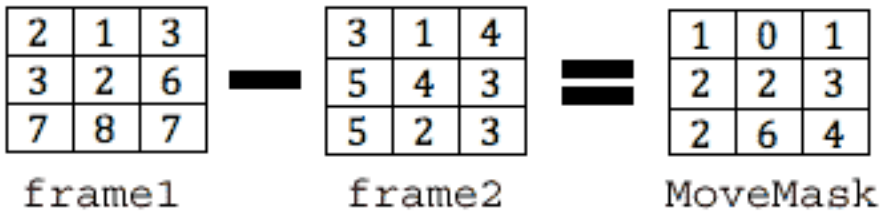

Рис. 19. Разность матриц фреймов

Получившуюся матрицу необходимо отфильтровать от шумов методом подбора «дельты». К первоначальной маске применяется «дельта» на первом шаге (рисунок 20), а затем, на втором шаге осуществляется переход в готовую маску.

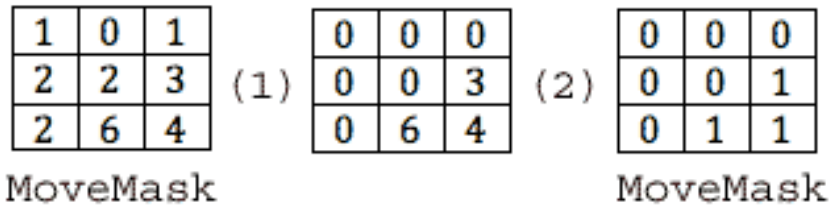

Рис. 20. Переход в маску без шумов

На выходе получаем кадр, с наложенной на него маской изменений изображений (рисунок 21).

Рис. 21 Наложение маски на кадр

Как видно на изображении – пиксели объектов, которые двигаются и отличаются от пикселей предыдущего кадра выделены маской.

После этого необходимо посчитать «силу» изменений. Одним из методов подсчета «силы» является горизонтальное суммирование блоков, расположенных рядом, но отнимая отдельные блоки, которые можно считать шумом. Возьмем, например, следующую матрицу.

0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	1	0	2
0	0	0	1	1	1	0	0	3
0	0	0	0	1	1	1	1	4
0	0	0	1	1	1	0	1	2
0	0	0	0	1	1	0	0	2
0	1	0	0	0	0	0	0	-1
								12

«Сила» изменений в данном примере равна 12. Следовательно, при установленном значении порога чувствительности T<12, пользователь системы видео аналитики панорамных изображений увидит следующую картину, представленную на рисунке 22.

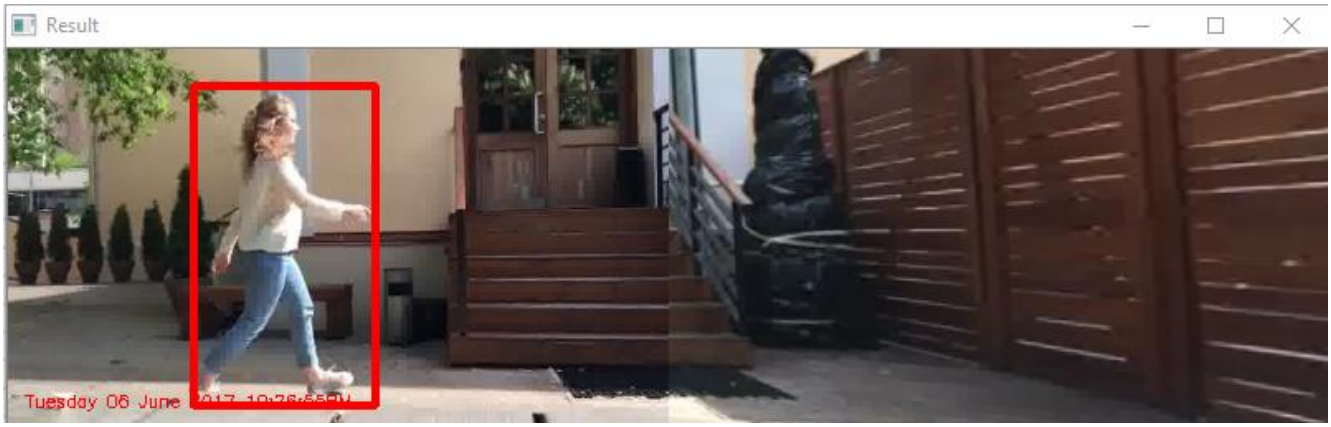

Рис.22. Определение движения

У алгоритма определения движения с помощью разностных кадров имеются следующие преимущества:

- Простота программирования
- Малая ресурсоемкость
- Хорошая и настраиваемая чувствительность

Данный алгоритм так же можно улучшить, чтобы избежать ложных срабатываний, связанных с освещенностью сцены.

2.6. Анализ производительности методов сшивания SIFT и SURF

Основным модулем нашей системы видео аналитики панорамных изображений является модуль сшивания панорамных изображений. В предыдущей главе мы рассмотрели два основных метода определения ключевых точек – алгоритмы SIFT и SURF.

Рассмотрим несколько тестовых ситуаций. Возьмем для одного набора видео фрагментов, но с разной частотой кадров и разрешением, разные методы определения ключевых точек.

Для тестирования на языке Java было написано приложение, которое загружало в буфер два изображения и обрабатывало их. Одновременно с этим подсчитывалось время, затраченное на операцию бесшовного сшивания изображений.

Тесты были проведены на рабочей станции со следующими характеристиками:

- ОС – Windows 10 Professional (Отключены дополнительные функции)
- RAM – 16GB (2x8GB DDR4 2133 MHz)
- SSD – 250GB
- CPU - Intel i5-6600 3.30 GHz

В таблице 1 представлено распределение времени в мс, необходимого на обработку и сшивание из двух кадров одного размера панорамного изображения. Алгоритм реализован на языке Java.

Таблица 1.

	640×360	848x480	1280×720	1600×900	2048×1152	2560×1440
SIFT	213	482	756	1034	3206	5286
SURF	336	668	867	1471	4935	8449

Так же, представлены графики 1 и 2, для сравнения производительности различных алгоритмов определения ключевых точек.

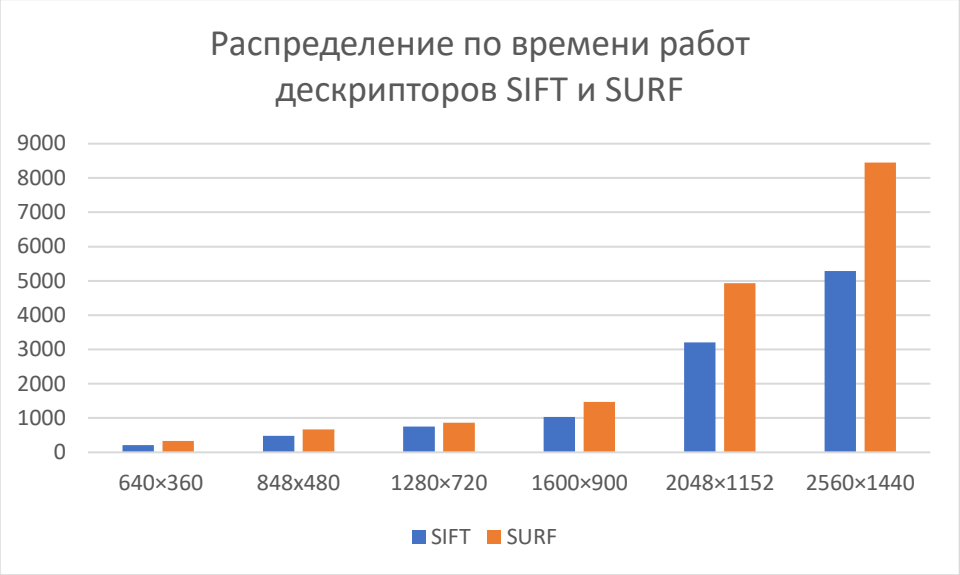

График 1. Гистограмма распределения по времени (Java)

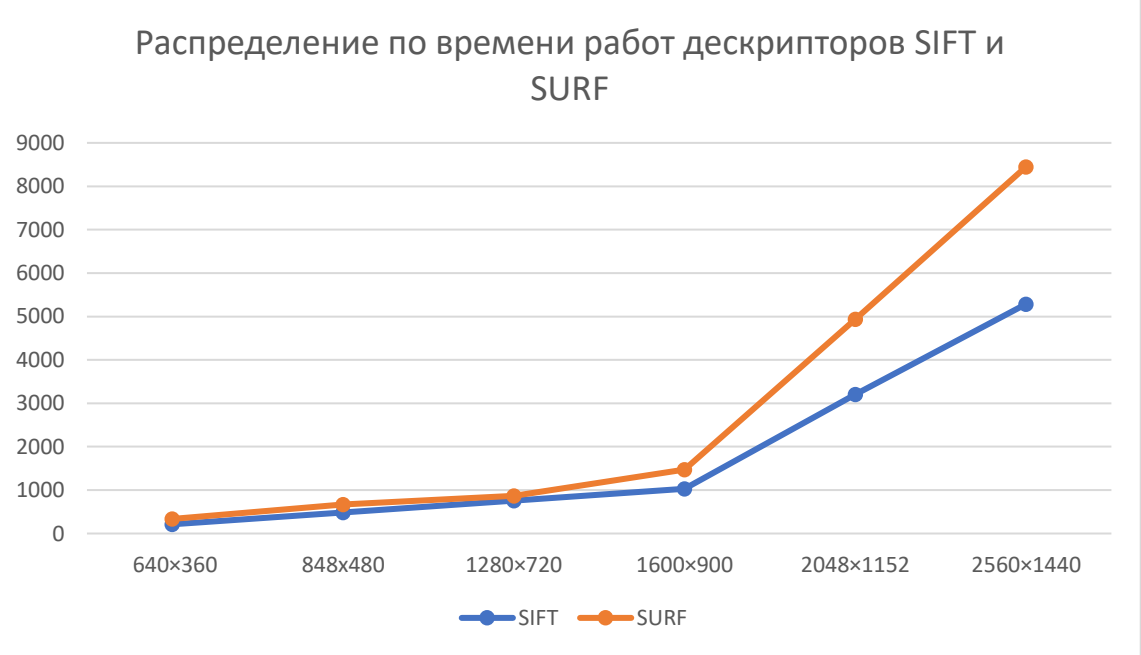

График 2. Линейное распределение производительности (Java)

Так же необходимо рассмотреть количество ключевых точек, определенных тем или иным методом дескрипторов. В таблице 2 представлено распределение количества опорных точек, найденных методом SURF и SIFT.

Таблица 2.

	640×360	848x480	1280×720	1600×900	2048×1152	2560×1440
SIFT	204	222	254	266	291	308
SURF	9	12	21	47	91	113

Ниже представлены графики 3 и 4, которые наглядно иллюстрируют распределение ключевых точек.

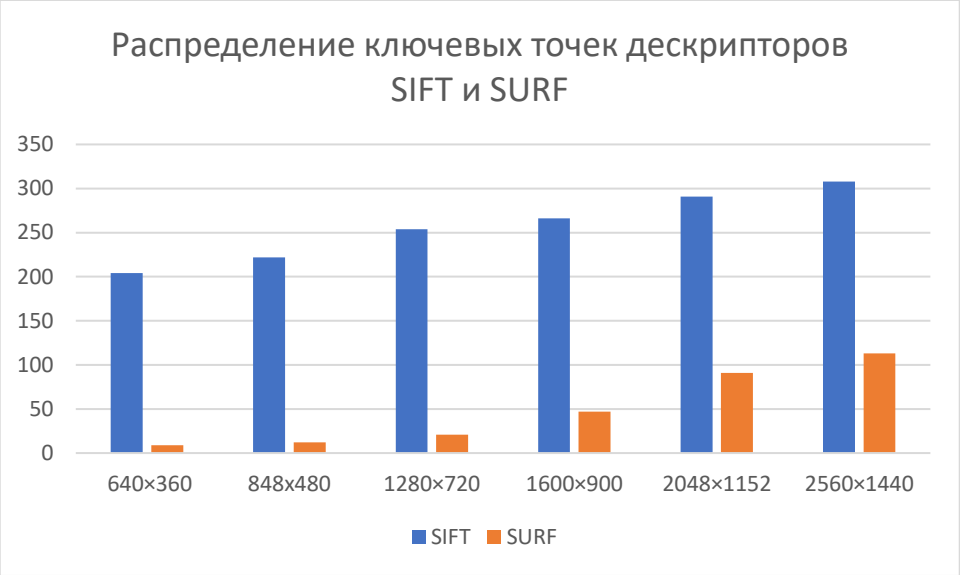

График 3. Гистограмма распределения ключевых точек (Java)

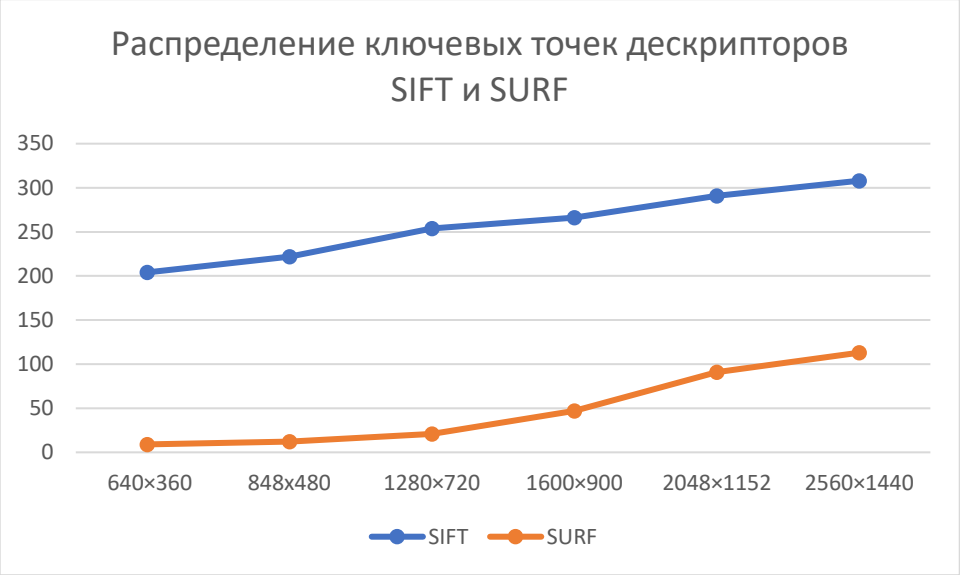

График 4. Линейное распределение ключевых точек (Java)

Так же стоит отметить, что при автоматизации процесса попарного сравнения и сшивания панорамного изображения с помощью специально написанного на языке Java модуля, были встречены дефекты. На рисунке 23 наглядно продемонстрированно сопоставление ключевых точек двух кадров из кортежей A и B.

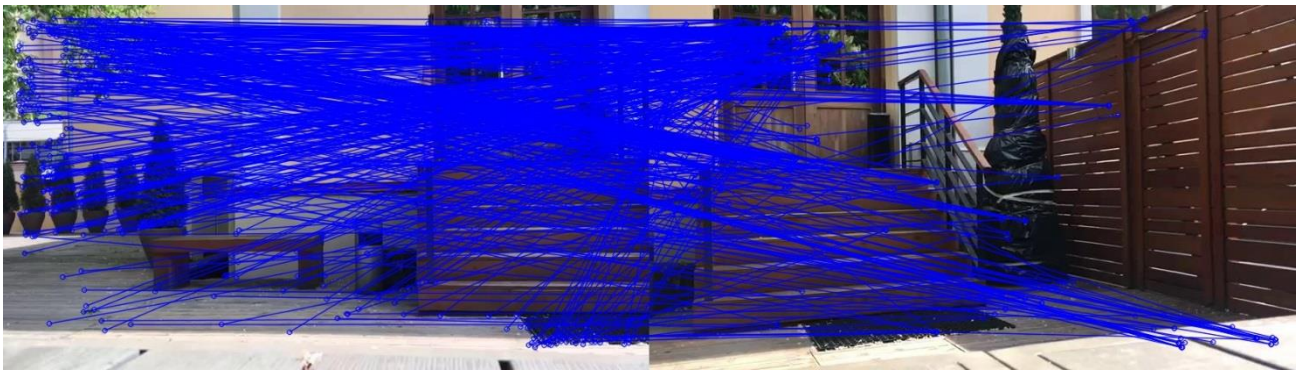

Рис. 23. Сопоставление ключевых точек.

Не смотря на данные удачные сопоставления, встречаются неудачные результаты сшивания фреймов. Один из таких неудачных результатов представлен на рисунке 24.

Рисунок 24. Неудачное сшивание панорамного изображения

Рассмотрим результаты более подробно. Обратимся к теории и вспомним, что ключевыми свойствами видео потока являются разрешение и количество кадров в секунду (fps). Минимальной частотой кадров, которая позволяет актуально следить за объектами и воспринимать последовательность фреймов как видео считается 12 fps. Но оптимальным является значение от 18 до 24 фреймов, чтобы была соблюдена плавность движений объектов.

Исходя из табл. 1 мы видим, что при использовании видео формата 2560×1440 нам необходимо 8449 мс для алгоритма SURF и 5286 для алгоритма SIFT. Следовательно, fps рассчитывается по формуле:

$fps = \frac{T}{t_{\text{затраченное}}}$, где T – 1000 мс. Таким образом, получаем, что для разрешения 2560×1440 $fps_{surf} = 0.11$ к/с и $fps_{sift} = 0.18$ к/с.

Рассмотрим минимальное разрешение 480×272. Для метода SIFT время для обработки и сшивания равняется 153 мс. Для данного разрешения метод SURF не рассматривается в виду того, что не находятся ключевые точки и панорамное изображение ни разу не сшивалось.

Таким образом получаем, $fps_{sift} = 6.53$ к/с.

В итоге получаем, что модуль обработки и сшивания панорамных изображений, написанный на языке Java смог достигнуть максимального числа fps, используя алгоритм SIFT. Но тем не менее, это число кадров в секунду слишком мало, чтобы хорошо воспринимать информацию, что так же усугубляется низким разрешением картинки.

Так же, результат работы модуля Java можно считать неудовлетворительным в виду наличия дефектов, которые представлены на рисунке 24.

Протестируем теперь модуль сшивания панорамных изображений, написанный на языке Python 2.7.

Тесты были проведены на рабочей станции со следующими характеристиками:

- ОС – Windows 10 Professional (Отключены дополнительные функции)
- RAM – 16GB (2x8GB DDR4 2133 MHz)
- SSD – 250GB
- CPU - Intel i5-6600 3.30 GHz

В таблице 3 представлено распределение времени в мс, необходимого на обработку и сшивание из двух кадров одного размера панорамного изображения. Алгоритм реализован на языке Python.

Таблица 3.

	640×360	848x480	1280×720	1600×900	2048×1152	2560×1440
SIFT	76	83	156	291	534	1328
SURF	110	194	334	427	936	2591

Так же, представлены графики 5 и 6, для сравнения производительности различных алгоритмов определения ключевых точек.

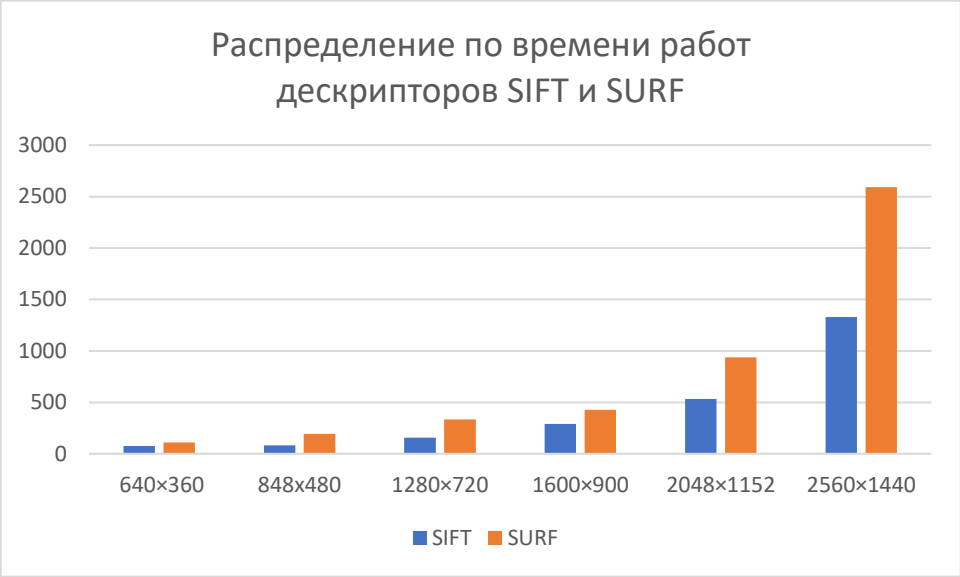

График 5. Гистограмма распределения по времени (Python)

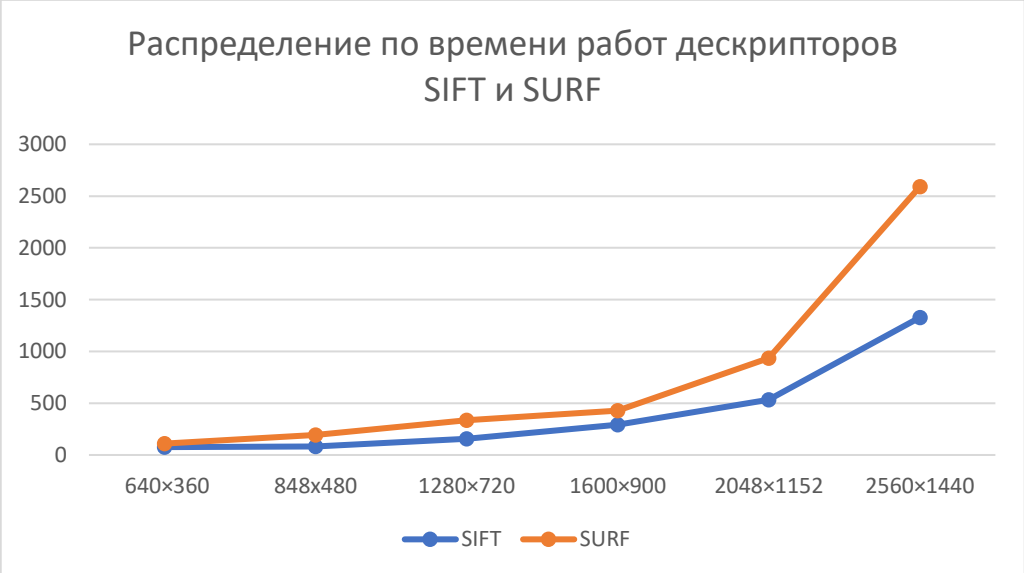

График 6. Линейное распределение ключевых точек (Python)

Теперь проверим, как хорошо определяются опорные точки в изображениях при помощи модуля, написанного на языке Python. В таблице 4

продемонстрировано распределение количества ключевых точек, определенных модулем.

Таблица 4.

	640×360	848x480	1280×720	1600×900	2048×1152	2560×1440
SIFT	201	213	229	256	284	307
SURF	47	59	73	86	91	111

)

Соответственно, на графиках 7 и 8 проиллюстрировано распределение ключевых точек, найденных алгоритмами SURF и SIFT.

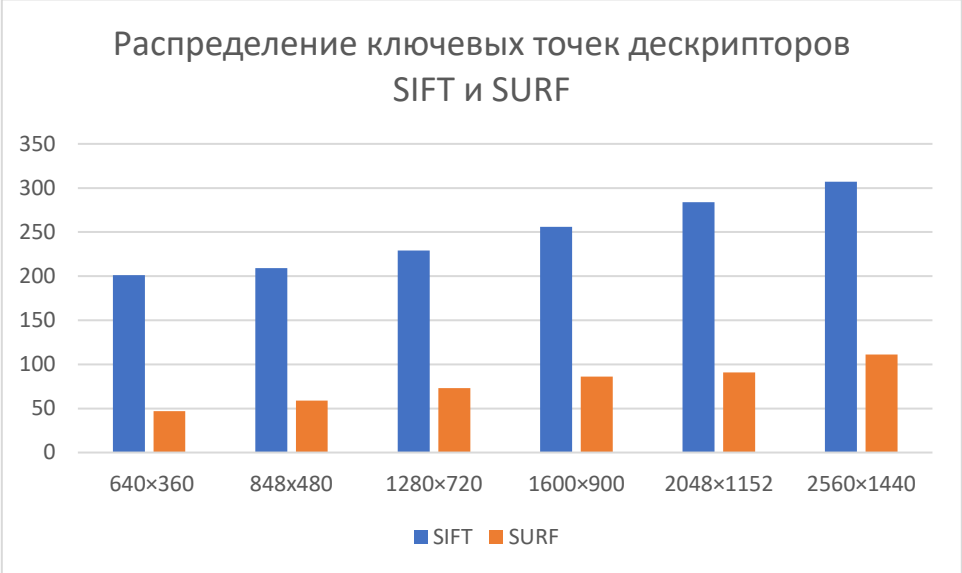

График 7. Гистограмма распределения ключевых точек (Python)

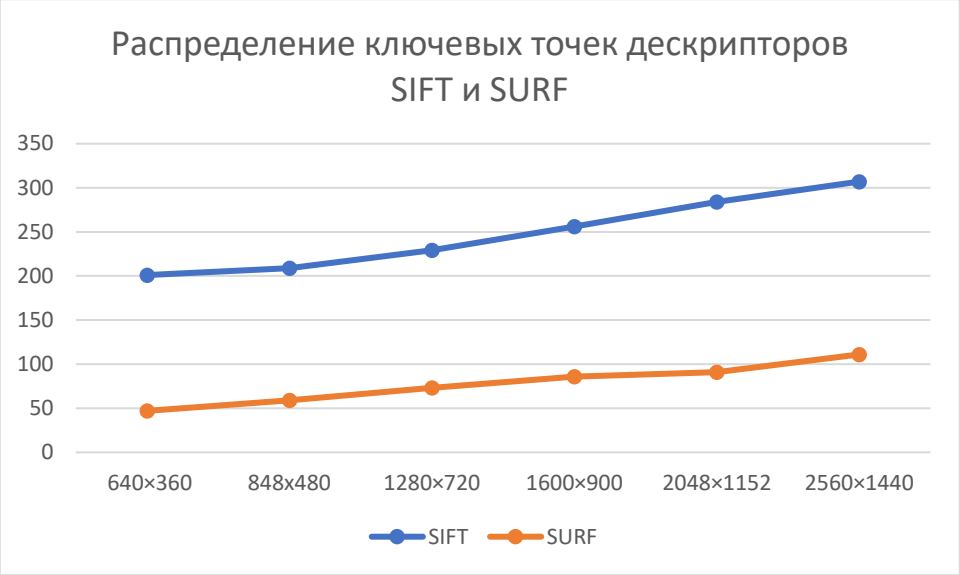

График 8. Линейное распределение ключевых точек.

При использовании приложения, написанного на языке Python, не было встречено дефектов, которые представлены на рисунке 24. Результаты сравнения представлены на рисунке 25.

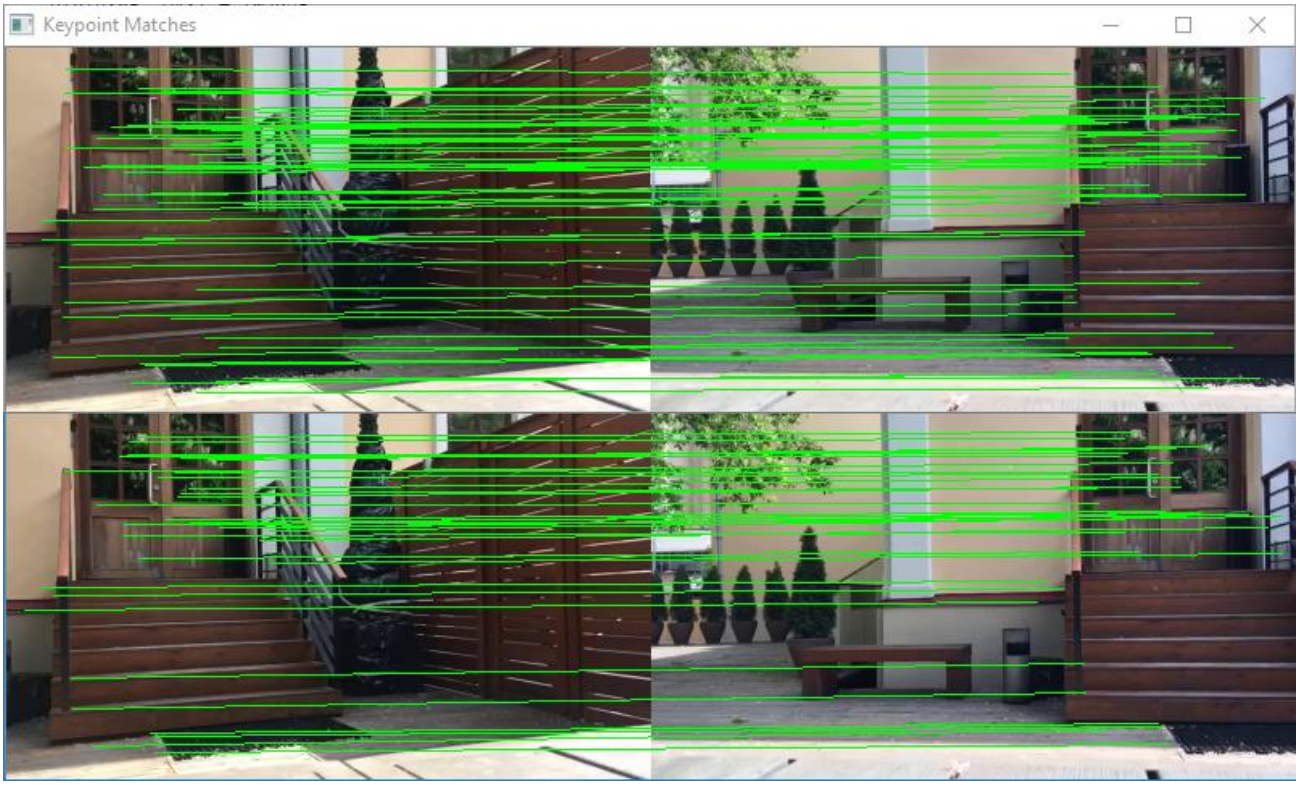

Рис. 25. Определение ключевых точек с помощью модуля Python (SURF – сверху, SIFT – снизу)

Теперь, так же, как и для модуля сшивания панорамных изображения, написанного на языке Java, рассчитаем fps.

Исходя из табл. 3, мы видим, что при использовании видео формата 2560×1440 нам необходимо 1328 мс для алгоритма SURF и 2591 для алгоритма SIFT. Следовательно, fps рассчитывается по формуле и получаем, что для разрешения 2560×1440 $fps_{surf} = 0.75$ к/с и $fps_{sift} = 0.38$ к/с.

Рассчитаем теперь fps для разрешения 640x360. Данное разрешение изображение видеопотока можно рассматривать в виду того, что для каждого из двух алгоритмов определения ключевых точек не было найдено ни одного дефектного сформированного панорамного изображения.

$fps_{surf} = 9.09$ к/с и $fps_{sift} = 13.15$ к/с.

Таким образом, исходя из опытов, проведенных в главе 2.6. можно сделать следующие выводы:

1. Алгоритм определения ключевых точек и формирования по ним цилиндрического панорамного изображения работает быстрее в модуле, написанном на языке Python, а не на языке Java
2. Алгоритм SIFT работает быстрее и определяет больше ключевых точек, чем алгоритм SURF

Из вышеперечисленных выводов следует, что при равных условиях аппаратной части, формирование панорамных изображений быстрее и качественней производится с помощью алгоритма SIFT, реализованным в модуле, написанном на языке Python. При соблюдении данных условий, можно добиться обрабатывании и формировании видео потока из панорамных изображений, которые были созданы из фреймов с разрешением 528x297 с fps равным 24 кадрам в секунду, что соответствует международным стандартам киносъемки.

ГЛАВА 3. ОЦЕНКА ЭФФЕКТИВНОСТИ РАЗРАБОТАННОГО РЕШЕНИЯ

3.1. Расчет необходимой памяти, для хранения и обработки дескрипторов

Не смотря на развитие современных технологий и увеличения размера памяти в современных серверах и рабочих станциях с одновременным уменьшением стоимости компонентов, вопрос о необходимой памяти, которая будет потребляться тем или иным процессом до сих пор очень актуальна.

В зависимости от разрядности операционной системы, приложение может рассчитывать на определенное количество памяти. Например, если в 64-разрядных системах возможно выделить приложению всю имеющуюся память, то 32-разрядных машинах невозможно передать приложению больше 4 GB RAM.

Следующим ограничением является возможность ограничения со стороны операционной системы, выделяемой одному приложению памяти. Обычно выделяется 2 GB RAM.

Мы имеем метрическое дерево, которое хранит информацию по точкам. Рассмотрим данные, которые необходимо хранить в этом дереве.

- 129 коэффициентов делящей плоскости гиперплоскости L – 132 байта
- координата центра узла -128 байт
- Радиус гиперсферы B – 4 байта
- Количество дескрипторов – 4 байта

- Указатель на индексы дескрипторов – 4 байта
- Флаг – узел с перекрытием – 1 байт
- Указатели на дочерние узлы – 8 байт

Получаем, что в итоге, на каждый узел дерева необходимо $C_t = 281$ байт.

Помимо узлов метрического дерева, так же постоянно обрабатываются массивы дескрипторов и индексы на них, а, следовательно, для всех этих элементов должна выделяться память. Индексы на дескрипторы позволяют увеличить скорость нахождения и обработки самих дескрипторов, так что их использование целесообразно. Например, операции над 4 байтовым индексом производятся быстрее, чем над 128-байтным дескриптором. Пусть $C_{id} = 4$ байт, а $C_d = 128$ байт, где C_{id} и C_d – память занимаемая индексом на дескриптор и самим дескриптором соответственно.

Тогда C_{mem} – память необходимая для обработки n дескрипторов

$$C_{mem} = n(\frac{2C_t}{l} + C_{id} + C_d)$$

где l – максимальное количество дескрипторов, представленные листовыми узлами. Пусть l=10 и $C_{mem} = 2$ GB, тогда выражая n, получим:

$$n = \frac{C_{mem}}{2C_t l^{-1} + C_{id} + C_d}$$

Подставляя значения в уравнение, получаем, что при памяти равной 2GB, можно построить дерево из ≈11.4 миллионов дескрипторов. Исходя из данных представленных в таблице 4, получаем, что при использовании видеофрагментов с разрешением 640x360, одновременно можно обрабатывать ≈56 тысяч изображений. Тогда как с разрешением 2560×1440 – 32 тысячи изображений.

3.2. Расчет необходимой области перекрытия

Весь принцип сшивания панорамных изображений основывается на совмещении кадров, имеющих некоторое «перекрытие», которое так же именуется общей зоной. Поэтому возникает задача объединения кадров по некоторому «шву».

Рассмотрим более подробно общую область двух изображений. На рисунке 26 проиллюстрировано схематическое изображение области «перекрытия» и шва соединения.

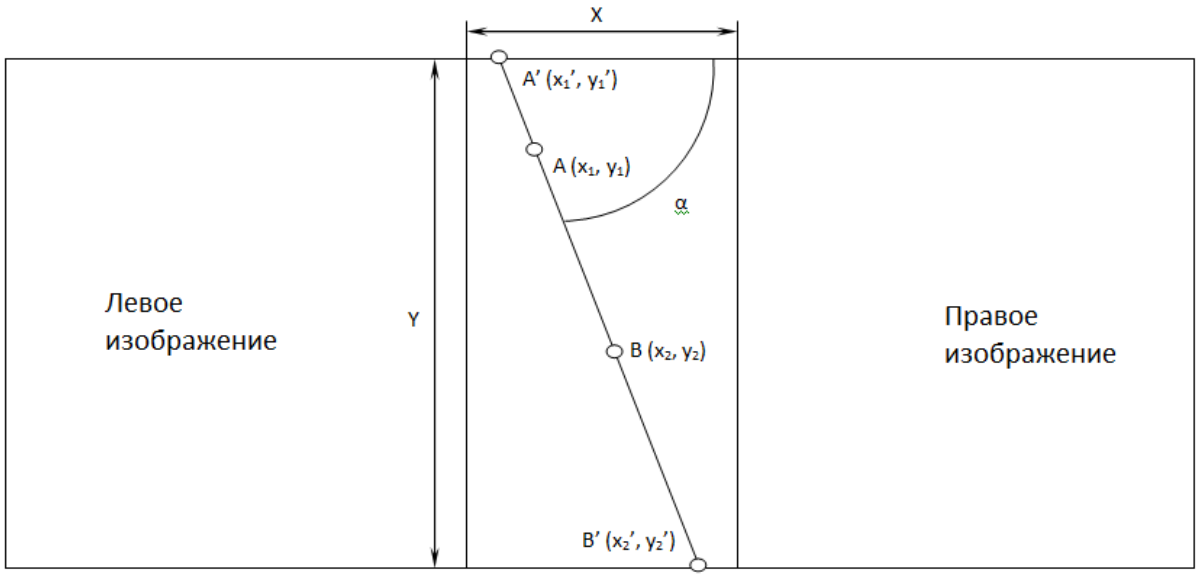

Рис. 26. Схематическое изображение области «перекрытия» и шва соединения.

Установим условие, что угловой коэффициент k должен быть больше или равен угловому коэффициенту диагонали общей полосы $K = {}^{Y}/_{X}$. В виду того, что при отсутствии области перекрытия, общая область из прямоугольной будет стремиться к линейной, а угол диагонали будет стремиться к 90°, что будет означать, что не будет найден угловой коэффициент. Для отрезка $A'B'$ представленного на рисунке 26

$$k = tg\alpha = \frac{y_2 - y_1}{x_2 - x_1}$$

В виду того, что сторона наклона шва не имеет значения для дальнейших расчетов, возьмем значение k по модулю.

$$\left|\frac{y_2 - y_1}{x_2 - x_1}\right| \geq \frac{Y}{X}$$

Наша искомая прямая «шва» не должна выходить за вертикальные границы, поэтому нужно оценить коэффициент b из уравнения $y = kx + b$. Зная угловой коэффициент k рассчитываем координаты точек A' и B'. Будем считать, что в нашей системе координат $y'_2 = Y$, а $y'_1 = 0$, тогда:

$$x'_1 = \frac{-b(y_2 - y_1)}{x_2 - x_1}$$

$$x'_2 = \frac{(Y - b)(y_2 - y_1)}{x_2 - x_1}$$

Соответственно, если координаты точек x'_1 и x'_2 не находятся в интервале (0; X), то следует перейти к поиску других точек.

Возьмем два кортежа последовательности изображений $\langle a_1, a_2, \dots, a_n\rangle \in A$ и $\langle b_1, b_2, \dots, b_m\rangle \in B$, где a_i, b_i – фреймы видеопотоков, и n=m=1000.

Разрешение фреймов возьмем 640x360 для хорошего быстродействия системы.

Пусть W – ширина кадра, а $X \in W$ –длинна общей зоны, тогда отношение $\frac{X}{W}$*100% - процентное отношение ширины общей зоны от ширины всего кадра.

Будем считать, что $X_l = X_r = X$, где X_l, X_r – ширина общих зон для левого и правого изображения соответственно.

Проведем ряд экспериментов, в которых будем постепенно уменьшать ширину X. В таблице 5 представлены результаты опытов.

Таблица 5.

X	160	140	120	100	80	60	40	20
N	201	198	205	186	171	146	86	34
L	1	7	3	19	27	29	13	9
V	1000	1000	1000	1000	1000	976	913	834
T	68	67	69	65	64	71	74	86

X – ширина общей зоны, выраженная в пикселях

N – количество ключевых точек

L – количество ложных совпадений

V – количество удачно сшитых кадров в панорамное изображение

T – время в с, затраченное на формирование 1000 панорамных снимков

На графике 9 представлена зависимости общего количества найденных ключевых точек и ложных совпадений от ширины общей зоны.

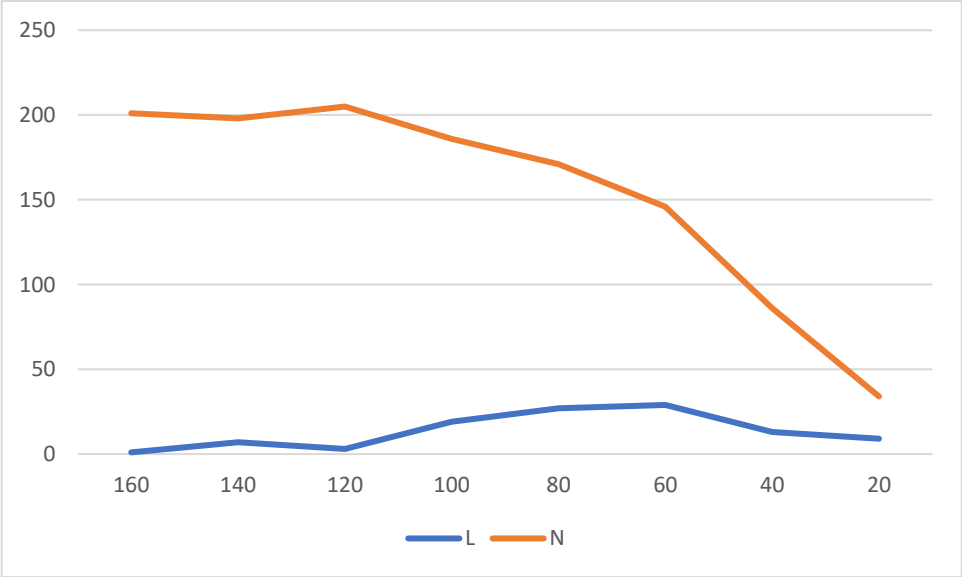

График 9. Зависимость ключевых точек от ширины общей зоны

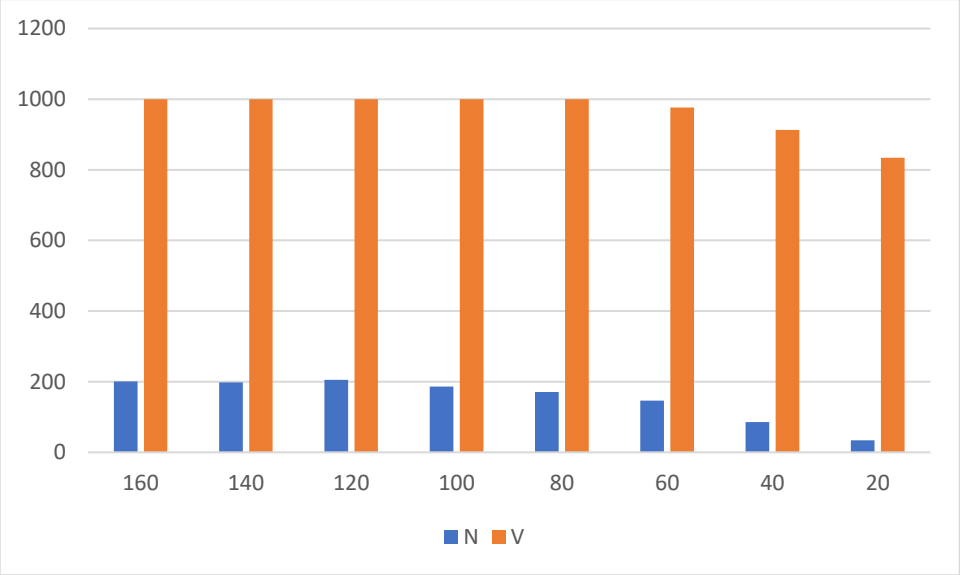

График 10. Гистограмма количества ключевых точек и панорамных снимков

Основываясь на результатах вышеописанного опыта, можно сделать следующий вывод:

В виду затраченного времени на формирование 1000 панорамных изображений в случае, когда X=100 pxl и X=80 pxl, что составляет 15.625% и 12.5% от общего размера кадра и равного 65 с и 64 с соответственно, следует располагать камеры так, что общая зона была в диапазоне от 16-12% от общей ширины кадра.

3.3. Перспективы разработанной системы видео аналитики панорамных изображений

Разработанная система видео аналитики имеет модульную систему и легко встраиваема в существующие системы. Благодаря использованию языка программирования Python была достигнута кроссплатформенность приложения.

Система видео аналитики панорамных изображений может найти свое применение в следующих областях, если дополнить ее некоторыми из описанных ниже функций:

- Торговля и бизнес – подсчет посетителей, составление маршрутов движения в магазинах и торговых центрах и анализ их, для увеличения прибыли
- Дорожное движение – определение и распознавание объектов. Распознавание номеров транспортных средств на широких магистралях.
- Безопасность – периметральное слежение за объектами и определение критичных ситуаций.

ЗАКЛЮЧЕНИЕ

В магистерской диссертации разработана система видео аналитики панорамных изображений.

Изучены существующие системы видео аналитики, их функции, возможности и недостатки и на основе анализа этой информации предложены варианты улучшения.

Разработана системы видео аналитики панорамных изображений, включающая комплекс программных средств для решения задач, поставленных перед данной системой. Содержится описание разработки программ, схематично представлено разделение задач системы по программам. Приведены контрольные примеры работы программ. Произведена оценка быстродействия и результативности создания панорамных изображений из кадров разного разрешения в системах, написанных на языке Java и Python. Была выбрана система, написанная на языке Python, которая полностью удовлетворяет установленным требованиям. На основе исследований сделаны выводы.

Разработанная система видеоаналитики полностью соответствует выдвинутым требованиям. В ходе тестирования решения было выявлено, что при разрешении входящего кадра 640x360 система может обрабатывать видео-поток с частотой кадров в 13 fps, тогда как при выборе минимального разрешения 480x272 система может работать с частотой кадров более 25 fps.

Разработка системы видео аналитики панорамных изображений полностью завершена.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Как устроенна видеоаналитика - https://habrahabr.ru/post/271207/; (Проверено – 18.06.2017)
2. К.Л. Тасов, А.Л. Федотов «Формирование панорамных изображений с камер дорожного движения» МГТУ им. Баумэна 2012г.;
3. Каскад Хаара - https://habrahabr.ru/company/recognitor/blog/228195/; (Проверено – 18.06.2017)
4. Что такое Java - https://www.java.com/ru/about/whatis_java.jsp; (Проверено – 18.06.2017)
5. Python is powerful… and fast; runs everywhere; is friendly & easy to learn - https://www.python.org/about/; (Проверено – 18.06.2017)
6. Д.В. Чеховский «Исследования сшивания нескольких кадров в единое изображение» Известия ТулГУ. Технические науки. 2013г;
7. Г.В. Дружинин, И.В. Сергеева Методические указания: Особенности разработки автоматизированных информационных систем. –М.:МИИТ, 2001.
8. А.А. Пивоваров Методы обеспечения достоверности информации в АСУ. М Радио и связь. – 1982 г.
9. ГОСТ Р ИСО/МЭК 9126-93 «Информационная технология. Оценка программной продукции. Характеристики качества и руководства по их применению».
10. ГОСТ 28195-89 «Оценка качества программных средств».
11. ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

ПРИЛОЖЕНИЕ

realtime_stitching.py

```
  # initialize the image stitcher, motion detector, and total
# number of frames read
stitcher = Stitcher()
motion = BasicMotionDetector(minArea=2500)
total = 0

# loop over frames from the video streams
while True:
   # grab the frames from their respective video streams
   ret, left = leftStream.read()
   ret, right = rightStream.read()

   # resize the frames
   left = imutils.resize(left, width=400)
   right = imutils.resize(right, width=400)

   # stitch the frames together to form the panorama
   # IMPORTANT: you might have to change this line of code
   # depending on how your cameras are oriented; frames
   # should be supplied in left-to-right order
   result = stitcher.stitch([left, right])

   # adding slitching video to fgbg
   fgmask = fgbg.apply(result)

   # no homograpy could be computed
   if result is None:
      print("[INFO] homography could not be computed")
      break

   # convert the panorama to grayscale, blur it slightly, update
   # the motion detector
   gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
   gray = cv2.GaussianBlur(gray, (21, 21), 0)
   locs = motion.update(gray)

   # only process the panorama for motion if a nice average has
   # been built up
   if total > 32 and len(locs) > 0:
      # initialize the minimum and maximum (x, y)-coordinates,
      # respectively
      (minX, minY) = (np.inf, np.inf)
      (maxX, maxY) = (-np.inf, -np.inf)

      # loop over the locations of motion and accumulate the
      # minimum and maximum locations of the bounding boxes
      for l in locs:
         (x, y, w, h) = cv2.boundingRect(l)
         (minX, maxX) = (min(minX, x), max(maxX, x + w))
         (minY, maxY) = (min(minY, y), max(maxY, y + h))

      # draw the bounding box
      cv2.rectangle(result, (minX, minY), (maxX, maxY),
```

```
            (0, 0, 255), 3)

        # increment the total number of frames read and draw the
        # timestamp on the image
        total += 1
        timestamp = datetime.datetime.now()
        ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")
        cv2.putText(result, ts, (10, result.shape[0] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.35, (0, 0, 255), 1)

        # show the output images
        cv2.imshow("Result", result)
        cv2.imshow("Left Frame", left)
        cv2.imshow("Right Frame", right)
        cv2.imshow("Backgroud Sb", fgmask)
        cv2.imshow("Gray",gray)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

# do a bit of cleanup
print("[INFO] cleaning up...")
cv2.destroyAllWindows()
leftStream.stop()
rightStream.stop()
```

panorama.py

```
# import the necessary packages
import numpy as np
import imutils
import cv2

class Stitcher:
    def __init__(self):
        # determine if we are using OpenCV v3.X and initialize the
        # cached homography matrix
        self.isv3 = imutils.is_cv3()
        self.cachedH = None

    def stitch(self, images, ratio=0.75, reprojThresh=4.0):
        # unpack the images
        (imageB, imageA) = images

        # if the cached homography matrix is None, then we need to
        # apply keypoint matching to construct it
        if self.cachedH is None:
            # detect keypoints and extract
            (kpsA, featuresA) = self.detectAndDescribe(imageA)
            (kpsB, featuresB) = self.detectAndDescribe(imageB)

            # match features between the two images
            M = self.matchKeypoints(kpsA, kpsB,
                featuresA, featuresB, ratio, reprojThresh)

            # if the match is None, then there aren't enough matched
```

```
            # keypoints to create a panorama
            if M is None:
                return None

            # cache the homography matrix
            self.cachedH = M[1]

        # apply a perspective transform to stitch the images together
        # using the cached homography matrix
        result = cv2.warpPerspective(imageA, self.cachedH,
            (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
        result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB

        # return the stitched image
        return result

    def detectAndDescribe(self, image):
        # convert the image to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # check to see if we are using OpenCV 3.X
        if self.isv3:
            # detect and extract features from the image
            descriptor = cv2.xfeatures2d.SIFT_create()
            (kps, features) = descriptor.detectAndCompute(image, None)

        # otherwise, we are using OpenCV 2.4.X
        else:
            # detect keypoints in the image
            detector = cv2.FeatureDetector_create("SIFT")
            kps = detector.detect(gray)

            # extract features from the image
            extractor = cv2.DescriptorExtractor_create("SIFT")
            (kps, features) = extractor.compute(gray, kps)

        # convert the keypoints from KeyPoint objects to NumPy
        # arrays
        kps = np.float32([kp.pt for kp in kps])

        # return a tuple of keypoints and features
        return (kps, features)

    def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB,
        ratio, reprojThresh):
        # compute the raw matches and initialize the list of actual
        # matches
        matcher = cv2.DescriptorMatcher_create("BruteForce")
        rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
        matches = []

        # loop over the raw matches
        for m in rawMatches:
            # ensure the distance is within a certain ratio of each
            # other (i.e. Lowe's ratio test)
            if len(m) == 2 and m[0].distance < m[1].distance * ratio:
                matches.append((m[0].trainIdx, m[0].queryIdx))

        # computing a homography requires at least 4 matches
```

```
        if len(matches) > 4:
            # construct the two sets of points
            ptsA = np.float32([kpsA[i] for (_, i) in matches])
            ptsB = np.float32([kpsB[i] for (i, _) in matches])

            # compute the homography between the two sets of points
            (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
                reprojThresh)

            # return the matches along with the homograpy matrix
            # and status of each matched point
            return (matches, H, status)

        # otherwise, no homograpy could be computed
return None
```

basicmotiondetector.py

```
# import the necessary packages
import imutils
import cv2

class BasicMotionDetector:
    def __init__(self, accumWeight=0.5, deltaThresh=5, minArea=5000):
        # determine the OpenCV version, followed by storing the
        # the frame accumulation weight, the fixed threshold for
        # the delta image, and finally the minimum area required
        # for "motion" to be reported
        self.isv2 = imutils.is_cv2()
        self.accumWeight = accumWeight
        self.deltaThresh = deltaThresh
        self.minArea = minArea

        # initialize the average image for motion detection
        self.avg = None

    def update(self, image):
        # initialize the list of locations containing motion
        locs = []

        # if the average image is None, initialize it
        if self.avg is None:
            self.avg = image.astype("float")
            return locs

        # otherwise, accumulate the weighted average between
        # the current frame and the previous frames, then compute
        # the pixel-wise differences between the current frame
        # and running average
        cv2.accumulateWeighted(image, self.avg, self.accumWeight)
        frameDelta = cv2.absdiff(image, cv2.convertScaleAbs(self.avg))

        # threshold the delta image and apply a series of dilations
        # to help fill in holes
        thresh = cv2.threshold(frameDelta, self.deltaThresh, 255,
            cv2.THRESH_BINARY)[1]
        thresh = cv2.dilate(thresh, None, iterations=2)

        # find contours in the thresholded image, taking care to
```

```
        # use the appropriate version of OpenCV
        cnts = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
           cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if self.isv2 else cnts[1]

        # loop over the contours
        for c in cnts:
           # only add the contour to the locations list if it
           # exceeds the minimum area
           if cv2.contourArea(c) > self.minArea:
              locs.append(c)

        # return the set of locations
        return locs
```

panoramastitcher.py

```
# import the necessary packages
from pyimagesearch.panorama import Stitcher
import argparse
import imutils
import cv2

# load the two images and resize them to have a width of 400 pixels
# (for faster processing)
imageA = cv2.imread('D:\Diploma\Frames_L\LFrame_1.jpg')
imageB = cv2.imread('D:\Diploma\Frames_R\RFrame_1.jpg')
imageA = imutils.resize(imageA, width=400)
imageB = imutils.resize(imageB, width=400)

# stitch the images together to create a panorama
stitcher = Stitcher()
(result, vis) = stitcher.stitch([imageA, imageB], showMatches=True)

# show the images
cv2.imshow("Image A", imageA)
cv2.imshow("Image B", imageB)
cv2.imshow("Keypoint Matches", vis)
cv2.imshow("Result", result)
cv2.waitKey(0)
```

FramesTogether.java

```
public class FramesTogether {
    /*image1 = Highgui.imread("D:/Diploma/Frames_R/RFrame_1.jpg");
    image2 = Highgui.imread("D:/Diploma/Frames_L/LFrame_1.jpg");*/
    static Mat image1;
    static Mat image2;

    static FeatureDetector fd;
    static DescriptorExtractor fe;
    static DescriptorMatcher fm;

    public static void initialise(){

        fd = FeatureDetector.create(FeatureDetector.BRISK);
        fe = DescriptorExtractor.create(DescriptorExtractor.SURF);
        fm = DescriptorMatcher.create(DescriptorMatcher.BRUTEFORCE);
        long start = System.currentTimeMillis();
```

```
        //images
        image1 = Highgui.imread("D:/Diploma/Frames_L/LFrame_2.jpg");
        image2 = Highgui.imread("D:/Diploma/Frames_R/RFrame_2.jpg");

/*        image1 = Highgui.imread("C:/Users/alcob/Downloads/img_11.jpg");
        image2 = Highgui.imread("C:/Users/alcob/Downloads/img_12.jpg");*/

        //structures for the keypoints from the 2 images
        MatOfKeyPoint keypoints1 = new MatOfKeyPoint();
        MatOfKeyPoint keypoints2 = new MatOfKeyPoint();

        //structures for the computed descriptors
        Mat descriptors1 = new Mat();
        Mat descriptors2 = new Mat();

        //structure for the matches
        MatOfDMatch matches = new MatOfDMatch();

        //getting the keypoints
        fd.detect(image1, keypoints1);
        fd.detect(image1, keypoints2);

        //getting the descriptors from the keypoints
        fe.compute(image1, keypoints1, descriptors1);
        fe.compute(image2, keypoints2, descriptors2);

        //getting the matches the 2 sets of descriptors
        fm.match(descriptors1, descriptors2, matches);

        //turn the matches to a list
        List<DMatch> matchesList = matches.toList();

        Double maxDist = 0.0; //keep track of max distance from the matches
        Double minDist = 100.0; //keep track of min distance from the matches

        //calculate max & min distances between keypoints
        for(int i=0; i<keypoints1.rows();i++){
            Double dist = (double) matchesList.get(i).distance;
            if (dist<minDist) minDist = dist;
            if(dist>maxDist) maxDist=dist;
        }

        System.out.println("max dist: " + maxDist );
        System.out.println("min dist: " + minDist);

        //structure for the good matches
        LinkedList<DMatch> goodMatches = new LinkedList<DMatch>();

        //use only the good matches (i.e. whose distance is less than
3*min_dist)
        for(int i=0;i<descriptors1.rows();i++){
            if(matchesList.get(i).distance<3*minDist){
                goodMatches.addLast(matchesList.get(i));
            }
        }

        //structures to hold points of the good matches (coordinates)
        LinkedList<Point> objList = new LinkedList<Point>(); // image1
        LinkedList<Point> sceneList = new LinkedList<Point>(); //image 2
```

```
        List<KeyPoint> keypoints_objectList = keypoints1.toList();
        List<KeyPoint> keypoints_sceneList = keypoints2.toList();

        //putting the points of the good matches into above structures
        for(int i = 0; i<goodMatches.size(); i++){

objList.addLast(keypoints_objectList.get(goodMatches.get(i).queryIdx).pt);

sceneList.addLast(keypoints_sceneList.get(goodMatches.get(i).trainIdx).pt);
        }

        System.out.println("\nNum. of good matches" +goodMatches.size());

        MatOfDMatch gm = new MatOfDMatch();
        gm.fromList(goodMatches);

        //converting the points into the appropriate data structure
        MatOfPoint2f obj = new MatOfPoint2f();
        obj.fromList(objList);

        MatOfPoint2f scene = new MatOfPoint2f();
        scene.fromList(sceneList);

        //finding the homography matrix
        Mat H = Calib3d.findHomography(obj, scene);

        //LinkedList<Point> cornerList = new LinkedList<Point>();
        Mat obj_corners = new Mat(4,1,CvType.CV_32FC2);
        Mat scene_corners = new Mat(4,1,CvType.CV_32FC2);

        obj_corners.put(0,0, new double[]{0,0});
        obj_corners.put(0,0, new double[]{image1.cols(),0});
        obj_corners.put(0,0, new double[]{image1.cols(),image1.rows()});
        obj_corners.put(0,0, new double[]{0,image1.rows()});

        Core.perspectiveTransform(obj_corners, scene_corners, H);

        //structure to hold the result of the homography matrix
        Mat result = new Mat();

        //size of the new image - i.e. image 1 + image 2
        Size s = new Size(image1.cols()+image2.cols(),image1.rows());

        //using the homography matrix to warp the two images
        Imgproc.warpPerspective(image1, result, H, s);
        int i = image1.cols();
        Mat m = new Mat(result,new Rect(i,0,image2.cols(), image2.rows()));

        image2.copyTo(m);

        Mat img_mat = new Mat();

        Features2d.drawMatches(image1, keypoints1, image2, keypoints2, gm,
img_mat, new Scalar(254,0,0),new Scalar(254,0,0) , new MatOfByte(), 2);

        //creating the output file
        boolean imageStitched =
Highgui.imwrite("D:/Diploma/Slitched/Slitched.jpg",result);
```

```
        boolean imageMatched =
Highgui.imwrite("D:/Diploma/Slitched/imageMatched.jpg",img_mat);
        System.out.println("run time (millis): " + (System.currentTimeMillis() -
start));
    }

    public static void main(String args[]){
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        initialise();
    }
        /*Mat gray_image1 = new Mat();
        Mat gray_image2 = new Mat();

        Imgproc.cvtColor(img1, gray_image1, Imgproc.COLOR_RGB2GRAY);
        Imgproc.cvtColor(img2, gray_image2, Imgproc.COLOR_RGB2GRAY);

        MatOfKeyPoint keyPoints1 = new MatOfKeyPoint();
        MatOfKeyPoint keyPoints2 = new MatOfKeyPoint();

        FeatureDetector detector = FeatureDetector.create(FeatureDetector.SURF);
        detector.detect(gray_image1, keyPoints1);
        detector.detect(gray_image2, keyPoints2);

        Mat descriptors1 = new Mat();
        Mat descriptors2 = new Mat();

        DescriptorExtractor extractor =
DescriptorExtractor.create(DescriptorExtractor.SURF);
        extractor.compute(gray_image1, keyPoints1, descriptors1);
        extractor.compute(gray_image2, keyPoints2, descriptors2);

        MatOfDMatch matches = new MatOfDMatch();

        DescriptorMatcher matcher =
DescriptorMatcher.create(DescriptorMatcher.FLANNBASED);
        matcher.match(descriptors1, descriptors2, matches);

        double max_dist = 0; double min_dist = 100;
        List<DMatch> listMatches = matches.toList();

        for( int i = 0; i < listMatches.size(); i++ ) {
            double dist = listMatches.get(i).distance;
            if( dist < min_dist ) min_dist = dist;
            if( dist > max_dist ) max_dist = dist;
        }

        System.out.println("Min: " + min_dist);
        System.out.println("Max: " + max_dist);

        LinkedList<DMatch> good_matches = new LinkedList<DMatch>();
        MatOfDMatch goodMatches = new MatOfDMatch();
        for(int i = 0; i < listMatches.size(); i++){
            if(listMatches.get(i).distance < 2*min_dist){
                good_matches.addLast(listMatches.get(i));
            }
        }
        goodMatches.fromList(good_matches);

        Mat img_matches = new Mat(new Size(img1.cols()+img2.cols(),img1.rows()),
```

```
CvType.CV_32FC2);

        LinkedList<Point> imgPoints1List = new LinkedList<Point>();
        LinkedList<Point> imgPoints2List = new LinkedList<Point>();
        List<KeyPoint> keypoints1List = keyPoints1.toList();
        List<KeyPoint> keypoints2List = keyPoints2.toList();

        for(int i = 0; i<good_matches.size(); i++){

imgPoints1List.addLast(keypoints1List.get(good_matches.get(i).queryIdx).pt);

imgPoints2List.addLast(keypoints2List.get(good_matches.get(i).trainIdx).pt);
        }

        MatOfPoint2f obj = new MatOfPoint2f();
        obj.fromList(imgPoints1List);
        MatOfPoint2f scene = new MatOfPoint2f();
        scene.fromList(imgPoints2List);

        Mat H = Calib3d.findHomography(obj, scene, Calib3d.RANSAC,3);

        Size s = new Size(img2.cols() + img1.cols(),img1.rows());

        Imgproc.warpPerspective(img1, img_matches, H, s);
        Mat m = new Mat(img_matches,new Rect(0,0,img2.cols(), img2.rows()));

        img2.copyTo(m);
        int n = 1;
        Highgui.imwrite("D:/Diploma/Slitched/Slitched.jpg", img_matches);
    }*/
}
```

JpegImagesToJava.java

```
public class JpegImagesToMovie implements ControllerListener, DataSinkListener {

    public boolean doIt(int width, int height, int frameRate, Vector inFiles,
                        MediaLocator outML) throws MalformedURLException {
        ImageDataSource ids = new ImageDataSource(width, height, frameRate,
                inFiles);

        Processor p;

        try {
            //System.err
            //      .println("- create processor for the image datasource ...");
            p = Manager.createProcessor(ids);
        } catch (Exception e) {
            System.err
                    .println("Yikes!  Cannot create a processor from the data
source.");
            return false;
        }

        p.addControllerListener(this);

        // Put the Processor into configured state so we can set
        // some processing options on the processor.
```

```
        p.configure();
        if (!waitForState(p, p.Configured)) {
            System.err.println("Failed to configure the processor.");
            return false;
        }

        // Set the output content descriptor to MPEG
        p.setContentDescriptor(new ContentDescriptor(
                FileTypeDescriptor.QUICKTIME));

        // Query for the processor for supported formats.
        // Then set it on the processor.
        TrackControl tcs[] = p.getTrackControls();
        Format f[] = tcs[0].getSupportedFormats();
        if (f == null || f.length <= 0) {
            System.err.println("The mux does not support the input format: "
                    + tcs[0].getFormat());
            return false;
        }

        tcs[0].setFormat(f[0]);

        //System.err.println("Setting the track format to: " + f[0]);

        // We are done with programming the processor. Let's just
        // realize it.
        p.realize();
        if (!waitForState(p, p.Realized)) {
            System.err.println("Failed to realize the processor.");
            return false;
        }

        // Now, we'll need to create a DataSink.
        DataSink dsink;
        if ((dsink = createDataSink(p, outML)) == null) {
            System.err
                    .println("Failed to create a DataSink for the given output
MediaLocator: "
                            + outML);
            return false;
        }

        dsink.addDataSinkListener(this);
        fileDone = false;

        System.out.println("Generating the video : "+outML.getURL().toString());

        // OK, we can now start the actual transcoding.
        try {
            p.start();
            dsink.start();
        } catch (IOException e) {
            System.err.println("IO error during processing");
            return false;
        }

        // Wait for EndOfStream event.
        waitForFileDone();
```

```
        // Cleanup.
        try {
            dsink.close();
        } catch (Exception e) {
        }
        p.removeControllerListener(this);

        System.out.println("Video creation completed!!!!!");
        return true;
    }

    /**
     * Create the DataSink.
     */
    DataSink createDataSink(Processor p, MediaLocator outML) {

        DataSource ds;

        if ((ds = p.getDataOutput()) == null) {
            System.err
                    .println("Something is really wrong: the processor does not
have an output DataSource");
            return null;
        }

        DataSink dsink;

        try {
            //System.err.println("- create DataSink for: " + outML);
            dsink = Manager.createDataSink(ds, outML);
            dsink.open();
        } catch (Exception e) {
            System.err.println("Cannot create the DataSink: " + e);
            return null;
        }
        return dsink;
    }
    Object waitSync = new Object();
    boolean stateTransitionOK = true;
    /**
     * Block until the processor has transitioned to the given state. Return
     * false if the transition failed.
     */
    boolean waitForState(Processor p, int state) {
        synchronized (waitSync) {
            try {
                while (p.getState() < state && stateTransitionOK)
                    waitSync.wait();
            } catch (Exception e) {
            }
        }
        return stateTransitionOK;
    }
    /**
     * Controller Listener.
     */
    public void controllerUpdate(ControllerEvent evt) {

        if (evt instanceof ConfigureCompleteEvent
```

```
            || evt instanceof RealizeCompleteEvent
            || evt instanceof PrefetchCompleteEvent) {
        synchronized (waitSync) {
            stateTransitionOK = true;
            waitSync.notifyAll();
        }
    } else if (evt instanceof ResourceUnavailableEvent) {
        synchronized (waitSync) {
            stateTransitionOK = false;
            waitSync.notifyAll();
        }
    } else if (evt instanceof EndOfMediaEvent) {
        evt.getSourceController().stop();
        evt.getSourceController().close();
    }
}

Object waitFileSync = new Object();
boolean fileDone = false;
boolean fileSuccess = true;

/**
 * Block until file writing is done.
 */
boolean waitForFileDone() {
    synchronized (waitFileSync) {
        try {
            while (!fileDone)
                waitFileSync.wait();
        } catch (Exception e) {
        }
    }
    return fileSuccess;
}

/**
 * Event handler for the file writer.
 */
public void dataSinkUpdate(DataSinkEvent evt) {

    if (evt instanceof EndOfStreamEvent) {
        synchronized (waitFileSync) {
            fileDone = true;
            waitFileSync.notifyAll();
        }
    } else if (evt instanceof DataSinkErrorEvent) {
        synchronized (waitFileSync) {
            fileDone = true;
            fileSuccess = false;
            waitFileSync.notifyAll();
        }
    }
}

/*public static void main(String args[]) {

    if (args.length == 0)
        prUsage();
```

```
        // Parse the arguments.
        int i = 0;
        int width = -1, height = -1, frameRate = 1;
        Vector inputFiles = new Vector();
        String outputURL = null;

        while (i < args.length) {
            if (args[i].equals("-w")) {
                i++;
                if (i >= args.length)
                    prUsage();
                width = new Integer(args[i]).intValue();
            } else if (args[i].equals("-h")) {
                i++;
                if (i >= args.length)
                    prUsage();
                height = new Integer(args[i]).intValue();
            } else if (args[i].equals("-f")) {
                i++;
                if (i >= args.length)
                    prUsage();
                frameRate = new Integer(args[i]).intValue();
            } else if (args[i].equals("-o")) {
                i++;
                if (i >= args.length)
                    prUsage();
                outputURL = args[i];
            } else {
                inputFiles.addElement(args[i]);
            }
            i++;
        }
        if (outputURL == null || inputFiles.size() == 0)
            prUsage();
        // Check for output file extension.
        if (!outputURL.endsWith(".mov") && !outputURL.endsWith(".MOV")) {
            System.err
                    .println("The output file extension should end with a .mov
extension");
            prUsage();
        }
        if (width < 0 || height < 0) {
            System.err.println("Please specify the correct image size.");
            prUsage();
        }
        // Check the frame rate.
        if (frameRate < 1)
            frameRate = 1;

        // Generate the output media locators.
        MediaLocator oml;

        if ((oml = createMediaLocator(outputURL)) == null) {
            System.err.println("Cannot build media locator from: " + outputURL);
            System.exit(0);
        }
        JpegImagesToMovie imageToMovie = new JpegImagesToMovie();
        imageToMovie.doIt(width, height, frameRate, inputFiles, oml);
```

```
            System.exit(0);
        }*/
        static void prUsage() {
            System.err
                    .println("Usage: java JpegImagesToMovie -w <width> -h <height> -
f <frame rate> -o <output URL> <input JPEG file 1> <input JPEG file 2> ...");
            System.exit(-1);
        }
        /**
         * Create a media locator from the given string.
         */
        static MediaLocator createMediaLocator(String url) {
            MediaLocator ml;
            if (url.indexOf(":") > 0 && (ml = new MediaLocator(url)) != null)
                return ml;
            if (url.startsWith(File.separator)) {
                if ((ml = new MediaLocator("file:" + url)) != null)
                    return ml;
            } else {
                String file = "file:" + System.getProperty("user.dir")
                        + File.separator + url;
                if ((ml = new MediaLocator(file)) != null)
                    return ml;
            }
            return null;
        }
        // ////////////////////////////////////////////
        //
        // Inner classes.
        // ////////////////////////////////////////////
        /**
         * A DataSource to read from a list of JPEG image files and turn that into a
         * stream of JMF buffers. The DataSource is not seekable or positionable.
         */
        class ImageDataSource extends PullBufferDataSource {

            ImageSourceStream streams[];
            ImageDataSource(int width, int height, int frameRate, Vector images) {
                streams = new ImageSourceStream[1];
                streams[0] = new ImageSourceStream(width, height, frameRate,
images);
            }
            public void setLocator(MediaLocator source) {
            }
            public MediaLocator getLocator() {
                return null;
            }
            /**
             * Content type is of RAW since we are sending buffers of video frames
             * without a container format.
             */
            public String getContentType() {
                return ContentDescriptor.RAW;
            }
            public void connect() {
            }
            public void disconnect() {
            }
            public void start() {
```

```
    }
    public void stop() {
    }
    /**
     * Return the ImageSourceStreams.
     */
    public PullBufferStream[] getStreams() {
        return streams;
    }
    /**
     * We could have derived the duration from the number of frames and
     * frame rate. But for the purpose of this program, it's not necessary.
     */
    public Time getDuration() {
        return DURATION_UNKNOWN;
    }

    public Object[] getControls() {
        return new Object[0];
    }

    public Object getControl(String type) {
        return null;
    }
}
/**
 * The source stream to go along with ImageDataSource.
 */
class ImageSourceStream implements PullBufferStream {

    Vector images;
    int width, height;
    VideoFormat format;

    int nextImage = 0; // index of the next image to be read.
    boolean ended = false;

    public ImageSourceStream(int width, int height, int frameRate,
                             Vector images) {
        this.width = width;
        this.height = height;
        this.images = images;

        format = new VideoFormat(VideoFormat.JPEG, new Dimension(width,
                height), Format.NOT_SPECIFIED, Format.byteArray,
                (float) frameRate);
    }
    /**
     * We should never need to block assuming data are read from files.
     */
    public boolean willReadBlock() {
        return false;
    }
    /**
     * This is called from the Processor to read a frame worth of video
     * data.
     */
    public void read(Buffer buf) throws IOException {
        // Check if we've finished all the frames.
```

```
        if (nextImage >= images.size()) {
            // We are done. Set EndOfMedia.
            //System.err.println("Done reading all images.");
            buf.setEOM(true);
            buf.setOffset(0);
            buf.setLength(0);
            ended = true;
            return;
        }
        String imageFile = (String) images.elementAt(nextImage);
        nextImage++;
        //System.err.println("  - reading image file: " + imageFile);
        // Open a random access file for the next image.
        RandomAccessFile raFile;
        raFile = new RandomAccessFile(imageFile, "r");
        byte data[] = null;
        // Check the input buffer type & size.
        if (buf.getData() instanceof byte[])
            data = (byte[]) buf.getData();
        // Check to see the given buffer is big enough for the frame.
        if (data == null || data.length < raFile.length()) {
            data = new byte[(int) raFile.length()];
            buf.setData(data);
        }
        // Read the entire JPEG image from the file.
        raFile.readFully(data, 0, (int) raFile.length());
        //System.err.println("    read " + raFile.length() + " bytes.");
        buf.setOffset(0);
        buf.setLength((int) raFile.length());
        buf.setFormat(format);
        buf.setFlags(buf.getFlags() | buf.FLAG_KEY_FRAME);
        // Close the random access file.
        raFile.close();
    }
    /**
     * Return the format of each video frame. That will be JPEG.
     */
    public Format getFormat() {
        return format;
    }

    public ContentDescriptor getContentDescriptor() {
        return new ContentDescriptor(ContentDescriptor.RAW);
    }

    public long getContentLength() {
        return 0;
    }

    public boolean endOfStream() {
        return ended;
    }

    public Object[] getControls() {
        return new Object[0];
    }

    public Object getControl(String type) {
        return null;
```

```
        }
    }
}
```

VideoToFrames.java

```
public class VideoToFrames {
    public static final double SECONDS_BETWEEN_FRAMES = 0.1;

    public static String inputName;
    public static String filePrefix;

    // The video stream index, used to ensure we display frames from one and
    // only one video stream from the media container.
    private static int mVideoStreamIndex = -1;

    // Time of last frame write
    private static long mLastPtsWrite = Global.NO_PTS;

    public static final long MICRO_SECONDS_BETWEEN_FRAMES =
            (long)(Global.DEFAULT_PTS_PER_SECOND * SECONDS_BETWEEN_FRAMES);

    /*public static void main(String[] args) {

        IMediaReader mediaReader = ToolFactory.makeReader(inputFilename);

        // stipulate that we want BufferedImages created in BGR 24bit color
space

mediaReader.setBufferedImageTypeToGenerate(BufferedImage.TYPE_3BYTE_BGR);

        mediaReader.addListener(new ImageSnapListener());

        // read out the contents of the media file and
        // dispatch events to the attached listener
        while (mediaReader.readPacket() == null) ;
    }*/

    public static class ImageSnapListener extends MediaListenerAdapter {

        int n = 0;

        public void onVideoPicture(IVideoPictureEvent event) {

            if (event.getStreamIndex() != mVideoStreamIndex) {
                // if the selected video stream id is not yet set, go ahead an
                // select this lucky video stream
                if (mVideoStreamIndex == -1)
                    mVideoStreamIndex = event.getStreamIndex();
                    // no need to show frames from this video stream
                else
                    return;
            }

            // if uninitialized, back date mLastPtsWrite to get the very first
frame
            if (mLastPtsWrite == Global.NO_PTS)
                mLastPtsWrite = event.getTimeStamp() -
```

```
MICRO_SECONDS_BETWEEN_FRAMES;

            // if it's time to write the next frame
            if (event.getTimeStamp() - mLastPtsWrite >=
                    MICRO_SECONDS_BETWEEN_FRAMES) {

                String outputFilename = dumpImageToFile(event.getImage());

                // indicate file written
                double seconds = ((double) event.getTimeStamp()) /
                        Global.DEFAULT_PTS_PER_SECOND;
                System.out.printf(
                        "at elapsed time of %6.3f seconds wrote: %s\n",
                        seconds, outputFilename);

                // update last write time
                mLastPtsWrite += MICRO_SECONDS_BETWEEN_FRAMES;
            }
        }

        private String dumpImageToFile(BufferedImage image) {
            try {
                String outputFilename = filePrefix +
                        n + ".jpg";
                ImageIO.write(image, "jpg", new File(outputFilename));
                n++;
                return outputFilename;
            }
            catch (IOException e) {
                e.printStackTrace();
                return null;
            }
        }
    }
}
```

CreateVideo.java

```
public class CreateVideo {

    public static void makeVideo (String movFile) throws MalformedURLException {

        JpegImagesToMovie imageToMovie = new JpegImagesToMovie();

        Vector<String> imgList = new Vector <String>();

        File f = new File("D:\\Diploma\\Frames\\");
        File[] fileList = f.listFiles();

        for (int i = 0; i < fileList.length; i++) {
            imgList.add(fileList[i].getAbsolutePath());
        }

        MediaLocator ml;

        if ((ml = imageToMovie.createMediaLocator(movFile)) == null) {
            System.exit(0);
        }

        imageToMovie.doIt(720, 480, (1000/100), imgList, ml);
```

```
    }
    public static void main(String[] args) throws MalformedURLException {
        try {
            makeVideo("Video.mp4");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

CreateVideo.java

```
public class Main{

    static class Frames {
        private String name;
        private String prefix;

        Frames(String name, String prefix) {
            this.name = name;
            this.prefix = prefix;
        }
        public String getName() {
            return this.name;
        }
        public String getPrefix() {
            return this.prefix;
        }

        public void makeFrames() {
            VideoToFrames mf = new VideoToFrames();

            mf.inputName = this.name;
            mf.filePrefix = this.prefix;

            IMediaReader mediaReader = ToolFactory.makeReader(mf.inputName);

            // stipulate that we want BufferedImages created in BGR 24bit color
space

mediaReader.setBufferedImageTypeToGenerate(BufferedImage.TYPE_3BYTE_BGR);

            mediaReader.addListener(new VideoToFrames.ImageSnapListener());

            // read out the contents of the media file and
            // dispatch events to the attached listener
            while (mediaReader.readPacket() == null) ;
        }
    }
    public static void makeFrames(String name, String prefix) {
        VideoToFrames mf = new VideoToFrames();

        mf.inputName = name;
        mf.filePrefix = prefix;

        IMediaReader mediaReader = ToolFactory.makeReader(mf.inputName);

        // stipulate that we want BufferedImages created in BGR 24bit color
```

```
space

mediaReader.setBufferedImageTypeToGenerate(BufferedImage.TYPE_3BYTE_BGR);

        mediaReader.addListener(new VideoToFrames.ImageSnapListener());

        // read out the contents of the media file and
        // dispatch events to the attached listener
        while (mediaReader.readPacket() == null) ;
    }

    public static void main(String[]args) throws InterruptedException {
        //makeFrames("D:/Diploma/B_Left.avi","D:/Diploma/Frames_L/LFrame_");
        makeFrames("D:/Diploma/Bryansk_Left.avi","D:/Diploma/Frames_L/LFrame_");
        //makeFrames("D:/Diploma/B_Right.avi","D:/Diploma/Frames_R/RFrame_");

//makeFrames("C:/Users/alcob/Dropbox/Right.avi","D:/Diploma/Frames_R/RFrame_");

makeFrames("C:/Users/alcob/Dropbox/Left.avi","D:/Diploma/Frames_L/LFrame_");

        /*Frames right = new
Frames("D:/Diploma/Bryansk_Left.avi","D:/Diploma/Frames_L/LFrame_");
        new Thread(new Runnable() {
            public void run() { right.makeFrames(); }
        }).start();
        Frames left = new
Frames("D:/Diploma/Bryansk_Left.avi","D:/Diploma/Frames_L/LFrame_");
        new Thread(new Runnable() {
            public void run() { left.makeFrames(); }
        }).start();*/
    }
}
```