МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт ИРИТ

Направление подготовки 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы и сети
(наименование)

Кафедра Вычислительные системы и технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

бакалавра
(бакалавра, магистра, специалиста)

Студента Густякова А.П. группы 14-В-1
(Ф.И.О.)

на тему Программная система детектирования вредоносного воздействия на камеру
(наименование темы работы)

СТУДЕНТ:

_____ Густякова А.П.
(подпись) (фамилия, и., о.)

_____
(дата)

РУКОВОДИТЕЛЬ:

_____ Гай В.Е.
(подпись) (фамилия, и., о.)

_____
(дата)

РЕЦЕНЗЕНТ:

_____ _____
(подпись) (фамилия, и., о.)

_____
(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

_____ Кондратьев В.В.
(подпись) (фамилия, и.о.)

_____
(дата)

КОНСУЛЬТАНТЫ:

1. По _____

_____ _____
(подпись) (фамилия, и., о.)

_____
(дата)

2. По _____

_____ _____
(подпись) (фамилия, и., о.)

_____
(дата)

3. По _____

_____ _____
(подпись) (фамилия, и., о.)

_____
(дата)

ВКР защищена _____
(дата)

протокол № _____

с оценкой _____

Оглавление

Введение

Задача детектирования вредоносного воздействия на камеру является одной из наиболее важных задач в области компьютерного зрения на сегодняшний день. В современном мире существует большое количество различных мест скопления людей, таких как торговые центры, аэропорты, общественный транспорт, которые оснащены камерами видеонаблюдения. Одновременное наблюдение за изображениями со всех камер в течение длительного времени является достаточно сложной задачей, даже для наиболее опытного персонала службы безопасности. Поэтому, очевидно, что введение автоматизации в данный процесс может значительно улучшить эффективность системы наблюдения. Отсюда можно сделать вывод о том, что разработка программной системы детектирования негативного воздействия на камеру является достаточно актуальной.

1. Постановка задачи

1.1 Назначение разработки и область применения

Разрабатываемая система предназначена для детектирования вредоносного воздействия на камеру.

Данная система может иметь применения для следующих случаев:

1) В местах, где существует вероятность внедрения в работу камер. Для этого требуется уведомление сотрудников службы безопасности о произошедшем воздействии на камеру.

2) Система может использоваться для проверки работоспособности камеры, для выявления правильности ее установки. Например, неподходящим местом установки камеры является место, в котором угол обзора камеры во время ветра загораживается ветками дерева.

Система может использоваться на портативных и стационарных компьютерах.

1.2 Технические требования

Разрабатываемая система рассчитана на работу на следующих ЭВМ:

1) Операционные системы MS Windows, начиная с версии XP. Аппаратное обеспечение ЭВМ должно поддерживать работу соответствующей ОС.

2) ЭВМ поддерживает взаимодействие с пользователем через монитор, клавиатуру и мышь.

Функциональные требования к системе следующие:

1) Пользователь системы имеет возможность выбрать файл с видео из директории, используя графический интерфейс программы.

2) Программа выполняет анализ видео, разбивая его на фреймы и проводя вычисления по сравнения кадров.

3) По завершению определенного периода времени, программа выводит пользователю результат о том, было ли совершено воздействие или нет.

2. Анализ поставленной задачи

2.1 Выбор операционной системы

Важным этапом в разработке ПО является определение целевой операционной системы. Рассмотрим наиболее распространенные варианты.

1)Windows – семейство операционных систем для персональных компьютеров и рабочих станций. *OS Windows* доминирует в мире персональных компьютеров, предлагая графический интерфейс пользователя, управление виртуальной памятью, многозадачность и поддержку многих периферийных устройств.

2)Mac OS-официальное название операционной системы *Apple Macintosh. Mac OS* имеет графический интерфейс пользователя, который использует окна, значки и все приложения, которые работают на компьютере *Macintosh*, имеют аналогичный пользовательский интерфейс.

3)Linux – свободно распространяемая операционная система с открытым исходным кодом, которая работает на нескольких аппаратных платформах. Ядро *Linux* было разработано главным образом Линусом Торвальдсом и оно основано на *Unix*.

По данным Интернет-ресурса *Statcounter* [3] на июль 2017 года, в основном на персональных компьютерах в мире используются следующие операционные системы:

- *Windows 7 (48.91%)*
- *Windows 10 (27.63%)*
- *Windows 8.1 (6.48%)*
- *Windows XP (6.10%)*
- *Mac OS X 10.12 (3.52%)*

- *Linux (2.53%)*
- *Windows 8 (1.42%)*
- *Mac OS X 10.11 (1.17%)*
- *Mac OS X 10.10 (0.76%)*

Таким образом, семейство *OS Windows* представляет собой наиболее используемые операционные системы среди пользователей персональных компьютеров. Большим преимуществом операционных систем *Windows* является то, что существует большое количество программного обеспечения, рассчитанного на работу на данной ОС, что предоставляет разработчику большой набор для выбора среды написания программ.

Поэтому было принято решение выбрать целевой операционной системой *Windows OS.*

2.2 Выбор языка программирования

Для разработки под *Windows* существует большой выбор языков программирования. Выбор конкретного языка должен зависеть от цели разработки и навыков разработчика. Следующие языки наиболее востребованы для разработки *Windows*-приложений:

1) C – структкрно-ориентированный язык программирования, в основном используется для разработки низкоуровненого ПО. С применятся в системном программировании, искусственном интеллекте, промышленной автоматизации, компьютерной графике, обработке изображений, разработке игр. Используется для разработки системных приложений, интегрированных в операционные системы, такие как *Windows, UNIX* и *Linux*, а также встроенные программные средства. Приложения включают графические пакеты, текстовые процессоры, электронные таблицы, разработку операционной системы, системы баз данных, компиляторы и сборщики, сетевые драйверы и интерпретаторы. Язык *C* был разработан в 1972 году в *Bell Labs* специально для внедрения системы *UNIX*. В конечном итоге это привело к появлению многих современных языков программирования, включая *C ++, Java C #, JavaScript* и *Pearl.*

2)C++ - универсальный, объектно-ориентированный язык программирования среднего уровня, является расширением языка *C.* Язык *C ++* используется для создания компьютерных программ, таких как игры, офисные приложения, графические и видеоредакторы и операционные системы. *ОС Blackberry* разрабатывается с использованием *C ++.* Новый пакет *Microsoft Office* был разработан с использованием *C ++.* Особенностью *C++* является надежная стандартная библиотека *STL*, а также механизм быстрой обработки и компиляции. Выпущен в 1983 году и часто рассматривается как объектно-ориентированная версия языка *C, C ++* был

создан для компиляции скудного и эффективного кода, обеспечивая при этом абстракции высокого уровня для лучшего управления крупными проектами развития.

3)Python – продвинутый интерпретируемый язык программирования, объектно-ориентирован и построен на гибкой и надежной семантике. Используется для *back-end* разработки, разработки баз данных, для научных и числовых вычислений, обработки изображений, разработки веб-сайтов и графического интерфейса. Применяется в таких компаниях как *Google, Pinterest, Instagram, YouTube, DropBox, NASA, ESRI*. *Python* позволяет быструю работу для интеграции систем в качестве скриптового языка. Особенностями *Python* являются лёгкость чтения, свободно распространяемый интерпретатор и стандартная библиотека, доступные в исходном коде. *Python* был разработан в конце 1980-х годов в *CWI* в Нидерландах и впервые выпущен для публики в 1991 году.

4)C# - язык программирования с несколькими парадигмами, характеризуется строгой типизацией, императивной, декларативной, функциональной, общей, объектно-ориентированной дисциплинами. Используется в области информационных технологий, инженерии, дизайна, профессиональных услуг, управления и контроля качества. Основные организации: *Microsoft, Intel, Hewlett Packard*. Специализации: платформы на базе *Windows. C #* помогает разработчикам создавать веб-службы *XML* и приложения *Microsoft .NET* для *ОС Windows* и сети.

5)Java – универсальный, объектно-ориентированный, высокоуровневый язык программирования. Используется работодателями в области коммуникаций, образования, финансов, здравоохранения, гостеприимства, розничной торговли. Основные организации: *V2COM, Eclipse Information Technologies, eBay, Eurotech*. Специализации и отрасли:

Интернет вещей (*IoT*), корпоративная архитектура, облачные вычисления. *Java* используется для разработки приложений уровня предприятия для видеоигр и мобильных приложений, а также для создания веб-приложений с *JSP* (страницами сервера *Java*). При использовании в Интернете *Java* позволяет загружать и использовать апплеты через браузер, который затем может выполнять функцию, которая обычно недоступна. Программы, которые используют или написаны на *Java*, включают в себя *Adobe Creative Suite, Eclipse, Lotus Notes, Minecraft* и *OpenOffice. Java* является основной основой для разработки приложений для *Android*. Особенности языка это: переносимость приложений, надежный и интерпретируемый язык, обширная сетевая библиотека. *Java* была основана в 1990 году компанией *Sun Microsystems*, чтобы добавить возможности языку *C ++*. *Java* была разработана в соответствии с принципом *WORA (Write Once Run Anywhere).* Язык был представлен общественности в 1995 году и теперь принадлежит *Oracle*.

6)R - это язык и среда для статистических вычислений и графики. Это проект *GNU*, который похож на язык *S* и среду, разработанную в *Bell Laboratories* (ранее *AT & T*, теперь *Lucent Technologies*) Джоном Чемберсом и его коллегами. R можно рассматривать как различную реализацию *S*. Есть некоторые важные отличия, но много кода, написанных для *S*, не изменяется в *R*. *R* предоставляет широкий спектр статистических (линейное и нелинейное моделирование, классические статистические тесты, анализ временных рядов, классификация, кластеризация) и графические методы и является весьма расширяемым. Язык *S* часто является средством выбора для исследований в области статистической методологии, а *R* предоставляет маршрут с открытым исходным кодом для участия в этой деятельности.

Для разработки системы был выбран язык программирования *Python*, так как данный язык поддерживает библиотеки для работы с изображениями,

показывает высокую скорость вычислений и предоставляет возможности для создания графических интерфейсов.

2.3 Выбор среды разработки

Правильный подход к выбору среды разработки – важный этап при решении задач для программиста. Грамотно выбранная *IDE*, соответствующая предпочтениям и целям, значительно повышает эффективность работы разработчика. Наиболее используемыми средами разработки для *Python* являются следующие:

1)Spyder - межплатформенная среда разработки с открытым исходным кодом. Включает в себя основные библиотеки для обработки больших наборов данных, такие как*: NumPy, SciPy, Matplotlib* и *IPython*, кроме того, он может быть расширен с помощью плагинов. *Spyder* предлагает интерактивную справку, которая позволяет искать конкретную информацию о библиотеках. Особенности *Spyder* заключаются в следующих функциях: это текстовый редактор с подсветкой синтаксиса, завершение кода и исследование переменных, которые можно редактировать значениями с помощью графического интерфейса пользователя.

2)PyCharm - это *IDE*, созданная компанией *JetBrains*, ответственной за одну из самых известных *Java IDE - IntelliJ IDEA*. *PyCharm* интегрирует инструменты и библиотеки, такие как *NumPy* и *Matplotlib*, что позволяет работать с обзором массивов и интерактивными графиками. Как и другие *IDE*, *PyCharm* имеет интересные функции, такие как редактор кода, подсветку ошибок, мощный отладчик с графическим интерфейсом, помимо интеграции *Git*, *SVN* и *Mercurial*. Кроме того, можно расширить возможности *PyCharm*, добавив плагины.

3)Rodeo - *IDE*, которая была специально разработана для анализа данных. *Rodeo* очень похож на *RStudio*, который на сегодняшний день является самой популярной средой *IDE* для языка *R*, не только из-за ее внешнего вида, но и функций: например, при запуске *Rodeo*, окно *IDE*

разделено на четыре группы: текстовый редактор, консоль, среда для визуализации переменных и графика / библиотеки / файлы, так же как и в *RStudio*. Основным преимуществом этой *IDE* является ее настройка. В ней можно легко проверять значения переменных и взаимодействия с графиками и кадрами данных. Кроме того, *IDE* поставляется с автозаполнением, подсветкой синтаксиса и поддержкой *IPython*, что помогает писать код лучше и быстрее.

4)Atom - текстовый редактор с открытым исходным кодом, разработанный *Github*. У *Atom* есть интересные функции, которые создают хороший опыт для разработчиков *Python*. Одним из лучших преимуществ *Atom* является его сообщество, главным образом благодаря усовершенствованиям констант и плагинам, которые они разрабатывают для настройки *IDE* и улучшения рабочего процесса. В Atom есть возможность визуализировать свои результаты, не открывая дополнительнх окон. Кроме того, есть плагин под названием *«Markdown Preview Plus»*, который предоставляет встроенную поддержку для редактирования и визуализации файлов *Markdown* и который позволяет открыть предварительный просмотр, визуализировать уравнения *LaTeX* и многие другие функции. К недостаткам *Atom* можно отнести слабую производительность на старых процессорах.

5)Jupyter Notebook - веб-приложение, основанное на архитектуре клиент-сервер и позволяющее создавать и обрабатывать документы. *Jupyter Notebook* предоставляет удобную интерактивную среду для научных исследований на многих языках программирования, которая работает не только как среда разработки, но и как инструмент презентации или образования. Благодаря *Jupyter* можно легко видеть и редактировать свой код, чтобы создавать интересные презентации. Например, можно использовать библиотеки визуализации данных, такие как *Matplotlib* и *Seaborn*, и показывать графики в том же документе, где находится код.

Помимо всего этого можно экспортировать работу в файлы *PDF* и *HTML*, или как *.py*-файл.

Для разработки программной системы была выбрана среда *PyCharm*, так как данная среда располагает удобным отладчиком, отлично интегрируется с *git* и легко работает с использованием нескольких версий *Python*. Также есть возможность использовать свободно распространяемую версию среды – *PyCharm Community* с открытым исходным кодом.

2.4 Обзор существующих способов детектирования воздействия на камеру

Различные исследования проводились в отношении выявления вредоносного воздействия на камеру. Например, работы [4],[9] и [10] посвящены технике «водяных знаков» для обнаружения искажений на изображениях. Техника «водяных знаков» заключается в выявлении искажающих изменений, проявившихся при обработке изображений, с помощью сравнения коррелированных значений из областей изображения. Данные работы рассматривают важные аспекты обнаружения дефектов на изображениях, однако они не являются напрямую связанными с проблемой фальсификации данных камеры, так как целью исследования является создания системы, контролирующей данные с камеры в реальном времени.

Некоторые работы посвящены оценке движения на видео. Так, в работе [11], используется метод факторизации используется для оценки структуры движения. В работе [7] для этой же цели используется метод наименьших квадратов. Эти работы являются наиболее значимыми в этой области. Данные методы и аналогичные им полезны при выявлении фальсификации данных в тех случаях, когда человек изменяет направление камеры на другой угол обзора. Однако обе работы не рассчитаны на применение в условиях изменчивости среды. Также, в задаче детектирования негативного влияния не так важна оценка движения на видео, сколько фиксирование события вмешательства в работу камер.

Для того чтобы выявить иные способы влияния на достоверность данных видеонаблюдения, такие как, например, попытка чем-либо закрыть объектив камеры или подмена изображения существует несколько алгоритмов.

Например, в работе [8] особое внимание уделяется рассмотрению переходов в видеоизображении с помощью интерполяцией B-сплайнами. Однако здесь подход рассчитан на медленные переходы в видеоряде, что не всегда будет иметь отношение к фальсификации.

В работе [5] авторы моделируют каждый снимок как набор граней на изображении, и отслеживают их изменения, делая снимок при резком изменении граней. Этот метод также недостаточно точен в определении вредоносного влияния, так как, например затемнение/загораживание объектива может происходить постепенно, без резких смен границ.

В работе [8] авторы пытаются определить изменения в изображениях видеоряда с камеры с помощью использования алгоритма, основанного на анализе цветовых гистограмм для выявления значительных изменений картины. Аналогичным способом в работе [6] используются цветовые гистограммы вместе со значениями пикселей изображения для нахождения различий между изображениями. Методы, основанные на использовании гистограмм, хорошо подходят для обнаружения фальсификации данных с камер, так как они предлагают глобальные измерения содержимого изображения и чувствительны к большинству типов изменений в изображении независимо от их характера или природы.

Таким образом, алгоритмы обнаружения негативного влияния на камеру должны быть чувствительны к любым значительным движениям на камере. Количество ложных обнаружений фальсификации должны быть сведено к минимуму, для того чтобы не отвлекать службы безопасности каждый раз, когда событие обнаружено. Важно отметить, что негативное воздействие на камеру является устойчивым событием, которое может происходить как быстро, так и плавно, и, следовательно, необходимо

производить сложные и правильные сравнения между кадрами видео, для обнаружения фальсификации.

2.5. Выбор подхода к решению задачи детектирования вредоносного воздействия на камеру

В общем смысле все известные подходы к детектированию влияния на камеру основаны на следующих этапах:

- Предварительная обработка изображения – изображения видеоряда представляются в подходящем виде для их последующей обработки.
- Построение признакового описания объекта – формируется набор признаков, позволяющий однозначно описать объект.
- Принятие решения – негативное воздействие произошло/не произошло.

В данной работе для решения задачи детектирования вредоносного воздействия на камеру было решено использовать теорию активного восприятия. В подходе, применяющем TAB, признаковое описание объекта производится с помощью нахождения ключевых точек. Определение этих точек позволяет представить объект в виде набора особых точек, которые свойственны данном объекту и могут быть в последствии использованы при принятии решения о воздействии на камеру. Представление изображений в виде набора ключевых точек позволяет сократить размер его описания и соответственно увеличить скорость обработки.

Схема, описанных выше этапов представлена на рисунке:

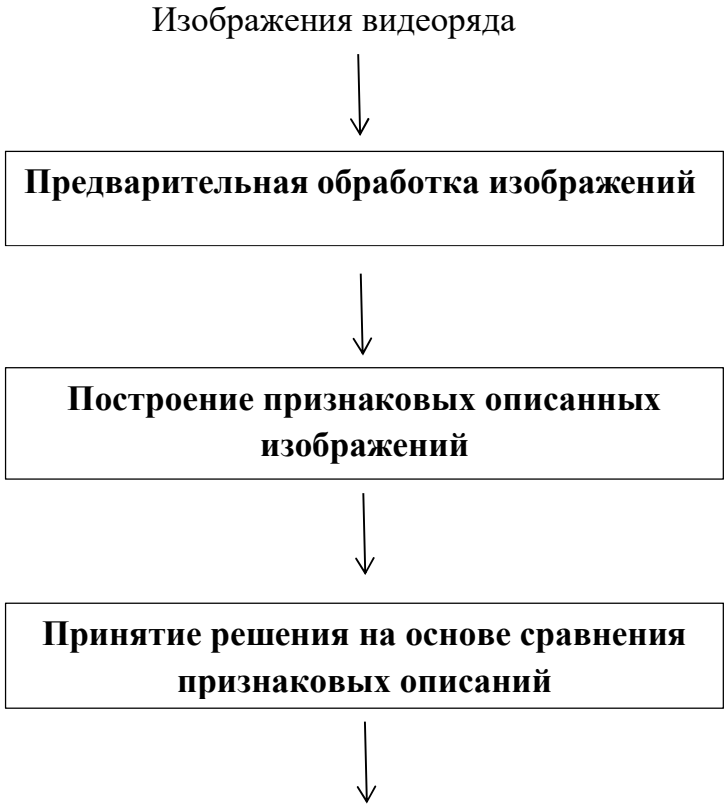

Рисунок 2.1. Схема этапов решения задачи

Теория активного восприятия была разработана Владимиром Александровичем Утробиным, профессором НГТУ им. Р.Е. Алексеева. Данная теория используется для формирования признакового описания и последующей обработки и анализа изображений. Теория активного восприятия базируется на операции *U* преобразования, применяемой к исследуемому изображению. *U* преобразование заключается в последовательном применении к изображению операций интегрирования и

дифференцирования. В результате этого формируется набор спектральных коэффициентов, который и является признаковым описанием изображения. На основе полученных спектральных коэффициентов формируется набор ключевых точек. На этапе принятия решения производится сравнение ключевых точек изображений видеоряда. Для большей точности, каждое изображение разбивается на несколько областей.

Во время видеосъемки производятся следующие вычисления: все кадры проходят предварительную обработку, вычисляются их спектральные коэффициенты, формируются наборы ключевых точек. Далее производится сравнение ключевых точек по их координатам. В «идеальном» случае – когда на камеру не оказано никакого воздействия – все точки будут оставаться на своих местах на протяжении всех кадров. Однако, в случае воздействия, будет иметь место сильное различие между набором ключевых точек в момент воздействия от того набора, что имел место до этого. В момент воздействия ключевые точки могут либо пропасть вовсе (закрыли объектив, подставили изображение), резко сменить координаты (изменили угол обзора), частично поменяться (закрыли часть объектива). На основе анализа координат ключевых точек производится принятие решения о совершении воздействия на камеру.

В данной работе точка изображения считается ключевой, если в области, в которой находится эта точка, имеется резкий перепад яркости. Признаковое описание областей изображения, состоящее из спектральных коэффициентов *U*-преобразования, позволяет легко выявлять области, содержащие перепады яркости. Критерий, по которому определяется, является ли точка ключевой выглядит следующим образом: $s_{ij} > k \times s_{max}$, где s_{ij} – среднеквадратичное отклонение коэффициентов спектрального представления области.

Отсюда следует, что область изображения содержит ключевую точку объекта, если среднеквадратичное отклонение спектральных коэффициентов этой области больше, чем максимальное среднеквадратичное отклонение среди всех областей рассматриваемого объекта, умноженное на коэффициент k. В данной работе оптимальным значением коэффициента k является 0.6, так как увеличение коэффициента отбора k приводит к уменьшению количества ключевых точек, выявленных на рассматриваемом объекте. Таким образом, выявление ключевых точек объекта позволяет существенно сократить число областей в признаковом описании. Это приводит к значительному снижению затрат памяти и вычислительной мощности на этапе составления признакового описания кадров.

3. Разработка структуры системы детектирования вредоносного воздействия на камеру

3.1 Разработка общей структуры системы

В соответствии с теорией активного восприятия, общая структура должна включать в себя следующие этапы:

- разработка алгоритма предварительной обработки данных
- формирование системы признаков

Результат предварительной обработки данных представляет собой матрицу визуальных масс, в используемой теории это определяется как *Q*-преобразование.

Результатом формирования системы признаков является вектор спектральных коэффициентов. Для данного этапа используется набор из 16 фильтров (Рисунок. 3.1). Затемненный элемент соответствует значению «-1», в то время как светлому соответствует «1».

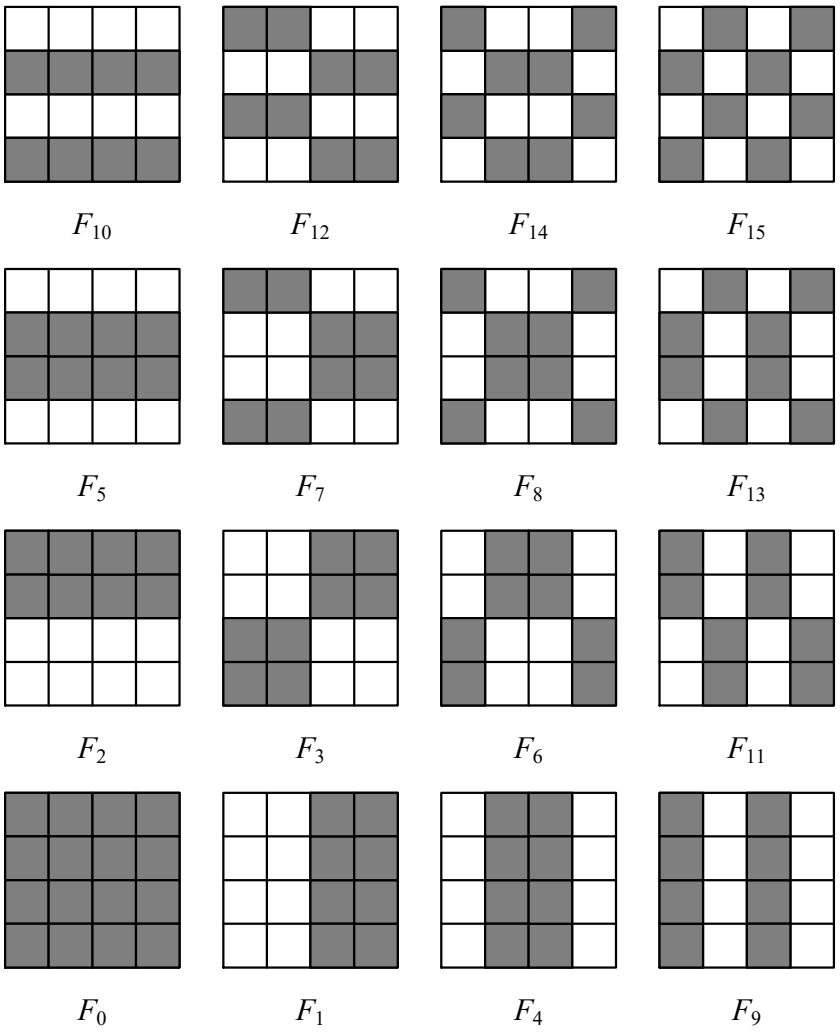

Рисунок 3.1. Фильтры

Совокупность *Q*-преобразования и вектора спектральных коэффициентов в теории активного восприятия называется *U*-преобразованием (рис.3.2).

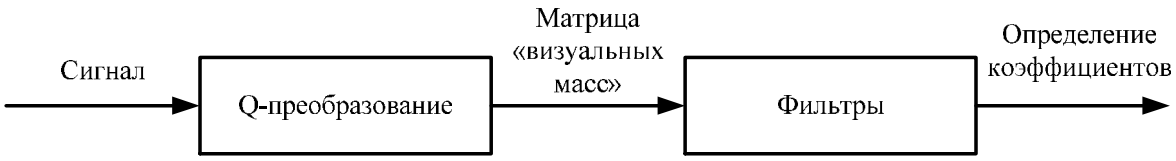

Рисунок 3.2. *U*-преобразование

Итоговая программа будет содержать в себе следующие блоки:

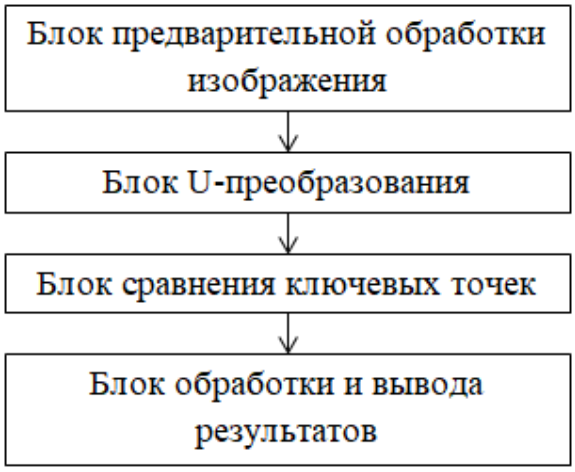

Рисунок 3.3. Структура системы

Рассмотрим в отдельности каждый блок:

1)Блок предварительной обработки включает в себя операции по нормированию изображения, представлению его в виде матрицы каналов изображения.

2)Блок *U*-преобразования заключается в получении матрицы спектральных коэффициентов для каждой подобласти нахождения среднеквадратических отклонений коэффициентов векторов.

3)Блок сравнения ключевых точек предназначен для вычислений, определяющих вмешательство в работу камеры. Для этого формируется таблица из ключевых точек, выявленных у обработанных кадров, и производится их сравнение с набором ключевых точек стартового кадра. Далее происходит оценка частоты появления каждой точки на кадрах, после чего отбирается процент точек, частота которых выше порогового значения. Пороговое значение вычисляется в ходе экспериментов и означает допустимый процент отсутствия точки в некоторых кадрах.

4)В блоке вывода результатов производится общая оценка присутствия ключевых точек в кадрах и выводится соответствующий результат о том, было ли произведено вмешательство в работу камеры или нет.

3.2 Разработка алгоритма вычисления особых точек изображений

Последовательность действий по вычислению особых точек изображений изображена на рисунке 3.4.

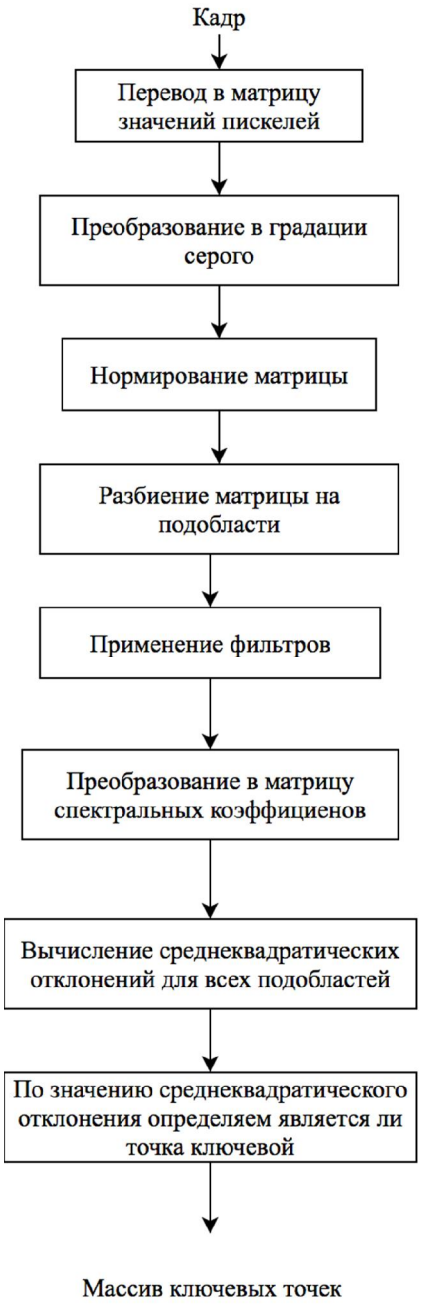

Рисунок 3.4 Алгоритм нахождения ключевых точек

3.3 Разработка алгоритма принятия решения

Для определения вмешательства в работу камеры, необходимо проводить сравнение ключевых точек. Рассмотрим алгоритм по принятию решения о воздействии на камеру см. рисунок. 3.5.

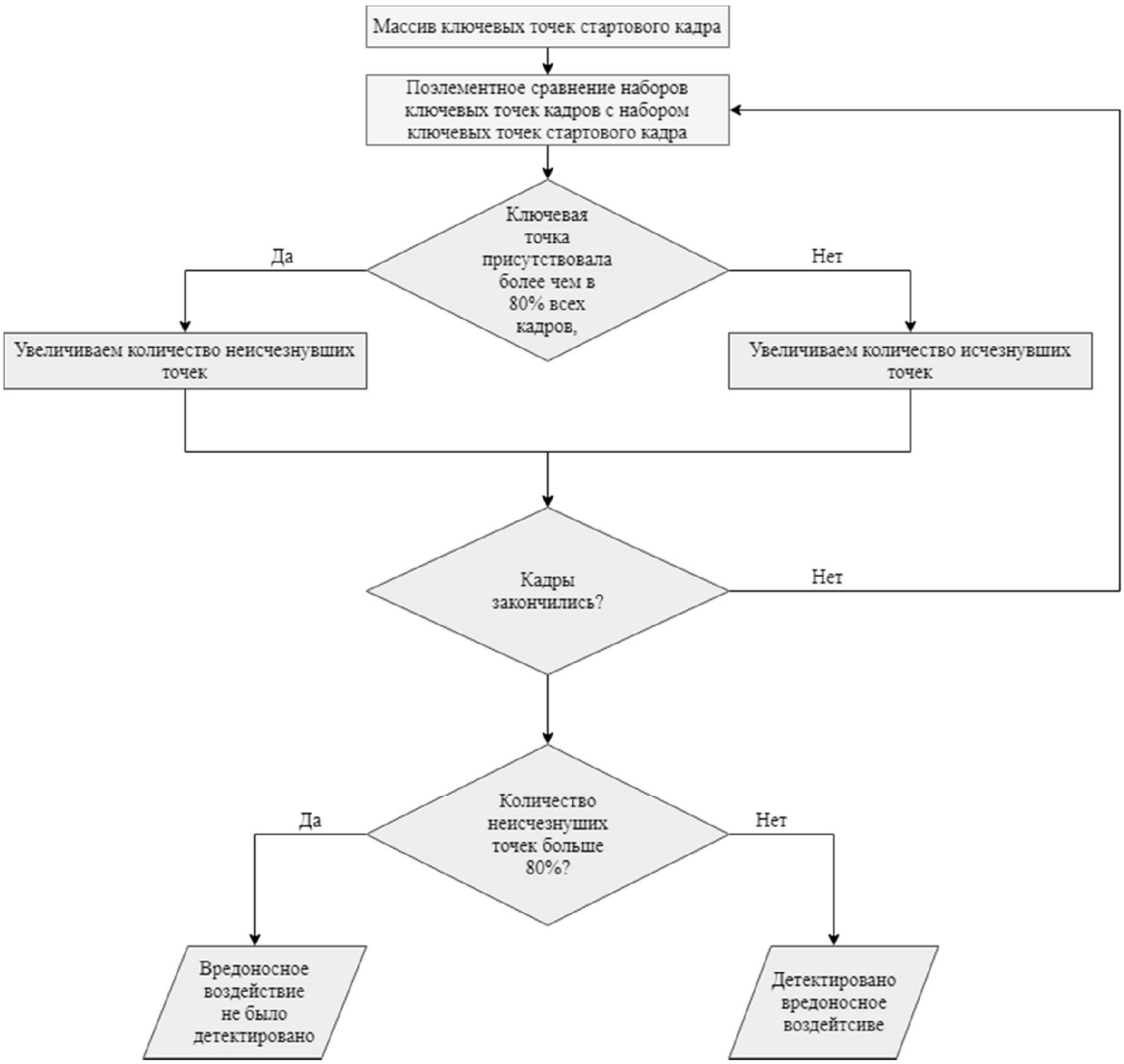

Рисунок 3.5 Алгоритм принятия решения

Необходимо выявить воздействие на камеру следуя данному алгоритму:

1)Находим массив ключевых точек первого кадра

2)Находим массивы ключевых точек для остальных кадров и проводим их поэлементное сравнение с первым массивом.

3)Для каждой точки из первого массива вычисляем процент присутствия в остальных массивах, следовательно, кадрах. Если это значение больше 80%, то в массив для вычисления результата заносим значение логической единицы.

4)Проверяем количество значений в массиве результата, если занесено более 80% значений *TRUE*, значит, воздействия не было, большинство ключевых точек сохранили свои позиции. Иначе детектируем вредоносное воздействие на камеру.

4. Разработка программных средств

4.1. Разработка интерфейса пользователя системы

Графический интерфейс программы был реализован на языке *Python* с помощью библиотеки *Tkinter*. *Tkinter* представляет собой стандартную *Python* библиотеку для разработки интерфейса. Данная библиотека свободно распространяется, поэтому является одной из наиболее простых и удобных для разработки *GUI*.

Tkinter поддерживает 15 основных виджетов:

1) *Button* - простая кнопка, используемая для выполнения команды или другой операции.

2) *Canvas* - структурированная графика. Этот виджет может использоваться для рисования графиков, создания графических редакторов и для реализации пользовательских виджетов.

3) *Checkbutton* -представляет переменную, которая может иметь два разных значения. Нажатие кнопки переключает между значениями.

4) *Entry* - поле ввода текста.

5) *Frame* - виджет контейнера. Фрейм может иметь границу и фон и используется для группировки других виджетов при создании приложения или макета диалога.

6) *Label* - отображает текст или изображение.

7) *Listbox* - отображает список альтернатив.

8) *Menu* - панель меню. Используется для реализации раскрывающихся меню и всплывающих меню.

9) *Menubutton* - меню. Используется для реализации выпадающих меню.

10) *Message* - отображает текст. Подобно виджету ярлыков, но может автоматически переносить текст в заданную ширину или соотношение сторон.

11) *Radiobutton* - представляет одно значение переменной, которое может иметь одно из нескольких значений. Нажатие кнопки устанавливает переменную в это значение и очищает все другие radiobutton, связанные с одной и той же переменной.

12) *Scale* - позволяет установить числовое значение, перетащив «ползунок».

13) *Scrollbar* - полоса прокрутки.

14) *Text* - форматированный текстовый дисплей. Позволяет отображать и редактировать текст с различными стилями и атрибутами. Также поддерживает встроенные изображения и окна.

15) *Toplevel* - виджет контейнера отображается как отдельное окно верхнего уровня

При разработке интерфейса программы использовались следующие элементы: *Button, Label, Frame*. Программная реализация *GUI* представлена на рисунке. 4.1.

```
def ui():
    global root
    root.title("Детектирование")
    root.geometry("500x200")
    root.configure(background="light gray")
    Label(root, text="Добро пожаловать в программу\n детектирования
    воздействия на камеру! ",bg="light gray", font = "Verdana 10 italic").pack
    ()
    Label(root, text="\n Для начала работы загрузите видео.", fg = "dark gray"
    ,bg="light gray", font = "Verdana 10").pack()
    Label(root, text="\n Как только воздействие будет обнаружено,\n появится
    соответствующее окошко.",
          fg = "red",bg="light gray", font = "Verdana 10").pack()
    Button(root, text = 'Загрузить видео', command = openFolder).place(x = 0,
    y = 150, width = 500, height = 50)
    root.mainloop()
```

Рисунок 4.1 Программная реализация интерфейса

Рассмотрим программные составляющие интерфейса

- *root* – переменная базового класса *Tkinter* приложения
- *root.geometry* задает параметры окна приложения
- *root.configure* используется для настройки цветовых параметров окна.
- *Label* элементы применяются для сообщения пользователю информации.
- *Button* – кнопка, по нажатию которой пользователь может загрузить видео.

С помощью функции *mainLoop()* запускается цикл обработки событий.

Событием в данном случае является нажатие кнопки «Загрузить видео».

Обработчик кнопки – функция *openFolder(),* см. рисунок 4.2.

```
def openFolder():
    root.withdraw()
    root.filename = filedialog.askopenfilename(initialdir =
    "C:\\Users\\Anastasia\\Desktop\\detector",title = "Выберите файл",
                                               filetypes = (("mp4 files",
                                               "*.mp4"),("all files","*.*")))
    global path
    path = root.filename
    createFrames()
    main()
```

Рисунок 4.2 Обработчик нажатия кнопки «Загрузить видео»

Также в программе для взаимодействия с пользователем предназначены диалоговые окошки, сообщающие о следующих событиях:

- Загрузка видео
- Начало обработки видео
- Результат обработки
- Окончание обработки

Данные окошки реализованы с помощью модуля *tkinter.messagebox* следующим способом, см. рисунок 4.3.

```
messagebox.showinfo("Сообщение", "Видео успешно загружено. Обработка
началась.")
```

Рисунок 4.3 Пример реализации диалогового окна

При выводе результата также используется *messagebox*, для предоставления информации о внедрении в работу камеры выводится снимок момента с помощью функции модуля *PIL.Image.show(),* см. рисунок 4.4:

```
im=Image.open(list[minIndex+1])
im.show()
messagebox.showinfo("Сообщение", "Детектировано вредоносное
воздействие на камеру.")
```

Рисунок 4.4 Вывод результата

4.2. Программная реализация модулей системы

4.2.1. Модуль разбиения входного видео на фреймы.

Для обработки видео необходимо представить его в видео набора фреймов. Для этого был реализован метод *createFrames(),* см. рисунок 4.5.

```
def createFrames():
    global duration
    vidcap = cv2.VideoCapture(path)
    clip = VideoFileClip(path)
    duration = double(clip.duration)
    success, image = vidcap.read()
    global count
    success = True
    while success:
        cv2.imwrite(os.path.join('frames', "frame%d.jpg" % count), image)
        success, image = vidcap.read()
        count += 1

   messagebox.showinfo("Сообщение", "Видео успешно загружено. Обработка
   началась.")
```

Рисунок 4.5 Реализация метода *createFrames*

4.2.2. Модуль обработки фреймов.

Для начала переводим фреймы в градации серого с помощью функции grayScale, см. рисунок 4.6:

```
def grayScale(image):
    newImage = Image.new('L', image.size)
    for y in range(0, image.size[1]):
        for x in range(0, image.size[0]):
            r,g,b = image.getpixel((x,y))
            g = round((r+g+b)/3)
            newImage.putpixel((x,y),g)

    return newImage
```

Рисунок 4.6 Метод преобразования изображения в градации серого

Затем производим нормирование матриц каналов, для этого находим минимальное и максимальное значения матрицы пикселей, для нормирования используем функцию *normalize*, см. рисунок 4.7.

```
def imageToArray(source):
    source = source.load()
    width = source.size[0]
    height = source.size[1]
    massive = np.zeros((height, width))
    for x in range(width):
        for y in range(height):
            gray = source.getpixel((x, y))
            massive[y][x] = gray
    return massive

def minimum(source):
    minimum = source.min()
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            source[x][y] = source[x][y] - minimum
    return source

def maximum(source):
    maximum = source.max()
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            source[x][y] = round(source[x][y]/maximum, 1)
    return source

def normalize(image):
    massive = imageToArray(image)
    massive = minimum(massive)
    massive = maximum(massive)
    return massive
```

Рисунок 4.7 Нормирование матрицы каналов

4.2.2. Модуль выполнения *U* – преобразования

Данный модуль включает в себя функцию *Q*-преобразования, на выходе которого получаем массив матриц «визуальных масс», см. рисунок 4.8.

```
def qTransform(source):
    q = 4
    N = int(source.shape[1]/4)
    M = int(source.shape[0]/4)
    result = np.zeros((q,q))
    sum = []
    for i in range(q):
        for j in range(q):
            sum = 0
            for y in  range(i*M,(i+1)*M):
                for x in range(j*N,(j+1)*N):
                    sum = sum + source[y][x]
            result[i][j] = sum

    return result
```

Рисунок 4.8 *Q*-преобразование

С помощью операции интегрирования получаем матрицы спектральных коэффициентов, с помощью следующих функций, см. рисунок 4.9.

```
def mCreate(source, f):
    res = 0
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            res = res + source[x][y]*f[x][y]
    return round(res, 1)

def mMassCreate(source):
    mass = []
    mass.append(mCreate(source, f1))
    mass.append(mCreate(source, f2))
    mass.append(mCreate(source, f3))
    mass.append(mCreate(source, f4))
    mass.append(mCreate(source, f5))
    mass.append(mCreate(source, f6))
    mass.append(mCreate(source, f7))
    mass.append(mCreate(source, f8))
    mass.append(mCreate(source, f9))
    mass.append(mCreate(source, f10))
    mass.append(mCreate(source, f11))
    mass.append(mCreate(source, f12))
    mass.append(mCreate(source, f13))
    mass.append(mCreate(source, f14))
    mass.append(mCreate(source, f15))
    return mass
```

Рисунок 4.9 Вспомогательные функции для получения вектора спектральных коэффициентов

4.2.4. Модуль отбора ключевых точек

Все описанные выше функции используются для нахождения ключевых точек фреймов.

Реализация метода определения ключевых точек представлена на рисунке 4.10:

```
def getKeyPoints(sourceName):
    source = Image.open(sourceName)
    grey = grayScale(source)
    height = grey.size[1]
    width = grey.size[0]
    matrix = norm(grey)
    res = []
    deviations = []
    points = []
    u = []
    i=1
    k=1
    windowSize=16
    while i<(height-windowSize+1):
        j=1
        while j < (width - windowSize+1):
            windowMatrix = matrix[i:i + windowSize, j:j + windowSize]
            u = mMassCreate(q_preobr(windowMatrix))
            points.append([j,i])
            deviations.append(np.std(u))
            j = j + 4
        i=i+4
    maxDev = max(deviations)
    for x in range(0, len(deviations)):
        if deviations[x] > maxDev * 0.6:
            res.append(points[x])
    return res
```

Рисунок 4.10 Метод *getKeyPoints*

Приходящие на вход метода *getKeyPoints* изображения проходит все операции, описанные выше: после преобразования в градации серого, нормирования, в цикле просматриваем изображения, с помощью «окошек», размером 16*16 пикселей. Для каждого окошка вычисляется *U*-преобразование, в свою очередь, для результата каждого *U*-преобразования вычисляется среднеквадратическое отклонение. Все найденные отклонения попадают в массив *deviations*. После того как получены *U*-преобразования всех подобластей изображения, находится максимальное значение массива *deviations*, используемое для отбора ключевых точек. На основе утверждения, что область изображения содержит ключевую точку объекта, если среднеквадратичное отклонение спектральных коэффициентов этой области

больше, чем максимальное среднеквадратичное отклонение среди всех областей рассматриваемого объекта, умноженное на коэффициент k=0.6, производится отбор ключевых точек в массив *res*.

4.2.5. Модуль записи в таблицу сравнения

Как только для фрейма получен набор ключевых точек, производится их сравнение с набором ключевых точек начального кадра и запись соответствующего значение в таблицу *table* в методе *writeToCompare*, см. рисунок 4.11.

```
def writeTocompare(shot):
    global j
    global original
    global table
    global arr
    global a
    i=0
    table[:len(original),0]=1
    for dot in original:
        if dot in shot:
            arr[j] = arr[j]+1;
            table[i,j]=1
        i=i+1;
    j=j+1;
```

Рисунок 4.11 Метод заполнения таблицы *writeToCompare*

4.2.6. Модуль принятия решения о негативном воздействии на камеру:

Согласно алгоритму, просматриваем таблицу *table*, в которой находятся все совпадения ключевый точек каждого фрейма с ключевыми точками первого кадра. Определяем суммарное количество появлений каждой точки во всех кадрах, с помощью переменной sum и цикла for по

элементам таблицы *table*. Ключевая точка считается «неисчезнувшей», если ее суммарное появление больше или равно 80%. Затем вычисляется процентое количество всех «неизчезнувших» точек, на основе этого значения происходит принятие решения, выводится соответствующий messagebox.

```
def final():
    global table
    h, w = table.shape
    sum=0
    mass= []
    perc = 0
    true_count=0;
    false_count=0;
    for j in range (0,h):
        for i in range (0,w):
            sum=sum+table[j,i]
        mass.append(sum)
        sum=0;
    global duration
    for item in mass:
        if item >=0.4*int(duration)/1.2:
            true_count=true_count+1
        if item>0 and item<0.4*int(duration)/1.2:
            false_count=false_count+1
    result = (true_count*100/(true_count+false_count))
    print(true_count,false_count)
    print(result)
    global list
    if(result<80):
        list.pop(0)
        minIndex = list.index(min(list))
        im=Image.open(list[minIndex+1])
        im.show()
        messagebox.showinfo("Сообщение", "Детектировано вредоносное воздействие на камеру.")
    else:
        messagebox.showinfo("Сообщение", "Вредоносного воздействия на данный момент не совершалось.")
```

5. Тестирование системы

5.1 Описание набора данных

Для тестирования системы детектирования вредоносного воздействия на камеру необходимы данные с камер наблюдения, то есть таких камер, положение которых зафиксировано и угол обзора остается постоянным.

В ходе тестирования необходимо определить, что система действительно определяет негативное воздействие, такое как перекрытие объектива, изменение угла обзора. Важным моментом является выявление того, что система не реагирует на обычные движения в кадре.

Поэтому, в качестве набора данных для тестирования использовались видео с явно зафиксированным внедрением в работу камеры, а также записи с камер наблюдений, работающих в обычном режиме.

5.2. Описание методики тестирования

Проверим работоспособность программы. Для этого запустим пользовательский интерфейс и загрузим тестовое видео. Пользовательский интерфейс представлен на рисунке 5.1.

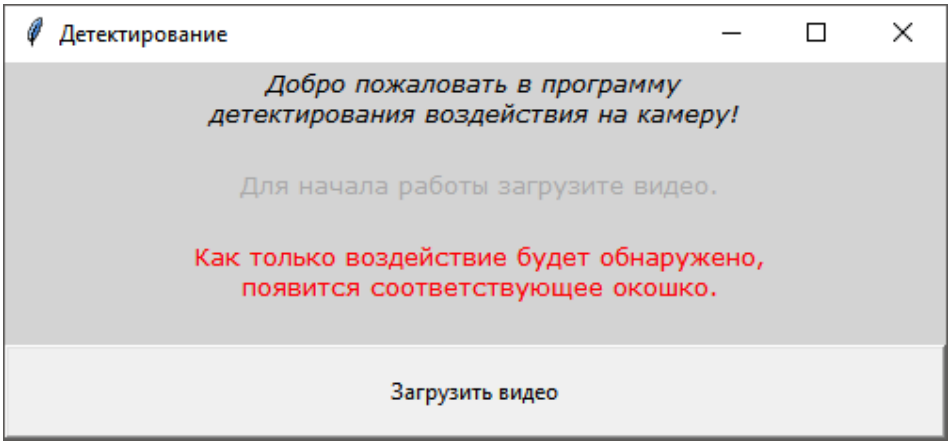

Рисунок 5.1 Пользовательский интерфейс

Для проверки работоспособности программы и действительность ее результатов были взяты следующие записи с видеокамер:

1) Запись, на которой резко поворачивают угол обзора камеры в противоположную сторону. Программа выявила данное внедрение в работу камеры, см. рисунок 5.2.

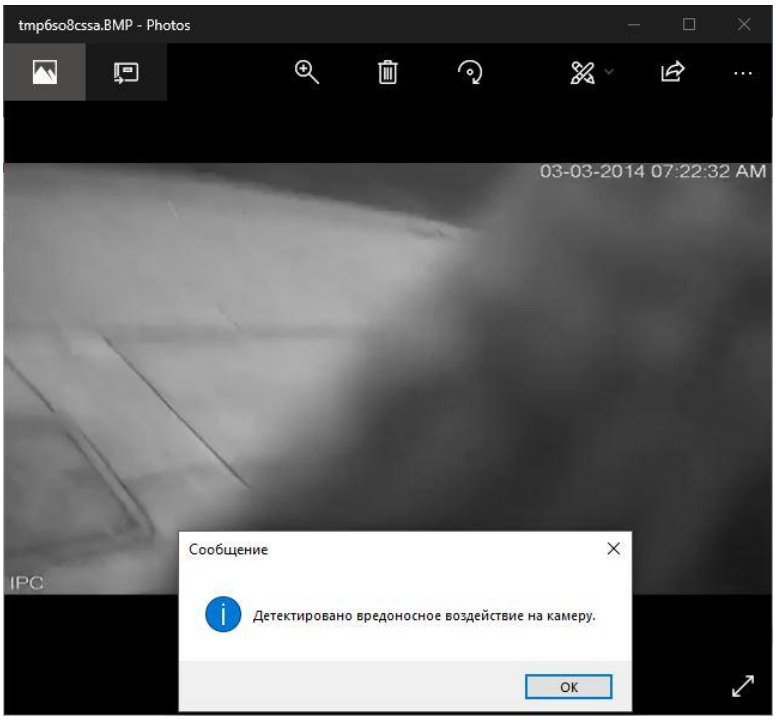

Рисунок 5.2 Внедрение детектировано

2) Запись, на которой происходит неоднократное внедрение в работу камеры, то есть было совершено некоторое внедрение в работу камеры, после чего камера работа в нормальном режиме до повторного внедрения. Задача данной программы в таком случае отследить каждое из этих воздействий. Программа действительно обнаружила оба случая, после чего завершила свою работу.

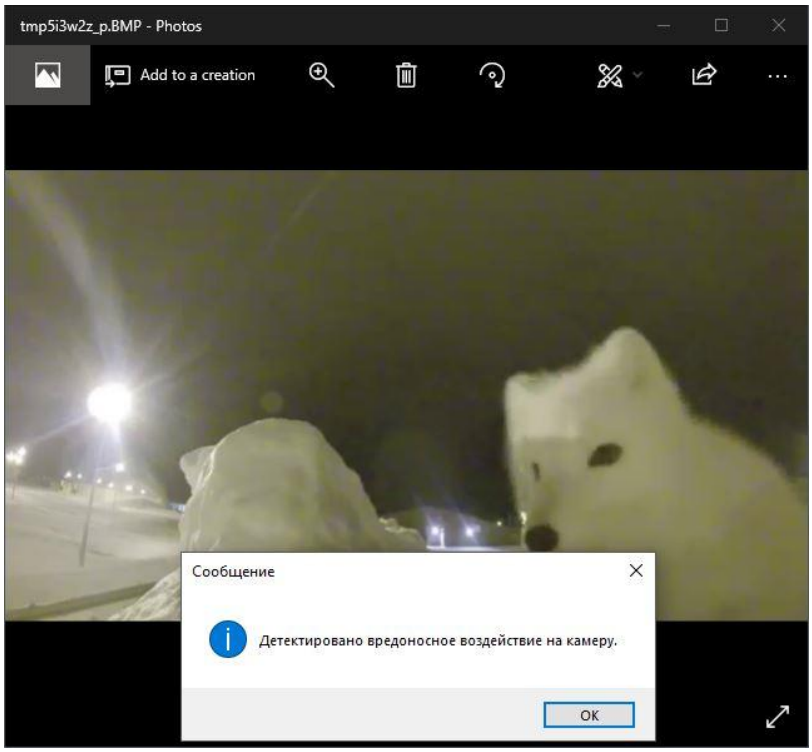

Рисунок 5.3 Детектировано внедрение в работу камеры

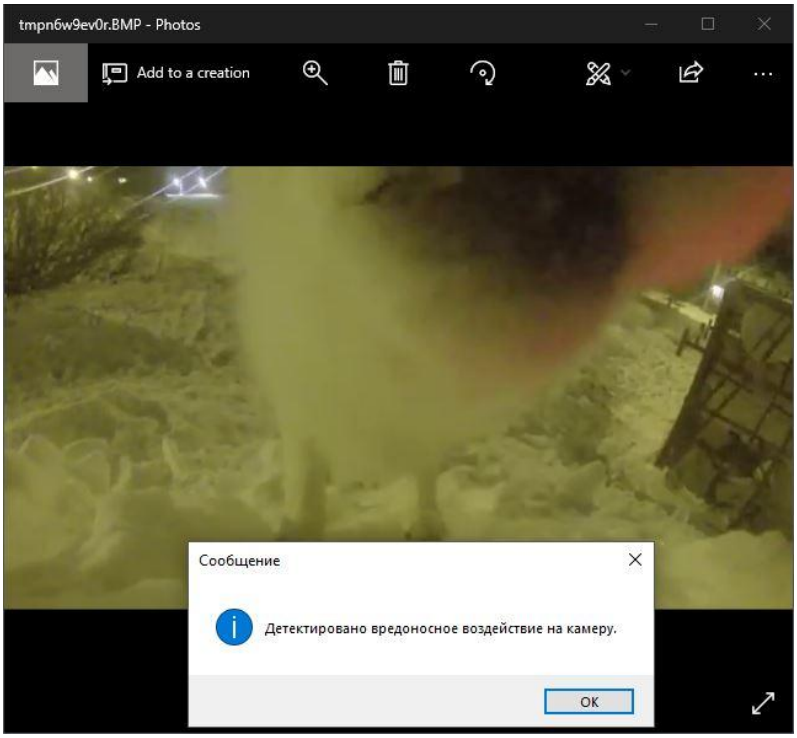

Рисунок 5.4 Детектировано повторное внедрение в работу камеры

3) Важным критерием качества работы программы является отсутствие «ложных» результатов. Таким образом, программа должна реагировать конкретно на помехи в работе камеры, и игнорировать в кадре такие действия как движение людей, машин, ветра и т. д.

Для проверки данного критерия была взята запись с уличной камеры, в кадре присутствует движение, но в работу камеры вмешательства не происходит.

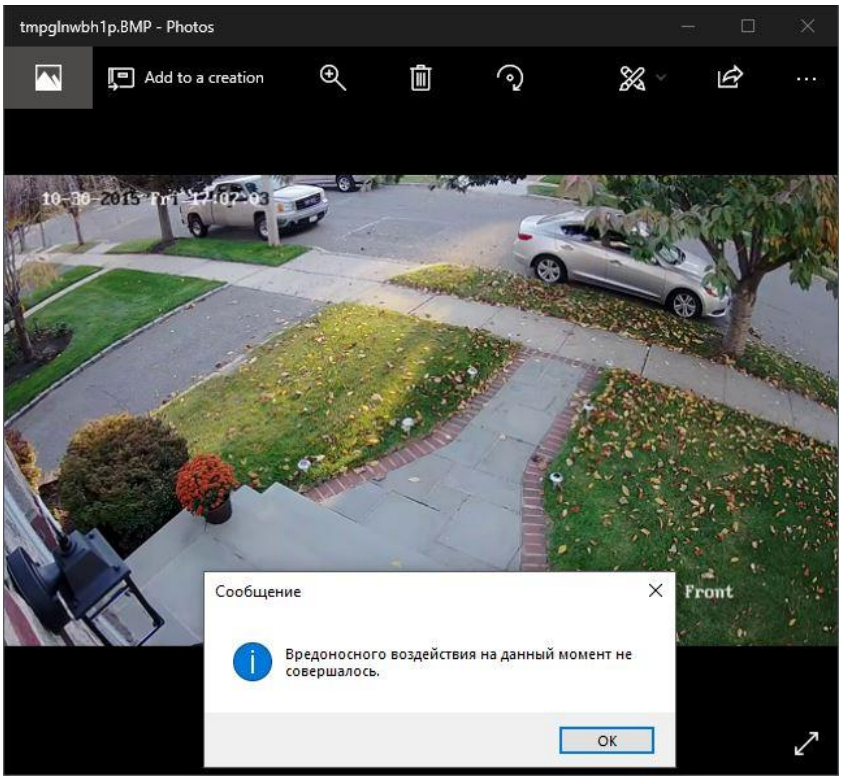

Рисунок 5.5 Работа программы при отсутствии внедрения в работу камеры

В ходе эксперимента было установлено, что при наличии в кадре вредоносного воздействия, система детектирует его с вероятностью 100%. Исходя из полученных результатов, можно сделать вывод о том, что программа устойчива к движениям в кадре, программа своевременно реагирует на изменения в работе камеры и предоставляет достоверные результаты.

Заключение

В результате выполнения выпускной квалификационной работы была спроектирована и программно реализована система детектирования негативного воздействия на камеру. Общий алгоритм работы программы был основан на теории активного восприятия.

Разработанная система предназначена для обнаружения внедрения в работу камер наблюдения. Тестирование программы подтвердило ее работоспособность и возможность дальнейшего использования для решения данной задачи.

Список литературы

1. Утробин В. А. «Компьютерная обработка изображений. Основы специальной теории восприятия» - Нижний Новгород : Нижегородский гос. технический ун-т им. Р. Е. Алексеева, 2015

2. Утробин В. А. Физические интерпретации элементов алгебры изображения // Успехи физических наук, Т. 174, № 10, 2004, С. 1089–1104.

3. *Statcounter* [Электронный ресурс] *URL: http://gs.statcounter.com/*

4. *Fridrich, J., "Image Watermarking for Tamper Detection", Proc. of ICIP 1998, vol. 2, pp. 404-408, 1998.*

5. Марк Саммерфилд. *Python* на практике. — Перевод с английского. — М.: ДМК Пресс, 2014. — 338 с. — ISBN 978-5-97060-095-5.

6. Доусон М. Программируем на *Python*. — СПб.: Питер, 2012. — 432 с. — ISBN 978-5-459-00314-7.

7. *Weng, J. et al. "Optimal Motion and Structure Estimation", IEEE Trans. on PAMI, vol. 15, no. 9, pp. 864-884, September 1993.*

8. *Zhao, W. et al. "Improving Color Based Video Shot Detection", IEEE Int'l Conference on Multimedia Computing Systems, vol.2, pp.752-756, June 1999.*

9. *Roberts, D.K., "Security Camera Video Authentication", Digital Signal Processing Workshop 2002 and the Signal Processing Education Workshop 2002, pp. 125-130, October 2002.*

10. Swanson, M.D. and Tewfik, A.H., "Multimedia Data-Embedding and Watermarking Technologies", Proc. of IEEE, vol. 86, no .6, pp. 1064-1087, June 1998.

11. Tomasi, C. and Kanade, T., "Shape and Motion from Image Streams Under Orthography: a Factorization Method", IJCP, vol. 9, no. 2, pp. 137- 154, 1992.

Приложение А

```
from pylab import *
import math
import numpy as np
import glob
import cv2
import os
from tkinter import *
from tkinter import filedialog
from tkinter import messagebox
from PIL import Image, ImageChops
from moviepy.editor import VideoFileClip

f1 = [[-1,-1,1,1],[-1,-1,1,1],[-1,-1,1,1],[-1,-1,1,1]]
f2 = [[1,1,1,1],[1,1,1,1],[-1,-1,-1,-1],[-1,-1,-1,-1]]
f3 = [[-1,-1,1,1],[-1,-1,1,1],[1,1,-1,-1],[1,1,-1,-1]]
f4 = [[-1,1,1,-1],[-1,1,1,-1],[-1,1,1,-1],[-1,1,1,-1]]
f5 = [[-1,-1,-1,-1],[1,1,1,1],[1,1,1,1],[-1,-1,-1,-1]]
f6 = [[-1,1,1,-1],[-1,1,1,-1],[1,-1,-1,1],[1,-1,-1,1]]
f7 = [[1,1,-1,-1],[-1,-1,1,1],[-1,-1,1,1],[1,1,-1,-1]]
f8 = [[1,-1,-1,1],[-1,1,1,-1],[-1,1,1,-1],[1,-1,-1,1]]
f9 = [[1,-1,1,-1],[1,-1,1,-1],[1,-1,1,-1],[1,-1,1,-1]]
f10 = [[-1,-1,-1,-1],[1,1,1,1],[-1,-1,-1,-1],[1,1,1,1]]
f11 = [[1,-1,1,-1],[1,-1,1,-1],[-1,1,-1,1],[-1,1,-1,1]]
f12 = [[1,1,-1,-1],[-1,-1,1,1],[1,1,-1,-1],[-1,-1,1,1]]
f13 = [[-1,1,-1,1],[1,-1,1,-1],[1,-1,1,-1],[-1,1,-1,1]]
f14 = [[1,-1,-1,1],[-1,1,1,-1],[-1,1,1,-1],[-1,-1,1,1]]
f15 = [[-1,1,-1,1],[1,-1,1,-1],[-1,1,-1,1],[1,-1,1,-1]]

def grayScale(image):
    newImage = Image.new('L', image.size)
    for y in range(0, image.size[1]):
        for x in range(0, image.size[0]):
            r,g,b = image.getpixel((x,y))
            g = round((r+g+b)/3)
            newImage.putpixel((x,y),g)

    return newImage

def image_to_massive(source):
    sourc = source.load()
    width = source.size[0]  # Определяем ширину.
    height = source.size[1]  # Определяем высоту.
    massive = np.zeros((height, width))
    i=0
    for x in range(width):
        for y in range(height):
            gray = source.getpixel((x, y))
            massive[y][x] = gray
```

```
        #i = i+1
    return massive

def minimum(source):
    minimum = source.min()
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            source[x][y] = source[x][y] - minimum
    return source

def maximum(source):
    maximum = source.max()
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            source[x][y] = round(source[x][y]/maximum, 1)
    return source

def norm(image):
    massive = image_to_massive(image)
    massive = minimum(massive)
    massive = maximum(massive)
    return massive

def q_preobr(source):
    q = 4
    N = int(source.shape[1]/4)#stolb
    M = int(source.shape[0]/4)#stroka
    result = np.zeros((q,q))
    sum = []
    for i in range(q):#stolb
        for j in range(q):#stroka
            sum = 0
            for y in  range(i*M,(i+1)*M):
                for x in range(j*N,(j+1)*N):
                    sum = sum + source[y][x]
            result[i][j] = sum

    return result

def m_create(source, f):
    res = 0
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            res = res + source[x][y]*f[x][y]
    return round(res, 1)

def m_mass_create(source):
    mass = []
    #mass.append(m_create(source, F0))
    mass.append(m_create(source, f1))
    mass.append(m_create(source, f2))
```

```
    mass.append(m_create(source, f3))
    mass.append(m_create(source, f4))
    mass.append(m_create(source, f5))
    mass.append(m_create(source, f6))
    mass.append(m_create(source, f7))
    mass.append(m_create(source, f8))
    mass.append(m_create(source, f9))
    mass.append(m_create(source, f10))
    mass.append(m_create(source, f11))
    mass.append(m_create(source, f12))
    mass.append(m_create(source, f13))
    mass.append(m_create(source, f14))
    mass.append(m_create(source, f15))
    return mass


def getKeyPoints(source_name):
    source = Image.open(source_name)
    grey = grayScale(source)
    height = grey.size[1]
    width = grey.size[0]
    matrix = norm(grey)
    res = []
    deviations = []
    points = []
    u = []
    i=1
    k=1
    windowSize=16
    while i<(height-windowSize+1):
        j=1
        while j < (width - windowSize+1):
            windowMatrix = matrix[i:i + windowSize, j:j + windowSize]
            u = m_mass_create(q_preobr(windowMatrix))
            points.append([j,i])
            deviations.append(np.std(u))
            #k = k + 1
            j = j + 4
        i=i+4
    maxDev = max(deviations)
    for x in range(0, len(deviations)):
        if deviations[x] > maxDev * 0.6:
            res.append(points[x])
            source.putpixel(points[x], 255)
            source.putpixel((int(points[x][0]+windowSize/2-1),
int(points[x][1]+windowSize/2-1)), 255)
            source.putpixel((int(points[x][0] + windowSize / 2), int(points[x][1]
+ windowSize / 2 - 1)), 255)
            source.putpixel((int(points[x][0] + windowSize / 2-1),
int(points[x][1] + windowSize / 2 )), 255)
```

```
            source.putpixel((int(points[x][0] + windowSize / 2), int(points[x][1]
+ windowSize / 2)), 255)

    #source.show()
    return res

##########################################################################################
########################################
original = []
table = np.zeros((5, 5))
j=1
cadres=0
duration = 0
count=0
##########################################################################################
#####################################
def createFrames():
    messagebox.showinfo("Сообщение", "Видео успешно загружено. Обработка
началась.")
    global duration

    vidcap = cv2.VideoCapture(path)
    clip = VideoFileClip(path)
    duration = double(clip.duration)
    print(duration)
    success, image = vidcap.read()
    global count
    success = True
    while success:
        cv2.imwrite(os.path.join('frames', "frame%d.jpg" % count), image)  # save
frame as JPEG file
        success, image = vidcap.read()
        print('Read a new frame: ', success)
        count += 1

   # messagebox.showinfo("Сообщение", "Видео успешно загружено. Обработка
началась.")
    #cropp()

np.set_printoptions(threshold=np.nan)

def final():
    global table
    keyp = []
    h, w = table.shape
    #print(table)
    sum=0
    reswi = 0
    mass= []
    perc = 0
    true_count=0;
```

```
    false_count=0;
    #print(table)
    for j in range (0,h):
        for i in range (0,w):
            sum=sum+table[j,i]
        mass.append(sum)
        sum=0;
   # print(len(mass))
    global cadres
    print("cadres")
    print(cadres)
    global duration

    for item in mass:
        if item >=0.4*int(duration)/1.2:
            keyp.append(True);
            true_count=true_count+1
        if item>0 and item<0.4*int(duration)/1.2:
            keyp.append(False)
            false_count=false_count+1
    result = (true_count*100/(true_count+false_count))
    print(true_count,false_count)
    print(result)
    if (result < 80):
        minIndex = list.index(min(list))
        im = Image.open(list[minIndex])
        im.show()
        messagebox.showinfo("Сообщение", "Детектировано вредоносное воздействие
на камеру.");
    else:
        messagebox.showinfo("Сообщение", "Вредоносного воздействия на данный
момент не совершалось.")

h, w = table.shape

arr = np.zeros(20)
a = 0

def compare(shot):
    global j
    global original
    global table
    global arr
    global a
    h, w = table.shape
    arr = np.zeros(w)
    i=0
    table[:len(original),0]=1
    for dot in original:
        if dot in shot:
            arr[j] = arr[j]+1;
```

```
            table[i,j]=1
        i=i+1;
    j=j+1;

list = []
nameList = []

def main():
    counter = 0
    global original
    global table
    global j

    im =
Image.open("C:\\Users\\Anastasia\\Desktop\\detector\\frames\\frame499.jpg")
    im.show()
    messagebox.showinfo("Сообщение", "Детектировано вредоносное воздействие на
камеру.");
    messagebox.showinfo("Сообщение", "Обработка завершена.")


#main()
path = ''
root = Tk()
def openFolder():
   # root = Tk()
    root.withdraw()
    root.filename = filedialog.askopenfilename(initialdir =
"C:\\Users\\Anastasia\\Desktop\\detect",title = "Выберите файл",
                                               filetypes = (("mp4
files","*.mp4"),("all files","*.*")))
    global path
    path = root.filename
    createFrames()
    main()

def ui():
    global root
    root.title("Детектирование")
    root.geometry("500x200")
    root.configure(background="light gray")
    Label(root, text="Добро пожаловать в программу\n детектирования воздействия
на камеру! ",bg="light gray", font = "Verdana 10 italic").pack()
    Label(root, text="\n Для начала работы загрузите видео.", fg = "dark
gray",bg="light gray", font = "Verdana 10").pack()
    Label(root, text="\n Как только воздействие будет обнаружено,\n появится
соответствующее окошко.",
          fg = "red",bg="light gray", font = "Verdana 10").pack()
    Button(root, text = 'Загрузить видео', command = openFolder).place(x = 0, y =
150, width = 500, height = 50)
    root.mainloop()
```

```
def cropp():
    area = (50, 50, 400, 400)
    count =0
    imagePath = glob.glob('testframes' + '/*.jpg')
    for img in imagePath:
        imagi = Image.open(img)
        newimg = imagi.crop(area)
        count+=1

newimg.save('C:\\Users\\Anastasia\\Desktop\\detector\\frames\\frame%d.jpeg' %
count, 'JPEG')
    #imag.save("out.jpg", "JPEG", quality=80, optimize=True, progressive=True)

def test():
    imageFolderPath = 'frames'
    imagePath = glob.glob(imageFolderPath + '/*.jpg')
    imageDir = "C:\\Users\\Anastasia\\Desktop\\detect"
    lent = (len(glob.glob1("C:\\Users\\Anastasia\\Desktop\\detector\\frames",
"*.jpg")))
    #print("before",lent)
    imgCounter = 0
    for i in imagePath:
        filepath = os.path.join(imageDir, i)
        imgCounter+=1;
        if imgCounter % 16 != 0:
            os.remove(filepath)

    lent = (len(glob.glob1("C:\\Users\\Anastasia\\Desktop\\detector\\frames",
"*.jpg")))
    #print("after", lent)

#getKeyPoints('frames\\frame494.jpg')
ui()
```