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

Институт Институт радиоэлектроники и информационных технологий (ИРИТ)

Направление подготовки (специальность) 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 Разумовского Сергея Александровича группы 15-В-2
(Ф.И.О.)

на тему Программная система дополненной реальности с многопользовательским режимом
(наименование темы работы)

СТУДЕНТ

Разумовский С.А.
(подпись) (фамилия, и., о.)

(дата)

РУКОВОДИТЕЛЬ

Гай В.Е.
(подпись) (фамилия, и., о.)

(дата)

РЕЦЕНЗЕНТ

(подпись) (фамилия, и., о.)

(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

Мякиньков А.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
протокол №
с оценкой

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

Кафедра Вычислительные системы и технологии

УТВЕРЖДАЮ
Зав. кафедрой
_____ А.В. Мякиньков
«___» _____ 20___ г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника
(код и наименование)

студенту Разумовскому Сергею Александровичу группы 15-В-2
(Ф.И.О.)

1. Тема ВКР Программная система дополненной реальности с многопользовательским режимом

(утверждена приказом по вузу от 28.03.2019 № 79715)

2. Срок сдачи студентом законченной работы
3. Исходные данные к работе Данные с камеры (видеопоток), набор изображений в формате PNG; язык разработки – Python; система включает приложение для персонального компьютера и сервер

4. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке) 1. Введение
2. Техническое задание
3. Анализ поставленной задачи
4. Разработка структуры системы
5. Разработка программных средств
6. Тестирование системы
7. Заключение
8. Список литературы
9. Приложение

5. Перечень графического материала (с точным указанием обязательных чертежей)
1. Общая структурная схема системы
2. Архитектура клиентского приложения
3. Общая схема работы сервера
4. Расширенная структурная схема работы сервера
5. Изображения схемы модулей клиентского приложения

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Гай В.Е.

7. Дата выдачи задания

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ПК-1, Способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов «человек - электронно-вычислительная машина»	Разработать структурную схему системы	Структурная схема системы	
ПК-2, Способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	Разработать алгоритмы работы системы, реализовать их на одном из языков программирования с помощью современной среды разработки	Схема алгоритма работы реализована на языке Python в среде разработки PyCharm Professional	
ПК-3, Способность обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности	Осуществить отбор лучших результатов, оценить их качество по нескольким критериям	Осуществлён отбор лучших результатов, оценено их качество	

Руководитель _____ В.Е. Гай
(подпись) (И.О. Фамилия)

Задание принял к исполнению _____
(дата)

Студент _____ С.А. Разумовский
(подпись) (И.О. Фамилия)

Примечания:
1. Это задание прилагается к законченной работе и в составе пояснительной записки предоставляется в ГЭК.
2. До начала консультаций студент должен составить и утвердить у руководителя календарный график работы на весь период выполнения ВКР (с указанием сроков выполнения и трудоемкости отдельных этапов).

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

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника
(код и наименование)

студента Разумовского Сергея Александровича группы 15-В-2
(Ф.И.О.)

по теме Программная система дополненной реальности с многопользовательским режимом

Выпускная квалификационная работа выполнена на 52 страницах, содержит 33 рисунка, библиографический список из 7 источников.

Актуальность: в настоящее время сфера технологий дополненной реальности получает всё большую популярность, но готовые полноценные системы отсутствуют. В связи с этим было принято решение о написании программной системы дополненной реальности с многопользовательским режимом.

Объект исследования: видеопоток.

Предмет исследования: алгоритмы распознавания маркеров, алгоритмы обработки видеопотока.

Цель исследования: разработка программной системы дополненной реальности с многопользовательским режимом с использованием персонального компьютера.

Задачи исследования: исследовать существующие реализации систем дополненной реальности; разработать собственную программную систему дополненной реальности с многопользовательским режимом; выполнить тестирование разработанной системы, с целью проверки её работоспособности.

Методы исследования: определение идентификатора маркера на основе данных с камеры.

Структура работы: выпускная квалификационная работа состоит из введения, пяти глав, заключения, списка литературы и приложения.

Во введении даётся описание проблемы, лежащей в основе данной работы.

В 1 разделе «Техническое задание» составлены технические требования к разрабатываемому продукту.

Во 2 разделе «Анализ поставленной задачи» производится выбор программных средств, обзор существующих систем, даётся краткое описание алгоритма решения поставленной задачи.

В 3 разделе «Разработка структуры системы» разрабатывается структурная схема, алгоритмы решения каждого из этапов обработки данных.

В 4 разделе «Разработка программных средств» разрабатываются программные средства для решения поставленной задачи.

В 5 разделе «Тестирование системы» описываются методы тестирования системы и полученные результаты.

В заключении производятся основные выводы о проделанной работе.

Выводы:

1. Разработана программная система дополненной реальности с многопользовательским режимом.
2. Тестирование системы подтвердило её работоспособность и возможность использования для поставленной задачи.

Рекомендации:

1. Дальнейшее развитие проекта.
2. Оптимизация существующего кода.

_____/ Разумовский С.А.
подпись студента /расшифровка подписи

«____» _____ 20_____ г.

Содержание

Введение

Сегодня одним из самых перспективных направлений в сфере IT-разработок является дополненная реальность. Данная технология представляет собой новый способ получения информации.

Дополненная реальность способна сделать восприятие информации человеком гораздо проще и нагляднее. Требуемые запросы будут автоматически доставляться пользователю. Дополненная реальность - это, прежде всего, технология, с помощью которой реальные объекты приобретают новые качества и раскрываются пользователю, с другой стороны. Принцип дополненной реальности заключается в совмещении виртуальных и существующих объектов в режиме реального времени. Взаимодействие техники с изображением реального мира отличает дополненную реальность от виртуальной.

Главной задачей дополненной реальности является увеличение возможностей пользователей, т. е. их взаимодействие с окружением, но уже на существенно новом уровне. С помощью компьютерного устройства на изображение реальной среды наносятся слои с набором объектов, несущих дополнительную информацию. Сейчас технологии позволяют считывать и распознавать изображения окружающей среды при помощи камер, а также дополнять их при помощи несуществующих объектов.

Целью данной работы является разработка системы дополненной реальности с многопользовательским режимом.

1. Техническое задание

1.1 Назначение разработки и область применения

В настоящее время такие международные компании, как Google (Google Glass, Magic Leap, ARCore), Apple (ARKit), Qualcomm (Vuforia), Microsoft (Hololense), а также множество российских компаний, заинтересованы в создании систем дополненной реальности. Также данные корпорации работают над созданием SDK (от англ. software development kit – набор средств разработки), для облегчения разработки программного обеспечения на своих устройствах.

Разрабатываемая система предназначена для дополнения сведений об окружении и улучшения восприятия информации. Для работы данной системы необходимо иметь портативный или стационарный компьютер со специализированным программным обеспечением.

1.2 Технические требования

Поскольку в разрабатываемой системе используются несколько устройств, то можно выделить две группы требований к различным устройствам: серверному компьютеру и клиентскому.

Требования к серверному компьютеру:

- Наличие подключения к Интернету;

- ОС, способная запустить Python скрипт.

Требования к клиентскому компьютеру:

- Операционная система Microsoft Windows 10 и выше;

- Требования к аппаратному обеспечению определяются операционной системой;

- Мышь, дисплей, клавиатура, веб камера;

- Доступ в интернет.

Рассмотрим, каким функционалом должна обладать разрабатываемая система дополненной реальности:

1. Система должна позволить пользователю изменять набор маркеров и загружать их в систему для обработки.

2. Должен быть реализован метод идентификации созданных маркеров.

3. Должна быть реализована система приёма/передачи и хранения информации.

4. Вывести на экран видеопоток с дополнительной информацией.

1.3 Требования к клиентской части приложения

Функционал продукта должен обеспечиваться несколькими взаимодействующими друг с другом модулями:

- Модуль генерации маркера, которым выполняется создание маркера для создания базы данных;

- Модуль сканирования маркера, которым выполняется распознавание маркера в клиентской части;

- Модуль для отрисовки интерфейса программы;

- Модуль для получения видеопотока с камеры;

- Связующий модуль обработки изображений для отображения видеопотока в окне интерфейса;

- Модуль событий для определения положения курсора в момент нажатия и сопоставления текущего положения с координатами обнаруженных ранее маркеров;

- Сетевой модуль, который реализует взаимодействие с сервером;

- Модуль, объединяющий полученную информацию и видеопоток;

- Центральный модуль, задача которого – обеспечение взаимодействия всех предыдущих модулей.

1.4 Требования к реализации серверной части

- В качестве протокола взаимодействия между различными частями системы на транспортном уровне будет использован протокол UDP;

- Для реализации обмена информацией между компонентами системы необходимо использовать протоколы прикладного уровня: HTTP и HTTPS;

- Для организации базы данных в которой хранится история сообщений будет использоваться хеш-таблица.

2. Анализ поставленной задачи

2.1 Выбор операционной системы

Первоначальным этапом является выбор операционной системы, поскольку каждая из них имеет различные преимущества и недостатки, влияющие, в свою очередь, на процесс разработки программного обеспечения. На данный момент наиболее распространёнными дистрибутивами операционных систем являются: Mac OS, Linux (Mint, Ubuntu), Windows 10.

1. Операционная система Windows 10 появилась относительно недавно – она стала доступной с 29 июля 2015 года. Компания Microsoft при разработке продолжала свой путь, направленный на унификацию. Допускается установка на компьютеры, ноутбуки, планшеты, а также смартфоны и консоли Xbox One. Единая платформа обеспечивает возможность синхронизации настроек, как это уже было на предшествующих версиях. Отдельного внимания заслуживает распространение операционной системы. Довольно большое количество пользователей не захочет переходить с полностью устраивающих их семерки и восьмерки. Как показывает статистика, именно они заняли значительную часть рынка. Разработчик предложил отличную возможность для пользователей данных ОС – выполнить обновление бесплатно в течение одного года с момента выпуска. Весьма интересным является тот факт, что Windows 10 будет последней версией, которая вышла в коробке. С этих пор выпуск будет осуществляться только в цифровом виде. Как и у всякого крупного программного продукта, после выхода появилось большое количество критики самого разного плана. Следует рассмотреть основные моменты, вызвавшие недовольство со стороны пользователей. В первую очередь, многим не понравилась система сбора данных. Ранее уже говорилось о том, насколько большое количество сведений о пользователе она отправляет Microsoft. Сюда можно отнести местоположение, контакты и частоту разговоров с ними, данные электронной почти и другое. Имелись и другие претензии не столь существенного плана. Они касались сложностей установки или проблем в ходе работы с некоторыми приложениями. На данный момент, "патчи" оперативно исправляют это.

2. Mac OS (Macintosh Operating System) — семейство проприетарных операционных систем производства корпорации Apple. Разработана для линейки персональных компьютеров Macintosh. Популяризация графического интерфейса пользователя в современных операционных системах часто считается заслугой Mac OS. Она была впервые представлена в 1984 году вместе с персональным компьютером Macintosh 128K. Apple

хотела, чтобы Macintosh представлялся как «компьютер для всех остальных». Самого термина «Mac OS» в действительности не существовало до тех пор, пока он не был официально использован в середине 1990-х годов. С тех пор термин применяется ко всем версиям операционных систем Макинтоша как удобный способ выделения их в контексте других операционных систем.

3. Linux – операционные системы, которые основаны на свободном и открытом программном обеспечении. Операционные системы на базе ядра Linux представляют собой группу Unix-подобных операционных систем. ОС Linux может быть установлена на большое количество компьютерных устройств: мобильные телефоны, планшетные компьютеры, маршрутизаторы, игровые консоли, настольные компьютеры, мэйнфреймы и суперкомпьютеры. У Linux нет единой «официальной» комплектации, так как она поставляется в виде дистрибутивов. Дистрибутивы Linux создаются на основе одноименного ядра, библиотек и системных программ, которые разрабатываются в рамках проекта GNU. ОС Linux является одним из наиболее ярких примеров свободного и открытого программного обеспечения, так как исходный код операционной системы находится в свободном доступе и может быть использован в других проектах, если они также, как и ОС выпускаются под лицензией GNU (General Public License).

Для разработки системы дополненной реальности с многопользовательским режимом мною была выбрана система из семейства Windows, так как у меня имеется больший опыт работы на данных системах и все необходимые средства разработки доступны на данной ОС.

2.2 Выбор языка программирования

Следующим, не менее важным этапом разработки программного обеспечения является выбор языка программирования. Возьмём наиболее распространённые на текущий момент языки программирования, такие как C++, Python, Ruby, Java и рассмотрим их более детально:

1. C++ - компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C, — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования.

2. Python - высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

3. Ruby - динамический, рефлективный, интерпретируемый высокоуровневый язык программирования. Язык обладает независимой от операционной системы реализацией многопоточности, сильной динамической типизацией, сборщиком мусора и многими другими возможностями. По особенностям синтаксиса он близок к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и Клу. Кроссплатформенная реализация интерпретатора языка является полностью свободной.

4. Java - сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). В настоящее время проект принадлежит OpenSource и распространяется по лицензии GPL. В OpenJDK вносят вклад крупные компании, такие как — Oracle, RedHat, IBM, Google, JetBrains. Так же на основе OpenJDK эти компании разрабатывают свои сборки JDK. Как утверждает компания Oracle — отличия между OpenJDK и OracleJDK практически отсутствуют за исключением лицензии, отрисовки шрифтов в Swing и некоторых библиотек, на которые лицензия GPL не распространяется. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины.

Решающим фактором в процессе выбора языка программирования являлось наличие библиотеки для распознавания маркеров – python-ar-markers. Её реализация есть только на Python, поэтому для разработки был выбран язык Python.

Рассмотрим полный состав используемых библиотек:

1. OpenCV - библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD.

2. python-ar-markers – библиотека для обнаружения и создания маркеров Хэмминга. Данная библиотека представляет из себя модуль для Python и является надстройкой над OpenCV.

3. Tkinter - кроссплатформенная графическая библиотека на основе средств Tk (широко распространённая в мире GNU/Linux и других UNIX-подобных систем, портирована также и на Microsoft Windows).

4. socket – предоставляет интерфейс для работы с сокетами в Python.

5. math – модуль для работы с числами (включает реализацию математических функций).

6. os - функции для работы с операционной системой.

7. Python Imaging Library — библиотека языка Python, предназначенная для работы с растровой графикой.

2.3 Выбор версии языка Python

Так как в настоящее время существует две основных версии языка, которые до сих пор поддерживаются, имеет смысл рассмотреть их.

Python 2 - Вышедшая в 2000 году версия Python 2 сделала процесс разработки более прозрачным и всеобъемлющим по сравнению с предыдущими версиями Python с реализацией PEP. Примечание: PEP (Python Enhancement Proposal) – техническая спецификация, которая предоставляет информацию членам сообщества Python или же описывает новую функцию языка. Кроме того, Python 2 предложил множество новых функций: циклический сборщик мусора для автоматизации управления памятью, расширенную поддержку Unicode для стандартизации символов, списковую сборку и т.п. По мере разработки Python 2 набор функций значительно расширился, в том числе появилась унификация типов и классов Python (версия 2.2).

Python 3 - Python 3, последняя разрабатываемая версия, уже считается будущим этого языка программирования. Python 3 был выпущен в конце 2008 года, его целью было устранение внутренних конструктивных недостатков предыдущих версий языка. Python 3 сосредоточен на поддержке чистой базы кода и устранении избыточности. Основные изменения в Python 3.0: замена оператора print встроенной функцией, улучшение способа деления целых чисел и продвинутая поддержка Unicode.

Python 2.7 - Версия Python 2.7 вышла 3 июля 2010 года и должна была стать последней версией Python 2.x. Версия Python 2.7 была предназначена для пользователей Python 2.x, которым трудно перейти на новую версию, Python 3, и должна была обеспечить совместимость этих версий. Она предоставляла усовершенствованные модули для версии 2.7 (например, unittest для поддержки автоматизации тестирования, argparse для разбора параметров командной строки, а также более удобные классы – коллекции). Таким образом, версия Python 2.7 оказалась в уникальном положении: он стал связующим звеном между Python 2 и Python 3.0, благодаря своей совместимости со многими надежными библиотеками он получил популярность среди программистов. Как правило, сегодня при упоминании Python 2 имеется в виду именно версия Python 2.7. Версия Python 2.7 по-прежнему остаётся в разработке, которая на данный момент почти полностью состоит из исправлений багов и будет полностью прекращена в 2020 году.

В итоге была выбрана версия языка Python 3.6, так как все необходимые библиотеки для разработки приложения были переписаны под новую версию языка и надобности в использовании старой версии языка нет.

2.3 Выбор среды разработки

Программировать на языке Python можно даже в блокноте, а запускать написанный код в командной строке с помощью интерпретатора, но всё же разумнее использовать специализированную IDE (англ. Integrated Development Environment – интегрированная среда разработки), так как она предоставляет более продвинутые средства работы с кодом.

Из всех вариантов мною была выбрана среда разработки PyCharm от JetBrains, её Community версия доступна для всех open-source проектов, а версия Professional является бесплатной для студентов.

Ключевыми особенностями PyCharm являются:

- Установка модулей Python прямо из IDE;
- Настройка конфигураций для запуска скриптов;
- Подсветка синтаксиса;
- Создание виртуальной среды Python для каждого проекта;
- Простое управление рабочими каталогами и файлами;
- Встроенный отладчик, для быстрого обнаружения ошибок.

2.4 Обзор существующих систем дополненной реальности

ARKit - использует подразумевает привязку реального мира к live-видео и системе координат XYZ. При этом ARKit работает исключительно на iOS. Для создания виртуальной модели реального мира, ARKit использует метод визуальной одометрии. В процессе используются показания с датчиков движения iOS-устройства и, кроме того, происходит оценка перемещения объекта в пространстве. Система ARKit выявляет основные характеристики видимых объектов, проводит трекинг изменения их положения. Далее идёт сравнение полученной информации с данными датчиков движения. В итоге, получаем сверхточную модель отслеживания нахождения и направления движения устройства. Более того, представленная система отслеживания анализирует и сканирует пространство с разных сторон. ARKit использует специальные методы тестирования, которые предназначены для нахождения и анализа реальных поверхностей и интеграции их с изображением камеры. При запуске planeDetection в параметрах, ARKit начнет обнаруживать плоские горизонтальные поверхности, сообщая об их размерах и расположении. Для перемещения или изменения местоположения виртуальных объектов можно использовать результаты проверки наличия данных или данные о размещённых плоскостях.

ARCore - платформа работает на седьмой версии Android, используя встроенные возможности телефонов. Программа ARCore формирует дополненную реальность на основании трех принципов: анализа освещенности, контроля движения, осознания и опознавания реального окружения. Созданный в пространстве «виртуальный» объект реагирует на изменения освещенности и способен отбрасывать тень. Колебания гаджета не изменяют положения этого объекта, когда он уже зафиксирован программой. Также программа определяет габариты, углы наклона горизонтальных поверхностей, на которые можно устанавливать новые «дополненные» фигуры. Для определения параметров физических объектов используется камера телефона. Она определяет ключевые точки пространства, точнее, их высчитывают программные средства при помощи камеры, а данные передает блок инерциальных измерений. Эта же система применяется для определения горизонтальных поверхностей, после чего создаваемые объекты надежно привязываются к столам, полкам и другим поверхностям. После закрепления AR-объект становится частью реального мира, и можно перемещать смартфон без опасения потерять созданную фигуру.

Vuforia – использует в качестве маркеров как 2D изображения, так и 3D объекты различных форм (куб, кубоид, цилиндр). После определения соответствующего изображения или объекта использует его как точку отсчёта. Благодаря интеграции с Unity позволяет размещать различные 2D и 3D объекты, создавая сцену вокруг точки отсчёта.

Все вышеописанные системы обладают различными недостатками: их использование не всегда бесплатно, их разработка требует существенных вложений, но главным недостатком является то, что использование данных технологий доступно в основном на ограниченном модельном ряде устройств.

Основными преимуществами разрабатываемой системы будут:

- Бесплатность использования;
- Открытость исходного кода;
- Простота в использовании;
- Можно запускать под управлением почти любого компьютера с веб камерой.

3. Разработка структуры системы

Программная система дополненной реальности с многопользовательским режимом состоит из нескольких ключевых частей.

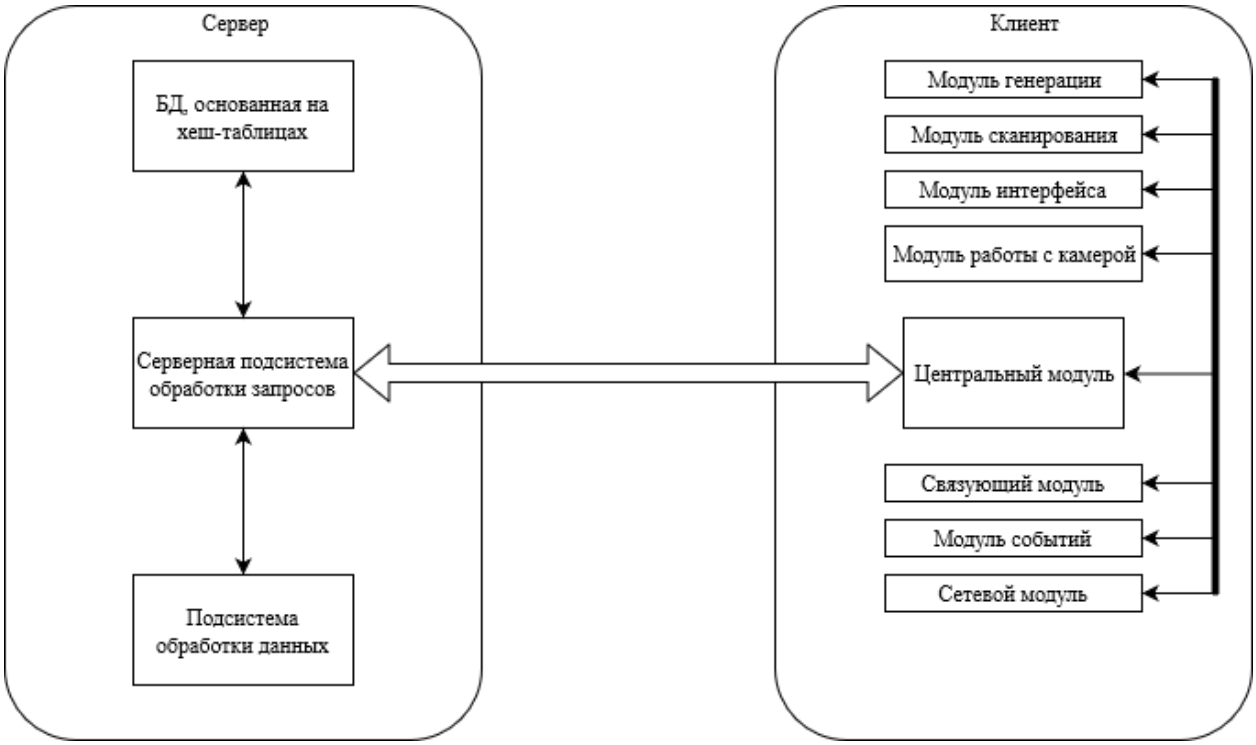

Рисунок 1. Структурная схема разрабатываемой системы

У разрабатываемой системы можно выделить две ключевых составляющих:

- Серверная часть;

- Приложение для персонального компьютера (клиентская часть).

В результате реализации клиент-серверной архитектуры были решены следующие проблемы:

- Необходимость централизованной обработки и хранения данных. Это позволит избавить клиентское устройство от некоторой нагрузки;

- Проблема синхронизации информации между несколькими клиентами.

Рассмотрим составляющие более подробно.

3.1 Структура клиентского приложения

Клиентское приложение реализовано с применением модульной архитектуры, в результате чего имеет большую гибкость и масштабируемость.

Так, например, при отказе какого-либо модуля, приложение продолжит свою работу. Однако при отказе критически важных модулей, например, камеры или центрального модуля имеет место быть некорректная работа приложения. А если откажет сетевой модуль, в результате отсутствия Интернет-соединения, то приложение продолжит свою работу, но перестанет получать данные (т.е. маркер будет распознан, но соответствующий ему объект получен не будет).

Если в дальнейшем возникнет потребность в расширении функционала программного продукта, то не будет необходимости переписывать весь код приложения. Также будет возможна работа в команде, достаточно лишь разделить задачи по модулям, в результате чего разработчики не будут обязаны знать о внутренних особенностях реализации модулей. Они смогут использовать готовые интерфейсы для работы с модулем. Благодаря такому подходу достигается больший уровень абстракции.

Рассмотрим структуру и назначение всех модулей.

Начнём с модуля генерации маркера. Его предназначение заключается в создании изображения, которое в будущем будет использовано в серверной части для создания базы данных маркеров.

Сам по себе маркер представляет чёрно-белое изображение 150×150 пикселей разбитое на зоны 7×7 (но зона с данными имеет размер 5×5 и объём информации 25 бит соответственно, а остальное это черный контур). Эти зоны закрашены в определённой последовательности, заданной алгоритмом. Каждый маркер имеет определённый идентификатор, который получается исходя из того, как были закрашены зоны.

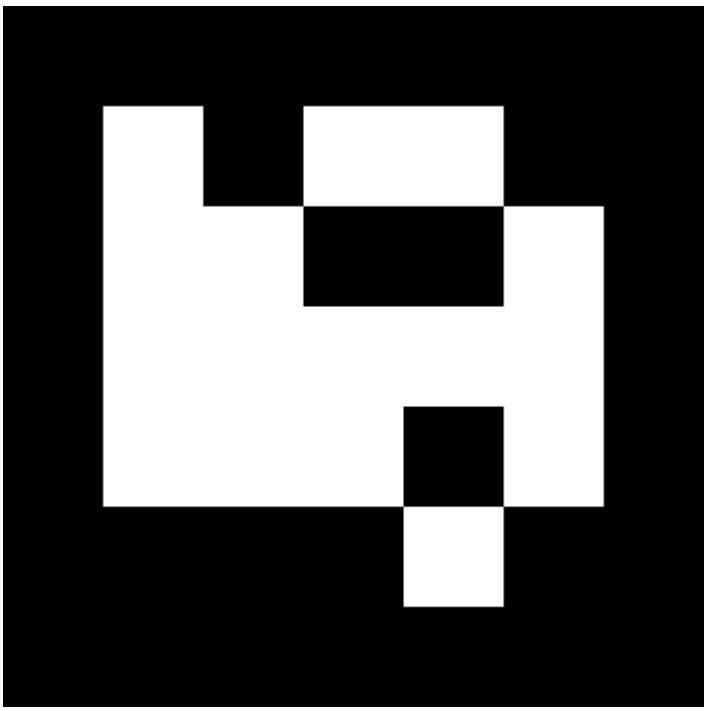

Рисунок 2. Чёрно-белый маркер, построенный на основе кода Хэмминга

Стоит пояснить, что из себя представляет код Хэмминга. Итак, Код Хэ́мминга — вероятно, наиболее известный из первых самоконтролирующихся и самокорректирующихся кодов. Построен применительно к двоичной системе счисления. Позволяет исправлять одиночную ошибку (ошибка в одном бите) и находить двойную. Другими словами, это алгоритм, который позволяет закодировать какое-либо информационное сообщение определённым образом и после передачи (например, по сети) определить появилась ли какая-то ошибка в этом сообщении (к примеру, из-за помех) и, при возможности, восстановить это сообщение.

Рассмотрим, что входит в 25 бит информации, содержащейся в маркере:

- 4 бита используются для определения ориентации (крайние точки);

- 9 бит используются для контроля и коррекции ошибок;

- 12 бит используются для кодировки сообщения.

Это значит, что в данные маркеры могут зашифровать $2^{12} = 4096$ значений.

Процесс генерации маркера происходит в несколько этапов:

1. Случайно генерируется число от 0 до 4095.

2. Создаётся пустое изображение 7×7.

3. Первый пиксель изображения – белый, необходим для определения ориентации.

4. Сгенерированное число переводится из десятичной системы счисления в двоичную. К примеру, было получено число 3313, его представление в двоичном виде 110011110001.

5. Полученное двоичное число кодируется при помощи Кода Хэмминга. Рассмотрим процесс подробнее. После получения двоичного числа, необходимо определиться с длинной информационного слова. Допустим, у нас длина слова будет равна 4, таким образом, полученное число необходимо разделить на блоки по 4 бита, которые будут кодироваться отдельно друг от друга. Итак, получены 3 бинарные строки по 4 бита: 1100, 1111, 0001. Далее все 3 строки кодируются параллельно и независимо друг от друга. Рассмотрим, как происходит кодирование, на примере первой строки. Прежде всего необходимо вставить контрольные биты. Они вставляются в строго определённых местах – в позициях с номерами, равными степеням двойки. В моём случае это будут позиции 1,2,4. Соответственно было получено 3 контрольных бита (выделены жирным) – **00**1**0**100. Таким образом, длина всего сообщения увеличилась на 3 бита. До того, как будут вычислены контрольные биты, им присваивается значение «0». Теперь необходимо вычислить значение каждого контрольного бита. Значение каждого контрольного бита зависит от значений информационных бит, но не от всех, а только от тех, которые этот контрольных бит контролирует. Для того, чтобы понять, за какие биты отвечает каждый контрольный бит необходимо понять определённую закономерность: контрольный бит с номером *N* контролирует все последующие *N* бит через каждые *N* бит, начиная с позиции *N*. На рис. 3 представлено более подробное пояснение.

1	2	3	4	5	6	7	
0	0	1	0	1	0	0	
X		X		X		X	1
	X	X			X	X	2
			X	X	X	X	4

Рисунок 3. Здесь знаком «X» обозначены те биты, которые контролирует контрольный бит, номер которого справа. К примеру, бит номер 6 контролируется битами 2 и 4.

Далее идёт вычисление контрольного бита. Берётся каждый контрольный бит и проверяется, сколько бит он контролирует (количество X), в результате получается целое число, если это число чётное, то ставится 1, а если нечётное – 0. Посчитав контрольные биты для информационного слова 1100, получается следующее: **0**1**11**100.

6. Полученное двоичное число длинной в 21 бит используется для заполнения оставшихся 21 клеток (0 в двоичной системе соответствует черному цвету, а вместо 1 подставляется 255, что означает белый цвет). Позиции, которые будут закрашены данным методом прописаны заранее в переменной HAMMINGCODE_MARKER_POSITIONS.

7. Изображение пропорционально увеличивается до 150×150 пикселей.

В результате на выходе имеется изображение, готовое к использованию на сервере.

Далее, рассмотрим модуль сканирования маркера.

Этапы работы модуля:

1. Применяется адаптивный порог для получения границ (рис. 4).

2. Нахождение контуров. После этого обнаруживаются не только настоящие маркеры, но и множество ненужных нам границ. Остальная часть алгоритма состоит в том, чтобы отфильтровать нежелательные границы. Удаляются границы с небольшим количеством точек (рис. 5).

3. Происходит полигональная аппроксимация контура с сохранением замкнутых контуров к четырьмя углами (т.е. прямоугольников) (рис. 6).

4. Убираются слишком маленькие прямоугольники. Это необходимо из-за того, что адаптивный порог обычно определяет внутреннюю и внешнюю часть границы маркера. На этом этапе сохраняется внешняя граница (рис. 7).

5. Происходит идентификация маркера. Если это маркер, то он содержит в себе код. Маркер поделён сеткой 7×7, из которой 5×5 это ячейки, в которых содержится информация об идентификаторе. Остальное относится к внешней чёрной границе. После считывания информации происходит её декодирование.

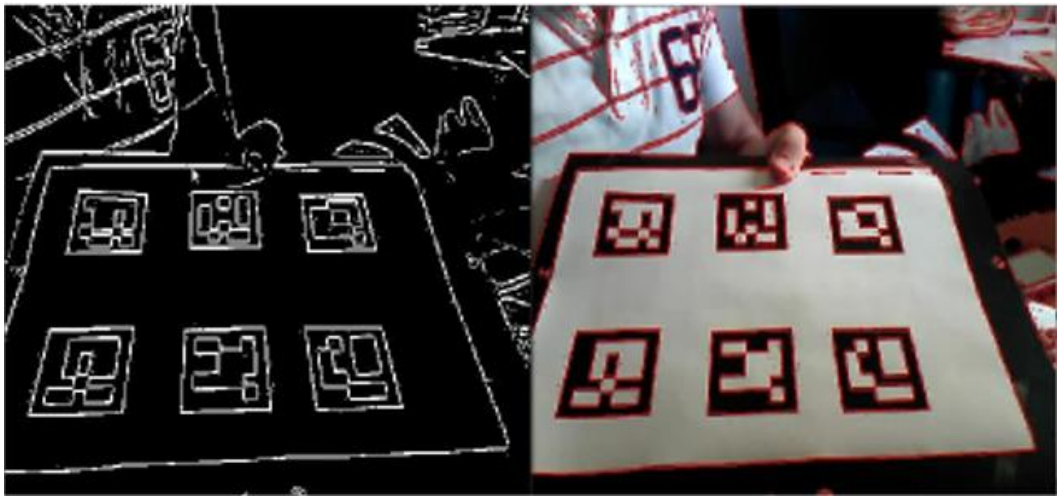

Рисунок 4. Получение границ

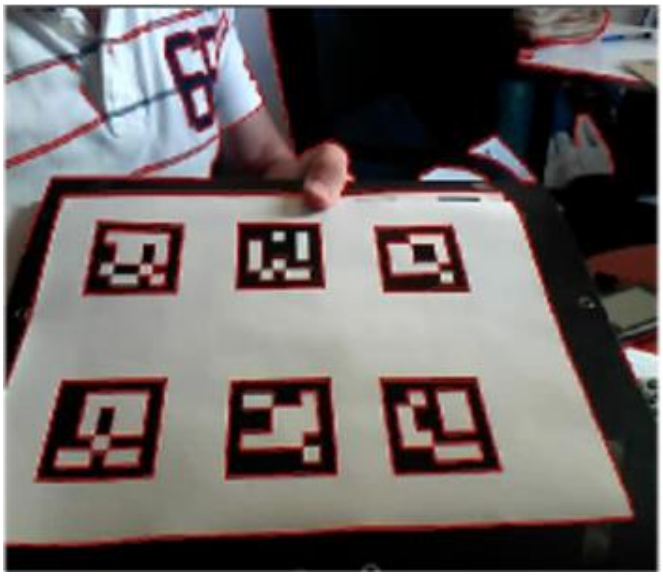

Рисунок 5. Нахождение контуров

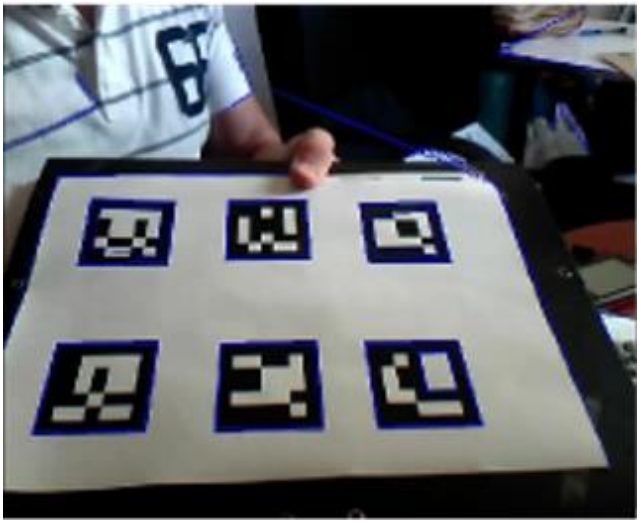

Рисунок 6. Аппроксимация контуров

Рисунок 7. Отсеивание несоответствующих прямоугольников

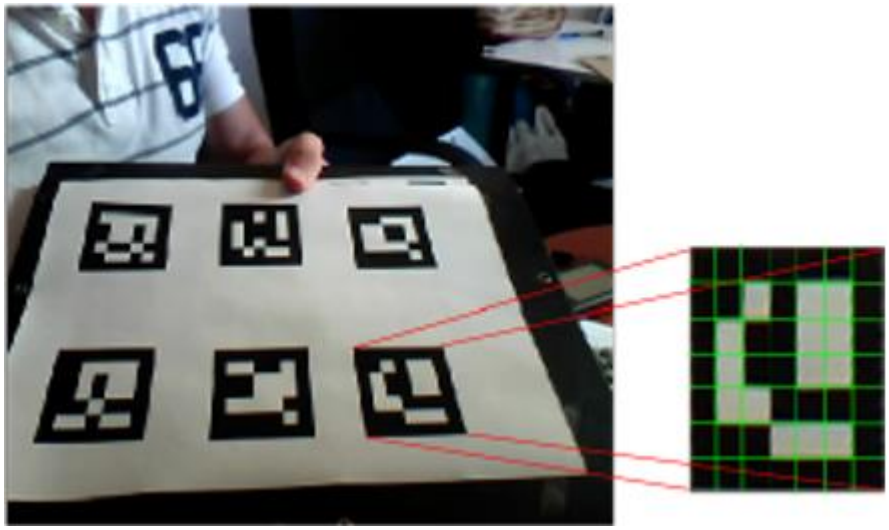

Рисунок 8. Идентификация маркера

6. Процесс декодирования. Допустим, было получено число, но с ошибкой, например, шестой бит был передан неправильно и в итоге вместо 0111100 было получено 0111110. Алгоритм заключается в том, что необходимо заново вычислить все контрольные биты и сравнить их с контрольными битами. Посчитав контрольные биты с неправильным шестым битом получится следующее: 0010110. Видно, что контрольные биты 2 и 4 не совпадают с такими же контрольными битами, которые были получены. Теперь сложив номера позиций неправильных контрольных бит (2+4=6) получается позиция ошибочного бита. Инвертируем ошибочный бит и отбрасываем контрольные биты. В результате имеем исходное число.

Для отрисовки интерфейса использовался библиотечный модуль Tkinter.

Модуль tkinter включает следующие классы:

- Button (кнопка)

Button - класс обычной кнопки. Общий синтаксис создания кнопки:

name = Button(window)

name - имя кнопки, window - имя окна, на котором она располагается.

- Radiobutton (радио-кнопка)

Radiobutton - класс радио - кнопки. Общий синтаксис создания кнопки:

name = Radiobutton(window)

name - имя радио-кнопки, window - имя окна, на котором она располагается.

- Checkbutton (флажок)

Checkbutton - класс кнопки-флажка (флажка). Общий синтаксис создания кнопки:

name = Checkbutton(window)

name - имя кнопки-флажка, window - имя окна, на котором он располагается.

- Entry (однострочное поле для ввода)

Entry - класс однострочного текстогого поля. Общий синтаксис создания однострочного текстового поля:

name = Entry(window)

name - имя однострочного текстового поля, window - имя окна, на котором оно располагается.

- Text (многострочное поле для ввода)

Text - класс многострочного текстогого поля. Общий синтаксис создания многострочного текстового поля:

name = Text(window)

name - имя многострочного текстового поля, window - имя окна, на котором оно располагается.

- Label (метка)

Label - класс метки. Общий синтаксис создания метки:

name = Label(window)

name - имя метки, window - имя окна, на котором она располагается.

- Scale (ползунок)

Scale - класс ползунка. Общий синтаксис создания ползунка:

name = Scale(window)

name - имя ползунка, window - имя окна, на котором он располагается.

- Scrollbar (полоса прокрутки)

Scrollbar - класс полосы прокрутки. Общий синтаксис создания полосы прокрутки:

name = Scrollbar(window)

name - имя полосы прокрутки, window - имя окна, на котором она располагается.

- Frame (виджет для группировки других виджетов)

Frame - класс фрейма. Общий синтаксис его создания:

name = Frame(window)

name - имя фрейма, window - имя окна, на котором он располагается.

- LabelFrame (аналог Frame, только с заголовком)

LabelFrame - класс фрейма с заголовком. Общий синтаксис его создания:

name = LabelFrame(window)

name - имя фрейма, window - имя окна, на котором он располагается.

- Listbox (список)

Listbox - класс списка. Общий синтаксис его создания:

name = Listbox(window)

name - имя списка, window - имя окна, на котором он располагается.

- Canvas (поле для рисования)

Canvas - класс полотна для рисования. Общий синтаксис создания полотна:

name = Canvas(window)

name - имя полотна для рисования, window - имя окна, на котором оно располагается.

- PanedWindow (элемент разделения окна)

PanedWindow - класс разделителя окна. Общий синтаксис его создания:

name = PanedWindow(window,orient=k)

name - имя разделителя окна; window - имя окна, на котором он располагается; k - его ориентация (VERTICAL - вертикальная, HORIZONTAL - горизонтальная).

- Menu (главное меню)

Menu - класс главного меню. Общий синтаксис создания экземпляра меню:

name_main = Menu(window)

window.config(menu=name_main)

name_main - имя экземпляра меню, window - имя окна, на котором оно располагается.

- Tk (главное единственное окно)

Tk - класс главной формы. Общий синтаксис создания окна:

name = Tk()

name - имя окна потомка.

- Toplevel (дочернее окно)

Toplevel - класс окна верхнего уровня. Общий синтаксис создания окна:

name = Button(window)

name - имя окна потомка, window - имя окна, потомком которого оно является.

- Курсор имеет два свойства:

event.xПозиция курсора по X во время события.

event.yПозиция курсора по Y во время события.

При разработке интерфейса программы использовались элементы: Tk, simpledialog, события курсора, bind, PhotoImage.

Модуль работы с камерой использует библиотеку OpenCV.

Захват видео с помощью камеры в реальном времени. OpenCV обеспечивает очень простой интерфейс. Чтобы захватить видео, необходимо создать объект VideoCapture . Его аргументом может быть индекс устройства, или имя видеофайла. Индекс устройства — это номер, указывающий, какая камера по счету подключена к компьютеру. Обычно всего одна камера будет подключена (как в моем случае). Поэтому я просто передаю 0 (или -1). После этого можно захватывать кадр за кадром, но в конце, необходимо освободить память с помощью cap.release(), cv2.destroyAllWindows().

Связующий модуль использует библиотеку pillow. Она предназначена для конвертации изображения полученного при помощи функций захвата камеры в OpenCV в изображение, поддерживаемое интерфейсом tkinter.

Модуль событий отслеживает нажатия мыши. При совпадении позиций курсора и площади маркера открывает диалоговое окно с предложением ввести текст для отправки. Отправляет полученный текст на сервер с присвоением определённому маркеру.

Наличие сетевого модуля обязательно для клиент-серверного приложения. Задачами этого модуля являются: обращение к удалённым ресурсам, получение информации от них, а также загрузка и передача данных.

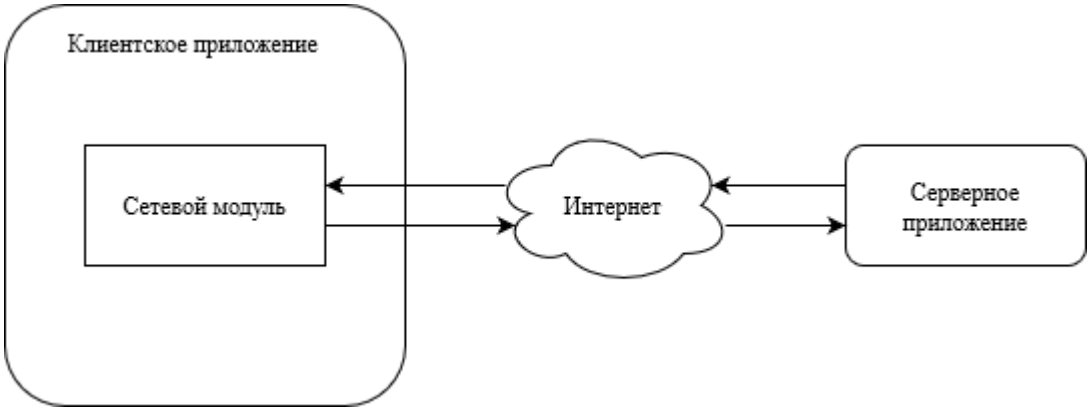

Рисунок 9. Схема сетевого модуля

Заключительным модулем является центральный модуль. Он является главным составляющим, можно сказать «ядром» программы, которое координирует всё взаимодействие между другими независимыми модулями.

За счёт такого модульного подхода к написанию приложения удаётся достичь как свободы и гибкости при модификации, так и централизованной структуры. Это очень удобно в больших проектах. Данный подход позволяет избежать одну очень неприятную проблему – при росте проекта, возрастает его функциональность, а, следовательно, запутанность, порой до такой степени, что для добавления простейшего функционала требуется переделывать большие участки уже функционирующего решения.

Можно считать, что применённый в результате модульный подход, позволяет избежать всех вышеописанных проблем и построение приложения по данной методике являлось правильным решением.

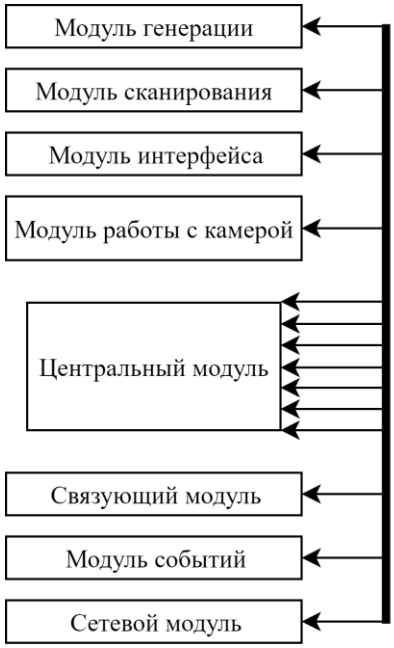

Рисунок 10. Модульная архитектура клиентского приложения

3.2 Структура серверной части

Так как к серверу не предъявляется больших требований (он выполняет только обработку текста), то его реализация является не столь сложной.

Для начала, рассмотрим немного терминологии. У сервера есть адрес (IP адрес), все компьютеры, подключенные к сети должны иметь хотя бы один. Адрес, как правило, состоит из четырёх чисел, разделённых точками, например, 127.0.0.1. Следующее, что имеет сервер – порт. Он сообщает программам, куда они должны посылать данные. Порты с номерами 1-1024 зарезервированы и не могут быть использованы для наших целей.

Можно использовать порты от 1024 до 65535. Сокет - это комбинация IP адреса и номера порта, которая однозначно определяет отдельный сетевой процесс во всей глобальной сети Интернет. Два сокета, один для хоста-получателя, другой для хоста-отправителя, определяют соединение для протоколов, ориентированных на установление связи, таких, как TCP и UDP. Сокеты можно использовать для приёма и передачи данных между программами.

Из библиотеки socket будем использовать одноимённую функцию для создания нового сокета. Синтаксис команды: socket(socket_family, socket_type). socket_family обычно имеет значение socket.AF_INET, означающее, что информация о порте и ip адресе будет передана в кортеже. socket_type в нашем случае, поскольку мы используем UDP, принимает значение socket.SOCK_DGRAM.

Метод bind связывает адрес и порт. На вход подаётся кортеж с адресом и портом. Синтаксис команды: bind((hostname, port)), где hostname – адрес сервера, port – порт.

UDP (User Datagram Protocol — пользовательский дейтаграмм-ный протокол). UDP позволяет приложениям отправлять инкапсулированные IP-дейтаграммы без установления соединений. UDP описан в RFC 768.

С помощью протокола UDP передаются сегменты, состоящие из 8-байтного заголовка, за которым следует поле полезной нагрузки. Два номера портов служат для идентификации конечных точек внутри отправляющей и принимающей машин. Когда прибывает пакет UDP, содержимое его поля полезной нагрузки передается процессу, связанному с портом назначения. В сущности, весь смысл использования UDP вместо обычного IP заключается как раз в указании портов источника и приемника. Без этих двух полей на транспортном уровне невозможно было бы определить действие, которое следует произвести с пакетом. В соответствии с полями портов производится корректная доставка сегментов.

Рисунок 11. Структура UDP пакета.

Одной из областей, где UDP применяется особенно широко, является область клиент-серверных приложений. Зачастую клиент посылает короткий запрос серверу и

надеется получить короткий ответ. Если запрос или ответ теряется, клиент по прошествии определенного временного интервала может попытаться еще раз. Это позволяет не только упростить код, но и уменьшить требуемое количество сообщений по сравнению с протоколами, которым требуется начальная настройка.

Изначально сервер ждёт запроса, это достигается путём использования функции s.recvfrom(1024). На выходе получаем данные, которые были переданы и обратный адрес. Поскольку, начиная с версии Python 3, строки в Python кодируются не в UTF-8, то после каждого приёма и перед каждой передачей всю текстовую информацию необходимо декодировать или кодировать в UTF-8.

После получения запроса сервер начинает его обработку. Запросы могут быть трёх типов:

- запрос на обновление БД (должен быть использован после добавления маркера на сервер);

- запрос на получение всей информации, ассоциированной с определённым идентификатором;

- запрос на добавление информации к какому-либо идентификатору.

После успешного выполнения запроса клиенту поступает сообщение «OK» символизирующее то, что всё в порядке. В случае с запросом всей информации отправляется вся имеющаяся информация.

В случае, если запрос не был выполнен имеется три кода ошибки:

- ERROR_1 – ошибка, означающая то, что длинна информационного сообщения, которое должно добавиться к определённому идентификатору, либо слишком велика, либо нулевая. Ограничение на количество символов в сообщении 50.

- ERROR_2 – ошибка, означающая, что запрошенный клиентом идентификатор отсутствует в базе данных.

- ERROR_3 – ошибка, указывающая на то, что запрос был некорректно сформирован, к тому же в сообщении нельзя использовать знак двоеточия, поскольку символы после двоеточия не будут обработаны.

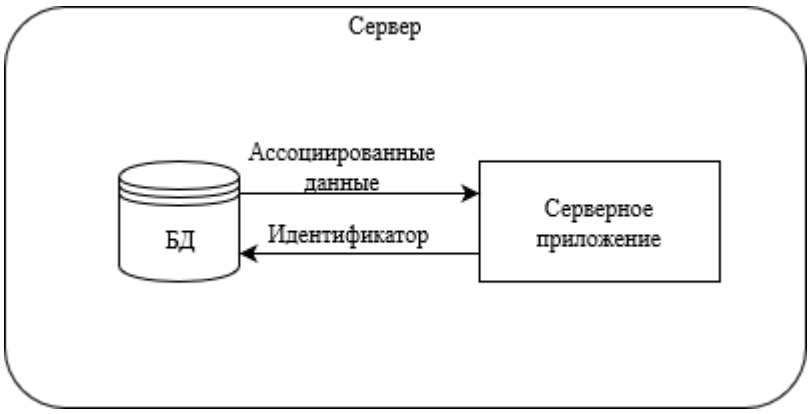

Рисунок 12. Схематическое изображение работы сервера

4. Разработка программных средств

Программная система дополненной реальности с многопользовательским режимом работает следующим образом: клиентские устройства распознают заранее подготовленные метки, их идентификатор отправляется на сервер с запросом на получение информации о данной метке, после чего сервер отправляет данные, либо код ошибки в случае неудачи. При нажатии на метку появляется окно ввода сообщения. После ввода сообщения, информация отправляется на сервер и присваивается определённому идентификатору. Сервер в ответ отправляет подтверждение об успешной операции или код ошибки.

Внешний вид интерфейса программы представлен на рис 13, 14.

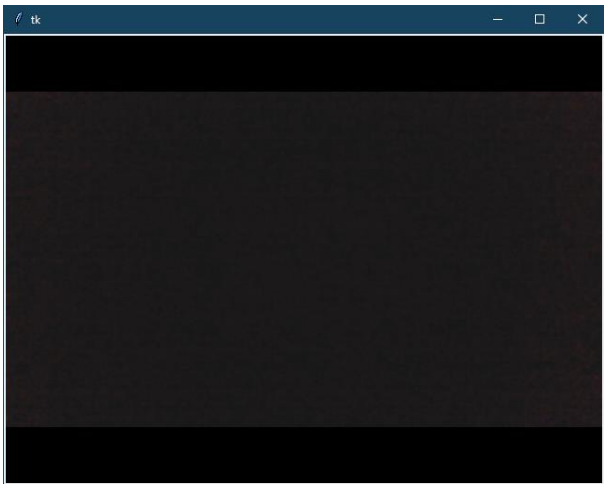

Рисунок 13. Пример интерфейса программы

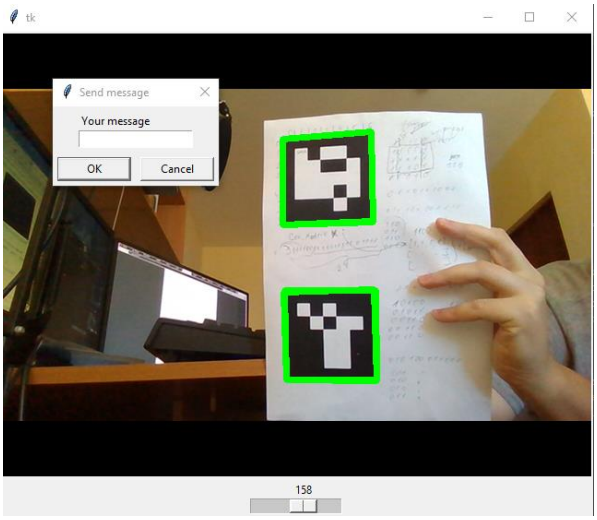

Рисунок 14. Окно ввода сообщения

Для того, чтобы видеть с чем будет работать алгоритм распознавания используется отладочное окно (рис. 15).

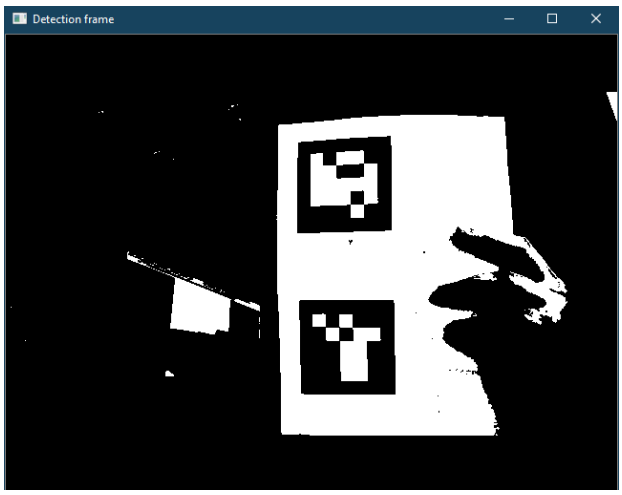

Рисунок 15. Отладочное окно программы

Схематическое представление работы программной системы представлено на рис. 16.

Рисунок 16. Схема работы

Удалённым сервером может являться компьютер, на любой операционной системе, поддерживающей Python. Компьютер также должен иметь доступ в интернет или быть в одной локальной сети с клиентом.

Сервер представляет из себя простое консольное приложение, не имеющее графического интерфейса (рис. 17).

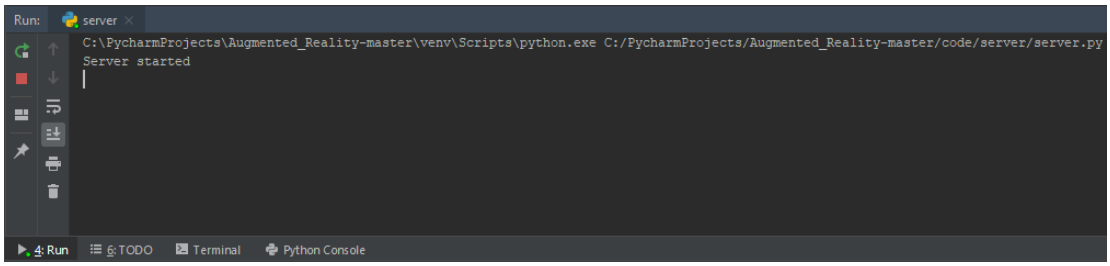

Рисунок 17. Выполнение серверной программы

Как говорилось ранее, система состоит из двух частей: клиентской и серверной. Клиентская часть включает в себя следующие файлы: ar_markers_skan.py, ar_generate_marker.py, detect.py, coding.py, marker.py. В файле ar_markers_skan.py реализуются следующие функции: show_frame – отображение текущего фрейма

изображения с использованием интерфейса tkinter, getorigin – функция, вызываемая по событию нажатия кнопки мыши, нужна для получения позиции курсора относительно отображаемого кадра, main – основное тело программы. Файл ar_generate_marker.py нужен только для генерации маркера и в нём присутствует только main-функция для этой цели. В файле detect.py реализованы две функции: validate_and_turn – функция, определяющая ориентацию маркера, detect_markers – функция, для обнаружения маркера на изображении. Файл coding.py описывает процессы кодирования и декодирования Кодов Хэмминга и содержит 6 функций: encode – функция кодирования, decode – функция декодирования, parity_correct – проверка кода на ошибки, matrix_array_multiply_and_format – функция перемножения матриц и их представления в формате, удобном для дальнейшей работы, generate_bit_array – разбивает полученную последовательность на элементы и записывает в массив (т.е. последовательность 0010101 будет представлена в виде [0, 0, 1, 0, 1, 0, 1]), extract_hamming_code – извлекает Код Хэмминга из поступившей на вход матрицы. Файл marker.py – содержит класс HamingMarker описывающий маркер Хэмминга, включает переменные id - идентификатор и contours – координаты углов для отрисовки контуров. Помимо переменных класс имеет 7 функций: center – вычисление центра маркера исходя из координат углов, generate_image – генерирует изображение с маркером, используется в том числе и в ar_generate_marker.py, draw_contour – функция отрисовки контура метки, highlite_marker – функция которая подсвечивает маркер (рисует контур) и накладывает необходимый текст, generate – функция создающая новый объект класса HamingMarker со случайным идентификатором, id_as_binary – функция для перевода числа из десятичной системы в двоичную, haming_code – функция кодирующая идентификатор.

Серверная часть состоит из одного файла server.py и использует библиотеки sockets и os. В файле описаны две функции: update – обновление базы данных маркеров, используется, если на сервер, во время его работы, был загружен новый маркер, main – тело программы, тут описан алгоритм работы сервера реализующий инициализацию и обработку ошибок.

5. Тестирование системы

5.1. Описание набора данных

Для создания тестовой базы данных был использован заранее написанный генератор маркеров. В результате работы генератора было создано четыре метки со случайными идентификаторами. Все маркеры, представленные в базе данных, имеют формат PNG. Этот формат был выбран так как имеет минимальные потери при сжатии (т.е. качество изображения не меняется при любой степени сжатия) и файлы, созданные в этом формате, имеют небольшой объём. Пример получившихся изображений приведён на рис. 18.

Рисунок 18. Маркеры, используемые в тестовой базе данных

5.2 Тестирование серверной части приложения

Для удобства тестирования сервера был написан простой текстовый клиент, способный отправлять команды через командную строку. Всего будет проверено 4 варианта событий.

Вариант 1, запрос сформирован корректно, произведено получение информации и она была сохранена в базе данных. Для эмуляции данного события возьмём идентификатор, имеющийся в базе данных сервера, например, 1355. Пошлём на сервер запрос формата

«1355:Входящее сообщение.». Ожидаем ответа от сервера в виде сообщения с фразой «OK».

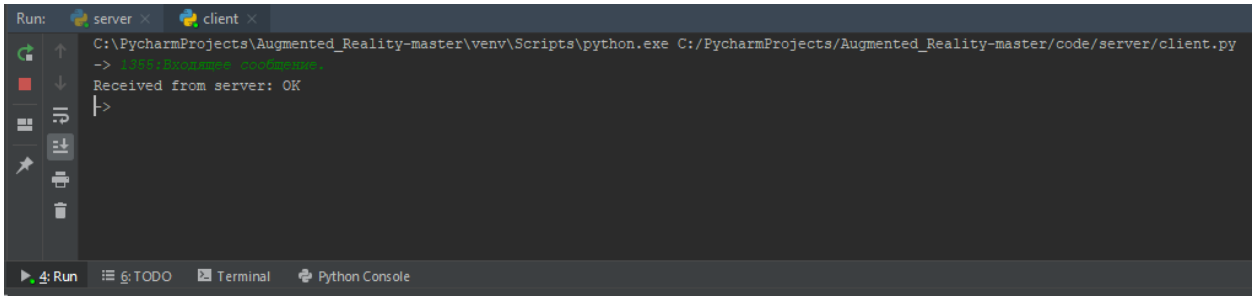

Рисунок 19. Лог из консоли клиента

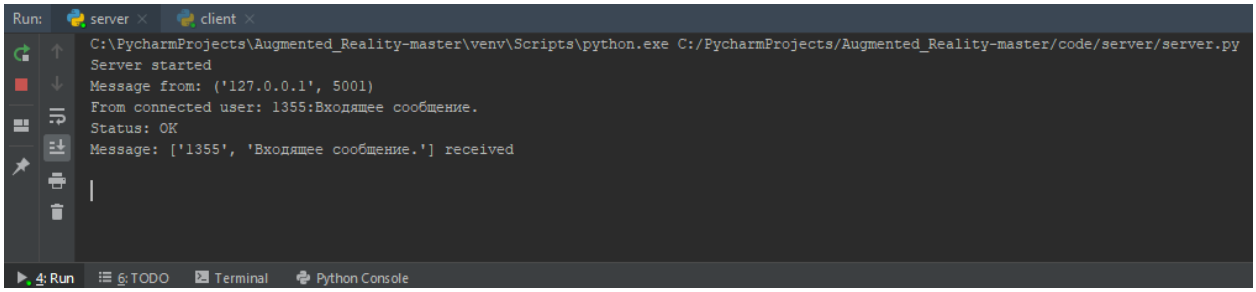

Рисунок 20. Лог из консоли сервера

На рис. 19, 20 видно, что сообщение было отправлено и получено сервером, сервер в свою очередь отправил сообщение подтверждения.

Теперь, попробуем запросить, у сервера информацию, связанную с идентификатором 1355, формируем запрос get:1355, ожидаем увидеть сообщение формата «Входящее сообщение.».

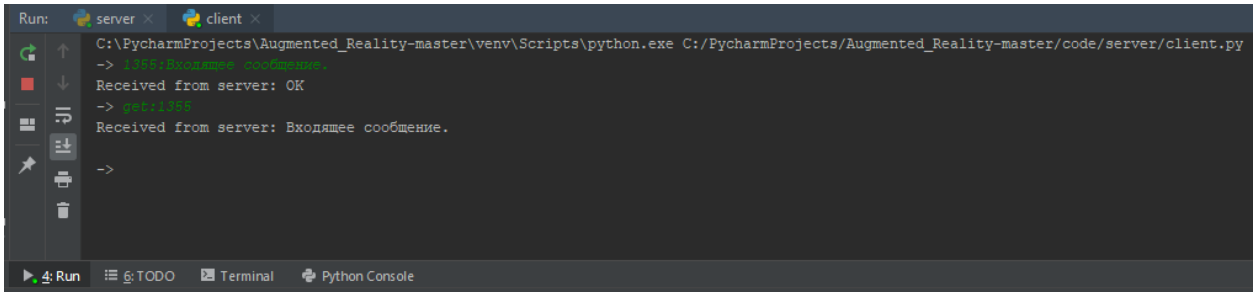

Рисунок 21. Лог из консоли клиента

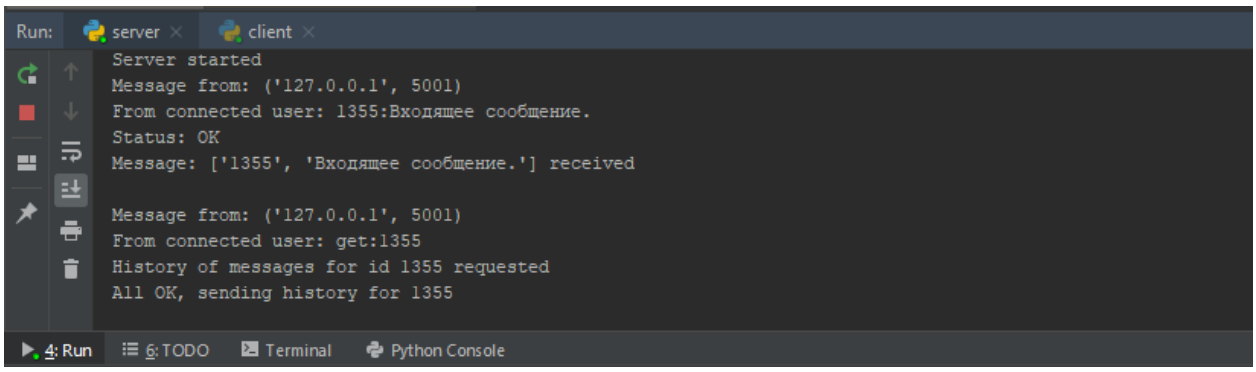

Рисунок 22. Лог из консоли сервера

На рис. 21, 22 показано, что после запроса, сервер прислал ответ с сообщением «Входящее сообщение.». Результат соответствует ожиданиям.

Вариант 2, запрос сформирован некорректно, был получен запрос с некорректным идентификатором (например, его не существует в базе данных), ожидаемый ответ от сервера «ERROR_2». Для проверки отправим запрос с несуществующим идентификатором, например, get:2222.

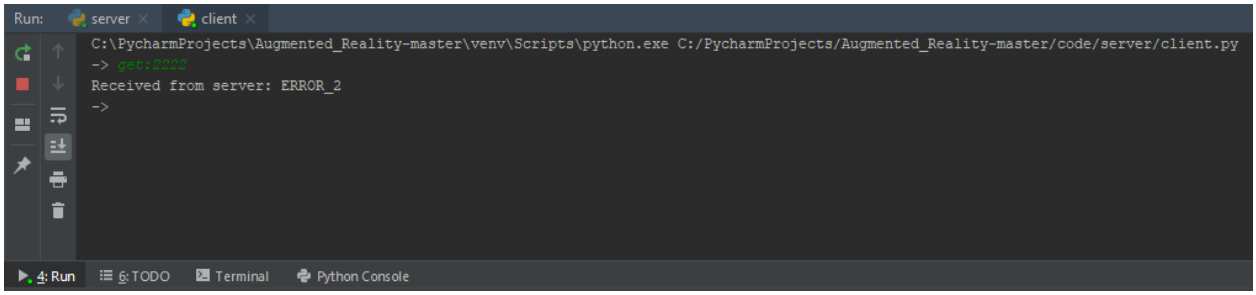

Рисунок 23. Лог из консоли клиента

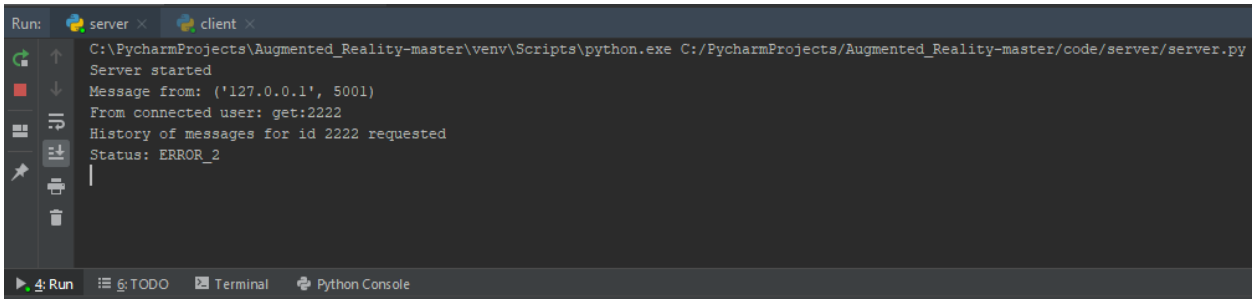

Рисунок 24. Лог из консоли сервера

В результате, от сервера получен ответ «ERROR_2» (рис. 23, 24), что полностью соответствует заявленным ожиданиям.

Вариант 3, запрос сформирован некорректно, получен запрос со слишком длинным сообщением (т.е. больше 50 символов), ожидаемый ответ «ERROR_1». Для проверки

данного события отправим запрос формата 1355:gEa9fBQwrrpaw8W2RVgX88DsGfG0g3CGYzBTjOeJbqfmBTI6tMx.

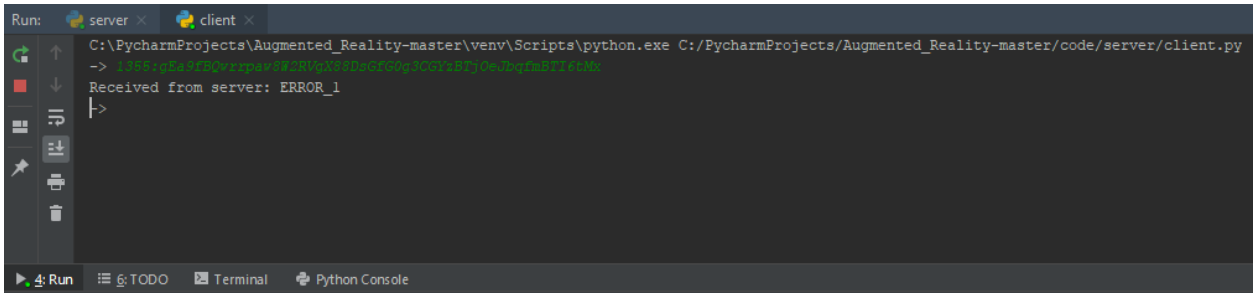

Рисунок 25. Лог из консоли клиента

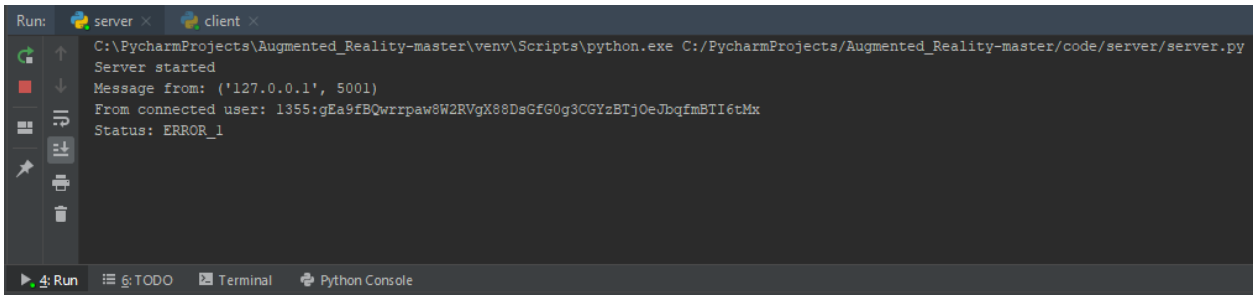

Рисунок 26. Лог из консоли сервера

В результате отправки данного запроса сервер выдал код ошибки «ERROR_1» (рис. 25, 26), что соответствует ожиданиям.

Вариант 4, запрос сформирован некорректно, в запросе не стоит разделитель в виде двоеточия. Данная ошибка может возникнуть только при неправильной реализации запросов в клиентском приложении, т.е. рядовой пользователь с ней столкнуться не должен. Тем не менее такой вариант тоже стоит предусмотреть. Итак, отправим запрос с содержимым «get», ожидаем ответ от сервера с кодом ошибки «ERROR_3».

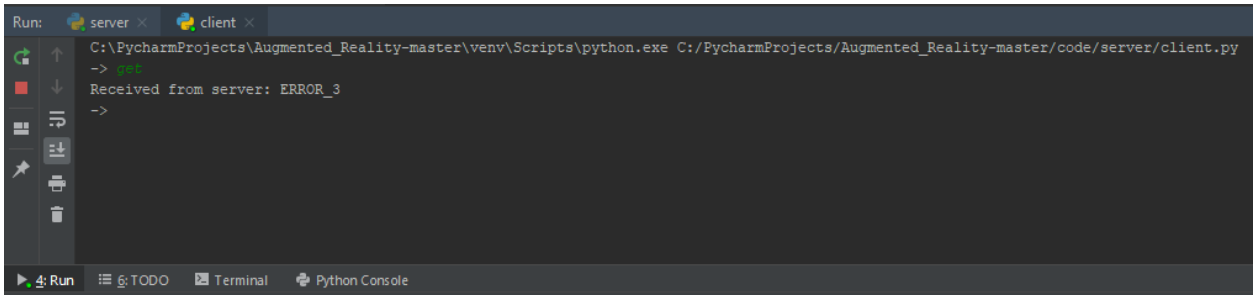

Рисунок 27. Лог из консоли клиента

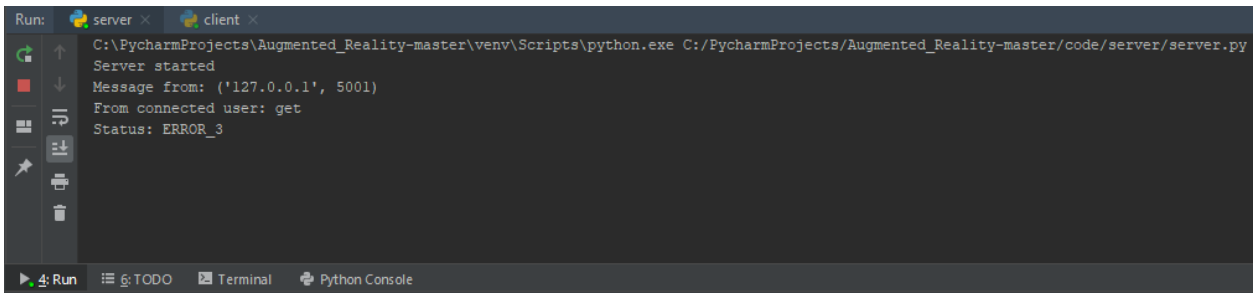

Рисунок 28. Лог из консоли сервера

Как показано на рис. 27, 28 в ответ на запрос клиента, сервер послал код ошибки «ERROR_3», что соответствует ожиданиям.

5.3 Тестирование клиентской части приложения

Проверка клиента не очень сложна. Всё что требуется, так это убедиться в том, что маркер распознаётся (т.е. он должен выделиться), появляется окно ввода сообщения, сообщение приходит на сервер, сообщение отображается поверх маркера (т.е. приходит информация об ассоциациях с конкретным идентификатором).

На рис. 29 видно, что маркеры выделяются, следовательно, они распознаются приложением.

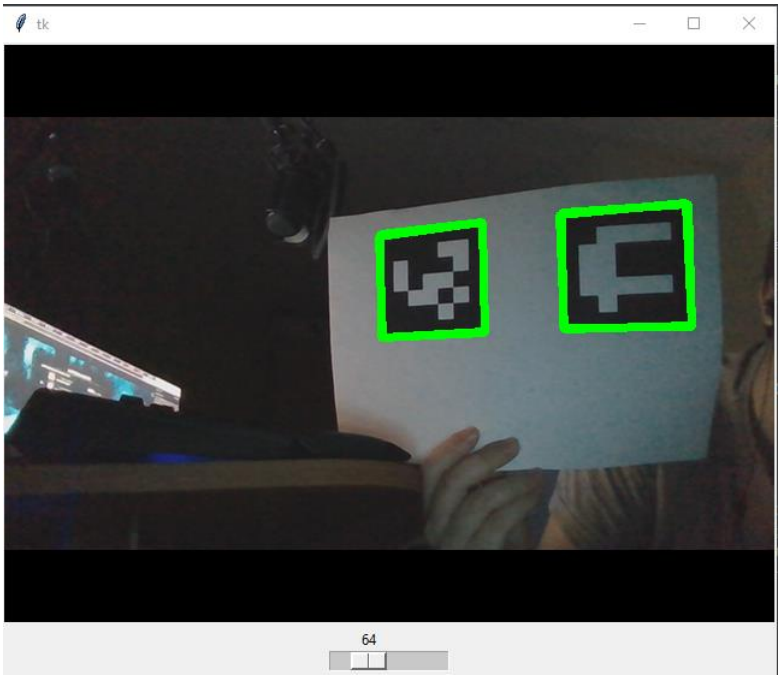

Рисунок 29. Результат распознавания маркера

На рис. 30 видно, что окно ввода после нажатия на маркер открылось, значит, всё в порядке.

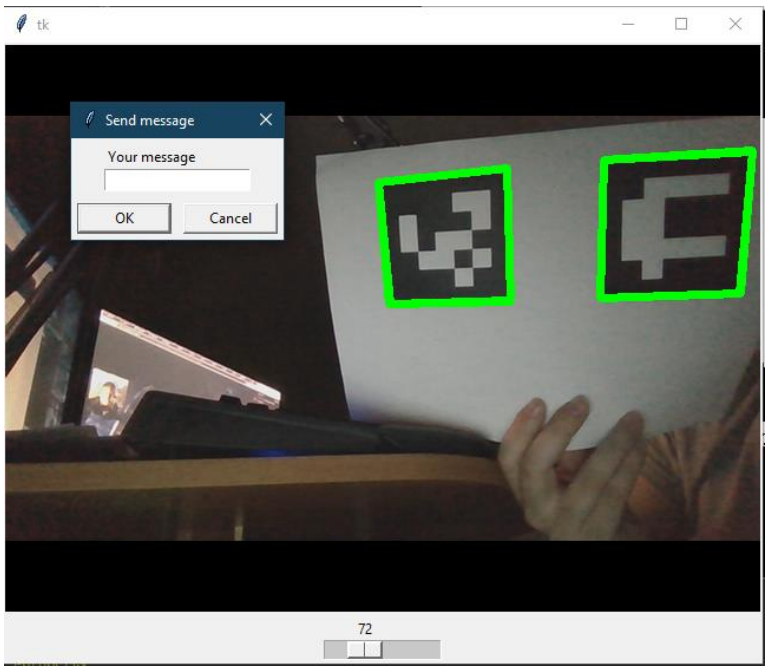

Рисунок 30. Демонстрация открытия диалогового окна

Отправим на два разных маркера сообщения, в первом сообщении будет написано «Hello», а во втором «world!». На рис. 31, 32 видно, что на сервер пришли сообщения для определённых маркеров. На рис. 33 показано, что сообщения успешно приходят с сервера и отображаются в интерфейсе программы.

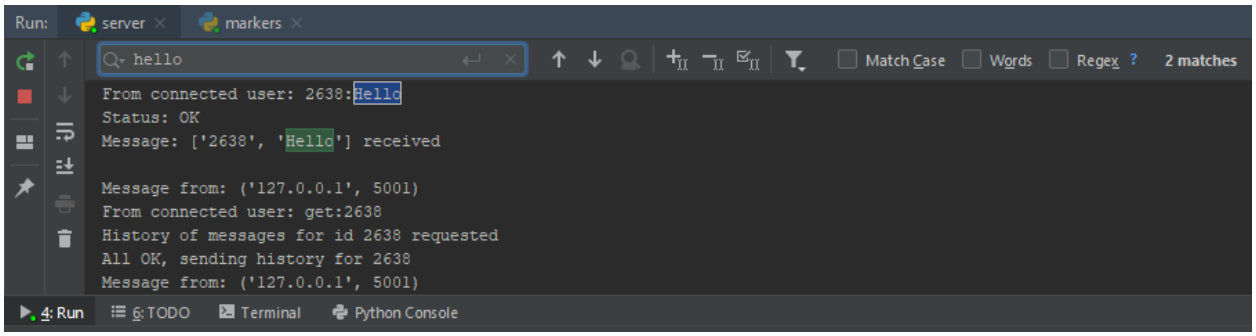

Рисунок 31. Лог из консоли сервера

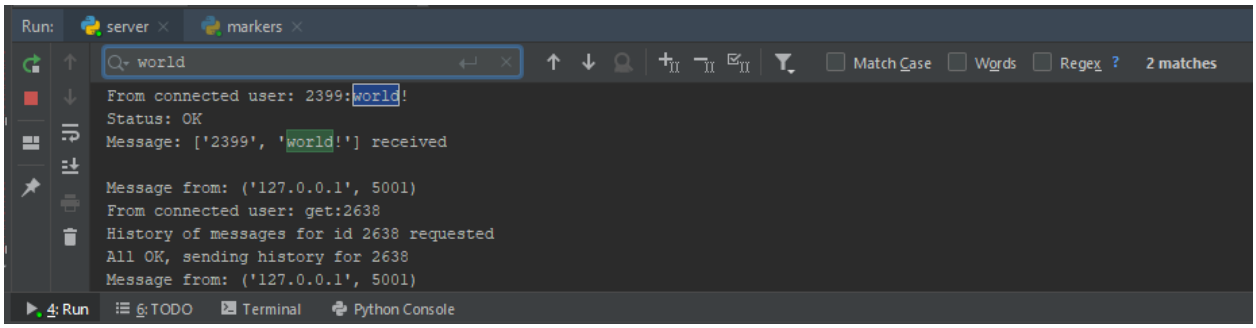

Рисунок 32. Лог из консоли сервера

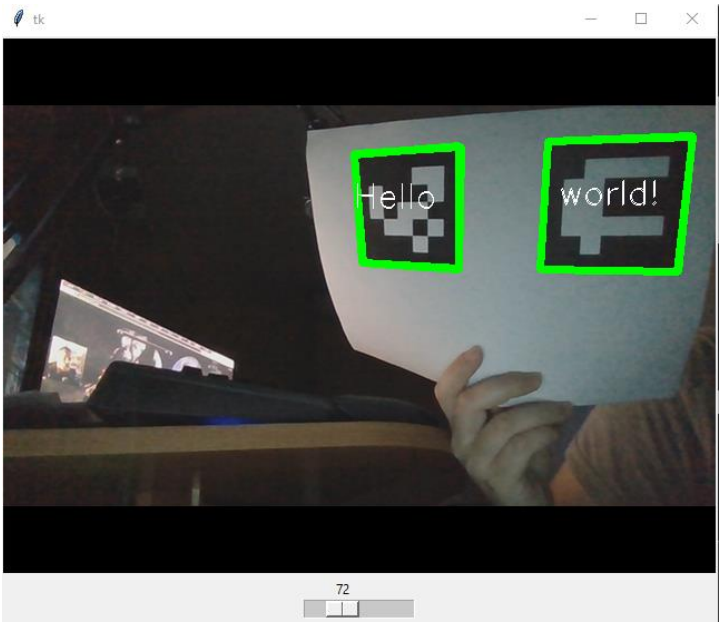

Рисунок 33. Демонстрация полученной информации

Заключение

В результате выполнения выпускной квалификационной работы была спроектирована и реализована программная система дополненной реальности с многопользовательским режимом, которая включает в себя два компонента: клиентское приложение для устройств на базе Windows 10 и серверное приложение для компьютеров с установленным интерпретатором Python.

Созданный программный продукт отвечает всем поставленным требованиям и решает все необходимые задачи. Тестирование данного продукта подтвердило его работоспособность и возможность дальнейшего использования для решения подобных задач.

Список литературы

1. "Speeded up detection of squared fiducial markers", Francisco J.Romero-Ramirez, Rafael Muñoz-Salinas, Rafael Medina-Carnicer, Image and Vision Computing, vol 76, pages 38-47, year 2018

2. "Generation of fiducial marker dictionaries using mixed integer linear programming",S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Pattern Recognition:51, 481-491,2016

3. Код Хэмминга. Пример работы алгоритма // Хабр URL: https://habr.com/ru/post/140611/

4. Код_Хэмминга // Википедия – свободная энциклопедия ru.wikipedia.org URL: https://ru.wikipedia.org/wiki/Код_Хэмминга

5. Detection of ArUco Markers // OpenCV URL: https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html

6. ArUco: a minimal library for Augmented Reality applications based on OpenCV // Aplicaciones de la Visión Artificial URL: http://www.uco.es/investiga/grupos/ava/node/26

7. A new approach to decode a black and white marker // iplimage (archived) URL: https://web.archive.org/web/20180104190717/http://iplimage.com/blog/approach-encodedecode-black-white-marker/

Приложение

ar_markers_skan.py

```
#!/usr/bin/env python

from __future__ import print_function
import cv2
import socket
from ar_markers import detect_markers
import PIL
from PIL import Image, ImageTk
import pytesseract
import cv2
from tkinter import *
from tkinter import simpledialog
from tkinter import Scale
import math

capture = cv2.VideoCapture(0)
radius = 20
detected_markers = {}
print(detected_markers.values())

root = Tk()
root.bind('<Escape>', lambda e: root.quit())
lmain = Label(root)
lmain.pack()
slide1 = Scale(root, from_=0, to=255, resolution=2, orient=HORIZONTAL)
slide1.set(128)
slide1.pack()

def show_frame(frame):
    # _, frame = capture.read()
    # frame = cv2.flip(frame, 1)
    cv2image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGBA)
    img = PIL.Image.fromarray(cv2image)
    imgtk = ImageTk.PhotoImage(image=img)
    lmain.imgtk = imgtk
    lmain.configure(image=imgtk)
    # lmain.after(10, show_frame, frame)

# Determine the origin by clicking
def getorigin(eventorigin):
    global x0,y0
    x0 = eventorigin.x
    y0 = eventorigin.y

    for m_key, m_value in detected_markers.items():
```

```
         if math.sqrt((x0 - m_value[0]) ** 2 + (y0 - m_value[1]) ** 2) <
radius:
                message = simpledialog.askstring("Send message", "Your
message")
                if message is not None:
                    message = f"{m_key}:{message}"
                    s.sendto(message.encode('utf-8'), server)
                    data, addr = s.recvfrom(1024)
                    data = data.decode('utf-8')
                    if "OK" in data:
                            print(message)
                            print(f"OK id: {m_key}")
                    else:
                            print("ERROR")
                else:
                    print("Empty message!")
    print(x0,y0)

if __name__ == '__main__':
        host = '127.0.0.1'
        port = 5001

        server = ('127.0.0.1', 5000)

        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.bind((host, port))
        lmain.bind("<Button 1>", getorigin)

        print('Press "q" to quit')

        if capture.isOpened():  # try to get the first frame
                frame_captured, frame = capture.read()
        else:
                frame_captured = False

        while frame_captured:
                img = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
                ret, img = cv2.threshold(img, slide1.get(), 255,
cv2.THRESH_BINARY)
                # img = cv2.adaptiveThreshold(img, slide1.get(),
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
                markers = detect_markers(img)
                if not markers:
                        detected_markers = {}
                for marker in markers:
                        detected_markers[marker.id] = marker.center
                        # print(marker.contours)
                        try:
                                message = f"get:{marker.id}"
```

```
                                s.sendto(message.encode('utf-8'), server)
                                data, addr = s.recvfrom(1024)
                                data = data.decode('utf-8')
                                if data == 'ERROR_1' or data == 'ERROR_2' or
data == 'ERROR_3':
                                    print('ERROR')
                                else:
                                    marker.highlite_marker(frame,
text_color=(255, 255, 255), text=data)
                        except ConnectionResetError as e:
                                print(f"Server unavailable: {e}")
                        # print(detected_markers)

                show_frame(frame)
                root.update_idletasks()
                root.update()
                cv2.imshow('Detection frame', img)
                # cv2.imshow('Test Frame', frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                        break
                frame_captured, frame = capture.read()

        # When everything done, release the capture
        capture.release()
        cv2.destroyAllWindows()
```

ar_generate_marker.py

```
import sys
from cv2 import imwrite

from ar_markers import HammingMarker


if __name__ == '__main__':
    if len(sys.argv) > 1:
        if sys.argv[1] == '--generate':
            for i in range(int(sys.argv[2])):
                marker = HammingMarker.generate()
                imwrite('marker_{}.png'.format(marker.id),
marker.generate_image())
                print("Generated Marker with ID {}".format(marker.id))
        else:
            marker = HammingMarker(id=int(sys.argv[1]))
            imwrite('marker_{}.png'.format(marker.id),
marker.generate_image())
            print("Generated Marker with ID {}".format(marker.id))
    else:
        marker = HammingMarker.generate()
        imwrite('marker_{}.png'.format(marker.id), marker.generate_image())
```

```
        print("Generated Marker with ID {}".format(marker.id))
    print('Done!')
```

server.py

```
import socket
import os

def update(id):
    """Function for id detection"""
    lst = os.listdir("./markers")
    for i in lst:
        id.append(int(i.split('_')[1].split('.')[0]))
    return id

def Main():
    host = '127.0.0.1'
    port = 5000

    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind((host, port))

    id = []
    update(id)

    # database initialization
    # key - id of marker, value - history of messages separated by \n
    db = {}
    for i in id:
        db[i] = ""

    # first and last symb delete [1:-1]
    print("Server started")
    # ERROR_1 - Incorrect message
    # ERROR_2 - Id not exists
    # ERROR_3 - Incorrect separators
    # OK - No errors
    while True:
        data, addr = s.recvfrom(1024)
        data = data.decode('utf-8')
        print(f"Message from: {addr}")
        print(f"From connected user: {data}")
        if data == "update":
            print("Update request received, updating...")
            update(id)
            s.sendto("OK".encode('utf-8'), addr)
            continue
        elif data[:4] == "get:":
            req_id = int(data.split(':')[1])
            print(f"History of messages for id {req_id} requested")
```

```
            if req_id in id:
                print(f"All OK, sending history for {req_id}")
                s.sendto(db[req_id].encode('utf-8'), addr)
                continue
            else:
                print(f"Status: ERROR_2")
                s.sendto("ERROR_2".encode('utf-8'), addr)
                continue
        else:
            if ':' in data:
                data = data.split(':')
                if int(data[0]) in id:
                    if str(data[1]).__len__() != 0 and str(data[1]).__len__()
< 50:
                        print(f"Status: OK\nMessage: {data} received\n")
                        db[int(data[0])] += f"{data[1]}\n"
                        s.sendto("OK".encode('utf-8'), addr)
                        continue
                    else:
                        print(f"Status: ERROR_1")
                        s.sendto("ERROR_1".encode('utf-8'), addr)
                        continue
                else:
                    print(f"Status: ERROR_2")
                    s.sendto("ERROR_2".encode('utf-8'), addr)
                    continue
            else:
                print(f"Status: ERROR_3")
                s.sendto("ERROR_3".encode('utf-8'), addr)
                continue

if __name__ == '__main__':
    Main()
```

coding.py

```
from numpy import matrix, array

GENERATOR_MATRIX = matrix([
    [1, 1, 0, 1],
    [1, 0, 1, 1],
    [1, 0, 0, 0],
    [0, 1, 1, 1],
    [0, 1, 0, 0],
    [0, 0, 1, 0],
    [0, 0, 0, 1],
])

REGENERATOR_MATRIX = matrix([
    [0, 0, 1, 0, 0, 0, 0],
```

```
    [0, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 1],
])

PARITY_CHECK_MATRIX = matrix([
    [1, 0, 1, 0, 1, 0, 1],
    [0, 1, 1, 0, 0, 1, 1],
    [0, 0, 0, 1, 1, 1, 1],
])

HAMMINGCODE_MARKER_POSITIONS = [
    [1, 2], [1, 3], [1, 4],
    [2, 1], [2, 2], [2, 3], [2, 4], [2, 5],
    [3, 1], [3, 2], [3, 3], [3, 4], [3, 5],
    [4, 1], [4, 2], [4, 3], [4, 4], [4, 5],
    [5, 2], [5, 3], [5, 4],
]

def encode(bits):
    encoded_code = ''
    if len(bits) % 4 != 0:
        raise ValueError('Only a multiple of 4 as bits are allowed.')
    while len(bits) >= 4:
        four_bits = bits[:4]
        bit_array = generate_bit_array(four_bits)
        hamming_code   =   matrix_array_multiply_and_format(GENERATOR_MATRIX,
bit_array)
        encoded_code += ''.join(hamming_code)
        bits = bits[4:]
    return encoded_code


def decode(bits):
    decoded_code = ''
    if len(bits) % 7 != 0:
        raise ValueError('Only a multiple of 7 as bits are allowed.')
    for bit in bits:
        if int(bit) not in [0, 1]:
            raise ValueError('The provided bits contain other values that 0 or
1: %s' % bits)
    while len(bits) >= 7:
        seven_bits = bits[:7]
        uncorrected_bit_array = generate_bit_array(seven_bits)
        corrected_bit_array = parity_correct(uncorrected_bit_array)
        decoded_bits   =   matrix_array_multiply_and_format(REGENERATOR_MATRIX,
corrected_bit_array)
        decoded_code += ''.join(decoded_bits)
        bits = bits[7:]
```

```
    return decoded_code


def parity_correct(bit_array):
    # Check the parity using the PARITY_CHECK_MATRIX
    checked_parity   =   matrix_array_multiply_and_format(PARITY_CHECK_MATRIX,
bit_array)
    parity_bits_correct = True
    # every value as to be 0, so no error accoured:
    for bit in checked_parity:
        if int(bit) != 0:
            parity_bits_correct = False
    if not parity_bits_correct:
        error_bit = int(''.join(checked_parity), 2)
        for index, bit in enumerate(bit_array):
            if error_bit == index + 1:
                if bit == 0:
                    bit_array[index] = 1
                else:
                    bit_array[index] = 0
    return bit_array


def matrix_array_multiply_and_format(matrix, array):
    unformated = matrix.dot(array).tolist()[0]
    return [str(bit % 2) for bit in unformated]


def generate_bit_array(bits):
    return array([int(bit) for bit in bits])


def extract_hamming_code(mat):
    hamming_code = ''
    for pos in HAMMINGCODE_MARKER_POSITIONS:
        hamming_code += str(int(mat[pos[0], pos[1]]))
    return hamming_code
```

marker.py

```
from __future__ import print_function
try:
    import cv2
except ImportError:
    raise Exception('Error: OpenCv is not installed')
from PIL import ImageDraw, Image, ImageFont

from numpy import mean, binary_repr, zeros
from numpy.random import randint
# from scipy.ndimage import zoom  # get rid of this! - replaced w/ cv2.Resize
```

```
from ar_markers.coding import encode, HAMMINGCODE_MARKER_POSITIONS

MARKER_SIZE = 7


class HammingMarker(object):
    def __init__(self, id, contours=None):
        self.id = id
        self.contours = contours

    def __repr__(self):
        return '<Marker id={} center={}>'.format(self.id, self.center)

    @property
    def center(self):
        if self.contours is None:
            return None
        center_array = mean(self.contours, axis=0).flatten()
        return (int(center_array[0]), int(center_array[1]))

    def generate_image(self):
        img = zeros((MARKER_SIZE, MARKER_SIZE))
        img[1, 1] = 255  # set the orientation marker
        for index, val in enumerate(self.hamming_code):
            coords = HAMMINGCODE_MARKER_POSITIONS[index]
            if val == '1':
                val = 255
            img[coords[0], coords[1]] = int(val)
        # return zoom(img, zoom=50, order=0)
        height, width = img.shape[:2]
        res       =       cv2.resize(img,       (50*width,       50*height),
interpolation=cv2.INTER_NEAREST)
        return res

    def draw_contour(self, img, color=(0, 255, 0), linewidth=5):
        cv2.drawContours(img, [self.contours], -1, color, linewidth)

    def highlite_marker(self, img, contour_color=(0, 255, 0), text_color=(255,
0, 0), linewidth=5, text_thickness=1, text = str(id)):
        """
        This draws a bounding box around the marker on the image. NOTE: it
returns
        a BGR image so the highlite is in color.

        Input:
          img: image with detected marker
          contour_color: bounding box color, default is Green (0,255,0)
          text_color: text color, default is Blue (255,0,0)
```

```
            linewidth: thickness of bonding box line
            text_thickness: thickness of marker number text

        Output:
            A color image with the marker drawn on it
        """
        # this is a grayscale, so make it a color image
        if len(img.shape) == 2:
            img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
        self.draw_contour(img, color=contour_color, linewidth=linewidth)
        dy = 25
        draw = ImageDraw.Draw(Image.fromarray(img))
        font = ImageFont.truetype('./Arial.ttf')
        for i, line in enumerate(text.split('\n')):
            y = self.center[1] + i*dy
            #  draw.text((self.center[0]  -  50,  y),  str(line),  font=font,
fill=(255, 255, 255))
            cv2.putText(img,    line,    (self.center[0]    -    50,    y),
cv2.FONT_HERSHEY_SIMPLEX, text_thickness, text_color)
        return img

    @classmethod
    def generate(cls):
        return HammingMarker(id=randint(4096))

    @property
    def id_as_binary(self):
        return binary_repr(self.id, width=12)

    @property
    def hamming_code(self):
        return encode(self.id_as_binary)
```

detect.py

```
from __future__ import print_function
from __future__ import division

try:
    import cv2
except ImportError:
    raise Exception('Error: OpenCv is not installed')

from numpy import array, rot90

from ar_markers.coding import decode, extract_hamming_code
from ar_markers.marker import MARKER_SIZE, HammingMarker

BORDER_COORDINATES = [
```

```
    [0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [1, 0], [1, 6], [2,
0], [2, 6], [3, 0],
    [3, 6], [4, 0], [4, 6], [5, 0], [5, 6], [6, 0], [6, 1], [6, 2], [6, 3], [6,
4], [6, 5], [6, 6],
]

ORIENTATION_MARKER_COORDINATES = [[1, 1], [1, 5], [5, 1], [5, 5]]


def validate_and_turn(marker):
    # first, lets make sure that the border contains only zeros
    for crd in BORDER_COORDINATES:
        if marker[crd[0], crd[1]] != 0.0:
            raise ValueError('Border contians not entirely black parts.')
    # search for the corner marker for orientation and make sure, there is only
1
    orientation_marker = None
    for crd in ORIENTATION_MARKER_COORDINATES:
        marker_found = False
        if marker[crd[0], crd[1]] == 1.0:
            marker_found = True
        if marker_found and orientation_marker:
            raise ValueError('More than 1 orientation_marker found.')
        elif marker_found:
            orientation_marker = crd
    if not orientation_marker:
        raise ValueError('No orientation marker found.')
    rotation = 0
    if orientation_marker == [1, 5]:
        rotation = 1
    elif orientation_marker == [5, 5]:
        rotation = 2
    elif orientation_marker == [5, 1]:
        rotation = 3
    marker = rot90(marker, k=rotation)
    return marker


def detect_markers(img):
    """
    This is the main function for detecting markers in an image.

    Input:
      img: a color or grayscale image that may or may not contain a marker.

    Output:
      a list of found markers. If no markers are found, then it is an empty
list.
    """
```

```
    if len(img.shape) > 2:
        width, height, _ = img.shape
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        width, height = img.shape
        gray = img

    edges = cv2.Canny(gray, 10, 100)
    contours, hierarchy = cv2.findContours(edges.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)[-2:]

    # We only keep the long enough contours
    min_contour_length = min(width, height) / 50
    contours = [contour for contour in contours if len(contour) >
min_contour_length]
    warped_size = 49
    canonical_marker_coords = array(
        (
            (0, 0),
            (warped_size - 1, 0),
            (warped_size - 1, warped_size - 1),
            (0, warped_size - 1)
        ),
        dtype='float32')

    markers_list = []
    for contour in contours:
        approx_curve = cv2.approxPolyDP(contour, len(contour) * 0.01, True)
        if not (len(approx_curve) == 4 and cv2.isContourConvex(approx_curve)):
            continue

        sorted_curve = array(
            cv2.convexHull(approx_curve, clockwise=False),
            dtype='float32'
        )
        persp_transf = cv2.getPerspectiveTransform(sorted_curve,
canonical_marker_coords)
        warped_img = cv2.warpPerspective(img, persp_transf, (warped_size,
warped_size))

        # do i really need to convert twice?
        if len(warped_img.shape) > 2:
            warped_gray = cv2.cvtColor(warped_img, cv2.COLOR_BGR2GRAY)
        else:
            warped_gray = warped_img

        _, warped_bin = cv2.threshold(warped_gray, 127, 255, cv2.THRESH_BINARY)
        marker = warped_bin.reshape(
```

```
            [MARKER_SIZE, warped_size // MARKER_SIZE, MARKER_SIZE, warped_size
// MARKER_SIZE]
        )
        marker = marker.mean(axis=3).mean(axis=1)
        marker[marker < 127] = 0
        marker[marker >= 127] = 1

        try:
            marker = validate_and_turn(marker)
            hamming_code = extract_hamming_code(marker)
            marker_id = int(decode(hamming_code), 2)
            markers_list.append(HammingMarker(id=marker_id,
contours=approx_curve))
        except ValueError:
            continue
    return markers_list
```