Министерство образования Республики Беларусь

Учреждение образования
ГРОДНЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ ЯНКИ КУПАЛЫ
Физико-технический факультет
Кафедра информационных систем и технологий

К защите допустить:
Заведующий кафедрой ИСиТ
_____Ю.Р.Бейтюк
«____»_____20__г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему

РЕАЛИЗАЦИЯ СИСТЕМЫ HMI ДЛЯ БЕСКОНТАКТНОГО УПРАВЛЕНИЯ ОБОРУДОВАНИЕМ НА ОСНОВЕ ЖЕСТОВЫХ КОМАНД

ГРГУ ДП 1-38 02 01 08 006 ПЗ

Студент	_____	М.Р. Ступакевич
Руководитель	_____	Б.А. Ассанович
Нормоконтролер	_____	
Рецензент	_____	

Гродно 2022

РЕФЕРАТ

Дипломный проект «Реализация системы HMI для бесконтактного управления оборудованием на основе жестовых команд» содержит 104 страницы, 31 изображение, 2 таблицы, 14 источников литературы.

Цель данной работы: реализация ПО идентификации жестовых команд в видеопотоке, поступающем с камеры.

Методы исследования: для реализации проекта использовались текстовый редактор Visual Studio Code и Jupyter Notebook, язык программирования Python.

Полученные результаты: приложение сбора данных для обучения нейронной сети и приложение системы HMI для бесконтактного управления оборудованием.

Область применения: проект может быть использован в системах требующих бесконтактного управления.

RESUME

The diploma project «Realisation the HMI system for contactless control of equipment based on gesture commands» contains 104 pages, 31 images, 2 tables, 14 literature sources.

The purpose of this work is to realize the identification of gesture commands in the video stream coming from the camera.

Research methods: Visual Studio Code text editor and Jupiter Notebook, Python programming language were used to realize the project.

The results obtained are: a data acquisition application for training a neural network and an HMI system application for contactless control of equipment.

Scope: the project can be used in systems requiring contactless control.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Обычно, когда мы хотим что-то сделать, то всегда сопровождаем это какими-либо физическими действиями для достижения ожидаемого результата. Например, идём в магазин, чтобы купить продукты, протягиваем руку, чтобы взять нужный предмет или присаживаемся, чтобы завязать шнурки, поднимаем руку, чтобы обратить на себя внимание, поворачиваем голову, чтобы лучше что-то услышать. И в этом смысле жесты или движения куда более интуитивны чем статическое нажатие на кнопку вкл/выкл.

Поэтому переход к жестам кажется таким же логичным как, в своё время, переход от кнопок к сенсору.

Как же определить руку с жестом по видео? Оказывается, есть алгоритмы, которые умеют распознавать в видеопотоке человеческую ладонь. Работает это так же, как и лицом.

В настоящее время существует несколько устройств, использующих управление жестами. Например, телевизоры Philips, управление которыми осуществляется следующим образом: следует поднять руку и направить ладонь в сторону телевизора, после чего активируется управление жестами, далее сжатием и расжатием кулака открывается меню, а навигация по меню реализована использованием руки в качестве указки с сжатием кулака как сигнал нажать на кнопку.

Или новое устройство от СБЕР «SberBox Top», в котором, на 2021 год, было реализовано использование пяти жестов (Рисунок - 1.1): «Салют» для старта/ответа на звонок, «Стоп» для паузы, «Лайк» и «Дизлайк» для оценки видеоконтента и «Тихо» для включения/выключения микрофона в звонке.

Рисунок – 1 Жесты SberBox

Идентификация жестов человека по видеопотоку для управления оборудованием имеет следующий ряд преимуществ по сравнению с другими методами управления:

1. Нет необходимости в других устройствах ввода. Камера имеет не только коммуникативную функцию, но и контролирующую.
2. Не нужен физический контакт с какими-либо устройствами, достаточно лишь находиться в области видимости камеры. Тем самым физический износ является минимальным.

Помимо достоинств компьютерное распознавание жестов имеет и свои недостатки:

1. Сам по себе один лишь способ не гарантирует стопроцентного распознавания.
2. При различном освещении матрица камеры может быть пересвечена.

Чтобы распознать жест, для начала нужно распознать саму кисть руки на изображении. В настоящее время используются различные методы распознавания лиц и вот некоторые из них:

- использование нейронных сетей (к примеру, используемый в курсовом проекте Google Mediapipe Hands)
- метод гибкого сравнения на графах (elastic graph matching)
- скрытые Марковские модели (СММ, HMM)
- метод главных компонентов (principal component analysis, PCA)

Для распознавания жестов используют алгоритмы, основанные на методах машинного обучения с использованием ключевых точек, распознанных алгоритмами распознавания кистей рук на изображении или с использованием самого изображения.

Целью данного проекта является разработка программного HMI (от англ. Human-Machine interface - интерфейс Человек-Машина) для распознавания жестов в видеопотоке и выполнением команд компьютера, согласно распознанному жесту.

Для реализации системы HMI требуется решить следующие задачи:

1. Разработать инструмент для сбора данных.
2. Собрать данные для обучения нейронной сети.
3. Провести анализ и обработку собранных данных для подготовки датасета.
4. Обучить рекуррентную нейронную сеть LSTM для классификации.
5. Разработать приложение системы HMI для идентификации жестовых команд в видеопотоке.

1 КРАТКОЕ ОПИСАНИЕ ТЕХНОЛОГИЙ ПРОЕКТА

В реализации проекта использовался комплекс программного обеспечения с целью создания составного программного продукта. Для идентификации жестовых команд в видеопотоке сначала необходимо распознавание и детерминация положения ключевых точек кистей рук в пространстве по изображению. Выполнение этой задачи достигалось использованием библиотеки с открытым исходным кодом Google Mediapipe Hands. Реализация системы HMI (от англ. Human-Machine interface - интерфейс Человек-Машина) требует ПО, необходимое для сбора датасета (от англ. Dataset - набор данных) и его обработки (анализа, визуализации, фильтрации, интерполяции и т.д.), для разработки которого использовался язык программирования Python версии 3.9 и ряд библиотек таких как numpy, pandas, matplotlib, seaborn, opencv2 и др. Также требовалось ПО необходимое для обучения модели машинного обучения с учителем (использовалась библиотека для Python tensorflow/Keras и sklearn) и для разработки приложения, в котором реализована имплементация обученной модели и которое реагирует на распознанные команды с помощью этой модели. Суммируя, можно выделить следующие технологии: Python, IPython и Jupyter notebook, Google Mediapipe Hands, opencv2, numpy, pandas, matplotlib, sklearn, Tensorflow Keras.

1.1 Язык программирования Python

Python (в русской интерпретации часто звучит как питон или пайтон) — высокоуровневый интерпретируемый 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 является мультипарадигменным языком программирования, в котором используются такие подходы как структурный, объектно-ориентированный, функциональный, императивный и аспектно-ориентированный.

Язык Python обзавёлся большим количеством интерпретаторов, таких как Cpython, IPython, IronPython, Anaconda, PyPy, Jython. [1]

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные.

Python — активно развивающийся язык программирования, новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года. Вследствие этого и некоторых других причин на Python отсутствуют стандарт ANSI, ISO или другие официальные стандарты, их роль выполняет CPython.

Разработка языка Python была начата в конце 1980-х годов сотрудником голландского института CWI Гвидо ван Россумом. Для распределенной ОС Amoeba требовался расширяемый скриптовый язык, и Гвидо начал писать Python на досуге, позаимствовав некоторые наработки для языка ABC (Гвидо участвовал в разработке этого языка, ориентированного на обучение программированию). В феврале 1991 года Гвидо опубликовал исходный текст в группе новостей alt.sources. С самого начала Python проектировался как объектно-ориентированный язык.

Наличие дружелюбного, отзывчивого сообщества пользователей считается наряду с дизайнерской интуицией Гвидо одним из факторов успеха Python. Развитие языка происходит согласно четко регламентированному процессу создания, обсуждения, отбора и

реализации документов PEP (англ. Python Enhancement Proposal) — предложений по развитию Python [2].

1.2 IPython и Jupyter notebook

IPython (англ. Interactive Python) — интерактивная оболочка для языка программирования Python, которая предоставляет расширенную интроспекцию, дополнительный командный синтаксис, подсветку кода и автоматическое дополнение. Является компонентом пакетов программ SciPy и Anaconda.

IPython позволяет осуществлять неблокирующее взаимодействие с Tkinter, GTK, Qt и WX. Стандартная библиотека Python включает лишь Tkinter. IPython может интерактивно управлять параллельными кластерами, используя асинхронные статусы обратных вызовов и/или MPI. IPython может использоваться как замена стандартной командной оболочки операционной системы, особенно на платформе Windows, возможности оболочки которой ограничены. Поведение по умолчанию похоже на поведение оболочек UNIX-подобных систем, но тот факт, что работа происходит в окружении Python, позволяет добиваться большей настраиваемости и гибкости.

Начиная с версии 4.0, монолитный код был разбит на модули, и независимые от языка модули были выделены в отдельный проект Jupyter. Наиболее известной веб-оболочкой для IPython является Jupyter Notebook (ранее известный как IPython Notebook), позволяющая объединить код, текст и изображения, и распространять их для других пользователей. [3]

1.3 OpenCV

OpenCV (англ. Open Source Computer Vision Library - библиотека компьютерного зрения с открытым исходным кодом) — 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.

Второе крупное обновление OpenCV было выпущено в октябре 2009 года. OpenCV2 включает в себя серьезные изменения в интерфейсе C++, направленные на упрощение, улучшение безопасности, введение новых функций и увеличение производительности (особенно для многоядерных систем). Официальные релизы теперь выпускаются каждые шесть месяцев, а разработка ведется независимой российской командой при поддержке коммерческих корпораций. [4]

1.4 Google Mediapipe Hands

MediaPipe - это бесплатно предоставляемый фреймворк с открытым исходным кодом от компании Google, который предлагает кроссплатформенные, настраиваемые решения ML (от англ. Machine Learning - машинное обучение) для прямых и потоковых медиа.

Преимущества Google MediaPipe состоят в следующем:

Сквозное ускорение: Встроенный быстрый вывод и обработка ML, ускоренные даже на обычном оборудовании.

Сборка один раз, развертывание в любом месте: Унифицированное решение работает на Android, iOS, настольных компьютерах / облаках, в Интернете и IoT (от англ. Internet of Things - интернет вещей).

Готовые к использованию решения: Передовые решения ML, демонстрирующие всю мощь фреймворка.

Бесплатно и с открытым исходным кодом: Фреймворк и решения под управлением Apache 2.0, полностью расширяемые и настраиваемые. [5]

MediaPipe Hands - это высококачественное решение для отслеживания рук и пальцев. Он использует машинное обучение (ML) для определения 21 3D-ориентира руки всего из одного кадра. В то время как современные подходы для вывода в основном основаны на мощных средах для персонального компьютера, метод Google обеспечивает производительность в реальном времени на мобильном телефоне и даже масштабируется для распознавания ключевых точек нескольких рук.

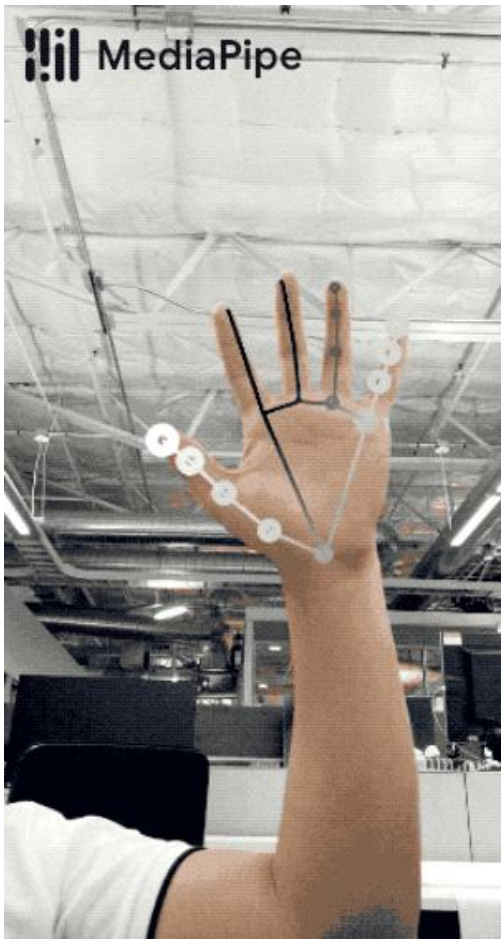


Рисунок – 1.1 Пример работы MediaPipe Hands

MediaPipe Hands использует конвейер ML, состоящий из нескольких моделей, работающих вместе: модель обнаружения ладони, которая работает с полным изображением и возвращает ориентированную ограничивающую рамку для рук. Модель ориентира руки, которая

работает с областью обрезанного изображения, определенной детектором ладони, и возвращает высококачественные 3D-ключевые точки руки.

Предоставление точно обрезанного изображения руки для модели ориентира руки значительно снижает потребность в увеличении данных (например, поворотах, переводе и масштабировании) и вместо этого позволяет сети использовать большую часть своих возможностей для обеспечения точности прогнозирования координат. Кроме того, в нашем конвейере посевы также могут быть сгенерированы на основе ориентиров руки, определенных в предыдущем кадре, и только когда модель ориентира больше не может идентифицировать присутствие руки, для перемещения руки вызывается функция обнаружения ладони.

Конвейер реализован в виде графика MediaPipe, который использует подграф отслеживания ориентиров руки из модуля ориентиров руки и визуализируется с использованием выделенного подграфа средства визуализации рук. Подграф отслеживания ориентира руки внутренне использует подграф ориентира руки из того же модуля и подграф обнаружения ладони из модуля обнаружения ладони. [6]

1.5 Numpy

NumPy (сокращенно от Numerical Python)— библиотека с открытым исходным кодом для языка программирования Python. Возможности:

- поддержка многомерных массивов (включая матрицы);
- поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Математические алгоритмы, реализованные на интерпретируемых языках (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на компилируемых языках (например, Фортран, Си, Java). Библиотека NumPy предоставляет реализации вычислительных алгоритмов (в виде функций и операторов),

оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB. [7]

1.6 Pandas

Pandas — программная библиотека на языке Python для обработки и анализа данных. Работа pandas с данными строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. Предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временны́ми рядами. Название библиотеки происходит от эконометрического термина «панельные данные», используемого для описания многомерных структурированных наборов информации. pandas распространяется под новой лицензией BSD.

Основная область применения — обеспечение работы в рамках среды Python не только для сбора и очистки данных, но для задач анализа и моделирования данных, без переключения на более специфичные для статобработки языки (такие, как R и Octave).

Также активно ведётся работа по реализации «родных» категориальных типов данных.

Пакет прежде всего предназначен для очистки и первичной оценки данных по общим показателям, например среднему значению, квантилям и так далее; статистическим пакетом он в полном смысле не является, однако наборы данных типов DataFrame и Series применяются в качестве входных в большинстве модулей анализа данных и машинного обучения (SciPy, Scikit-Learn[en] и других). [8]

1.7 MatPltLib

Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной (2D) графикой (3D графика также поддерживается). Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях.

Matplotlib написан и поддерживался в основном Джоном Хантером (англ. John Hunter) и распространяется на условиях BSD-подобной лицензии. Генерируемые в различных форматах изображения могут быть использованы в интерактивной графике, в научных публикациях, графическом интерфейсе пользователя, веб-приложениях, где требуется построение диаграмм (англ. plotting). В документации автор признаётся, что Matplotlib начинался с подражания графическим командам MATLAB, но является независимым от него проектом.

Версия 2.1.1 — последняя стабильная — требует Python версии 2.7 или от 3.4 и выше и версию NumPy от 1.7.1 и выше.

Библиотека Matplotlib построена на принципах ООП, но имеет процедурный интерфейс pylab, который предоставляет аналоги команд MATLAB.

Matplotlib является гибким, легко конфигурируемым пакетом, который вместе с NumPy, SciPy и IPython предоставляет возможности, подобные MATLAB. В настоящее время пакет работает с несколькими графическими библиотеками, включая wxWindows и PyGTK.

Пакет поддерживает многие виды графиков и диаграмм:

- Графики (line plot)
- Диаграммы разброса (scatter plot)
- Столбчатые диаграммы (bar chart) и гистограммы (histogram)
- Круговые диаграммы (pie chart)
- Ствол-лист диаграммы (stem plot)

- Контурные графики (contour plot)
- Поля градиентов (quiver)
- Спектральные диаграммы (spectrogram)

Пользователь может указать оси координат, решетку, добавить надписи и пояснения, использовать логарифмическую шкалу или полярные координаты.

Несложные трёхмерные графики можно строить с помощью набора инструментов (toolkit) mplot3d. Есть и другие наборы инструментов: для картографии, для работы с Excel, утилиты для GTK и другие.

С помощью Matplotlib можно делать и анимированные изображения. [9]

1.8 Scikit-learn

Scikit-learn (ранее scikits.learn, также известный как sklearn) - это бесплатная библиотека машинного обучения для языка программирования Python. Он включает в себя различные алгоритмы классификации, регрессии и кластеризации, включая машины опорных векторов, случайные леса, повышение градиента, k-means и DBSCAN, и предназначен для взаимодействия с числовыми и научными библиотеками Python NumPy и SciPy. Scikit-learn - это финансируемый NumFOCUS проект.

Проект scikit-learn начинался как scikits.learn - проект Google Summer of Code французского специалиста по обработке данных Дэвида Курнапо. Его название происходит от представления о том, что это "SciKit" (SciPy Toolkit), отдельно разработанное и распространяемое стороннее расширение для SciPy. Оригинальная кодовая база позже была переписана другими разработчиками. В 2010 году Фабиан Педрегоса, Гаэль Вароко, Александр Грамфор и Винсент Мишель, все из Французского института исследований в области компьютерных наук и автоматизации в Рокенкуре, Франция, взяли на себя руководство проектом и сделали первый

публичный релиз 1 февраля 2010 года. Из различных scikits scikit-learn, а также scikit-image были описаны как "хорошо поддерживаемые и популярные" в ноябре 2012 года. Scikit-learn - одна из самых популярных библиотек машинного обучения на GitHub. [10]

1.9 Tensorflow Keras

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для R, C Sharp, C++, Haskell, Java, Go и Swift.

Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0. [11]

Keras — открытая библиотека, написанная на языке Python и обеспечивающая взаимодействие с искусственными нейронными сетями. Она представляет собой надстройку над фреймворком TensorFlow. До версии 2.3 поддерживались разные версии нейросетевых библиотек, такие как TensorFlow, Microsoft Cognitive Toolkit, Deeplearning4j, и Theano. Нацелена на оперативную работу с сетями глубинного обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Она была создана как часть исследовательских усилий проекта ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System), а ее основным автором и поддерживающим является Франсуа Шолле (фр. François Chollet), инженер Google.

Планировалось что Google будет поддерживать Keras в основной библиотеке TensorFlow, однако Шолле выделил Keras в отдельную надстройку, так как согласно концепции Keras является скорее интерфейсом, чем сквозной системой машинного обучения. Keras предоставляет высокоуровневый, более интуитивный набор абстракций, который делает простым формирование нейронных сетей, независимо от используемой в качестве вычислительного бэкенда библиотеки научных вычислений. Microsoft работает над добавлением к Keras и низкоуровневых библиотек CNTK. [12]

1.10 Рекуррентная нейронная сеть LSTM

Долгая краткосрочная память (Long short-term memory; LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям. Они были представлены Зеппом Хохрайтер и Юргеном Шмидхубером (Jürgen Schmidhuber) в 1997 году, а затем усовершенствованы и популярно изложены в работах многих других исследователей. Они прекрасно решают целый ряд разнообразных задач и в настоящее время широко используются [13].

LSTM разработаны специально, чтобы избежать проблемы долговременной зависимости. Запоминание информации на долгие периоды времени – это их обычное поведение, а не что-то, чему они с трудом пытаются обучиться.

На рисунке – 1.5 изображена структурная схема LSTM слоя.

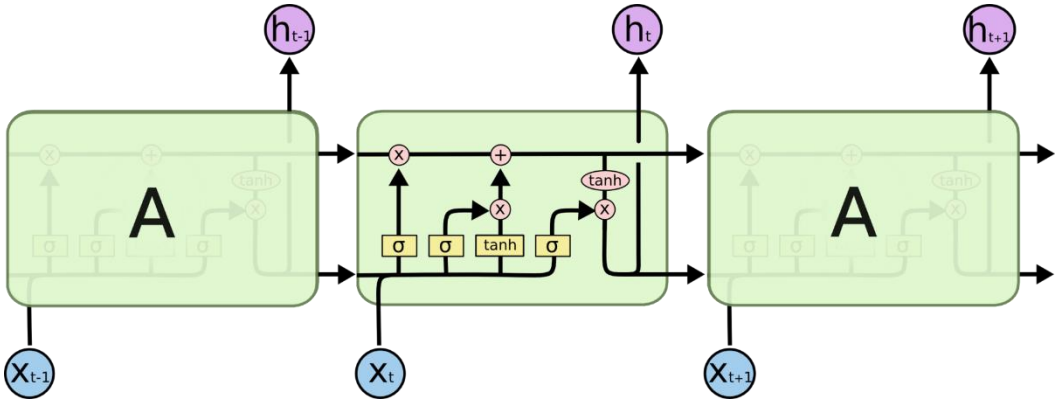


Рисунок – 1.2 Структурная схема LSTM слоя

Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы (Рисунок - 1.6).

Состояние ячейки напоминает конвейерную ленту. Она проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях. Информация может легко течь по ней, не подвергаясь изменениям.

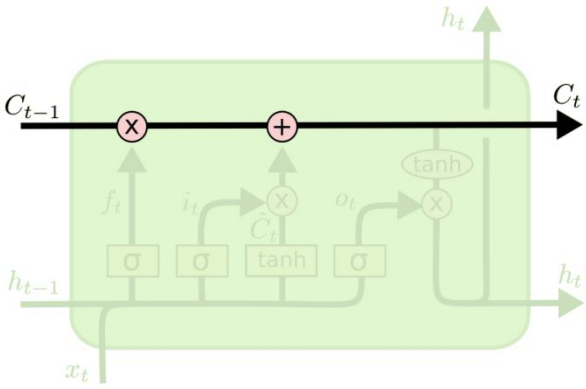


Рисунок - 1.3 Состояние ячейки

Тем не менее, LSTM может удалять информацию из состояния ячейки; этот процесс регулируется структурами, называемыми фильтрами (gates).

Фильтры позволяют пропускать информацию на основании некоторых условий. Они состоят из слоя сигмоидальной нейронной сети и

операции поточечного умножения (Рисукок – 1.7 Сигмоидальный слой).

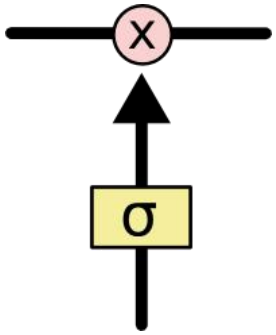
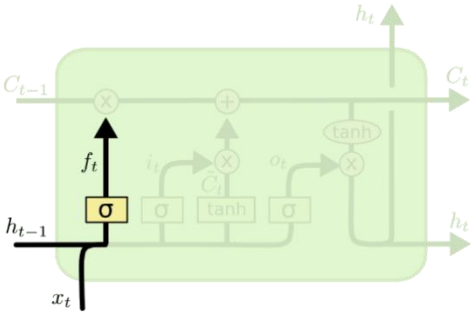


Рисунок - 1.4 Сигмоидальный слой

Сигмоидальный слой возвращает числа от нуля до единицы, которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети. Ноль в данном случае означает “не пропускать ничего”, единица – “пропустить все”.

В LSTM три таких фильтра, позволяющих защищать и контролировать состояние ячейки.

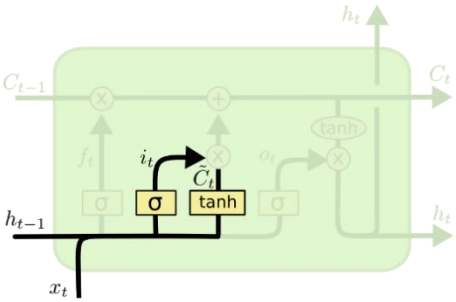
Первый шаг в LSTM – определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает сигмоидальный слой, называемый “слоем фильтра забывания” (forget gate layer) (Рисунок – 1.8). Он смотрит на $h_t−1$ и x_t и возвращает число от 0 до 1 для каждого числа из состояния ячейки $C_{t–1}$. 1 означает “полностью сохранить”, а 0 – “полностью выбросить”.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Рисунок – 1.5 Слой фильтра забывания

Следующий шаг – решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой под названием “слой входного фильтра” (input layer gate) определяет, какие значения следует обновить (Рисунок – 1.9). Затем tanh-слой строит вектор новых значений-кандидатов C_t, которые можно добавить в состояние ячейки.

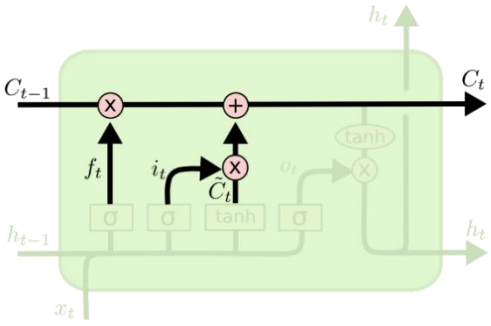


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Рисунок – 1.6 Слой входного фильтра

Настало время заменить старое состояние ячейки C_{t-1} на новое состояние C_t.

Необходимо умножить старое состояние на f_t, забывая то, что решено забыть. Затем прибавляется $i_t * C_t$. Это новые значения-кандидаты, умноженные на t – на сколько следует обновить каждое из значений состояния (Рисунок - 1.10).



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Рисунок – 1.7 Слой обновления состояния

Наконец, нужно решить, какую информацию необходимо получать на выходе. Выходные данные будут основаны на состоянии ячейки, к ним будут применены некоторые фильтры. Сначала применяется сигмоидальный слой, который решает, какую информацию из состояния ячейки будет нужно выводить. Затем значения состояния ячейки проходят через tanh-слой (слой с функцией активации «Гиперболический тангенс»), чтобы получить на выходе значения из диапазона от -1 до 1, и перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию.

Именно поэтому в решении поставленной задачи следует использовать сеть LSTM, выделяя, запоминая и используя только необходимые данные из последовательности при обучении и последующем использовании модели для предсказания результата. [13]

2 ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

2.1 Общая структура

Конечное приложение, разработанное в этом дипломном проекте, должно работать согласно приведённой ниже структурной схеме:

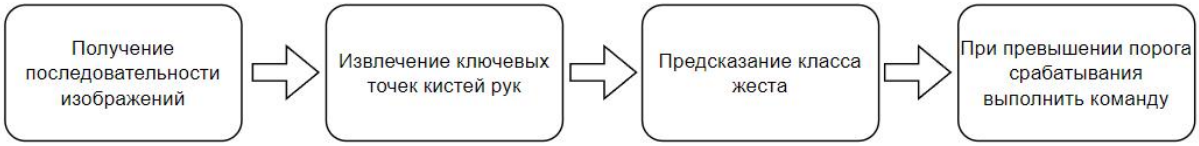


Рисунок - 2.1 Структурная схема системы

Основным действующим элементом HMI системы является модель рекуррентной нейронной сети LSTM, выполняющая функцию машинного обучения с учителем, а именно - классификацию. Для использования её сначала нужно обучить на реальных размеченных данных. Таким образом, формулируются следующие промежуточные задачи: реализация инструмента сбора и мгновенной разметки данных, формулирование жестовых команд, которые система должна будет распознать, сбор данных, обработка собранных данных.

Для решения первой задачи разработано приложение, работающее по представленной ниже структурной схеме:

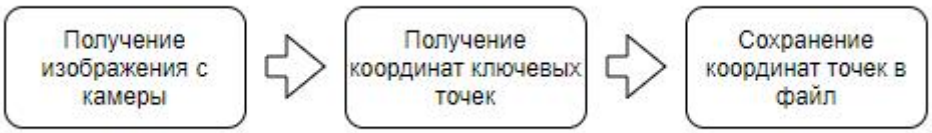


Рисунок - 2.2 Структурная схема работы приложения сбора данных

Далее, согласно сформулированным для классификации жестовым командам, следует собрать данные и произвести их обработку

(визуализация, анализ, интерполяция и другие преобразования), что будет выполнено согласно следующей схеме:

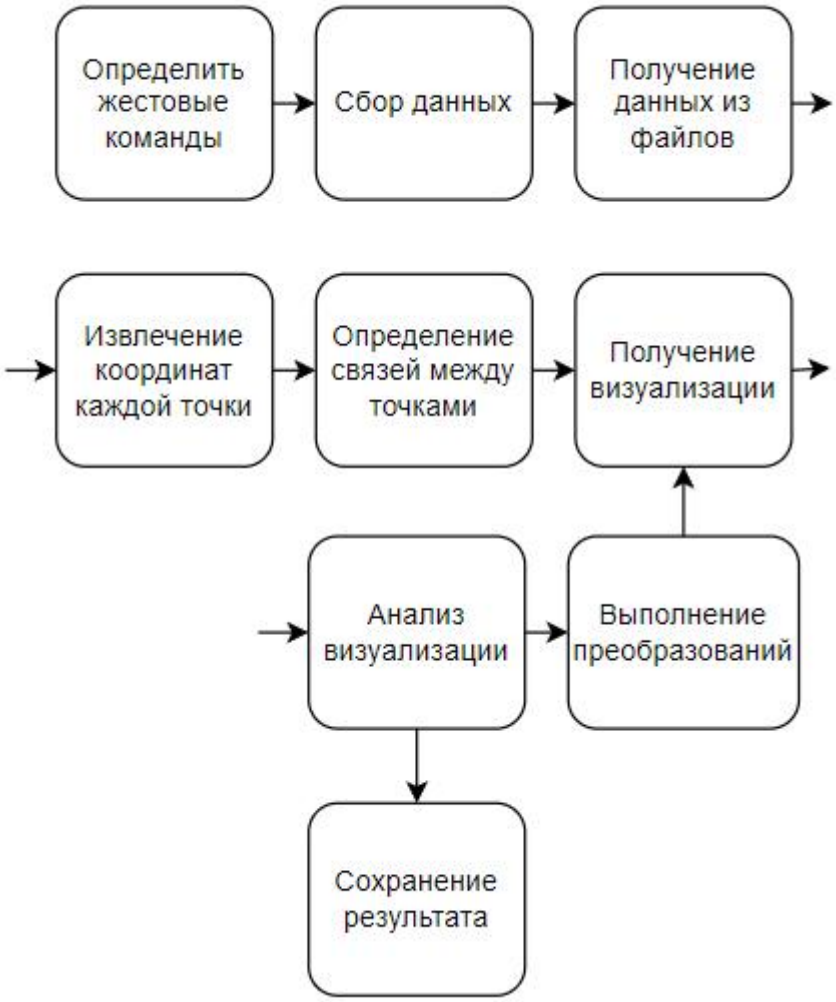


Рисунок - 2.3 Структурная схема этапов сбора и обработки данных

Таким образом, мы получаем всё необходимое для обучения модели рекуррентной нейронной сети LSTM. Последующей задачей является проведение экспериментов по обучению нейросети (перебор гиперпараметров модели для достижения более успешного результата валидации модели). В конечном итоге необходима реализация приложения, в которое будет внедрена модель, и реагирующее на жестовые команды, выполняя заданные функции.

2.2 Приложение сбора данных

Более подробный алгоритм работы приложения сбора данных представлен ниже:

Рисунок - 2.4 Блок-схема основной задачи приложения сбора данных

Основная задача приложения - последовательно получать изображение с камеры, детектировать на каждом изображении ключевые точки кистей рук и сохранять полученные координаты точек в файл для каждого последующего кадра.

Перед тем как начать работу приложения следует создать директории на диске для хранения файлов. Принято решение сохранять файлы следующим образом. Создаётся директория с указанным именем. Внутри созданной директории создаются директории для каждого указанного ранее жеста. Внутри директории жеста создаются директории записей в указанном количестве, названные по номерам записей. Эти директории в последствии будут хранить .npy файлы, содержащие данные о пространственном расположении ключевых точек кистей рук в каждом кадре видеопотока, указанном на этапе конфигурации.

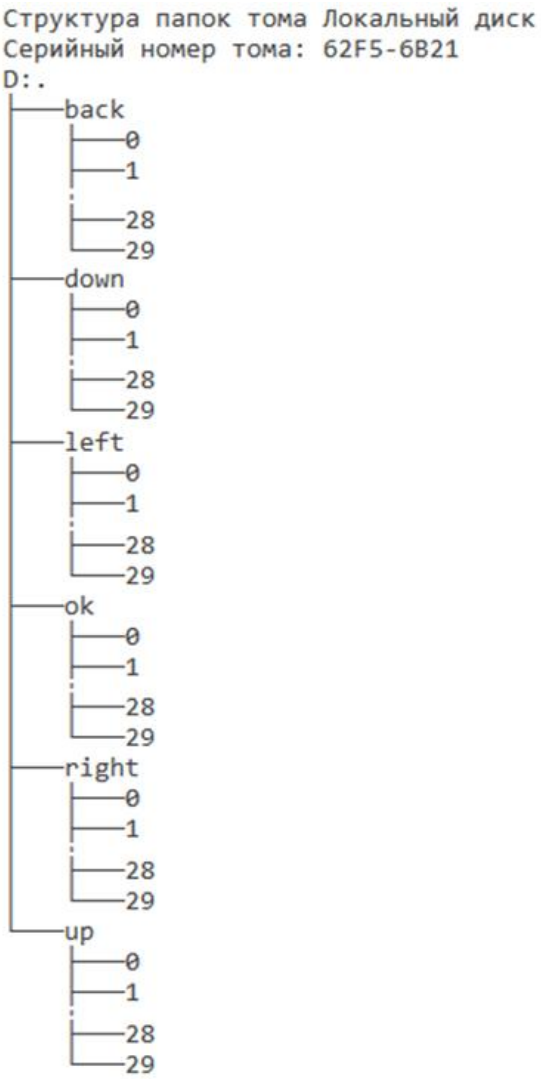


Рисунок - 2.5 Пример структуры папок

Следующий код выполняет данные операции:

```
def make_dirs(data_path, actions, no_sequences):
for action in actions:
  for sequence in range(no_sequences):
    try:
      os.makedirs(os.path.join(data_path, action, str(sequence)))
    except:
      pass
```

После создания директорий для хранения данных можно приступать к сбору данных.

Извлечение ключевых точек из изображения выполняет следующая функция process объекта класса Hands фреймворка MediaPipe,

принимающая на входе кадр изображения и возвращающая объект результата, содержащий вместе с координатами ключевых точек различную дополнительную информацию (т.н. мировые координаты, вероятностно оценивающие реальные размеры кистей рук, вероятность нахождения ключевой точки в указанном положении в пространстве и т.д.). Для извлечения необходимых координат ключевых точек двух рук из вышеописанного объекта реализована следующая функция:

```
def extract_keypoints(results):
try:
  for idx, handLms in enumerate(results.multi_hand_landmarks):
    lh = np.array([[res.x, res.y, res.z] for res in
            results.multi_hand_landmarks[idx].landmark]).flatten() \
      if results.multi_handedness[idx].classification[0].label == 'Left' else np.zeros(21 * 3)
    rh = np.array([[res.x, res.y, res.z] for res in
            results.multi_hand_landmarks[idx].landmark]).flatten() \
      if results.multi_handedness[idx].classification[0].label == 'Right' else np.zeros(21 * 3)
  return np.concatenate([lh, rh])
except:
  return np.concatenate([np.zeros(21 * 3), np.zeros(21 * 3)])
```

В случае отсутствия ключевых точек кистей руки на изображении массив координат заменяется массивом, наполненным нулями.

Для извлечения ключевых точек кистей рук реализована функция, работающая следующим образом. На вход функции принимаются: объекты классов Hands, Drawing, DrawingStyles из MediaPipe, набор жестов, количество записей и их размер, путь сохранения файлов, минимальная граница детектирования (по умолчанию 0.5 или 50%), флаг установки статического режима (по умолчанию False) и номер камеры в операционной системе (по умолчанию 0).

В начале сбора из библиотеки cv2 получается объект, указанной на этапе конфигурации камеры. Далее начинается итерация по указанным жестам, для каждого жеста итерируется каждая запись, для каждой записи итерируется каждый кадр указанное количество раз.

В каждой итерации кадра проверяется успешность получения изображения с камеры. В случае неудачи, попытка повторяется. Далее происходит вычисление времени с получения предыдущего кадра и вычисляется количество кадров в секунду (FPS).

В последствии изображение передается в функцию process объекта Hands класса MediaPipe и на выходе получается объект, содержащий всю информацию о пройденном детектировании.

На основе полученных данных ключевые точки кистей рук и связи между ними отображаются на изображении. Также на изображении выводится информация о текущем жесте, номере записи.

В конечном итоге полученный объект детектирования передаётся в реализованную ранее функцию extract_keypoints и данные сохраняются в .npy файл от библиотеки Numpy.

По окончании итераций окна закрываются, и работа функции завершается.

Данная функция имеет следующий код:

```
def get_keypoints(mp_hands, mp_drawing, mp_drawing_styles,
          actions, no_sequences, sequence_length, data_path,
          threshold=0.5, static_mode=False, camera=0):
cap = cv2.VideoCapture(camera)
p_time = 0
with mp_hands.Hands(
      min_detection_confidence=threshold,
      min_tracking_confidence=threshold,
      static_image_mode=static_mode) as hands:
   # Loop through actions
   for action in actions:
      # Loop through sequences aka videos
      for sequence in range(no_sequences):
         # Loop through video length aka sequence length
         for frame_num in range(sequence_length):

            # Read feed
            success, image = cap.read()
            if not success:
               print("Ignoring empty camera frame.")
               # If loading a video, use 'break' instead of 'continue'.
               continue

            # FPS counter
```

```
            c_time = time.time()
            fps = 1 / (c_time - p_time)
            p_time = c_time
            cv2.putText(image, f'FPS: {int(fps)}', (40, 70), cv2.FONT_HERSHEY_DUPLEX, 3, (255, 245,
215), 3)

            # Make detections
            image.flags.writeable = False
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            results = hands.process(image)

            # Draw landmarks
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            if results.multi_hand_landmarks:
               for hand_landmarks in results.multi_hand_landmarks:
                  mp_drawing.draw_landmarks(
                     image,
                     hand_landmarks,
                     mp_hands.HAND_CONNECTIONS,
                     mp_drawing_styles.get_default_hand_landmarks_style(),
                     mp_drawing_styles.get_default_hand_connections_style())

            # NEW Apply wait logic
            if frame_num == 0:
               cv2.putText(image, 'STARTING COLLECTION', (120, 200),
                           cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 245, 215), 4, cv2.LINE_AA)
               cv2.putText(image,  'Collecting  frames  for  {}  Video  Number  {}'.format(action,
sequence),
                           (15, 12),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 245, 215), 1, cv2.LINE_AA)
               # Show to screen
               cv2.imshow('OpenCV Feed', image)
               cv2.waitKey(1500)
            else:
               cv2.putText(image,  'Collecting  frames  for  {}  Video  Number  {}'.format(action,
sequence),
                           (15, 12),
                           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 245, 215), 1, cv2.LINE_AA)
               # Show to screen
               cv2.imshow('OpenCV Feed', image)

            # NEW Export keypoints
            keypoints = extract_keypoints(results)
            npy_path = os.path.join(data_path, action, str(sequence), str(frame_num))
            np.save(npy_path, keypoints)

            # Break gracefully
            if cv2.waitKey(5) & 0xFF == 27:
               cap.release()
               cv2.destroyAllWindows()
               return
```

Также, для «холостой» работы алгоритма, разработана функция, работающая аналогично, за исключением сохранения результата. Эту функцию можно назвать демонстрирующей.

Реализованный алгоритм сбора данных является самодостаточным и вышеописанные функции можно смело назвать «чистыми функциями», которые получают на вход изображение и создают файлы результатов на выходе. Таким образом, разработанные функции можно использовать в приложении, доступному обычному пользователю без опасений влияния приложения на операционную систему.

Таким образом, для «оборачивания» функций в пользовательское приложение был разработан с помощью инструмента QtDesigner следующий интерфейс (Рисунок - 2.3):

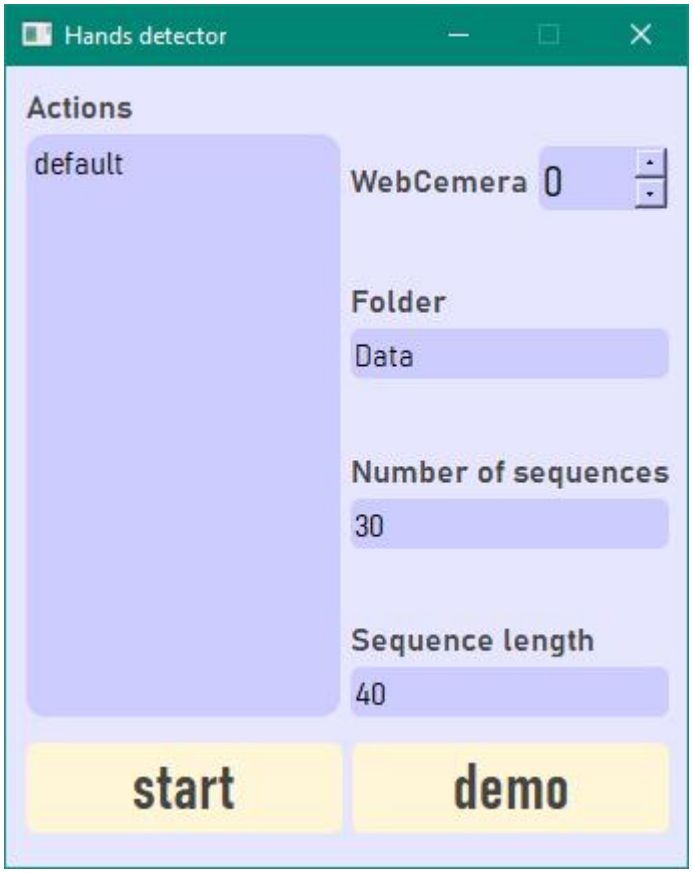


Рисунок - 2.6 Интерфейс приложения

С помощью библиотеки PyQt5 полученный с помощью QtDesigner интерфейс связан с функциями сбора данных. Разработанное приложение

работает следующим образом: Конфигурируются параметры сбора данных. После нажатия кнопки “Start” разворачивается окно, начинающее сбор данных, которое выглядит следующим образом:

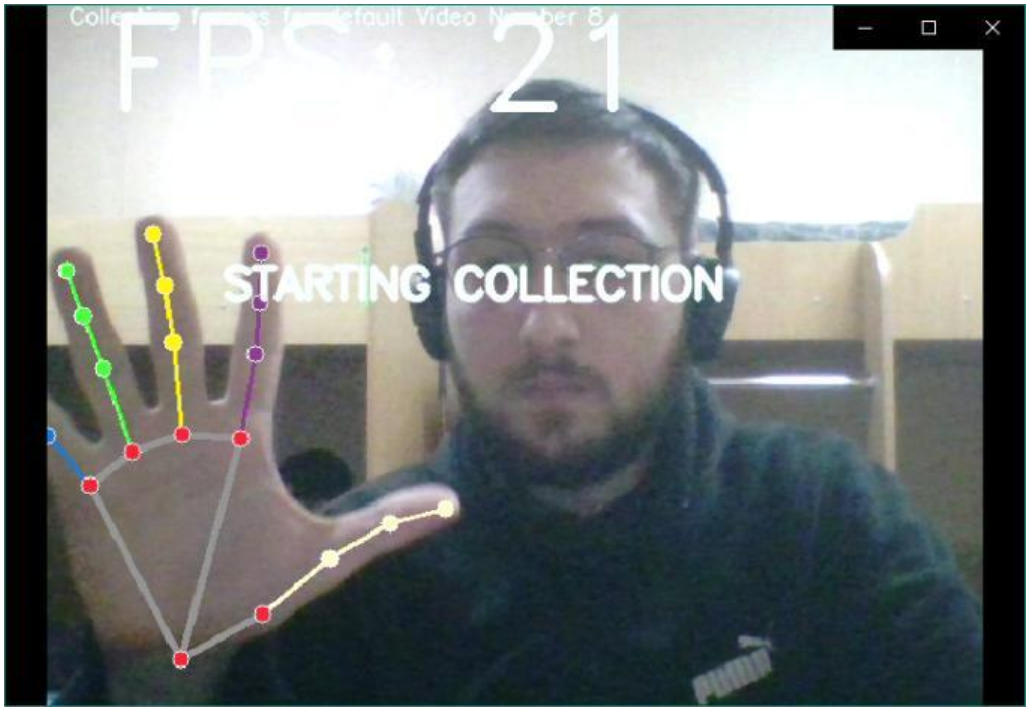


Рисунок - 2.7 Окно сбора данных приложения

Приложение также может работать в демонстрационной режиме, не собирающем данные. Данный режим доступен по нажатию кнопки “Demo”.

2.3 Формулирование команд, сбор и обработка данных

Для выполнения задачи формулирования набора жестовых команд управления, необходимо определить интуитивно понятные человеку движения и достаточно простые для распознавания алгоритмом, во избежание нечётких срабатываний.

Так же жесты должно в полной мере удовлетворять базовые потребности в управлении. За пример был взят обычный пульт с стандартными кнопками: Вверх, Вниз, Влево, Вправо, Назад, ОК. Таким образом, были определены следующие движения рукой с ладонью направленной в сторону камеры (Рисунок - 1.2):

Рисунок – 2.8 Жесты управления

Во избежание непреднамеренных активаций команд было принято использовать так называемый «синхрожест», который представляет собой наполовину сжатый кулак (Рисунок - 1.3) и сопровождает управляющий жест от начала до конца.

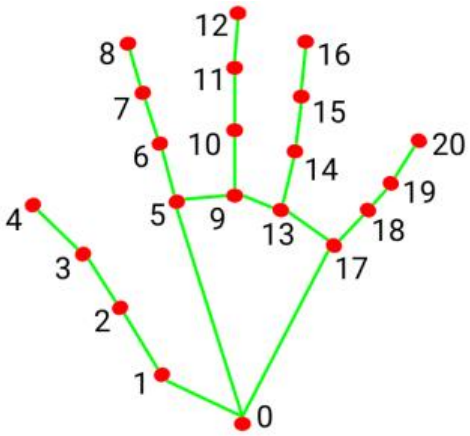
Рисунок – 2.9 Пример синхрожеста

Собранные данные представляют собой набор из 6 жестов, по 30 записей, по 40 кадров, в сумме 7200 .npy файлов (Библиотеки Numpy для Python), который состоит из 126 чисел с плавающей точкой, описывающих 3D-координаты двух рук по 21 ключевой точке и выглядящие следующим образом:

```
[array([ 0.42938599,  0.56767941,  0.       ,  0.46548966,  0.5553869 ,
       -0.00987106,  0.49247044,  0.53347236, -0.01593102,  0.51416409,

       …
        0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
        0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
        0.       ]),
…
array([ 0.42413726,  0.5693543 ,  0.       ,  0.44785389,  0.53808248,
       -0.00742947,  0.46761057,  0.49912149, -0.01242292,  0.48233014,
       …
              0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
              0.       ,  0.       ,  0.       ,  0.       ,  0.       ,
              0.       ])]
```

Обрабатывать и визуализировать было принято только одну правую руку с целью экономии вычислительных ресурсов, т.к. наличие двух рук увеличивает количество фич (от англ. feature - признак) вдвое и обучение нейронной сети на данных большей размерности увеличивается на порядок, что уж говорить о большем количестве рук. Для демонстрации и решения поставленной задачи достаточно лишь одной руки (При наличии более высоких мощностей возможно легко адаптировать представленный алгоритм для обработки и визуализации двух и более рук).

Коллекция обнаруженных/отслеженных рук, где каждая рука представлена в виде списка из 21 ориентира руки, и каждый ориентир состоит из **x, y, z**. Где **x** и **y** нормализуются от 0.0 до 1.0 по ширине и высоте изображения соответственно. А **z** представляет глубину ориентира, причем глубина на запястье является началом координат, и чем меньше значение, тем ближе ориентир к камере. Величина **z** вычисляется примерно того же масштаба, **x** что и **y**. Ключевые точки в файлах расположены следующим образом (Рисунок - 1.4):



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Рисунок - 2.10 Карта ключевых точек кисти руки

Следующей задачей является получение данных из файлов и проведение обработки.

Рисунок - 2.11 Блок-схема получения датасета

Разработку алгоритма необходимо начать с получения данных. Для этого необходимо указать относительный путь к папке где расположены все файлы.

Так как алгоритм заранее не знает какое количество жестов, сколько записей и их размер, необходимо явно это указать.Так же необходимо указать связи между ключевыми точками кисти руки.

Далее, с учётом известных параметров, данные извлекаются из файлов и помещаются в объект (переменную).

Данные операции выполняет следующий код:

```
import os
DATA_PATH = os.path.join('MP_Data\\Hands_Data')
actions = np.array([ 'up', 'down', 'left', 'right', 'ok', 'back'])
no_sequences = 30
sequence_length = 40
label_map = {label:num for num, label in enumerate(actions)}
parts = [
   [0, 1, 0, 5, 0, 17, 'green'],
   [5,9,9,13,13,17,'red'],
   [1,2,2,3,3,4,'blue'],
```

```
    [5,6,6,7,7,8,'yellow'],
    [9,10,10,11,11,12,'orange'],
    [13,14,14,15,15,16,'pink'],
    [17,18,18,19,19,20,'purple']
]
sequences, labels = [], []
for action in actions:
    for sequence in range(no_sequences):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])
```

Перед тем как визуализировать данные необходимо их обработать. Произведя визуализацию можно увидеть отсутствие данных в некоторых кадрах, т.н. пробелы. Таким образом, для заполнения пробелов необходимо использовать интерполяцию, в частности линейную. Алгоритм интерполяции данных следующий:

Рисунок - 2.12 Блок-схема интерполяции данных

Данные, извлеченные из файлов, передаются в функцию. Далее происходит разделение данных по координатам на наборы X, Y и Z соответственно. В последствии каждый набор линейно интерполируется по всей последовательности. Эти операции выполняет следующая функция:

```
def interp_coords(x):
    coords = []
    for num,i in enumerate(x):
        if np.count_nonzero(i) != 0:
            coords.append([i,num])
    result = []
    for i in range(63):
        result.append(np.interp(range(40),[e[1] for e in coords],[e[0][i] for e in coords]))
    return np.array(result).transpose()
```

Рисунок - 2.13 Блок-схема получения визуализации

Затем можно приступать к визуализации. Для визуализации реализована функция, принимающая следующие параметры: координаты ключевых точек кисти руки, связи ключевых точек, флаги сохранения, фиксированного масштаба координатных осей и динамического изменения точки зрения. В функции для каждого кадра извлекаются ключевые точки и помещаются в трёхмерный график. Затем на график помещаются связи между ключевыми точками. После этого выбирается

масштаб осей автоматический или фиксированный, демонстрирующий всю область видимости ключевых точек, в зависимости от установленного флага fixed_axes. И выбирается динамическая или статическая точка зрения на график флагом dynamic_view. В зависимости от значения флага save результат сохраняется. В конечном итоге с помощью библиотеки MatPlotLib получаем визуализацию каждого отдельного кадра.

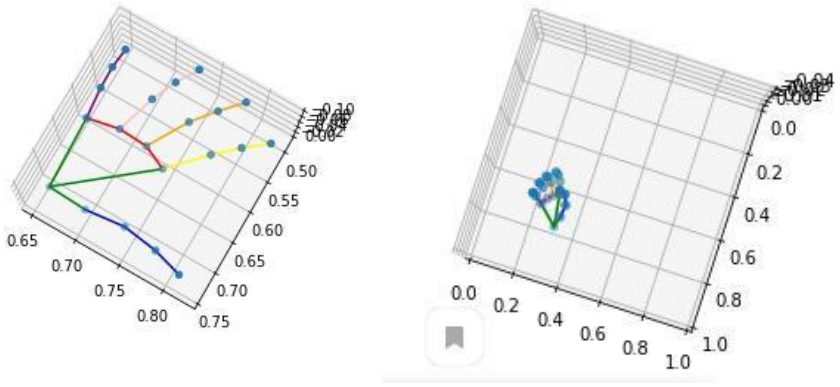


Рисунок - 2.14 Пример визуализации кадра

Код данной функции и используемых внутри неё:

```
    def draw_parts(ax, dx, dy, dz, parts):
        ax.plot3D([dx[parts[0]],dx[parts[1]]],                    [dy[parts[0]],dy[parts[1]]],
[dz[parts[0]],dz[parts[1]]], parts[6])
        ax.plot3D([dx[parts[2]],dx[parts[3]]],                    [dy[parts[2]],dy[parts[3]]],
[dz[parts[2]],dz[parts[3]]], parts[6])
        ax.plot3D([dx[parts[4]],dx[parts[5]]],                    [dy[parts[4]],dy[parts[5]]],
[dz[parts[4]],dz[parts[5]]], parts[6])

    def get_coords(X):
        dx = X[:63:3]
        dy = X[1:63:3]
        dz = X[2:63:3]
        return dx,dy,dz

    def draw_hand(x, parts, save=False, fixed_axes=False, dinamic_view=False):
        for num,i in enumerate(x):
            dx,dy,dz = get_coords(i)

            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d', label=num)
            ax.scatter(dx, dy, dz)

            for i in parts:
                draw_parts(ax, dx, dy, dz, i)
```

```
ax.view_init(260, -120)
figure = plt.gcf()

if fixed_axes:
   plt.xlim([0,1])
   plt.ylim([0,1])

if dinamic_view:
   ax.view_init(260+2*num, -120+4*num)

plt.show()
if save:
   figure.savefig(os.path.join("images", "{}.jpg".format(num)))
plt.close(fig)
```

Описанная функция ничего не возвращает, только демонстрирует matplotlib графики для каждого кадра записи движения. Внутри функции используются функции get_coords для получения координат конкретного кадра и функция draw_parts для визуализации связей между ключевыми точками.

2.4 Обучение LSTM сети

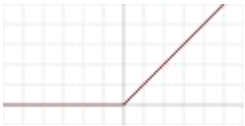
2.4.1 Конфигурация

Исходя из объёма и формата имеющихся данных (6 жестов, по 30 записей, по 40 кадров каждая) была сконфигурирована следующая модель рекуррентной нейронной сети LSTM:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 40, 64)            48896

 lstm_4 (LSTM)               (None, 40, 128)           98816

 lstm_5 (LSTM)               (None, 64)                49408

 dense_3 (Dense)             (None, 64)                4160

 dense_4 (Dense)             (None, 32)                2080

 dense_5 (Dense)             (None, 6)                 198

=================================================================
Total params: 203,558
Trainable params: 203,558
Non-trainable params: 0
```

Рисунок - 2.4 Блок-схема получения визуализации

На протяжении всей структуры модели основной функцией активации является так называется функция ReLU (от англ. Rectified linear unit - Линейный выпрямитель или полулинейный элемент):



$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geqslant 0 \end{cases}$$

Рисунок - 2.15 Функция активации ReLU

Эта функция является универсальной функцией обучения различного рода и форм нейронных сетей благодаря своей линейности. С помощью только этой функции активации можно добиться хорошего результата обучения независимо от количества слоёв нейронной сети.

Другой функцией активации является функция активации Softmax, выполняющая активацию в суммирующем финальном слое, выводя на выходе вероятности отношения классифицируемого объекта к i-тому классу. Softmax — это обобщение логистической функции для многомерного случая. Функция преобразует вектор z размерности K в вектор σ той же размерности, где каждая координата σ_i полученного вектора представлена вещественным числом в интервале [0,1] и сумма координат равна 1. Координаты σ_i вычисляются следующим образом:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

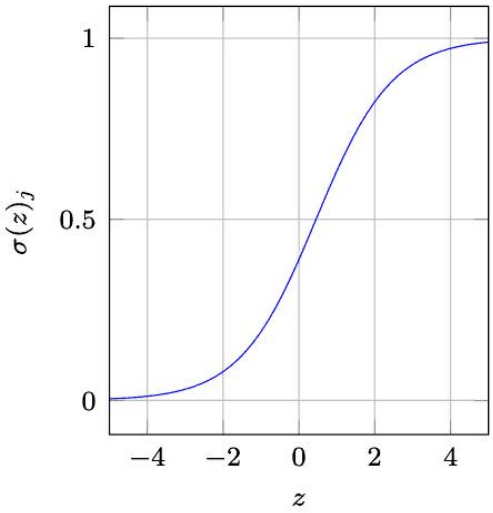


Рисунок - 2.16 Функция активации Softmax

В качестве входного слоя модели выступают элементы LSTM в количестве 40 на 64. На выходе модели находится суммирующий слой с функцией softmax, и имеющий 6 выходов, выводящих вероятности отношения классифицируемого объекта к i-тому классу от 0 до 1.

Весь код, выполняющий конфигурацию модели рекуррентной нейронной сети LSTM:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard

model = Sequential()
model.add(LSTM(64,         return_sequences=True,         activation='relu',
input_shape=(40,126)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
```

```
        model.add(LSTM(64, return_sequences=False, activation='relu'))
        model.add(Dense(64, activation='relu'))
        model.add(Dense(32, activation='relu'))
        model.add(Dense(actions.shape[0], activation='softmax'))

        model.compile(optimizer='Adam',            loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
```

2.4.2 Проведение экспериментов по обучению и валидации модели

Исходя из целей проекта метрикой оценивания модели будет являться Accuracy (от англ. Точность). Accuracy - это то, насколько близок или далек данный набор измерений (наблюдений или показаний) от их истинного значения.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Рисунок - 2.17 Формула вычисления Accuracy

Где:

TP = True positive; (положительный исход)

FP = False positive; (Ложноположительный исход)

TN = True negative; (Отрицательный исход)

FN = False negative (Ложноотрицательный исход)

Более наглядно эту формулу можно понять следующим образом. Учитывая, что возможны два варианта ответа системы и два правильных ответа, то всего возможно 4 исхода:

Реальные показания	Система отвечает «да»	Система отвечает «нет»
Да	Положительный исход	Ложноотрицательный исход
Нет	Ложноположительный исход	Отрицательный исход

Пример обучения нейронной сети:

```
Epoch 1/100
```

```
  6/6  [==============================]  -  8s  109ms/step  -  loss:  1.7829  -
categorical_accuracy: 0.1728
  Epoch 2/100
  6/6  [==============================]  -  1s  122ms/step  -  loss:  1.7460  -
categorical_accuracy: 0.2778
  Epoch 3/100
  6/6  [==============================]  -  1s  116ms/step  -  loss:  1.6570  -
categorical_accuracy: 0.2407
  Epoch 4/100
  6/6  [==============================]  -  1s  111ms/step  -  loss:  1.5603  -
categorical_accuracy: 0.2963
  Epoch 5/100
  6/6  [==============================]  -  1s  109ms/step  -  loss:  1.5202  -
categorical_accuracy: 0.3148
  Epoch 6/100
  6/6  [==============================]  -  1s  107ms/step  -  loss:  1.9403  -
categorical_accuracy: 0.4136
  Epoch 7/100
  6/6  [==============================]  -  1s  109ms/step  -  loss:  1.6135  -
categorical_accuracy: 0.3889
  Epoch 8/100
  6/6  [==============================]  -  1s  109ms/step  -  loss:  1.6216  -
categorical_accuracy: 0.3395
  Epoch 9/100
  6/6  [==============================]  -  1s  94ms/step  -  loss:  1.5447  -
categorical_accuracy: 0.3210
  Epoch 10/100
  6/6  [==============================]  -  1s  132ms/step  -  loss:  1.6248  -
categorical_accuracy: 0.3210
  Epoch 11/100
  6/6  [==============================]  -  1s  118ms/step  -  loss:  1.3468  -
categorical_accuracy: 0.3889
```

Как видно из примера выше, библиотека Keras сама вычисляет и выводит это значение на каждом этапе обучения модели рекуррентной нейронной сети. Таким образом, можно отслеживать результативность обучения на каждой эпохе и тем самым вовремя обнаруживать наличие переобучения модели, останавливать обучение и повторять эксперимент с другим количеством эпох. До тех пор пока мы не достигнем максимально возможного Accuracy, оценённого на тестовом наборе данных. Следую этому принципу на всём ходу проведения экспериментов по обучению рекуррентной нейронной сети удалось добиться максимального значения Accuracy score на тестовом наборе равном 0.(8) уходя от 200 эпох обучения и придя к оптимальному количеству равному 60.

Все эти операции производились повторяя следующий код:

```
model.fit(X_train, y_train, epochs=60, callbacks=[tb_callback])

from sklearn.metrics import accuracy_score
accuracy_score(ytrue, yhat)

# 0.8888888888888888888888888
```

2.5 Приложение системы HMI для бесконтактного управления оборудованием

Получив обученную модель рекуррентной нейронной сети LSTM можно приступать к разработке основного приложения системы HMI для бесконтактного управления оборудованием с помощью жестовых команд посредством камеры.

Как и было описано ранее, командами, обученными для распознавания, являются команды: Вверх, Вниз, Влево, Вправо, Ок, Назад.

Исходя из этого, соответствуя этим командам будут выполняться симуляции нажатия аналогичными кнопок на клавиатуре компьютера. А именно кнопки: Стрелка вверх, Стрелка вниз, Стрелка влево, Стрелка вправо, Enter, Backspace.

```
actions = np.array([ 'up', 'down', 'left', 'right', 'ok', 'back'])
key_actions = {'up': 'up',
               'down': 'down',
               'left': 'left',
               'right': 'right',
               'ok': 'enter',
               'back': 'backspace'}
```

Далее реализованы следующие функции:

Интерполяция координат последних сорока кадром последовательности:

```
def interp_coords(x):
    coords = []
    for num,i in enumerate(x):
        if np.count_nonzero(i) != 0:
            coords.append([i,num])
    if not coords:
        return x
    result = []
    for i in range(126):
        result.append(np.interp(range(40),[e[1] for e in coords],[e[0][i] for e
in coords]))
    return np.array(result).transpose()
```

Выделение ключевых точек кистей рук с помощью Google MediaPipe Hands:

```
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results
```

Прорисовка ключевых точек на выходном изображении:

```
def draw_landmarks(image, results):
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_holistic.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())
```

Извлечение ключевых точек каждой отдельной руки и приведение к необходимому формату:

```
def extract_keypoints(results):
    try:
        for idx, handLms in enumerate(results.multi_hand_landmarks):
            lh = np.array([[res.x, res.y, res.z] for res in
                           results.multi_hand_landmarks[idx].landmark]).flatten()
\
                if    results.multi_handedness[idx].classification[0].label    ==
'Left' else np.zeros(21 * 3)
            rh = np.array([[res.x, res.y, res.z] for res in
                           results.multi_hand_landmarks[idx].landmark]).flatten()
\
                if    results.multi_handedness[idx].classification[0].label    ==
'Right' else np.zeros(21 * 3)
        return np.concatenate([lh, rh])
    except:
        return np.concatenate([np.zeros(21 * 3), np.zeros(21 * 3)])
```

Прорисовка интерфейса приложения с вероятностями распознавания жестовых команд и историей детектирования:

```
  colors = [(245,117,16), (117,245,16), (16,117,245), (16,217,245), (116,117,245),
(116,217,245)]
  def prob_viz(res, actions, input_frame, colors):
      output_frame = input_frame.copy()
      for num, prob in enumerate(res):
          cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40),
colors[num], -1)
          cv2.putText(output_frame,       actions[num],       (0,       85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

      return output_frame
```

Таким образом, имея все вышеописанные функции и обученную модель рекуррентной нейронной сети LSTM, реализована следующая основная функция приложения:

```
  sequence = []
  sentence = []
  predictions = []
  threshold = 0.5
  prev_len_sentence = 0

  cap = cv2.VideoCapture(0)
  with                              mp_holistic.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
      while cap.isOpened():
          ret, frame = cap.read()

          image, results = mediapipe_detection(frame, holistic)

          draw_landmarks(image, results)

          keypoints = extract_keypoints(results)
          sequence.append(keypoints)
          sequence = sequence[-40:]

          if len(sequence) == 40:
              res      =      model.predict(np.expand_dims(interp_coords(sequence),
axis=0))[0]
              print(actions[np.argmax(res)])
              predictions.append(np.argmax(res))

              if np.unique(predictions[-10:])[0]==np.argmax(res):
                  if res[np.argmax(res)] > threshold:

                      if len(sentence) != prev_len_sentence:
                          pag.press(key_actions[actions[np.argmax(res)]])
                          prev_len_sentence = len(sentence)
                      if len(sentence) > 0:
                          if actions[np.argmax(res)] != sentence[-1]:
```

```
                    sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        cv2.imshow('Detector', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Реализованное приложение имеет следующий интерфейс:

Рисунок - 2.18 Пример выполнения жестовой команды «Вверх»

Рисунок - 2.19 Пример выполнения жестовой команды «Вниз»

Рисунок - 2.20 Пример выполнения жестовой команды «Влево»

Рисунок - 2.21 Пример выполнения жестовой команды «Вправо»

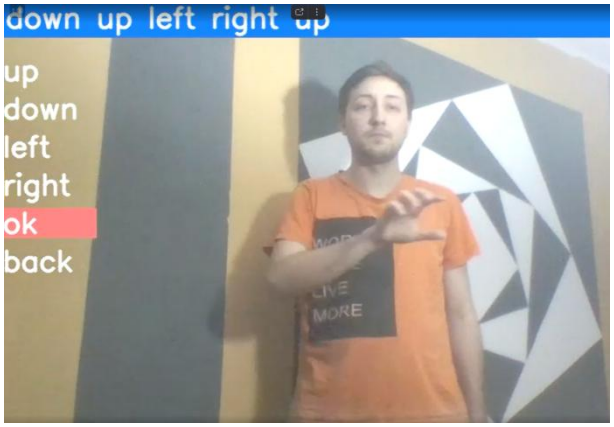


Рисунок - 2.22 Пример выполнения жестовой команды «ОК»

Рисунок - 2.23 Пример выполнения жестовой команды «Назад»

Как можно было видеть выше, реализованное приложение достаточно точно определяет команду. Единственным минусом на данный момент являются, выявленные на момент тестирования приложения, требования к вычислительным способностям компьютера и чёткости и скорости камеры. В следствии чего, в работе приложения возникают так называемые баги, вызванные уменьшением операционной системой выделенных на работу приложения ресурсов, замедляя работу приложения когда приложение работает в основном для него фоновом режиме. Приложение было разработано и протестировано на ноутбуке ASUS Tuf-Gaming FX505DY с процессором AMD Ryzen 5 3550H 4 ядра 8 потоков и базовой частотой 2.1 ГГц, 8 гигабайтами оперативной памяти DDR4 SODIMM. Увеличение вычислительной мощности компьютера и повышения приоритета процесса приложения значительно повышает его производительность, что было выявлено при тестировании приложения на ноутбуке HP с процессором Intel Core i7 10700 8 ядер 16 потоков и базовой частотой 3.8 ГГц и оперативной памятью 32 гигабайта DDR4 SODIMM.

3 ТЕХНИКА БЕЗОПАСНОСТИ ПРИ РАБОТЕ ЗА ПЕРСОНАЛЬНЫМ КОМПЬЮТЕРОМ

Персональный компьютер — электроприбор. От прочих электроприборов он отличается тем, что для него предусмотрена возможность длительной эксплуатации без отключения от электрической сети. Кроме обычного режима работы компьютер может находиться в режиме работы с пониженным электропотреблением или в дежурном режиме ожидания запроса. В связи с возможностью продолжительной работы компьютера без отключения от электросети следует уделить особое внимание качеству организации электропитания.

3.1 Требования по электрической безопасности

Недопустимо использование некачественных и изношенных компонентов в системе электроснабжения, а также их суррогатных заменителей: розеток, удлинителей, переходников, тройников. Недопустимо самостоятельно модифицировать розетки для подключения вилок, соответствующих иным стандартам. Электрические контакты розеток не должны испытывать механических нагрузок, связанных с подключением массивных компонентов (адаптеров, тройников и т. п.).

Все питающие кабели и провода должны располагаться с задней стороны компьютера и периферийных устройств. Их размещение в рабочей зоне пользователя недопустимо.

Запрещается производить какие-либо операции, связанные с подключением, отключением или перемещением компонентов компьютерной системы без предварительного отключения питания.

Компьютер не следует устанавливать вблизи электронагревательных приборов и систем отопления.

Недопустимо размещать на системном блоке, мониторе и периферийных устройствах посторонние предметы: книги, листы бумаги, салфетки, чехлы для защиты от пыли. Это приводит к постоянному или временному перекрытию вентиляционных отверстий.

Запрещается внедрять посторонние предметы в эксплуатационные или вентиляционные отверстия компонентов компьютерной системы.

Особенности электропитания монитора. Монитор имеет элементы, способные сохранять высокое напряжение в течение длительного времени после отключения от сети. Вскрытие монитора пользователем недопустимо ни при каких условиях. Это не только опасно для жизни, но и технически бесполезно, так как внутри монитора нет никаких органов, регулировкой или настройкой которых пользователь мог бы улучшить его работу. Вскрытие и обслуживание мониторов может производиться только в специальных мастерских.

3.2 Особенности электропитания системного блока

Все компоненты системного блока получают электроэнергию от блока питания. Блок питания ПК — это автономный узел, находящийся в верхней части системного блока. Правила техники безопасности не запрещают вскрывать системный блок, например при установке дополнительных внутренних устройств или их модернизации, но это не относится к блоку питания. Блок питания компьютера — источник повышенной пожаро-опасности, поэтому вскрытию и ремонту он подлежит только в специализированных мастерских.

Блок питания имеет встроенный вентилятор и вентиляционные отверстия. В связи с этим в нем неминуемо накапливается пыль, которая может вызвать короткое замыкание. Рекомендуется периодически (один - два раза в год) с помощью пылесоса удалять пыль из блока питания через вентиляционные отверстия без вскрытия системного блока. Особенно

важно производить эту операцию перед каждой транспортировкой или наклоном системного блока.

3.3 Система гигиенических требований

Длительная работа с компьютером может приводить к расстройствам состояния здоровья. Кратковременная работа с компьютером, установленным с грубыми нарушениям гигиенических норм и правил, приводит к повышенному утомлению. Вредное воздействие компьютерной системы на организм человека является комплексным. Параметры монитора оказывают влияние на органы зрения. Оборудование рабочего места влияет на органы опорно-двигательной системы. Характер расположения оборудования в компьютерном классе и режим его использования влияет как на общее психофизиологическое состояние организма, так и им органы зрения.

В прошлом монитор рассматривали в основном как источник вредных излучений, воздействующих прежде всего на глаза. Сегодня такой подход считается недостаточным. Кроме вредных электромагнитных излучений (которые на современных мониторах понижены до сравнительно безопасного уровня) должны учитываться параметры качества изображения, а они определяются не только монитором, но и видеоадаптером, то есть всей видеосистемы в целом.

Монитор компьютера должен удовлетворять следующим между народным стандартам безопасности:

- по уровню электромагнитных излучений — TCO 95
- по параметрам качества изображения (яркость, контрастность, мерцание, антибликовые свойства и др.) — TCO 99

Узнать о соответствии конкретной модели данным стандартам можно в сопроводительной документации. Для работы с мониторами,

удовлетворяющими данным стандартам, специальные защитные экраны не требуется.

На рабочем месте монитор должен устанавливаться таким образом, чтобы исключить возможность отражения от его экрана в сторону пользователя источников общего освещения помещения.

Расстояние от экрана монитора до глаз пользователя должно составлять от 50 до 70 см. Оптимально размещение монитора на расстоянии 1,5 D от глаз пользователя, где D — размер экрана монитора, измеренный по диагонали. Завыщенное расстояния от глаз до монитора приводит к дополнительному напряжению органов зрения, сказывается на затруднении перехода от работы с монитором к работе с книгой и проявляется в преждевременном развитии дальнозоркости.

Важным параметром является частота кадров, которая зависит от свойств монитора, видеоадаптера и программных настроек видеосистемы. Для работы с текстами минимально допустима частота кадров 72 Гц. Для работы с графикой рекомендуется частота кадров от 85 Гц и выше.

3.4 Требования к рабочему месту

В требования к рабочему месту входят требования к рабочему столу, посадочному месту (стулу, креслу), Подставкам для рук и ног. Несмотря на кажущуюся простоту, обеспечить правильное размещение элементов компьютерной системы и правильную посадку пользователя чрезвычайно трудно. Полное решение проблемы требует дополнительных затрат, сопоставимых по величине со стоимостью отдельных узлов компьютерной системы, поэтому и биту и на производстве этими требованиями часто пренебрегают.

Монитор должен быть установлен прямо перед пользователем и не требовать поворота головы или корпуса тела.

Рабочий стол и посадочное место должны иметь такую высоту, чтобы уровень глаз пользователя находился чуть выше центра монитора. На экран монитора следует смотреть сверху вниз, а не наоборот. Даже кратковременная работа с монитором, установленным слишком высоко, приводит к утомлению шейных отделов позвоночника.

Если при правильной установке монитора относительно уровня глаз выясняется, что ноги пользователя не могут свободно покоиться на полу, следует установить подставку для ног, желательно наклонную. Если ноги не имеют надежной опоры, это непременно ведет к нарушению осанки и утомлению позвоночника. Удобно, когда компьютерная мебель (стол и рабочее кресло) имеют средства для регулировки по высоте. В этом случае проще добиться оптимального положения.

Клавиатура должна быть расположена на такой высоте, чтобы пальцы рук располагались на ней свободно, без напряжения, а угол между плечом и предплечьем составлял 100° — 110°. При использовании обычных школьно-письменных столов добиться одновременно правильного положения и монитора, и клавиатуры практически невозможно. Для работы рекомендуется использовать специальные компьютерные столы, имеющие выдвижные полочки для клавиатуры.

При длительной работе с клавиатурой возможно утомление сухожилий кистевого сустава. Известно тяжелое профессиональное заболевание — кистевой туннельный синдром, связанное с неправильным положением рук на клавиатуре. Во избежание чрезмерных нагрузок на кисть желательно предоставить рабочее кресло с подлокотниками, уровень высоты которых, замеренный от пола, совпадает с уровнем высоты расположения клавиатуры.

При работе с мышью рука не должна находиться на весу. Локоть руки или хотя бы запястье должны иметь твердую опору. Если предусмотреть необходимое расположение рабочего стола и кресла затруднительно,

рекомендуется применить коврик для мыши, имеющий специальный опорный валик. Нередки случаи, когда в поисках опоры для руки (обычно правой) располагают монитор сбоку от пользователя (соответственно, слева), чтобы он работал вполоборота, опирая локоть или запястье правой руки о стол. Этот прием недопустим. Монитор должен обязательно находиться прямо перед пользователем.

3.5 Требования к организации пространства

Экран монитора — не единственный источник вредных электромагнитных излучений. Разработчики мониторов достаточно давно и успешно занимаются их преодолением. Меньше внимания уделяется вредным побочным излучениям, возникающим со стороны боковых и задней стенок оборудования. В современных компьютерных системах эти зоны наиболее опасны.

Монитор компьютера следует располагать так, чтобы задней стенкой он был обращен не к людям, а к стене помещения. В компьютерных помещениях, имеющих несколько компьютеров, рабочие места должны располагаться по периферии помещения, оставляя свободным центр. При этом дополнительно необходимо проверить каждое из рабочих мест на отсутствие прямого отражения внешних источников освещения. Как правило, добиться этого для всех рабочих мест одновременно достаточно трудно. Возможное решение состоит в использовании штор на окнах и продуманном размещении искусственных источников общего и местного освещения.

Сильными источниками электромагнитных излучений являются устройства бесперебойного питания. Располагать их следует как можно дальше от посадочных мест пользователей.

4 ЭНЕРГОСБЕРЕЖЕНИЕ

Энергосбережение — реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на эффективное (рациональное) использование (и экономное расходование) топливно-энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии. При разработке ПО самые большие траты приходятся на электроэнергию, потребляемую компьютерами.

4.1 Основные способы экономии энергии

Ключевыми мероприятиями оптимизации потребления электроэнергии на освещение являются:

- максимальное использование дневного света (повышение прозрачности и увеличение площади окон, дополнительные окна);
- повышение отражающей способности (белые стены и потолок);
- оптимальное размещение световых источников (местное освещение, направленное освещение);
- использование осветительных приборов только по необходимости;
- повышение светоотдачи существующих источников (замена люстр, плафонов, удаление грязи с плафонов, применение более эффективных отражателей);
- замена ламп накаливания на энергосберегающие (люминесцентные, компактные люминесцентные, светодиодные);
- применение устройств управления освещением (датчики движения и акустические датчики, датчики освещенности, таймеры, системы дистанционного управления);
- внедрение автоматизированной системы диспетчерского управления наружным освещением (АСДУ НО);

- установка интеллектуальных распределённых систем управления освещением (минимизирующих затраты на электроэнергию для данного объекта).

4.2 Энергосберегающие режимы работы компьютера

Спящий режим — это режим пониженного потребления электроэнергии, который позволяет быстро возобновить работу в режиме обычного потребления энергии (обычно в течение нескольких секунд) по требованию пользователя.

Перевод компьютера в спящий режим напоминает нажатие кнопки "Пауза" на проигрывателе DVD — компьютер немедленно останавливает все операции и готов к возврату в рабочий режим при необходимости.

Режим гибернации — это режим пониженного потребления электроэнергии, разработанный в первую очередь для ноутбуков. При переходе в спящий режим все открытые документы и параметры сохраняются в памяти и компьютер переходит в режим пониженного потребления электроэнергии, а при переходе в режим гибернации все открытые документы и программы сохраняются на жестком диске и затем компьютер выключается. Из всех энергосберегающих режимов, используемых в ОС Windows, для поддержания режима гибернации требуется наименьшее количество электроэнергии. Если в течение длительного промежутка времени не планируется использовать ноутбук и нет возможности подзарядить батарею, рекомендуется перевести ноутбук в режим гибернации.

Гибридный спящий режим — это режим, который разработан преимущественно для настольных компьютеров. Гибридный спящий режим сочетает в себе спящий режим и режим гибернации, поскольку все открытые документы и программы сохраняются в памяти и на жестком диске и компьютер переводится в режим пониженного потребления

электроэнергии. При неожиданном сбое питания операционная система Windows может легко восстановить данные с диска. Если гибридный спящий режим включен, переход в спящий режим автоматически переводит компьютер в гибридный спящий режим. На настольных компьютерах гибридный спящий режим обычно включен по умолчанию.

Таким образом, наиболее эффективное энергосбережение достигается при применении спящего режима в сочетании с режимом гибернации, либо при применении гибридного спящего режима.

4.3 Энергоэффективность в серверных помещениях

Обслуживание касается как серверного оборудования, так и системы кондиционирования. Нужно поддерживать правильные настройки. Например, не нужно задавать на кондиционере слишком низкую температуру, которая не нужна для работы серверов. Нужно следить за расположением коммутационных кабелей в серверных шкафах. Обычно со временем клубок кабелей копится, потоки воздуха начинают испытывать излишнее сопротивление. Пустые юниты в стойках стоит закрывать панелями-заглушками, чтобы холодный воздух не перемешивался с горячим. Нужно вовремя чистить и мыть оборудование, т.к. грязный наружный блок кондиционера или грязные воздушные фильтры во внутренних блоках существенно снижают теплообмен. Если не следить за всем выше перечисленным, существенно увеличиваются энергозатраты (компрессор и вентиляторы вынуждены работать более производительно).

4.4 Стандарты энергосберегающих технологий

В начале 90-х годов компания EPA (Environmental Protection Agency — Агентство по защите окружающей среды) начало проводить кампанию по сертификации энергосберегающих персональных компьютеров и периферийного оборудования. Компьютер или монитор во время

продолжительного простоя должен снизить энергопотребление до 30 Вт и более. Система, удовлетворяющая этим требованиям, может получить сертификат Energy Star.

4.4.1 Стандарт усовершенствованной системы управления питанием (Advanced Power Management — APM)

Разработан фирмой Intel совместно с Microsoft и определяет ряд интерфейсов между аппаратными средствами управления питанием и операционной системой компьютера. Полностью реализованный стандарт APM позволяет автоматически переключать компьютер между пятью состояниями в зависимости от текущего состояния системы. Каждое последующее состояние в приведенном ниже списке характеризуется уменьшением потребления энергии.

- Full On

Система полностью включена.

- APM Enabled

Система работает, некоторые устройства являются объектами управления для системы управления питанием. Неиспользуемые устройства могут быть выключены, может быть также остановлена или замедлена (т.е. снижена тактовая частота) работа тактового генератора центрального процессора.

- APM Standby (резервный режим)

Система не работает, большинство устройств находятся в состоянии потребления малой мощности. Работа тактового генератора центрального процессора может быть замедлена или остановлена, но необходимые параметры функционирования хранятся в памяти. Пользователь или операционная система могут запустить компьютер из этого состояния почти мгновенно.

- APM Suspend (режим приостановки).

Система не работает, большинство устройств пассивны. Тактовый генератор центрального процессора остановлен, а параметры функционирования хранятся на диске и при необходимости могут быть считаны в память для восстановления работы системы. Чтобы запустить систему из этого состояния, требуется некоторое время.

- Off (система отключена).

Система не работает. Источник питания выключен.

Для реализации режимов APM требуются аппаратные средства и программное обеспечение. Источниками питания ATX можно управлять с помощью сигнала Power_On и факультативного разъема питания с шестью контактами. (Необходимые для этого команды выдаются программой.) Изготовители также встраивают подобные устройства управления в другие элементы системы, например в системные платы, мониторы и дисководы. Операционные системы (такие как Windows), которые поддерживают APM, при наступлении соответствующих событий запускают программы управления питанием, наблюдая за действиями пользователя и прикладных программ. Однако операционная система непосредственно не посылает сигналы управления питанием аппаратным средствам. Система может иметь множество различных аппаратных устройств и программных функций, используемых при выполнении функций APM. Чтобы разрешить проблему сопряжения этих средств в операционной системе и аппаратных средствах предусмотрен специальный абстрактный уровень, который облегчает связь между различными элементами архитектуры

При запуске операционной системы загружается программа — драйвер APM, который связывается с различными прикладными программами и программными функциями. Именно они запускают действия управления питанием, причем все аппаратные средства, совместимые с APM,

связываются с системной BIOS. Драйвер APM и BIOS связаны напрямую; именно эту связь использует операционная система для управления режимами аппаратных средств.

Таким образом, чтобы функционировали средства APM, необходим стандарт, поддерживаемый схемами, встроенными в конкретные аппаратные устройства системы, системная BIOS и операционная система с драйвером APM. Если хотя бы один из этих компонентов отсутствует, APM работать не будет.

4.4.2 Стандарт усовершенствованной системы управления питанием (Advanced Power Management—APM)

Усовершенствованная конфигурация и интерфейс питания (Advanced Configuration and Power Interface— ACPI) впервые реализованы в современных BIOS и операционных системах Windows 98 и более поздних. Если BIOS компьютера поддерживает систему ACPI, то все управление питанием передается операционной системе. Это упрощает конфигурирование параметров, все они находятся в одном месте — в операционной системе.

4.4.3 Стандарт DPMS (Display Power Management Signaling - система сигналов управления питанием монитора)

Стандарт ассоциации VESA. Определяет состав сигналов, передаваемых компьютером в монитор, при вхождении системы от состояния простоя в режимы пониженного потребления энергии. В этих системных процедурах контроль берет на себя драйвер, посылающий соответствующие сигналы через графическую карту. При нажатии клавиши на клавиатуре или движении "мыши" монитор переходит в нормальный режим работы.

5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

В разделе представлена информация, из которой выводится целесообразность (или нецелесообразность) создания ПО. Содержит анализ затрат и результатов проекта. Позволяет определить, стоит ли вкладывать деньги в предлагаемый проект.

5.1 Оптимальные характеристики оборудования

Как инструмент разработки программ использовался ноутбук ASUS Tuf-Gaming FX505DY со следующими ключевыми техническими характеристиками:

- Процессор AMD Ryzen 5 CPU 3550H @ 2.1GHz
- Встроенный дисплей с разрешением 1920x1080
- ОЗУ DDR4 SODIMM 8 Гбайт

Отметим пункты для значительного увеличения скорости разработки программ:

- Наличие дополнительного монитора
- Мониторы с высоким разрешением экрана
- Высокопроизводительный многоядерный процессор
- Большой размер ОЗУ (16Гбайт - 32Гбайт)
- Наличие SSD в качестве основного дискового пространства (для операционной системы и запуска программ)

Указанные пункты помогут уменьшить время разработки программ за счет увеличения скорости работы сборщика проектов, возможности одновременного доступа к нескольким источникам информации без необходимости переключения внимания и скорости проведения тестов разработанного ПО за счет возможности параллельной работы сразу с несколькими эмуляторами вместо последовательной в случае работы с устаревшим оборудованием.

Минимальные системные требования:

- Процессор 2ГГц
- Разрешение экрана 1024x768
- ОЗУ 4Гбайта
- Наличие монитора, клавиатуры и мыши

5.2 Расчет себестоимости программного обеспечения

При расчете себестоимости необходимо учитывать целый ряд технико-экономических показателей, к которым относятся время разработки, размер и объем программного кода.

Совокупные затраты на создание скрипта состоят из нескольких компонентов. При расчете все эти компоненты необходимо приводить к одной единице измерения с целью обеспечения их однородности и удобства сопоставления. В качестве такой единицы удобнее всего рассматривать денежные выражения либо человеко-часы.

При разработке программных комплексов наибольшее значение в составе затрат имеют следующие составляющие:

1. затраты на подготовку и применение технологии и программных средств автоматизации разработки программ;
2. затраты на проектирование, программирование, отладку и испытания программ согласно требований пользователя или заказчика;
3. затраты на ЭВМ, используемые при разработке данного программного продукта;
4. затраты на изготовление опытного образца программного продукта как продукции производственно–технического назначения;
5. затраты на подготовку и повышение квалификации специалистов–разработчиков.

Затраты типа 1 и 4 можно исключить, учитывая условия и характер разработки данного программного продукта. Затраты типа 5 тоже не учитываются, так как их наиболее трудно учитывать и формализовать в конкретной работе. Имеются и дополнительные составляющие стоимости, которые не учитываются исходя из условий работы. К этим составляющим относятся: заработная плата, мебель, коммунальные услуги, охрана, противопожарные мероприятия, налоги и т.д.

Для определения составляющих затрат труда воспользуемся расчетом условного числа выполняемых строк программного кода в программе:

$$Q = q \ \ c \ \ (1 + p), \ (1)$$

где c = 1,2 — коэффициент увеличения затрат, характеризует увеличение затрат труда вследствие недостаточно полного описания задачи, уточнений и некоторой доработки. Этот коэффициент может принимать значения от 1,2 до 5;

p = 0.08 — коэффициент коррекции программы в ходе ее разработки, принимает значения от 0.05 до 0.1;

q = 1604— число строк программного кода.

В итоге получаем, что условное число операторов в программе принимает значение:

$$Q = 1604 \ \ 1{,}2 \ \ (1 + 0{,}08) = 2079$$

Важной составляющей при расчете себестоимости программного продукта является коэффициент квалификации k. Коэффициент квалификации берется из таблицы коэффициентов, приведенной ниже.

Опыт работы	Коэффициент
До 2–х лет	0,8
2 — 3 года	1,0
3 — 5 лет	1,1–1,2
5 — 7 лет	1,3–1,4

Больше 7 лет	1,5–1,6

Таблица 1 Коэффициенты квалификации

Согласно данным таблицы, необходимый коэффициент квалификации принимаем равным k = 1,2.

Для определения трудоемкости разработки приложения необходимо рассчитать следующие виды затрат:

- затраты труда на подготовку описания задачи;
- затраты труда на разработку блок–схемы приложения;
- затраты труда на программирование по блок–схеме;
- затраты труда на отладку кода;
- затраты труда на подготовку документации/описания.

Затраты труда на подготовку описания задачи рассчитываются по следующей формуле:

$$t_{\text{оп}} = \frac{T_{\min} + 4\ T_{\text{наиб.вероят.}} + T_{\max}}{6}, \quad (2)$$

где $T_{\min}$ — минимально возможная трудоемкость выполнения работы;

$T_{\text{наиб.вероя т.}}$ — наиболее вероятная трудоемкость;

$T_{\max}$ — максимально возможная трудоемкость.

По формуле 2 получаем, что затраты составляют

$$t_{\text{оп}} = \frac{8 + 4\ 14 + 20}{6} = 14 \quad \text{(чел/час)}.$$

Затраты труда на разработку блок–схемы алгоритма:

$$t_{\text{ал}} = \frac{Q\ \ B}{(75\ \ 85)\ \ k}, \quad (3)$$

где B — коэффициент увеличения затрат в зависимости от качества постановки задачи [1,2...5];

k — коэффициент квалификации берется из диапазона [75…85], (примем его равным 80):

$$t_{ал} = \frac{Q \;\; B}{(75 \;\; 85) \;\; k} = \frac{1604 \;\; 1{,}2}{80 \;\; 1.2} \approx 20 \;\; \text{(чел/час)}.$$

Затраты труда на программирование по блок–схеме рассчитывается по формуле (коэффициент квалификации берется из диапазона [60…75], примем его равным 70):

$$t_{пр} = \frac{Q}{(60 \;\; 75) \;\; k} \qquad (4)$$

$$t_{пр} = \frac{Q}{(60 \;\; 75) \;\; 1{,}2} = \frac{1604}{24} \approx 67 \,\text{(чел/час)}.$$

Затраты труда на отладку кода (коэффициент квалификации берется из диапазона [40…50], примем его равным 40):

$$t_{отл} = \frac{Q}{(40 \;\; 50) \;\; k} \qquad (5)$$

$$t_{отл} = \frac{Q}{(40 \;\; 50) \;\; k} = \frac{1604}{40 \;\; 1{,}2} \approx 33 \,\text{(чел/час)}.$$

Затраты труда на подготовку документации:

$$t_{Д} = t_{пр} + t_{оф} \,\text{(чел/час)}, \qquad (6)$$

где $t_{пр}$ — затраты труда на подготовку

$t_{оф}$ — время на оформление документов

$$t_{пр} = \frac{Q}{(150 \;\; 200) \;\; k} = \frac{1604}{175 \;\; 1{,}2} = 8\,,$$

$$t_{оф} = t_{пр} \;\; 0{,}75 \,\text{(чел/час)},$$

$$t_{оф} = 6$$

В итоге получаем:

$$t_{Д} = 14 \,\text{(чел/час)}.$$

Тогда итоговая трудоемкость разработки данного программного обеспечения равна:

$$t_{\Sigma} = t_{оп} + t_{ал} + t_{пр} + t_{отл} + t_{Д} \,\text{(чел/час)},$$

$$t_{\Sigma} = 14 + 20 + 67 + 33 + 14 = 148 \,\text{(чел/час)}.$$

Это время приблизительно равняется одному месяцу при длине рабочего дня в 8 часов, что соответствует реальному времени разработки данного программного продукта.

При расчете затрат на разработку программного обеспечения необходимо разделить на составляющие: заработная плата (основная и дополнительная), отчисления на социальные нужды, эксплуатационные затраты (электроэнергия, техническое обслуживание и текущий ремонт), накладные расходы, материалы и комплектующие.

Основная заработная плата рассчитывается по формуле:

$$З_{осн} = \frac{t}{t_{ср}\ 8}\ TC\ (\text{руб.}), \quad (7)$$

где t – суммарные затраты труда, 148(чел/час);

$t_{ср}$ – среднее число рабочих дней в месяце: (365–107)/12 = 21,5 дней;

Тарифная ставка (ТС) представляет собой минимальный размер оплаты труда (МРОТ), увеличенный в зависимости от тарифного коэффициента $k_т$, соответствующего данному разряду работ. Для 13–ого разряда работ, который соответствует работе программиста, тарифный коэффициент равен 3,04.

$$ТС = МРОТ \cdot k_т = 480 \cdot 3,04 = 1459,2\ (\text{руб/мес}),$$

$$З_{осн} = \frac{148}{21,5\ 8}\ 1459,2 = 1255,59\ (\text{руб.}),$$

$$t_з = \frac{t}{t_{ср}\ 8} = \frac{1604}{21,5\ 8} = 0,84\ (\text{мес.}),$$

где $t_з$ - время, затраченное на создание программного обеспечения (в месяцах).

Надбавка за стаж составляет 5% от основной заработной платы

$$З_{доп} = 5\%\ \ З_{осн} = 0,05\ \ 1255,59 = 62,7795\ (\text{руб.})$$

Премия 30 % от основной заработной платы за успешно выполненное задание в срок

$$З_{прем} = 30\% \ З_{осн} = 0{,}3 \ 1255{,}59 = 376{,}677 \text{ (руб.)}.$$

Отчисление на социальное страхование составляют 35% от всей заработной платы:

$$З_{соц.страх.} = 35\% \ (З_{осн} + З_{доп} + З_{прем}) = 0{,}35 \ (1255{,}59 + 62{,}7795 + 376{,}677) = 593{,}27 \text{ (руб.)}$$

Эксплуатационные затраты возникают при эксплуатации ЭВМ:

Стоимость электроэнергии:

$$C_{ээ} = M \ \kappa_з \ F_{эф} \ C_{квт.ч.}, \quad (8)$$

где M — мощность ЭВМ (0,06 кВт);

$\kappa_з$ — коэффициент загрузки (0,7);

$C_{кВт.ч.}$ — стоимость 1 кВт час электроэнергии (0,165 руб.);

$F_{эф}$ — эффективный фонд, рассчитывается по формуле:

$$F_{эф} = Д_{ном} \ 8 \ \left(1 - \frac{f}{100}\right), \quad (9)$$

где $Д_{ном}$ — номинальное число рабочих дней в году (255);

f — планируемый процент времени на ремонт ЭВМ (2%);

тогда

$$F_{эф} = 255 \ 8 \ \left(1 - \frac{2}{100}\right) = 1999 \text{ (час.)},$$

тогда стоимость электроэнергии составит:

$$C_{ээ} = 0{,}06 \ 0{,}7 \ 1999 \ 0{,}165 = 13{,}85 \text{ (руб.)}.$$

Это стоимость электроэнергии за год, а за период разработки программного обеспечения стоимость электроэнергии составит:

$$C_{ээ.разраб.} = C_{ээ} \ \frac{t}{Д_{ном} \ 8}, \quad (10)$$

где t — суммарные затраты труда, 1273 (чел/час);

$Д_{ном}$ — номинальное число рабочих дней в году (255);

$C_{ээ}$ — стоимость электроэнергии за год.

Тогда стоимость электроэнергии за период разработки программного обеспечения составит:

$$C_{ээ.разраб.} = 13{,}85 \ \frac{148}{255 \ 8} = 1 \text{(руб.)}.$$

Предположительная стоимость компьютера с необходимыми параметрами будет составлять 2000 (руб.).

Техническое обслуживание и ремонт составляют 2.5% от стоимости компьютера, тогда получаем:

$$C_{то} = 2{,}5\% \ C_{ЭВМ} = 0{,}025 \ 2000 = 50 \text{(руб.)}.$$

Это стоимость и обслуживание технического ремонта за год, а за период разработки стоимость технического обслуживания и текущего ремонта составит:

$$C_{то.разраб.} = C_{то} \ \frac{t}{Д_{ном} \ 8} = 25 \ \frac{148}{255 \ 8} = 1{,}8 \text{(руб.)},$$

где t — суммарные затраты труда, 1273 (чел.час);

$Д_{ном}$ — номинальное число рабочих дней в году (255).

Накладные расходы составляют 10% от основной заработной платы:

$$C_{накл} = З_{осн} \ 0{,}1 = 1255{,}59 \ 0{,}1 = 125{,}6 \text{(руб.)}.$$

Материалы и комплектующие составляют 1.5% от стоимости оборудования:

$$C_{МиК} = 1{,}5\% \ C_{ЭВМ} = 0{,}015 \ 2000 = 30 \text{(руб.)}$$

Таким образом, суммарные расходы на разработку программного обеспечения составили:

$$C = З_{осн} + З_{доп} + З_{прем} + З_{соц.страх.} + C_{ээ.разраб} + C_{накл} + C_{МиК} + C_{ТО.Разраб} \quad \text{(руб.)} \quad (11)$$

$$C = 1255{,}59 + 62{,}78 + 376{,}677 + 593{,}27 + 1 + 1{,}8 + 30 + 125{,}6 = 2446{,}717 \quad \text{(руб.)}$$

Составляющие себестоимости	Сумма, BYN
Зарплата	1255,59
Социальное страхование	593,27
Эксплуатационные расходы	1
Накладные расходы	125,6
Материалы и комплектующие	30
Техническое обслуживание	1,8
Итого	2007,26

Таблица 2 Смета затрат а разработку

Согласно данным таблицы 10 себестоимость разработанного программного продукта составляет 2007,26 (руб.).

ЗАКЛЮЧЕНИЕ

В ходе разработки дипломного проекта были поставлены и выполнены следующие задачи:

6. Разработан инструмент для сбора данных.
7. Собраны данные для обучения нейронной сети.
8. Проведён анализ и обработка собранных данных для подготовки датасета.
9. Обучена рекуррентная нейронная сеть LSTM для классификации.
10. Разработано приложение системы HMI для идентификации жестовых команд в видеопотоке.

Основное приложение идентификации жестовых команд в видеопотоке выполняет классификацию жеста, относя жест к одному из шести жестов, по последовательности распознанных ключевых точек кистей рук с помощью обученной в ходе разработки дипломного проекта рекуррентной нейронной сети LSTM и реагирует на жест, выполняя заданную в коде программы задачу. Точность обученной модели рекуррентной нейронной сети на тестовых данных составила 0.(8) (ноль целых и восемь в периоде), что равно 88.(8)%. С помощью приложения для сбора данных возможен сбор большего большего количества данных для обучения, тем самым повышая качество и точность модели, и большего количества жестов для реализации более функциональной системы в будущем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Интерприторы python — Microsoft Documents [Электронный ресурс] / Режим доступа:https://docs.microsoft.com/ru-ru/visualstudio/python/installing-python-interpreters?view=vs-2019. Дата доступа: 07.05.2022.

2. Python — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Python. Дата доступа: 07.05.2022.

3. IPython — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/IPython. Дата доступа: 07.05.2022.

4. OpenCV — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/OpenCV. Дата доступа: 07.05.2022.

5. Google MediaPipe — Google Github [Электронный ресурс] / Режим доступа: https://google.github.io/mediapipe/. Дата доступа: 07.05.2022.

6. Google MediaPipe Hands — Google Github [Электронный ресурс] / Режим доступа: https://google.github.io/mediapipe/solutions/hands. Дата доступа: 07.05.2022.

7. Numpy — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/numpy. Дата доступа: 07.05.2022.

8. Pandas — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/pandas. Дата доступа: 07.05.2022.

9. Matplotlib — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Matplotlib. Дата доступа: 07.05.2022.

10. Scikit-learn— Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Scikit-learn. Дата доступа: 07.05.2022.

11. TensorFlow — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/TensorFlow. Дата доступа: 07.05.2022.

12. Keras — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Keras. Дата доступа: 07.05.2022.

13. Long short-term memory — Habr [Электронный ресурс] / Режим доступа: https://habr.com/ru/company/wunderfund/blog/331310/. Дата доступа: 07.05.2022.

14. Softmax — Википедия [Электронный ресурс] / Википедия — свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Softmax. Дата доступа: 07.05.2022.

ПРИЛОЖЕНИЕ

Исходный код приложения сбора данных

hands.py

```
import cv2
import numpy as np
import os
import time
import mediapipe as mp


def make_dirs(data_path, actions, no_sequences):
    for action in actions:
        for sequence in range(no_sequences):
            try:
                os.makedirs(os.path.join(data_path, action, str(sequence)))
            except:
                pass


def only_hands(mp_hands, mp_drawing, mp_drawing_styles,
               flip=False, threshold=0.5, static_mode=False, camera=0):
    cap = cv2.VideoCapture(camera)
    p_time = 0
    with mp_hands.Hands(
            min_detection_confidence=threshold,
            min_tracking_confidence=threshold, static_image_mode=static_mode)
as hands:
        while cap.isOpened():
            success, image = cap.read()
            if not success:
                print("Ignoring empty camera frame.")
                # If loading a video, use 'break' instead of 'continue'.
                continue

            if flip:
                image = cv2.flip(image, 1)

            # FPS counter
            c_time = time.time()
            fps = 1 / (c_time - p_time)
            p_time = c_time
            cv2.putText(image, f'FPS: {int(fps)}', (40, 70),
cv2.FONT_HERSHEY_DUPLEX, 3, (255, 245, 215), 3)

            # To improve performance, optionally mark the image as not
writeable to
            # pass by reference.
            image.flags.writeable = False
```

```
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            results = hands.process(image)

            # Draw the hand annotations on the image.
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(
                        image,
                        hand_landmarks,
                        mp_hands.HAND_CONNECTIONS,
                        mp_drawing_styles.get_default_hand_landmarks_style(),

mp_drawing_styles.get_default_hand_connections_style())
            # Flip the image horizontally for a selfie-view display.
            cv2.namedWindow('Hands', cv2.WINDOW_NORMAL)
            cv2.imshow('Hands', image)
            if cv2.waitKey(5) & 0xFF == 27:
                break
    cap.release()
    cv2.destroyAllWindows()


def extract_keypoints(results):
    try:
        for idx, handLms in enumerate(results.multi_hand_landmarks):
            lh = np.array([[res.x, res.y, res.z] for res in

results.multi_hand_landmarks[idx].landmark]).flatten() \
                if results.multi_handedness[idx].classification[0].label ==
'Left' else np.zeros(21 * 3)
            rh = np.array([[res.x, res.y, res.z] for res in

results.multi_hand_landmarks[idx].landmark]).flatten() \
                if results.multi_handedness[idx].classification[0].label ==
'Right' else np.zeros(21 * 3)
        return np.concatenate([lh, rh])
    except:
        return np.concatenate([np.zeros(21 * 3), np.zeros(21 * 3)])


def get_keypoints(mp_hands, mp_drawing, mp_drawing_styles,
                  actions, no_sequences, sequence_length, data_path,
                  threshold=0.5, static_mode=False, camera=0):
    cap = cv2.VideoCapture(camera)
    p_time = 0
    with mp_hands.Hands(
            min_detection_confidence=threshold,
            min_tracking_confidence=threshold,
            static_image_mode=static_mode) as hands:
```

```
        # Loop through actions
        for action in actions:
            # Loop through sequences aka videos
            for sequence in range(no_sequences):
                # Loop through video length aka sequence length
                for frame_num in range(sequence_length):

                    # Read feed
                    success, image = cap.read()
                    if not success:
                        print("Ignoring empty camera frame.")
                        # If loading a video, use 'break' instead of
'continue'.
                        continue

                    # FPS counter
                    c_time = time.time()
                    fps = 1 / (c_time - p_time)
                    p_time = c_time
                    cv2.putText(image, f'FPS: {int(fps)}', (40, 70),
cv2.FONT_HERSHEY_DUPLEX, 3, (255, 245, 215), 3)

                    # Make detections
                    image.flags.writeable = False
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                    results = hands.process(image)

                    # Draw landmarks
                    image.flags.writeable = True
                    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                    if results.multi_hand_landmarks:
                        for hand_landmarks in results.multi_hand_landmarks:
                            mp_drawing.draw_landmarks(
                                image,
                                hand_landmarks,
                                mp_hands.HAND_CONNECTIONS,

mp_drawing_styles.get_default_hand_landmarks_style(),

mp_drawing_styles.get_default_hand_connections_style())

                    # NEW Apply wait logic
                    if frame_num == 0:
                        cv2.putText(image, 'STARTING COLLECTION', (120, 200),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 245,
215), 4, cv2.LINE_AA)
                        cv2.putText(image, 'Collecting frames for {} Video
Number {}'.format(action, sequence),
                                    (15, 12),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 245,
215), 1, cv2.LINE_AA)
```

```
                        # Show to screen
                        cv2.imshow('OpenCV Feed', image)
                        cv2.waitKey(1500)
                    else:
                        cv2.putText(image, 'Collecting frames for {} Video
    Number {}'.format(action, sequence),
                                    (15, 12),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 245,
    215), 1, cv2.LINE_AA)
                        # Show to screen
                        cv2.imshow('OpenCV Feed', image)

                    # NEW Export keypoints
                    keypoints = extract_keypoints(results)
                    npy_path = os.path.join(data_path, action, str(sequence),
    str(frame_num))
                    np.save(npy_path, keypoints)

                    # Break gracefully
                    if cv2.waitKey(5) & 0xFF == 27:
                        cap.release()
                        cv2.destroyAllWindows()
                        return
```

app.py

```
from PyQt5 import QtCore, QtGui, QtWidgets
import sys
import os
import mediapipe as mp
import hands
from ui import Ui_MainWindow

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()
ui = Ui_MainWindow()
ui.setupUi(MainWindow)
MainWindow.show()

mp_h = mp.solutions.hands
mp_d = mp.solutions.drawing_utils
mp_ds = mp.solutions.drawing_styles

ui.textEdit.setText("default")

get_actions = lambda text: text.split("\n")


def demo():
    hands.only_hands(mp_h, mp_d, mp_ds, camera=int(ui.spinBox.text()))
```

```
    def start():
        hands.make_dirs(os.path.join(ui.lineEdit.text()),
                        actions=get_actions(ui.textEdit.toPlainText()),
                        no_sequences=int(ui.lineEdit_2.text()))
        hands.get_keypoints(mp_h, mp_d, mp_ds,
                            get_actions(ui.textEdit.toPlainText()),
                            no_sequences=int(ui.lineEdit_2.text()),
                            sequence_length=int(ui.lineEdit_3.text()),
                            data_path=os.path.join(ui.lineEdit.text()),
                            camera=int(ui.spinBox.text()))


    ui.pushButton.clicked.connect(start)
    ui.pushButton_2.clicked.connect(demo)

    sys.exit(app.exec_())
```

ui.py

```
    # -*- coding: utf-8 -*-

    from PyQt5 import QtCore, QtGui, QtWidgets


    class Ui_MainWindow(object):
        def setupUi(self, MainWindow):
            MainWindow.setObjectName("MainWindow")
            MainWindow.setWindowModality(QtCore.Qt.NonModal)
            MainWindow.setEnabled(True)
            MainWindow.resize(340, 400)
            MainWindow.setMinimumSize(QtCore.QSize(340, 400))
            MainWindow.setMaximumSize(QtCore.QSize(340, 400))
            MainWindow.setBaseSize(QtCore.QSize(340, 410))
            MainWindow.setMouseTracking(False)
            MainWindow.setAcceptDrops(False)
            MainWindow.setWindowOpacity(1.0)
            MainWindow.setStyleSheet("QMainWindow{\n"
    "    background-color: #E6E6FF;\n"
    "}")
            MainWindow.setWindowFilePath("")
            MainWindow.setIconSize(QtCore.QSize(32, 32))
            MainWindow.setAnimated(True)
            self.centralwidget = QtWidgets.QWidget(MainWindow)
            self.centralwidget.setObjectName("centralwidget")
            self.horizontalLayoutWidget = QtWidgets.QWidget(self.centralwidget)
            self.horizontalLayoutWidget.setGeometry(QtCore.QRect(10, 330, 321, 61))
            self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")
            self.horizontalLayout                                                =
QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)
```

```
            self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
            self.horizontalLayout.setSpacing(5)
            self.horizontalLayout.setObjectName("horizontalLayout")
            self.pushButton = QtWidgets.QPushButton(self.horizontalLayoutWidget)
            self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            self.pushButton.setStyleSheet("QPushButton{\n"
    "    height: 40px;\n"
    "    padding-bottom: 5px;\n"
    "    border: none;\n"
    "    font: 63 24pt \"Bahnschrift SemiBold Condensed\";\n"
    "    color: #444444;\n"
    "    background-color:#FFF5D7;\n"
    "    border-radius: 5px;\n"
    "}\n"
    "QPushButton:hover{\n"
    "    color: #222222;\n"
    "    background-color: #FFF9E9;\n"
    "}\n"
    "QPushButton:pressed{\n"
    "    background-color:#bbbbff;\n"
    "}")
            self.pushButton.setObjectName("pushButton")
            self.horizontalLayout.addWidget(self.pushButton)
            self.pushButton_2 = QtWidgets.QPushButton(self.horizontalLayoutWidget)
            self.pushButton_2.setCursor(QtGui.QCursor(QtCore.Qt.PointingHandCursor))
            self.pushButton_2.setStyleSheet("QPushButton{\n"
    "    height: 40px;\n"
    "    padding-bottom: 5px;\n"
    "    border: none;\n"
    "    font: 63 24pt \"Bahnschrift SemiBold Condensed\";\n"
    "    color: #444444;\n"
    "    background-color:#FFF5D7;\n"
    "    border-radius: 5px;\n"
    "}\n"
    "QPushButton:hover{\n"
    "    color: #222222;\n"
    "    background-color: #FFF9E9;\n"
    "}\n"
    "QPushButton:pressed{\n"
    "    background-color:#bbbbff;\n"
    "}")
            self.pushButton_2.setObjectName("pushButton_2")
            self.horizontalLayout.addWidget(self.pushButton_2)
            self.horizontalLayoutWidget_2 = QtWidgets.QWidget(self.centralwidget)
            self.horizontalLayoutWidget_2.setGeometry(QtCore.QRect(10, 10, 321, 315))
            self.horizontalLayoutWidget_2.setObjectName("horizontalLayoutWidget_2")
            self.horizontalLayout_2                                                 =
QtWidgets.QHBoxLayout(self.horizontalLayoutWidget_2)
            self.horizontalLayout_2.setContentsMargins(0, 0, 0, 0)
            self.horizontalLayout_2.setSpacing(5)
            self.horizontalLayout_2.setObjectName("horizontalLayout_2")
```

```
            self.verticalLayout_2 = QtWidgets.QVBoxLayout()
            self.verticalLayout_2.setObjectName("verticalLayout_2")
            self.label_4 = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
            self.label_4.setStyleSheet("color: #444;\n"
   "font: 63 12pt \"Bahnschrift SemiBold\";")
            self.label_4.setObjectName("label_4")
            self.verticalLayout_2.addWidget(self.label_4)
            self.textEdit = QtWidgets.QTextEdit(self.horizontalLayoutWidget_2)
            self.textEdit.setStyleSheet("QTextEdit{\n"
   "     font: 63 12pt \"Bahnschrift\";\n"
   "    background-color:#ccccff;\n"
   "    border-radius: 10px;\n"
   "}\n"
   "QTextEdit:hover{\n"
   "    background-color:#bbbbff;\n"
   "}")
            self.textEdit.setObjectName("textEdit")
            self.verticalLayout_2.addWidget(self.textEdit)
            self.horizontalLayout_2.addLayout(self.verticalLayout_2)
            self.verticalLayout_3 = QtWidgets.QVBoxLayout()
            self.verticalLayout_3.setObjectName("verticalLayout_3")
            spacerItem = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
QtWidgets.QSizePolicy.Expanding)
            self.verticalLayout_3.addItem(spacerItem)
            self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
            self.horizontalLayout_3.setObjectName("horizontalLayout_3")
            self.label_5 = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
            self.label_5.setStyleSheet("color: #444;\n"
   "font: 63 12pt \"Bahnschrift SemiBold\";")
            self.label_5.setObjectName("label_5")
            self.horizontalLayout_3.addWidget(self.label_5)
            self.spinBox = QtWidgets.QSpinBox(self.horizontalLayoutWidget_2)
            self.spinBox.setStyleSheet("QSpinBox{\n"
   "    height: 32px;\n"
   "     font: 63 16pt \"Bahnschrift\";\n"
   "    background-color:#ccccff;\n"
   "    border-radius: 5px;\n"
   "}\n"
   "QSpinBox:hover{\n"
   "    background-color:#bbbbff;\n"
   "}")
            self.spinBox.setObjectName("spinBox")
            self.horizontalLayout_3.addWidget(self.spinBox)
            self.verticalLayout_3.addLayout(self.horizontalLayout_3)
            spacerItem1           =           QtWidgets.QSpacerItem(20,           40,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
            self.verticalLayout_3.addItem(spacerItem1)
            self.label = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
            self.label.setStyleSheet("color: #444;\n"
   "font: 63 12pt \"Bahnschrift SemiBold\";")
            self.label.setObjectName("label")
```

```
          self.verticalLayout_3.addWidget(self.label)
          self.lineEdit = QtWidgets.QLineEdit(self.horizontalLayoutWidget_2)
          self.lineEdit.setStyleSheet("QLineEdit{\n"
    "    height: 25px;\n"
    "     font: 63 12pt \"Bahnschrift\";\n"
    "    background-color:#ccccff;\n"
    "    border-radius: 5px;\n"
    "}\n"
    "QLineEdit:hover{\n"
    "    background-color:#bbbbff;\n"
    "}")
          self.lineEdit.setObjectName("lineEdit")
          self.verticalLayout_3.addWidget(self.lineEdit)
          spacerItem2           =           QtWidgets.QSpacerItem(20,           40,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
          self.verticalLayout_3.addItem(spacerItem2)
          self.label_2 = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
          self.label_2.setStyleSheet("color: #444;\n"
    "font: 63 12pt \"Bahnschrift SemiBold\";\n"
    "word-wrap: break-word;")
          self.label_2.setObjectName("label_2")
          self.verticalLayout_3.addWidget(self.label_2)
          self.lineEdit_2 = QtWidgets.QLineEdit(self.horizontalLayoutWidget_2)
          self.lineEdit_2.setStyleSheet("QLineEdit{\n"
    "    height: 25px;\n"
    "     font: 63 12pt \"Bahnschrift\";\n"
    "    background-color:#ccccff;\n"
    "    border-radius: 5px;\n"
    "}\n"
    "QLineEdit:hover{\n"
    "    background-color:#bbbbff;\n"
    "}")
          self.lineEdit_2.setObjectName("lineEdit_2")
          self.verticalLayout_3.addWidget(self.lineEdit_2)
          spacerItem3           =           QtWidgets.QSpacerItem(20,           40,
QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
          self.verticalLayout_3.addItem(spacerItem3)
          self.label_3 = QtWidgets.QLabel(self.horizontalLayoutWidget_2)
          self.label_3.setStyleSheet("color: #444;\n"
    "font: 63 12pt \"Bahnschrift SemiBold\";")
          self.label_3.setObjectName("label_3")
          self.verticalLayout_3.addWidget(self.label_3)
          self.lineEdit_3 = QtWidgets.QLineEdit(self.horizontalLayoutWidget_2)
          self.lineEdit_3.setStyleSheet("QLineEdit{\n"
    "    height: 25px;\n"
    "     font: 63 12pt \"Bahnschrift\";\n"
    "    background-color:#ccccff;\n"
    "    border-radius: 5px;\n"
    "}\n"
    "QLineEdit:hover{\n"
    "    background-color:#bbbbff;\n"
```

```
"}")
        self.lineEdit_3.setObjectName("lineEdit_3")
        self.verticalLayout_3.addWidget(self.lineEdit_3)
        self.horizontalLayout_2.addLayout(self.verticalLayout_3)
        MainWindow.setCentralWidget(self.centralwidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Hands detector"))
        self.pushButton.setText(_translate("MainWindow", "start"))
        self.pushButton_2.setText(_translate("MainWindow", "demo"))
        self.label_4.setText(_translate("MainWindow", "Actions"))
        self.label_5.setText(_translate("MainWindow", "WebCemera"))
        self.label.setText(_translate("MainWindow", "Folder"))
        self.lineEdit.setText(_translate("MainWindow", "Data"))
        self.label_2.setText(_translate("MainWindow", "Number of sequences"))
        self.lineEdit_2.setText(_translate("MainWindow", "30"))
        self.label_3.setText(_translate("MainWindow", "Sequence length"))
        self.lineEdit_3.setText(_translate("MainWindow", "40"))
```

Исходный код визуализации

Visualization.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import numpy as np
import os
import matplotlib.pyplot as plt


# In[10]:


actions = np.array([ 'up', 'down', 'left', 'right', 'ok', 'back'])

no_sequences = 30

sequence_length = 40


# In[11]:


label_map = {label:num for num, label in enumerate(actions)}
```

```
# In[12]:


parts = [
    [0, 1, 0, 5, 0, 17, 'green'],
    [5,9,9,13,13,17,'red'],
    [1,2,2,3,3,4,'blue'],
    [5,6,6,7,7,8,'yellow'],
    [9,10,10,11,11,12,'orange'],
    [13,14,14,15,15,16,'pink'],
    [17,18,18,19,19,20,'purple'],

    #[0+21, 1+21, 0+21, 5+21, 0+21, 17+21, 'green'],
    #[5+21,9+21,9+21,13+21,13+21,17+21,'red'],
    #[1+21,2+21,2+21,3+21,3+21,4+21,'blue'],
    #[5+21,6+21,6+21,7+21,7+21,8+21,'yellow'],
    #[9+21,10+21,10+21,11+21,11+21,12+21,'orange'],
    #[13+21,14+21,14+21,15+21,15+21,16+21,'pink'],
    #[17+21,18+21,18+21,19+21,19+21,20+21,'purple']
]

def draw_parts(ax, dx, dy, dz, parts):
    ax.plot3D([dx[parts[0]],dx[parts[1]]],          [dy[parts[0]],dy[parts[1]]],
[dz[parts[0]],dz[parts[1]]], parts[6])
    ax.plot3D([dx[parts[2]],dx[parts[3]]],          [dy[parts[2]],dy[parts[3]]],
[dz[parts[2]],dz[parts[3]]], parts[6])
    ax.plot3D([dx[parts[4]],dx[parts[5]]],          [dy[parts[4]],dy[parts[5]]],
[dz[parts[4]],dz[parts[5]]], parts[6])


# In[13]:


def get_coords(X):
    dx = X[:63:3]
    dy = X[1:63:3]
    dz = X[2:63:3]
    return dx,dy,dz


# In[14]:


label_map


# In[15]:
```

```
# sequences, labels = [], []
# for action in actions:
#     for sequence in range(no_sequences):
#         window = []
#         for frame_num in range(sequence_length):
#                 res = np.load(os.path.join(DATA_PATH, action, str(sequence),
"{}.npy".format(frame_num)))
#             window.append(res)
#         sequences.append(window)
#         labels.append(label_map[action])


# In[4]:


sequences = np.load('data/sequences.npy').tolist()
labels = np.load('data/labels.npy').tolist()


# In[24]:


np.array(labels).shape
np.array(sequences).shape


# In[25]:


np.save('sequences.npy', np.array(sequences))
np.save('labels.npy', np.array(labels))


# In[26]:


print(sequences[4])


# In[9]:


X = np.array(sequences)


# In[10]:


X.shape
```

```
# In[11]:


def only_right_hand(X):
    res = []
    for i in X:
        res_j = []
        for j in i:
            res_j.append(j[:63])
        res.append(res_j)
    return res


# In[13]:


X.shape


# In[14]:


def interp_coords(x):
    coords = []
    for num,i in enumerate(x):
        if np.count_nonzero(i) != 0:
            coords.append([i,num])
    result = []
    for i in range(63):
        result.append(np.interp(range(40),[e[1] for e in coords],[e[0][i] for e
in coords]))
    return np.array(result).transpose()


# In[15]:


def draw_hand(x, parts, save=False, fixed_axes=False, dinamic_view=False):
    for num,i in enumerate(x):
        dx,dy,dz = get_coords(i)

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d', label=num)
        ax.scatter(dx, dy, dz)

        for i in parts:
            draw_parts(ax, dx, dy, dz, i)

        ax.view_init(260, -120)
        figure = plt.gcf()
```

```
        if fixed_axes:
            plt.xlim([0,1])
            plt.ylim([0,1])

        if dinamic_view:
            ax.view_init(260+2*num, -120+4*num)

        plt.show()
        if save:
            figure.savefig(os.path.join("hand_frames", "{}.jpg".format(num)))
        plt.close(fig)


# In[18]:


sample = only_right_hand(X)
draw_hand(interp_coords(sample[139]), parts, save=False, dinamic_view=False,
fixed_axes=True)
#draw_hand(sample[139], parts, save=True)
```

Исходный код исследования

detection research.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # 1.

# In[1]:


get_ipython().run_line_magic('pip', 'install tensorflow==2.4.1 tensorflow-
gpu==2.4.1 opencv-python mediapipe sklearn matplotlib')


# In[1]:


import cv2
import numpy as np

import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

```
# In[ ]:




# # 2.

# In[2]:


mp_holistic = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles


# In[3]:


def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results


# In[4]:


def draw_landmarks(image, results):
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_holistic.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())


# In[5]:


def extract_keypoints(results):
    try:
        for idx, handLms in enumerate(results.multi_hand_landmarks):
            lh = np.array([[res.x, res.y, res.z] for res in
                           results.multi_hand_landmarks[idx].landmark]).flatten()
\
```

```
                if    results.multi_handedness[idx].classification[0].label    ==
'Left' else np.zeros(21 * 3)
             rh = np.array([[res.x, res.y, res.z] for res in
                           results.multi_hand_landmarks[idx].landmark]).flatten()
\
                if    results.multi_handedness[idx].classification[0].label    ==
'Right' else np.zeros(21 * 3)
          return np.concatenate([lh, rh])
      except:
          return np.concatenate([np.zeros(21 * 3), np.zeros(21 * 3)])


  # # 3.

  # In[6]:


  from sklearn.model_selection import train_test_split
  from tensorflow.keras.utils import to_categorical


  # In[7]:


  actions = np.array([ 'up', 'down', 'left', 'right', 'ok', 'back'])


  # In[8]:


  label_map = {label:num for num, label in enumerate(actions)}


  # In[9]:


  label_map


  # In[11]:


  sequences = np.load("data/sequences.npy")
  labels = np.load("data/labels.npy")


  # In[12]:


  def interp_coords(x):
      coords = []
```

```
        for num,i in enumerate(x):
            if np.count_nonzero(i) != 0:
                coords.append([i,num])
        if not coords:
            return x
        result = []
        for i in range(126):
            result.append(np.interp(range(40),[e[1] for e in coords],[e[0][i] for e
in coords]))
        return np.array(result).transpose()


    # In[13]:


    sequences = np.array([interp_coords(e) for e in sequences])


    # In[14]:


    np.array(sequences).shape


    # In[15]:


    sequences[117,:,13]


    # In[16]:


    np.array(labels).shape


    # In[17]:


    X = np.array(sequences)


    # In[18]:


    X.shape


    # In[19]:
```

```
y = to_categorical(labels).astype(int)


# In[20]:


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1)


# In[21]:


y_test.shape


# # 4.

# In[22]:


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import TensorBoard


# In[23]:


log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)


# In[38]:


model = Sequential()
model.add(LSTM(64,           return_sequences=True,           activation='relu',
input_shape=(40,126)))
model.add(LSTM(128, return_sequences=True, activation='relu'))
model.add(LSTM(64, return_sequences=False, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))


# In[39]:


model.compile(optimizer='Adam',                loss='categorical_crossentropy',
metrics=['categorical_accuracy'])
```

```
# In[42]:


model.summary()


# In[27]:


model.fit(X_train, y_train, epochs=100, callbacks=[tb_callback])


# In[28]:


model.summary()


# # 5.
# In[29]:


res = model.predict(X_test)


# In[30]:


actions[np.argmax(res[4])]


# In[31]:


actions[np.argmax(y_test[4])]


# # 6.
# In[33]:


model.save('models/gesture.h5')


# In[32]:


del model
```

```
# In[24]:


model.load_weights('models/action.h5')


# # 7.

# In[32]:


from sklearn.metrics import multilabel_confusion_matrix, accuracy_score,
recall_score


# In[33]:


yhat = model.predict(X_test)


# In[34]:


ytrue = np.argmax(y_test, axis=1).tolist()
yhat = np.argmax(yhat, axis=1).tolist()


# In[35]:


multilabel_confusion_matrix(ytrue, yhat)


# In[36]:


accuracy_score(ytrue, yhat)


# # 8.

# In[39]:


from scipy import stats


# In[40]:
```

```
   colors = [(245,117,16), (117,245,16), (16,117,245), (16,217,245), (116,117,245),
(116,217,245)]
   def prob_viz(res, actions, input_frame, colors):
       output_frame = input_frame.copy()
       for num, prob in enumerate(res):
           cv2.rectangle(output_frame,  (0,60+num*40),  (int(prob*100),  90+num*40),
colors[num], -1)
           cv2.putText(output_frame,       actions[num],       (0,       85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)

       return output_frame


   # In[41]:


   sequence = []
   sentence = []
   predictions = []
   threshold = 0.5

   cap = cv2.VideoCapture(0)
   with                              mp_holistic.Hands(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as holistic:
       while cap.isOpened():
           ret, frame = cap.read()

           image, results = mediapipe_detection(frame, holistic)
           print(results)

           draw_landmarks(image, results)

           keypoints = extract_keypoints(results)
           sequence.append(keypoints)
           sequence = sequence[-40:]

           if len(sequence) == 40:
               res      =      model.predict(np.expand_dims(interp_coords(sequence),
axis=0))[0]
               print(actions[np.argmax(res)])
               predictions.append(np.argmax(res))

               if np.unique(predictions[-10:])[0]==np.argmax(res):
                   if res[np.argmax(res)] > threshold:

                       if len(sentence) > 0:
                           if actions[np.argmax(res)] != sentence[-1]:
                               sentence.append(actions[np.argmax(res)])
                       else:
```

```
                        sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                    cv2.FONT_HERSHEY_SIMPLEX,   1,   (255,   255,   255),   2,
cv2.LINE_AA)

        cv2.imshow('OpenCV Feed', image)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Исходный код основного приложения

detection.py

```
import cv2

import numpy as np

import mediapipe as mp

from keras.models import Sequential

from keras.layers import LSTM, Dense

from keras.callbacks import TensorBoard

from keras.models import load_model

import pyautogui as pag


actions = np.array([ 'up', 'down', 'left', 'right', 'ok', 'back'])

key_actions = {'up': 'up',

               'down': 'down',

               'left': 'left',

               'right': 'right',

               'ok': 'enter',

               'back': 'backspace'}


mp_holistic = mp.solutions.hands

mp_drawing = mp.solutions.drawing_utils

mp_drawing_styles = mp.solutions.drawing_styles


model = load_model('models/action.h5')
```

```
# model = Sequential()

# model.add(LSTM(64, return_sequences=True, activation='relu',
input_shape=(40,126)))

# model.add(LSTM(128, return_sequences=True, activation='relu'))

# model.add(LSTM(64, return_sequences=False, activation='relu'))

# model.add(Dense(64, activation='relu'))

# model.add(Dense(32, activation='relu'))

# model.add(Dense(actions.shape[0], activation='softmax'))



# model.compile(optimizer='Adam', loss='categorical_crossentropy',
metrics=['categorical_accuracy'])



# model.load_weights('action.h5')



def interp_coords(x):

    coords = []

    for num,i in enumerate(x):

        if np.count_nonzero(i) != 0:

            coords.append([i,num])

    if not coords:

        return x

    result = []

    for i in range(126):

        result.append(np.interp(range(40),[e[1] for e in coords],[e[0][i] for e in
coords]))

    return np.array(result).transpose()
```

```
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image.flags.writeable = False
    results = model.process(image)
    image.flags.writeable = True
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
    return image, results


def draw_landmarks(image, results):
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_holistic.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())


def extract_keypoints(results):
    try:
        for idx, handLms in enumerate(results.multi_hand_landmarks):
            lh = np.array([[res.x, res.y, res.z] for res in
                           results.multi_hand_landmarks[idx].landmark]).flatten() \
                if results.multi_handedness[idx].classification[0].label == 'Left'
else np.zeros(21 * 3)
            rh = np.array([[res.x, res.y, res.z] for res in
```

```
                        results.multi_hand_landmarks[idx].landmark]).flatten() \

                if results.multi_handedness[idx].classification[0].label == 'Right'
else np.zeros(21 * 3)

        return np.concatenate([lh, rh])

    except:

        return np.concatenate([np.zeros(21 * 3), np.zeros(21 * 3)])



colors = [(245,117,16), (117,245,16), (16,117,245), (16,217,245), (116,117,245),
(116,217,245)]

def prob_viz(res, actions, input_frame, colors):

    output_frame = input_frame.copy()

    for num, prob in enumerate(res):

        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40),
colors[num], -1)

        cv2.putText(output_frame, actions[num], (0, 85+num*40),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)



    return output_frame




sequence = []

sentence = []

predictions = []

threshold = 0.5

prev_len_sentence = 0



cap = cv2.VideoCapture(0)
```

```
with mp_holistic.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5)
as holistic:

    while cap.isOpened():

        ret, frame = cap.read()



        image, results = mediapipe_detection(frame, holistic)



        draw_landmarks(image, results)



        keypoints = extract_keypoints(results)

        sequence.append(keypoints)

        sequence = sequence[-40:]



        if len(sequence) == 40:

            res = model.predict(np.expand_dims(interp_coords(sequence), axis=0))[0]

            print(actions[np.argmax(res)])

            predictions.append(np.argmax(res))



            if np.unique(predictions[-10:])[0]==np.argmax(res):

                if res[np.argmax(res)] > threshold:



                    if len(sentence) != prev_len_sentence:

                        pag.press(key_actions[actions[np.argmax(res)]])

                        prev_len_sentence = len(sentence)

                    if len(sentence) > 0:

                        if actions[np.argmax(res)] != sentence[-1]:
```

```
                    sentence.append(actions[np.argmax(res)])
                else:
                    sentence.append(actions[np.argmax(res)])

        if len(sentence) > 5:
            sentence = sentence[-5:]

        image = prob_viz(res, actions, image, colors)

    cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
    cv2.putText(image, ' '.join(sentence), (3,30),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

    cv2.imshow('Detector', image)

    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Плакаты к дипломному проекту

ДИПЛОМНЫЙ ПРОЕКТ

РЕАЛИЗАЦИЯ СИСТЕМЫ HMI ДЛЯ БЕСКОНТАКТНОГО УПРАВЛЕНИЯ ОБОРУДОВАНИЕМ НА ОСНОВЕ ЖЕСТОВЫХ КОМАНД

Разработал: Ступакевич М.Р. СДП-ИИТ-171
Руководитель: Ассанович Б.А. Доцент

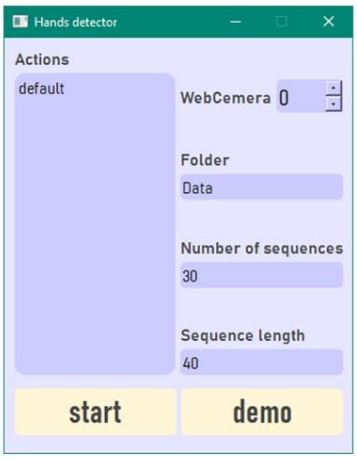
Цель проекта:

Разработка программного HMI для распознавания жестов в видеопотоке и выполнением команд компьютера, согласно распознанному жесту.

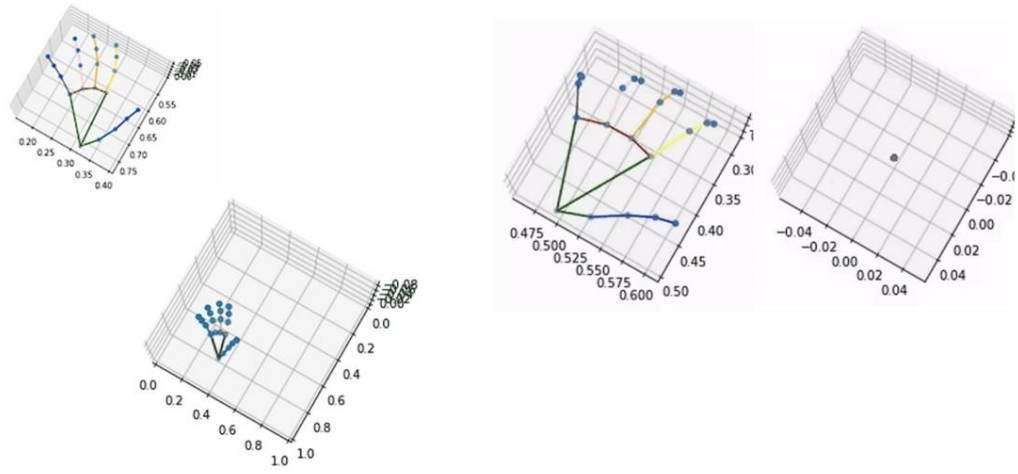
Поставленные задачи:

1. Разработать инструмент сбора данных для обучения нейронной сети.
2. Собрать данные.
3. Провести анализ и обработку собранных данных.
4. Обучить рекуррентную нейронную сеть LSTM.
5. Разработать приложение системы HMI, идентифицирующее жестовые команды в видеопотоке.

Приложение сбора данных



Обработка собранных данных



Структура нейронной сети

Layer (type)	Output Shape	
lstm_3 (LSTM)	(None, 40, 64)	Вход – 40x126
lstm_4 (LSTM)	(None, 40, 128)	ReLU
lstm_5 (LSTM)	(None, 64)	ReLU
dense_3 (Dense)	(None, 64)	ReLU
dense_4 (Dense)	(None, 32)	ReLU
dense_5 (Dense)	(None, 6)	Softmax Выход

40x21x3 (5040) => 6

Интерфейс приложения системы HMI