МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет им. Янки Купалы»

Физико-технический факультет

|  |  |
| --- | --- |
| Кафедра информационных  систем и технологий | ДОПУЩЕН К ЗАЩИТЕ  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.  Зав. Кафедрой информационных  систем и технологий  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ю.Р.Бейтюк |

Специальность: I-38.02.01 Информационно-измерительная техника

КУРСОВОЙ ПРОЕКТ

Тема: Аппаратно-программный интерфейс, реализующий сбор данных для системы управления на основе жестовых команд

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

|  |  |  |
| --- | --- | --- |
| Разработал: |  | Ступакевич М.Р. |
|  | (подпись, дата) |  |
| Руководитель проекта: |  | Ассанович Б.А. |
|  | (подпись, дата) |  |

2021

Содержание

[ВВЕДЕНИЕ 3](#_Toc90238997)

[1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc90238998)

[1.1 Определение набора жестовых команд 5](#_Toc90238999)

[1.2 Google MediaPipe Hands 7](#_Toc90239000)

[1.3 Описание полученных данных 9](#_Toc90239001)

[2. ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc90239002)

[2.1 Структурная схема работы приложения 11](#_Toc90239003)

[2.2 Используемое программное обеспечение 11](#_Toc90239004)

[2.3 Блок-схема и описание работы алгоритма 14](#_Toc90239005)

[2.4 Реализация алгоритма в приложении 20](#_Toc90239006)

[2.5 Визуализация и обработка полученных данных 22](#_Toc90239007)

[ЗАКЛЮЧЕНИЕ 23](#_Toc90239008)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 24](#_Toc90239009)

[ПРИЛОЖЕНИЕ 25](#_Toc90239010)

# 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.

У Google есть своя открытая библиотека Google MediaPipe Hands, которая используется в разработке этого курсового проекта.

Google MediaPipe Hands - это высококачественное решение для отслеживания рук и пальцев. Он использует машинное обучение (ML) для определения 21 3D-ориентира руки всего по одному кадру.

Целью данного курсового проекта является разработка программного решения для сбора данных ключевых точек кистей рук для дальнейшего использования в обучении нейронной сети LSTM, определяющей жестовые команды на видеопотоке.

Для разработки проекта необходимо решить следующие задачи:

1. Определить набор жестовых команд, данные которых будут получены и в дальнейшем использованы для обучения нейронной LSTM сети.
2. Разработать десктопное приложение, собирающее данные и сохраняющее их в удобном для использования формате.
3. Собрать данные для определённых ранее жестовых команд, в количестве достаточном для обучения нейронной сети.
4. Проанализировать полученные данные и произвести их обработку.

# 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## **1.1 Определение набора жестовых команд**

В настоящее время есть 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 данный момент, 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. Кстати, алгоритмы SberBox Top, основаны и обучены на данных, также полученных с помощью Google MediaPipe Hands.



Рисунок – 1.1 Жесты SberBox

Для выполнения задачи формирования 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):

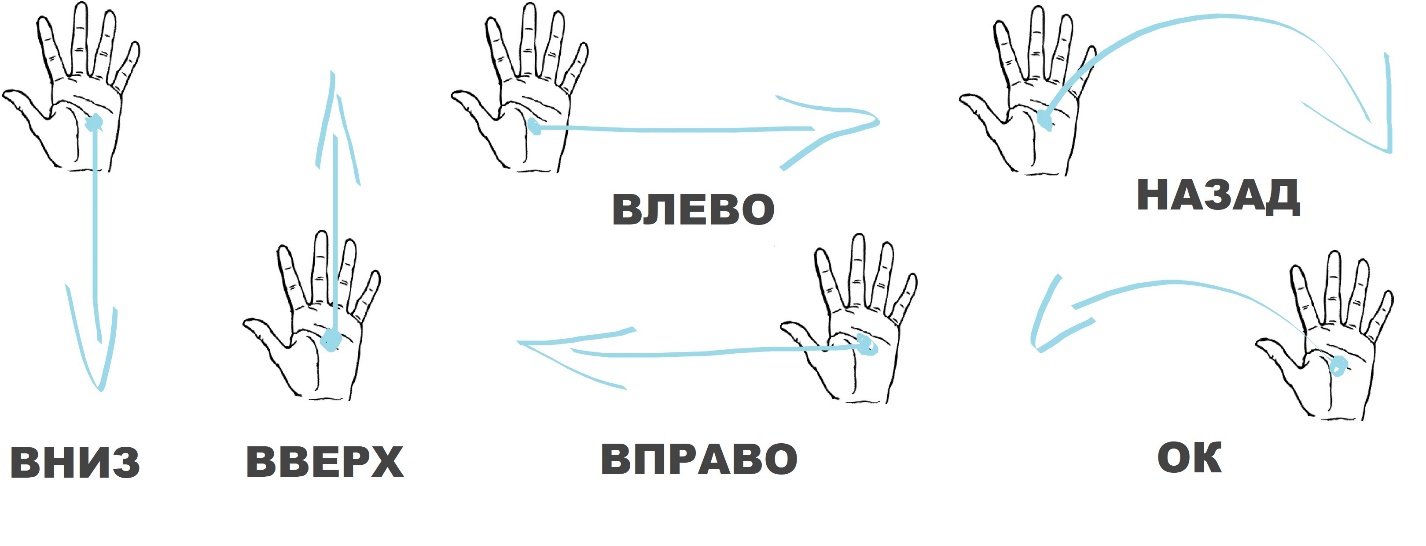


Рисунок – 1.2 Жесты управления

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 – 1.3 Пример синхрожеста

## **1.2 Google MediaPipe Hands**

MediaPipe Hands [1] - 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.

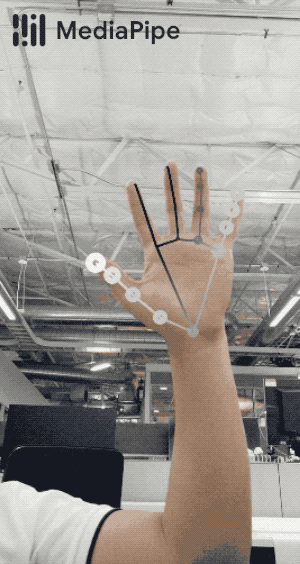


Рисунок – 1.4 Пример работы MediaPipe Hands

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.

## **1.3 Описание полученных данных**

Для последующего обучения рекуррентной нейронной сети LSTM [2] принято решение собрать для определённых ранее 6 жестовых команд по 30 записей каждого жеста. Каждая запись жестовой команды состоит из 40 кадров. Из практического опыта определено, что такого количества записей достаточно для классификации рекуррентной нейронной сети.

Таким образом, с помощью приложения, разработанного в этом курсовом проекте собраны данные. Полученные 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.

Обрабатывать, визуализировать и использовать в обучении рекуррентной нейронной сети LSTM было принято только одну правую руку с целью экономии вычислительных ресурсов, т.к. наличие двух рук увеличивает количество фич (признаков) 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):

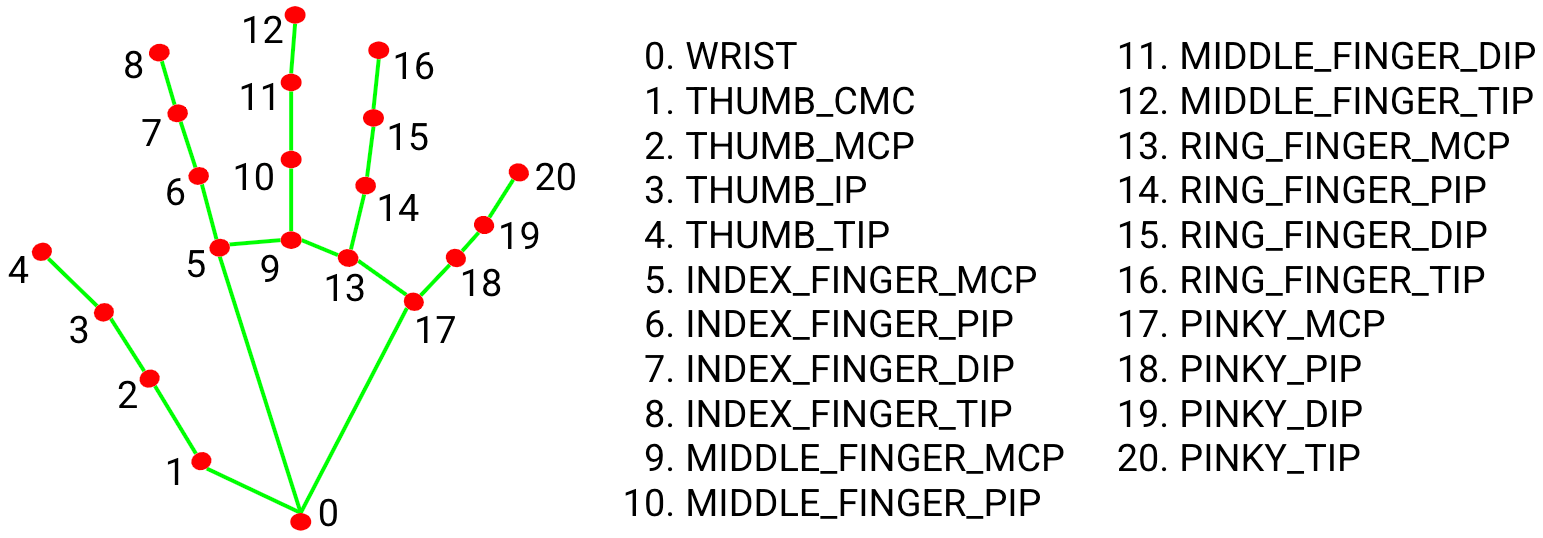


Рисунок – 1.4 Нумерация ключевых точек

# 2. ПРАКТИЧЕСКАЯ ЧАСТЬ

## **2.1 Структурная схема работы приложения**

Приложение, выполняющее поставленную цель должна на входе получить видеопоток и вывести результаты распознавания ключевых точек кистей рук на выходе. Для этого в приложении должны быть реализованы следующие функции (Рисунок – 2.1):

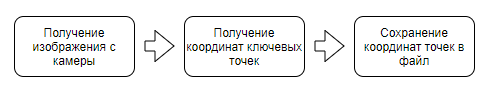


Рисунок - 2.1 Структурная схема системы

## **2.2 Используемое программное обеспечение**

Решение разрабатывается на языке Python и дополнительных библиотек для работы с данными и изображениями с использованием стандартного интерпретатора Python и среды PyCharm от JetBrains

Используемые Python библиотеки:

OS ­­­­­­­— стандартная библиотека Python, позволяющая работать с операционной системой. В программе использовался метод path.join, создающий относительный путь к файлу независимо от типа операционной системы. Это необходимо для пути сохранения файлов. И makedirs для создания директорий по указанному пути для сохранения в них полученных файлов ключевых точек кистей руки.

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами. Использовались такие методы как: array для создания массива, concatenate для конкатенации нескольких массивов данных, zeros для создания массива указанной формы и типа, заполненного нулями, save для сохранения данных в файлы, load для загрузки данных из файлов, shape для определения размерности массива данных и interp для линейной интерполяции данных на этапе обработки полученных данных.

Matplotlib — это мощная бибилиотека двумерной графики для Python, с помощью которой можно создавать высококачественные рисунки различных форматов. В программе использование данный библиотеки ограничивалось применение для визуализации и анализа полученных данных на этапе обработки. Использовались методы plot и show для создания и отображения трёхмерных графиков, представляющих собой эвклидово пространство с расположенными в нём ключевыми точками кисти руки. Также, для установки «точки зрения» и масштабирования видимой области, использовались методы view\_init и xlim/ylim.

Time - модуль для работы со временем в Python. Использовался метод time для получения времени с начала эпохи, выраженное в секундах. Данная операция необходима для вычисления количества обрабатываемых кадров в секунду, т.н. FPS.

Cv2 – это open source библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений. Ипользовались: VideoCapture для получения объекта выбранной камеры, read для получения изображения с камеры, flip для отзеркаливания полученного изображения, putText для помещения текста на изображения (использовалось для отображения количества кадров в секунду и информации о сборе данных: тип жеста, номер записи, времени до начала записи), waitKey для ожидания , namedWindow для создания окна, демонстрирующего изображение, imshow для отображения изображения, release для окончания работы получения изображений и destroyAllWindows для закрытия окон отображения. Также использовались встроенные шрифты, цвета и цветовые схемы.

Mediapipe ­­­­- фреймворк для запуска пайплайнов (предобработка данных, запуск (inference) модели, а также постобработка результатов модели) машинного обучения, позволяющий решить описанные выше проблемы и упростить написание кроссплатформенного кода для запуска моделей. В коде использовались Hands для получения ключевых точек руки с изображения, с разными параметрами (минимальная вероятность детектирования, статический режим – режим детектирования ключевых точек по конкретному изображению, независимо от предыдущих результатов).

## **2.3 Блок-схема и описание работы алгоритма**

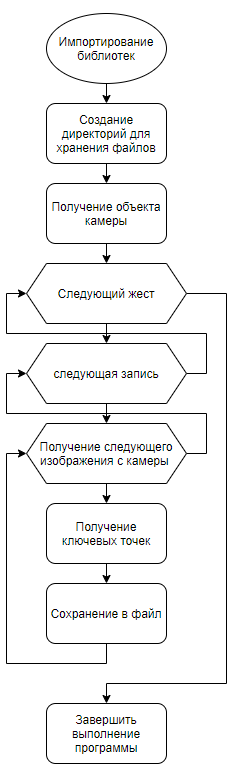


Рисунок - 2.2 Блок-схема основной задачи приложения

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.

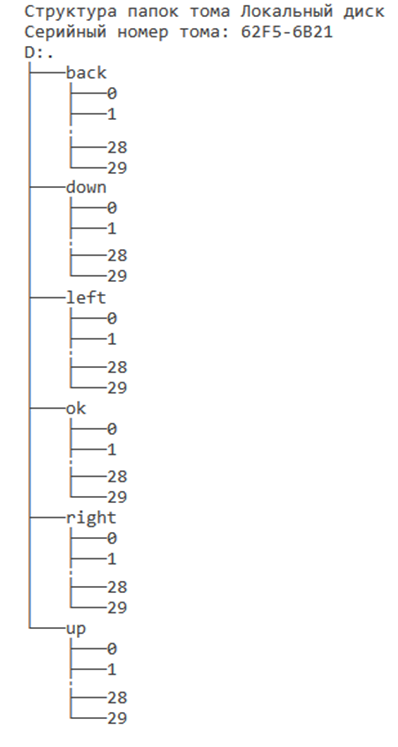


Рисунок - 2.2 Пример структуры папок

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.

## **2.4 Реализация алгоритма в приложении**

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

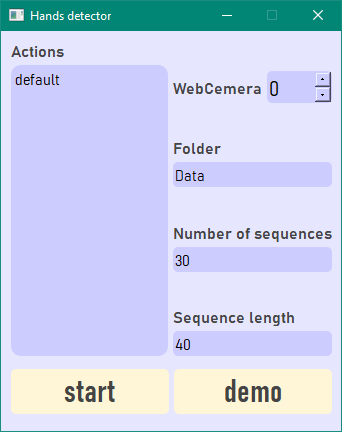


Рисунок - 2.3 Интерфейс приложения

С помощью библиотеки PyQt5 [3] 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:

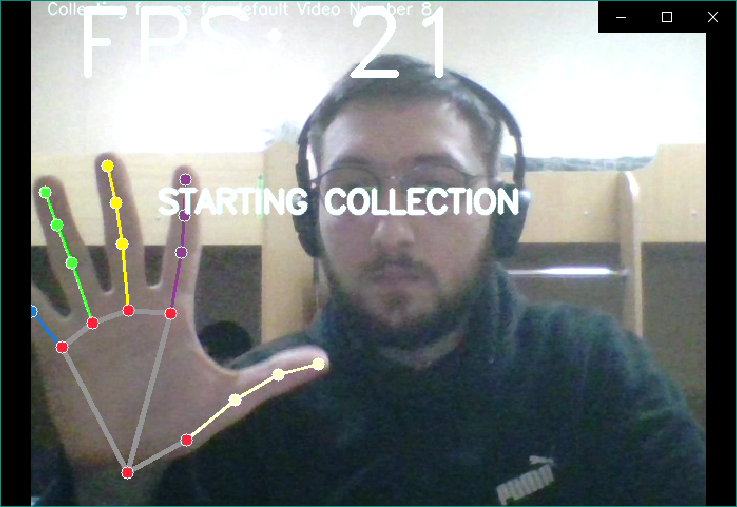


Рисунок - 2.3 Окно сбора данных приложения

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

## **2.5 Визуализация и обработка полученных данных**

Собранные данные выглядят 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. ])]

С помощью библиотеки MatPlotLib [4] данные визуализированы и представлены следующим образом:

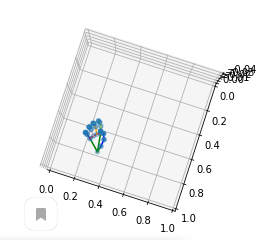
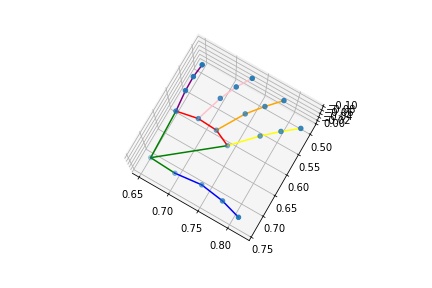


Рисунок - 2.4 Пример визуализации данных

Произведя визуализацию можно наблюдать отсутствие данных в некоторых кадрах, т.н. пробелы. Таким образом для заполнения пробелов необходимо использовать интерполяцию, в частности линейную, реализованную с помощью функции interp библиотеки Numpy.

# ЗАКЛЮЧЕНИЕ

Для достижения поставленной цели были выполнены следующие задачи:

1. Определен набор жестовых команд, данные которых будут получены и в дальнейшем использованы для обучения нейронной LSTM сети.
2. Разработано приложение, собирающее данные и сохраняющее их в удобном для использования формате.
3. Собраны данные для определённых ранее жестовых команд, в количестве достаточном для обучения нейронной сети.
4. Проанализированы полученные данные и произведена их обработку.

Таким образом в ходе выполнения курсового проекта были определены жестовые команды управления и удалось реализовать алгоритм сбора ключевых точек кистей рук.

С помощью приложения собраны данные, состоящие из 6 жестов по 30 записей, по 40 кадров. В сумме получилось 7200 .npy файлов. Эти данные визуализированы, и после визуального анализа определены т.н. пробелы, которые были устранены с помощью линейной интерполяции. Собранные данные будут использованы в дальнейшем для обучения рекуррентной нейронной LSTM [2] сети, определяющей жестовые команды на видеопотоке.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Google MediaPipe Hands [Электронный ресурс]. – Режим доступа: https://google.github.io/mediapipe/solutions/hands.html - Дата доступа: 16.11.2021

2. LSTM [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/wunderfund/blog/331310/ - Дата доступа: 26.04.2021

3. PyQt [Электронный ресурс]. – Режим доступа: https://habr.com/ru/sandbox/153546/ - Дата доступа 26.11.2021

4. MatPlotLib [Электронный ресурс]. – Режим доступа: https://matplotlib.org/ - Дата доступа 16.11.2021

# ПРИЛОЖЕНИЕ

Приложение А - Программный код, выполняющий основные функции

**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**

Приложение Б - Программный код интерфейса приложения:

**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"**))

Приложение В – Программный код приложения:

**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 *# Holistic model*mp\_d = mp.solutions.drawing\_utils *# Drawing utilities*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\_())

Приложение Г – Интерфейс приложения:

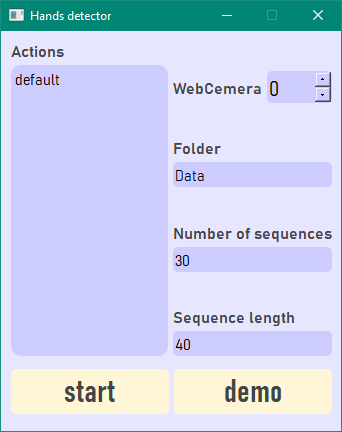


Рисунок – П.4.1

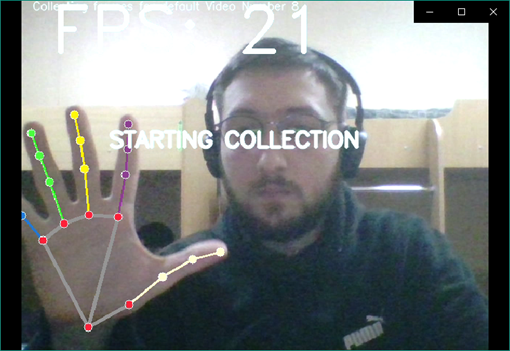


Рисунок – П.4.2

Приложение Д – Структурная схема алгоритма:

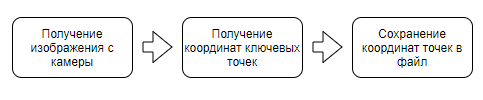


Рисунок – П.5.1

Приложение Е - Блок-схемы алгоритма

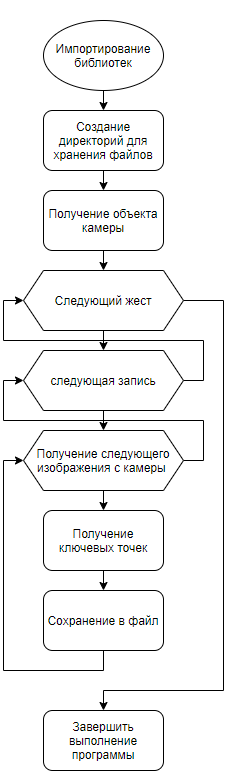


Рисунок – П.6.1

Приложение Ж – Пример структуры папок

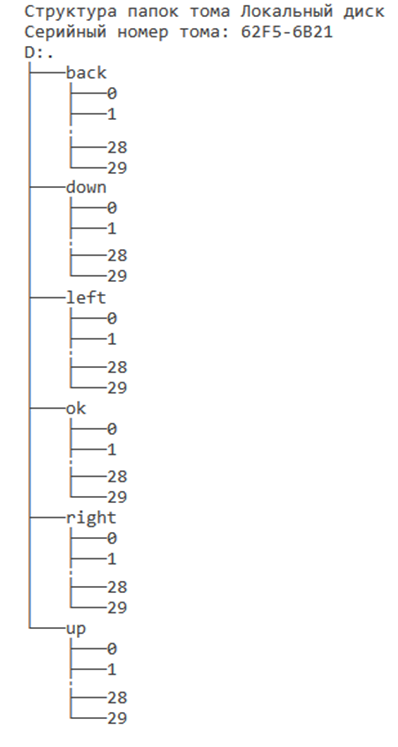


Рисунок – П.7.1

Приложение З – Пример содержания файлов

[array([ 0.42938599, 0.56767941, 0. , 0.46548966, 0.5553869 ,

-0.00987106, 0.49247044, 0.53347236, -0.01593102, 0.51416409,

0.516195 , -0.02286603, 0.53283322, 0.50564444, -0.03002752,

0.45469558, 0.45870686, -0.00292747, 0.46471769, 0.41450295,

-0.01102093, 0.46984217, 0.38775542, -0.01809588, 0.47449937,

0.36475754, -0.02303408, 0.43543926, 0.45218444, -0.0070549 ,

0.43921083, 0.40121913, -0.01362257, 0.44438624, 0.36989373,

-0.02065281, 0.44956419, 0.34298426, -0.02635392, 0.41795516,

0.45479655, -0.01353704, 0.41675597, 0.40510413, -0.02168544,

0.41969445, 0.37392634, -0.02933038, 0.42373306, 0.34772027,

-0.0348063 , 0.40187961, 0.4651227 , -0.02174575, 0.39501756,

0.42526639, -0.02981236, 0.39430583, 0.39780676, -0.03549252,

0.39652619, 0.37338078, -0.03956991, 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

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,

0.46820682, -0.01836461, 0.49301189, 0.44496268, -0.02450593,

0.43159527, 0.45567632, -0.00351434, 0.43321002, 0.4080134 ,

-0.01151449, 0.43276775, 0.37888885, -0.01818659, 0.43125865,

0.35360682, -0.02351789, 0.41231561, 0.45928288, -0.00930303,

0.40633932, 0.41122344, -0.01611581, 0.40113413, 0.38204661,

-0.02225555, 0.394824 , 0.35786849, -0.02785919, 0.39534357,

0.47250044, -0.0171991 , 0.38383883, 0.43061465, -0.02711632,

0.37708384, 0.40310526, -0.03500047, 0.37158644, 0.3802363 ,

-0.04080683, 0.38017702, 0.49374044, -0.02638531, 0.35859042,

0.47116315, -0.0362888 , 0.34428221, 0.45572633, -0.04210992,

0.3324005 , 0.44171456, -0.046393 , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. , 0. , 0. , 0. , 0. ,

0. ])]