Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

КУРСОВОЙ ПРОЕКТ

По курсу **Технологии компьютерного зрения**

по направлению 09.03.01 − Информатика и вычислительная техника

Образовательная программа (профиль) «Системная и программная инженерия»

ТЕМА

«Распознавание жестов»

Студент: _____ /Обухов Алексей Сергееивч, 221-3210/

Проверил: _____ /зав. кафедры ИКТ Пухова. Е.А/

Москва, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)

УТВЕРЖДАЮ
заведующая кафедрой
«Инфокогнитивные технологии»
_____ / Е. А. Пухова /
«___» _____ 2025 г.

ЗАДАНИЕ
на выполнение курсовой работы (проекта)
Обухову Алексею Сергеевичу
(ФИО обучающегося)

обучающемуся группы _221-3210,
направления подготовки 09.03.01 «Информатика и вычислительная техника»
по дисциплине «Технологии компьютерного зрения»
на тему «Распознавание жестов»
1. Исходные данные к работе (проекту): информационные ресурсы в сети интернет, научные публикации в открытой печати.
2. Содержание задания по курсовой работе (проекту) – перечень вопросов, подлежащих разработке:

Разрабатываемый вопрос	Объем от всего задания, %	Срок выполнения	Примечание
Раздел 1. Анализ предметной области	30	01.04.2025	
Задача 1.1. Определение проблем	10	20.03.2025	
Задача 1.2. Поиск решений определенных проблем	20	01.04.2025	
Раздел 2. Разработка программного решения	55	14.06.2025	
Задача 2.1. Выбор технологии для разработки программного решения	10	15.04.2025	
Задача 2.2. Определение структуры программного решения	15	30.04.2025	
Задача 2.3. Реализация программного решения	20	10.06.2025	
Задача 2.4. Тестирование разработанное программное решение	10	14.06.2025	
Раздел 3. Оформление итогов работы	15	18.06.2025	
Задача 3.1. Оформление пояснительной записки	15	18.06.2025	

Руководитель курсовой работы (проекта): Заведующий кафедрой «Инфокогнитивные технологии»

«13» марта 2025 г. _____ (подпись) Е.А. Пухова (И. О. Фамилия)

Дата выдачи задания « 13 » марта 2025 г.
Дата сдачи выполненной работы (проекта) «___» _____ 2025 г.

Задание принял к исполнению

«13» марта 2025 г. _____ (подпись) А. С. Обухов (И. О. Фамилия)

ЛИСТ ЗАМЕЧАНИЙ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время распознавание жестов используется виртуальными ассистентами, в сурдопереводе, в приложениях с дополненной реальностью, а также в развлекательных сервисах. Широкое применения данной технологии является следствием удобства взаимодействия как с людьми, так и с программным обеспечением при помощи жестов.

Использование жестов вместо речи позволяет общаться людям, которые не способны общаться при помощи речи или лишены слуха, либо в шумных местах, в которых уровень шума превышает громкость человеческой речи. Жесты также позволяют расширить возможности пользовательского интерфейса, особенно в приложениях с дополненной реальностью. Управление жестами также используется в мультимедийных системах некоторых автомобилей, что позволяет водителю не отвлекаться от дороги, при этом передавая команду мультимедийной системе автомобиля.

Целью данной курсовой работы является закрепление навыков, знаний и умений, полученных в ходе прохождения курса «Технологии компьютерного зрения», а также создание приложения, способного определять жесты рук на изображениях, поступающих с камеры в реальном времени.

Ниже определены задачи, которые необходимо выполнить для достижения цели данной курсовой работы:

1. Провести анализ предметной области;
2. Определить возможные проблемы и продумать их решения;
3. Разработать приложение для распознавания жестов в реальном времени;
4. Протестировать разработанное приложение;
5. Сделать выводы о проделанной работе.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Цели использования систем распознавания жестов

Основной целью систем распознавания жестов является определение жестов, показанных людьми на изображениях. В дальнейшем сведения о показанном жесте могут быть использованы для взаимодействия с пользовательским интерфейсом, составления субтитров для сурдоперевода, либо для печати символа, соответствующего показанному жесту.

1.2 Аппаратное обеспечение

Поскольку данная технология имеет широкое распространения, спектр используемого аппаратного обеспечения также значительно разнится. Зачастую, используемое аппаратное обеспечение представляет собой камеру, характеристики которой могут значительно различаться, в зависимости от модели.

Все камеры можно условно разделить на три категории: веб-камеры, камеры смартфонов и профессиональные камеры. Каждый из видов камер имеет свои преимущества и недостатки, которые могут сказаться на работе системы распознавания жестов. Преимущества и недостатки каждого из трех видов камер представлены ниже.

1.2.1 Веб-камеры

Зачастую данный вид камер обладает самим низким разрешением, среди трех описанных видов, что усложнить выделение контуров руки и отдельных её частей (фаланги пальцев). Однако низкое разрешение изображение означает, что для его обработки требуется меньше вычислительных ресурсов, что позволяет быстрее производить обработку изображения.

1.2.2 Камеры смартфонов

Камеры смартфонов способны снимать изображения в достаточно высоком разрешении, которое обычно варьируется от 12 до 16 мегапикселей,

что обеспечивает высокую детализацию снимков и позволяет легко выделять контуры на изображении.

Высокая детализация также ведет к тому, что на изображении появляется множество мелких деталей и шумов, которые будут препятствием в распознавании жестов.

Из-за сравнительно высокого разрешение и небольшого размера матрицы пиксели на матрице являются маленькими, в сравнение с пикселями матрицы на веб-камере или профессиональной камере. Маленький размер пикселя ведет к тому, что малое количество света успевает попасть в пиксель за время выдержки, из-за чего изображения получается «темной». Для борьбы с этим обычно используется либо более длительная задержка, которая ведет к повышенному размытию в движении, из-за возможного изменение относительного расположения камеры и руки, либо искусственное повышение яркости изображения, которое ведет к появлению дополнительных шумов.

1.2.3 Профессиональные камеры

Данные камеры зачастую обладают высоким разрешением, обычно от 8 до 25 мегапикселей, и большим размером матрицы. В следствии данных характеристик изображение к данного вида камер является высоко детализированным и при этом с небольшим количеством шумов.

Однако данные камеры составляют лишь небольшую долю из всех используемых камер из-за их высокой стоимости.

1.3 Основные проблемы

Как написано выше, в системах распознавания жестов используется огромное множество различных моделей и видов камер, из-за чего изображения будут значительно различаться по качеству в зависимости от камеры, с которой оно было получено. В следствии этого невозможно будет использовать простой алгоритм для обработки изображений.

Алгоритм должен быть способен предобработать изображение таким образом, чтобы на нем было минимальное количество шумов, при этом

контуры руки могли быть распознаны, чтобы в дальнейшем можно было определить показанный на изображении жест. В виду различия камер, необходимо, чтобы алгоритм мог подобрать наиболее подходящие параметры для предобработки изображения.

Для систем распознавания жестов, которые используются в пользовательских интерфейсах, необходимо быстрое распознавание жестов для плавной и отзывчивой работы интерфейса. Поскольку обработка изображений является ресурсоёмкой задачей, алгоритм предобработки изображений и распознавания жестов необходимо оптимизировать.

Большинство пользовательских интерфейсов обновляют свое состояние 60 раз в секунду или более. Для того, чтобы управление при помощи жестов не замедляло общую работу пользовательского интерфейса и не выглядело ступенчатым, необходимо, чтобы система распознавания жестока была способна обрабатывать 60 или более изображений в секунду. Таким образом на обработку одного изображения должно уходить не более 16,67 миллисекунд.

1.4 Возможные решения проблем

1.4.1 Проблема слишком низкой яркости изображений

Изображения, полученные с камеры смартфона или веб-камеры могут иметь низкую яркость, которая будет препятствовать определению контуров руки, а как следствие и распознаванию жестов.

Повысить яркость путем простого умножения яркости пикселей на некоторую константу можно, однако это приведет к повышению уровня на изображении.

Чтобы повысить яркость изображения, минимально увеличив количество присутствующих на изображении шумов, рациональным будет использование градационных фильтров, например кусочно-линейного или степенного преобразований.

При использовании таких фильтров можно повысить яркость изображения, практически не добавив новых шумов, а возможно даже и удалив уже имеющиеся шумы.

1.4.2 Проблема большого количества шумов на изображении

Поскольку большая часть шумов, которые могут находится на изображении, для борьбы с ними достаточно провести частотный анализ на предмет наличия высокочастотных сигналов в частотной спектре изображения. Затем для избавления изображения от шумов можно использовать высокочастотный фильтр, например фильтр Гаусса.

Данный фильтр идеально подходит для устранения высокочастотных шумов. Кроме того, данный фильтр не требует больших вычислительных затрат, а соответственно и время обработки изображений будет минимально. Фильтр также не влияет на общее качество изображения при подборе параметров, в частности размера ядра и интенсивности фильтра.

1.4.3 Проблема высокого разрешения

Хоть высокое разрешение изображения и позволяет легко выделить контуры руки, оно также добавляет большое количество излишних мелких деталей.

Для снижения количества мелких деталей, которые могут помешать выделению контуров рук, можно использовать частотный фильтр, например фильтр Гаусса.

Также высокое разрешение изображения означает большее количество пикселей, которые придется обрабатывать и хранить в оперативной памяти.

Рациональным решением будет снижение разрешения изображения путем его обрезания и масштабирования. Это позволит снизить время обработки кадра, что повысит общую производительность системы без потерь в точности работы алгоритма.

Снижение разрешения также избавит изображение от мелких деталей, которые могут помешать нахождению контуров руки или замедлить этот процесс.

2 РАЗРАБОТКА ПРОГРАММНОГО РЕШЕНИЯ

2.1 Цель и назначение приложения

Целью приложения, разрабатываемого в рамках данной курсовой работы, является анализ изображений, поступающих с камеры в реальном времени и определение положения руки на них.

Данное приложение может быть использовано как часть большей системы, например в качестве модуля распознавания жестов пользователя, сведения о которых могут быть использованы для управления пользовательским интерфейсом.

2.2 Используемые технологии

2.2.1 Язык программирования

Данное приложение будет разрабатываться на языке программирования высокого уровня Python 3, поскольку код, написанный на данном языке, может быть запущен на различных устройствах, вне зависимости от их операционной системы. Кроме того, разработка приложений на Python требует меньше временных затрат чем разработка на других языках, например C++, Java, Go.

Для Python3 также существует множество библиотек, которые дополнительно упростят разработку данного приложения.

2.2.2 NumPy

Данная библиотека используется многими другими библиотеками, в частности библиотеками для работы с изображениями, например OpenCV, PIL.

NumPy упрощает работу с массивами различных размерностей, матрицами и тензорами, что позволит снизить время разработки данного приложения.

В данном приложении NumPy будет выполнять функцию манипулятора данными, т.е. будет основополагающей для других используемых в проекте библиотек.

2.2.3 OpenCV

OpenCV является одной из крупнейших библиотек для обработки изображений. Эта библиотека обладает широчайшим функционалом, от чтения и сохранения изображений в файлы до поиска контуров на изображениях.

Кроме того, код данной библиотеки написан на языке C, что обеспечивает высокую скорость ее работы.

2.2.4 MediaPipe

Данная библиотека от компании Google, разработанная специально для работы с аудио, видео и фото, предоставляет доступ к различным переобученным нейросетевым моделям, которые показывают высокое качество своей работы.

Одна из нейросетевых моделей, предоставляемых данной библиотекой, будет использована в разрабатываемом приложении для определения положения рук на видео, получаемом с видеокамеры.

2.2.5 PyGame

Для реализации и отображения пользовательского интерфейса приложения будет использоваться библиотека PyGame. Хоть данная библиотека и предназначена в первую очередь для создания игр, а её возможности в плане создания графических интерфейсов весьма малы, данная библиотека позволит создать простой графический интерфейс.

2.3 Структура приложения

Приложение будет разработано в соответствие с объектно-ориентированной парадигмой программирования, что позволит модифицировать программу в дальнейшем, а также легко встраивать отдельные её части в другие приложения.

Приложение будет состоять из четырех классов:

1. Window;

2. HandRecognizer;
3. ImagePreprocessor;
4. Button

2.3.1 Класс Window

Данный класс отвечает за отрисовку пользовательского интерфейса приложения и взаимодействие с пользователем. Также при каждом цикле обновления пользовательского интерфейса, данный класс будет получать новый кадр и камеры.

2.3.2 Класс HandRecognizer

Данный класс отвечает за распознавание положения рук на изображении, полученном с камеры и построении тензора, описывающего найденное положение рук.

2.3.3 Класс ImagePreprocessor

Данный класс будет выполнять роль контейнера для функций обработки изображений, таких как фильтр Гаусса.

2.3.4 Класс Button

Этот класс нужен для описания кнопок пользовательского интерфейса и обработки нажатия на них. Данный класс необходим, поскольку PyGame не имеет встроенного класс для кнопок.

2.4 Принцип работы

Принцип работы данного приложения достаточно прост: приложение считывает изображения с камеры в режиме реального, после чего производит их предобработку, после чего производит анализ изображений при помощи предобученной модели.

В анализ изображений входят следующие этапы:

1. Получение изображения с камеры
2. Предобработка изображения;

3. Определение положения рук на изображении;
4. Построение тензора, описывающего положение руки;

Ниже будут подробно описаны все этапы анализа изображений.

2.4.1 Получение изображения с камеры

При помощи библиотеки OpenCV приложение будет получать кадр с видеокамеры, выбранной в качестве камеры по умолчанию

2.4.2 Предобработка изображения

На данном этапе приложение будет масштабировать изображение под оптимальные размеры, если исходное изображение будет слишком велико или мало.

После масштабирования приложение будет применять производить повышение контрастности изображения, путем применения кусочно-линейного фильтра или гистограммной экранизации.

Затем приложение применит фильтр Гаусса к изображению, чтобы снизить количество шумов на нем.

2.4.3 Определение положения руки на изображении

Определение положения руки на изображении будет выполняться путем поиска контура руки. Контуры, в свою очередь, будут определяться по границам цветов на изображении.

2.4.4 Создание тензора, описывающего положение руки

После определения положения руки на изображении, приложение построить тензор, описывающий положение руки на изображении, который затем может быть передан нейронной сети для распознавания жеста или использован для определения перемещения руки.

2.5 Ход разработки

2.5.1 Реализация структуры приложения

На данном этапе были созданы классы, описанные в пункте 2.3, налажено их взаимодействие и инициализация при запуске приложения.

2.5.2 Реализация графического пользовательского интерфейса

При помощи библиотеки PyGame был создан простой графический интерфейс, которые тем не менее отображает всю необходимую информацию, также предоставляет пользователю возможности по настройке предобработки изображений.

Полученный графический интерфейс изображен на рисунке 3.

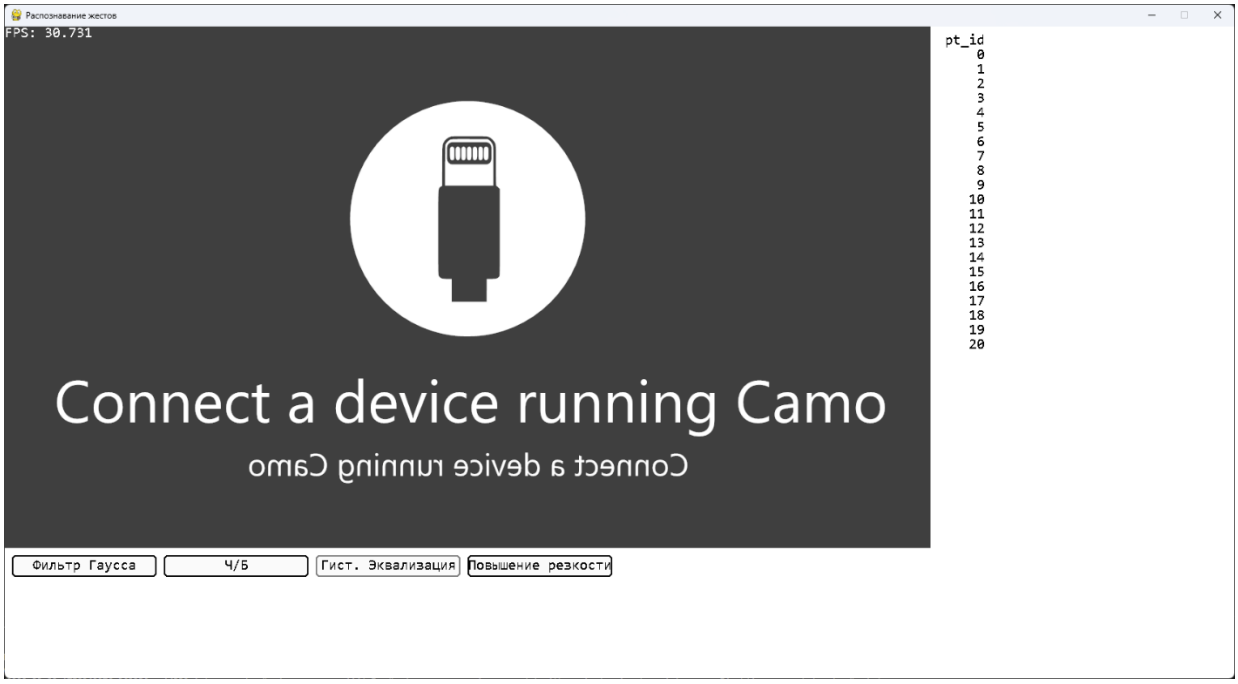

Рисунок 1 — Изображение графического пользовательского интерфейса приложения

2.5.3 Получение изображения с камеры

Для получения изображения с камеры был использован класс VideoCapture из библиотеки OpenCV.

Частота получения кадров с видеокамеры синхронизирована с частотой обновления графического пользовательского интерфейса. Частота получения кадров с камеры отображается в левом верхнем углу приложения.

2.5.4 Предобработка полученных изображений

На данном этапе были реализованы функции класса ImagePreprocessor по предобработке изображений. Параметры для предобработки определяются пользователем.

В листинге 1 показана реализация класса, реализующего предобработку изображений.

Листинг 1 — реализация класса ImagePreprocessor

```
class ImagePreprocessor:
    def grayscale(frame):
        return cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    def gaussian_blur(frame, kernel=(5, 5)):
        return cv2.GaussianBlur(frame, kernel, 0)

    def hist_equalization(frame):
        return cv2.equalizeHist(frame)

    def laplacian_sharpening(frame):
        kernel = np.array([
            [0, -1, 0],
            [-1, 4, -1],
            [0, -1, 0]], dtype=np.float32)
        laplacian = cv2.filter2D(frame, cv2.CV_32F,
kernel).astype(np.uint8)
        sharpened = cv2.addWeighted(frame, 0.9, laplacian, 0.1,
0)

        return sharpened
```

При реализации функций данного класса было решено использовать функции, реализованные в библиотеке OpenCV вместо написания собственных реализаций в виду более высокой скорости работы функций из библиотеке.

2.5.5 Определение жестов рук

На данном этапе был реализован алгоритм передачи кадров предобученной модели и последующее получение результатов её работы.

Также была реализована функция для отрисовки положения руки, обнаруженного моделью, на кадре, который затем будет отображен в пользовательском интерфейсе приложения.

Код функции, отвечающий определение и отрисовку положения рук в кадре показан в листинге 2.

Листинг 2 — функция определения положения рук в кадре

```
def trace_fingers(self, frame, draw=True):
        self.results = self.hands.process(frame)

        if not self.results.multi_hand_landmarks:
            return frame

        for hand in self.results.multi_hand_landmarks:
            if draw:
                self.mp_draw_utils.draw_landmarks(
                    frame, hand, self.hands_mp.HAND_CONNECTIONS)

        return frame
```

2.6 Тестирование разработанного приложения

Теперь, чтобы убедиться в работоспособности разработанного приложения, была выполнена проверка работы приложения в различных условиях и сценариях, которые описаны в разделах ниже.

2.6.1 Одна рука в кадре на простом фоне

В данном тестовом сценарии проверяется, как приложение определить положение руки в кадре с простым фоном. Результат работы приложения показан на рисунке 2.

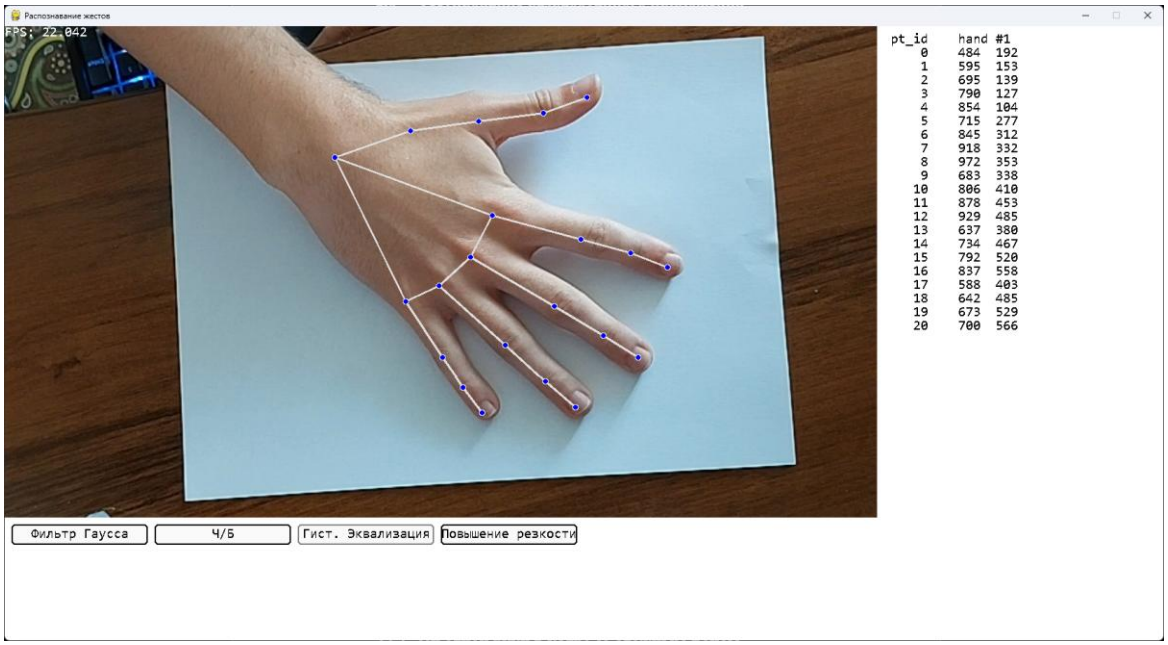

Рисунок 2 — Распознавание одной руки на простом фоне

Как видно на рисунке 2, приложение успешно определило положение руки, а также положение всех пальцев на изображении.

2.6.2 Две руки в кадре на простом фоне

Данный тестовый сценарий схож с предыдущим. Единственным отличием является наличие второй руки в кадре. Результат работы приложения показан на рисунке 3.

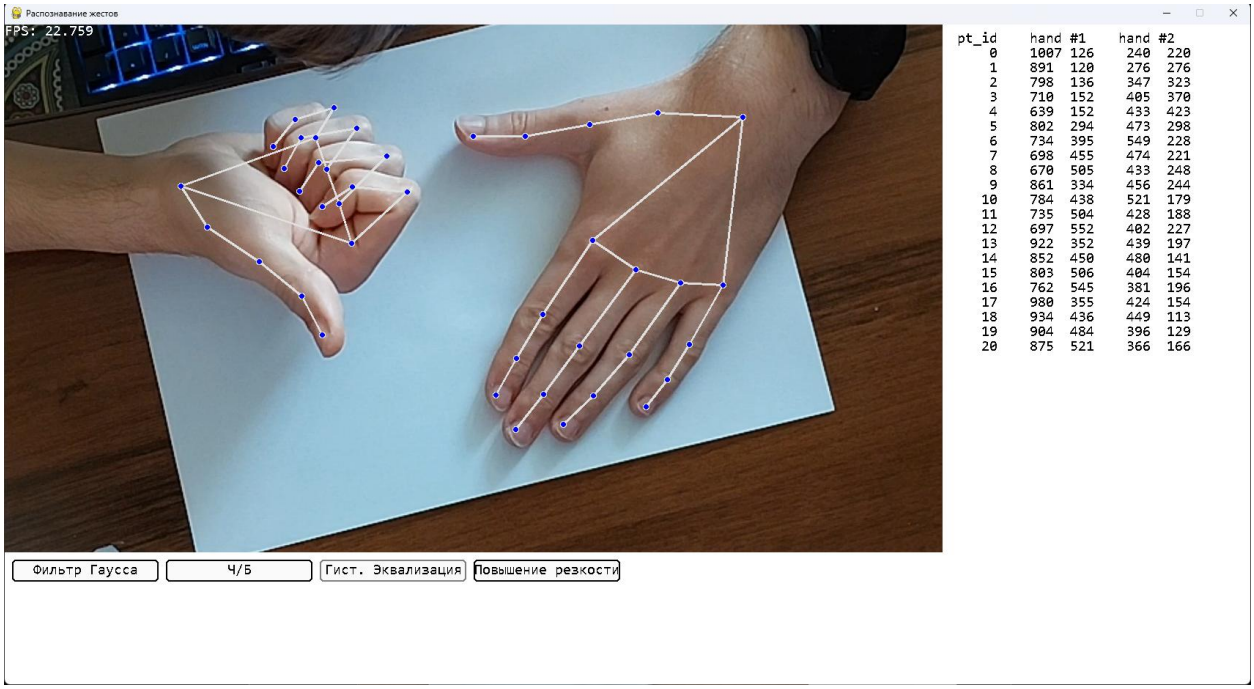

Рисунок 3 — Распознавание двух рук на простом фоне.

2.6.3 Попытка распознания нарисованной руки

В данном тестовом сценарии была попытка ввести предобученную модель в заблуждения путем показа ей нарисованной руки. Результат работы приложения показан на рисунке 4, из которого видно, что модель не распознала нарисованную руку.

Рисунок 4 — Распознавание нарисованной руки

2.6.4 Одна рука в кадре на сложном фоне

Данный тестовый сценарий повторяет сценарий, описанный в пункте 2.6.1, но теперь фон имеет больше деталей, которые могут помешать приложению определить положение руки. Приложение корректно определило положение руки в кадре, что подтверждается рисунком 5.

Рисунок 5 — Распознавание одной руки на сложном фоне

2.6.5 Две руки в кадре на сложном фоне

В Этом тестовом сценарии проверяется корректность распознания двух рук на сложном фоне, с чем приложение успешно справилось, что показано на рисунке 6.

Рисунок 6 — Распознание двух рук на сложном фоне

2.6.6 Ни одной руки в кадре со сложным фоном

В этом тестовом сценарии проверяется, не распознает ли приложение в сложной фоне руки. Как видно из рисунка 7, приложение не опознало в фоне рук.

Рисунок 7 — Работа со сложным фоном без рук

2.6.7 Две взаимодействующие руки на простом фоне

В данном тестовом сценарии проверялось, как приложение отработает, в случае если две руки, показанные в кадре, будут взаимодействовать друг с другом. В данном случае, одна рука взяла большой палец второй руки. Приложение смогло определить обе руки, однако предсказанное положение большого пальца незначительно отличается от реального, что показано на рисунке 8.

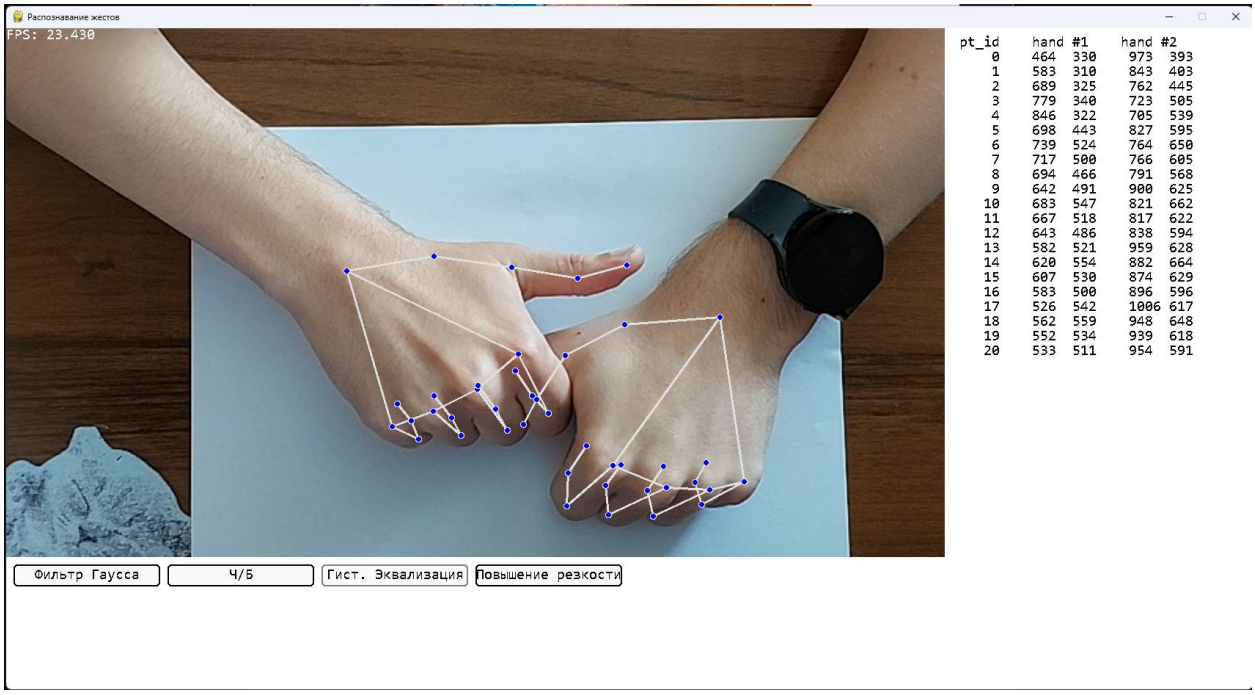

Рисунок 8 — Распознавание взаимодействующих рук

2.6.8 Рука в кадре частично перекрыта посторонним объектом

В данном тестовом сценарии проверялось, сможет ли приложение определить положение руки, если её часть будет закрыта от камеры каким-либо посторонним предметом. В данном случае, посторонним предметом выступил лист бумаги, которые перекрывает примерно 50% руки. Однако даже в таком случае, приложение корректно определило положение видимых камере пальцев и предсказало положение пальцев, которые камере не видны. Результат работы приложения показан на рисунке 9.

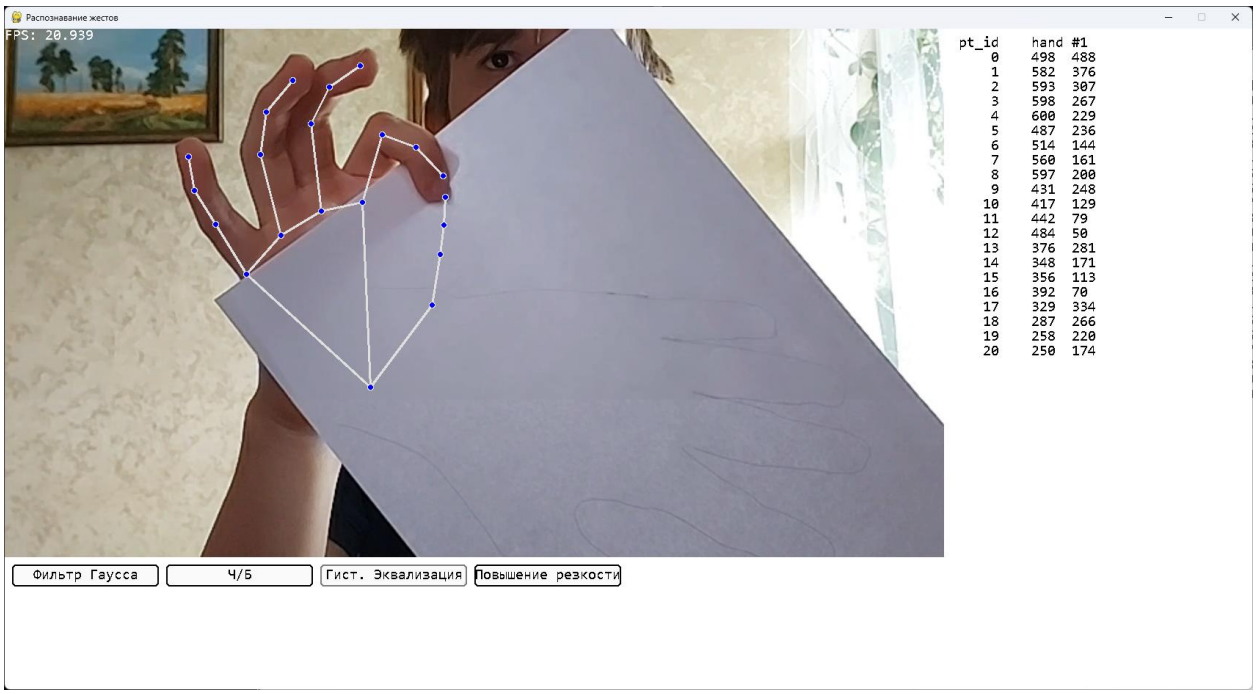

Рисунок 9 — Распознавание руки, частично закрытой от камеры посторонним объектом.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были проанализированы сценарии использования систем распознавания жестов, области их применения, а также проблемы, мешающие корректной работе таких систем.

В целях закрепления навыков и умений, полученных при прохождении курса «Технологии компьютерного зрения», а также в целях решения проанализированных проблем было создано приложение для распознавания положение рук в кадре.

В целом все цели данной курсовой работы были достигнуты, а задачи поставленные в ней – выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по решениям MediaPipe [Электронный ресурс]. — URL: https://ai.google.dev/edge/mediapipe/solutions/guide?hl=ru (дата обращения: 18.06.2025);
2. OpenCV Modules [Электронный ресурс]. — URL: https://docs.opencv.org/4.x/index.html (дата обращения: 18.06.2025);
3. NumPy Documentation [Электронный ресурс]. — URL: https://numpy.org/doc/ (дата обращения 18.06.2025);
4. Владимир И. Павовлович, Раджив Шарма, Томас С. Хуан. Visual interpretation of hand gestures for human-computer interaction: A Review // IEEE Transactions on pattern analysis and machine intelligence. Сер. 19. — 1997. — №7 —. С. 677-695;
5. Мухаммад Захид Икбал, Авраам Г. Кэмпбелл. The emerging Need for touchless interaction technologies // Interactions. Сер 27. — 2020. — С. 51-52;
6. Gesture Recognition [Электронный ресурс]. — URL: https://en.wikipedia.org/wiki/Gesture_recognition#cite_note-16 (дата обращения: 18.06.2025);
7. Размытие по Гауссу [Электронный ресурс]. — URL: https://ru.wikipedia.org/wiki/Размытие_по_Гауссу (дата обращения: 18.06.2025).

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

Листинг А.1 — исходный код приложения

```
import pygame as pg
from pygame.locals import *
from time import time
import cv2
import mediapipe as mp
import numpy as np


WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
GRAY = (125, 125, 125)


class ImagePreprocessor:
    def grayscale(frame):
        return cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)

    def gaussian_blur(frame, kernel=(5, 5)):
        return cv2.GaussianBlur(frame, kernel, 0)

    def hist_equalization(frame):
        return cv2.equalizeHist(frame)

    def laplacian_sharpening(frame):
        kernel = np.array([
            [0, -1, 0],
            [-1, 4, -1],
            [0, -1, 0]], dtype=np.float32)
        laplacian = cv2.filter2D(frame, cv2.CV_32F,
kernel).astype(np.uint8)
        sharpened = cv2.addWeighted(frame, 0.9, laplacian, 0.1,
0)

        return sharpened


class HandRecognizer:
    def __init__(self, mode=False, max_hands: int = 2,
detection_con: float = 0.5, track_con: float = 0.5) -> None:
        self.__mode__ = mode
        self.max_hands = max_hands
        self.__detectionCon__ = detection_con
        self.__trackCon__ = track_con
        self.hands_mp = mp.solutions.hands
        self.hands = self.hands_mp.Hands()
        self.mp_draw_utils = mp.solutions.drawing_utils
        self.tip_ids = [4, 8, 12, 16, 20]
```

```
    def trace_fingers(self, frame, draw=True):
        self.results = self.hands.process(frame)

        if not self.results.multi_hand_landmarks:
            return frame

        for hand in self.results.multi_hand_landmarks:
            if draw:
                self.mp_draw_utils.draw_landmarks(
                    frame, hand, self.hands_mp.HAND_CONNECTIONS)

        return frame

    def find_hand_possition(self, frame, hand_id, draw=True):
        xList = []
        yList = []
        self.landmarks = []

        if self.results.multi_hand_landmarks:
            if hand_id + 1 >
len(self.results.multi_hand_landmarks):
                return None
            hand = self.results.multi_hand_landmarks[hand_id]
            h, w, _ = frame.shape

            for id, lm in enumerate(hand.landmark):
                cx, cy = int(lm.x * w), int(lm.y * h)
                xList.append(cx)
                yList.append(cy)
                self.landmarks.append([id, cx, cy])

                if draw:
                    cv2.circle(frame, (cx, cy), 5, GREEN,
cv2.FILLED)

        return self.landmarks


class Button:
    def __init__(self, text: str, screen, x: int, y: int, w:
int, h: int) -> None:
        self.rect = pg.Rect(x, y, w, h)
        self.text = text
        self.active_color = 45, 45, 250
        self.enabled_color = 250, 250, 250
        self.disabled_color = 250, 250, 250
        self.is_enabled = True
        self.is_active = False
        self.font = pg.font.SysFont("Consolas", 20)
        self.screen = screen

    def draw(self) -> None:
```

```
        if self.is_enabled:
            color = self.enabled_color
            if self.is_active:
                color = self.active_color
        else:
            color = self.disabled_color

        pg.draw.rect(self.screen, color, self.rect,
border_radius=5)
        pg.draw.rect(self.screen, BLACK if self.is_enabled else
GRAY,
                     self.rect, 2, border_radius=5)

        text_surface = self.font.render(
            self.text, False, BLACK if self.active_color else
WHITE)
        text_rect =
text_surface.get_rect(center=self.rect.center)
        self.screen.blit(text_surface, text_rect)

    def is_clicked(self, mouse_pos):
        return self.rect.collidepoint(mouse_pos) and
self.is_enabled

    def set_active(self, active_status: bool) -> None:
        self.is_active = active_status

    def set_enabled(self, enabled_status: bool) -> None:
        self.is_enabled = enabled_status


class Window:
    def __init__(self, hand_detector: HandRecognizer, width: int
= 1700, height: int = 900, framerate: int = 30, video_cap: int =
None) -> None:
        pg.init()
        pg.font.init()
        self.font = pg.font.SysFont("Consolas", 20)
        self.running = True
        self.framerate = framerate
        self.screen = pg.display.set_mode((width, height))
        pg.display.set_caption("Распознавание жестов")
        self.clock = pg.time.Clock()
        self.hand_detector = hand_detector

        self.gausse_btn = Button(
            "Фильтр Гаусса", self.screen, 10, 730, 200, 30)
        self.grayscale_btn = Button("Ч/Б", self.screen, 220,
730, 200, 30)
        self.hist_eq_btn = Button(
            "Гист. Эквализация", self.screen, 430, 730, 200, 30)
        self.hist_eq_btn.set_enabled(False)
        self.unsharp_btn = Button(
```

```
            "Повышение резкости", self.screen, 640, 730, 200,
30)

        if video_cap is None:
            self.video_cap = None
        else:
            self.video_cap = cv2.VideoCapture(video_cap)

            if not self.video_cap.isOpened():
                print("Could not open new video_cap")
                pg.quit()
                exit()

        self.cycle()

    def set_video_cap(self, new_cap_id: int):
        self.video_cap.release()
        self.video_cap = cv2.VideoCapture(new_cap_id)

        if not self.video_cap.isOpened():
            print("Could not open new video_cap")
            pg.quit()
            exit()

    def print_landmarks(self, landmarks_list):
        y_pos = 10
        y_gap = 20
        header = 'pt_id'

        for i in range(len(landmarks_list)):
            header += f'    hand #{i + 1}'

        header_surface = self.font.render(header, False, BLACK)
        self.screen.blit(header_surface, (1300, y_pos))
        y_pos += y_gap

        for pt_id in range(21):
            pt_info = f'{pt_id:>5}'

            for hand_id in range(len(landmarks_list)):
                pt_info += f'
{landmarks_list[hand_id][pt_id][1]:<4}
{landmarks_list[hand_id][pt_id][2]:<3}'

            line_surface = self.font.render(pt_info, False,
BLACK)
            self.screen.blit(line_surface, (1300, y_pos))
            y_pos += y_gap

    def cycle(self):
        ptime = 0
        while self.running:
            for event in pg.event.get():
```

```
                if event.type == QUIT or (event.type == KEYDOWN
and event.key == K_ESCAPE):
                    self.running = False
                    break
                if event.type == pg.MOUSEBUTTONDOWN and
event.button == 1:
                    mouse_pos = pg.mouse.get_pos()
                    if self.gausse_btn.is_clicked(mouse_pos):
                        self.gausse_btn.set_active(
                            not self.gausse_btn.is_active)
                    if self.grayscale_btn.is_clicked(mouse_pos):
                        self.grayscale_btn.set_active(
                            not self.grayscale_btn.is_active)
                        self.hist_eq_btn.set_enabled(
                            self.grayscale_btn.is_active)
                        if self.grayscale_btn.is_active ==
False:
                            self.hist_eq_btn.set_active(False)
                    if self.hist_eq_btn.is_clicked(mouse_pos):
                        self.hist_eq_btn.set_active(
                            not self.hist_eq_btn.is_active)
                    if self.unsharp_btn.is_clicked(mouse_pos):
                        self.unsharp_btn.set_active(
                            not self.unsharp_btn.is_active)

            self.screen.fill(WHITE)
            ret, frame = self.video_cap.read()

            if not ret:
                print("Could not get a frame from a camera")
                break

            rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

            if self.gausse_btn.is_active:
                rgb_frame =
ImagePreprocessor.gaussian_blur(rgb_frame)

            if self.unsharp_btn.is_active:
                rgb_frame =
ImagePreprocessor.laplacian_sharpening(rgb_frame)

            if self.grayscale_btn.is_active:
                rgb_frame =
ImagePreprocessor.grayscale(rgb_frame)

            if self.hist_eq_btn.is_active:
                rgb_frame =
ImagePreprocessor.hist_equalization(rgb_frame)

            if self.grayscale_btn.is_active:
                rgb_frame = cv2.merge([rgb_frame, rgb_frame,
rgb_frame])
```

```
            traced_fingers_frame =
self.hand_detector.trace_fingers(rgb_frame)

            landmarks_list = []
            for hand_id in range(self.hand_detector.max_hands):
                landmarks =
self.hand_detector.find_hand_possition(
                    frame, hand_id)
                if landmarks:
                    landmarks_list.append(landmarks)

            landmarks_info =
self.print_landmarks(landmarks_list)
            landmarks_info_surface = self.font.render(
                landmarks_info, False, BLACK)
            self.screen.blit(landmarks_info_surface, (0, 800))

            img_surface = pg.surfarray.make_surface(
                traced_fingers_frame.swapaxes(0, 1))

            self.screen.blit(img_surface, (0, 0))

            self.gausse_btn.draw()
            self.grayscale_btn.draw()
            self.hist_eq_btn.draw()
            self.unsharp_btn.draw()

            ctime = time()
            fps = 1 / (ctime - ptime)
            fps_surface = self.font.render(f"FPS: {fps:.3f}",
False, WHITE)
            self.screen.blit(fps_surface, (0, 0))

            pg.display.flip()

            ptime = time()
            self.clock.tick(self.framerate)

        self.video_cap.release()
        pg.quit()


if __name__ == "__main__":
    detector = HandRecognizer()
    w = Window(detector, video_cap=0)
```