)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Вычислительные машины, комплексы, системы и сети»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ПРОГРАММНЫЙ МОДУЛЬ ОТСЛЕЖИВАНИЯ РУК ДЛЯ ДОПОЛНЕННОЙ РЕАЛЬНОСТИ

Студент 31.05.2023 (подпись, дата) Кудряшов Ярослав Валерьевич (ФИО полностью)

Руководитель 31.05.2023 (подпись, дата) Митрохин М.А. (фамилия, инициалы)

Нормоконтролёр 31.05.2023 (подпись, дата) Бычков А.С. (фамилия, инициалы)

Рецензент гл. специалист проектного офиса АО «ПКК "Рубин", к.т.н. (должность, место работы) (подпись, дата) Фасеев (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 г. № 14)

Заведующий кафедрой (подпись) Митрохин М.А. (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 13.06.23 № 10)

Секретарь ГЭК (подпись) Гурин Е.И. (фамилия, инициалы)

Пенза 2023

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«29» ноября 2021 г.

ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Кудряшов Ярослав Валерьевич гр. 21ВВВм1 факультета вычислительной техники, направления 09.04.01
2. Тема работы «Программный модуль отслеживания рук для дополненной реальности».

Тема утверждена приказом ПГУ № 924/о от "29" ноября 2021 г.

3. Руководитель магистерской диссертации д.т.н., профессор Митрохин М.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: модуль с поддержкой жестов рук, предназначенный для облегчения взаимодействия с виртуальной доской.

Разработать виртуальную доску, оснащенную компактным набором инструментов и возможностью распознавания жестов рук в режиме реального времени.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие методы отслеживания рук, их точность, надежность и пригодность для различных приложений дополненной реальности. Кроме того,

следует изучить проблемы и ограничения, связанные с отслеживанием движений рук в средах дополненной реальности, включая окклюзию, различные условия освещения и различные позы рук.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области отслеживания и распознавания рук в реальном времени	13%	29.11.2021-10.12.2021	
2.	Анализ предметной области распознавания жестов рук в реальном времени	13%	11.12.2021-28.02.2022	
3.	Проектирование структуры разрабатываемого модуля	10%	01.03.2022-01.05.2022	
4.	Разработка всех компонентов модуля отслеживания рук для дополненной реальности	39%	02.05.2022-27.01.2022	
5.	Отладка модуля отслеживания рук для дополненной реальности	15%	28.12.2022-29.03.2023	
6.	Оформление выпускной работы	10%	30.03.2023-31.05.2023	

Дата выдачи задания "29" ноября 2021

Руководитель магистерской диссертации _____ Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент _____ Кудряшов Я.В.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)
16.06.2023

РЕФЕРАТ

Магистерская работа содержит 83 л., 30 рис., 8 источн., 2 прил.

КОМПЬЮТЕРНОЕ ЗРЕНИЕ, БИБЛИОТЕКА MEDIAPIPE, ОБНАРУЖЕНИЕ, ОТСЛЕЖИВАНИЕ И РАСПОЗНАВАНИЕ РУК, РАСПОЗНАВАНИЕ ЖЕСТОВ РУК В РЕАЛЬНОМ ВРЕМЕНИ.

Цель работы - создать модуль, который сможет взаимодействовать с виртуальной доской при помощи жестов рук.

Объектом разработки является модуль отслеживания рук для дополненной реальности. Данный модуль облегчает взаимодействие с виртуальной доской, устраняя необходимость в клавиатуре и мыши. Благодаря использованию алгоритмов распознавания жестов рук пользователи получают возможность выполнять широкий спектр манипуляций на виртуальной доске в режиме реального времени, исключительно используя свои руки.

В результате проделанной работы была создана виртуальная доска, которая выполняет все поставленные задачи. Программа прошла все испытания и теперь доступна для использования.

Результаты данной работы могут быть применимы в области виртуальных совещаний и совместной работы, позволяя участникам взаимодействовать с виртуальной доской и участвовать в совместной деятельности в режиме реального времени.

					ПГУ1.090401.05.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Кудряшов Я.В.			Программный модуль отслеживания рук для дополненной реальности Пояснительная записка	Лит.	Лист	Листов
Провер.		Митрохин М.А.					4	83
Н. Контр		Бычков А.С.				ФВТ, каф. ВТ		
						гр. 21ВВВм1		
Утверд.		Митрохин М.А.						

(подпись, дата)

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6
ВВЕДЕНИЕ 7
1 Анализ предметной области 9
1.1 Отслеживание рук в реальном времени с помощью MediaPipe 10
1.2 Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей 18
1.3 Выводы по главе 26
2 Реализация 28
2.1 Разработка виртуальной доски 28
2.2 Разработка жестов рук 41
2.3 Выводы по главе 66
ЗАКЛЮЧЕНИЕ 67
СПИСОК ЛИТЕРАТУРЫ 68
ПРИЛОЖЕНИЕ А Листинги программы 69
ПРИЛОЖЕНИЕ Б Презентация 73

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

CPU – центральный процессор

GPU – графический процессор

AR – дополнительная реальность

CNN – сверточная нейронная сеть

DNN – деконволюционная нейронная сеть

API – программный интерфейс приложения

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств

ВВЕДЕНИЕ

Технология дополненной реальности в последние годы демонстрирует значительный рост, предоставляя пользователям уникальный способ взаимодействия с реальным миром. Важнейшим компонентом AR являются программные модули отслеживания рук, которые позволяют пользователям манипулировать виртуальными объектами естественным и интуитивно понятным способом. Программные модули отслеживания рук используют алгоритмы компьютерного зрения для отслеживания движения рук пользователя в режиме реального времени, что позволяет им взаимодействовать с виртуальными объектами естественным образом. Существует два основных типа технологий отслеживания движения рук: основанные на маркерах и безмаркерные. Системы на основе маркеров требуют предварительно установленных маркеров, таких как цветные перчатки, для отслеживания движений рук, в то время как системы без маркеров используют алгоритмы компьютерного зрения для отслеживания рук без маркеров. Разработка точных и эффективных программных модулей для отслеживания движения рук является сложной задачей из-за окклюзии и сложности вычислений. Окклюзия возникает, когда части руки скрыты от взгляда камеры, а вычислительная сложность возникает из-за обработки большого количества данных в режиме реального времени. Для преодоления этих проблем разработчики используют комбинацию методов, таких как алгоритмы машинного обучения и датчики глубины, чтобы повысить точность и скорость отслеживания руки. Потенциальные возможности применения программных модулей для отслеживания рук в AR очень широки: от игр до здравоохранения. В играх отслеживание рук может обеспечить более захватывающий опыт, позволяя игрокам использовать свои руки вместо контроллеров. В здравоохранении отслеживание рук может помочь в физиотерапии, предоставляя обратную связь в реальном времени о движениях рук. Программные модули отслеживания движений рук являются

важнейшим компонентом дополненной реальности, позволяя пользователям взаимодействовать с виртуальными объектами естественным и интуитивно понятным способом. Разработка точного и эффективного программного обеспечения для отслеживания движений рук является сложной задачей, но постоянное развитие компьютерного зрения и машинного обучения позволяет добиться значительного прогресса в этой области. Потенциальные возможности применения отслеживания рук в AR очень широки, от игр до здравоохранения, и мы можем ожидать, что в будущем мы увидим еще больше инновационных применений отслеживания рук в дополненной реальности.

Основной целью данной работы является создание «Программный модуль отслеживания рук для дополненной реальности». Модуль позволит пользователям взаимодействовать с виртуальными объектами, в частности с виртуальной доской, используя жесты рук в качестве метода ввода. Разработка этого модуля сопряжена с рядом технических проблем, таких как обеспечение точности и надежности обнаружения и интерпретации жестов рук в режиме реального времени. Для достижения этой цели мы изучим и оценим различные существующие методы отслеживания жестов рук. Эта оценка будет включать критический анализ сильных и слабых сторон каждого подхода с точки зрения точности, скорости и надежности.

1 Анализ предметной области

При разработке программных продуктов очень важно изучить и оценить различные варианты эффективного решения проблем. Одной из таких проблем является отслеживание рук, которое становится все более популярным в разработке программного обеспечения. В этом разделе мы подробно рассмотрим различные доступные методы отслеживания рук, проанализируем их преимущества и недостатки. Эти методы включают отслеживание на основе камеры. В каждом методе используются различные технологии и подходы, что приводит к разным уровням точности и скорости. Отслеживание на основе камер включает в себя использование камер для получения изображений руки пользователя, а затем с помощью методов обработки изображений для идентификации и отслеживания движений руки. На рисунке 1 представлена модель отслеживания рук для дополненной реальности.

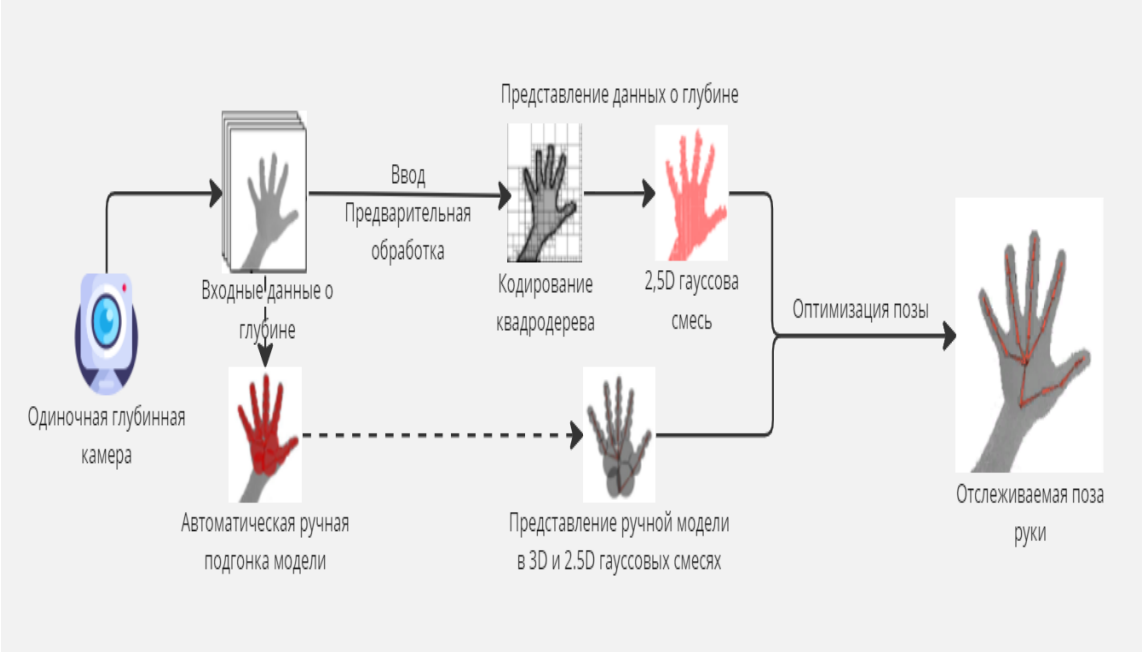


Рисунок 1 – Модель отслеживания рук для дополненной реальности

Понимая сильные и слабые стороны каждого метода отслеживания движения рук, разработчики программного обеспечения могут выбрать наиболее подходящий метод в соответствии с требованиями своего продукта.

Этот процесс выбора может существенно повлиять на общую производительность и удобство использования программного продукта. Поэтому очень важно провести тщательный анализ доступных вариантов перед внедрением модуля отслеживания рук [1].

1.1 Отслеживание рук в реальном времени с помощью MediaPipe

Восприятие формы и движения рук может иметь решающее значение для улучшения взаимодействия с пользователем в целом ряде технических областей и платформ. Например, он может служить основой для понимания языка жестов и управления жестами, а также для наложения цифрового материала и информации на реальную среду в дополненной реальности. В то время как надежное восприятие рук в реальном времени дается людям без особых усилий, это гораздо более сложная работа с компьютерным зрением, поскольку руки часто смыкаются сами по себе или друг другу (например, окклюзия пальца / ладони и дрожание рук) и не имеют высококонтрастных паттернов.

MediaPipe Hands — это решение для отслеживания рук и пальцев с высоким разрешением. Он использует машинное обучение, чтобы определить 21 трехмерный ориентир руки из одного изображения. В то время как существующие современные системы сильно зависят от сложных настольных сред для вывода, наш метод работает в реальном времени на мобильном телефоне и даже масштабируется для многих рук. Мы ожидаем, что предоставление этой возможности восприятия руками более широкому сообществу исследователей и разработчиков будет стимулировать создание новых приложений и направлений исследований. Отслеживаемые трехмерные ориентиры рук представлены точками разного оттенка, при этом более яркие точки обозначают ориентиры, расположенные ближе к камере. Рисунок 2 демонстрирует отслеживаемые трехмерные ориентиры рук.

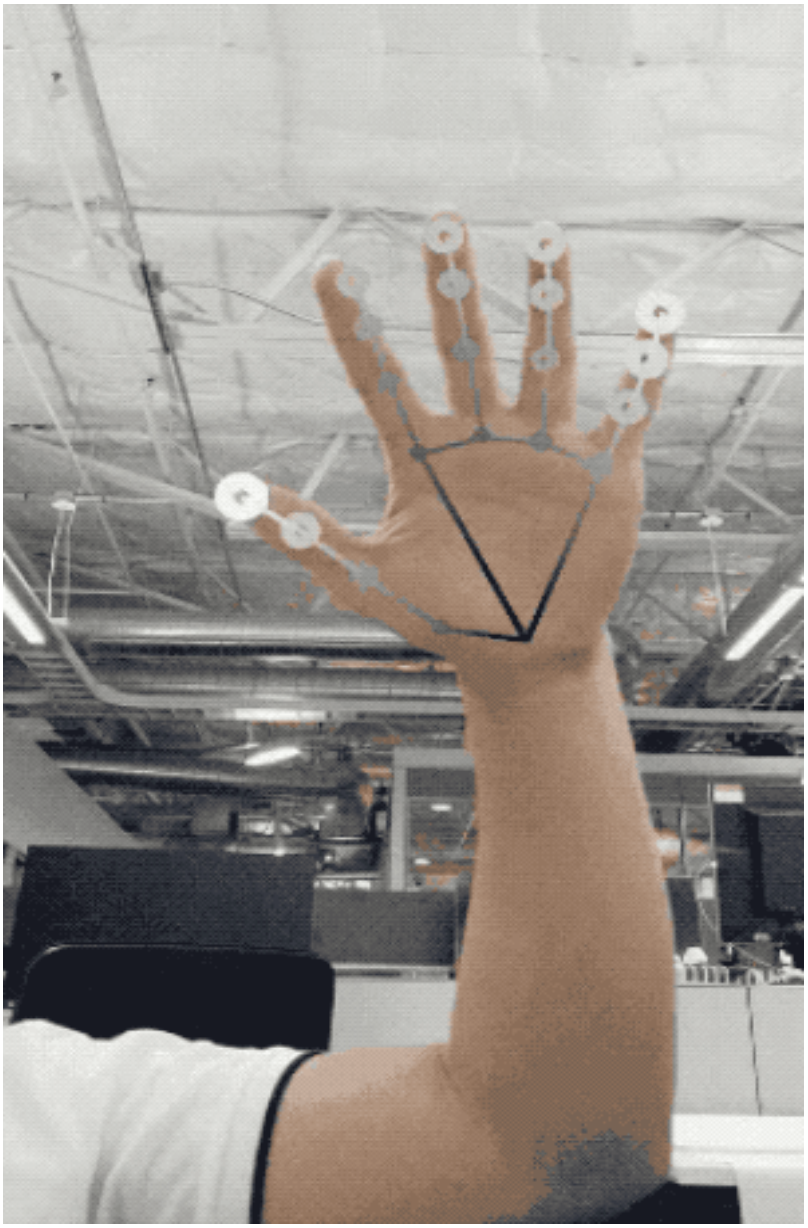


Рисунок 2 – Отслеживаемые трехмерные ориентиры рук

MediaPipe Hands использует конвейер машинного обучения, состоящий из множества взаимодействующих моделей: модель обнаружения ладони, которая действует на все изображение и создает ограничивающую рамку для ориентированной руки. Модель ориентира руки, которая воздействует на обрезанную область изображения детектора ладони и обеспечивает высокоточные трехмерные ключевые точки руки. Этот метод аналогичен тому, который используется в нашем решении MediaPipe Face Mesh, которое сочетает в себе детектор лиц с отображением лицевых ориентиров. Конвейер восприятия рук показан на рисунке 3.

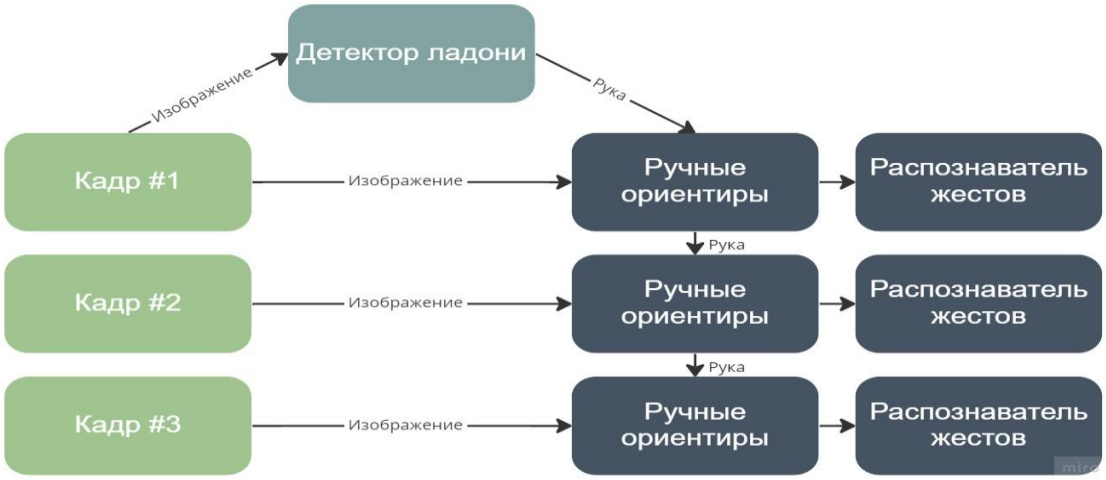


Рисунок 3 – Конвейер восприятия рук

Путем подачи модели ориентира руки соответствующим образом обрезанного изображения руки сеть может посвятить большую часть своих возможностей координации точности прогнозирования, а не увеличению данных (например, вращению, перемещению и масштабированию). Кроме того, в нашем конвейере могут быть созданы культуры в зависимости от ориентиров руки, распознанных в предыдущем кадре, и обнаружение ладони запускается для перемещения руки только тогда, когда модель ориентира не может определить ее существование.

Конвейер построен как граф MediaPipe, который использует подграф отслеживания ориентира руки модуля ориентира руки и визуализирует с помощью подграфа определенного средства рендеринга руки. Внутри подграф отслеживания ориентира руки использует подграф того же модуля ориентира руки и подграф обнаружения ладони модуля обнаружения ладони [2].

Чтобы определить начальное положение рук, мы разработали модель детектора одиночного выстрела, аналогичную той, которая используется в MediaPipe Face Mesh, которая адаптирована для мобильного приложения в реальном времени. Обнаружение рук — очень сложная задача: как наши облегченные, так и полные модели должны работать в диапазоне размеров рук со значительным диапазоном шкалы (20x) по сравнению с рамкой

изображения и быть способными обнаруживать закрытые и само закрытые руки. В то время как лица содержат высококонтрастные узоры, например вокруг глаз и рта, у рук отсутствуют аналогичные характеристики, что затрудняет их точное различение только на основе их визуальных аспектов. Вместо этого предоставление дополнительного контекста, такого как рука, тело или индивидуальные характеристики, помогает в точной локализации руки.

В нашем подходе используются различные способы решения вышеупомянутых проблем. Для начала мы обучаем детектор ладони, а не детектор руки, поскольку оценить ограничивающие рамки жестких объектов, таких как ладони и кулаки, значительно проще, чем распознавать сочлененные пальцы на руках. Кроме того, поскольку ладони представляют собой более мелкие объекты, метод подавления без максимума хорошо работает в ситуациях самоокклюзии двумя руками, таких как рукопожатия. Кроме того, ладони можно моделировать с помощью квадратных ограничивающих рамок (якоря на языке машинного обучения), игнорируя другие соотношения сторон, что приводит к сокращению количества якорей в 3-5 раз. Во-вторых, экстрактор функций кодера-декодера используется, чтобы гарантировать, что даже маленькие объекты знают свой контекст в более крупном изображении (аналогично подходу RetinaNet). Наконец, мы ограничиваем потерю фокуса во время обучения, чтобы приспособиться к огромному количеству привязок, возникающих из-за большого изменения масштаба.

Используя вышеупомянутые стратегии, мы получаем среднюю точность идентификации ладони 95,7%. Без декодера и стандартной потери кросс-энтропии достигается базовый уровень всего 86,22%.

После обнаружения ладони по всему изображению наша следующая модель ориентира руки использует регрессию для точной локализации 21 трехмерной координаты сустава руки внутри идентифицированных областей

руки. Модель получает последовательное внутреннее представление поз рук и, следовательно, устойчива к частично видимым рукам и само окклюзии. На рисунке 4 представлены ручные ориентиры.

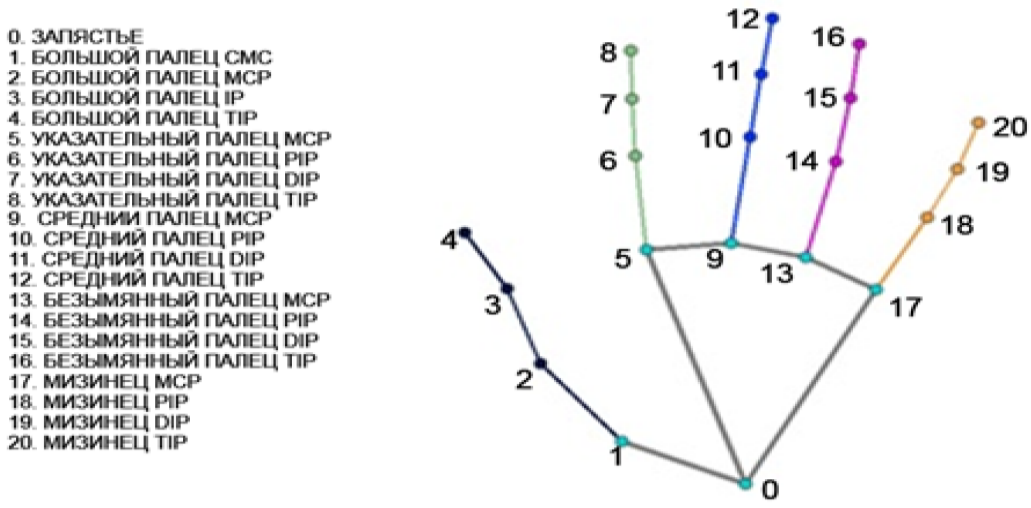


Рисунок 4 – Ручные ориентиры

Чтобы собрать достоверные данные, мы вручную пометили тридцать тысяч реальных фотографий двадцатью одной трехмерной координатой, как показано ниже (мы берем Z-значение из карты глубины изображения, если оно существует для соответствующей координаты). Чтобы лучше представить диапазон возможных положений руки и обеспечить дополнительный контроль над природой геометрии руки, мы дополнительно визуализируем высококачественную синтетическую модель руки на различных фонах и сопоставляем ее с правильными трехмерными координатами. С помощью аннотации наземной информации выровненные урожаи рук передаются в сеть слежения. Внизу: сгенерированные синтетические изображения рук с аннотациями, основанными на реальных данных. Рисунок 5 демонстрирует изображения рук с аннотациями [3].

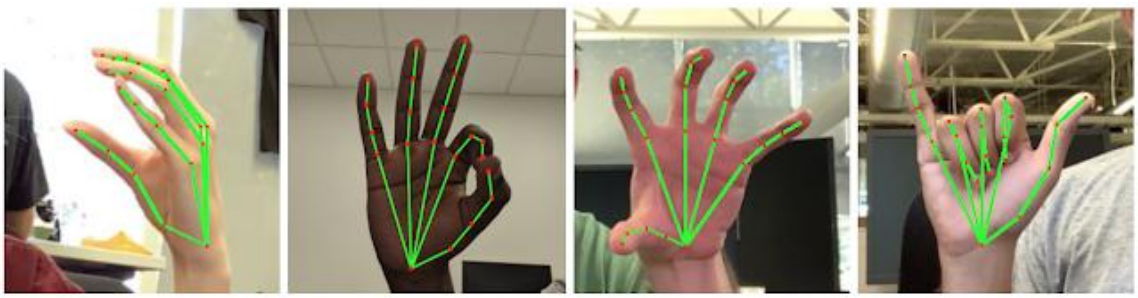


Рисунок 5 – Изображения рук с аннотациями

Однако исключительно сгенерированные данные плохо поддаются обобщению в реальном мире. Чтобы решить эту проблему, мы используем подход к обучению со смешанной схемой. На следующем рисунке изображена высокоуровневая обучающая диаграмма для модели. Для сети отслеживания рук используется схема смешанного обучения. Обрезанные фотографии реального мира и созданные синтетические изображения используются в качестве входных данных для прогнозирования 21 трехмерной ключевой точки. Схема смешанного обучения показана на рисунке 6.

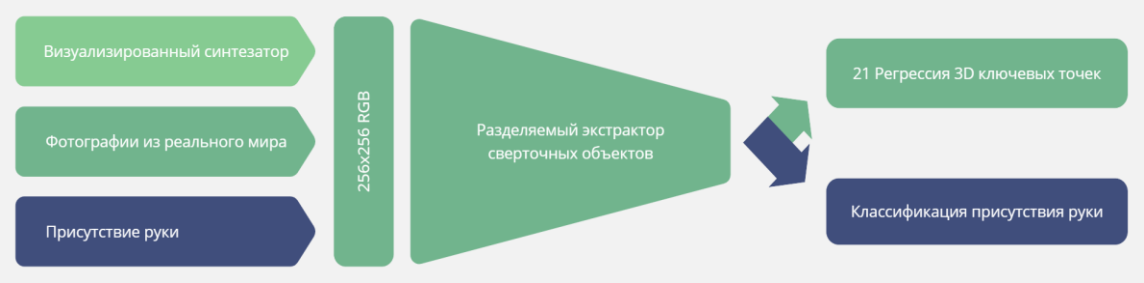


Рисунок 6 – Схема смешанного обучения

Затем мы используем простой алгоритм для ожидаемого скелета руки для создания движений. Для начала, совокупные углы суставов определяют состояние каждого пальца, например, согнутый или прямой. Затем набор состояний пальцев сопоставляется с набором заранее определенных движений. Этот простой, но эффективный подход позволяет нам оценивать основные статические движения с приличной степенью точности. Настоящий конвейер поддерживает множество счетных жестов из разных культур, включая американскую, европейскую и китайскую, а также различные жесты руками, в том числе «Большой палец вверх», сжатый кулак, «ОК», «Рок» и «Человек-паук». На рисунке 7 представлено «Рок», «Человек-паук» и «ОК».

Рисунок 7 – «Рок», «Человек-паук» и «ОК»

Этот конвейер восприятия может быть построен с использованием MediaPipe в виде ориентированного графа модульных компонентов, получивших название «Калькуляторы». Mediapipe включает в себя расширяемую коллекцию калькуляторов для выполнения таких действий, как вывод модели, алгоритмы обработки мультимедиа и преобразование данных на различных устройствах и платформах. Отдельные задачи, такие как кадрирование, рендеринг и вычисления нейронной сети, могут выполняться полностью на графическом процессоре. Например, на большинстве современных телефонов мы используем вывод графического процессора TFLite [4].

На приведенном ниже графике показано отслеживание рук MediaPipe. График разделен на два подграфа: один для обнаружения руки, а другой для вычисления ключевых точек руки (т.е. Ориентиров). Одним из заметных улучшений, предоставляемых MediaPipe, является то, что детектор ладони запускается только тогда, когда это необходимо (очень редко), что приводит к значительной экономии времени на вычисления. Это достигается путем определения положения руки в будущих кадрах видео из вычисленных ключевых точек руки в текущем кадре, что позволяет избежать необходимости запускать детектор ладони для каждого кадра. Чтобы обеспечить надежность, модель ручного трекера создает дополнительный скаляр, выражающий степень уверенности в том, что рука присутствует и адекватно выровнена внутри входного поля. Модель обнаружения рук повторно запускается для всего кадра только тогда, когда достоверность

падает ниже определенного уровня. Когда модель обнаружения руки активирована, выходные данные модели ориентира руки (REJECT HAND FLAG) используются для регулирования, когда срабатывает модель обнаружения руки. Такое поведение обеспечивается сложными компонентами построения синхронизации MediaPipe, которые обеспечивают отличную производительность и оптимальную пропускную способность конвейера машинного обучения. Рисунок 8 демонстрирует отслеживание рук MediaPipe.

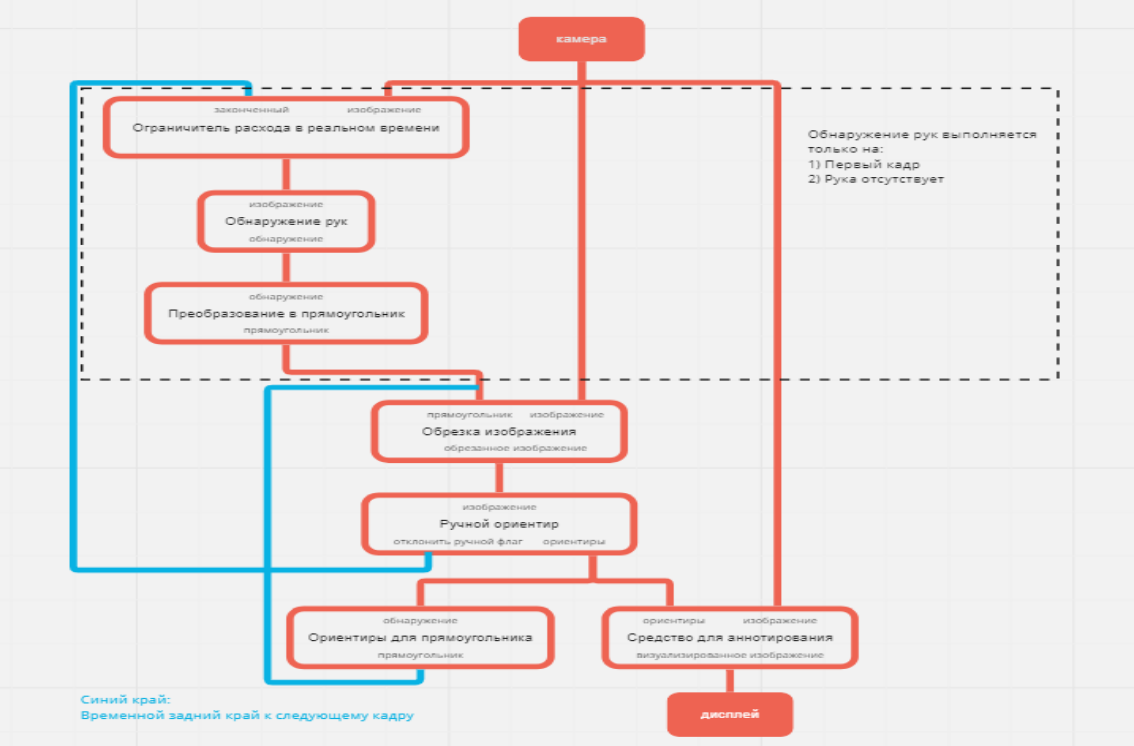


Рисунок 8 – Отслеживание рук MediaPipe

Высокоэффективная система машинного обучения, которая работает в режиме реального времени и на различных платформах и форм-факторах, влечет за собой значительно большую сложность, чем показано в простом примере, приведенном выше. С этой целью мы открываем исходный код вышеупомянутого конвейера отслеживания рук и обнаружения жестов в платформе MediaPipe вместе с соответствующим сценарием сквозного использования и исходным кодом. Это предлагает исследователям и

разработчикам обширный стек для разработки и экспериментов с новыми концепциями, основанными на нашей методологии [5].

Мы хотим улучшить эту технологию, добавив более надежное и устойчивое отслеживание, увеличив количество движений, которые могут быть надежно обнаружены, и позволив динамическим жестам разворачиваться во времени. Мы думаем, что распространение этой технологии будет стимулировать разработку новых инновационных идей и приложений членами академического сообщества и сообщества разработчиков. Надежное и устойчивое отслеживание показано на рисунке 9.

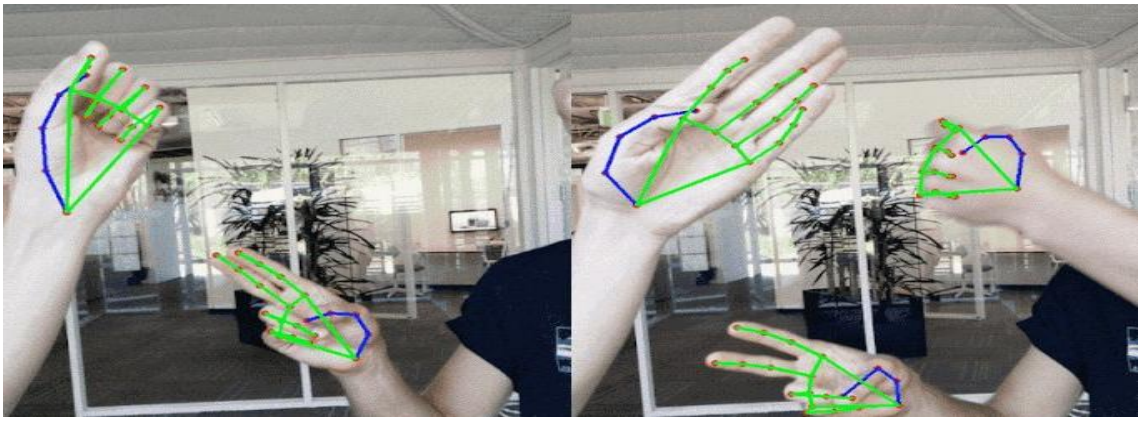


Рисунок 9 – Надежное и устойчивое отслеживание

1.2 Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей

Недавний эксперимент с использованием API идентификации объектов Tensorflow для отслеживания рук на фотографиях дал впечатляющие результаты. В результате обученная модель и исходный код были выложены в открытый доступ и уже использованы для прототипирования некоторых интригующих приложений, таких как инструмент для помощи детям с произношением, расширения для предсказания языка жестов и даже настольного тенниса. Несмотря на потенциал этой технологии, многие люди сталкиваются с проблемами при настройке Tensorflow, среди которых часто встречаются проблемы с установкой, версионированием и экспортом графов. К счастью, Tensorflow разработан для работы в стандартизированной среде,

предоставляемой браузерами, что устраняет многие из этих проблем с настройкой дистрибутива. Чтобы еще больше упростить процесс создания взаимодействия рук и жестов с помощью обученной модели обнаружения рук, группа разработчиков создала фреймворк под названием Handtrack. Этот фреймворк абстрагируется от процедур, связанных с загрузкой файлов моделей, и предлагает полезные функции, которые позволяют распознавать руки на изображениях даже тем, кто не имеет предварительных знаний в области машинного обучения. Более того, у пользователей есть возможность узнать больше о разработке моделей, если они пожелают, и нет необходимости экспортировать какие-либо замороженные графики или сохраненные модели. На рисунке 10 представлено обнаружение в реальном времени видеопотока с веб-камеры [6].

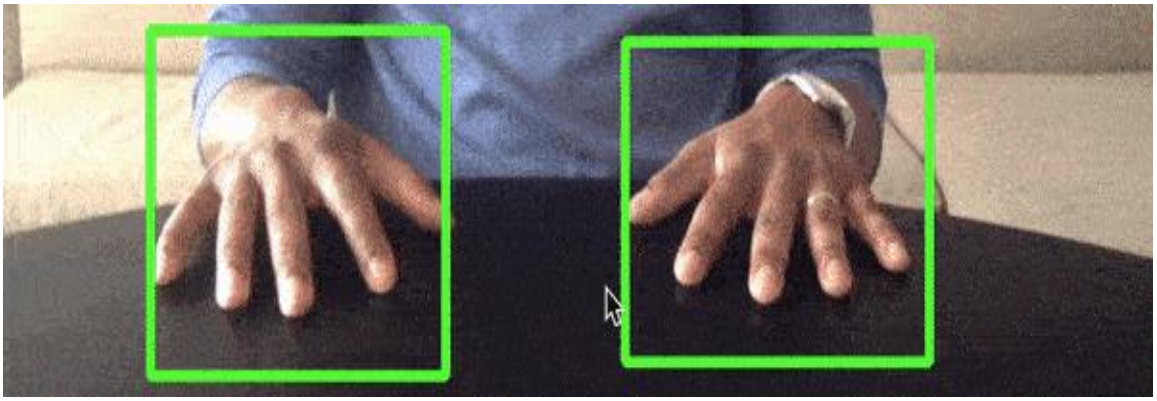


Рисунок 10 – Обнаружение в реальном времени видеопотока с веб-камеры

Одним из наиболее сложных и дорогостоящих аспектов обучения нейронных сетей для задач компьютерного зрения является получение подходящего набора данных с аннотированными примерами. Исследователи часто тратят значительное количество времени на поиск или создание набора данных, подходящего для конкретной задачи. В случае с отслеживанием рук этот процесс ничем не отличается. Однако с появлением различных наборов данных для отслеживания рук бывает сложно определить, какой из них лучше всего подходит для решения поставленной задачи. В недавнем исследовании ученые экспериментировали с двумя популярными наборами

данных: Oxford Hands и Egohands, чтобы определить, какой из них лучше подходит для их нужд. Они обнаружили, что набор данных Egohands, включающий аннотированные примеры рук в эгоцентрических ракурсах, высококачественные изображения и ручные аннотации, лучше подходит для их конкретной задачи отслеживания рук, чем набор данных Oxford Hands. Это подчеркивает важность выбора набора данных, который отвечает конкретным потребностям поставленной задачи.

Компьютерное зрение предлагает множество методов для отслеживания рук, но многие из этих методов опираются на фиксированные правила. Эти основанные на правилах подходы, например, использующие текстуру и особенности границ для удаления фона, или цветовые гистограммы и классификаторы для отличия рук от фона, не всегда надежны. В случаях, когда фон необычный, или условия освещения быстро меняются, или отслеживаемый объект становится загороженным, эти алгоритмы могут быть подвержены ошибкам.

Нейронные сети способны преодолеть ограничения традиционных методов отслеживания объектов благодаря эффективному обучению моделей на больших массивах данных. Они способны решать такие задачи, как плохое освещение, шум, несколько ракурсов и окклюзии. Однако основными недостатками использования этих алгоритмов для слежения в реальном времени являются их сложность, низкая скорость по сравнению с методами, использующими только слежение, и высокая стоимость создания надежных наборов данных. Тем не менее, для решения этих проблем разрабатываются быстрые нейронные сети.

Кроме того, фреймворки глубокого обучения, такие как Tensorflow, облегчили работу исследователей в этой области. Эти фреймворки упрощают процесс создания моделей, способных идентифицировать конкретные объекты. Более того, быстрые модели нейронных сетей, включая ssd, более быстрый rcnn, rfcn и другие, сделали нейронные сети привлекательным

вариантом для приложений обнаружения и отслеживания объектов в реальном времени.

Чтобы обучить пользовательский детектор объектов с помощью API обнаружения объектов Tensorflow, необходимо выполнить несколько шагов, включая сбор набора данных, его очистку, разбиение на обучающую и тестовую части и построение выходного графа.

Модель, используемая для обнаружения рук, обучается с помощью набора данных Egohands, который обладает многочисленными преимуществами. Набор данных содержит точные аннотации на уровне пикселей для более чем 15 000 ориентиров и состоит из более чем 4800 фотографий, снятых с эгоцентрической точки зрения. Набор данных охватывает широкий спектр из 48 сценариев в помещении и на улице, включая различные виды деятельности, такие как игра в карты, шахматы, дженга и решение головоломок. Рисунок 11 демонстрирует модель детектора рук из набора данных Egohands.

Рисунок 11 – Модель детектора рук из набора данных Egohands

Набор данных Egohands состоит из 48 папок, каждая из которых содержит 100 помеченных изображений положения рук, полученных из различных мест, запечатленных на видеоматериалах. Набор данных Egohands показан на рисунке 12.

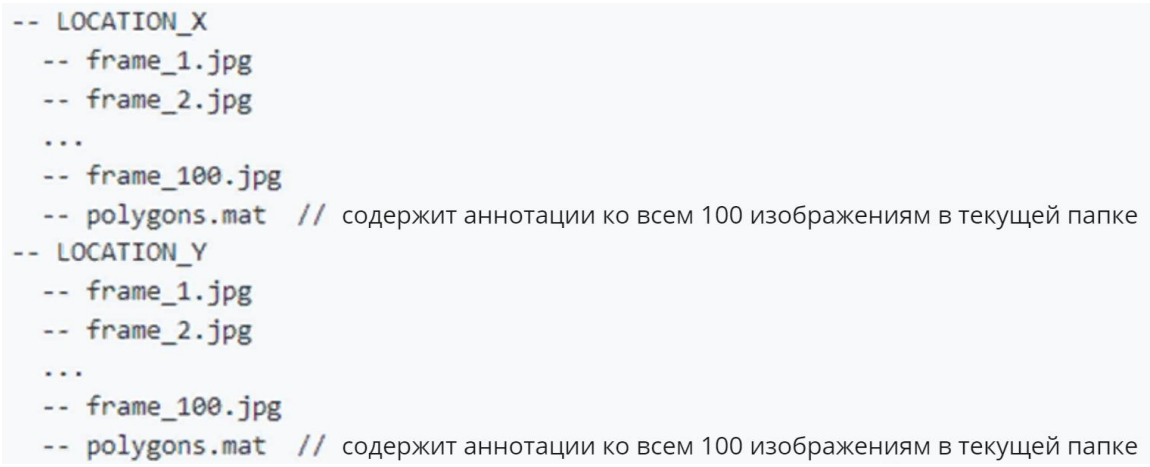


Рисунок 12 – Набор данных Egohands

Перед обучением модели с набором данных Egohands необходимо выполнить некоторые начальные шаги для преобразования его в формат Tensorflow (tfrecord). К счастью, репозиторий предоставляет полезный набор данных egohands clean.py script для создания необходимых файлов CSV. Этот скрипт, выполняет несколько операций.

Во-первых, всем файлам присваиваются уникальные имена путем добавления к ним соответствующих имен каталогов. Затем данные делятся на три папки для обучения (80%), тестирования (10%) и оценки (10%). Файл polygons.mat для каждой папки считывается, и для них создаются ограничительные рамки. Точность ограничительных рамок подтверждается визуализацией. После выполнения сценария на выходе должна получиться папка с изображениями, содержащая три файла с именами train, test и eval соответственно. Кроме того, для каждой папки должен быть CSV-документ с метками, названный train labels.csv и test labels.csv соответственно, который можно использовать для создания tfrecords.

В наборе данных egohands есть четыре типа меток для рук - собственная левая, собственная правая, другая левая и другая правая. Однако для нашей цели нам нужно обозначить руки как общий класс и не делать различий между левыми и правыми руками. Следовательно, на этапе подготовки данных мы пометим все обучающие данные как руки. При необходимости

сценарий может быть изменен для создания четырех различных меток tfrecords для четырех типов рук, упомянутых в наборе данных.

Далее необходимо преобразовать набор данных и связанные с ним файлы CSV в формат tfrecords. Это преобразование позволит вам создать необходимые файлы train.record и test.record для каждой папки, которые будут использоваться в процессе обучения.

Для того чтобы обучить модель на созданном наборе данных, для сокращения времени обучения модели можно использовать трансферное обучение. Трансферное обучение предполагает использование существующей модели, успешно обученной в аналогичной области (например, классификации изображений), и переобучение ее последнего слоя (слоев) для распознавания рук. Эта техника позволяет обучать нейронные сети с миллионами параметров за часы, а не за недели или месяцы. Модель ssd mobilenet v1 является одной из самых быстрых моделей для трансферного обучения в Tensorflow. Процесс обучения может быть выполнен на локальной системе CPU, но обучение на GPU (например, на облачном компьютере) более эффективно. Обучение на CPU-системе MacBook Pro достигло максимальной производительности в 5 секунд на шаг, в то время как использование GPU привело к 0,5 секунды на шаг. Для выполнения 200 тысяч шагов на MacBook Pro потребуется около 12 дней, тогда как на GPU те же шаги можно выполнить за 5 часов [7].

В области машинного обучения одной из ключевых целей является минимизация ошибок по мере продвижения процесса обучения. Типичный способ определить, происходит ли это, посмотреть на графики тензорной доски на предмет суммарных потерь, которые должны уменьшаться по мере увеличения числа итераций. В недавнем эксперименте задача обучения была решена после 200 000 шагов, что заняло около пяти часов, а общий убыток составил 2,575. Однако, оглядываясь назад, можно было бы завершить задачу примерно за 50 000 шагов при сопоставимых общих потерях. TensorFlow

также можно использовать для параллельного оценивания, чтобы оценить производительность модели на тестовых данных. Обычно используемой мерой производительности является средняя точность, которая представляет собой единичное значение, суммирующее площадь под кривой «точность-отдача». Эта мера показывает, насколько хорошо модель генерирует граничные рамки, которые перекрывают истинные граничные рамки не менее чем на 50% в тестовом наборе данных. В данном эксперименте для обучения ручного детектора использовалось значение 0,96860. Средние значения точности варьируются от 0 до 1, при этом 1 является оптимальным значением. На рисунке 13 представлены графики обучения ручного детектора.

Рисунок 13 – Графики обучения ручного детектора

По завершении процесса обучения полученные графики ручного детектора экспортируется и сохраняется в указанной папке выходных графиков. На этом этапе можно сделать интересный вывод. Значительное улучшение производительности объясняется несколькими факторами.

Важным аспектом приложений компьютерного зрения является возможность получения изображений в режиме реального времени, что требует эффективной обработки данных изображения. Считывание изображений с камеры может быть ресурсоемким, что может замедлить выполнение программы, если оно выполняется в основном потоке

приложения. Для решения этой проблемы был применен подход Адриана Роузбака, заключающийся в распараллеливании захвата изображений на несколько рабочих потоков, что привело к увеличению частоты кадров примерно на пять пунктов.

Еще одним фактором, который необходимо учитывать, является цветовой формат изображений, возвращаемых cv2.read(). Эти изображения возвращаются в формате BGR, и перед обнаружением объектов их необходимо преобразовать в RGB. Если этого не сделать, точность детектора может значительно снизиться.

В дополнение к этим факторам, уменьшение размера входных изображений также может повысить частоту кадров без потери точности. Это было достигнуто путем использования разрешения около 320 x 240 вместо 1280 x 720, предоставляемых камерой.

Квантование моделей - еще одна стратегия, которая может быть использована для снижения требований к памяти для загрузки и хранения моделей. Уменьшение размера модели с 32 до 8 бит может привести к четырехкратному снижению требований к памяти. Дополнительным методом оптимизации является использование 8-битного квантования с фиксированной точкой [8].

В настоящее время мы изучаем возможность повышения производительности нашей системы обнаружения путем сочетания методов отслеживания с уже имеющимися у нас достойными методами обнаружения. Мы обнаружили, что производительность детектора недостаточна, когда речь идет о случайных фотографиях рук, и мы работаем над устранением этих недостатков. Используя методы отслеживания, мы надеемся еще больше повысить точность и аккуратность нашей системы обнаружения. Таким образом, мы сможем получить лучшие результаты и повысить общую эффективность системы. Рисунок 14 демонстрирует работу детектора на случайных фотографиях.

Рисунок 14 – Работа детектора на случайных фотографиях

Ограничения обучающего набора отражены в детекторе, включая неэгоцентрические точки зрения, сложные фоны, такие как море рук, и различные оттенки кожи. Дополнительные данные могут помочь преодолеть эти ограничения. Один из подходов к улучшению работы детектора заключается в том, чтобы соединить его новые знания о «руках» с детекторами, специализирующимися на распознавании других объектов. К сожалению, детектор рук не может распознавать другие объекты, а создание детектора, способного распознавать множество объектов, потребует создания наборов данных для каждого класса, что является длительным и громоздким процессом. Потенциальным решением является исследование структур, которые могут эффективно переключаться между несколькими предварительно обученными моделями, представляющими различные классы объектов, что позволяет им распознавать несколько объектов на одном изображении. Например, мы могли бы использовать этот подход для определения положения объектов на столе относительно положения рук на том же столе.

1.3 Выводы по главе

В этой главе основное внимание было уделено изучению различных методов отслеживания движения рук и анализу их эффективности с выделением их преимуществ. В главе приводится всесторонний обзор

различных подходов к отслеживанию рук в реальном времени, и после тщательного рассмотрения был сделан вывод, что фреймворк MediaPipe является наиболее подходящей платформой для создания алгоритмов, способных обнаруживать и отслеживать движения рук человека в видеопотоке в реальном времени. Описанная в этой главе работа является важным компонентом разработки продукта и закладывает основу для создания приложений, которые полагаются на технологию отслеживания движений рук.

2 Реализация

Для создания «Программный модуль отслеживания рук для дополненной реальности» процесс разработки разделен на два отдельных этапа. На первом этапе создается виртуальная доска, которая является интерфейсом, где жесты рук будут распознаваться и использоваться для взаимодействия с ней. Второй этап посвящен разработке жестов рук, которые будут использоваться для рисования, стирания и других манипуляций на виртуальной доске.

Для разработки виртуальной доски программное обеспечение должно иметь набор инструментов и быть способна создавать графическое представление движений рук пользователя в режиме реального времени для взаимодействия с этим набором инструментов.

Следующим шагом идёт определение жестов рук, которые будут использоваться для взаимодействия с виртуальной доской. Этот процесс достигается путем анализа положения пальцев в данных изображения, полученных с камеры.

Наконец, определение жестов рук должно быть интегрировано с программой виртуальной доски. Программа будет постоянно захватывать кадры с камеры и применять алгоритмы обработки изображений для определения движений рук пользователя. Обнаруженные жесты рук будут использоваться для запуска определенных действий на виртуальной доске. Таким образом, пользователь сможет взаимодействовать с доской, просто двигая руками и пальцами. Полученный программный модуль позволит пользователям рисовать, стирать и проделывать различные манипуляции на виртуальной доске с помощью интуитивных жестов рук.

2.1 Разработка виртуальной доски

Ранее в тексте упоминалось, что виртуальная доска должна включать в себя набор инструментов и генерировать визуальное отображение движений рук пользователя в режиме реального времени. Это означает, что

виртуальная доска позволит пользователям рисовать, делать наброски и писать, как если бы они использовали традиционную физическую доску. Кроме того, планируется создать виртуальный курсор мыши, который будет использовать указательный палец пользователя в качестве указателя. Этот курсор позволит пользователям легко перемещаться по виртуальной доске и взаимодействовать с ее инструментами. Курсор будет отслеживать движения пальца пользователя, позволяя ему легко выбирать, тот или иной инструмент и манипулировать им на доске.

2.1.1 Панель инструментов

Для разработки панели инструментов мы будем использовать библиотеку компьютерного зрения OpenCV для создания холста, на котором будут размещены наши инструменты, что позволит нам настроить размер и расположение панели инструментов в соответствии с нашими конкретными потребностями, а также онлайн-редактор фотографий для легкого проектирования и рисования панели инструментов. Но, прежде чем приступить к рисованию панели инструментов, необходимо определить набор инструментов. Для нашего проекта мы выбрали стандартный набор инструментов, включающий:

— квадрат,

— круг,

— линия,

— перо,

— ластик.

Каждый инструмент будет размещен в квадрате размером 50 на 50 пикселей, где мы присвоим каждому инструменту определенный цвет, чтобы сделать их легко узнаваемыми. Таким образом, мы создадим компактную и интуитивно понятную панель инструментов, которая позволит пользователям быстро получить доступ к необходимым им инструментам. Такой подход широко используется в различных приложениях, включая программы для

редактирования фотографий и инструменты для дизайна, и доказал свою эффективность в повышении удобства работы пользователя. Панель инструментов показана на рисунке 15.

Рисунок 15 – Панель инструментов

2.1.2 Виртуальный курсор мыши

После реализации графического интерфейса нашей программы, который включает в себя холст и панель инструментов, следующим шагом будет добавление функциональности для взаимодействия с интерфейсом с помощью виртуального курсора мыши. Для этого нам сначала нужно импортировать необходимые модули.

Листинг программы:

```
import cv2
import mediapipe as mp
import autopy
```

Здесь мы импортируем три модуля, которые мы будем использовать в нашем коде. cv2 - модуль OpenCV, mediapipe - модуль MediaPipe, и autopy - модуль, который мы будем использовать для управления курсором мыши.

Далее мы проинициализируем объекты MediaPipe и OpenCV.

Листинг программы:

```
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
cap = cv2.VideoCapture(0)
```

Здесь мы инициализируем объекты mp_drawing и mp_hands из модуля MediaPipe, которые мы будем использовать для рисования ориентиров на изображении и обнаружения ориентиров на руках, соответственно. Мы также инициализируем объект cap из метода OpenCV cv2.VideoCapture(), который мы будем использовать для захвата видео с веб-камеры.

Теперь настроим управление мышью.

Листинг программы:

```
autopy.mouse.toggle(autopy.mouse.Button.LEFT, False)
```

Здесь мы используем метод autopy.mouse.toggle() для имитации щелчка левой кнопкой мыши, при этом параметр False указывает на то, что кнопку мыши следует отпустить.

После того как мы проинициализировали наши объекты, можем перейти к нашему главному циклу программы.

Листинг программы:

```
with mp_hands.Hands(
    max_num_hands=1,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")
            continue
```

Здесь мы используем оператор with для инициализации объекта hands из модуля mp_hands, при этом параметры max_num_hands, min_detection_confidence и min_tracking_confidence установлены на 1, 0.5 и 0.5 соответственно. Затем мы запускаем цикл while, который считывает кадры с веб-камеры с помощью cap.read(), и продолжаем только в том случае, если флаг установлен в True.

Далее преобразуем изображение в RGB и обработаем его с помощью MediaPipe

Листинг программы:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image.flags.writeable = False
```

```
results = hands.process(image)
image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
```

Здесь мы сначала преобразуем изображение из формата BGR (синий-зеленый-красный) в формат RGB (красный-зеленый-синий) с помощью cv2.cvtColor(). Затем мы устанавливаем флаг возможности записи объекта изображения на False и используем метод hands.process() для обнаружения ориентиров руки на изображении. Затем мы устанавливаем флаг записи объекта изображения на True и конвертируем изображение обратно в формат BGR.

Теперь рисуем на изображении ориентиры для рук.

Листинг программы:

```
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.DrawingSpec(color=(255, 255, 255), thickness=2, circle_radius=2),
                mp_drawing.DrawingSpec(color=(0, 0, 255), thickness=2, circle_radius=2),
                )
```

Здесь мы используем оператор if, чтобы проверить, содержит ли объект результатов, возвращенный hands.process(), какие-либо обнаруженные ориентиры для рук (results.multi_hand_landmarks). Если да, то мы выполняем итерации по каждому обнаруженному ориентиру руки (для hand_landmarks в results.multi_hand_landmarks:) и используем метод mp_drawing.draw_landmarks() для рисования ориентиров на изображении.

Параметр mp_hands.HAND_CONNECTIONS указывает, что между ориентирами рук должны быть проведены линии, чтобы показать связи между ними. Объекты DrawingSpec задают цвет, толщину и радиус окружности ориентиров и связей.

Как упомяналось ранее виртуальный курсор мыши, будет использовать указательный палец пользователя в качестве виртуального указателя мыши, поэтому давайте получим положение кончика указательного пальца.

Листинг программы:

```
index_finger_tip = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
x, y = int(index_finger_tip.x * image.shape[1]), int(index_finger_tip.y * image.shape[0])
```

Здесь мы используем синтаксис hand_landmarks.landmark[] для доступа к ориентиру, соответствующему кончику указательного пальца (mp_hands.HandLandmark.INDEX_FINGER_TIP). Затем мы умножаем координаты x и y кончика указательного пальца на ширину и высоту изображения, соответственно, чтобы получить пиксельные координаты кончика на изображении. На рисунке 16 представлен кончик указательного пальца.

Рисунок 16 – Кончик указательного пальца

Теперь нам необходимо перемещать курсор мыши вслед за кончиком указательного пальца.

Листинг программы:

```
autopy.mouse.move(x, y)
```

Здесь мы используем метод autopy.mouse.move() для перемещения курсора мыши в пиксельные координаты кончика указательного пальца.

Следует отметить, что разрешение дисплея может быть слишком велико и дотянуться до краёв дисплея курсором мыши может оказаться не такой простой задачей, как кажется на первый взгляд. Поэтому чтобы решить эту проблему, необходимо уменьшить (или обрезать) разрешение дисплея на определенное количество пикселей по ширине и высоте. После завершения уменьшения (или обрезки) разрешения дисплея, можно использовать линейную интерполяцию, чтобы перенести координаты местоположения курсора мыши из исходного разрешения в новое, более легкодоступное разрешение. Впоследствии нет необходимости тянуться рукой к краям дисплея, достаточно всего лишь упереться в край фиолетового прямоугольника чтобы курсор мыши оказался в краю дисплея.

Листинг программы:

```
x2 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
y2 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
autopy.mouse.move(x2, y2)
```

Здесь мы вычисляем координату x местоположения объекта после интерполяции. Функция np.interp() принимает три аргумента: первый аргумент x1 - входное значение, подлежащее интерполяции, второй аргумент (frameR, wCam - frameR) задает диапазон входного значения, третий аргумент (0, wScr) задает диапазон выходного значения. Здесь frameR - радиус круговой области интереса в кадре камеры с центром вокруг объекта, wCam - ширина кадра камеры, а wScr - ширина выходного экрана. Функция

np.interp() переводит входное значение x1 из диапазона (frameR, wCam - frameR) в диапазон (0, wScr) и возвращает интерполированное значение x2. Аналогично, вторая строка кода вычисляет y - координату местоположения объекта после интерполяции. Аргумент y1 - входное значение, подлежащее интерполяции, (frameR, hCam - frameR) задает диапазон входного значения, а (0, hScr) - диапазон выходного значения. hCam - высота кадра камеры, а hScr - высота выходного экрана. Рисунок 17 демонстрирует уменьшение разрешения дисплея и линейную интерполяцию координат.

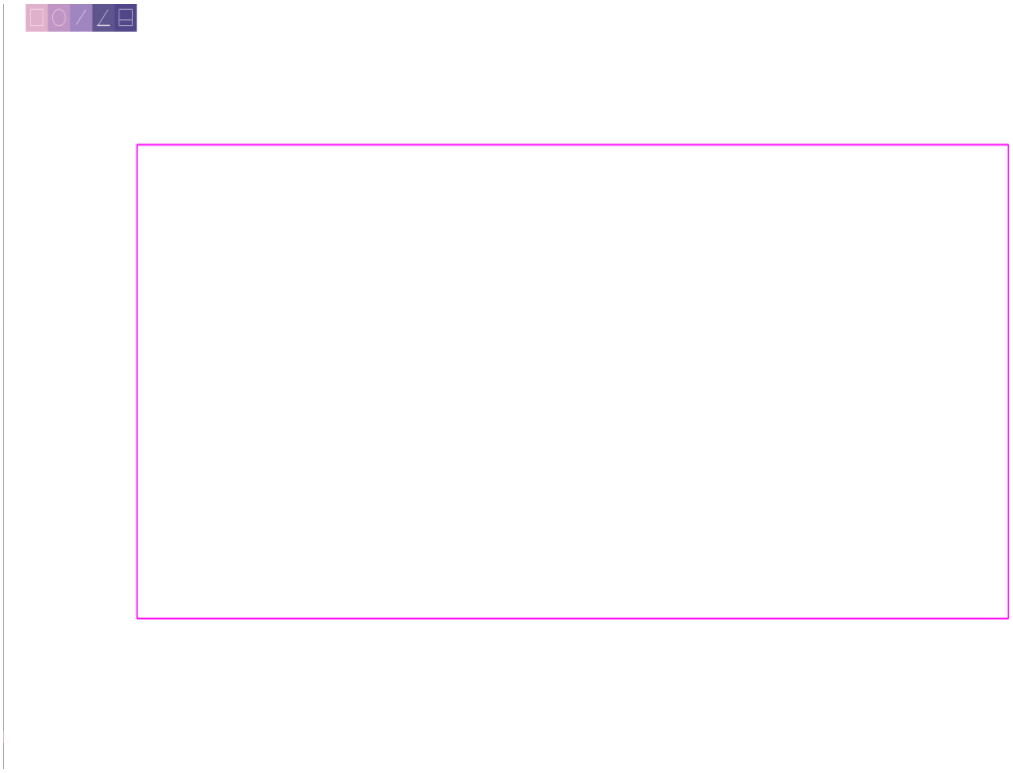


Рисунок 17 – Уменьшение разрешения дисплея и линейная интерполяция координат

Так же не стоит забывать о необходимости сглаживания движения виртуального курсора мыши для обеспечения плавного взаимодействия с пользователем. Для решения этой проблемы можно применить алгоритм, использующий методы сглаживания или фильтрации.

Листинг программы:

```
x2 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
y2 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
```

```
clocX = plocX + (x2 - plocX) / smoothening
clocY = plocY + (y2 - plocY) / smoothening
autopy.mouse.move(clocX, clocY)
```

Здесь выходные значения clocX и clocY представляют текущее местоположение объекта после сглаживания, а входные значения plocX и plocY представляют предыдущее местоположение объекта до сглаживания. Во второй строке кода вычисляется сглаженная координата X местоположения объекта. Для этого к предыдущей X-координате plocX добавляется доля разницы между текущей X-координатой x2 и предыдущей X-координатой plocX, где доля определяется переменной smoothhening. В третей строке кода вычисляется сглаженная Y-координата местоположения объекта по той же формуле, что и в первой строке, но вместо нее используются Y-координаты. Переменная smoothhening - параметр, который управляет количеством сглаживания, применяемого к местоположению объекта. Большее значение сглаживания приводит к большему сглаживанию, что уменьшает дрожание или шум в положении объекта, но также может привести к замедлению времени отклика. И наоборот, меньшее значение сглаживания приводит к меньшему сглаживанию, что приводит к более быстрому отклику, но более шумному отслеживанию.

После того как мы научились перемещать виртуальный курсор, следующим шагом будет реализация имитации нажатия левой кнопки мыши, которая позволит нам взаимодействовать с объектами графического интерфейса (панель инструментов). В отличие от физической мыши, имитация нажатия левой кнопки на виртуальной мыши немного отличается. При использовании физической мыши вы обычно подводите курсор к объекту, наводите на него курсор и затем нажимаете левую кнопку мыши, чтобы взаимодействовать с ним. Однако в виртуальной мыши нет кнопок, которые нужно нажимать, и поэтому механизм взаимодействия отличается, хотя и незначительно. Вместо нажатия кнопки мы просто удерживаем

виртуальный курсор над объектом в течение короткого промежутка времени, после чего происходит взаимодействие с объектом.

Листинг программы:

```
if x < max_x and y < max_y and x > ml:
    if time_init:
        ctime = time.time()
        time_init = False
    ptime = time.time()
    cv2.circle(frm, (x, y), rad, (0, 255, 255), 2)
    rad -= 1
    if (ptime - ctime) > 0.8:
        curr_tool = getTool(x)
        print("your current tool set to : ", curr_tool)
        time_init = True
        rad = 40
else:
    time_init = True
    rad = 40
```

Здесь мы используем условный оператор, который проверяет, находятся ли x и y в определенных границах, и больше ли x определенного значения. Если все три условия выполнены, проверяется, истинно ли значение time_init. Если да, то текущее время сохраняется в переменной ctime, а переменная time_init устанавливается в false, чтобы указать, что начальное время было установлено. Затем текущее время сохраняется в переменной ptime. Далее на изображении, рисуется круг в позиции (x, y) с радиусом rad, цветом (0, 255, 255) и толщиной 2 пикселя. После этого переменная rad уменьшается на 1. Если разница между ptime и ctime больше 0,8 секунды, вызывается функция getTool с параметром x, а возвращаемое значение сохраняется в curr_tool, эта функция возвращает определёный инструмент, на который в данный момент

указывает виртуальный курсор. Затем на консоль выводится сообщение о том, что текущий инструмент был установлен, переменная time_init установлена в true, а переменная rad сброшена на 40. Если ни одно из условий не выполняется, переменная time_init устанавливается в true, а переменная rad сбрасывается до 40.

Далее в главном цикле программы изображение выводится на экран (или перерисовывается), чтобы убедиться, что программа работает так, как задумано. Это включает обновление изображения с учетом всех изменений, внесенных программой. Чтобы выйти из программы, пользователь может нажать клавишу на клавиатуре. Это дает пользователю возможность остановить программу и вернуть управление операционной системе.

Листинг программы:

```
cv2.imshow('Virtual Mouse', image)
if cv2.waitKey(5) & 0xFF == 27:
   break
```

Здесь мы используем метод cv2.imshow() для отображения изображения с заголовком окна «Виртуальная мышь». Затем мы используем метод cv2.waitKey(), чтобы дождаться нажатия клавиши и выйти из цикла, если нажатой клавишей является клавиша Escape (которая имеет значение 27 в коде ASCII). Взаимодействие виртуального курсора с объектом показан на рисунке 18.

Рисунок 18 – Взаимодействие виртуального курсора с объектом

В заключении важно отметить правильное управление системными ресурсами, такими как память, дескрипторы файлов и другие системные ресурсы. Это особенно важно, когда программа выполнила свою задачу и больше не нужна, поскольку ресурсы, которые она использовала, могут быть выделены и недоступны для других процессов. Для решения этой проблемы рекомендуется, чтобы программы освобождали ресурсы, когда они больше не нужны. Это означает освобождение ресурсов после завершения работы программы т.е. нажатия пользователем клавиши Escape. Освобождение ресурсов подразумевает освобождение ресурсов, которые были выделены во время выполнения программы, например, памяти, выделенной для хранения переменных или буферов, используемых для чтения или записи данных. Этот процесс обычно выполняется с помощью специальных команд или функций, которые предоставляются языком программирования или операционной системой. Неспособность освободить ресурсы может привести к целому ряду проблем, включая снижение производительности, проблемы со стабильностью и сбои системы. Правильное управление ресурсами и их высвобождение, когда они больше не нужны, позволяет программам работать более эффективно и результативно, а также помогает предотвратить возникновение проблем, связанных с ресурсами.

Листинг программы:

```
cap.release()
cv2.destroyAllWindows()
```

Здесь мы используем метод cap.release() для освобождения ресурсов, используемых веб-камерой, и cv2.destroyAllWindows() для закрытия всех открытых окон, созданных OpenCV.

На этом все! Полный листинг программы представлен ниже:

```
import cv2
import mediapipe as mp
import autopy
```

```
mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
cap = cv2.VideoCapture(0)
autopy.mouse.toggle(autopy.mouse.Button.LEFT, False)
with mp_hands.Hands(
    max_num_hands=1,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            print("Ignoring empty camera frame.")
            continue
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        results = hands.process(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.DrawingSpec(color=(255, 255, 255),
thickness=2, circle_radius=2),
                    mp_drawing.DrawingSpec(color=(0, 0, 255), thickness=2,
circle_radius=2),
                )
```

```
                index_finger =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP
]
                x1 = int(index_finger.x * image.shape[1])
                y1 = int(index_finger.y * image.shape[0])
                x2 = np.interp(x1, (frameR, wCam - frameR), (0, wScr))
                y2 = np.interp(y1, (frameR, hCam - frameR), (0, hScr))
                clocX = plocX + (x2 - plocX) / smoothening
                clocY = plocY + (y2 - plocY) / smoothening
                autopy.mouse.move(clocX, clocY)
        cv2.imshow('Virtual Mouse', image)
        if cv2.waitKey(5) & 0xFF == 27:
            break
    cap.release()
    cv2.destroyAllWindows()
```

2.2 Разработка жестов рук

После успешного создания виртуальной доски следующим шагом будет разработка жестов рук, которые можно использовать для работы с ней. Жесты рук — это способ взаимодействия с виртуальной доской без использования каких-либо физических инструментов, таких как мышь или клавиатура. Жесты рук могут использоваться для выполнения различных задач, таких как рисование, стирание, выбор различных инструментов или цветов. Разработка жестов рук включает в себя определение набора движений и положений рук, которые могут быть распознаны системой. Жесты рук могут быть распознаны с помощью традиционных методов компьютерного зрения, моделей глубокого обучения или их комбинации. Популярной библиотекой для распознавания жестов рук является библиотека MediaPipe, которая предоставляет предварительно обученные модели для отслеживания рук и распознавания жестов. Эта библиотека с открытым

исходным кодом предоставляет набор мощных конвейеров, которые разработчики могут использовать для создания передовых систем распознавания жестов рук. На рисунке 19 представлен обзор конвейера восприятия рук.

Рисунок 19 – Обзор конвейера восприятия рук

Конвейеры, включенные в библиотеку, предназначены для работы с алгоритмами компьютерного зрения для отслеживания движения рук пользователя и определения конкретных жестов в режиме реального времени.

2.2.1 Отслеживание рук

Для разработки жестов рук с помощью библиотеки MediaPipe необходимо сначала обнаружить и отследить ориентиры руки, которые представляют собой ключевые точки на руке, такие как кончики пальцев, ладонь и запястье. Поэтому сначала импортируем необходимые модули: cv2 для захвата и отображения видеопотоков и mediapipe для использования модуля отслеживания рук.

Листинг программы:

```
import cv2
import mediapipe as mp
```

Затем мы инициализируем модуль отслеживания рук, создав объект Hands. Мы также открываем видеопоток с помощью функции cv2.VideoCapture.

Листинг программы:

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
cap = cv2.VideoCapture(0)
```

В главном цикле мы считываем кадр из видеопотока и конвертируем его в формат RGB, ожидаемый модулем отслеживания рук MediaPipe. Затем мы обрабатываем изображение с помощью функции hands.process, которая обнаруживает руки и возвращает объект HandLandmark, содержащий ориентиры каждой обнаруженной руки.

Листинг программы:

```
while cap.isOpened():
  success, image = cap.read()
  if not success:
    break
  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  results = hands.process(image)
```

Если руки обнаружены, мы перебираем все обнаруженные объекты HandLandmark и рисуем ориентиры на изображении с помощью функции mp_drawing.draw_landmarks.

Листинг программы:

```
if results.multi_hand_landmarks:
  for hand_landmarks in results.multi_hand_landmarks:
    mp_drawing.draw_landmarks(
      image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
```

Рисунок 20 демонстрирует отслеживаемые трехмерные ориентиры рук.

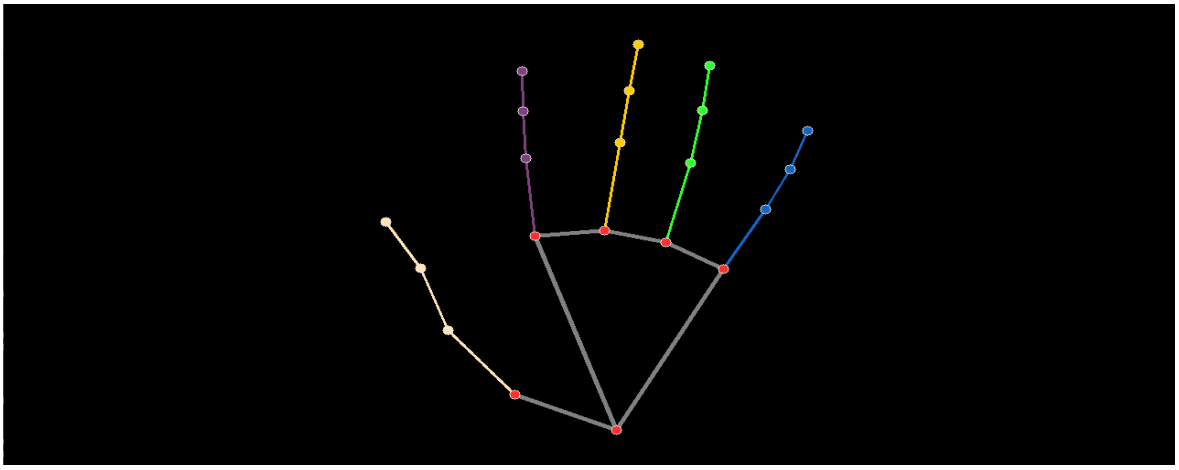


Рисунок 20 – Отслеживаемые трехмерные ориентиры рук

Для определения типа руки для каждого объекта HandLandmark мы используем атрибут multi_handedness переменной results. Атрибут multi_handedness представляет собой список объектов классификации, по одному для каждой обнаруженной руки, а индекс каждого объекта классификации соответствует индексу соответствующего объекта HandLandmark в списке multi_hand_landmarks. Мы извлекаем атрибут label первого объекта классификации в каждом списке multi_handedness, который указывает, является ли обнаруженная рука левой или правой. Затем мы рисуем на изображении ориентиры руки и метку руки с помощью функций mp_drawing.draw_landmarks и cv2.putText соответственно.

Листинг программы:

```
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        handedness = results.multi_handedness[
results.multi_hand_landmarks.index(hand_landmarks)].classification[0].
label
            mp_drawing.draw_landmarks(
              image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
            cv2.putText(image, handedness, (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
```

Отслеживаемые трехмерные ориентиры рук (тип руки) показаны на рисунке 21.

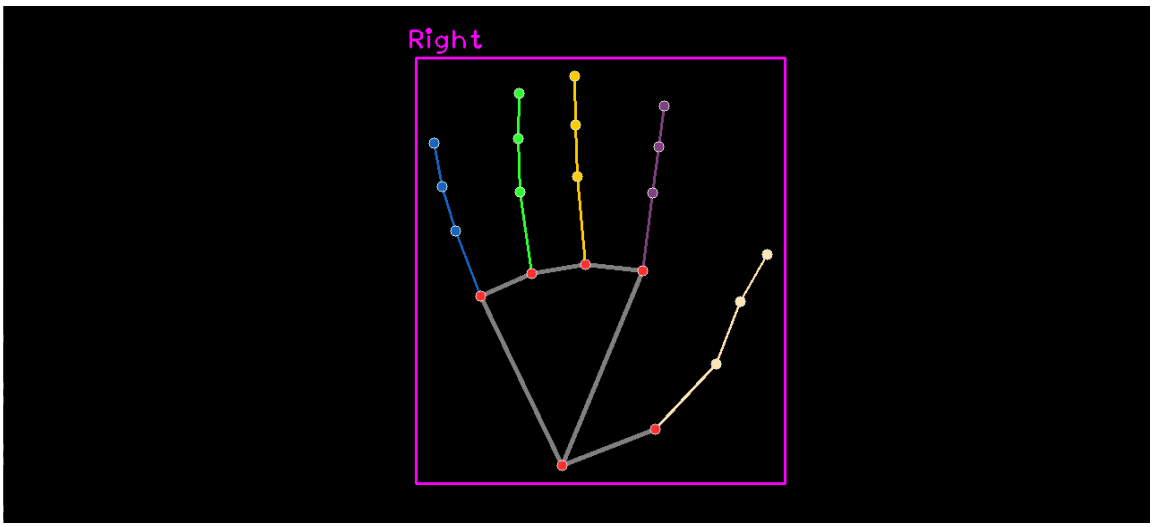


Рисунок 21 – Отслеживаемые трехмерные ориентиры рук (тип руки)

Наконец, мы выводим полученное изображение на экран с помощью функции cv2.imshow и ждем нажатия клавиши 'q' для выхода из цикла.

Полный листинг программы выглядит следующим образом:

```
import cv2
import mediapipe as mp
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()
mp_drawing = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)
while cap.isOpened():
    success, image = cap.read()
    if not success:
        break
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    results = hands.process(image)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            handedness = results.multi_handedness[
results.multi_hand_landmarks.index(hand_landmarks)].classification[0].
label
```

```
                mp_drawing.draw_landmarks(
                    image, hand_landmarks, mp_hands.HAND_CONNECTIONS)
                cv2.putText(image, handedness, (10, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        cv2.imshow('MediaPipe Hand Tracking', image)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

Обратите внимание, что нам также необходимо импортировать модуль mp_drawing для рисования ориентиров на изображении.

2.2.2 Жесты по умолчанию

Теперь, когда у нас есть возможность обнаруживать и отслеживать ориентиры на руке пользователя, мы можем использовать эту информацию для реализации жестов по умолчанию. Эти жесты аналогичны основным функциям компьютерной мыши. При использовании мыши нам не нужно разбираться в ее внутреннем устройстве. Мы просто знаем, что она поставляется с запрограммированными функциями, которые позволяют нам работать с ней без предварительной настройки. Однако для тех, кто хочет получить более продвинутые функции, существует возможность написания макросов. В нашей программе рука пользователя будет действовать как виртуальная мышь, которая поддерживает несколько основных функций, необходимых для правильной работы программы. Например, жест «Клик» может быть распознан как удержание на выбранном объекте кончика указательного пальца в течении нескольких миллисекунд, что напоминает нажатие левой кнопки компьютерной мыши. Эта виртуальная рука-мышь служит более естественным и интуитивно понятным интерфейсом для взаимодействия пользователей с программой. Она устраняет необходимость использования внешних периферийных устройств, таких как физическая

мышь, что делает ее более удобной для пользователя. Теперь вернёмся к нашим ориентирам, как упомяналось ранее, ориентиры рук, которые определяются и отслеживаются с помощью технологии компьютерного зрения, могут предоставить нам ценную информацию о точном положении и ориентации каждого из пальцев пользователя. Анализируя расположение ориентиров, мы можем с высокой точностью определить, в каком положении находятся наши пальцы (согнутый или выпрямленный палец). Для этого мы написали функцию с именем "fingersUp", которая принимает три аргумента: "results", "tipIds" и "myHand". Аргумент "results" - объектом, содержащим результаты алгоритма отслеживания рук, а "tipIds" - список целочисленных значений, представляющих ориентиры (в частности, ориентиры кончиков пальцев), которые соответствуют каждому пальцу руки. Наконец, "myHand" - объект, содержащий информацию об анализируемой руке, такую как ее тип (левая или правая) и положение ориентиров. Функция начинает работу с извлечения типа руки («Правая» или «Левая») и списка ориентиров ("myLmList") из словаря "myHand".

Листинг программы:

```
def fingersUp(results, tipIds, myHand):
    myHandType = myHand["type"]
    myLmList = myHand["lmList"]
```

Затем, если атрибут "multi_hand_landmarks" в объекте "results" равен True, функция переходит к анализу положения пальцев руки. Сначала функция проверяет положение большого пальца, сравнивая координаты ориентира кончика пальца и ориентира, расположенного непосредственно под ним. Если координата кончика пальца выше, чем координата ниже (для правой руки) или ниже (для левой руки), считается, что большой палец поднят, и в список "fingers" добавляется значение 1. В противном случае в список добавляется значение 0.

Листинг программы:

```
if results.multi_hand_landmarks:
    fingers = []
    if myHandType == "Right":
        if myLmList[tipIds[0]][0] > myLmList[tipIds[0] - 1][0]:
            fingers.append(1)
        else:
            fingers.append(0)
    else:
        if myLmList[tipIds[0]][0] < myLmList[tipIds[0] - 1][0]:
            fingers.append(1)
        else:
            fingers.append(0)
```

Затем функция перебирает оставшиеся четыре пальца и сравнивает координаты у их кончиков с координатами ориентиров, расположенных на две позиции ниже. Если координата у кончика пальца меньше, чем у соответствующего ориентира, расположенного двумя позициями ниже, считается, что палец поднят, и в список "fingers" добавляется значение 1. В противном случае в список добавляется значение 0.

Листинг программы:

```
for id in range(1, 5):
    if myLmList[tipIds[id]][1] < myLmList[tipIds[id] - 2][1]:
        fingers.append(1)
    else:
        fingers.append(0)
```

Наконец, функция возвращает список "fingers", который содержит 5 значений, представляющих, поднят или нет каждый палец (1 - поднят, 0 - не поднят).

Полный листинг программы выглядит так:

```
def fingersUp(results, tipIds, myHand):
```

```
            myHandType = myHand["type"]
            myLmList = myHand["lmList"]
            if results.multi_hand_landmarks:
                fingers = []
                if myHandType == "Right":
                    if myLmList[tipIds[0]][0] > myLmList[tipIds[0] - 1][0]:
                        fingers.append(1)
                    else:
                        fingers.append(0)
                else:
                    if myLmList[tipIds[0]][0] < myLmList[tipIds[0] - 1][0]:
                        fingers.append(1)
                    else:
                        fingers.append(0)
                for id in range(1, 5):
                    if myLmList[tipIds[id]][1] < myLmList[tipIds[id] - 2][1]:
                        fingers.append(1)
                    else:
                        fingers.append(0)
            return fingers
```

Если по-простому, то это выглядит следующим образом, мы берём окончания наших ориентиров под номерами [4, 8, 12, 16, 20].

На рисунке 22 представлен ориентир руки MediaPipe.

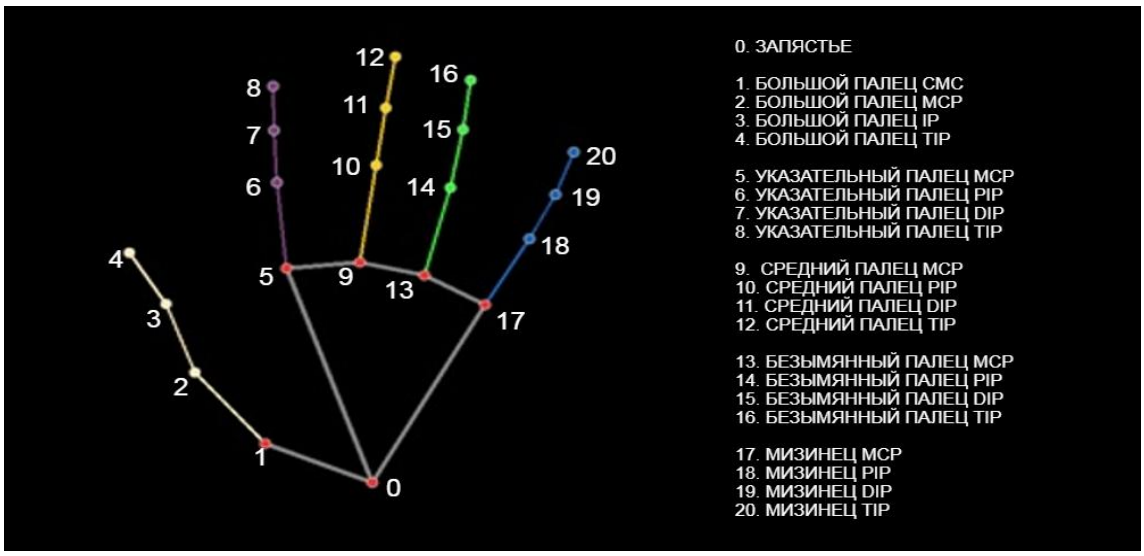


Рисунок 22 – Ориентир руки MediaPipe

И тот ориентир, который был поднят (выпрямленный палец) устанавливаем в единицу и наоборот который был опущен (согнутый палец) устанавливаем в ноль. Рисунок 23 демонстрирует ориентир руки (1- выпрямленный палец, 0 - согнутый палец).

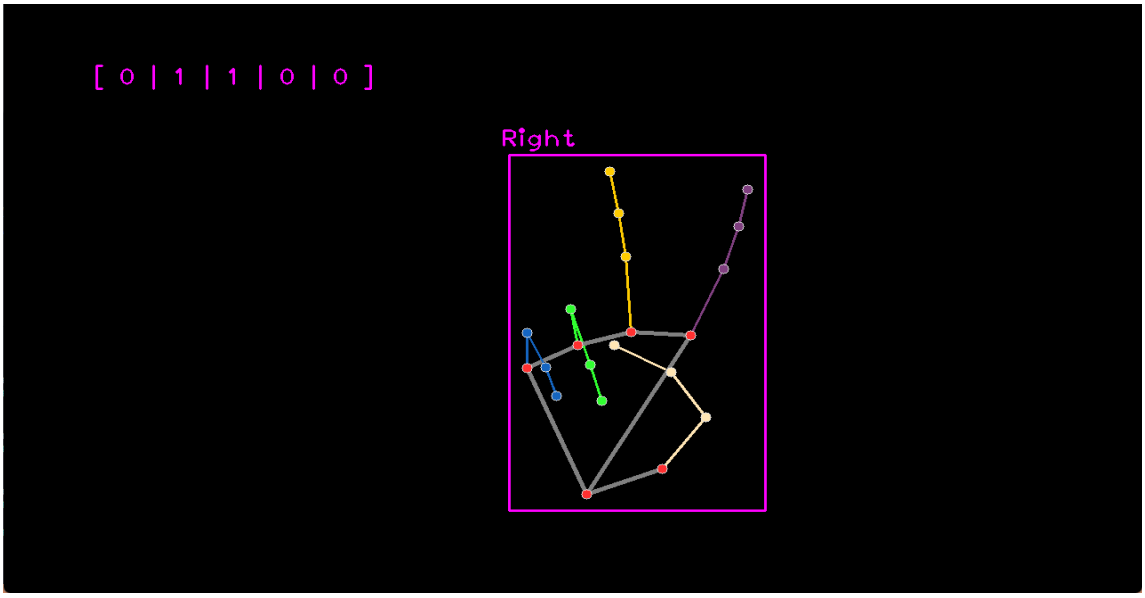


Рисунок 23 – Ориентир руки (1- выпрямленный палец, 0 - согнутый палец)

Для повышения точности и эффективности распознавания жестов мы применили механизм, использующий алгоритм поиска расстояния между двумя ключевыми ориентирами, реализовав функцию "findDistance", которая принимает два аргумента: "p1" и "p2", которые представляют собой

координаты двух точек (p1 = (x1,y1) и p2 = (x2,y2)) на изображении. В самой функции вычисляется центральная точка (cx, cy) отрезка прямой, соединяющего p1 и p2, путем взятия среднего значения x-координат и y-координат двух точек. Затем вычисляет евклидово расстояние между точками p1 и p2 с помощью функции "hypot" из модуля "math", которая берет разность координат x и y между двумя точками и вычисляет квадратный корень из их суммы квадратов. Функция создает кортеж "info", содержащий все координаты точек (x1, y1, x2, y2, cx, cy).

Листинг программы:

```
def findDistance(p1, p2, image=None):
    x1, y1 = p1
    x2, y2 = p2
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
    length = math.hypot(x2 - x1, y2 - y1)
    info = (x1, y1, x2, y2, cx, cy)
```

Если указан необязательный аргумент "image", функция также рисует окружности в точках p1, p2 и средней точке между ними, а также линию, соединяющую p1 и p2, используя библиотеку "cv2" (OpenCV).

Листинг программы:

```
if image is not None:
    cv2.circle(image, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
    cv2.circle(image, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
    cv2.line(image, (x1, y1), (x2, y2), (255, 0, 255), 3)
    cv2.circle(image, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
    return length, info, image
```

Если изображение нарисовано, функция возвращает кортеж, содержащий длину отрезка линии, кортеж "info" и измененное "image". Если изображение не нарисовано, функция просто возвращает кортеж длины и "info".

Полный листинг программы выглядит следующим образом:

```
def findDistance(p1, p2, image=None):
    x1, y1 = p1
    x2, y2 = p2
    cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
    length = math.hypot(x2 - x1, y2 - y1)
    info = (x1, y1, x2, y2, cx, cy)
    if image is not None:
        cv2.circle(image, (x1, y1), 15, (255, 0, 255), cv2.FILLED)
        cv2.circle(image, (x2, y2), 15, (255, 0, 255), cv2.FILLED)
        cv2.line(image, (x1, y1), (x2, y2), (255, 0, 255), 3)
        cv2.circle(image, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
        return length, info, image
    else:
        return length, info
```

Этот алгоритм, по сути, повторяет естественное движение карандаша, позволяя пользователю виртуально «брать» и «отпускать» карандаш, как это происходит в физическом мире. Отслеживая расстояния между этими ориентирами, мы можем определить начальную и конечную точки жеста рисования, что позволяет точно записать нарисованные линии или фигуры. Эта техника не только обеспечивает более реалистичное и интуитивно понятное рисование для пользователя, но и улучшает общее качество и четкость конечного результата. Поиск расстояния между двумя ориентирами показан на рисунке 24.

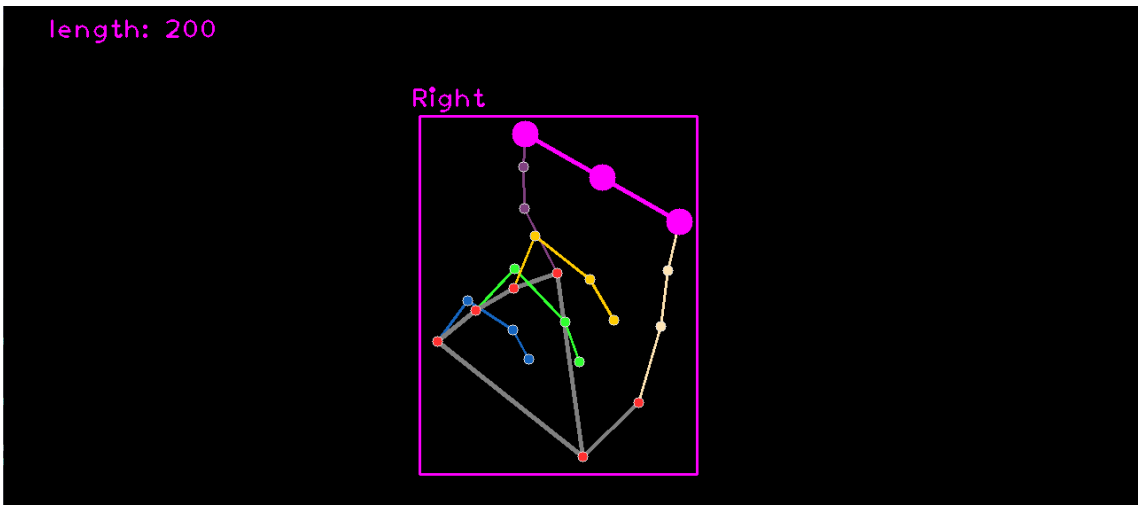


Рисунок 24 – Поиск расстояния между двумя ориентирами

Теперь осталось дело за малым, а именно интегрировать жесты рук в программу виртуальной доски. Одной из важнейших задач такой интеграции является разработка логики, которая позволяет жестам взаимодействовать с панелью инструментов. Чтобы добиться этого, необходимо реализовать логику взаимодействия жестов с каждым из инструментов, представленных на этой панели:

— перо,

— линия,

— квадрат,

— круг,

— ластик.

Начнём по порядку, сначала проработаем логику использования жестов для взаимодействия с инструментом «Перо».

Для этого проверим текущий инструмент.

Листинг программы:

```
if curr_tool == "draw":
```

Эта строка кода проверяет, выбран ли текущий инструмент «Перо». Если это не так, блок кода пропускается, и программа переходит к следующему набору инструкций.

Далее извлечём координаты ориентиров.

Листинг программы:

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
```

Этот код извлекает координаты ориентира для двух определенных точек на руке. Первая строка извлекает координаты x и y 12-й опорной точки и масштабирует их для соответствия изображению размером 640 на 480 пикселей. Вторая строка извлекает координату y 9-й опорной точки и масштабирует ее для соответствия тому же изображению.

Теперь проверим, поднят ли указательный палец.

Листинг программы:

```
if index_raised(yi, y9):
```

Эта строка кода вызывает функцию "index_raised", чтобы проверить, поднят ли указательный палец. Эта функция принимает два аргумента, yi и y9, которые являются y-координатами 12-й и 9-й опорных точек. Если функция возвращает True, это означает, что указательный палец поднят, и программа переходит к следующему шагу. В противном случае программа пропускает этот набор инструкций и переходит к блоку "else".

Если условие истинно, то нарисуем линию.

Листинг программы:

```
cv2.line(mask, (prevx, prevy), (x, y), 0, thick)
prevx, prevy = x, y
```

Если указательный палец поднят, программа рисует линию на изображении. Линия рисуется с помощью функции OpenCV "cv2.line", которая принимает пять аргументов: изображение для рисования (mask), начальную и конечную точки линии (prevx, prevy) и (x, y), цвет линии (0) и толщину линии (thick). Последняя строка этого блока кода обновляет предыдущие координаты до текущих, так что следующая нарисованная линия будет начинаться с конца предыдущей.

В противном случае производим обновление предыдущих координат.

Листинг программы:

```
        else:
            prevx = x
            prevy = y
```

Если указательный палец не поднят, программа обновляет предыдущие координаты до текущих. Это необходимо потому, что следующая нарисованная линия будет начинаться с обновленных координат.

Полный листинг программы выглядит следующим образом:

```
        if curr_tool == "draw":
            xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
            y9 = int(i.landmark[9].y*480)
            if index_raised(yi, y9):
                cv2.line(mask, (prevx, prevy), (x, y), 0, thick)
                prevx, prevy = x, y
            else:
                prevx = x
                prevy = y
```

На рисунке 25 представлен инструмент «Перо».

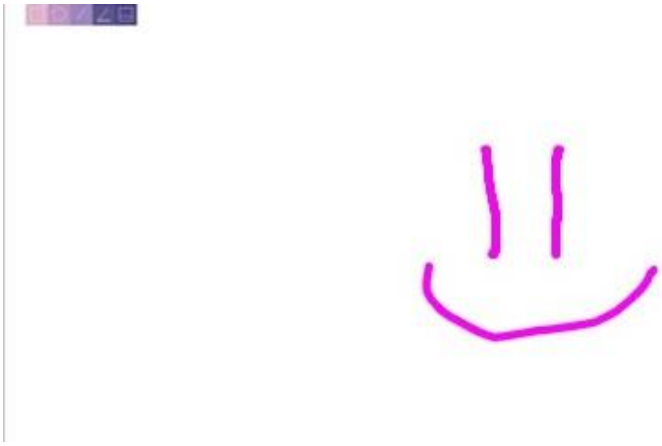


Рисунок 25 – Инструмент «Перо»

Теперь сосредоточимся на разработке логики использования жестов для взаимодействия с инструментом «Линия».

Как и с предыдущем инструментом, первым делом проверяем, является ли текущий инструмент инструментом «Линия».

Листинг программы:

```
elif curr_tool == "line":
```

Этот код вложен в структуру if-elif, которая проверяет текущий инструмент, используемый пользователем. Если текущим инструментом является «Линия», то будет выполнен код в этом блоке.

Далее получаем координаты двух ориентиров на руке.

Листинг программы:

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
```

Этот код извлекает координаты x и y двух ориентиров на руке с помощью библиотеки MediaPipe. Ориентиры идентифицируются по их индексному номеру в атрибуте landmark обнаруженной руки. Функция int используется для преобразования дробных координат в целые числа, которые могут быть использованы в качестве координат пикселей в изображении. Значения 640 на 480 - ширина и высота обрабатываемого изображения или видеопотока.

И проверяем, поднят ли указательный палец.

Листинг программы:

```
if index_raised(yi, y9):
```

Этот код вызывает функцию index_raised, которая, проверяет, поднят ли указательный палец, основываясь на координатах y ориентиров, извлеченных на предыдущем шаге. Если указательный палец поднят, будет выполнен код в блоке if.

Если указательный палец поднят, то проведём линию на изображении.

Листинг программы:

```
if not (var_inits):
   xii, yii = x, y
```

```
            var_inits = True
        cv2.line(frm, (xii, yii), (x, y), (50, 152, 255), thick)
```

Этот код проверяет, не является ли переменная var_inits значением False. Если она False, то инициализирует две переменные xii и yii текущими координатами x и y руки, соответственно, и устанавливает var_inits в True. Это делается для того, чтобы начать новый сегмент линии, когда пользователь впервые поднимает указательный палец. Затем вызывается функция cv2.line, чтобы нарисовать сегмент линии на изображении, используя начальные координаты (xii, yii) и текущие координаты (x, y) руки. Кортеж (50, 152, 255) задает цвет линии в формате BGR, а thick - толщину линии.

Иначе, если указательный палец не поднимался, то прекратить рисовать линию.

Листинг программы:

```
        else:
            if var_inits:
                cv2.line(mask, (xii, yii), (x, y), 0, thick)
                var_inits = False
```

Этот код выполняется, если указательный палец не поднят. Он проверяет, имеет ли var_inits значение True, что означает, что пользователь ранее поднял указательный палец и начал рисовать сегмент линии. Если var_inits равен True, код использует функцию cv2.line для рисования последнего сегмента линии на отдельном изображении под названием mask. Начальные координаты - (xii, yii), а текущие координаты - (x, y). Цвет линии устанавливается равным 0, то есть черным, а значение thick задает толщину линии. Наконец, `var_inits устанавливается обратно в False, чтобы показать, что линия завершена.

Полный листинг программы выглядит следующим образом:

```
        elif curr_tool == "line":
```

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
if index_raised(yi, y9):
    if not (var_inits):
        xii, yii = x, y
        var_inits = True
    cv2.line(frm, (xii, yii), (x, y), (50, 152, 255), thick)
else:
    if var_inits:
        cv2.line(mask, (xii, yii), (x, y), 0, thick)
        var_inits = False
```

Рисунок 26 демонстрирует инструмент «Линия».

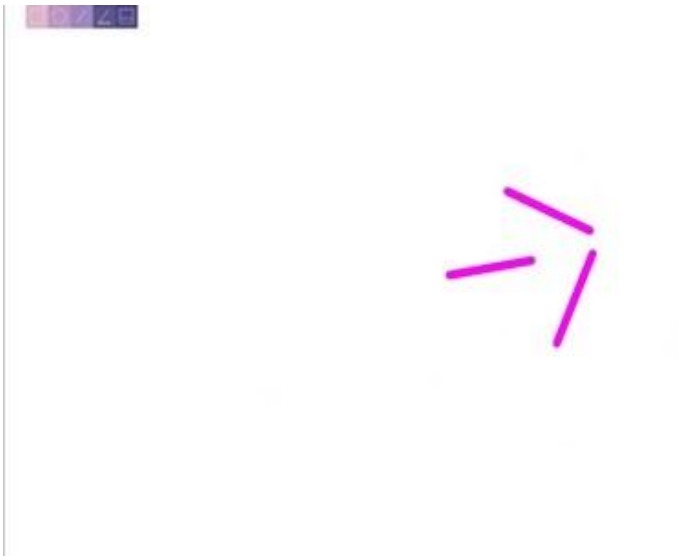


Рисунок 26 – Инструмент «Линия»

Следующим шагом будет разработка логики использования жестов для взаимодействия с инструментом «Квадрат».

Начнём как всегда с небольшой проверки, а именно оператора elif. Это означает, что код в этом блоке будет выполнен только в том случае, если не было выполнено условие, указанное в предыдущем(их) операторе(ах) if или elif. В этом блоке проверяется, равна ли переменная curr_tool строке "rectangle". Если это так, то код в этом блоке будет выполнен.

Листинг программы:

```
elif curr_tool == "rectangle":
```

Следующие две строки кода, используют функцию int() для преобразования чисел с плавающей точкой в целые числа. Эти строки извлекают координаты x и y двух ориентиров на руке, которая отслеживает программа. Эти ориентиры идентифицируются по их индексам (12 и 9, соответственно) в объекте i.landmark.

Листинг программы:

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
```

После этого вызываем функцию index_raised(), которая принимает два аргумента (переменные yi и y9) и возвращает булево значение, указывающее, поднят ли указательный палец или нет.

Листинг программы:

```
if index_raised(yi, y9):
```

Если функция index_raised() возвращает значение True, то выполняется код в блоке if. Первое, что делает этот блок кода, - проверяет, является ли булева переменная var_inits значением False. Если это так, то текущие координаты x и y (x и y) присваиваются двум новым переменным xii и yii соответственно, а var_inits устанавливается в True.

Листинг программы:

```
if not (var_inits):
    xii, yii = x, y
    var_inits = True
```

После этого вызываем функцию cv2.rectangle(), которая является частью библиотеки OpenCV. Эта функция рисует прямоугольник на изображении (в данном случае на изображении frm), используя указанные координаты и цвет. Кортеж (0, 255, 255) задает желтый цвет, а thick - переменная, которая определяет толщину границы прямоугольника.

Листинг программы:

```
cv2.rectangle(frm, (xii, yii), (x, y), (0, 255, 255), thick)
```

Если функция index_raised() возвращает False, то выполняется код в блоке else. Этот блок сначала проверяет, является ли var_inits значением True. Если да, то на изображении mask рисуется прямоугольник с теми же координатами, что и раньше, но с цветом 0 (что означает черный) и той же толщиной, что и раньше. Наконец, var_inits снова устанавливается в False.

Листинг программы:

```
else:
    if var_inits:
        cv2.rectangle(mask, (xii, yii), (x, y), 0, thick)
        var_inits = False
```

Вот и все. Полный листинг программы выглядит следующим образом:

```
elif curr_tool == "rectangle":
    xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
    y9 = int(i.landmark[9].y*480)
    if index_raised(yi, y9):
        if not (var_inits):
            xii, yii = x, y
            var_inits = True
        cv2.rectangle(frm, (xii, yii), (x, y), (0, 255, 255), thick)
    else:
        if var_inits:
            cv2.rectangle(mask, (xii, yii), (x, y), 0, thick)
            var_inits = False
```

Инструмент «Квадрат» показан на рисунке 27.

Рисунок 27 – Инструмент «Квадрат»

Наконец пришло время реализовать логику для жестового взаимодействия с инструментом «Круг».

Знакомый всем блок elif, проверяет определенное условие, которое не выполняется предыдущим оператором if. Здесь проверяется, равна ли переменная curr_tool значению "circle".

Листинг программы:

```
elif curr_tool == "circle":
```

Внутри блока elif код инициализирует три переменные: xi, yi и y9. Эти переменные устанавливаются на основе координат определенных ориентиров, обнаруженных в видеопотоке.

Листинг программы:

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
```

Вызывается функция index_raised с аргументами yi и y9. Она возвращает булево значение, определяющее, поднят или опущен указательный палец.

Листинг программы:

```
if index_raised(yi, y9):
```

Если index_raised возвращает True, код проверяет, является ли var_inits значением False. Если да, то код устанавливает значения xii и yii в x и y, соответственно, и устанавливает var_inits в True.

Листинг программы:

```
if not (var_inits):
    xii, yii = x, y
    var_inits = True
```

Если var_inits равен True, код рисует круг на изображении frm с помощью функции cv2.circle. Центр окружности находится в точке (xii, yii), а радиус рассчитывается на основе расстояния между (xii, yii) и (x, y) с помощью теоремы Пифагора. Цвет окружности - (255, 255, 0) (то есть желтый), а thick - толщина.

Листинг программы:

```
cv2.circle(frm, (xii, yii), int(((xii-x)**2 + (yii-y)**2)**0.5),
 (255, 255, 0), thick)
```

Если index_raised возвращает False, код проверяет, равно ли var_inits True. Если да, то код рисует круг на изображении маски с помощью функции cv2.circle. Центр окружности находится в точке (xii, yii), а радиус вычисляется на основе расстояния между (xii, yii) и (x, y) с помощью теоремы Пифагора. Цвет круга - (0, 255, 0) (это зеленый), а толщина - thick. Наконец, код устанавливает var_inits в False.

Листинг программы:

```
else:
    if var_inits:
        cv2.circle(mask, (xii, yii), int(((xii-x)**2 + (yii-y)**2)**0.5),
         (0, 255, 0), thick)
        var_inits = False
```

На этом всё. Полный листинг программы выглядит следующим образом:

```
elif curr_tool == "circle":
```

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
if index_raised(yi, y9):
    if not (var_inits):
        xii, yii = x, y
        var_inits = True
    cv2.circle(frm, (xii, yii), int(
        ((xii-x)**2 + (yii-y)**2)**0.5), (255, 255, 0), thick)
else:
    if var_inits:
        cv2.circle(mask, (xii, yii), int(
            ((xii-x)**2 + (yii-y)**2)**0.5), (0, 255, 0), thick)
        var_inits = False
```

На рисунке 28 представлен инструмент «Круг».

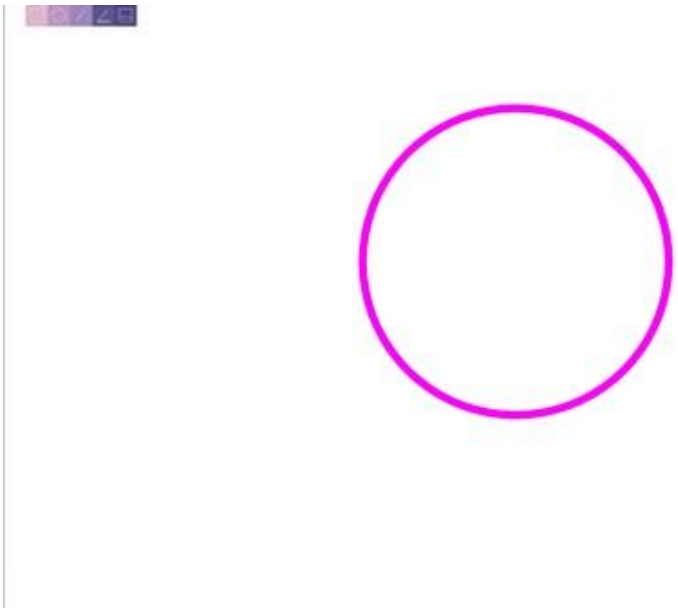


Рисунок 28 – Инструмент «Круг»

Ну и напоследок осталось разработать логику использования жестов для работы с инструментом «Ластик», который является последним инструментом в списке.

Поэтому снова выполняем проверку текущего инструмента.

Листинг программы:

```
elif curr_tool == "erase":
```

Эта строка кода проверяет, является ли текущий используемый инструмент инструментом «Ластик». Если это так, то будет выполнен код в операторе if.

Следующим шагом получаем координаты x и y определенных ориентиров на руке.

Листинг программы:

```
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)
```

Этот код извлекает координаты x и y двух определенных ориентиров на отслеживаемой руке. Затем он сохраняет эти значения в переменных xi, yi и y9. Эти ориентиры соответствуют определенным точкам на руке, которые будут использоваться для определения необходимости применения инструмента «Ластик».

Далее проверяем, поднят ли указательный палец.

Листинг программы:

```
if index_raised(yi, y9):
```

Эта строка кода вызывает функцию index_raised(), передавая координаты y ориентиров для указательного пальца. Эта функция, проверяет, поднят ли указательный палец или нет.

Если условие истинно, то применяем инструмент «Ластик».

Листинг программы:

```
cv2.circle(frm, (x, y), 30, (0, 0, 0), -1)
cv2.circle(mask, (x, y), 30, 255, -1)
```

Если поднять указательный палец, будет выполнен этот код. Этот код рисует две окружности на обрабатываемом изображении, эффективно применяя инструмент «Ластик». Первая окружность рисуется - на frm, а

вторая - на mask. Параметры, передаваемые функции cv2.circle(), определяют положение, размер и цвет круга.

Полный листинг программы выглядит следующим образом:

```
elif curr_tool == "erase":
    xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
    y9 = int(i.landmark[9].y*480)
    if index_raised(yi, y9):
        cv2.circle(frm, (x, y), 30, (0, 0, 0), -1)
        cv2.circle(mask, (x, y), 30, 255, -1)
```

Рисунок 29 демонстрирует инструмент «Ластик».

Рисунок 29 – Инструмент «Ластик»

Внедрение вышеописанных функции распознавания жестов рук позволяют пользователю эффективно и качественно выполнять различные действия на виртуальной доске, такие как рисование линий или фигур, стирание, изменение цвета или толщины инструмента рисования, а также доступ к другим интерактивным функциям. Рукописное изображение показано на рисунке 30.

Рисунок 30 – Рукописное изображение

2.3 Выводы по главе

В этом разделе дается подробное объяснение различных компонентов, составляющих внутренний механизм приложения. В частности, рассматривается поиск и отслеживание рук, что является неотъемлемым аспектом работы приложения. Также необходимо отметить важность жестов по умолчанию для обеспечения работы приложения. Без этих жестов программа не смогла бы функционировать. Кроме того, в данном разделе рассматривается реализация виртуальной доски, которая включает в себя всю её логистику.

ЗАКЛЮЧЕНИЕ

В рамках магистерской диссертации была разработана виртуальная доска, оснащенная различными инструментами, которыми можно управлять с помощью жестов рук. Программа непрерывно снимает видео с камеры, а алгоритмы обработки изображений применяются для обнаружения движений рук пользователя. Определенные жесты рук используются для инициирования определенных действий на виртуальной доске, позволяя пользователям рисовать, стирать и выполнять другие манипуляции с помощью естественных жестов рук. Что позволяет пользователям взаимодействовать с виртуальной доской плавно и интуитивно, без необходимости использования дополнительных инструментов или оборудования.

В процессе разработки программы были учтены все критерии, указанные в техническом задании. Для обеспечения соответствия требованиям был проведен процесс тщательного тестирования созданного программного обеспечения. Тестирование было направлено на обеспечение соответствия программы всем спецификациям и стандартам, изложенным в техническом задании. В результате программа была успешно разработана и полностью соответствует техническому заданию.

СПИСОК ЛИТЕРАТУРЫ

1. Fast Tracking of Hand and Finger - интернет ресурс. URL: http://vcai.mpi-inf.mpg.de/files/MPI-I-2014-4-002.pdf (Дата обращения: 06.04.2023).

2. Hand tracking and Gesture Recognition - интернет ресурс. URL: https://www.intelrealsense.com/hand-tracking-overview/ (Дата обращения: 14.03.2023).

3. Hand Recognition Using Depth Cameras - интернет ресурс. URL: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1909-36672015000200011 (Дата обращения: 27.03.2023).

4. Hand Tracking and Gesture Recognition Using Lensless Smart Sensors - интернет ресурс. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6164357/ (Дата обращения: 07.05.2023).

5. MediaPipe Hands - интернет-ресурс. URL: https://google.github.io/mediapipe/solutions/hands.html (Дата обращения: 12.04.2023).

6. On-Device, Real-Time Hand Tracking with MediaPipe - интернет ресурс. URL: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html (Дата обращения: 09.04.2023).

7. Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow - интернет ресурс. URL: https://github.com/victordibia/handtracking (Дата обращения: 18.03.2023).

8. How to Build a Real-time Hand-Detector using Neural Networks (SSD) on Tensorflow - интернет ресурс. URL: https://medium.com/@victor.dibia/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce (Дата обращения: 21.05.2023).

ПРИЛОЖЕНИЕ А Листинги программы

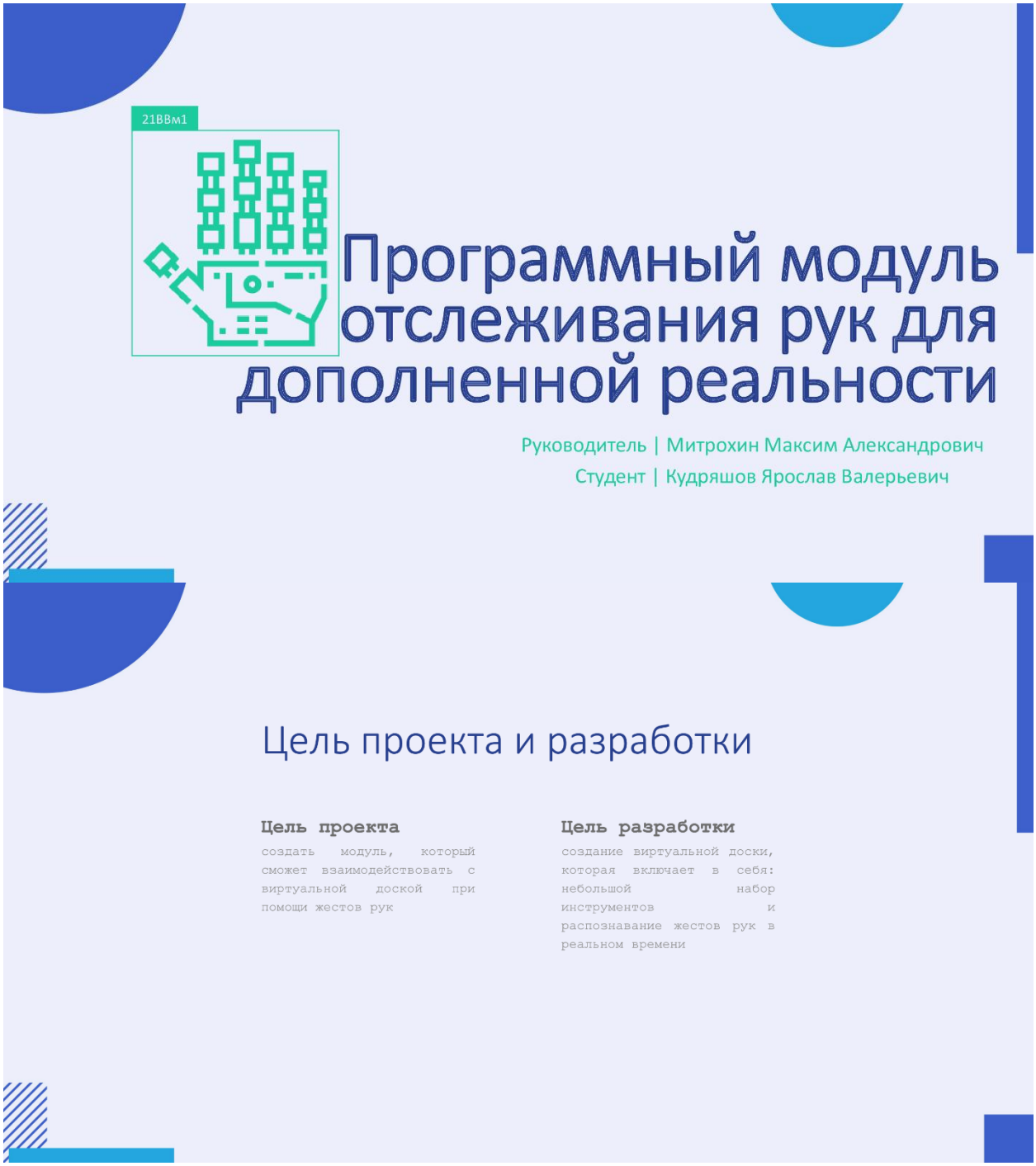
```
import mediapipe as mp
import cv2
import numpy as np
import time

# contants
ml = 150
max_x, max_y = 250+ml, 50
curr_tool = "select tool"
time_init = True
rad = 40
var_inits = False
thick = 4
prevx, prevy = 0, 0

# get tools function
def getTool(x):
if x < 50 + ml:
return "draw"
elif x < 100 + ml:
return "erase"
elif x < 150 + ml:
return "line"
elif x < 200 + ml:
return "circle"
else:
return "rectangle"

def index_raised(yi, y9):
if (y9 - yi) > 40:
return True
return False

hands = mp.solutions.hands
hand_landmark = hands.Hands(
min_detection_confidence=0.6, min_tracking_confidence=0.6,
max_num_hands=1)
draw = mp.solutions.drawing_utils

# drawing tools
tools = cv2.imread("tools.png")
```

```
tools = tools.astype('uint8')

mask = np.ones((480, 640))*255
mask = mask.astype('uint8')
cap = cv2.VideoCapture(0)

while True:
_, frm = cap.read()
frm = cv2.flip(frm, 1)
rgb = cv2.cvtColor(frm, cv2.COLOR_BGR2RGB)
op = hand_landmark.process(rgb)

if op.multi_hand_landmarks:
for i in op.multi_hand_landmarks:
draw.draw_landmarks(frm, i, hands.HAND_CONNECTIONS)
x, y = int(i.landmark[8].x
*640), int(i.landmark[8].y*480)

if x < max_x and y < max_y and x > ml:
if time_init:
ctime = time.time()
time_init = False
ptime = time.time()

cv2.circle(frm, (x, y), rad, (0, 255, 255), 2)
rad -= 1

if (ptime - ctime) > 0.8:
curr_tool = getTool(x)
print("your current tool set to : ", curr_tool)
time_init = True
rad = 40

else:
time_init = True
rad = 40

if curr_tool == "draw":
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)

if index_raised(yi, y9):
cv2.line(mask, (prevx, prevy), (x, y), 0, thick)
```

```
prevx, prevy = x, y

else:
prevx = x
prevy = y

elif curr_tool == "line":
xi, yi = int(i.landmark[12].x
*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)

if index_raised(yi, y9):
if not (var_inits):
xii, yii = x, y
var_inits = True

cv2.line(frm, (xii, yii), (x, y),
(50, 152, 255), thick)

else:
if var_inits:
cv2.line(mask, (xii, yii), (x, y), 0, thick)
var_inits = False

elif curr_tool == "rectangle":
xi, yi = int(i.landmark[12].x
*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)

if index_raised(yi, y9):
if not (var_inits):
xii, yii = x, y
var_inits = True

cv2.rectangle(frm, (xii, yii),
(x, y), (0, 255, 255), thick)

else:
if var_inits:
cv2.rectangle(mask, (xii, yii), (x, y), 0, thick)
var_inits = False

elif curr_tool == "circle":
```

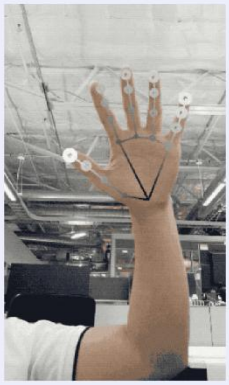
```
xi, yi = int(i.landmark[12].x
*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)

if index_raised(yi, y9):
if not (var_inits):
xii, yii = x, y
var_inits = True

cv2.circle(frm, (xii, yii), int(
((xii-x)**2 + (yii-y)**2)**0.5), (255, 255, 0), thick)

else:
if var_inits:
cv2.circle(mask, (xii, yii), int(
((xii-x)**2 + (yii-y)**2)**0.5), (0, 255, 0), thick)
var_inits = False

elif curr_tool == "erase":
xi, yi = int(i.landmark[12].x*640), int(i.landmark[12].y*480)
y9 = int(i.landmark[9].y*480)

if index_raised(yi, y9):
cv2.circle(frm, (x, y), 30, (0, 0, 0), -1)
cv2.circle(mask, (x, y), 30, 255, -1)

op = cv2.bitwise_and(frm, frm, mask=mask)
frm[:, :, 1] = op[:, :, 1]
frm[:, :, 2] = op[:, :, 2]

frm[:max_y, ml:max_x] = cv2.addWeighted(
tools, 0.7, frm[:max_y, ml:max_x], 0.3, 0)

cv2.putText(frm, curr_tool, (270+ml, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
cv2.imshow("paint app", frm)

if cv2.waitKey(1) == 27:
cv2.destroyAllWindows()
cap.release()
break
```

ПРИЛОЖЕНИЕ Б Презентация



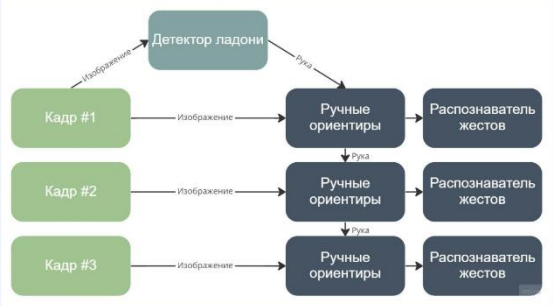
Модель отслеживания рук для дополненной реальности

Отслеживание рук в реальном времени с помощью MediaPipe



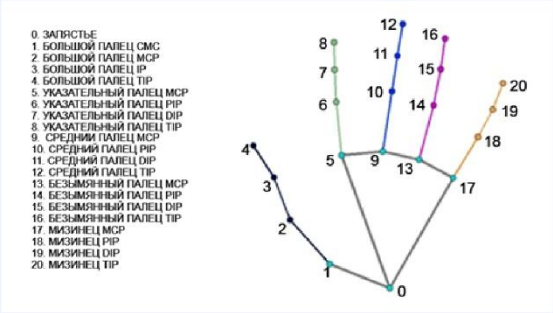
Отслеживаемые трехмерные ориентиры рук

Отслеживание рук в реальном времени с помощью MediaPipe



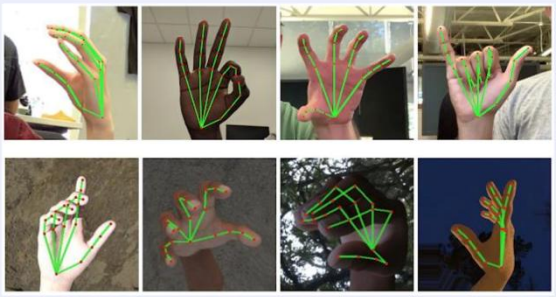
Конвейер восприятия рук

Отслеживание рук в реальном времени с помощью MediaPipe



Ручные ориентиры

Отслеживание рук в реальном времени с помощью MediaPipe



Изображения рук с аннотациями

Отслеживание рук в реальном времени с помощью MediaPipe

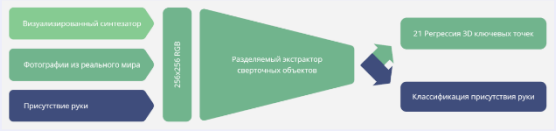
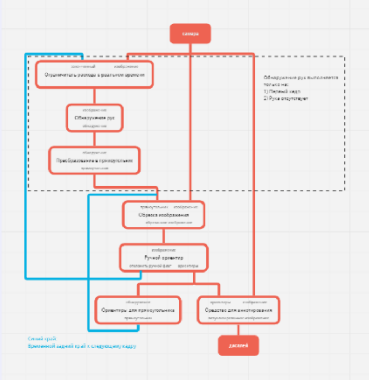


Схема смешанного обучения

Отслеживание рук в реальном времени с помощью MediaPipe

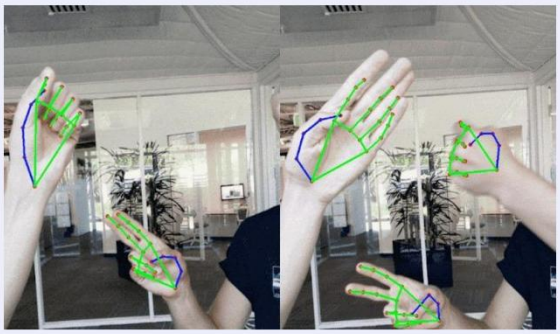
«Рок», «Человек-паук» и «ОК»

Отслеживание рук в реальном времени с помощью MediaPipe



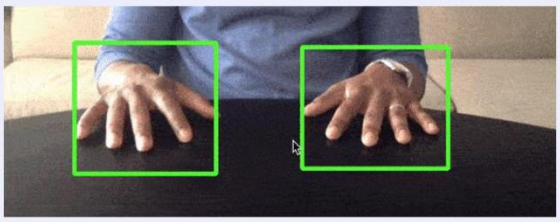
Отслеживание рук MediaPipe

Отслеживание рук в реальном времени с помощью MediaPipe



Надежное и устойчивое отслеживание

Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей



Обнаружение в реальном времени видеопотока с веб-камеры

Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей

Модель детектора рук из набора данных Egohands

Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей

```
-- LOCATION_X
  -- frame_1.jpg
  -- frame_2.jpg
  ...
  -- frame_100.jpg
  -- polygons.mat  // содержит аннотации ко всем 100 изображениям в текущей папке
-- LOCATION_Y
  -- frame_1.jpg
  -- frame_2.jpg
  ...
  -- frame_100.jpg
  -- polygons.mat  // содержит аннотации ко всем 100 изображениям в текущей папке
```

Набор данных Egohands

Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей

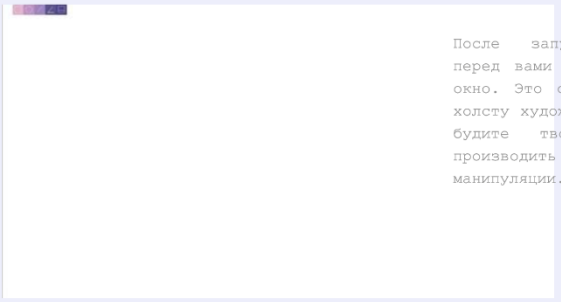
График обучения ручного детектора

Отслеживание рук в реальном времени с использованием Tensorflow и нейронных сетей

Работа детектора на случайных фотографиях

Инструкции

Виртуальная доска



После запуска программы перед вами появиться белое окно. Это окно равносильно холсту художника, на нём вы будите творить или же производить разного рода манипуляции.

Главное окно программы

Инструкции

Виртуальная доска

Обратите внимание на верхний левый угол главного окна программы, там расположена панель с инструментами.

Всего на этой панели пять инструментов:

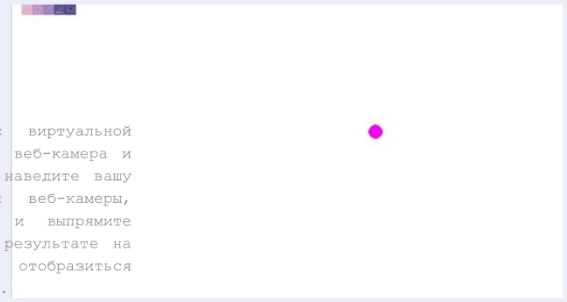
Для полноценной работы вам потребуются только два из них («Перо» и «Ластик»). Остальные инструменты являются вспомогательными («Квадрат» «Круг» и «Линия»), так как от руки трудно рисовать ровные фигуры.

Панель инструментов

Инструкции

Взаимодействие с виртуальной доской при помощи жестов рук

Для взаимодействия с виртуальной доской вам потребуется веб-камера и рука человека. Просто наведите вашу руку в поле зрения веб-камеры, сожмите её в кулак и выпрямите указательный палец, в результате на белом окне (холсте), отобразиться виртуальный курсор мыши.



Виртуальный курсор мыши

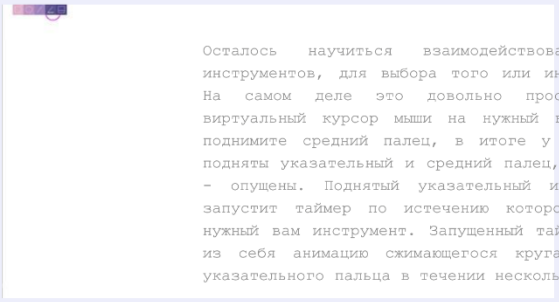
Инструкции

Взаимодействие с виртуальной доской при помощи жестов рук

Перемещение виртуального курсора мыши

Инструкции

Взаимодействие с виртуальной доской при помощи жестов рук

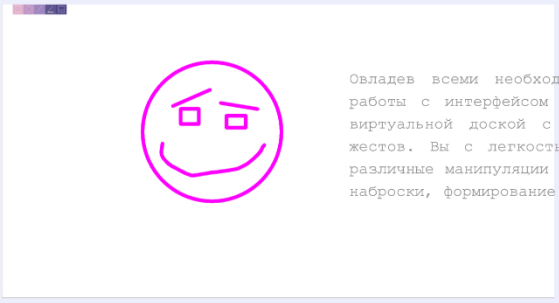


Осталось научиться взаимодействовать с панелью инструментов, для выбора того или иного инструмента. На самом деле это довольно просто, переместите виртуальный курсор мыши на нужный вам инструмент и поднимите средний палец, в итоге у вас должны быть подняты указательный и средний палец, а все остальные - опущены. Поднятый указательный и средний палец, запустит таймер по истечению которого будет выбран нужный вам инструмент. Запущенный таймер представляет из себя анимацию сжимающегося круга вокруг кончика указательного пальца в течении нескольких миллисекунд.

Взаимодействие виртуального курсора мыши с панелью инструментов

Инструкции

Взаимодействие с виртуальной доской при помощи жестов рук



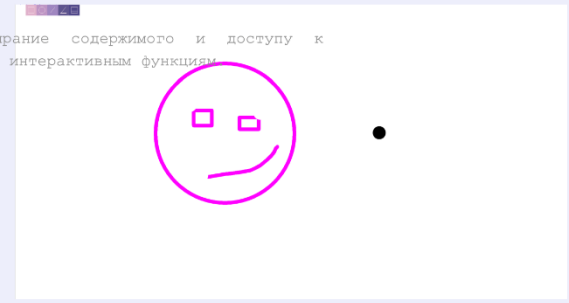
Овладев всеми необходимыми навыками для работы с интерфейсом и взаимодействию с виртуальной доской с помощью физических жестов. Вы с легкостью можете выполнять различные манипуляции на доске, такие как наброски, формирование фигур…

Набросок «Эмодзи»

Инструкции

Взаимодействие с виртуальной доской при помощи жестов рук

… стирание содержимого и доступу к другим интерактивным функциям

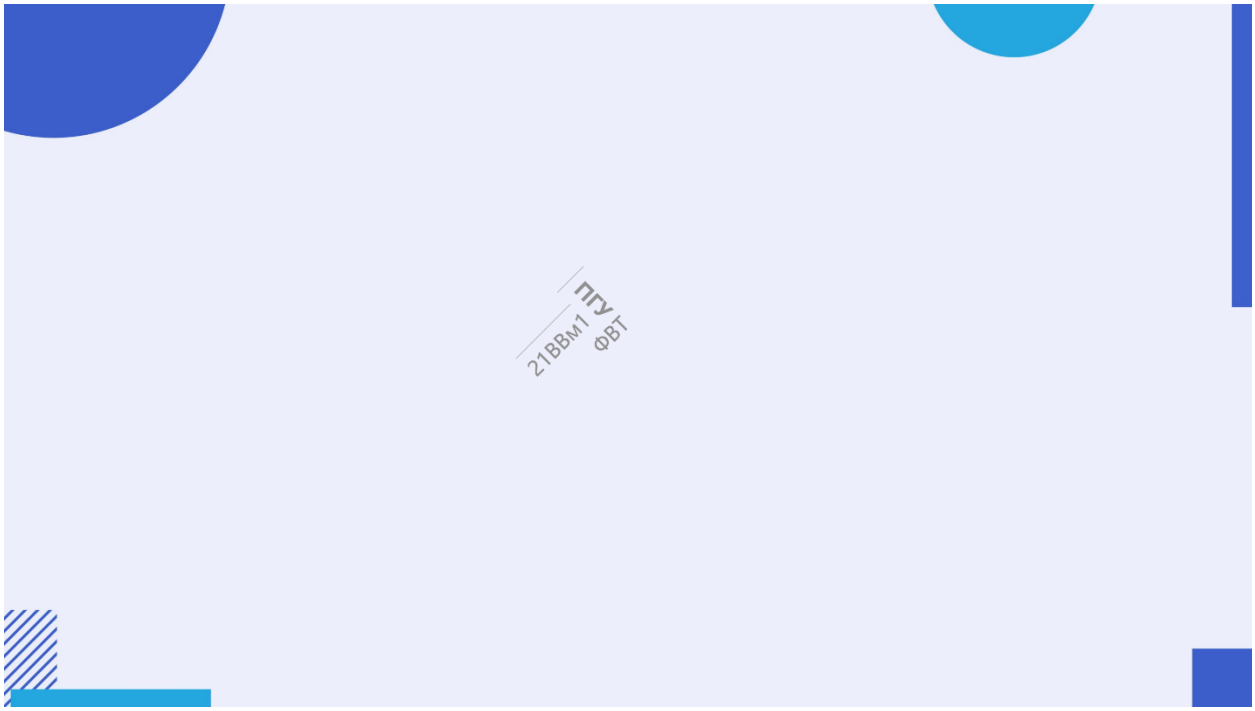


Стирание фрагмента «Эмодзи»

Подведение итогов

Целью магистерской диссертации было изучение технологии дополненной реальности, с особым акцентом на разработку программы виртуальной доски, использующей компьютерное зрение и распознавание жестов рук:

- разработана программа виртуальной доски
- разработан модуль для взаимодействия с виртуальной доской при помощи жестов рук


ПГУ
ФВТ
21ВВм1