Министерство образования и науки Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой
_____Г.И. Радченко
«___»_____2018 г.

Разработка программного комплекса для анализа моторно-двигательной функции и сопровождения реабилитации больных, перенесших инсульт

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУРГУ-090301.2018.131 ПЗ ВКР

Руководитель работы,
к.т.н., доцент
«Электронные
вычислительные машины»
_____И.Л. Надточий
«___»_____2018 г.

Автор работы
студент группы КЭ-484
_____В.А. Синеглазов
«___»_____2018 г.

Нормоконтролёр, ст. преп. каф.
«Электронные вычислительные
машины»
_____В.В. Лурье
«___»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Синеглазов В.А. Разработка программного комплекса для анализа моторно-двигательной функции и сопровождения реабилитации больных, перенесших инсульт. – Челябинск: ЮУрГУ, ВШЭКН; 2018, 83 с., 35 ил., библиогр. список – 10 наим., 1 прил.

В данной выпускной квалификационной работе выполнена разработка программного обеспечения, позволяющего анализировать моторно-двигательные функции, а также осуществлять проведение реабилитационных упражнений для пациентов, перенесших инсульт. В ходе выполнения был произведен обзор классических и информационно технологических методов реабилитации, также рассмотрены методы анализа и распознавания крупной и мелкой моторики. Разработан алгоритм распознавания руки в изображении глубины, её анализ и выделение основных параметров. Создана программная оболочка, включающая реабилитационные упражнения и систему мониторинга динамических показателей человека.

Пояснительная записка включает в себя введение, оглавление, основную часть, заключение и список литературы.

Оглавление

ВВЕДЕНИЕ

Актуальность

В современном мире человек все чаще сталкивается с серьезными проблемами со здоровьем, который приводят к необратимым последствиям. Одним из таких грозных заболеваний является инсульт. Инсульт – это острое нарушение мозгового кровообращения, с повреждением ткани мозга и расстройством его функций. За последние годы зафиксирован рост сосудистых заболеваний. Ежегодно в России инсульт переносят около 450 тыс. человек, а в мире порядка 6 млн. В России инсульт занимает 3-е место по причинам смертности, но даже при благоприятном исходе человеку редко удается вернуться к привычному образу жизни. Одним из основных расстройств является нарушение опорно-двигательных функций. Поэтому важнейшую роль играет своевременная помощь, а также комплексная реабилитация больного, для его успешной социальной адаптации.

Цели исследования

Ознакомившись с методиками реабилитации больных с нарушением мелкой и крупной моторики, следует разработать реабилитационный комплекс, который будет включать в себя:

1. Набор реабилитационных игр.
2. Способность распознавать мелкую и крупную моторику.
3. Систему мониторинга.
4. Интерактивное (бесконтактное) взаимодействие.

Задачи:

1. Проведение анализа требований к системе.
2. . Исследование классических методов реабилитации.
3. . Исследование методов распознавания жестов.
4. . Исследование технических средств.
5. . Разработка игр, направленных на реабилитацию.

6. . Разработка алгоритма распознавания жестов.

7. . Разработка системы мониторинга.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор существующих решений

Основным методом реабилитации инсультных больных с нарушениями движений является лечебная физкультура (кинезотерапия), в задачи которой входит восстановление (полное или частичное) объема движений, силы и ловкости в пораженных конечностях, функции равновесия, навыков самообслуживания. Для этого применяются классические средства реабилитации:

- физиотерапия;
- трудотерапия;
- лечебная физкультура;
- массаж;
- иглорефлексотерапия.

Несмотря на положительные результаты, а также накопленный с годами опыт классических методов реабилитации, существует ряд недостатков:

- низкая доступность;
- высокая стоимость;
- высокие затраты по времени.

Помимо классических методов реабилитации существует также методы, основанные на информационных технологиях. Основным подходом к реабилитации является интерактивное взаимодействие с системой, выполнение физических упражнений, контролируемых системой, а также анализ моторно-двигательных характеристик. На данный момент существующие программные продукты в области интерактивной реабилитации больных имеют обширный набор развивающих игр для крупной моторики, но не имеют возможности также развивать и мелкую моторику. Поэтому стоить уделить большое внимание к выбору и разработке методов распознавания мелкой моторики. Это позволит пациенту восстановить утраченные способности объемного движения рук, точности и скорости манипулирования предметами, а также взаимодействия в повседневной жизни с мелкими предметами.

Следующие проекты используют данные методы для реабилитации больных, перенесших инсульт.

«MIRA Rehab» [1] – Румынская компания, занимающаяся разработкой программного обеспечения в формате видео игр, мотивируя тем самым детей проходить курс реабилитации. Система позволяет получать статистику о пациенте, что отличает компанию от других аналогичных решений. Главным недостатком является высокая цена (около 450000 рублей в год), а также отсутствие корректирующего наставника, способствующего правильному течению реабилитации пациента

«Jintronix» [2] – Американская компания, занимающаяся разработкой приложений, основанных на захвате движений, так же, как и аналогичные приложения, предлагает набор приложений для развития различных навыков или реабилитации определенных заболеваний. Существенным недостатком системы является то, что компания работает только в Северной Америке.

«Habilect» [3] – Российская компания, занимающаяся разработкой комплексов спортивной медицины, в том числе и комплекса виртуальной реабилитации людей с нарушением моторно-двигательных функций. Недостатком является малая мобильность системы, а также как и во всех описанных аналогах, отсутствие реабилитации мелкой моторики.

Таблица 1 – Аналоги системы

	MIRA Rehab	Jintronix	Habilect
Упражнения для мелкой моторики	+	-	-
Распознавания мелкой моторики	-	-	-
Стоимость	400тыс. р.	120тыс. р. в год	свыше 200тыс.
Мобильность системы	-	+	-

1.2 Анализ требований к системе

1.2.1 Группы требований и задач

Бизнес-требования

1. Система должна обеспечить частичную автоматизацию реабилитационной деятельности больных, перенесших инсульт.
2. Реабилитация должна проводиться на основе интерактивных упражнений с бесконтактным взаимодействием пациента и ПО.
3. Система должна обеспечить мониторинг динамических показателей пациента на протяжении прохождения реабилитации.

Требования группы пользователей

Приложение для ПК должно представлять из себя полноэкранное приложение, содержать режим мониторинга и режим упражнений, включающий в себя набор игр для реабилитации больных.

Интерфейс ПО делится на две группы для врачей и для пациентов.

Интерфейс врача должен содержать меню настройки параметров игр (выбор игры, выбор уровня, выбор продолжительности игры), меню настройки параметров мониторинга (выбор измерений: скорость, угол, активность и т.д., выбор зоны отслеживания: локтевой сустав, кисть, плечевой сустав и т.д.), страница результатов мониторинга. Доступ к меню настроек должен обеспечиваться из любой игры по запросу врача для изменения параметров либо выбора другой игры.

Интерфейс пациента должен содержать «игровую» область, представленную в виде графической реализации упражнений с элементами анимации и объектами взаимодействия. Каждая игра должна сопровождаться заданием (в графическом или звуковом представлении), временем продолжительности игры и набранными пользователем очками за правильное выполнение заданий.

Управление системой должно осуществляться интерактивно (бесконтактно). Пользователь находится напротив монитора, где отображается игровой контент. Взаимодействие происходит с помощью следующих функций:

1. На экране появляется курсор (кисть), пациент с помощью движений своих рук, а также моторики кистей управляет этим курсором и взаимодействует с функциональными объектами;
2. На экране пациент видит сам себя через камеру, направленную на него и выполняет действия, указанные в программе.

Процесс работы с ПО должен реализовывать следующие шаги:

1. Врач настраивает параметры мониторинга;
2. Врач настраивает параметры игры;
3. Врач запускает игру;
4. В рабочей зоне появляется игровой контент, а также выводится задание;
5. Врач вслух повторяет задание или наглядно показывает пациенту как нужно выполнить задание;
6. Автоматически запускается процесс мониторинга, фиксирующий показатели пациента;
7. В случае правильного выполнения задания, выводится надпись о положительном результате, увеличивается счетчик «очков» и запускается следующее задание.
8. В случае неправильного выполнения задания, выводится оповещение об отрицательном результат и задание запускается заново.
9. Если пациент не может выполнить задание, врач должен вручную изменить задание или же изменить параметры игры в меню, уменьшив сложность задания.
10. После завершения игры врач может просмотреть результаты игрового процесса, в виде количества правильно выполненных задний, чис-

ло попыток, время выполнения и самое главное результаты мониторинга пациента.

Пациенты делятся на несколько групп:

1. Группа пациентов, которые только перенесли инсульт Игры для реабилитации данной группы не должны содержать в себе ярких, пестрых, красочных цветов, эффектов, предметов, так как на данной стадии это может губительно повлиять на пациента и испугать его;
2. Группа пациентов с нарушением крупной моторики. Игры для реабилитации крупной моторики должны содержать упражнения на развитие амплитуды, скорости и интенсивности движений верхних конечностей;
3. Группа пациентов с нарушением мелкой моторики. Игры для реабилитации данной группы должны содержать упражнение на развитие точности и правильности выполнения захватов.

Концепция каждой игры должна быть основана на одном из базовых упражнений:

1. Дотянуться до предмета

В рабочей области приложения появляется предмет (цветной объект, изображение, фигура) которого должен коснуться пациент. После касания предмет исчезает и появляется в другом месте. Если пациент не может дотянуться до предмета продолжительное время, то предмет должен появится ближе к пациенту. Данный тип упражнения должен развивать амплитуду движений рук, подъем и разведение.

2. Захватить и перенести предмет

В рабочей области появляется предмет, который пациент должен захватить и перенести в одну из определенных для данного типа предмета зону. Предметы делятся на три типа: сферические, цилиндрические, кубические. Каждый предмет можно захватить лишь определенным типом захвата. После захвата, в зависимости от уровня, пациент должен перенести предмет в определенную зону либо по заданной траектории, либо так как он может это сделать.

3. Повторить движение

Данное упражнение подразумевает повторение пациентом действий, указанных на экране с помощью графического представления. В рабочей области появляется указатели действий (стрелки вверх – поднять руки, стрелки в стороны – развести руки и т.д.), либо силуэт (рисунок) человека с анимацией действия, либо видеофрагмент с человеком, демонстрирующим правила выполнения упражнения. Пациент должен повторить данное упражнение по указанным правилам с определенной точностью в зависимости от своих возможностей и выбранного уровня игры.

Весь игровой контент должен делиться на уровни, которые будут выбираться врачом в зависимости от состояния пациента. От выбранного уровня должны зависеть следующие параметры:

1. Скорость передвижения объектов на экране.
2. Сложность траектории движения.
3. Сложность упражнений, предоставляемых пациент.;
4. Оценка качества захвата предметов.

Для развития мелкой моторики система должна содержать игры, управление в которых происходит с помощью кистей рук. Следует распознавать 5 состояний кисти:

1. Открыта.
2. Закрыта.

3. Сферический захват. Большой палец, пальцы и ладонь кисти охватывают полностью какой-нибудь сферический предмет — мяч или яблоко.

Рисунок 1 – Сферический захват

4. Цилиндрический захват. Вся ладонная поверхность ладони и пальцев охватывают какой-нибудь цилиндрический предмет, а большой палец образует около него кольцо.

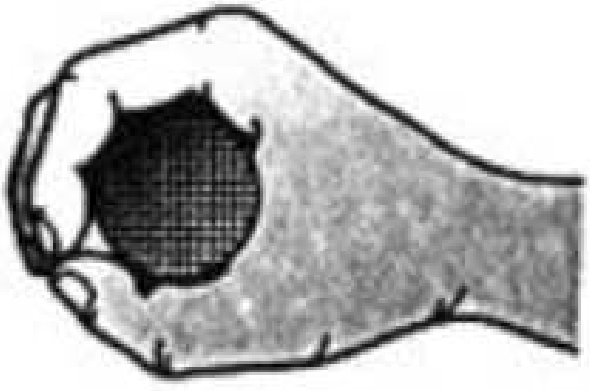

Рисунок 2 – Цилиндрический захват

5. Щипковый захват. Щипковый захват образуется кончиками большого и указательного или среднего пальцев при взятии мелкого предмета, например иголки. Иногда его называют чувствительным захватом.

Рисунок 3 – Щипковый захват

ПО должно обеспечить мониторинг динамических показателей пациента. При запуске игры должен запуститься процесс мониторинга и действовать на всем протяжении игры. Должны отслеживаться и фиксироваться следующие показатели пациента:

1. Угол (подъема и разгибания рук).
2. Скорость и ускорение точек рук.
3. Время реакции на событие (появление предмета, изменение формы, цвета).
4. Активность (время и процент активного движения).

Функциональные требования

Для реализации реабилитационных упражнений должен быть разработан комплекс интерактивных игр, где каждая игра реализует один из базовых типов упражнений (п. «Требования пользователя»). Каждая игра должна иметь уровни сложности, отличающиеся следующими характеристиками:

1. Скорость передвижения предметов.
2. Траектория передвижения предмета.

3. Размер предметов.
4. Способы захвата предмета.

Для интерактивного взаимодействия с ПО следует использовать камеру Kinect, которая способна отслеживать позиции точек тела человека в пространстве. Оперируя этими данными, система должна осуществить взаимодействие между пациентом и ПО.

Реабилитация мелкой моторики требует распознавания кисти руки. Под распознаванием подразумевается получение с помощью камеры Kinect следующих данных:

1. Положение кисти в пространстве.
2. Ориентация кисти относительно оси руки.
3. Положение точек пальцев.

Следует разработать алгоритм классификации состояния кисти. Входными данными будет являться карта глубины с камеры Kinect.

Мониторинг динамических показателей должен осуществляться на протяжении выполнения упражнения. Для этого следует получать данные с камеры Kinect о положениях точек тела и высчитывать упомянутые показатели:

1. Угол сгибания в суставе рассчитывается как угол между тремя точками в декартовой системе координат.
2. Скорость передвижения точек рук рассчитывается как отношение пройденного пути точки к затраченному времени.
3. Средняя скорость движения рассчитывается как отношение суммы моментальных скоростей к количеству замеров.

Нефункциональные требования

Спроектировать недорогостоящую, компактную систему, с мобилизационными характеристиками, а также с возможностью использования в домашних условиях.

Системные требования

Система должна состоять из аппаратной и программной части.

Программная часть - приложение для персонального компьютера, задача которого выполнять диагностику и реабилитацию пациентов, перенесших инсульт.

Программа может быть запущена только на операционной среде Windows. На персональных компьютерах системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows 8 или выше.

Для создания гибкого и красивого интерфейса, а также игр с графической составляющей, следует использовать универсальную систему WPF языка C# для более простой и быстрой разработки системы.

Аппаратная часть - система из монитора, компьютера, датчика. В качестве приемного датчика использовать внешнее подключаемое устройство.

Взаимодействие с системой должно иметь интерактивный подход, следовательно, для управления и взаимодействия пациента с ПО следует использовать внешнее периферийное устройство – камеру.

2 МЕТОДЫ РАСПОЗНАВАНИЯ ЖЕСТОВ

2.1 Обзор существующих методов распознавания руки

Существующие методы распознавания жестов руки можно разбить на две большие категории:

1. методы, основанные на анализе внешних признаков жеста.
2. методы, основанные на анализе трехмерной модели руки.

Особенностью методов, основанных на анализе внешних признаков жеста, является анализ только внешнего вида (формы, позиции и т.д.) целевого объекта. Для распознавания не хранится никакой информации о физических свойствах рассматриваемого объекта.

Рассмотрим известные работы и методы, посвященные распознаванию жестов руки человека на основе анализа внешних признаков жеста.

1. Распознавание движений на основе анализа разностей изображений

Использование разностей кадров видеоряда (MEI) позволяет в реальном времени анализировать движения объекта в видеоряде при стабильном, но необязательно однородном фоне изображения.

Вычисление разностей изображений является одним из самых простых методов, чтобы обнаружить движение в последовательности изображения.

Движение является очень важной функцией в последовательностях изображения, которая показывает отношение между изменением времени и локальными свойствами. Этот метод достаточно быстрый и позволяет в реальном времени анализировать движения объекта в видео потоке при стабильном, но необязательно однородном фоне изображения. На практике данная технология и ее усовершенствованные виды (например, motion history image - MHI) нашли применение в таких приложениях как интерактивный виртуальный тренер по аэробике [4] и интерактивная комната для рассказа историй.

2. Распознавание конфигурации и позиции с применением цветных перчаток

Для распознавания жестов руки часто применяют цветные перчатки. Предложенный в работе метод позволяет с помощью одной видеокамеры в реальном времени распознавать конфигурацию руки и отслеживать движения ладони в трехмерном пространстве рис. 5.

Используемая перчатка сконструирована из двадцати сегментов десяти разных цветов. Использование небольшого количества цветов позволяет распознать цвет выбранной точки изображения перчатки при разных освещениях, а специальное размещение цветовых сегментов не позволяет получить идентичные изображения при разных конфигурациях руки. Данная система нашла применение в задачах управления анимационным персонажам и распознавания жестов ручной азбуки глухонемых ASL.

Системы на основе захвата движения при помощи маркеров ориентированы на точность за счет простоты использования и установки. Недостатком систем, основанных на захвате движения по маркерам, является использование светоотражающих или цветных маркеров.

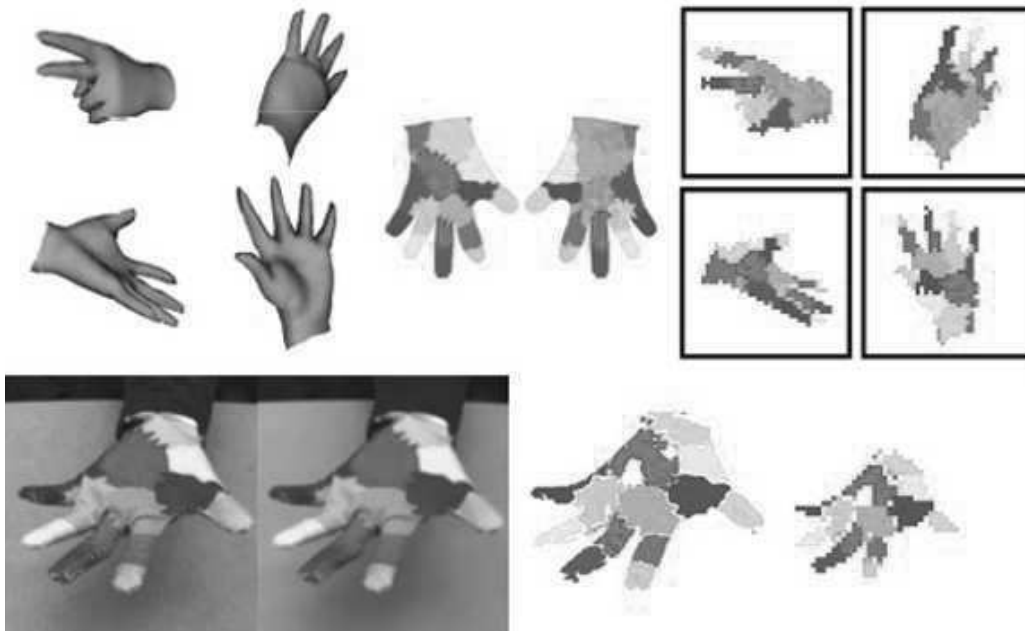

Рисунок 5 – Применение цветных перчаток в задаче распознавания жестов

3. Распознавание жестов на основе метод HOG

При решении задач по распознаванию жестов в приложениях компьютерного зрения, кроме позиции и ориентации руки человека, требуется дополнительная информация о ее конфигурации. Гистограмма направленных градиентов (HOG) это дескрипторы особых точек, которые используются в компьютерном зрении и обработке изображений с целью распознавания объектов.

В большинстве приложений компьютерного зрения, кроме позиции и ориентации руки человека, требуется дополнительная информация о ее конфигурации. Данная задача в работах [5,6] решается анализом, так называемых гистограмм направлений (orientation histograms) и карт направлений (orientation maps) изображения, которые менее чувствительны к изменениям освещения наблюдаемой среды.

Данная техника основана на подсчете количества направлений градиента в локальных областях изображения. Впервые использование HOG было предложено Navneet Dalal и Bill Triggs [7] в качестве набора признаков для нахождения пешеходов на статичных изображениях. Кроме того, HOG широко используется для распознавания жестов [8,9] из-за высокой производительности. Основной идеей алгоритма является допущение, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краев.

Для вычисления дескрипторов может применяться разделение изображения на маленькие связанные области, называемые ячейками, и затем вычислением для каждой ячейки HOG или направлений краев для пикселов, находящихся внутри ячейки. Локальные гистограммы могут подвергаться нормализации по контрасту для увеличения точности. Для этого вычисляется мера интенсивности на большем фрагменте изображения, который называется блоком, затем полученное значение используется для нормализации.

Нормализованные дескрипторы обладают лучшей инвариантностью к условиям освещения. Дескриптор HOG имеет несколько преимуществ над другими дескрипторами. Так как HOG работает локально, поддерживается инвариантность

геометрических и фотометрических преобразований, за исключением ориентации объекта. Ниже приведены изображения жестов руки и соответствующих гистограмм направлений рис. 6.

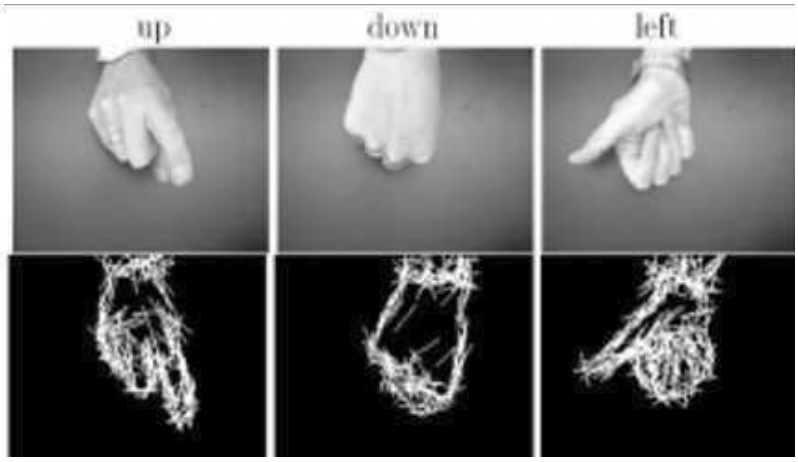

Рисунок 6 - Жесты руки, карты направлений и гистограммы направлений

4. Анализ контура руки с использованием выпуклой геометрии

Данный алгоритм не представляет серьезной сложности для реализации. Поэтому он часто используется для решения тривиальных задач распознавания жестов руки.

В своей работе Agarwal и Thakur [10] разработали метод по распознаванию языка жестов, используя изображения глубины с сенсора Kinect. Глубина и профиль движения извлекаются из изображений глубины Kinect и используются для построения матрицы особенностей для каждого жеста. Распознавание производится на основе линейного классификатора на основе метода опорных векторов (SVM).

В алгоритме сначала находится контур руки, как показано на рис. 7. Контур -кривая функции двух переменных, вдоль которой функция имеет постоянное значение. Контуры представляют собой прямые или кривые линии, описывающие резкие изменения яркости на изображении.

Рисунок 7 - Полученный контур руки

Контур находится по бинарному изображению руки, которое вычисляется по порогу входного изображения. Существует большая вероятность получения более одного контура, которые формируются на изображении из-за наличия шумов на заднем фоне. Все образованные из-за шума контуры невелики, поэтому наибольший контур рассматривается как контур руки для дальнейших вычислений.

По контуру находится выпуклая геометрия (ConvexHull) объекта руки. Выпуклая геометрия набора точек в евклидовом пространстве – это наименьшая выпуклая оболочка, содержащая все исходные точки. На рис. 8 показана выпуклая геометрия контура руки.

В качестве алгоритма нахождения выпуклой геометрии используется алгоритм сканирования Грэма со сложностью O(n log n). На рис. 8 показан пример работы алгоритма Грэма.

На первом шаге при использовании этого алгоритма находится точка P(x,y)с наименьшим значением по координате y. Если такая точка с координатой y существует, а полученный набор точек имеет размер больше 1, то находится точка P(x,y)с наименьшим значением по координате x из имеющегося набора точек. На этом шаге время выполнения алгоритма составляет O(n), где n–число точек, о которых идет речь.

Рисунок 8 –Выпуклая геометрия контура руки

На следующем шаге рис.9 множество точек сортируется в порядке возрастания угла, который каждая точка и точка P образуют с осью x.

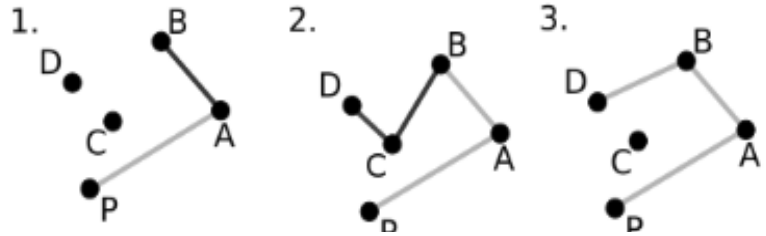

Рисунок 9 – Алгоритм нахождения выпуклой геометрий с использованием сканирования Грэма

Затем последовательно рассматривается каждая точка в отсортированном массиве. Если точка вместе с предыдущей и последующей точками образует правосторонний поворот, то эта точка не принадлежит выпуклой геометрии. Если точка с предыдущей и последующей точками образует левосторонний поворот, то точка принадлежит выпуклой геометрии исходного множества точек. Алгоритм выполняется до тех пор, пока не будет достигнута начальная точка P.

Выпуклая геометрия рисуется вокруг контура руки и все точки контура находятся внутри нее. После этого вычисляются дефекты выпуклой геометрии. Дефекты выпуклой геометрии –это точки на контуре, наиболее отдаленные от

выпуклой геометрии. Каждая такая точка характеризуется следующими принадлежащими контуру точками:

1. точкой начала дефекта (принадлежит выпуклой геометрии).
2. точкой конца дефекта (принадлежит выпуклой геометрии).
3. дефект выпуклой геометрии (наиболее отдаленная от выпуклой геометрии точка между точкой начала и точкой конца дефекта).

На рис. 10 изображен пример найденных дефектов выпуклой геометрии имеющегося контура руки.

Рисунок 10 –Найденный дефекты выпуклой оболочки

Кончики пальцев находятся при помощи анализа найденных точек. Полученный набор точек сравнивается с шаблоном, который описывает некоторый жест. Недостатком данного метода является невозможность распознавать сложные жесты.

2.2 Анализ технических средств, используемых для распознавания жестов

Распознавание жестов рук в последнее время становится очень популярным вариантом управления компьютером. Способность отслеживать состояние рук, а потом определять к какому жесту это относится, может быть достигнуто с помощью специальных устройств. Существует несколько типов устройств, ис-

пользуемых в распознавании жестов. Основные из них это видеокамера либо камера глубины.

2.2.1 Обычная камера

Обычная камера может быть использована в системе распознавания жестов. Главной сложностью является выделение руки от общего изображения. Для этого требуется либо однотонный фон, либо маркеры, нанесенные на руку (цветные перчатки). Из-за двумерного представления функционал камеры ограничивается лишь распознаванием простых жестов.

2.2.2 3D-сенсор

В ноябре 2010 года Microsoft выпустила бесконтактный сенсорный игровой контроллер Kinect, что поспособствовало появлению новых возможностей для распознавания жестов. На рис. 11 показаны сенсоры Kinect первого и второго поколения. Несмотря на то, что сенсоры глубины существовали уже относительно давно, сенсор Kinect имел ряд значительных преимуществ: большее распространение, относительно невысокая стоимость и наличие RGB камеры. Карта глубины, которая может быть получена при помощи сенсора Kinect инвариантна по отношению к освещению и фону, поскольку основана на инфракрасном излучении.

а) б)

Рисунок 11 – (а) Kinect первого поколения, (б) Kinect второго поколения

Отличительными особенностями сенсора Kinect являются RGB камера, сенсор глубины и массив из четырех цифровых микрофонов, что позволяет от-

слеживать движения всего тела, распознавать выражения и мимику лица, а также распознавание голоса и речи.

Сенсор глубины состоит из инфракрасного лазерного проектора и монохромной CMOS-матрицы, что позволяет получать карту глубины даже в темном помещении. Диапазон работы сенсора Kinect составляет от 0,5-4,5 м. Частота RGB потока зависит от выбранного разрешения и варьируется от 9 Гц до 30 гц. Стандартный RGB поток использует фильтр Байера и имеет разрешение по умолчанию 8 бит VGA разрешение (640x480 пикселей), три канала цвета.

На рис. 12 показан принцип работы сенсора Kinect первого поколения. Поток видео, получаемый с сенсора глубины, имеет разрешение QVGA (320 × 240 пикселей) и 11-битную глубину, которая обеспечивает 2,048 уровней чувствительности. Kinect может также предоставлять поток из ИК-камеры непосредственно (прежде, чем он был преобразован в карту глубины) с разрешением 640x480.

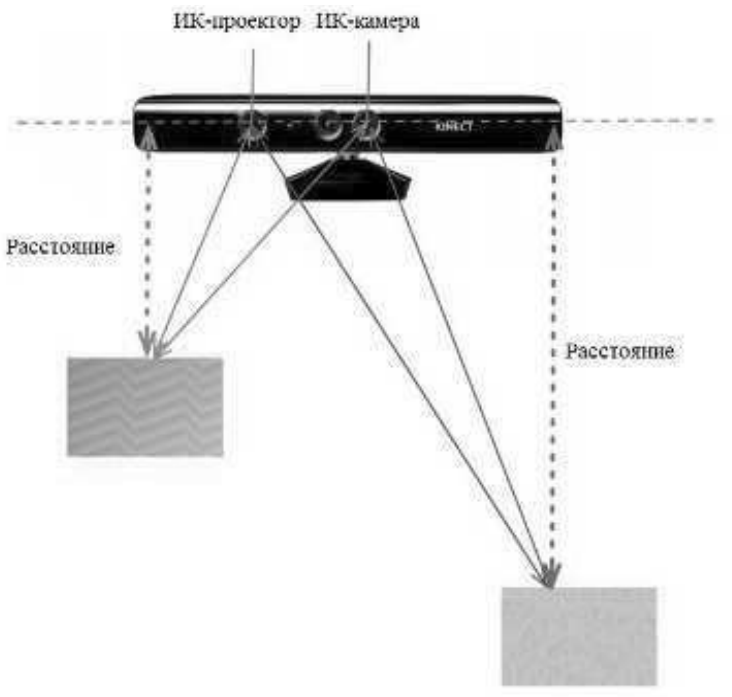

Рисунок 12 – Пример работы сенсора Kinect

Сенсор Kinect имеет практический диапазон 1.2-3.5 м при использовании. Площадь, необходимая для использования сенсора Kinect, составляет примерно 6

м2. Сенсор имеет угловое поле зрения 57° по горизонтали и 43° по вертикали, в то время как встроенный мотор способен вращаться сенсор дополнительно на 27° вверх или вниз. Поэтому горизонтальное поле сенсор Kinect на минимальном расстоянии просмотра от ~0,8 м составляет ~87 см, а также вертикального поля составляет ~63 см, что приводит к разрешению чуть более 1,3 мм на пиксель. Также сенсор имеет массив микрофонов, каждый из которых предоставляет 16-битный цифровой звуковой сигнал с частотой дискретизации 16 кГц.

Microsoft Kinect имеет две версии для игровой приставки Xbox и для Windows. Версия для Windows предназначена для непосредственного использования с компьютером и имеет особый диапазон камеры глубины Near Mode рис. 13. Другим отличием версий Kinect является небольшое увеличение точности сенсора для компьютера.

Рисунок 13 - Режимы работы камеры глубины сенсора Kinect

Сенсор Kinect второго поколения основан на инфракрасном излучении, как и сенсор Kinect предыдущего поколения, но имеет более высокую точность по сравнению с его предшественником и позволяет обрабатывать до 2 гигабит данных в секунду. В отличие от предыдущей версии, для вычисления глубины сенсор Kinect второго поколения используется другой принцип работы TOF, как показано на рис. 14. Разрешение камеры восприятия глубины равно 512x424 пикселей. Сенсор имеет большее поле зрения, меньшее расстояние между игроком и сенсором, необходимое для обеспечения оптимальной производительности Kinect.

Сенсор также включает в себя RGB камеру с разрешением 1920x1080 пикселей, которую можно использовать для записи видео и видеочата. Так же существует возможность отслеживать до 6 скелетов сразу, вычислить частоту биения сердца игрока, сканировать QR-коды. Kinect использует массив четыре микрофона, который позволяет более широкую поддержку голосовых команд через программное обеспечение.

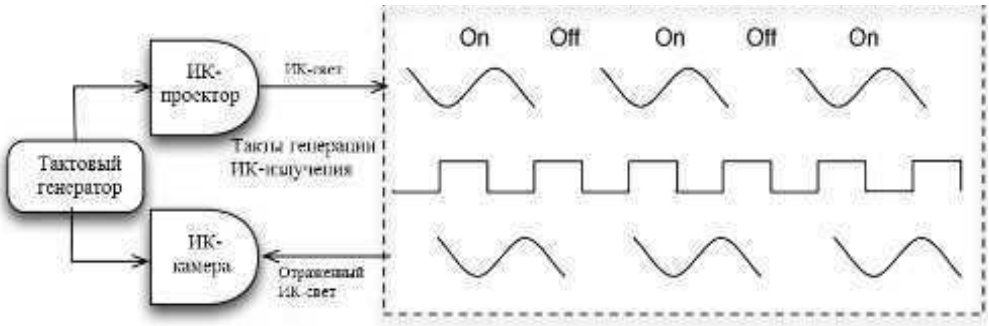

Рисунок 14 – Пример работы сенсора Kinect второго поколения

Еще одним популярным устройством, которое используется для распознавания, является Asus Xtion Pro Live, изображенный на рис. 15. Сенсор предоставляет возможность получения карты глубины, RGB изображения, а также имеет 2 микрофона. Подробные характеристики предоставлены в таблице 1. К сожалению, данное устройство снято с производства.

Рисунок 15 – Asus Xtion Pro Live

Разработчики устройства под названием Leap Motion использовали другой подход к получению карты глубины. Сенсор Leap Motion показан на рис. 16. Как и другие устройства, Leap Motion также оснащен камерой восприятия глубины, однако устройство не имеет таких дополнительных функций, как RGB камера или

микрофон. Leap Motion не позволяет работать с скелетом человека из-за маленького радиуса действия устройства, также из-за отсутствия RGB камеры функции отслеживания лица, распознавания эмоций и мимики недоступны.

Рисунок 16 – Leap Motion

Intel RealSense является платформой для реализации HCI на основе методов распознавания жестов. Сенсор изображен на рис. 17. Сенсор Intel RealSense содержит следующие четыре компонента: обычная RGB камера, инфракрасный лазерный проектор, инфракрасная камера и микрофонная решетка. Как и в Kinect первого поколения инфракрасный проектор проецирует сетку на сцену (в инфракрасном свете, который невидим для человеческого глаза) и инфракрасная камера анализирует ее для вычисления информации о глубине. Микрофонная решетка позволяет локализовать источники звука в пространстве и выполнения отмены фонового шума. Программное обеспечение для разработчиков Intel RealSense SDK имеет встроенные функции для анализа эмоций человеческого лица, отслеживания руки и пальцев, распознавания речи, а также для создания дополненной реальности.

Рисунок 17 – Intel RealSense

Таблица 1 - Характеристики сенсоров

Параметр\ Сенсор	Microsoft Kinect	Asus Xtion Pro Live
Видео камера	RGB (1280*960)	RGB (1280*1024)
Камера глубины	640*480 (от 0,8м до 4м)	640*480 (от 0,8 м до 3м)
Вес	1500 г	250 г
ПО	Microsoft SDK	Open NI
Регулировка угла	Программное	Ручное

2.3 Выводы

В этой главе были рассмотрены существующие методы и аппаратные средства для распознавания жестов руки.

Из материалов раздела можно сделать вывод что:

1. Самым оптимальным методом по времени и сложности реализации, а также точности распознавания в реальном времени является алгоритм анализа контура руки.
2. Устройством, которое удовлетворяет всем требованиям проекта, легкодоступно и имеет простое и понятное API для быстрой реализации, является Kinect v2.

Алгоритм анализа контура руки требует меньшей затраты времени на реализацию, предоставляет удовлетворяемую точность распознавания, так как не требуется распознавание сложных жестов, а лишь 5 состояний. Может работать в режиме реального времени с частотой в 30 кадров в секунду, что является одним из главных критериев выбора, так как адекватная и своевременная отзывчивость программы на действия пользователя очень важна.

Алгоритм работает с контуром руки, получать этот контур можно как с помощью обычной камеры, так и с 3D сенсора с помощью карты глубины, выделив объект на заданном расстоянии. Для обычной камеры нужны особые условия

(однородный фон, яркое и равномерное освещение, по возможности статичное расположение руки, дополнительные алгоритмы выделения и распознавания местоположения человека и рук), которые затруднительно выполнять в режиме реального использования. 3D сенсор не требует однородного фона, а также не зависит от освещенности помещения, так как работает в инфракрасном диапазоне волн, имеет встроенные функции распознавания контрольных точек тела, что понадобиться для алгоритмов взаимодействия пользователя с системой.

Выбор датчика был остановлен на Kinect второго поколения. Он имеет достаточную размерность карты глубины, для точного определения кисти на расстоянии. Также имеет RGB камеру, что непосредственно пригодиться в разработке игр разной направленности. Доступность и цена удовлетворяют потребности заказчика. Также датчик от Microsoft имеет удобную API Kinect SDK для работы с датчиком, что позволит уменьшить срок и затраты разработок.

3 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

По результатам исследования предметной области, а также существующих программных продуктов реабилитации больных с нарушением моторно-двигательных функции, можно сделать вывод, что для создания востребованного и уникального продукта надо реализовать следующие функции:

1. Интерактивное (бесконтактное) взаимодействие пользователя с системой. Это позволит пациенту, вне зависимости от его физического состояния, взаимодействовать с системой в свободной форме, выполнять реабилитационные упражнения, которые будут в режиме реального времени отслеживаться системой.
2. Набор развивающих игр. Для реабилитации больных система должна предоставить набор игр, основанных на базовых физических упражнениях (п. 1.2.1). Выполнение этих упражнений позволяет пациенту улучшать свои моторно-двигательные характеристики.
3. Распознавание мелкой и крупной моторики. Это позволит взаимодействовать с игрой с помощью кинематических движений тела, а также состояний кистей рук.
4. Мониторинг. Данный режим позволит врачу отследить динамические показатели (п. 1.2.1) пациента на протяжении реабилитационной деятельности. И на основании полученных показаний формировать результат об успешности реабилитации пациента.

Для решения поставленной задачи, на основе анализа методов распознавания и датчиков приема информации, формируется общая архитектура проекта, которая будет содержать в себе основные компоненты в виде монитора и компьютера, а также 3D сенсора Kinect (выбор ранее обоснован см. п. 2.3).

Ключевой этап разработки приложения – проектирование. Для нашего программного продукта мы выделили следующие пункты:

1. проектирование архитектуры приложения.
2. проектирование раздельного интерфейса для врача и пациента.
3. проектирование логики программы.

3.1 Проектирование архитектуры системы

3.1.1 Описание общей структуры проекта

Вся система состоит из 4 частей:

1. Набор развивающих игр.
2. Система распознавания мелкой и крупной моторики.
3. Система мониторинга.
4. Интерфейс работы с датчиком Kinect.

На рисунке 18 представлена структурная схема программной части системы.

Рисунок 18 – Структурная схема

Система содержит набор развивающих игр, направленных на реабилитацию больных, перенесших инсульт. Каждая игра включает в себя упражнения, тренирующие определенные навыки пациента. Пациент взаимодействует с приложением с помощью специальной камеры Kinect.

Что бы развивать мелкую и крупную моторику пациента, система содержит специальные алгоритм распознавания моторки (положение точек тела в пространстве, состояние кистей, положение пальцев и т.д.). Алгоритм работает с данными, получаемыми из камеры Kinect (кадр глубины, кадр контрольных точек тела).

На протяжении всей игры работает система мониторинга, она отслеживает и фиксирует данные о пациенте (угол подъема, сгибания и разгибания, скорость и ускорение точек тела, активность отдельных частей тела). После каждой игры эти данные формируются и выводятся на экран в виде полной информации о пациенте.

3.1.2 Проектирование структуры приложения

Приложение состоит из следующих частей:

1. Набор игр для реабилитации
2. Система распознавания мелкой и крупной моторики
3. Система мониторинга.

Камера Kinect собирает информацию о пациенте и передает ее во все три части системы, которые обрабатывают полученные данные. Игры используют данные для управления игровым процессом, взаимодействием с объектами и т.д. Система распознавания использует данные для определения состояния кистей рук. Система мониторинга формирует показатели пациента в результирующую таблицу.

На рис. 19 представлена функциональная схема взаимодействия элементов.

Рисунок 19 – Функциональная схема

Данная схема отображает работу системы реабилитации. Из функциональной схемы видно, что центральным объектом управления является работа с датчиком Kinect, с помощью которого производится взаимодействие со всеми частями системы. Пользователи могут управлять системой, фиксировать свои физические показатели, а также проходить реабилитационные упражнения лишь с помощью сдатчика Kinect.

3.2 Проектирование БД

Схема базы данных представлена на рис. 20.

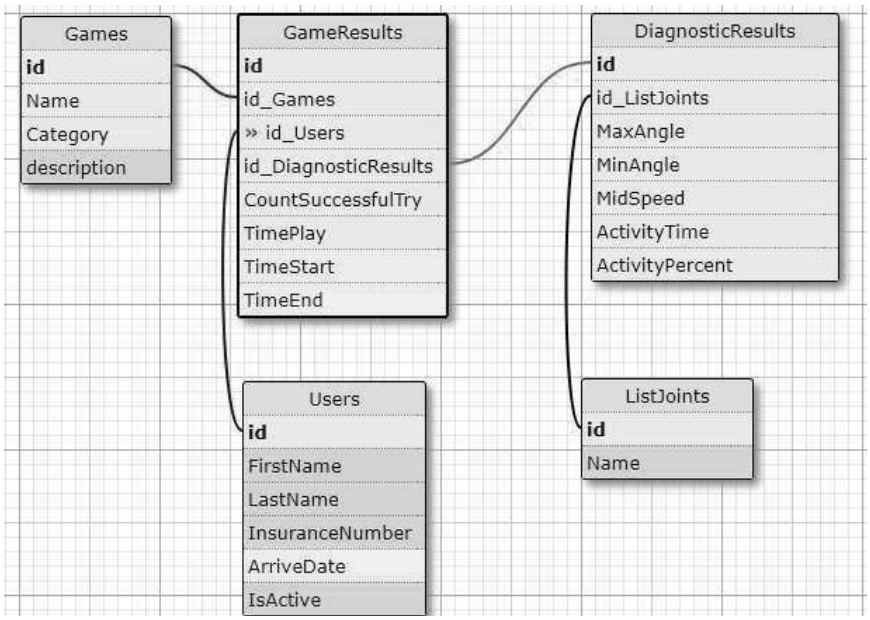

Рисунок 20 – Схема базы данных

Используемые типы данных:

- int – целочисленный тип данных;
- nvarchar – строковый тип данных переменной длины;
- datetime – тип данных, представляющий дату и время;
- bit – булевский тип данных.

Краткое описание таблиц:

- Users – таблица пользователей;
- Games – таблица доступных игр;
- GameResults – таблица результатов прохождения игр пациентами;
- DiagnosticResults – таблица результатов мониторинга пациентов;
- ListJoints – список контрольных точек тела;
- Result – полные результаты о реабилитации пациента.

При описании атрибутов используются следующие обозначения:

* – первичный ключ;

^ – внешний ключ и ссылающийся на указанный атрибут.

В таблице 2 содержится информация обо всех пользователях системы, о их правах, ролях в системе, а также логин и пароль для авторизации.

Таблица 2 – Пользователи

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	FirstName		Nvarchar	Имя пациента
3	LastName		Nvarchar	Фамилия пациента
4	InsuranceNumber		Nvarchar	Номер страхового полиса
5	ArriveDate		Datetime	Дата прибытия
6	IsActive		Bit	Активность пациента

В таблице 3 содержится информация о доступных играх, их категории и описании.

Таблица 3 – Игры

№	Название	Ключ	Тип	Семантика
1	id	*	int	Уникальный идентификатор
2	Name		Nvarchar	Название игры
3	Category		Nvarchar	Категория игры
4	Description		Nvarchar	Описание игры

В таблице 4 содержится информация о результатах прохождения развивающих игр пациентами.

Таблица 4 – Результат игры

№	Название	Ключ	Тип	Семантика
1	id	*	Int	Уникальный идентификатор
2	id_Games	^	Int	Пройденная игра
3	id_Users	^	Int	Пользователь
4	id_DiagnosticResults	^	Int	Результат мониторинга
5	CountSuccessfulTry		Int	Кол-во правильных попыток
6	TimePlay		DateTime	Затраченное время
7	TimeStart		DateTime	Время начала игры
8	TimeEnd		DateTime	Время окончания игры

В таблице 5 содержится информация о результатах мониторинга пациента.

Таблица 5 – Результат диагностики

№	Название	Ключ	Тип	Семантика
1	id	*	Int	Уникальный идентификатор
2	id_ListJoints	^	Int	Контрольная точка тела
3	MaxAngle		Int	Максимальный угол
4	MinAngle		Int	Минимальный угол
6	MidSpeed		Int	Средняя скорость движения
7	ActivityTime		Int	Время активности
8	ActivityPersent		Int	Процент активности

В таблице 6 содержится информация о всех доступных контрольных точках тела, которые могут отслеживаться системой мониторинга.

Таблица 6 – Список контрольных точек

№	Название	Ключ	Тип	Семантика
1	id	*	Int	Уникальный идентификатор
2	Name		Nvarchar	Контрольная точка тела

В таблице 7 содержится информация о полном результате реабилитации пациента.

Таблица 7 – Результаты

№	Название	Ключ	Тип	Семантика
1	id	*	Int	Уникальный идентификатор
2	id_GamesResult	^	Int	Результат игр
3	id_DiagnosticResults	^	Int	Результат мониторинга
4	id_Users	^	Int	Пользователь

Спроектирована база данных, удовлетворяющая всем требованиям системы.

3.3 Проектирование графического интерфейса настольного приложения

Графический интерфейс ПО делиться на две часть:

1. Интерфейс управления для врача;
2. Интерфейс игры для пациента.

3.3.1 Графический интерфейс для врача

Врач должен иметь возможность управлять реабилитационной деятельностью, изменять параметры игр, а также мониторинга, запускать и останавливать игры, просматривать результаты реабилитации. В настройках параметров игры врач может выбрать игру и выставить для нее следующие параметры:

1. Уровень сложности. Влияет на основные характеристики игр (скорость движения предметов, траектория перемещения и т.д.);
2. Время продолжительности игры;
3. Упрощенный режим. Режим без графической нагрузки, с минимальной сложностью упражнений;
4. Группа пациентов. (п 1.2.1). В зависимости от группы выбирается графическое оформление и особенности игры.

Настройка параметров мониторинга содержит следующие составляющие:

1. Выбор точки отслеживания;
2. Выбор стороны;
3. Выбор характеристик мониторинга

Интерфейс меню для врача изображен на рис. 21.

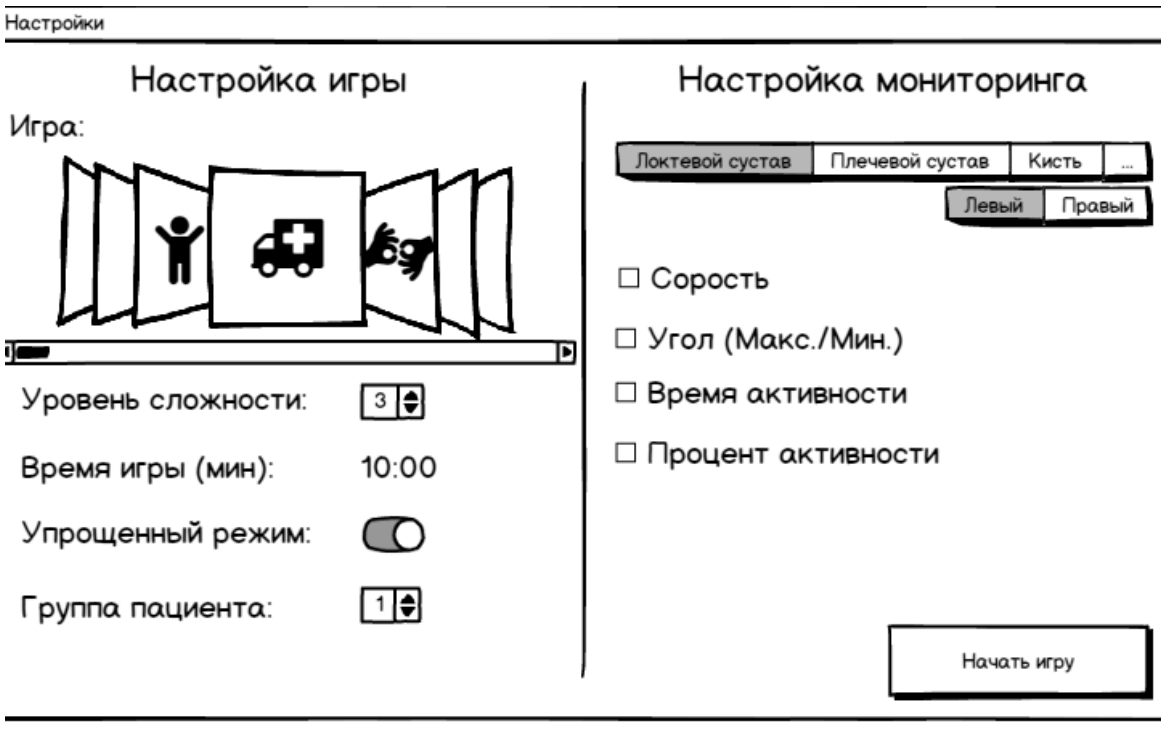

Рисунок 21 – Интерфейс врача (меню настроек)

Врач может просматривать результаты по окончанию прохождения пациентом игры. Результат содержит информацию о правильно выполненных заданиях, количестве повторений упражнения, общем затраченном времени. Также в результатах выводиться информация о мониторинге за период выполнения упражнения.

Интерфейс вывода результатов пациента для врача изображен на рисунке 22.

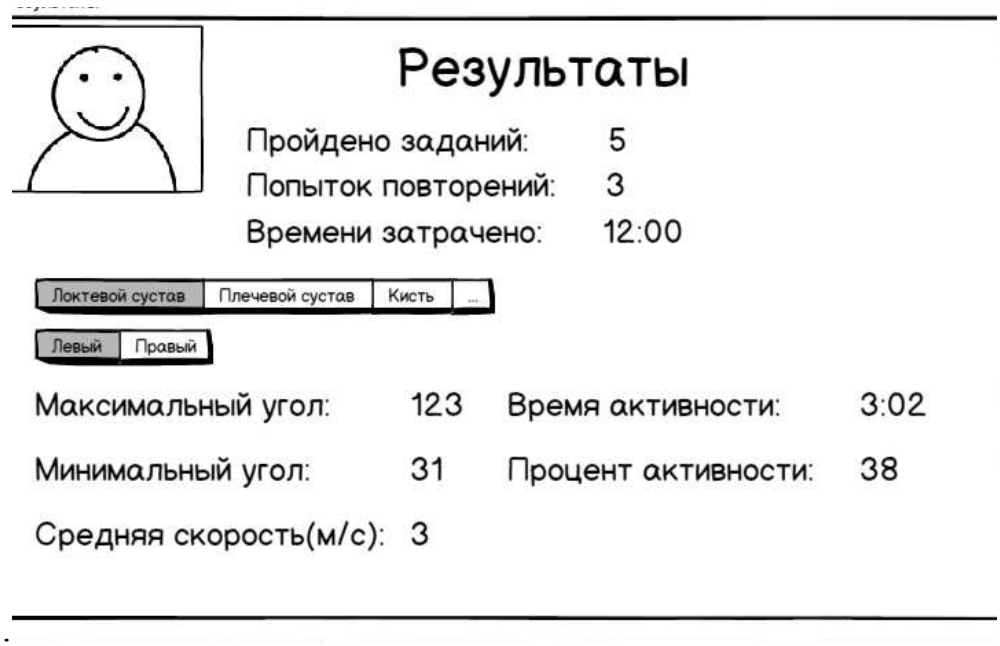

Рисунок 22 – Интерфейс врача (результаты пациента)

3.3.2 Графический интерфейс для пациента

Графический интерфейс пациента должен содержать рабочую зону (на весь экран), в которой находится игровой контент. При запуске игры выводится задание, а в углу ведется счет времени и очков правильно выполненных заданий. Интерфейс для пациента на примере игры «Перенеси предмет» представлен на рис. 23.

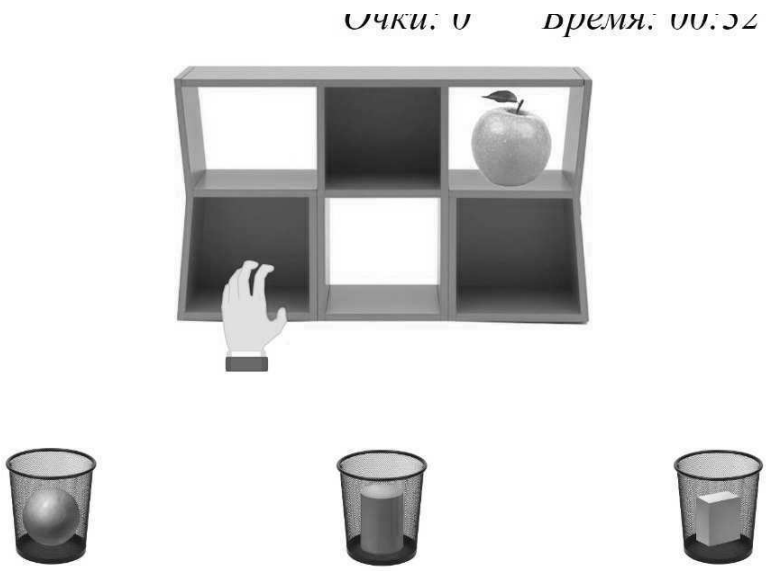

Рисунок 23 – Интерфейс пациента

3.4 Выбор платформы для реализации системы

Для реализации системы распознавания жестов языка глухонемых была выбрана платформа .NET Framework версии 4.6. .NET Framework - программная платформа, предоставляющая широкий спектр возможностей для запущенных на ней приложений. Эта платформа состоит из двух основных компонентов: общеязыковой среды выполнения (CLR), обрабатывающей запущенные приложения, и библиотеки классов .NET Framework, которые разработчики могут использовать для написания собственных приложений.

Существует 3 основных направления развития платформы .NET Framework:

1. Усовершенствование существующих языков, компиляторов, библиотек базовых классов и инструментов;
2. Развитие платформы в открытом доступе при поддержке сообщества;
3. Расширение платформы для операционных систем Linux и Mac OS.

Благодаря открытию исходных кодов платформы .NET Framework появилось большое количество различных библиотек, фреймворков.

3.5 Выбор языка программирования для реализации системы

В качестве языка программирования в соответствии с выбранной платформой использовался C#. Поскольку C# является основным языком платформы .NET, он был выбран реализации программы для распознавания жестов.

C# - мощный объектно-ориентированный язык программирования, предлагающий решения для малых и больших потребностей в области развития программирования. C# является типобезопасным языком. По умолчанию только неявные преобразования считаются безопасными. Это обеспечивается во время компиляции и, в некоторых случаях, во время выполнения.

По мнению многих источников, это один из самых ценных языков программирования для обучения программированию. Кроме того, по сравнению с другими языками программирования, C# относительно прост и легок в освоении, что делает его лучшим выбором для начинающих и даже опытных разработчиков.

3.6 Выбор библиотеки компьютерного зрения

В качестве библиотеки компьютерного зрения используется библиотека OpenCVSharp, являющаяся кроссплатформенным .NET враппером библиотеки OpenCV. OpenCVSharp позволяет вызывать функции OpenCV из .NET совместимых языков, таких как C#, Visual Basic, Visual C++ и других. Исходный код библиотеки OpenCVSharp написан на C# и находится в открытом доступе. Преимущество библиотеки в отличие от других врапперов библиотеки OpenCV заключается в возможности запуска приложений в таких операционных системах, как Windows, Windows Phone, Linux, Mac OS X, iOS и Android.

3.7 Выбор базы данных

Критериями выбора СУБД были: производительность, стабильность, наличие полной документации. В результате анализа рынка были выбраны следующие СУБД:

- PostgreSQL;

- MySQL;
- MSSQL.

MySQL – очень популярна, но данная СУБД имеет закрытые части, платна для коммерческого использования и имеет проблемы с надежностью из-за некоторых способов обработки данных

PostgreSQL – свободная объектно-реляционная система управления базами данных СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. Но у неё есть существенные недостатки, такие как не самая высокая производительность и невысокая популярность.

Microsoft SQL Server – система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL). Ограничения бесплатной версии Microsoft SQL не имеют в данном случае определяющего значения

Исходя из результатов сравнения, для использования в проектируемом программном продукте была выбрана СУБД Microsoft SQL Server 2016.

4 РАЗРАБОТКА МЕТОДА РАСПОЗНАВАНИЯ ЖЕСТА РУКИ НА ОСНОВЕ ГЛУБИННОГО ИЗОБРАЖЕНИЯ

Для реализации реабилитации мелкой моторики следует разработать алгоритм распознавания жестов. В данной задаче достаточно будет распознать 5 состояний кисти (п. 1.2.1).

Положение пальцев и ориентация руки человека имеют важную составляющую в распознавании жестов. Для этого следует найти ключевые точки руки и по этим точкам определять текущий жест.

В данной работе был выбран алгоритм распознавания на основе анализа контура рук, полученного из карты глубины с помощью 3D ссенсора Kinect. Структура системы распознавания жестов изображена на рис.24

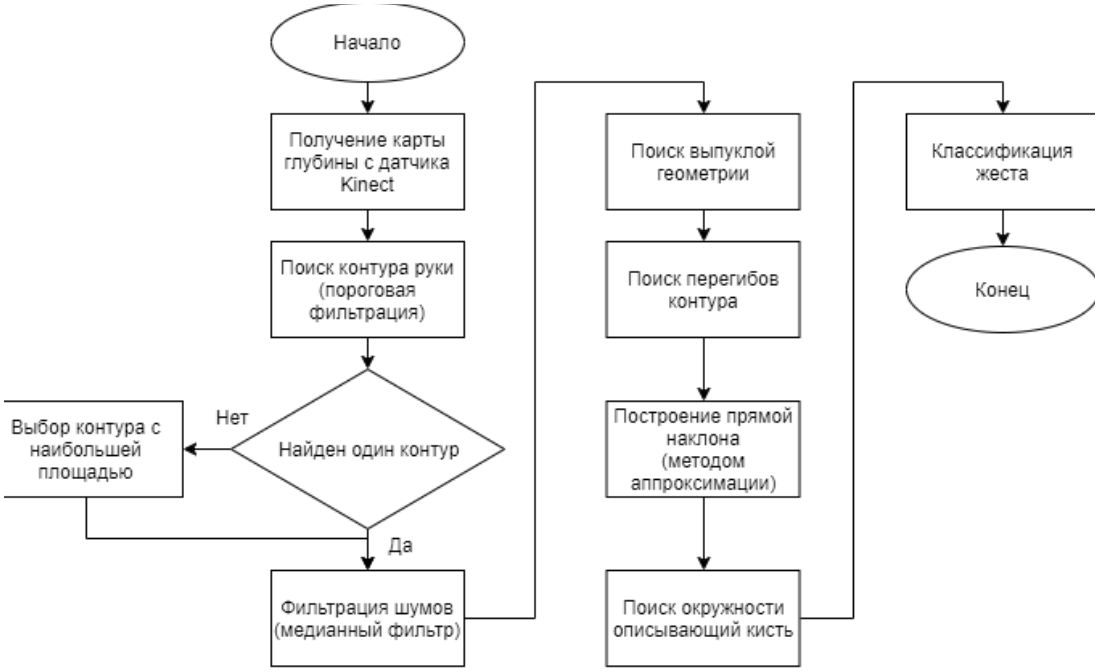

Рисунок 24 – Структура системы распознавания

4.1 Карта глубины

Обзор существующих решений показал целесообразность использования изображения глубины для распознавания образа руки. В этом пункте работы представлен алгоритм получения данных с сенсора Kinect и их обработки. Для получе-

ния информации с устройства используется программный драйвер и пакет программного обеспечения Microsoft SDK 1.8.

После обнаружения подключенного к компьютеру сенсора Kinect устанавливаем соединения. Программное обеспечение позволяет получать поток данных с RGB камеры, поток данных скелета и изображение глубины. Изображение глубины представляет собой массив байтов размером 512x424. В каждом бите содержится расстояние $d(x, y)$ от сенсора до соответствующей точки пространства, выраженное в миллиметрах. Частота получения кадров глубины достигает 30 кадров в секунду.

Ниже на рис. 25 представлен пример изображения глубины, полученного с сенсора Kinect. На изображении отчетливо видны точки, создающие поверхность руки. Заметны неровности поверхности и контуров руки, которые свойственны всем объектам в изображении глубины.

а) б)

Рисунок 25 - Укрупненные изображения руки

Однако, несмотря на преимущества у значений карты глубины есть ограничения, которые влияют на разработку новых алгоритмов:

1. Карта глубины и цветное изображение получены различными датчиками, таким образом, необходимо знать точные характеристики каждого датчика, чтобы использовать эти два набора данных;
2. Как и при использовании цветных изображений, карта глубины не имеет точных значений, так как зависит от влияния и наличия различного шума.

Такой шум должен фильтроваться или по крайней мере приниматься во внимание в алгоритме распознавания, чтобы минимизировать ошибочную классификацию;

3. Карте глубины свойственно содержать разрывы, соответствующие частям поверхности, которые не достигнуты датчиком. Могут присутствовать также пиксели в цветном изображении, соответствующие разрывам в карте глубины;
4. Скорость захвата (кадры в секунду) недорогостоящих датчиков глубины обычно ниже, чем уровень, на котором получаются цветные изображения.

4.2 Получение изображения и предобработка

Полученная с сенсора Kinect карта глубин содержит лишнюю информацию о заднем фоне руки. Для выделения области руки используется пороговая фильтрация (Thresholding), позволяющая преобразовать исходное изображение в бинарное. Исходное изображение получается с сенсора Kinect в виде карты глубин. При выполнении пороговой фильтрации каждый пиксель изображения I(x ,y), имеющий значение меньше постоянной T, заменяется значением равным единице, в противном случае пиксель заменяется значением равным нулю. Постоянной T будет являться расстояние от датчика до кисти руки, которое мы определяем с помощью камеры глубины рис. 26.

Рисунок 26 – Бинарное изображение руки

В обработке изображений часто необходимо уменьшить уровень шума на изображении для улучшения результатов последующей обработки. Для этого ис-

пользуются различные морфологические операции, цифровые фильтры рис 27. В работе в качестве цифрового фильтра для уменьшения шума использовался медианный фильтр. Медианный фильтр является нелинейным цифровым методом фильтрации, основанным на статистике. Основной идеей медианного фильтра является замена каждого значения пикселя средним из значений соседних пикселей.

Рисунок 27 – Результат обработки фильтром

В программе [Приложение А], рис. 28, для размытия была использована функция библиотеки OpenCvSharp – CV.Smooth() с ядром размытия размерностью 3 на 3.

```
  1);
IplImage imgContour = new IplImage(new CvSize(imgDepth.ROI.Width, imgDepth.ROI.Height), BitDepth.U8,
  1);
IplImage outImgHand = new IplImage(new CvSize(400, 400), BitDepth.U8, 1);
imgContour.Zero();
imgLeftHand = imgDepth.Clone(imgDepth.ROI);
Cv.Smooth(imgLeftHand, imgLeftHand, SmoothType.Median, 3, 3);
Cv.Resize(imgLeftHand, outImgHand);
imageHand = outImgHand.ToWriteableBitmap(PixelFormats.Gray8);
```

Рисунок 28 – Часть кода размытия контура

4.3 Обрезка кисти руки по запястью

Информативной частью руки для нас является кисть. Поэтому обрезается честь предплечья и остается только кисть. Для начала стоит определить рабочую зону, то есть окрестность вокруг центра кисти. Для этого нужно найти радиус окружности с центром в точки центра ладони. Точку центра ладони мы получим из камеры Kinect. Радиусом окружности будет являться расстояние от центра до са-

мого удаленного пальца. Таким образом после наложения окружности на изображение, а также исключения точек, лежащих снаружи окружности, мы получаем лишь рабочую область – кисть рис. 29.

Рисунок 29 – Рабочая область кисти руки

4.4 Распознавания жеста руки

По контуру находится выпуклая геометрия (ConvexHull) объекта руки. Выпуклая геометрия набора точек в евклидовом пространстве – это наименьшая выпуклая оболочка, содержащая все исходные точки. В качестве алгоритма нахождения выпуклой геометрии используется алгоритм сканирования Грэма со сложностью O (n log n).

После внутри выпуклой геометрии на основе анализа контура находятся контрольные точки. Из контура берется 3 точки и ищется угол между ними, если он превышает критический угол, значит это перегиб и здесь находится контрольная точка. На рис. 30 отмечены такие точки.

Рисунок 30 – Кисть с контрольными точками

Для того что бы отсечь точки между пальцев следует брать только положительные углы перегиба. На рис. 31 представлена часть кода [Приложение А] поиска контрольных точек.

```
Vector v2;
if (counterPoint - 2 * koef < 0)
{
    v1 = new Vector(c[len - 1 + (counterPoint - 2 * koef)].Value.X - c[counterPoint -
      koef].Value.X, c[len - 1 + (counterPoint - 2 * koef)].Value.Y - c[counterPoint -
      koef].Value.Y);
    v2 = new Vector(c[counterPoint].Value.X - c[counterPoint - koef].Value.X, c
      [counterPoint].Value.Y - c[counterPoint - koef].Value.Y);
}
else
{
    v1 = new Vector(c[counterPoint - 2 * koef].Value.X - c[counterPoint - koef].Value.X, c
      [counterPoint - 2 * koef].Value.Y - c[counterPoint - koef].Value.Y);
    v2 = new Vector(c[counterPoint].Value.X - c[counterPoint - koef].Value.X, c
      [counterPoint].Value.Y - c[counterPoint - koef].Value.Y);
}

double angleBetween;
angleBetween = Vector.AngleBetween(v1, v2);
if (angleBetween == 0)
{
    counterPoint++;
    continue;
}
if ((angleBetween < angle && angleBetween > 0))
{
    point[k++] = c[counterPoint - koef].Value;
}
counterPoint++;
```

Рисунок 31 – Код поиска контрольных точек

Для определения жеста руки нужно определить поворот руки. Осуществлено это с помощью нахождения прямой, аппроксимированной по точкам пальцев, с помощью которой можно определить вертикальное либо горизонтальное положение кисти, что важно для распознавания сферического и цилиндрического захвата. На рис. 28 представлена кисть и прямая, как результат аппроксимации.

Рисунок 32 – Кисть с прямой горизонта

Также определяется положение большого пальца для того, чтобы исключить его из метода аппроксимации. На рис 32 большой палец обозначен желтым кругом. На рис. 33 представлен кусок кода [Приложение А], где по двум массивам координат точек Xval и Yval находятся координаты аппроксимированной прямой и определяется угол наклона к вертикали и горизонтали.

```
    Tuple<double, double> pVert = Fit.Line(yVal, xVal);
    double aV = pVert.Item1;
    double bV = pVert.Item2;
    double aG = pGoriz.Item1;
    double bG = pGoriz.Item2;
    arrGoriz[0].X = 0;
    arrGoriz[1].X = handinfo.Padding*2;

    arrGoriz[0].Y = (int)(bG * arrGoriz[0].X + aG);
    arrGoriz[1].Y = (int)(bG * arrGoriz[1].X + aG);

    arrVert[0].X = 0;
    arrVert[1].X = handinfo.Padding*2;

    arrVert[0].Y = (int)((arrVert[0].X - aV) / bV);
    arrVert[1].Y = (int)((arrVert[1].X - aV) / bV);
}

Vector vertical = new Vector(0,1);
Vector gorizont = new Vector(1,0);
Vector vVert;
Vector vGoriz;
if (arrGoriz[0].X < arrGoriz[1].X)
    vGoriz = new Vector(arrGoriz[1].X - arrGoriz[0].X, arrGoriz[1].Y - arrGoriz[0].Y);
else
    vGoriz = new Vector(arrGoriz[0].X - arrGoriz[1].X, arrGoriz[0].Y - arrGoriz[1].Y);
if (arrVert[0].Y < arrVert[1].Y)
    vVert = new Vector(arrVert[1].X - arrVert[0].X, arrVert[1].Y - arrVert[0].Y);
else
    vVert = new Vector(arrVert[0].X - arrVert[1].X, arrVert[0].Y - arrVert[1].Y);

var angleVert = (int)Math.Abs(Vector.AngleBetween(vertical, vVert));
var angleGoriz = (int)Math.Abs(Vector.AngleBetween(gorizont, vGoriz));
```

Рисунок 33 – Часть кода с методом аппроксимации прямой

Для распознавания степени сжатия кисти, находится окружность вокруг кисти. Окружность строится по трем точкам, выбранным из набора контрольных точек пальцем, где обязательно одна из этих точек большой палец. При сжатии пальцев радиус окружности уменьшается, а при закрытии кисти радиус становится ноль. На рис. 34 представлен пример распознавания сжатия кисти.

Рисунок 34 – Сжатие кисти

После получения всех данных (контрольные точки пальцев, точка большого пальца, линия горизонта и вертикали, окружность кисти) производиться классификация жеста по заданным параметрам для одного из 5 вариантов.

ЗАКЛЮЧЕНИЕ

Целью данной работы являлось создание программного комплекса, состоящего из 3 частей (набор развивающих игр, система распознавания мелкой и крупной моторики, система мониторинга), направленных на реабилитацию больных, перенесших инсульт.

В ходе выполненной работы решены следующие задачи:

- проведен анализ классических методов реабилитации;
- проведен анализ существующих на рынке решений по реабилитации больных, перенесших инсульт;
- в процессе проведения анализа действующих решений были сформулированы требования к разрабатываемой системе;
- спроектирована база данных, обслуживающая систему;
- реализованы игры, обеспечивающие реабилитацию;
- проведен анализ методов распознавания мелкой моторики;
- проведен анализ технических средств;
- разработан алгоритм распознавания жестов [Приложение А].

Разработанный программный комплекс имеет следующие перспективы на будущее развития:

- добавление новых игр в имеющийся курс;
- монетизация за счет продажи подписок на курсы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Официальный сайт проекта «MIRA Rehab». [Электронный ресурс] URL: http://www.mirarehab.com. (дата обращения: 03.02.2018).

2. Официальный сайт проекта «Jintronix». [Электронный ресурс] URL: http://www.jintronix.com. (дата обращения: 03.02.2018).

3. Официальный сайт проекта «Habilect». [Электронный ресурс] URL: http://www.habilect.com. (дата обращения: 05.02.2018).

4. Interactive Virtual Aerobics Trainer. [Электронный ресурс] URL: http://www.cse.ohio-state.edu/~jwdavis/CVL/Research/VirtualAerobics/aerobics.html (дата обращения: 12.02.2018).

5. Freeman W.T., Roth M. Orientation Histograms for Hand Gesture Recognition // In International Workshop on Automatic Face and Gesture Recognition, 1994. — p. 296 – 301

6. Zhou H., Lin D.J., Huang T.S. Static hand gesture recognition based on local orientation histogram feature distribution model // in Proc. Conference on Computer vision and Pattern Recognition Workshop, vol. 10, 2004. — p. 161–169.

7. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In Proceedings of IEEE Computer Society Conference, pp. 886-893, 2005.

8. C.-C. Chen and J. Aggarwal. Recognizing human action from a far field of view. Motion and Video Computing, pp.1-7, 2009.

9. S. Satkin and M. Hebert. Modeling the temporal extent of actions. In Proceedings of European Conference on Computer Vision. pp. 536-548, 2010.

10. A. Agarwal and M. K. Thakur. Sign language recognition using Microsoft Kinect. In IEEE Sixth International Conference on Contemporary Computing, pp. 181-185, 2013.

ПРИЛОЖЕНИЕ А

Листинг исходного кода проекта TrackingHand

TrackingHand.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using Microsoft.Kinect;
using OpenCvSharp;
using System.Windows.Media.Media3D;
using Microsoft.Kinect.Input;
using Microsoft.Kinect.Toolkit.Input;
using OpenCvSharp.Extensions;
using OpenCvSharp.Utilities;
using MathNet.Numerics;

namespace HandTracking
{
    public class TrackingHand
    {

        public struct HandType
        {
            public static readonly int LeftHand = 1;
            public static readonly int RightHand = 2;
        }

        public struct TypeState
        {
            public static readonly int Open = 1;
            public static readonly int Close = 2;
            public static readonly int Plucked = 3;
        }
        public class HandState
        {
            public int State;
        }

        public HandState LeftHand = new HandState();
        public HandState RightHand = new HandState();
        public int ActiveHand;
        public enum TypeGrip
        {
            Sphere = 1,
            Cylinder,
            Plucked,
            Open,
            Close
        }
```

```
public WriteableBitmap imageHand;

public class Circle
{
    public int X;
    public int Y;
    public int Rad;

    public Circle(int X, int Y, int Rad)
    {
        this.X = X;
        this.Y = Y;
        this.Rad = Rad;
    }
}

public  class HandInfo
{
    public class ProbabilityGrip
    {
        public double Sphere;
        public double Cylinder;
        public bool Plucked;
        public bool Open;
        public bool Close;
    }

    public class LevelDepth
    {
        public int countLevel = 10;
        public float step = 8; // millimeters
        public int currentLevel = 0;
        public float[] arrContourLength = new float[10];
        public float[] arrContourSquare = new float[10];
        public int[] arrContourCount = new int[10];

        public void CleanArr()
        {
            for (int i = 0; i < 10; i++)
            {
                arrContourCount[i] = 0;
                arrContourLength[i] = 0;
                arrContourSquare[i] = 0;
            }
        }

    }

    public struct Position
    {
        public int X;
        public int Y;
        public float Z;
    }

    public IplImage imgHand = new IplImage(new CvSize(400, 400), BitDepth.U8, 3);
    public IplImage imgFingers = new IplImage(new CvSize(400, 400), BitDepth.U8, 3);
    public IplImage imgState = new IplImage(new CvSize(800, 100), BitDepth.U8, 3);
    public IplImage imgLevel = new IplImage(new CvSize(1600, 640), BitDepth.U8, 1);
    public IplImage imgContour;

    public Circle midPointCircle;

    public Position CenterPosition;
```

```
        public bool IsTracked = false;
        public int Padding;
        public int width;
        public int height;
        public float X;
        public float Y;
        public float midSquareFinger;
        public float koefSquare = 30;

        public double angleGoriz = 0;
        public double angleVert = 0;
        public bool GorizBlock = false;
        public bool VertBlock = false;
        public double lastLengthBigFinger = 0;
        public double lengthBigFinger = 0;
        public int currentState = 0;
        public int lastState = 0;
        public int countState = 0;
        public Vector lastVectGrip = new Vector();
        public Vector vectGrip = new Vector();

        public CvPoint[] pointOutsideFingers = null;
        public CvPoint[] pointInsideFingers = null;
        public CvPoint[] pointFingers = null;
        public Vector[] pointVector;

        public bool changeBigFinger = false;
        public bool pluckedGrip = false;
        public bool pointInRect = false;
        public CvRect pluckedRect = new CvRect();
        public int countFingers = 0;

        public int indexBigFingers = -1;
        public int handType;



        public LevelDepth levelDepth = new LevelDepth();
        public ProbabilityGrip probability = new  ProbabilityGrip();

        public HandInfo(int handType, int frameWidth, int frameHeight)
        {
            this.handType = handType;
            imgContour = Cv.CreateImage(new CvSize(200, 200), BitDepth.U8, 1);
        }



    }


    private List<Circle> arrCircle = new List<Circle>();
    public event method ChangeState;
    public delegate void method(int state, int handState);



    ushort[] depthData;
    private byte[] depthPixels;
    private byte[] depthFingersPixels;

    int frameWidth;
    int frameHeight;
    CvPoint3D32f pointHand;
```

```
        IplImage imgDepth;
        IplImage imgDepthFingers;
        private FileInfo file;
        private FileInfo file2;
        public HandInfo handinfo = null;
        private HandInfo leftHandInfo;
        private HandInfo rightHandInfo;
        public TrackingHand(int width, int height)
        {

            frameWidth = width;
            frameHeight = height;
            imgDepth = Cv.CreateImage(new CvSize(frameWidth, frameHeight), BitDepth.U8, 1);
            imgDepthFingers = Cv.CreateImage(new CvSize(frameWidth, frameHeight), BitDepth.U8,
1);
            this.depthPixels = new byte[frameWidth * frameHeight];
            this.depthFingersPixels = new byte[frameWidth * frameHeight];


            leftHandInfo = new HandInfo(HandType.LeftHand, frameWidth, frameHeight);
            rightHandInfo = new HandInfo(HandType.RightHand, frameWidth, frameHeight);


        }



        public bool GetGripState(TypeGrip typeGrip)
        {
            if (typeGrip == TypeGrip.Sphere)
            {
                if (handinfo.probability.Sphere > 70 && handinfo.probability.Open == false)
                    return true;
                else
                    return false;
            }
            if (typeGrip == TypeGrip.Cylinder)
            {
                if (handinfo.probability.Cylinder > 70 && handinfo.probability.Open == false)
                    return true;
                else
                    return false;
            }
            if (typeGrip == TypeGrip.Plucked && handinfo.probability.Close == false)
            {
                if(handinfo.probability.Plucked)
                    return true;
                else
                    return false;

            }
            if (typeGrip == TypeGrip.Open)
            {

            }
            if (typeGrip == TypeGrip.Close)
            {
            }
            return false;
        }

        public bool GetLooseState(TypeGrip typeGrip)
        {
            if (typeGrip == TypeGrip.Sphere)
```

```
            {
                if (handinfo.probability.Sphere < 55 || handinfo.probability.Open)
                    return true;
                else
                    return false;
            }
            if (typeGrip == TypeGrip.Cylinder)
            {
                if (handinfo.probability.Cylinder < 55 || handinfo.probability.Open)
                    return true;
                else
                    return false;
            }
            if (typeGrip == TypeGrip.Plucked)
            {
                if (handinfo.probability.Open || handinfo.pointFingers?.Length == 5)
                    return true;
                else
                    return false;
            }
            if (typeGrip == TypeGrip.Open)
            {

            }
            if (typeGrip == TypeGrip.Close)
            {
            }
            return false;
        }

        public void CloseWindows()
        {
            //LeftHandWindows.Close();
            //RightHandWindows.Close();
            //FingersLeftHandWindows.Close();
            //FingersRightHandWindows.Close();
            //LeftStateWindows.Close();
            //RightStateWindows.Close();
        }

        private void DrawDepthPixel(ref HandInfo hand, CvPoint3D32f point, int Padding, int
width)
        {
            ushort min = 8000;
            ushort max = (ushort)point.Z;
            for (int y = (int)point.Y - Padding; y < (int)point.Y + Padding; y++)
            {
                for (int x = (int)point.X - Padding; x < (int)point.X + Padding; x++)
                {
                    if (x < 0) x = 0;
                    if (y < 0) y = 0;
                    var index = y * width + x;
                    if (index >= depthData.Length)
                        index = depthData.Length - 1;
                    var depth = depthData[index];
                    if (depth < point.Z && depth < min && depth != 0)
                        min = depth;

                    this.depthPixels[index] =
                        (byte)(depth >= (int)(point.Z) - 150 && depth <= (int)(point.Z) + 30 ?
(255) : 0);

                }
            }
```

```
            hand.levelDepth.CleanArr();
            for (int k = 3; k < hand.levelDepth.countLevel-2; k++)
            {
                byte[] depthFingersLevelPixels = new byte[200 * 200];
                int y1 = 0;
                for (int y = (int)point.Y - Padding; y < (int)point.Y + Padding; y++)
                {
                    int x1 = 0;
                    for (int x = (int)point.X - Padding; x < (int)point.X + Padding; x++)
                    {
                        if (x < 0) x = 0;
                        if (y < 0) y = 0;
                        var index = y * width + x;
                        if (index >= depthData.Length)
                            index = depthData.Length - 1;
                        var depth = depthData[index];

                        var index2 = y1 * 200 + x1;
                        if (index2 >= depthFingersLevelPixels.Length)
                            index2 = depthFingersLevelPixels.Length - 1;
                        depthFingersLevelPixels[index2] =
                            (byte)
                                (depth <= (int)(point.Z) - (hand.levelDepth.step *
(hand.levelDepth.countLevel - k)) &&
                                 (depth != 0)
                                    ? (255)
                                    : 0);
                        x1++;
                    }
                    y1++;
                }



                IplImage dst = new IplImage(new CvSize(hand.imgContour.Width,
hand.imgContour.Height), BitDepth.U8, 1);
                IplImage imgOut = new IplImage(new CvSize(320, 320), BitDepth.U8, 1);

                dst.Zero();
                imgOut.Zero();




                var ptr = hand.imgContour.ImageData;
                for (int i = 0; i < depthFingersLevelPixels.Length; i++)
                {
                    Marshal.WriteByte(ptr, i, depthFingersLevelPixels[i]);
                }

                dst = Cv.CloneImage(hand.imgContour);
                Cv.Smooth(dst, dst, SmoothType.Median, 3, 3);


                int count = 0;
                var storage = new CvMemStorage(0);
                CvContourScanner scanner = Cv.StartFindContours(dst, storage, CvCon-
tour.SizeOf, ContourRetrieval.List,
                    ContourChain.ApproxNone);
                CvSeq<CvPoint> c = null;
                while (true)
                {
                    c = Cv.FindNextContour(scanner);
```

```
                    if (c != null)
                    {
                        float len = (float)c.ArcLength();
                        float area = Math.Abs((float)c.ContourArea());
                        if (len > 2/* && TestSquare(point.Z/1000,area)*/)
                        {

                            if(hand.levelDepth.arrContourSquare[k-1] == 0)
                            hand.levelDepth.arrContourSquare[k - 1] += area;
                            else
                            {
                                hand.levelDepth.arrContourSquare[k - 1] += area;
                                hand.levelDepth.arrContourSquare[k - 1] =
hand.levelDepth.arrContourSquare[k - 1]/2;
                            }
                            hand.levelDepth.arrContourLength[k - 1] += len;
                            count++;
                        }
                    }
                    else
                    {
                        hand.levelDepth.arrContourCount[k - 1] = count;
                        break;
                    }
                }
                Cv.EndFindContours(scanner);

                if (k < 5)
                    Cv.SetImageROI(hand.imgLevel, (new CvRect((k - 1) * 320, 0, 320, 320)));
                else
                {
                    }
                Cv.Resize(hand.imgContour, imgOut);
                Cv.PutText(imgOut, point.Z.ToString(), new CvPoint(10, 20), new
CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
                Cv.PutText(imgOut, hand.levelDepth.arrContourSquare[k - 1].ToString(), new
CvPoint(10, 45), new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
                Cv.PutText(imgOut, count.ToString(), new CvPoint(10, 65), new
CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
                Cv.Copy(imgOut, hand.imgLevel);
            }

            float maxT = 0;
            int indexT = 0;
            for (int i = 0; i < hand.levelDepth.arrContourCount.Length; i++)
            {
                if (hand.levelDepth.arrContourCount[i] > maxT &&
hand.levelDepth.arrContourCount[i] <= 5)
                {
                    maxT = hand.levelDepth.arrContourCount[i];
                    indexT = i;
                    continue;
                }
                if (hand.levelDepth.arrContourCount[i] == maxT)
                {
                    if (hand.levelDepth.arrContourSquare[i] >
hand.levelDepth.arrContourSquare[indexT])
                        indexT = i;

                }
            }

            if (indexT > 6)
                indexT = 6;
```

```
            if (hand.levelDepth.arrContourSquare[indexT] <= (75 - (Math.Pow(point.Z / 100, 2)
/ 4)))
                hand.midSquareFinger = hand.levelDepth.arrContourSquare[indexT];


            for (int y = (int)point.Y - Padding; y < (int)point.Y + Padding; y++)
            {
                for (int x = (int)point.X - Padding; x < (int)point.X + Padding; x++)
                {
                    if (x < 0) x = 0;
                    if (y < 0) y = 0;
                    var index = y * width + x;
                    if (index >= depthData.Length)
                        index = depthData.Length - 1;
                    var depth = depthData[index];

                    this.depthFingersPixels[index] =
                        (byte)
                            (depth <= (int)(point.Z) - (hand.levelDepth.step *
(hand.levelDepth.countLevel - indexT - 1)) &&
                             (depth != 0)
                                ? (255)
                                : 0);
                }
            }

            //LeftHandWindows.ShowImage(hand.imgContour);

            Cv.ResetImageROI(hand.imgLevel);
            Cv.PutText(hand.imgLevel, hand.midSquareFinger.ToString(), new CvPoint(10, 20),
new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            Cv.PutText(hand.imgLevel, (75 - (Math.Pow(point.Z/100,2)/4)).ToString(), new
CvPoint(10, 45), new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            //LeftLevelWindows.ShowImage(handinfo.imgLevel);
            hand.imgLevel.Zero();


        }

        public CvPoint ComputeRadius(CvPoint a, CvPoint b, CvPoint c, ref int rad)
        {
            double x1 = a.X;
            double y1 = a.Y;
            double x2 = b.X;
            double y2 = b.Y;
            double x3 = c.X;
            double y3 = c.Y;
            double mr = (double)((y2 - y1) / (x2 - x1));
            double mt = (double)((y3 - y2) / (x3 - x2));

            double xc = (double)((mr * mt * (y3 - y1) + mr * (x2 + x3) - mt * (x1 + x2)) / (2
* (mr - mt)));

            double yc = (double)((-1 / mr) * (xc - (x1 + x2) / 2) + (y1 + y2) / 2);
            double d = (xc - x1) * (xc - x1) + (yc - y1) * (yc - y1);
            rad  = (int)Math.Sqrt(d);
            //return Math.Sqrt(d);
            return new CvPoint((int)xc, (int)yc);
        }

        private bool TestSquare(float Z, float Square)
        {
            float b = 1/(Z/30);
            float proc = 0.5f;
```

```
            if (Z > 1200)
                proc = 0.6f;
            if (Z > 1300)
                proc = 0.8f;
            if (Z > 1400)
                proc = 0.9f;
            if (Z > 1500)
                proc = 1;
            if (Z > 1600)
                proc = 1.1f;


            if (Math.Abs(1 - Square/b) <= proc)
                return true;
            else
                return false;

        }

        private CvPoint DepthMin(CvRect rect, CvRect rectImg, int width)
        {
            ushort depth_min = 8000;
            int X_min = 0, Y_min = 0;
            for (int y = rect.Y; y < rect.Y + rect.Height; y++)
                for (int x = rect.X; x < rect.X + rect.Width; x++)
                {
                    var index = (y + rectImg.Y) * width + (x + rectImg.X);
                    var depth = depthData[index];
                    if (depth < depth_min && depth != 0)
                    {
                        depth_min = depth;
                        X_min = x;
                        Y_min = y;
                    }
                }
            return new CvPoint(X_min, Y_min);
        }

        CvSeq<CvPoint> FindContour(IplImage img, IplImage imgContour,ref CvMemStorage storage)
        {
            IplImage dst = new IplImage(new CvSize(img.Width, img.Height), BitDepth.U8, 1);
            dst = Cv.CloneImage(img);
            CvContourScanner scanner = Cv.StartFindContours(dst, storage, CvContour.SizeOf,
ContourRetrieval.List,
            ContourChain.ApproxNone);
            CvSeq<CvPoint> c = null;
            while (true)
            {
                c = Cv.FindNextContour(scanner);
                if (c != null)
                {
                    if ((int)Cv.PointPolygonTest(c, new CvPoint(img.Width / 2, img.Height /
2), false) == 1)
                    {
                        //Cv.DrawContours(imgLeftHand, c, CvColor.DimGray, CvColor.Coral, 2,
1);
                        Cv.DrawContours(imgContour, c, CvColor.DimGray, CvColor.Coral, 2, 1);
                        break;
                    }
                }
                else
                { break; }
            }
```

```
            Cv.EndFindContours(scanner);
            return c;
        }

        CvPoint[] SearchPointOutsideFingers(CvSeq<CvPoint> c, IplImage img)
        {
            int counterPoint = 0;
            int koef = 6;
            int angle = 80;
            CvPoint[] point = new CvPoint[500];
            int k = 0;

            if (c != null)
            {
                int len = 0;
                foreach (var contour in c)
                    len++;

                foreach (var contour in c)
                {


                    if (counterPoint < koef)
                    {
                        counterPoint++;
                        continue;
                    }
                    Vector v1;
                    Vector v2;
                    if (counterPoint - 2 * koef < 0)
                    {
                        v1 = new Vector(c[len - 1 + (counterPoint - 2 * koef)].Value.X -
c[counterPoint - koef].Value.X, c[len - 1 + (counterPoint - 2 * koef)].Value.Y -
c[counterPoint - koef].Value.Y);
                        v2 = new Vector(c[counterPoint].Value.X - c[counterPoint -
koef].Value.X, c[counterPoint].Value.Y - c[counterPoint - koef].Value.Y);
                    }
                    else
                    {
                        v1 = new Vector(c[counterPoint - 2 * koef].Value.X - c[counterPoint -
koef].Value.X, c[counterPoint - 2 * koef].Value.Y - c[counterPoint - koef].Value.Y);
                        v2 = new Vector(c[counterPoint].Value.X - c[counterPoint -
koef].Value.X, c[counterPoint].Value.Y - c[counterPoint - koef].Value.Y);
                    }

                    double angleBetween;
                    angleBetween = Vector.AngleBetween(v1, v2);
                    if (angleBetween == 0)
                    {
                        counterPoint++;
                        continue;
                    }
                    if ((angleBetween < angle && angleBetween > 0))
                    {
                        point[k++] = c[counterPoint - koef].Value;
                    }
                            counterPoint++;
}
            }



            CvPoint[] pointFingers = new CvPoint[100];
```

```
            int count = 0;
            for (int i = 0; i < k - 1; i++)
            {
                for (int j = 0; j < pointFingers.Length; j++)
                {
                    if (point[i].Y > img.Height - 10)
                    { continue; }
                    if (point[i].Y > img.Height * ((float)3 / 5))
                    { continue; }
                    if (pointFingers[j].X == 0 && pointFingers[j].Y == 0)
                    {
                        pointFingers[j] = point[i];
                        count++;
                        break;
                    }
                    else
                    {
                        if ((Math.Abs(point[i].X - pointFingers[j].X) <= img.Width * 0.05) &&
(Math.Abs(point[i].Y - pointFingers[j].Y) <= img.Height * 0.05))
                        {
                            pointFingers[j].X = (pointFingers[j].X + point[i].X) / 2;
                            pointFingers[j].Y = (pointFingers[j].Y + point[i].Y) / 2;
                            break;
                        }
                    }
                }
            }

            CvPoint[] t = new CvPoint[count];
            for (int i = 0; i < count; i++)
                t[i] = pointFingers[i];
            return t;
        }

        CvPoint[] SearchPointInsideFingers(IplImage img, IplImage img2, CvRect r)
        {
            CvPoint[] pointInsideFingers = new CvPoint[10];
            IplImage dst = new IplImage(new CvSize(img.Width, img.Height), BitDepth.U8, 1);
            dst = Cv.CloneImage(img);
            var storage = new CvMemStorage(0);
            CvContourScanner scanner = Cv.StartFindContours(dst, storage, CvContour.SizeOf,
ContourRetrieval.List,
            ContourChain.ApproxSimple);
            CvSeq<CvPoint> c = null;
            int count = 0;
            while (true)
            {
                c = Cv.FindNextContour(scanner);
                if (c != null)
                {
                    CvRect rectContour = Cv.BoundingRect(c);
                    Cv.DrawRect(img, rectContour, CvColor.White);
                    //Cv.DrawRect(img2, rectContour, CvColor.Red);

                    if (count < pointInsideFingers.Length)
                        //pointInsideFingers[count++] = DepthMin(img, rectContour, r);
                        Cv.DrawCircle(img, pointInsideFingers[count - 1], 2, CvColor.Red);

                }
                else
                { break; }
            }
            Cv.EndFindContours(scanner);
            CvPoint[] t = new CvPoint[count];
```

```
            for (int i = 0; i < count; i++)
                t[i] = pointInsideFingers[i];
            return t;
        }

        void SearchPointFingers(ref HandInfo hand, ref IplImage imgOutside, ref IplImage im-
gOutsideContour, ref IplImage imgInside, CvRect r, int width)
        {
            var storage = new CvMemStorage(0);
            CvSeq<CvPoint> c = FindContour(imgOutside, imgOutsideContour,ref storage);
            Cv.FloodFill(imgOutsideContour, new CvPoint(imgOutside.Width / 2, imgOut-
side.Height / 2), CvColor.White);

            IplImage tmp = new IplImage(new CvSize(imgDepth.ROI.Width, imgDepth.ROI.Height),
BitDepth.U8, 1);
            tmp.Zero();
            imgOutside.Copy(tmp, imgOutsideContour);
            imgOutside.Zero();
            imgOutside = Cv.CloneImage(tmp);

            IplImage temp = new IplImage(new CvSize(imgDepth.ROI.Width, imgDepth.ROI.Height),
BitDepth.U8, 3);
            Cv.CvtColor(imgOutside, temp, ColorConversion.GrayToBgr);
            imgOutside = new IplImage(new CvSize(imgDepth.ROI.Width, imgDepth.ROI.Height),
BitDepth.U8, 3);
            imgOutside.Zero();
            imgOutside = Cv.CloneImage(temp);

            hand.pointOutsideFingers = SearchPointOutsideFingers(c, imgOutside);
            CvPoint[] pointTemp = new CvPoint[hand.pointOutsideFingers.Length];

            for (int i = 0; i < hand.pointOutsideFingers.Length; i++)
                pointTemp[i] = hand.pointOutsideFingers[i];

            double[] square = new double[10];
            CvPoint[] pointInsideFingerstemp = new CvPoint[10];
            CvPoint[] pointInsideFingerstemp2 = new CvPoint[10];
            CvPoint[] pointFingerstemp = new CvPoint[20];
            IplImage dst = new IplImage(new CvSize(imgInside.Width, imgInside.Height), Bit-
Depth.U8, 1);
            dst = Cv.CloneImage(imgInside);
            var storage2 = new CvMemStorage(0);
            CvContourScanner scanner = Cv.StartFindContours(dst, storage2, CvContour.SizeOf,
ContourRetrieval.List,
            ContourChain.ApproxSimple);
            CvSeq<CvPoint> c2 = null;
            int count = 0;
            int countFing = 0;
            while (true)
            {
                c2 = Cv.FindNextContour(scanner);
                if (c2 != null)
                {
                    if (pointTemp.Length != 0)
                    {
                        int d = (int)Cv.PointPolygonTest(c2, pointTemp[0], false);
                    }

                    CvRect rectContour = Cv.BoundingRect(c2);
                    Cv.DrawRect(imgInside, rectContour, CvColor.White);
                    //Cv.DrawRect(img2, rectContour, CvColor.Red);
                    if (c2.ArcLength() < 5)
                        continue;
```

```
                    var sq = Math.Abs(c2.ContourArea());


                    if (count < pointInsideFingerstemp.Length)
                    {
                        pointInsideFingerstemp[count++] = DepthMin(rectContour, r, width);
                        pointInsideFingerstemp2[count - 1] = pointInsideFingerstemp[count -
1];
                        //int d = (int)Cv.PointPolygonTest(c2, pointInsideFingerstemp[count -
1], false);
                        square[count - 1] = sq;

                    }

                    for (int i = 0; i < pointTemp.Length; i++)
                    {
                        if (pointTemp[i].X == 0 && pointTemp[i].Y == 0)
                        { continue; }



                        if (Math.Abs(pointTemp[i].X - pointInsideFingerstemp[count - 1].X) <
(imgInside.Width * 0.05f) && Math.Abs(pointTemp[i].Y - pointInsideFingerstemp[count - 1].Y) <
(imgInside.Height * 0.05f))
                        //if((int)Cv.PointPolygonTest(c2, new
CvPoint(pointTemp[i].X,pointTemp[i].Y+1), false) == 1)
                        {
                            if (countFing < pointFingerstemp.Length)
                            pointFingerstemp[countFing++] = pointTemp[i];
                            pointTemp[i].X = 0;
                            pointTemp[i].Y = 0;
                            pointInsideFingerstemp[count - 1].X = 0;
                            pointInsideFingerstemp[count - 1].Y = 0;
                            break;
                        }
                    }


                    Cv.DrawCircle(imgInside, pointInsideFingerstemp[count - 1], 2, CvCo-
lor.Red);

                }
                else
                { break; }
            }

            Cv.EndFindContours(scanner);


            hand.pointInsideFingers = new CvPoint[count];
            for (int i = 0; i < count; i++)
                hand.pointInsideFingers[i] = pointInsideFingerstemp2[i];

            for (int i = 0; i < pointTemp.Length; i++)
            {
                if (countFing < pointFingerstemp.Length)
                    if (pointTemp[i].X != 0 && pointTemp[i].Y != 0)
                    {
                        pointFingerstemp[countFing++] = pointTemp[i];
                    }
            }
            for (int i = 0; i < pointInsideFingerstemp.Length; i++)
            {
```

```
            if(countFing < pointFingerstemp.Length)
                if (pointInsideFingerstemp[i].X != 0 && pointInsideFingerstemp[i].Y != 0)
                {
                    pointFingerstemp[countFing++] = pointInsideFingerstemp[i];
                }
        }
        // Отбор лишних точек

        hand.pointFingers = new CvPoint[countFing];
        for (int i = 0; i < countFing; i++)
        {
            if (pointFingerstemp[i].X != 0 && pointFingerstemp[i].Y != 0)
                hand.pointFingers[i] = pointFingerstemp[i];
        }




        if (hand.pointFingers.Length != 0)
        {

            int index = SearchBigFinger(hand.pointFingers, hand);

            if (index != -1)
                for (int i = 0; i < hand.pointInsideFingers.Length; i++)
                {
                    if (hand.pointFingers[index] == hand.pointInsideFingers[i])
                    {
                        if (hand.lengthBigFinger != 0)
                            hand.lastLengthBigFinger = hand.lengthBigFinger;
                        else
                            hand.lastLengthBigFinger = square[i];
                        hand.lengthBigFinger = square[i];
                    }
                }
        }


    }

    int SearchBigFinger(CvPoint[] pointFingers, HandInfo hand)
    {
        int mid_x = 0, mid_y = 0;
        double len = 0;
        int k = -1;
        for (int i = 0; i < pointFingers.Length; i++)
        {
            mid_x += pointFingers[i].X;
            mid_y += pointFingers[i].Y;
        }
        mid_x = mid_x / pointFingers.Length;
        mid_y = mid_y / pointFingers.Length;

        for (int i = 0; i < pointFingers.Length; i++)
        {
            if (hand.handType == HandType.LeftHand)
                if (pointFingers[i].Y > hand.Padding - 30)
                {
                    Vector v = new Vector(pointFingers[i].X - mid_x, pointFingers[i].Y -
mid_y);
                    if (len < v.Length)
                    {
                        len = v.Length;
                        k = i;
```

```
                    }
                }
            if (hand.handType == HandType.RightHand)
            {
                if ( pointFingers[i].Y > hand.Padding - 30)
                {
                    Vector v = new Vector(pointFingers[i].X - mid_x, pointFingers[i].Y -
mid_y);
                    if (len < v.Length)
                    {
                        len = v.Length;
                        k = i;
                    }
                }
            }
        }

        return k;
    }

    private void Hand_changeState(ref HandInfo hand, int state, ref IplImage imgState)
    {
        imgState.Zero();

        if (state == 1)
            //Cv.PutText(imgState, "Open hand grip", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Yellow);
            Cv.PutText(imgState, "Open", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Yellow);
        if (state == 2)
            //Cv.PutText(imgState, "Closed hand grip", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Red);
            Cv.PutText(imgState, "Close", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Red);
        if (state == 3)
            //Cv.PutText(imgState, "Open hand", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Green);
            Cv.PutText(imgState, "Plucked Grip!!!!", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.BlueViolet);
        if (state == 4)
            Cv.PutText(imgState, "Closed fingers", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Aquamarine);
        //if (state == 5)
        //    Cv.PutText(imgState, countFingers.ToString() + "Fingers", new CvPoint(0,
70), new CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.Aquamarine);
        if (state == 6)
            Cv.PutText(imgState, "Plucked Grip!!!!", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.CadetBlue);
        if (state == 0)
            Cv.PutText(imgState, "Unknown", new CvPoint(0, 70), new
CvFont(FontFace.HersheyDuplex, 3f, 1f), CvColor.BlueViolet);

        hand.imgState.Zero();
        hand.imgState = Cv.CloneImage(imgState);


    }

    int CheckState(ref HandInfo hand, CvPoint[] pointFingers, CvPoint[] pointInsideFin-
gers, CvPoint pointHand, ref bool pluckedGrip)
    {
        if ((pluckedGrip && hand.midSquareFinger / hand.lengthBigFinger < 1.4f) || (point-
Fingers.Length == 5) || (pointInsideFingers.Length == 5))
        {
```

```
                pluckedGrip = false;
            }
            if (!pluckedGrip)
            {
                if (pointFingers.Length == 0 && pointInsideFingers.Length == 1)
                {
                    Vector v = new Vector(pointInsideFingers[0].X - pointHand.X, pointInside-
Fingers[0].Y - pointHand.Y);
                    if (v.Length > pointHand.X / 2.7f)
                    {
                        //lastState = 4;
                        //return 4;
                    }

                    hand.currentState = 2;
                    return 2;
                }
                if (hand.lengthBigFinger / hand.midSquareFinger > 1.4f &&
/*hand.lastVectGrip.Length < 10 && hand.lastVectGrip.Length != 0 &&*/ pointInsideFin-
gers.Length != 1 && pointFingers.Length != 5)
                {
                    pluckedGrip = true;
                    hand.currentState = 3;
                    return 3;
                }
                // Открытый захват
                if (pointFingers.Length >= 2 && pointInsideFingers.Length >= 4)
                {
                    hand.currentState = 1;
                    return 1;
                }
                //Закрытый захват

                if (pointFingers.Length == 0 && pointInsideFingers.Length > 2)
                {
                    hand.currentState = 1;
                    return 1;
                }
                //Открытая рука
                if (pointFingers.Length >= 4 && pointInsideFingers.Length <= 2)
                {
                    hand.currentState = 1;
                    return 1;
                }

            }
            return hand.currentState;
        }

        bool CheckPointInRect(CvPoint p, CvRect r)
        {
            if (p.X > r.X && p.Y > r.Y && p.X < r.X + r.Width && p.Y < r.Y + r.Height)
                return true;
            else
                return false;
        }

        void CheckPlucked(ref HandInfo hand, IplImage img, CvPoint[] pointFingers, int index-
BigFingers, ref CvRect pluckedRect)
        {
            if (hand.handType == HandType.LeftHand)
                pluckedRect = new CvRect(pointFingers[indexBigFingers].X - 25, pointFin-
gers[indexBigFingers].Y - 25, 30, 30);
            if (hand.handType == HandType.RightHand)
```

```
                pluckedRect = new CvRect(pointFingers[indexBigFingers].X - 5, pointFin-
gers[indexBigFingers].Y - 25, 30, 30);
            hand.lastVectGrip = hand.vectGrip;
            hand.vectGrip = new Vector();
            for (int i = 0; i < pointFingers.Length; i++)
            {
                if (CheckPointInRect(pointFingers[i], pluckedRect) && i != indexBigFingers)
                {
                    //pointInRect = true;
                    //pluckedGrip = false;
                    Vector v = new Vector(pointFingers[i].X - pointFingers[indexBigFingers].X,
pointFingers[i].Y - pointFingers[indexBigFingers].Y);
                    if (v.Length < hand.vectGrip.Length || hand.vectGrip.Length == 0)
                        hand.vectGrip = new Vector(v.X, v.Y);
                    //img.DrawLine(pointFingers[i], pointFingers[indexBigFingers], CvCo-
lor.BlueViolet);
                }

            }
        }

        private void FindProbability()
        {

            double countFinger = 0;
            double angleGoriz;
            double angleVert;
            int count = 0;
            double koef1 = 0.4d;
            double koef2 = 0.8d;
            //Открытая рука

            if ((int)(Math.Abs(1000 - handinfo.CenterPosition.Z) / 46 + handin-
fo.midPointCircle.Rad) > 25)
            {
                handinfo.probability.Open = true;
                handinfo.probability.Plucked = false;
            }
            else
            {
                handinfo.probability.Open = false;
            }

            if (handinfo.midPointCircle.Rad == 0)
            { handinfo.probability.Close = true;
                handinfo.probability.Plucked = false;
            }
            else
            {
                handinfo.probability.Close = false;
            }
            if (((handinfo.midSquareFinger / handinfo.lengthBigFinger < 1.3f) && (handin-
fo.midSquareFinger / handinfo.lengthBigFinger > 0.7f)) || (handinfo.pointFingers.Length == 5)
|| (handinfo.pointInsideFingers.Length == 5))
                handinfo.probability.Plucked = false;
            else
            if (handinfo.lengthBigFinger / handinfo.midSquareFinger > 1.5f && handin-
fo.midSquareFinger != 0 && handinfo.lengthBigFinger < 300 &&
                /*hand.lastVectGrip.Length < 10 && hand.lastVectGrip.Length != 0 &&*/
                handinfo.pointInsideFingers.Length != 1 && handinfo.pointFingers.Length != 5
&& handinfo.probability.Open == false)
                handinfo.probability.Plucked = true;
```

```
            //Сферический захват
            if (handinfo.GorizBlock)
            {
                handinfo.probability.Sphere = 0;
            }
            else
            {
                for (int i = 0; i < handinfo.pointFingers.Length; i++)
                {
                    if (handinfo.pointFingers[i].Y < handinfo.Padding)
                        count++;
                }

                koef1 += (float) count / 6.6665f;
                koef2 += (float) count / 20;
                angleGoriz = (100 - handinfo.angleGoriz)/100;
                //handinfo.probability.SphereGrip = (int) ((angleGoriz*koef1)*100);
                handinfo.probability.Sphere = (int)((angleGoriz * 0.4f + koef2 * 0.6f) * 100);

            }
            //Цилиндрический захват
            count = 0;
            koef1 = 0.6d;
            koef2 = 0.8d;
            if (handinfo.VertBlock)
                handinfo.probability.Cylinder = 0;
            else
            {
                for (int i = 0; i < handinfo.pointInsideFingers.Length; i++)
                {
                    if(handinfo.handType == HandType.LeftHand)
                    if (handinfo.pointInsideFingers[i].X < handinfo.Padding)
                        count++;
                    if (handinfo.handType == HandType.RightHand)
                        if (handinfo.pointInsideFingers[i].X > handinfo.Padding)
                            count++;
                }
                koef1 += (float)count / 10;
                koef2 += (float)count / 20;
                angleVert = (100 - handinfo.angleVert) / 100;

                //handinfo.probability.Cylinder = (int)((angleVert * koef1) * 100);
                handinfo.probability.Cylinder = (int)((angleVert * 0.6f + koef2 * 0.4f) *
100);
            }

        }

        private CvPoint[] approx()
        {
            double[] xVal;
            double[] yVal;
            if (handinfo.pointFingers.Length > 2)
            {
                if (handinfo.indexBigFingers == -1)
                {
                    xVal = new double[handinfo.pointFingers.Length];
                    yVal = new double[handinfo.pointFingers.Length];
                }
                else
                {
                    if (handinfo.pointFingers.Length != 2)
                    {
```

```
                    xVal = new double[handinfo.pointFingers.Length - 1];
                    yVal = new double[handinfo.pointFingers.Length - 1];
                }
                else
                {
                    return null;
                }
            }
        }
        else
        {
            return null;
        }

        CvPoint[] arrVert = new CvPoint[2];
        CvPoint[] arrGoriz = new CvPoint[2];
        int count = 0;
        for (int i = 0; i < handinfo.pointFingers.Length; i++)
        {
            if (i == handinfo.indexBigFingers)
            {
                continue;
            }
            xVal[count] = handinfo.pointFingers[i].X;
            yVal[count] = handinfo.pointFingers[i].Y;
            count++;
        }
        if (xVal.Length >= 2)
        {
            Tuple<double, double> pGoriz = Fit.Line(xVal, yVal);
            Tuple<double, double> pVert = Fit.Line(yVal, xVal);
            double aV = pVert.Item1;
            double bV = pVert.Item2;
            double aG = pGoriz.Item1;
            double bG = pGoriz.Item2;
            arrGoriz[0].X = 0;
            arrGoriz[1].X = handinfo.Padding*2;

            arrGoriz[0].Y = (int)(bG * arrGoriz[0].X + aG);
            arrGoriz[1].Y = (int)(bG * arrGoriz[1].X + aG);

            arrVert[0].X = 0;
            arrVert[1].X = handinfo.Padding*2;

            arrVert[0].Y = (int)((arrVert[0].X - aV) / bV);
            arrVert[1].Y = (int)((arrVert[1].X - aV) / bV);
        }

        Vector vertical = new Vector(0,1);
        Vector gorizont = new Vector(1,0);
        Vector vVert;
        Vector vGoriz;
        if (arrGoriz[0].X < arrGoriz[1].X)
            vGoriz = new Vector(arrGoriz[1].X - arrGoriz[0].X, arrGoriz[1].Y - arrGo-
riz[0].Y);
        else
            vGoriz = new Vector(arrGoriz[0].X - arrGoriz[1].X, arrGoriz[0].Y - arrGo-
riz[1].Y);
        if (arrVert[0].Y < arrVert[1].Y)
            vVert = new Vector(arrVert[1].X - arrVert[0].X, arrVert[1].Y - arrVert[0].Y);
        else
            vVert = new Vector(arrVert[0].X - arrVert[1].X, arrVert[0].Y - arrVert[1].Y);

        var angleVert = (int)Math.Abs(Vector.AngleBetween(vertical, vVert));
```

```
        var angleGoriz = (int)Math.Abs(Vector.AngleBetween(gorizont, vGoriz));



        if (angleVert >= 35)
            handinfo.VertBlock = true;
        if (angleGoriz >= 47 && angleGoriz != 90)
            handinfo.VertBlock = false;
        if (!handinfo.VertBlock)
            handinfo.angleVert = angleVert;


        if (angleGoriz >= 59)
            handinfo.GorizBlock = true;
        if (angleVert >= 33 && angleVert != 90)
            handinfo.GorizBlock = false;
        if (!handinfo.GorizBlock)
            handinfo.angleGoriz = angleGoriz;


        if (!handinfo.GorizBlock)
            return arrGoriz;
        return arrVert;
    }

    private void SearchCircle()
    {
        Circle circ = null;

        if (handinfo.pointFingers.Length > 3)
        {
            int Xmin = 200;
            int Xmax = 0;
            int Ymin = 200;
            CvPoint XminPoint = new CvPoint();
            CvPoint XmaxPoint = new CvPoint();
            CvPoint YminPoint = new CvPoint();
            if(handinfo.indexBigFingers != -1)
            XmaxPoint = handinfo.pointFingers[handinfo.indexBigFingers];
            else
                foreach (var point in handinfo.pointFingers)
                {
                    if (point.X > Xmax )
                    {
                        Xmax = point.X;
                        XmaxPoint = point;
                    }
                }
            foreach (var point in handinfo.pointFingers)
            {
                if (point.X < Xmin && point != XmaxPoint)
                {    Xmin = point.X;
                    XminPoint = point;
                }
            }


            foreach (var point in handinfo.pointFingers)
            {
                if (point.Y < Ymin && point != XminPoint && point != XmaxPoint)
                {
                    Ymin = point.Y;
                    YminPoint = point;
```

```
                    }
                }

                int rad = 0;
                var pointC = ComputeRadius(XminPoint, XmaxPoint, YminPoint, ref rad);
                if (pointC.X >= 0 && pointC.Y >= 0)
                {
                    //imgLeftHand.DrawCircle(pointC.X, pointC.Y, rad, CvColor.Red, 7);
                    circ = new Circle(pointC.X, pointC.Y, rad);
                }
            }

            if (handinfo.pointFingers.Length == 3)
            {
                int rad = 0;
                var pointC = ComputeRadius(handinfo.pointFingers[0], handinfo.pointFingers[1],
handinfo.pointFingers[2], ref rad);
                if (pointC.X >= 0 && pointC.Y >= 0)
                {
                    circ = new Circle(pointC.X, pointC.Y, rad);
                }
            }


            if (circ != null && circ.Rad < handinfo.Padding)
            {
                if (arrCircle.Count < 5)
                    arrCircle.Add(circ);
                else
                {
                    arrCircle.RemoveAt(0);
                    arrCircle.Add(circ);
                }

                Circle midPoint = new Circle(0, 0, 0);
                foreach (var p in arrCircle)
                {
                    midPoint.X += p.X;
                    midPoint.Y += p.Y;
                    midPoint.Rad += p.Rad;
                }
                midPoint.X = midPoint.X / arrCircle.Count;
                midPoint.Y = midPoint.Y / arrCircle.Count;
                midPoint.Rad = midPoint.Rad / arrCircle.Count;

                handinfo.midPointCircle = midPoint;
            }
            else
            {
                handinfo.midPointCircle = new Circle(0,0,0);
            }
        }

        public void Tracking(int handType, Point3D point, ushort[] depth)
        {

            if (handType == 1)
                handinfo = leftHandInfo;
            if (handType == 2)
                handinfo = rightHandInfo;
            if (handType != 1 && handType != 2)
                return;

            ActiveHand = handinfo.handType;
```

```
            depthData = new ushort[depth.Length];

            depth.CopyTo(depthData, 0);
            pointHand = new CvPoint3D32f(point.X, point.Y, point.Z);

            //
            IplImage imgState = new IplImage(new CvSize(800, 100), BitDepth.U8, 3);
            //

            GetInfoHand(handinfo, pointHand);
            DrawDepthPixel(ref handinfo, pointHand, handinfo.Padding, frameWidth);
            //Write.write("0)Получили градиент");


            var ptr = imgDepth.ImageData;
            for (int i = 0; i < depthPixels.Length; i++)
            {
                Marshal.WriteByte(ptr, i, depthPixels[i]);
            }
            if (handinfo.X >= 0 && handinfo.X < imgDepth.Width && handinfo.Y >= 0 && handin-
fo.Y < imgDepth.Height)
                imgDepth.ROI = new CvRect((int)handinfo.X, (int)handinfo.Y, handinfo.width,
handinfo.height);
            else
                return;
            IplImage imgLeftHand = new IplImage(new CvSize(imgDepth.ROI.Width, im-
gDepth.ROI.Height), BitDepth.U8, 1);
            IplImage imgContour = new IplImage(new CvSize(imgDepth.ROI.Width, im-
gDepth.ROI.Height), BitDepth.U8, 1);
            IplImage outImgHand = new IplImage(new CvSize(400, 400), BitDepth.U8, 1);
            imgContour.Zero();
            imgLeftHand = imgDepth.Clone(imgDepth.ROI);
            Cv.Smooth(imgLeftHand, imgLeftHand, SmoothType.Median, 3, 3);
            Cv.Resize(imgLeftHand, outImgHand);
            imageHand = outImgHand.ToWriteableBitmap(PixelFormats.Gray8);
            var ptr2 = imgDepthFingers.ImageData;

            for (int i = 0; i < depthFingersPixels.Length; i++)
            {
                Marshal.WriteByte(ptr2, i, depthFingersPixels[i]);
            }
            //Write.write("2)Копируем второе изображение");
            //Cv.SetData(imgDepthFingers, depthFingersPixels, frameWidth);
            if (handinfo.X >= 0 && handinfo.X < imgDepthFingers.Width && handinfo.Y >= 0 &&
handinfo.Y < imgDepthFingers.Height)
                imgDepthFingers.ROI = new CvRect((int)handinfo.X, (int)handinfo.Y, handin-
fo.width, handinfo.height);
            else
                return;
            IplImage imgLeftHandFingers = new IplImage(new CvSize(imgDepthFingers.ROI.Width,
imgDepthFingers.ROI.Height), BitDepth.U8, 1);
            IplImage imgContourFingers = new IplImage(new CvSize(imgLeftHandFingers.Width, im-
gLeftHandFingers.Height), BitDepth.U8, 1);
            IplImage outImgFingers = new IplImage(new CvSize(400, 400), BitDepth.U8, 1);
            imgLeftHandFingers.Zero();
            imgContourFingers.Zero();
            imgLeftHandFingers = imgDepthFingers.Clone(imgDepthFingers.ROI);
            Cv.Smooth(imgLeftHandFingers, imgLeftHandFingers, SmoothType.Median, 3, 3);
            Cv.Resize(imgLeftHandFingers, outImgFingers);

            handinfo.imgFingers.Zero();
            handinfo.imgFingers = Cv.CloneImage(outImgFingers);
```

```
            SearchPointFingers(ref handinfo, ref imgLeftHand, ref imgContour, ref imgLeftHand-
Fingers, imgDepthFingers.ROI, frameWidth);
            //Write.write("3)Распознали пальцы");
            if (handinfo.pointFingers.Length != 0)
            {
                handinfo.indexBigFingers = SearchBigFinger(handinfo.pointFingers, handinfo);


                if (handinfo.indexBigFingers != -1)
                {
                    imgLeftHand.DrawCircle(handinfo.pointFingers[handinfo.indexBigFingers], 4,
CvColor.Yellow);

                }

            }

            SearchCircle();

            CvPoint[] arr = new CvPoint[2];
            arr = approx();
            FindProbability();
            //Write.write("4)Определили вероятность захватов");
            int state = CheckState(ref handinfo, handinfo.pointOutsideFingers, handin-
fo.pointInsideFingers, new CvPoint(handinfo.Padding, handinfo.Padding), ref handin-
fo.pluckedGrip);


            if (arr != null)
            imgLeftHand.DrawLine(arr[0],
                    arr[1], CvColor.Blue);
            Cv.PutText(handinfo.imgLevel, "Gorizont: "+"-"+handinfo.angleGoriz.ToString()+"-",
new CvPoint(10, 70), new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            Cv.PutText(handinfo.imgLevel, "Vertical: " + "-"+handinfo.angleVert.ToString()+"-
", new CvPoint(10, 95), new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            Cv.PutText(handinfo.imgLevel, "Sphere: " + handin-
fo.probability.Sphere.ToString()+"%", new CvPoint(10, 120), new CvFont(FontFace.HersheyDuplex,
1f, 1f), CvColor.Aqua);
            Cv.PutText(handinfo.imgLevel, "Cylinder: " + handin-
fo.probability.Cylinder.ToString()+"%", new CvPoint(10, 145), new
CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            Cv.PutText(handinfo.imgLevel, "Rad: " + "-" + handin-
fo.midPointCircle.Rad.ToString() + "-", new CvPoint(10, 165), new
CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            Cv.PutText(handinfo.imgLevel, "RadMax: " + "-" + ((int)(Math.Abs(1000 - handin-
fo.CenterPosition.Z)/46 + handinfo.midPointCircle.Rad)).ToString() + "-", new CvPoint(10,
190), new CvFont(FontFace.HersheyDuplex, 1f, 1f), CvColor.Aqua);
            //imgLeftHand.DrawRect(handinfo.pluckedRect, CvColor.Blue);




            Hand_changeState(ref handinfo, state, ref imgState);

            if (handinfo.lastState == state)
                handinfo.countState++;
            else
            {
                handinfo.countState = 1;
                handinfo.lastState = state;
            }

            if (handinfo.countState >= 2)
```

```
            {
                if (state == 3)
                    handinfo.changeBigFinger = true;
                else
                {
                    handinfo.changeBigFinger = false;
                }

                //if (ChangeState != null) ChangeState(state, handType);

                        if (handType == 1)
    LeftHand.State = state;
                if (handType == 2)
                    RightHand.State = state;
            }

            //Write.write("5)Событие состояния руки");



            for (int i = 0; i < handinfo.pointFingers.Length; i++)
            {
                imgLeftHand.DrawLine(new CvPoint(handinfo.Padding, handinfo.Padding), handin-
fo.pointFingers[i], CvColor.Blue);
            }



            imgLeftHand.DrawCircle(new CvPoint(handinfo.Padding, handinfo.Padding), 3, CvCo-
lor.Red);

            if (handinfo.pointFingers.Length != 0)
            {
                imgLeftHand.DrawCircle(handinfo.pointFingers[0], 2, CvColor.Red);
                for (int i = 1; i < handinfo.pointFingers.Length; i++)
                {

                    if (handinfo.pointFingers[i].X != 0 && handinfo.pointFingers[i].Y != 0)
                    {
                        imgLeftHand.DrawCircle(handinfo.pointFingers[i], 2, CvColor.Red);
                    }
                }
            }

            IplImage outImg2 = new IplImage(new CvSize(400, 400), BitDepth.U8, 3);
            Cv.Resize(imgLeftHand, outImg2);

            handinfo.imgHand.Zero();
            handinfo.imgHand = Cv.CloneImage(outImg2);

            if (handinfo.handType == HandType.LeftHand)
            {
                //LeftHandWindows.ShowImage(handinfo.imgHand);
            }
            if (handinfo.handType == HandType.RightHand)
                //LeftHandWindows.ShowImage(handinfo.imgHand);



            for (int i = 0; i < depthFingersPixels.Length; i++)
            {
```

```
                depthFingersPixels[i] = 0;
            }

            //Write.write("6)Закончили распознавание");
            //Write.write("=========================================");
            //Write.write("=========================================");

            imgContour.Dispose();
            imgLeftHand.Dispose();
            imgContour.Dispose();
            outImgHand.Dispose();
            imgLeftHandFingers.Dispose();
            imgContourFingers.Dispose();
            outImgFingers.Dispose();
            outImg2.Dispose();

        }

        public void Close()
        {

        }

        private static void GetInfoHand(HandInfo hand, CvPoint3D32f pointHand)
        {
            int Padding = GetPadding(pointHand.Z);

            hand.IsTracked = true;
            hand.Padding = Padding;
            hand.CenterPosition.X = (int)pointHand.X;
            hand.CenterPosition.Y = (int)pointHand.Y;
            hand.CenterPosition.Z = (float)pointHand.Z;
            hand.X = (int)pointHand.X - Padding;
            hand.Y = (int)pointHand.Y - Padding;
            hand.width = Padding * 2;
            hand.height = Padding * 2;

            //Canvas.SetLeft(ell_L, pointHand.X - ell_L.Width);
            //Canvas.SetTop(ell_L, pointHand.Y - ell_L.Height);
        }

        private static int GetPadding(double Z)
        {
            if ((int)(Z / 10) < 95)
                return (155 - (int)(Z / 10));
            else
                return 60;
        }
    }
}
```

KinectKursorHand.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Kinect;
using PointF = System.Drawing.PointF;

namespace KinectCursorHand
{
    class KinectCursor
    {
        private CoordinateMapper coordinateMapper = null;
        private KinectSensor kinectSensor = null;
        private BodyFrameReader reader = null;
        private Body[] bodies;
        private static int padding;
        private ulong bodyId = 0;
        private int TypeHand;
        public delegate void change(System.Drawing.PointF pointCursor, ulong bodyId);
        public event change CursorMoved;
        private List<PointF> arrPoint = new List<PointF>();
        public KinectCursor()
        {
            this.kinectSensor = KinectSensor.GetDefault();
            this.coordinateMapper = this.kinectSensor.CoordinateMapper;
            reader = kinectSensor.BodyFrameSource.OpenReader();
            reader.FrameArrived+= ReaderOnFrameArrived;
            kinectSensor.Open();

        }
        private void ReaderOnFrameArrived(object sender, BodyFrameArrivedEventArgs bodyFra-
meArrivedEventArgs)
        {
            var reference = bodyFrameArrivedEventArgs.FrameReference.AcquireFrame();
            using (BodyFrame bodyFrame = reference)
            {
                if (bodyFrame != null)
                {
                    if (this.bodies == null)
                    {
                        this.bodies = new Body[bodyFrame.BodyCount];
                    }
                    bodyFrame.GetAndRefreshBodyData(this.bodies);

                    var pointCursor = GetHandPointCursor(bodies);
                    if(arrPoint.Count < 5)
                        arrPoint.Add(pointCursor);
                    else
                    {
                        arrPoint.RemoveAt(0);
                        arrPoint.Add(pointCursor);
                    }

                    PointF midPoint = new PointF();
                    foreach (var p in arrPoint)
                    {
                        midPoint.X += p.X;
                        midPoint.Y += p.Y;
```

```
                    }
                    midPoint.X = midPoint.X/arrPoint.Count;
                    midPoint.Y = midPoint.Y/arrPoint.Count;
                    if (pointCursor != new Point(0, 0))
                        if (CursorMoved != null) CursorMoved(midPoint, bodyId);
                }
            }
        }
        private PointF GetHandPointCursor(Body[] bodies)
        {
            int index = -1;
            float min_Z = 8000;
            int count = 0;
            int Padding_X = 0;
            int Padding_Y = 0;
            foreach (Body body in bodies)
            {
                if (body.IsTracked)
                {
                    if (bodyId == body.TrackingId)
                    {
                        index = count;
                        break;
                    }
                    if (body.Joints[JointType.SpineMid].Position.Z * 1000 < min_Z)
                    {
                        min_Z = body.Joints[JointType.SpineMid].Position.Z * 1000;
                        index = count++;
                    }
                    else
                    {
                        count++;
                    }
                }
                else
                    count++;
            }
            if (index == -1)
            {
                return new PointF();
            }
            ColorSpacePoint position_Hand;
            int X = 0;
            int Y = 0;
            float Xout = 0;
            float Yout = 0;
            var position_Spine = coordinateMap-
per.MapCameraPointToColorSpace(bodies[index].Joints[JointType.SpineMid].Position);
            var position_SpineShoulder = coordinateMap-
per.MapCameraPointToColorSpace(bodies[index].Joints[JointType.SpineShoulder].Position);
            if (padding != 0)
            {
                padding += (int)(position_Spine.Y - position_SpineShoulder.Y);
                padding = (int)(padding / 2);
            }
            else
            {
                padding = (int)(position_Spine.Y - position_SpineShoulder.Y);
            }
            Padding_X = padding * 3;
            Padding_Y = padding * 3;

            if (bodies[index].Joints[JointType.HandLeft].Position.Y >=
                bodies[index].Joints[JointType.HandRight].Position.Y)
```

```
            {
                position_Hand = coordinateMap-
per.MapCameraPointToColorSpace(bodies[index].Joints[JointType.HandLeft].Position);
                TypeHand = 0;
                X = (int)(position_SpineShoulder.X - 50 - position_Hand.X);
                Y = (int)(position_SpineShoulder.Y - position_Hand.Y);
                Xout = (float)Math.Round((1 - ((float)X / Padding_X)), 3);
                Yout = (float)Math.Round((1 - ((float)Y / Padding_Y)), 3);
            }
            else
            {
                position_Hand = coordinateMap-
per.MapCameraPointToColorSpace(bodies[index].Joints[JointType.HandRight].Position);
                TypeHand = 1;
                X = (int)(position_Hand.X - position_SpineShoulder.X - 50);
                Y = (int)(position_SpineShoulder.Y - position_Hand.Y);
                Xout = (float)Math.Round((((float)X / Padding_X)), 3);
                Yout = (float)Math.Round((1 - ((float)Y / Padding_Y)), 3);
            }


            if (Xout < 0) Xout = 0;
            if (Xout > 1) Xout = 1;
            if (Yout > 1) Yout = 1;
            if (Yout < 0) Yout = 0;
            bodyId = bodies[index].TrackingId;
            return new System.Drawing.PointF(Xout, Yout);
        }
    }
}
```