МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СИМВОЛЬНОГО РАСПОЗНАВАНИЯ РЕЛЬЕФНО-ТОЧЕЧНОГО ТАКТИЛЬНОГО ШРИФТА БРАЙЛЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Фишера Евгения Алексеевича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Очень часто у преподавателя, работающего в школе для детей с нарушениями зрения, возникает потребность во время урока беглым взглядом проверить классную или домашнюю работу обучающихся. К сожалению, у учителя на проверку одного такого задания уходит длительное время, так как перевод с рельефно-точечного шрифта Брайля в плоский вариант текста для зрячего человека является весьма трудоемким процессом.

В данной работе рассматриваются различные системы и стандарты рельефно-точечных шрифтов Брайля, их способы представления и возможности хранения в цифровом виде. На основе анализа таких доступных способов описываются алгоритмы оптического распознавания и разработка мобильного приложения, представляющее собой аналог сканера, который распознает символы рельефно-точечного тактильного шрифта Брайля, написанные учеником по определенным правилам в специализированной тетради, и предоставляет на экране транскрипцию написанного на русском языке. Приложение адаптировано для устройств, работающих на ОС Android.

Проанализировав информацию в Интернете, был изучен рынок мобильных приложений в этой области. На сегодняшний день приложение, полностью удовлетворяющее требованиям преподавателей, не представлено ни на одной доступной платформе.

Таким образом, целями данной работы является исследование систем записи и хранения рельефно-точечных шрифтов, а также разработка мобильного приложения, в качестве основного пользовательского интерфейса.

Для достижения данных целей были поставлены следующие задачи:

— анализ современных стандартов записи текста шрифтом Брайля;
— написание алгоритмов обработки изображений, способных определять нахождение на них текста, набранного шрифтом Брайля и осуществлять его перевод;
— реализация интерфейса мобильного приложения, который будет отображать превью с камеры устройства и давать возможность пользователю сделать фотографию и осуществить обработку получившейся фотографии вышеописанными алгоритмами для получения и отображения переведенного текста.

1 Рельефно-точечный тактильный шрифт Брайля

В данной главе содержится информация об основных системах и стандартах шрифтов Брайля в России. Все спецификации, упоминаемые в данной главе, взяты из ГОСТ Р 56832-2015 «Шрифт Брайля. Требования и размеры».

Шрифт Брайля является рельефно-точечным тактильным шрифтом, что означает, что его можно читать посредством осязания. Знаки (буквы, цифры, знаки препинания и служебные знаки) образуются шестью рельефными точками, значение которых получается с одной стороны из количества и позиции точек в основной форме и, с другой стороны, из позиции знака в системе шрифта Брайля. Брайль в основном предназначен для письма и чтения незрячими и плохо видящими людьми.

1.1 Особенности реализаций систем шрифта Брайля

Каждый символ в шрифте Брайля для русского, английского и некоторых других языков изображается с использованием шести точек. Также перевод в систему Брайля является симметричной операцией, то есть каждому символу исходного алфавита будет соответствовать один символ шрифта Брайля. Данный факт существенно ограничивает количество используемых в тексте символов, и не каждый язык способен уместить весь свой алфавит с цифрами и знаками препинания в 63 символа. Однако существуют и применяются также и другие более сложные системы, имеющие восьмиточечную запись или контекстно-зависимые шеститочечные системы. Все эти усложнения добавляются для расширения количества возможных символов алфавита. Важно упомянуть, что при разработке приложения такие усложненные системы не были приняты во внимание ввиду их редкого применения в России, перевод полученного текста осуществляется относительно стандартной шеститочечной системы Брайля. Точки в такой системе расположены в два столбца и три строки.

1.2 Компьютерная обработка и хранение текста

Для простоты запоминания и компьютерного перевода обычно используют так называемую координатную сетку Брайля, где каждая точка символа имеет свой порядковый номер как закрепленную позицию от 1 до 6.

Такая сетка позволяет записывать Брайль не только имеющимися символами Unicode, но и последовательностями цифр, каждая из которых — это своя позиция следующей точки при прокалывании. Или же двоичными шестибитовыми последовательностями, где 0 будет означать отсутствие выпуклой точки в позиции с номером текущего разряда, начиная со старших, а 1 — ее наличие.

Данный способ записи при хранении символов шрифта Брайля используется в разработке приложения. Так хранятся словари соответствия брайлевских знаков буквам алфавита используемого языка. Подробнее об этом описано в следующей главе.

1.3 Некоторые спецификации стандартной системы Брайля

Из-за того, что осязание в отличие от глаза имеет слабую разрешающую способность, ввиду ограниченности тактильной поверхности на кончиках пальцев, символы шрифта Брайля должны быть очень большими по сравнению с символами плоскопечатного шрифта, чтобы обеспечить точное распознавание и свободное чтение. Аналогично, все точки и отступы должны быть строго одинакового размера в пределах одного документа, отсюда следует и строго закрепленная высота и ширина одного символа, включая пробелы. Следует отметить, что текст шрифтом Брайля крайне редко набирается от руки без дополнительных приспособлений, чтобы уменьшить возможность возникновения ошибок и сохранять рекомендуемые спецификации для улучшения общей читаемости текста.

Данные факты позволяют строить алгоритмы символьного распознавания на основе того, что большая часть отступов и размеров символов и точек одинаковы, что существенно облегчает их реализацию. Установленные спецификации также позволяют использовать шрифты различного размера. Отличаются они как диаметрами точек так и общими размерами отступов, однако все размеры пропорциональны, поэтому при реализации алгоритмов распознавания за единицу измерения внутри документа берется самый часто встречающийся диаметр точки. Отступы между точек внутри одного символа, отступы между двумя разными символами и различные погрешности набора рассчитываются относительно найденного диаметра, что позволяет сохранять эти пропорции.

2 Разработка мобильного приложения

В начале текущей главы рассказывается про разработку нативных приложений на ОС Android, про средства и механизмы сборки Android проектов, а также про некоторые аспекты запуска C++ кода на Android устройствах. Далее объясняется про последовательность предварительной обработки и тонкости алгоритмов адаптивного трешхолдинга и морфологических преобразований изображений. После чего следует объяснение самих алгоритмов распознавания, основанных на поиске контуров. За этим следует описание пользовательского интерфейса и способов его построения в нативных Android приложениях.

2.1 Нативные мобильные приложения на ОС Android

Нативное приложение — это мобильное приложение, спроектированное на определенном языке программирования для конкретной операционной системы устройства, iOS или Android. Нативные приложения iOS написаны на языке Swift или Objective-C, а нативные приложения Android написаны на Java. То есть в отличие от веб-приложений, написанных в основном на Javascript и его фреймворках, нативные приложения — это приложения, написанные на языках, которые допускает платформа, для которой они создаются.

Нативные приложения имеют ряд существенных преимуществ перед альтернативами. Прежде всего, они предлагают пользователям самый быстрый, надежный и отзывчивый интерфейс. Это вряд ли изменится в пользу веб-приложений в ближайшем будущем. Нативный стек разработки обеспечивает легкий доступ к камере, микрофону, компасу, акселерометру и жестам смахивания. Все это также доступно, используя альтернативы, однако весомая часть функционала там доступна не будет. Ввиду этого для реализации поставленной задачи было решено написать именно нативное мобильное приложение, так как для отображения превью с наложенными преобразованиями требуется доступ к покадровой обработке видео с камеры устройства, а также использование на стороне самого устройства C++ библиотек с алгоритмами компьютерного зрения.

Для сборки Android приложений часто используют сборщик Gradle, который позволяет устанавливать гибкие пользовательские конфигурации

сборки. Каждая конфигурация сборки может определять свой собственный набор кода и ресурсов, одновременно используя части, общие для всех версий приложения. Сам же процесс сборки включает в себя множество инструментов и этапов, которые преобразуют Java проект в так называемый Android Application Package, или APK.

Также очень часто в современной мобильной разработке перед инженерами стоит задача организации мультиплатформенности мобильного приложения. И, если речь идет только о нативной разработке, приходится прибегать к написанию двух разных отдельных версий приложения для iOS и для Android. Дизайн пользовательского интерфейса должен в любом случае отличаться в зависимости от используемой платформы, поэтому какого-либо повторения кода здесь не возникает. Однако если мобильное приложение имеет более сложную логику, а также активно использует возможности нативной платформы, то неизбежно будут возникать ситуации, когда разработчикам приходится поддерживать две версии своего приложения одновременно, включая и такие части, написанные на разных языках для разных платформ, отвечающие за одну и ту же логику. Поддержка таких проектов становится затруднительной с увеличением объема и количества таких кусков кода и усложнением бизнес-логики по очевидным причинам, поэтому все чаще в индустрии можно видеть применение C/C++ библиотек, в которых описывается основная часть приложения, которая не отвечает за внешний вид и интерфейс, а только лишь за обработку логики. Такие библиотеки подключаются уже непосредственно как внутри iOS проект проекта, так и внутри Android проекта. В результате получается, что для изменения внутренней логики приходится изменять только код самой библиотеки, а не код двух приложений одновременно. Если речь идет о написании приложения под ОС Android, то тогда разрабочики обычно прибегают к использованию готовых решений интеграции и использования библиотек в нативном коде. Одним из таких решений является NDK.

Разработчик может использовать NDK для компиляции C и C++ кода в нативные библиотеки, которые упаковываются в APK с помощью Gradle на этапе сборки приложения, который был описан выше. Из кода самого приложения методы нативных библиотек доступны через интерфейс JNI. Java Native Interface (JNI) определяет способ взаимодействия байт-кода, который

компилируется из кода Android приложения, написанного на языках программирования Java или Kotlin, с кодом нативных библиотек, написанных на C или C++.

2.2 Предварительная обработка изображения

Для успешного распознавания символов шрифта Брайля, необходимо предварительно обработать полученную фотографию таким образом, чтобы отсеять возможный шум и дополнительные нежелательные помехи, получив бинарное (черно-белое) изображение, на котором бы присутствовали только рельефные точки символов текста. Для этой задачи было решено применить инструментарий библиотеки OpenCV.

Разработчики библиотеки предоставляют официальный пакет OpenCV Android SDK, который содержит в себе саму библиотеку OpenCV, Java интерфейс для нее, а также несколько базовых классов и элементов пользовательского интерфейса, которые взаимодействуют с нативным Android API. Эти базовые классы представляют необходимый для реализации приложения функционал, такой как: покадровое получение изображений с камеры устройства, отображение и модификация превью полученных кадров. Методы OpenCV будут активно использоваться на всех этапах символьного распознавания, включая этап предварительной обработки, о котором было упомянуто ранее.

Начальным этапом предварительной обработки является задание размера полученного изображения. Ввиду того, что современные смартфоны имеют достаточно большое разрешение фотографий, обрабатываемые изображения, могут быть довольно большого размера, что, в свою очередь, будет негативно сказываться на производительности реализуемых алгоритмов. Для решения данной проблемы разрешение получаемых с камеры устройства кадров было ограничено сверху разрешением `864x480`.

Готовые классы для элементов интерфейса в пакете OpenCV Android SDK, которые предоставляют окно для вывода кадров с камеры устройства, предназначены только для работы устройства в ландшафтной ориентации экрана. Вывод камеры всегда ориентирован на поворот 270° по часовой стрелке относительно портретного режима по умолчанию. Ввиду этого в процессе разработки были написаны свои классы для отображения превью, код которых подробно разобран в этой главе.

Основным изменением в наследуемом классе будет наложение преобразований поворота и масштабирования на матрицу, которая соответствует отображаемому кадру относительно текущей ориентации экрана.

Для перевода изображения в бинарный вид, оно должно пройти ряд фильтров, которые снизят количество возможного шума. Так как в дальнейшей обработке будут использоваться алгоритмы бинаризации, требующие того, чтобы исходное изображение находилось в градациях серого, следующим шагом предварительной обработки является перевод изображения в одноканальный формат. Это можно выполнить, ограничить количество получаемых цветовых каналов одним. В результате получается изображение в градациях серого, которое можно использовать в алгоритмах адаптивного трешхолдинга.

Однако перед непосредственным проведением бинаризации, с целью уменьшить количество возможных ошибок, появляющихся в результате зашумления исходного изображения и ошибок набора текста, требуется использовать алгоритмы размытия изображения, которые также представлены в нескольких разновидностях в библиотеке OpenCV. В рамках поставленной задачи не ожидается сильного зашумления получаемых изображений, поэтому для предварительной обработки исходного изображения используется метод быстрого размытия по Гауссу.

После всех описанных преобразований получается размытое изображение в оттенках серого, к которому можно безопасно применять алгоритмы трешхолдинга.

2.3 Бинаризация изображения

Алгоритмы трешхолдинга описывают процесс бинаризации изображения. Бинарное (черно-белое) изображение по своей сути является изображением, у каждого пикселя которого все три компоненты RGB равны 0 или 255. В рамках данного проекта используются так называемые «адаптивные» версии таких алгоритмов. Так как все три компоненты RGB каждого пикселя исходного изображения равны, ввиду того, что оно было представлено в оттенках серого, то происходит сравнение этого значения с некоторым порогом (трешхолдом), и если значение ниже данного порога, то цвет данного пикселя меняется на черный, иначе — на белый. В целях реализации данного приложения был выбран алгоритм трешхолдинга по Гауссу. Также после при-

менения трешхолдинга последовательно применяются два морфологических преобразования: эрозия и наращивание. В этой главе присутствует подробное обоснование выбора адаптивных порогов и наложения морфологических преобразований.

Морфологические преобразования — это специальные операции, которые обычно выполняются на бинарных изображениях и которые основаны на форме самого изображения. В результате применения метода, который в библиотеке OpenCV соответствует эрозии, черные регионы на бинарном изображении будут сужены. После чего применяется наращивание, которое оказывает обратный эффект, расширяя черные регионы, чтобы привести остальные точки в исходный размер.

В конечном итоге получается бинарное изображение, которое уже может быть использовано для поиска контуров рельефных точек (см. рисунок 1).

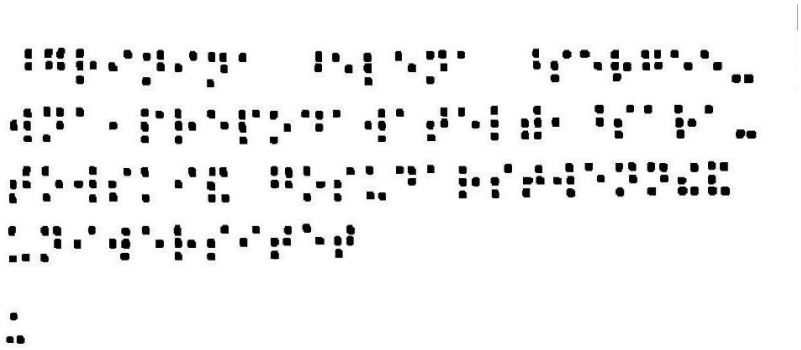

Рисунок 1 – Бинарное изображение, полученное адаптивным трешхолдингом

2.4 Символьное распознавание и перевод текста

OpenCV предоставляет ряд возможностей для нахождения так называемых «контуров» объектов на изображении. Контур — это кривая, соединяющая все непрерывные точки (вдоль границы), имеющие одинаковый цвет или интенсивность. Контуры являются полезным инструментом для анализа формы, а также для обнаружения и распознавания объектов. В официальной документации OpenCV до непосредственного поиска каких-либо контуров рекомендуется бинаризировать исходное изображение, что и происходит на этапах предварительной обработки.

Само символьное распознавание начинается именно с поиска контуров. После чего каждый контур рассматривается отдельно для нахождения минимального прямоугольника, который его ограничивает. Прямоугольники такого вида в OpenCV называются «Bounding Rectangles» и являются одним из способов упрощения геометрии контура для ускорения работы или вычисления приблизительных значений его размерности. После нахождения описываемого прямоугольника контура вычисляется его площадь. Далее находится самое популярное значение из списка площадей всех прямоугольников. На основе полученной площади вычисляется диаметр одной рельефной точки, как квадратный корень этого значения. После чего происходит фильтрация текущих найденных контуров: каждый контур сравнивается с вычисленным диаметром и, если он отличается от него на установленную допустимую погрешность, которая по умолчанию пропускает контуры с диаметром больше 50% и меньше 200% от вычисленного, то он отфильтровывается.

Так как известны контуры каждой точки и описывающие их прямоугольники, то можно найти их геометрические центры и перевести каждый контур в пару координат этих центров для большего удобства работы с контурами и для их наглядного представления. Для корректного перевода текста после нахождения центров каждой рельефной точки нужно узнать, к какому символу они относятся. Для этого проводится процесс сегментации изображения. На изображении выделяются сегменты, каждый из которых относится к своему символу из текста. После успешной сегментации изображения можно переходить на стадию перевода. Сам процесс разделен на две основные части: сопоставление точек внутри сегментов бинарным последовательностям и перевод полученной бинарной матрицы.

Каждый сегмент обрабатывается отдельно, и, в конечном счете, они объединяются в одну двоичную матрицу, где, 1 обозначает наличие рельефной точки, а 0 — ее отсутствие. После получения описанной двоичной матрицы перевод в плоскопечатный текст не составляет больших затруднений.

2.5 Интерфейс приложения

Разметка пользовательского интерфейса в Android осуществляется при помощи так называемых макетов. Макет определяет визуальную структуру пользовательского интерфейса, например, пользовательского интерфейса операции или виджета приложения. В данной работе весь интерфейс описы-

вается при помощи XML-файлов.

Функционал самого пользовательского интерфейса реализуемого приложения не предполагался обширным. Как было описано ранее, главными функциями должны быть возможность видеть превью кадров с камеры устройства, делать фото и возможность просматривать перевод текста с фото.

Для реализации этого функционала был использован один главный экран. Основной макет на данном экране — это макет, предоставляемый вместе с пакетом OpenCV Android SDK, который осуществляет отображение кадров с камеры устройства. Помимо превью камеры на экране также присутствуют две кнопки, одна из которых осуществляет снимок, а вторая очищает текущий сохраненный снимок. При нажатии на первую будет скопирован текущий кадр с камеры устройства, пропущен через конвейер препоцессинга, работа которого была описана ранее.

Результаты сегментации и поиска контуров сохраняются в отдельной матрице, которая показывается в качестве превью. При нажатии на другую кнопку матрица очищается, и на экран продолжает выводиться оригинальное превью камеры. После попытки распознавания также будет открыто модальное окно с распознанным текстом или с предупреждением о том, что текст не удалось распознать.

Изначально предполагалось, что у пользователя будет возможность видеть, какие именно рельефные точки участвуют в распознавании, поэтому в меню верхней панели также доступен переключатель наложения фильтра, влияющий на то, какое изображение выводится в качестве превью: исходное или прошедшее предварительную обработку.

ЗАКЛЮЧЕНИЕ

В результате написания данной работы были проанализированы стандартные системы записи и компьютерного хранения текстов, набранных рельефно-точечным тактильным шрифтом Брайля, а также разработано нативное приложение для ОС Android для символьного распознавания таких текстов, где способом хранения словаря для перевода в плоскопечатный вид была выбрана основная координатная сетка Брайля. Приложение способно обрабатывать кадры с камеры устройства, обнаруживать символы и переводить найденный текст в плоскопечатный формат. Приложение адекватно распознает весь текст, используемый при разработке в качестве тестовых данных, при условии создания приемлемой для распознавания фотографии. Для определения, по каким именно критериям будет распознан сам текст, пользователю предоставлен интерфейс, с помощью которого можно наблюдать результат предварительной обаботки изображения, на котором будут видны обрабатываемые рельефные точки шрифта.

К сожалению, реализованные алгоритмы распознавания являются несовершенными и имеют ряд недостатков. Как было отмечено ранее, с помощью данного приложения невозможно сделать перевод большого текста за один раз. А также на данный момент отсутствует какая-либо нормализация угла наклона получаемого изображения. То есть, нигде не учитывается был ли снимок произведен под некоторым углом наклона или с некоторой перспективой, поэтому такие изображения будут переведены некорректно.

Однако за счет того, что все действия осуществляются на стороне приложения при помощи нативных библиотек, написанных на C++, то нужные вычисления проводятся почти моментально, что позволяет покадрово выводить пользователю превью предварительной обработки без значительной потери в количестве кадров. Также после каждого распознавания пользователь может наблюдать сетку символов, по которой была осуществлена текущая попытка, с помощью которой можно будет увидеть отклонение символов от горизонтальной оси, что позволит поправить наклон камеры при следующей попытке.