**МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ**

**РОссИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)»

**Институт № 3 «Системы управления, информатика и электроэнергетика»**

**Кафедра 304 «Вычислительные машины, системы и сети» Группа М3О-409Б-19**

**Направление подготовки 09.03.01 «Информатика и вычислительная техника»**

**Профиль «Вычислительные машины, комплексы, системы и сети»   
Квалификация бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: Разработка программы распознавания нотных символов на изображениях, содержащих нотную нотацию с использованием технологий компьютерного зрения.

Автор ВКРБ (\_\_\_\_\_\_\_\_\_\_\_\_\_)

(фамилия, имя, отчество полностью)

Руководитель (\_\_\_\_\_\_\_\_\_\_\_\_\_)

(фамилия, имя, отчество полностью)

Консультант (\_\_\_\_\_\_\_\_\_\_\_\_\_)

(фамилия, имя, отчество полностью)

Консультант (\_\_\_\_\_\_\_\_\_\_\_\_\_)

(фамилия, имя, отчество полностью)

Рецензент (\_\_\_\_\_\_\_\_\_\_\_\_\_)

(фамилия, имя, отчество полностью)

**К защите допустить**

Заведующий кафедрой 304 (\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(№ каф) (фамилия, имя, отчество полностью)

“\_\_\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023г.

Москва 2023г.

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc137377090)

[1. Анализ исходных данных и постановка задачи. 6](#_Toc137377091)

[1.1. Роль музыки в повседневной жизни. 6](#_Toc137377092)

[1.2. Основные элементы музыкальной теории. 7](#_Toc137377093)

[1.3. Компьютерное зрение 11](#_Toc137377094)

[1.4. История компьютерного зрения. 12](#_Toc137377095)

[1.5. Математические основы компьютерного зрения. 14](#_Toc137377096)

[1.6. Методы обработки изображений и видео. 15](#_Toc137377097)

[1.7. Анализ существующих программ для распознавания нот. 21](#_Toc137377098)

[1.8. Постановка задачи дипломной работы. 29](#_Toc137377099)

[1.9. Выбор программных средств. 30](#_Toc137377100)

[2. Программная реализация 31](#_Toc137377101)

[2.1. Подготовка изображения. 32](#_Toc137377102)

[2.2. Поиск и обнаружение нотных станов на изображении. 34](#_Toc137377103)

[2.3. Поиск и обнаружение на изображении объектов, похожих на ноту. 38](#_Toc137377104)

[2.4. Поиск среди найденных элементов нот, находящихся на нотных станах и определения их высот, относительно их положения. 41](#_Toc137377105)

[2.5. Создание из найденных нот MIDI-файла для прослушивания и анализа. 43](#_Toc137377106)

[3. Тестирование разработанного алгоритма. 45](#_Toc137377107)

[3.1. Подготовка тестовых изображений. 45](#_Toc137377108)

[3.1.1. Тестирование работы на примере скана листа. 47](#_Toc137377109)

[3.1.2. Проверка корректности работы программы. 52](#_Toc137377110)

[3.1.3. Тестирование на примере фотографии, содержащей лист с музыкальным произведением. 55](#_Toc137377111)

[3.1.4. Проверка корректности работы программы. 61](#_Toc137377112)

[3.2. Подведение итогов тестирования. 64](#_Toc137377113)

[ЗАКЛЮЧЕНИЕ 66](#_Toc137377114)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 68](#_Toc137377115)

[ПРИЛОЖЕНИЕ А 69](#_Toc137377116)

[ПРИЛОЖЕНИЕ Б 71](#_Toc137377117)

[ПРИЛОЖЕНИЕ В 77](#_Toc137377118)

[ПРИЛОЖЕНИЕ Г 78](#_Toc137377119)

[ПРИЛОЖЕНИЕ Д 84](#_Toc137377120)

[ПРИЛОЖЕНИЕ Е 85](#_Toc137377121)

[ПРИЛОЖЕНИЕ Ж 90](#_Toc137377122)

[ПРИЛОЖЕНИЕ З 95](#_Toc137377123)

# ВВЕДЕНИЕ

Современный мир невозможно представить без технологий, позволяющих автоматизировать процессы, которые ранее требовали участия человека. Одной из таких технологий автоматизации является компьютерное зрение.

Компьютерное зрение – область искусственного интеллекта, которая занимается разработкой методов и алгоритмов для автоматического анализа, обработки и интерпретации изображений и видео. Она находит применение во многих сферах современного мира. Например, в медицине, технологии компьютерного зрения используются для диагностики и лечения различных заболеваний. Также компьютерное зрения играет важнейшую роль в развитии робототехники, создании виртуальной и дополненной реальности.

Музыкальные ноты – главные элементы музыкальной нотации, представляющие собой символы, используемые для создания и записи музыкальных произведений. Знание и умение читать ноты расширяет возможности для музыкантов. Многие начинают свой путь с изучения нот, чтобы впоследствии использовать эти знания для эффективного обучения игры на инструменте. Умение читать ноты с листа – трудоемкий процесс, который занимает много времени, особенно для начинающих.

Существует множество методов и технологий для распознавания объектов на изображении, в том числе и нотной нотации. Распознавание музыкальной нотации с использованием технологий компьютерного зрения имеет практическое применение для музыкантов, которые смогут использовать автоматизированную систему для чтения нот и для обучения игры на инструменте.

В данной выпускной квалификационной работе будет рассмотрен подход на основе методов компьютерного зрения. В качестве основной технологии будет использоваться библиотека OpenCV языка Python, предоставляющая весь набор инструментов для работы с изображением и видео.

Основная задача заключается в определении нот на изображении, а также определение их высот и прослушивания их звучания. Это позволит сократить время, затрачиваемое на самостоятельное распознавание нот, и повысит точность результатов.

В рамках работы будут изучены существующие методы распознавания объектов на фото, будет разработан и реализован алгоритм распознавания на основе свойств нот и их расположения на изображении. Также будет проведена оценка точности разработанного алгоритма на различных изображениях, а также на фотографиях, содержащих музыкальные произведения, описанных с помощью нотной нотации.

1. Анализ исходных данных и постановка задачи.

## **1.1. Роль музыки в повседневной жизни.**

В настоящее время музыка является важнейшей формой искусства, оказывающей значительное влияние на культуру и общество. Музыка воплощает эмоции, выражает идеи, соединяет людей и служит средством самовыражения. Она играет важную роль в жизни людей и оказывает значительное влияние на наше самочувствие, настроение и эмоции.

Важность музыки в повседневной жизни можно подчеркнуть несколькими аспектами:

* Эмоциональное выражение: музыка является мощным средством эмоционального выражения. Она может передать и усилить различные чувства, включая радость, грусть, восторг, меланхолию и многие другие.
* Индивидуальное самовыражение: благодаря музыке каждый человек может выразить свою уникальность и индивидуальность. Через музыку мы можем передавать свои мысли, чувства и идеи, отражая наше внутреннее мироощущение и самовыражение.
* Культурное наследие: музыка является важной частью культурного наследия различных народов и обществ. Она передает и сохраняет традиции, историю и ценности определенной культуры. Изучение и понимание музыки позволяет нам глубже познать и уважать культурное многообразие мира.
* Социальная связь: музыка объединяет людей и способствует социальной связи. Музыкальные события, концерты и фестивали создают возможность для людей с разных культур и сообществ собраться вместе и насладиться общим музыкальным опытом. Музыка является языком, позволяющим людям коммуницировать и взаимодействовать друг с другом.

Исходя из этого, у огромного количества людей в мире возникает желание изучать музыкальную теорию, уметь играть на музыкальных инструментах и петь.

## **1.2. Основные элементы музыкальной теории.**

Музыкальная теория – систематическое изучение музыки, ее элементов, структуры, характеристик и взаимоотношений между ними. Она охватывает широкий спектр тем, связанных с музыкой, включая музыкальные ноты, ритм гармонию, мелодию, темп и т.д.

Основным аспектом в данной дипломной работе являются ноты и нотация.

Ноты – основные элементы, используемые в музыке для записи и передачи музыкальных идей. Они играют важную роль в передачи музыкальной информации от композитора к исполнителю и слушателю.

Основные ноты, которые используются в музыке возникли в Средневековье и сегодня, в музыкальной практике, представлены двумя системами – слоговой и буквенной.

За основу буквенной системы в раннем Средневековье был взят латинский алфавит, первые буквы которого дали название ступеням средневекового семиступенного звукоряда. Таким образом, тоны данного звукоряда получили следующее буквенное обозначение: A (a), B (b), C (c), D (d), E (e), F (f), G (g), соответствующие по современной слоговой системе обозначений звукам «ля», «си», «до», «ре», «ми», «фа», «соль». На рисунке 1.1 представлены основные ноты современной хроматической музыкальной системы.

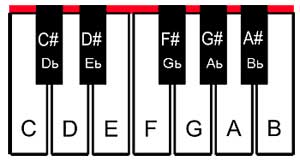


Рисунок 1.1 – Основные ноты современной хроматической музыкальной системы

Ноты представляют собой символы, которые обозначают высоту и длительность звуков. Они располагаются на нотном стане, который состоит из горизонтальных линий и промежутков между ними. Каждая линия и промежуток на нотном стане соответствует определенной ноте.

Нотация – система записи музыки с использованием нот и других символов. Существует несколько систем нотации, которые используются в музыке, но наиболее распространенной является стандартная нотация. В этой системе ноты представлены символами, размещенными на нотном стане.

Стандартная нотация включает в себя следующие элементы:

1. Нотный стан – горизонтальные линии и промежутки, на которых размещаются ноты. В стандартной нотации используется пять горизонтальных линий, но при необходимости могут использоваться дополнительные линии.
2. Ключ – символ, который указывает высоту определенной ноты на нотном стане. Наиболее часто используемые ключи – скрипичный (также известный как третий ключ) и басовый ключ (также известный как четвертый ключ).
3. Знаки длительности — это символы, которые обозначают длительность нот. Они включают целую ноту, половинку, четверть, восьмую, шестнадцатую и так далее.
4. Дополнительные символы – различные символы для обозначения динамики (громкости), артикуляции, темпа, фразировки и других музыкальных выражений.

Октава – концепция в музыке, которая относится к диапазону звуков по их высоте. Каждая октава представляет собой интервал между двумя звуками с одинаковым названием, но различающимся по высоте. Например, нота «до» первой октавы и нота «до» второй октавы.

В музыке используется система октавного деления, основанная на 12 полутонах (черных и белых клавиш фортепиано) в рамках одной октавы. Каждая октава состоит из 12 полутонов, которые повторяются в последовательных октавах.

Тон – интервал между двумя звуками, который воспринимается как музыкально распределенный и имеет определенное музыкальное расстояние. Один тон эквивалентен двум полутонам.

Полутон – наименьший интервал, используемый в западной музыкальной системе. Он представляет собой минимальное музыкальное расстояние между двумя соседними звуками на клавишной клавиатуре (например, на фортепиано) или между двумя соседними ладами на грифе струнного инструмента.

Пример тона и полутона изображен на рисунке 1.2.

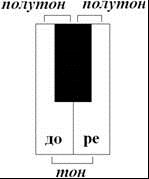


Рисунок 1.2 – Пример тона и полутона на фортепиано

Октавы обычно обозначаются числовыми значениями и имеют буквенные названия. В стандартной системе нумерации октавы начинаются с номера 0 и увеличиваются вверх по мере увеличения высоты звуков. Таким образом, каждая последующая октава имеет номер, увеличенный на 1.

Первая нота октавы – C (до), затем следуют ноты D, E, F, G, A, B. После ноты B (си) следует нота C (до) следующей октавы.

Всего на фортепиано есть 8 октав: 0 - субконтроктава, 1 – контроктава, 2 – большая октава, 3 – малая октава, 4 – первая октава, 5 – вторая октава, 6 – третья октава, 7 – четвертая октава и нота до 8й – пятой октавы.

Каждая октава включает в себя различные ноты, расположенные в пределах данной октавы (пример на рисунке 1.3). Использование числовых обозначений октав позволяет однозначно определить высоту звука и установить связь между различными инструментами и исполнителями.



Рисунок 1.3 – Пример записи нот на нотных станах и октав

Изучение музыкальной теории – трудоемкий процесс, поэтому существует множество программ, помогающих начинающим музыкантам выучить ноты, нотную нотацию, написать свое собственное музыкальное произведение и провести анализ любого произведения, написанного с помощью нотной нотации.

С помощью программы для определения и классификации нот на нотном листе пользователь сможет ускорить свое обучение нотной грамоте. Сможет проверить себя и проработать ошибки, допущенные при собственном анализе музыкального произведения.

## **1.3. Компьютерное зрение**

Компьютерное зрение (англ. computer vision) является разделом искусственного интеллекта и науки о компьютерном зрении, который изучает методы и техники, позволяющие компьютерам “видеть” и интерпретировать изображения и видеоаналитику таким образом, чтобы понимать и анализировать содержание исходных данных.

Компьютерное зрение стремится имитировать способность человеческого зрения, которое воспринимает, анализирует и понимает визуальные данные. Основные задачи компьютерного зрения включают обнаружение объектов, распознавание образов, сегментацию изображений, трекинг движущихся объектов, классификацию и интерпретацию содержимого изображений и видео.

Для выполнения этих задач компьютерное зрения использует различные методы и алгоритмы обработки изображения и анализа данных.

Приведем примеры некоторых методов:

* Предобработка изображений. Включает операции такие как изменение размера изображений, фильтрация, сглаживание, улучшение контрастности и яркости, а также удаление шума и искажений, чтобы подготовить изображение к дальнейшему анализу.
* Извлечение признаков. Операции, направленные на извлечение информативных признаков из изображений, таких как границы, углы, текстуры, цвета и формы. Это может включать алгоритмы детектирования краев, дескрипторы особых точек или градиентные методы.
* Классификация и распознавание. Процесс определения и идентификации объектов или образов на изображении. Это может включать обучение моделей машинного обучения, таких как нейронные сети или классификаторы, для автоматического распознавания и классификации объектов.
* Сегментация изображений. Разделение изображения на отдельные сегменты или регионы, обозначающие объекты или области интереса. Это может быть полезно для обнаружения и извлечения объектов из фона или для анализа структуры и контуров объектов.
* Трекинг движущихся объектов: Отслеживание и анализ перемещения объектов на последовательных кадрах видео. Это позволяет отслеживать движущиеся объекты и анализировать их поведение или изменения во времени.
* 3D компьютерное зрение. Включает в себя методы и техники для восстановления трехмерной информации из двухмерных изображений или видео. Это может быть полезно для анализа глубины, размеров и пространственной структуры объектов.

Компьютерное зрение находит широкое применение во многих областях, таких как робототехника, медицина, автоматическое управление, безопасность, анализ изображений и видео, дополненная виртуальная реальность и многие другие. Оно играет ключевую роль в развитии автономных систем, распознавании лиц, машинном зрении, медицинской диагностике и многих других приложениях, где обработка и анализ визуальных данных являются критически важными задачами.

## **1.4. История компьютерного зрения.**

История компьютерного зрения уходит корнями во вторую половину 20го века, когда начали развиваться первые идеи и концепции, связанные с использованием компьютеров для обработки и анализа изображений.

Рассмотрим некоторые ключевые моменты в истории компьютерного зрения:

1. Ранние работы (1950-е – 1960-е годы):

* В 1956 году Аллен Ньюэлл и Герберт Саймон предложили идею использования компьютеров для символьной обработки информации и применения визуальных задач.
* В 1966 году Сеймур Паперт и Марвин Минский представили искусственные нейронные сети (ИНС) в своей работе “Персептроны”, которые стали основой для развития компьютерного зрения.

1. Развитие методов обработки изображений (1970-е – 1980-е годы):

* В 1973м году Хэрри Канн публикует свою работу “Распознавание образов: методология и реализация для синтетического и аналогового компьютерного зрения”, где предлагают системный подход к компьютерному зрению.
* В 1980-х годах активно развиваются методы обработки и анализа изображений, такие как выделение признаков, фильтрация, сегментация и другие.

1. Прорыв в распознавании образов (1990-е – 2000-е годы):

* В 1998 году Ян Лекун представляет метод сверточных нейронных сетей (CNN), который стал ключевым прорывом в области распознавания образов.
* В 2012 году Алекс Криспан представляет нейронную сеть под названием “AlexNet”, которая существенно улучшает результаты в задаче классификации изображений на наборе данных ImageNet.

1. Применение глубокого обучения (2010-е годы и далее):

* В 2012 году глубокое обучение становится основным направлением в компьютерном зрении благодаря успехам в использовании глубоких нейронных сетей.
* С появлением больших объемов данных, мощных вычислительных ресурсов и развитием графических процессоров (GPU), глубокое обучение становится еще более эффективным и точным.

1. Расширение применений (настоящее время):

* Компьютерное зрение находит широкое применение в различных областях, таких как медицина, автомобильная промышленность, робототехника, безопасность, анализ изображений и видео, дополненная реальность и многое другое.
* С развитием технологий машинного обучения и глубокого обучения, компьютерное зрение продолжает улучшаться и проникать в новые сферы деятельности

Сегодня компьютерное зрение является активной и быстроразвивающейся областью исследований и разработок. С появлением новых методов, алгоритмов и технологий, компьютерное зрение становится все более точным, быстрым и многосторонним инструментом автоматизации и анализа визуальной информации.

## **1.5. Математические основы компьютерного зрения.**

Математические основы компьютерного зрения включают в себя множество методов и алгоритмов, которые позволяют обрабатывать изображения, распознавать образы и анализировать данные.

Основные методы и концепции, используемые в компьютерном зрении:

* Линейная алгебра позволяет представлять изображения в виде матриц и проводить с ними различные операции, такие как поворот, сжатие, изменение размера и т.д.
* Статистика используется для анализа данных, полученных из изображений. Статистические методы используются для распознавания объектов на изображениях, определение их размера и формы, а также для анализа текстур и цветов.
* Теория вероятностей позволяет определять точность распознавания образов.
* Методы оптимизации используются для настройки параметров алгоритмов и выбора наилучшего решения. Наиболее популярные методы – градиентный спуск и методы оптимизации с ограничениями.
* Машинное обучение используется для обучения компьютерных систем распознаванию образов и классификации. Например, нейронные сети и алгоритмы классификации.

## **1.6. Методы обработки изображений и видео.**

Методы обработки изображений и видео широко используется в компьютерном зрении. Они позволяют автоматически извлекать нужную информацию из изображений и видео, выделять объекты, анализировать свойства и траектории движения объектов и многое другое.

Для работы с изображением, его нужно сначала представить в виде матрицы пикселей, где каждый пиксель представляет собой маленький квадратик, содержащий информацию об интенсивности цвета или яркости в этой точке изображения.

Если изображение имеет размерность , то его можно представить в виде матрицы размером , где каждый элемент матрицы , представляет собой яркость пикселя в *i*-й строке и *j*-м столбце.

Для цветных изображений обычно используется трехмерный массив пикселей, где каждый пиксель имеет три значения – интенсивность красного (Red), зеленого (Green) и синего (Blue) цветов (RGB). Таким образом, каждый элемент матрицы представляет собой вектор (R, G, B) для данного пикселя.

Некоторые из основных методов обработки и изображений и видео:

* Фильтрация.

Фильтры используются для обработки изображений и видео с целью улучшения качества изображений, устранения шума и других артефактов.

Например, черно-белый (ЧБ) фильтр – один из самый простых фильтров обработки изображений, который преобразует изображение в черно-белое.

Он работает по принципу преобразования каждого пикселя изображения из цветного представления RGB (Red, Green, Blue) в оттенки серого используя следующую формулу:

Здесь R, G, B -значения красной, зеленой и синей компонент цвета в диапазоне от 0 до 255. Значение Gray будет находиться в диапазоне от 0 до 255 и будет использоваться в качестве нового значения пикселя на месте цветного значения.

На рисунке 1.4 продемонстрированно применение ЧБ фильтра.

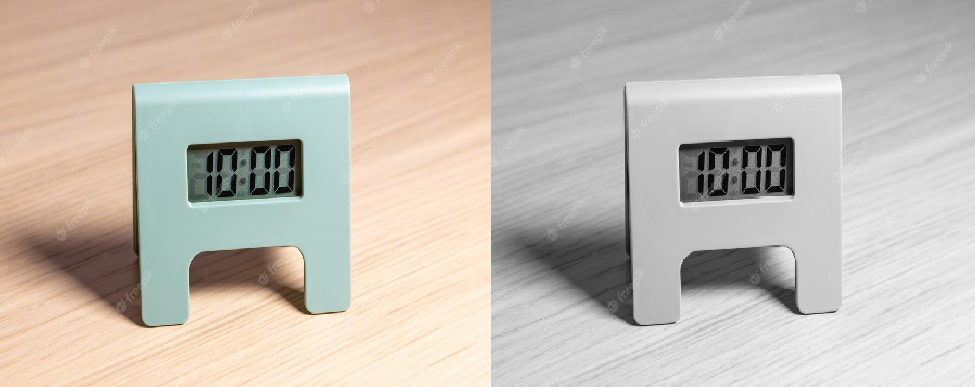


Рисунок 1.4 – Применение ЧБ фильтра

* Сегментация.

Сегментация – это процесс разделения изображения на части или объекты. Это может быть полезным, например, для определения области интереса на изображении. Для сегментации используются различные методы, такие как, пороговая бинаризация, кластерный анализ и методы на основе границ. Приведем пример бинаризации.

Бинаризация - метод обработки изображений, который позволяет разделить изображение на 2 части: переднего и фонового планов. В результате бинаризации каждый пиксель изображения получает значение либо 0 (черный), либо 1 (белый) в зависимости от того, находится ли он в переднем или фоновом плане.

Существует несколько методов бинаризации, но самый распространенный – метод пороговой бинаризации, при котором каждый пиксель изображения превращается в черный или белый пиксель, в зависимости от того, превышает ли его интенсивность некоторый порог.

Процесс пороговой бинаризации можно описать следующим образом:

1. Преобразование изображения в оттенки серого.
2. Выбор порогового значения t. Это значение является границей между черным и белым цветами и определяет, какие пикселя будут отнесены к черному, а какие к белому цвету.
3. Проход по всем пикселям изображения
4. Если значение пикселя больше t, установить значение пикселя в 1 (белый), иначе установить его в 0 (черный).

Применение бинаризации часто используется для выделения объектов на изображении и упрощения его дальнейшей обработки. На рисунке 1.5 продемонстрирована пороговая бинаризация рисунка 1.4.

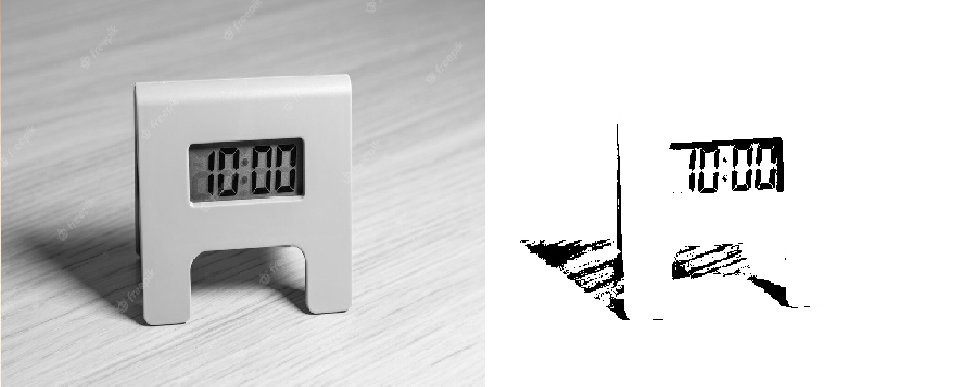


Рисунок 1.5 – Применение пороговой бинаризации

* Детектирование границ.

Детектирование границ – процесс поиска различий в интенсивности пикселей изображения, которые могут указывать на границы объектов на изображении. Это один из ключевых шагов в обработке изображений.

Для детектирования границ на изображении применяются различные операторы, которые работают с различными аспектами изображения, такими как градиенты яркости, вторые производные, различные фильтры и т.д. Наиболее популярными методами детектирования являются операторы Собеля, Прюитта и Кэнни.

Оператор Кэнни – один из наиболее распространенных алгоритмов детектирования границ на изображении. Он был разработан Джоном Кэнни в 1986 году и широко используется в компьютерном зрении и обработке изображений. Оператор Кэнни является многопроходным алгоритмом, который состоит из следующих шагов:

1. Сглаживание изображения: с помощью фильтра Гаусса производится сглаживание изображения, чтобы уменьшить шумы и сохранить только наиболее важные детали.
2. Расчет градиентов: вычисляются градиенты в каждой точке изображения. Градиенты показывают направление и силу изменения яркости в каждой точке
3. Подавление «ложных» максимумов: в каждой точке изображения выбирается максимальное значение градиента по направлению нормали к границе.
4. Пороговая бинаризация: значения градиента сравниваются с двумя пороговыми значениями, и только пиксели с градиентом выше максимального порога считаются граничными, пиксели с градиентом ниже минимального порога считаются неграничными, а пиксели со значениями градиента между двумя порогами анализируются на наличие связности с граничными пикселями.
5. Определение связности: анализируются пиксели, соединяющие граничные пиксели и определяется, являются ли они частью одной границы или нет.
6. Уточнение границ: границы могут быть уточнены с помощью метода трассировки контура, который использует информацию о направлении границы и расстоянии между пикселями для точного определения границы.

На рисунке 1.6 продемонстрировано детектирование границ с помощью оператора Кэнни рисунка 1.4.

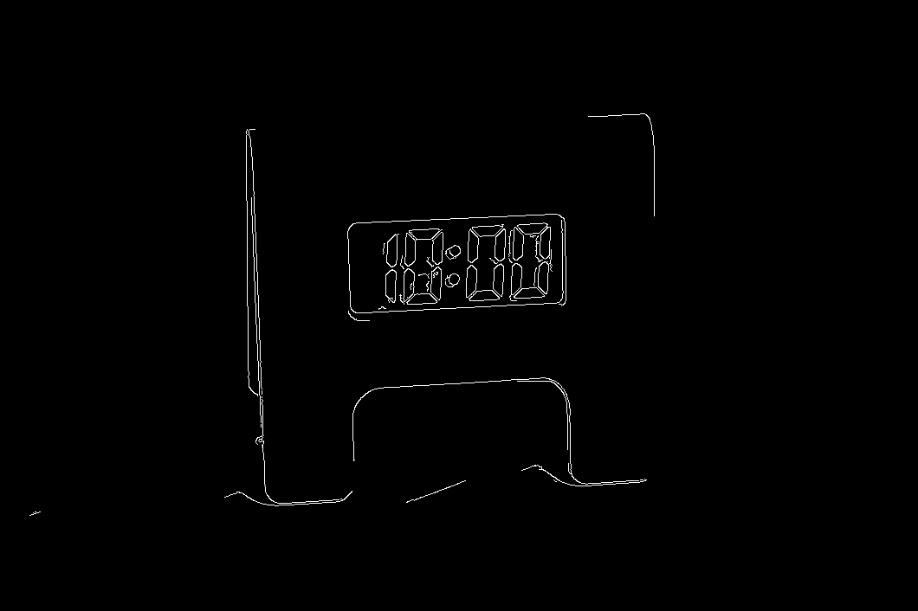


Рисунок 1.6 – Применение оператора Кэнни

* Выравнивание изображений.

Выравнивание – процесс приведения изображения к стандартной форме или ориентации.

Выравнивание может быть достигнуто путем изменения масштаба, поворота и переноса изображения. Некоторые алгоритмы выравнивания также могут применять деформации для изменения формы изображения.

Одним из распространенных методов выравнивания является выравнивание на основе контуров. Он используется для выравнивания изображений с нестандартной формой или обрезанных изображений. Алгоритм сначала находит контуры объекта на изображении, затем вычисляет угол поворота и масштабирует изображение для выравнивания контуров.

Например, к рисунку 1.4 можно применить выравнивание с помощью контуров, путем нахождения наибольшего контура, выделяющего нужный объект на изображении, затем вырезать соответствующую область изображения.

На рисунке 1.7 продемонстрировано выделение нужного контура на изображении.

.

Рисунок 1.7 – Выравнивание и вырезка нужного контура

Помимо представленных методов обработки изображений существует множество других методов изображений, таких как: морфологическая обработка, цветовая обработка, обнаружение движения, обработка видео на основе глубокого обучения и т.д.

## **1.7. Анализ существующих программ для распознавания нот.**

Существует множество программ, которые занимаются распознаванием нот из изображения нотного листа. Некоторые из них бесплатны, а некоторые являются коммерческими продуктами.

Musitek Smart Score – программное обеспечение для распознавания нот из отсканированных документов или PDF ­– файлов. Для автоматического распознавания нот, аккордов, текста и других элементов нотных листов и сохранения их в электронном формате используется технология Optical Music Recognition.

Технология Optical Music Recognition (OMR) или оптическое распознавание музыки – область компьютерного зрения и обработки изображений, которая занимается распознаванием и интерпретацией нотного материала с помощью анализа нотных изображений или сканированных нотных страниц.

На официальном сайте Musitek доступны различные демоверсии программного обеспечения, которые можно бесплатно скачать и использовать на своем персональном компьютере.

Для примера, будет использована программа SmartScore X2 Songbook Edition. Данная программа, как и все остальные версии, обладает функцией распознавания музыкальных нот и аккордов, позволяет создавать полноценные нотные партитуры. Также имеются инструменты для редактирования ноты, текстов, шрифтов и т.д. Распознанную и отредактированную партитуру можно воспроизвести, чтобы прослушать и проверить правильность распознавания и оформления.

Протестируем работу данной программы. Для начала, нужно выбрать нотное произведение, ноты которого нужно распознать. На рисунке 1.8 представлены ноты одного из самых известных в мире романсов на русском языке “Очи черные” в виде простой мелодии.

На начальном экране, изображенном на рисунке 1.9, нажмем на кнопку “Recognition”, которая представлена на рисунке 1.10. Далее, в появившемся окне, изображенном на рисунке 1.11 нажмем на кнопку “Add Files to List”, выберем нужное нам изображение, загрузим, и нажмем на кнопку “Begin Recognition”. Далее, после процесса распознавания, на рисунке 1.12 мы увидим лист, содержащий ноты, которые можно отредактировать, а также, с помощью встроенного плеера, прослушать звучание мелодии от начала и до конца. Нажав на вкладку “Playback” и выбрав кнопку “Piano Roll View”, изображенную на рисунке 1.13, мы сможем увидеть визуальное представление музыкальных нот и их длительностей.

На рисунке 1.14 мы видим отображение нот в виде горизонтальных полосок на оси времени. Каждая полоска представляет отдельную ноту и позволяет пользователю определить ее высоту и длительность. Также, на рисунке 1.15 мы можем выбрать другое отображение, например, в виде списка событий (Event list), который изображен на рисунке 1.16, который представляет собой структурированный список музыкальных событий, таких как ноты, аккорды, длительности и другие параметры, связанные с музыкальной композицией. Обычно, список событий представляет собой таблицу или список, где каждая строка соответствует отдельному музыкальному событию, а столбцы содержат информацию о времени, длительности, высоте ноты, динамике и других параметрах.



Рисунок 1.8 – Пример музыкального произведения

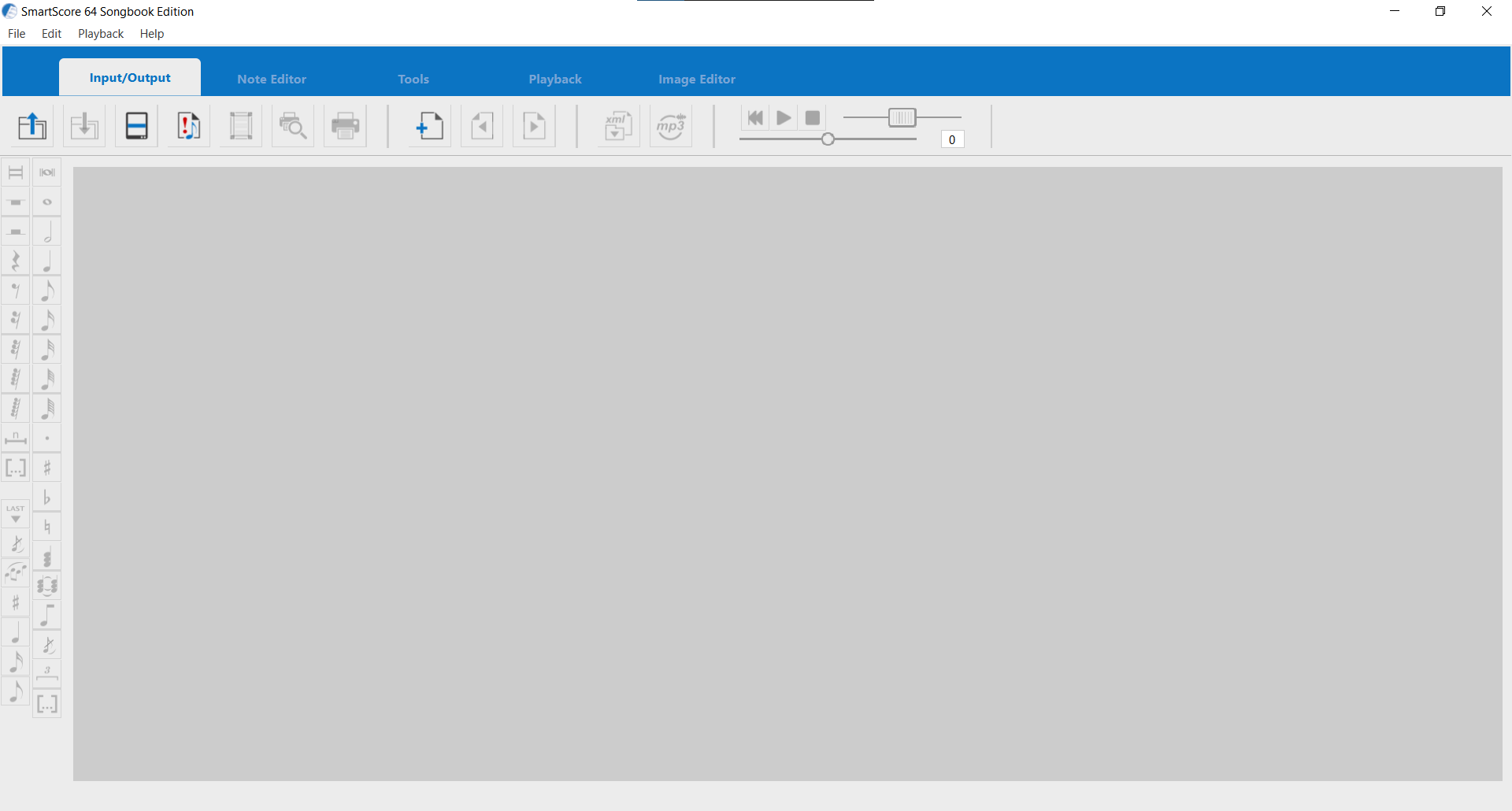


Рисунок 1.9 – Начальная страница приложения

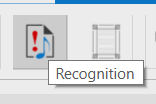


Рисунок 1.10 – Кнопка для распознавания

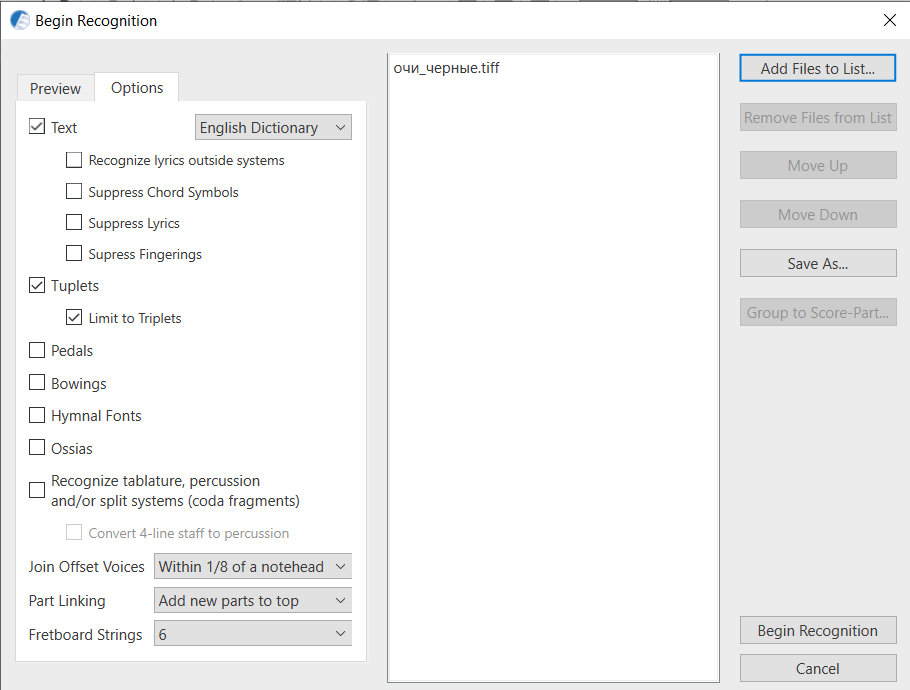


Рисунок 1.11 – Меню выбора нотного файла

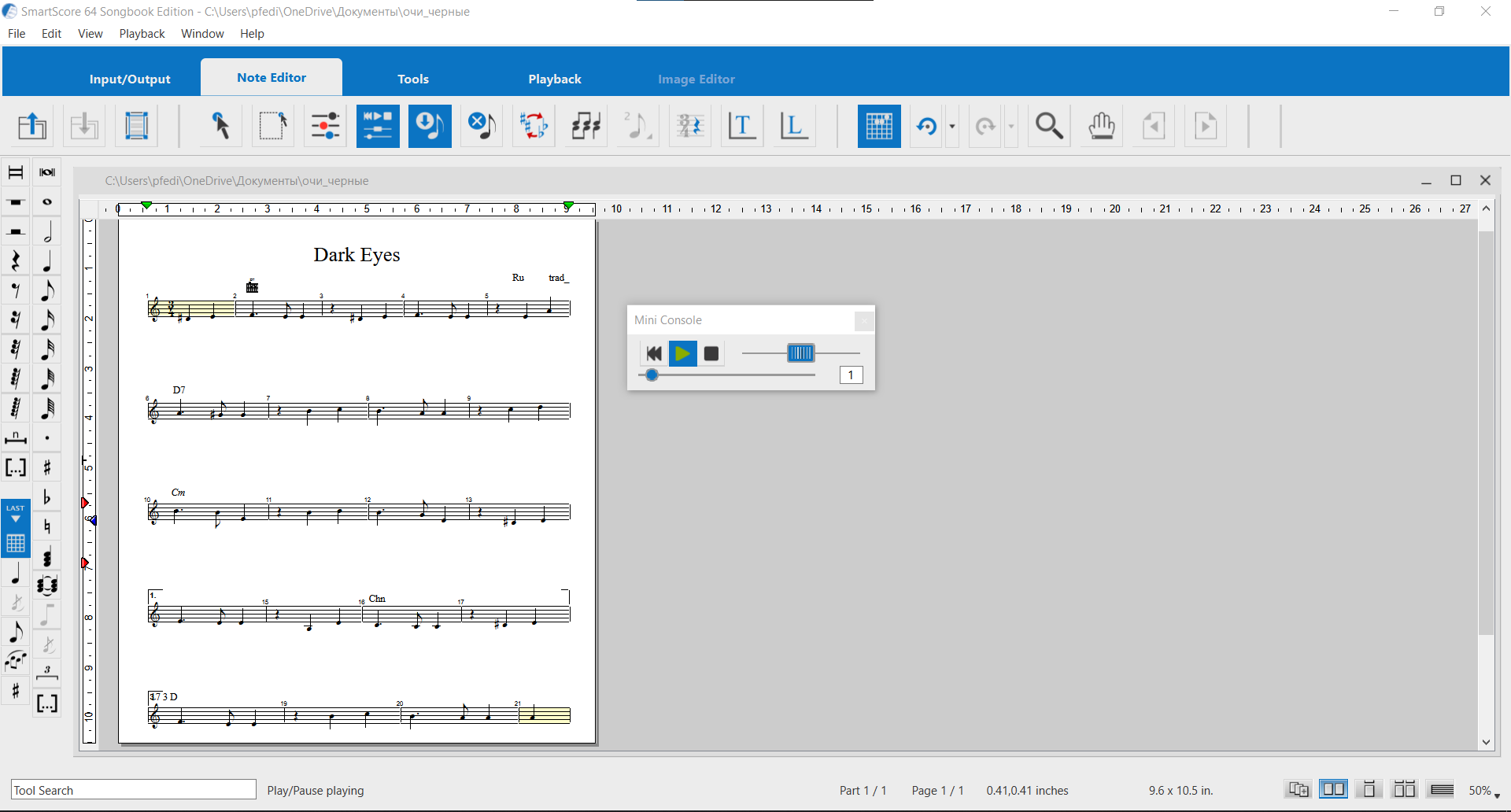


Рисунок 1.12 - Распознанный нотный лист

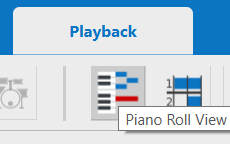


Рисунок 1.13 – Кнопка для выбора просмотра в виде ленты для пианино

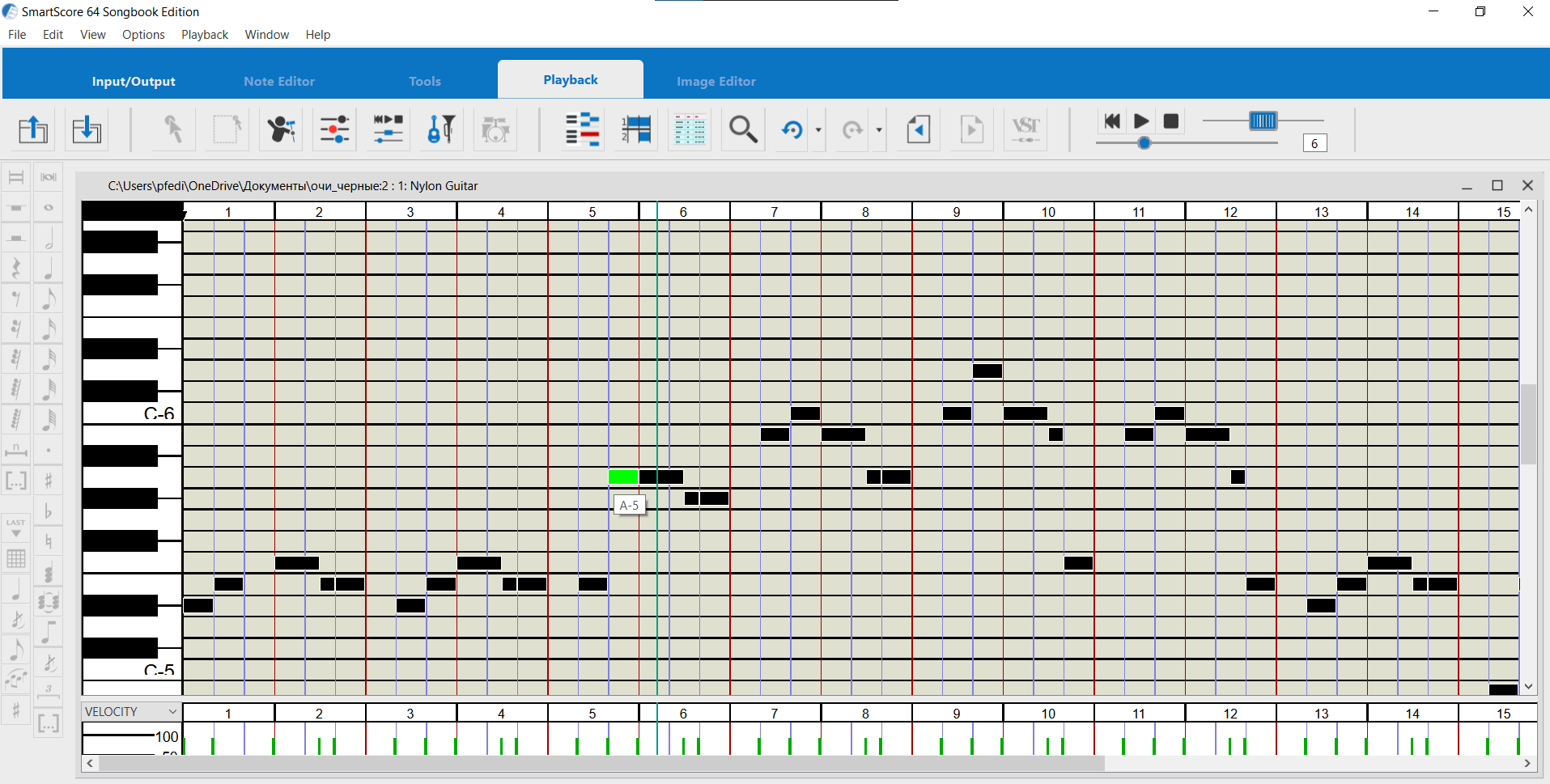


Рисунок 1.14 – Положение нот и их длительности

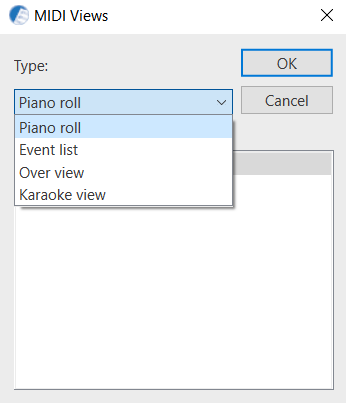


Рисунок 1.15 - выбор отображения текущего произведения

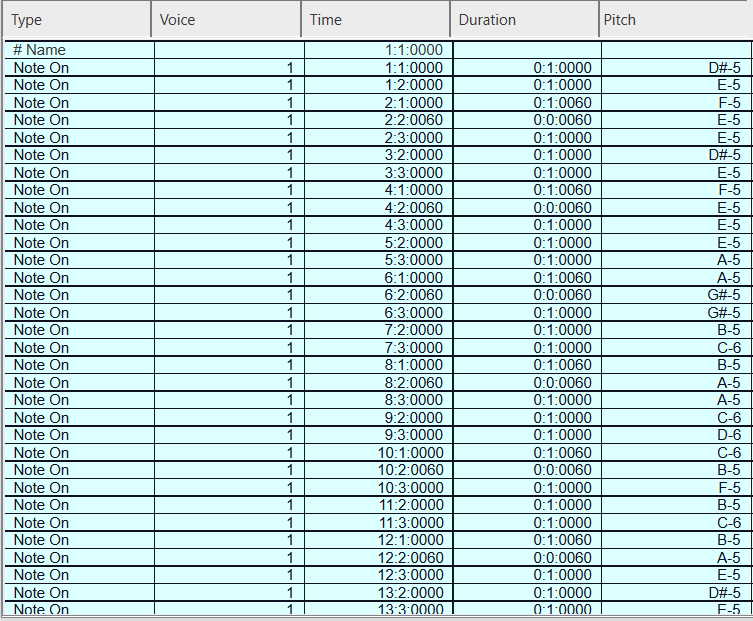


Рисунок 1.16 – Список нот в виде буквенного обозначения, время начала их воспроизведения и продолжительность каждой.

Все версии программ SmartScore имеют в своем функционале распознавание нот. Поэтому процесс распознавания нот на нотном листе в них одинаков.

Также, существует множество подобных программ, у некоторых из которых меньший или больший функционал. Но принцип их работы одинаков. В некоторых из них нельзя прослушать мелодию, нельзя редактировать нотный стан. Но, процесс распознавания в них одинаков. Нужно загрузить изображение, содержащее мелодию, написанную с помощью нот, затем дождаться окончания процесса распознавания и обработки изображения, содержащее нотные символы, чтобы, в конечном итоге, получить нотную нотацию.

К сожалению, данные программы не лишены недостатков. Например, если на изображении будет что-то помимо листа, содержащего нотную нотацию, программа не сможет корректно обработать изображение и не выдаст результат. Также, изображение должно иметь определенный формат, такие как TIFF, PDF. То есть, файл с форматом PNG или JPEG нужно сначала конвертировать в любой из нужных форматов, после этого только будет возможно распознать и обработать изображение.

## **1.8. Постановка задачи дипломной работы.**

Цель данной дипломной работы заключается в разработки и реализации системы компьютерного зрения для автоматического распознавания нот на нотных листах. Главная цель работы заключается в создании алгоритма, способного обнаруживать и распознавать ноты на изображении нотного листа, а также извлекать информацию о их положении на нотном стане.

Конкретные задачи, которые необходимо выполнить для достижения цели работы, включают:

1. Предварительная обработка изображений нотных листов, включая преобразования для повышения качества изображения для улучшения дальнейшего обнаружения нот.
2. Разработка алгоритма для обнаружения нотных станов и извлечение информации об их положении на изображении.
3. Разработка алгоритма для обнаружения элементов, представляющих собой ноту.
4. Разработка алгоритма для распознавания высоты каждой ноты.
5. Проведение экспериментов и оценка эффективности разработанной системы на наборе изображений.
6. Анализ результатов и формулирование выводов о преимуществах и недостатках разработанной системы.

В итоге дипломная работа должная представлять собой полноценную систему, способную автоматически распознавать ноты на листах и предоставлять информацию о их положении и музыкальных характеристиках.

## **1.9. Выбор программных средств.**

В качестве языка программирования был выбран Python. Данный язык известен своей простотой и понятностью синтаксиса. Он имеет высокий уровень абстракции, что делает код легко читаемым и понятным для разработчиков. Python обладает обширной экосистемой библиотек и фреймворков, специализированных для обработки изображений. Для разработки алгоритма автоматического распознавания и классификации нотных символов на музыкальном листе была выбрана библиотека OpenCV.

Для воспроизведения отсканированного произведения была выбрана библиотека MIDIUtil. Также, для работы с многомерными массивами была использована выбрана библиотека NumPy.

OpenCV (Open-Source Computer Vision Library) – это библиотека с открытым исходным кодом, которая предоставляет набор функций и алгоритмов для обработки изображений и видео. Она была написана на C++ и предоставляет интерфейсы для использования на разных языках программирования, включая Python, Java, MATLAB и другие. Python-интерфейс OpenCV является наиболее популярным и широко используется в сообществе благодаря своей простоте и мощности.

Несомненными плюсами данной библиотеки являются обширный функционал, мультиплатформенность, большое количество документации и интеграция с другими библиотеками и инструментами, такими как NumPy, Scipy и Matplotlib.

MIDIUtil – это библиотека для Python, предназначенная для создания и манипулирования файлами формата MIDI (Musical Instrument Digital Interface). Она обеспечивает простой и удобный способ создания и редактирования музыкальных композиций в MIDI-формате. Данная библиотека предлагает простой и интуитивно понятный интерфейс для создания и редактирования MIDI-файлов. Она обеспечивает гибкость в настройке различных параметров и удобные методы для добавления музыкальных событий.

NumPy (Numerical Python) – это библиотека для языка программирования Python, предназначенная для работы с многомерными массивами и выполнения математических операций над ними. Она обеспечивает эффективные и удобные инструменты для выполнения операций линейной алгебры, статистики, случайных чисел и других вычислительных задач. NumPy реализована на языке C, что обеспечивает высокую скорость выполнения операций. Благодаря своей производительности и широкой поддержке в научно-исследовательском сообществе, является неотъемлемой частью стека инструментов для анализа данных и машинного обучения.

# Программная реализация

В ходе разработки системы для распознавания и воспроизведения музыкального произведения был реализован алгоритм, состоящий из нескольких подпрограмм:

1. Подготовка изображения.
2. Поиск и обнаружение нотных станов на подготовленном изображении.
3. Поиск и обнаружение на изображении объектов, похожих на ноту.
4. Поиск среди найденных элементов нот, находящихся на нотных станах и определение их высоты, относительно их положения на нотных станах.
5. Создание из найденных нот MIDI файла, для прослушивания и выявления неточностей работы.

Блок-схема приведенного алгоритма представлена на рисунке 2.1.

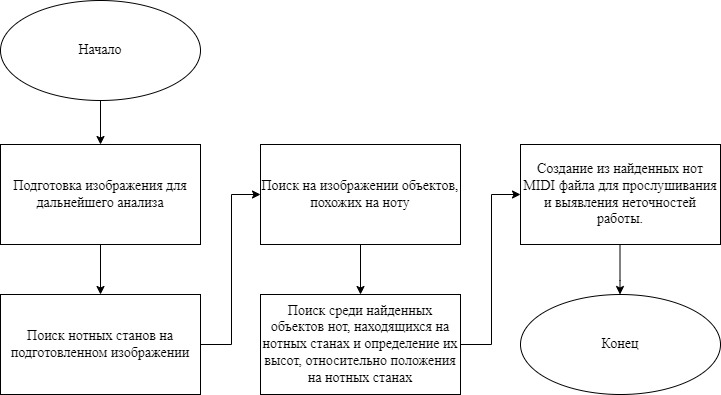


Рисунок 2.1 – Блок-схема алгоритма

## **2.1. Подготовка изображения.**

Данная подпрограмма предназначена для обработки изображения, поступающего на вход, с целью выделения нужного объекта на изображении и его коррекции или коррекция уже готового скана листа.

Подпрограмма содержит 3 функции: check(), adjust\_image() и adjust\_scan().

1. Функция check() принимает на вход изображение image формата JPEG или PNG, проверяет наличие посторонних объектов, помимо бумажного листа формата А4. Если присутствует посторонние объекты, то вызывается функция adjust\_image(), иначе вызывается функция adjust\_scan(). В зависимости от выбора одной из этих двух функций, на возврат подается обработанное изображение.
2. Функция adjust\_image() принимает на вход изображение image и выполняет следующие шаги:
   * + Преобразование изображения в оттенки серого с помощью функции cv2.cvtColor().
     + Применение размытия с использованием фильтра Гаусса с помощью функции cv2. GaussianBlur().
     + Обнаружение границ на размытом изображении с помощью алгоритма Кэнни с использованием функции cv2.Canny().
     + Поиск контуров на закрытом изображении с помощью функции cv2.findCountours().
     + Сортировка контуров по площади, для поиска наибольшего контура, который представляет собой лист формата А4 с нотами.
     + Поиск четырехугольника с четырьмя углами, который охватывает найденный контур.
     + Вычисление координат четырех углов листа.
     + Вычисление максимальной ширины и длины листа.
     + Создание массива с новыми координатами для преобразования перспективы, чтобы конечное изображение представляло из себя скан листа.
     + Применение преобразования перспективы с помощью функции cv2.getPerspectiveTransform() и cv2.warpPerspective().
     + Применение пороговой бинаризации с использованием функции cv2.threshold() для получения четкого изображения.
     + Изменение размера изображения в три раза с помощью функции cv2.resize().
     + Сохранение промежуточных изображений.
     + Возвращение результата коррекции изображения.
3. Функция adjust\_scan() принимает на вход изображение image и выполняет следующие шаги:
   * + Преобразование изображения в оттенки серого с помощью функции cv2.cvtColor().
     + Применение пороговой бинаризации для получения четкого изображения с помощью функции cv2.threshold().
     + Изменение размера изображения в три раза с помощью функции cv2.resize().
     + Сохранение измененного изображения в файл.
     + Возвращение результата коррекции.

## **2.2. Поиск и обнаружение нотных станов на изображении.**

Данная подпрограмма предназначена для того, чтобы на изображении, полученном в результате коррекции из предыдущей подпрограммы, найти нотные станы, для дальнейшего определения нот, содержащихся на них.

Подпрограмма состоит из класса Staff и пяти функций: preprocess\_image(), detect\_lines(), detect\_staffs(), get\_staffs() и draw\_staffs().

1. Функция preprocess\_image() принимает на вход скан нотного листа image, обработанного в предыдущей подпрограмме.

Данная функция делает предварительную обработку входного изображения и возвращает изображение с границами объектов. Предварительная обработка включает в себя преобразование изображения в оттенки серого, бинаризацию с пороговым значением, применение эрозии для удаления шумов, а также обнаружение границ с помощью алгоритма Кэнни. Обработанное изображение сохраняется в файл и затем возвращается.

1. Функция detect\_lines() используется для обнаружения горизонтальных линий на изображении и добавления их в список. На вход функции подаются следующие аргументы:
   * + processed\_image – обработанное черно-белое изображение, полученное из предыдущей функции preprocess\_image.
     + image – исходное изображение, для того, чтобы нарисовать на нем линии.
     + nlines – максимальное количество линий, которое требуется обнаружить.

Функция использует преобразование Хафа (cv2.HoughLines()) для поиска горизонтальных линий на обработанном изображении.

Преобразование Хафа (Hought transform) – это метод, используемый для обнаружения геометрических фигур, таких как прямые линии, окружности или эллипсы на изображении. Преобразование Хафа было разработано в 1962 году Ричардом Дудли Хаформ и было первоначально предложено для обнаружения линий на изображениях.

Преобразование основано на математической теории, согласно которой каждая точка на изображении может быть представлена в виде параметрического пространства. Например, для прямых линий, каждая точка (x, y) на изображении может быть представлена в виде параметрической формы , где *r* – длина радиус-вектора ближайшей к началу координат точки на прямой (длина нормали к прямой, проведенной из начала координат), а – угол между этим вектором и осью абсцисс.

Функция cv2.HoughLines() применяет преобразование Хафа для обнаружения прямых линий на изображении. Она возвращает массив линий, представленных в виде ().

Затем, функция проходит по первым nlines линиям из результатов преобразования Хафа для извлечения параметров каждой линии (расстояние и угол). Далее, с использованием этих параметров, вычисляются начальные и конечные координаты каждой линии.

После этого идет проверка разницы по вертикали между начальной и конечной координатой линии. Если эта разница меньше некоторого заданного количества пикселей, то средняя координата по оси y добавляется во множество all\_lines. Кроме того, функция отрисовывает найденные линии на цветном изображении image.

В конце функция сохраняет цветное изображение с отрисованными линиями в файл и возвращает список всех обнаруженных линий all\_lines и цветное изображение lines\_image\_color.

1. Функция detect\_staffs() используется для нахождения нотных станов на изображении на основе обнаруженных линий. Она принимает список всех обнаруженных линий all\_lines в качестве входного аргумента.

Вначале создается пустой список staffs, в котором будут храниться координаты нотных станов. Далее функция проходит по всем линиям в списке all\_lines, который был отсортирован по возрастанию. Если текущая и предыдущая линии расположены на расстоянии больше заданного порога, то считается, что это новый нотный стан, и в список staffs добавляются координаты первой и последней линии предыдущего нотного стана.

После обработки всех линий, функция возвращает список staffs начала и конца каждого нотного стана.

1. Функция draw\_staffs() используется для отрисовки нотных станов на исходном изображении. Она принимает два аргумента: image – исходное изображение, на котором отрисованы линии и staffs – список координат нотных станов.

Данная функция для каждого нотного стана рисует две горизонтальные линии, которые являются его началом и концом. В конце функция сохраняет изображение в файл.

1. Класс Staff представляет модель нотного стана и содержит информацию о нотном стане, такую как координаты и приблизительные положения строк на пятилинейном стане.

Атрибуты класса:

* + - min\_range – минимальное значение координаты нижней линии нотного стана. Это значение определяет нижнюю границу нотного стана на изображении.
    - max\_range – Максимальное значение координаты верхней линии нотного стана. Это значение определяет верхнюю границу нотного стана на изображении.
    - lines\_location – Список приблизительных позиций строк на пятилинейном стане. Каждый элемент списка представляет собой координату по вертикали, где находится строка на нотном стане.
    - lines\_distance – Расстояние между строками на пятилинейном стане. Это значение используется для вычисления приблизительных позиций строк на нотном стане.

Методы класса Staff:

* + - get\_lines\_locations() – метод вычисляет приблизительное положение каждой строки на пятилинейном нотном стане. Он используется min\_range и max\_range для определения высоты нотного стана и вычисляет равные интервалы между строками на основе этого диапазона. Метод возвращает список приблизительных позиций строк на нотном стане.

1. Функция get\_staffs() используется для нахождения нотных станов на изображении и возвращает список найденных нотных станов. Данная функция является главной функцией, в которой вызываются все предыдущие функции.

Сначала вызывается функция preprocess\_image(), которая возвращает обработанное изображение processed\_image и бинаризированное изображение thresholded. Далее вызывается функция detect\_lines(), которая находит горизонтальные линии на изображении processed\_image и возвращает список координат этих линий all\_lines и изображение с отрисованными линиями lines\_image\_color.

После нахождения линий вызывается функция detect\_staffs, которая на основе списка координат линий all\_lines определяет координаты нотных станов и возвращает список найденных нотных станов staffs.

## **2.3. Поиск и обнаружение на изображении объектов, похожих на ноту.**

Блоб (англ. blob) в компьютерном зрении обычно относится к группе связанных пикселей или регионов на цифровом изображении, которые имеют общие характеристики или свойства. Блобы могут представлять объекты или области определенного интереса на изображении.

Данная подпрограмма используется для поиска блобов на изображении, которые представляют ноты на нотном листе. Она состоит из одной функции detect\_blobs(), которая принимает на вход два аргумента: input\_image – изображение, на котором нужно найти ноты и staffs – список нотных станов, ранее найденных с помощью функции get\_staffs().

В подпрограмме используется SimpleBlobDetector (Простой детектор блобов) – это алгоритм компьютерного зрения, предназначенный для обнаружения и извлечения простых объектов (блобов) на изображении. Этот алгоритм основан на анализе яркости и формы регионов на изображении.

Простой детектор блобов обычно используется для обнаружения и извлечения объектов, которые имеют определенную яркости и размер, и являются изолированными на изображении. Примерами таких объектов могут быть круглые маркеры, точки, пузырьки, монеты и другие подобные структуры.

Детектор обладает различными параметрами, которые можно настроить для достижения желаемых результатов. В данном случае для того, чтобы обнаружить ноту, были использованы следующие параметры:

* filterByArea (Фильтрация по площади) – определяет, следует ли фильтровать блобы на основе их площади. Принимает значения True (Истина) или False (Ложь). В данном случае значение – True.
* minArea (Минимальная площадь) задает минимальную площадь блоба, которая должна быть обнаружена. Принимает целочисленное значение, представляющее минимальную площадь в пикселях.
* maxArea (Максимальная площадь) задает максимальную площадь блоба, которая должна быть обнаружена. Принимает целочисленное значение, представляющее максимальную площадь в пикселях.
* filterByCircularity (Фильтрация по округлости) – определяет, следует ли фильтровать блобы на основе их округлости. Принимает значения True (Истина) или False (Ложь). В данном случае значение – True.
* minCircularity (Минимальная округлость) – задает минимальное значение округлости блоба, которое должно быть обнаружено. Принимает значения от 0 до 1, где 1 – полностью круглый блоб.
* filterByConvexity (Фильтрация по выпуклости) – определяет, следует ли фильтровать блобы на основе их выпуклости. Принимает значения True или False. В данном случае значение – True.
* minConvexity (Минимальная выпуклость) – задает минимальное значение выпуклости блоба, которое должно быть обнаружено. Принимает значения от 0 до 1, где 1 – полностью выпуклый блоб.
* filterByInertia (Фильтрация по инерции) – определяет, следует ли фильтровать блобы на основе их инерции. Принимает значения True или False. В данном случае значение – True.
* minInertiaRatio (Минимальное значение инерции) – задает минимальное значение инерции блоба, которое должно быть обнаружено. Принимает значения от 0 до 1, где 1 – высокая инерция (полностью круглый блоб).

Алгоритм работы функции detect\_blobs:

1. Исходное изображение копируется, копия изображения инвертируется для удобства последующей обработки.
2. Производится морфологическая трансформация для удаления горизонтальных линий на изображении.
3. Также производится морфологическая трансформация для удаления вертикальных линий на изображении.
4. Создается объект cv2.SimpleBlobDetector\_Params() и настраиваются параметры поиска нот. Нота представляет из себя круг или похожий на круг объект.
5. Создается детектор блобов с настроенными параметрами cv2.SimpleBlobDetector\_create().
6. Производится поиск блобов на изображении с помощью детектора, найденные блобы сохраняются в переменную keypoints.
7. Найденные блобы отображаются на изображении im\_with\_blobs с помощью функции cv2.drawKeypoints().
8. Выполняется нумерация найденных нот на изображении с использованием информации о расположении нотных станов. Для того, чтобы определить, что блоб находится на нотном станах, сначала вычисляется координаты начала и конца всех нотных станов. Далее, с помощью функции библиотеки NumPy – digitize(), каждый блоб помещается в одну из категорий. Например, если у нас 2 нотных стана, то у нас получается 5 категорий: 0 – блоб находится над началом первого стана, 1 – находится внутри первого нотного стана (координата блоба по вертикали находится между координатами нотного стана), 2 – находится между первым и вторым нотном станом, 3 – находится внутри второго нотного стана и 4 – находится ниже второго нотного стана. Следовательно, каждая нота относится к нечетной категории.
9. Номера нот отображаются на изображении im\_with\_numbers, а также на изображении im\_with\_blobs.
10. Изображения с нумерацией нот сохраняются в файлы.
11. Возвращается отсортированный список найденных нот в формате (keypoints, keypoints\_staff), где keypoints содержит информацию о найденных блобах(нотах), а keypoints\_staff – номера нотных станов, к которым относятся блобы.

Подрограмма позволяет обнаружить и пронумеровать ноты на нотном листе с использованием информации о нотных станах, найденных ранее.

## **2.4. Поиск среди найденных элементов нот, находящихся на нотных станах и определения их высот, относительно их положения.**

Данная подпрограмма предназначена для обнаружения нотных символов на изображении нотных станов и их позиции и высоты, а также для визуализации результатов и конвертации в формат midi. Данная подпрограмма состоит из класса Note и трех функций: extract\_notes(), draw\_notes\_pitch(), notes\_to\_midi().

1. Словарь clef представляет соответствие между позициями на нотном стане и нотами скрипичного ключа. Ключами в словаре являются целочисленные значения, которые представляют позиции на нотном стане, а значениями являются строковые представления соответствующих нот.
2. Класс Note представляет собой модель для представления ноты в контексте музыкального нотного стана. Он содержит методы и атрибуты, которые позволяют обрабатывать и анализировать информацию о ноте.

Атрибуты класса:

* + - staff\_no – номер нотного стана, на котором находится нота.
    - staffs – список всех нотных станов.
    - blob – блоб с нотным символом (ключевая точка, обнаруженная на изображении).
    - clef – словарь, содержащий соответствие позиции на нотном стане и ноты. В данном случае в качестве соответствия позиция – нота взят скрипичный ключ. Скрипичный ключ – символ, используемый в нотной записи для обозначения позиции ноты Соль(G) на нотном стане.

Методы класса:

* detect\_position\_on\_staff() – метод для определения позиции ноты на нотном стане. Принимает объект staff и блоб с нотным символом - blob. Внутри метода вычисляется расстояние от координат блоба до каждой линии на нотном стане. Затем определяется позиция ноты на основе этих расстояний.
* detect\_pitch() – метод определения высоты ноты относительно ее позиции на нотном стане. Принимает позицию ноты на нотном стане position\_on\_staff и использует словарь clef, чтобы получить соответствующую ноту для данной позиции.

Объект класса представляет отдельную ноту, обнаруженную на изображении. При создании объекта ноты передаются необходимые параметры, такие как номер нотного стана, список всех нотных станов, “блоб” с нотным символом и словарь с соответствием позиции ноты на нотном стане и названии ноты.

1. Функция extract\_notes() выполняет обнаружение нот из блобов и возвращает список всех обнаруженных нот. Функция принимает три аргумента: blobs – список блобов, staffs – информация о нотных станах. Внутри функции происходит итерация по каждому блобу. Если номер блоба нечетный, то определяется номер нотного стана, к которому принадлежит блоб, и создается объект класса Note с помощью найденной информации. В конце функции выводится информация о количестве найденных нот, и список notes возвращается.
2. Функция draw\_notes\_pitch() принимает изображение image и список нот notes. Она выполняет отрисовку названия нот на изображении. Для каждой ноты в списке она использует функцию cv2.putText() для размещения названия ноты на изображении. Функция только отрисовывает ноты, ничего не возвращая.
3. Функция notes\_to\_midi() принимает список нот notes и преобразует в список, который содержит в себе высоту каждой ноты. Этот список возвращается в конце функции.

## **2.5. Создание из найденных нот MIDI-файла для прослушивания и анализа.**

Данная подпрограмма создает MIDI-файл на основе переданных нот.

MIDI (Musical Instrument Digital Interface) – стандартный протокол, используемый для передачи музыкальной информации между электронными инструментами, компьютерами и другими устройствами, связанными с музыкой.

Формат MIDI представляет собой набор числовых данных, которые описывают различные аспекты музыки, такие как нажатие и отпускание клавиш, длительность звуков, силу звучания и другие параметры. MIDI-данные могут быть записаны, сохранены и воспроизведены с помощью поддерживающего MIDI-устройства или программного обеспечения.

Файлы, содержащие MIDI-данные, могут быть воспроизведены на компьютере, синтезаторе, цифровом фортепиано, секвенсоре или другом устройстве. Это позволяет музыкантам записывать, обмениваться и исполнять музыкальные произведения с использованием различных инструментов и звуковых эффектов.

Подпрограмма состоит из одной функции mk\_track(), которая принимает на вход список нот (notes) в виде их названий, полученный из предыдущей подпрограммы.

Вначале создается файл с одним треком для последующего добавления в него нот. Далее задается время начала ноты и длительность каждой ноты. Из-за того, что в данной программе анализировалась только высота нот, длительность нот не учитывается. Поэтому конечное произведение представляет из себя набор звуков для сверки, а не готовое музыкальное произведение. После задания времени начала ноты и длительностей, устанавливается темп произведения и громкость.

Далее создается словарь, который связывает названия нот с их числовыми позициями в MIDI.

В стандарте MIDI каждая нота представлена числовой позицией, называемой «номером ноты» или «MIDI-номером». Номер ноты определяет высоту звука ноты.

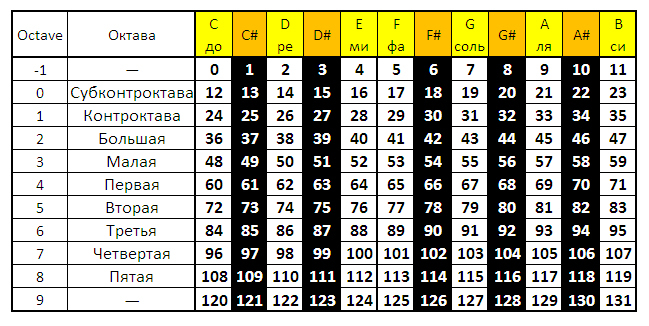


Рисунок 2.2 – Таблица MIDI-номеров

После создания словаря, происходит подготовка списка номеров нот, соответствующих переданным нотам. Если нота, присутствует в словаре, ее числовая позиция добавляется в список, который после передается в MIDI-файл.

В конце происходит сохранения MIDI-файла для дальнейшего прослушивания и анализа распознавания нот.

# Тестирование разработанного алгоритма.

## **3.1. Подготовка тестовых изображений.**

В ходе подготовки тестовых данных были собраны разнообразные фотографии, на которых присутствуют нотные листы с музыкальными произведениями. Также были собраны сканы нотных листов. Качество изображение существенно влияет на точность конечных данных.

Так как алгоритм не распознает музыкальный ключ, в зависимости от которого названия нот может варьироваться, был выбран только скрипичный ключ. Также алгоритм не распознает музыкальные знаки, повышающие или понижающие ноту на полутон или тон, а только распознает конкретную ноту, прослушивание MIDI-файла исказит восприятие музыкального произведения.

Для примера было выбрано два изображение. Первое изображение представляет собой скан музыкального произведения (рисунок 3.1), второе – фотографию, содержащую лист с музыкальным произведением (рисунок 3.2).

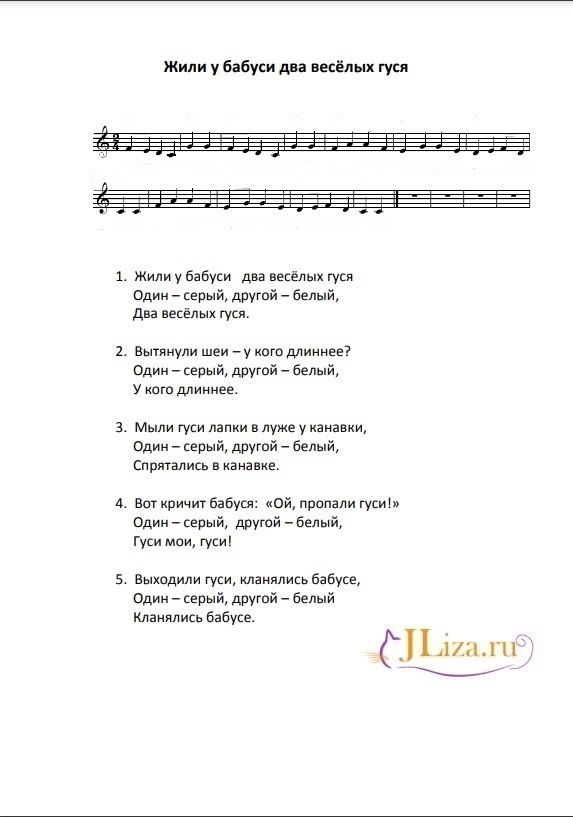


Рисунок 3.1 – Скан музыкального произведения

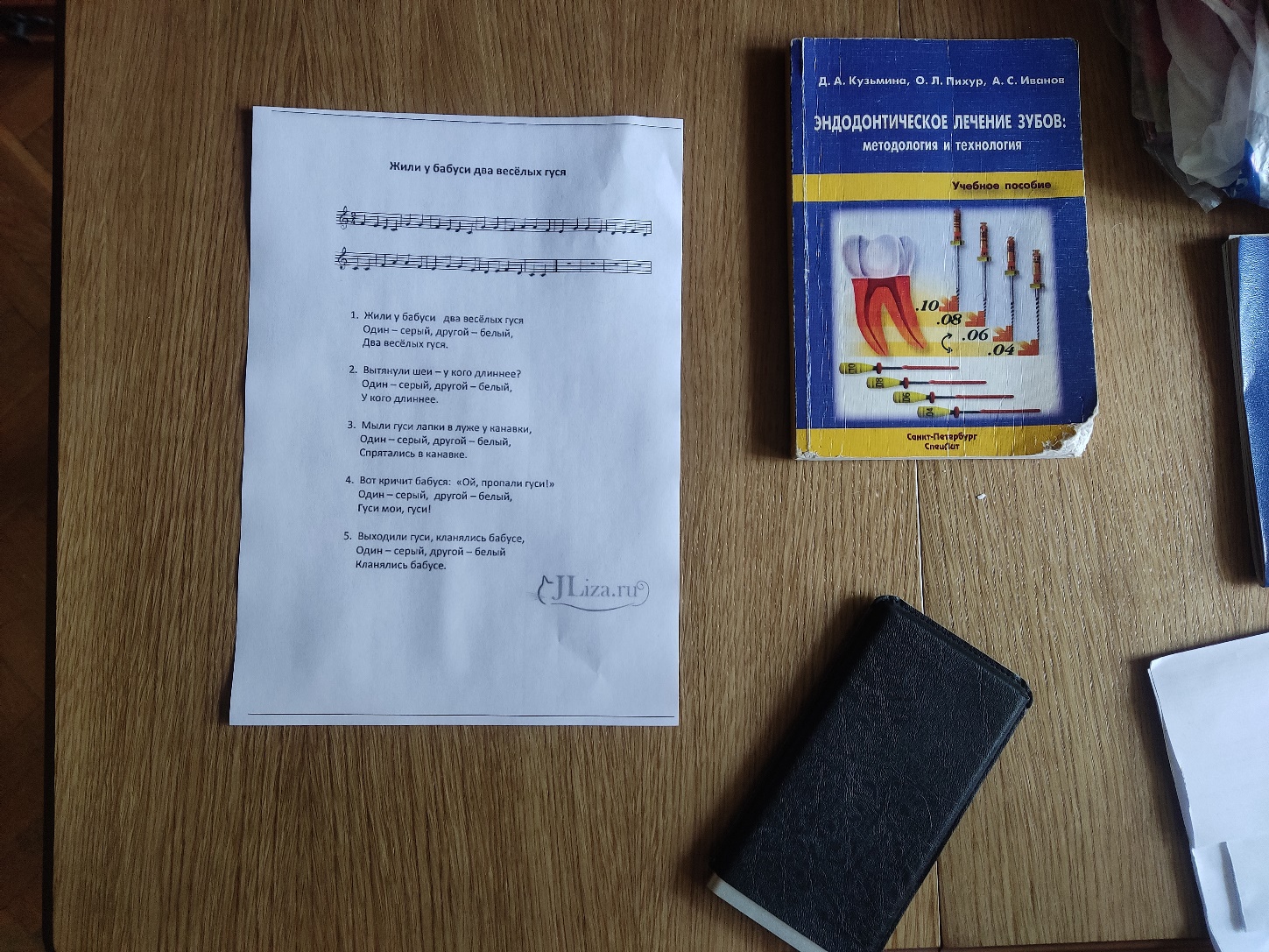


Рисунок 3.2 – Фотография, содержащая лист с музыкальным произведением

### **Тестирование работы на примере скана листа.**

Протестируем работу алгоритма для рисунка 3.1. Так как рисунок 3.1 представляет собой скан, процесс поиска листа формата А4 для него бесполезен. Следовательно, сразу на скане будет произведен поиск линий.

На рисунке 3.3 представлен результат поиска линий на изображении. Каждая найденная линия выделяется красным цветом для наглядности.



Рисунок 3.3 – Результат поиска линий на изображении

Далее, из найденных линий нужно определить те линии, которые являются началом и концом нотного стана. На рисунке 3.4 представлен результат поиска нотных станов. Каждый нотный стан сверху и снизу помечен зеленой линией для наглядности.



Рисунок 3.4 – Результат поиска нотных станов

После нахождения нотных станов происходит поиск всех блобов на изображении для дальнейшего определения нот, находящихся на нотных станах. На рисунке 3.5 представлен результат поиска. Каждый найденный блоб помечен красным кругом. На рисунке убираются горизонтальные и вертикальные линии для более корректного поиска нужных элементов.



Рисунок 3.5 – Результат поиска блобов на изображении

Для того, чтобы определить, какой блоб является нотой, а какой ей не является, нужно определить, к какой категории он относится. На рисунке 3.6 представлены результаты соотнесения блобов к категориям. Если блоб имеет нечетный номер – то он находится на нотном стане и является нотой.

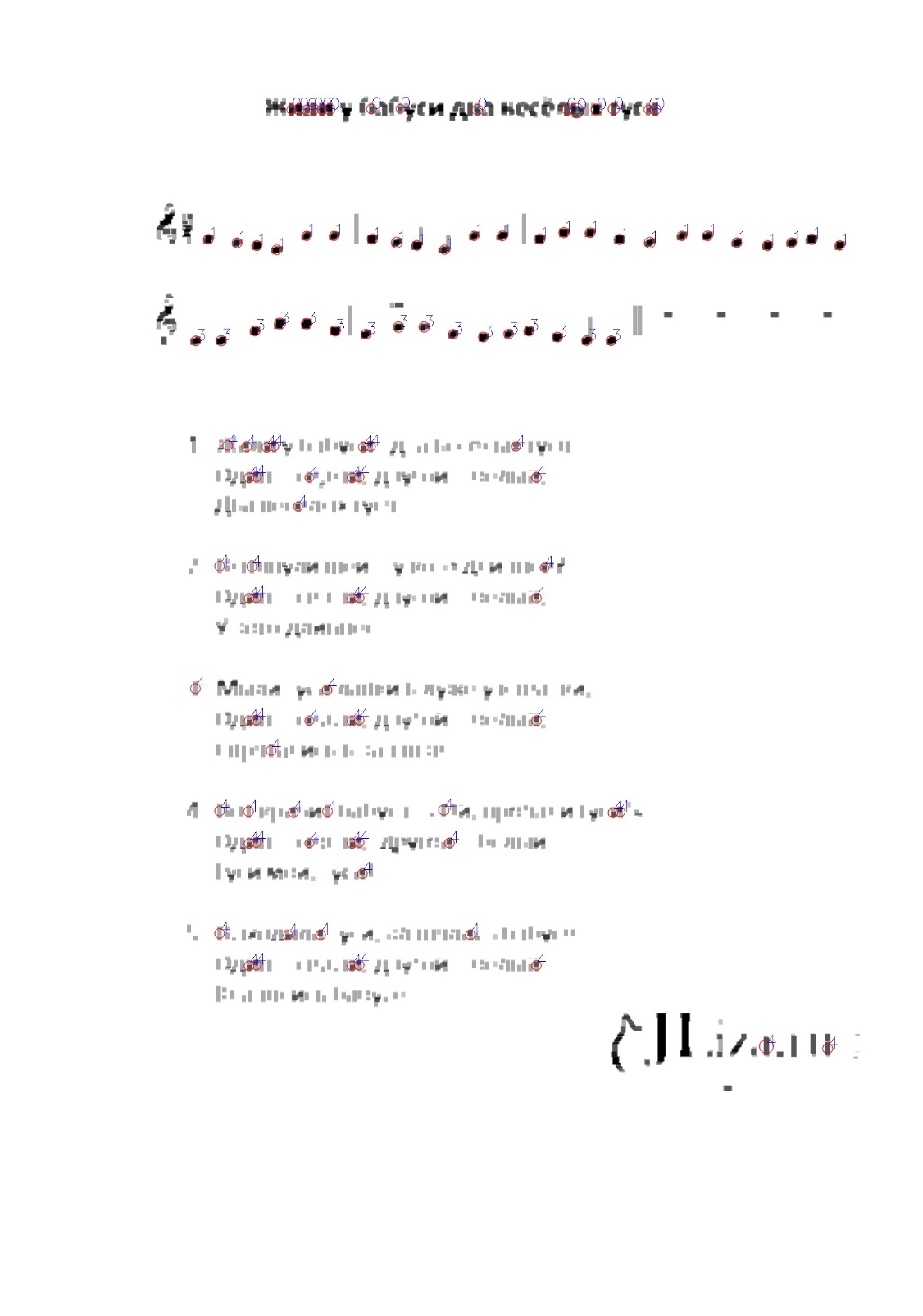


Рисунок 3.6 – Классификация блобов

Последним является определение высоты каждой ноты. На рисунке 3.7 для каждой ноты определена их высота. Для каждой ноты определяется название и номер октавы, к которой она относится. Названия и номера октав подписываются на изображении.

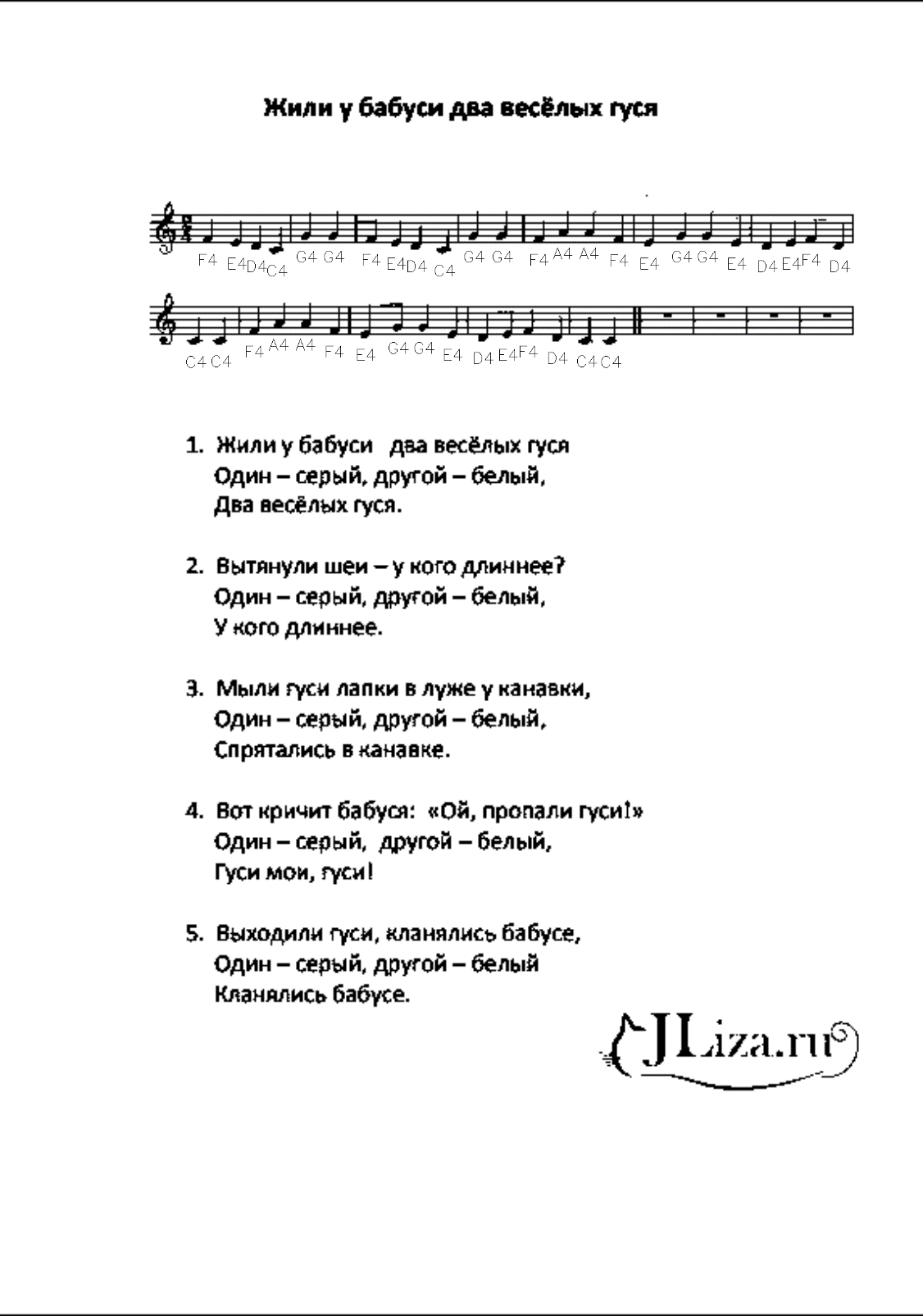


Рисунок 3.7 – Результат работы программы, каждая нота подписана

### **Проверка корректности работы программы.**

Для того, чтобы проверить корректность каждой определенной ноты, можно самостоятельно проверить каждую ноту, или воспользоваться программой SmartScore X2 Songbook.

Для самостоятельного определения ноты нужно иметь знания в музыкальной теории. Поэтому данный способ проверки рассмотрен не будет.

Для проверки в SmartScore X2 Songbook можно самостоятельно создать нотный лист с данным произведением с помощью редактора. Для того, чтобы создать файл нужно выполнить следующие шаги:

1. Создать новый проект.
2. Выбрать параметры проекта, такие как название произведения, композитор, ключевые особенности и т.д. (данный шаг ни на что не влияет, поэтому его можно пропустить).
3. После создания проекта откроется редактор нотного произведения. Здесь можно создать ноты, добавить нотные символы, изменять ритм и другие свойства.
4. Для добавления ноты необходимо выбрать инструмент из панели инструментов и щелкнуть на нотном стане в нужном месте. Нота будет добавлена в выбранной позиции.
5. Когда произведение готово, сохранить файл.

На рисунке 3.8 представлено тестируемое произведение, созданное в редакторе.

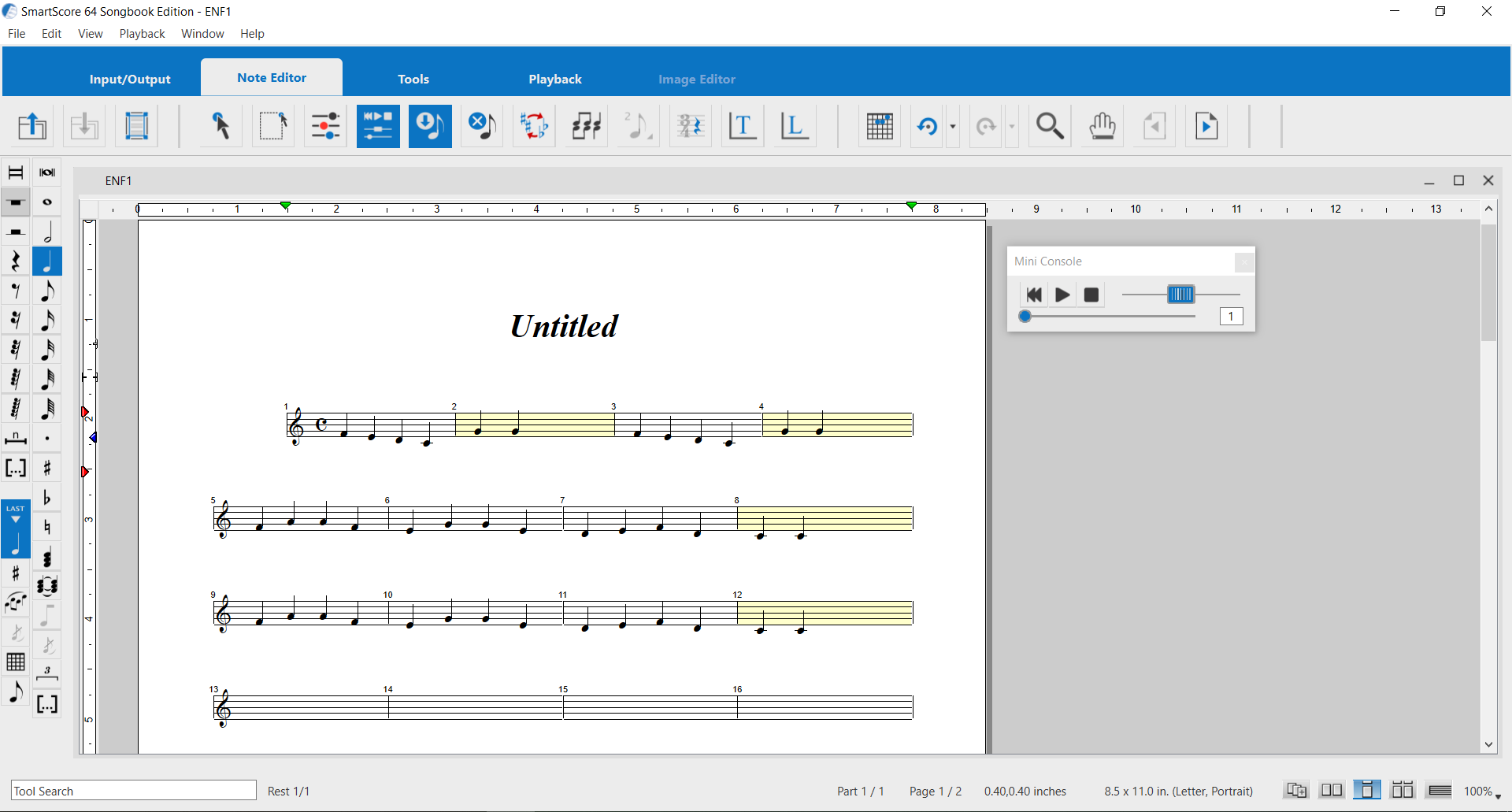


Рисунок 3.8 – Тестируемое произведение

Для того, чтобы получить список нот с названиями, воспользуемся просмотром списка событий (Event list). На рисунке 3.9 показан список нот данного тестируемого произведения.

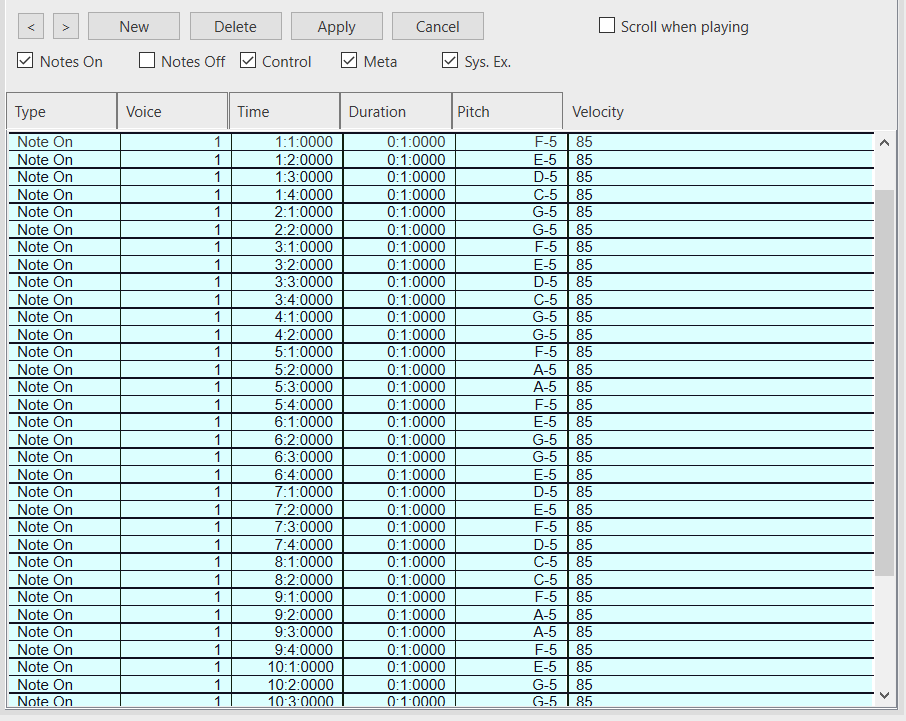


Рисунок 3.9 – Список, содержащий информацию о каждой ноте

Перепишем список в удобный формат – [F-5, E-5, D-5, C-5, G-5, G-5, F-5, E-5, D-5, C-5, G-5, G-5, F-5, A-5, A-5, F-5, E-5, G-5, G-5, E-5, D-5, E-5, F-5, D-5, C-5, C-5, F-5, A-5, A-5, F-5, E-5, G-5, G-5, E-5, D-5, E-5, F-5, D-5, C-5, C-5]

Также, получим список нот, распознанных с помощью программы. На рисунке 3.10 представлен вывод списка нот в консоль.

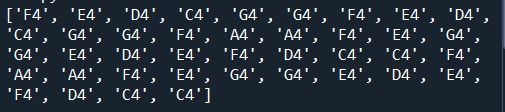


Рисунок 3.10 – Список распознанных нот на нотном листе

Перепишем список в удобный формат - ['F4', 'E4', 'D4', 'C4', 'G4', 'G4', 'F4', 'E4', 'D4', 'C4', 'G4', 'G4', 'F4', 'A4', 'A4', 'F4', 'E4', 'G4', 'G4', 'E4', 'D4', 'E4', 'F4', 'D4', 'C4', 'C4', 'F4', 'A4', 'A4', 'F4', 'E4', 'G4', 'G4', 'E4', 'D4', 'E4', 'F4', 'D4', 'C4', 'C4']

После сверки можно увидеть, что списки идентичны, следовательно программа распознала все ноты без ошибок. Цифры 5 и 4 соответствуют одной и той же октаве – первой.

### **Тестирование на примере фотографии, содержащей лист с музыкальным произведением.**

Протестируем работу алгоритма для рисунка 3.2. Рисунок 3.2 представляет собой фотографию, на которой присутствует лист с тем же музыкальным произведением, что и на рисунке 3.1. Лист специально был помят для того, чтобы показать все неточности выбранного алгоритма.

Сперва на фотографии нужно определить лист, выделить его и создать изображение, представляющее собой скан. На рисунке 3.10 представлен скан найденного листа с музыкальной нотацией. Можно заметить, что линии нотных станов на изображении не прямые, а, так как алгоритм вычисляет примерное положение линий нотного стана, в распознавании нот будут допущены ошибки.

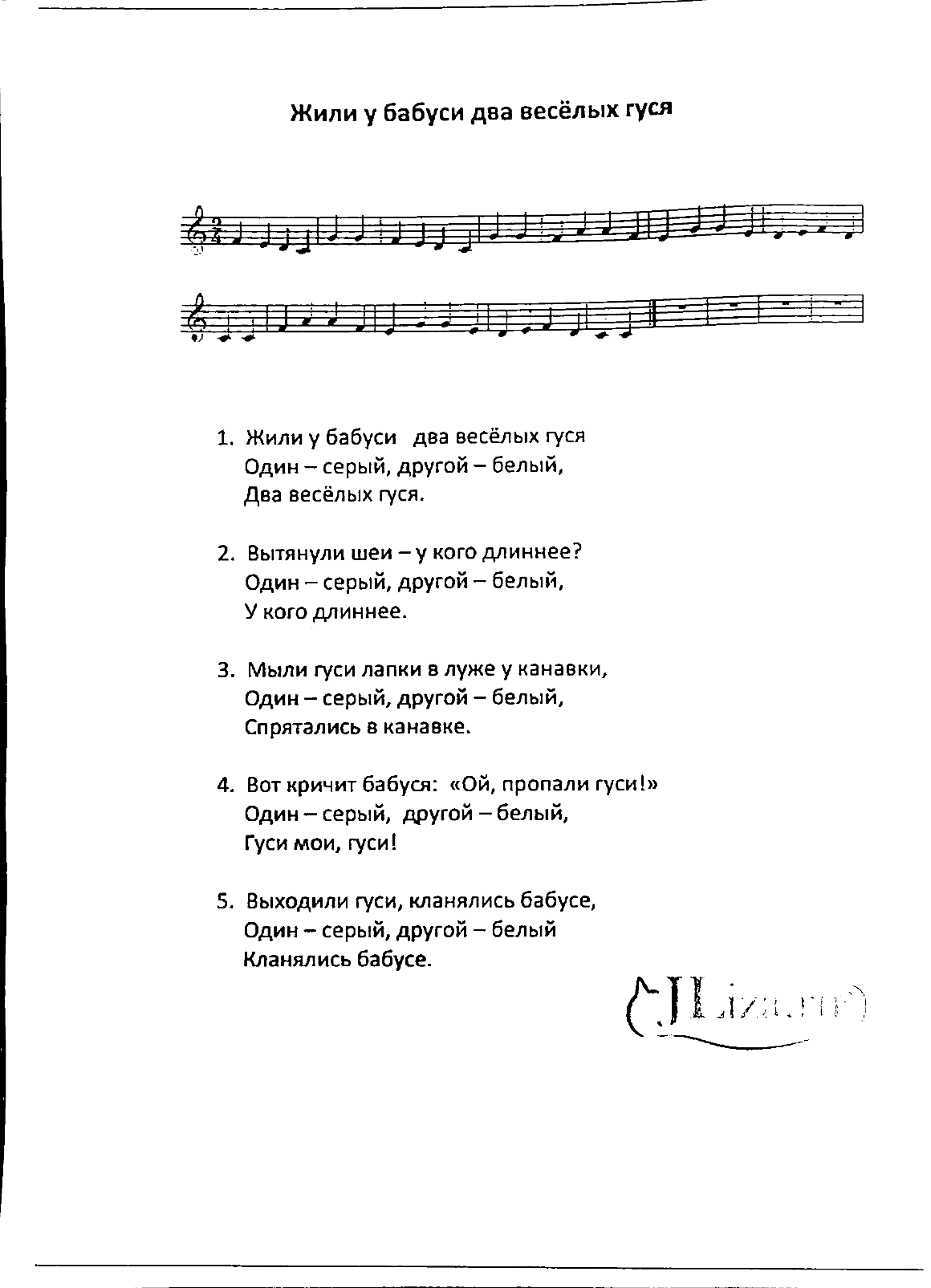


Рисунок 3.10 – Лист, найденный на фотографии и представленный в виде скана

Далее, проводится поиск линий на изображении. На рисунке 3.11 представлен результат поиска линий на изображении.

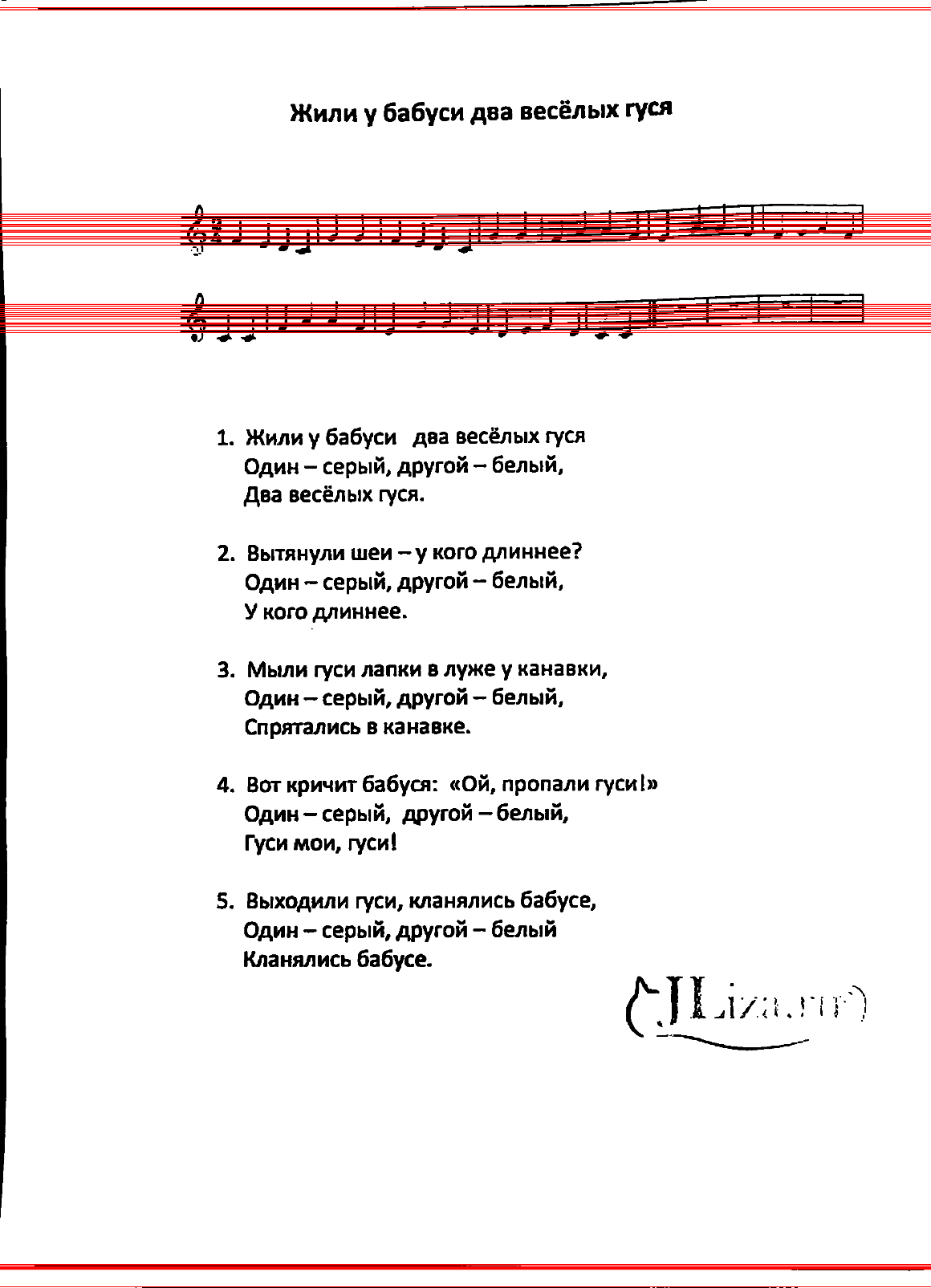


Рисунок 3.11 – Результат поиска линий на изображении

Далее, из найденных линий нужно определить те линии, которые являются началом и концом нотного стана. На рисунке 3.12 представлен результат поиска нотных станов. Каждый нотный стан сверху и снизу помечен зеленой линией для наглядности.

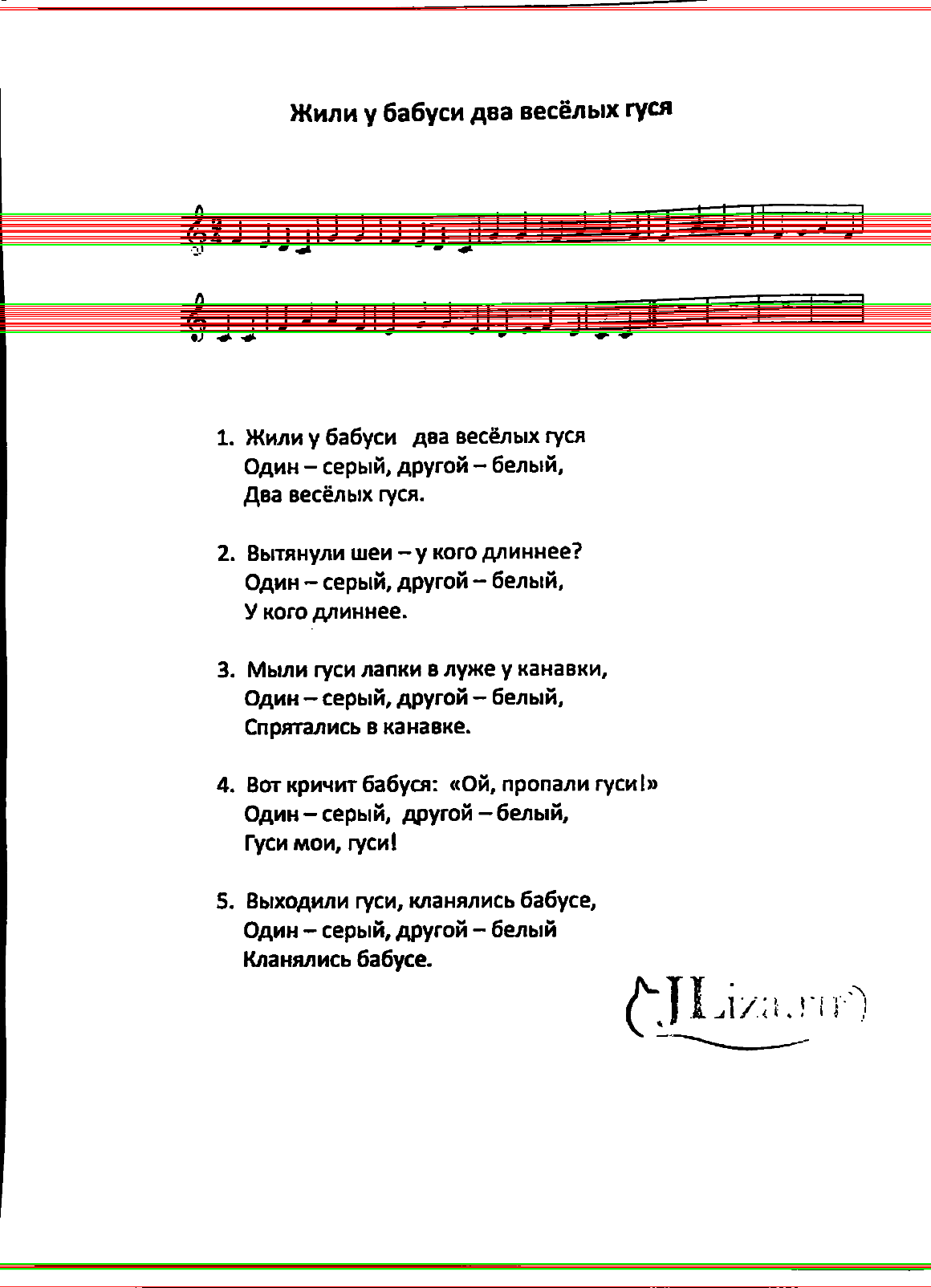


Рисунок 3.12 – Результат поиска нотных станов на изображении

На рисунке 3.12 можно заметить, что ноты на нотных станах не будут определены верно, потому что изображение искажено. Линии нотных станов уходят выше, чем нотный стан, определенный программой.

После нахождения нотных станов происходит поиск всех блобов на изображении для дальнейшего определения нот, находящихся на нотных станах. На рисунке 3.13 представлен результат поиска. Каждый найденный блоб помечен красным кругом. На рисунке убираются горизонтальные и вертикальные линии для более корректного поиска нужных элементов.

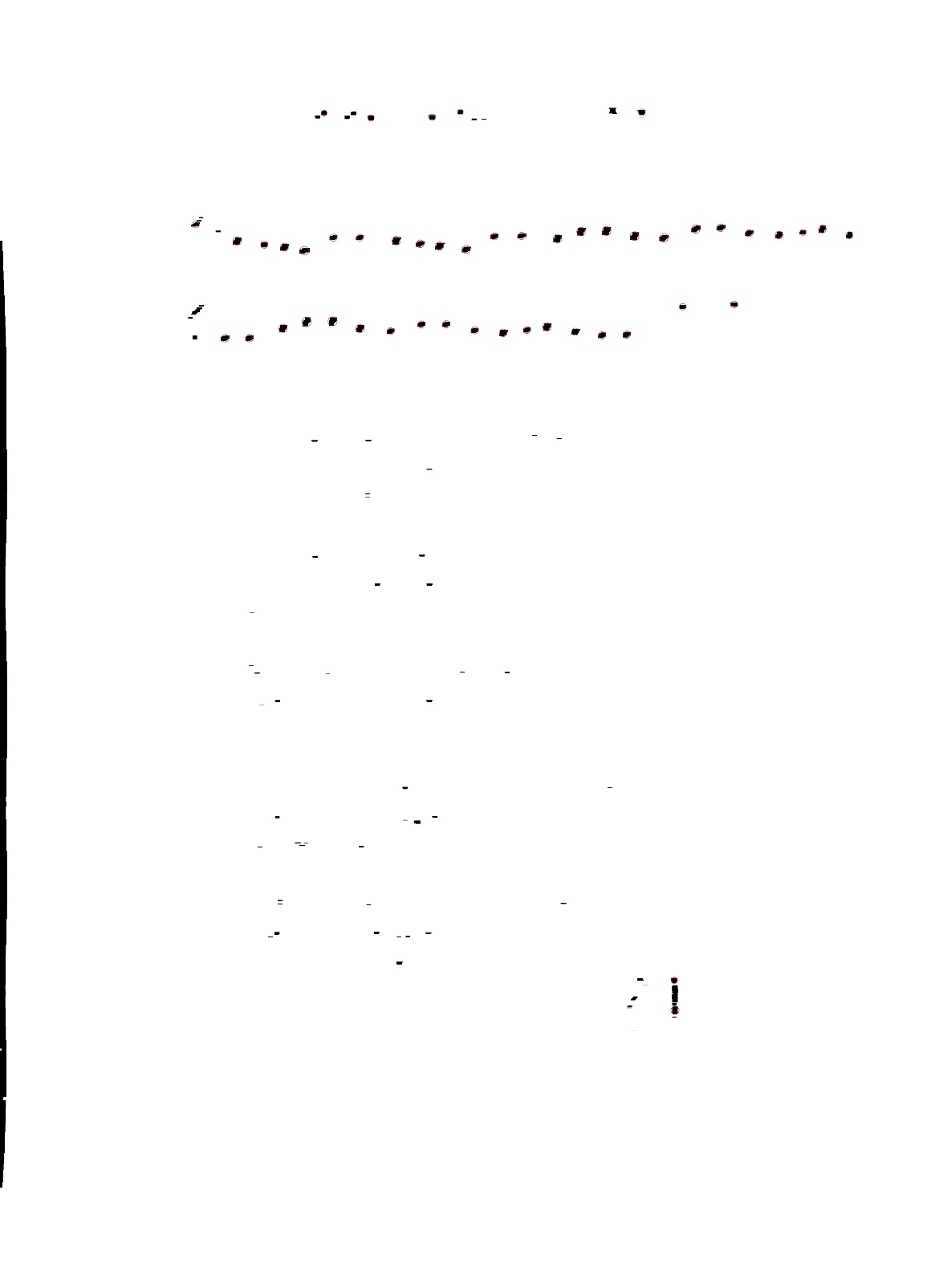


Рисунок 3.13 – Результат поиска блобов на изображении

Для того, чтобы определить, какой блоб является нотой, а какой ей не является, нужно определить, к какой категории он относится. На рисунке 3.14 представлены результаты соотнесения блобов к категориям. Если блоб имеет нечетный номер – то он находится на нотном стане и является нотой.

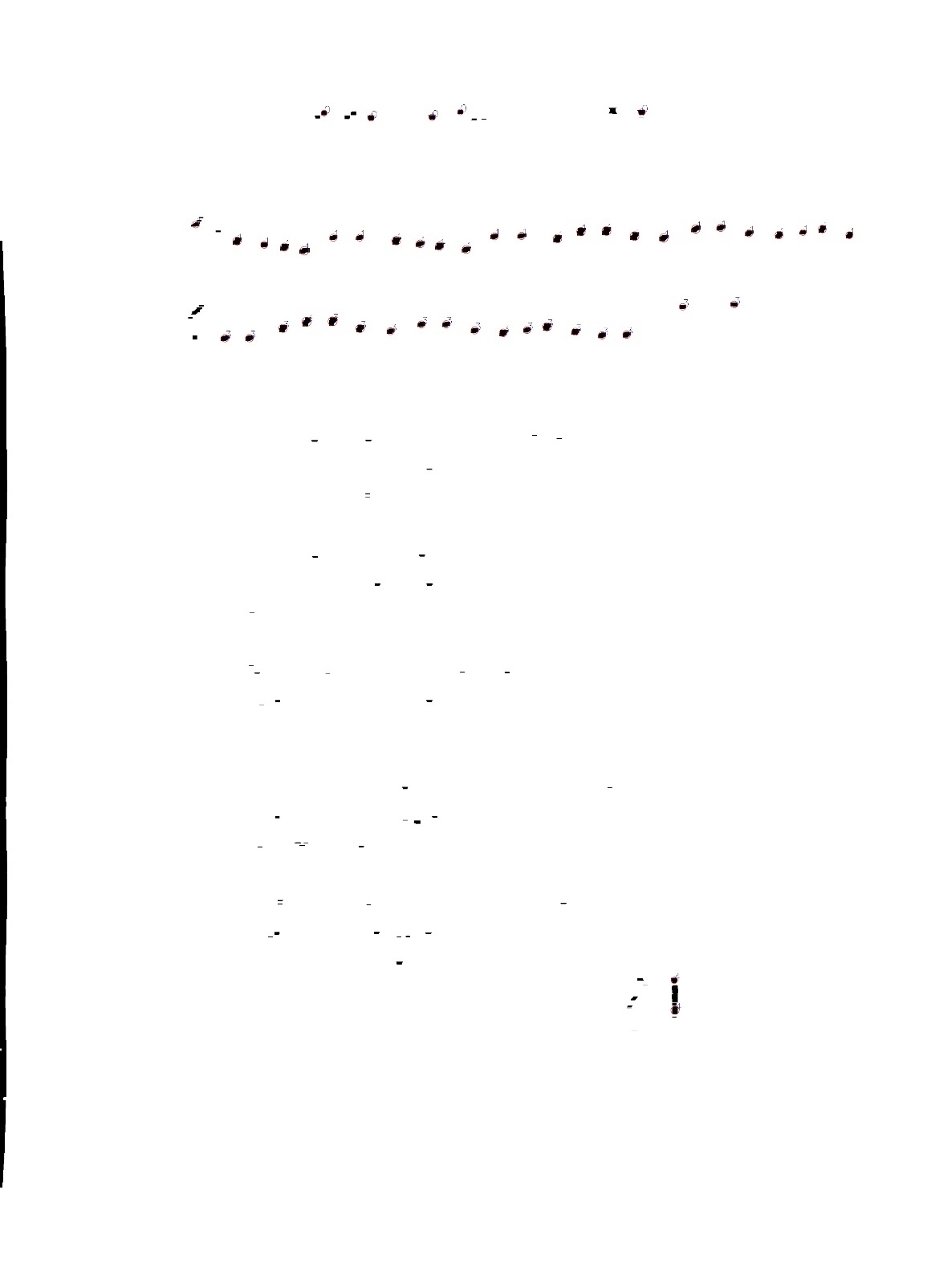


Рисунок 3.14 – результат поиска блобов

Последним является определение высоты каждой ноты. На рисунке 3.15 для каждой ноты определена их высота. Для каждой ноты определяется название и номер октавы, к которой она относится. Названия и номера октав подписываются на изображении.

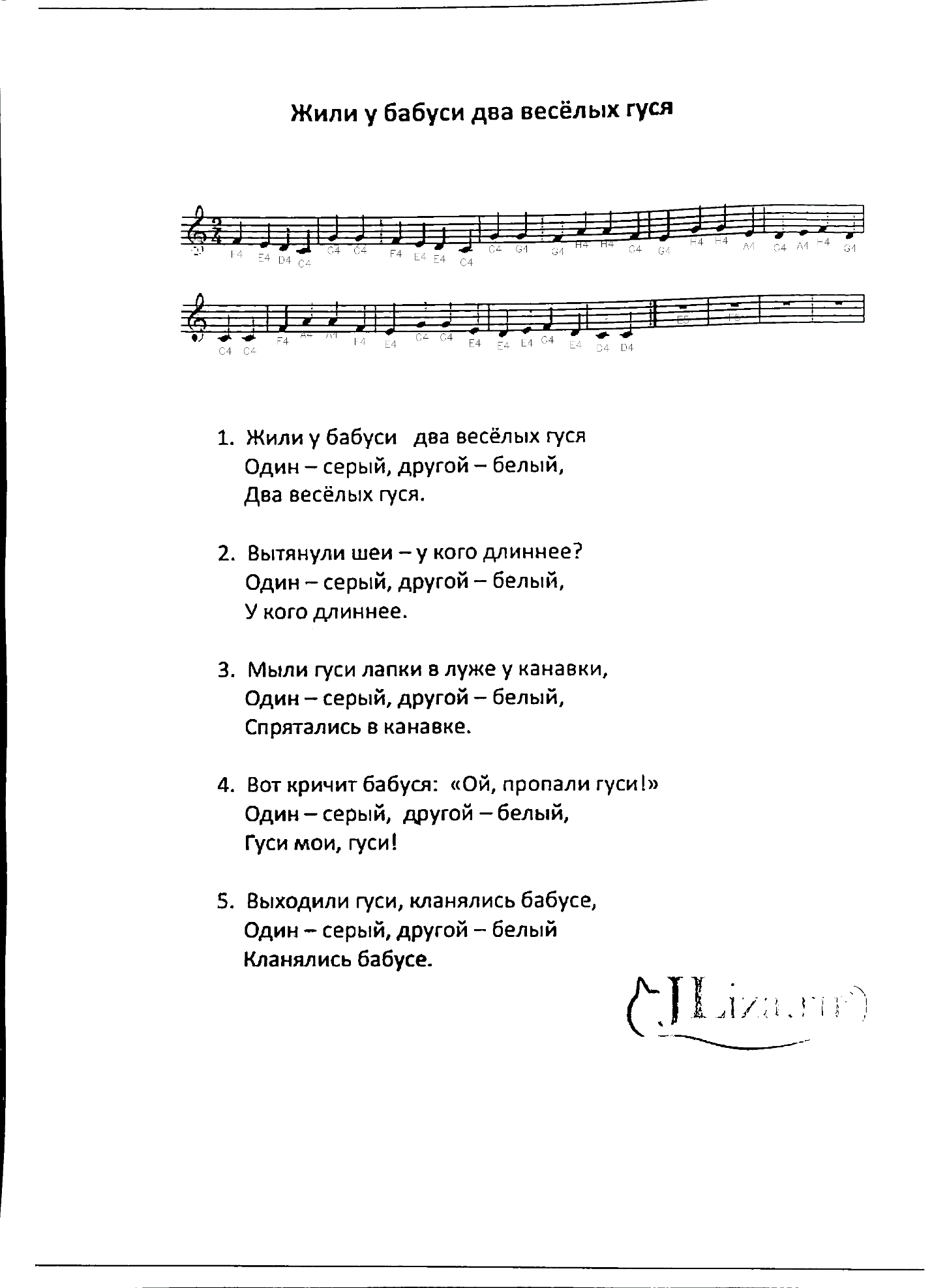


Рисунок 3.15 – Результат работы программы, каждая нота подписана

### **Проверка корректности работы программы.**

Для проверки правильности определения высоты каждой ноты воспользуемся тем же методом, что и в пункте 11.2.2.

Возьмем из пункта 11.2.2 список нот, полученный из программы SmartScore X2 Songbook - [F-5, E-5, D-5, C-5, G-5, G-5, F-5, E-5, D-5, C-5, G-5, G-5, F-5, A-5, A-5, F-5, E-5, G-5, G-5, E-5, D-5, E-5, F-5, D-5, C-5, C-5, F-5, A-5, A-5, F-5, E-5, G-5, G-5, E-5, D-5, E-5, F-5, D-5, C-5, C-5].

Список нот, распознанных с помощью программы. На рисунке 3.16 представлен вывод списка нот в консоль.

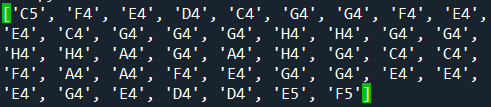


Рисунок 3.16 – Список распознанных нот на нотном листе

Перепишем список в удобный формат - ['C5', 'F4', 'E4', 'D4', 'C4', 'G4', 'G4', 'F4', 'E4', 'E4', 'C4', 'G4', 'G4', 'G4', 'H4', 'H4', 'G4', 'G4', 'H4', 'H4', 'A4', 'G4', 'A4', 'H4', 'G4', 'C4', 'C4', 'F4', 'A4', 'A4', 'F4', 'E4', 'G4', 'G4', 'E4', 'E4', 'E4', 'G4', 'E4', 'D4', 'D4', 'E5', 'F5'].

Сразу можно заметить, что в распознавании допущены ошибки. Первая найденная нота C5 является найденным блобом, который находится на первом нотном стане, но не является нотой, а является частью срипичного ключа. На рисунке 3.17 слева выделен желтым найденный блоб, который является частью скрипичного ключа, а справа желтым выделена подписанная нота, вычисленная из позиции блоба.



Рисунок 3.17 – Ошибочно определенная нота

Также, из-за того, что линии нотных станов не прямые, были допущены ошибки в распознавании некоторых нот. На рисунке 3.18 выделены желтым ошибочно определенные ноты.

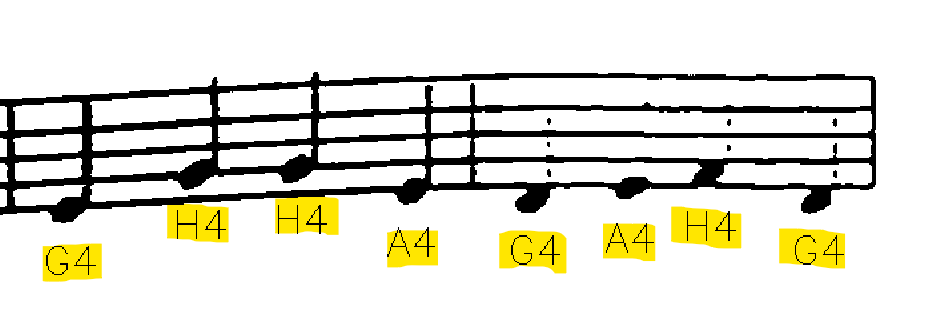


Рисунок 3.18 – Неверно определенная высота нот из-за искривления изображения

Высота нот была определена выше на 2 ступени в первом такте и на 3 ступени во втором такте. Верные значения - 'E4', 'G4', 'G4', 'E4', 'D4', 'E4', 'F4', 'D4'. Если бы лист бумаги на фото был ровным, без вмятин, то алгоритм определил бы верные высоты каждой ноты.

На рисунке также можно увидеть верно определенные высоты нот, из-за того, что в этой части лист был не деформирован. На рисунке 3.19 зеленым выделены правильно определенные ноты.

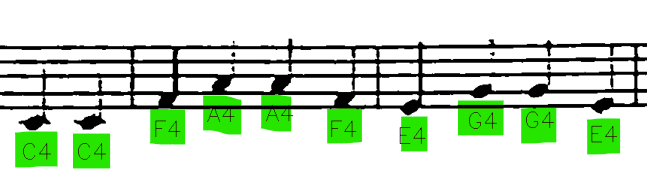


Рисунок 3.19 – Верно определенная высота нот на ровном участке изображения

## **3.2. Подведение итогов тестирования.**

При проведении тестирования были выявлены неточности в работе алгоритма распознавания нот. Эти неточности могут быть обусловлены следующими факторами:

1. **Качество входных данных:** Неточности могут быть связаны с качеством входных изображений нотных листов. Если изображения имеют низкое разрешение, недостаточную яркость и контрастность, это может затруднить правильное распознавание нот. Рекомендуется использовать изображения высокого качества с хорошим освещением и четкими деталями.
2. **Разнообразие нотных символов:** Нотные листы могут содержать различные типы нот, аккорды и другие символы. Если алгоритм не был разработан для распознавания всех возможных символов, это может привести к неточностям. В таком случае, необходимо расширить функциональность алгоритма, чтобы он справлялся с большим разнообразием нотных символов.
3. **Сложные условия изображения:** Некоторые нотные листы могут иметь сложные условия, такие как перекрытие нотных символов, неоднородное освещение или искажения. Эти факторы могут затруднить правильное распознавание. Рекомендуется использоваться методы предварительное обработки, такие как фильтры или сегментация, чтобы улучшить качество изображения и устранить некоторые искажения.
4. **Оптимизация параметров алгоритма:** Некоторые неточности могут быть связаны с неоптимальными параметрами алгоритма распознавания. Необходимо провести анализ параметров и оптимизировать их для достижения наилучших результатов. Можно экспериментировать с различными значениями параметров и оценивать их влияние на точность распознавания.

Для устранения наблюдаемых неточностей, в будущем, будет проведен более детальный анализ проблемных случаев, и расширен функционал алгоритма. Также будут использованы методы машинного и глубокого обучения для повышения точности распознавания высот нот.

В будущем будут разработаны алгоритмы для распознавания музыкального ключа, музыкального размера, тональности, знаков альтерации, длительности каждой ноты и различных символов музыкальной нотации.

# ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы был разработан и реализован алгоритм распознавания нот на нотном листе. Целью работы было создание программного инструмента, способного автоматически определять и классифицировать ноты на изображении нотного листа.

В ходе исследования были рассмотрены основные методы и подходы компьютерного зрения, используемые для распознавания объектов на изображениях. Была проведена аналитическая работы, основанная на изучении существующих методов и алгоритмов. Также были протестированы существующие программы, способные распознавать ноты. На основе этой информации был разработан алгоритм распознавания нот и реализован в виде программного обеспечения на языке Python с использованием библиотек OpenCV и NumPy.

Для тестирования алгоритма были подготовлены тестовые наборы данных, состоящие из различных фото нотных листов. Алгоритм был применен к этим наборам данных, и результаты были оценены.

В ходе тестирования были выявлены недочеты и неточности в распознавании нот, были выявлены причины, которые влияют на неточную работу программы.

В целом, результаты данной работы являются основой для дальнейшего улучшения алгоритма распознавания нот на нотном листе. Для дальнейших исследований и разработки рекомендуется уделить внимание следующим аспектам: расширение функциональности алгоритма для обработки большего разнообразия нотных символов, применение дополнительных методов предварительной обработки изображений и оптимизация параметров алгоритма. Это позволит достичь более точного и надежного распознавания нот на нотных листах.

В итоге, разработанный алгоритм представляет собой важный шаг в автоматизации процесса распознавания нот на нотных листах и может быть полезен в музыкальных приложениях, обучении музыке и других областях, где требуется точное распознавание музыкальных символов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Лутц М. Изучаем Python, том 1, 5-е изд.: Пер. с англ. — СПб.: ООО «Диалектика», 2019. — 832 с.: ил. — Парал, тит. англ.
2. Рейнхард Клетте. Компьютерное зрение. Теория и алгоритмы / пер. с англ. А. А. Слинкин. – М.: ДМК Пресс, 2019. – 506 с.: ил.
3. <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>
4. <https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html>
5. <https://www.geeksforgeeks.org/perspective-transformation-python-opencv/>
6. <https://habr.com/ru/articles/448618/>
7. <https://learnopencv.com/blob-detection-using-opencv-python-c/>
8. <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>
9. <https://numpy.org/doc/stable/reference/generated/numpy.digitize.html>
10. <https://readthedocs.org/projects/midiutil/downloads/pdf/latest/>

# ПРИЛОЖЕНИЕ А

import cv2

from adjustingimage import check

from get\_lines import get\_staffs

import os.path as pth

from blob\_detector import detect\_blobs

from note import extract\_notes, draw\_notes\_pitch,notes\_to\_midi

from notes\_to\_midi import mk\_track

file\_path = 'input/21.jpg'

def main():

if not pth.exists(file\_path):#проверка существования файла

return 'Исходный файл не существует'

else: image = cv2.imread(file\_path)#читаем изображение

processed\_photo = check(image)#подготовка изображения для дальнейшей работы

staffs = get\_staffs(processed\_photo)#нахождение и отрисовка нотных станов на изображении

blobs = detect\_blobs(processed\_photo, staffs)#поиск блобов на изображении по заданным параметрам

notes = extract\_notes(blobs, staffs)#поиск и классификация нот на нотных станах

draw\_notes\_pitch(processed\_photo, notes)#отрисовка нот

nlist = notes\_to\_midi(notes)#получение списка нот, обнаруженных на изображении

mk\_track(nlist)

print(nlist)

if \_\_name\_\_ == '\_\_main\_\_':

main()

# ПРИЛОЖЕНИЕ Б

from config import distance

import cv2

import numpy as np

import sys

def check(image):

img = image.copy()

# Конвертация в оттенки серого

im\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

# Применение пороговой бинаризации

ret, threshold = cv2.threshold(im\_gray, 100, 255, cv2.THRESH\_BINARY)

# Нахождение контуров на бинаризованном изображении

contours, \_ = cv2.findContours(threshold, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

# Рассчитываем площадь изображения и площадь наибольшего контура

image\_area = img.shape[0] \* img.shape[1]

largest\_contour\_area = max([cv2.contourArea(cnt) for cnt in contours])

# Если площадь наибольшего контура меньше 90% площади изображения, значит на изображении есть фон

if largest\_contour\_area < 0.9 \* image\_area:

print("Изображение содержит фон")

result = adjust\_image(image)

else:

print("На изображении только объекты без фона")

result = adjust\_scan(image)

return result

def adjust\_image(image):

#преобразования изображения rgb в оттенки серого

gray = cv2.cvtColor(image.copy(), cv2.COLOR\_BGR2GRAY)#0.114 \* blue\_pixel + 0.299 \* red\_pixel + 0.587 \* green\_pixel

cv2.imwrite("output/gray.jpg", gray)#сохранение изображения в оттенках серого

blur = cv2.GaussianBlur(gray, (7,7), 0)#размытие с помощью фильтра гаусса

cv2.imwrite('output/blur.jpg',blur)#сохранение размытого изображения

edged = cv2.Canny(blur, 0, 50)#обнаружение краев, представление изображения массивом ч/б пикселей

cv2.imwrite('output/canny.jpg', edged)

kernel = cv2.getStructuringElement(cv2.MORPH\_RECT, (7, 7))

closed = cv2.morphologyEx(edged, cv2.MORPH\_CLOSE, kernel)

cv2.imwrite('output/closed.jpg', closed)

contours, \_ = cv2.findContours(closed, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_NONE) # поиск контуров

contours = sorted(contours, key=cv2.contourArea, reverse=True)#сортировка по площади контура

for cnt in contours:

# Douglas Pecker algorithm - для упрощения контуров изображений

# поиск на изображении контура, который имеет четыре угл

epsilon = cv2.arcLength(cnt, True)

approx = cv2.approxPolyDP(cnt, 0.01 \* epsilon, True)

if len(approx) == 4:

sheet = approx

break

if 'sheet' not in locals():

print("невозможно найти лист на фото")

sys.exit()

approx = np.asarray([x[0] for x in sheet.astype(dtype=np.float32)])#массив из 4х углов изображения

# top\_left has the smallest sum, bottom\_right has the biggest

top\_left = min(approx, key=lambda t: t[0] + t[1])

bottom\_right = max(approx, key=lambda t: t[0] + t[1])

top\_right = max(approx, key=lambda t: t[0] - t[1])

bottom\_left = min(approx, key=lambda t: t[0] - t[1])

max\_width = int(max(distance(bottom\_right, bottom\_left), distance(top\_right, top\_left)))#максимальная ширина изображения

max\_height = int(max(distance(top\_right, bottom\_right), distance(top\_left, bottom\_left)))#максимальная высота изображения

arr = np.array([#массив, состоящий из координат исходного изображения, 4 координаты которого - вершины прямоугольника

[0, 0],

[max\_width - 1, 0],

[max\_width - 1, max\_height - 1],

[0, max\_height - 1]], dtype="float32")

rectangle = np.asarray([top\_left, top\_right, bottom\_right, bottom\_left])#координаты прямоугольника(листа)

#преобразование перспективы, фотография становится сканом листа

m = cv2.getPerspectiveTransform(rectangle, arr)

dst = cv2.warpPerspective(image, m, (max\_width, max\_height))

cv2.drawContours(image, contours, -1, (0, 255, 0), 2)

cv2.imwrite("output/2with\_contours.png", image)

dst = cv2.cvtColor(dst, cv2.COLOR\_BGR2GRAY)

\_, result = cv2.threshold(dst, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)

cv2.imwrite("output/3adjusted\_photo.png", result)

return result

def adjust\_scan(image):

gray = cv2.cvtColor(image.copy(), cv2.COLOR\_RGB2GRAY)

\_, result = cv2.threshold(gray, 200,255, cv2.THRESH\_BINARY)

h,w = result.shape[:2]

new\_h = h \* 3

new\_w = w \* 3

resized\_img = cv2.resize(result, (new\_w, new\_h)) # изменение размера

cv2.imwrite('output/3resized.png', resized\_img)

return resized\_img

# ПРИЛОЖЕНИЕ В

import numpy as np

def distance(point1, point2):

"""

Возвращает расстояние между двумя точками в декартовой системе координат.

"""

return np.sqrt((point1[0] - point2[0]) \*\* 2 + (point1[1] - point2[1]) \*\* 2)

# ПРИЛОЖЕНИЕ Г

import cv2

import numpy as np

from staff import Staff

import sys

def get\_staffs(image):

"""

Данная функция возвращает список нотных станов

на входе - изображение,для которого нужно найти нотные станы

на выходе - список найденных нотных станов

"""

processed\_image, thresholded = preprocess\_image(image)

all\_lines, lines\_image\_color = detect\_lines(processed\_image,thresholded, 80)

staffs = detect\_staffs(all\_lines)

draw\_staffs(lines\_image\_color, staffs)

return [Staff(staff[0], staff[1]) for staff in staffs]

def preprocess\_image(image):

#Приготовление изображения для последующего преобразования

print("Предварительная обработка изображения.")

gray = image.copy()

\_, thresholded = cv2.threshold(gray, 160, 255, cv2.THRESH\_BINARY)#всё,что выше - белый, ниже - черный

element = np.ones((3, 3))#ядро эррозии

#применение эрозии для изображения. если все пиксели в ядре равны 1, то пиксель будет 1,

#если же хотя бы 1 не будет равен 1, то текущий пиксель - 0

thresholded = cv2.erode(thresholded, element)

edges = cv2.Canny(thresholded, 10, 100, apertureSize=3)

cv2.imwrite('output/5.1edges.png', edges)

return edges, thresholded

def detect\_lines(processed\_image,image, nlines):

"""

Данная функция используется для обнаружения горизонтальных линий

на изображении и добавляет их в список.

"""

print("Распознавание линий.")

#ищем линии по заданным параметрам

#преобразование Хафа для поиска линий на изображении

hough = cv2.HoughLines(processed\_image, 1, np.pi / 150, 200)

all\_lines = set()#множество линий

width, height = image.shape#получаем высоту и ширину изображения

#делаем изображение цветным, чтобы можно было увидеть найденные линии.

lines\_image\_color = cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR)

for result\_arr in hough[:nlines]:

#линия представлена в параметрическом виде:

#r = x\*cos(theta) + y\*sin(theta),

#где r - длина перпендикуляра от начала координа до линии

#а theta - угол между r и x

rho = result\_arr[0][0]

theta = result\_arr[0][1]

a = np.cos(theta)

b = np.sin(theta)

x0 = a \* rho

y0 = b \* rho

shape\_sum = width + height

#координаты линий

x1 = int(x0 + shape\_sum \* (-b))

y1 = int(y0 + shape\_sum \* a)

x2 = int(x0 - shape\_sum \* (-b))

y2 = int(y0 - shape\_sum \* a)

start = (x1, y1)#начальная координата линии

end = (x2, y2)#конечная координата

diff = y2 - y1#разность по вертикали

if abs(diff) < 10:#если разность меньше 10, то добавляем

all\_lines.add(int((start[1] + end[1]) / 2))#средняя координата по y

cv2.line(lines\_image\_color, start, end, (0, 0, 255), 2)#отрисовка найденной линии

if len(all\_lines) < 5:

print('На изображении нет достаточного количества линий для нотного стана')

sys.exit()

cv2.imwrite("output/5lines.png", lines\_image\_color)

return all\_lines, lines\_image\_color

def detect\_staffs(all\_lines):

"""

Нахождение нотных станов на изображении

возвращает начало и конец обнаруженного нотного стана

"""

print("Поиск нотных станов")

LINES\_DISTANCE\_THRESHOLD = 50

staffs = []#список, в котором хранится начало и конец нотного стана

lines = []#список координат линий текущего стана

all\_lines = sorted(all\_lines)#сортируем линии по возрастанию

for current\_line in all\_lines:

# если список не пустой и расстояние между текущей лtинией и последней

#обнаруженной линией больше порога Lines\_distant

if lines and abs(lines[-1] - current\_line) > 50:

if len(lines) >= 5:

# Если больше 5, то начало это начало нового стана

# если меньше 5, то не нотный стан

staffs.append((lines[0], lines[-1]))#добавляем в список нотных станов 1ю и 5ю линию

lines.clear()

lines.append(current\_line)

# Process the last line

if len(lines) >= 5:#если больше 5, то нотный стан

#расстояние между предпоследней и последней меньше 50

if abs(lines[-2] - lines[-1]) <= LINES\_DISTANCE\_THRESHOLD:

staffs.append((lines[0], lines[-1]))

return staffs

def draw\_staffs(image, staffs):

"""

отрисовка станов на нотном листе

"""

# Draw the staffs

width = image.shape[0]

for staff in staffs:

cv2.line(image, (0, staff[0]), (width, staff[0]), (0, 255, 0), 4)

cv2.line(image, (0, staff[1]), (width, staff[1]), (0, 255, 0), 4)

cv2.imwrite("output/6staffs.png", image)

# ПРИЛОЖЕНИЕ Д

class Staff:

"""

для поиска нотных станов

"""

def \_\_init\_\_(self, min\_range, max\_range):

self.min\_range = min\_range

self.max\_range = max\_range

self.lines\_location, self.lines\_distance = self.get\_lines\_locations()

def get\_lines\_locations(self):

"""

вычисление приблизительных позиций строк на пятилинейном стане

:return: список примерных положений строк.

"""

lines = []

lines\_distance = int((self.max\_range - self.min\_range) / 4)

for i in range(5):

lines.append(self.min\_range + i \* lines\_distance)

return lines, lines\_distance

# ПРИЛОЖЕНИЕ Е

import cv2

import numpy as np

def detect\_blobs(input\_image, staffs):

"""

поиск блобов по заданным параметрам.

"""

print("поиск нот")

im\_with\_blobs = input\_image.copy()

im\_inv = (255 - im\_with\_blobs)#инвертируем изображение

#ядро для морф.трансформации, является квадратом с высотой = 1,шириной=шириналиста/250

kernel = cv2.getStructuringElement(ksize=(1, int(im\_inv.shape[0] / 500)), shape=cv2.MORPH\_RECT)#для фото

#kernel = cv2.getStructuringElement(ksize=(1, int(im\_inv.shape[0] / 300)), shape=cv2.MORPH\_RECT)

horizontal\_lines = cv2.morphologyEx(im\_inv, cv2.MORPH\_OPEN, kernel)

#убираем горизонтальные линии просто меняя их цвет

horizontal\_lines = (255 - horizontal\_lines)

#kernel = cv2.getStructuringElement(ksize=(int(im\_inv.shape[1] / 250), 1), shape=cv2.MORPH\_RECT)#для фото

kernel = cv2.getStructuringElement(ksize=(int(im\_inv.shape[1] / 200), 1), shape=cv2.MORPH\_RECT)

vertical\_lines = cv2.morphologyEx(255 - horizontal\_lines, cv2.MORPH\_OPEN, kernel)

vertical\_lines = (255 - vertical\_lines)

cv2.imwrite("output/8a\_lines\_horizontal\_removed.png", horizontal\_lines)

cv2.imwrite("output/8a\_lines\_vertical\_removed.png", vertical\_lines)

im\_with\_blobs = vertical\_lines

im\_with\_blobs = cv2.cvtColor(im\_with\_blobs, cv2.COLOR\_GRAY2BGR)

#настраиваем параметры blob-ов для поиска на изображении

params = cv2.SimpleBlobDetector\_Params()

params.filterByArea = True#фильтрация по площади

params.minArea = 225#минимальная площадь в пикселях

params.maxArea = 1500#максимальная площадь

params.filterByCircularity = True#фильтрация по круглости, тк нота круглая

params.minCircularity = 0.6#минимальная круглость

params.filterByConvexity = True#фильтрация по выпуклости

params.minConvexity = 0.9#минимальная выпуклость

params.filterByInertia = True

params.minInertiaRatio = 0.01

detector = cv2.SimpleBlobDetector\_create(params)#создание детектора с существующими параметрами

keypoints = detector.detect(im\_with\_blobs)#нахождение нот с помощью детектора

cv2.drawKeypoints(im\_with\_blobs, keypoints=keypoints, outImage=im\_with\_blobs, color=(0, 0, 255),

flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)#отрисовка блобов на изображении

cv2.imwrite("output/8b\_with\_blobs.jpg", im\_with\_blobs)

'''

нумерация нот

'''

staff\_diff = 3 / 5 \* (staffs[0].max\_range - staffs[0].min\_range)#разница между максимальным и минимальным значением стана \* 3/5

bins = [x for sublist in [[staff.min\_range - staff\_diff, staff.max\_range + staff\_diff] for staff in staffs] for x in

sublist]#список, содержащий диапазоны координат y для каждого нотного стана по y

keypoints\_staff = np.digitize([key.pt[1] for key in keypoints], bins)#индекс каждого блоба на основе его координаты

#np.digitize - используется для нахождения индекса ближайшего значения в заданном массиве для каждого элемента входного массива.

sorted\_notes = sorted(list(zip(keypoints, keypoints\_staff)), key = lambda tup: (tup[1], tup[0].pt[0]))

im\_with\_numbers = im\_with\_blobs.copy()

for idx, tup in enumerate(sorted\_notes):

cv2.putText(im\_with\_numbers, str(idx), (int(tup[0].pt[0]), int(tup[0].pt[1])),

fontFace=cv2.FONT\_HERSHEY\_SIMPLEX,

fontScale=1, color=(255, 0, 0))

cv2.putText(im\_with\_blobs, str(tup[1]), (int(tup[0].pt[0]), int(tup[0].pt[1])),

fontFace=cv2.FONT\_HERSHEY\_SIMPLEX,

fontScale=1, color=(255, 0, 0))

cv2.imwrite("output/8c\_with\_numbers.jpg", im\_with\_numbers)

cv2.imwrite("output/8d\_with\_staff\_numbers.jpg", im\_with\_blobs)

print("всего найденно элементов : " + str(len(keypoints)))

return sorted\_notes

# ПРИЛОЖЕНИЕ Ж

import cv2

from config import distance

NOTE\_PITCH\_DETECTION\_MIDDLE\_SNAPPING = 6

violin\_key = {#словарь, содержащий соотв позиция-нота

-6: 'E6',

-5: 'D6',

-4: 'C6',

-3: 'H5',

-2: 'A5',

-1: 'G5',

0: 'F5',

1: 'E5',

2: 'D5',

3: 'C5',

4: 'H4',

5: 'A4',

6: 'G4',

7: 'F4',

8: 'E4',

9: 'D4',

10: 'C4',

11: 'H3',

12: 'A3',

13: 'G3',

14: 'F3',

}

def extract\_notes(blobs, staffs):

#принимает блобы, информацию о нотных станах и изображение

#возвращает список всех нот

notes = []

print('Обнаружение нот из блобов.')

for blob in blobs:

if blob[1] % 2 == 1:

staff\_no = int((blob[1] - 1) / 2)

notes.append(Note(staff\_no, staffs, blob[0], violin\_key))

print('Обнаружено ' + str(len(notes)) + ' нот.')

return notes

def draw\_notes\_pitch(image, notes):

im\_with\_pitch = image.copy()

im\_with\_pitch = cv2.cvtColor(im\_with\_pitch, cv2.COLOR\_GRAY2BGR)

for note in notes:

cv2.putText(im\_with\_pitch, note.pitch, (int(note.center[0]) - 20, int(note.center[1]) + 50),#размещение названия нот

fontFace=cv2.FONT\_HERSHEY\_SIMPLEX,

fontScale=1, color=(0, 0, 0))

cv2.imwrite('output/9\_with\_pitch.png', im\_with\_pitch)

#для того, чтобы воспроизвести ноты, сначала их нужно добавить в массив

def notes\_to\_midi(notes):

list\_of\_notes = []

for i in notes:

list\_of\_notes.append(i.detect\_pitch(i.position\_on\_staff))

return list\_of\_notes

class Note:

"""

Конструктор принимает номер нотного стана,

список всех нотных станов, блоб с нотным символом и ключ

"""

def \_\_init\_\_(self, staff\_no, staffs, blob, clef):

self.position\_on\_staff = self.detect\_position\_on\_staff(staffs[staff\_no], blob)#позиция ноты на стане

self.staff\_no = staff\_no#номер нотного стана

self.center = blob.pt#центр нотного стана

self.pitch = self.detect\_pitch(self.position\_on\_staff)#высота ноты относительно позиции

#функция определения позиции ноты на нотном стане

def detect\_position\_on\_staff(self, staff, blob):

distances\_from\_lines = []#список, содержащий в себе расстояния от координаты x,y блоба до каждой линиии на нотном стане

x, y = blob.pt#координаты x,y каждого блоба

for line\_no, line in enumerate(staff.lines\_location):

distances\_from\_lines.append((2 \* line\_no, distance((x, y), (x, line))))

# генерация 3х дополнительных линий сверху

for line\_no in range(5, 8):

distances\_from\_lines.append((2 \* line\_no, distance((x, y), (x, staff.min\_range + line\_no \* staff.lines\_distance))))

# генерация 3х доп линиий снизу

for line\_no in range(-3, 0):

distances\_from\_lines.append((2 \* line\_no, distance((x, y), (x, staff.min\_range + line\_no \* staff.lines\_distance))))

distances\_from\_lines = sorted(distances\_from\_lines, key=lambda tup: tup[1])

# Проверьте, находится ли разница между двумя ближайшими расстояниями в пределах значения шага

if distances\_from\_lines[1][1] - distances\_from\_lines[0][1] <= NOTE\_PITCH\_DETECTION\_MIDDLE\_SNAPPING:

# размещение ноты между 2мя линиями

return int((distances\_from\_lines[0][0] + distances\_from\_lines[1][0]) / 2)

else:

# размещение ноты, являющейся ближайшей к центру блоба

return distances\_from\_lines[0][0]

def detect\_pitch(self, position\_on\_staff):

return violin\_key[position\_on\_staff]

# ПРИЛОЖЕНИЕ З

from midiutil import MIDIFile

def mk\_track(notes):

# Создание MIDI-файла

midi = MIDIFile(1) # Создание нового MIDI-файла с одним треком

track = 0 # Номер трека

channel = 0 # Номер канала

time = 1 # Время начала ноты

duration = 3 # Длительность ноты

tempo = 100 # Темп в ударах в минуту

volume = 100

# Установка темпа

midi.addTempo(track, time, tempo)

# Пример названий нот

notes\_midi = {#словарь, содержащий соотв нота-позиция

'E6':88,

'D6':86,

'C6':84,

'H5':83,

'A5':81,

'G5':79,

'F5':77,

'E5':76,

'D5':74,

'C5':72,

'H4':71,

'A4':69,

'G4':67,

'F4':65,

'E4':64,

'D4':62,

'C4':60,

'H3':59,

'A3':57,

'G3':55,

'F3':53,

}

l\_of\_midi = []

for i in notes:

if i in notes\_midi:

l\_of\_midi.append(notes\_midi[i])

# Передача нот и их высоты в MIDI-файл

for i in l\_of\_midi:

midi.addNote(track, channel, i, time, duration,volume)

time += 1

# Сохранение MIDI-файла

with open('output.mid', 'wb') as file:

midi.writeFile(file)