Теоретическая часть к лабораторной работе 4.1

1. Введение в задание

Лабораторная работа 4.1 посвящена разработке приложения на языке программирования Python с использованием библиотеки PyQt6 для создания графического интерфейса. Основная цель задания — создание программы, которая позволяет накладывать русскую озвучку на видеофайлы, независимо от исходного языка видео. Приложение должно выполнять следующие этапы обработки: загрузка видеофайла, извлечение аудиодорожки, транскрибация аудио в текст, перевод текста на русский язык, синтез новой аудиодорожки на основе переведенного текста и наложение её на исходное видео. Результатом является новый видеофайл с русской озвучкой.

Работа предполагает использование современных библиотек Python для обработки мультимедиа, распознавания речи, перевода текста и синтеза речи. Также требуется оформление графического интерфейса с кастомными стилями и соблюдение стандартов PEP 8, включая использование докстрингов и аннотаций типов (type hints).

---

2. Основные термины и понятия

2.1. Графический интерфейс (GUI)

Графический интерфейс пользователя (GUI, Graphical User Interface) — это способ взаимодействия пользователя с программой через визуальные элементы, такие как кнопки, окна, текстовые поля и т.д. В рамках данной лабораторной работы используется библиотека PyQt6 для создания GUI.

2.2. CRUD-операции

Хотя CRUD-операции (Create, Read, Update, Delete) напрямую не применяются в данном задании, их принципы могут быть использованы для работы с файлами или промежуточными данными (например, сохранение транскрибированного текста или переведённого текста в временные файлы).

2.3. Транскрибация аудио

Транскрибация (или транскрипция) аудио — процесс преобразования звуковой дорожки в текст. Для этого используются алгоритмы автоматического распознавания

речи (ASR, Automatic Speech Recognition). В данной работе предполагается использование библиотеки speech_recognition.

2.4. Синтез речи

Синтез речи (Text-to-Speech, TTS) — это процесс преобразования текстовой информации в звуковую форму, воспроизводящую речь человека. Этот процесс востребован во многих областях, включая голосовых помощников, навигационные системы, доступность для людей с ограниченными возможностями, автоматическое озвучивание текста в образовательных приложениях и многое другое.

Современные системы синтеза речи используют различные методы для обеспечения естественности звучания, управления интонацией, скоростью и тембром голоса. В данной работе рассматривается синтез речи с использованием библиотеки **gTTS (Google Text-to-Speech)**, которая позволяет конвертировать текст в аудиофайлы с минимальными затратами вычислительных ресурсов.

Существует несколько подходов к синтезу речи, различающихся по качеству звука, требуемым ресурсам и сложности реализации.

1. **Формантный синтез**
    - Основан на моделировании речевого тракта человека с использованием формантных частот.
    - Преимущества: малые вычислительные затраты, возможность полного контроля параметров речи.
    - Недостатки: низкая естественность звучания, роботизированный голос.
2. **Конкатенативный синтез**
    - Использует предварительно записанные фрагменты речи, которые комбинируются для формирования новых слов и предложений.
    - Вариации:
        - **Диплофонный метод** (использует пары фонем).
        - **Синтез на основе больших корпусов речи** (WaveNet, Tacotron).
    - Преимущества: высокая натуральность голоса.
    - Недостатки: большие объемы данных, сложность управления интонацией.
3. **Нейросетевой синтез**
    - Использует глубокие нейронные сети для прогнозирования параметров речевого сигнала (например, Tacotron, WaveNet, FastSpeech).
    - Преимущества: высокая естественность и гибкость.
    - Недостатки: большие вычислительные затраты, необходимость значительных объемов обучающих данных.
4. **Гибридные методы**
    - Комбинируют элементы различных подходов для достижения баланса между качеством и производительностью.

Библиотека gTTS (Google Text-to-Speech)

В данной работе используется **gTTS (Google Text-to-Speech)** — это облачная технология синтеза речи, разработанная Google. Она позволяет преобразовывать текст в речь с использованием предобученных моделей, обеспечивающих естественное звучание.

Основные возможности gTTS:

- Поддержка множества языков, включая русский, английский, китайский и другие.
- Различные варианты голосов и акцентов.
- Настройка скорости речи.
- Генерация аудиофайлов в формате MP3.
- Работа через API, что позволяет использовать сервис в веб-приложениях и чат-ботах.

Преимущества использования gTTS:

- **Легкость интеграции**: простая установка и удобный API.
- **Высокое качество синтеза**: звучание приближено к человеческому голосу.
- **Облачное исполнение**: не требует больших вычислительных мощностей.
- **Поддержка множества языков**: более 30 языков и диалектов.

Недостатки gTTS:

- **Требуется интернет-соединение**: так как обработка осуществляется на серверах Google.
- **Ограниченные возможности кастомизации**: нельзя изменять интонацию и тембр голоса вручную.

```
from gtts import gTTS

text = "Привет, как дела?"

tts = gTTS(text, lang="ru")
```

2.5. Обработка ошибок

Обработка ошибок — это программистская практика, которая позволяет программе корректно реагировать на исключительные ситуации, такие как отсутствие файла, проблемы с интернет-соединением или ошибки ввода-вывода. В Python для этого используются конструкции try-except.

2.6. PEP 8

PEP 8 — это стандарт стиля кода для языка Python, который определяет правила оформления кода, включая отступы, длину строк, имена переменных и т.д. Соблюдение PEP 8 делает код более читаемым и поддерживаемым.

2.7. Докстринги и аннотации типов

- **Докстринги** (docstrings) — это строковые литералы, которые используются для документирования функций, классов и модулей. Они описывают назначение, параметры и возвращаемые значения.
- **Аннотации типов** (type hints) — это синтаксические конструкции, добавленные в Python 3.5+, которые позволяют указывать типы аргументов и возвращаемых значений функций. Например, def func(arg: int) -> str:.

---

3. Используемые библиотеки и их описание

3.1. PyQt6

PyQt6 — это библиотека Python для создания кроссплатформенных графических интерфейсов, основанная на фреймворке Qt. Она предоставляет набор классов и методов для работы с окнами, виджетами, событиями и стилями.

Основные классы и методы

- **QApplication**: Класс, который управляет основным циклом событий приложения. Создаётся в начале программы для инициализации приложения.
    - Пример: app = QApplication(sys.argv)
- **QMainWindow**: Основной класс для создания главного окна приложения.
    - Методы: setWindowTitle(str), setGeometry(x, y, width, height).
- **QPushButton**: Класс для создания кнопок.
    - Методы: setText(str), clicked.connect(func) (подключение обработчика события).
- **QFileDialog**: Класс для создания диалогов выбора файлов.
    - Методы: getOpenFileName(parent, caption, dir, filter) — открывает диалог выбора файла.
- **QLabel**: Класс для отображения текста или изображений.
    - Методы: setText(str).

Кастомные стили

PyQt6 позволяет задавать кастомные стили через **Qt Style Sheets** (аналог CSS). Стили задаются в виде строк и применяются с помощью метода setStyleSheet(). Пример стиля для кнопки:

```
QPushButton {
    background-color: #4CAF50;
    color: white;
```

```
    padding: 10px;
    border-radius: 5px;
}
QPushButton:hover {
    background-color: #45a049;
}
```

Стили можно хранить в отдельном файле (например, styles.qss) и загружать в приложение.

3.2. moviepy

moviepy — это библиотека Python для работы с видео и аудио. Она позволяет извлекать аудиодорожки из видео, редактировать видео и накладывать новые аудиодорожки.

Основные классы и методы

- **VideoFileClip**: Класс для работы с видеофайлами.
    - Параметры: filename (путь к файлу).
    - Методы: audio (извлечение аудиодорожки), write_videofile(filename) (сохранение видео).
- **AudioFileClip**: Класс для работы с аудиофайлами.
    - Методы: write_audiofile(filename) (сохранение аудио).

3.3. speech_recognition

speech_recognition — библиотека Python для распознавания речи. Она поддерживает несколько API, включая Google Speech Recognition.

Основные классы и методы

- **Recognizer**: Класс для распознавания речи.
    - Методы: recognize_google(audio) (распознавание речи через Google API).
- **AudioFile**: Класс для работы с аудиофайлами.
    - Используется как контекстный менеджер: with AudioFile(filename) as source:.
- **record**: Метод для записи аудио из файла.

3.4. googletrans

googletrans — библиотека Python для перевода текста с помощью Google Translate API.

Основные классы и методы

- **Translator**: Класс для перевода текста.
    - Методы: translate(text, dest='ru') — переводит текст на указанный язык.

        - Параметры: text (текст для перевода), dest (цель перевода, например, 'ru' для русского).

3.5. gTTS

gTTS (Google Text-to-Speech) — библиотека для синтеза речи на основе текста.

Основные классы и методы

- **gTTS**: Класс для создания аудио из текста.
    - Параметры: text (текст для синтеза), lang (язык, например, 'ru').
    - Методы: save(filename) (сохранение аудиофайла).

3.6. pydub

pydub — библиотека для работы с аудиофайлами. Может использоваться для анализа или преобразования аудио (например, изменения формата).

Основные классы и методы

- **AudioSegment**: Класс для работы с аудиофайлами.
    - Методы: from_file(filename), export(filename, format).

---

4. Основные этапы работы приложения

4.1. Загрузка видеофайла

Пользователь выбирает видеофайл через диалог QFileDialog. После выбора путь к файлу сохраняется в переменную для дальнейшей обработки.

4.2. Извлечение аудио

С помощью библиотеки moviepy из видеофайла извлекается аудиодорожка. Она сохраняется во временный файл (например, temp_audio.wav).

4.3. Транскрибация аудио

Извлечённая аудиодорожка передаётся в библиотеку speech_recognition для распознавания речи. Полученный текст сохраняется для дальнейшей обработки.

4.4. Перевод текста

Транскрибированный текст переводится на русский язык с помощью библиотеки googletrans. Для этого требуется интернет-соединение.

4.5. Синтез нового аудио

Переведённый текст преобразуется в аудио с помощью gTTS. Новый аудиофайл сохраняется (например, translated_audio.mp3).

4.6. Соединение видео и нового аудио

С помощью moviepy новая аудиодорожка накладывается на исходное видео, заменяя оригинальную аудиодорожку. Итоговый видеофайл сохраняется (например, output_video.mp4).

---

5. Описание методов и параметров

5.1. Методы PyQt6

1. **QApplication(sys.argv)**:
    - Назначение: Инициализация приложения.
    - Параметры: sys.argv — список аргументов командной строки.
    - Возвращает: Экземпляр приложения.
2. **QMainWindow.setWindowTitle(str)**:
    - Назначение: Устанавливает заголовок окна.
    - Параметры: str — строка заголовка.
    - Возвращает: Ничего.
3. **QFileDialog.getOpenFileName()**:
    - Назначение: Открывает диалог выбора файла.
    - Параметры:
        - parent — родительский виджет (может быть None).
        - caption — заголовок диалога.
        - dir — начальная директория.
        - filter — фильтр файлов (например, "Video Files (*.mp4 *.avi)").
    - Возвращает: Кортеж (filename, filter) — путь к файлу и выбранный фильтр.
4. **QPushButton.clicked.connect(func)**:
    - Назначение: Подключает функцию-обработчик к событию нажатия кнопки.
    - Параметры: func — функция-обработчик.
    - Возвращает: Ничего.

5.2. Методы moviepy

1. **VideoFileClip(filename)**:
    - Назначение: Загружает видеофайл.
    - Параметры: filename — путь к файлу.
    - Возвращает: Экземпляр VideoFileClip.
2. **VideoFileClip.audio**:
    - Назначение: Извлекает аудиодорожку из видео.
    - Параметры: Нет.

    - Возвращает: Экземпляр AudioFileClip.
3. **AudioFileClip.write_audiofile(filename)**:
    - Назначение: Сохраняет аудиофайл.
    - Параметры: filename — путь для сохранения.
    - Возвращает: Ничего.

5.3. Методы speech_recognition

1. **Recognizer.recognize_google(audio)**:
    - Назначение: Распознаёт речь через Google API.
    - Параметры: audio — объект AudioData.
    - Возвращает: Строку с распознанным текстом.
    - Исключения: Требует интернет-соединение; может выбросить UnknownValueError или RequestError.

5.4. Методы googletrans

1. **Translator.translate(text, dest='ru')**:
    - Назначение: Переводит текст на указанный язык.
    - Параметры:
        - text — текст для перевода.
        - dest — целевой язык (например, 'ru').
    - Возвращает: Объект с атрибутами text (переведённый текст), src (исходный язык).

5.5. Методы gTTS

1. **gTTS(text, lang='ru')**:
    - Назначение: Создаёт объект для синтеза речи.
    - Параметры:
        - text — текст для синтеза.
        - lang — язык (например, 'ru').
    - Возвращает: Экземпляр gTTS.
2. **gTTS.save(filename)**:
    - Назначение: Сохраняет синтезированное аудио в файл.
    - Параметры: filename — путь для сохранения.
    - Возвращает: Ничего.

---

6. Обработка ошибок

Обработка ошибок является важной частью приложения. Примеры возможных ошибок и их обработка:

1. **Отсутствие файла**:

    - Если пользователь не выбрал видеофайл, программа должна вывести сообщение об ошибке.
    - Решение: Проверка пути файла перед обработкой.
2. **Проблемы с интернет-соединением**:
    - Методы recognize_google() и gTTS требуют интернет-соединения. При отсутствии сети программа должна корректно сообщить об этом.
    - Решение: Использование try-except для перехвата исключений.
3. **Некорректный формат файла**:
    - Если видеофайл повреждён или не поддерживается moviepy, программа должна обработать это исключение.
    - Решение: Использование try-except вокруг операций с moviepy.

Демонстрационный пример кода №1. Открытие аудио файла и извлечение данных их аудио файла.

```
import numpy as np
from scipy.io import wavfile
import matplotlib.pyplot as plt

# 1. Открытие аудиофайла
file_path = "input.wav"
sample_rate, audio_data = wavfile.read(file_path)

# 2. Извлечение информации
print(f"Частота дискретизации (Sample Rate): {sample_rate} Гц")
print(f"Количество отсчетов: {len(audio_data)}")
print(f"Длительность аудио: {len(audio_data) / sample_rate:.2f} секунд")

# 3. Проверка типа данных и каналов
if len(audio_data.shape) == 1:
    print("Моно-аудио (1 канал)")
    channels = 1
else:
    print(f"Стерео-аудио ({audio_data.shape[1]} канала)")
    channels = audio_data.shape[1]
    audio_data = audio_data[:, 0]  # Используем только первый канал для
простоты

# 4. Нормализация данных (но можно и не делать как бы ))
audio_data_normalized = audio_data / np.max(np.abs(audio_data))

# 5. создание временной оси
time_axis = np.linspace(0, len(audio_data) / sample_rate, num=len(audio_data))

# 6. построение графика амплитуды
plt.figure(figsize=(10, 4))
plt.plot(time_axis, audio_data_normalized, label="Амплитуда аудио")
plt.xlabel("Время (с)")
plt.ylabel("Амплитуда (нормализованная)")
plt.title("Форма волны аудиофайла")
plt.grid(True)
plt.legend()
plt.show()

# 7. показываю пример сохранение данных
# например, сохранение в новый .wav файл
output_file = "output.wav"
wavfile.write(output_file, sample_rate, audio_data)
print(f"Аудиофайл сохранен как: {output_file}")
```

Демонстрационный пример кода №2. Извлечение аудио из видео файла.

```
from moviepy.editor import VideoFileClip
import time

# путь к видеофайлу (входному) и выходному аудио файлу
video_path = "input_video.mp4"
output_audio_path = "output_audio.mp3"

# + тут еще замер времени выполнения
start_time = time.time()

# загрузка видеофайла и извлечение аудио
try:
    video = VideoFileClip(video_path)
    audio = video.audio  # тут как раз извлекаем аудиодорожку
    print(f"Длительность видео: {video.duration:.2f} секунд")

    # сохранение аудио в файл
    audio.write_audiofile(output_audio_path, codec='mp3')
    print(f"Аудио успешно извлечено и сохранено как: {output_audio_path}")

except Exception as e:
    print(f"Ошибка при обработке видеофайла: {e}")

# закрытие объектов
finally:
    if 'video' in locals():
        video.close()
    if 'audio' in locals():
        audio.close()

# вывод времени выполнения программы
end_time = time.time()
execution_time = end_time - start_time
print(f"Время выполнения извлечения аудио: {execution_time:.2f} секунд")
```

*если используемый файл не **mp3** формата а например **.wav** то строчка сохранения поменяется на:*

```
audio.write_audiofile(output_audio_path, codec='pcm_s16le')
print(f"Аудио успешно извлечено и сохранено как: {output_audio_path}")
```

Демонстрационный пример кода №3. Процесс транскрибации аудио файла.

```
import speech_recognition as sr
import time
from scipy.io import wavfile


audio_path = "input_audio.wav"

# проверка аудиофайла
sample_rate, audio_data = wavfile.read(audio_path)
print(f"Частота дискретизации аудио: {sample_rate} Гц")
print(f"Длительность аудио: {len(audio_data) / sample_rate:.2f} секунд")


start_time = time.time()

# инициализация распознавателя через Recognizer
recognizer = sr.Recognizer()

# далее идет процесс транскрибации аудио
try:
    # загрузка аудиофайла
    with sr.AudioFile(audio_path) as source:
        audio = recognizer.record(source)  # читаем весь аудиофайл

        # распознавание с помощью Google Speech API
        print("Выполняется транскрибация...")
        text = recognizer.recognize_google(audio, language="ru-RU")
        print(f"Распознанный текст: {text}")

    # сохранение текста в файл
    with open("transcription.txt", "w", encoding="utf-8") as file:
        file.write(text)
    print("Текст сохранен в файл: transcription.txt")

except sr.UnknownValueError:
    print("Ошибка: Google Speech Recognition не смог распознать аудио")
except sr.RequestError as e:
```

```
    print(f"Ошибка запроса к Google Speech API: {e}")
except Exception as e:
    print(f"Произошла ошибка: {e}")

# вывод времени выполнения
end_time = time.time()
execution_time = end_time - start_time
print(f"Время выполнения транскрибации: {execution_time:.2f} секунд")
```