ДОДАТОК А

Апробація результатів роботи

РОЗПІЗНАВАННЯ ГОЛОСУ ЗА ДОПОМОГОЮ ОФЛАЙН-БІБЛІОТЕКИ VOSK В РОБОТОТЕХНІЦІ

Бєлий Я.В, Сичова О.В.
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: yaroslav.bielyi@nure.ua

Анотація: У цій статті розглянуто використання офлайн-бібліотеки Vosk для розпізнавання мовлення. Представлено огляд можливостей бібліотеки, процес встановлення та налаштування, а також наведено приклади використання для розпізнавання голосу з аудіофайлів та в реальному часі. Бібліотека Vosk є ефективним рішенням для офлайн-розпізнавання мовлення, зокрема на пристроях з обмеженими ресурсами.

Ключові слова: розпізнавання мовлення, Vosk, офлайн-розпізнавання, голосові команди, штучний інтелект, автоматизація, Python, Kaldi, голосові моделі, нейромережі.

VOICE RECOGNITION USING THE VOSK OFFLINE LIBRARY IN ROBOTICS

Bielyi Y.V, Sychova O.V.
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky ave., 14
E-mail: yaroslav.bielyi@nure.ua

Annotation: This article discusses the use of the Vosk offline library for speech recognition. It provides an overview of the library's capabilities, the installation and configuration process, and examples of how to use it for voice recognition from audio files and in real time. The Vosk library is an effective solution for offline speech recognition, particularly on devices with limited resources.

Key words: speech recognition, Vosk, offline recognition, voice commands, artificial intelligence, automation, Python, Kaldi, voice models, neural networks.

Розпізнавання мовлення є однією з ключових технологій у сфері штучного інтелекту та автоматизації. Більшість сучасних систем розпізнавання голосу, таких як Google Speech-to-Text або Microsoft Azure Speech, потребують підключення до інтернету для передачі та обробки аудіоданих на сервері. Це може бути проблемою в умовах обмеженого доступу до мережі або коли потрібна максимальна автономність.

Vosk є відмінною альтернативою, оскільки працює повністю офлайн, що забезпечує швидку обробку та захист конфіденційності даних. На відміну від популярного CMU Sphinx, який також працює без інтернету, Vosk використовує нейромережеві моделі, що забезпечує вищу точність розпізнавання. У цій статті розглянемо можливості Vosk, процес встановлення та налаштування, а також приклади використання для розпізнавання голосу на різних пристроях.

Vosk – це потужна бібліотека для офлайн-розпізнавання мовлення, яка підтримує різні мови та працює на Windows, Linux, macOS, Android. Вона дозволяє обробляти аудіо навіть на пристроях з обмеженими ресурсами, використовуючи нейромережеві моделі для досягнення високої точності. Завдяки можливості роботи з аудіопотоком у реальному часі, Vosk підходить для розробки інтерактивних голосових систем.

Архітектура системи мобільного робота з використанням розпізнавання голосових команд має вигляд як на рисунку 1.

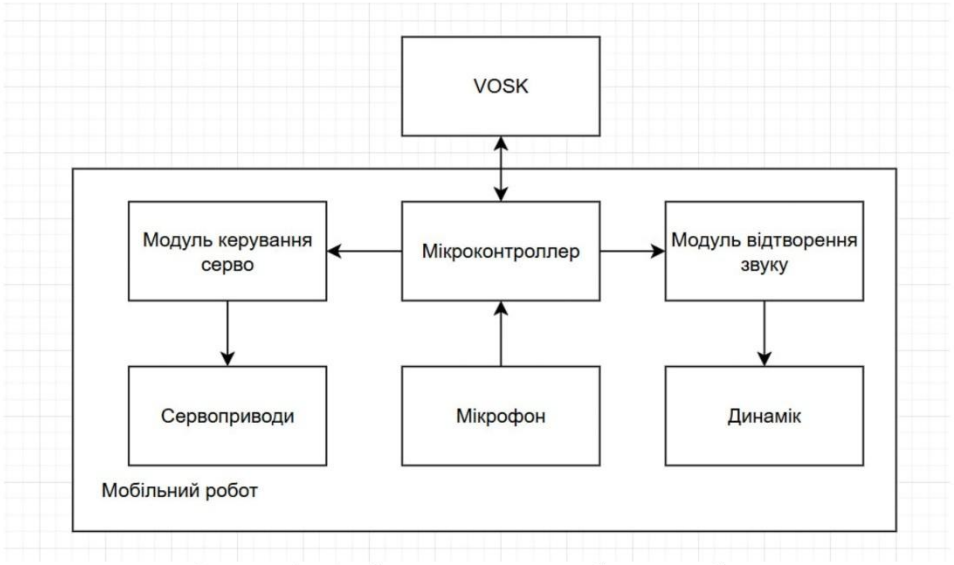

Рисунок 1 – Архітектура системи мобільного робота

Така система може виконувати прості голосові запрограмовані команди або бути частиною голосового асистента зі штучним інтелектом. Наприклад, можна навчити систему відповідати на складніші запити чи взаємодіяти з іншими пристроями.

Модуль керування серво можна замінити майже на будь-які виконавчі механізми або прибрати зовсім, створивши голосового асистента для розумного будинку. У такому випадку Vosk буде розпізнавати команди та передавати їх на інші пристрої, наприклад, для керування освітленням, температурою чи безпекою.

Для використання Vosk у Python необхідно встановити пакет vosk та завантажити відповідну мовну модель. Встановлення пакета виконується через термінал командою pip install vosk.

Мовні моделі доступні у відкритому доступі на офіційному сайті alphacephei.com. Вибір моделі залежить від необхідної мови та рівня деталізації, оскільки існують як компактні, так і розширені версії. Завантажену модель потрібно розпакувати та вказати шлях до неї під час ініціалізації розпізнавача в коді.

Vosk використовує попередньо навчені нейромережеві моделі, які базуються на Kaldi – одному з найпотужніших інструментів для обробки мовлення. Існує кілька типів моделей, які відрізняються за точністю та розміром:

– малі моделі мають компактний розмір (50–150 МБ), швидко працюють навіть на слабких пристроях, але мають нижчу точність;

– середні моделі це баланс між розміром (~400 МБ) і точністю;

– великі моделі займають більше місця (~1,5 ГБ), але забезпечують високу точність. Використовуються для додатків, де потрібне детальне розпізнавання.

Розглянемо простий приклад використання Vosk для обробки аудіофайлу.

```
from vosk import Model, KaldiRecognizer
import wave
import json
```

```
# Завантаження моделі
model = Model("vosk-model-uk-v3")

# Відкриття аудіофайлу
wf = wave.open("C:\\Users\\Kodaku\\source\\repos\\PythonApplication2\\PythonApplication2\\Recording.wav", "rb")
rec = KaldiRecognizer(model, wf.getframerate())

# Обробка аудіофайлу
while True:
    data = wf.readframes(4000)
    if len(data) == 0:
        break
    if rec.AcceptWaveform(data):
        print(json.loads(rec.Result())["text"])

# Вивід фінального результату
print(json.loads(rec.FinalResult())["text"])
```

Як результат отримаємо текст який виведемо до консолі рис.2.

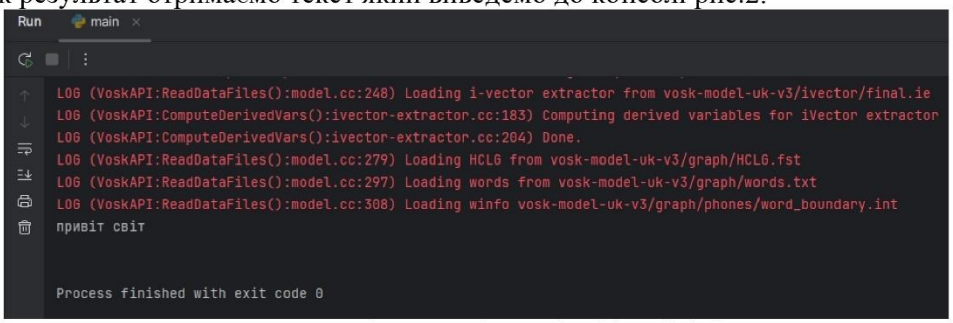

Рисунок 2 – Вміст консолі при обробці аудіофайлу

Якщо необхідно розпізнавати голосові команди у реальному часі, можна використовувати бібліотеку pyaudio та словник.

Приклад коду для розпізнавання команд в реальному часі:

```
import pyaudio
from vosk import Model, KaldiRecognizer
import json

model = Model("vosk-model-uk-v3")
rec = KaldiRecognizer(model, 16000)
pa = pyaudio.PyAudio()
stream = pa.open(format=pyaudio.paInt16, channels=1, rate=16000, input=True, frames_per_buffer=4000)
stream.start_stream()
commands = {
    "вперед": lambda: print("Робот рухається вперед"),
    "назад": lambda: print("Робот рухається назад"),
    "ліворуч": lambda: print("Робот повертає ліворуч"),
    "праворуч": lambda: print("Робот повертає праворуч"),
}

while True:
```

```
data = stream.read(1024, exception_on_overflow=False)

if rec.AcceptWaveform(data):
    result = json.loads(rec.Result())
    text = result["text"]
    print(f"Розпізнано: {text}")

    for word in text.split():
        if word in commands:
            print(f"Виконую команду: {word}")
            commands[word]()
```

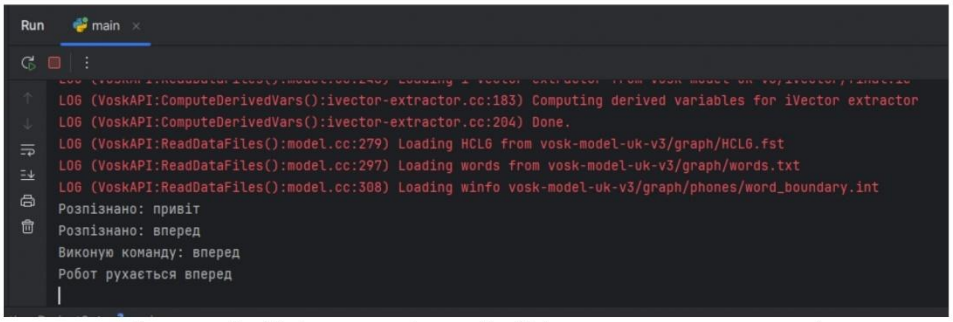

Рисунок 3 – Вміст консолі при обробці команд в реальному часі

Vosk – це ефективне рішення для офлайн-розпізнавання мовлення, особливо на пристроях з обмеженими ресурсами. Бібліотека проста у використанні, підтримує широкий спектр мов і забезпечує високу точність завдяки нейромережевим моделям. Її застосування значно підвищує інтерактивність розумних пристроїв, дозволяючи їм розпізнавати голосові команди без доступу до інтернету. Це робить Vosk хорошим вибором для вбудованих систем, робототехніки та автономних голосових асистентів.

ЛІТЕРАТУРА

1. Невлюдов, І.Ш. Інтелектуальне проектування технологічних процесів роботизованого складання [Текст]/І.Ш. Невлюдов, О.М. Цимбал, С.С. Мілютіна. – Харків: НТМТ, 2010. – 206 с.

2. Офіційний сайт alphacephei. Speech Recognition [Електронний ресурс]. –Режим доступу: https://alphacephei.com/ – 31.03.2025.

ДОДАТОК Б

Код програмного засобу

```
# файл main.py
import json
import os
from flask import Flask, Response, render_template, redirect, url_for,
request, jsonify
import cv2
import threading
import time
import subprocess
from picamera2 import Picamera2
from vad import process_audio
import servo_controller
import time
from dfplayer import dfplayer
import RPi.GPIO as GPIO
from views.voice_anim_control import voice_bp
from views.servo_control import servo_bp

CONFIG_PATH = "config/config.json"

def handle_voice_command(command):
    try:
        track = int(os.path.basename(command)[:4])  # Отримуємо номер треку з
назви
        servo_controller.play_mouth_animation_async(command)
        dfplayer.play(track)
        print(f"Відтворення треку #{track} та анімації")
    except Exception as e:
        print(f"handle_voice_command помилка: {e}")

def load_config():
    with open(CONFIG_PATH, "r") as f:
        return json.load(f)

def save_config(updated_config):
    with open(CONFIG_PATH, "w") as f:
        json.dump(updated_config, f, indent=2)

app = Flask(__name__)
app.secret_key = 'some_secret'
app.register_blueprint(voice_bp, url_prefix='/voice_anim_control')
app.register_blueprint(servo_bp)
```

```
face_detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
servo_controller.init_servos()
threading.Thread(target=process_audio, kwargs={"command_callback":
handle_voice_command}, daemon=True).start()

class Camera:
    def __init__(self, enabled=True):
        self.enabled = enabled
        if not self.enabled:
            return
        self.config = load_config()
        self.picam2 = Picamera2()
        self.picam2.configure(self.picam2.create_preview_configuration(
            main={"format": 'XRGB8888', "size":
(self.config.get("camera_width"),
                  self.config.get("camera_height"))}))
        self.picam2.start()
        self.frame = None
        self.lock = threading.Lock()
        threading.Thread(target=self.process_frames, daemon=True).start()

    def detect_faces_and_log(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
        gray = cv2.equalizeHist(gray)
        faces = face_detector.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

        if len(faces) > 0:
            largest = max(faces, key=lambda rect: rect[2] * rect[3])
            x, y, w, h = largest

            center_x = x + w // 2
            center_y = y + h // 2

            servo_controller.move_servos(center_x, center_y)

            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        return frame

    def process_frames(self):
        if not self.enabled:
            return
```

```
        while True:
            frame = self.picam2.capture_array()
            frame = self.detect_faces_and_log(frame)

            with self.lock:
                self.frame = frame
            time.sleep(self.config.get("frame_delay"))

    def get_jpeg_frame(self):
        if not self.enabled:
            return
        with self.lock:
            if self.frame is None:
                return None
            ret, jpeg = cv2.imencode('.jpg', self.frame)
            if ret:
                return jpeg.tobytes()
            else:
                return None
    def stop(self):
        try:
            self.picam2.stop()
            self.picam2.close()
        except Exception as e:
            print("Помилка зупинки picam2:", e)

cam = Camera()
camera_enabled = True

def get_cpu_temp():
    try:
        output = subprocess.check_output(['vcgencmd',
'measure_temp']).decode()
        temp_str = output.strip().split('=')[1].split("'")[0]
        return float(temp_str)
    except Exception as e:
        print(f"Помилка читання температури CPU через vcgencmd: {e}")
        return None

def generate():
    while True:
        if cam is None:
            time.sleep(0.5)
```

```
            continue
        frame = cam.get_jpeg_frame()
        if frame:
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
        else:
            time.sleep(0.5)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video')
def video():
    config = load_config()
    return render_template('video.html', config=config,
camera_enabled=camera_enabled)

@app.route('/cpu_temp')
def cpu_temp():
    temp = get_cpu_temp()
    if temp is not None:
        return jsonify({"cpu_temp": temp})
    else:
        return jsonify({"error": "Не вдалося отримати температуру"}), 500

@app.route('/video_feed')
def video_feed():
    return Response(generate(), mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/shutdown', methods=['POST'])
def shutdown_pi():
    try:
        subprocess.Popen(['sudo', 'shutdown', 'now'])
        return "Вимкнення Raspberry Pi...", 200
    except Exception as e:
        print(f"Помилка вимкнення: {e}")
        return "Помилка при вимкненні Raspberry Pi", 500

@app.route('/reboot', methods=['POST'])
def reboot_pi():
    try:
```

```
            subprocess.Popen(['sudo', 'reboot'])
            return "Перезапуск Raspberry Pi...", 200
        except Exception as e:
            print(f"Помилка перезапуску: {e}")
            return "Помилка при перезапуску Raspberry Pi", 500

@app.route('/update_config', methods=['POST'])
def update_config():
    config = load_config()
    try:
        config['camera_width'] = int(request.form.get('camera_width',
config['camera_width']))
        config['camera_height'] = int(request.form.get('camera_height',
config['camera_height']))
        config['frame_delay'] = float(request.form.get('frame_delay',
config['frame_delay']))
        save_config(config)
        stop_camera()
        start_camera()
    except Exception as e:
        print("Помилка збереження конфігу або перезапуску камери:", e)
    return redirect(url_for('video'))

@app.route("/toggle_camera")
def toggle_camera():
    if cam is None:
        start_camera()
    else:
        stop_camera()
    return redirect(url_for("video"))

def stop_camera():
    global cam
    if cam:
        try:
            cam.stop()
        except Exception as e:
            print("Помилка при вимиканні камери:", e)
        cam = None

def start_camera():
    global cam
    if cam is None:
```

```
        try:
            cam = Camera()
        except Exception as e:
            print("Помилка при запуску камери:", e)
            cam = None

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

```
# файл servo_controller.py
import time
import threading
import json
import board
import busio
from adafruit_pca9685 import PCA9685
from adafruit_motor import servo

# Глобальні змінні
servo_horizontal = None
servo_vertical = None
current_horizontal = 90
current_vertical = 90
target_horizontal = 90
target_vertical = 90

servo_head_horizontal = None
servo_head_vertical = None
current_head_horizontal = 90
current_head_vertical = 90
target_head_horizontal = 90
target_head_vertical = 90
servo_mouth = 0

running = True

# Конфігурація
config = {}

def init_servos():
    global servo_horizontal, servo_vertical, servo_head_horizontal,
servo_head_vertical, servo_mouth
    global config
    global servo_thread, monitor_thread

    # Завантаження конфігурації
    with open('config/config.json') as config_file:
        config = json.load(config_file)

    # Ініціалізація
    i2c = busio.I2C(board.SCL, board.SDA)
    pca = PCA9685(i2c)
```

```
    pca.frequency = 50

    # Ініціалізація серв
    servo_horizontal = servo.Servo(
        pca.channels[config["servo_horizontal_channel"]],
        **get_pulse_range(config, "servo_horizontal_pulse")
    )

    servo_vertical = servo.Servo(
        pca.channels[config["servo_vertical_channel"]],
        **get_pulse_range(config, "servo_vertical_pulse")
    )

    servo_head_horizontal = servo.Servo(
        pca.channels[config["head_servo_horizontal_channel"]],
        **get_pulse_range(config, "head_horizontal_pulse")
    )

    servo_head_vertical = servo.Servo(
        pca.channels[config["head_servo_vertical_channel"]],
        **get_pulse_range(config, "head_vertical_pulse")
    )

    servo_mouth = servo.Servo(
        pca.channels[config["mouth_servo_channel"]],
        **get_pulse_range(config, "mouth_pulse")
    )

    # Запускаємо новий потік
    servo_thread = threading.Thread(target=servo_loop, daemon=True)
    monitor_thread =
threading.Thread(target=monitor_eye_offset_and_adjust_head, daemon=True)
    servo_thread.start()
    monitor_thread.start()

def play_mouth_animation_async(json_path):
    threading.Thread(target=play_mouth_animation, args=(json_path,),
daemon=True).start()

def play_mouth_animation(json_path):
    try:
        with open(json_path, 'r') as f:
            anim_data = json.load(f)
```

```
            angles = anim_data["angles"]
            delay_ms = anim_data.get("frame_duration_ms", 100)

            invert = config.get("invert_mouth_animation", False)

            for angle in angles:
                if invert:
                    angle = 180 - angle
                servo_mouth.angle = angle
                time.sleep(delay_ms / 1000.0)

    except Exception as e:
        print(f"Помилка під час програвання анімації: {e}")

def get_pulse_range(config, key):
    pulse = config.get(key, [500, 2500])
    return {'min_pulse': pulse[0], 'max_pulse': pulse[1]}

def move_head(x, y):
    global target_head_horizontal, target_head_vertical

    # Обчислення кута з обмеженнями
    target_head_horizontal = max(config["head_min_horizontal_angle"],
min(config["head_max_horizontal_angle"], int(x)))
    target_head_vertical = max(config["head_min_vertical_angle"],
min(config["head_max_vertical_angle"], int(y)))

def move_servos(x, y):
    global target_horizontal, target_vertical

    # Завантаження параметрів з конфігу
    invert_horizontal = config.get("invert_horizontal")
    invert_vertical = config.get("invert_vertical")
    camera_width = config.get("camera_width")
    camera_height = config.get("camera_height")

    # Інверсія координат
    if invert_horizontal:
        x = camera_width - x
    if invert_vertical:
        y = camera_height - y

    # Конвертація в кути з урахуванням обмежень
```

```
    target_horizontal = max(
        config["horizontal_min_angle"],
        min(config["horizontal_max_angle"], int((x / camera_width) * 180))
    )
    target_vertical = max(
        config["vertical_min_angle"],
        min(config["vertical_max_angle"], int((y / camera_height) * 180))
    )

def monitor_eye_offset_and_adjust_head():
    global current_horizontal, current_vertical
    global current_head_horizontal, current_head_vertical
    global running

    # Параметри з config.json
    deviation_threshold_percent =
config.get("eye_deviation_threshold_percent")
    deviation_duration_sec = config.get("eye_deviation_duration_sec")
    head_adjustment_angle = config.get("head_adjustment_angle")

    center_angle = 90
    full_range = 180
    deviation_threshold = full_range * deviation_threshold_percent / 100

    # Таймери для горизонтального і вертикального відхилення
    horizontal_deviation_start = None
    vertical_deviation_start = None

    while running:
        eye_offset_h = current_horizontal - center_angle
        eye_offset_v = current_vertical - center_angle

        # Горизонталь
        if abs(eye_offset_h) > deviation_threshold:
            if horizontal_deviation_start is None:
                horizontal_deviation_start = time.time()
            elif time.time() - horizontal_deviation_start >=
deviation_duration_sec:
                if config.get("invert_head_horizontal"):
                    if eye_offset_h > 0:
                        move_head(current_head_horizontal -
head_adjustment_angle, current_head_vertical)
                    else:
```

```
                        move_head(current_head_horizontal +
head_adjustment_angle, current_head_vertical)
                else:
                    if eye_offset_h > 0:
                        move_head(current_head_horizontal +
head_adjustment_angle, current_head_vertical)
                    else:
                        move_head(current_head_horizontal -
head_adjustment_angle, current_head_vertical)
                horizontal_deviation_start = None
        else:
            horizontal_deviation_start = None

        # Вертикаль
        if abs(eye_offset_v) > deviation_threshold:
            if vertical_deviation_start is None:
                vertical_deviation_start = time.time()
            elif time.time() - vertical_deviation_start >=
deviation_duration_sec:
                if config.get("invert_head_vertical"):
                    if eye_offset_v > 0:
                        move_head(current_head_horizontal,
current_head_vertical - head_adjustment_angle)
                    else:
                        move_head(current_head_horizontal,
current_head_vertical + head_adjustment_angle)
                else:
                    if eye_offset_v > 0:
                        move_head(current_head_horizontal,
current_head_vertical + head_adjustment_angle)
                    else:
                        move_head(current_head_horizontal,
current_head_vertical - head_adjustment_angle)
                vertical_deviation_start = None
        else:
            vertical_deviation_start = None

        time.sleep(0.1)

def clamp(value, min_val, max_val):
    return max(min_val, min(max_val, value))

def servo_loop():
```

```
    global current_horizontal, current_vertical
    global target_horizontal, target_vertical
    global servo_horizontal, servo_vertical
    global current_head_horizontal, current_head_vertical
    global target_head_horizontal, target_head_vertical
    global servo_head_horizontal, servo_head_vertical
    global running

    Kp_horizontal = 0.5
    Kp_vertical = 0.5

    while running:
        # Горизонталь очей
        error_h = target_horizontal - current_horizontal
        delta_h = Kp_horizontal * error_h
        current_horizontal += delta_h
        current_horizontal = clamp(current_horizontal,
config["horizontal_min_angle"], config["horizontal_max_angle"])
        servo_horizontal.angle = current_horizontal

        # Вертикаль очей
        error_v = target_vertical - current_vertical
        delta_v = Kp_vertical * error_v
        current_vertical += delta_v
        current_vertical = clamp(current_vertical,
config["vertical_min_angle"], config["vertical_max_angle"])
        servo_vertical.angle = current_vertical

        # Горизонталь голови
        diff_h_head = target_head_horizontal - current_head_horizontal
        if abs(diff_h_head) >= 0.5:
            step_h_head = max(config["head_min_step"],
min(config["head_max_step"], abs(diff_h_head) / 2))
            current_head_horizontal += step_h_head if diff_h_head > 0 else -
step_h_head
            current_head_horizontal = clamp(current_head_horizontal,
config["head_min_horizontal_angle"], config["head_max_horizontal_angle"])
            servo_head_horizontal.angle = current_head_horizontal

        # Вертикаль голови
        diff_v_head = target_head_vertical - current_head_vertical
        if abs(diff_v_head) >= 0.5:
```

```
            step_v_head = max(config["head_min_step"],
min(config["head_max_step"], abs(diff_v_head) / 2))
            current_head_vertical += step_v_head if diff_v_head > 0 else -
step_v_head
            current_head_vertical = clamp(current_head_vertical,
config["head_min_vertical_angle"], config["head_max_vertical_angle"])
            servo_head_vertical.angle = current_head_vertical

        time.sleep(0.2)

def stop_servo():
    global running, servo_thread, monitor_thread
    running = False
    print("Waiting for threads to stop")

    if servo_thread and servo_thread.is_alive():
        servo_thread.join(timeout=2)

    if monitor_thread and monitor_thread.is_alive():
        monitor_thread.join(timeout=2)
    print("Servo  stopped")

def restart_servo():
    global running
    stop_servo()
    running = True
    init_servos()

def start_servo():
    global running
    running = True
    init_servos()
```

```
# файл класу dfplayer.py
import serial
import time
import json

CONFIG_PATH = "config/config.json"

def load_config():
    with open(CONFIG_PATH, "r") as f:
        return json.load(f)

class DFPlayer:
    def __init__(self, port="/dev/ttyS0", baudrate=9600):
        self.ser = serial.Serial(port, baudrate, timeout=1)
        time.sleep(2)
        self.config = load_config()
        self.set_volume(self.config.get("volume"))

    def send_command(self, cmd, param1=0, param2=0):
        command = bytearray([
            0x7E,       # Початок
            0xFF,       # Версія
            0x06,       # Довжина
            cmd,        # Команда
            0x00,       # Високий байт
            param1,     # Параметр 1
            param2,     # Параметр 2
            0x00,       # Контрольна сума
            0x00,       # Контрольна сума
            0xEF        # Кінець
        ])
        checksum = -sum(command[1:7]) & 0xFFFF
        command[7] = (checksum >> 8) & 0xFF
        command[8] = checksum & 0xFF

        self.ser.write(command)
        time.sleep(0.2)

    def play(self, track_number):
        high_byte = (track_number >> 8) & 0xFF
        low_byte = track_number & 0xFF
        self.send_command(0x03, high_byte, low_byte)
```

```
    def set_volume(self, volume):
        volume = max(0, min(30, volume))
        self.send_command(0x06, 0, volume)

    def reload_volume(self):
        """Оновити гучність із config.json"""
        self.config = load_config()
        volume = self.config.get("volume", 25)
        self.set_volume(volume)
        print(f"Гучність оновлено: {volume}")

dfplayer = DFPlayer()
```

```
# файл скрипту створення анімацій роту mp3animation.py
from pydub import AudioSegment
import numpy as np
import json
import os

def rms(frame):
    return np.sqrt(np.mean(np.square(frame.astype(np.float32))))

def audio_to_servo_animation(mp3_path, frame_duration_ms=50, servo_range=(0,
180)):
    audio =
AudioSegment.from_mp3(mp3_path).set_channels(1).set_frame_rate(16000)

    samples = np.array(audio.get_array_of_samples())
    sample_rate = audio.frame_rate
    frame_size = int(sample_rate * (frame_duration_ms / 1000))

    angles = []
    max_rms_seen = 0.0

    for i in range(0, len(samples), frame_size):
        frame = samples[i:i+frame_size]
        if len(frame) == 0:
            break
        value = rms(frame)
        max_rms_seen = max(max_rms_seen, value)
        angles.append(value)

    max_rms = max_rms_seen if max_rms_seen > 0 else 1
    angles_scaled = [int(np.interp(a, [0, max_rms], servo_range)) for a in
angles]

    animation_data = {
        "frame_duration_ms": frame_duration_ms,
        "angles": angles_scaled
    }

    json_path = os.path.splitext(mp3_path)[0] + "_mouth.json"
    with open(json_path, "w") as f:
        json.dump(animation_data, f, indent=2)

    print(f"Збережено: {json_path}")
```

```
    return animation_data


if __name__ == "__main__":
    target_folder = os.path.join(os.getcwd(), "Mouth_anim")

    if not os.path.exists(target_folder):
        print("Папка 'Mouth_anim' не знайдена.")
    else:
        mp3_files = [f for f in os.listdir(target_folder) if
f.lower().endswith('.mp3')]
        if not mp3_files:
            print("MP3-файли в 'Mouth_anim' не знайдено.")
        else:
            for mp3 in mp3_files:
                mp3_path = os.path.join(target_folder, mp3)
                json_path = os.path.splitext(mp3_path)[0] + "_mouth.json"
                if not os.path.exists(json_path):
                    try:
                        audio_to_servo_animation(mp3_path)
                    except Exception as e:
                        print(f"Помилка обробки {mp3}: {e}")
                else:
                    print(f"Пропущено (вже існує): {json_path}")
```

```
# файл визначення мовлення в аудіопотоці vad.py
import queue
import sounddevice as sd
import numpy as np
import webrtcvad
import collections
import RPi.GPIO as GPIO
import json
import time
import struct
import os
from vosk_stt import recognize_vosk

os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
"/home/Zero2W/my_project/robot-promoter-stt-d0ffd0b23517.json"
from google_stt import recognize_google

# GPIO для BUSY плеєру
BUSY_PIN = 4
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUSY_PIN, GPIO.IN)

# Конфігурація
SAMPLE_RATE = 16000
FRAME_DURATION_MS = 30  # мс (10, 20 або 30)
FRAME_SIZE = int(SAMPLE_RATE * FRAME_DURATION_MS / 1000)
GAIN = 4.0
VAD_MODE = 3  # 0-3: агресивність детекції
MIN_VOICE_LEN = SAMPLE_RATE // 2
MAX_SILENCE_FRAMES = int(0.5 * 1000 / FRAME_DURATION_MS)  # 0.5 сек
BUSY_DELAY = 2.0
running = True
google_stt = False

# Команди
def load_commands(path="commands/commands.json"):
    try:
        with open(path, 'r', encoding='utf-8') as f:
            return json.load(f)
    except Exception as e:
        print(f"Помилка завантаження команд: {e}")
        return {}
```

```
commands = load_commands()

def check_command(text: str):
    for keyword, action in commands.items():
        if keyword in text:
            return keyword, action
    return None, None

q = queue.Queue()
buffer = []
pre_voice_buffer = collections.deque(maxlen=int(0.5 * SAMPLE_RATE))
recording = False
silence_frames = 0

# WebRTC VAD
vad = webrtcvad.Vad()
vad.set_mode(VAD_MODE)

def float_to_pcm16(audio):
    return struct.pack("%dh" % len(audio), *(np.int16(audio * 32767)))

# Callback
def audio_callback(indata, frames, time, status):
    if status:
        print("Помилка потоку:", status)
    q.put(indata.copy())

def recognize(audio_data, sample_rate=16000):
    if google_stt:
        print(f"Розпізнавання за допомогою Google STT")
        return recognize_google(audio_data, sample_rate)
    else:
        print(f"Розпізнавання за допомогою Vosk STT")
        return recognize_vosk(audio_data, sample_rate)

def enable_google_stt():
    global google_stt
    google_stt = True

def enable_vosk_stt():
    global google_stt
    google_stt = False
```

```
# Обробка аудіо
def process_audio(command_callback=None):
    global recording, silence_frames, buffer, running

    prev_busy_state = GPIO.input(BUSY_PIN)
    busy_block_until = 0

    with sd.InputStream(samplerate=SAMPLE_RATE, channels=1, dtype='float32',
                        blocksize=FRAME_SIZE, callback=audio_callback,
latency='low'):

        while running:
            current_time = time.time()
            busy_state = GPIO.input(BUSY_PIN)

            # Виявлення зміни стану плеєра
            if busy_state != prev_busy_state:
                if busy_state == GPIO.LOW:
                    print("Плеєр зайнятий. Затримка розпізнавання")
                    busy_block_until = current_time + BUSY_DELAY
                else:
                    print("Плеєр вільний")
                    pre_voice_buffer.clear()
                    buffer.clear()
                    while not q.empty():
                        q.get()
                prev_busy_state = busy_state

            if q.empty():
                continue

            audio = q.get().flatten()
            pre_voice_buffer.extend(audio)

            # Підсилення + перевірка голосу
            audio = np.clip(audio * GAIN, -1.0, 1.0)
            pcm_data = float_to_pcm16(audio)
            is_speech = vad.is_speech(pcm_data, SAMPLE_RATE)

            if is_speech:
                if not recording:
                    recording = True
                    silence_frames = 0
```

```
                    print("Початок запису голосу")
                    if current_time > busy_block_until:
                        buffer = list(pre_voice_buffer)
                    else:
                        print("Затримка активна — пропускаємо
pre_voice_buffer")
                        buffer = []
                buffer.extend(audio)

            elif recording:
                buffer.extend(audio)
                silence_frames += 1

                if silence_frames > MAX_SILENCE_FRAMES:
                    print("Кінець запису голосу")
                    data = np.array(buffer)
                    recording = False
                    silence_frames = 0
                    buffer = []

                    # Перевірка на мінімальну довжину
                    if len(data) < MIN_VOICE_LEN:
                        print("Занадто короткий фрагмент — ігнорується")
                        continue

                    # Затримка після BUSY
                    if current_time < busy_block_until:
                        print(f"Очікування роботи плеєра")
                        continue

                    print(f"Відправляю фрагмент ({len(data)/SAMPLE_RATE:.2f}
сек) на розпізнавання")
                    text = recognize(data, SAMPLE_RATE)

                    if text:
                        print("Розпізнано:", text)
                        keyword, action = check_command(text)
                        if action:
                            print(f"Команда: '{keyword}' {action}")
                            if command_callback:
                                command_callback(action)
                    else:
                        print("Нічого не розпізнано")
```

```
# файл модуля розпізнавання мовлення vosk_stt.py
import vosk
import json
import numpy as np
import wave
import io

model = vosk.Model("vosk-model-small-uk-v3-nano")

def recognize_vosk(audio_data: np.ndarray, sample_rate: int = 16000) -> str:
    # Перетворення float [-1.0, 1.0] в int16 [-32768, 32767]
    int16_data = (audio_data * 32767).astype(np.int16)

    with io.BytesIO() as wav_io:
        with wave.open(wav_io, "wb") as wav_file:
            wav_file.setnchannels(1)
            wav_file.setsampwidth(2)
            wav_file.setframerate(sample_rate)
            wav_file.writeframes(int16_data.tobytes())

        wav_io.seek(0)

        # Створюємо розпізнавач
        rec = vosk.KaldiRecognizer(model, sample_rate)

        while True:
            data = wav_io.read(4000)
            if len(data) == 0:
                break
            rec.AcceptWaveform(data)

        result = json.loads(rec.FinalResult())
        return result.get("text", "")

# файл модуля розпізнавання мовлення google_stt.py
import io
import wave
import numpy as np
from google.cloud import speech

# Ініціалізація клієнта
client = speech.SpeechClient()
```

```
def recognize_google(audio_data: np.ndarray, sample_rate: int = 16000,
language: str = "uk-UA") -> str:
    # Перетворення float [-1.0, 1.0] в int16 [-32768, 32767]
    int16_data = (audio_data * 32767).astype(np.int16)

    # Створюємо WAV у пам’яті
    with io.BytesIO() as wav_io:
        with wave.open(wav_io, "wb") as wav_file:
            wav_file.setnchannels(1)
            wav_file.setsampwidth(2)  # 2 байти = 16-біт
            wav_file.setframerate(sample_rate)
            wav_file.writeframes(int16_data.tobytes())

        content = wav_io.getvalue()

    # Формуємо запит до Google STT
    audio = speech.RecognitionAudio(content=content)
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=sample_rate,
        language_code=language,
    )

    # Відправляємо на розпізнавання
    response = client.recognize(config=config, audio=audio)

    if response.results:
        return response.results[0].alternatives[0].transcript
    return ""
```

```
# блюпрінт для серверу Flask сторінки контролю сервоприводами
from flask import Blueprint, render_template, request, redirect, url_for,
flash, jsonify
import servo_controller
import json
import os

servo_bp = Blueprint('servo_control', __name__,
template_folder='../templates')
CONFIG_PATH = 'config/config.json'

def load_config():
    with open(CONFIG_PATH, 'r') as f:
        return json.load(f)

def save_config(data):
    with open(CONFIG_PATH, 'w') as f:
        json.dump(data, f, indent=2)

@servo_bp.route('/servo-control')
def servo_control_page():
    config = load_config()
    return render_template('servo_control.html', config=config)

@servo_bp.route('/update-servo-angle', methods=['POST'])
def update_servo_angle():
    from servo_controller import servo_horizontal, servo_vertical,
servo_head_horizontal, servo_head_vertical, servo_mouth
    data = request.json
    angle = int(data.get('angle'))
    servo_id = data.get('servo')

    try:
        if servo_id == 'horizontal':
            servo_horizontal.angle = angle
        elif servo_id == 'vertical':
            servo_vertical.angle = angle
        elif servo_id == 'head_horizontal':
            servo_head_horizontal.angle = angle
        elif servo_id == 'head_vertical':
            servo_head_vertical.angle = angle
        elif servo_id == 'mouth':
            servo_mouth.angle = angle
```

```
        return jsonify({'status': 'ok'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@servo_bp.route('/update-config', methods=['POST'])
def update_config():
    new_config = request.form.to_dict(flat=True)
    config = load_config()
    for key in new_config:
        try:
            val = json.loads(new_config[key])
        except:
            val = new_config[key]
        config[key] = val
    save_config(config)
    servo_controller.restart_servo()
    flash('Конфігурацію оновлено')
    return redirect(url_for('servo_control.servo_control_page'))

@servo_bp.route('/servo-stop', methods=['POST'])
def servo_stop():
    servo_controller.stop_servo()
    flash('Автоматичний контроль сервоприводами вимкнено')
    return redirect(url_for('servo_control.servo_control_page'))

@servo_bp.route('/start-servo', methods=['POST'])
def start_servo():
    servo_controller.start_servo()
    flash('Автоматичний контроль сервоприводами увімкнено')
    return redirect(url_for('servo_control.servo_control_page'))

# блюпрінт для серверу Flask сторінки налаштування голосових команд
from flask import Blueprint, render_template, request, redirect, url_for,
flash
from dfplayer import dfplayer
from vad import enable_google_stt, enable_vosk_stt
import os
import subprocess
import json

voice_bp = Blueprint('voice_anim_control', __name__,
template_folder='../templates')
```

```
UPLOAD_FOLDER = 'Mouth_anim'
COMMANDS_FILE = 'commands/commands.json'
ALLOWED_EXTENSIONS = {'mp3'}
CONFIG_PATH = "config/config.json"

os.makedirs(UPLOAD_FOLDER, exist_ok=True)
os.makedirs(os.path.dirname(COMMANDS_FILE), exist_ok=True)
if not os.path.exists(COMMANDS_FILE):
    with open(COMMANDS_FILE, 'w') as f:
        json.dump({}, f)

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

def list_files(folder, ext):
    return sorted([f for f in os.listdir(folder) if f.lower().endswith(ext)])

def load_config():
    with open(CONFIG_PATH) as f:
        return json.load(f)

def save_config(data):
    with open(CONFIG_PATH, "w") as f:
        json.dump(data, f, indent=2)

def load_commands():
    try:
        with open(COMMANDS_FILE, 'r') as f:
            return json.load(f)
    except Exception as e:
        print(f"Помилка завантаження команд: {e}")
        return {}

def save_commands(commands):
    with open(COMMANDS_FILE, 'w') as f:
        json.dump(commands, f, indent=2, ensure_ascii=False)

@voice_bp.route('/', methods=['GET', 'POST'])
def index():
    config = load_config()
    if request.method == 'POST':
        file = request.files.get('file')
```

```
        if not file or file.filename == '':
            flash('Файл не вибрано')
        elif allowed_file(file.filename):
            file.save(os.path.join(UPLOAD_FOLDER, file.filename))
            flash(f'Файл {file.filename} збережено успішно!')
        else:
            flash('Дозволені тільки MP3-файли')
        return redirect(request.url)

    return render_template(
        'voice_anim_control.html',
        mp3_files=list_files(UPLOAD_FOLDER, '.mp3'),
        json_files=list_files(UPLOAD_FOLDER, '.json'),
        commands=load_commands(),
        result="",
        current_volume=config.get("volume")
    )

@voice_bp.route('/add_command', methods=['POST'])
def add_command():
    command = request.form.get('command')
    json_file = request.form.get('json_file')
    if not command or not json_file:
        flash('Потрібно вказати команду та файл.')
    else:
        commands = load_commands()
        commands[command] = os.path.join(UPLOAD_FOLDER, json_file)
        save_commands(commands)
        flash(f'Команду "{command}" додано.')
    return redirect(url_for('voice_anim_control.index'))

@voice_bp.route('/delete_command', methods=['POST'])
def delete_command():
    command_name = request.form.get('command_name')
    commands = load_commands()
    if command_name in commands:
        del commands[command_name]
        save_commands(commands)
        flash(f'Команду "{command_name}" видалено.')
    else:
        flash(f'Команда "{command_name}" не знайдена.')
    return redirect(url_for('voice_anim_control.index'))
```

```
@voice_bp.route('/edit_command', methods=['POST'])
def edit_command():
    command_name = request.form.get('command_name')
    new_json_file = request.form.get('new_json_file')
    commands = load_commands()
    if command_name in commands:
        commands[command_name] = os.path.join(UPLOAD_FOLDER, new_json_file)
        save_commands(commands)
        flash(f'Команду "{command_name}" оновлено.')
    else:
        flash(f'Команда "{command_name}" не знайдена.')
    return redirect(url_for('voice_anim_control.index'))

@voice_bp.route('/run_script', methods=['POST'])
def run_script():
    try:
        result = subprocess.run(['python3', 'mp3animation.py'],
capture_output=True, text=True)
        output = result.stdout if result.returncode == 0 else f"Помилка:
{result.stderr}"
    except Exception as e:
        output = f"Сталася помилка: {e}"

    return render_template(
        'voice_anim_control.html',
        mp3_files=list_files(UPLOAD_FOLDER, '.mp3'),
        json_files=list_files(UPLOAD_FOLDER, '.json'),
        commands=load_commands(),
        result=output
    )
@voice_bp.route('/delete_file', methods=['POST'])
def delete_file():
    filename = request.form.get('filename')
    if not filename:
        flash('Файл не вказано.')
        return redirect(url_for('voice_anim_control.index'))

    file_path = os.path.join(UPLOAD_FOLDER, filename)
    if os.path.exists(file_path):
        try:
            os.remove(file_path)
            flash(f'Файл "{filename}" видалено.')
        except Exception as e:
```

```
            flash(f'Помилка при видаленні файлу: {e}')
    else:
        flash(f'Файл "{filename}" не знайдено.')

    return redirect(url_for('voice_anim_control.index'))

@voice_bp.route("/set_volume", methods=["POST"])
def set_volume():
    try:
        config = load_config()
        volume = int(request.form.get("volume"))
        volume = max(0, min(30, volume))
        config["volume"] = volume
        save_config(config)
        dfplayer.reload_volume()
        flash(f"Гучність оновлено до {volume} ")
    except Exception as e:
        flash(f"Помилка встановлення гучності: {e}")
    return redirect(url_for("voice_anim_control.index"))

@voice_bp.route('/enable_google_stt', methods=['POST'])
def enable_google_stt_route():
    try:
        enable_google_stt()
        flash("Google STT увімкнено.")
    except Exception as e:
        flash(f"Помилка увімкнення Google STT: {e}")
    return redirect(url_for('voice_anim_control.index'))

@voice_bp.route('/enable_vosk_stt', methods=['POST'])
def enable_vosk_stt_route():
    try:
        enable_vosk_stt()
        flash("Vosk STT увімкнено.")
    except Exception as e:
        flash(f"Помилка увімкнення Vosk STT: {e}")
    return redirect(url_for('voice_anim_control.index'))
```

```
#файл розмітки головної сторінки
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Головна панель</title>
</head>
<body>
    <h1>Головна Панель Робота</h1>

    <ul>
        <li><a href="{{ url_for('video') }}">Налаштування та перегляд
відеопотоку</a></li>
        <li><a href="{{ url_for('voice_anim_control.index') }}">Налаштування
голосових команд</a></li>
        <li><a href="{{ url_for('servo_control.servo_control_page')
}}">Налаштування сервоприводів</a></li>
    </ul>

    <h2>Температура CPU: <span id="cpu-temp">Завантаження...</span> °C</h2>
    <h4>Керування Raspberry Pi</h4>
    <form action="{{ url_for('shutdown_pi') }}" method="post"
onsubmit="return confirm('Ви дійсно хочете вимкнути Raspberry Pi?');">
        <button type="submit">Вимкнути Raspberry Pi</button>
    </form>

    <form action="{{ url_for('reboot_pi') }}" method="post" onsubmit="return
confirm('Ви дійсно хочете перезапустити Raspberry Pi?');">
        <button type="submit">Перезапустити Raspberry Pi</button>
    </form>
    <script>
        function updateCPUTemp() {
            fetch('/cpu_temp')
                .then(response => response.json())
                .then(data => {
                    if (data.cpu_temp !== undefined) {
                        document.getElementById('cpu-temp').textContent =
data.cpu_temp.toFixed(1);
                    } else {
                        document.getElementById('cpu-temp').textContent =
'Помилка';
                    }
                })
```

```
                .catch(() => {
                    document.getElementById('cpu-temp').textContent =
'Помилка';
                });
        }

        setInterval(updateCPUTemp, 2000);  // оновлення кожні 2 секунди
        updateCPUTemp();  // перший виклик одразу
    </script>
</body>
</html>

#файл розмітки сторінки відеопотоку
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Потік з камери</title>
</head>
<body>
    <h1>Потік з камери</h1>
    <a href="{{ url_for('index') }}">← Назад на головну</a><br><br>

    <img src="/video_feed" width="640" height="480">

    <h2>Налаштування камери</h2>
    <form action="{{ url_for('update_config') }}" method="post">
        <label for="camera_width">Ширина:</label>
        <input type="number" id="camera_width" name="camera_width" value="{{
config['camera_width'] }}"><br><br>

        <label for="camera_height">Висота:</label>
        <input type="number" id="camera_height" name="camera_height"
value="{{ config['camera_height'] }}"><br><br>

        <label for="frame_delay">Затримка (сек):</label>
        <input type="number" step="0.01" id="frame_delay" name="frame_delay"
value="{{ config['frame_delay'] }}"><br><br>

        <button type="submit">Зберегти</button>
    </form>
    <form action="/toggle_camera" method="get">
        <button type="submit">Увімкнути/Вимкнути
```

```
        </button>
    </form>

    <script>
    window.addEventListener("beforeunload", function (e) {
        navigator.sendBeacon('/stop_video_feed');
    });
    </script>
</body>
</html>

#файл розмітки сторінки керування сервоприводами
<!DOCTYPE html>
<html>
<head>
    <title>Керування Сервами</title>
    <style>
        .slider-container { margin: 15px 0; }
    </style>
</head>
<body>
    <h1>Керування Сервами</h1>

    {% for servo_id, label in {
        'horizontal': 'Очі горизонтально',
        'vertical': 'Очі вертикально',
        'head_horizontal': 'Голова горизонтально',
        'head_vertical': 'Голова вертикально',
        'mouth': 'Рот'
    }.items() %}
        <div class="slider-container">
            <label>{{ label }}:
                <input type="range" min="0" max="180" value="90"
onchange="updateServo(this.value, '{{ servo_id }}')">
                <span id="{{ servo_id }}_val">90</span>°
            </label>
        </div>
    {% endfor %}
    <form method="post" action="{{ url_for('servo_control.start_servo') }}"
style="display:inline;">
        <button type="submit">Старт автоматичного керування</button>
    </form>
```

```
    <form method="post" action="{{ url_for('servo_control.servo_stop') }}"
style="display:inline;">
        <button type="submit">Стоп автоматичного керування</button>
    </form>
    <h2>Редагування конфігурації</h2>
    <form method="post" action="{{ url_for('servo_control.update_config')
}}">
        {% for key, value in config.items() %}
            <label>{{ key }}: <input type="text" name="{{ key }}" value="{{
value|tojson }}"></label><br>
        {% endfor %}
        <button type="submit">Зберегти</button>
    </form>

    <script>
        function updateServo(angle, servoId) {
            document.getElementById(servoId + "_val").innerText = angle;
            fetch("/update-servo-angle", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ angle: angle, servo: servoId })
            }).then(res => res.json()).then(data => {
                if (data.status !== 'ok') alert(data.message);
            });
        }
    </script>
</body>
</html>

#файл розмітки сторінки голосового керування
<!doctype html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <title>Керування голосовими командами</title>
</head>
<body>
<a href="{{ url_for('index') }}">Назад на головну</a>
<h2>Завантаження MP3-файлу у папку Mouth_anim</h2>
<form method="post" enctype="multipart/form-data">
  <input type="file" name="file">
  <input type="submit" value="Завантажити">
</form>
```

```
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul style="color:green;">
    {% for message in messages %}
      <li>{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}

<hr>
<h3>MP3-файли:</h3>
<ul>
  {% for mp3 in mp3_files %}
    <li>
      {{ mp3 }}
      <form method="post" action="/voice_anim_control/delete_file"
style="display:inline;">
        <input type="hidden" name="filename" value="{{ mp3 }}">
        <button type="submit">❌ Видалити</button>
      </form>
    </li>
  {% else %}
    <li>Немає MP3-файлів</li>
  {% endfor %}
</ul>

<h3>JSON-анімації:</h3>
<ul>
  {% for jsonf in json_files %}
    <li>
      {{ jsonf }}
      <form method="post" action="/voice_anim_control/delete_file"
style="display:inline;">
        <input type="hidden" name="filename" value="{{ jsonf }}">
        <button type="submit">❌ Видалити</button>
      </form>
    </li>
  {% else %}
    <li>Немає JSON-файлів</li>
  {% endfor %}
```

```
</ul>

<hr>
<h3>Гучність DFPlayer:</h3>
<form method="post" action="{{ url_for('voice_anim_control.set_volume') }}">
  <input type="range" name="volume" min="0" max="30" value="{{ current_volume
}}" oninput="this.nextElementSibling.value = this.value">
  <output>{{ current_volume }}</output>
  <input type="submit" value="Зберегти">
</form>

<h3>Тип розпізнавання:</h3>
<form action="{{ url_for('voice_anim_control.enable_google_stt_route') }}"
method="post">
  <button type="submit">Увімкнути Google STT</button>
</form>

<form action="{{ url_for('voice_anim_control.enable_vosk_stt_route') }}"
method="post">
  <button type="submit">Увімкнути Vosk STT</button>
</form>

<hr>
<h3>Додати нову голосову команду:</h3>
<form method="post" action="/voice_anim_control/add_command">
  Команда: <input type="text" name="command" required><br>
  JSON-файл:
  <select name="json_file" required>
    {% for jsonf in json_files %}
      <option value="{{ jsonf }}">{{ jsonf }}</option>
    {% endfor %}
  </select>
  <br><input type="submit" value="Додати команду">
</form>

<h3>Існуючі голосові команди:</h3>
<ul>
  {% for command, path in commands.items() %}
    <li>
      <strong>{{ command }}</strong>: {{ path }}
      <form method="post" action="/voice_anim_control/delete_command"
style="display:inline;">
        <input type="hidden" name="command_name" value="{{ command }}">
```

```
        <button type="submit">❌ Видалити</button>
      </form>
    </li>
  {% else %}
    <li>Команди відсутні</li>
  {% endfor %}
</ul>

<hr>
<h3>Редагувати існуючу команду:</h3>
<form method="post" action="/voice_anim_control/edit_command">
  Виберіть команду:
  <select name="command_name" required>
    {% for command in commands.keys() %}
      <option value="{{ command }}">{{ command }}</option>
    {% endfor %}
  </select><br>
  Новий JSON-файл:
  <select name="new_json_file" required>
    {% for jsonf in json_files %}
      <option value="{{ jsonf }}">{{ jsonf }}</option>
    {% endfor %}
  </select><br>
  <input type="submit" value="Оновити команду">
</form>

<hr>
<h3>Запуск створення анімацій:</h3>
<form method="post" action="/voice_anim_control/run_script">
  <input type="submit" value="Запустити скрипт">
</form>

<h3>Результат виконання:</h3>
<pre>{{ result }}</pre>
</body>
</html>
```

```
#файл конфігурації
{
  "servo_horizontal_channel": 0,
  "servo_vertical_channel": 1,
  "head_servo_horizontal_channel": 2,
  "head_servo_vertical_channel": 3,
  "mouth_servo_channel": 4,
  "horizontal_min_angle": 40,
  "horizontal_max_angle": 140,
  "vertical_min_angle": 0,
  "vertical_max_angle": 180,
  "loop_delay": 0.05,
  "min_step": 1,
  "max_step": 10,
  "camera_width": 320,
  "camera_height": 240,
  "frame_delay": 0.2,
  "head_min_step": 1,
  "head_max_step": 10,
  "head_min_horizontal_angle": 0,
  "head_max_horizontal_angle": 180,
  "head_min_vertical_angle": 0,
  "head_max_vertical_angle": 180,
  "invert_horizontal": true,
  "invert_vertical": false,
  "invert_head_horizontal": false,
  "invert_head_vertical": false,
  "invert_mouth_animation": true,
  "servo_horizontal_pulse": [
    400,
    2500
  ],
  "servo_vertical_pulse": [
    400,
    2500
  ],
  "head_horizontal_pulse": [
    500,
    2400
  ],
  "head_vertical_pulse": [
    2100,
    2400
```

```
  ],
  "mouth_pulse": [
    600,
    2200
  ],
  "eye_threshold_ratio": 0.5,
  "eye_deviation_threshold_percent": 15,
  "eye_deviation_duration_sec": 3,
  "head_adjustment_angle": 15,
  "volume": 30
}
```

ДОДАТОК В

Демонстраційний матеріал

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кільк.	Примітки
			Текстові документи		
А4	1	ГЮІК.412150.009 ПЗ	Пояснювальна записка	52 с.	
			Інші документи	37 с.	
А4	2		Апробація результатів роботи	4 с.	
А4	3		Код програмного засобу		
А4	4		Демонстраційний матеріал	6 с.	

Змін.	Арк	Номер докум.	Підпис	Дата	ГЮІК 412150.009 ВД			
Розроб.		Бєлий Я. В.			Розроблення технічного засобу візуалізації стану робота-промоутера Відомість кваліфікаційної роботи	Літера	Арку	Аркуш
Перевір.		Сичова О.В.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Демська Н.П.						
Затв.		Невлюдов І.Ш.						