Smart Voice-Controlled Car System Documentation

---

1. Introduction

This project presents a **Smart Voice-Controlled Car System** that allows a robotic car to be controlled using **Arabic and English voice commands**. The system integrates **Artificial Intelligence (Natural Language Processing)** with **IoT and Embedded Systems** to convert human speech into real motor movements.

> ✅Flutter mobile application is **excluded** from this documentation as requested.

---

2. Project Objective

The main goal of this project is to: - Recognize Arabic/English voice commands - Classify the command intent using an NLP model - Convert the intent into motor control commands - Control a physical robotic car remotely using cloud communication

---

3. System Architecture Overview

End-to-End Flow:

Voice / Text Input
→ NLP Intent Classification Model
→ Flask Backend API
→ Intent-to-Command Mapping
→ MQTT (HiveMQ Cloud)
→ ESP8266 (Wi-Fi + MQTT Subscriber)
→ Arduino UNO (Motor Controller)
→ DC Motors

---

4. Technologies Used

Software Technologies

- Python 3
- Flask (REST API)
- scikit-learn
- joblib

- Paho-MQTT
- SpeechRecognition

Cloud Service

- HiveMQ Cloud (Secure MQTT Broker)

Hardware Components

- ESP8266 (NodeMCU)
- Arduino UNO
- L298N Motor Driver
- DC Motors
- External battery supply

---

5. NLP Intent Classification Model

5.1 Purpose of the Model

The NLP model converts **spoken or typed commands** into predefined intents that represent car movements.

Example Command	Intent
امشي قدام	forward
امشي ورا	backward
امشي يمين	right
امشي شمال	left
اقف	stop

5.2 Dataset Structure

The dataset is stored in a CSV file with two columns:

Column	Description
text	Raw command text
intent	Target intent label

5.3 Text Preprocessing

To improve accuracy, the following preprocessing steps are applied: - Lowercasing - Removing extra spaces - Arabic normalization (أ → ا, ة → ه, ى → ي) - Normalizing command synonyms (وقف → اقف)

5.4 NLP Model Code (Final Working Version)

```
# Intent Classification Model for Smart Car Control

import os, sys, re, joblib
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Load dataset
df = pd.read_csv('dataset.csv')

# Text cleaning function
def clean_text(text):
    text = str(text).lower().strip()
    text = re.sub(r"[أإآ]", "ا", text)
    text = re.sub(r"ة", "ه", text)
    text = re.sub(r"ى", "ي", text)
    text = re.sub(r"(وقف|اوقف|توقف|ستوب)", "اقف", text)
    text = re.sub(r"قدامي", "قدام", text)
    text = re.sub(r"ورى", "ورا", text)
    return text

df['text_clean'] = df['text'].apply(clean_text)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    df['text_clean'], df['intent'], test_size=0.15, random_state=42,
stratify=df['intent']
)

# Training pipeline
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(ngram_range=(1,3))),
    ('clf', LogisticRegression(max_iter=2000, C=10, solver='lbfgs'))
])
```

```python
pipeline.fit(X_train, y_train)

# Evaluation
y_pred = pipeline.predict(X_test)
print(classification_report(y_test, y_pred))

# Save model
os.makedirs('models', exist_ok=True)
joblib.dump(pipeline, 'models/nlp_intent_model.joblib')
```

5.5 Model Performance

- Accuracy achieved: **~93%**
- Robust for both Arabic and English commands
- Handles spelling variations and informal Arabic

---

6. Flask Backend API

6.1 Backend Responsibilities

- Load NLP model once at startup
- Receive command text via HTTP POST
- Predict intent
- Map intent to motor command
- Publish command to MQTT broker

6.2 Intent to Command Mapping

Intent	Motor Command
forward	F
backward	B
left	L
right	R
stop	S

6.3 Flask API (Final Version)

```python
# Flask API for Smart Car Control

from flask import Flask, request, jsonify
import joblib, re
import paho.mqtt.client as mqtt
import ssl

app = Flask(__name__)

INTENT_TO_COMMAND = {
    "forward": "F",
    "backward": "B",
    "left": "L",
    "right": "R",
    "stop": "S"
}

# MQTT setup
MQTT_BROKER = "<HiveMQ_URL>"
MQTT_PORT = 8883
MQTT_TOPIC = "car/control"
MQTT_USERNAME = "<USERNAME>"
MQTT_PASSWORD = "<PASSWORD>"

client = mqtt.Client()
client.username_pw_set(MQTT_USERNAME, MQTT_PASSWORD)
client.tls_set(cert_reqs=ssl.CERT_REQUIRED)
client.connect(MQTT_BROKER, MQTT_PORT)

model = joblib.load('models/nlp_intent_model.joblib')

# Clean text
def clean_text(text):
    text = text.lower().strip()
    text = re.sub(r"[إأآ]", "ا", text)
    text = re.sub(r"ö", "o", text)
    text = re.sub(r"ى", "ي", text)
    text = re.sub(r"(وقف|اوقف|توقف|ستوب)", "اقف", text)
    return text

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()
    text = data.get('text', '')
```

```
    clean_cmd = clean_text(text)
    intent = model.predict([clean_cmd])[0]
    command = INTENT_TO_COMMAND.get(intent, 'S')

    client.publish(MQTT_TOPIC, command)

    return jsonify({
        'input': text,
        'intent': intent,
        'command': command
    })

if __name__ == '__main__':
    app.run(debug=True)
```

---

7. MQTT Communication (HiveMQ Cloud)

- Secure TLS connection
- Real-time command delivery
- Topic used:

  ```
  car/control
  ```

Payload is a **single character** representing motor action.

---

8. ESP8266 (MQTT Subscriber)

Role

- Connect to Wi-Fi
- Subscribe to MQTT topic
- Receive command
- Send command to Arduino via Serial

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

WiFiClientSecure espClient;
PubSubClient client(espClient);

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.write((char)payload[0]);
```

```
}

void setup() {
  Serial.begin(9600);
  espClient.setInsecure();
  client.setServer("<HiveMQ_URL>", 8883);
  client.setCallback(callback);
}

void loop() {
  if (!client.connected()) {
    client.connect("ESP8266", "<USER>", "<PASS>");
    client.subscribe("car/control");
  }
  client.loop();
}
```

---

9. Arduino UNO (Motor Control)

```
#define IN1 8
#define IN2 9
#define IN3 10
#define IN4 11
#define ENA 5
#define ENB 6

char command;

void setup() {
  Serial.begin(9600);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
}

void loop() {
  if (Serial.available()) {
    command = Serial.read();
    if (command == 'F') forward();
    else if (command == 'B') backward();
    else if (command == 'L') left();
```

```
    else if (command == 'R') right();
    else stopMotors();
  }
}
```

---

10. Final Results

- The car responds accurately to Arabic/English commands
- NLP accuracy achieved **~93%**
- Real-time cloud communication
- Full AI + IoT integration

---

11. Conclusion

This project demonstrates a **complete real-world application** of Artificial Intelligence integrated with embedded systems. It proves that natural language can be reliably used to control physical devices.

---

12. Flutter Mobile Application

12.1 Overview

The Flutter mobile application represents the **user interface and voice input layer** of the Smart Car system. It allows the user to control the car using Arabic voice commands and communicates directly with the Flask backend server.

Flutter is responsible only for: - Capturing voice commands - Converting speech to text - Sending text to the backend server - Displaying system status and responses

> Flutter does **NOT** perform NLP processing or motor control logic.

12.2 Hardware Components Used

- Arduino Board
- Motor Driver (L298N)
- External Power Supply
- Communication Module (ESP8266)

12.3 Installation and Setup

12.3.1 Flutter Application Setup

A. Prerequisites

```
flutter doctor
```

Install dependencies

```
cd flutter_application_1
flutter pub get
```

B. Assets Configuration

Add the following to `pubspec.yaml`:

```
flutter:
  assets:
    - assests/image/logo.png
```

Ensure the asset file exists at:

```
assests/image/logo.png
```

C. Build Commands

Development mode:

```
flutter run
```

Production (Android):

```
flutter build apk --release
```

Production (iOS):

```
flutter build ios --release
```

12.3.2 Backend Server Setup

```
pip install -r requirements.txt
python app.py
```

12.3.3 Hardware Setup

- Upload Arduino motor control code to the board
- Connect DC motors to the motor driver
- Connect ESP8266 to Arduino via Serial
- Power motors using an external power supply
- Verify MQTT connectivity

---

13. Application Usage

13.1 Initial Configuration

1. Launch the Flutter application
2. Wait for the splash screen to finish
3. Open **Settings** by tapping the 🛑icon
4. Enter the backend server address:

Using ngrok:

```
https://your-app.ngrok-free.dev
```

Using local network:

```
192.168.1.100:5000
```

1. Press **Save Settings (💾)**

13.2 Voice Control Operation

1. Ensure connection status shows **Connected** (green)
2. Press the microphone button 🆎
3. Speak a command clearly (Arabic)
4. The recognized text is sent to the Flask server
5. The server responds with intent and command
6. The car executes the movement in real time

13.3 Supported Voice Commands

Movement Commands

- "تقدم" / "تقدم للأمام" / "امشي"
- "تراجع" / "ارجع للخلف"
- "يمين" / "اتجه يمين"
- "يسار" / "اتجه يسار"
- "قف" / "توقف"

Speed Commands (Future Support)

- "أسرع" / "زود السرعة"
- "أبطأ" / "قلل السرعة"

Status Commands (Future Support)

- "ما الحالة؟" / "وضع السيارة"

---

14. Flutter Application Features

User Interface

- Modern gradient-based UI
- Full Arabic language support
- Dark mode by default
- Smooth animations and transitions

Voice Recognition

- High accuracy Arabic recognition (ar_SA)
- Partial results support
- Automatic stop after silence
- Comprehensive error handling

Connectivity

- Dynamic backend server configuration
- HTTPS and HTTP support
- Automatic connection validation
- Clear status indicators

Local Storage

- Persistent server settings
- Session continuity
- Reset configuration option

---

15. Additional Project Information

- Application Version: 1.0.0+1
- Flutter SDK: ^3.10.0
- Total Project Files: ~100
- Dart Files: 6
- Application Screens: 3
- Services: 2

---

16. Notes and Warnings

Important Notes

- Missing files:
- Flask server Python files
- Arduino (.ino) motor control code

These components are mandatory for full system functionality.

- Default server URL:

  ```
  https://ungraduating-kaylee-nakedly.ngrok-free.dev
  ```

  This is a temporary ngrok link and must be updated when restarted.

- Language:

- Arabic (ar_SA) supported

- Language can be modified from `voice_service.dart`

• Security:

• HTTPS is recommended for production environments
• Microphone permissions must be granted

---

17. Conclusion

The Flutter application completes the Smart Car ecosystem by providing a **reliable, user-friendly voice interface**. When integrated with the Flask AI backend and IoT hardware, the system demonstrates a powerful real-world application of **AI-driven voice control for embedded systems**.

---

18. Future Improvements

• Manual control mode
• Speed control by voice
• Obstacle avoidance sensors
• Command history logging
• Multi-language support
• Permanent cloud deployment