Instituto Politécnico da Guarda

Escola Superior de Tecnologia e Gestão

Sistemas de Aquisição e Registo de Dados

Relatório do Projeto – Aquisição de Dados e Reconhecimento de Voz

Janeiro de 2026

Curso:

Licenciatura em Ciência de Dados e Inteligência Artificial (LCDIA)

Unidade Curricular: Sistemas de Aquisição e Registo de Dados

(SARD)

Ano Letivo: 2025/2026

Autor: Ericleiton Mendes – nº1710344

Índice

Lista de Abreviaturas

- **IoT** – Internet of Things
- **CSV** – Comma-Separated Values
- **HMI** – Human-Machine Interface
- **UC** – Unidade Curricular

Resumo

O presente trabalho teve como objetivo o desenvolvimento de um sistema de aquisição e registo de dados ambientais, recorrendo a sensores físicos, um microcontrolador e software desenvolvido em Python. O sistema permite a recolha de dados de temperatura e humidade, bem como a associação de informação contextual fornecida pelo utilizador através de reconhecimento de voz.

Os dados recolhidos são tratados e armazenados num ficheiro CSV, possibilitando a sua análise posterior. O projeto enquadra-se no contexto de sistemas IoT e demonstra a integração entre hardware, comunicação série, processamento de dados e interação homem-máquina.

Palavras-chave: Aquisição de Dados, IoT, Sensores, Python, Reconhecimento de Voz

Introdução

A utilização de microcontroladores associados a plataformas IoT tem sido amplamente aplicada em sistemas de aquisição de dados ambientais, permitindo a recolha, armazenamento e visualização remota da informação em tempo real (Rachmawati et al., 2023).

Neste contexto, o objetivo deste exercício da unidade curricular de **Sistemas de Aquisição e Registo de Dados (SARD)** foi desenvolver um sistema modular capaz de adquirir dados provenientes de sensores físicos, capturar input humano através de reconhecimento de voz e armazenar toda a informação de forma estruturada. Projetos baseados em sensores e microcontroladores são comuns em sistemas IoT, sendo amplamente utilizados em aplicações de monitorização ambiental e recolha de dados em tempo real.

O projeto desenvolvido foca-se na leitura de um sensor de temperatura e humidade (SHT30) e na recolha do estado do tempo indicado pelo utilizador, registando todas as informações num ficheiro CSV (*registos.csv*) para posterior análise.

Resolução Teórica

Aquisição de Dados

A leitura da temperatura e humidade é realizada através de um sensor digital SHT30, amplamente utilizado em aplicações IoT devido à sua fiabilidade e facilidade de integração com microcontroladores.

A comunicação entre o microcontrolador e o computador é feita através da interface série, abordagem comum em sistemas de aquisição de dados baseados em Arduino.

Comunicação com Python

No lado do software, foi utilizada a biblioteca PySerial, que permite a leitura de dados provenientes da porta série de forma simples e eficiente, sendo uma solução frequentemente utilizada em projetos de aquisição de dados em Python.

Reconhecimento de Voz

Para permitir a interação do utilizador com o sistema, foi implementado um módulo de reconhecimento de voz recorrendo às bibliotecas SpeechRecognition e PyAudio, possibilitando a captura de comandos de voz em tempo real.

Este tipo de abordagem é comum em aplicações de interação homem-máquina, permitindo associar dados técnicos a informação contextual fornecida pelo utilizador.

Armazenamento de Dados

Os dados recolhidos são armazenados num ficheiro CSV, opção simples e eficaz para registo estruturado de informação, permitindo posterior análise ou integração com bases de dados.

Detalhes Práticos

Bibliotecas e Tecnologias Utilizadas

Comunicação e Aquisição de Dados

A comunicação entre o microcontrolador e a aplicação desenvolvida em Python é realizada através de comunicação série, uma abordagem comum em sistemas de aquisição de dados que permite a transferência de informação em tempo real entre dispositivos físicos e software (Arduino, n.d.; PySerial Development Team, n.d.).
Para este efeito, foi utilizada a biblioteca **PySerial**, cuja principal função é estabelecer e gerir a ligação à porta série, permitindo a leitura contínua dos dados enviados pelo microcontrolador. Esta biblioteca é responsável por receber os valores medidos pelo sensor SHT30 e disponibilizá-los à aplicação principal para posterior tratamento.

Reconhecimento de Voz

O reconhecimento de voz do utilizador é realizado com recurso à biblioteca **speech_recognition**, que permite capturar áudio a partir de um microfone e convertê-lo em texto. Esta funcionalidade é utilizada para recolher o estado do tempo indicado pelo utilizador (por exemplo, “sol”, “chuva” ou “nevoeiro”), associando essa informação contextual aos dados recolhidos pelos sensores.

Para a captura de áudio em tempo real, é utilizada a biblioteca **PyAudio**, que funciona como interface entre o sistema operativo e o microfone. Bibliotecas deste tipo permitem implementar mecanismos simples de reconhecimento de voz em aplicações Python (PyAudio, n.d.), sendo o PyAudio integrado diretamente com a biblioteca speech_recognition.

Armazenamento e Tratamento de Dados

O armazenamento dos dados recolhidos é realizado de forma estruturada, recorrendo inicialmente a ficheiros CSV. Para esse efeito, é utilizada a biblioteca **csv**, que permite a escrita organizada dos dados num ficheiro (*registos.csv*), incluindo o timestamp, a temperatura, a humidade e a condição meteorológica indicada pelo utilizador.

Adicionalmente, é utilizada a biblioteca **sqlite3** para a criação e manipulação de uma base de dados SQLite (*sard.db*). Bases de dados leves como o SQLite são amplamente utilizadas em sistemas embebidos e aplicações de pequena escala, devido à sua simplicidade e eficiência (SQLite Consortium, n.d.). Esta abordagem permite a importação dos dados do ficheiro CSV para a base de dados, possibilitando consultas e análises futuras.

A biblioteca **datetime** é utilizada para a manipulação de datas e horários, garantindo que cada registo armazenado contém um timestamp preciso, tanto no ficheiro CSV como na base de dados.

Hardware e Microcontrolador

Relativamente ao hardware, o sistema utiliza um **sensor digital de temperatura e humidade SHT30**, amplamente aplicado em sistemas IoT devido à sua precisão e facilidade de integração (Sensirion AG, n.d.). Este sensor é responsável pela medição contínua dos parâmetros ambientais.

A leitura dos dados do sensor e o seu envio para o computador são realizados por um microcontrolador baseado na plataforma **ESP8266** (LOLIN/Wemos). Microcontroladores deste tipo são frequentemente utilizados em projetos IoT de pequena escala, devido ao seu baixo custo e conectividade integrada (Wemos Electronics, n.d.). O microcontrolador atua como intermediário entre o sensor físico e a aplicação Python, enviando os dados recolhidos através da comunicação série.

Módulos e Estrutura do Sistema

app.py – Módulo Principal

O módulo `app.py` constitui o núcleo do sistema, sendo responsável por coordenar todos os restantes módulos e garantir o correto fluxo de execução do programa. A aplicação segue uma abordagem síncrona típica de sistemas de aquisição de dados, onde cada leitura do sensor é processada antes de avançar para a etapa seguinte, assegurando consistência entre os dados recolhidos.

O fluxo de funcionamento consiste na leitura dos dados do sensor através do módulo `read_data`, seguida da recolha de input humano via reconhecimento de voz e, por fim, do registo persistente da informação. Este tipo de organização é comum em sistemas de aquisição de dados em tempo real, onde a sequência de operações garante a integridade dos registos (Arduino, n.d.; PySerial Development Team, n.d.).

read_data.py – Leitura de Sensores

O módulo `read_data.py` é responsável pela comunicação entre a aplicação Python e o microcontrolador através da porta série. A comunicação série é amplamente utilizada em sistemas de aquisição de dados por permitir a transferência contínua e fiável de informação entre dispositivos físicos e software (Arduino, n.d.).

Este módulo inicializa a ligação série, lê as mensagens enviadas pelo microcontrolador e procede à decodificação dos dados recebidos. São também implementados mecanismos de filtragem para ignorar linhas inválidas ou bytes corrompidos, aumentando a robustez do processo de aquisição de dados (PySerial Development Team, n.d.).

voice_recognition.py – Captura de Input de Voz

O módulo `voice_recognition.py` permite a interação do utilizador com o sistema através de reconhecimento de voz. Para esse efeito, recorre à biblioteca `speech_recognition`, que possibilita a conversão de áudio captado por um microfone em texto, utilizando serviços externos de reconhecimento de fala.

A utilização de reconhecimento de voz como forma de input permite enriquecer os dados recolhidos com informação contextual fornecida pelo utilizador, sendo uma abordagem cada vez mais comum em sistemas interativos e aplicações IoT (PyAudio, n.d.). Caso o áudio não seja corretamente reconhecido, o sistema solicita novamente o input ao utilizador, garantindo a validade da informação registada.

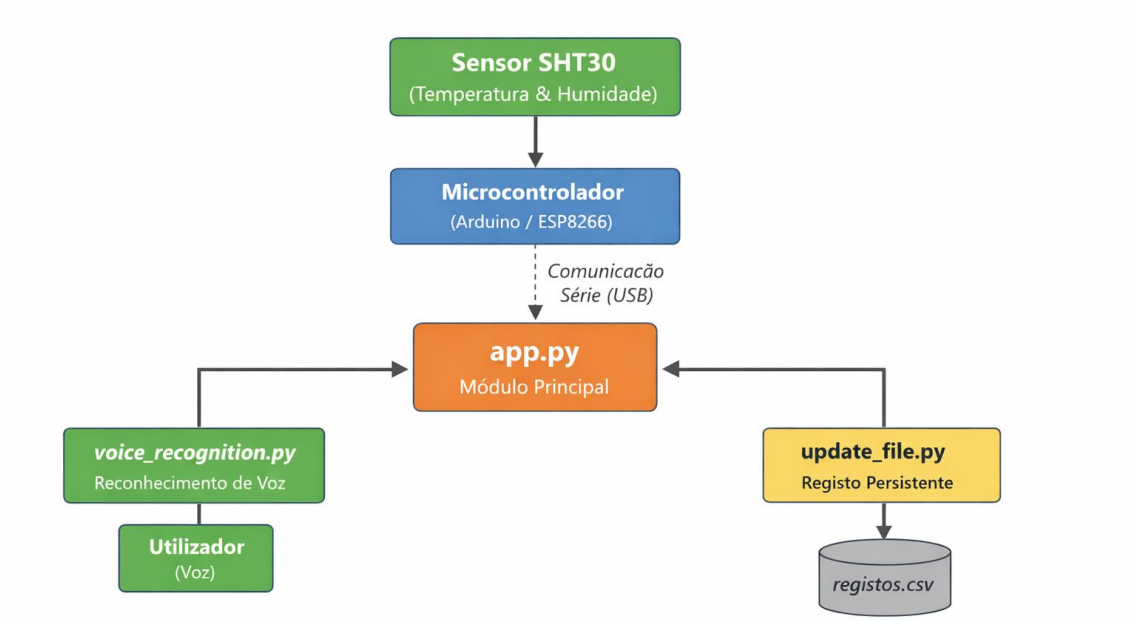
update_file.py – Registo Persistente

O módulo `update_file.py` é responsável pelo armazenamento persistente dos dados recolhidos. Os registos são guardados num ficheiro CSV (*registos.csv*), formato

amplamente utilizado pela sua simplicidade e compatibilidade com diversas ferramentas de análise.

O ficheiro é criado apenas caso não exista previamente, assegurando a presença do cabeçalho, e cada novo registo é adicionado sem eliminar dados anteriores. Esta abordagem de armazenamento estruturado é comum em sistemas de pequena escala e pode ser posteriormente complementada com bases de dados leves, como o SQLite, frequentemente utilizadas em aplicações embebidas (SQLite Consortium, n.d.).

Após a descrição dos módulos que compõem o sistema, a **Figura 1** apresenta o diagrama geral de funcionamento da solução desenvolvida, evidenciando a interação entre o sensor de temperatura e humidade, o microcontrolador e os diferentes módulos da aplicação Python. Este diagrama permite compreender o fluxo de dados desde a aquisição dos valores ambientais até ao seu registo persistente. Complementarmente, a **Figura 2** ilustra a execução prática do sistema no terminal, onde é possível observar a leitura dos dados do sensor, o reconhecimento do input de voz fornecido pelo utilizador e o consequente armazenamento da informação, confirmando o correto funcionamento do sistema implementado.



(Figura 1- Diagrama de Funcionamento do Sistema)

```
PS C:\Users\cleit\Downloads\ProjetoSARD\ProjetoSARD\python> python .\app.py
Sistema iniciado...
Dados do sensor: FF;20.17;74.32
Fala agora...
Diz o estado do tempo (ex: sol, chuva, nevoeiro):
Reconhecido: sol
Registo guardado

Dados do sensor: FF;20.17;74.55
Fala agora...
Diz o estado do tempo (ex: sol, chuva, nevoeiro):
Reconhecido: chuva
Registo guardado

Dados do sensor: FF;20.19;74.68
Fala agora...
Diz o estado do tempo (ex: sol, chuva, nevoeiro):
Reconhecido: nevoeiro
Registo guardado
```

(Figura 2 – Execução do sistema no terminal Python)

Conclusão

O sistema desenvolvido segue uma arquitetura semelhante à utilizada em vários projetos IoT descritos na literatura, integrando sensores físicos, comunicação série, tratamento de dados em Python e persistência da informação.

Como trabalho futuro, seria possível expandir o projeto através da criação de uma interface gráfica para visualização dos dados em tempo real, tornando o sistema mais intuitivo para o utilizador final.

Referências Bibliográficas

Arduino. (n.d.). *Serial communication – Arduino reference*. https://www.arduino.cc

GeeksforGeeks. (n.d.). *Serial communication in Arduino*. https://www.geeksforgeeks.org

Kumar, S., & Reddy, A. (n.d.). *IoT-based weather monitoring system using NodeMCU ESP8266*. https://www.geeksforgeeks.org

Kurniawan, A. (2017). *Internet of Things with ESP8266: Build amazing Internet of Things projects using the ESP8266 Wi-Fi chip*. Packt Publishing.

Patel, R., & Shah, K. (n.d.). *ESP32-based data logger*. https://www.researchgate.net

PyAudio. (n.d.). *Python bindings for PortAudio: Documentation*. https://people.csail.mit.edu/hubert/pyaudio/

PySerial Development Team. (n.d.). *PySerial documentation: Serial port access in Python*. https://pyserial.readthedocs.io

Rahman, M., Islam, S., & Hossain, M. (n.d.). *A proposed approach to utilizing ESP32 microcontroller for data acquisition*. https://www.researchgate.net

Sensirion AG. (n.d.). *SHT3x-DIS digital humidity and temperature sensor: Datasheet*. https://www.sensirion.com

SQLite Consortium. (n.d.). *SQLite documentation*. https://www.sqlite.org/docs.html

Wemos Electronics. (n.d.). *LOLIN D1 mini (ESP8266) documentation*. https://www.wemos.cc

Apêndices

Apêndice A - Código do Microcontrolador:

Neste apêndice é apresentado o código desenvolvido para o microcontrolador, responsável pela leitura dos dados do sensor de temperatura e humidade e pelo envio dessa informação para o computador através da comunicação série.

```
#include <WEMOS_SHT3X.h>

SHT3X sht30(0x45);

void setup() {

  Serial.begin(115200);
  Wire.begin();

}

void loop() {

  if(sht30.get()==0){
    Serial.print("FF;");
    Serial.print(sht30.cTemp);
    Serial.print(";");
    Serial.println(sht30.humidity);

    Serial.println();

  }
  else
  {
    Serial.println("Error!");
  }
  delay(1000);

}
```

Apêndice B – Código Python

Este apêndice contém o código desenvolvido em Python, responsável pela leitura dos dados provenientes da porta série, captura do input de voz do utilizador e armazenamento da informação num ficheiro CSV.

Este apêndice está dividio em 4 partes:

- B.1 app.py
- B.2 read_data.py
- B.3 voice_recognition.py
- B.4 update_file.py

B.1 - app.py

```
from read_data import ler_sensor
from voice_recognition import ouvir_condicao
from update_file import guardar_registo, inicializar_csv
import time

inicializar_csv()
print("Sistema iniciado...")

while True:
    dados = ler_sensor()

    if dados:
        print("Dados do sensor:", dados)

        try:
            _, temperatura, humidade = dados.split(";")
            temperatura = float(temperatura)
            humidade = float(humidade)

            pressao = None  # ainda não tens sensor de pressão

            print("Fala agora...")
            condicao = ouvir_condicao()

            guardar_registo(temperatura, pressao, humidade,
condicao)
            print("Registo guardado\n")

        except ValueError:
            print("Erro a processar os dados:", dados)

        time.sleep(2)
```

B.2 read_data.py

```
import serial
import time

ser = serial.Serial('COM6', 115200, timeout=2)
time.sleep(2)
ser.reset_input_buffer()

def ler_sensor():
    linha = ser.readline()
    if not linha:
        return None

    texto = linha.decode('utf-8', errors='ignore').strip()

    if texto.startswith("FF"):
        return texto

    return None
```

B.3 voice_recognition.py

```
import speech_recognition as sr

def ouvir_condicao():

    """
    Captura o input de voz do utilizador e retorna o texto em
minúsculas.
    Repete se não for possível reconhecer.
    """
    r = sr.Recognizer()
    with sr.Microphone() as source:
        print("Diz o estado do tempo (ex: sol, chuva, nevoeiro):")
        audio = r.listen(source)
        try:
            texto = r.recognize_google(audio, language='pt-PT')
            print("Reconhecido:", texto)
            return texto.lower()
        except sr.UnknownValueError:
            print("Não percebi, tenta de novo.")
            return ouvir_condicao()
        except sr.RequestError:
            print("Erro no serviço de reconhecimento.")
            return None
```

B.4 update_file.py

```
import csv
from datetime import datetime

CSV_FILE = 'registos.csv'


def inicializar_csv():
    try:
        with open(CSV_FILE, 'x', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(['Timestamp', 'Temperatura', 'Pressao',
'Humidade', 'Condicao'])
    except FileExistsError:
        pass  # já existe

def guardar_registo(temperatura, pressao, humidade, condicao):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(CSV_FILE, 'a', newline='') as f:
        writer = csv.writer(f)
        writer.writerow([
            timestamp,
            temperatura,
            pressao if pressao is not None else "",
            humidade,
            condicao
        ])
```