UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

Lucas Martins Primo
Raul Nicolini Rodrigues
Renato Souza Santana Filho

Roteiro para aquisição de dados

UBERLÂNDIA
2024

- Aquisição de dados com potenciômetro

1. Hardware

O hardware consiste em um ESP32 como microcontrolador conectado a um potenciômetro conforme indica a Figura 1. Colocou-se o pino 1 do knob na porta de 3V3 do ESP32, enquanto o pino 2 foi conectado à uma porta de ADC e o pino 3 disposto no GND do microcontrolador. A imagem não mostra, mas o circuito é alimentado pelo USB, fornecendo uma tensão de 5V para o circuito.

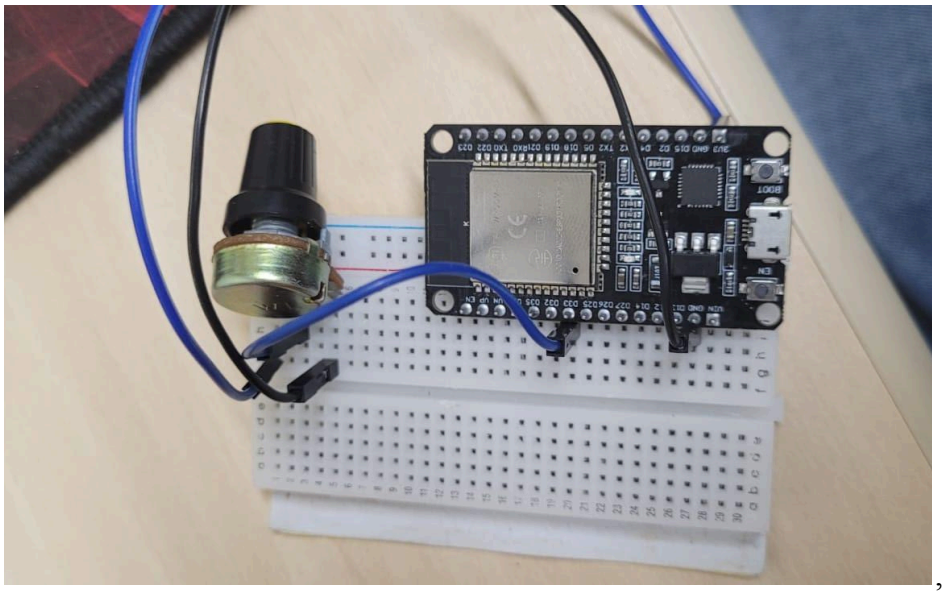 ,

Figura 1. Hardware da aquisição de dados com um potenciômetro.

2. Software

O software consiste num sistema de aquisição de dados ADC (Conversor Analógico-Digital) utilizando um microcontrolador ESP32 conectado a uma rede WiFi.

A configuração sem fio é feita através da função *sendCORSHeaders()* da biblioteca *WebServer* funcionando como um método *full duplex*, ou seja, é feita uma requisição juntamente ao envio de dados. As credenciais da rede WiFi são definidas pelo usuário, assim para conexão entre o ESP32 e um dispositivo celular deve-se colocar o mesmo login e senha tanto no código para o microcontrolador quanto para a rede, assim o endereço IP é exibido no monitor serial.

No geral, o código utiliza RTOS (*Real-Time Operating System*) para que sejam executadas tarefas distintas, porém ao mesmo tempo. A leitura ADC é feita pela tarefa *void vADCTask()* na qual sempre é requisita e usa a função *float readvADCValue()* que é usada para ler o valor do ADC no pino especificado e converter para volts.

Além disso, encontram-se as funções *void startAcquisition(), void stopAcquisition()* e *void clearBuffer()* que são, respectivamente, para iniciar a aquisição, parar a aquisição e deletar valores antigos da aquisição. Nada irá acontecer se a função *WiFi.begin()* não estiver pronta, sendo ela responsável pela conexão sem fio.

Já no aplicativo, Figura 2, no dispositivo celular temos a visualização do gráfico formado acima do slider em azul. O slider é responsável por alterar a taxa de aquisição do sinal, ou seja, se ele deslizar para esquerda o tempo de aquisição é menor, em contrapartida, se colocado para direita esse tempo será aumentado, e também há o slider “ajuste da escala de Tempo” Que permite visualizar diferentes trechos do gráfico se fosse uma aproximação da escala facilitando olhar menores oscilações . O botão da direita “Iniciar” é responsável por começar a coleta de dados, já o botão da direita “Parar” pausa a aquisição de dados, reiniciando o gráfico. Abaixo do símbolo da UFU será mostrado o valor nominal lido do sinal.

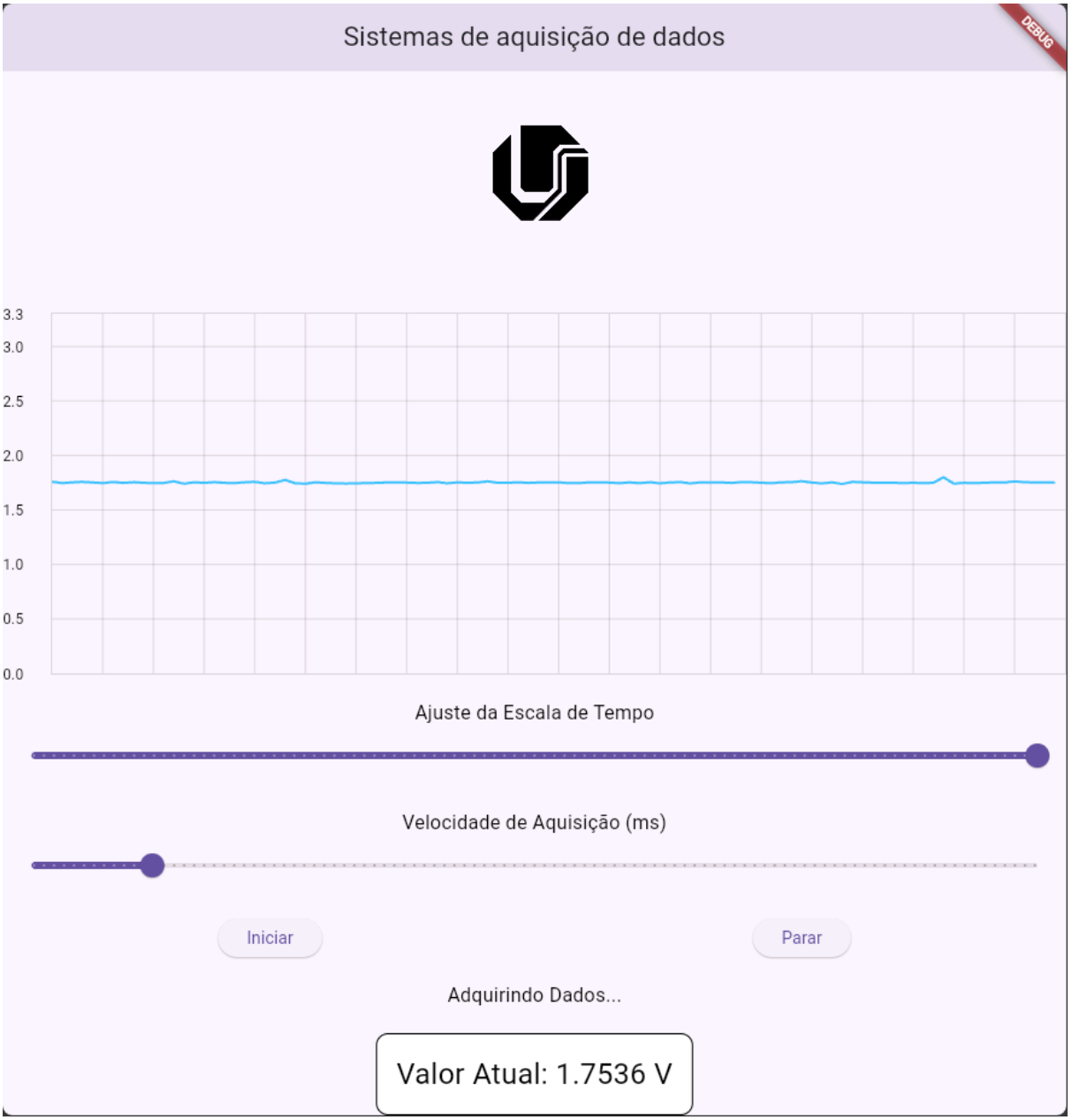

Figura 2. Interface do software no celular.

O aplicativo foi construído na plataforma Visual Studio Code, utilizando-se de uma extensão da plataforma FlutterFlow, específica para produção de aplicativos. Assim, com a criação de um arquivo Flutter, cerca de 90% da camada de software foi obtida, visto que são comandos padrão para todo aplicativo. Por isso, os esforços se focaram nos arquivos main.dart, data_aquisition_state.dart e chart_widget.dart. Neste primeiro, exporta-se outros códigos de outras dependências para criar conteineres, obter imagens no local desejado e toda estilização do aplicativo.

No desenvolvimento de sistemas de aquisição de dados em tempo real, os códigos DataAcquisition e ChartWidget desempenham papéis fundamentais. DataAcquisition é um widget stateful do Flutter, responsável por gerenciar a coleta contínua de dados de um potenciômetro. Esta classe estabelece a conexão com o hardware, lê os valores e mantém um fluxo constante de dados, utilizando um estado interno e uma lista para armazenar os valores adquiridos. Ela também configura um listener para atualizar os dados em tempo real e garante a liberação adequada dos recursos ao fechar o StreamController quando o widget é destruído. O método build() retorna um ChartWidget, que exibe os dados atualizados em um gráfico.

ChartWidget, um StatelessWidget do Flutter, renderiza os dados adquiridos em um gráfico de linhas interativo utilizando a biblioteca Syncfusion. Recebe uma lista de objetos ChartData e configura um gráfico cartográfico com eixos apropriados, aplicando animações suaves para melhorar a experiência do usuário. Juntas, essas classes proporcionam uma solução completa e eficiente para a aquisição e visualização de dados em sistemas de monitoramento em tempo real, destacando-se pela clareza e interatividade.

Entre as importações necessárias, destaca-se a biblioteca http, visto que o aplicativo é baseado no acesso a uma webpage criada com o endereço específico do microcontrolador, a biblioteca provider e a biblioteca fluter_spinkit, necessárias para a plotagem do gráfico. Após aplicar o comando flutter pub get, seguido do comando flutter analyze e flutter run, a aplicação estará pronta para utilização. Por fim, só é necessário inserir o comando flutter build apk, para gerar o arquivo do aplicativo que, se baixado em um smartphone, o torna passível de utilização desse sistema de plotagem de nível DC de um potenciômetro.

- **Software Embarcado**

```
#include <WiFi.h>
#include <WebServer.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

// Definição das credenciais de WiFi
const char *ssid = "Martins Wifi6";
const char *password = "17031998";
```

```
WebServer server(80);
const int bufferSize = 100;  // Número máximo de pontos no gráfico
float vADCBuffer[bufferSize];
int bufferIndex = 0;
bool updatingData = false;
unsigned long acquisitionRate = 50; // Taxa de aquisição fixa
const int pinvADC = 34;

TaskHandle_t vADCTaskHandle = NULL;

// Função para ler o valor do sinal vADC
float readvADCValue() {
  digitalWrite(26, !digitalRead(26));
  int valorADC = analogRead(pinvADC);
  float tensao = ((valorADC * 3.3) / 4095); // Convertendo para volts
  Serial.println(valorADC);
  return tensao;
}

void sendCORSHeaders() {
  server.sendHeader("Access-Control-Allow-Origin", "*");
  server.sendHeader("Access-Control-Allow-Methods", "GET, POST,
OPTIONS");
  server.sendHeader("Access-Control-Allow-Headers", "*");
}

void vADCTask(void *pvParameters) {
  while (1) {
    if (updatingData) {
      float vADCvalue = readvADCValue();
      vADCBuffer[bufferIndex] = vADCvalue;
      bufferIndex = (bufferIndex + 1) % bufferSize;
      vTaskDelay(acquisitionRate);
    } else {
      vTaskDelay(5 / portTICK_PERIOD_MS);
    }
  }
}

void startAcquisition() {
  updatingData = true;
  Serial.println("Aquisição de dados iniciada.");
```

```
}

void stopAcquisition() {
  updatingData = false;
  Serial.println("Aquisição de dados parada.");
}

void clearBuffer() {
  memset(vADCBuffer, 0, sizeof(vADCBuffer));  // Limpa o buffer
  bufferIndex = 0;
  Serial.println("Buffer limpo.");
}

void setup() {
  pinMode(26, OUTPUT);
  pinMode(pinvADC, INPUT);
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando ao WiFi...");
  }

  Serial.println("Conectado ao WiFi");
  Serial.print("Endereço IP: ");
  Serial.println(WiFi.localIP());

  server.on("/", HTTP_OPTIONS, []() {
    sendCORSHeaders();
    server.send(200);
  });

  server.on("/vADCvalue", HTTP_GET, []() {
    sendCORSHeaders();
    float value = readvADCValue();
    server.send(200, "text/plain", String(value, 4)); // Precisão de 4
casas decimais
  });

  server.on("/startAcquisition", HTTP_GET, []() {
    sendCORSHeaders();
    startAcquisition();
```

```
    server.send(200, "text/plain", "Aquisição iniciada");
  });

  server.on("/stopAcquisition", HTTP_GET, []() {
    sendCORSHeaders();
    stopAcquisition();
    server.send(200, "text/plain", "Aquisição parada");
  });

  server.on("/clearBuffer", HTTP_GET, []() {
    sendCORSHeaders();
    clearBuffer();
    server.send(200, "text/plain", "Buffer limpo");
  });

  server.begin();
  Serial.println("Servidor iniciado");

  xTaskCreate(vADCTask, "vADCTask", 2048, NULL, 1, &vADCTaskHandle);
}

void loop() {
  server.handleClient();
}
```

- **Aplicativo do Celular**

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'data_acquisition_state.dart';
import 'chart_widget.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
```

```
    return ChangeNotifierProvider(
      create: (context) => DataAcquisitionState('192.168.3.20'), //
Substitua pelo IP do seu ESP32
      child: MaterialApp(
        title: 'Sistemas de aquisição de dados',
        theme: ThemeData(
          brightness: Brightness.light, // Mudar para tema claro
          primaryColor: Colors.blue,
          textTheme: Theme.of(context).textTheme.apply(
                bodyColor: Colors.black,
                displayColor: Colors.black,
              ),
        ),
        home: const MyHomePage(),
      ),
    );
  }
}

class MyHomePage extends StatelessWidget {
  const MyHomePage({super.key});

  @override
  Widget build(BuildContext context) {
    final state = Provider.of<DataAcquisitionState>(context);

    return Scaffold(
      appBar: AppBar(
        title: const Center(child: Text('Sistemas de aquisição de
dados')),
      ),
      body: Center(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              // Adicionar a imagem com tamanho ajustado
              Image.asset(
                'assets/images/45 Anos UFU-19.png', // Atualizado para
o novo caminho da imagem
                width: 200, // Defina a largura desejada
                height: 200, // Defina a altura desejada
              ),
```

```
              const SizedBox(height: 20),
              const ChartWidget(),
              const SizedBox(height: 20),
              // Título para o Slider de escala de tempo
              const Text(
                'Ajuste da Escala de Tempo',
                style: TextStyle(fontSize: 16),
              ),
              // Slider para ajustar a escala de tempo
              Slider(
                value: state.timeScale.toDouble(),
                min: 1,
                max: 100, // Defina um valor máximo adequado para a
escala de tempo
                divisions: 100,
                label: state.timeScale.toString(),
                onChanged: (value) {
                  state.updateTimeScale(value.toInt());
                },
              ),
              const SizedBox(height: 20),
              // Título para o Slider de velocidade de aquisição
              const Text(
                'Velocidade de Aquisição (ms)',
                style: TextStyle(fontSize: 16),
              ),
              // Novo Slider para ajustar o atraso do fetchData
              Slider(
                value: state.fetchDelay.toDouble(),
                min: 1,
                max: 1000, // Ajuste conforme necessário para controlar
a velocidade
                divisions: 100,
                label: state.fetchDelay.toString(),
                onChanged: (value) {
                  state.updateFetchDelay(value.toInt());
                },
              ),
              const SizedBox(height: 20),
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceAround,
                children: [
                  ElevatedButton(
```

```
                    onPressed: () {
                      debugPrint("Start button pressed");
                      state.startAcquisition();
                    },
                    child: const Text('Iniciar'),
                  ),
                  ElevatedButton(
                    onPressed: () {
                      debugPrint("Stop button pressed");
                      state.stopAcquisition();
                    },
                    child: const Text('Parar'),
                  ),
                ],
              ),
              const SizedBox(height: 20),
              if (state.isAcquiring) ...[
                const Text('Adquirindo Dados...', style:
TextStyle(fontSize: 16)),
              ],
              const SizedBox(height: 20),
              // Container para mostrar o valor atual da aquisição
              Container(
                padding: const EdgeInsets.all(16.0),
                decoration: BoxDecoration(
                  color: Colors.white,
                  borderRadius: BorderRadius.circular(10.0),
                  border: Border.all(color: Colors.black),
                ),
                child: Text(
                  'Valor Atual:
${state.currentValue.toStringAsFixed(4)} V',
                  style: const TextStyle(fontSize: 24, color:
Colors.black),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

class DataAcquisitionState with ChangeNotifier {
  final String esp32Ip;
  List<double> dataPoints = [];
  double currentValue = 0.0;
  int timeScale = 50; // Escala de tempo
  int fetchDelay = 10; // Novo atributo para atraso do fetchData
  bool isAcquiring = false;

  DataAcquisitionState(this.esp32Ip);

  void startAcquisition() async {
    if (!isAcquiring) {
      isAcquiring = true;
      notifyListeners();
      debugPrint("Starting data acquisition...");
      try {
        final response = await
http.get(Uri.parse('http://$esp32Ip/startAcquisition'));
        if (response.statusCode == 200) {
          debugPrint("Acquisition started successfully.");
          _fetchData();
        } else {
          debugPrint("Failed to start acquisition:
${response.statusCode}");
        }
      } catch (e) {
        debugPrint("Error starting acquisition: $e");
      }
    }
  }

  void stopAcquisition() async {
    if (isAcquiring) {
      isAcquiring = false;
      debugPrint("Stopping data acquisition...");
      try {
        final response = await
http.get(Uri.parse('http://$esp32Ip/stopAcquisition'));
        if (response.statusCode == 200) {
          debugPrint("Acquisition stopped successfully.");
```

```
        } else {
          debugPrint("Failed to stop acquisition:
${response.statusCode}");
        }
      } catch (e) {
        debugPrint("Error stopping acquisition: $e");
      }
      notifyListeners();
    }
  }

  // Atualiza a escala de tempo
  void updateTimeScale(int scale) {
    timeScale = scale;
    notifyListeners();
  }

  // Atualiza o atraso do fetchData
  void updateFetchDelay(int delay) {
    fetchDelay = delay;
    notifyListeners();
  }

  void _fetchData() async {
    while (isAcquiring) {
      try {
        final response = await
http.get(Uri.parse('http://$esp32Ip/vADCvalue'));
        if (response.statusCode == 200) {
          currentValue = double.parse(response.body);
          if (dataPoints.length >= 100) {
            dataPoints.removeAt(0);
          }
          dataPoints.add(currentValue);
          debugPrint("Fetched value: $currentValue");
          notifyListeners();
        } else {
          debugPrint("Failed to fetch data. Status code:
${response.statusCode}");
        }
      } catch (e) {
        debugPrint("Error fetching data: $e");
      }
```

```
      await Future.delayed(Duration(milliseconds: fetchDelay)); //
Usando o novo atraso
    }
  }
}

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:fl_chart/fl_chart.dart';
import 'data_acquisition_state.dart';

class ChartWidget extends StatelessWidget {
  const ChartWidget({super.key});

  @override
  Widget build(BuildContext context) {
    final state = Provider.of<DataAcquisitionState>(context);

    return SizedBox(
      height: 300,
      width: double.infinity,
      child: LineChart(
        LineChartData(
          minY: 0,
          maxY: 3.3,
          minX: 0,
          maxX: state.timeScale.toDouble(), // Usando a escala de tempo
          titlesData: FlTitlesData(
            leftTitles: AxisTitles(
              sideTitles: SideTitles(
                showTitles: true,
                reservedSize: 40,
                getTitlesWidget: (value, meta) {
                  return Text(
                    value.toStringAsFixed(1),
                    style: const TextStyle(color: Colors.black,
fontSize: 12),
                  );
                },
              ),
            ),
            bottomTitles: const AxisTitles(
              sideTitles: SideTitles(
```

```
            showTitles: false, // Escondendo os valores do eixo X
          ),
        ),
        topTitles: const AxisTitles(
          sideTitles: SideTitles(
            showTitles: false,
          ),
        ),
        rightTitles: const AxisTitles(
          sideTitles: SideTitles(
            showTitles: false,
          ),
        ),
      ),
      gridData: FlGridData(
        show: true,
        drawVerticalLine: true,
        getDrawingHorizontalLine: (value) {
          return const FlLine(
            color: Colors.black12,
            strokeWidth: 1,
          );
        },
        getDrawingVerticalLine: (value) {
          return const FlLine(
            color: Colors.black12,
            strokeWidth: 1,
          );
        },
      ),
      borderData: FlBorderData(
        show: true,
        border: Border.all(color: Colors.black12, width: 1),
      ),
      lineBarsData: [
        LineChartBarData(
          spots: state.dataPoints
              .asMap()
              .entries
              .map((e) => FlSpot(e.key.toDouble(), e.value))
              .toList(),
          color: Colors.lightBlueAccent,
          barWidth: 2,
```

```
              dotData: const FlDotData(show: false),
            ),
          ],
        ),
      ),
    );
  }
}
```