Sistema de Monitoramento Remoto da Temperatura de Ninhos de uma Ave Símbolo da Zona Costeira do Sul do Brasil

Luís Vitor Vaz de Mattos
Universidade Estadual do Rio Grande do Sul - UERGS Guaíba / Brasil
luis-mattos@uergs.edu.br

Fernando de Souza Oliveira
Universidade Estadual do Rio Grande do Sul - UERGS Guaíba / Brasil
fernando-oliveira@uergs.edu.br

Celso Maciel da Costa
Universidade Estadual do Rio Grande do Sul - UERGS Guaíba / Brasil
celso-costa@uergs.edu.br

Letícia Vieira Guimarães
Universidade Estadual do Rio Grande do Sul - UERGS Guaíba / Brasil
leticia-guimaraes@uergs.edu.br

Paulo Henrique Ott
Universidade Estadual do Rio Grande do Sul - UERGS Litoral Norte-Osório / Brasil
paulo-ott@uergs.edu.br

ABSTRACT

This paper details the design and development of a bird monitoring system, focusing on the study of bird breeding. The project encompasses the entire development lifecycle: selecting components, assembling the hardware, programming the ESP32 microcontroller, and developing a web-based data visualization interface. The methodology begins with component selection, including the design of a custom temperature sensor, as well as considerations for connectivity and energy efficiency essential for remote deployments. Programming the ESP32 involves configuring modules, implementing time-stamped data logging, and optimizing for low-power operation. The final phase focuses on creating a user-friendly web interface using HTML, CSS, and JavaScript, providing a seamless experience for data access and visualization. This system offers a practical, end-to-end solution for ecological monitoring, enabling researchers to track environmental parameters in real-time and over extended periods to support habitat preservation effort

CCS CONCEPTS

Computer systems organization. Embedded and cyber-physcal systems. Embedded systems

KEYWORDS

Environmental monitoring, IoT (Internet of Things), DataLogger, Remote sensing, Microprocessor Esp32.

1. Introdução

Diante da crescente preocupação com a preservação da biodiversidade e dos impactos negativos das atividades humanas no meio ambiente, sistemas baseados em IoT (Internet Of Things) têm sido amplamente utilizados para monitoramento ambiental [1, 2, 3, 4, 5]. Nesse contexto, o monitoramento de espécies de aves costeiras se tornou uma ferramenta essencial para apoiar a conservação ambiental. Na costa do Rio Grande do Sul. Uma ave de grande importância ecológica, o piru-piru (*Haematopus palliatus*), vem sendo afetada por mudanças no habitat e pela pressão das atividades humanas[6]. O piru-piru tem uma distribuição tipicamente costeira, sendo fundamental a existência de ambientes com dunas preservadas para nidificação e desenvolvimento dos filhotes[7]. Nesse contexto, o entendimento do comportamento da ave no período de reprodução é de suma importância para estabelecer medidas de manejo que garantam condições favoráveis à sobrevivência e ao sucesso reprodutivo da espécie, contribuindo para sua conservação em longo prazo.

O sistema desenvolvido coleta a temperatura dos ninhos por meio de sensores posicionados no solo, próximos aos ninhos, e integra esses dados com informações de data e hora, obtidas pelo microcontrolador ESP32. Esses dados são armazenados de forma permanente em um arquivo na memória flash. Ao término do período de coleta, as informações são enviadas para o banco de dados para posterior visualização. Este artigo descreve o desenvolvimento do sistema de monitoramento, abordando desde a seleção e montagem dos componentes — incluindo os sensores de temperatura e outros módulos — até a programação da ESP32 e o desenvolvimento de uma interface gráfica para visualização dos dados em uma aplicação web.

2. Descrição do sistema

Utilizando conceitos de IoT[12,13], o sistema foi projetado para estudar aves costeiras, especificamente o piru-piru, na costa do Rio Grande do Sul, ao longo de vários dias. O sistema possui um

microcontrolador ESP32, um sensor de temperatura, um servidor na nuvem que acessa um banco de dados Firebase. Os dados coletados são enviados ao servidor que os armazena no Firebase, de onde podem ser recuperados e exibidos por um cliente web. A Figura 1 mostra uma visão geral do sistema.

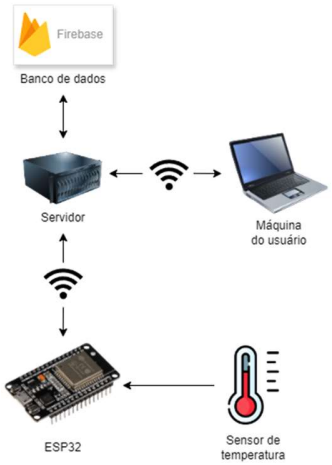

Figura 1: Visão geral do sistema

Uma estação de coleta de dados, DataLogger, foi desenvolvida para monitorar a temperatura do solo no ninho da ave, que é caracterizado por uma pequena depressão nas dunas costeiras e contém, usualmente, dois ovos no período de incubação (Figura 2). A cada intervalo programado (atualmente 15 minutos), o DataLogger registra a temperatura do solo, juntamente com a data e a hora, armazenando essas informações em um arquivo na memória flash da ESP32.

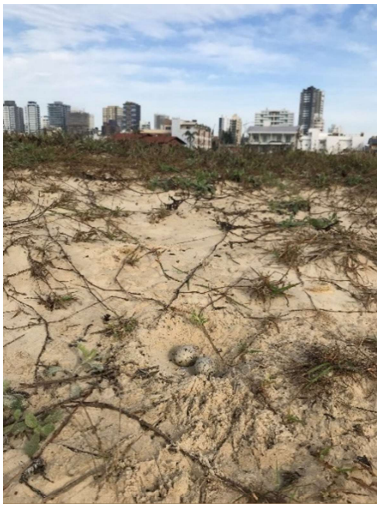

Figura 2 – Ninho de piru-piru (*Haematopus palliatus*).

Ao final do período de coleta, definido pela duração da bateria (limitado a dois dias nos testes realizados), a estação é retirada dos ninhos e levada para um local com conexão WiFi. Conectado à rede, o sistema lê os dados armazenados na memória Flash e os envia a um servidor central, onde são guardados em um banco de dados para processamento e análise posterior.

Os dados armazenados podem ser acessados por meio de um cliente web, que permite visualizar a temperatura no ninho, bem como a data e a hora de cada registro realizado ao longo do período de monitoramento. A interface web é intuitiva e facilita a visualização dos dados organizados por data, hora e temperatura.

A interface web é implementada em JavaScript, que consulta o banco de dados Firebase para verificar e recuperar os dados disponíveis. Uma vez recuperados, os dados são exibidos de forma dinâmica na página, atualizando-se conforme novas leituras são obtidas. Essa visualização permite acompanhar o número total de leituras e analisar a variação de temperatura no ninho ao longo do período de coleta, possibilitando comparações ao longo dos dias de monitoramento.

3. Detalhamento do Hardware desenvolvido

Foi desenvolvida uma placa protótipo, Figura 3 , utilizando o microcontrolador ESP32 como núcleo principal, alimentada por uma bateria recarregável, para possibilitar operações em locais remotos. A placa integra um sensor de temperatura para monitoramento ambiental, além de chaves liga/desliga para controlar o consumo de energia e facilitar a operação manual. Quando o carregador está conectado, o circuito permanece inativo, sendo ativado somente quando a chave seletora é pressionada. Se o carregador não estiver conectado, o circuito permanece totalmente desligado, uma medida adotada para evitar problemas durante o transporte das placas. Ao pressionar a chave seletora, o regulador de tensão é ativado, recebendo uma entrada de 16.8V e 2Ah da bateria, que é então convertida para 5V e 2A. Essa alimentação é distribuída para dois componentes principais: a ESP32, responsável pelo controle do sistema e pela comunicação via Wi-Fi e o sensor de temperatura, desenvolvido em laboratório, que monitora as condições ambientais. Após o regulador fornecer a tensão correta, a ESP32 inicia seu ciclo de operação, aguardando 10 segundos para realizar a primeira leitura da temperatura. Quando essa leitura ocorre e o sistema está configurado corretamente, um LED na ESP32 pisca rapidamente, indicando que o ciclo foi iniciado. Após a leitura da temperatura, a data e a hora são obtidas da própria ESP32, que entra no modo light *sleep*, um estado de baixo consumo de energia programado especificamente para este projeto. Nesse modo, a ESP32 permanece por cerca de 15 minutos, reiniciando

em seguida para repetir o ciclo de leitura do sensor de temperatura e obtenção da data/hora. Este ciclo se repete até que seja atingido o número predefinido de dados a serem coletados. Após alcançar a quantidade necessária de dados, a ESP32 interrompe o ciclo de *light sleep* e começa a tentar enviar os dados para o servidor. Caso não consiga estabelecer conexão com a rede Wi-Fi após dez tentativas, o sistema retorna à coleta de dados. Se a conexão for bem-sucedida, os dados são enviados ao banco de dados. Caso o envio falhe devido a um sinal fraco ou outro problema de conexão, o sistema aguarda e tenta novamente.

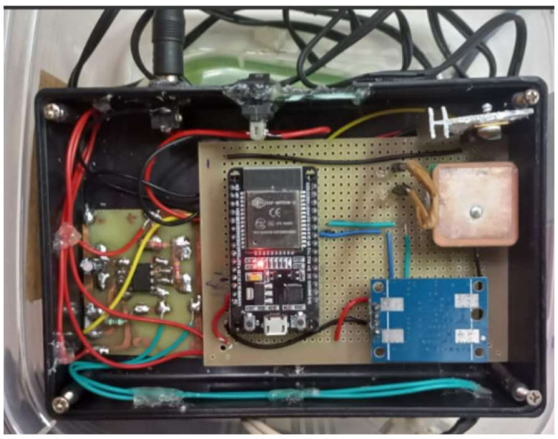

Figura 3: Placa desenvolvida

O desenvolvimento do sensor de temperatura se baseou na utilização de um circuito integrado 555, atuando como oscilador no modo estável. O sinal gerado pelo C.I. 555 é uma onda quadrada com período $T = T_1 + T_2$ onde T_1 é o tempo em que a tensão é igual a alimentação e T_2 o tempo em que o sinal é igual a zero. Desta forma a razão $\frac{T_1}{T_2} \times 100\%$ é denominada duty cycle. O duty cycle se modifica com a alteração da tensão no pino de controle, desta forma é possível obter a alteração do duty cycle conectando um termistor no pino de controle e assim relacionar o duty cicle à temperatura a qual o termistor estiver submetido. Dessa maneira, observando a variação do duty cycle conforme a modificação da temperatura sobre o termistor foi possível modelar este comportamento criar uma função genérica D(T), a qual relaciona um valor de temperatura a uma taxa de duty cycle.

Portanto, foi possível desenvolver um software a ser integrado em um micro controlador (nesse caso, o ESP32), que tenha como propósito a obtenção de um valor de temperatura aproximado daquele ao qual o termistor está sendo submetido.

O código apresentado a seguir, na Figura 4, mostra a função "lerTemperatura()" responsável por computar os tempos de alto e baixo da onda quadrada gerada pelo C.I. 555, através de diversas medidas, armazená-los em um vetor global, e chamar a função "dutyCycle()".

```
#define PINO_SENSOR 7
#define N_AMOSTRAL 200
int  coletasRealizadas = 0;
float vetorOn[N_AMOSTRAL] = { 0 }, vetorOff[N_AMOSTRAL] = { 0 };

void lerTemperatura() {
  bool coletaEmAndamento = true;
  int contadorAmostras = 0, tempoAlto = 0, tempoBaixo = 0;

  estadoPino = digitalRead(PINO_SENSOR);
  while (coletaEmAndamento) {
    // Captura o tempo de onda alta
    while (estadoPino == HIGH) {
      tempoAlto += 1;
      estadoPino = digitalRead(PINO_SENSOR);
    }

    // Captura o tempo de onda alta
    while (estadoPino == LOW) {
      tempoBaixo += 1;
      estadoPino = digitalRead(PINO_SENSOR);
    }

    tempoOn[contadorAmostras] = tempoAlto;
    tempoOff[contadorAmostras] = tempoBaixo;

    contadorAmostras++;
    if(contadosAmostras == N_AMOSTRAL) {
      dutyCycle();
      coletasRealizadas++;
      contadorAmostras = 0;
      coletaEmAndamento = false;
    }
  }
}
```

Figura 4: Função de cômputo dos tempos da onda

A função "dutyCycle()" (Figura 5), ao ser chamada, utiliza os tempos coletados pela função "lerTemperatura()", resume-os em uma média e utiliza essa média nas funções obtidas através da correlação anteriormente explicada. Como resultado da função, é possível obter um valor em ºC de temperatura através da medição do duty cycle da onda de saída do C.I. 555.

```
void dutyCycle() {
  float dutyCycle = 0, tempoAltoMedio = 0, somaTempoAlto = 0;
  float tempoBaixoMedio = 0, somaTempoBaixo = 0, temperatura = 0;

  for (int i = 0; i < N_AMOSTRAL; i++) {
    somaTempoAlto += vetorOn[i];
  }
  tempoAltoMedio = somaTempoAlto / N_AMOSTRAL;

  for (int i = 0; i < N_AMOSTRAL; i++) {
    somaTempoBaixo += vetorOff[i];
  }
  tempoBaixoMedio = somaTempoBaixo / N_AMOSTRAL;

  dutyCycle = tempoAltoMedio / tempoBaixoMedio;
  if (dutyCycle < 1.6) {
    temperatura = (dutyCycle - 1.0182) / 0.0119;
  } else {
    temperatura = (dutyCycle + 0.45) / 0.04;
  }

  // Processamento da temperatura obtida
  ...

  delay(100);
}
```

Figura 5: Código da função duty cycle

4. TRABALHOS RELACIONADOS

Zhang et al. (2021) abordam um sistema IoT projetado para monitoramento da qualidade da água em tempo real, aplicável à proteção ambiental. Utilizando sensores conectados a um sistema central, o estudo explora a captura e transmissão contínua de parâmetros de qualidade da água, como temperatura e pH. Os dados são enviados para um servidor na nuvem, permitindo acesso remoto e análise para identificação rápida de mudanças ambientais que possam afetar o ecossistema aquático. Esse estudo destaca a importância da integração de sensores para capturar dados ambientais em intervalos frequentes, semelhante ao uso de um DataLogger para monitoramento de temperatura ao longo do tempo.

Bose et al. (2021) apresentam um sistema IoT voltado para monitoramento ambiental em florestas e proteção de vida selvagem, com coleta de temperatura, umidade e movimento. O sistema usa um DataLogger para armazenar os dados temporariamente, enquanto a transmissão é feita em intervalos determinados. Este sistema permite o acompanhamento do microclima da floresta, fornecendo informações que ajudam na gestão de habitats naturais. Esse artigo tem relevância direta ao projeto, uma vez que explora uma estrutura similar, com sensores coletando dados em períodos definidos e armazenando-os para envio posterior, garantindo uma coleta consistente e segura.

Mishra et al. (2022) exploram o desenvolvimento de um sistema de monitoramento da qualidade do ar em tempo real, projetado para aplicação em cidades inteligentes e sustentáveis. O sistema coleta dados de poluentes atmosféricos, como dióxido de carbono e partículas finas, além de informações sobre temperatura e umidade. Estes dados são armazenados em um servidor de nuvem, permitindo a análise de tendências e alertas de poluição em tempo real. A similaridade com o projeto do DataLogger está na coleta e transmissão regular de dados ambientais, além da utilização de uma rede de sensores para capturar as mudanças no ambiente ao longo do tempo.

Gupta et al. (2023) apresentam uma revisão do uso de sistemas IoT combinados com inteligência artificial para monitoramento ambiental, focando em aplicações como agricultura e conservação de habitats. O artigo descreve a coleta de dados como temperatura, umidade e qualidade do solo, que são armazenados em nuvem e analisados por modelos de IA para identificar padrões e anomalias. Esse estudo é relevante para o projeto do DataLogger, pois sugere como técnicas de análise avançadas podem ser aplicadas a dados ambientais para melhorar a eficiência e precisão da coleta de dados, como ocorre no monitoramento de temperatura em ninhos de aves. O trabalho apresentado evidencia que o DataLogger compartilha com os outros sistemas o objetivo de monitoramento ambiental com armazenamento temporário dos dados, porém, difere na frequência de coleta (que é configurável) e na aplicação, que é voltada para conservação de aves costeiras. Um aspecto de importância é que o DataLogger armazena os dados em uma memória Flash, permanente, o que permite coleta de dados em áreas remotas, sendo adequado para cenários sem conectividade com a Internet. Outra característica relevante é a capacidade de definição dos intervalos de coleta de dados, que possibilita também o envio em tempo real para o servidor na nuvem. Trabalhos similares são também apresentados em [12, 13].

5 Estrutura e funcionamento do Sistema

O sistema de monitoramento desenvolvido inicia se conectando a uma rede wireless e realizando uma leitura inicial de hora, sincronizando-se com um servidor de horário na internet para obter a data e a hora exatas. Em seguida, o sistema configura e inicializa o sistema de arquivos LittleFS na ESP32, permitindo a criação e manipulação de arquivos na memória Flash do dispositivo.

Uma vez configurado o sistema de arquivos, o sistema começa a coletar leituras de temperatura, utilizando o sensor integrado. A cada leitura, o sistema obtém a hora local da ESP32 e registra as informações de data, mês, ano e temperatura no arquivo previamente criado na Flash. Após o registro dos dados, o sistema executa a função littlesleep, entrando em um modo de baixo consumo de energia pelo tempo especificado. Esse ciclo de coleta, registro e repouso se repete ao longo do período de monitoramento, otimizando o uso da bateria e garantindo a coleta contínua de dados ambientais.

Após o período de coleta, realizado em áreas remotas e sem conexão à internet, o DataLogger é transportado para um local onde a rede wireless inicial está disponível. Nesse ambiente com conexão, o sistema se conecta automaticamente à rede e executa a transmissão dos dados coletados e armazenados no arquivo da Flash para um banco de dados no Firebase, que foi configurado para armazenar permanentemente as informações.

O Firebase funciona como o repositório central dos dados, permitindo a consulta e análise das informações de temperatura registradas durante o período de monitoramento. Dessa forma, os usuários podem acessar os dados remotamente, visualizando e analisando as informações de temperatura coletadas ao longo do período em que o sistema

esteve em campo. A transmissão para o Firebase é realizada de forma automática, garantindo que todas as leituras sejam salvas e eliminando a necessidade de intervenção manual para envio dos dados.

Além disso, com os dados disponíveis no Firebase, é possível gerar relatórios e visualizações gráficas que facilitam o acompanhamento do ambiente monitorado.

O sistema foi projetado para operar de forma confiável em ambientes remotos, priorizando a economia de energia e a segurança dos dados. Ao final de cada período de coleta, o DataLogger retoma a conectividade e envia automaticamente as informações para o Firebase, criando um ciclo de coleta e transmissão de dados.

5.1 Detalhes da Implementação

A implementação do DataLogger na ESP32 é um programa de cerca de 300 linhas. As funções "setup()" e "loop()" do sistema são apresentadas a seguir (Figura 6).

```
void setup() {
  pinMode(PINO_SENSOR, INPUT);
  pinMode(LED_BUILT_IN, OUTPUT);
  Serial.begin(115200);
  initTime();
  iniciarLittleFS();
  delay(1000);
}

void loop() {
  firebase() ;
  callTime();
  delay(1000);
}
```

Figura 6: Implementação do DataLogger

O código apresentado implementa o sistema de monitoramento DataLogger, configurando e executando todas as etapas necessárias para a coleta, armazenamento e transmissão de dados.

Na função "setup()", executada uma única vez no início, o sistema inicializa e configura o ambiente. Primeiramente, define os modos dos pinos: o pino conectado ao sensor é configurado como entrada para permitir a leitura dos dados de temperatura, enquanto o pino LED_BUILT_IN é configurado como saída, para ser usado como um indicador de status. Em seguida, o sistema configura a comunicação serial para 115200 bps, enviando mensagens de depuração para o console para indicar que o DataLogger foi iniciado, junto com a versão atual do programa.

A próxima etapa no "setup()" envolve a conexão à rede Wi-Fi, necessária para sincronizar o relógio da ESP32 e transmitir os dados coletados. Uma vez conectada, a função "initTime()" sincroniza o relógio interno da ESP32 com um servidor de hora na internet, assegurando que os registros de dados terão uma marcação precisa de data e hora. Após a sincronização do horário, a função "initLittleFS()" inicializa o sistema de arquivos LittleFS na ESP32, permitindo que os dados de data/hora e temperatura sejam armazenados na memória Flash. Um atraso de 1 segundo (delay(1000)) é então adicionado para estabilizar o sistema antes do início do "loop()".

A função "loop()" é executada continuamente, realizando as tarefas principais do sistema. Em cada iteração a função "firebase()" envia os dados coletados para o banco de dados Firebase, onde ficam armazenados de maneira permanente para acesso e análise posteriores. Em seguida, a função "callTime()" é executada. Essa função chama a função que realiza a leitura do sensor de temperatura e obtém a hora atual da ESP32, que é associada a temperatura lida. Essas informações são gravadas no arquivo na Flash. Um atraso de 1 segundo é adicionado ao final de cada iteração, permitindo que o sistema opere de forma estável e repetitiva.

A função "firebase()" (Figura 7) realiza o envio de dados coletados e armazenados na memória Flash da ESP32 para um banco de dados no Firebase. A função "firebase()" gerencia o envio de dados coletados pela ESP32 para um banco de dados no Firebase, centralizando o processo de conectividade, leitura e formatação dos dados, além de lidar com a verificação de erros e limpeza de dados após o envio. Primeiramente, ela verifica se a ESP32 está conectada à rede Wi-Fi; caso não esteja, tenta se reconectar. Se a conexão falhar, a função exibe uma mensagem informativa e interrompe a execução, evitando tentativas de envio sem conectividade.

Em seguida, configura o acesso ao Firebase, definindo as credenciais e o host, e aguarda um curto período para garantir que a conexão seja estabelecida. A função verifica se o Firebase está pronto para uso e, em caso positivo, abre o arquivo dados.bin armazenado na memória Flash (LittleFS) da ESP32, onde os dados coletados de temperatura, data e hora foram gravados anteriormente. Esse arquivo é lido sequencialmente, com cada registro sendo extraído e formatado em JSON. Cada conjunto de dados é então enviado ao Firebase, que exibe uma confirmação de sucesso para cada registro transmitido. Caso ocorra algum erro durante o envio, a função sinaliza a falha e registra o erro.

Após o envio de todos os dados, se não houver falhas, a função "limparArquivo()" é chamada para esvaziar o arquivo dados.bin, garantindo que não haja duplicação de dados no próximo ciclo de coleta. Essa função de limpeza abre o arquivo em modo de escrita e fecha-o imediatamente, o que apaga seu conteúdo. Esse processo assegura que a ESP32 mantenha os dados atualizados e sincronizados no Firebase, além de preservar o espaço de armazenamento na Flash, pronto para novas coletas. A função "firebase()" realiza, assim, um gerenciamento robusto dos dados, desde a conectividade até a persistência segura no banco de dados remoto.

```
void firebase() {
  bool erro = false;  // Inicializa a variável erro como false

  // Verifica conexão WiFi e tenta reconectar, se necessário
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Tentando conectar ao WiFi.");
    wifi();
  }
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Impossível conectar ao Firebase.");
    return;
  }

  // Configurações do Firebase
  config.host = FIREBASE_HOST;
  config.signer.tokens.legacy_token = FIREBASE_AUTH;

  // Inicializa o Firebase
  Firebase.begin(&config, &auth);
  delay(3000);  // Aguarda para dar tempo de conexão

  // Verifica se o Firebase está pronto
  if (!Firebase.ready()) {
    Serial.println("Firebase não está pronto para uso.");
    return;
  }

  // Abre o arquivo para leitura
  File file = LittleFS.open("/dados.bin", FILE_READ);
  if (!file) {
    Serial.println("Erro ao abrir arquivo para leitura.");
    return;
  }

  int indice = 0;
  while (file.available()) {
    // Lê dados do arquivo
    file.read((uint8_t*)&dados_finais[indice], sizeof(Data));

    // Formata a data e a hora
    char dateString[9];
    char timeString[9];
    snprintf(dateString, sizeof(dateString), "%02d/%02d/%02d",
             dados_finais[indice].dia,
             dados_finais[indice].mes,
             dados_finais[indice].ano);

    snprintf(timeString, sizeof(timeString), "%02d:%02d:%02d",
             dados_finais[indice].hora,
             dados_finais[indice].minuto,
             dados_finais[indice].segundo);

    // Prepara o JSON
    json.set("data", String(dateString));
    json.set("hora", String(timeString));
    json.set("temperatura", dados_finais[indice].temperatura);

    // Envia dados ao Firebase
    if (Firebase.pushJSON(firebaseData, FIREBASE_PATH, json)) {
      Serial.println("Dados enviados com sucesso.");
    } else {
      Serial.printf("Erro ao enviar dados: %s\n", firebaseData.errorReason().c_str());
      erro = true;  // Falha no envio
    }

    indice++;
    delay(2000);  // Intervalo entre envios
  }

  // Limpa o arquivo após envio bem-sucedido
  if (!erro) {
    limparArquivo();
    Serial.println("Arquivo limpo após envio ao Firebase.");
  }

  // Fecha o arquivo
  file.close();

  // Função para limpar o arquivo após enviar os dados
  void limparArquivo() {
    // Abre o arquivo em modo de escrita, o que limpa o conteúdo automaticamente
    File file = LittleFS.open("/dados.bin", FILE_WRITE);
    if (!file) {
      Serial.println("Erro ao abrir o arquivo para limpeza.");
      return;
    }

    // Fecha o arquivo imediatamente após abrir para garantir que fique vazio
    file.close();
    Serial.println("Arquivo limpo após envio para o BD.");
  }
}
```

Figura 7: Implementação da função "firebase()"

A função "callTempTime()" (Figura 8), desenvolve uma função fundamental no sistema: realiza a coleta e o armazenamento de dados de temperatura e horário em um arquivo no sistema de arquivos LittleFS da ESP32. Essa função é responsável por captar os dados em intervalos específicos e armazená-los para posterior envio ao Firebase.

```
void callTempTime() {
  struct tm timeinfo;

  // Abre o arquivo para anexar dados
  File file = LittleFS.open("/dados.bin", FILE_APPEND);
  if (!file) {
    Serial.println("Erro ao abrir arquivo para escrever.");
    return;
  }

  for (int i = 0; i < TAM_BUFFER; i++) {
    // Obtém a temperatura
    temperatura();

    // Obtém o horário local
    if (getLocalTime(&timeinfo)) {
      // Armazena os dados no buffer
      dados_finais[i].ano = timeinfo.tm_year - 100;
      dados_finais[i].mes = timeinfo.tm_mon + 1;
      dados_finais[i].dia = timeinfo.tm_mday;
      dados_finais[i].hora = timeinfo.tm_hour;
      dados_finais[i].minuto = timeinfo.tm_min;
      dados_finais[i].segundo = timeinfo.tm_sec;

      // Escreve os dados no arquivo
      file.write((uint8_t*)&dados_finais[i], sizeof(Data));
      file.flush();  // Garante que os dados sejam gravados no
armazenamento
    } else {
      Serial.println("Erro ao obter a hora local.");
    }

    light_sleep();  // Coloca o dispositivo em modo de baixo
consumo
  }

  // Fecha o arquivo após a gravação
  file.close();
  Serial.println("Dados registrados com sucesso.");
}
```

Figura 8: Implementação da função "callTempTime()"

A função começa com a estrutura tm, que armazena as informações de tempo e data. Em seguida, abre o arquivo dados.bin no sistema LittleFS no modo de escrita. Caso o arquivo não seja aberto corretamente, a função encerra a execução. O loop principal da função itera até atingir o tamanho máximo do buffer (TAM_BUFFER), captando a temperatura e o horário local a cada ciclo.

Dentro do loop, a função chama "temperatura()"lê a temperatura atual do sensor. Em seguida, utiliza "getLocalTime(&timeinfo)" para obter o horário local. Se a função retornar sucesso, os dados de ano, mês, dia, hora, minuto e segundo são extraídos da estrutura timeinfo e armazenados em uma estrutura de dados denominada "dados_finais", na posição *i*, que representa o número da iteraçãoApós formatar os dados, a função grava a estrutura "dados_finais[i]" no arquivo dados.bin, usando "file.write()", seguido de "file.flush()" para garantir que a escrita seja concluída.

A função ainda chama "light_sleep()" após cada coleta de dados, colocando a ESP32 em modo de baixo consumo até o

próximo ciclo de coleta. Ao final do loop o arquivo é fechado com "file.close()". Isso completa o ciclo de coleta e armazenamento dos dados de temperatura e hora na Flash, prontos para sincronização com um servidor quando houver conectividade.

5.2. Desenvolvimento da Interface Gráfica

O desenvolvimento da interface gráfica foi realizado utilizando HTML e CSS. A construção foi pensada para oferecer simplicidade e clareza, proporcionando aos usuários uma experiência intuitiva, especialmente na tela inicial onde é feita a seleção dos dados de temperatura de cada estação (Figura 9). Essa abordagem buscou minimizar a complexidade para que os usuários, mesmo sem experiência técnica, conseguissem utilizar o sistema facilmente.

Figura 9: Tela inicial de seleção do município para visualização da data/hora e temperatura

JavaScript foi utilizado para acesso ao banco de dados Firebase e para exibição das informações. Este recurso permitiu que data/hora e temperatura pudessem ser recuperadas nos intervalos programados, podendo ser em tempo real, convertidos para o formato adequado e apresentados de maneira clara na interface (Figura 10).

Uma das funcionalidades mais importantes da interface foi a capacidade de recuperar os dados diretamente do Firebase e utilizá-los para alimentar o sistema de visualização (Figura 10). Assim, os usuários podem visualizar os dados ao final do intervalo de coleta programado, quando o sistema se conectar à rede wireless previamente definida.

Dados de Temperatura de Mostardas

ID	Data	Hora	Temperatura(°C)
0	03/11/24	19:40:23	26.9421
1	03/11/24	19:41:25	27.68389
2	03/11/24	19:42:28	27.34431
3	03/11/24	19:43:30	27.27847
4	03/11/24	19:44:32	27.20003
5	03/11/24	19:45:50	27.49767
6	03/11/24	19:46:52	27.91371
7	03/11/24	19:47:54	27.33508
8	03/11/24	19:48:50	27.99185
9	03/11/24	19:49:53	26.94905

Figura 10: Dados do Munícipio de Mostardas

6. Análise dos Resultados

Na análise dos resultados, aspectos importantes foram avaliados para garantir a funcionalidade e a eficiência do sistema. Em relação ao consumo de energia, nos testes realizados, a bateria utilizada mostrou ser capaz de sustentar operações contínuas por até dois dias, desde que mantida em boas condições de conservação. No entanto, situações em que a bateria é descarregada completamente a cada ciclo de coleta de dados podem reduzir sua vida útil, comprometendo o tempo de operação esperado.

Outro ponto importante identificado foi o impacto das tentativas repetitivas de conexão em áreas com sinal Wi-Fi fraco. Esse cenário aumenta significativamente o consumo de energia, uma vez que cada tentativa exige um maior fluxo de corrente, gerando aquecimento no módulo de conexão. Esse aquecimento constante pode, com o tempo, criar um ciclo de desgaste acelerado tanto para a bateria quanto para os demais componentes do sistema, especialmente em condições de uso intensivo ou em ambientes onde a conectividade é intermitente.

A interface gráfica é intuitiva, fácil de usar e permite aos usuários uma rápida leitura das informações, facilitando o processo de análise.

Outro aspecto relevante foi a necessidade de buscar soluções para diminuir o consumo de energia, visto que a duração da bateria é um ponto crítico para o funcionamento contínuo do sistema. Como a coleta dos dados de data/hora e temperatura ocorre em intervalos previamente definidos, como a cada 15 minutos, a alternativa escolhida para reduzir o consumo foi colocar o microcontrolador ESP32 em light sleep durante os períodos de inatividade. O light sleep é um modo de economia de energia oferecido pelo ESP32, no qual grande parte dos componentes do microcontrolador é desativada temporariamente, enquanto outros permanecem ativos para garantir a funcionalidade essencial. Nesse estado, o processador principal entra em suspensão, mas periféricos como o RTC (Real-Time Clock) e pinos configurados para interrupções podem continuar funcionando. O modo light sleep foi utilizado para manter o ESP32 em um estado de baixo consumo entre as coletas de dados. O dispositivo "acorda" apenas nos momentos pré-programados, realiza a leitura dos sensores e sincroniza informações com o servidor quando necessário. Esse comportamento reduz significativamente o consumo energético, já que o ESP32 consome muito menos corrente em light sleep em comparação ao modo ativo.

7. Conclusão

O desenvolvimento do sistema utilizando ESP32, um sensor de temperatura e uma interface web baseada em Firebase, mostrou-se uma solução viável e eficaz para monitorar a

temperatura dos ninhos de piru-piru. A pesquisa abordou vários aspectos críticos, incluindo a seleção e montagem de componentes, programação do microcontrolador, desenvolvimento de uma interface gráfica e testes, conectividade, eficiência energética e usabilidade.

Os resultados obtidos demonstraram que o sistema é capaz de coletar dados, com a energia sendo fornecida por uma bateria, durante o tempo previamente definido, e enviar esses dados para armazenamento no servidor, quando houver rede wireless disponível.

A interface gráfica criada é intuitiva e de fácil utilização, possibilitando ao usuário selecionar a estação que deseja visualizar os dados, que são apresentados sob a forma de uma tabela. O consumo de energia do sistema é um pouco crítico. Em operação, a duração da bateria foi, em média, de dois dias, o que leva a necessidade de recargas frequentes. Essa característica cria um problema para uso em cenários onde a manutenção constante não é viável, como em áreas remotas.

Trabalhos futuros incluem a integração de sensores adicionais para monitoramento ambiental (unidade, pressão, qualidade do ar) e uma avaliação precisa do consumo de bateria, ponto sensível em sistemas de monitoramento que são dispostos em áreas remotas. Com isso, o sistema poderá contribuir, ainda mais, na gestão sustentável dos recursos naturais.

REFERENCES

[1] ZHANG, Y.; LIU, X.; WANG, J. Environmental Monitoring Systems for Air Quality: IoT and Data Analytics. Environmental Science and Technology, v. 54, n. 2, p. 55-67, 2018.

[2] LOPEZ, P.; FERNANDEZ, M. Application of IoT in Environmental Monitoring for Climate Change Adaptation. Journal of Environmental Management, v. 232, p. 459-467, 2019.

[3] SINGH, A.; KUMAR, S.; MEHTA, N. IoT and Cloud Computing for Real-Time Environmental Monitoring Systems. International Journal of Environmental Technology and Management, v. 21, n. 4, p. 345-362, 2021.

[4] JONES, M.; DAVIS, J. Monitoring of Water Quality Using IoT: A New Approach for Environmental Protection. Environmental Monitoring and Assessment, v. 193, p. 210, 2021.

[5] KUMAR, P.; RANI, P.; SHARMA, A. IoT-Based Framework for Environmental Quality Monitoring in Urban Areas. Urban Environmental Management, v. 34, p. 215-230, 2021.

[6] CANABARRO, P.L., FEDRIZZI, C.E. Aspectos da reprodução do Piru-piru *Haematopus palliatus* (Charadriiformes: Haematopodidae) na Praia do Hermenegildo, Rio Grande do Sul, Brasil. Revista Brasileira de Ornitologia, v. 18, p. 249–255, 2010.

[7] LINHARES, B. A.; BORDIN, J.; NUNES, G. T.; OTT, P. H. Breeding biology of the American Oystercatcher *Haematopus palliatus* on a key-site for conservation in southern Brazil. Ornithology Research, v. 29, p.16–21, 2021.

[8] GUPTA, S.; SHARMA, R. Remote Sensing and IoT-based Environmental Monitoring for Sustainable Development. Sustainable Development and Innovations in Environmental Engineering, v. 15, p. 121-133, 2020.

[9] ZHANG, D., ZHENG, H., CHEN, T., & HE, X. (2021). IoT-Based Water Quality Monitoring System for Environmental Protection. Environmental Science and Pollution Research, 28(5), 1-12.

[10] BOSE, S., DAS, S., & BISWAS, S. (2021). IoT-Based Smart Forest: An Approach for Forest Environment Monitoring and Wildlife Protection. Journal of Ambient Intelligence and Humanized Computing, 12(4), 1-13.

[11] MISHRA, S., TRIPATHI, M., & VERMA, S. (2022). Air Quality Monitoring Using IoT and Cloud Computing for Smart City Applications. International Journal of Environmental Science and Technology, 19(3), 2531-2545.

[12] ALMEIDA, R.; COSTA, E.; PEREIRA, M. Internet of Things (IoT) for Environmental Monitoring: A Review of Applications and Challenges. Environmental Engineering & Management Journal, v. 18, n. 9, p. 1947-1960, 2019.

[13] RUIZ, M.; LOPEZ, J.; GUTIERREZ, F. A Comprehensive Review of IoT Applications in Environmental Monitoring and Disaster Management. International Journal of Environmental Research and Public Health, v. 19, n. 8, p. 1051, 2022.