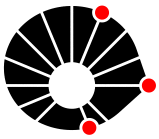
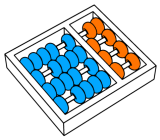




Internet das Coisas e visão computacional para monitoramento de colmeias e contagem de fluxo de abelhas

J. A. R. Feltran *J. M. de O. Guimarães* *R. L. M. de Sousa*
C. A. A. Trujillo *L. F. Bittencourt*

Relatório Técnico - IC-PFG-24-22
Projeto Final de Graduação
2024 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Internet das Coisas e visão computacional para monitoramento de colmeias e contagem de fluxo de abelhas

João A. R. Feltran* João M. de O. Guimarães† Renan L. M. de Sousa‡
Carlos A. A. Trujillo§ Luiz F. Bittencourt¶

Resumo

Este trabalho é um relatório realizado como Projeto Final de Graduação do Instituto de Computação, em parceria com o Prof. Roberto Greco do Instituto de Geociências. O objetivo do projeto é desenvolver um sistema para a coleta e processamento de dados de colmeias, para estudo em escolas.

O sistema é composto por placas embarcadas, e uma aplicação Web. As placas fazem a coleta de dados de temperatura, umidade, pressão e som, se comunicando com a aplicação via Wi-Fi por meio de requisições HTTP. A aplicação permite observar os dados de diferentes colmeias e sensores, assim como fazer o upload de um vídeo que mostre o fluxo de abelhas ao redor da entrada da colmeia. Esse vídeo é, então, processado para realizar a contagem de abelhas que entram e saem, expondo os resultados em gráficos.

1 Introdução

Em 2022, o professor Roberto Greco, do Instituto de Geociências da UNICAMP, iniciou um projeto para instalação de colmeias no assentamento Milton Santos, em Americana, SP. Esse projeto foi desenvolvido em parceria com o Instituto de Computação da UNICAMP, com alguns alunos do instituto desenvolvendo uma ferramenta para o monitoramento das colmeias como Projeto Final de Graduação sob orientação do professor Luiz Fernando Bittencourt [19]. A ferramenta era composta por sensores de temperatura, umidade e som, assim como um microcontrolador que enviava através de conexão Bluetooth os dados coletados para um aplicativo também desenvolvidos pelos alunos.

Este relatório detalha uma atualização do projeto, incluindo o desenvolvimento de uma aplicação Web que visa aumentar a escalabilidade do sistema e a eficiência de apresentação das informações. O projeto também conta com a adição de abordagens de aprendizado de máquina para a detecção de abelhas em vídeos, permitindo contar quantas entraram

*j174083@dac.unicamp.br
†j174358@dac.unicamp.br
‡r243792@dac.unicamp.br
§carlosat@ic.unicamp.br
¶bit@ic.unicamp.br

e saíram ao longo da gravação. Esses dados podem, então, ser relacionados às métricas coletadas, fornecendo informações adicionais sobre o comportamento das abelhas.

O relatório está estruturado em seções que abordam detalhadamente diversos aspectos do projeto. As seções são:

- **Evolução do Projeto e Objetivos:** Apresentação do histórico do projeto, assim como objetivo geral e suas subdivisões, especificando o propósito e os resultados esperados.
- **Arquitetura e Ferramentas**: Descrição da arquitetura do projeto como um todo e das ferramentas utilizadas em seu desenvolvimento.
- **Placa:** Descrição da implementação do hardware e software da placa utilizada para coleta de dados das colmeias.
- **Aplicação Web:** Descrição do aplicativo utilizado para observação dos dados coletados e processamento de vídeos da colmeia.
- **Banco de Dados:** Explicação do banco de dados utilizado para armazenamento dos dados e cadastro das colmeias e sensores.
- **API:** Explicação da API desenvolvida para comunicação entre as diferentes partes do sistema.
- **Detecção e Contagem de Abelhas**: Descrição das técnicas utilizadas para a análise de vídeos com objetivo de analisar o fluxo de abelhas na colmeia.
- **Documentação:** Explicação sobre a estrutura do repositório com o código fonte do sistema.
- **Resultados:** Descrição dos resultados qualitativos e quantitativos da implementação do sistema, incluindo também benchmarks.
- **Modo de Uso:** Instruções de como utilizar todas as partes do sistema, da placa ao aplicativo.
- **Conclusão e Trabalhos Futuros:** Considerações finais sobre o projeto, discussões sobre seu impacto e sugestões de melhorias para trabalhos futuros.
- **Referências:** Fontes e referências utilizadas no desenvolvimento do projeto, incluindo também material desenvolvido pelos autores.

2 Evolução do Projeto e Objetivos

Após a implementação inicial, o professor Greco procurou a ajuda do professor Fabiano Fruett, da Faculdade de Engenharia Elétrica e Computação, para desenvolver uma nova placa. Esta contava com um Raspberry Pi Pico W, com microcontrolador RP2040, assim

como sensores de temperatura, umidade, pressão, proximidade e som. Em 2023, outro grupo de alunos do Instituto de Computação aprimorou o sistema de monitoramento, expandindo seu escopo para a utilização em escolas, novamente sob orientação do professor Bittencourt, mas agora acompanhado pelo professor Carlos Alberto Astudillo Trujillo [7]. Essa nova versão utilizou a placa desenvolvida pelo professor Fruett, com dados sendo armazenados em um microSD e enviados por Wi-Fi ao invés de Bluetooth. O aplicativo móvel também foi atualizado, passando a gerenciar a placa, além de expor um gráfico dos dados coletados. Porém, alguns problemas surgiram no envio de dados e na integração dos sensores de som, temperatura, pressão e umidade.

Em 2024, um novo projeto foi iniciado para a correção dos problemas encontrados, assim como a adição de uma plataforma Web para exposição dos dados [1]. Esse projeto também foi orientado pelo professor Bittencourt, dessa vez acompanhado pelo professor Fruett. Além das alterações descritas anteriormente, esse projeto substituiu a placa pela BitDogLab, uma placa voltada para projetos pedagógicos, desenvolvida pelo professor Fruett e com código aberto.

Para a atualização, os objetivos do projeto foram definidos com base nas necessidades levantadas pelo Prof. Roberto, visando o uso do sistema para fins educacionais. O principal destes é a adição da contagem de abelhas que entram e saem das colmeias, utilizando técnicas de visão computacional e inteligência artificial. Além disso, reformular a aplicação para apresentar e armazenar os dados de forma mais eficiente.

Para organização do trabalho, o objetivo principal foi dividido em metas menores:

- Adicionar funcionalidade de upload e processamento de vídeos para realizar a contagem de abelhas.
- Aplicar técnicas de inteligência artificial para detectar e rastrear abelhas nas proximidades da colmeia, usando ferramentas como OpenCV e YOLO.
- Substituir o Firebase como banco de dados pelo MongoDB, preservando a segurança do armazenamento e da transmissão.
- Desenvolver uma nova página web para exibição dos dados coletados, com gráficos separados para métricas diferentes.
- Tornar o sistema facilmente expansível, contemplando múltiplas colmeias e sensores.

3 Arquitetura e Ferramentas

3.1 Arquitetura Geral

O sistema é composto por cinco partes que se comunicam entre si, representados na Figura 1.

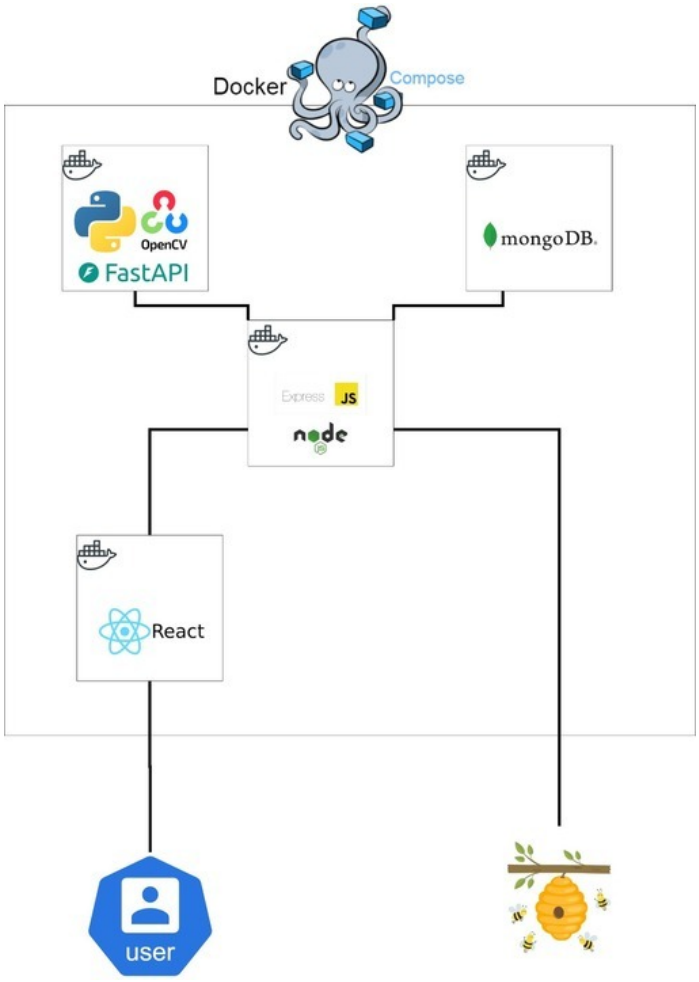


Figura 1: Diagrama da arquitetura do sistema, com ênfase na organização dos contêineres.

- **Placa**: Representada no diagrama por uma colmeia, inclui um microprocessador e sensores de temperatura, umidade, pressão e ruído. Por meio de código em MicroPython, são realizadas coletas periódicas nos sensores. Após a coleta, os dados são enviados para o backend JS por meio de requisições HTTP.

- **Aplicação Web**: Representada no diagrama pelo retângulo com a logo do React, é responsável pela interação com o usuário. Inclui gráficos personalizáveis e uma seção para upload de vídeos para processamento e contagem de abelhas. Se comunica com o backend JS através de requisições HTTP. Como indicado pelo diagrama, foi implementado em JavaScript, utilizando componentes da biblioteca React.

- **Banco de Dados**: Representado no diagrama pelo retângulo com a logo do MongoDB. É responsável por armazenar os dados obtidos pela placa, recebendo-os ao se comunicar com o backend JS. Foi implementado com o uso do MongoDB, gerando assim um banco de dados não relacional definido por *schemas* em alguma linguagem

de programação, como JavaScript.

- **Backend JS**: Representado pelo retângulo no centro do diagrama, também pode ser considerado o centro da arquitetura. Através de requisições HTTP recebe os dados da placa e os armazena no banco de dados. Também recebe requisições da aplicação Web para acessar os dados armazenados e retorná-los, assim como para armazenar os dados dos vídeos enviados pelo usuário. Por fim, se comunica com o backend Python para iniciar o processamento dos vídeos. Foi implementado em JavaScript, com o framework Express e executando em um ambiente Node.js.

- **Backend Python**: Representado pelo retângulo no canto superior esquerdo do diagrama. É responsável pelo processamento dos vídeos, realizando a contagem de quantas abelhas entram e saem da colmeia utilizando técnicas de aprendizado de máquina. Se comunica com o backend JS para receber os vídeos e seus metadados e enviar os resultados. Foi implementado em Python, utilizando o framework FastAPI, a biblioteca OpenCV e um modelo YOLO.

3.2 Docker Compose

O Docker Compose [2] foi utilizado para orquestrar as diferentes partes do sistema de maneira eficiente e modular, evitando conflitos de ambiente. Foi decidido utilizar essa ferramenta por sua simplicidade de configuração, sem sacrificar a eficiência. Ela permite configurar e executar múltiplos contêineres em conjunto, simplificando o desenvolvimento, a integração e a replicação do ambiente. No arquivo `docker-compose.yml`, foram definidos os serviços `backend-mongo`, `frontend-react`, `backend-nodejs-api` e `backend-python`.

Embora o código da placa de microprocessador faça parte do sistema, ele não foi incluído na orquestração, uma vez que é executado diretamente no hardware e não interage como um serviço conteinerizado.

3.2.1 backend-mongo

Expõe a porta padrão do MongoDB (27017) para interação com a API do backend. Para garantir a persistência de dados entre reinicializações do contêiner, foi utilizado o volume nomeado `mongo-data`.

3.2.2 frontend-react

O serviço `frontend-react` executa a interface do usuário e expõe a aplicação na porta 5173, frequentemente utilizada por frameworks modernos como Vite. Durante o desenvolvimento, o *hot reload* foi configurado, permitindo que alterações no código sejam automaticamente refletidas na aplicação sem necessidade de reiniciar o servidor manualmente. Para gerenciar os pacotes de dependências, foi utilizada a ferramenta pnpm, uma alternativa ao npm que reduz o uso de espaço em disco.

3.2.3 backend-nodejs-api

Expõe a API na porta 3003, mapeada para 3000 dentro do contêiner. O serviço foi configurado para ser iniciado apenas após o `backend-mongo`, garantindo que o banco de dados esteja disponível para conexões. Um volume foi configurado para sincronizar o contêiner e o código local, facilitando o desenvolvimento ao permitir alterações em tempo real. Para gerenciar as dependências de forma automática, foi utilizada a ferramenta npm.

3.2.4 backend-python

Expõe a porta 5000, também mapeada para 3000 dentro do contêiner. Assim como o `backend-nodejs-api`, o `backend-python` depende do `backend-mongo` e é inicializado apenas após o banco de dados estar disponível. O contêiner também foi configurado para sincronizar o código local com o diretório interno, permitindo atualizações em tempo real. Além disso, o comando de execução foi sobrescrito para rodar a aplicação com o servidor ASGI `uvicorn`, configurado com recarregamento automático (*reload*), facilitando o processo de desenvolvimento.

4 Placa

A principal placa utilizada durante o projeto foi a BitDogLab [15], desenvolvida por uma iniciativa do projeto Escola 4.0 da UNICAMP. Além dela, também foi utilizada outra placa desenvolvida pelos professores Fruett e Greco [13]. Essa placa tem aplicações menos amplas que a BitDogLab, mas para os propósitos deste trabalho seu funcionamento é idêntico.

4.1 Componentes

Segue lista dos componentes e sensores disponíveis para uso projeto:

- **Raspberry Pi Pico W:** Inclui um microcontrolador RP2040 com 2MB de memória flash, suporte a conexões sem fio (Bluetooth e Wi-Fi), entrada micro USB B para potência e dados e 40 pinos para funcionalidades variadas [8].
- **Sensor de Proximidade Ultrassônico E18-D80NK - NPN:** Alcance de detecção de 3 a 80 cm. Utilizado por grupos anteriores para detectar a passagem de abelhas pela entrada da colmeia.
- **Módulo GY-MAX4466 com Microfone**: Captura sons nas proximidades da colmeia, permitindo análises das atividades internas e das condições do ambiente [12] [3].
- **Sensor Ambiental BME680:** Mede temperatura, pressão atmosférica e umidade, permitindo análise da correlação entre condições ambientais e comportamento das abelhas. Também faz a detecção de gases, especialmente compostos orgânicos voláteis (VOCs) [17].

4.2 Ciclo de Funcionamento

Para o monitoramento contínuo de colmeias, o software da placa envolve um ciclo operacional que visa garantir que a coleta e armazenamento dos dados ocorra, lidando com erros se necessário. O ciclo pode ser dividido em três etapas:

1. **Inicialização:** É efetuada a conexão à rede previamente definida e a definição do horário local atual. Além disso, é feita a verificação dos IDs da colmeia e dos sensores. Caso eles ainda não possuam identificadores, estes são criados e enviados ao banco de dados. Por fim, é feita a inicialização dos sensores, incluindo testes para minimizar erros durante a operação.

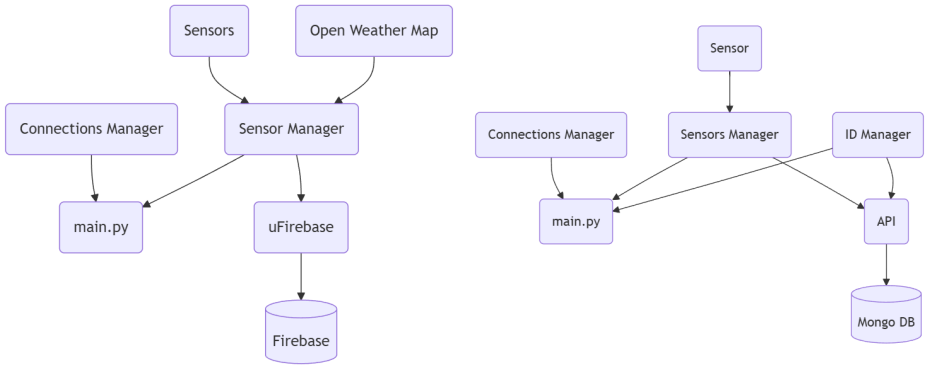
2. **Leitura dos Sensores:** São realizadas leituras periódicas nos sensores da placa, sendo o intervalo entre cada medição configurável, podendo ser adaptado para cada caso. Durante essa etapa, o horário local também é continuamente atualizado, de modo a vincular as métricas obtidas com um timestamp para visualização temporal do comportamento da colmeia.

3. **Envio e Armazenamento:** Os dados obtidos são empacotados e enviados para a API em JavaScript com o método POST do protocolo HTTP/1.1. A API possui lógica própria que cuida do armazenamento dos dados, reduzindo a carga computacional sobre o microcontrolador.

4.3 Arquitetura da Placa

Assim como nos trabalhos anteriores, o software da placa foi desenvolvido com a linguagem MicroPython [9], uma implementação de Python3 compatível com microcontroladores. Assim, foi possível utilizar como base o código desenvolvido em projetos anteriores, modificando-o sem que fosse necessário remodelar toda a lógica de funcionamento para se adequar à outras linguagens. A arquitetura do trabalho anterior, representada como diagrama na Fig. (2a), foi utilizada como base para o desenvolvimento desta.

No caso deste projeto, foi decidido que os dados obtidos pela API OpenWheaterMap não seriam mais utilizados, assim como o Firebase, que foi substituído pelo MongoDB. Adicionalmente, para permitir a expansão do sistema, foi adicionado um sistema de identificadores para colmeias e sensores. Assim, a arquitetura anterior foi modificada, como mostrado na Fig. (2b).



(a) Arquitetura anterior (b) Arquitetura atual

Figura 2: Comparativo de arquitetura da placa entre projeto anterior e projeto atual

Cada componente da arquitetura do código da placa tem suas próprias funções e responsabilidades.

- **API:** Responsável pela conexão com o backend do sistema.
- **Connections Manager:** Gerencia a conexão da placa à internet via Wi-Fi, sem mudanças em relação ao projeto anterior.
- **ID Manager:** Gerencia os identificadores da colmeia e dos sensores.
- **Sensors Manager:** Responsável pela inicialização dos sensores, assim como suas leituras.

4.4 Bibliotecas Externas e APIs

Foram utilizadas algumas bibliotecas padrão do MicroPython, mas também outras bibliotecas específicas, descritas a seguir:

- *bme680:* Utilizada para leituras do sensor BME680. É um port do MicroPython de um driver originalmente desenvolvido pela Adafruit [10].
- *urequests:* Utilizada para envio de requisições HTTP a API. É uma implementação em MicroPython similar à biblioteca *requests* do Python.
- *ntptime:* Utilizada para obter o horário atual de um servidor NTP.

Além das bibliotecas, foi utilizada uma API desenvolvida especificamente para este projeto, para envio de dados para o backend NodeJS, descrita na Seção 7.

4.5 Estrutura do Software

4.5.1 Laço Principal

Por padrão do microcontrolador, o arquivo `main.py` é executado assim que a placa é energizada. Assim, ele é responsável pela inicialização do software da placa, fazendo chamadas aos outros arquivos. As funções desses arquivos serão explicadas com mais detalhes nas seções a seguir.

Inicialmente, a função `connect_to_wifi` do arquivo `connections_manager.py` é chamada para iniciar a conexão. Em seguida, é utilizada a biblioteca *ntptime* para calcular o horário atual no fuso horário de Brasília. Então, é feita uma chamada à função `get_ids` do arquivo `id_manager.py` para verificar os IDs dos sensores e colmeia. Por fim, é criada uma instância da classe `SensorsManager`, seguida por uma chamada à função `start_reading`, que inicia o laço de coletas e envios.

4.5.2 Connections Manager

Contido no arquivo `connections_manager.py` e responsável pela conexão à rede Wi-Fi, utilizando a biblioteca nativa *network*. De forma geral, ele não mudou em relação ao projeto anterior. Para mudar a rede, basta alterar as variáveis `REDE` e `SENHA`.

4.5.3 ID Manager

Gerencia os identificadores da placa e dos sensores, que são utilizados no banco de dados para diferenciar entre múltiplas placas, facilitando a expansão do sistema e diminuindo sua rigidez.

A função `get_ids` do arquivo `id_manager.py` verifica os identificadores armazenados no arquivo `ids.txt`. Caso o arquivo não exista ou não esteja no formato correto, são feitas chamadas à biblioteca *ubinascii* para criar identificadores aleatórios, compostos por números e letras. Em seguida, são feitas chamadas às funções `write_hiveId` e `write_sensorId` do arquivo `api.py`, registrando os sensores.

4.5.4 Conexão com API

Gerenciada pelo arquivo `api.py` que efetua requisições HTTP através da função `post` da biblioteca *urequests*, enviando para o backend as informações necessárias para cadastrar IDs ou atualizar os dados coletados. De forma geral, as funções enviam dados no formato *json* de acordo com os endpoints definidos pelo backend. No caso do envio de dados coletados, uma requisição é feita para cada dado coletado.

4.5.5 Sensors Manager

Representada pela classe `SensorsManager`, responsável pela inicialização dos sensores, coleta e envio dos dados e definida no arquivo `sensors_manager.py`. Em relação ao trabalho

anterior, o código foi simplificado, visto que somente os sensores exteriores à colmeia estavam sendo utilizados. Além disso, foram adicionados trechos de código para gerenciar o microfone e removidos trechos referentes ao sensor de proximidade e de gás.

Ao instanciar a classe, é realizada também a inicialização dos sensores. No caso do sensor BME680, são utilizadas funções da biblioteca *bme680*, que realizam testes para garantir o funcionamento correto do equipamento. Quanto ao microfone, por ser analógico, basta conectar em algum pino que possua capacidades de conversor analógico digital e no software declarar que esse está sendo utilizado no modo *ADC*. Pelo mesmo motivo, não há rotinas que validam seu funcionamento correto o que torna fundamental a validação humana para garantir o comportamento adequado do microfone analógico.

Além da inicialização dos sensores, o construtor da classe associa cada métrica a um sensor e um método de leitura. No caso das métricas do BME680, o método de leitura é simples: basta verificar se o objeto do sensor não está com erros e então retornar o valor de um de seus atributos. Em relação ao microfone, a simplicidade do equipamento analógico se traduz em uma maior complexidade de leitura dos níveis de intensidade sonora. Para tanto é necessário ler os valores medidos em um pequeno intervalo de tempo, converter as medidas para Volts e utilizando a variação de tensão gerada pelo microfone calcular o nível de pressão sonora a partir de uma fórmula logarítmica que tem como parâmetros a tensão medida e depende de constantes específicas ao equipamento. Vale adicionar que essa complexidade é o motivo de trabalhos anteriores relatarem o não funcionamento do microfone. Esses trabalhos que utilizaram fórmulas lineares para calcular a níveis de pressão sonora de forma que a medida obtida era na verdade a tensão gerada pela variação da intensidade sonora no ambiente. Combinado com o fato de utilizarem fórmulas incorretas as medidas realizadas nesse trabalho eram realizadas em um único clico da placa o que torna efetivamente impossível detectar qualquer frequência de sonora no espectro audível (20Hz - 20 kHz).

A principal função desse arquivo é `start_reading`, responsável pelo laço principal de funcionamento da placa. A cada iteração, a primeira etapa é a leitura dos sensores utilizando os métodos de leitura definidos pelo construtor. Em seguida, é realizada a atualização do horário local, utilizando a biblioteca padrão *time*, e a criação do timestamp no padrão ISO-8601. Então, a função `write_data` do arquivo `api.py` é chamada, passando como parâmetros as leituras, os IDs e o timestamp das medidas. Por fim, a função `sleep` é utilizada para garantir que a próxima medida ocorra somente depois de um intervalo definido previamente.

Em relação ao trabalho anterior, deixamos de utilizar os dados da API `OpenWeatherMap` pois eles eram redundantes com as medições dos sensores. Apesar dos dados fornecidos pela API serem precisos, eles são referentes a uma área extensa, e não necessariamente correspondem às condições nas proximidades da colmeia. Adicionalmente, as chamadas à API consomem recursos computacionais do microcontrolador. Deixando de usar esses recursos, torna-se possível adicionar mais sensores no futuro. Assim, foi decidido que somente as informações coletadas pelos sensores seriam utilizadas.

4.6 Sensores

4.6.1 Microfone

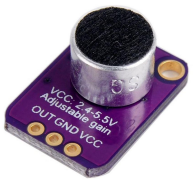


Figura 3: Módulo GY-MAX4466

Fonte: https://www.smartkits.com.br/modulo-microfone-c-ganho-ajustavel-gy-max4466

O sensor GY-MAX4466 (Fig. 3) é um módulo composto por um microfone de eletreto, um amplificador MAX4466 [12] e um potenciômetro. Abaixo estão as principais características do módulo:

- **Modelo:** GY-MAX4466
- **Dimensões:** 9.7mm x 6.7mm x 4.5mm
- **Tensão de Operação:** 2.4V - 5.5V DC
- **Corrente de Operação:** $<$ 0.85mA
- **Saída:** Analógica
- **Resposta em Frequência:** 20Hz a 20kHz
- **Ganho do Amplificador:** Ajustável (potenciômetro)
- **Temperatura de Operação:** -40ºC a 85ºC

Por sua capacidade de capturar os sons do ambiente, o sensor pode ser utilizado para estudar como a poluição sonora e a proximidade à atividade humana afetam as abelhas. Adicionalmente, se o sensor for posicionado perto da colmeia ou em seu interior, ele pode captar os sons emitidos pelas abelhas em si, como zumbidos e bater de asas. Características como a frequência e intensidade desses sons podem ser utilizadas para analisar o comportamento das abelhas, incluindo momentos de repouso ou alta atividade.

Afim de detectar sons no espectro audível, as medidas no microfone são realizados durante intervalos de 50ms. Nesses intervalos, é calculada a tensão RMS gerada pela variação da intensidade sonora detectada pelo microfone. Em seguida, utiliza-se a Eq. [1] para calcular o nível de pressão sonora. Nessa equação, Gain representa o ganho do amplificador, Sensitivity a sensibilidade do microfone medida nas condições normais (1V/Pa) e V_{Ref} é

constante derivada da sensibilidade do microfone ao calcular V_{RMS} nas condições normais utilizando somente a parte logarítmica da equação. Os valores para as constantes características do microfone podem ser obtidos a partir de seu datasheet [3]. Foram feitos testes variando o valor de ganho, mas não foram percebidas mudanças significativas nas medições. Esse comportamento é inesperado, e pode ser investigado mais a fundo em trabalhos futuros.

$$\text{SPL} = 20 \cdot \log\left(\frac{V_{RMS}}{V_{Ref}}\right) + (\text{Gain} - \text{Sensitivity}) \tag{1}$$

4.6.2 Sensor BME680

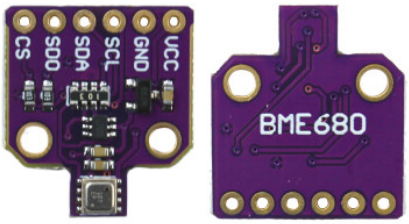


Figura 4: Sensor BME680

Fonte: https://www.usinainfo.com.br/sensor-de-pressao-arduino/sensor-de-pressao-umidade-temperatura-e-gases-bme680-8541.html

Este sensor (Fig. 4) combina medidas de umidade relativa, pressão barométrica, temperatura ambiente e gases. Desenvolvido especialmente para aplicações móveis, com ênfase em baixo consumo de energia. Esse tipo de sensor é comumente utilizado para avaliar a qualidade do ar, especialmente em dispositivos pessoais. Abaixo estão os principais pontos abordados no datasheet do sensor [17]:

- **Modelo:** BME680
- **Fabricante:** Bosch Sensortec
- **Dimensões:** 3.0mm x 3.0mm x 0.93mm
- **Tensão de Operação:** 1.17V - 3.6V DC
- **Corrente de Operação:** 0.15mA a 0.7mA (modo normal)
- **Saída:** I2C, SPI
- **Faixa de Temperatura:** -40ºC a 85ºC

No caso desse projeto, o sensor tem um papel importante no entendimento das relações entre a atividade das abelhas e os fatores ambientais, especialmente se utilizado em conjunto com a ferramenta de análise de vídeos. Além disso, a temperatura e umidade ambientes afetam a temperatura interna da colmeia, um parâmetro crítico para a saúde e desenvolvimento das abelhas, assim como para a produção do mel. Valores inadequados de umidade podem levar ao crescimento de fungos e à propagação de doenças, colocando em risco a saúde das abelhas.

4.6.3 Sensor de Proximidade

Apesar de ter sido incluído nos projetos de grupos anteriores, foi decidido que o sensor de proximidade não seria utilizado neste trabalho. Como havia apenas um sensor, era possível detectar se houve passagem de abelhas pela entrada, mas não a direção em que elas se moviam. Assim, era impossível utilizá-lo para a contagem do fluxo de abelhas, um dos principais objetivos do projeto.

5 Aplicação Web

Embora seja possível visualizar os dados utilizando diversas ferramentas, como scripts que buscam os dados na API Express JS e geram gráficos, a alternativa mais prática é uma aplicação Web. Nesse sentido, foi desenvolvida uma aplicação com o objetivo de facilitar acesso aos dados nos mais diversos contextos, sejam eles educação ou pesquisa.

5.1 Tecnologias

Antes de iniciar o desenvolvimento é importante considerar alguns pontos para determinar as tecnologias a serem utilizados em um sistema Web: facilidade de manutenção, performance e escalabilidade. Considerando que o objetivo é desenvolver um dashboard que deve ser atualizado em intervalos de alguns minutos fica claro que performance na renderização, apesar de importante, não é crítica. Ademais, como o sistema deve ser reutilizado e estendido em trabalhos futuros, a facilidade de manutenção e extensibilidade são fundamentais. Sendo assim, foi definido que a tecnologia a ser utilizada seria o React.

React [14] é uma biblioteca JavaScript de código aberto criada pelo Facebook em 2013. Utilizada para desenvolver interfaces de usuário (UI) interativas, que se destaca pela sua abordagem declarativa e baseada em componentes. Assim, permitindo a construção aplicações web dinâmicas e responsivas com facilidade e eficiência. Comparado com frameworks como Angular e Vue.js, React oferece um equilíbrio entre simplicidade e poder, tornando a experiencia de desenvolvimento para este sistema mais positiva.

Mesmo que o objetivo principal desse projeto seja a funcionalidade do sistema, a aplicação ser agradável e intuitiva é um requisito não funcional de grande importância. Devido ao curto tempo de desenvolvimento, foi decidido que não seria viável desenvolver um design totalmente original ao mesmo tempo em que as funcionalidades são implementadas. Sendo assim, para agilizar o processo de desenvolvimento sem comprometer muito a qualidade

visual, utiliza-se da biblioteca de componentes Shadcn [18] como ferramenta fundamental no dashboard.

Por fim, como o sistema será utilizado por pesquisadores e em situações educacionais, a internacionalização (i18n) se torna outro requisito importante, garantindo acesso ao sistema nos mais diversos meios. Com objetivo de alcançar a internacionalização, vale-se dos módulos *i18next*, *react-i18next* e *i18next-browser-languagedetector*, permitindo alcançar essa meta de forma simples e escalável.

5.2 Implementação

Ainda que não seja o principal, a performance do dashboard não deixa de ser importante. Por isso, foram utilizadas boas práticas em React para evitar ciclos de renderização desnecessários causados por mudanças no estado de componentes. Foi identificado que um estado fundamental para a aplicação é aquele que representa qual a colmeia monitorada atualmente e quais dos seus sensores são de interesse. Por serem utilizados por diversos componentes na aplicação, esse estado é fornecido a partir de componentes que implementam o padrão *Provider* permitindo que todos os componentes *children* tenham acesso de leitura e escrita a esses dados de forma simples, sem a necessidade de propagá-los como *props* por diversas camadas.

Ademais, é comum realizar chamadas nos diversos endpoints de dados para os gráficos. Implementar isso diretamente nos componentes aumentaria muito a complexidade do código e a dificuldade de manutenção. Por esse motivo, utilizam-se *hooks* para cada endpoint da aplicação Express JS, que periodicamente buscam os dados na API . Assim permitindo maior aproveitamento de código.

Por fim, para garantir que diversos componentes não seriam renderizados novamente sem necessidade, são utilizadas técnicas de memoização como *useMemo* (dados), *useCallback* (funções) e *memo* (componentes simples). Dessa forma, efeitos colaterais de atualizações nos dados originados nos *hooks* são propagadas somente se tornam os dados memorizados inválidos.

6 Banco de Dados

Uma decisão de grande impacto na performance de qualquer aplicação é a forma de organizar os dados de interesse. Por apresentarem uma estrutura mais rígida, bancos de dados relacionais são capazes de criar registros mais precisos e melhor organizados dos dados. Contudo, como consequência dessa organização se tornam menos escaláveis e flexíveis a alterações. Em um contexto de sistemas de IoT, que devem ingerir uma grande quantidade de dados de forma rápida e eficiente, utilizar um banco de dados relacional pode não ser a melhor decisão. Por esse motivo, optou-se pelo uso do Mongo DB, especificamente sua versão "Community"[11]: um banco de dados não relacional com documentação amplamente disponível e que poderia ser facilmente adaptado para o caso de uso de séries temporais.

Foram definidos *schemas* para quatro modelos de dados, descritos a seguir.

6.1 Colmeias

```
const hiveSchema = new mongoose.Schema({
  hiveId: { type: String, required: true, unique: true },
  location: { type: String, required: true },
  description: { type: String, required: false },
}).index({ hiveId: 1 });
```

Figura 5: *Schema* do modelo *hives*

Indexadas por um identificador único, cadastrado na primeira utilização da placa, conforme descrito na seção 4.5.3. Cada colmeia tem, além do seu identificador, uma localização associada e uma descrição opcional.

6.2 Métricas

```
const schema = new mongoose.Schema({
  sensorId: { type: String, required: true },
  hiveId: { type: String, required: true },
  metricType: { type: String, required: true },
  value: { type: Number, required: true },
  timestamp: { type: Date, default: Date.now },
}).index({ timestamp: 1 });
```

Figura 6: *Schema* do modelo *metrics*

Nesse caso, a base de dados é indexada pelo timestamp da medida, de forma que filtrar dados por períodos se torna um operação muito simples e eficiente. Ademais, indexar os dados por timestamps permite a interpolação dos dados por períodos, de forma que o tamanho dos payloads enviados pela API possa ser reduzido. Portanto, aumentando a escalabilidade da API e o desempenho da aplicação Web. Por exemplo, não são necessários todos os dados coletados no último mês para visualização no dashboard. Basta uma medida representativa por dia. Assim, é possível reduzir o número de pontos transmitidos em uma requisição de 2880 para somente 30. Além do timestamp, cada item tem os identificadores únicos do sensor que fez a medição, assim como da colmeia onde este estava acoplado. Há também o tipo de métrica e seu valor, que são utilizado para a criação dos gráficos.

6.3 Sensores

```
const sensorSchema = new mongoose.Schema({
  sensorId: { type: String, required: true, unique: true },
  description: { type: String, required: false },
}).index({ sensorId: 1 });
```

Figura 7: *Schema* do modelo *sensors*

Também indexados por seus identificadores único, cadastrados na primeira utilização da placa, conforme descrito na seção 4.5.3. Além do identificador, cada sensor tem uma descrição opcional, que pode guardar informações como o tipo do sensor, onde ele está localizado e qual tipo de dados ele obtém.

6.4 Vídeos

```
const crossingEventSchema = new mongoose.Schema({
    direction: { type: String, required: true},
    timestamp: { type: Number, required: true}
});

const videoSchema = new mongoose.Schema({
    filename: {type: String, required: true },
    duration: { type: Number, required: true },
    resolution: { type: String, required: true },
    detector_type: { type: String, required: true },
    processing_time: { type: Number, required: true },
    events: { type: [crossingEventSchema], required: true },
    status: { type: String, required: true }
  }, { timestamps: true });
```

Figura 8: *Schemas* dos modelos *videos* e *crossingEvent*

Esse *schema* armazena as informações de cada vídeo enviado para processamento, incluindo também os resultados obtidos.

Não são indexados nem por identificador único nem por timestamp. Cada item inclui parâmetros do vídeo (nome do arquivo, duração e resolução) e parâmetros do processamento. Dentre estes, o primeiro é tipo de detector, que pode ser *YOLO* ou *background-subtraction*, definindo qual abordagem será utilizada na análise do vídeo. O parâmetro status começa como "Sending", sendo atualizado para "Processing"e então "Done"conforme o processamento ocorre. Por fim, os parâmetros de tempo de processamento e eventos são inicializados

vazios, sendo preenchidos após o fim da detecção. Os itens da base também são acompanhados por timestamps, permitindo armazenar quando eles foram enviados e exibir essa informação para o usuário na aplicação web.

O campo *events* é composto por uma lista de *crossingEvents*, um *schema* auxiliar definido para melhorar a legibilidade do código. Cada item dessa categoria representa um evento detectado em que uma abelha passa pela entrada da colmeia. Assim, são armazenadas a direção do evento (entrando ou saindo da colmeia) e a timestamp do momento em que ele é detectado no vídeo.

7 API

7.1 Backend JavaScript

Nos trabalhos anteriores, verifica-se as vantagens de utilizar o Firebase como backend para uma aplicação dessa escala, como atualização em tempo real dos dados a partir de Server Sent Events (SSE). Contudo, a capacidade do Firebase é diretamente associada ao seu custo. Para tornar esse sistema escalável e acessível para educação e pesquisa, utilizar um Backend as a Service (*Baas*) como o Firebase, se torna inviável. Sendo assim, foi desenvolvido nesse projeto um backend em Express JS, com objetivo de substituir os serviços do Firebase e reduzir os custos de implantação e manutenção da aplicação.

Há uma gama de frameworks para JS disponíveis para desenvolver serviços de backend e todos apresentam suas vantagens e desvantagens. A escolha do Express JS [4] para implementar esse projeto se deu por conta da familiaridade dos autores com ele. Esse framework é muito escalável por permitir utilizar diversos serviços de middleware para auxiliar o desenvolvimento. Por esse mesmo motivo, desenvolver aplicações de larga escala utilizando-o pode se tornar rapidamente complexo. Sendo assim, foi implementado um padrão Model-View-Controller (MVC) para permitir ao máximo reuso e manutenção do código. Em um primeiro momento são definidos os modelos dos dados a partir de um *schema* para o MongoDB. Em sequência, implementam-se os controladores para as diversas métricas. Nesse momento são definidas as respostas para chamadas HTTP, as validações e as operações realizadas sobre o banco de dados. Por fim, definem-se as "Views"que, nesse caso, são as rotas expostas para uso.

7.1.1 Colmeias

GET /hive

Parâmetro	Tipo	Localização	Descrição
hiveId	String	query	Filtro por identificador único da colmeia
location	String	query	Filtro por descrição do local da colmeia.
description	String	query	Filtro por descrição da colmeia.

POST /hive

Parâmetro	Tipo	Localização	Descrição
hiveId	String	body	Obrigatório. Identificador único da colmeia
location	String	body	Obrigatório. Descrição do local da colmeia.
description	String	body	Orbigatório. Descrição da colmeia.

7.1.2 Métricas

GET /metrics/<metric>

Parâmetro	Tipo	Localização	Descrição
hiveId	String	query	Filtro por identificador único da colmeia.
sensorId	String	query	Filtro por identificador único do sensor.
limit	String	query	Filtro por número de aferições.
startDate	ISO Date	query	Filtro por data inicial.
endDate	ISO Date	query	Filtro por data final.
timestamp	ISO Date	query	Filtro por timestamp.
value	Number	query	Filtro por valor.

POST /metrics/<metric>

Parâmetro	Tipo	Localização	Descrição
hiveId	String	body	Obrigatório. Identificador único da colmeia.
sensorId	String	body	Obrigatório. Identificador único do sensor.
value	String	body	Obrigatório. Valor da aferição.
timestamp	ISO Date	body	Orbigatório. Timestamp da aferição.

7.1.3 Sensores

GET /sensor

Parâmetro	Tipo	Localização	Descrição
hiveId	String	query	Filtro por identificador único da colmeia.
metricType	String	query	Filtro por tipo de métrica (e.g. Temperatura).

POST /sensor

Parâmetro	Tipo	Localização	Descrição
hiveId	String	body	Obrigatório. Identificador único da colmeia.
sensorId	String	body	Obrigatório. Identificador único do sensor.
metric	String	body	Obrigatório. Tipo do sensor (e.g. Temperatura)
description	String	body	Obrigatório. Descrição do sensor.

7.1.4 Vídeos

GET /videos

Parâmetro	Tipo	Localização	Descrição
id	String	query	Filtro por identificador único do vídeo.

POST /videos/upload

Parâmetro	Tipo	Localização	Descrição
detector_type	String	body	Obrigatório. Algoritmo de Detecção que deve ser utilizado.
videoFile	File	body	Obrigatório. Arquivo MP4 para análise.

POST /videos/callback

Parâmetro	Tipo	Localização	Descrição
video_id	String	body	Obrigatório. Algoritmo de Detecção que deve ser utilizado.
filename	String	body	Obrigatório. Arquivo MP4 para análise.
duration	Number	body	Obrigatório. Duração do vídeo em segundos.
resolution	String	body	Obrigatório. Resolução do vídeo.
processing_time	Number	body	Obrigatório. Tempo de processamento em segundos.
events	Object[]	body	Obrigatório. Array de Eventos de detecção.
status	String	body	Obrigatório. Novo status do processo (e.g. Concluído).

7.2 Backend Python

Além do backend JS, foi desenvolvido um em Python, com o framework FastAPI [5], responsável pelo processamento dos vídeos. Esse framework foi escolhido por sua ampla documentação e suporte extensivo a operações assíncronas, necessárias para o processamento. Há um endpoint responsável por receber, através de uma chamada HTTP, tanto o arquivo do vídeo quanto os seus metadados determinados pelo *schema*. Após o recebimento e a verificação de que o arquivo não foi corrompido, o processamento em si é iniciado como uma Background Task (ou seja, assíncrono), permitindo que o usuário continue a operação. Por fim, é devolvida uma resposta, confirmando que o processamento foi iniciado ou avisando que houve erro. Quando o processamento acaba, os resultados (ou a mensagem de erro relevante) são enviados para o backend JS, que atualiza o banco de dados.

8 Detecção e Contagem de Abelhas

A contagem de abelhas é um requisito que tem por objetivo, a partir de um vídeo de entrada, obter o fluxo de abelhas que entram e saem da colmeia ao longo do vídeo, substituindo a contagem manual. De início foram levantados alguns dos desafios de efetuar essa contagem, tais quais: o tamanho pequeno das abelhas jataí, aproximadamente 5mm; a grande velocidade nas quais elas se movimentam, resultando em borrões nas imagens; a sua coloração amarela e preta que as camuflava contra fundos de madeira e vegetação; qualidade e taxa de

atualização do vídeo. Com base nisso, é possível dividir a contagem em 3 partes principais: Detecção, rastreamento e fluxo. Cada uma com sua função bem definida e utilizando de mecanismos específicos para lidar com os diferentes desafios levantados.

8.1 Detecção

Essa etapa consiste em, a partir de um quadro de um vídeo, identificar se haviam abelhas e onde elas estariam localizadas na imagem. Como mencionado, foi necessário lidar com alguns dos desafios levantados anteriormente, em especial, os relacionados com o tamanho e identificabilidade das abelhas. Para solucionar isso, foram utilizadas duas abordagens independentes, cada uma com suas vantagens e limitações que contemplam cenários diferentes, sendo elas ***processamento de imagem por meio de transformações morfológicas*** e ***aprendizado de máquina***. Mas antes de entrar nas técnicas utilizadas, é necessário salientar alguns pontos importantes.

8.1.1 Identificabilidade

(a) Abelha presente (b) Sem abelha

Figura 9: Duas imagens de um mesmo local com e sem abelha, respectivamente

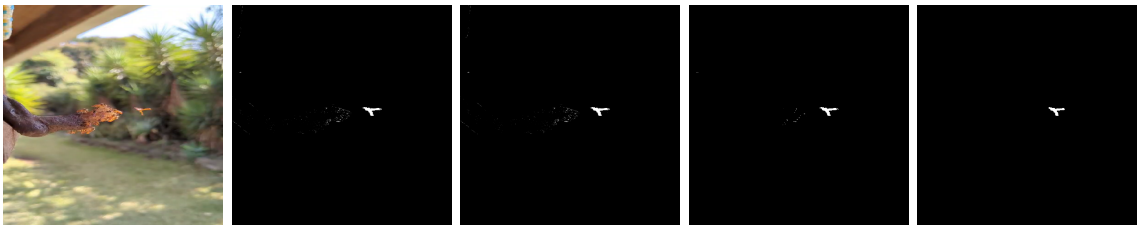
Com relação a detecção manual feita por seres humanos, a identificação das abelhas não se dá apenas por similaridade dos pontos da imagem com as formas e cores que caracterizam uma abelha. Parte do esforço em detectar as abelhas é analisar o contexto do vídeo e identificar diferenças temporais. Por exemplo, uma pessoa, muitas vezes inconscientemente, consegue prever onde uma abelha vai estar com base em sua velocidade aparente e no fato de que ela não pode desaparecer ou aparecer em pontos fora das extremidades do vídeo. Com base nessas heurísticas construídas, é possível identificar áreas de interesse onde abelhas podem ou não estar e ajustar o rigor com o qual é concluído se um conjunto de pixels representam uma abelha com certa confiança. Portanto, uma mancha em um local próximo a onde uma abelha deveria estar Fig. (9a) será considerada uma abelha, mesmo que sua forma esteja bastante borrada e distorcida a partir do momento em que há contexto Fig. (9b).

8.1.2 Processamento de imagem

A detecção por meio de técnicas de processamento de imagem é constituída por uma pipeline de transformações nas imagens utilizando técnicas já conhecidas. Seu objetivo é obter, a partir do vídeo de entrada, as bounding boxes das abelhas nas imagens. Para tanto, foram assumidos alguns comportamentos do vídeo: câmera estável e estática; pouco ou nenhum ruído de imagem; fundo com pouco ou nenhum movimento; abelhas compõem a vasta maioria dos objetos em movimentos na imagem.

Figura 10: Pipeline de processamento de vídeo para referência

Portanto, com base na Fig. 10, passando por cada estado partindo da imagem original, são aplicadas as seguintes técnicas: subtração de fundo, desfoque gaussiano, máscara binária e abertura. Um exemplo da aplicação do Pipeline é apresentado abaixo na Fig. 11.



(a) Original (b) Bg. Sub. (c) Gaussian Blur (d) Binary Mask (e) Result

Figura 11: Quadros em cada estágio da pipeline

8.1.2.1 Subtração de Fundo

Figura 12: Fundo estimado para três quadros consecutivos.

A subtração de fundo (***background subtraction***) é uma técnica que diferencia o plano de fundo do primeiro plano em um vídeo. Em especial, busca detectar objetos em movimento (plano da frente) a partir de um vídeo por meio da subtração entre o quadro atual e o fundo estimado, onde o resultado deve ser maior que um limiar pré-estabelecido. Esta técnica possui alguns requisitos como o vídeo ser estático, assim como o seu fundo.

Para a elaboração desta etapa, foi utilizado o subtrator MOG2 do OpenCV, com os valores de limiar e histórico de quadros sendo, respectivamente 50 e 500. Esses valores foram obtidos empiricamente, de forma que os objetos de interesse sejam mantidos mesmo com pequenas oscilações no vídeo. Finalmente, os resultados obtidos consistem em imagens em escalas de cinza (*grayscale*) onde os pixels transicionam de preto até branco, tornando-se mais claros quanto mais diferente do fundo estimado eles forem. Analogamente, é possível associar uma interpretação a cor: quanto mais branco, maior a confiança de que aquele pixel em destaque pertence a um objeto em movimento.

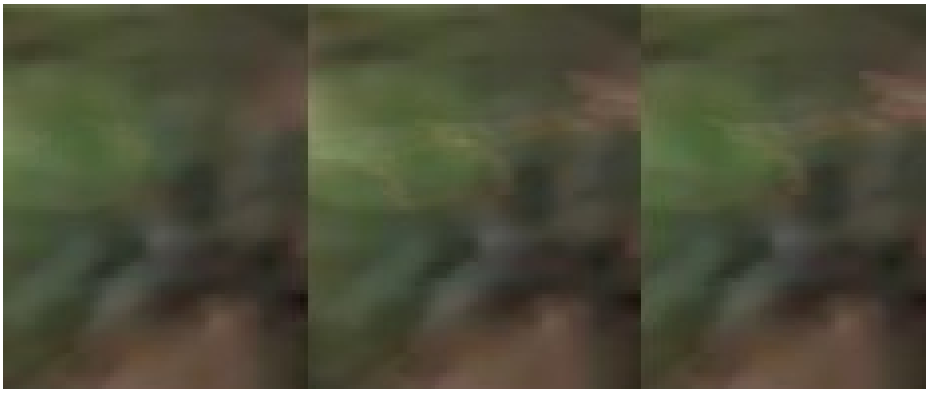


Figura 13: Aproximação na região de interesse de fundo estimado em diferentes quadros, é possível observar a pós imagem de uma abelha passando

8.1.2.2 Desfoque Gaussiano

É uma técnica que busca aumentar a suavidade de uma imagem por meio da aplicação de um filtro gaussiano, ou seja, é uma convolução de uma imagem onde a matriz é definida a partir de uma função Gaussiana. Embora o desfoque gaussiano seja uma técnica bastante poderosa, os parâmetros utilizados para tratamento da imagem buscaram minimizar alterações na imagem resultante e tinham como objetivo apenas remover pequenos artefatos do vídeo.

8.1.2.3 Máscara Binária

A aplicação de uma máscara binária consiste em, a partir da imagem em escala de cinza, convertê-las para preto e branco, onde os valores dos pixels são arredondados com base em um limiar definido. Logo, se os pixels estiverem abaixo do limiar, eles serão pretos, senão, serão brancos.

O resultado desta e das etapas seguintes é bastante sensível ao valor do limiar escolhido, dado que um valor elevado irá remover toda a parte da imagem onde não há uma confiança elevada, em um primeiro momento isso pode parecer interessante, porém é importante ressaltar que isso tornaria a análise ineficiente em ambientes onde a diferença de cor entre o foreground e o background são menores.

Para o propósito da detecção de abelhas, um valor obtido empiricamente para o limiar foi de 160. Embora valores convencionais para a máscara binaria tendam a ser maiores, isso aumenta a garantia que nenhuma abelha irá passar despercebida. Por outro lado, alguns ruídos ou movimentos imprevistos não serão eliminados nesta etapa.

8.1.2.4 Abertura

Por fim, para concluir o tratamento da imagem e iniciar o reconhecimento, é aplicada a técnica de abertura. A abertura consiste na aplicação das operações morfológicas de erosão seguida de dilatação. De forma sucinta, a erosão consiste redução da largura dos contornos e objetos enquanto a dilatação é responsável por extravasar os contornos.

Portanto, a abertura causa um desgaste seguido de reconstrução dos componentes da imagem, levando a remoção de linhas e ruídos. Assim, resultando em uma imagem onde os objetos em movimento (abelhas) estão em branco.

8.1.2.5 Reconhecimento

Para o reconhecimento das abelhas dentro das imagens são aplicadas técnicas de encontro de contorno sobre os quadros resultantes da pipeline de transformações morfológicas. Assim, são retornadas caixas delimitadoras onde as abelhas estão.

Em seguida, é aplicada uma heurística para excluir falsos positivos das caixas retornadas: elas devem ter ao menos 300 pixels de área. Isso evita que ruídos ou objetos ao menores fundo sejam indevidamente classificados como abelhas. Ademais, foram testadas outras heurísticas para este mesmo propósito: definir um tamanho máximo e definir uma razão máxima entre os lados da caixa delimitadora. Embora essas técnicas, de fato, reduzissem o

número de falsos positivos, elas também aumentavam o número de falsos negativos, o que levou ao descarte das opções.

Figura 14: Caixas delimitadoras aplicadas sobre os quadros dos vídeos após a detecção

8.1.3 Aprendizado de máquina

Uma abordagem alternativa à pipeline de operações morfológicas para a detecção de abelhas foi a utilização de modelos de aprendizado de máquina especializados na detecção de objetos. O modelo escolhido foi o YOLO, que apresenta um bom equilíbrio entre performance, acurácia e velocidade de treino.

8.1.3.1 YOLO

YOLO [20] é um conjunto de modelos bastante consolidados atualmente, com um de seus principais usos sendo a detecção de objetos em vídeos, por conta de operar em tempo real (30 FPS) em hardwares mais acessíveis e em uma resolução de 640x640 pixels.

A performance vem do fato de YOLO se tratar de um *single-shot object detector*, isto é, o algoritmo passa apenas uma vez pela imagem original. A partir daí, em conjunto com uma rede neural convolucional, ele gera as predições.

8.1.3.2 Dataset

Para que fosse possível fazer a detecção das abelhas jataí, era necessário um dataset no qual o modelo seria treinado. Um dos desafios iniciais foi que, embora existissem inúmeros datasets cobrindo abelhas africanas, não foi encontrado nenhum disponível publicamente que continha anotações de abelhas jataí. Portanto, foi necessário montar o dataset do zero.

Foi utilizada a ferramenta Roboflow para a anotação manual e subsequente automação com inteligência artificial. A automação consistiu em treinar um modelo com algumas das imagens anotadas manualmente e utilizá-lo para anotar imagens novas que seriam revisadas manualmente posteriormente. Embora o modelo de anotação apresente suas limitações, foi possível acelerar a anotação de imagens em até 10 vezes em alguns casos, aumentando também a qualidade das anotações. O resultado final consistiu em uma base de 1035 imagens com aproximadamente 7600 abelhas anotadas.

Outro ponto importante para entender a eficácia do resultado é compreender como o dataset foi montado. O conteúdo foi escolhido com o intuito de cobrir várias situações distintas para garantir a acurácia do modelo em situações comuns ou adversas, tais quais: iluminação, tipo de fundo, tamanho e velocidade das abelhas. Isso foi possível pois os seguintes fatores foram variados: locais de gravação; horário do dia; câmera; distância até a colmeia; ângulo de gravação e velocidade das abelhas.

Ademais, o dataset e o modelo evoluíram em conjunto: conforme o modelo apresentava limitações, mais imagens eram inseridas ou removidas para cobrir os casos onde a performance era subótima. Inicialmente o dataset era majoritariamente composto por imagens próximas das abelhas com fundos simples, fazendo com que o modelo ignorasse abelhas pequenas ou em fundos complexos. Para solucionar isso, foram inseridas imagens que cobrissem esses casos. Outra limitação encontrada posteriormente foi que o modelo não detectava adequadamente as abelhas em movimento. Portanto, foram inseridas mais imagens com abelhas em movimento, assim como a utilização de técnicas de aumento, especialmente de borrão (gaussian blur).

Por fim, além das imagens, foram utilizadas técnicas de aumento do dataset para reduzir overfitting e aumentar a acurácia do modelo. A versão final após os aumentos era composta por 2573 imagens (640x640 pixels), no qual a divisão entre treino, validação e teste foi de 2307, 171 e 95. É importante observar que o aumento é apenas sobre o treino, por isso a discrepância aparente. As técnicas de aumento utilizadas foram: rotação de 90 graus; rotação de até 15 graus; variação de brilho; aplicação de ruído e desfoque. Essas técnicas foram escolhidas pois geram imagens novas com orientações e características que, de fato, poderiam ser encontradas.

8.1.3.3 Especificações e Treinamento

Com relação ao modelo, foi utilizado o YOLOv10 nano, por se tratar do modelo mais recente com maior performance. O treinamento foi incremental e feito duas vezes em uma Tesla T4 sobre a versão mais recente do dataset previamente descrito com os seguintes valores para os hiperparâmetros principais: 30 épocas; batches de 85 imagens; imagens de 640x640; optimizador Adam.

O resultado desses treinamentos gerou um modelo que satisfatoriamente detecta abelhas em cenários diversos. Porém, há algumas limitações nos resultados associados ao dataset, natureza do problema e funcionamento do modelo.

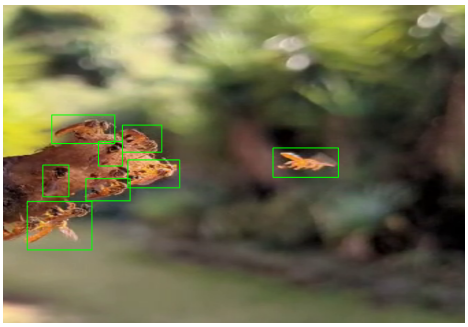


Figura 15: Predições do modelo

É importante salientar algumas das vantagens e desvantagens da utilização da detecção de objetos utilizando o modelo YOLO:

- **Performance:** Comparado à pipeline de transformações morfológicas, a performance do modelo de aprendizado de máquina gasta mais memória e oferece performance inferior no melhor caso (ambas rodando na GPU).

- **Comportamento com fundos simples:** Em fundos simples onde as abelhas estão claramente visíveis, o modelo YOLO performa extremamente bem, sendo o cenário de melhor caso.

- **Comportamento com fundos complexos:** Este é um dos casos no qual algumas das limitações principais desta técnica se apresentam. Como o modelo analisa as imagens individualmente e não armazena contexto, abelhas que se camuflem no fundo não serão identificadas, dado que é impossível distingui-las.

- **Qualidade da predição de abelhas em diferentes velocidades:** Este é um ponto que se mostrou bastante contraintuitivo e gerou certa surpresa. A princípio pensou-se que o modelo não iria conseguir detectar abelhas em alta velocidade, pois elas se assemelham muito a um borrão. Porém, por conta das altas velocidades, observou-se que algumas das câmeras possuíam artefatos de gravação onde as abelhas são claramente distintas de outros objetos, mesmo sendo impossível associar o formato com uma jataí. Na Fig. 16 é possível ver um artefato retangular em volta das abelhas. Embora seres humanos não consigam detectar isso claramente, o modelo utiliza esse artefato para aumentar a confiança da predição.

Figura 16: Imagens de abelhas com artefatos visíveis

8.2 Rastreamento

Independente do método de detecção utilizado, conseguimos as bounding boxes das abelhas nos quadros dos vídeos. A partir daí é necessário fazer o rastreamento delas para poder efetuar a contagem. Atualmente há uma série de algoritmos que lidam com o rastreamento de objetos.

8.2.1 DeepSort

O DeepSort é um algoritmo estado da arte de rastreamento de múltiplos objetos que utiliza redes neurais profundas para associar identificadores (Id) aos objetos em cena e identifica-los através de diferentes quadros. Embora seja um técnica bastante eficiente, os testes envolvendo DeepSort não apresentaram resultados satisfatórios, dado que é necessário que os objetos fiquem alguns quadros em cena para terem um Id atribuído, o que não ocorre considerando a alta velocidade das abelhas e o enquadramento próximo à colmeia (necessário para visualizar as abelhas). Portanto, as abelhas saiam muito rápido do vídeo e, embora fossem detectadas, não era possível rastrea-las.

8.2.2 Técnicas Ingênuas

Técnicas ingênuas ("naive") consistem em algoritmos que assumem cenários muito simples e tentam extrapolar para o caso geral, geralmente se tratando de soluções simples. Muitas vezes é possível abstrair um cenário complexo em casos mais simples e obter um resultado satisfatório.

Uma das técnicas testadas é o que chamaremos de "similaridade por proximidade", que é consiste em aplicar um identificador as abelhas que entram em quadro e assumir que as bounding boxes mais próximas entre quadros consecutivos pertencem a mesma abelha. Essa técnica se mostrou bastante eficiente para casos onde há até 2 abelhas e elas estavam

distantes. Porém, para os demais casos ela era ineficiente, tendo dificuldade em diferenciar abelhas novas e abelhas saindo de quadro.

8.2.3 Distribuição de Probabilidade

Com base nos resultados dos testes anteriores, decidiu-se que era necessário desenvolver um rastreador próprio específicos para o problema que tínhamos. De forma geral, precisávamos de um rastreador que:

- Lidasse com objetos que se deslocassem muitas vezes o seu comprimento entre quadros.
- Fosse resiliente com objetos que fiquem poucos quadros em vídeo.
- Mantivesse a identidade entre objetos que possam ou não ter alguma intersecção entre quadros.
- Lidasse com múltiplos objetos que poderiam estar próximos uns dos outros.

Para começar o desenvolvimento das soluções, primeiro foi necessário simplificar ao máximo o que seria necessário para efetivar o processo subsequente de contagem. Primeiramente, não era necessário atribuir um identificador para as abelhas, pois não tínhamos interesse em rastrear o caminho todo de uma abelha, apenas o seu deslocamento entre dois quadros. Portanto, o problema pode ser entendido como identificar o deslocamento das abelhas (bounding boxes) entre os quadros $N-1$ e N, pois não tínhamos interesse na individualidade das abelhas, apenas se elas cruzariam algum limiar (será detalhado em breve).

Desta forma, o primeiro passo foi avaliar a situação do ponto de vista do rastreador. Ele recebe as informações das bounding boxes para os quadros N (atual) e $N-1$ (anterior), sendo essa toda a informação que ele tem para tomar as decisões, puramente geométrica e sem quaisquer atributos dos objetos.

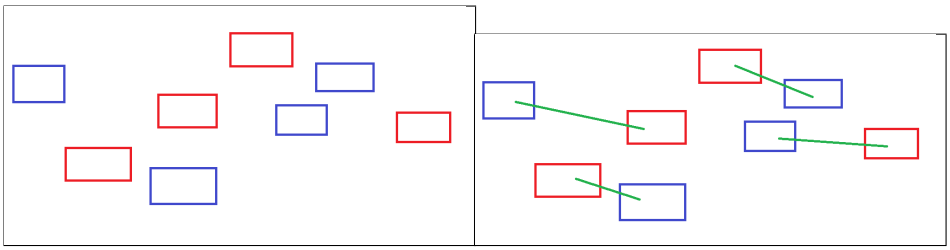


Figura 17: Posição das bounding boxes do quadro $N-1$ (Azul) e N (Vermelho). Respectivamente: entrada recebida e saída esperada do rastreador, onde as linhas verdes ligam um mesmo objeto em N e $N-1$

Com base na comparação da Figura 17, é possível observar que é complicado, senão, impossível fazer a associação de caixas corretamente sem informações adicionais. Após muita observação do comportamento das abelhas, notou-se que, embora o movimento longe

da entrada da colmeia seja caótico, ao se aproximar de uma região próxima à entrada o padrão de movimento se torna mais previsível, no qual:

- As abelhas saem da colmeia muito mais rápido do que entram.
- As abelhas dentro da região de interesse tendem a ir em direção a boca da colmeia quase que em linha reta.
- As abelhas raramente fazem movimentos perpendiculares a colmeia.
- As abelhas não mudam abruptamente de velocidade ou direção.

Com base nessas heurísticas e sabendo que as abelhas não podem desaparecer, a proposta de solução foi criar uma função $P(x, y, Cx, Cy)$, que para o centro (x, y) de uma bounding box no quadro N e a posição (Cx, Cy) da entrada da colmeia, retorna a probabilidade $P(0 < P \leq 1)$ de uma bounding box no quadro $N - 1$ pertencer ao mesmo objeto, ou seja, ser a mesma abelha. Com tal função, conseguiríamos mapear todas as abelhas em N e ver quais caixas de $N - 1$ equivalem a cada uma.

A entrada da colmeia está em:

$$(hive_x, hive_y)$$

O ângulo t, formado pela reta que conecta (c_x, c_y) e $(hive_x, hive_y)$ com a horizontal, é dado por:

$$t = -\arctan\left(\frac{c_y - hive_y}{c_x - hive_x}\right)$$

As coordenadas transladadas e rotacionadas são:

$$x_1 = x - c_x, \quad y_1 = y - c_y$$

$$x_2 = x_1\cos(t) - y_1\sin(t), \quad y_2 = y_1\cos(t) + x_1\sin(t)$$

O plano que corta as elipses ao meio é dado por:

$$k_x = \begin{cases} \frac{1}{t}x + \left(c_y - \frac{c_x}{t}\right) & \text{se } t \neq 0 \\ \infty & \text{caso contrário} \end{cases}$$

As metades das elipses são definidas como:

$$f_{xy} = \begin{cases} -\left(\frac{x_2}{a}\right)^2 - \left(\frac{y_2}{b}\right)^2 + 1, & \text{se } c_y > hive_y \text{ e } k_x \geq y \\ -\left(\frac{x_2}{c}\right)^2 - \left(\frac{y_2}{b}\right)^2 + 1, & \text{se } c_y > hive_y \text{ e } k_x < y \\ -\left(\frac{x_2}{a}\right)^2 - \left(\frac{y_2}{b}\right)^2 + 1, & \text{se } c_y \leq hive_y \text{ e } -k_x \geq -y \\ -\left(\frac{x_2}{c}\right)^2 - \left(\frac{y_2}{b}\right)^2 + 1, & \text{se } c_y \leq hive_y \text{ e } -k_x < -y \end{cases}$$

Finalmente, aplica-se uma transformação exponencial para suavizar a curva e manter a imagem da função entre 0 e 1:

$$\mathrm{P(x,y)} = \frac{\exp(f_{xy})}{e}$$

Dada a equação apresentada acima, vamos descrever como ela foi obtida e as motivações por trás de cada passo:

I - A princípio, foi criada a equação de uma elipse centralizada na origem.
II - A elipse é transladada para ficar na posição (x, y) da abelha.
III - A elipse é rotacionada de forma que seu semi-eixo maior fique alinhado com a colmeia.
IV - É criado K, um plano que sempre divide a elipse no semi-eixo menor.

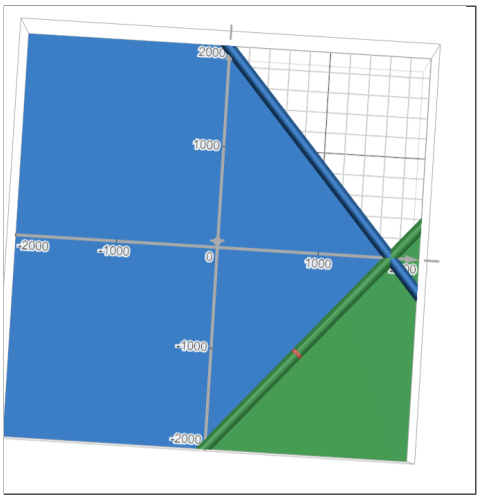


Figura 18: Plano K e sua normal para valores acima (azul) e abaixo (verde) da colmeia. É possível observar a inversão de direção.

V - A elipse é dividida em duas semi-elipses menores que se encontram no semi-eixo menor (divididas por K) de mesmo tamanho. Vamos chamar de E_a a elipse próxima a colmeia e de E_b a elipse mais distante. Onde o semieixo maior de E_b é menor que o semieixo maior de E_a

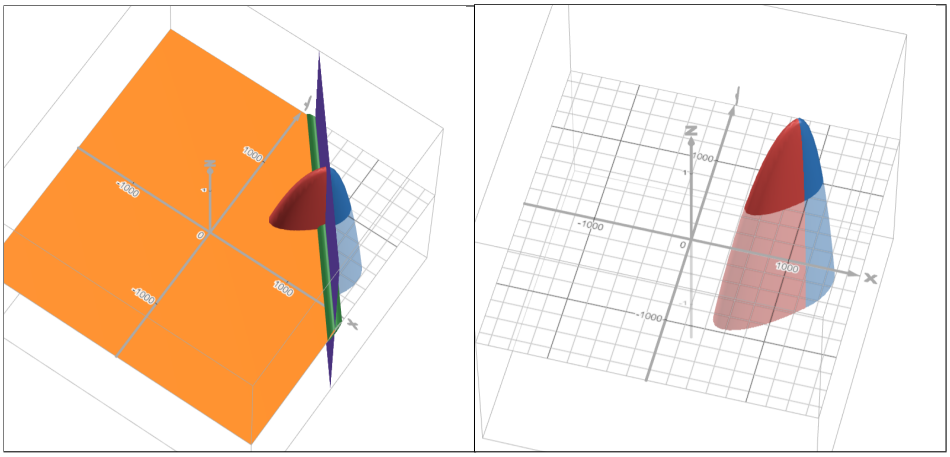


Figura 19: Respectivamente: $F(x, y)$ com o plano K e a reta $y = k(x)$ visíveis; apenas $F(x, y)$

VI - $f(x, y)$ é dada pela união de E_a e E_b onde a reta $y = k(x)$ determina qual elipse vai ser responsável pela imagem de f. Resultando, portanto, em uma elipse assimétrica no semi-eixo menor. É necessário observar que a reta $y = k(x)$ possui sua tangente apontando a favor ou contra a origem, dependendo se a abelha possui um y maior ou menor que a colmeia, por isso é necessário tratar esses dois casos separadamente.
VII - Por fim, aplica-se uma transformação exponencial para limitar a imagem da função entre 0 e 1.

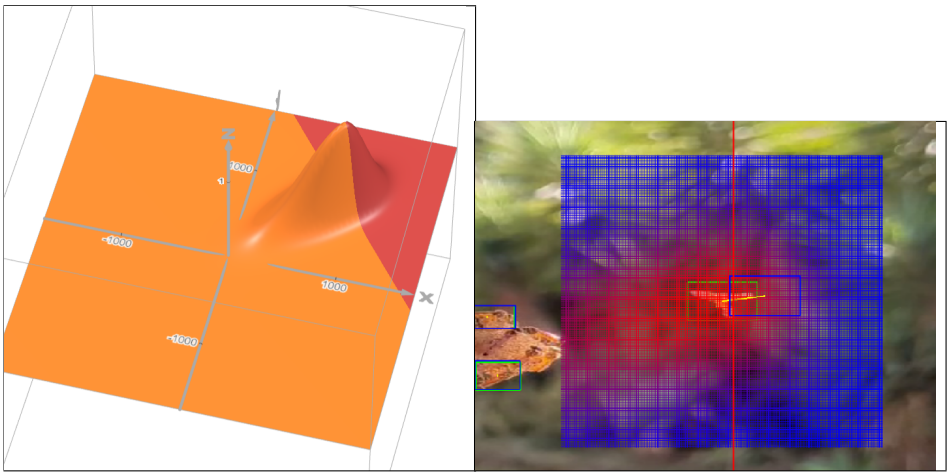


Figura 20: $P(x, y)$ respectivamente em um plano tridimensional e sobreposta sobre uma abelha, onde a cor varia de vermelho até azul conforme a probabilidade decresce.

8.3 Fluxo de Abelhas

O objetivo destes processos de detecção e rastreamento culmina na possibilidade de contagem das abelhas que entram e saem da colmeia. Como a detecção das abelhas do lado de dentro da colmeia era impossível, foi considerado que uma abelha entra ou sai na colmeia de acordo com a direção que ela cruza uma linha horizontal centralizada no quadro. Essa decisão foi tomada com base em observação empírica, visto que dentro de uma zona de interesse próxima da colmeia, as abelhas que atravessavam essa linha quase sempre estavam entrando ou saindo da colmeia.

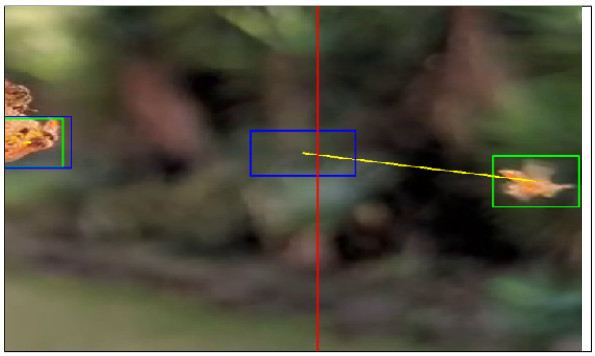


Figura 21: Abelha cruzando a linha de detecção, é possível observar a sua posição no quadro $N-1$ em azul.

Portanto, quando alguma abelha cruza essa linha em direção oposta a colmeia, ela é contabilizada como saindo. Caso o movimento seja em direção a colmeia, é considerado que ela entrou.

9 Documentação

O código gerado ao longo do projeto está no repositório público hive-mind [16]. Sua estrutura de diretórios é descrita a seguir.

- **backend-node:** Arquivos relativos à API, como controladores, modelos e rotas, incluindo os *schemas* do banco de dados.
- **backend-python:** Arquivos relacionados ao processamento dos vídeos, incluindo o tracker e detectores implementados.
- **embedded:** Arquivos inseridos nas placas para coleta e envio dos dados.
- **frontend:** Arquivos relativos à aplicação Web, como rotas, componentes e hooks.

Adicionalmente, a descrição, incluindo firmwares e documentação, da principal placa utilizada para desenvolvimento e testes está presente no repositório BitDogLab [15].

10 Resultados

10.1 Quantitativos - Backend Node JS

Afim de validar a performance da aplicação em situações reais, foram realizados benchmarks utilizando a ferramenta K6 [6] para similar situações de estresse na aplicação. O K6 é uma ferramenta open source extensível para realizar testes de carga a partir de scripts escritos em JavaScript. Definem-se as métricas de interesse para o endpoint, por exemplo taxa de falhas, e executa-se o script com um número arbitrário de Usuários Virtuais (vus) por um determinado tempo. Ademais, é possível estágios de teste simulando picos de requisições no tempo.

Testar todos os endpoints é pouco eficiente e, por isso, optou-se por testar somente aqueles cuja performance é crítica: GET metrics e POST metrics. Além disso, pela generalidade da implementação, testar endpoints para uma métrica, como a temperatura, é representativo da performance para as outras.

Além das métricas padrões da ferramenta, foram medidas: duração médias das requisições, taxa de sucesso, taxa de falha e número total de requisições realizadas para os endpoints GET e POST. O teste foi executado com um pico de carga constante gerado por 1200 usuários virtuais em um período de 5 minutos e foi considerado que uma chamada era falha ao levar mais de 1s para retornar uma resposta.

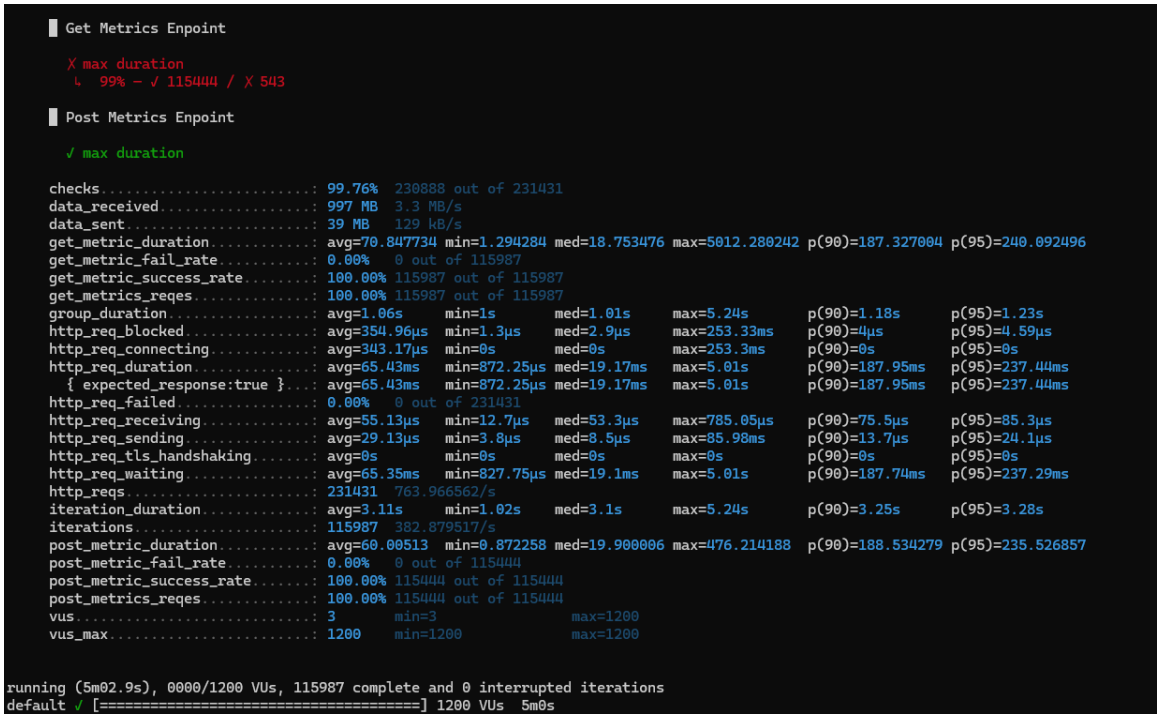


Figura 22: Resultados para benchmarks de endpoints GET e POST para métricas utilizando a ferramenta K6 para simular um pico de carga de 1200 usuários por um período de 5 minutos.

Considerando os resultados apresentados na Fig. (22), é possível verificar que 98% das chamadas para o endpoint GET foram bem sucedidas e 100% das chamadas POST retornaram em tempo adequado. Executando o script é possível que as falhas no endpoint GET ocorram somente no início do teste, demonstrando as capacidades de cache do sistema que o tornam mais capaz de responder a altas cargas de estresse. Vale ainda mencionar que foi possível obter um médias de 763.97 requisições/s. Existe espaço para otimizações que poderiam aumentar essa taxa, mas considerando a complexidade da implementação e os frameworks utilizados ela é satisfatória.

10.2 Quantitativos - Detecção de Abelhas

Por fim, com propósito de verificar as diferenças de resultados entre os algoritmos de detecção de abelhas e criar uma expectativa quanto ao tempo médio de execução, foram feitos alguns testes quantitativos. Para execução dos testes foram utilizados dois vídeos. Em ambos os casos a colmeia encontra-se a esquerda do vídeo, como descrito nas instruções de uso. Vale ainda mencionar que os testes foram executados em um computador com processador i7-10750H e uma GPU RTX 2060.

Algoritmo de Detecção	**Entrada**	**Duração do Vídeo (s)**	**Previsões**		**Real**	
			Entrada	**Saída**	**Entrada**	**Saída**
Background Subtractor	Vídeo 1	11	3	4	6	11
Rede Neural (YOLO)	Vídeo 2	45	13	15	13	14

Tabela 1: Comparação de desempenho em termos da diferença entre valores reais e previstos para ambos algoritmos de detecção de abelhas.

Algoritmo de Detecção	**Tempo do Vídeo (s)**	**Tempo de Execução do Algorítmo (s)**
Background Subtractor	11	16.447
	45	71.7
Rede Neural (YOLO)	45	38.349

Tabela 2: Comparação de desempenho em termos da tempo de execução para os algoritmos de detecção de abelhas.

Em relação ao algoritmo de subtração de fundo, os resultados de contagem obtidos para o vídeo 1 se explicam pelo fato de que o vídeo não foi gravado nas condições previstas pelo algoritmo. Neste caso, a câmera se mexia muito, prejudicando a detecção das abelhas. Já o vídeo 2 foi utilizado somente para propósitos de tempo de execução, como as condições não permitiriam a aplicação da técnica de detecção de forma eficiente. É importante observar também que esse mecanismo roda majoritariamente na CPU.

Quanto ao algoritmo baseado na rede neural YOLO, os resultados foram bastante satisfatórios. Em sua maior parte porque o modelo de detecção funcionou muito bem já que o vídeo se encaixava nos pré-requisitos dessa técnica. Como a detecção funcionou de

forma satisfatória, o rastreamento e contagem foram bastante eficientes. Com relação a performance, a execução ocorre em tempo real e é majoritariamente processada na GPU.

10.3 Qualitativos

Além dos resultados numéricos, vale destacar alguns pontos fortes do sistema desenvolvido.

Primeiramente, a arquitetura foi totalmente reformulada, tendo em mente a expansibilidade do sistema. Ao deixar de utilizar o Firebase para backend e armazenamento de dados, o sistema se torna escalável sem adquirir um custo monetário, viabilizando seu uso educacional. O uso de um banco de dados não relacional, com *schemas* bem definidos e indexados por identificadores também garante que o sistema possa operar em escalas muito maiores que as testadas.

Em relação aos anos anteriores, a aplicação Web também foi melhorada. Anteriormente, havia um único gráfico com todas as medições, mesmo que em escalas diferentes. Para melhorar a visualização dos dados, ele foi substituído por uma variedade de gráficos mais especializados. Ademais, a aplicação foi desenvolvida tendo em mente a escalabilidade do sistema, podendo filtrar por diferentes colmeias e sensores para permitir análises mais amplas ou específicas.

Também foram feitas melhorias consideráveis ao software da placa. A remoção das coletas de dados da API OpenWeatherMap, utilizada em versões anteriores do projeto, reduz a carga posta sobre o processador. Assim, a própria placa se torna mais escalável, com um mesmo microcontrolador podendo ser responsável por mais sensores, reduzindo possíveis custos com a fabricação de novas placas. Adicionalmente, foi integrado o sensor sonoro, ausente em trabalhos anteriores.

Por fim, a principal adição foi o processamento de vídeos. Com essa ferramenta, se tornam possíveis análises mais profundas de como as medidas obtidas impactam o comportamento das abelhas.

11 Modo de Uso

Em preparação para o primeiro uso deve-se conectar a placa a um computador e transferir o software da placa configurando no arquivo `main.py` informações gerais como nome e senha da rede Wi-Fi a ser utilizada e o endereço da API que receberá os dados coletados pela placa. Em seguida, basta iniciar o software com a placa conectada a uma fonte de energia, seja esse uma tomada ou uma bateria, e a placa vai passar pelo ciclo inicial se cadastrando no sistema. Finalizado a ciclo inicial de cadastro do dispositivo IoT os dados serão enviados para o sistema em intervalos de 15 minutos e estarão disponíveis em forma de gráficos e tabelas na aplicação Web.

Quanto ao processamento de vídeos para contagem de abelhas desenvolvido nesse projeto, basta que a aplicação Web esteja disponível que será possível realizar o upload dos arquivos para processamento. Para tanto basta acessar a seção "Monitoramento de Vídeos"na aplicação web e enviar o video de interesse. Após o envio na área de resultados, estará disponível uma entrada na tabela identificando o vídeo e assim que seu status de processamento

seja "Processado", clicar nessa linha da tabela vai redirecionar o usuário a página de resultados específica para aquele vídeo. Vale adicionar que um vídeo pode ser analisado mais de uma vez e o uso de diferentes algoritmos de detecção é aconselhável para validar resultados.

12 Conclusão e Trabalhos Futuros

Ao longo deste projeto, foi desenvolvida uma nova versão do sistema de monitoramento de colmeias. Ele é composto por um servidor para armazenamento e processamento de dados e uma aplicação web para visualização dos dados e upload de vídeos. O sistema é capaz de se conectar por Wi-Fi a diversas placas, cada uma com um microcontrolador Raspberry Pi Pico W e diversos sensores, permitindo coleta de dados ambientais.

Em suma, os objetivos definidos no início do projeto foram concluídos. Foi desenvolvido um sistema robusto e completo, com funcionalidades que apoiam os fins educacionais estabelecidos. Além de manter as capacidades de versões anteriores do projeto, foi adicionado o processamento de vídeos para contagem das abelhas, integrando técnicas de inteligência artificial e visão computacional. Adicionalmente, a escalabilidade do sistema como um todo se tornou muito maior, permitindo lidar com mais colmeias e sensores sem um aumento considerável de custo.

Apesar dos objetivos terem sido atingidos, foram identificados pontos de melhoria a serem considerados em futuros trabalhos. Tendo em mente a meta da escalabilidade, a base de dados poderia ser modificada para permitir diversos usuários simultâneos enquanto mantém suas informações privadas através de um sistema de autenticação. Para melhorar a explicabilidade dos resultados, seria interessante retornar ao usuário quadros do vídeo onde as abelhas foram detectadas, assim como métricas de acurácia e confiança. Além disso, o sistema poderia ser melhorado para se tornar mais resistente a falhas, com uma fila de requisições e cacheamento de respostas.

Referências

[1] J. C. Gonçalves e D. M. Dos Santos e L. J. S. Dos S. P. Monroe e V. A. Scholze e F. Fruett e L. F. Bittencourt. *Monitoramento de colmeias com Internet das Coisas*. Rel. técn. 2024. URL: `https://www.ic.unicamp.br/~reltech/PFG/2024/PFG-24-03.pdf`.

[2] *Docker Compose*. Acesso em 08 de Dezembro de 2024. URL: `https://docs.docker.com/compose/`.

[3] Voise Eletronics. *CNZ-15E - Datasheet*. Acesso em 20 de Novembro de 2024. 2020. URL: `https://www.quartz1.com/price/techdata/VS9767S32(CZN-15E).pdf`.

[4] *Express*. Acesso em 08 de Dezembro de 2024. URL: `https://expressjs.com/`.

[5] *FastAPI*. Acesso em 08 de Dezembro de 2024. URL: `https://github.com/fastapi/fastapi`.

[6] *K6*. Acesso em 08 de Dezembro de 2024. URL: `https://k6.io/`.

[7] M. C. Rosa e L. C. Castello e C. A. A. Trujillo e L. F. Bittencourt. *Monitoramento de colmeias com Internet das Coisas*. Rel. técn. 2023. URL: https://www.ic.unicamp.br/~reltech/PFG/2023/PFG-23-28.pdf.

[8] Raspberry Pi Ltd. *Raspberry Pi Pico W - Datasheet*. Acesso em 20 de Novembro de 2024. 2024. URL: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf.

[9] *MicroPython*. Acesso em 08 de Dezembro de 2024. URL: https://micropython.org/.

[10] *Micropython Driver for a BME680 breakout*. Acesso em 08 de Dezembro de 2024. URL: https://github.com/robert-hh/BME680-Micropython.

[11] *MongoDB*. Acesso em 08 de Dezembro de 2024. URL: https://github.com/mongodb/mongo.

[12] Maxim Integrated Products. *MAX4466 - Datasheet*. Acesso em 20 de Novembro de 2024. 2012. URL: https://makerhero.com/img/files/download/MAX4466-Datasheet.pdf.

[13] F. Fruett e R. Greco. *Multiple sensor beehive monitoring*. Acesso em 07 de Dezembro de 2024. URL: https://docs.google.com/document/d/1Mz-MCR6UlPnqEdyOTakaD4TGH3IIKm6izRvWRS6

[14] *React*. Acesso em 08 de Dezembro de 2024. URL: https://react.dev/.

[15] *Repositório BitDogLab*. Acesso em 07 de Dezembro de 2024. URL: https://github.com/BitDogLab/BitDogLab.

[16] *Repositório hive-mind*. Acesso em 07 de Dezembro de 2024. URL: https://github.com/Jonhyog/hive-mind/.

[17] Bosch Sensortec. *BME680 - Datasheet*. Acesso em 20 de Novembro de 2024. 2024. URL: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf.

[18] *Shadcn*. Acesso em 08 de Dezembro de 2024. URL: https://ui.shadcn.com/.

[19] G. J. S. Moraes e V. A. M. Dantas e A. C. L. C. C. F. Renoldi e H. F. Zimerman e P. P. Alves e J. V. F. Costa e L. S. L. Carmo e L. F. Bittencourt. *Monitoramento de colmeias com Internet das Coisas*. Rel. técn. 2022. URL: https://www.ic.unicamp.br/~reltech/PFG/2022/PFG-22-37.pdf.

[20] *YOLO*. Acesso em 08 de Dezembro de 2024. URL: https://docs.ultralytics.com/pt.