



FACULDADES E ESCOLAS TÉCNICAS QI

BRUNA GISELE PIRES DA SILVA

BRUNA.GPIRES.S@GMAIL.COM

Gerenciador de estoque com ESP32-CAM

NOVO HAMBURGO

2024

BRUNA GISELE PIRES DA SILVA

Gerenciador de estoque com ESP32-CAM

Gerenciador de estoque com ESP32-CAM é apresentado como requisito parcial para a aprovação na disciplina de Projeto Aplicado, na Faculdade e Escola Técnicas QI.

Orientador: Prof. Esp. Silvio Cesar Viegas

NOVO HAMBURGO

2024

RESUMO

Esse trabalho tem como objetivo demonstrar os conhecimentos adquiridos ao longo do curso Sistemas para Internet. Trata-se de um projeto de desenvolvimento de software que foca em um sistema inteligente de gerenciamento de estoque, substituindo o tradicional código de barras por um processo automatizado de reconhecimento de objetos. Para a captura de imagens, é utilizado o ESP32-CAM, enquanto o sistema usa TensorFlow Lite para identificar os itens em tempo real, possibilitando a atualização automática e precisa do estoque. Esse método torna o processo mais eficiente e oferece uma solução moderna e escalável para otimizar a gestão de inventário.

O software desenvolvido utiliza a linguagem de programação C++, amplamente usada na programação de microcontroladores. Além disso, o Edge Impulse é empregado para o treinamento de modelos de *machine learning* (aprendizado de máquina), permitindo a identificação dos itens. Como resultado dessa pesquisa, o sistema está disponível em sua primeira versão.

Palavras-chave: ESP32, Edge Impulse, gerenciamento de estoque, automação, reconhecimento de objetos, *machine learning*, C++.

LISTA DE ILUSTRAÇÕES

FIGURA 1 - DIAGRAMA DE ATIVIDADES (FONTE: LUCIDCHART).....	16
FIGURA 2 - DIAGRAMA CASOS DE USO (FONTE: LUCIDCHART).....	17
FIGURA 3 - DIAGRAMA DE CASOS DE USO DO SISTEMA DE GERENCIAMENTO DE ESTOQUE	21
FIGURA 4 - DIAGRAMA DE ATIVIDADES DO SISTEMA DE GERENCIAMENTO DE ESTOQUE.....	22
FIGURA 5 - CONECTANDO COMPONENTES AO ESP32-CAM	23
FIGURA 6 - SELECIONANDO O MODELO DA PLACA ESP	25
FIGURA 7 - EDIÇÃO DO ARQUIVO CONV.CPP	25
FIGURA 8 – EDIÇÃO DO ARQUIVO DEPTHWISE_CONV.CPP	26
FIGURA 9 - CÓDIGO DE TESTE DO BANCO DE DADOS	27
FIGURA 10 - CÓDIGO DE TESTE DA PORTA SERIAL	27
FIGURA 11 - CÓDIGO FINAL DO SISTEMA	28

SUMÁRIO

1.	INTRODUÇÃO.....	8
1.1	Tema.....	8
1.2	Delimitação do tema	9
1.3	Problema	9
1.4	Justificativa	9
1.5	Objetivo geral.....	10
1.6	Objetivos específicos	10
2.	FUNDAMENTAÇÃO TEÓRICA	11
2.1	Gerenciamento	11
2.2	A visão computacional e o reconhecimento de objetos	11
2.3	Tecnologias utilizadas.....	12
2.3.1	C++	12
2.3.2	ESP32-CAM.....	13
2.3.3	Edge Impulse	14
2.3.4	Python.....	14
2.3.5	SQLite.....	15
2.4	Modelagem do sistema	15
2.4.1	Diagrama de atividade	16
2.4.2	Diagrama de casos de uso.....	16
2.5	Prototipagem em eletrônica.....	17
3.	METODOLOGIA	19
4.	SISTEMA DE GESTÃO DE ESTOQUE COM ESP32-CAM.....	20
4.1	Estrutura do sistema	20
4.1.1	Diagrama de caso de uso	20
4.1.2	Diagrama de atividades	21
4.1.3	Diagrama de conexão dos componentes.....	22
4.2	Desenvolvimento do projeto	23
4.2.1	Treinamento de máquina	23
4.2.2	Enviando código para o ESP32-CAM	24
4.2.3	Banco de dados	26
4.2.4	Teste de conexão com banco de dados e porta serial	26

4.2.5 Código principal.....	28
5. CONSIDERAÇÕES FINAIS.....	30
REFERÊNCIAS.....	31

1. INTRODUÇÃO

A gestão eficiente de estoque é um fator crucial para o sucesso de qualquer empreendimento. O controle inadequado de mercadorias pode gerar severas consequências negativas, como o acúmulo excessivo de produtos ou venda de itens que não possuem mais, resultando em prejuízos financeiros e operacionais. Tradicionalmente, a gestão de estoque tem sido realizada com métodos manuais ou utilizando códigos de barras.

Com os avanços tecnológicos e a crescente demanda por automação nos processos empresariais, soluções baseadas em inteligência artificial e visão computacional têm ganhado destaque. Essas tecnologias permitem o desenvolvimento de sistemas mais inteligentes e precisos, capazes de realizar o controle de estoque de maneira automatizada, proporcionando maior eficiência e precisão.

Este trabalho propõe o desenvolvimento de um sistema automatizado de gerenciamento de estoque para um empreendimento local, utilizando o ESP32-CAM para captura de imagens e Edge Impulse para reconhecimento de objetos. O sistema visa substituir os métodos tradicionais por um processo moderno e escalável.

1.1 Tema

O gerenciamento de estoque é essencial para a eficiência de qualquer empreendimento, sendo responsável por controlar o fluxo de mercadorias e garantir que o inventário reflita a realidade operacional. A automação nesse processo permite um controle mais preciso, reduzindo erros manuais e otimizando o tempo de reposição de produtos.

Diante disso, a implementação de soluções tecnológicas como o reconhecimento de objetos e visão computacional surge como uma alternativa eficaz aos métodos tradicionais, como o uso de códigos de barras. O desenvolvimento de um sistema inteligente de gerenciamento de estoque, utilizando ESP32-CAM e Edge Impulse, tem o potencial de transformar a forma como pequenos empreendimentos locais administram seus inventários, proporcionando maior agilidade, precisão e redução de custos operacionais.

1.2 Delimitação do tema

O sistema de gerenciamento de estoque será voltado para um empreendimento local, com o objetivo de automatizar o controle de inventário. O escopo inicial incluirá a implementação de reconhecimento de objetos através do ESP32-CAM, utilizando o Edge Impulse para identificação e atualização de itens em tempo real. O sistema também permitirá a geração automática de relatórios de vendas, facilitando o acompanhamento do desempenho de cada item e contribuindo para uma melhor gestão do estoque.

O foco principal da solução é otimizar a gestão de estoque, reduzindo erros manuais e melhorando a precisão nas reposições, com possibilidade de expansão futura para a inclusão de novos recursos e melhorias.

1.3 Problema

A ausência de um sistema de gerenciamento no empreendimento local resulta em desorganização na preparação dos pedidos, dificuldades no reestoque de itens e falta de informações sobre as preferências dos clientes e vendas realizadas.

1.4 Justificativa

Com o crescimento da base de clientes e o aumento da participação em eventos e feiras, o empreendimento enfrenta dificuldades na gestão manual das vendas e do estoque. A falta de um sistema estruturado leva a dificuldades na venda de itens que estão acabando, falta de visibilidade sobre os ganhos diários e a ausência de insights sobre os produtos mais populares. A implementação de um sistema automatizado de gerenciamento de estoque irá otimizar a organização, melhorar a eficiência nas operações e fornecer insights valiosos para o planejamento estratégico, contribuindo para um melhor desempenho operacional e financeiro.

1.5 Objetivo geral

Desenvolver um sistema de gerenciamento de estoque que automatize a atualização de itens e forneça relatórios detalhados sobre as vendas, otimizando o controle de estoque e melhorando a eficiência operacional.

1.6 Objetivos específicos

- a. Treinar o modelo de aprendizado de máquina utilizando Edge Impulse com uma base de dados de imagens dos produtos, para o reconhecimento e classificação das imagens enviadas.
- b. Implementar um sistema que utilize o ESP32-CAM para captura de imagens dos itens no momento da venda e enviar essas imagens a um servidor.
- c. Desenvolver um mecanismo no servidor para comparar as imagens recebidas com os itens registrados no banco de dados, atualizar o estoque e gerar relatórios diários e mensais de vendas.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Gerenciamento

O gerenciamento de estoque é uma prática fundamental para a organização e eficiência de qualquer negócio, pois envolve o controle de produtos desde a sua entrada até o momento da venda. Segundo Jaqueline Amaral Fraga (2024),

“Uma gestão eficiente de estoque possibilita que as empresas atendam aos pedidos com precisão e agilidade, o que garante a satisfação do cliente. Além disso, ajuda a reduzir custos operacionais e otimizar o fluxo de caixa, resultando em maior lucratividade para o negócio.”

Percebe-se que muitas empresas estão em busca de soluções automatizadas para a melhoria de precisão e eficiência no controle de estoque, pois a automação facilita a integração com outras áreas do negócio, como compra e vendas, assim permitindo uma visão unificada das operações. Dessa forma é possível evitar problemas como a falta de produtos ou o excesso de estoque.

2.2 A visão computacional e o reconhecimento de objetos

A visão computacional é uma das áreas da inteligência artificial, ela permite que máquinas processem e interpretem imagens ou vídeos, reconhecendo padrões, formas e objetos, de forma precisa e eficiente. Essa tecnologia tem sido aplicada em diversas áreas, como:

- **Segurança:** utilizada para a melhoria da segurança residencial, detectando visitantes ou entrega de encomendas.
- **Operacional:** pode identificar defeitos de qualidade de itens antes que saiam para venda, problemas de segurança em máquinas, analisar mídias sociais e descobrir tendências e autenticar funcionários com o reconhecimento facial.

Segundo a AWS (2023), “A visão computacional é uma tecnologia que as máquinas usam para reconhecer imagens automaticamente e descrevê-las com precisão e eficiência.”

O reconhecimento de objetos é uma técnica que utiliza modelos de machine learning para detectar e classificar itens em imagens capturadas. O processo de aprendizado dessas

máquinas envolve uma combinação de conceitos matemáticos e estatísticos, pois é por meio dessas técnicas que o computador consegue associar as formas das imagens fornecidas aos objetos que deve identificar. Para isso é fornecido imagens previamente classificadas, como uma foto de uma caneca acompanhada da legenda “caneca”. Por exemplo, assim como ocorre com os seres humanos quando crianças, que aprendem o que é uma bola por meio de repetidas exposições a esse objeto.

Ferramentas como o Edge Impulse e TensorFlow tornam possível a implementação dessa tecnologia em diversos dispositivos. Os algoritmos de machine learning são capazes de identificar padrões comuns nas imagens e vídeos capturados e aplicam o conhecimento para o qual foram treinados, identificando objetos com precisão.

2.3 Tecnologias utilizadas

Para o desenvolvimento de softwares é necessário utilizar linguagem de programação, que são uma forma de humanos instruírem máquinas a realizarem determinadas tarefas. Essas instruções são conhecidas como algoritmos.

2.3.1 C++

C++ é uma linguagem de programação criada por Bjarne Stroustrup em 1983. Essa linguagem de programação é uma extensão da linguagem C e suporta múltiplos paradigmas, mas o mais utilizado é a programação orientada a objetos (POO). C++ é amplamente utilizada em diversas áreas, como no desenvolvimento de sistemas operacionais, jogos, sistemas embarcados, software de alta performance e sistemas críticos. Segundo a Microsoft (2023),

“Desde sua criação, o C++ se tornou uma das linguagens de programação mais usadas do mundo, os programas bem escritos que a utilizam são rápidos e eficientes. Ele é mais flexível do que outras linguagens: pode funcionar nos níveis mais elevados de abstração e no nível do silício.”

Suas características permitem que desenvolvedores modelem o software como uma coleção de “objetos”, que representam entidades do mundo real e como interagem entre si, sendo eles:

- Classe: define um conjunto de atributos e funções.
- Objeto: uma instancia de uma classe.

- Herança, polimorfismo e encapsulamento: princípios que facilitam o reuso de código e a organização hierárquica dos componentes.

Como é uma linguagem compilada, o desenvolvedor tem o controle direto sobre o hardware de baixo nível, como o gerenciamento de memória e alocação de recursos, tornando-se ideal para sistemas que exigem alto desempenho. Dessa forma, é oferecido o controle detalhado da alocação e liberação de memória através de ponteiros e funções. Esse controle é importante para aplicações onde a eficiência de memória é crítica, mas também pode introduzir complexidades, como problemas com vazamento de memória.

2.3.2 ESP32-CAM

Esses pequenos dispositivos são microcontroladores utilizados em projetos de robótica, IoT e outras aplicações devido ao seu baixo consumo de energia.

O ESP32 é um processador de 32bits com dois núcleos que podem operar a até 240 MHz. Como sucessor do ESP8266, ele oferece maior versatilidade e poder de processamento, além de incluir uma antena integrada RF balun para comunicação via *Wi-Fi* e *Bluetooth*, e uma maior quantidade de GPIOs.

Seu diferencial é a inclusão de uma câmera, enquanto o chip de processamento permanece o mesmo utilizado em outras placas da Espressif. O ESP32-CAM possui uma entrada para cartão SD, permitindo o armazenamento de imagens, e vem equipado com uma câmera OV2640 de 2 megapixel.

Para programar essa placa, é necessário o uso de um conversor USB-Serial, que permite tanto a alimentação da placa quanto o envio do código. O ESP32 suporta programação nas linguagens C++, MicroPython e LUA.

Por sua placa ter AIoT (Inteligência Artificial e Internet das Coisas), ou seja, o módulo ESP32 tem suporte nativo para reconhecimento facial. Ele é amplamente utilizado em projetos que exigem monitoramento remoto por vídeo, como câmeras de segurança, bem como soluções voltadas para o reconhecimento de objetos e automação. Devido a sua conectividade *Wi-Fi* e *Bluetooth*, os desenvolvedores podem criar projetos com comunicação em tempo real.

A câmera OV2640, tem uma resolução de 2 megapixels, o que é suficiente para o seu tamanho. Pode ser feita a captura de imagens nos formatos JPEG e BMP, permitindo a flexibilidade no armazenamento e transmissão de dados. O ESP32-CAM pode ser programado

usando a IDE Arduino, uma plataforma acessível e muito utilizada pela comunidade de desenvolvedores. A IDE e a placa oferecem suporte para o uso de bibliotecas, que simplificam a captura e o processamento de imagens em tempo real.

2.3.3 Edge Impulse

Essa é uma plataforma desenvolvida para a criação, treinamento e implantação de modelos de *machine learning* (aprendizado de máquina) em dispositivos embarcados e IoT, seu foco é especialmente no *TinyML* (*machine learning* em dispositivos com recursos limitados, como microcontroladores). Permite que desenvolva soluções diretamente em dispositivos, ou seja, onde os dados são coletados, sem a necessidade de enviar grandes volumes de informações para a nuvem.

Seus pontos principais são:

A coleta de dados é feita diretamente de sensores conectados a dispositivos IoT como câmeras, acelerômetros, entre outros.

Um ambiente fácil de usar para treinar modelos de machine learning, com uma interface gráfica que auxilia na criação de algoritmos baseados em redes neurais, classificação de dados e processamentos de sinais. Após o treinamento, a plataforma gera o código necessário para a implementação no dispositivo, compatível com diversos microcontroladores e placas, como o ESP32, Raspberry Pi e até chips como o STM32. O código pode ser integrado diretamente no *firmware*, possibilitando a execução do modelo localmente.

Embora ofereça suporte a fluxos de trabalho automatizados, o Edge Impulse também permite que desenvolvedores avancem com modelos personalizados em *frameworks* populares como o TensorFlow Lite. É possível integrar modelos já treinados ou modificar os algoritmos conforme necessário.

2.3.4 Python

O Python é uma linguagem de programação, amplamente utilizada devido à sua simplicidade, legibilidade e versatilidade. Criada em 1991 por Guido van Rossum, ele é orientada a objetos, mas também suporta outros paradigmas de programação.

Algumas das principais características do Python, são:

- A ampla coleção de bibliotecas e frameworks que facilitam o desenvolvimento de diversas aplicações, como inteligência artificial, ciência de dados, automação e desenvolvimento web.
- É multiplataforma, ou seja, o mesmo código pode ser executado em diferentes sistemas operacionais, como Windows, macOS e Linux.

2.3.5 SQLite

O SQLite é um banco de dados relacional leve e de código aberto. Diferente de sistemas de banco de dados mais complexos, como MySQL ou PostgreSQL, o SQLite é um banco **embutido** que não requer um servidor separado para funcionar. Todos os dados são armazenados em um único arquivo no sistema de arquivos, tornando-o ideal para aplicações locais, dispositivos embarcados e aplicativos móveis. Como os dados são armazenados em um único arquivo, o banco de dados pode ser facilmente movido entre sistemas.

Esse banco de dados consome poucos recursos, sendo eficiente para aplicações que não precisam de um sistema de banco de dados robusto.

2.4 Modelagem do sistema

Modelagem de sistema é uma forma de visualizar, especificar e documentar os componentes de sistemas de software. O processo de modelagem ajuda a criar representações visuais da estrutura, comportamento e interações dos componentes de um sistema, facilitando o entendimento e design do sistema em desenvolvimento.

Uma ferramenta muito utilizada para a modelagem é a UML (Unified Modeling Language – Linguagem de Modelagem Unificada). Desenvolvida na década de 1990, a UML fornece um conjunto de diagramas que ajudam a representar a estrutura e o comportamento dos sistemas. Esses diagramas são fundamentais para a comunicação entre as partes interessadas e para a documentação do projeto. Os diagramas mais utilizados são o de: Atividade, Casos de Uso e Classes.

2.4.1 Diagrama de atividade

Esse padrão é utilizado para modelar o fluxo de atividades dentro de um processo. Ele é útil para visualizar a sequência de ações e decisões, permitindo identificar a lógica do processo de forma clara e concisa.

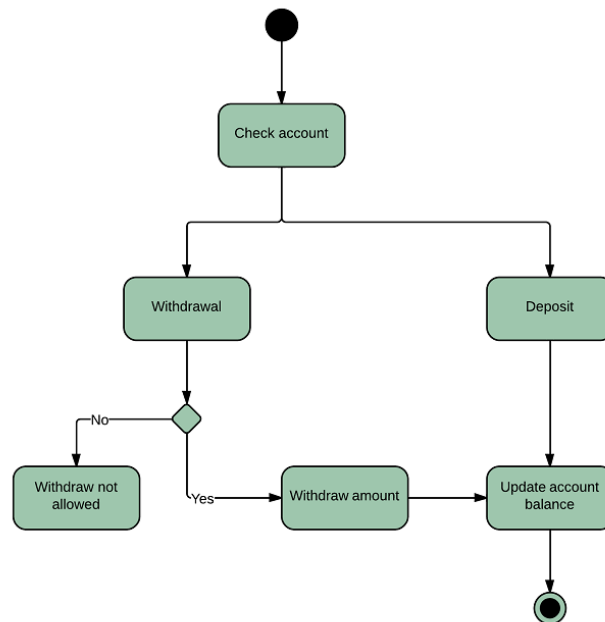


Figura 1 - Diagrama de atividades (fonte: Lucidchart)

Suas características são:

- Nós de atividades que representam as ações ou tarefas a serem realizadas.
- Transições que indicam o fluxo de controle entre as atividades e são representadas por setas.
- Decisões que são utilizadas para descrever pontos de bifurcação no fluxo, onde pode se tomar diferentes caminhos dependendo de condições em específico.
- Início e fim que são os símbolos que demarcar o início e o término do processo.

2.4.2 Diagrama de casos de uso

Essa ferramenta ajuda a descrever e organizar as funcionalidades de um sistema do ponto de vista do usuário. Focando nas interações entre os atores – que podem ser usuários

ou sistemas externos – e o sistema, destacando quais funcionalidades estão disponíveis e como diferentes atores utilizam essas funcionalidades.

Seus principais elementos são:

- Atores: que representam os usuários ou outros sistemas que interagem com o sistema em desenvolvimento. São representados por figuras de “bonecos palito”.
- Casos de uso: são as funcionalidades ou os serviços que o sistema oferece aos atores. São descritos como verbos ou ações, como “atualizar estoque”. São representados como elipses.
- Relações: é as conexões entre os atores e os casos de uso. Nos mostram quem executa qual ação.
- Sistema: é representado como um retângulo que envolve os casos de uso, delimitando quais funcionalidades pertencem ao sistema.

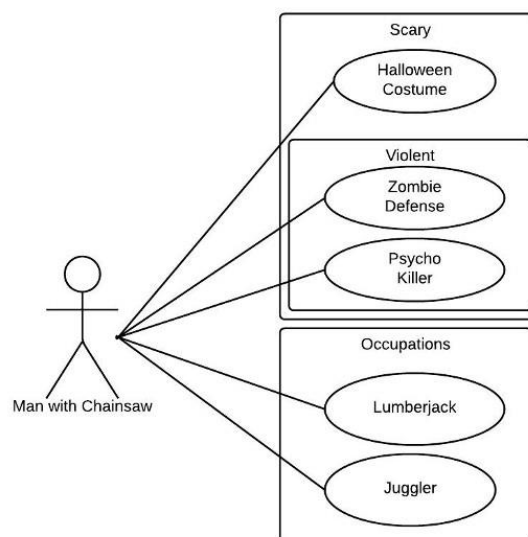


Figura 2 - Diagrama casos de uso (fonte: Lucidchart)

2.5 Prototipagem em eletrônica

É uma ferramenta de código aberto amplamente utilizada para prototipagem de circuitos eletrônicos de forma visual. Um exemplo desse tipo de ferramenta é o *software* Fritzing, que permite criar circuitos, diagramas de prototipagem e *layouts* de PCBs (Placas de Circuito Impresso). Trata-se de uma ferramenta ideal para quem trabalha com eletrônica, pois

sua interface é simples e facilita a criação de projetos sem exigir muito esforço. Além disso, é útil para documentar projetos de forma clara e apresentável.

No Fritzing é possível utilizar o modo de placa de ensaio, onde podemos montar um esquema visual de como seria o circuito, arrastando e soltando componentes e conectá-los com fios, tecnicamente conhecidos como *jumpers*.

A ferramenta converte automaticamente o esquema da placa de ensaio em um diagrama esquemático, que é uma representação mais técnica do circuito, permitindo entender de forma mais clara as conexões.

O Fritzing oferece uma grande diversidade de componentes eletrônicos, como sensores, displays, motores e diversos microcontroladores. Se necessário, também é possível criar os componentes personalizados ou importar novos componentes por meio de arquivos.

3. METODOLOGIA

O presente trabalho seguiu um método de pesquisa aplicada, contendo algumas etapas. A pesquisa envolveu a seleção e definição dos recursos necessários para o desenvolvimento do projeto, identificando o *software* e o ambiente de desenvolvimento (IDE) adequados, incluindo ferramentas de aprendizado de máquina, como o TensorFlow e Edge Impulse, além de componentes eletrônicos essenciais, como o ESP32-CAM. O estudo contemplou a análise das conexões desses componentes e a escolha de metodologias de organização, como a UML para a modelagem de processos e o Fritzing para a representação esquemáticas das conexões eletrônicas. Além disso, foram verificados os requisitos mínimos de hardware necessários para o funcionamento dos softwares em questão, garantindo que o microcontrolador tivesse a capacidade de processar o aprendizado de máquina com eficiência. O planejamento consistiu em organizar e priorizar os testes dos softwares necessários. Paralelamente, foi realizada a aquisição dos componentes eletrônicos, como o ESP32-CAM e os adaptadores necessários para o seu funcionamento. Para a melhor visualização e organização do projeto, foram criados diagramas UML e esquemas Fritzing, detalhando tanto a estrutura do software quanto as conexões dos componentes. Os testes incluíram a experimentação do *software* de aprendizado de máquina Edge Impulse e TensorFlow, juntamente com a integração dos componentes eletrônicos, como o ESP32-CAM. Além disso, foram realizados testes com códigos padrão de detecção de objetos para avaliar a precisão do componente e o *software* utilizado, garantindo a funcionalidade adequada para o sistema de gerenciamento de estoque automatizado. O processo de desenvolvimento envolveu a coleta de fotos, utilizadas para o treinamento do programa de aprendizado de máquina. As imagens foram devidamente rotuladas para permitir a identificação precisa dos itens durante o reconhecimento. Com o modelo treinado, foi feito o *deploy* do código gerado para a integração com o sistema. Teste específicos foram realizados para garantir o funcionamento correto da câmera ESP32-CAM e do código de detecção de objetos criado a partir das fotos rotuladas. Além disso, foi desenvolvido um sistema de código e banco de dados capaz de atualizar automaticamente o estoque conforme os produtos fossem reconhecidos e vendidos.

4. SISTEMA DE GESTÃO DE ESTOQUE COM ESP32-CAM

O sistema foi desenvolvido com o objetivo de aprimorar a gestão de estoque. Com isso em mente, foram utilizados os seguintes componentes para o desenvolvimento:

- C++: Utilizado para programar o microcontrolador com o código desenvolvido para a detecção de objetos.
- Edge Impulse: Foi utilizado para treinamento de modelos de aprendizado de máquina, onde foram fornecidas as imagens devidamente rotuladas para garantir a eficácia do aprendizado.
- SQLite: Utilizado para armazenar as informações do estoque.
- Python: Utilizado para programar o monitoramento de porta serial e atualizar o banco de dados.

4.1 Estrutura do sistema

Esta seção descreve as estruturas utilizadas para o planejamento e desenvolvimento do sistema.

4.1.1 Diagrama de caso de uso

O diagrama a seguir ilustra os casos de uso que envolvem o usuário, permitindo a captura de imagem em tempo real do objeto desejado. O sistema realiza a detecção do objeto e atualiza automaticamente o banco de dados do estoque, conforme representado no diagrama.

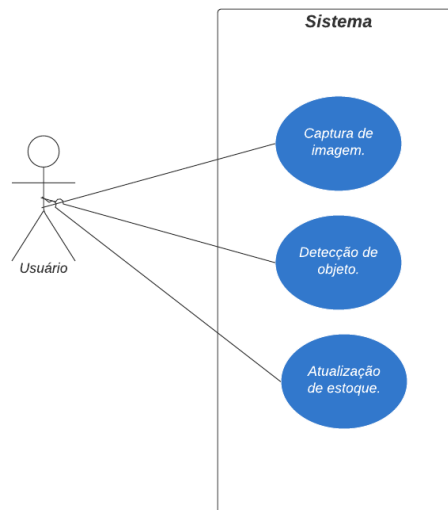


Figura 3 - Diagrama de casos de uso do sistema de gerenciamento de estoque

4.1.2 Diagrama de atividades

O diagrama de atividades descreve o fluxo de execução dentro do sistema, representando o comportamento dinâmico das atividades. Ele detalha as etapas desde a captura da imagem de um produto até a atualização do banco de dados do estoque, demonstrando o fluxo de ações automáticas que ocorrem em sequência e destacando a interação entre o reconhecimento de objetos e a atualização do sistema.

Na figura, são mostradas as seguintes funções: o sistema inicia a captura da imagem quando o usuário o ativa, e essa imagem é enviada para o próximo processo. Após ativado, a câmera captura imagens continuamente. Essas imagens passam pelo processo de reconhecimento de objetos, que identifica e classifica o produto. Em seguida, o sistema verifica o banco de dados e atualiza automaticamente a quantidade do item no estoque, refletindo a venda.

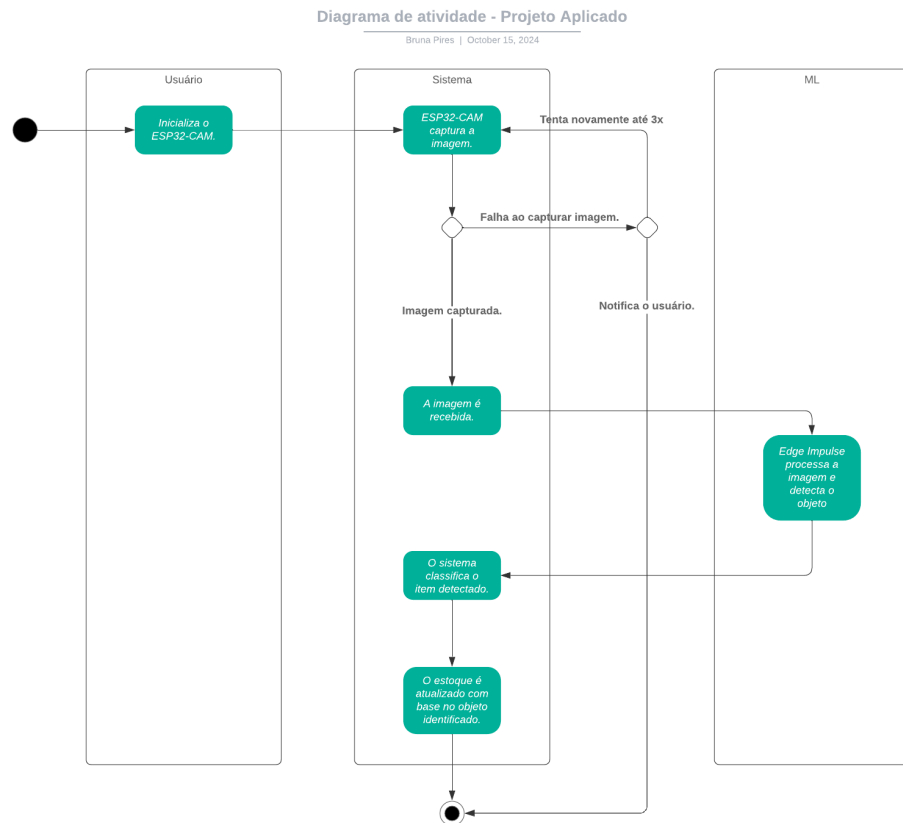


Figura 4 - Diagrama de atividades do sistema de gerenciamento de estoque

4.1.3 Diagrama de conexão dos componentes

O diagrama de componentes eletrônicos apresenta a conexão entre os componentes principais do sistema, incluindo uma placa Arduino UNO, o módulo ESP32-CAM e um conversor FTDI. Esses componentes trabalham em conjunto para realizar a captura de imagens e enviar os dados para o processamento e atualização do banco de dados do estoque.

A placa Arduino UNO é a responsável por fornecer energia ao módulo ESP32-CAM. Embora o conversor FTDI tenha os pinos para fornecer energia ao módulo, essa alimentação não é o suficiente, sendo necessário que o Arduino faça a função de fonte externa. O módulo ESP32-CAM contém uma câmera embutida que captura imagens dos produtos. Ele fica conectada ao Arduino para receber energia e ao conversor FTDI, que é responsável por enviar o código ao módulo. O conversor FTDI (USB para serial) é utilizado para carregar o código e depurar o módulo ESP32-CAM. Ele converte os sinais de comunicação entre a porta USB do computador e o ESP32-CAM, facilitando a transferência de dados durante a programação e o teste do sistema.

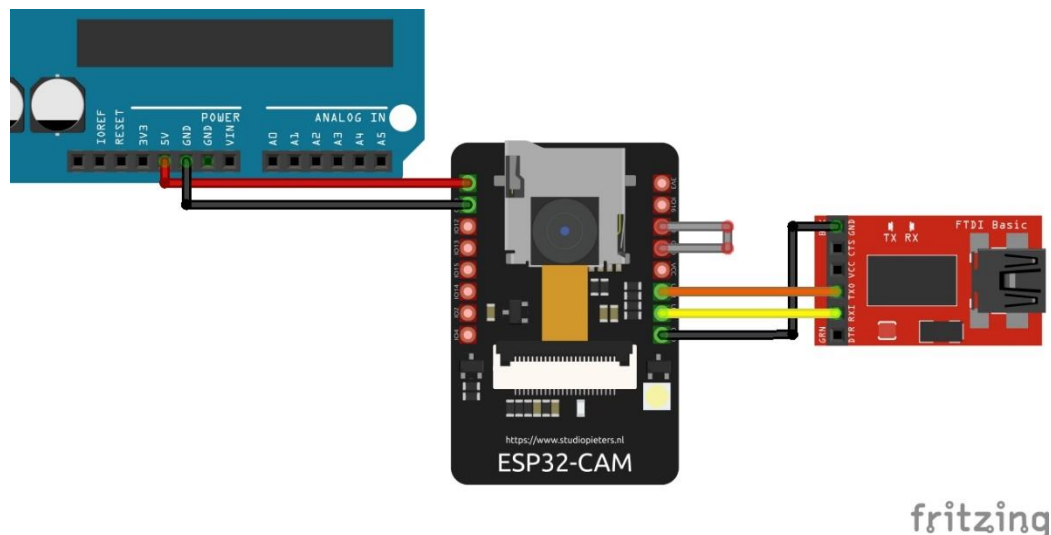


Figura 5 - Conectando componentes ao ESP32-CAM

Conexões:

- O Arduino UNO está conectado ao ESP32-CAM por meio de *jumpers* (cabos) nas conexões de 5V e GND de cada placa.
- O conversor FDTI está conectado ao ESP32-CAM por meio de *jumpers* (cabos) nas portas GND de cada placa. O UoR da placa ESP32-CAM deve ser conectado ao TX do FDTI, e o pino UoT da placa ESP32-CAM deve ser conectado ao RX do FDTI.
- Para que seja possível o envio do código, é necessário que o GND e ID0 da placa ESP32-CAM estejam conectados. Após o envio, é preciso remover esta conexão e reiniciar a placa utilizando o botão **RST**.

4.2 Desenvolvimento do projeto

A seguir, será apresentado o passo a passo do desenvolvimento do sistema.

4.2.1 Treinamento de máquina

Para iniciar o projeto de detecção de objetos, é necessário coletar diversas fotos dos objetos que o sistema deve detectar. A primeira decisão é optar pelo treinamento com imagens em RGB (colorido) ou em *grayscale* (tons de cinza). Esse projeto seguiu a abordagem em *grayscale*, pois as imagens geradas ocupam menos espaço de memória, o que é vantajoso para o microcontrolador. No entanto, se os resultados obtidos não forem satisfatórios, é possível realizar o treinamento utilizando RGB.

Após essa escolha, criamos uma conta no Edge Impulse para iniciar o treinamento. No caso de utilizar imagens RGB, o envio de fotos ou tirar fotos usando outros dispositivos. No entanto, para grayscale, recomenda-se a captura das imagens diretamente com o dispositivo que será utilizado no sistema final, melhorando a precisão do modelo.

Depois da coleta, é necessário categorizar as imagens, demarcando manualmente os objetos a serem detectados. Esse passo é simples e essencial para o sucesso do treinamento, por isso faça com bastante atenção.

Na plataforma Edge Impulse, na seção “*Impulse Design*”, devemos criar um novo “impulso” clicando em **Create Impulse**. Em seguida:

- Na seção “*Image data*”, ajustamos o tamanho da imagem para 48x48.
- No campo “*Add processing block*”, adicionamos a opção **Image**.
- No campo “*Add learning block*”, selecionamos **Object detection (Images)** e clicamos em **Save Impulse** para prosseguir.

Na próxima etapa, alteramos o parâmetro **Image color depth** para *grayscale* e salvamos as configurações em **Save Parameters**.

Agora, dentro da seção **Object detection**, configuramos o treinamento:

- **Number of training cycles**: para 30 ciclos.
- **Learning rate**: alteramos para 0.005.
- O modelo utilizado será o **FOMO (Faster Objects, More Objects) MobileNetV2.0.1**. Após isso, clicamos em **Save & Train** para iniciar o treinamento.

Finalizando o treinamento, vamos à etapa de geração do código. Na seção **Deploy**, adicionamos a biblioteca Arduino e clicamos em **Build**. Após alguns segundos, o sistema gerará um arquivo **.zip**, que será utilizado na Arduino IDE para programar o dispositivo. Importante: para realizar o envio do código, é necessário utilizar a versão 2.X.X da Arduino IDE.

4.2.2 Enviando código para o ESP32-CAM

Após baixar o arquivo .zip gerado pelo Edge Impulse, deve-se adicionar a biblioteca na Arduino IDE para possibilitar o uso do código fornecido e os *imports* necessários para o funcionamento do sistema.

Em seguida, é importante ajustar o código para ser compatível com o modelo da placa ESP32-CAM que está sendo utilizado. Para enviar o código à placa, é crucial garantir que os pinos GND e IOD estejam conectados corretamente, o que coloca a placa em modo de *upload*.

```

33 // https://github.com/espressif/arduino-esp32/blob/main
34
35 // #define CAMERA_MODEL_ESP_EYE // Has PSRAM
36 #define CAMERA_MODEL_AI_THINKER // Has PSRAM
37
38 #if defined(CAMERA_MODEL_ESP_EYE)
39 #define PWDN_GPIO_NUM -1

```

Figura 6 - Selecionando o modelo da placa ESP

Caso ocorra algum erro durante o processo de upload, será necessário fazer uma pequena modificação nos arquivos da biblioteca. Acesse o diretório de bibliotecas da IDE Arduino, geralmente fica na pasta documentos do computador, `C:\Users\Usuario\Documents\Arduino\libraries\projeto_inferencing\src\edge-impulse-sdk\tensorflow\lite\micro\kernels\` e edite os arquivos `conv.cpp` e `depthwise_conv.cpp` para que o código funcione.

```

1785 #if ESP_32
1786 if (input->type == kTfLiteInt8) {
1787     data_dims_t input_dims = {
1788         .width = input_width, .height = input_height,
1789         .channels = input->dims->data[3]
1790     };
1791     data_dims_t output_dims = {
1792         .width = output_width, .height = output_height,
1793         .channels = output->dims->data[3]
1794     };
1795     data_dims_t filter_dims = {filter_width, filter_height, 0, 0};
1796     conv_params_t conv_params = {
1797         .in_offset = 0, .out_offset = 0,
1798         .stride = {params.stride_width, params.stride_height},
1799         .padding = {data->op_data.padding.width, data->op_data.padding.height},
1800         .dilation = {0, 0}, .activation = {-128, 127}
1801     };
1802
1803     const int input_size = input_width * input_height * input_depth;
1804     const int output_size = output_width * output_height * output_depth;
1805
1806     data_dims_t input_dims = {
1807         .width = input_width, .height = input_height,
1808         .channels = input_depth
1809     };
1810     data_dims_t output_dims = {
1811         .width = output_width, .height = output_height,
1812         .channels = output_depth
1813     };
1814     data_dims_t filter_dims = {filter_width, filter_height, 0, 0};
1815     conv_params_t conv_params = {
1816         .in_offset = input_offset, .out_offset = output_offset,
1817         .stride = {stride_width, stride_height},
1818         .padding = {pad_width, pad_height},
1819         .dilation = {0, 0},
1820         .activation = {activation_min, activation_max}
1821     };
1822     quant_data_t quant_data = {
1823         .shift = data.op_data.per_channel_output_shift,
1824         .mult = data.op_data.per_channel_output_multiplier
1825     };

```

Figura 7 - Edição do arquivo conv.cpp

```

1723     esp_nn_set_depthwise_conv_scratch_buf(scratch_buf);
1724
1725     data_dims_t input_dims = {
1726         .width = input_width, .height = input_height,
1727         .channels = input_depth
1728     };
1729     data_dims_t output_dims = {
1730         .width = output_width, .height = output_height,
1731         .channels = output_depth
1732     };
1733     data_dims_t filter_dims = {filter_width, filter_height, 0, 0};
1734     dw_conv_params_t conv_params = {
1735         .in_offset = input_offset, .out_offset = output_offset,
1736         .ch_mult = depth_multiplier,
1737         .stride = {stride_width, stride_height},
1738         .padding = {pad_width, pad_height}, .dilation = {0, 0},
1739         .activation = {activation_min, activation_max}
1740     };
1741     quant_data_t quant_data = {
1742         .shift = data.op_data.per_channel_output_shift,
1743         .mult = data.op_data.per_channel_output_multiplier
1744     };

```

```

1832 #if ESP_NN
1833 if (input->type == kFlliteInt8) {
1834     data_dims_t input_dims = {
1835         .width = input_width, .height = input_height,
1836         .channels = input->dims->data[3]
1837     };
1838     data_dims_t output_dims = {
1839         .width = output_width, .height = output_height,
1840         .channels = output->dims->data[3]
1841     };
1842     data_dims_t filter_dims = {filter_width, filter_height, 0, 0};
1843     dw_conv_params_t conv_params = {
1844         .in_offset = 0, .out_offset = 0,
1845         .ch_mult = params.depth_multiplier,
1846         .stride = {params.stride_width, params.stride_height},
1847         .padding = {data->op_data.padding.width, data->op_data.padding.height},
1848         .dilation = {0, 0}, .activation = {-128, 127}
1849     };
1850 }

```

Figura 8 – Edição do arquivo `depthwise_conv.cpp`

4.2.3 Banco de dados

Utilizaremos o SQLite como banco de dados, já que o driver do SQLite é instalado por padrão juntamente com o Python, tornando-o uma escolha prática e eficiente.

Para iniciar o banco de dados, basta executar o seguinte comando em um terminal ou no VSCode utilizando Python: `sqlite3 nome_banco_de_dados.bd`.

Esse comando cria, ou abre, o arquivo de banco de dados. Após isso, podemos criar a tabela de produtos executando os comandos SQL adequados.

4.2.4 Teste de conexão com banco de dados e porta serial

Antes de implementar o código de atualização de estoque, é necessário verificar a conexão com o banco de dados utilizando Python. Para isso, faremos os seguintes passos:

- Importar o módulo `sqlite3` no código.
- Utilizar o comando `sqlite3.connect(path_do_banco)` para estabelecer a conexão com o banco de dados.
- Por último, devemos executar um comando para buscar as tabelas existentes no banco de dados.

Esse processo retornará as tabelas existentes no banco de dados ou apresentará um erro caso não consiga conectar.

```
testDatabase.py > ...
1  import sqlite3
2
3  def test_connection():
4      try:
5          print("Tentando conectar ao banco de dados SQLite...")
6          # Conectando-se ao banco de dados (um será criado se não existir)
7          db = sqlite3.connect('C:/Users/GamePlay/estoque.db')
8
9          # Verificando a conexão e executando um comando simples
10         cursor = db.cursor()
11         cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
12         tables = cursor.fetchall()
13
14         if tables:
15             print("Conexão com o banco de dados SQLite estabelecida com sucesso!")
16             print("Tabelas existentes:", tables)
17         else:
18             print("Nenhuma tabela encontrada no banco de dados.")
19
20         db.close()
21     except sqlite3.Error as err:
22         print(f"Erro ao conectar ao banco de dados: {err}")
23     except Exception as e:
24         print(f"Ocorreu um erro: {e}")
25
26 test_connection()
```

Figura 9 - Código de teste do banco de dados

Para testar a comunicação serial com a ESP32-CAM, primeiro é necessário instalar a biblioteca PySerial usando o comando `pip install pyserial` em um terminal.

Após a instalação, importamos a biblioteca PySerial no código e configuramos a porta à qual a placa ESP32-CAM está conectada, juntamente com a taxa de baud que a placa utiliza. Essa taxa é a velocidade de comunicação da placa. Isso exibirá os dados recebidos do ESP32-CAM diretamente no terminal do VSCode permitindo verificar se a comunicação está funcionando perfeitamente.

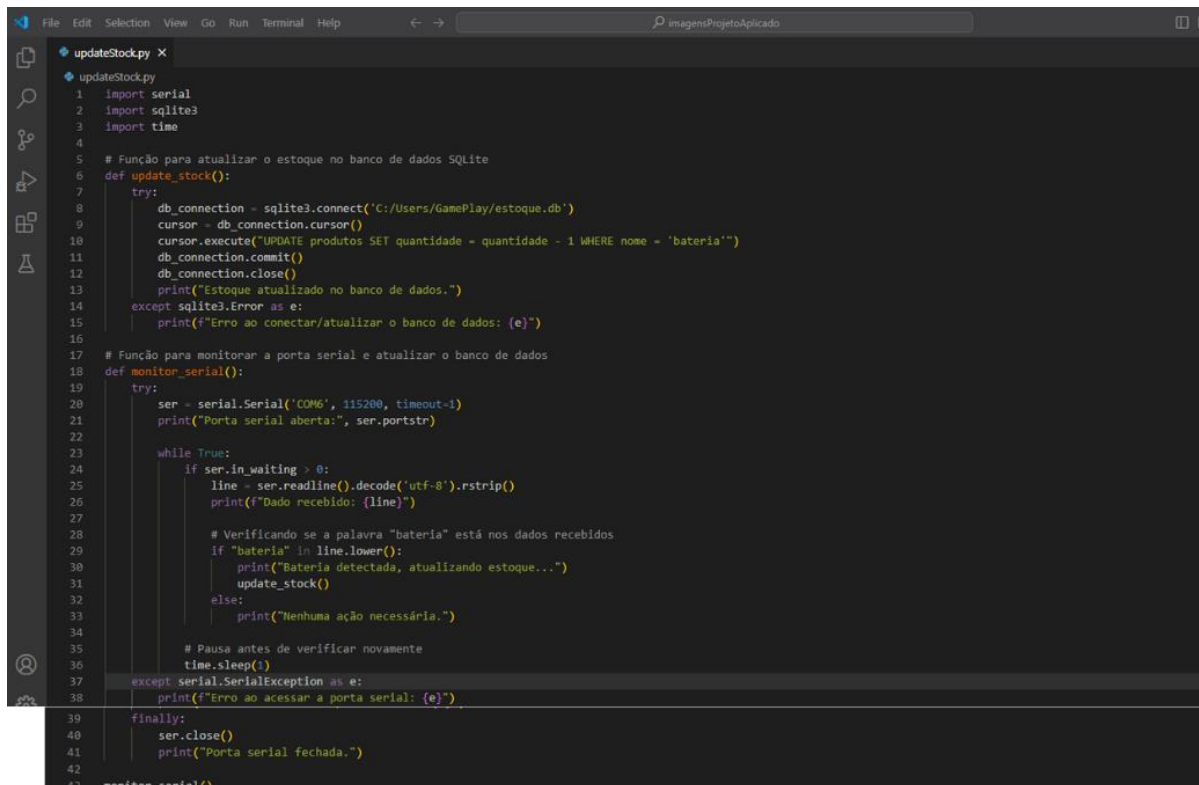
```
testSerialPort.py > ...
1  import serial
2
3  def test_serial():
4      try:
5          # Inicializando a porta serial
6          ser = serial.Serial('COM6', 115200, timeout=1)
7          print("Porta serial aberta:", ser.portstr)
8          line = ser.readline()
9          print("Dados recebidos da serial:", line.decode('utf-8').strip())
10         ser.close()
11     except serial.SerialException as e:
12         print(f"Erro ao acessar a porta serial: {e}")
13
14 test_serial()
```

Figura 10 - Código de teste da porta serial

4.2.5 Código principal

Após verificar que a conexão com o banco de dados e a comunicação serial estão funcionando corretamente, podemos implementar o código principal para monitorar a porta serial e atualizar o banco de dados automaticamente com base nos dados recebidos.

O sistema será responsável por ler continuamente os dados da **ESP32-CAM** através da porta serial. Quando o nome de um produto for detectado, uma função será chamada para atualizar o estoque correspondente no banco de dados.



```

1  import serial
2  import sqlite3
3  import time
4
5  # Função para atualizar o estoque no banco de dados SQLite
6  def update_stock():
7      try:
8          db_connection = sqlite3.connect('C:/Users/GamePlay/estoque.db')
9          cursor = db_connection.cursor()
10         cursor.execute("UPDATE produtos SET quantidade = quantidade - 1 WHERE nome = 'bateria'")
11         db_connection.commit()
12         db_connection.close()
13         print("Estoque atualizado no banco de dados.")
14     except sqlite3.Error as e:
15         print("Erro ao conectar/atualizar o banco de dados: (e)")
16
17 # Função para monitorar a porta serial e atualizar o banco de dados
18 def monitor_serial():
19     try:
20         ser = serial.Serial('COM6', 115200, timeout=1)
21         print("Porta serial aberta:", ser.portstr)
22
23         while True:
24             if ser.in_waiting > 0:
25                 line = ser.readline().decode('utf-8').rstrip()
26                 print(f"Dado recebido: {line}")
27
28                 # Verificando se a palavra "bateria" está nos dados recebidos
29                 if "bateria" in line.lower():
30                     print("Bateria detectada, atualizando estoque...")
31                     update_stock()
32                 else:
33                     print("Nenhuma ação necessária.")
34
35                 # Pausa antes de verificar novamente
36                 time.sleep(1)
37     except serial.SerialException as e:
38         print(f"Erro ao acessar a porta serial: (e)")
39     finally:
40         ser.close()
41         print("Porta serial fechada.")
42
43 monitor_serial()

```

Figura 11 - Código final do sistema

Neste código:

- A função **update_stock()** é responsável por conectar-se ao banco de dados SQLite e atualizar a quantidade do produto especificado.
- A função **monitor_serial()** lê continuamente os dados da porta serial. Quando o nome de um produto é identificado (por exemplo, "bateria"), a função de atualização de estoque é chamada.

Um aspecto inovador que se destaca é a integração eficaz entre o ESP32-CAM e o SQLite para o gerenciamento automatizado de estoque, utilizando técnicas de detecção de objetos. A solução se diferencia pela sua capacidade de atualizar o estoque em tempo real, ao

detectar produtos por meio da câmera embarcada e, em seguida, comunicar-se com o banco de dados local de forma simples e eficiente.

A implementação de um sistema de detecção utilizando imagens em *grayscale*, que reduz significativamente o uso de recursos, tornando o processamento mais ágil e adequado para dispositivos de baixo custo. Esse ajuste otimiza a performance do microcontrolador, sem comprometer a precisão na identificação dos produtos.

5. CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema automatizado de gerenciamento de estoque utilizando ESP32-CAM mostrou-se promissor no contexto de empreendimento local. O projeto atingiu os objetivos principais, como o reconhecimento de produtos em tempo real e a atualização automática de estoque. A solução foi capaz de abordar o problema de controle manual de inventário, fornecendo um sistema que minimiza erros e otimiza a gestão dos produtos.

Ao longo do projeto, verificou-se que a integração entre a ESP32-CAM e o Edge Impulse facilitou o treinamento do modelo e a implementação da detecção diretamente no dispositivo. Além disso, o uso do SQLite como banco de dados local no computador proporcionou a simplicidade necessária para o armazenamento e recuperação de dados de forma ágil e eficiente.

Conceitos de visão computacional e automação foram embasados em pesquisas existentes, garantindo que o sistema fosse desenvolvido sobre uma base sólida. O uso de tecnologias como o Arduino IDE para programação e o Python para comunicação com o banco de dados demonstrou ser uma combinação adequada para o cenário proposto.

Com base nos resultados alcançados, acredita-se que este sistema pode ser aprimorado em estudos futuros, como a implementação de uma interface gráfica para facilitar o monitoramento do estoque e o controle de vendas em tempo real. Além disso, o sistema pode ser expandido para suportar múltiplas categorias de produtos e integrar-se com plataformas de gerenciamento financeiro e de vendas online.

O projeto não apenas resolve o problema inicial de atualização manual de estoque, mas também abre caminho para novos desenvolvimentos, mostrando que a tecnologia de IoT, em conjunto com inteligência artificial, pode transformar significativamente a maneira como pequenas e médias empresas gerenciam seus recursos.

REFERÊNCIAS

AMAZON WEB SERVICES. O que é visão computacional? [S.l.], 2023. Disponível em: <https://aws.amazon.com/pt/what-is/computer-vision/>. Acesso em: 19 set. 2024.

CIGAM. Gestão de estoque: o que é e qual a importância para a sua empresa? [S.l.], 2023. Disponível em: <https://www.cigam.com.br/blog/723/gestao-de-estoque-o-que-e-e-qual-a-importancia-para-a-sua-empresa#:~:text=Uma%20gest%C3%A3o%20eficiente%20de%20estoque%20possibilita%20que%20as%20empresas%20atendam,maior%20lucratividade%20para%20o%20neg%C3%B3cio>. Acesso em: 1 out. 2024.

EDGE IMPULSE. Documentation. Disponível em: <https://docs.edgeimpulse.com/docs>. Acesso em: 27 set. 2024.

ESPRESSIF. ESP32 Overview. 2023. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 26 set. 2024.

GOOGLE AI. Getting Started with TensorFlow Lite for Microcontrollers. 2023. Disponível em: https://ai.google.dev/edge/litert/microcontrollers/get_started?hl=pt-br. Acesso em: 27 set. 2024.

LUCIDCHART. Diagrama de Atividades UML. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-atividades-uml>. Acesso em: 15 out. 2024.

LUCIDCHART. Diagrama de Caso de Uso UML. Disponível em: <https://www.lucidchart.com/pages/pt/diagrama-de-caso-de-uso-uml>. Acesso em: 15 out. 2024.

LUCIDCHART. Diagrama de Classe UML. Disponível em: <https://www.lucidchart.com/pages/pt/o-que-e-diagrama-de-classe-uml>. Acesso em: 15 out. 2024.

MAKERHERO. Módulo ESP32-CAM com câmera OV2640 2MP. 2023. Disponível em: <https://www.makerhero.com/produto/modulo-esp32-cam-com-camera-ov2640-2mp/>. Acesso em: 16 set. 2024.

MICROSOFT. Bem-vindo de volta ao C++ - C++ moderno. 2023. Disponível em: <https://learn.microsoft.com/pt-br/cpp/cpp/welcome-back-to-cpp-modern-cpp?view=msvc-170>. Acesso em: 2 out. 2024.

TENSORFLOW. TensorFlow Lite for Microcontrollers. 2023. Disponível em: <https://www.tensorflow.org/lite/microcontrollers?hl=pt-br>. Acesso em: 19 set. 2024.