

Sistema de Registro de Ponto com Reconhecimento Facial para o Setor da Construção Civil

Tífany Severo
Universidade de Brasília
Brasília, Brasil

Abstract— This project presents a facial recognition-based time attendance system designed to operate autonomously and in a distributed manner across multiple job sites. The solution employs a Raspberry Pi 4 with a PiCam to perform real-time facial recognition locally, without relying on a central server. Employee registration is handled through an C++ HTTP server, accessed via a computer browser, allowing image capture and transmission to the Raspberry Pi. After local preprocessing, the facial model is trained using OpenCV and synchronized with a GitLab repository via C++ scripts. The recognition client performs a git pull to fetch the latest models, displays the identified employee's name on an OLED screen, and awaits confirmation through a physical button. Attendance logs are saved in .csv files and automatically pushed to GitLab, ensuring backup and traceability. The system provides a low-cost, portable, and scalable solution, ideal for construction companies requiring mobility, robustness, and unit integration.

Keywords— *Distributed Access Control, Raspberry Pi, Cloud Server, Linux Operating System*

Resumo — Este projeto apresenta um sistema de registro de ponto eletrônico baseado em reconhecimento facial, desenvolvido para operar de forma autônoma e distribuída em ambientes com múltiplas unidades, como canteiros de obras. A solução utiliza a Raspberry Pi 4 com câmera PiCam para executar o reconhecimento facial em tempo real, sem necessidade de servidor central. O cadastro de novos funcionários ocorre por meio de um servidor HTTP em C++, acessado via navegador por computador, onde as imagens são capturadas e enviadas à Raspberry Pi. Após o pré-processamento local, o modelo facial é treinado com OpenCV e os arquivos gerados são sincronizados com um repositório GitLab por meio de scripts em C++. O cliente de reconhecimento realiza git pull para obter os modelos mais recentes, exibe o nome reconhecido em um display OLED e solicita a confirmação de ponto por botão físico. Os registros são salvos em arquivos .csv e enviados automaticamente ao GitLab, garantindo backup e rastreabilidade. A solução oferece baixo custo, alta portabilidade e escalabilidade, sendo ideal para empresas do setor da construção civil que demandam mobilidade, robustez e integração entre unidades.

Palavras-chave — *Controle de Acesso Distribuído, Raspberry Pi, Servidor em Nuvem, Sistema Operacional Linux*

I. INTRODUÇÃO

A crescente busca por soluções eficientes de gestão de jornada e segurança no ambiente de trabalho tem impulsionado o desenvolvimento de tecnologias aplicadas ao controle de ponto eletrônico. No setor da construção civil, onde há constante

mobilidade de equipes entre canteiros de obras e diferentes unidades da mesma empresa, o desafio de monitorar e registrar os horários dos trabalhadores de forma precisa, segura e integrada torna-se ainda mais evidente. Sistemas convencionais de registro de ponto, como cartões magnéticos, folhas de papel ou até mesmo dispositivos fixos com senha ou biometria digital, frequentemente falham em oferecer a flexibilidade, escalabilidade e robustez necessárias para esse cenário específico.

Frente a essa problemática, este artigo propõe o desenvolvimento de um sistema de controle de ponto com reconhecimento facial, baseado em hardware embarcado de baixo custo (Raspberry Pi 4 com PiCam), capaz de identificar funcionários em diferentes unidades da construtora, mesmo quando se deslocam entre bairros distintos ao longo do dia. O sistema opera de forma distribuída, com comunicação entre o dispositivo embarcado e um servidor em nuvem, permitindo sincronização de dados em tempo real e centralização das informações.

O reconhecimento facial é executado localmente com base em modelos LBPH treinados na própria Raspberry Pi. A atualização dos modelos e o compartilhamento dos registros de ponto entre unidades ocorrem de forma automatizada por scripts em C++, permitindo que novas unidades se atualizem com dados da nuvem sem qualquer intervenção técnica complexa.

Com esse modelo, um funcionário pode ser cadastrado em uma unidade (por exemplo, em Taguatinga-DF) e, em poucos minutos, ser reconhecido em outra (como em Planaltina-DF), com total sincronização dos registros de ponto e sem duplicidade de cadastro.

A proposta visa solucionar uma lacuna existente entre os sistemas caros e proprietários oferecidos por grandes empresas do setor, como Control iD, Dimep e Ahgora, e as necessidades práticas de construtoras de médio e pequeno porte que requerem mobilidade, baixo custo e integração em múltiplas unidades. Enquanto essas soluções de mercado frequentemente dependem de dispositivos fixos e infraestrutura dedicada, o sistema aqui proposto diferencia-se por sua portabilidade, independência de rede proprietária, e autenticação facial baseada em aprendizado de máquina, executada localmente, com sincronização de eventos e imagens na nuvem.

A justificativa para o uso de reconhecimento facial, em vez de outros métodos biométricos, deve-se à maior agilidade no processo de autenticação e ao fato de não exigir contato físico com o equipamento — uma vantagem significativa em

ambientes de obra, onde as condições higiênicas e de acessibilidade podem variar.

Estudos recentes indicam que o uso de reconhecimento facial em sistemas de controle de acesso tem se mostrado eficaz em diversos contextos, apresentando precisão superior a 95% em ambientes controlados e viabilidade crescente em ambientes abertos com o uso de algoritmos como LBPH (Local Binary Patterns Histograms), Haar Cascades e redes neurais convolucionais (ZHAO et al., 2020; AHONEN et al., 2006). Além disso, plataformas embarcadas como o Raspberry Pi têm sido amplamente adotadas em projetos de IoT e automação industrial devido ao seu custo acessível, versatilidade e suporte a bibliotecas como OpenCV (BRADBURY, 2021). Isso reforça a escolha técnica adotada neste trabalho, unindo processamento local com conectividade em nuvem.

II. OBJETIVO

O objetivo deste projeto é desenvolver um sistema de controle de ponto eletrônico por reconhecimento facial, utilizando uma plataforma embarcada de baixo custo (Raspberry Pi 4 com PiCam), capaz de operar em diferentes unidades de uma mesma empresa de forma sincronizada e centralizada. O sistema deve ser capaz de:

- Permitir o **cadastro local de novos funcionários** e o treinamento de modelos de reconhecimento facial diretamente na Raspberry Pi;
- Executar o reconhecimento facial em tempo real embarcado, sem necessidade de servidor externo;
- Armazenar os registros de ponto em um arquivo .csv e sincronizá-los com uma planilha central online, compartilhada entre todas as unidades;
- Automatizar a distribuição e atualização dos modelos faciais entre as Raspberry Pis da empresa, garantindo que todas as unidades reconheçam todos os funcionários;
- Operar de forma **autônoma**, sem necessidade de supervisão técnica local;
- Oferecer uma solução de baixo custo, portátil e escalável, que possa ser facilmente implementada em ambientes industriais ou de obra, com foco em robustez, simplicidade e eficiência.

O projeto visa atender à demanda por mobilidade, praticidade e integração entre filiais da construção civil, promovendo automação no processo de registro de jornada de trabalho e possibilitando melhor rastreabilidade das equipes em campo.

III. DESENVOLVIMENTO

A. Descrição de Hardware

A arquitetura de hardware do sistema proposto foi desenvolvida com foco em **baixo custo, portabilidade, facilidade de implementação e compatibilidade** com bibliotecas de visão computacional. Para isso, utilizou-se a plataforma Raspberry Pi 4, aliada a uma câmera oficial PiCam v2.1 conectada à interface CSI (Camera Serial Interface), permitindo captura de vídeo em tempo real com alta eficiência.

O hardware responsável pelo reconhecimento facial localmente é composto por:

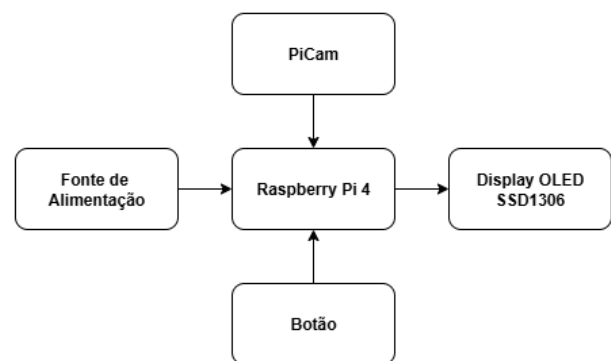
- **Raspberry Pi 4 Model B – 4GB RAM:** unidade de processamento principal, equipada com processador quad-core Cortex-A72 de 1.5GHz, que garante desempenho suficiente para tarefas de detecção de imagem, compressão JPEG e comunicação via rede.
- **Pi Camera v2.1:** sensor de imagem de 8MP (Sony IMX219) com suporte a captura de vídeo em HD (até 1080p30), integrado via CSI para minimizar o uso de USBs e reduzir latência.
- **Cartão microSD de 64GB Classe 10:** armazenamento local do sistema operacional, imagens capturadas e arquivos de registro temporários.
- **Fonte de alimentação 5V 3A com cabo USB-C:** necessária para garantir estabilidade elétrica do sistema embarcado.
- **Conectividade via Wi-Fi ou Ethernet:** utilizada exclusivamente para sincronização de modelos e registros de ponto com os serviços da nuvem (GitLab). O funcionamento principal do sistema não depende de conexão contínua, mas a rede é necessária para a atualização de modelos e envio de dados.
- **Display OLED:** para visualização de informações
- **Botão Push:** para registrar o ponto.

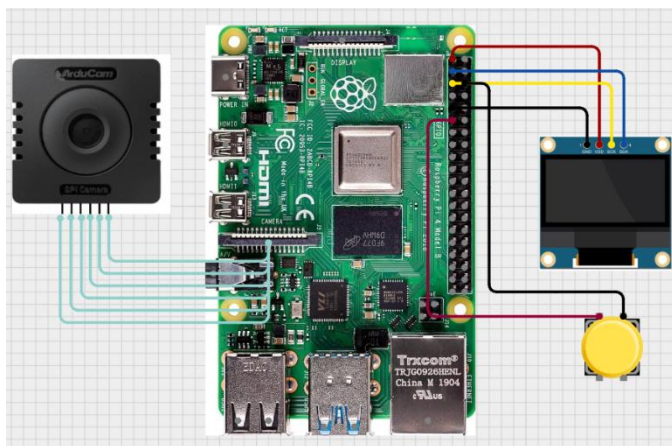
A escolha do Raspberry Pi 4 justifica-se pela ampla comunidade de suporte, compatibilidade com bibliotecas como OpenCV, e conectividade integrada (Wi-Fi e Ethernet), permitindo comunicação com o servidor central via sockets TCP/IP. Sua capacidade de rodar sistemas Linux completos (como Raspberry Pi OS) facilita a instalação de dependências, o gerenciamento remoto por SSH e a execução de scripts em Python ou binários em C++.

Além disso, o uso de uma câmera dedicada (PiCam) permite que o dispositivo capture imagens com qualidade suficiente para o treinamento e reconhecimento facial, mesmo em ambientes com variações de iluminação moderadas.

1) Diagrama de Blocos

A montagem do sistema consiste na conexão da PiCam ao conector CSI da Raspberry Pi, alimentação elétrica estável, acesso à rede via Wi-Fi ou cabo Ethernet, display OLED e um botão. Todo o restante do sistema está embarcado via software. A seguir, o esquemático do sistema e o Diagrama de Blocos:





Esquemático

2) Lista BOM

Item	Componente	Quantidade	Descrição Técnica	Observações
01	Raspberry Pi 4 Model B (4GB RAM)	1	Quad-core Cortex-A72, Wi-Fi, Bluetooth, USB, HDMI, CSI	Unidade de processamento central
02	Pi Camera v2.1	1	Sensor Sony IMX219, 8MP, conexão via CSI	Câmera para captura de rostos
03	Cartão microSD Classe 10 – 32GB	1	Armazenamento do sistema e dados locais	Sistema operacional e logs
04	Fonte de alimentação 5V 3A USB-C	1	Fonte com alimentação estável para Raspberry Pi	Evita reinicializações acidentais
05	Cabo de rede ou conexão Wi-Fi	1	Conexão com servidor central	Para transmissão dos dados
06	Display Oled ssd1306	1	Display para informações	Interação com o usuário
07	Botão	1	Botão push	Interação com o usuário

B. Descrição de Software

O sistema foi desenvolvido com uma arquitetura distribuída composta por três principais componentes: um servidor de cadastro remoto, um servidor local e um programa embarcado de reconhecimento facial. O código-fonte é escrito majoritariamente em C++, aproveitando a performance nativa da linguagem em tempo real, com o suporte de bibliotecas como OpenCV para visão computacional e Mongoose para comunicação HTTP. O treinamento de modelos ainda é realizado com um script Python (train.py), mas toda a orquestração do fluxo é conduzida por aplicações C++.

Servidor de Cadastro (HTTP via Mongoose - PC)

Este módulo é responsável por capturar imagens do rosto de novos funcionários e organizá-las em pastas nomeadas com o respectivo identificador do colaborador.

Funcionamento geral:

- **Cadastro:** o cadastro de novos funcionários é realizado remotamente. O operador acessa uma interface web hospedada em um servidor HTTP escrito em C++ com a biblioteca Mongoose. Nessa página, o usuário pode capturar imagens pelo navegador (usando a câmera do celular ou notebook) e preencher o nome do colaborador.
- **Envio das imagens:** As imagens são enviadas por meio de um formulário multipart contendo múltiplos arquivos e um campo de texto com o nome. O servidor HTTP, ao receber esse conteúdo, compacta e envia os arquivos para a Raspberry Pi via curl.

Processamento de Cadastro (Raspberry Pi – cadastro_server.cpp)

Este módulo opera continuamente na Raspberry Pi, recebendo as imagens, processando-as, treinando os novos modelos e enviando-os para o repositório.

Funcionamento geral:

- **Processamento de imagens:** cada face é detectada com Haar Cascade e imagem recebida é recortada na detecção da face, convertida para escala de cinza (200×200 pixels) e armazenada na pasta data/<nome>/.
- **Treinamento do modelo:** após o processamento, o cadastro_server.cpp executa o código check_and_pull.cpp para garantir que os modelos locais são os mais atuais e em seguida chama o script train.py. Esse script percorre todas as subpastas de data/, constrói os vetores faces e labels, treina um modelo LBPH (Local Binary Patterns Histograms) com as imagens e salva os arquivos lbph_model.yml e labels.txt na pasta models/.
- **Upload para o repositório:** ao final do treinamento, o upload_modelos.cpp é chamado. Ele envia os arquivos gerados (lbph_model.yml, labels.txt e timestamp.txt) para um repositório no gitlab, garantindo que outras unidades possam baixar o modelo atualizado.

Módulo de Reconhecimento e Registro de ponto (attend_client.cpp): Este módulo opera continuamente em cada Raspberry Pi, realizando o reconhecimento facial em tempo real e registrando o ponto do funcionário reconhecido.

Funcionamento geral:

- O programa ativa a câmera da Raspberry Pi e utiliza novamente um classificador Haar Cascade para detectar rostos.
- A cada rosto detectado, o sistema utiliza o modelo LBPH (carregado a partir de lbph_model.yml) para identificar a pessoa. Caso a predição atinja um nível de confiança aceitável, o sistema exibe no display OLED: "Funcionário: [Nome]. Deseja bater ponto? (S/N)".
- Se o usuário confirmar pressionando o botão, o nome e o timestamp são adicionados ao arquivo registro.csv. Cada funcionário possui uma linha única, e cada nova batida de ponto gera uma nova coluna com a data e hora.
- Após o registro de ponto, o .csv gerado é enviado para o repositório no gitlab.
- Para evitar registros em duplicidade, é imposta uma restrição de 30 minutos entre registros para o mesmo funcionário.

Scripts auxiliares

Além dos módulos principais, o sistema conta com quatro scripts que realizam operações específicas:

- **Check_and_pull_modelos.cpp:** sincroniza os modelos locais com a versão mais recente no GitLab, garantindo que todos os dispositivos estejam atualizados. Esse script é executado todos os dias às 7h através do agendamento pelo Linux com a função "contrab".
- **train.py:** treina o modelo LBPH com base nas imagens armazenadas em data/, salvando os arquivos lbph_model.yml e labels.txt.
- **upload_modelos.cpp:** envia os arquivos treinados para o repositório, junto com um timestamp.txt que marca a data da última atualização.
- **sync_to_sheets.cpp:** envia o .csv para o GitLab e sincroniza os dados de ponto do registro.csv, linha por linha.

1) Fluxograma e Diagrama de Camadas:

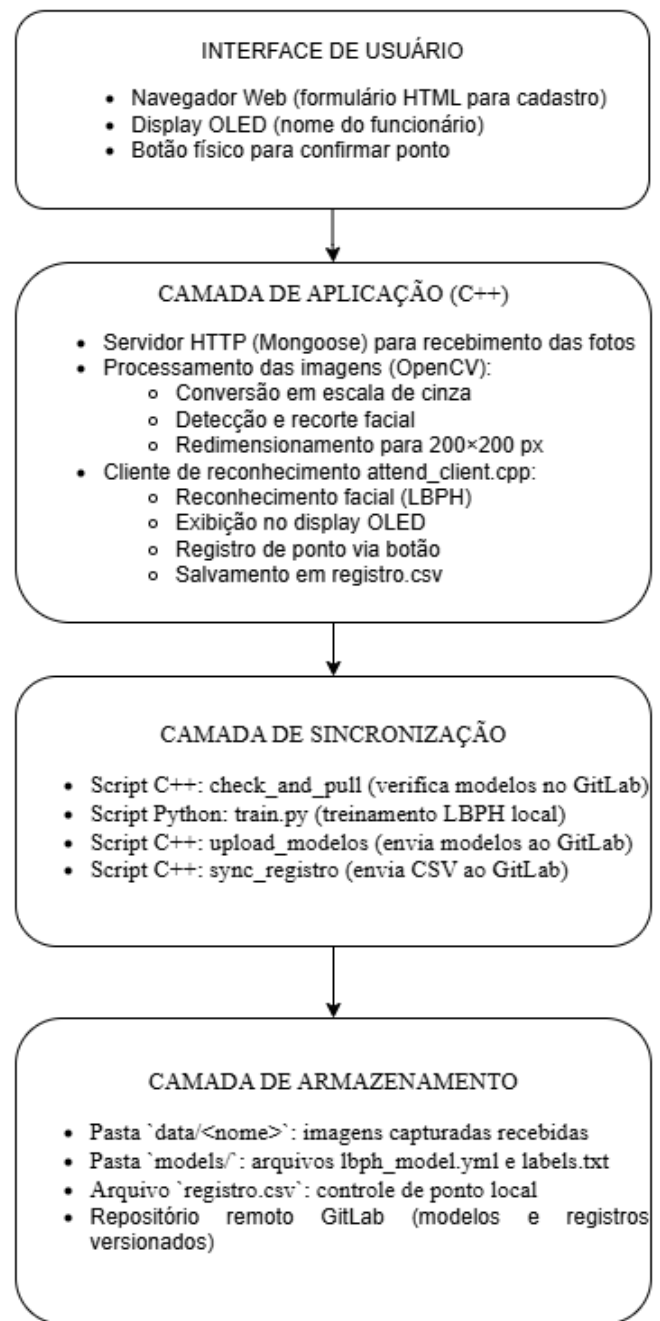
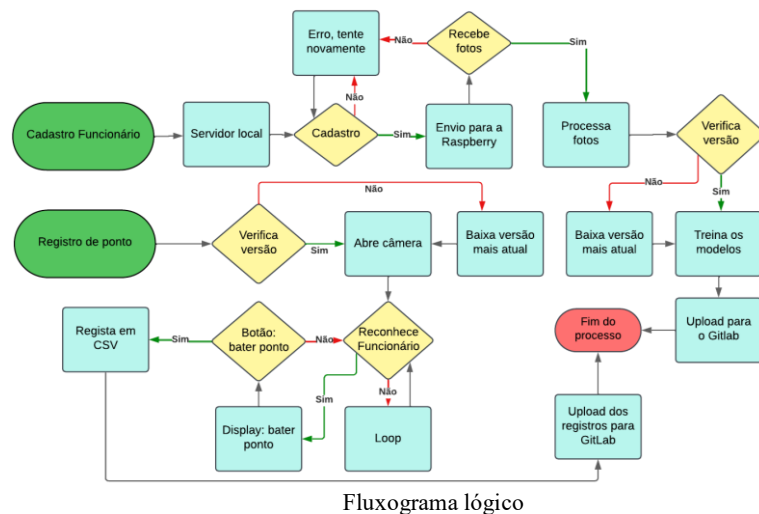


Diagrama de Camadas

Esse diagrama mostra como os módulos escritos em C++ e script em Python interagem entre si, por meio de imagens, modelos treinados e troca de dados via rede. A divisão clara entre cliente (na Raspberry Pi) e servidor (na máquina central ou na nuvem) facilita a escalabilidade.



Fluxograma lógico

IV. REQUISITOS

A. Funcionais:

Os requisitos funcionais definem as funcionalidades que o sistema deve obrigatoriamente oferecer para cumprir seu propósito principal. Entre eles, destacam-se:

- O sistema deve checar periodicamente se há atualizações dos modelos treinados na nuvem e baixar os novos modelos.
- O sistema deve permitir o **cadastro de novos funcionários** por meio de captura facial utilizando a câmera do dispositivo, com armazenamento local das imagens.
- Deve ser possível armazenar **múltiplas imagens por funcionário**, organizadas em pastas nomeadas conforme o identificador (nome).
- Após a captura das imagens, o sistema deve enviar as imagens para a RPi e executar automaticamente o treinamento do modelo facial (LBPH) com as novas imagens.
- O modelo treinado (lbph_model.yml, labels.txt) e um timestamp.txt devem ser enviados para o GitLab, permitindo sincronização com outras unidades.
- O programa attend_client.cpp deve detectar rostos em tempo real, utilizar o modelo LBPH embarcado para reconhecimento e, em caso de sucesso, perguntar se o funcionário deseja bater ponto.
- O ponto deve ser registrado localmente em um arquivo CSV com uma linha por funcionário, onde cada nova batida adiciona uma coluna com data e hora.
- O sistema deve aplicar uma restrição mínima de 30 minutos entre registros de ponto para o mesmo funcionário.
- Deve haver a possibilidade de sincronização manual do arquivo registro.csv com o repositório.
- A sincronização dos modelos e registros deve funcionar em qualquer Raspberry Pi conectada à internet, permitindo operação distribuída e cooperativa entre unidades remotas
- O sistema deve permitir que o operador encerre o programa manualmente ao pressionar uma tecla (por exemplo, “q”), garantindo controle local.

B. Não Funcionais:

Os requisitos não funcionais dizem respeito à qualidade, desempenho, escalabilidade e segurança do sistema, e foram definidos da seguinte forma:

- O sistema deve funcionar sem necessidade de servidor externo, sendo totalmente embarcado na Raspberry Pi.
- O tempo de resposta entre a **captura da imagem** e a **exibição do nome reconhecido** deve ser inferior a dois segundos, garantindo fluidez e usabilidade.
- Os dados devem ser armazenados em formato .csv legível e portátil, e sincronizados com a nuvem para redundância e segurança.

- O sistema deve ser capaz de operar temporariamente offline, sincronizando com a nuvem assim que a conexão for restabelecida.
- A arquitetura deve permitir que novas Raspberry Pis sejam instaladas sem necessidade de recadastrar funcionários, bastando baixar os modelos atualizados.
- O sistema deve ser compatível com dispositivos com até 4GB de RAM, garantindo viabilidade com Raspberry Pi 4.
- A interface deve ser minimalista e clara, exibindo vídeo da câmera, retângulo de detecção e nome do funcionário
- O código deve ser modular, facilitando manutenção, atualização e extensão futura.
- A segurança dos dados e modelos deve ser garantida por acesso restrito à conta de serviço e uso de diretórios protegidos
- A interface gráfica do sistema deve ser **minimalista e intuitiva**, exibindo apenas o vídeo da câmera com a detecção facial e o nome do colaborador.
- O sistema deve ser de fácil manutenção, com geração de logs legíveis e possibilidade de **reutilização do modelo treinado** por múltiplos dispositivos clientes.

V. RESULTADOS

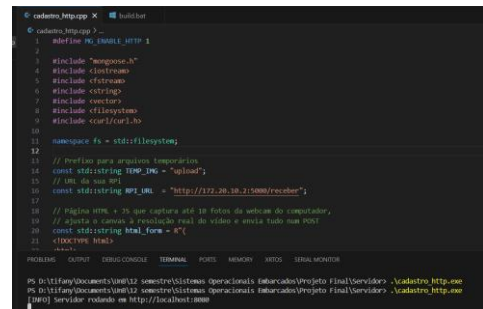
A validação do sistema foi realizada por meio de testes em ambiente real, utilizando uma Raspberry Pi 4 conectada a um display OLED, um botão físico e uma câmera PiCam, com comunicação em rede com um computador local (servidor de cadastro). Todas as etapas propostas no projeto foram implementadas com sucesso, confirmando o funcionamento autônomo, sincronizado e distribuído da solução — desde o cadastro remoto até o reconhecimento facial e registro de ponto com versionamento de dados.

Cadastro de Funcionário via Servidor HTTP

O processo de cadastro foi testado acessando-se a interface web fornecida pelo servidor HTTP em C++ (usando a biblioteca Mongoose). Ao preencher o nome do funcionário e capturar as imagens diretamente por navegador, os dados foram enviados com sucesso à Raspberry Pi, onde foram automaticamente:

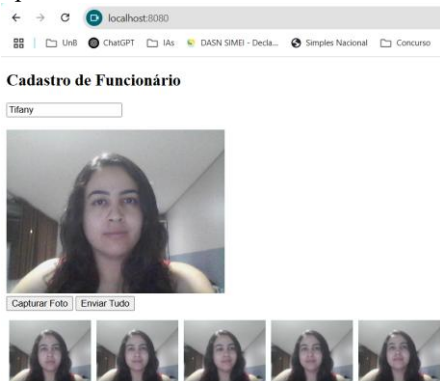
- Convertidos para escala de cinza;
- Detectados com Haar Cascade frontal;
- Recortados e redimensionados para 200×200 pixels;
- Armazenados localmente na pasta data/<nome>.

Servidor local rodando:

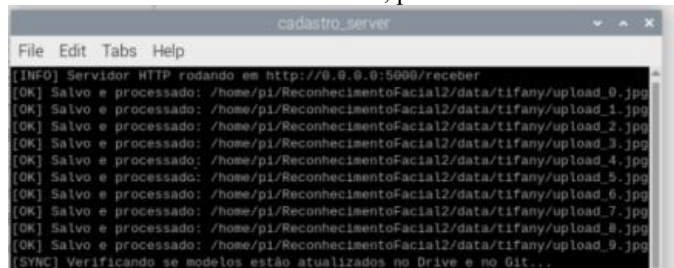


```
PS C:\Users\luciano> cd .\semestre\ sistemas operacionais\ embarcado\ projeto final\ servidor > .\cadastro_http.exe
[INFO] Servidor rodando em http://0.0.0.0:8080
```


Página web para cadastro de funcionário:



Servidor na RPi recebendo as fotos, processando e salvando:

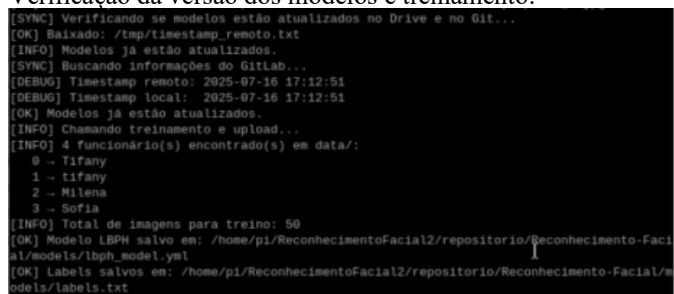


Treinamento do Modelo e Sincronização via GitLab

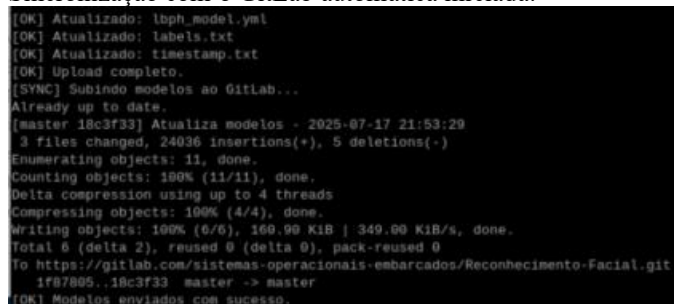
Após o recebimento das imagens, o script em C++ check_and_pull foi executado, verificando se os modelos locais estavam atualizados com o repositório GitLab remoto. Em seguida, o script train.py treinou um novo modelo LBPH com base nas imagens recém-adicionadas. O sistema então:

- Gerou os arquivos lbph_model.yml e labels.txt;
- Realizou o commit automático dos modelos;
- Executou push para o repositório GitLab via chamadas em C++.

Verificação da versão dos modelos e treinamento:



Sincronização com o GitLab automática iniciada:

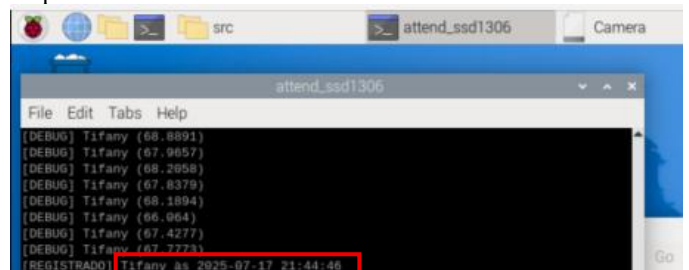


Reconhecimento Facial Local e Git Pull

No cliente embarcado (attend_client.cpp), antes do início do reconhecimento, o sistema executou um git pull para garantir que o modelo mais recente estivesse disponível localmente. Com a câmera ativada, o sistema detectou rostos em tempo real, identificando os funcionários com alta confiabilidade.

Ao detectar um rosto conhecido, o sistema exibiu no display OLED a mensagem com o nome identificado e solicitou a confirmação do ponto via botão físico. O acionamento do botão resultou na gravação imediata do ponto no arquivo registro.csv e em seguida foi feito o commit do registro no gitlab.

Detecção Facial com segurança de 3 confirmações e registro de ponto:



Display mostrando rosto reconhecido:



Após apertar o botão:



Envio automático do arquivo .CSV para o GitLab:

```
[SYNC] Enviando registro.csv ao GitLab...
Already up to date.
[master 1f87895] Atualiza registro - 2025-07-17 21:44:52
1 file changed, 1 insertion(+), 1 deletion(-)
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 328 bytes | 328.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To https://gitlab.com/sistemas-operacionais-embarcados/Reconhecimento-Facial.git
2209e81..1f87895 master -> master
[OK] registro.csv sincronizada com sucesso.
```

Planilha atualizada no GitLab:

registro.csv

Atualiza registro - 2025-07-17 21:44:52

Find file Blame Edit

3f878954 History

registro.csv 330 B

Code Preview

Tiffany	2025-06-26 15:15:46	2025-06-27 10:47:34	2025-06-27 14:48:59	2025-07-16 09:26:49	2025-07-16 10:14:02	2025-07-16 16:19:15	2025-07-16 17:00:17	2025-07-17 21:44:46
Paula	2025-06-26 15:15:46	2025-06-27 10:47:34						
Sofia	2025-06-27 16:37:19	2025-07-17 21:43:20						
Teste2	2025-07-15 10:17:29	2025-07-15 16:35:53						
Milena	2025-07-16 17:14:49							

Restrição de Tempo entre Registros

Testou-se também a funcionalidade de restrição mínima de 30 minutos entre registros para o mesmo funcionário. Ao tentar registrar um novo ponto antes desse intervalo, o sistema exibiu mensagem informando a impossibilidade e recusou o registro, como esperado.

```
[DEBUG] Previsto: Tiffany | Label: 0 | Confiança: 43.0471
[✓] Reconhecido: Tiffany
Funcionário: Tiffany. Deseja bater ponto? (s/N): s
[NEGADO] Último ponto de Tiffany foi há 1 minutos. Aguarde 30 minutos.
```

VI. CONCLUSÃO

Este projeto teve como objetivo desenvolver uma solução de controle de ponto eletrônico por reconhecimento facial, embarcada em Raspberry Pi 4, com foco em portabilidade, autonomia, baixo custo e sincronização em nuvem. A proposta se mostrou especialmente adequada para ambientes com múltiplas unidades descentralizadas, como canteiros de obras e empresas do setor da construção civil.

Ao longo do desenvolvimento, foram implementados e validados todos os módulos do sistema, desde o cadastro de novos funcionários, passando pelo treinamento e distribuição dos modelos faciais, até o reconhecimento em tempo real e o registro de ponto local e na nuvem.

Conclui-se que o sistema atende plenamente às exigências de robustez, escalabilidade e simplicidade de operação. A ausência de servidor central reduz os custos e simplifica a manutenção, enquanto a sincronização em nuvem garante a integridade e unificação dos dados em tempo real. Com isso, o sistema representa uma alternativa viável, eficiente e inovadora às soluções comerciais existentes, com grande potencial de aplicação prática em contextos industriais.

VII. REFERÊNCIAS

[1] M. Novak, R. Picek, e D. Androžec, “Overview of Smart Home Security with Raspberry,” *Proc. 7th Int. Conf. on Information Science and Systems (ICISS 2024)*, 2024

[2] [Controle de acesso - Produtos - Hikvision Brasil](#)

[3] [Controle de Acesso | Control iD](#)

[4] [Controle de Acesso | Intelbras](#)

[5] V. Kartikey e J. Arora, “Security of Smart Premises to Prevent Unauthorized Entrance by using IoT Enabled Face Recognition Technique,” *Proc. 2nd Int. Conf. on Edge Computing and Applications (ICECAA 2023)*, 20