UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Faculdade de Engenharia e Ciências de Guaratinguetá

TALES HIRO CARDOSO ISHIDA

Sistema de controle de acesso por reconhecimento facial utilizando sistemas embarcados

Guaratinguetá
2023

Tales Hiro Cardoso Ishida

Sistema de controle de acesso por reconhecimento facial utilizando sistemas embarcados

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia e Ciências do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica .

Orientador: Profº Thiago José Michelin

Guaratinguetá
2023

Ishida, Tales Hiro Cardoso
I79s Sistema de controle de acesso por reconhecimento facial utilizando sistemas embarcados / Tales Hiro Cardoso Ishida - Guaratinguetá, 2023.
50 f : il.
Bibliografia: f. 49

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia e Ciências de Guaratinguetá, 2023.
Orientador: Prof. Me. Thiago José Michelin

1. Reconhecimento facial (Computação).
2. Microcontroladores. 3. Processamento de imagens.
I. Título.

CDU 621.381

Luciana Máximo
Bibliotecária/CRB-8 3595

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

TALES HIRO CARDOSO ISHIDA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE **"GRADUANDO EM ENGENHARIA ELÉTRICA "**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Profº Dr. DANIEL JULIEN BARROS DA SILVA SAMPAIO
Coordenador

BANCA EXAMINADORA:

Profº Thiago José Michelin
Orientador/UNESP-FEG

Profº Dr. Daniel Julien Barros da Silva Sampaio
UNESP-FEG

Profº Mestrando Cristóvão José Dias da Cunha
Membro Externo

Fevereiro , 2023

DADOS CURRICULARES

TALES HIRO CARDOSO ISHIDA

NASCIMENTO 26/05/2000 - Campinas / SP

FILIAÇÃO Renato Sussumu Ishida
Patricia Borges Cardoso

A minha companheira que esteve presente em minha
infância, adolescência e se foi na metade da jornada de graduação
minha gata Hermione.

“A simplicidade é o mais alto grau de sofisticação“
(Leonardo da Vinci)

RESUMO

O avanço das tecnologias permitiu um maior grau de segurança para diversas facetas da sociedade, como biometria para operações bancárias, verificação de identidade por meio de dispositivos como celulares e até mesmo o reconhecimento facial. O reconhecimento facial é utilizado amplamente para controle de acesso em espaços como prédios, condomínios, empresas, instituições de ensino e etc. O trabalho propõe um sistema de controle de acesso por reconhecimento facial.
Neste trabalho, a obtenção de informações de face é feita por meio de fotos que são transmitidas pela internet para um servidor. As imagens são capturadas por uma câmera embarcada em uma placa de desenvolvimento que contém um microcontrolador para processamento da imagem e conexão com a internet. O servidor que recebe a imagem, utiliza um modelo pré-treinado de reconhecimento facial para cruzar os dados da imagem recebida com os dados de imagens já cadastradas no servidor, permitindo a identificação de uma pessoa já cadastrada.

PALAVRAS-CHAVE: acesso; reconhecimento facial; microcontrolador; processamento de imagem; cliente-servidor.

ABSTRACT

Technological advances have allowed a greater degree of security for various facets of society, such as biometrics for banking operations, identity verification through devices such as cell phones and even facial recognition. Facial recognition is widely used for access control in spaces such as buildings, condominiums, companies, educational institutions, etc.
In this work, obtaining face information is done through photos that are transmitted over the internet to a server. The images are captured by a camera embedded in a development board that contains a microcontroller for image processing and internet connection. The server that receives the image uses a pre-trained facial recognition model to cross the image data received with data from images already registered on the server, allowing the identification of a person already registered.

KEYWORDS: access; facial recognition; microcontroller; image processing; client-server.

LISTA DE ILUSTRAÇÕES

Figura 1 Resultado da pesquisa com *reconhecimento* e *facial* 14
Figura 2 Microcontrolador PIC18F4550 15
Figura 3 Placa de desenvolvimento baseada em STM32 16
Figura 4 ESP32-CAM 16
Figura 5 ESP32-CAM mapa de pinos 17
Figura 6 Diagrama cliente-servidor 17
Figura 7 Um exemplo de uma mensagem HTTP 18
Figura 8 Linguagens utilizadas para servidores web 19
Figura 9 Crescimento de linguagens de programação 20
Figura 10 Uma arquitetura de rede neural 21
Figura 11 Representação da relação entre saída e parâmetros da rede neural 22
Figura 12 Função de erro ao longo do treinamento 23
Figura 13 Descritores da face e seus valores 23
Figura 14 Representação de código fonte até código alvo de máquina 24
Figura 15 Diagrama interno do ESP32 27
Figura 16 Mensagem do cliente contendo a imagem. 28
Figura 17 Mensagem do servidor se o rosto estiver cadastrado. 28
Figura 18 Mensagem do servidor se o rosto não estiver cadastrado. 29
Figura 19 Linguagens mais utilizadas para embarcados 30
Figura 20 Algumas marcações de face obtidas pelo modelo face_recognition 37
Figura 21 Arquitetura geral do cliente 41
Figura 22 Arquitetura geral do servidor 42
Figura 23 Rostos cadastrados no servidor 43
Figura 24 Imagens enviadas para validação 44

LISTA DE TABELAS

Tabela 1 – Exemplos de tipos de mídia 18
Tabela 2 – Tabela de índices (BASE64) 26
Tabela 3 – Especificações do HC-SR04 33
Tabela 4 – Resultados dos testes de rostos e tempos de resposta 44
Tabela 5 – Tabelas de custo de hardware do sistema 47

LISTA DE ABREVIATURAS E SIGLAS

ARM	Advanced Risc Machine
CPF	Cadastro de pessoa física
ESP-IDF	Espressif official IoT Development Framework
GPIO	General Purpose Input Output
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
I2S	Inter-Integrated Circuit Sound Bus
JSON	JavaScript Object Notation
μC	Microcontrolador
PWM	Pulse Width Modulation
PIC	Peripheral Interface Controller
RISC	Reduced Instruction Set Computer
TCC	Trabalho de Conclusão de Curso
UART	Universal Asynchronous Aeceiver Transmitter
UNESP	Universidade Estadual Paulista
USART	Universal Synchronous Asynchronous Receiver Transmitter
Wi-Fi	Wireless Fidelity

SUMÁRIO

1 INTRODUÇÃO . . . 13
1.1 Contextualização . . . 13
1.2 Justificativas . . . 13
1.3 Objetivos . . . 14

2 REFERENCIAL TEÓRICO . . . 15
2.1 Microcontroladores . . . 15
2.1.1 Exemplos de Microcontroladores . . . 15
2.1.2 O ESP32 . . . 16
2.2 Protocolo HTTP . . . 17
2.3 Servidor Web . . . 18
2.3.1 Servidores web de maneira aplicada . . . 19
2.4 Redes neurais . . . 20
2.4.1 Um pouco da história das redes neurais . . . 20
2.4.2 Redes Neurais para reconhecimento facial . . . 23
2.5 ESP-IDF . . . 24
2.6 Base 64 . . . 25

3 DESENVOLVIMENTO . . . 27
3.1 O ESP32-CAM . . . 27
3.2 Mensagens HTTP para transmissão de imagens . . . 27
3.2.1 Implementação em código . . . 29
3.2.2 Wi-Fi . . . 29
3.2.3 Capturando uma imagem e transformando para base64 . . . 30
3.3 Enviando uma mensagem HTTP . . . 35
3.4 Servidor para processamento de imagens . . . 36
3.4.1 Python para servidores e inteligência artificial . . . 36
3.4.2 Identificando uma face . . . 38
3.5 Implementação do sistema . . . 40
3.5.1 Implementação do cliente . . . 40
3.5.2 Implementação do servidor . . . 40

4 RESULTADOS . . . 43
4.1 Testes com pessoas públicas e autor . . . 43
4.1.1 Viés algorítmico . . . 45
4.1.2 Possíveis explicações dos viéses . . . 45

5 CUSTOS DO SISTEMA . . . 47

6 CONCLUSÃO 48
6.1 Possibilidades de trabalhos futuros 48

REFERÊNCIAS 49

ANEXO A – LINK PARA OS CÓDIGOS 50

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A identificação de pessoas ao longo da história aconteceu de diversas maneiras, seja por meio de números identificadores de cadastro de pessoa física (CPF), biometria ou reconhecimento facial.

Esse último método de identificação só foi possível com o advento da tecnologia computacional, que com certas topologias de algoritmos tem a capacidade de detectar e descobrir padrões baseando-se em dados apresentados, como imagens, tabelas, áudio, textos, etc (PATTNAIK; MOHANTY, 2020).

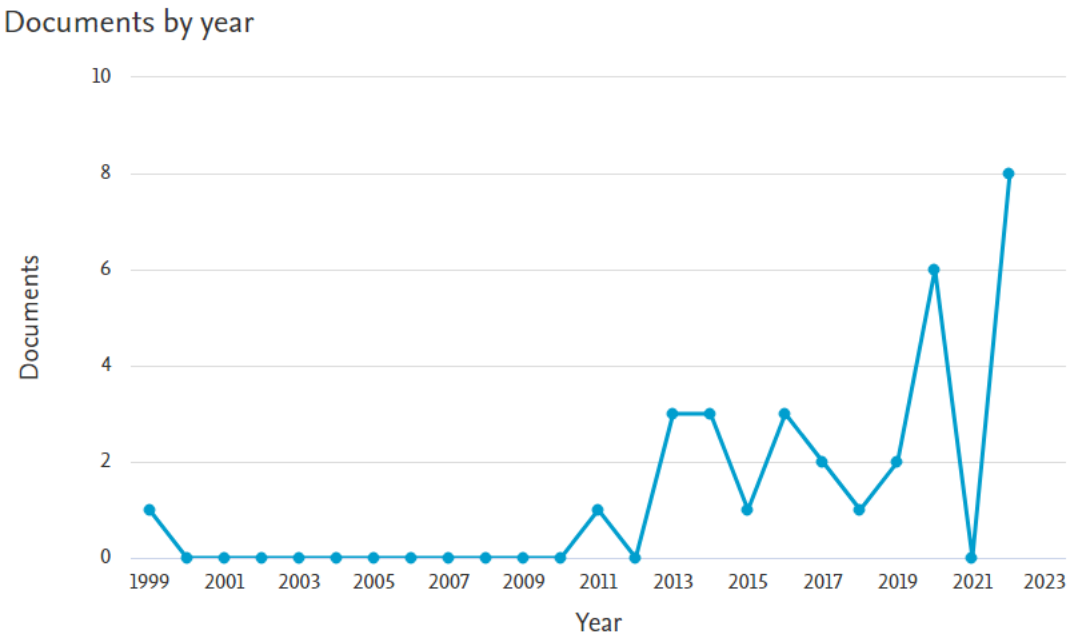
Modelos computacionais são somente uma parte do processo de identificação por meio de reconhecimento facial, também é necessário todo um aparato eletrônico capaz de captar imagens. O *Hardware* de obtenção de imagens é composto, na maioria das vezes, por um sensor óptico capaz de captar luz, aliado a unidades de processamento para, assim, obter uma imagem ou vídeo por meio de composição sucessiva de imagens.

O armazenamento de imagens para reconhecimento facial obtidas por meio de aparato eletrônico possibilita estabelecer uma relação entre imagens que já foram processadas e imagens a serem processadas. Isso permite uma identificação de algo, ou alguém, que já foi previamente classificado pelo modelo computacional. O modelo para classificação muitas vezes utiliza algoritmos de inteligência artificial para reconhecer os padrões da imagem. Alguns desses algoritmos se inspiram no comportamento do cérebro humano para identificação de padrões com estruturas chamadas redes neurais artificiais, enquanto outros partem de métodos estatísticos para classificação ou regressão (WANG, 2003).

1.2 JUSTIFICATIVAS

O trabalho a ser apresentado aborda reconhecimento facial, sistemas embarcados e comunicação do tipo cliente-servidor. Executando uma pesquisa na plataforma *Scopus* com as palavras chave *facial* e *reconhecimento*, é possível notar uma crescente (Figura 1) na literatura de reconhecimento facial. Uma pesquisa adicionando a palavra *embarcados* não retornou resultados.

Figura 1 – Resultado da pesquisa com *reconhecimento* e *facial*



fonte: Scopus (2023).

1.3 OBJETIVOS

Com um modelo de reconhecimento facial e o aparato eletrônico necessário é possível implementar um sistema para identificação de pessoas por meio das características únicas de uma face humana. Essa implementação é fundamental para a construção de sistema de controle de acesso por reconhecimento facial.

Este trabalho explora as possíveis implementações e propõe uma maneira de abordar o enunciado acima. A implementação tem a proposta de utilizar um microcontrolador capaz de se comunicar por HTTP *(Hypertext Transfer Protocol)* com um servidor e enviar imagens que serão validadas por meio de um algoritmo de reconhecimento facial. Esse algoritmo cruza dados recebidos com dados já cadastrados em uma base de dados para permitir o controle de acesso genérico baseado em reconhecimento facial.

2 REFERENCIAL TEÓRICO

2.1 MICROCONTROLADORES

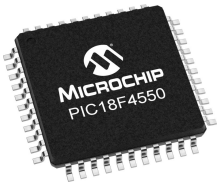
Tarefas de manipulação de dados necessitam de uma unidade de processamento computacional, isto é, um dispositivo capaz de executar operações bem definidas, sejam sequenciais ou simultâneas. Esses dispositivos podem ser computadores convencionais, servidores de rede, celulares, *notebooks* ou microcontroladores.

Microcontroladores são dispositivos que possuem unidades de processamento de dados, porém há uma grande diferença entre um microprocessador e um microcontrolador. Os microprocessadores são *chips eletrônicos* capazes de executar instruções lógicas definidas por meio de combinações de nível de tensão elétrica em seus terminais e também possuem terminais para interfaceamento com outros *chips eletrônicos*. Um microcontrolador também possui a capacidade de executar instruções lógicas, entretanto, esses dispositivos possuem circuitos internos chamados de periféricos que acrescentam mais funções, como um módulo de conversão analógica, um gerador de PWM *(Pulse Width Modulation)*, temporizadores, contadores, etc (GRIDLING; WEISS, 2007).

2.1.1 Exemplos de Microcontroladores

No mercado existem diversos modelos de microcontroladores disponíveis. A empresa *Microchip* possui a linha PIC *(Peripheral Interface Controller)* (Figura 2) que foi muito popular nos anos 2000. Em 2002 os PICs foram os microcontroladores de 8 bits mais vendidos no mundo (SMITH, 2005). A *STMicroelectronics* (Figura 3) possui uma linha de microcontroladores com a arquitetura ARM *(Advanced RISC Machine)* com um número de transistores reduzidos, que traz uma resposta térmica mais agradável (KOMMU; KANCHI, 2013). A arquitetura RISC possui um conjunto de instruções mais reduzido e o código compilado é formado de instruções simples desse conjunto.

Figura 2 – Microcontrolador PIC18F4550



fonte: Microchip Tecnology (2023).

Figura 3 – Placa de desenvolvimento baseada em STM32

fonte: STMicroelectronics (2023).

2.1.2 O ESP32

Em 2016, a empresa chinesa *Espressif Systems* lançou sua linha de microcontroladores chamada de *ESP32*, com a proposta de ser um dispositivo voltado para *IOT (Internet das coisas)*, ou seja, possibilita interconexão de dispositivos físicos (como sensores, atuadores e dispositivos eletrônicos) através da internet, permitindo que eles se comuniquem e troquem informações entre si e com sistemas baseados em nuvem. O ESP32 possui suporte nativo para conectividade *Wi-Fi* e *Bluetooth*, que pode ser mais um padrão de comunicação sem fio de curto alcance, que permite a conexão entre dispositivos eletrônicos para troca de informações, como arquivos, música, chamadas telefônicas e outros tipos de dados. Além de possuir alta popularidade e uma comunidade extremamente ativa, é um dispositivo com um amplo suporte na parte de desenvolvimento de *firmware* pela própria fabricante. Outra vantagem do *ESP32* são as abstrações para aplicações comuns já prontas e novas funcionalidades se baseiam na manipulação do *framework* de codificação da *Espressif Systems*

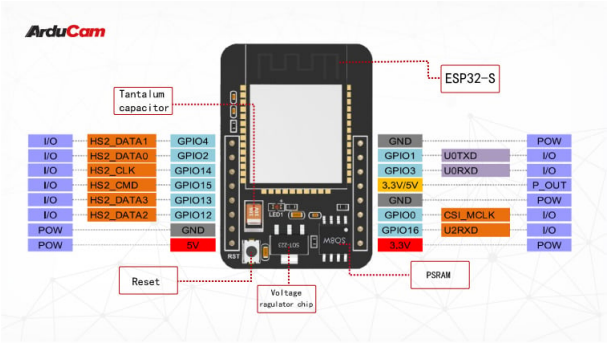
Além do suporte ao desenvolvimento fornecido pela fabricante da família de microcontroladores ESP32, existem diversas placas de desenvolvimento presentes no mercado para diferentes aplicações.

Para atender diversos requisitos de forma barata e eficiente é possível utilizar uma placa de desenvolvimento criada pela *Espressif Systems* chamada de ESP32-CAM (Figura 4). Se trata de um *hardware* provido de uma pequena câmera e um microcontrolador ESP32, mais toda a circuitaria de alimentação do microcontrolador, além de uma interface USB e uma antena capaz de captar sinais de Wi-Fi permitindo conexão à internet (Figura 5).

Figura 4 – ESP32-CAM

fonte: Espressif Systems (2020).

Figura 5 – ESP32-CAM mapa de pinos

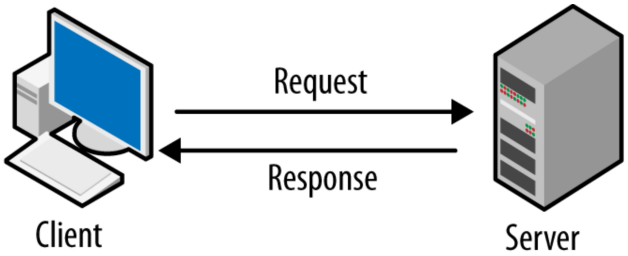


fonte: ArduCAM (2020).

2.2 PROTOCOLO HTTP

O protocolo HTTP permite a comunicação no modelo cliente-servidor (Figura 6) entre diferentes dispositivos conectados à internet pela Rede Mundial de Computadores *(World Wide Web)*. O protocolo HTTP utiliza um sistema de requisição-resposta, ilustrando a aplicação desse estudo como exemplo: O dispositivo cliente (ESP32) faz uma requisição a um servidor e essa requisição possui uma mensagem que pode ter diferentes tipos de informação. Após a mensagem ser recebida, o servidor envia uma resposta ao cliente.

Figura 6 – Diagrama cliente-servidor



fonte: Ali Madooei, Github pages (2021).

As requisições HTTP tem diferentes métodos com determinadas funções. Os métodos são informados na hora de transmitir a mensagem. O método mais interessante para o trabalho é o método POST. O método POST envia dados incluídos no corpo da mensagem. Em uma análise semântica, se trata do método HTTP para incluir informações no servidor.

O protocolo define cabeçalhos (*header*) que transmitem informações adicionais. Esses cabeçalhos podem carregar informações sobre a mensagem transmitida, sobre os clientes e as requisições. Além disso, a mensagem pode possuir um corpo não vazio. Se a mensagem for de requisição, o corpo contém as informações a serem transmitidas para o servidor e se a mensagem for de resposta, o corpo contém o que foi requisitado pelo cliente. Requisições que possuem corpo contém um cabeçalho de entidade, que descreve o tipo de mídia que será utilizado para formatação do corpo. Alguns exemplos de tipo de mídia são apresentados na Tabela 1:

Tabela 1 – Exemplos de tipos de mídia

text/plain	Formato texto (ASCII)
application/json	Formato JSON
application/zip	Arquivo compactado

fonte: Compilação do autor (2023).

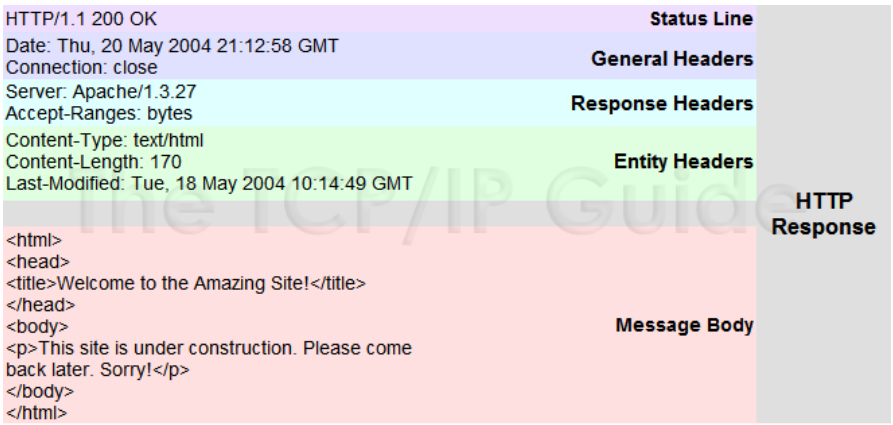
Mensagens de resposta possuem informações sobre o estado da requisição, além de todas as informações citadas acima. Essa informação de estado é transmitida ao cliente por meio de códigos de status, onde cada código de estado representa uma resposta diferente. Alguns exemplos de código de status são: 200 OK, 404 Not Found, 505 Internal Server Error (BERNERS-LEE, 1996).

2.3 SERVIDOR WEB

Um servidor *web* é um *software* ou sistema que é responsável por receber solicitações de um cliente ou usuário, processá-las e fornecer uma resposta adequada.

O servidor web utiliza protocolos de rede como o HTTP (Hypertext Transfer Protocol) citado acima, que define as regras para a troca de informações entre o cliente e o servidor. A troca de informações é feita por meio das mensagens HTTP (Figura 7) que podem enviar vários tipos de dados na forma de documentos HTML, imagens, ou simplesmente texto. Essas mensagens são transmitidas para um *endpoint*. Um *endpoint* é um ponto final de uma comunicação em uma rede, geralmente na internet. Ele é usado para descrever a localização de um recurso específico que pode ser acessado por meio de uma requisição de rede, como uma solicitação HTTP.

Figura 7 – Um exemplo de uma mensagem HTTP



fonte: The TCP/IP guide (2005).

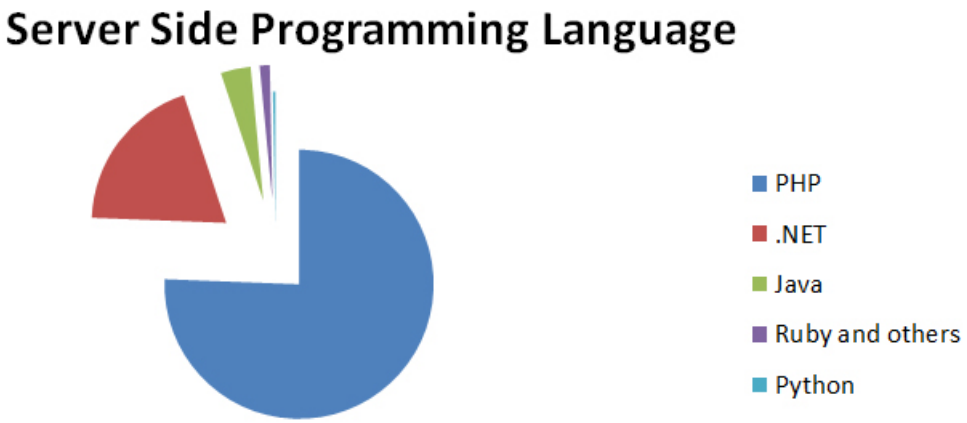
As aplicações de um servidor web são muito vastas e compreendem a maior parte da internet como conhecemos. Sites, jogos, aplicativos móveis e serviços gerais acessados pela internet interagem com servidores web para conseguir informações a todo momento.

2.3.1 Servidores web de maneira aplicada

Software de servidores web podem ser escritos com uma linguagem de programação. Várias linguagens permitem a criação de um servidor web.

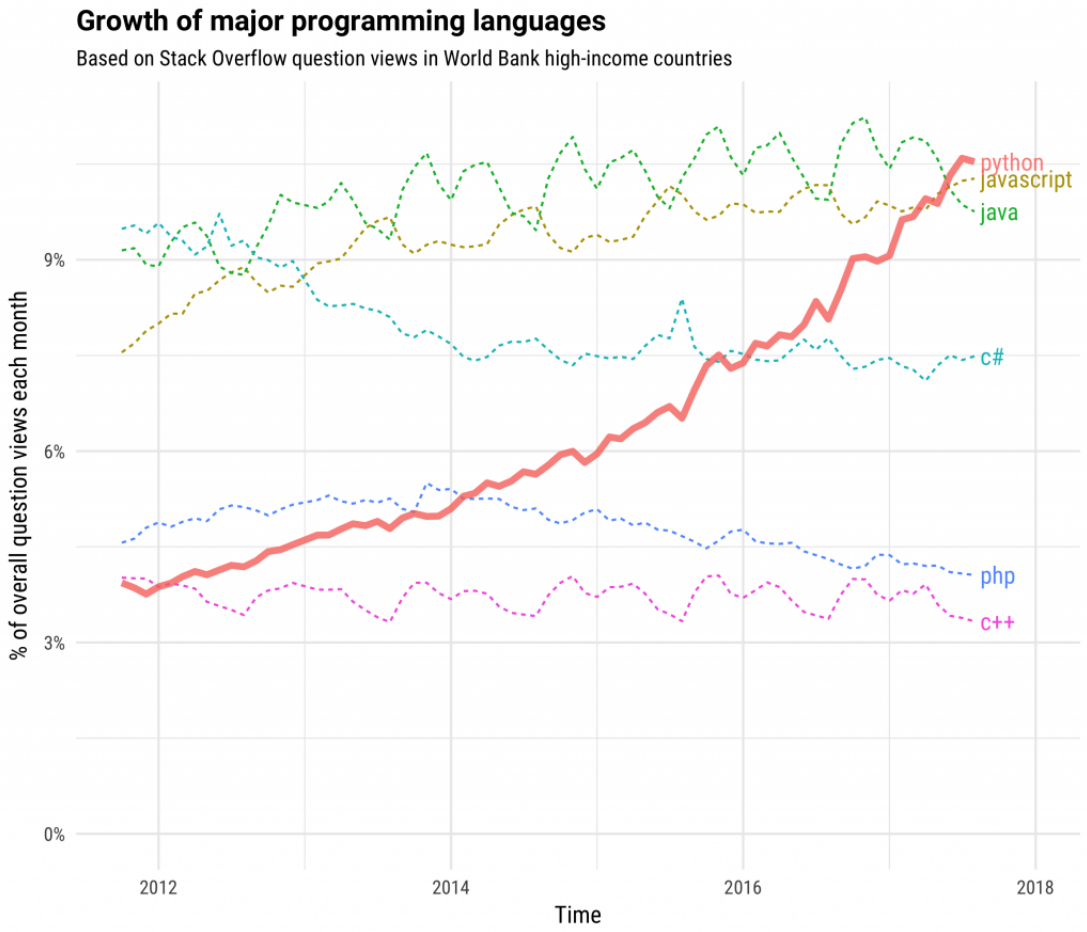
Olhando o gráfico abaixo (Figura 8) é possível notar a predominância de PHP nas aplicações de servidores web. Entretanto, uma pequena parcela é escrita em Python, que é uma linguagem que vem crescendo em popularidade ao longo dos anos. Isso pode ser explicado pela simplicidade de sua sintaxe e elegância do Python, mantendo o poder da linguagem de criar aplicações complexas.

Figura 8 – Linguagens utilizadas para servidores web



fonte: Stack Overflow (2017).

Figura 9 – Crescimento de linguagens de programação



fonte: Gerard Milles (2018).

Além das funções básicas de processamento de mensagens HTTP, um servidor também precisa dispor de funcionalidades diversas para executar funções de processamento de informações para difentes aplicações. A linguagem Python tem uma diversa comunidade de usuários por conta de sua crescente popularidade (Figura 9), isso acarreta em muitos *frameworks* e bibliotecas já validadas e prontas para uso (TANEJA; GUPTA, 2014).

2.4 REDES NEURAIS

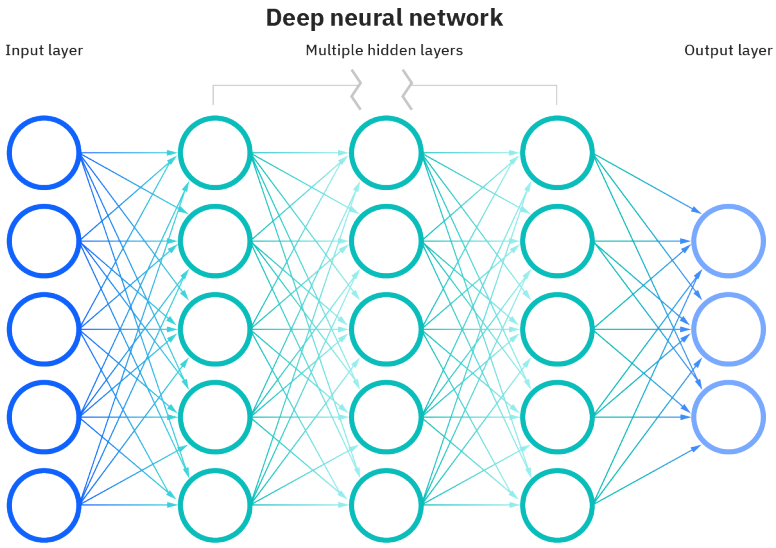
2.4.1 Um pouco da história das redes neurais

As redes neurais são um modelo matemático inspirado na estrutura biológica do cérebro humano. Elas foram propostas, pela primeira vez, na década de 1940 por Warren McCulloch e Walter Pitts, que descreveram como neurônios artificiais implementados em código poderiam ser ligados para formar uma rede capaz de processar informações de forma semelhante à forma como o cérebro humano o faz.

A história das redes neurais passa por várias etapas, desde sua concepção até sua consolidação como uma técnica de inteligência artificial. As primeiras tentativas de construir redes neurais foram feitas com elementos eletrônicos físicos, como a rede Perceptron inventada por Rosen Blatt em 1958. Porém, estas redes eram limitadas pelas tecnologias disponíveis na época (STANFORD, 2019).

As tecnologias foram evoluindo ao longo do tempo, porém sempre seguindo os princípios básicos de uma rede neural: trata-se de um aglomerado de neurônios artificiais com conexões a outros neurônios artificiais (Figura 10). As conexões são chamadas de pesos e determinam valores de saída da rede neural.

Figura 10 – Uma arquitetura de rede neural



fonte: IBM (2023).

Neurônios artificiais são elementos que recebem um número como entrada e, após executar uma função matemática, exportam como saída outro número de acordo com a função contida nesse neurônio e a entrada. Esses neurônios são interligados em camadas e assim a rede neural é alimentada com uma sequência de números que determinam uma sequência de números de saída.

Esses números de saída ativam outra camada de neurônios de acordo com os respectivos pesos conectados e esse processo se repete até a última camada, chamada de camada de saída. Então a ativação da camada de saída nada mais é do que o resultado de várias operações matemáticas, assim como a ativação de todas as outras camadas.

Na definição de ativação de um neurônio, x representa a entrada do neurônio e w representa o peso (magnitude da conexão) do neurônio e b é um valor inicial chamado de *"bias"* para inicializar a ativação dos neurônios em uma certa parte do espaço de ativação.

$$output = f(x) = 1 \text{ se } \sum w_1 x_1 + b \geq 0 \tag{1}$$

$$output = f(x) = 0 \text{ se } \sum w_1 x_1 + b < 0 \tag{2}$$

As redes neurais mais utilizadas são redes neurais supervisionadas onde o ajuste dos pesos de ativação é feito a partir do cálculo de erro entre o que é previsto e o que deve ser previsto, para isso são

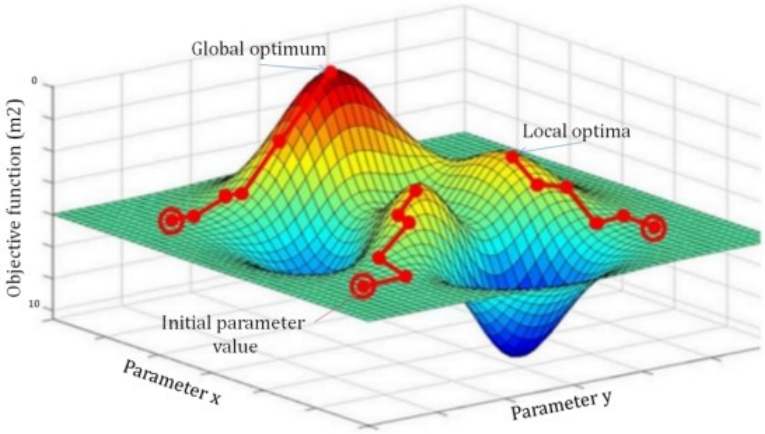
necessários dados que possuam valores já observados e confirmados como corretos e a entrada que previu esses valores.

Quando a rede observa alguma entrada, faz uma previsão e compara com o que deve ser previsto de acordo com aquela entrada, assim, é possível calcular o erro. Após isso, os pesos dos neurônios são ajustados a fim de minimizar a função de erro da rede neural.

Uma rede neural pode possuir muitos neurônios, até milhares. Então o espaço de ajuste dos pesos pode possuir n dimensões, e as alterações dos pesos causa uma saída diferente. Uma representação famosa do erro pelos ajustes dos parâmetros é a representação tridimensional (Figura 11).

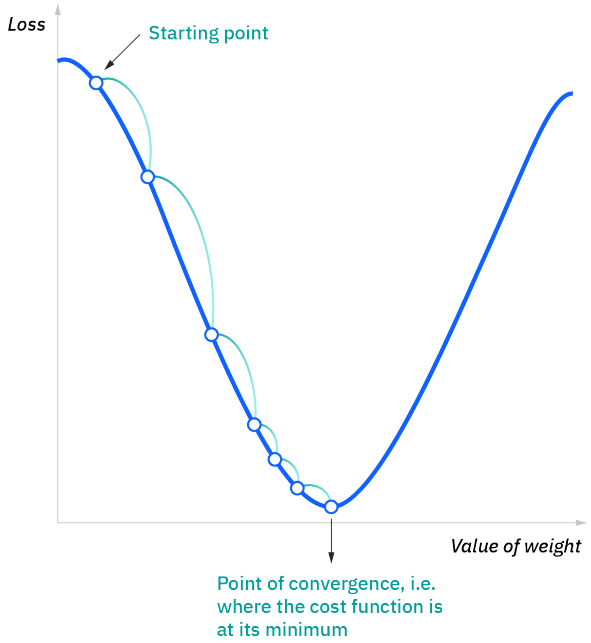
Na imagem da Figura 12 é possível ver que a função se altera de acordo com os parâmetros, e isso é a finalidade geral da rede neural. Achar o conjunto e valores de pesos e parâmetros ótimos para que a rede encontre uma função que descreva as saídas já observadas a partir de entradas já também observadas para depois ser utilizada para prever saídas com entradas não observadas, executando uma previsão acurada e precisa. A diferença entre redes neurais simples e redes neurais profundas são somente a quantidade de camadas e dimensão da entrada do modelo de rede neural.

Figura 11 – Representação da relação entre saída e parâmetros da rede neural



fonte: UFPR (2021).

Figura 12 – Função de erro ao longo do treinamento

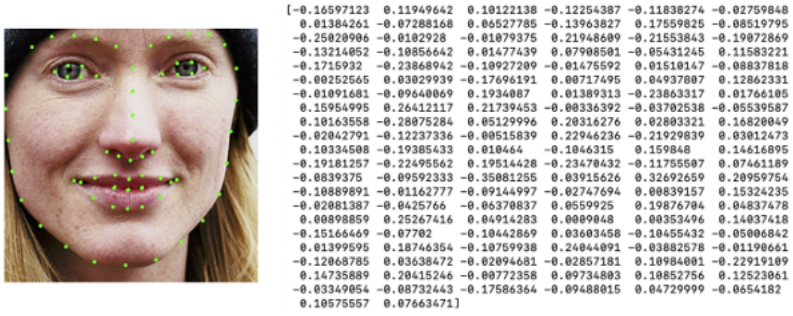


fonte: IBM (2023).

2.4.2 Redes Neurais para reconhecimento facial

As redes neurais podem ser construídas para atender diferentes aplicações, como identificação de fraudes bancárias ao extrair padrões anômalos de transferência de crédito, predição de séries temporais como previsão de temperatura e até mesmo reconhecimento facial. Os padrões a serem buscados por uma rede neural em uma tarefa de reconhecimento facial são chamados de descritores da face. Os descritores da face são uma série de distâncias, medidas e características de uma face, esses descritores podem ser mais tangíveis como a distância entre olhos, tamanho da boca, etc. Porém, uma rede neural tem a capacidade de reconhecer padrões de descritores não mapeados por seres humanos, por isso a arquitetura de rede neural se mostra útil para tarefa de reconhecimento facial (Figura 13).

Figura 13 – Descritores da face e seus valores

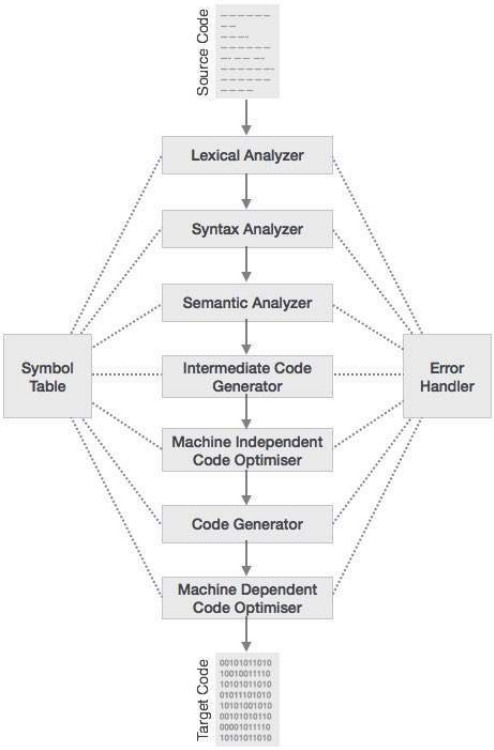


fonte: Tutorials point (2020).

2.5 ESP-IDF

A programação de microcontroladores segue os mesmos axiomas de qualquer outro tipo de codificação: sequencial e orientada a instruções bem definidas. A codificação pode ser feita em diversas linguagens como C, assembly, C++ e até Python. Independente da linguagem escolhida, o código se transforma em instruções sequenciais a nível de hardware (Figura 14), determinadas por diferentes níveis de tensão em terminais da unidade de processamento do microcontrolador.

Figura 14 – Representação de código fonte até código alvo de máquina



fonte: Tutorials point (2020).

A forma como o usuário interage com o código pode conter diferentes níveis de abstração. O nível mais baixo é a manipulação de registradores do μC (Microcontrolador), ou seja, determinar valores em posições de memória que determinam as funções do μC.

Outros códigos de microcontroladores podem utilizar abstrações da programação orientada a objetos para codificação. Exemplos aparecem em dispositivos mais recentes como o Arduino com a Arduino framework (ARDUINO, 2023, Disponível em: https://www.arduino.cc/. Acesso em 17 Jan. 2023), que tem diversas funções que envelopam as instruções de máquina e de manipulação de registradores, tornando a velocidade de desenvolvimento de *firmwares* muito mais rápida. Inicializar uma porta serial com a Arduino framework é tão simples quanto chamar um método de uma classe.

Muitos fabricantes de microcontroladores, como a *Espressif Systems*, desenvolvem bibliotecas, *frameworks* e middlewares próprios para seus dispositivos. O framework desenvolvido pela *Espressif Systems* se chama ESP-IDF. O framework tem estruturas de dados, classes, métodos, funções e macros que abstraem muitas configurações necessárias de periféricos de microcontroladores. Um exemplo de código utilizando a ESP-IDF é mostrado abaixo.

Exemplo de código utilizando a ESP-IDF

```
### CONFIGURANDO UM PWM COM A ESP-IDF

void pwmConfig(){

   mcpwm_config_t pwm_config;

   pwm_config.frequency = 5000;
   pwm_config.cmpr_a = 0;
   pwm_config.cmpr_b = 0;
   pwm_config.counter_mode = MCPWM_UP_COUNTER
   pwm_config.duty_mode = MCPWM_DUTY_MODE_0;

   mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
   mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, PWMA);
   mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0B, PWMB);
}
```

2.6 BASE 64

Base64 é um método de codificação de dados que é usado para representar arquivos binários (como imagens, vídeos, áudios) na forma de uma sequência de caracteres ASCII. Ele permite que esses arquivos sejam transmitidos através de canais que não suportam a transmissão direta de dados binários, como mensagens de texto.

A codificação Base64 é realizada dividindo os dados binários em conjuntos de seis bits e, em seguida, representando cada conjunto como um caractere ASCII correspondente de acordo com a Tabela 2. Isso permite que os dados binários sejam representados com uma taxa de compressão de 75%. Além disso, ele adiciona uma camada adicional de segurança às informações, pois os dados codificados são menos propensos a serem alterados durante a transmissão.

Por exemplo, os bits de informação que descrevem uma imagem podem ser transformados em uma sequência de caracteres para transmitir a imagem como uma variável do tipo *string* em C++.

Tabela 2 – Tabela de índices (BASE64)

Tabela de índices BASE64							
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

fonte: Compilação do autor (2023).

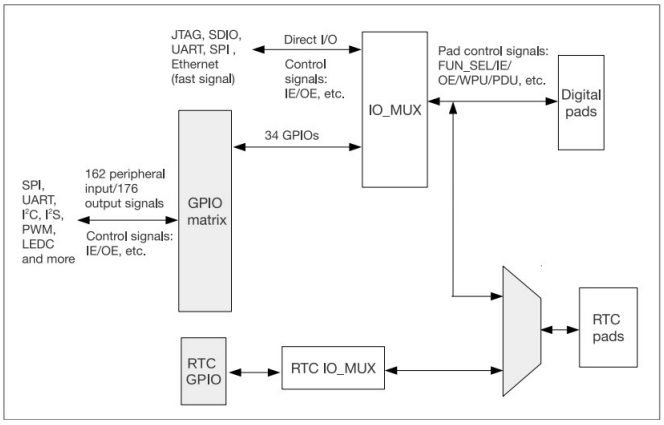
3 DESENVOLVIMENTO

3.1 O ESP32-CAM

A placa de desenvolvimento ESP32-CAM pode ser utilizada em diversas aplicações. A fabricante projetou uma placa capaz de se comunicar por diversos protocolos de comunicação para troca de informações como o I2C *(Inter-Integrated Circuit)*, UART *(Universal Asynchronous Receiver / Transmitter)*, USART *(Universal Synchronous/Asynchronous Receiver/Transmitter)*, I2S *(Integrated Inter-IC Sound Bus)*, SPI *(Serial Peripheral Interface)*. O chip possui geradores de PWM, diversos pinos de GPIO *(General Purpose Input Output)*, timers e outros periféricos que tornam esse μC um dispositivo muito completo e com alta aplicabilidade. A complexidade de multiplexação dos pinos do ESP32 pode ser observada no diagrama abaixo (Figura 15).

Com o intuito de construir um sistema de controle de acesso baseado em reconhecimento facial, é necessário um sensor óptico aliado de uma unidade de processamento e esses requisitos estão disponíveis com a placa ESP32-CAM. A funcionalidade presente mais importante e com maior peso para este estudo é a possibilidade de conexão à internet. Essa conectividade nativa na placa de desenvolvimento, aliada a bibliotecas de código para manipulação de mensagens HTTP, abstraem muitas tarefas na implementação de um sistema que se comunica por meio da internet. O protocolo HTTP será utilizado para a transferência de imagens capturadas pela câmera da placa de desenvolvimento até um servidor, que será encarregado de fazer o processamento dos dados enviados pelo μC.

Figura 15 – Diagrama interno do ESP32



fonte: Espressif Systems (2017).

3.2 MENSAGENS HTTP PARA TRANSMISSÃO DE IMAGENS

As mensagens HTTP vão efetivamente transmitir os dados, e para a aplicação precisa-se ter um sistema bem simples de mensagens. Primeiramente é necessário enviar uma mensagem contendo a imagem que será processada pelo servidor. Essa imagem contém um rosto que será identificado pelo ser-

vidor. Com essa necessidade, uma proposta de mensagem contendo a imagem é mostrada na Figura 16:

Figura 16 – Mensagem do cliente contendo a imagem.

```
POST /publish-image HTTP/1.1

Host:server.com

{"id": "new-image",
"image": "fUOiDp0rQ/wC...QM4nNHxmNNAAAA===="\}

 Content-Type: application/json
```

fonte: Compilação do autor (2023).

Essa mensagem HTTP funciona da seguinte forma. O método da mensagem é o POST, ou seja, tem a função de transmitir dados para o servidor e os dados são transmitidos por meio de um formato denominado JSON *(JavaScript Object Notation)*. O formato é especificado pelo cabeçalho de entidade: "application/json".

Na mensagem acima, o corpo da mensagem no formato JSON possui dois campos: "id", que é um identificador do tipo de imagem sendo transmitida, neste caso, uma nova imagem e "image" que é uma sequência de caracteres que correspondem a uma imagem codificada em base 64.

Como é mencionado na seção 2.6, a codificação por base 64 é simples, uma imagem é convertida em uma sequência de caracteres que pode ser transmitida e decodificados para obter a imagem original. (JOSEFSSON, 2006)

A resposta da requisição deve informar se a imagem enviada contém um rosto que está cadastrado no servidor. Se o rosto estiver cadastrado, o servidor enviará a mensagem da Figura 17 para o cliente.

Figura 17 – Mensagem do servidor se o rosto estiver cadastrado.

```
HTTP/1.1 200 OK

Content-type: text/plain

Content-length: 10

Registered
```

fonte: Compilação do autor (2023).

Se o rosto da imagem não estiver cadastrado, o servidor enviará a mensagem da Figura 18 para o cliente.

Figura 18 – Mensagem do servidor se o rosto não estiver cadastrado.

```
HTTP/1.1 200 OK
Content-type: text/plain
Content-length: 14
Not Registered
```

fonte: Compilação do autor (2023).

3.2.1 Implementação em código

Com essas mensagens definidas, podemos utilizar o ESP32 para fazer a captura da imagem, processar e converter para base64 e, após isso, deve-se montar a mensagem HTTP e enviar para o servidor.

Para essa implementação, pode-se utilizar bibliotecas já existentes disponibilizadas pela fabricante do ESP32, espressif systems (ESPRESSIF, 2022, Disponível em: https://github.com/espressif/arduino-esp32. Acesso em: 19 Jan. 2023). Essas bibliotecas tornam a implementação mais rápida e extensível por já possuírem diversas estruturas de dados, funções, classes e outras entidades de código que já estão prontas para uso em diversas aplicações.

As principais bibliotecas vão lidar com o protocolo HTTP, a conexão Wi-Fi, e a conversão de imagens para base 64. São escritas na linguagem C++, que é uma das linguagens mais utilizadas para programação de aplicações embarcadas e microcontroladores no mundo (Figura 19).

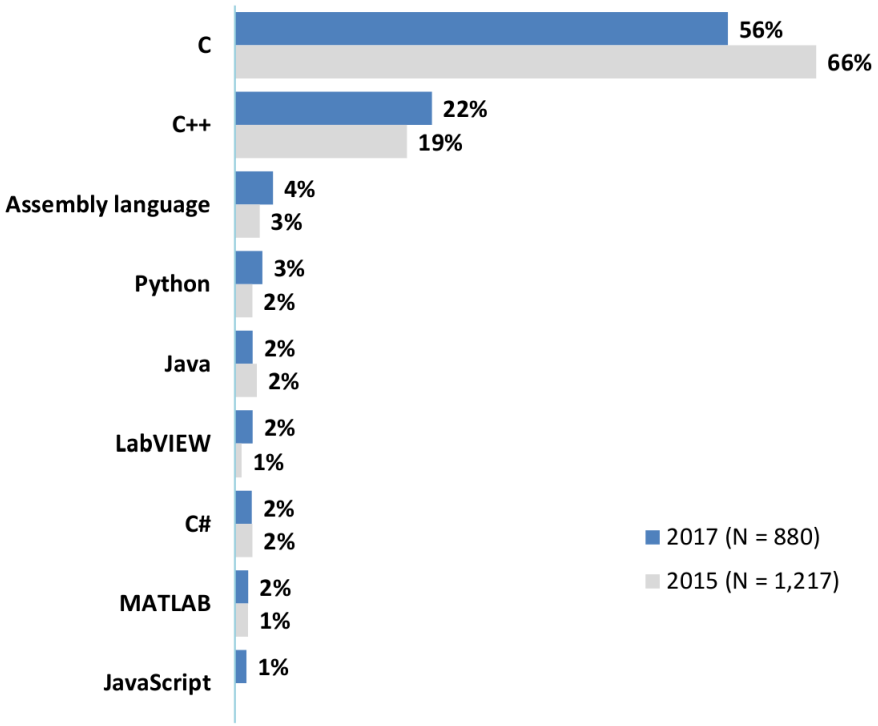
3.2.2 Wi-Fi

Antes de conectar-se à rede Wi-fi, podemos inicializar uma das portas seriais do ESP32 para atuar como *log* de informações. Para tanto, inicia-se com *baud rate* de 115200 bps utilizando a linha *Serial.begin(115200)*, após isso podemos iniciar a conexão ao Wi-Fi com as funções *WiFi.mode()* e *Wifi.begin()* (ESPRESSIF, 2022a) da biblioteca "WiFi"(<WiFi.h>) para ESP32. Os parâmetros necessários são o *ssid* e a senha da rede a qual o ESP32 será conectado. Finalmente, transmite-se um caractere de ponto (".") na porta serial enquanto a conexão Wi-Fi não estiver completa.

Inicialização para conexão à internet

```
#include <WiFi.h>

Serial.begin(115200);
delay(20);

WiFi.mode(WIFI_STA);

Serial.println("");
Serial.print("Connecting to ");
```

Figura 19 – Linguagens mais utilizadas para embarcados



fonte: Aspencore (2017).

```
Serial.println(ssid);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED)
{
  Serial.print(".");
  delay(500);
}

Serial.println("");
Serial.println("STAIP address: ");
Serial.println(WiFi.localIP());
```

3.2.3 Capturando uma imagem e transformando para base64

Para capturar uma imagem utilizando a ESP32-CAM, temos que definir algumas *macros* no código para atribuição de pinos. As *macros* são pedaços de códigos em que é atribuído um nome a um certo valor, que pode ser uma *string*, um valor númerico, etc. Esse nome, se encontrado pelo pré-processador na compilação, é substituído pelo pedaço de código atribuído a ele. Segue um exemplo de utilização de uma *macro*.

Exemplo de utilização de MACRO

```
#define MACRO "Hello, World!"

int main()
{
   printf(MACRO);

   return 0;
}
```

Esse programa é equivalente ao seguinte código, se não houvesse a utilização de *macros*.

Código equivalente ao exemplo de utilização de MACRO

```
int main()
{
   printf("Hello, World!");

   return 0;
}
```

Algumas macros são utilizadas para definir as conexões dos pinos da câmera embarcada na placa de desenvolvimento. A atribuição correta dos pinos é necessária para utilizar o *framework* de desenvolvimento da fabricante. As *macros* definidas a seguir seguem as informações contidas na documentação do ESP32 (ESPRESSIF, 2022b).

Atribuição de pinos da câmera.

```
#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27

#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22
```

Concluída esta etapa, devemos configurar a câmera criando uma *struct* que define a atribuição de cada pino com as macros anteriores.

Configuração da câmera.

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.frame_size = FRAMESIZE_UXGA;
config.pixel_format = PIXFORMAT_RGB565;
config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
config.fb_location = CAMERA_FB_IN_PSRAM;
config.jpeg_quality = 10;
config.fb_count = 1;
```

Finalmente, para terminar a configuração, uma função que recebe a *struct* montada como parâmetro é invocada para executar as instruções de configuração. Caso haja algum erro de configuração, uma mensagem é colocada na porta serial indicando o erro e o microcontrolador é reiniciado com a linha *ESP.restart()*.

Inicialização da câmera.

```
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK)
{
  Serial.printf("Camera init failed with error 0x%x", err);
  delay(1000);
```

```
    ESP.restart();
  }
```

Com essas instruções, a câmera já foi configurada porém ainda é preciso determinar quando será necessário tirar uma foto. Isso pode ser feito por meio de um sensor ultrassônico de distância simples de baixo alcance que será responsável por detectar se algo se aproximou da câmera. Uma das soluções possíveis é um sensor ultrassônico como o HC-SR04.

Esse sensor funciona de maneira muito simples: é alimentado com uma fonte de tensão de 5V e, ao detectar um objeto em sua frente, envia um nível lógico de tensão alto em um dos terminais. O seu alcance pode variar de 2 cm até 4 m e o consumo de corrente é cerca de 15 mA. Algumas especificações do sensor contidas na folha de dados do componente são mostradas na Tabela 3.

Tabela 3 – Especificações do HC-SR04

Property	Value
Working Voltage	DC 5V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
Measuring Angle	15 degree
Trigger Input Signal	10 μS TTL pulse

fonte: Folha de dados do HC-SR04 (2023).

O sensor é conectado em uma das portas do microcontrolador com capacidade de leitura digital e a configuração dessa porta é simples. É definida uma macro para o nome desse pino e é chamada a função *pinMode(pino, modo)* com os parâmetros para configurar o terminal como uma entrada de sinal.

Configuração da porta do microcontrolador para utilização do sensor ultrassônico.

```
#define INFRARED_SENSOR 2

pinMode(INFRARED_SENSOR, INPUT);
```

Após essa configuração, podemos iniciar um laço infinito que checa se o sensor foi ativado. Caso o sensor seja ativado, o microcontrolador tira uma foto chamando a função *saveCapturedImage()* e espera 3 segundos para tirar outra foto.

Laço infinito para captura de imagem.

```
void loop()
{
  if(digitalRead(infrared) == true){
    saveCapturedImage();
    delay(3000);
  }
}
```

A função *saveCapturedImage()* é responsável por capturar uma foto e enviar para o servidor que irá processar a foto de uma pessoa. Essa foto será utilizada para cruzamento de dados com um banco de dados de rosto para determinar se alguém está cadastrado ou não em um sistema de controle de acesso.

Para isso, é preciso criar um objeto, disponível na ESP-IDF (ESPRESSIF, 2022, Disponível em: https://github.com/espressif/arduino-esp32. Acesso em: 19 Jan. 2023) da classe *WifiClient*, que tem diversos métodos para utilizar uma conexão Wi-Fi no modo cliente. Depois da criação do objeto, uma foto é tirada e armazenada em um *buffer* com tamanho específico determinado pelo tipo *camera_fb_t()*, também presente na ESP-IDF. Finalmente, o programa checa se a foto foi tirada com sucesso e reinicia caso a captura de foto falhe.

Procedimento para captura de imagem.

```
void saveCapturedImage()
   {
    WiFiClient client;

    camera_fb_t *fb = NULL;
    fb = esp_camera_fb_get();
    if (!fb)
    {
     Serial.println("Captura Falhou!");
     delay(1000);
     ESP.restart();
     return;
    }
```

Para enviar a foto para o servidor, é preciso montar uma mensagem HTTP, que vai possuir dois campos de informação em formato JSON que será inserido no corpo da mensagem e um dos campos será a imagem a ser transmitida. Para a transmissão, codifica-se a imagem em base64 e o resultado da codificação será o conteúdo do corpo da mensagem HTTP.

Para codificar a imagem em base 64, podemos utilizar bibliotecas disponibilizadas pela comunidade de desenvolvimento de sistemas microcontrolados. A biblioteca "Base64.h"(ADAMVR, 2013, Disponível em: https://github.com/adamvr/arduino-base64. Acesso em: 18 Jan. 2023) possui métodos prontos para conversão de qualquer sequência de bits para base64. Primeiro, uma cópia do *buffer* é criada com o tipo adequado para servir como parâmetro da função de conversão, depois é criado um novo *buffer* que armazena os caracteres após a conversão. Converte-se o *buffer* de saída para o tipo *string*, em um formato *url-safe*, ou seja, que só possui caracteres ASCII.

Conversão da imagem para base64.

```
void saveCapturedImage(){
  ...
```

```
  char *input = (char *)fb->buf;
  char output[base64_enc_len(3)];
  String image_photo = "";
  for (int i = 0; i < fb->len; i++)
  {
    base64_encode(output, (input++), 3);
    if (i % 3 == 0)
      image_photo += urlencode(String(output));
  }
}
```

3.3 ENVIANDO UMA MENSAGEM HTTP

Após montar a *string* que representa a imagem, pode-se utilizar a biblioteca <*HTTPClient.h*> para iniciar um cliente HTTP com o objeto da classe *HTTPClient* presente na ESP-IDF e passar os parâmetros para a conexão, como o objeto responsável pela conexão de internet e o *endpoint* do servidor para acessar o serviço de processamento da imagem.

A montagem da mensagem começa definindo os cabeçalhos da mensagem HTTP. Neste caso o cabeçalho determina o tipo de mídia a ser transferida pelo corpo da mensagem. Para transferir uma mensagem no formato JSON, o método *addHeader* é chamado com os parâmetros adequados e o corpo da mensagem é montado logo em seguida com a *string* que representa uma imagem. O JSON pode seguir o seguinte formato para se comunicar com o servidor.

JSON para transmissão de mensagens

```
{
   "id": "new-image",
   "image":"/9jxmpr...o7VRLGUUWPbUn//2Q=="
}
```

Sendo *id* o campo de identificação do conteúdo da mensagem e o campo *image* a string contendo a imagem.

Por fim, a mensagem HTTP pode ser enviada com o método POST, que especifica uma adição de dados ao servidor. Para efeito de visualização, a resposta do servidor é enviada para a porta serial do microcontrolador e os cabeçalhos e dados do objeto *http* são limpos.

Montagem da mensagem HTTP

```
void saveCapturedImage(){
...
  HTTPClient http;

  http.begin(client, serverName);
```

```
    http.addHeader("Content-Type", "application/json");
    String post_data = "{\"id\":\"new-image\",\"image\":\"" +
       image_photo + "\"}";
    int httpResponseCode = http.POST(post_data);

    Serial.print("HTTP Response code: ");
    Serial.println(httpResponseCode);


    http.end();
  }
```

3.4 SERVIDOR PARA PROCESSAMENTO DE IMAGENS

As mensagens montadas nas seções anteriores devem ser enviadas para um servidor que executará a validação das imagens recebidas. Essa validação pode ser feita por meio de algoritmos de reconhecimento facial executados pelo servidor.

Existem diversas maneiras de criar um servidor para recebimento das mensagens HTTP. O conceito de servidor é agnóstico em relação a ferramenta de criação, ou seja, a linguagem de programação, *framework* ou método de implementação podem ser diferentes entre servidores porém as funcionalidades do servidor não são limitadas nem definidas por essas variáveis.

3.4.1 Python para servidores e inteligência artificial

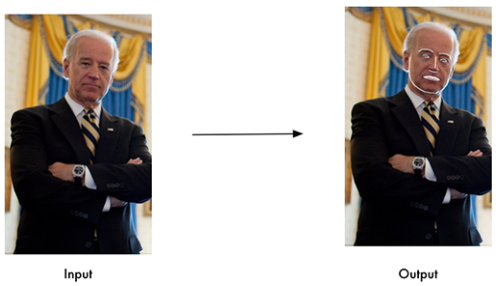
Levando em consideração as funcionalidades necessárias do servidor que executa uma tarefa de reconhecimento facial, a linguagem mais apropriada para sua criação é o Python devido à enorme quantidade de bibliotecas. Algumas destas bibliotecas incluem funcionalidades de manipulação de imagens, manipulação de grande quantidade de dados, modelagem de algoritmos de inteligência artificial e uma infinidade de funcionalidades.

O Python conta com diversos pacotes de tarefas de aprendizado de máquina como o *Scikit Learn* (PEDREGOSA et al., 2011) ou o pyTorch (PASZKE et al., 2019). Entretanto, esses pacotes são para criação de arquiteturas e de treinamento de algoritmos de aprendizado de máquina. Isso permite um maior controle sobre o algoritmo desenvolvido, porém requer uma capacidade computacional muito mais elevada. Em uma tarefa de aprendizado de máquina, o tempo de treino de um algoritmo é absurdamente maior do que o tempo de predição de um algoritmo já treinado.

Nessa ótica, a melhor opção para a tarefa de reconhecimento facial por meio de redes neurais, combinada com a necessidade de resposta rápida de um servidor, é a utilização de um modelo pré-treinado, ou seja, uma arquitetura já criada com o intuito de executar uma tarefa extremamente específica e treinada para a mesma.

Um modelo pré-treinado de reconhecimento facial que utiliza redes neurais profundas pode ser encontrado na biblioteca *Face Recognition* (AGEITGEY, 2022, Disponível em: https://github.com/ageitgey/face_recog Acesso em: 19 Jan. 2023). O modelo promete 99,38% de acurácia na tarefa de reconhecimento facial. O modelo tem a capacidade de identificar características da face como distância dos olhos, tamanho, relação entre olhos, tamanho da boca, simetria da boca, distância do nariz até a boca e diversos outros descritores de uma face (Figura 20).

Figura 20 – Algumas marcações de face obtidas pelo modelo face_recognition



fonte: Ageitgey (2022).

Além do algoritmo de reconhecimento facial, também é necessário a implementação de um servidor, que recebe e processa mensagens HTTP. Para isso, utiliza-se as classes *BaseHTTPRequestHandler* e *HTTPServer* contidas no pacote *http.server*. As outras funcionalidades são importadas com suas respectivas bibliotecas demonstradas abaixo

Importação de bibliotecas.

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import simplejson
import base64
import face_recognition
from glob import glob
```

O código utiliza o pacote *simplejson* para manipulação de arquivos do formato JSON, o pacote *base64* para manipulação de dados no formato base64, o pacote *face_recognition* para a tarefa de reconhecimento facial e o pacote *glob* para acesso de arquivos.

A definição das funcionalidades do servidor pode ser mapeada para uma classe em Python que possui métodos para atender às diferentes requisições de uma mensagem HTTP. Cada método de mensagem de requisição (POST, PUT, GET, DELETE) é mapeado a um método da classe. Na aplicação desse trabalho, o único método necessário será o POST, pois as mensagens enviadas ao servidor somente transmitirão uma nova imagem para ser validada para o servidor e este responderá conforme o status de cadastro do rosto identificado pelo algoritmo de reconhecimento facial.

O método que atende a requisição HTTP POST é chamado de *do_POST(self)*. O método acessa o tamanho do conteúdo da mensagem e armazena em uma variável *content_len*, lê o conteúdo contido no corpo da mensagem e mapeia para um objeto do formato JSON.

Processamento de uma mensagem HTTP.

```
class MyServer(BaseHTTPRequestHandler):
   def do_POST(self):
         content_len = int(self.headers.get('content-length',
            0))
         post_body = self.rfile.read(content_len)
         test_data = simplejson.loads(post_body)
         print("post_body(%s)" % (test_data))
```

3.4.2 Identificando uma face

Com as informações da mensagem, a imagem contida na mensagem é remontada a partir da decodificação da sequência de caracteres em base64. Essa conversão é feita a partir do pacote base64. A imagem contendo um rosto, agora em formato JPEG, é armazenada em disco no servidor para posteriormente ser comparada com rostos cadastrados em um diretório do servidor.

Inicialmente uma variável booleana que aponta se a pessoa está cadastrada no sistema é iniciada como falsa. A imagem é carregada como um objeto suportado pelo pacote *face_recognition* a partir da função *load_image_file()* e os descritores da face são obtidos a partir de *face_encodings()*.

Obtenção de descritores da face.

```
def do_POST(self):
   ...
   with open("imageToSave.jpg", "wb") as fh:
      fh.write(base64.urlsafe_b64decode(test_data['image']))

   in_register = False

   unknown_image =
      face_recognition.load_image_file("imageToSave.jpg")
   unknown_encoding =
      face_recognition.face_encodings(unknown_image)[0]
```

Depois de obtidos os descritores de uma face a ser validada, deve-se comparar as faces armazenadas no servidor com a face a ser validada. As faces armazenadas no servidor são as faces cadastradas e podem se encontrar em um diretório com o nome *registered*, que contém arquivos ".jpg" com as faces que serão lidas pela biblioteca de reconhecimento facial. Para obter o nome de todos os arquivos com a extensão ".jpg", a biblioteca *glob* é utilizada.

O programa itera pelas faces cadastradas e obtém os descritores das faces cadastradas da mesma maneira que obtém os descritores da face a ser validada. Depois, a função para comparar dois conjuntos de descritores de face é chamada: *compare_faces()*. Se a função retornar *True*, a variável de registro *in_register* é atribuída como verdadeira. Após isso, uma mensagem HTTP é enviada com o código de resposta 200 OK, e um corpo que contém a palavra "Registered" caso a face esteja cadastrada ou "Not

Registered"caso a face não esteja cadastrada.

Procedimento para validação do cadastro de um rosto.

```
    def do_POST(self):
        ...
        registered_images = glob("registered/*jpg")

        for image in registered_images:
            known_image = face_recognition.load_image_file(image)
            known_encoding =
                face_recognition.face_encodings(known_image)[0]

            if face_recognition.compare_faces([known_encoding],
                unknown_encoding)[0]:
                in_register = True
                break


        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.end_headers()

        if in_register:
            self.wfile.write(bytes("Registered","utf-8"))
        else:
            self.wfile.write(bytes("Not Registered","utf-8"))
```

Finalmente, a classe é passada como parâmetro para o construtor do objeto que inicializa o servidor. O servidor agora consegue processar uma mensagem de método POST.

O endereço do servidor é mostrado no terminal e há a tentativa de incialização do servidor. Caso haja o aborto de incialização por uma interrupção do teclado, o servidor para a sua execução e é fechado.

Inicialização do servidor

```
    if __name__ == "__main__":
        webServer = HTTPServer((hostName, serverPort), MyServer)
        print("Server started http://%s:%s" % (hostName, serverPort))

        try:
            webServer.serve_forever()
        except KeyboardInterrupt:
            pass
```

```
        webServer.server_close()
        print("Server stopped.")
```

3.5 IMPLEMENTAÇÃO DO SISTEMA

O sistema agora está completo e pronto para a troca de informações. A maneira que a informação será utilizada é aberta a modo de deixar o sistema extensível. A informação de que um rosto está cadastrado pode ser utilizada para alguma outra comunicação como CAN, UART, I2C, entre outros, para ativar outros dispositivos conectados ao ESP32-CAM, ou simplesmente utilizar um GPIO *(General Purpose Input Output)* como sinal para a ativação de algum outro circuito responsável por uma catraca, portão ou qualquer outra maneira de acesso.

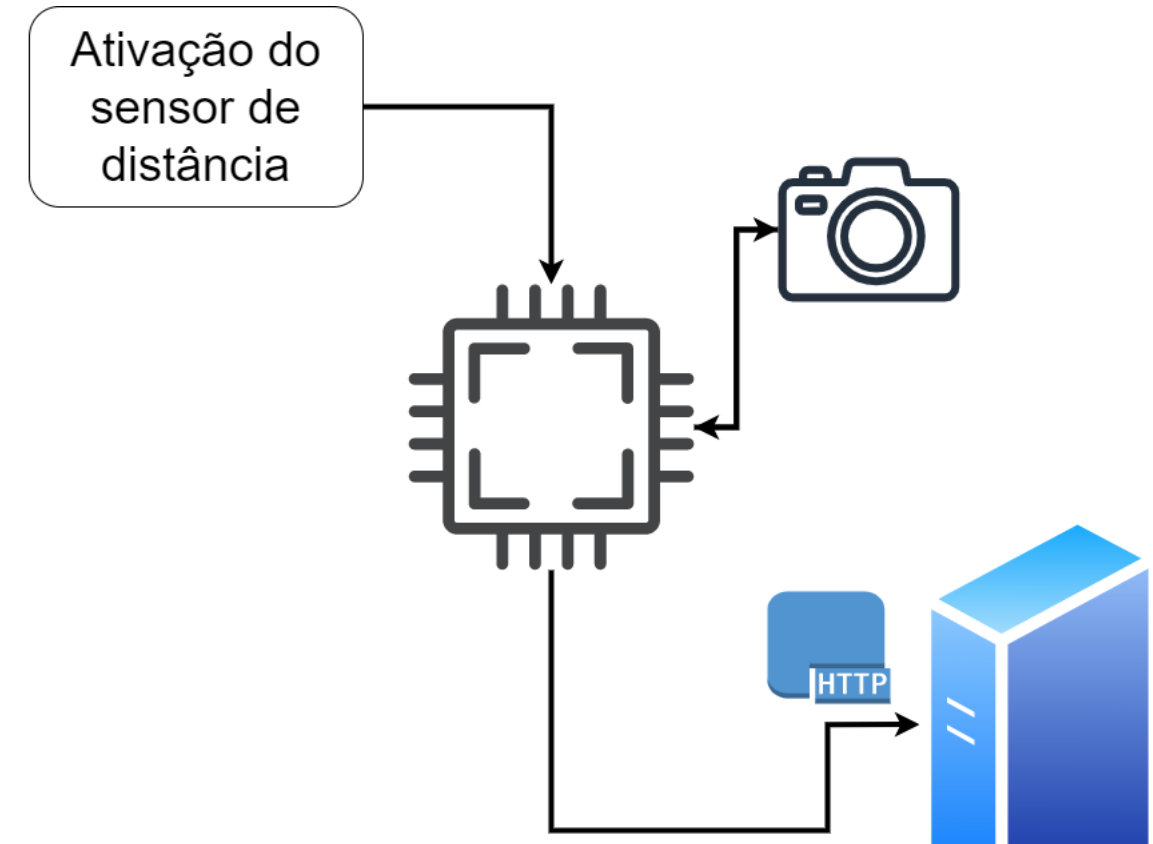
3.5.1 Implementação do cliente

O microcontrolador detecta uma presença a partir do sensor ultrassônico de distância, captura uma foto, codifica em uma mensagem HTTP e envia ao servidor (Figura 21).

3.5.2 Implementação do servidor

O servidor recebe a mensagem HTTP contendo um texto que representa a imagem, monta a imagem novamente e responde a solicitação de acordo com o estado de registro do rosto cadastrado (Figura 22).

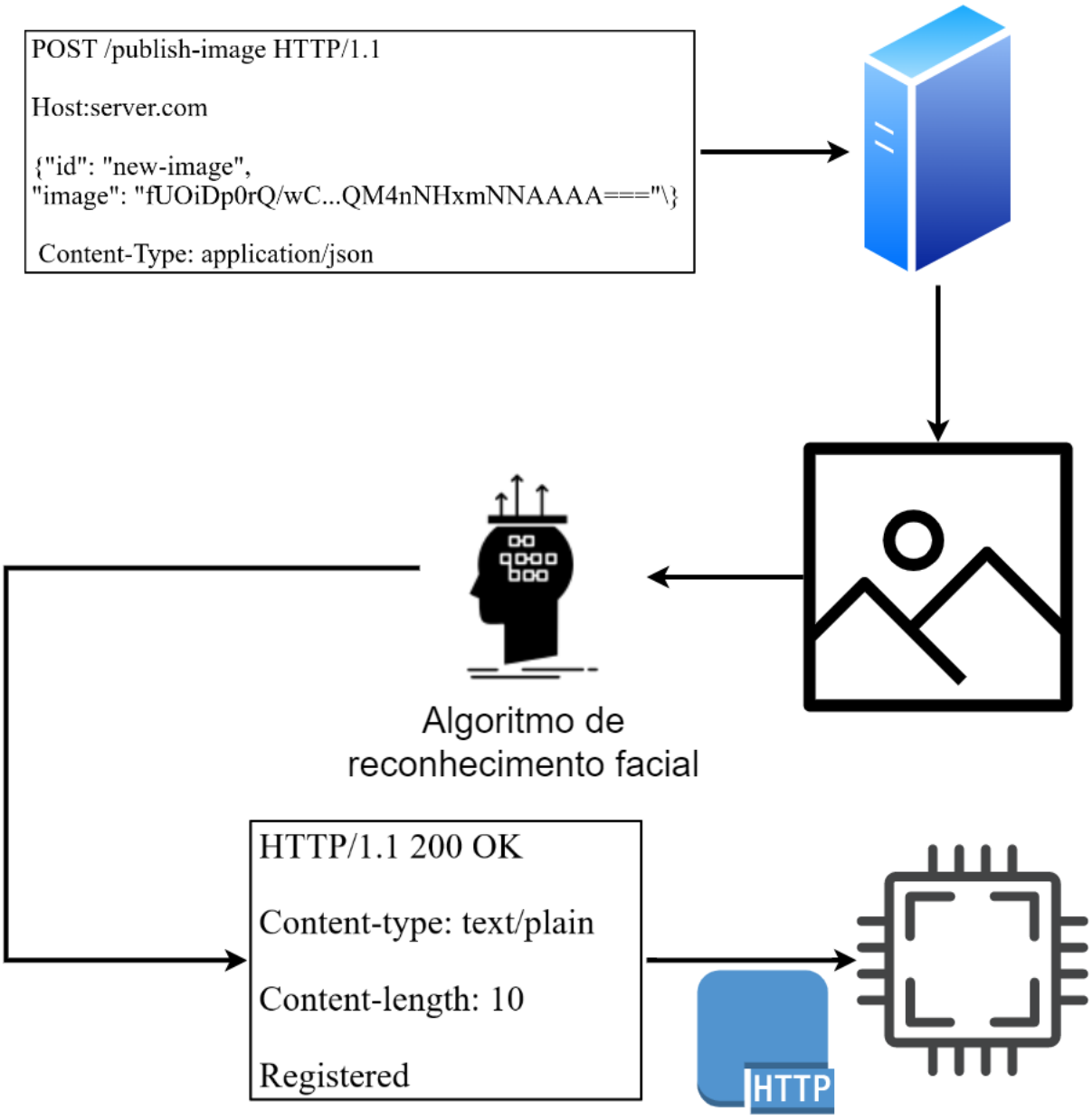
Figura 21 – Arquitetura geral do cliente




```
POST /publish-image HTTP/1.1

Host:server.com

{"id": "new-image",
"image": "fUOiDp0rQ/wC...QM4nNHxmNNAAAA==="\}

 Content-Type: application/json
```

fonte: Compilação do autor (2023).

Figura 22 – Arquitetura geral do servidor



fonte: Compilação do autor (2023).

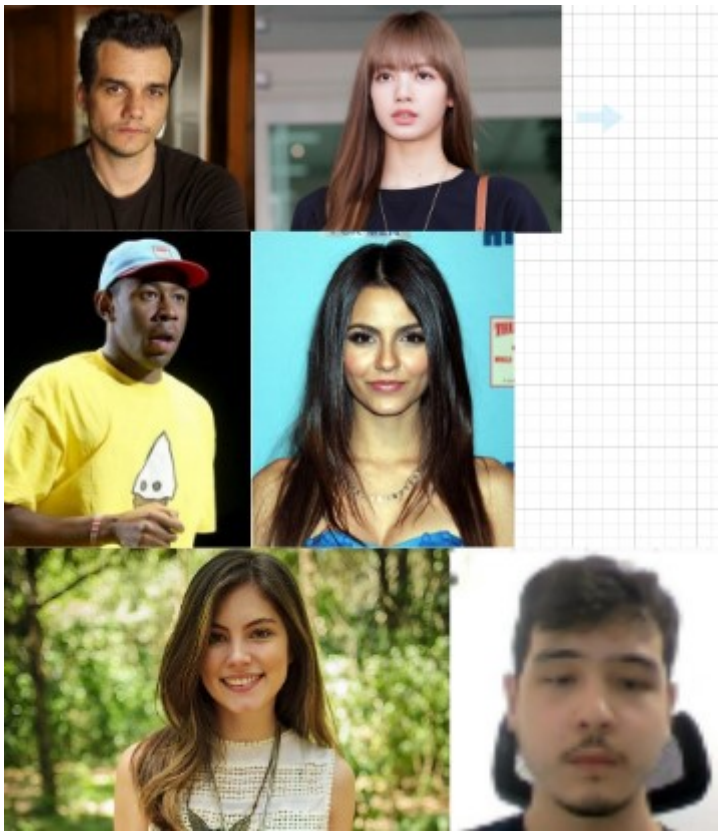
4 RESULTADOS

Foram executados testes com pessoas públicas e com fotos do autor (Figura 23) para validação do sistema. Todos os testes seguem o mesmo procedimento: enviar fotos diferentes por meio de requisições HTTP, sobrescrevendo o *buffer* de captura de foto para simular uma imagem capturada. Somente as fotos do próprio autor foram capturadas com a câmera do sistema.

4.1 TESTES COM PESSOAS PÚBLICAS E AUTOR

Foram cadastradas imagens de pessoas públicas e do próprio autor no servidor.

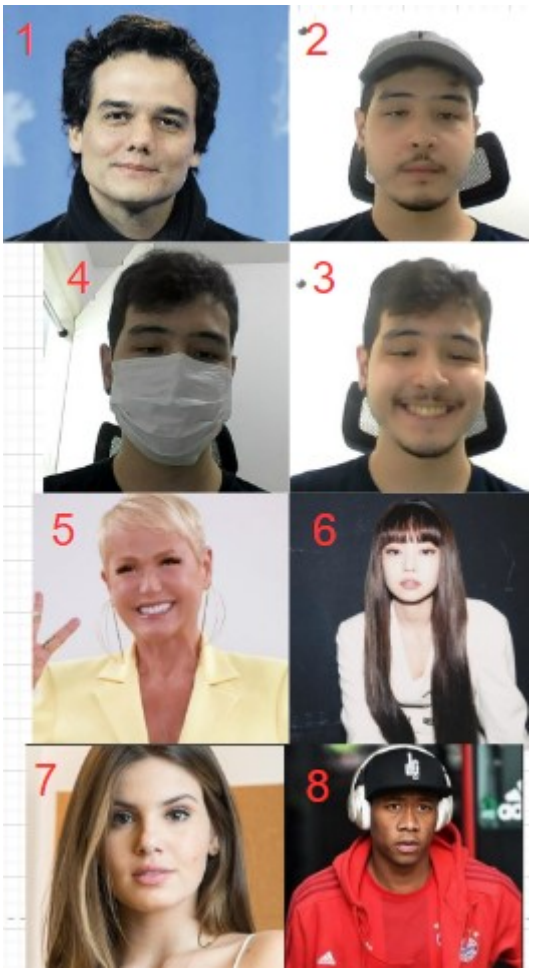
Figura 23 – Rostos cadastrados no servidor



fonte: Compilação do autor / Imagens de figuras públicas retiradas da internet (2023).

Foram enviadas algumas imagens de teste (Figura 24) para fins de validação, anotados os tempos de resposta do servidor e se a pessoa estava cadastrada no servidor. Os resultados revelaram pontos a serem estudados para futuros dos trabalhos.

Figura 24 – Imagens enviadas para validação



fonte: Compilação do autor (2023).

Tabela 4 – Resultados dos testes de rostos e tempos de resposta

ID	Corpo da resposta	Tempo de resposta (em segundos)
1	"Registered"	9.55
2	"Registered"	4.36
3	"Registered"	4.44
4	"Not Registered"	4.11
5	"Not Registered"	10.04
6	"Registered"	5.9
7	"Not Registered"	9.99
8	"Registered"	6.8

fonte: Compilação do autor (2023).

Os rostos enviados de numeração 1, 2, 3, 5 e 7 obedeceram o comportamento esperado do sistema. Os casos em que os sistema não respondeu de maneira correta podem ser analisados com mais profundidade. Os resultados com os tempos de resposta e a resposta do servidor estão contidos na Tabela 4.

Após o envio das imagens contidas na Figura 24 para o servidor, pode-se discutir os resultados compilados na Tabela 4.

A imagem 4 mostra o autor vestindo uma máscara, o que dificultou a detecção do rosto pelo algoritmo e causou um problema no reconhecimento. A imagem 6 mostra uma pessoa do sexo feminino com traços faciais asiáticos. Também há uma pessoa do sexo feminino com traços faciais asiáticos cadastrada no servidor, porém não são a mesma pessoa. A pessoa cadastrada no banco é uma cantora coreana chamada Lalisa Manobal e a foto da pessoa enviada para validação é uma cantora coreana chamada Kim Jennie. A semelhança entre as duas é notável, entretanto um modelo deveria conseguir capturar os descritores que fariam a distinção entre elas. Isso levanta uma questão sobre o modelo escolhido: "A identificação de pessoas semelhantes sempre falha?"

A imagem 7 mostra uma pessoa loira, branca do sexo feminino que foi identificada como não cadastrada no servidor. Isso contradiz a informação anterior, pois há uma pessoa loira, branca do sexo feminino extremamente semelhante com a pessoa da imagem 7 cadastrada no servidor e o modelo se mostrou falho em caso de pessoas extremamente semelhantes. A pessoa cadastrada é a atriz brasileira Bruna Hamu, enquanto a pessoa enviada para ser validada é a atriz brasileira Camila Queiroz.

O último caso (imagem 8) pode fornecer mais algumas informações sobre essa inconsistência do modelo. A imagem 8 mostra o jogador de futebol David Alaba, uma pessoa negra do sexo masculino que foi identificada como cadastrada no servidor. No servidor, o *rapper* Tyler Gregory Okonma está cadastrado e possui uma grande semelhança com David Alaba, isso levou o modelo a classificar o *rapper* e o jogador de futebol como a mesma pessoa.

Em suma, uma pessoa branca foi classificada corretamente mesmo que haja uma outra pessoa de semelhança extremamente aparente, enquanto uma pessoa asiática e uma pessoa negra não foram classificadas de maneira correta. O algoritmo apresenta inconsistências na classificação dependendo da etnia da pessoa a ser identificada.

4.1.1 Viés algorítmico

O algoritmo performa melhor para pessoas brancas do que para outras etnias. Esse fenômeno não é inédito na literatura e algumas ferramentas para aplicações de leis e reconhecimento de indivíduos classificam incorretamente pessoas negras. A razão por trás dessas classificações errôneas é resultado da propagação de viéses sociais em modelos de inteligência artificial.

> Alguns estudos apontam que tecnologias aparentemente neutras utilizadas como instrumentos de aplicação da lei, a exemplo de testes de drogas, leitores automáticos de placas de veículos, não foram criados a partir de uma posição de neutralidade e tal circunstância tem afetado de modo desproporcional pessoas negras (PORTO; ROLIM, , p.1)

4.1.2 Possíveis explicações dos viéses

Um algoritmo de reconhecimento facial é simplesmente um conjunto de operações matemáticas com parâmetros ajustados a fim de melhor se comportar em direção à um resultado já observado, permitindo a predição desse resultado. No estudo de reconhecimento facial, o algoritmo é treinado com imagens de rostos para determinar os descritores de uma face. Para essa tarefa é necessário um

grande volume de imagens para que o modelo consiga inferir os padrões e detectar semelhanças entre faces. Esse treino do algoritmo deve englobar diferentes tipos de rosto para que não haja uma injustiça na identificação, ou seja, o modelo precisa de rostos diferentes para detectar descritores diferentes.

Isso é uma possível causa de viés: uma rede neural de reconhecimento facial treinada somente com rostos de pessoas brancas será muito boa em identificar diferença entre pessoas brancas, enquanto outras etnias serão classificadas erroneamente. O pacote de reconhecimento facial desse sistema de reconhecimento facial provavelmente apresenta viés dessa natureza, um desbalanço na precisão de classificação de acordo com a etnia.

5 CUSTOS DO SISTEMA

Abaixo segue a Tabela 5 com os custos do hardware do sistema, o custo total foi de R$95,87. Sistemas comerciais de controle de acesso por reconhecimento facial, combinado com biometria chegam, a custar uma média de R$1800,00 (CONTROLE. . . , 2023. Disponível em: https://produto.mercadolivre.com.br/MLB-2759858122-controle-de-acesso-ds-k1t671mfl-facial-biometria-hikvision-_JM. Acesso em 17 Jan. 2023). Esses sistemas tem funcionalidades mais complexas, como *displays* de LED e tecnologia de biometria. Entretanto o sistema proposto consegue atingir o objetivo com um custo bem reduzido.

Tabela 5 – Tabelas de custo de hardware do sistema

Dispositivo	Custo (em reais)
Módulo esp32-cam com camera ov2640 2mp	49,90
OEM Sensor Ultrassônico Hc-sr04	11,99
Fonte Alimentação 5V 2A Bivolt	14,00
Cabo Jumper 20 cm Fêmea x Fêmea 40 unidades	19,98
Total	95,87

fonte: Compilação do autor (2023).

6 CONCLUSÃO

O sistema proposto funcionou com tempos de resposta que não ultrapassaram os 11 segundos, permitindo uma identificação rápida que traz agilidade e conforto para qualquer tipo de acesso em que o sistema será utilizado. A codificação por base 64 se mostrou válida e prática para a transmissão de imagens por meio de mensagens HTTP, apresentou boa performance na compressão e descompressão de imagens com uma redução baixa de qualidade que permitiu a operação correta do algoritmo de reconhecimento facial.

O servidor construído em Python foi de fácil e intuitiva implementação. O pacote Python de reconhecimento facial escolhido cumpriu parcialmente seu objetivo. Os resultados errôneos ou problemáticos provém de problemas como obstrução de face e viés algorítmico introduzido no modelo de reconhecimento facial por meio dos criadores do modelo.

O sistema se mostrou agnóstico ao tipo de controle de acesso, ele cumpre a tarefa de reconhecimento facial sem limitar as aplicações, ou seja, o sistema pode atuar em diferentes tipos de acesso, devido à capacidade de extensão das funcionalidades da placa controladora do sistema. Isso amplia as possíveis aplicações do sistema somente pela forma de arquitetura proposta. O sistema pode ser utilizado para ativar portões, portas, catracas, garagens, etc.

6.1 POSSIBILIDADES DE TRABALHOS FUTUROS

Em trabalhos futuros, o ponto principal para revisita é a questão do enviesamento do modelo utilizado. Estudos mais robustos sobre o viés do modelo, maneiras de contornar vieses ou até mesmo vieses direcionados seriam de fundamental complementação para aprimoramento do sistema. Além disso, a construção de um modelo de reconhecimento facial novo a partir de uma nova arquitetura de rede neural pode ser uma alternativa para o problema de enviesamento.

Uma estrutura de mensagem mais robusta também é interessante e benéfica para aprimoramento do modelo, podendo propor registro por meio da própria câmera do sistema, identificação de obstrutores de rosto e até mesmo alguma integração por meio de IOT para relatórios de acesso do sistema. Os relatórios podem conter data de entrada em um local em que o sistema foi implantado, tempo de estadia, hora de entrada, informações sobre rosto de indivíduos que utilizaram o sistema, etc.

REFERÊNCIAS

ADAMVR. Base64. **GitHub repository**, adamvr, 2013, Disponível em: https://github.com/adamvr/arduino–base64. Acesso em: 18 Jan. 2023.

AGEITGEY. Face recognition. **GitHub repository**, GitHub, 2022, Disponível em: https://github.com/ageitgey/face_recognition. Acesso em: 19 Jan. 2023.

ARDUINO. Arduino framework. Arduino, 2023, Disponível em: https://www.arduino.cc/. Acesso em 17 Jan. 2023.

BERNERS-LEE, e. a. Hypertext transfer protocol. RFC Editor, n. 1945, p. 1–60, May 1996. ISSN 2070-1721.

CONTROLE De Acesso Ds-k1t671mfl Facial Biometria Hikvision. 2023. Disponível em: https://produto.mercadolivre.com.br/MLB–2759858122–controle–de–acesso–ds–k1t671mfl–facial–biometria–hikvision–_JM. Acesso em 17 Jan. 2023.

ESPRESSIF. arduino-esp32-wifi. **GitHub repository**, GitHub, 2022.

ESPRESSIF. Esp-idf api reference. Espressif Systems, 2022.

ESPRESSIF. arduino-esp32. **GitHub repository**, GitHub, 2022, Disponível em: https://github.com/espressif/arduino–esp32. Acesso em: 19 Jan. 2023.

GRIDLING, G.; WEISS, B. Introduction to microcontrollers. **Vienna University of Technology Institute of Computer Engineering Embedded Computing Systems Group**, 2007.

JOSEFSSON, S. The base16, base32, and base64 data encodings. 2006.

KOMMU, A.; KANCHI, R. R. Arm based temperature measurement and processing to remote computer using fiber optic cable. **International Conference on Communication and Signal Processing, 2013**, Coimbatore, p. 423–427, 2013.

PASZKE, A. et al. Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 32, p. 8024–8035, 2019.

PATTNAIK, P.; MOHANTY, K. K. Ai-based techniques for real-time face recognition-based attendance system- a comparative study. **2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA)**, Coimbatore, p. 1034–1039, 2020.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, Boston, v. 12, p. 2825–2830, 2011.

PORTO, V. B.; ROLIM, E. K. C. O reconhecimento facial e o viés algorítmico racista facial recognition and racist algorithmic bias. **Brazilian Journal of Development**, Curitiba, 2022.

SMITH, J. **Programming the PIC Microcontroller with MBASIC**. [S.l.]: Elsevier, 2005.

STANFORD. Neural networks history. cs.stanford.edu, 2019.

TANEJA, S.; GUPTA, P. R. Python as a tool for web server application development. **Int. J. Information, Commun. Comput. Technol**, Pittsburgh, v. 2, n. 1, p. 2347–7202, 2014.

WANG, S.-C. Artificial neural network. Springer US, 2003.

ANEXO A – LINK PARA OS CÓDIGOS

Repositório do servidor: https://github.com/hiroci/tcc-server
Repositório do microcontrolador: https://github.com/hiroci/tcc