Tracking de objetos utilizando ESP32-CAM, FLASK, Opencv

Marco Antônio da Silva Santos *
Davi Lima Felix dos Santos †
Helen da Silva Bispo ‡

São Cristovão
2025

Disciplina: Sistemas Embarcados.
Professor: RODOLFO BOTTO DE BARROS GARCIA.

Objetivo da matéria:
A matéria de sistemas embarcados, para projetar, desenvolver e implementar sistemas computacionais dedicados a aplicações especficas, geralmente integrados em dispositivos eletrônicos maiores.

Projeto do grupo:
Nosso grupo projetou uma forma de inspecionar e ver os objetos que estão presentes no campo de visão dela, recebendo as informações em um servidor web local

*ma2022@academico.ufs.br - Engenharia de Computação
†srfelix@academico.ufs.br - Engenharia de Computação
‡helen.bispo@dcomp.ufs.br - Ciência da Computação

1.0 Introdução

Este projeto visa desenvolver um sistema utilizando:

- **ESP32-CAM**: Para captura de imagens.
- **YOLOv8**: Para detecção de objetos em tempo real.
- **Flask**: Como servidor web para exibição dos resultados.

1.1Especificações do Hardware

- **ESP32-CAM:** Módulo com microcontrolador ESP32 integrado à câmera OV2640 (resolução de 2MP).
- **Sensor OV2640:** Câmera com resolução de 1600 x 1200, com suporte a saída JPEG.
- **Consumo:** 2A no máximo.
- **Fonte de energia:** 3.3 or 5 vdc.
- **Wi-Fi:** IEEE 802.11 b/g/n para transmissão dos frames.
- **Outputs:** 3.3 vdc.
- **GPIO pins:** UART, SPI e I2C.

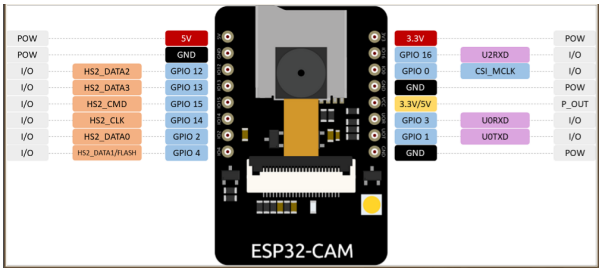

Figura 1: Diagrama de pinos do ESP32-CAM. Fonte: `https://www.nitrathor.com/data-sheets/esp32-cam`.

1.2 Introdução a YOLOv8

O Yolo(You only look once) é um sistema de detecção de objetos de ultima geração processando imagens a 30 FPS, revolucionou a detecção de objetos ao substituir os métodos tradicionais (como *sliding windows* e R-CNN) por uma abordagem unificada baseada em redes neurais [**?**]. Seu funcionamento básico inclui:

- **Caixas delimitadoras**: são chamadas de bounding boxes, utilizadas para demarcar o possível objeto.
- **Mapa de classes de probabilidades**: probabilidades do que tem na tela (ex: cachorro, bike, ...).
- **Confiança**: confiança da detecção.

O algoritmo divide a imagem em uma grade (ex: 13×13 células), onde cada célula faz previsões independentes.

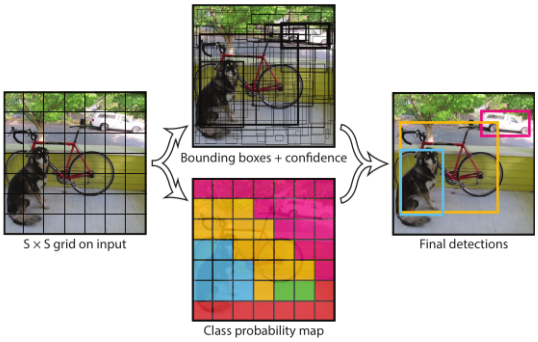

Figura 2: Representação do Modelo, Fonte: `https://arxiv.org/abs/1712.03316`.

1.3 Introdução Flask

Flask é um *microframework* web em Python projetado para ser **leve, flexível e fácil de usar**, ideal para desenvolver aplicações web e APIs RESTful. Suas principais características incluem:

- **Roteamento**: Roteamento de URLs com decoradores (ex: `@app.route`) e sistema de templates.
- **Leveza**: Core mínimo (500 linhas de código) sem dependências desnecessárias.
- **Extensibilidade**: Suporte a extensões para banco de dados (SQLAlchemy), autenticação (Flask-Login), etc.
- **Ideal para APIs**: Comunicação eficiente com dispositivos IoT (como o ESP32-CAM neste projeto).

Funcionamento no sistema proposto:

1. ESP32-CAM envia imagens via HTTP POST
2. Flask recebe e processa as imagens
3. Retorna resultados para visualização web

Documentação oficial: `https://flask.palletsprojects.com`

2.0 Conclusão

Este projeto demonstrou a viabilidade de implementar um sistema completo de visão computacional embarcada, integrando:

- **ESP32-CAM** como dispositivo de captura, configurado para transmitir video via Wi-Fi em `http://[ip-do-esp]/stream`
- **OpenCv** para processamento da stream em tempo real.
- **Yolov8** para detecção de objetos com precisão.
- **Flask** como servidor para gerenciar requisições web e funcionalidades adicionais:
  - Ligar/Desligar camera.
  - Tracking Objetos.
  - Gravação de video.
  - Captura de telas.

A arquitetura proposta mostrou-se eficiente para aplicações de monitoramento inteligente, combinando baixo custo hardware com técnicas modernas de visão computacional[1].
Próximos passos incluem:

1. Otimização do consumo energético do ESP32-CAM
2. Implementação de múltiplos clientes simultâneos
3. Adição de reconhecimento facial

Referências

[1] J. Redmon et al., "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779-788.

[2] NitThor, *ESP32-CAM Datasheet*, 2023. [Online]. Disponível em: `https://www.nitrathor.com/data-sheets/esp32-cam`

[3] J. Redmon, *YOLO: Real-Time Object Detection*, 2017. [Online]. Disponível em: `https://arxiv.org/abs/1712.03316`

[4] Pallets Team, *Flask Documentation*, 2023. [Online]. Disponível em: `https://flask.palletsprojects.com`

[1] Os desafios enfrentados durante o desenvolvimento contribuíram significativamente para nosso aprendizado em sistemas embarcados e processamento de imagens.

Apêndices

[

Esquema do projeto

Digrama da implementação do código..

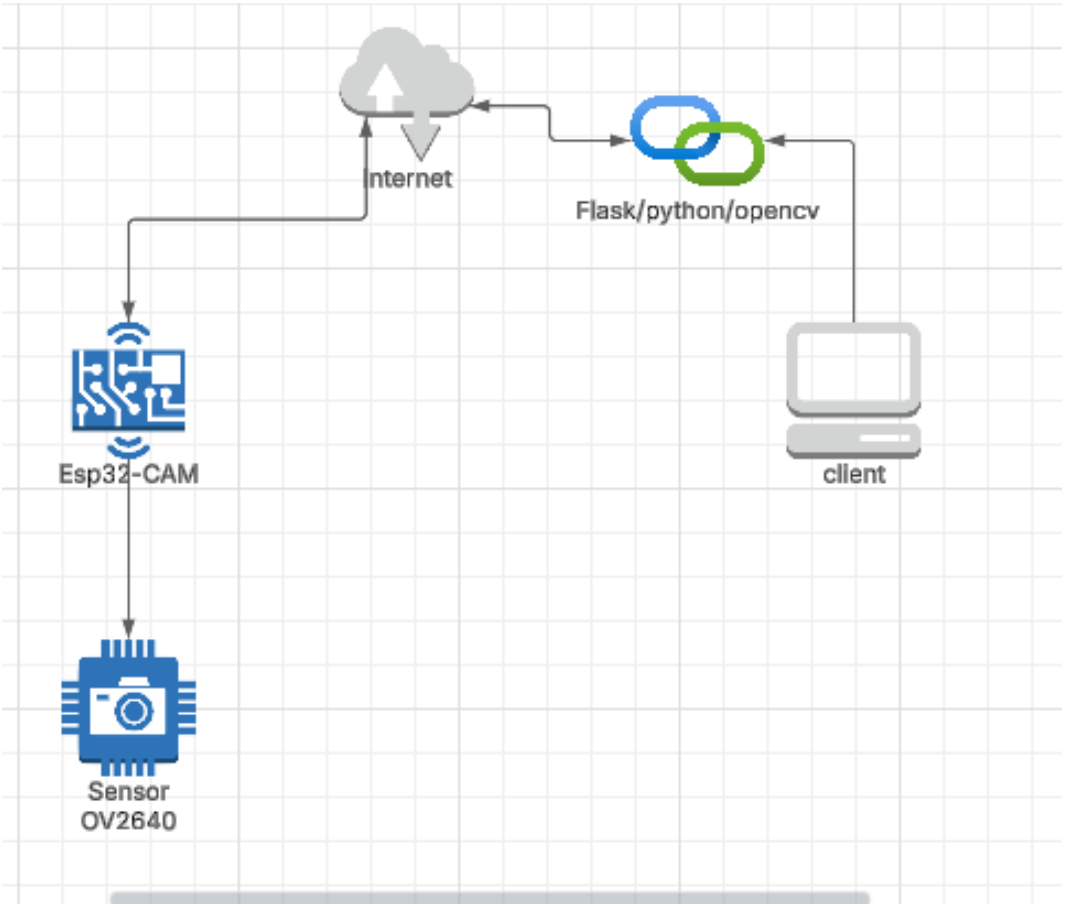

Figura 3: Fluxo de dados do sistema.

Resultados

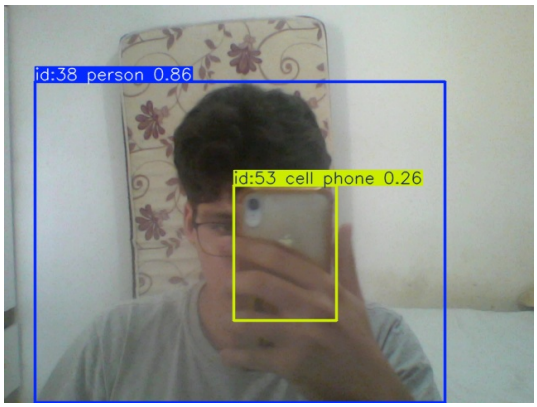

Figura 4: capture_26_03_25_150032.jpg

Código-fonte

Firmware do ESP32-CAM

Função: Configuração de pinos

```
// Copyright 2020-2021 Espressif Systems
static esp_err_t init_camera(uint32_t xclk_freq_hz, pixformat_t pixel_format, framesize_t frame_
{
    camera_config_t camera_config = {
        .pin_pwdn = CAMERA_PIN_PWDN,
        .pin_reset = CAMERA_PIN_RESET,
        .pin_xclk = CAMERA_PIN_XCLK,
        .pin_sscb_sda = CAMERA_PIN_SIOD,
        .pin_sscb_scl = CAMERA_PIN_SIOC,

        .pin_d7 = CAMERA_PIN_D7,
        .pin_d6 = CAMERA_PIN_D6,
        .pin_d5 = CAMERA_PIN_D5,
        .pin_d4 = CAMERA_PIN_D4,
        .pin_d3 = CAMERA_PIN_D3,
        .pin_d2 = CAMERA_PIN_D2,
        .pin_d1 = CAMERA_PIN_D1,
        .pin_d0 = CAMERA_PIN_D0,
        .pin_vsync = CAMERA_PIN_VSYNC,
        .pin_href = CAMERA_PIN_HREF,
        .pin_pclk = CAMERA_PIN_PCLK,

        //EXPERIMENTAL: Set to 16MHz on ESP32-S2 or ESP32-S3 to enable EDMA mode
        .xclk_freq_hz = xclk_freq_hz,
        .ledc_timer = LEDC_TIMER_0, // // This is only valid on ESP32/ESP32-S2. ESP32-S3 use LCD
        .ledc_channel = LEDC_CHANNEL_0,

        .pixel_format = pixel_format, //YUV422,GRAYSCALE,RGB565,JPEG
        .frame_size = frame_size,    //QQVGA-UXGA, sizes above QVGA are not been recommended whe

        .jpeg_quality = 10, //0-63
        .fb_count = fb_count,       // For ESP32/ESP32-S2, if more than one, i2s runs in continu
        .grab_mode = CAMERA_GRAB_LATEST,
        .fb_location = CAMERA_FB_IN_PSRAM
    };

    //initialize the camera
    esp_err_t ret = esp_camera_init(&camera_config);

    sensor_t *s = esp_camera_sensor_get();
    s->set_vflip(s, 1);//flip it back
    //initial sensors are flipped vertically and colors are a bit saturated
    if (s->id.PID == OV3660_PID) {
        s->set_saturation(s, -2);//lower the saturation
    }

    if (s->id.PID == OV3660_PID || s->id.PID == OV2640_PID) {
        s->set_vflip(s, 1); //flip it back
    } else if (s->id.PID == GC0308_PID) {
        s->set_hmirror(s, 0);
    } else if (s->id.PID == GC032A_PID) {
```

```
        s->set_vflip(s, 1);
    }

    camera_sensor_info_t *s_info = esp_camera_sensor_get_info(&(s->id));

    if (ESP_OK == ret && PIXFORMAT_JPEG == pixel_format && s_info->support_jpeg == true) {
        auto_jpeg_support = true;
    }

    return ret;
}
```

Fluxo de Trabalho

1. Inicializa conexão Wi-Fi (`app_wifi_main()`)
2. Configura hardware da câmera (`init_camera()`)
3. Inicia servidor de streaming (`start_stream_server()`)
4. Captura frames em loop e envia para fila RTOS

Python Stream config

Criação da classe camera

```
    import cv2 as cv
from ultralytics import YOLO
import os
from datetime import datetime
""" Classe da camera que utiliza o modelo de ia YOLOV8n.pt
    @param capture_cam -> a camera que vai ser usad
"""
class Camera:
    def __init__(self, capture_cam="http:/[ip-esp32-cam]/stream") -> None:
        self.captura = capture_cam
        self.streaming = cv.VideoCapture(self.captura)
        if not self.streaming.isOpened():
            print(f"Erro: Não foi possível abrir a câmera {self.captura}")
            return

        self.camRunning = True
        self.modelo = YOLO('yolov8n.pt')
        self.frame = None
        self.isRecording = False
        self.videoWriter = None
```

- **captura** index da camera.
- **stream** streaming da camera.

Metodos da camera:

1. **record_frame(save_dir)**: manda imagem da câmera para o site, com yolo executando
2. **parar_gravacao()**: para a trasmissão da câmera.
3. **capture_frame(save_dir)**: printa um frame da câmera.
4. **generate_frame()**: manda as imgs em forma de byte para web.
5. **stop()**: fecha tudo da câmera.

Configuração de rotas com o Flask

configuração da rota inicial

```
numero_da_camera = "http:/[ip-ESP32-CAM]/stream"
app = Flask(__name__)
cam = None


""" Rota inicial do site '/'
  renderiza a pagina
"""
@app.route('/')
def index():
  return render_template('index.html')
```

Mostrando câmera na pagina:

```
""" rota /iniciar_tracking
  usado para o butão que se n tiver camera, cria uma camera.
  retorna o stream gravado pela camera
"""
@app.route('/iniciar_tracking')
def iniciar_tracking():
  global cam
  if cam is None:
    cam = Camera(numero_da_camera)
  return Response(cam.generate_frame(), mimetype='multipart/x-mixed-replace; boundary=frame')

""" rota /parar_tracking
  desabilita a camera e retorna a pagina inicial
"""
@app.route('/parar_tracking')
def parar_tracking():
  global cam
  if cam is not None:
    cam.stop()
    cam = None
  return render_template('index.html')
```