

Fernando Trentin e
Nicolas Fajardo

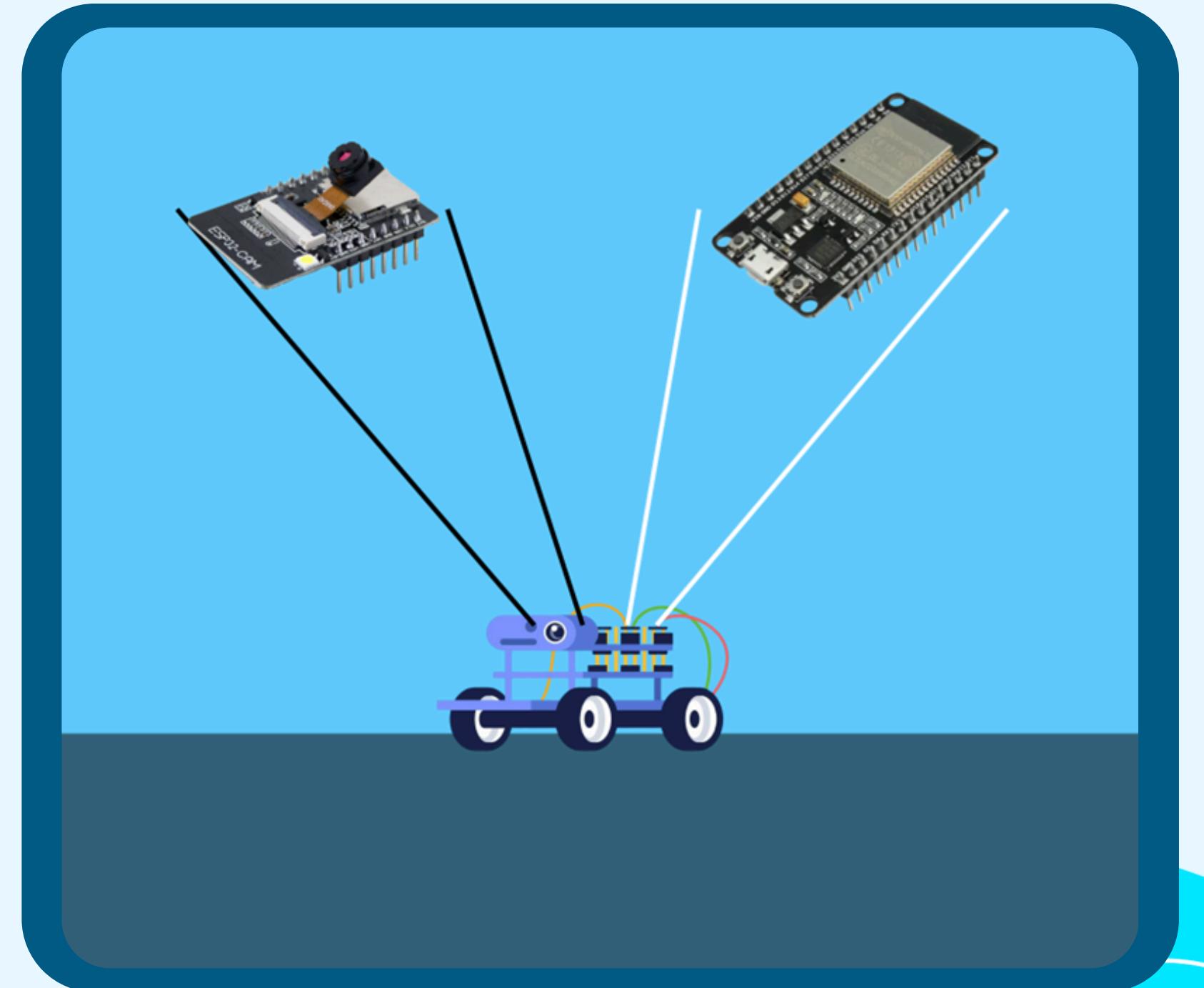
Sistema de Rastreamento e Movimentação Autônoma com Visão Computacional e Processamento Remoto

PROJETO DE SISTEMAS
EMBARCADOS

INTRODUÇÃO

O que é o projeto?

- Robô móvel capaz de detectar um alvo,
- calcular posição 3D (X, Y, Z) usando visão estéreo,
- e se mover automaticamente para mantê-lo centralizado e a uma distância fixa.
- Utiliza três ESP32-CAMs + ESP32 para controle + PC para processamento.



Porque fazer isso?

- ESP32-CAM é barata, mas limitada para visão.
- Computar visão em tempo real embarcado é difícil.
- Perdas de pacote, latência e instabilidade atrapalham robôs móveis.
- Poucos projetos documentam arquitetura completa (multi-câmeras, visão estéreo, TCP, controle real).
- Sistemas como drones e câmeras de segurança utilizam do rastreamento e movimentação.

Desafio central:

- Criar um sistema embarcado baixa potência + alta complexidade de visão, usando processamento remoto.

Diferenciais:

- Suporte a três câmeras simultâneas.
- Processamento estéreo real com profundidade (Eixo Z).
- Envios e respostas em tempo real entre PC e ESP32.
- Uso de controle autônomo real: motores + servo.
- Documentação estruturada — serve como projeto de portfólio.

PROJETO

Requisitos Funcionais: Sistema Embarcado

RF01	O sistema deve ajustar a câmera para manter um objeto centralizado e na mesma distância.
RF02	O sistema deve realizar os cálculos de controle com os dados recebidos.
RF03	O sistema deve utilizar os valores de controle para deixar o objeto centralizado
RF04	O sistema deve utilizar os valores de controle para manter sempre uma distância do objeto.
RF05	O sistema deve parar caso a conexão seja perdida
RF06	O sistema deverá realizar transmissão remota de imagens.
RF07	O sistema deve estar montado em uma estrutura única
RF08	O sistema deve ter alimentação própria

Requisitos Funcionais: Leitura da câmera

RF01	O programa deve receber as imagens enviadas remotamente.
RF02	O programa deve detectar o objeto a ser reconhecido
RF03	O programa deve calcular as posições cartesianas do objeto
RF04	O programa deve mostrar as posições calculadas.
RF05	O programa deve enviar uma mensagem caso o alvo não seja encontrado
RF06	O programa deve realizar a transmissão das posições calculados
RF07	O programa deve receber as imagens enviadas remotamente
RF08	O programa deve exibir as imagens recebidas

Requisitos Não Funcionais: Sistema Embarcado

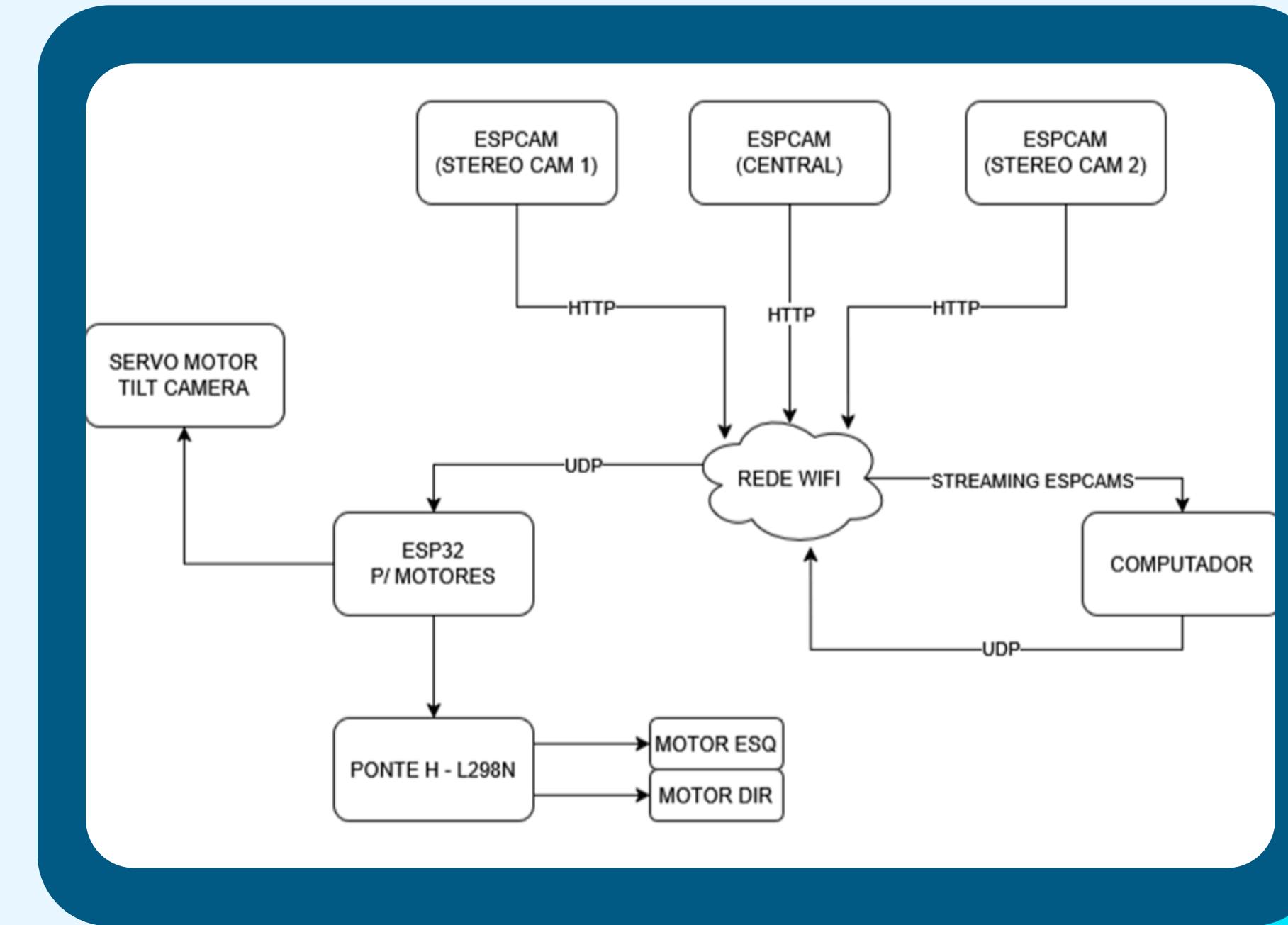
RNF01	O sistema deve ser implementado em linguagem C
RNF02	O sistema deve ser implementado em ESP32 DEVKIT1
RNF03	O sistema deverá utilizar ESPCAMs para capturar as imagens
RNF04	O sistema deve estar em uma estrutura impressa em 3D
RNF05	O recebimento das posições deve ser via Wi-Fi e utilizar o protocolo TCP
RNF06	O sistema deve utilizar protocolo HTTP para streaming de imagens
RNF07	O sistema deve utilizar motores DC para movimentação
RNF08	O sistema deve utilizar servo motor para controle do ângulo das câmeras
RNF09	Os motores devem ser controlados por uma Ponte-H
RNF10	O sistema deve ser alimentado por duas baterias LiPO 18650 3.7v

Requisitos Não Funcionais: Leitura da câmera

RNF01	O sistema deve ser implementado em Python utilizando Socket, Threads e OpenCV.
RNF02	O processamento de detecção não deve exceder o tempo estabelecido pela captura dos frames.
RNF03	A comunicação do sistema por Wi-Fi terá protocolo TCP
RNF04	O sistema deve utilizar processamento estereoscópico para identificar a distância do objeto para o sistema embarcado.

Arquitetura Geral do Sistema

- ESP32-CAMs
 - Captura e envia vídeo para o PC (HTTP).
- Computador (Python + OpenCV)
 - Processamento de visão estéreo, detecção e cálculo 3D.
- ESP32 Controle
 - Recebe posições e ajusta motores + servo.



Arquitetura de Hardware

- Cada ESP32-CAM transmite vídeo individualmente.
- O PC recebe os 3 fluxos simultâneos.
- O ESP32 recebe comandos via TCP.
- Motores e servo são acionados a partir desse módulo.

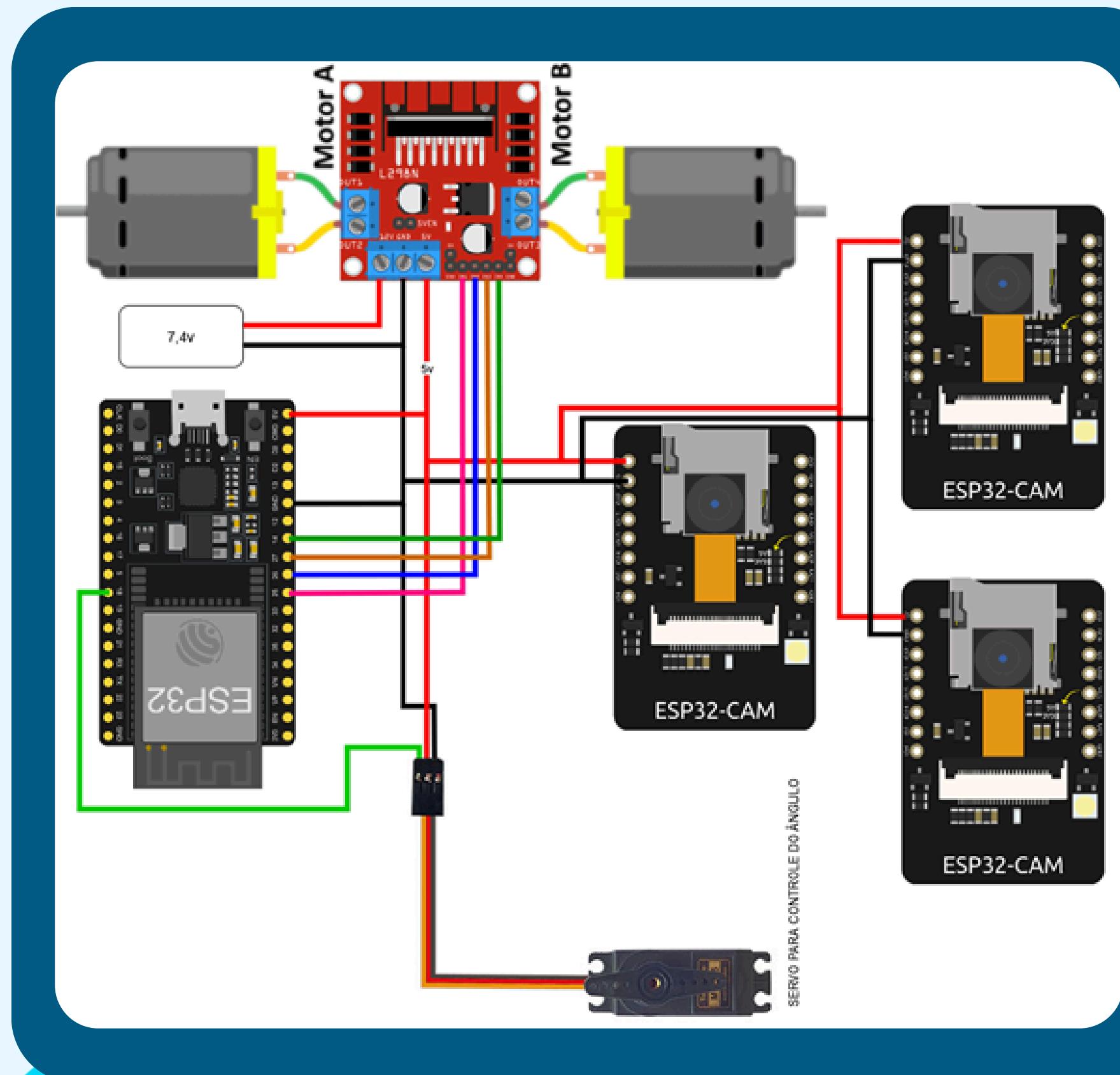


Diagrama de Sequência

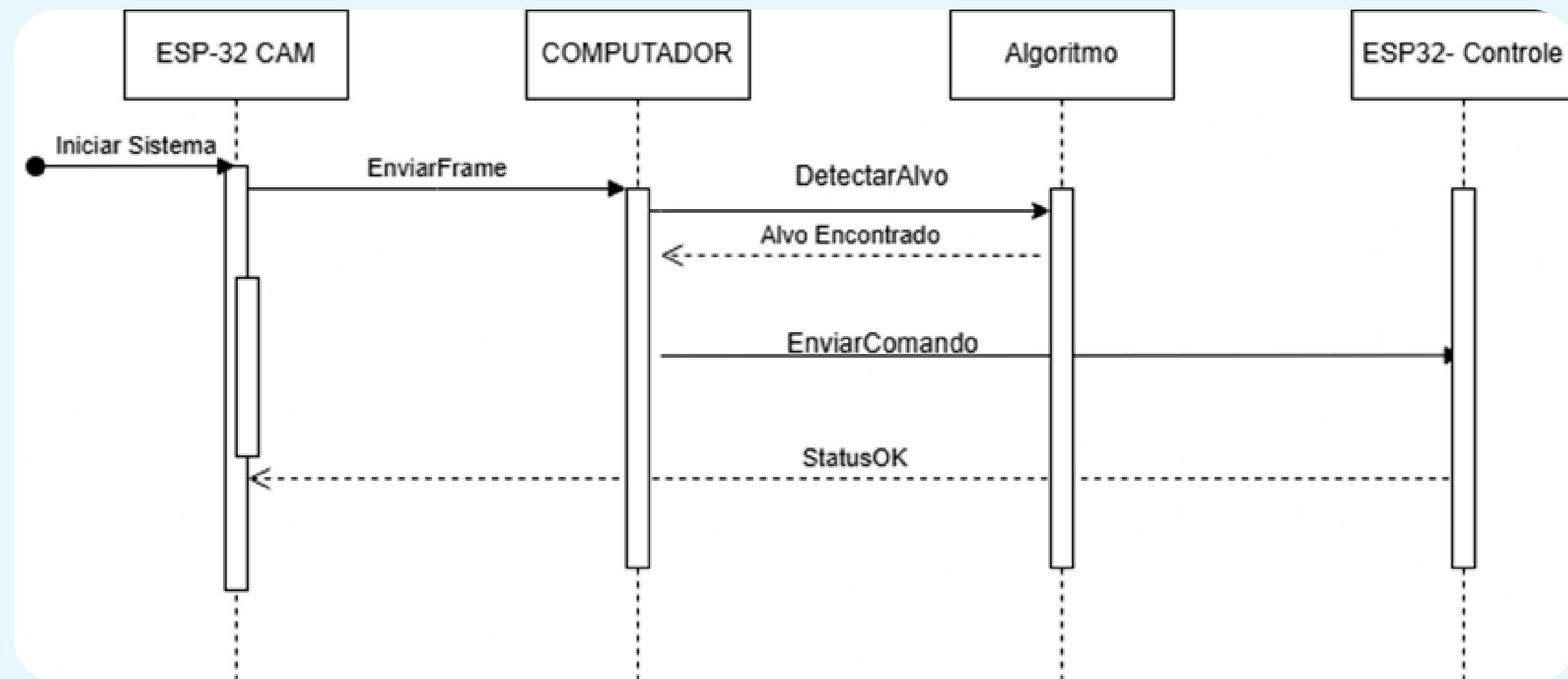
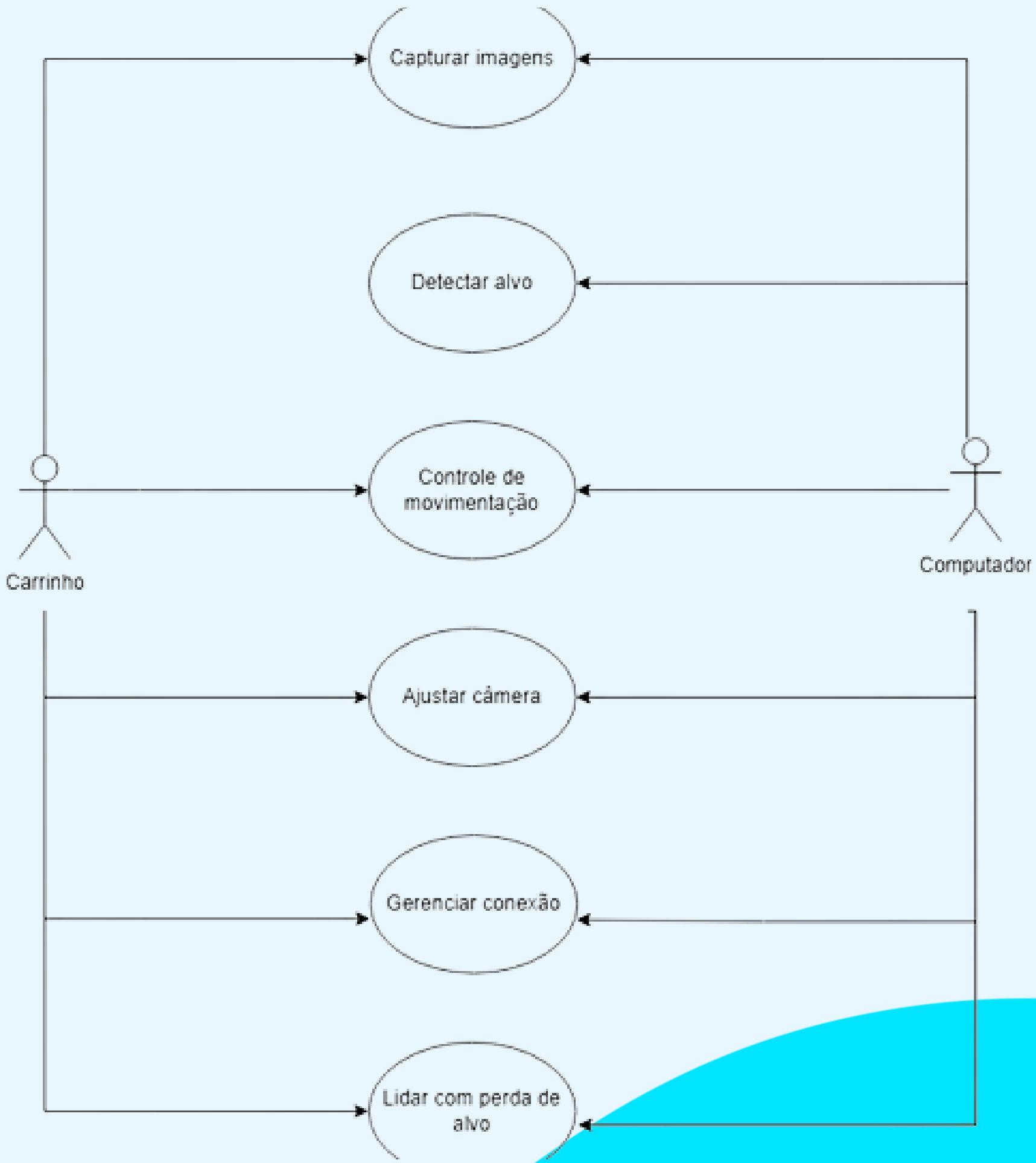
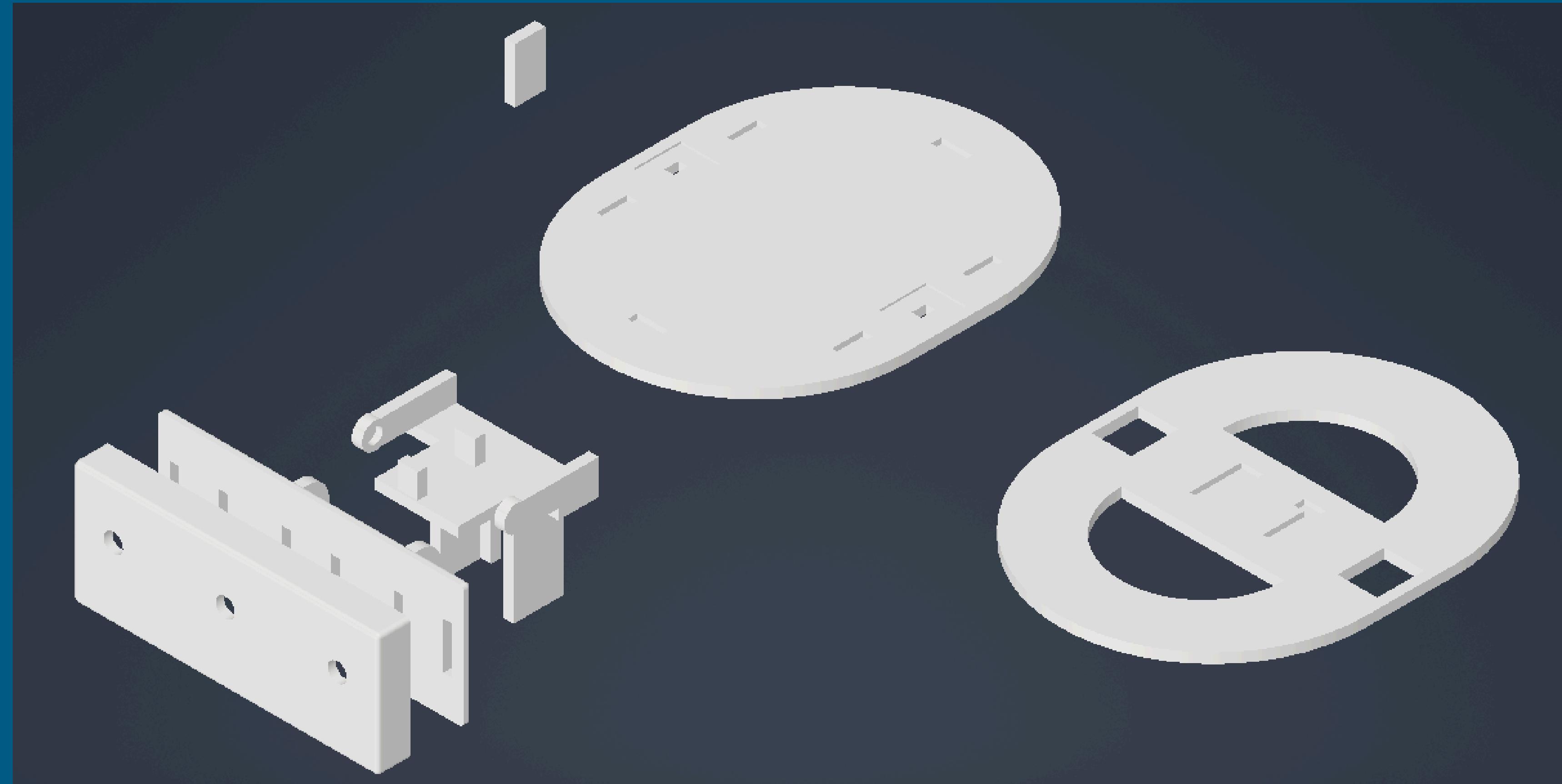


Diagrama de Casos de Uso



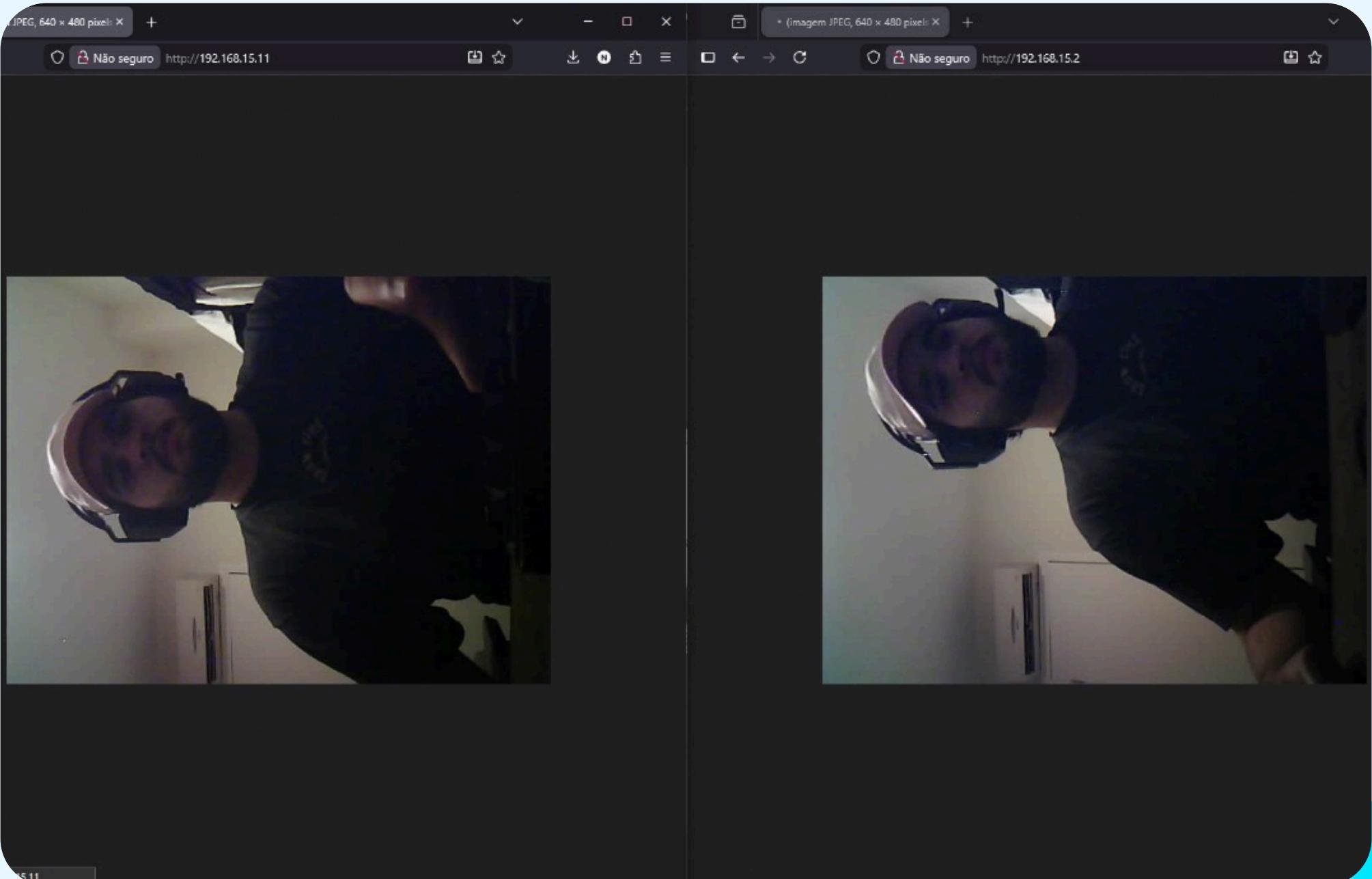
Modelagem 3D



IMPLEMENTAÇÃO

Etapa 1 - ESPCAMs

- Configuração das e inicialização das ESPCAMs
- Uso de biblioteca Open
Source(<https://github.com/ESP32Tutorials/ESP32-CAM-ESP-IDF-Live-Streaming-Web-Server>)
- Streaming por HTTP em formato MJPEG
- ~10fps (100ms por frame) quando rede boa
- Uso de 3 ESPCAMs congestiona a rede.



Etapa 2 - Visualização da câmera

- Classificador Cascade
- Identifica o rosto e as posições cartesianas
- Cálculo das distâncias

```
cap = cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + "haarcascade_frontalface_default.xml")

while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)

    h, w, _ = frame.shape
    frame_center = (w // 2, h // 2)
    cv2.circle(frame, frame_center, 5, (0,255,0), -1)

    yaw_output = 0
    text = "Nenhuma face detectada"

    for (x, y, fw, fh) in faces:
        face_center = (x + fw//2, y + fh//2)

        cv2.circle(frame, face_center, 5, (0,0,255), -1)
        cv2.rectangle(frame, (x,y), (x+fw,y+fh), (255,0,0), 2)

        #distancia do centro da face - frame
        dx = face_center[0] - frame_center[0]
        dy = -face_center[1] + frame_center[1]
```

Etapa 3 - Classe VideoStream

```
class VideoStream:  
    def __init__(self, src=0):  
        self.src = src  
        self.stream = cv2.VideoCapture(src)  
        self.ret, self.frame = self.stream.read()  
        self.stopped = False  
        if not self.ret:  
            self.frame = None  
  
    def start(self):  
        Thread(target=self.update, args=()).start()  
        return self
```

Etapa 3 - Classe VideoStream

```
def update(self):
    while True:
        if self.stopped:
            self.stream.release()
            return

        # Tenta ler o frame
        grabbed, frame = self.stream.read()

        if grabbed:
            self.frame = frame
            self.ret = True
        else:
            # Se falhar, marca como erro e tenta reconectar
            self.ret = False
            # Não printa toda hora para não poluir, mas tenta reabrir
            try:
                self.stream.release()
                time.sleep(1) # Espera um pouco antes de tentar de novo
                self.stream = cv2.VideoCapture(self.src)
                print(f"Tentando reconectar a {self.src}...")
            except:
                pass
```

Etapa 3 - Classe VideoStream

- self.ret, boolean para verificar se conseguiu encontrar um frame válido

```
def read(self):  
    return self.ret, self.frame  
  
def stop(self):  
    self.stopped = True
```

Etapa 3 - Classe VideoStream

- Função adaptada para leitura do rosto frontal, perfil e espelho da imagem

```
# Função auxiliar para detectar rostos (Frente, Perfil ou Invertido)
def detect_one_face(img):
    if img is None: return None
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 1. Frontal
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    if len(faces) > 0: return faces[0]

    # 2. Perfil
    faces_prof = profile_cascade.detectMultiScale(gray, 1.3, 5)
    if len(faces_prof) > 0: return faces_prof[0]

    # 3. Perfil Invertido, espelha a imagem
    flipped = cv2.flip(gray, 1)
    faces_flip = profile_cascade.detectMultiScale(flipped, 1.3, 5)
    if len(faces_flip) > 0:
        (x, y, w, h) = faces_flip[0]
        x = img.shape[1] - x - w
        return (x, y, w, h)

    return None
```

Etapa 3 - Leitura das duas câmeras

- Captura das duas ESPCAMS

```
# === INICIALIZAÇÃO ===
cam_left = VideoStream("http://192.168.15.22").start()
cam_right = VideoStream("http://192.168.15.19").start()
time.sleep(2.0)

last_send_time = 0

while True:
    retL, frameL = cam_left.read()
    retR, frameR = cam_right.read()

    # Se alguma câmera caiu, mostra aviso mas NÃO TRAVA O LOOP
    if frameL is None or frameR is None:
        print("Aguardando sinal das câmeras...")
        time.sleep(0.5)
        continue

    # Setup da imagem (Esquerda é a principal)
    h, w, _ = frameL.shape
    frame_center = (w // 2, h // 2)
    cv2.circle(frameL, frame_center, 5, (0, 255, 0), -1) # Centro da tela (Verde)

    # Detecção
    rectL = detect_one_face(frameL)
    rectR = detect_one_face(frameR)

    text_info = "Procurando..."
```

Etapa 3 - Cálculo da profundidade

- Profundidade (Z) é calculada a partir da diferença entre as posições horizontais do rosto nas duas imagens (disparidade).

```
# === LÓGICA PRINCIPAL (Só se achar rosto nas duas) ===
if rectL is not None and rectR is not None:

    # Coordenadas Esquerda
    (xL, yL, wL, hL) = rectL
    face_center_L = (xL + wL//2, yL + hL//2)

    # Coordenadas Direita
    (xR, yR, wR, hR) = rectR
    cxR = xR + wR//2

    # 1. CÁLCULO DE ERRO (X, Y)
    dx = face_center_L[0] - frame_center[0]
    dy = -(face_center_L[1] - frame_center[1])

    # 2. CÁLCULO DE PROFUNDIDADE (Z)
    cxL = face_center_L[0]
    disparity = abs(cxL - cxR)

    if disparity > 2:
        Z = (FOCAL_LENGTH * BASELINE) / disparity
    else:
        Z = 0.0

    # Desenho Visual
    cv2.circle(frameL, face_center_L, 5, (0, 0, 255), -1) # Centro do rosto (Vermelho)
    cv2.rectangle(frameL, (xL, yL), (xL+wL, yL+hL), (255, 0, 0), 2)

    text_info = f"X:{dx} Y:{dy} Z:{Z:.2f}"
```

Etapa 4 - Envio das posições

- Enviar a cada 100ms
- Leitura da ESP
- Exibição

```
# 3. ENVIO
current_time = time.time()
if (current_time - last_send_time) > SEND_INTERVAL:

    try:
        msg = f"posX:{int(dx)}, posY:{int(dy)}, posZ:{z:.2f}"
        s.send(msg.encode())

    # Ler resposta do ESP32
    try:
        resp = s.recv(1024).decode()
        print("ESP32:", resp)
    except:
        pass

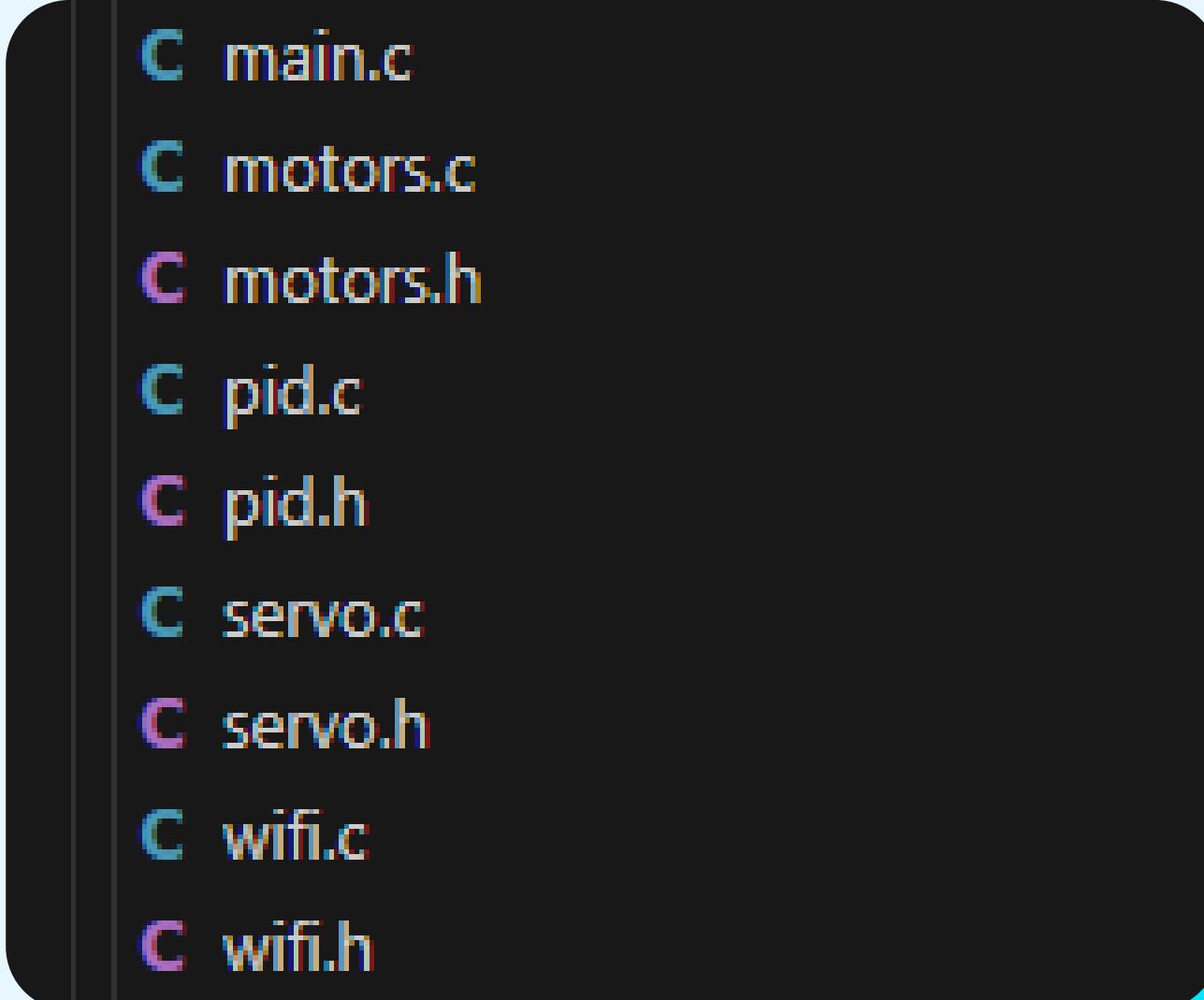
        print(f"ENVIADO! >> {msg}")
        last_send_time = current_time

    except Exception as e:
        print("Erro ao enviar TCP:", e)
# === EXIBIÇÃO ===
cv2.putText(frameL, text_info, (20, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,255,255), 2)

try:
    frameL_s = cv2.resize(frameL, (0,0), fx=0.6, fy=0.6)
    frameR_s = cv2.resize(frameR, (0,0), fx=0.6, fy=0.6)
    combined = cv2.hconcat([frameL_s, frameR_s])
    cv2.imshow("Sistema Stereo XYZ", combined)
except Exception as e:
    print(f" ERRO ENVIO: O ESP32 não respondeu. {e}")
pass
```

Etapa 5 - Controle

- Recebe as coordenadas cartesianas do objeto e calcula o valor de PWM para cada motor e o ângulo do servo.
- Uso de biblioteca próprias:
 - motors.c: Inicialização dos motores e funções para funcionamento da ponte H.
 - servo.c: Inicialização do servo e calculo de ângulo → pulso + trabalho
 - pid.c: Cálculos de controle e declaração de struct.
 - wifi.c: Conexão wifi e tcp_server_task()
 - main.c: Inicialização geral e control_task()



- main.c
- motors.c
- motors.h
- pid.c
- pid.h
- servo.c
- servo.h
- wifi.c
- wifi.h

Etapa 5 - Controle, main

```
void app_main(void)
{
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }

    newDataSemaphore = xSemaphoreCreateBinary();
    if (newDataSemaphore == NULL) {
        ESP_LOGE("MAIN", "Falha ao criar newDataSemaphore");
        esp_restart();
    }

    posMutex = xSemaphoreCreateMutex();
    if (posMutex == NULL) {
        ESP_LOGE("MAIN", "Falha ao criar posMutex");
        esp_restart();
    }

    ESP_LOGI("MAIN", "Inicializando Wi-Fi...");
    wifi_init_sta();
```

Etapa 5 - Controle, main

```
BaseType_t ok;
ok = xTaskCreate(tcp_server_task, "tcp_server", 8192, NULL, 5, NULL);
if (ok != pdPASS) {
    ESP_LOGE("MAIN", "Falha ao criar tcp_server_task");
}

ok = xTaskCreate(control_task, "control", 8192, NULL, 6, NULL);
if (ok != pdPASS) {
    ESP_LOGE("MAIN", "Falha ao criar control_task");
}

while (1) {
    //ESP_LOGI("MAIN", "Sistema rodando...");
    vTaskDelay(pdMS_TO_TICKS(5000));
}
```

Etapa 5 - Controle, control_task()

```
void control_task(void *pvParameters)
{
    motors_init();
    servo_init();

    ESP_LOGI("CONTROL", "Teste inicial: servo 90, motores 0");
    servo_set_angle(90.0f);
    motors_set(0, 0);
    vTaskDelay(pdMS_TO_TICKS(500));

    PID_t pid_x, pid_z;
    PID_Init(&pid_x, 1, 0.01, 0.10, 0.10);
    PID_Init(&pid_z, 0.60, 0.00, 0.20, 0.10);

    PID_SetOutputLimits(&pid_x, -255, 255);
    PID_SetOutputLimits(&pid_z, -255, 255);
```

Etapa 5 - Controle, control_task()

```
while (1)
{
    // aguarda dado novo do Wi-Fi (bloqueante). Se quiser ver logs periódicos,
    if (xSemaphoreTake(newDataSemaphore, portMAX_DELAY) == pdTRUE) {
        float x, y, z;

        // leitura segura das variáveis globais
        if (xSemaphoreTake(posMutex, pdMS_TO_TICKS(1000)) == pdTRUE) {
            x = posX;
            y = posY;
            z = posZ;
            xSemaphoreGive(posMutex);
        } else {
            ESP_LOGW("CONTROL", "Falha ao tomar posMutex");
            continue;
        }

        ESP_LOGI("CONTROL", "Pos recebida: x=%.3f y=%.3f z=%.3f", x, y, z);
    }
}
```

Etapa 5 - Controle, control_task()

```
float err_x = x;
float err_z = Z_target - z;

//float out_x = PID_Update(&pid_x, err_x);
float out_x = err_x; // só proporcional para teste
float out_z = PID_Update(&pid_z, err_z);

//float motor_esq = out_z - out_x;
//float motor_dir = out_z + out_x;

float motor_esq = -out_x;
float motor_dir = out_x;

// saturação
motor_esq = fminf(fmaxf(motor_esq, -255.0f), 255.0f);
motor_dir = fminf(fmaxf(motor_dir, -255.0f), 255.0f);

// log antes de enviar aos motores
ESP_LOGI("CONTROL", "Motors raw: L=%f R=%f", motor_esq, motor_dir);
motors_set((int)motor_esq, (int)motor_dir);
```

Etapa 5 - Controle, control_task()

```
float angulo = angulo_atual + (y * 0.03);
if (angulo < 0.0f) angulo = 0.0f;
if (angulo > 180.0f) angulo = 180.0f;

angulo_atual = angulo;

ESP_LOGI("CONTROL", "Servo angulo pedido: %.2f", angulo);
servo_set_angle(angulo);
} else {
    // raro se usar portMAX_DELAY, mas deixa para robustez
    ESP_LOGW("CONTROL", "xSemaphoreTake(newDataSemaphore) falhou/timeout");
}
```

Etapa 5 - Controle, tcp_server_task()

```
void tcp_server_task(void *pvParameters) {
    char rx_buffer[128];
    char addr_str[128];
    int addr_family = AF_INET;
    int ip_protocol = IPPROTO_IP;

    struct sockaddr_in dest_addr;
    dest_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(PORT);

    int listen_sock = socket(addr_family, SOCK_STREAM, ip_protocol);
    if (listen_sock < 0) {
        ESP_LOGE(TAGWIFI, "Unable to create socket: errno %d", errno);
        vTaskDelete(NULL);
        return;
    }

    bind(listen_sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    listen(listen_sock, 1);

    ESP_LOGI(TAGWIFI, "Servidor TCP escutando na porta %d", PORT);
```

Etapa 5 - Controle, tcp_server_task()

```
while (1) {
    struct sockaddr_in source_addr;
    socklen_t addr_len = sizeof(source_addr);

    int sock = accept(listen_sock, (struct sockaddr *)&source_addr, &addr_len);
    if (sock < 0) {
        ESP_LOGE(TAGWIFI, "Erro no accept: errno %d", errno);
        continue;
    }

    inet_ntoa_r(((struct sockaddr_in *)&source_addr)->sin_addr.s_addr,
                addr_str, sizeof(addr_str) - 1);
    ESP_LOGI(TAGWIFI, "Cliente conectado: %s", addr_str);

    send(sock, "Conexao estabelecida!\n", 23, 0);
```

Etapa 5 - Controle, tcp_server_task()

```
while (1) {
    int len = recv(sock, rx_buffer, sizeof(rx_buffer) - 1, 0);
    if (len <= 0) break;

    rx_buffer[len] = 0;
    // ESP_LOGI(TAGWIFI, "Recebido: %s", rx_buffer);

    float x, y, z;

    // Tenta ler com ou sem espaços
    if (sscanf(rx_buffer, "posX:%f, posY:%f, posZ:%f", &x, &y, &z) == 3 ||
        sscanf(rx_buffer, "posX:%f , posY:%f , posZ:%f", &x, &y, &z) == 3) {

        // Protege com mutex
        if(posMutex != NULL) {
            xSemaphoreTake(posMutex, portMAX_DELAY);
            posX = x;
            posY = y;
            posZ = z;
            xSemaphoreGive(posMutex);
        }

        // Notifica a task de controle
        if(newDataSemaphore != NULL) {
            xSemaphoreGive(newDataSemaphore);
        }

        ESP_LOGI(TAGWIFI, "POS -> X=%f Y=%f Z=%f", posX, posY, posZ);
        send(sock, "OK\n", 3, 0);
    }
}
```

Etapa 6 - testes validação e calibração

Aqui que os problemas apareceram...

- O uso de 3 ESPCAMS não deu certo, usamos duas.
- Zona Morta do motor
 - PWM é gerado pelo erro
 - Se o PWM é $> 40\%$ a roda não gira por conta do atrito/inercia
 - Apenas controles brutos
 - Inicialmente a frequência de amostragem era 1Hz, modificamos para 10Hz
 - Equilíbrio entre frequência de amostragem e parâmetros do PID
- Apenas controle Proporcional para não ter influencia dos erros anteriores.
- ESPCAMS puxam muita corrente.