UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
ENGENHARIA MECATRÔNITA

THALLES JONATHAN TRIGUEIRO DE ALMEIDA

IMPLEMENTAÇÃO DE ROBÔ MÓVEL PARA RASTREIO DE PESSOA COM VISÃO COMPUTACIONAL

Uberlândia

2026

THALLES JONATHAN TRIGUEIRO DE ALMEIDA

IMPLEMENTAÇÃO DE ROBÔ MÓVEL PARA RASTREIO DE PESSOA COM VISÃO COMPUTACIONAL

Projeto de Fim de Curso apresentado a Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica:
Orientador: Prof. Dr. Éder Alves de Moura

VERSÃO PARA A BANCA

Uberlândia
2026

THALLES JONATHAN TRIGUEIRO DE ALMEIDA

IMPLEMENTAÇÃO DE ROBÔ MÓVEL PARA RASTREIO DE PESSOA COM VISÃO COMPUTACIONAL

Projeto de Fim de Curso apresentado a Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica:
Orientador: Prof. Dr. Éder Alves de Moura

Uberlândia - MG, 17 de março de 2026: Banca Examinadora:

Éder Alves de Moura – Dr. (FEELT/UFU)
(Orientador)

Gabriela Vieira Lima – Dra. (FEELT/UFU)

Jeovane Vicente de Sousa – Dr. (FEELT/UFU)

Leonardo Muttoni – Dr. (FACOM/UFU)

AGRADECIMENTOS

Primeiramente agradeço a Deus pela oportunidade de poder concluir um sonho, e perseverança para não desistir em momentos difíceis.

Aos meus pais, por me darem força e um apoio incondicional, vocês são o alicerce que me mantém sempre querendo avançar.

Ao meu orientador Éder Alves de Moura por toda paciência e ensinamentos, sem o seu apoio o caminho mais difícil.

Aos demais professores que contribuíram para a minha formação.

Aos meus amigos e colegas de curso que conviveram mais comigo que muitos familiares, passamos juntos, momentos de alegrias e sofrimentos.

A Universidade Federal de Uberlândia, pela infraestrutura e pelas oportunidades que moldaram o meu caráter profissional como Engenheiro Mecatrônico.

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para a minha formação.

“Se você não gosta do seu destino,
não aceite. Em vez disso, tenha a
coragem de mudá-lo do jeito que
você quer que ele seja.”
(Masashi Kishimoto)

RESUMO

O desenvolvimento de um robô autônomo com a capacidade de rastrear pessoas integra áreas de conhecimento como a visão computacional, sistema de controle e protocolos de comunicação. Nesse contexto, este trabalho tem como objetivo desenvolver e implementar um robô móvel que realiza o rastreio de uma pessoa em ambientes não estruturados. A arquitetura do robô é baseada na unidade de processamento de alto nível, implementada em uma Raspberry Pi, que executa o algoritmo de visão computacional e a lógica de controle; a unidade de processamento de baixo nível, baseada na ESP32, faz interface com o sistema de potência e controla o acionamento dos motores por meio de uma Ponte H; por fim os atuadores do sistema consistem em dois motores de corrente contínua que são acionados de forma diferencial e independentes. O algoritmo de visão computacional, implementado em *Python*, processa o fluxo de imagens da captura até a extração das informações necessárias. Ao capturar as imagens são convertidas para matrizes numéricas, por meio das bibliotecas *OpenCV* e *NumPy*, para alimentar a rede neural YOLO11 (*Ultralytics*). Esta, por sua vez, é responsável por realizar a detecção de pessoas e extrair os parâmetros da *bounding box* da imagem detectada, e utilizam esses valores de coordenada horizontal central e a altura da caixa delimitadora, para respectivamente, realizar a correção de orientação, mantendo o robô alinhado em relação ao alvo e distância estabilizando o robô em uma faixa de repouso. Esses valores são normalizados, e passam pelo processo filtragem estocástica por meio do filtro de Kalman, que realiza a estimação das variáveis, reduzindo ruídos das detecções. As variáveis estimadas são fornecidas para a máquina de estados finitos, onde são definidos os estados de operação do robô. As velocidades são definidas e enviadas via protocolo de comunicação MQTT em rede local, essas informações coordenam o sistema de tração diferencial controlado por uma Ponte H e motores de corrente contínua aplicando técnicas de histerese, zona morta e acionamento intermitente para mitigar o atrito estático e o *chattering*. Os resultados experimentais mostram a capacidade do robô de manter o rastreamento do alvo com um comportamento estável, operando em tempo necessário para o acompanhamento de uma pessoa enquanto anda pelo ambiente. Este resultado evidencia a viabilidade da integração entre técnicas de visão computacional e controle embarcado em sistemas robóticos móveis autônomos.

Palavras-chave: robótica móvel, visão computacional, rastreamento de pessoas, YOLO, filtro de Kalman.

ABSTRACT

The development of an autonomous robot capable of tracking people integrates knowledge areas such as computer vision, control systems, and communication protocols. In this context, this work aims to develop and implement a mobile robot that performs human tracking in unstructured environments. The robot architecture is based on a high-level processing unit, implemented on a Raspberry Pi, which executes the computer vision algorithm and the control logic; the low-level processing unit, based on the ESP32, interfaces with the power system and controls motor actuation through an H-bridge; finally, the system actuators consist of two direct current motors driven in a differential and independent manner. The computer vision algorithm, developed in *Python*, processes the image stream from acquisition to the extraction of the required information. Upon acquisition, the images are converted into numerical matrices using the *OpenCV* and *NumPy* libraries to feed the YOLO11 (*Ultralytics*) neural network. This network is responsible for detecting people and extracting the parameters of the image *bounding box*, using the horizontal center coordinate and the height of the bounding box to perform, respectively, orientation correction, keeping the robot aligned with the target, and distance control, stabilizing the robot within a resting range. These values are normalized and undergo stochastic filtering using the Kalman filter, which estimates the variables and reduces detection noise. The estimated variables are then provided to the finite state machine, where the robot's operational states are defined. The velocities are defined and transmitted via the MQTT communication protocol over a local network; this information coordinates the differential traction system controlled by an H-bridge and direct current motors, applying hysteresis, deadband, and intermittent actuation techniques to mitigate static friction and *chattering*. The experimental results demonstrate the robot's ability to maintain target tracking with stable behavior, operating within the required time to follow a person walking through the environment. This result highlights the feasibility of integrating computer vision techniques with embedded control in autonomous mobile robotic systems.

Keywords: mobile robotics, computer vision, person tracking, YOLO, Kalman filtering.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BJT	Bipolar Junction Transistors
CNNs	Convolutional Neural Networks
DC	Direct Current
DL	Deep Learning
FSM	Finite State Machine
HTTP	Hypertext Transfer Protocol
IA	Inteligência Artificial
MQTT	Message Queuing Telemetry Transport
PWM	Pulse Width Modulation
QoS	Quality of Services
RNAs	Redes Neurais Artificiais
SBC	Single Board Computers
SoC	System on a Chip
YOLO	You Only Look Once

SUMÁRIO

1 INTRODUÇÃO

Este trabalho consiste no desenvolvimento de um robô móvel com sistema de rastreio de pessoas, implementado com técnicas de Visão Computacional e *Deep Learning* (DL), de modo a manter uma distância e orientação constante do alvo.

Este capítulo apresenta introdução deste trabalho. Para isso, a Seção 1.1 descreve o contexto geral em que a temática desta proposta está inserida, a Seção 1.2 determina os objetivos gerais e específicos e a Seção 1.3 discorre sobre as justificativas sustentadas para o desenvolvimento. Por fim, a Seção 1.4 apresenta como o restante deste trabalho está organizado.

Este trabalho desenvolveu um robô móvel, cujo código-fonte utilizado foi disponibilizado em repositório público[1], conforme documentado por Almeida (2026).

1.1 Contextualização

A robótica tem sido vislumbrada como uma ferramenta de auxílio em diversas áreas da atividade humana. Nesse sentido, ela estende o conceito de aplicação de sistemas eletromecânicos como suporte para diversas áreas e, principalmente, no contexto da atuação profissional, como forma de auxiliar no emprego de força, em situações de risco e de desgaste do corpo. Essa possibilidade diz muito sobre a incorporação de elementos de raciocínio artificial no sistema de controle, permitindo que a máquina decida de forma autônoma e amplie suas possibilidades e seu uso.

Historicamente, sempre foi um desafio a criação de robôs capazes de tomar decisões para executar tarefas complexas, mas foi o advento da Inteligência Artificial (IA), que permitiu vislumbrar esse potencial (AMARAL; GASPAROTTO, 2021). A IA diferentemente da programação convencional, utiliza dados para a criação de soluções adaptativas, expandindo as possibilidades para automação de sistemas. Segundo Zhong et al. (2017), esta tecnologia, confere aos processos a capacidade cognitiva de aprendizado, processamento lógico e ações autônomas.

Para esse evento ocorrer, foram necessários muitos anos de desenvolvimento teórico e tecnológico, a ponto de existirem ferramentas computacionais e capacidade de processamento disponíveis para integrar a construção de sistemas embarcados capazes de implementar os conceitos de IA que estão em uso (e em desenvolvimento) nos dias atuais. Dentre as ferramentas de IA que ganham grande atenção e, por conseguinte, perceberam um grande desenvolvimento, foi a Visão Computacional, principalmente considerando os avanços alcançados com as técnicas de DL.

Um marco decisivo nesse cenário foi a introdução da arquitetura AlexNet em 2012. Segundo Aloysius e Geetha (2017), o modelo com enfoque na sua capacidade de operar com

[1] Disponível em: https://github.com/thallesj5/UFU_Engenharia_Mecatronica_PFC. Acesso em: 29 abr. 2026.

aprendizado supervisionado, atuou como o principal responsável pela disseminação das Redes Neurais Convolucionais (*Convolutional Neural Networks*, CNNs), dispensando rotinas complexas de pré-treinamento não supervisionado, ao superar a precisão dos algoritmos tradicionais. Conforme Alom et al. (2018), esta arquitetura se destaca por desencadear o crescimento acelerado nas pesquisas e no desenvolvimento de DL aplicado à análise de imagens.

O DL utiliza redes neurais artificiais com múltiplas camadas, inspiradas em redes neurais biológicas, que são particularmente eficientes no processamento de dados não estruturados (GOODFELLOW; BENGIO; COURVILLE, 2016). Conforme Voulodimos et al. (2018), o DL tem aplicabilidade na análise de dados complexos, mesmo sem um pré-processamento extenso, característica fundamental para o seu crescimento em diversas áreas.

A detecção de objetos tem a capacidade de identificação e localização de objetos em imagens. Os detectores de objetos possuíam arquiteturas de múltiplas etapas, que requeriam alta demanda computacional, resultando em limitações para sistemas de baixa latência (KOTTHAPALLI; RAVIPATI; BHATIA, 2025). Para Sohan, Ram e Reddy (2024), as CNNs assumiram um papel de destaque ao permitir a extração de características em múltiplos níveis para a detecção de objetos com maior precisão.

Além dos sistemas de múltiplas etapas, com a evolução dessas arquiteturas, os modelos baseados em DL passaram a ser divididos em duas categorias principais (SOHAN; RAM; REDDY, 2024). Enquanto os detectores clássicos dependem de propostas de região antes da realização da classificação, os detectores de estágio único (*one-stage*) processam simultaneamente as previsões de classe e localização em uma única etapa . Para Kotthapalli, Ravipati e Bhatia (2025), essa transição para arquiteturas de estágio único foi essencial para eliminar gargalos computacionais e viabilizar aplicações sensíveis à latência que exigem processamento em tempo hábil de operação, como a robótica e a navegação autônoma.

Dentre esses tipos de redes, foi utilizada para este trabalho uma rede denominada YOLO, conhecida por sua eficiência, precisão na detecção e rastreamento de objetos em tempo hábil. Conforme Sohan, Ram e Reddy (2024), Jiang et al. (2022), a rede YOLO utiliza uma única CNN para avaliar o contexto global da imagem ao prever as classes e coordenadas espaciais dos objetos diretamente em um único passo. A aptidão da rede YOLO para identificar e seguir objetos em tempo hábil oferece muitas aplicações, como o rastreamento de múltiplos alvos e o controle preciso e adaptativo de servomecanismos.

Considerando todos esses avanços, os desafios tecnológicos atuais estão relacionados ao desenvolvimento de sistemas robóticos autônomos para diversas aplicações e vários campos da atividade humana. Assim, neste contexto, este trabalho descreve o processo de implementação de um protótipo de robô seguidor, utilizando técnicas de visão computacional.

1.2 Objetivos

Assim, em vista do que foi apresentado até este ponto, foi definido como objetivo geral deste trabalho, o desenvolvimento de um robô móvel capaz de rastrear uma pessoa enquanto esta se movimenta em um ambiente não estruturado, utilizando visão computacional para identificação e guiamento.

Em acréscimo, são definidos os seguintes objetivos específicos para o desenvolvimento deste trabalho:

- Desenvolver uma plataforma móvel com unidade de servomecanismo, integrando uma unidade de processamento computacional embarcada sob uma estrutura robótica de duas rodas de tração diferencial.
- Implementar o modelo de visão computacional YOLO11 de forma embarcada para a detecção e o rastreamento de objetos.
- Projetar e implementar o sistema de controle de posição e velocidade para o robô, em função da distância e orientação relativa ao alvo.

1.3 Justificativa

Este projeto tem sua relevância evidenciada em seu caráter multidisciplinar, integrando diversos conhecimentos teóricos e aplicando-os na prática, o que é fundamental para a graduação em Engenharia Mecatrônica. Para sua execução, foram exigidos conhecimentos em eletrônica, programação e controle para, respectivamente, a montagem do hardware, o desenvolvimento dos algoritmos, tanto de processamento de imagens (com o uso de visão computacional) quanto de comunicação, sincronização e otimização da resposta do mecanismo. Com isso, o projeto demonstra a capacidade de aplicação em um sistema embarcado real dos conceitos teóricos.

O desenvolvimento deste projeto explora o desafio de integrar o meio físico e digital (*hardware* e *software*) e de construir algoritmos que processam imagens em tempo hábil. Isso proporciona o desenvolvimento de habilidades cruciais para a evolução profissional, por meio da resolução de problemas complexos. Sendo essas habilidades requisitadas em diversos setores do mercado de trabalho, principalmente no ambiente industrial.

Um sistema de rastreamento de pessoas possui uma gama de aplicações em diversos setores. Na implementação de sistemas de segurança, para monitoramento inteligente em que é necessário o acompanhamento de um alvo em meio ao ambiente, na qual a base móvel garante a permanência do alvo no campo de visão. Na logística, possibilita o manuseio de objetos de risco ou o auxílio a operadores com o transporte de carga, em ambientes de depósitos e armazéns. Na melhoria na qualidade de vida de Pessoas com Deficiência (PcD), com robôs autônomos auxiliando-as na mobilidade e acessibilidade,

como por exemplo, a implementação de sistemas de rastreamento em cadeiras de roda. Logo, a proposta deste trabalho pode servir como uma base para o desenvolvimento de tecnologias que aumentam a integração de sistemas robóticos autônomos no cotidiano.

1.4 Estrutura do Trabalho

Este trabalho será dividido em cinco capítulos, organizados da seguinte forma:

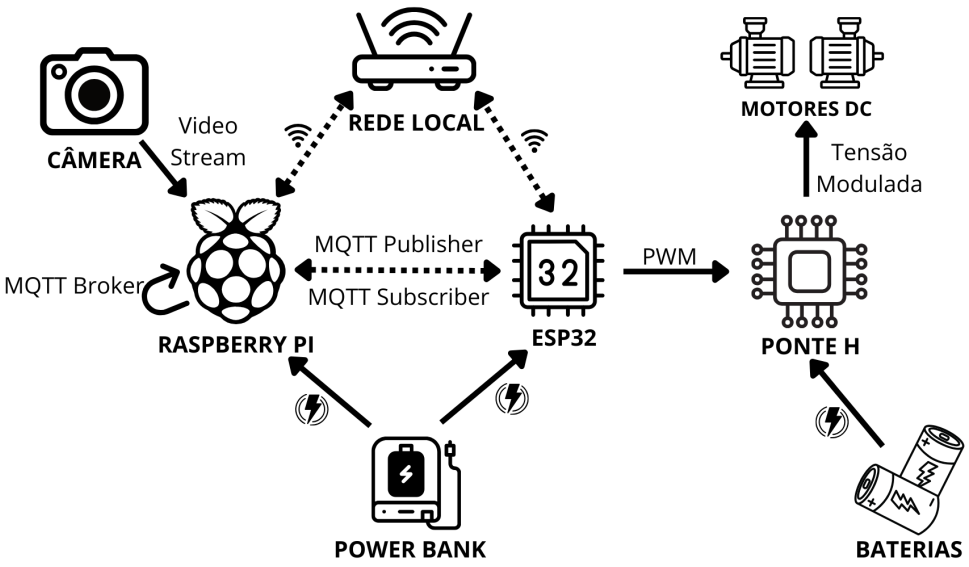
- Capítulo 2 - Fundamentação Teórica: Mostra o embasamento teórico necessário para a compreensão do projeto. Detalha o funcionamento de servomecanismos, o modelo de visão computacional, o sistema de controle, e apresenta as fontes bibliográficas e trabalhos relacionados, em que este projeto foi embasado.
- Capítulo 3 - Metodologia: Descreve a metodologia utilizada. Detalha a escolha de componentes, a montagem do servomecanismo e o desenvolvimento do algoritmo. Abordando a integração entre ambos e os critérios de avaliação.
- Capítulo 4 - Resultados e Discussões: Apresenta os resultados do projeto por meio de diagramas de bloco e descrições dos modelos e algoritmos utilizados. Compara os resultados obtidos com os objetivos propostos, discutindo o desempenho do sistema e as limitações encontradas.
- Capítulo 5 - Conclusão: Sintetiza os principais pontos encontrados no trabalho e sugere ideias para projetos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo realiza o embasamento teórico que fundamenta o desenvolvimento do projeto. O projeto trata-se de um sistema autônomo baseado em visão computacional, exigindo uma pluralidade disciplinar e mesclando conceitos de Robótica, Protocolos de Comunicação em Rede Wi-Fi, Visão Computacional e Sistemas de Controle.

Para o desenvolvimento do robô seguidor, foi definida uma arquitetura que segmenta o custo computacional do processamento do algoritmo de rastreio do sistema de atuação do protótipo. A estratégia adotada neste trabalho está representada na Figura 1, que detalha o diagrama de conexões e partes do projeto.

Figura 1 – Diagrama da arquitetura do projeto.



Fonte: O Autor (2026).

No sistema proposto, observa-se o funcionamento e a integração dos componentes utilizados. Para o melhor entendimento da arquitetura do sistema, a progressão deste capitulo será segmentada em três partes:

- **Seção 2.1 - Arquitetura de Hardware e Comunicação:** Define-se a unidade de processamento de alto nível (Raspberry Pi), responsável pelo algoritmo de rastreamento. A unidade de processamento de baixo nível (ESP32), unidade periférica de interface, é responsável por converter as variáveis recebidas para o *driver* de potência (Ponte H) para o acionamento dos atuadores (Motores DC - *Direct Current*). Adicionalmente, a arquitetura de rede local realiza a integração de ambos os nós do sistema, por meio do protocolo de comunicação (MQTT - *Message Queuing Telemetry Transport*).

- **Seção 2.2 - Sistema de Visão Computacional:** Aborda os fundamentos da visão computacional, desde a aquisição do sinal pelo sensor óptico (Câmera) e os processos de amostragem e quantização da imagem até o uso de Redes Neurais Artificiais e Convolucionais. Com foco na arquitetura YOLO11, para detecção de objetos no tempo operacional e as técnicas de filtragem estocástica (Filtro de Kalman) para mitigação dos ruídos.
- **Seção 2.3 - Estratégia de Controle:** Discute-se os modelos cinemáticos aplicados a robôs de tração diferencial, bem como o embasamento de Máquinas de Estados Finitos (do inglês *Finite State Machine* – FSM), e a sua lógica de tomada de decisão que rege o comportamento do robô seguidor.

2.1 Arquitetura de Hardware e Comunicação

Para o contexto deste projeto, foi selecionada uma arquitetura fundamentada nos conceitos de sistemas embarcados. Estes se definem como a integração de hardware e software, projetados para executarem funções específicas, com recursos computacionais restritos. Numa perspectiva clássica, Wolf (2012) descreve um sistema embarcado como qualquer dispositivo com capacidade de processamento programável, mas que não atua como um computador de propriamente dito.

Uma aplicação deste conceito é o robô seguidor proposto, que se justifica pelos requisitos de visão computacional com uma baixa latência de resposta, exigindo uma infraestrutura de processamento com capacidade multitarefa. Adotando-se um sistema embarcado, no qual, segundo Delai e Coelho (2012) e Orlandini (2012), a estratégia é separar as tarefas de alto custo computacional das de baixo custo, evitando que o processamento de alto nível comprometa a baixa latência da dinâmica do sistema.

Para mitigar o gargalo de alto nível e evitar que o atraso de processamento pesado prejudique a movimentação do robô, adotou-se uma arquitetura baseada na decomposição temporal, separando o planejamento estratégico do controle reativo em tempo hábil (SIEGWART ROLAND E NOURBAKHSH, 2004). Com isso, selecionaram-se dois componentes para essas tarefas: uma Raspberry Pi, para alta performance, e uma ESP32, como microcontrolador dedicado, que se comunicam via protocolo MQTT em uma rede local.

Enquanto a Raspberry Pi gerencia as demandas das variáveis da rede neural, o ESP32 mantém continuamente o controle sobre os mecanismos de atuação. Essa independência garante que o microcontrolador gerencie o driver de potência e os atuadores de forma estável, sem ser interrompido pelos atrasos temporais intrínsecos ao módulo de visão de alto nível.

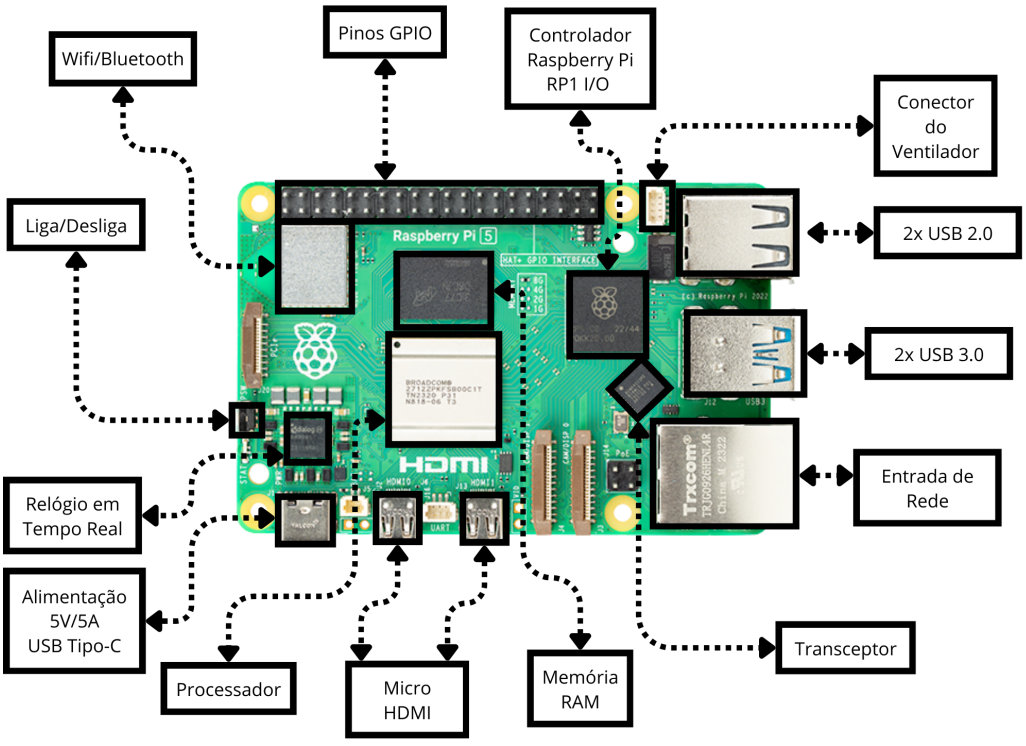
2.1.1 Unidade de Processamento de Alto Nível (Single Board Computers - SBC)

SBCs são componentes baseados em microprocessadores que, diferentemente dos microcontroladores, têm a capacidade de executar sistemas operacionais robustos, com acesso ao gerenciamento avançado de memória RAM e bibliotecas de alto nível. Esta arquitetura implica uma grande aplicabilidade em sistemas de alta demanda de processamento, como em algoritmos de visão computacional, que exigem arquiteturas de microprocessadores.

Segundo Upton e Halfacree (2016), a placa Raspberry Pi é considerada um computador muito pequeno que, pelo seu tamanho e baixo custo, possibilita uma gama de aplicações. As SBCs são de suma importância no projeto de robôs autônomos, pois em arquiteturas de controle distribuído, elas atuam como núcleo de inteligência, tendo a função de gerenciar de forma concorrente o fluxo de dados da câmera, o protocolo MQTT e o processamento da rede neural (como YOLO11).

Como ressaltado por Wolf (2012), para realizar os cálculos matemáticos demandados pelas redes neurais e processamento de imagens, os sistemas embarcados necessitam de alto desempenho. Neste contexto, o Raspberry Pi 5 de 8 GB, supre a base necessária para a resposta temporal, que como referido por Lee e Seshia (2016), não se trata apenas de uma métrica de desempenho, mas sim da garantia, de realizar a tarefa com altas restrições temporais. A arquitetura da Raspberry Pi 5, pode ser observada na Figura 2.

Figura 2 – Arquitetura de uma Raspberry pi 5.



Fonte: O Autor (2026).

A SBC Raspberry Pi supracitada é eficiente para o processamento de alto nível, devido a integração de componentes de alto desempenho e multitarefas, conforme especificações do fabricante [1]. Na Figura 2, alguns componentes se destacam, entre eles:

- **CPU:** formado por um processador ARM Cortex-A76 Quad-core de 2,4 Ghz, centraliza a execução do sistema operacional e a lógica dos algoritmos requeridos.
- **GPU:** Integrado ao CPU, o módulo VideoCore VII operando a 800MHz, garante o suporte de hardware necessário no pré-processamento de imagens e nas operações matriciais de redes neurais.
- **Memória RAM:** Com 8GB, esta memória tem a capacidade para o armazenamento temporário dos pesos das redes neurais e buffers de vídeo.
- **Controlador RP1:** Chip dedicado a gerenciar interfaces de entradas e saídas.
- **Interface de Conectividade:** 4 Portas USB (2x 2.0 e 2x 3.0), para aquisição de dados como os provenientes da câmera, e módulos Wi-fi para protocolos de comunicação como o MQTT.

2.1.2 Unidade de Processamento de Baixo Nível (System on a Chip - SoC)

Os microcontroladores são unidades de processamento de baixo nível que possuem uma arquitetura de sistema em um único chip (SoC), neste chip ocorre o encapsulamento de todos os componentes principais de um computador, como processador, memória, módulo Wi-Fi/bluetooth, contadores, entre outros periféricos. Com a capacidade de comunicação direta com sensores e atuadores, os projetos que utilizam microcontroladores podem ter rigorosas restrições temporais.

Isso posto, fornece aos SoC uma característica distinta das SBC que possuem a latência de sistemas operacionais complexos, além de garantir estabilidade na geração de sinais de controle, como a modulação por largura de pulso, do inglês *Pulse Width Modulation* (PWM).

Um microcontrolador utilizado em diversos projetos é o ESP32, de acordo com [2] a criadora do SoC ESP32, ele é integrado com um processador dual-core *Tensilica Xtensa LX6*. Esta arquitetura dual-core é crucial para o desenvolvimento deste projeto, pois concede a cada núcleo exclusividade em sua função, o primeiro processa as tarefas de rede e a comunicação do sistema, enquanto o segundo lida com o controle e a atuação do sistema.

Esta dedicação permite que o protótipo reaja com precisão em sua trajetória, conforme as variações do ambiente.

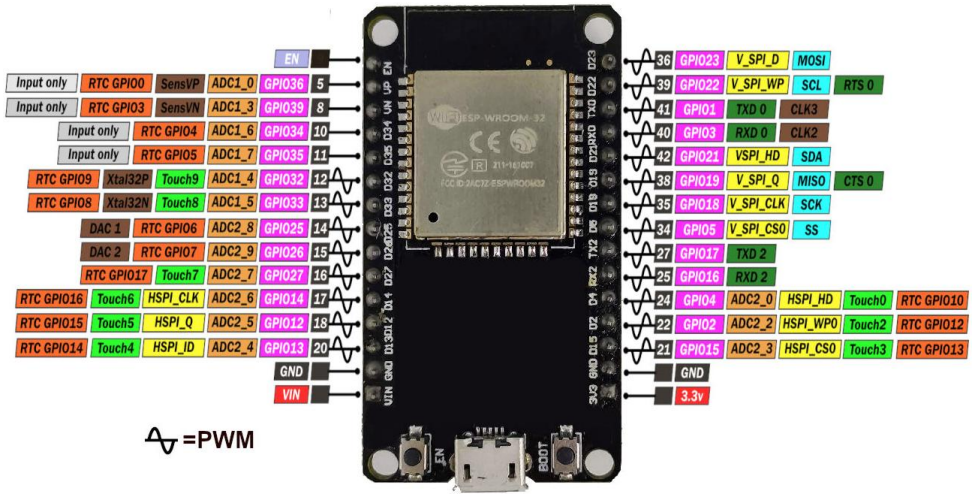
---

[1] Disponível em: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf Acessado em: 6 de fevereiro

[2] Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf Acessado em: 06 de fevereiro

A versatilidade da ESP32 pode ser observada na Figura 3, considerando o conjunto de funções disponíveis em seus pinos.

Figura 3 – Pinagem do ESP32 de 30 pinos.



Fonte: https://99tech.com.au/product/esp32-devkit-30u/. Acesso em: 4 mar. 2026

Segundo as especificações do fabricante[3], a ESP32 possui diversas entradas e saídas, o que permite a sua conexão com vários componentes e módulos. Isso é viável devido a pluralidade de funções de seus pinos GPIO mostrados na Figura 3, que podem operar como:

- **Saídas de Controle Digital**: Tem a capacidade de enviar sinais lógicos, como definir os estados do motor.
- **Canais de Modulação (PWM)**: Permitem o ajuste preciso da potência fornecida aos motores, sendo importantes para o controle em sistemas diferenciais.
- **Barramentos de Comunicação**: Portas dedicadas para protocolos, como I2C e SPI, viabilizam a integração de sensores.

No contexto deste protótipo, sua interface foi usada estrategicamente para a ligação com um driver de potência, a ponte H. Estas são características que tornaram o ESP32, o nó de execução física do sistema.

2.1.3 Eletrônica de Potência e Atuação

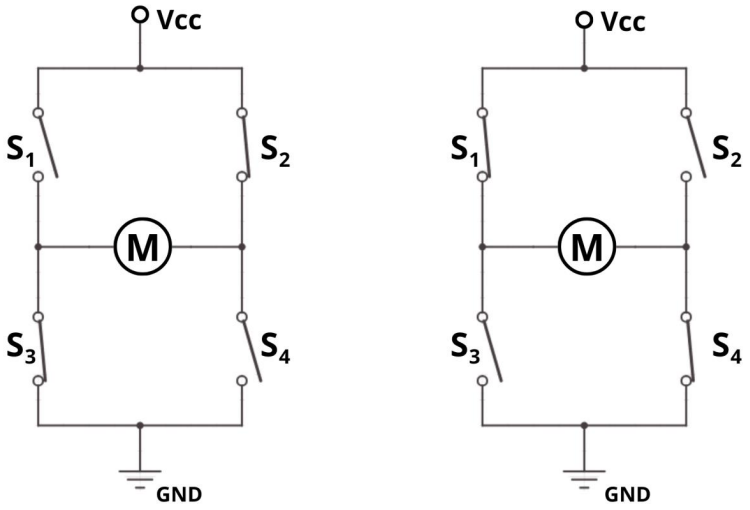
A eletrônica de potência detalha os componentes e técnicas responsáveis por converter as decisões lógicas do controlador em potência elétrica para a movimentação física do robô.

[3] Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf Acessado em: 06 de fevereiro

Para o acionamento dos motores de corrente contínua, utilizou-se o módulo driver L298N, baseado na topologia da Ponte H. Para Peerzada, Larik e Mahar (2021), este é um módulo de alta potência para o acionamento de motores CC e de passo, fazendo a intermediação entre o microcontrolador e os atuadores.

A operação interna do L298N pode ser dada por 4 chaves, transistores BJT (*Bipolar Junction Transistors*), na qual as chaves devem sempre ser ligadas em diagonal. O esquema de funcionamento por meio de chaves pode ser observado na Figura 4.

Figura 4 – Representação do circuito de uma Ponte H.



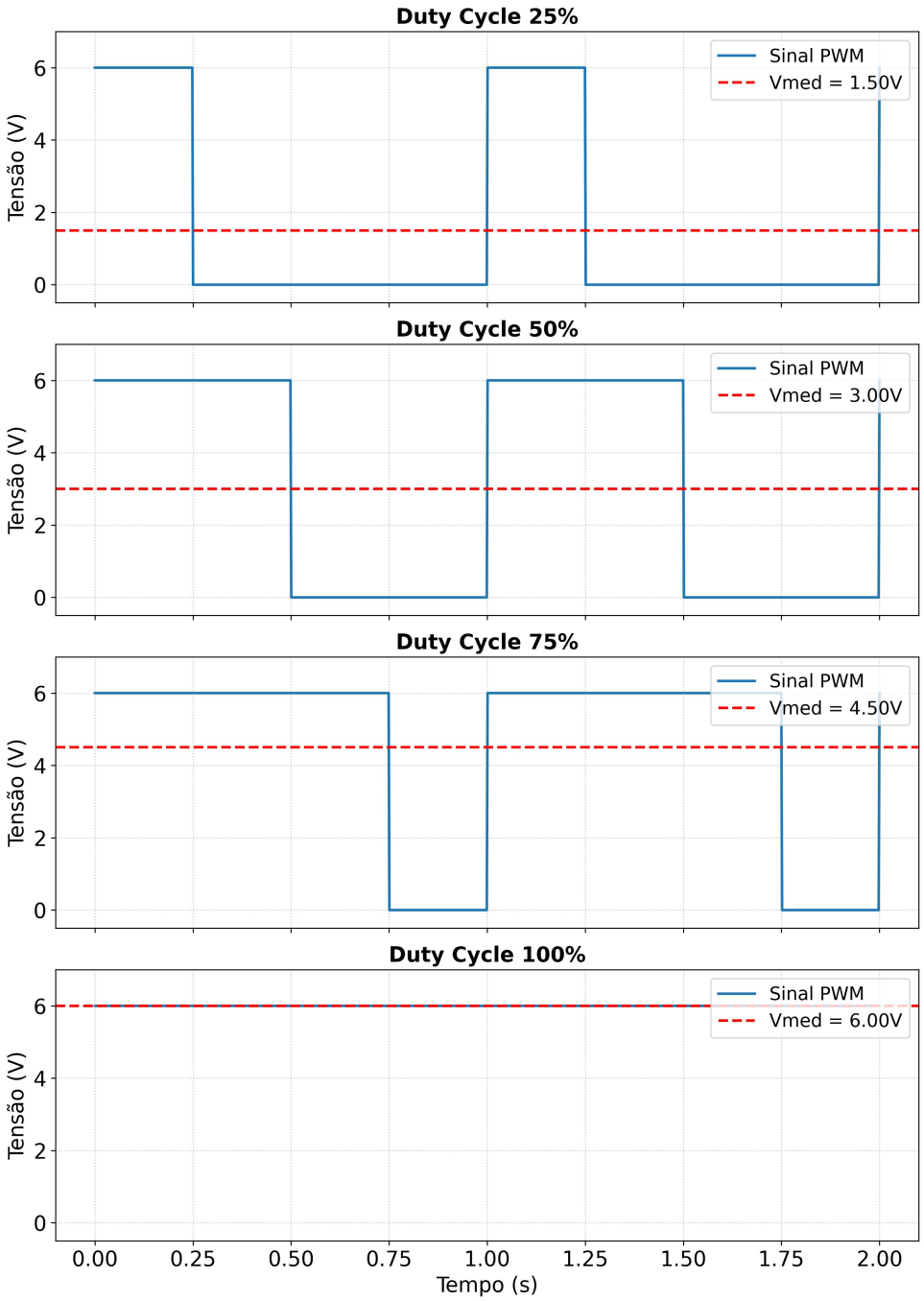
Fonte: Autor (2026)

A combinação entre as quatro chaves define se o motor girará no sentido horário, anti-horário ou ficará em repouso. Segundo Peerzada, Larik e Mahar (2021), em seus testes empíricos, notou-se uma queda de tensão interna do L298N, devido a saturação dos transistores BJT. Esta perda impacta diretamente na tensão efetiva disponível para os atuadores. onde V_{max} é a tensão máxima e $D \in [0, 1]$ é o ciclo de trabalho (*Duty cycle*), representando o percentual do valor máximo a ser aplicado ao sistema.

No ESP32, o parâmetro de ciclo de trabalho do PWM possui 8 bits, ou seja, 256 níveis (variando de 0 a 255), permitindo o controle de translação e rotação do robô. A ponte H permite o controle da velocidade e do sentido de rotação de dois motores CC, através do PWM que, para Rashid (2017), muito utilizada para a implementação de controle em conversores de potência. O PWM se baseia em ajustar a largura de um pulso de modo a manter uma tensão média desejada.

De acordo com Ogata (2010), um sistema de controle de duas posições é basicamente um sistema on-off. Devido a simplicidade e baixo custo, é muito utilizado em sistemas de controle industriais. Os sinais on-off em alta frequência geram uma tensão média proporcional ao tempo de permanência no estado “on”. A relação entre o ciclo de trabalho (*Duty Cycle*) e a tensão média (V_{med}) é apresentada na Figura 5.

Figura 5 – Relação entre o *Duty Cycle* e a tensão média de saída (VCC = 6 V).



Fonte: O Autor (2026)

2.1.4 Gerenciamento de Energia

O gerenciamento de energia está intrinsecamente ligado a estabilidade do protótipo e seu desempenho temporal. Segundo Orlandini (2012), o processamento pode gerar um consumo de energia alto. A Raspberry Pi 5 necessita de uma alimentação que atenda às demandas do rastreamento. Além de riscos a integridade física dos componentes, a insuficiência de corrente, em última instância, impede o funcionamento do protótipo.

A indutância dos motores de corrente contínua origina transitórios elétricos no momento da comutação, induzindo tensões parasitas no sistema. Para evitar o efeito da sensibilidade cruzada (cross sensitivity), descrita por Siegwart Roland e Nourbakhsh (2004) como a influência de fatores ambientais ou internos, como o campo magnético dos motores, no funcionamento dos componentes, adotou-se a estratégia de separar fisicamente os circuitos de potência e de lógica.

Considerando que o sistema de alimentação é distribuído, há isolamento entre a alimentação de potência e alimentação lógica ressaltando que o aterramento comum (GND) é obrigatório, para estabelecer o nível de 0 V como única referência (RASHID, 2017). O que também pode ser validado por Peerzada, Larik e Mahar (2021), o módulo L298N exige um aterramento comum entre a alimentação dos motores e a lógica do microcontrolador utilizado. Sem a referência única de 0 V, os sinais de PWM não podem ser interpretados corretamente, o que inviabiliza a comunicação entre a ponte H, L298N e o microcontrolador (como a ESP32).

2.1.5 Protocolo de Comunicação

Os protocolos de rede são conjuntos de regras que realizam a comunicação entre os diferentes nós de uma arquitetura. Conforme Tanenbaum (1981), uma rede é organizada como uma hierarquia de camadas, onde cada camada é construída sobre a anterior e também é um processo que comunica com outro processo em outra máquina.

O MQTT tem em sua estrutura três elementos: o Publicador (Publisher) que envia os dados; o Subscritor (Subscriber) que recebe os dados; e o Broker, que atua como servidor central, intermediando e distribuindo as mensagens. A seleção do MQTT como protocolo de comunicação entre o alto e o baixo nível, ao invés do padrão HTTP (*Hypertext Transfer Protocol*) justifica-se no comparativo técnico apresentado na Tabela 1.

Tabela 1 – Comparativo técnico entre os protocolos MQTT e HTTP.

Característica	**MQTT**	**HTTP**
Modelo de Comunicação	Publicar / Assinar	Requisição / Resposta
Tamanho do Cabeçalho	Mínimo (2 bytes)	Grande (Texto/Headers)
Latência Relativa	Baixa	Alta
Consumo de Energia	Reduzido	Elevado

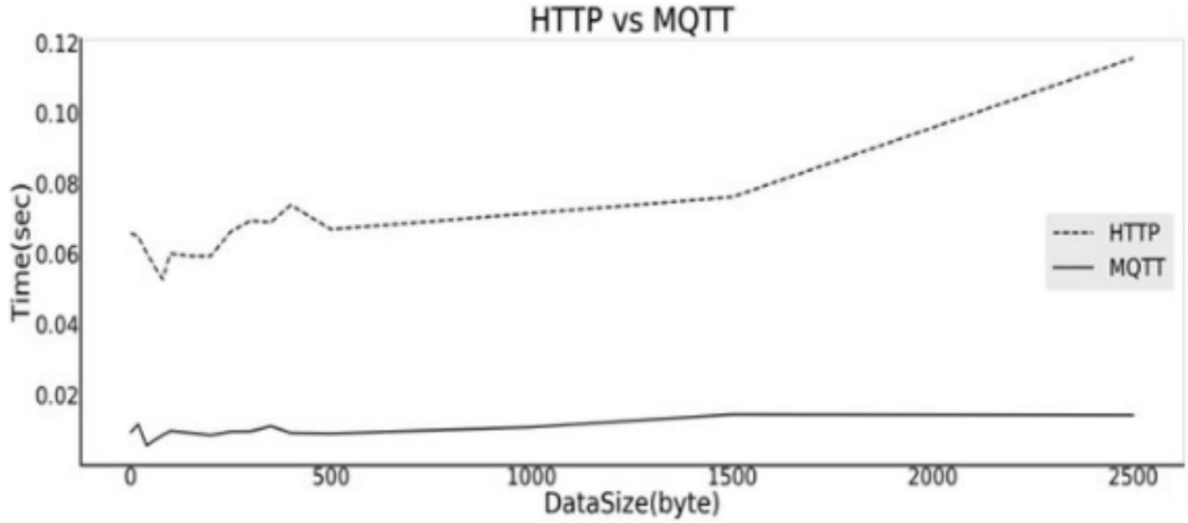
Fonte: Elaborado pelo Autor (2026), adaptado de Atmoko, Riantini e Hasin (2017) e Puthiyidam e Joseph (2024).

A adoção de uma arquitetura distribuída e separada em níveis pode ser embasada na teoria clássica de redes. Conforme Tanenbaum (1981), redes de computadores são hierarquicamente estruturadas como uma série de camadas independentes nas quais os processos podem se comunicam por meio de protocolos de comunicação.

O Protocolo HTTP é o mais usado para lidar com o trafego da internet, mas segundo Puthiyidam e Joseph (2024), mostram atrasos na transferência de mensagens que podem ser várias vezes maiores do que os observados no MQTT, o que ocorre principalmente por causa do tamanho dos cabeçalhos do protocolo HTTP.

Devido a necessidade de eficiência temporal, para este projeto, a comunicação utiliza o protocolo MQTT, que segundo Atmoko, Riantini e Hasin (2017), atua na camada de aplicação sobre o protocolo de transporte TCP/IP. No gráfico apresentado pela Figura 6, nota-se que o tempo de resposta apresentado pelo HTTP cresce proporcionalmente ao volume de dados (*data size*), enquanto o MQTT, praticamente se mantém constante no mesmo cenário.

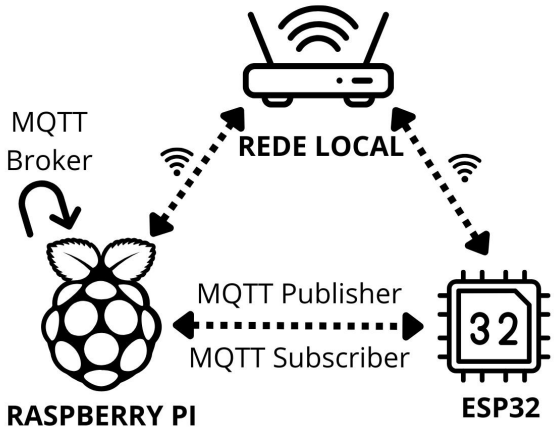
Figura 6 – Comparativo do Tempo de Resposta entre os protocolos HTTP e MQTT.



Fonte: Puthiyidam e Joseph (2024).

Segundo Dinculeana e Cheng (2019), o MQTT não necessita que clientes façam requisições constantes de atualizações, assim reduzindo o consumo de recursos. Para o protótipo detalhado na Figura 7, as funções de Publicador e Broker foram implementadas no Raspberry Pi 5, enquanto o Subscritor no ESP32. Conforme Atmoko, Riantini e Hasin (2017), esta arquitetura simplifica a rede local, ao utilizar overhead de 2 bytes para garantir que o erro chegue ao ESP32 com o mínimo de atraso.

Figura 7 – Arquitetura de controle distribuído e fluxo de comunicação via protocolo MQTT



Fonte: O Autor (2026).

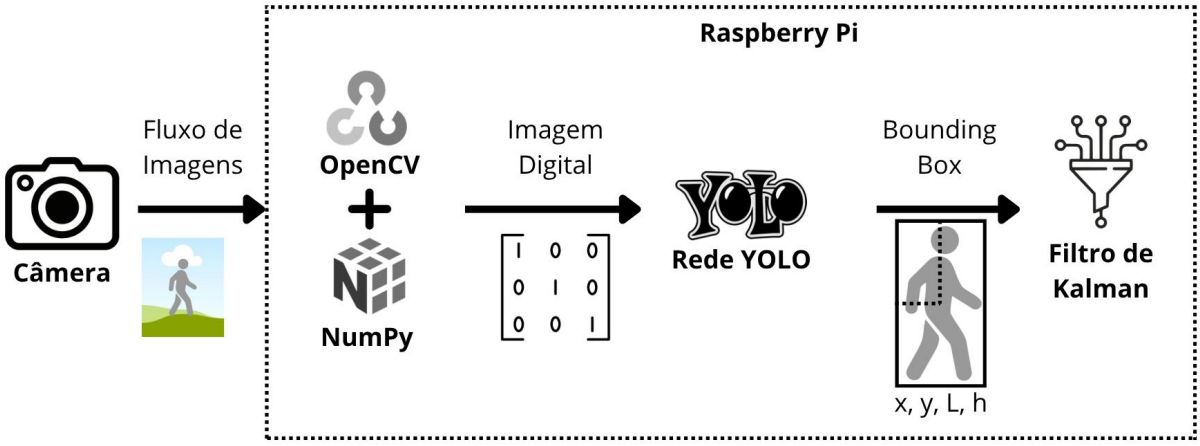
Para o ajuste de confiabilidade de entrega, o protocolo MQTT abrange níveis de *Quality of Services* (Qualidade de serviços, QoS). No nível QoS 0 (no máximo uma vez), a mensagem é enviada sem confirmação de recebimento. No nível QoS 1 (pelo menos uma vez), a mensagem é enviada e há espera de confirmação, se não confirmada envia de novo, havendo a possibilidade de receber a mensagem duas vezes. Já no QoS 2 (exatamente uma vez), utiliza-se de quatro etapas, de forma a garantir que a mensagem seja entregue e processada exatamente uma vez. Como pode ser analisado e confirmado por Puthiyidam e Joseph (2024), o nível QoS 0 apresenta a menor latência, tornando-o ideal para aplicações onde a velocidade é mais importante do que a garantia de entrega.

O QoS 0 é utilizado em sistemas de tempo real, já que o atraso na entrega dos dados torna essas informações obsoletas. Em contrapartida, o recebimento com menor atraso é crucial para que o algoritmo do controlador atenda os requisitos determinísticos.

2.2 Sistema de Visão Computacional

A estrutura lógica do sistema de percepção visual é detalhada no diagrama da Figura 8. O processo transita do domínio analógico para o digital na aquisição, por meio de um tratamento de dados em linguagem Python com suporte das bibliotecas OpenCV (BRADSKI, 2000) e NumPy. Esse fluxo permite que as informações atinjam as camadas da rede neural YOLO11 e filtragem estocástica (Filtro de Kalman). Este arranjo permite que o sistema interprete o ambiente e forneça os dados necessários para o controle de trajetória do protótipo.

Figura 8 – Diagrama do Sistema de Visão Computacional.



Fonte: O Autor (2026).

Essas características promovem a aquisição e manipulação de quadro a quadro de forma eficiente, para que o algoritmo execute com baixa latência. Segundo Barelli (2018), o OpenCV utiliza outra biblioteca, a NumPy, que oferece a estrutura matemática necessária para manipular essas matrizes de imagem de forma produtiva. Ao interpretar cada imagem como um *array* multidimensional, o NumPy permite que cálculos complexos sejam realizados com alta performance. Para Howse (2013), a NumPy é definida não apenas como uma ferramenta útil, mas como um requisito técnico para o funcionamento do OpenCV em Python, devido à sua capacidade de lidar com dados numéricos de forma eficiente.

O OpenCV utiliza uma abordagem moderna de integrar modelos de redes neurais profundas, do inglês *Deep Neural Networks* (DNN). Em vez de depender somente de algoritmos manuais para a extração de formas ou bordas, ele se integra a modelos como o YOLO11. Essa integração permite identificar e realizar a extração de características de modo autônomo, de propriedades como forma, cor e textura. Assim, proporciona a detecção mais eficiente e a tomada de decisão mais precisa para o sistema de controle.

A visão do sistema foi realizada por meio da implementação de uma DNN. O diferencial deste campo de estudo não é apenas realizar o tratamento de imagem, mas também reconhecer objetos ou ações nos quadros analisados (BARELLI, 2018, p. 1). A visão computacional tem como alicerce o Processamento Digital de Imagens (PDI) para a estruturação dos dados que serão processados pelas camadas de inteligência.

O PDI baseia-se em um conjunto de técnicas que são aplicadas em imagens digitais, as quais no escopo deste trabalho, não têm como propósito a melhoria da qualidade visual para interpretação humana, mas sim a otimização do seu processamento por sistemas computacionais. Para Solomon (2015), o PDI tem seu funcionamento pela transformação de sinais multidimensionais, de forma a atenuar ruídos e realçar características antes da classificação.

A interação de sistemas autônomos com o ambiente e as tomadas de decisão dinâmicas conferem ao sistema o comportamento mais confiável. No escopo deste trabalho, o PDI é utilizado para uniformizar, redimensionar e realizar a conversão de cores, além de normalizar os dados dos quadros capturados pela câmera, otimizando o processo de rastreamento do alvo pela visão computacional.

Logo, a sequência lógica da visão computacional possui, em seu processo, desde a aquisição das imagens até a interpretação dos resultados (AKBAR et al., 2024). A Figura 9 ilustra as quatro etapas principais da sequencia lógica de um algoritmo de visão computacional genérico.

Figura 9 – Etapas do algoritmo de visão computacional.

Fonte: O Autor (2026)

Contextualizando as etapas supracitadas, o processo começa com a aquisição das imagens por meio de uma câmera, um quadro (*frame*) recebido pela Raspberry Pi e pré-processadas por uma série de modificações, como a normalização, para garantir a uniformidade dos dados. Na extração de características, o algoritmo deixa de processar a imagem apenas como um conjunto de *pixels* e passa a identificar características estruturais, ou seja, padrões que resultam em dados computacionalmente relevantes, como forma, cor, textura e posição de bordas. Na análise e classificação, as informações são processadas por um modelo de rede neural, cujo objetivo é classificar os objetos encontrados. Por fim, na interpretação dos resultados, os dados gerados pela rede neural (classe, ID, caixa delimitadora e confiança) são convertidos em informações relevantes para o mapeamento do protótipo.

2.2.1 Aquisição e Pré-processamento de Imagens

Diferente da visão humana, que analisa imagens de forma analógica, para o PDI, imagens são dados que as máquinas interpretam como matrizes de dados numéricos (sinais discretos). Quando um sinal analógico é captado e discretizado, de acordo com Barelli (2018), é transformado de um sinal analógico de infinitos pontos, para um sinal discreto com finitos pontos, assim a resolução é dada a partir de dois processos, o de amostragem e de quantização. Na amostragem, o número de pontos espaciais é limitado a uma grade finita, enquanto na quantização os valores de intensidade (brilho) são limitados para cada *pixel*.

No âmbito matemático, como apresentado por Queiroz e Gomes (2006), o processo de amostragem transforma um sinal analógico representado pela função $f(x, y)$ em uma matriz M, com dimensões $H \times W$ (Largura x Altura, 640 x 480 *pixels*), no qual cada elemento representa um pixel na posição (i, j). Essa Matriz é construída da forma mostrada pela Equação (2.1):

$$M_{H\times W} = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,W-1) \\ f(1,0) & f(1,1) & \cdots & f(1,W-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(H-1,0) & f(H-1,1) & \cdots & f(H-1,W-1) \end{bmatrix} \tag{2.1}$$

Já no processo de quantização, conforme Queiroz e Gomes (2006), o brilho é estimado na escala de cinza do sinal analógico no ponto amostrado para um valor finito. Esse Valor é convencionalmente dado por uma potência de 2, indo de 0 a $L-1$, de acordo com a profundidade de bits adotada. Isto posto, o número de bits (B) que representa uma imagem na escala de cinza é dado pela Equação (2.2):

$$B = H \times W \times l \tag{2.2}$$

onde l representa a profundidade de bits e $L = 2^l$ define a quantidade de tons de cinza disponíveis para cada *pixel*.

2.2.2 Detecção de Objetos

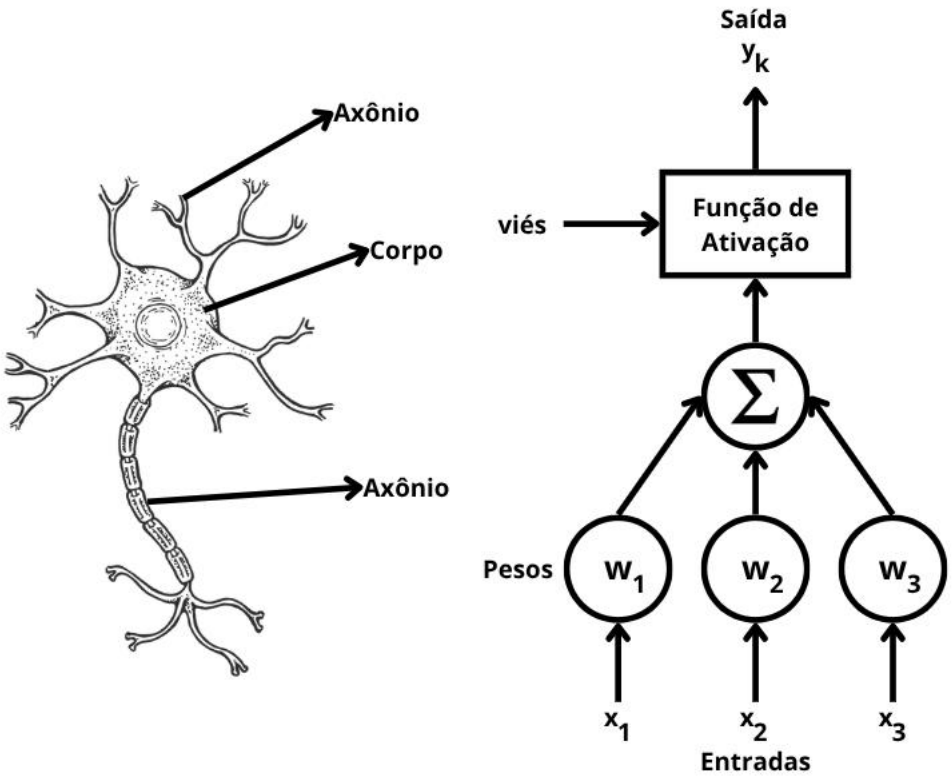
Nesta seção, o objetivo é apresentar o processamento de imagens com fins a determinação os parâmetros do alvo em relação ao sistema da câmera, que será implementado com a rede YOLOv11.

As Redes Neurais Artificiais (RNAs) foram projetadas de maneira a mimetizar o comportamento de uma rede neural biológica. Redes neurais biológicas que segundo Fleck et al. (2016), possuem um extenso processamento paralelo e a informação depende da rede como um todo, e não somente de um conjunto de neurônios.

Em concordância, esta analogia também foi usada por Haykin (2001), onde redes neurais artificiais (RNAs) são algoritmos que mimetizam o comportamento cerebral ao resolver tarefas particulares, através de conexões massivas entre células, de processamento simples, em que são denominadas de neurônios.

Assim, o cérebro atua, com sua rede de bilhões de neurônios, como uma função que mapeia estímulos captados por um conjunto de receptores, os quais alimentam uma rede neural, que analisa o sinal e entrega a resposta para a ação dos atuadores. Essa analogia entre o neurônio biológico e artificial, não é só metafísica, como física também, o neurônio biológico possui dendrito, soma e axônio, que representam respectivamente, a entrada, a função de ativação e a saída do neurônio artificial. Esta comparação pode ser observada na Figura 10 (CNNs e YOLO11).

Figura 10 – Comparação entre neurônios biológicos e artificiais.



Fonte: O Autor (2026)

Matematicamente, o neurônio artificial calcula a saída através do processamento de estímulos por meio da soma ponderada das entradas e da função de ativação, conforme a Equação (2.3).

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.3}$$

Onde x_i são as entradas, w_i os pesos sinápticos, b é o viés (bias), φ é a função de ativação e y a saída do sistema.

A partir da estrutura básica das RNAs, surgem as CNNs, que têm como foco o processamento de dados visuais, como imagens ou vídeos. De acordo com Aloysius e Geetha (2017), as CNNs foram baseadas no funcionamento da percepção visual, na qual os sinais gerados pelos olhos são processados em diversas camadas de neurônios, criando uma abstração do que está sendo observado.

A operação de convolução bidimensional, em que se baseia a extração de características em CNNs, consiste no produto escalar entre um núcleo de pesos (w) e a vizinhança de cada *pixel* da imagem de entrada (x), conforme (HAYKIN, 2001):

$$G(x, y) = (x * w)(x, y) = \sum_u \sum_v w(u, v) \cdot x(x - u, y - v) \tag{2.4}$$

Onde $G(x, y)$ é o valor do *pixel* resultante no mapa de características na posição (x, y); x é a matriz da imagem de entrada e w é o *kernel* bidimensional que contém os pesos sinápticos organizados espacialmente; já u e v são índices que percorrem as dimensões do filtro w.

Para compor a saída do neurônio convolucional, a operação linear descrita na equação 2.4 deve ser somada a um viés (b) e depois multiplicada pela função de ativação (φ). Como mostrado na Equação (2.5):

$$G(x, y) = \varphi \left(\sum_u \sum_v w(u, v) \cdot x(x - u, y - v) + b \right) \tag{2.5}$$

A comparação entre o neurônio RNA e CNN pode ser observado na Tabela 2.

Tabela 2 – Comparação entre RNA e CNN

Função	**RNA**	**CNN**
Entrada	x	$x(x - u, y - v)$
Pesos	w	$w(u, v)$
Saída	y	$G(x, y)$
Ativação	φ	φ (ReLU)

Fonte: O autor.

A função de ativação mais utilizada nas CNNs é a ReLu, que, segundo Alom et al. (2018), Aloysius e Geetha (2017) possui a capacidade de mitigar o problema da dissipação do gradiente (*vanishing gradient*), uma limitação inerente a funções não lineares clássicas, como a sigmoide e a tangente hiperbólica, durante o treinamento de redes profundas.

Segundo Jiang et al. (2022) e Kotthapalli, Ravipati e Bhatia (2025), a rede YOLO, desde sua concepção, representou um marco histórico na Visão Computacional ao tratar a detecção de objetos como um problema de regressão de um único estágio. Até então,

os sistemas baseados em DL eram de dois estágios, que geravam propostas de regiões de forma sequencial antes da classificação, tornando-os de alto custo computacional e inadequados para aplicações sensíveis à latência. A YOLO redefiniu esse cenário ao processar as classes e as coordenadas das caixas delimitadoras diretamente da imagem completa em uma única passagem pela rede, estabelecendo as bases para a detecção fluida em tempo hábil.

A arquitetura da YOLO, baseia-se na divisão da imagem de entrada em uma grade espacial. Cada célula dessa grade atua de forma autônoma e é responsável por prever objetos cujo centro geométrico está dentro de seus limites. Em uma única iteração pela rede neural, o algoritmo prevê simultaneamente as coordenadas espaciais das caixas delimitadoras, os escores de confiança e as classificações de classe (TERVEN; CóRDOVA-ESPARZA; ROMERO-GONZáLEZ, 2023; KOTTHAPALLI; RAVIPATI; BHATIA, 2025). Desde a sua concepção, o framework passou por um intenso ciclo de desenvolvimento estrutural para otimizar continuamente o balanço entre velocidade e precisão.

Criado pela Ultralytics em 2024, a YOLO11 foi projetada para melhorar o desempenho na detecção de objetos em tempo real. E segundo Kotthapalli, Ravipati e Bhatia (2025), tem um maior equilibro entre velocidade e precisão, devido ao aprimoramento da extração de características e da eficiência computacional.

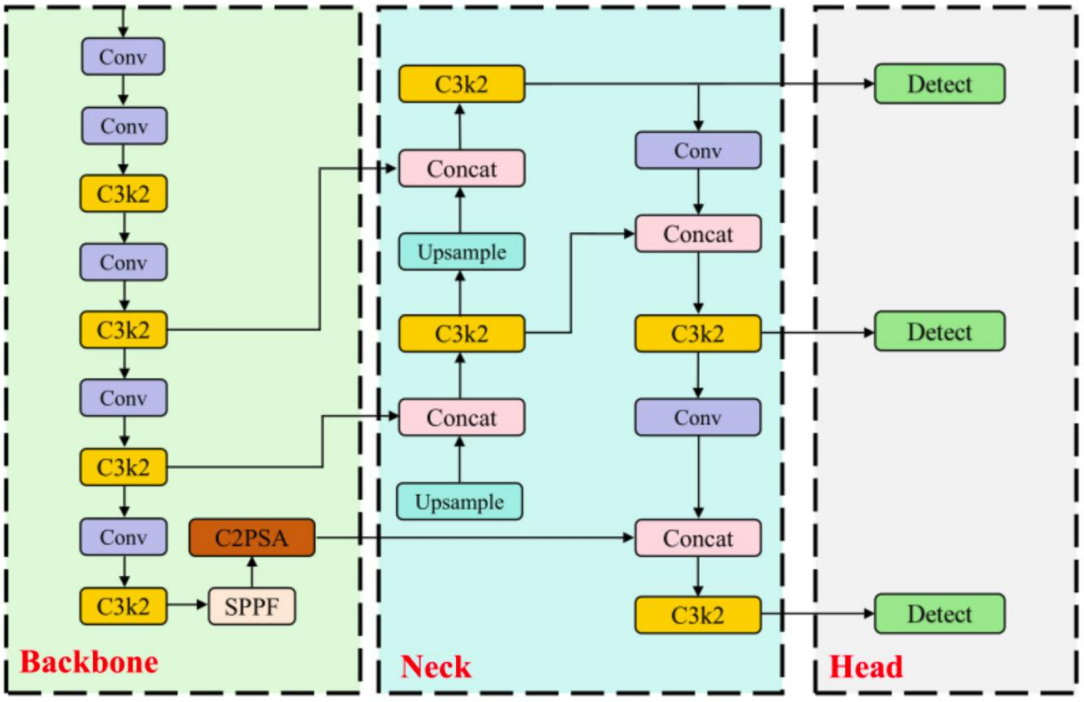
A sua estrutura interna é segmentada em três componentes primordiais: *Backbone*, *Neck* e *Head*. O *Backbone* atuando como extrator de características, transformando *pixels* brutos em mapa de características. *Neck* atuando entre extrator e preditor, fundindo características, de modo a combinar mapas de tamanhos diferentes para a rede conseguir detectar um mesmo objeto independente de estar longe ou perto. já o *Head* é o preditor e na YOLO11 é desacoplado, o que segmenta probabilidade de classe com a regressão de coordenadas. A arquitetura completa desta rede pode ser observado na Figura 11 (Arquitetura YOLO11).

Na arquitetura da rede YOLO11, houve mudanças em seus blocos fundamentais. No *Backbone*, o módulo C2f, foi substituído por C3k2 (*Cross Stage Partial with Kernel Size 2*). Onde Khanam e Hussain (2024) afirma que este novo bloco usa convoluções menores e otimizadas, implicando um processamento mais ágil e eficiente, sem prejudicar a extração de características.

Introduzido na rede YOLO11, logo depois do bloco de agrupamento SPPF (*Spatial Pyramid Pooling Fast*), o módulo de atenção espacial C2PSA (*Cross-Stage Partial Spatial Attention*), foi mencionado por Alif (2024), como sendo essencial para a detecção de objetos menores, ou que tenham partes ocultadas, isso ocorre devido ao seu maior foco em regiões críticas da imagem.

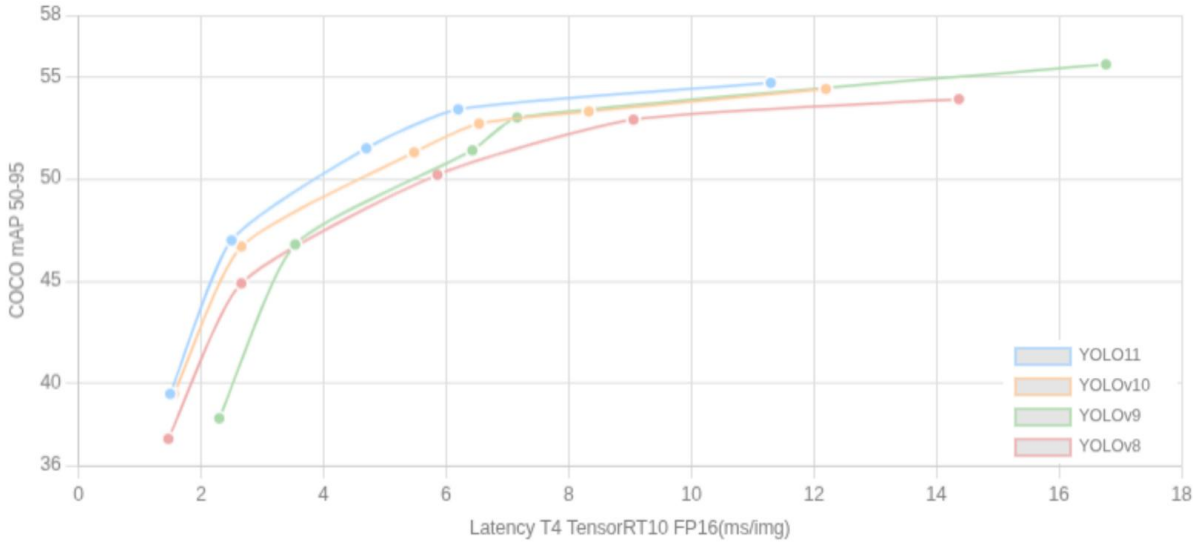
A rede YOLO11 para He et al. (2025b), consegue discernir e enfatizar melhor as características relevantes em ambientes complexos. Isso também se deve a menor latência e maior precisão média (mAP) apresentados em comparação com seus antecessores, YOLOv8, YOLOv9 e YOLOv10, o que pode ser analisado na Figura 12.

Figura 11 – Arquitetura da rede YOLO11.



Fonte: He et al. (2025b).

Figura 12 – Comparação de precisão (mAP) e latência entre as versões do modelo YOLO.



Fonte: Jocher e Qiu (2024)

A YOLO11n (Nano) conforme He et al. (2025a) é um modelo especializado e otimizado para ambientes com recursos limitados. Para este projeto, foi utilizada a versão do modelo pré-treinado no conjunto de dados COCO (*Common Objects in Context*). A quantificação deste desempenho foi descrita por Kotthapalli, Ravipati e Bhatia (2025), que destaca a precisão média (mAP) de 39,5 no conjunto de dados citado, com uma latência de 1,5 ms.

A variante nano possui 2,6 milhões de parâmetros e tem a capacidade de realizar 6,5 bilhões de FLOPs (*Floating Point Operations*), tornando-a ideal para dispositivos de borda. Conforme Lee et al. (2025), esta é a versão que exige menor quantidade de FLOPs e de parâmetros, o que provoca menor consumo, respectivamente, de processamento e de memória RAM. Para Khanam e Hussain (2024), a variante nano apresenta melhorias impressionantes de velocidade e eficiência em relação aos seus antecessores, tornando-a ideal para aplicações com restrições temporais. Segundo Rasheed e Zarkoosh (2025), a utilização do YOLO11n permite focar na detecção de objetos específicos com alta eficiência. A comparação de desempenho pode ser observada na Tabela 3.

Tabela 3 – Comparativo de desempenho entre as versões da arquitetura YOLO11.

Modelo	**Tam. (px)**	**mAPval**	**CPU (ms)**	**T4 (ms)**	**Parâm. (M)**	**FLOPs (B)**
YOLO11n	640	39,5	56,1 ± 0,8	1,5 ± 0,0	2,6	6,5
YOLO11s	640	47,0	90,0 ± 1,2	2,5 ± 0,0	9,4	21,5
YOLO11m	640	51,5	183,2 ± 2,0	4,7 ± 0,1	20,1	68,0
YOLO11l	640	53,4	238,6 ± 1,4	6,2 ± 0,1	25,3	86,9
YOLO11x	640	54,7	462,8 ± 6,7	11,3 ± 0,2	56,9	194,9

Fonte: (JOCHER; QIU, 2024).

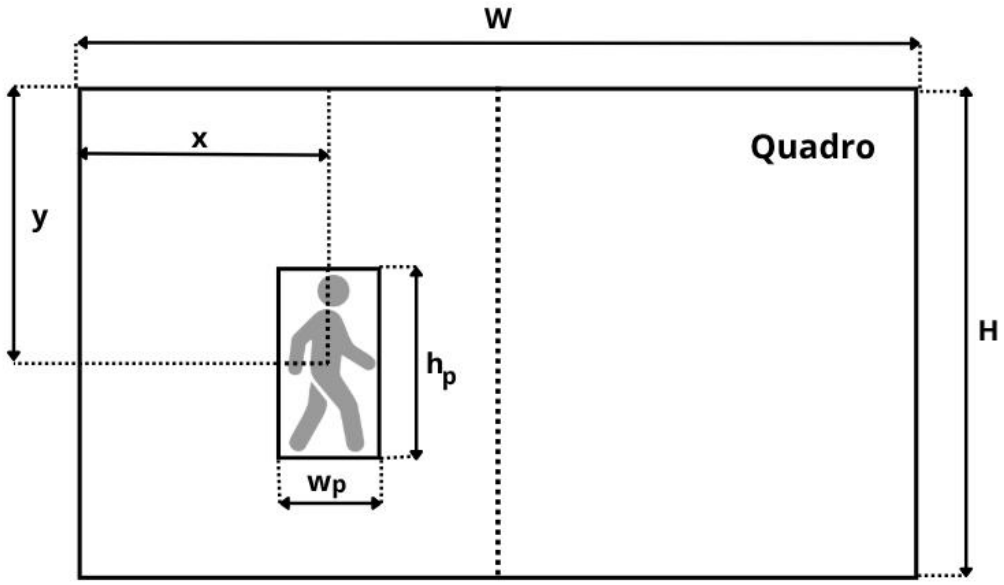
Para avaliar o processo de treinamento e a convergência da rede, analisa-se o comportamento da função de perda (*loss function*), que mede quanto a previsão do modelo erra em relação ao valor real. Durante a etapa de treinamento, o modelo aprimora sua precisão minimizando, simultaneamente, três componentes distintos da função de perda (LEE et al., 2025; HE et al., 2025a):

- **Perda de Regressão da Caixa:** Mede o erro na precisão espacial das *bounding boxes* previstas pelo modelo em comparação com as caixas reais da base de dados (*ground-truth*). Seu objetivo é assegurar a localização exata do objeto detectado, penalizando caixas que estejam mal ajustadas ao alvo.
- **Perda de Classificação:** Avalia a exatidão das categorias previstas pela rede, minimizando a discrepância entre a classe inferida e a classe real do objeto. Essa métrica indica o grau de certeza de que o alvo detectado pertence, de fato, à categoria correta.
- **Perda Focal de Distribuição:** Atua no refinamento das bordas das caixas delimitadoras, modelando suas coordenadas como uma distribuição de probabilidade. Essa função permite à rede refinar detalhes sutis das margens, melhorando a precisão em casos de bordas pouco claras, sobreposições ou alvos muito pequenos.

Após o processamento da CNN, o sistema de detecção gera uma caixa delimitadora no alvo identificado. Esta representação geométrica retangular, ilustrada na Figura 13, é

fundamental para a estratégia de controle, pois é ela que fornece as coordenadas centrais (x,y), a Largura (w_p) e Altura (h_p). Das quais calcula-se a distância do centro da tela ao centro do objeto Δx.

Figura 13 – Definição da caixa delimitadora (*bounding box*) no sistema de coordenadas da imagem.



Fonte: O Autor (2026)

O parâmetro utilizado para o controle do alinhamento do protótipo foi a posição horizontal do centro da caixa delimitadora (x), pois quanto mais próximo do centro do quadro capturado, mais próximo o protótipo fica do alinhamento em relação ao alvo, com maior desvio nas extremidades da imagem.

Quanto à distancia relativa do protótipo ao alvo, são três as possibilidades, largura, altura e área da caixa delimitadora. Ao serem analisados alguns pontos como a estabilidade e constância da média em diferentes situações, foram preteridas a largura e a área. Isso decorre pois qualquer gesticulação ou posição de braços tende a aumentar a largura e consequentemente a área. Assim, visto que a altura mantém valores próximos na maioria dos cenários, exceto em casos de inclinação ou movimentação de braços, como levantar os braços acima da cabeça. Portanto, a fim de mitigar erros que não advêm da movimentação em relação ao protótipo, foi selecionada a Altura da caixa delimitadora (h_p).

Além do mais, a altura (h_p) da caixa delimitadora é inversamente proporcional a distância, veja a Equação (2.6), em exemplo, quanto mais próxima uma pessoa esta da câmera maior é a sua altura em *pixels*, todavia, quanto menor for a pessoa na câmera mais longe ela está. Porém, não pode-se afirmar que seja linear, sendo suficiente essa relação para a continuidade deste trabalho.

$$h \propto \frac{1}{d} \tag{2.6}$$

Dessa forma foram utilizadas a normalização de x_p e h_p, ressaltando a aplicação do filtro de Kalman nessas variáveis, que foram normalizadas, conforme apresentado nas Equações (2.7) e (2.8):

$$x_n = (x - W/2)/W \tag{2.7}$$

$$h_n = h_p/H \tag{2.8}$$

2.2.3 Estimação de Estados e Filtro de Kalman

O Filtro de Kalman foi introduzido em 1960 pelo matemático húngaro Rudolf Emil Kalman como uma solução recursiva para o problema de filtragem linear de dados discretos (KALMAN, 1960). Desde a sua proposição, o algoritmo tornou-se uma das ferramentas matemáticas mais importantes para a estimativa ótima de estados na ciência e na engenharia, tendo uma de suas primeiras grandes aplicações práticas no projeto Apollo, onde foi utilizado para estimar a trajetória e mitigar erros de sensores (WANG et al., 2023). Diferente de métodos estatísticos anteriores, o Filtro de Kalman introduziu a modelagem no espaço de estados, o que eliminou a necessidade de armazenar todo o histórico de medições, pois o estado estimado e sua covariância contêm toda a informação estatística necessária (FADALI, 2024; BROWN; HWANG, 1997; BAR-SHALOM; LI; KIRUBARAJAN, 2002).

Embora o modelo original exija que os sistemas sejam estritamente lineares e os ruídos gaussianos, o contínuo desenvolvimento teórico expandiu sua aplicação para cenários complexos. Para lidar com sistemas não lineares, desenvolveu-se, por exemplo, o Filtro de Kalman Estendido (EKF), que lineariza o modelo por meio da aproximação de primeira ordem da série de Taylor (matriz Jacobiana), e posteriormente, o Filtro de Kalman *Unscented* (UKF), que utiliza a Transformada *Unscented* para aproximar distribuições de probabilidade sem a necessidade de matrizes jacobianas complexas (FENG et al., 2023; WANG et al., 2023; FADALI, 2024).

Neste projeto, o filtro de Kalman atua como um estimador de estados que minimiza o erro médio quadrático em sistemas dinâmicos lineares com ruídos. Segundo Kalman (1960), o filtro incorpora um modelo físico do sistema, o que possibilita a predição de estados futuros, bem como o gerenciamento otimizado das incertezas de medições.

O funcionamento do Filtro de Kalman opera dentro da estrutura de modelos de espaço de estados, separando o processo recursivo em duas etapas principais: a predição (a *priori*) e a atualização (a *posteriori*) (KALMAN, 1960; KIM; BANG, 2018; BROWN; HWANG, 1997; FADALI, 2024; BAR-SHALOM; LI; KIRUBARAJAN, 2002). Na etapa de predição, o filtro utiliza as equações da cinemática do sistema para prever o estado atual e a matriz de covariância do erro antes que uma nova medição do sensor seja feita. Em seguida, na etapa

de atualização, o algoritmo calcula o resíduo (a diferença entre a medição observada e a prevista) e o multiplica pelo Ganho de Kalman. Esse ganho atua como um peso matemático que equilibra a confiança entre a predição do modelo físico e a leitura do sensor, gerando a estimativa linear ótima no sentido de mínimo erro quadrático médio. O filtro fornece estimativas ótimas quando os modelos são lineares e os ruídos de processo e medição são brancos, independentes e gaussianos (FADALI, 2024; FENG et al., 2023; BROWN; HWANG, 1997).

Para o contexto deste projeto, o filtro foi aplicado no rastreamento. O espaço de estados no instante k é dado pela posição (p_k) e pela velocidade (v_k) do centro do alvo no quadro capturado, que definem o vetor x_k, conforme apresentado na Equação (2.9). Embora o detector YOLO11 forneça a medição direta da posição, a velocidade, por sua vez, é estimada pelo próprio filtro de Kalman. A progressão desse estado no tempo é regida pela equação de transição, na qual a matriz F projeta o estado anterior para o instante atual com base no intervalo de amostragem (Δt), somado ao ruído do processo w_k. A Equação (2.10) exemplifica esta operação:

$$x_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} \tag{2.9}$$

$$x_k = Fx_{k-1} + w_k, \quad \text{sendo} \tag{2.10}$$

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{2.11}$$

onde $w_k \sim \mathcal{N}(0, Q)$ representa o ruído do processo, modelado como uma variável aleatória Gaussiana de média nula e covariância Q (BAR-SHALOM; LI; KIRUBARAJAN, 2002; BROWN; HWANG, 1997; FADALI, 2024).

Nessa formulação, a primeira linha calcula a posição atual como sendo a posição no instante anterior somada ao deslocamento causado pela velocidade do último instante no intervalo de amostragem, enquanto a segunda linha mantém a velocidade, expressa na Equação (2.12):

$$\begin{cases} p_k = p_{k-1} + \Delta t \; v_{k-1} \\ v_k = v_{k-1} \end{cases} \tag{2.12}$$

A medição z_k fornecida pelo YOLO11n é mapeada pela matriz H. Como o sensor de visão entrega apenas a posição, H isola essa variável do vetor de estados:

$$z_k = Hx_k + r_k, \quad \text{sendo} \tag{2.13}$$

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.14}$$

e $r_k \sim \mathcal{N}(0, R)$ representa o ruído da medição associado ao detector YOLO11n, também assumido Gaussiano de média nula e covariância R.

A variância da posição estimada da caixa delimitadora medida da YOLO11n é expressa pela equação:

$$R = \left[\sigma_z^2\right]. \tag{2.15}$$

Para representar as incertezas inerentes ao sistema, adotou-se o modelo de aceleração com ruído branco. Nesse cenário, a matriz de covariância do ruído do processo Q mensura as variações que fogem à previsão do modelo matemático, tais como mudanças bruscas de direção e acelerações do alvo. Para o modelo de velocidade constante, a formulação de Q assume a forma apresentada por:

$$Q = \sigma_a^2 \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix} \tag{2.16}$$

sendo σ_a^2 a variância da aceleração, assumida como uma variável aleatória Gaussiana de média nula. Esta matriz trata-se de um parâmetro de calibração essencial, ao configurá-lo com valores maiores possibilita o acompanhamento de alvos com movimentos erráticos, enquanto o uso de valores menores produz uma trajetória com variações reduzidas.

Um resumo dos parâmetros associados à formulação do Filtro de Kalman de tempo discreto adotado nesta implementação estão listados na Tabela 4.

Tabela 4 – Descrição das Variáveis de Estado e Matrizes do Filtro.

Símbolo	**Nome Técnico**	**Função no Projeto**
x	Vetor de Estado	Armazena a posição (p) e a velocidade (v) do alvo.
F	Matriz de Transição	Aplica a física do movimento ($p_k = p_{k-1} + v_{k-1} \cdot \Delta t$).
H	Matriz de Medição	Mapeia o estado para a observação (extrai a posição do YOLO11n).
Q	Ruído do Processo	Representa a incerteza no modelo físico (movimentos bruscos).
R	Ruído da Medição	Representa a confiança na detecção do sensor (YOLO11n).
P	Covariância do Erro	Quantifica a incerteza da estimativa atual do filtro.
K	Ganho de Kalman	Peso que decide entre confiar na física ou na câmera.

Fonte: O Autor (2026).

Na etapa de previsão, o filtro usa a matriz F (matriz de transição) para estimar o estado

do sistema, dado por:

$$\hat{x}_k^- = F\hat{x}_{k-1}^+ \tag{2.17}$$

e a matriz de covariância P:

$$P_k^- = FP_{k-1}F^T + Q \tag{2.18}$$

onde $\hat{x}_k^-$ expressa a estimativa *a priori* do estado no instante k, e P_k^- corresponde à covariância *a priori* do erro de estimação associada a essa predição.

Enquanto na etapa de atualização, ao receber uma nova detecção da rede neural (YOLO11n), o filtro calcula a diferença entre o valor observado e o valor estimado. Ao multiplicar esta diferença pelo ganho de Kalman (K_k) para o ajuste da estimativa, reduzindo assim a incerteza e corrigindo a trajetória. O Ganho de Kalman é calculado conforme a Equação (2.19):

$$K_k = P_k^- H^T \left(HP_k^- H^T + R\right)^{-1} \tag{2.19}$$

A atualização do estado estimado é calculada pela Equação (2.20):

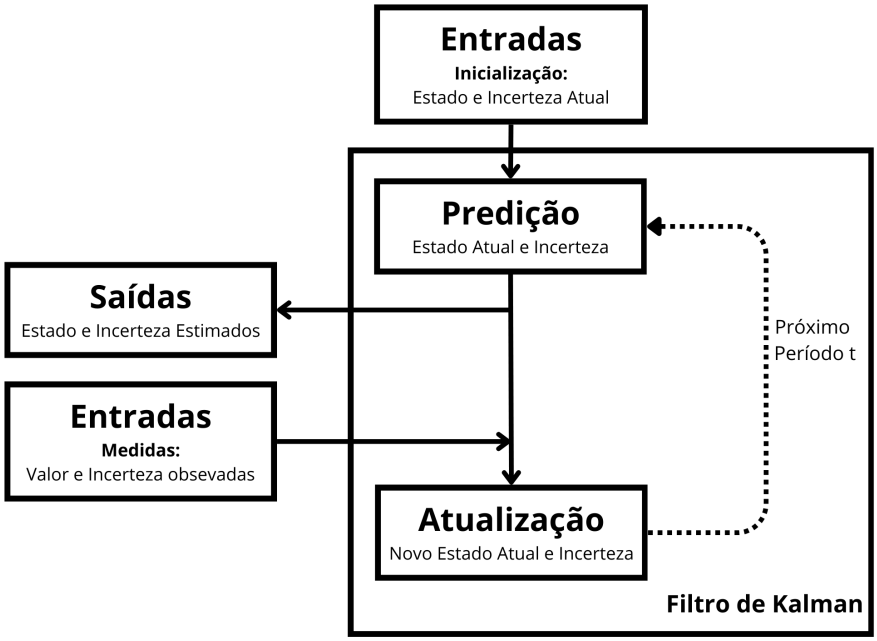
$$\hat{x}_k^+ = \hat{x}_k^- + K_k \left(z_k - H\hat{x}_k^-\right) \tag{2.20}$$

Por fim, a matriz de covariância do erro é atualizada conforme a Equação (2.21):

$$P_k = (I - K_k H)P_k^- \tag{2.21}$$

O Filtro de Kalman opera através de um ciclo recursivo que segundo Feng et al. (2023), minimiza o traço da matriz de covariância do erro, garantindo que o estado estimado seja o valor mais provável, dada a sequência de dados observados e as incertezas do modelo. A arquitetura do filtro de Kalman pode ser observada na Figura 14 (O Algoritmo de Predição e Correção).

Figura 14 – Fluxograma Filtro de Kalman.



Fonte: O Autor (2026).

Assim, desempenhando um papel importante para a autonomia do projeto, especialmente na capacidade de tratar oclusões do alvo. Em caso de oclusão, Kim e Bang (2018) cita que o filtro tem a capacidade de pular a etapa de correção e utilizar somente o resultado da etapa de predição. Assim, o filtro concede ao protótipo a capacidade de interpolação para estimar os estados em instantes de tempo quando não há resposta do YOLO11n.

A matriz de Covariância do Erro (P) é fundamental porque reflete a incerteza da estimativa atual do estado e é diretamente utilizada para derivar o Ganho de Kalman (K). Segundo Feng et al. (2023) o Ganho de Kalman é calculado para minimizar o traço dessa matriz de covariância P no passo de atualização.

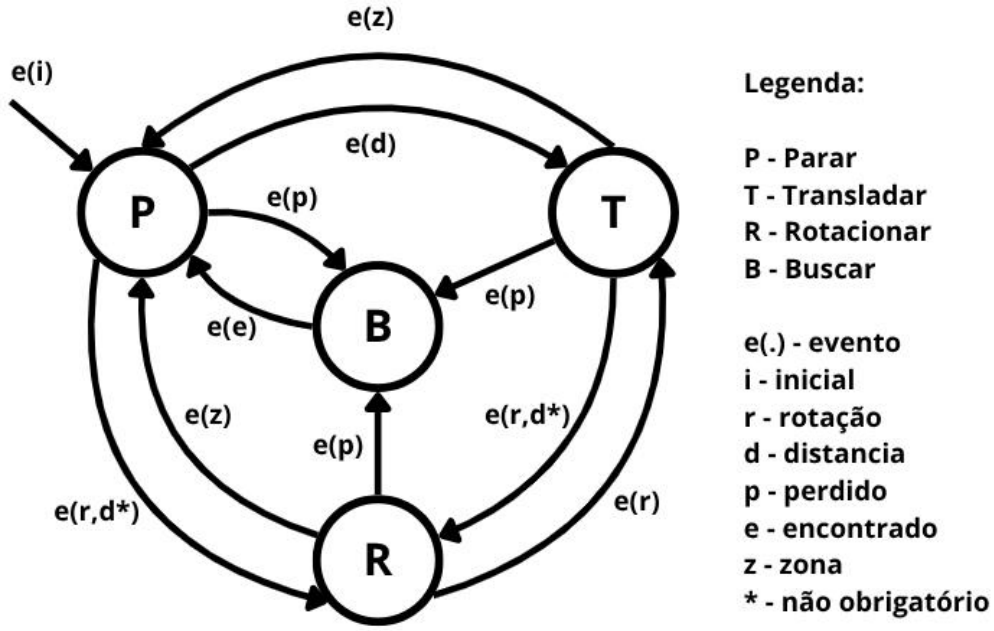
A importância do ganho K é ponderar o peso da nova medida para o sistema, se a incerteza for grande, o ganho aumenta, dando maior peso aos dados observados, já se o ruído de medição for alto, o filtro dá prioridade a estimativa realizada (KIM; BANG, 2018).

Este ajuste dinâmico proporciona ao filtro de Kalman uma performance superior à de filtros determinísticos em mudanças bruscas de movimento, pois filtros de média móvel introduzem atraso e não operam com um modelo de sistema dinâmico. Conforme Wang et al. (2023), o Filtro de Kalman possui estados que são atualizados dinamicamente com base nos movimentos do protótipo, para estimar mudanças bruscas de movimento e oclusões, de forma a suavizar o rastreio do alvo.

2.3 Estratégia de Controle

Nesta seção, é apresentada a fundamentação teórica que converte um erro captado pela percepção visual em movimento coordenado. A arquitetura de controle do protótipo é estruturada como uma FSM, implementada em linguagem *Python* na Raspberry Pi, cujo Diagrama de Transição de Estados é apresentado na Figura 15.

Figura 15 – Diagrama de Transição de Estados Para o Controle do Protótipo.



Fonte: O Autor (2026).

A arquitetura de controle proposta proporciona ao mecanismo a transição entre os comportamentos da FSM, com base no sistema de percepção. Os Estados de modos de operação são: Buscar (B), Rotacionar (R), Transladar (T) e Parar (P). Enquanto as transições, são condicionais lógicos, que proporcionam ao protótipo alternar entre os estados a cada iteração de modo a manter o controle determinístico do projeto.

O sistema desenvolvido realiza a utilização dos parâmetros adquiridos pelo sistema de visão computacional: posição (x_f), altura (h_f) e `busca`. O controle via máquina de estados utiliza-se desses parâmetros como métricas de transição de estados a cada ciclo, de forma que o protótipo mantenha estabilidade na orientação e distância em relação ao alvo. A exemplificação do modelo é ilustrada na Figura 15, cuja descrição é apresentada na Tabela 5.

Tabela 5 – Definição de Eventos do Sistema FSM

Símbolo	**Função**	**Descrição**
P	Estado Parar	Indica estar na zona morta dos dois eixos
R	Estado Rotacionar	Realiza os movimentos de alinhamento angular
T	Estado Transladar	Realiza os movimentos avançar e recuar
B	Estado Buscar	Realiza varredura pelo alvo
$e(\cdot)$	Função evento	Indica alteração no sistema
$e(i)$	Inicial	Início do sistema
$e(r)$	Rotação	x_f e/ou h_f
$e(d)$	Distância	h_f
$e(p)$	Perdido	busca
$e(e)$	Encontrado	busca
$e(z)$	Zona	x_f e h_f
$*$	Opcional	x_f e h_f

Fonte: O Autor (2026).

Além da dinâmica do controlador, outro desafio crítico da integração entre visão e atuação é a assincronia na realimentação. Enquanto o processamento de visão computacional da rede YOLO11n fornece medições a uma taxa de atualização reduzida, a atuação operação necessita de uma frequência maior. Para contornar os problemas de latência e medições com atrasos desiguais, o sistema utiliza a predição do Filtro de Kalman. Como supracitado na Seção 2.2.3, essa estrutura atua como um preditor (baseado no modelo) que preenche as lacunas de processamento da câmera, mantendo uma estimativa contínua e estável do estado do alvo, mesmo nos intervalos de tempo em que não há detecção visual válida.

2.3.1 Sistema de Tração Diferencial

O protótipo utiliza um sistema de tração diferencial muito usado em robótica móvel devido a sua mecânica simples e alta manobrabilidade. Conforme Siegwart Roland e Nourbakhsh (2004), a grande vantagem dessa configuração é seu controle: um robô de tração diferencial consegue controlar tanto sua taxa de mudança de orientação (rotacionar), quanto sua velocidade linear (avançar ou recuar) de forma direta, simplesmente manipulando as velocidades individuais de cada roda. O movimento é realizado em torno de um ponto, conhecido como Centro Instantâneo de Rotação (CIR). Este se localiza na linha que passa pelos dois eixos das rodas.

O comportamento cinemático de um sistema de tração diferencial opera da seguinte forma, quando cada roda gira com a mesma velocidade, mas em direções opostas, a velocidade linear frontal será zero e o robô apenas girará em torno de seu próprio eixo central (SIEGWART ROLAND E NOURBAKHSH, 2004), neste caso o CIR se localiza no centro do eixo das rodas. Já para o movimento de translação ocorre são aplicadas as mesmas velocidades e direção em ambas as rodas. Adicionalmente, existe um caso

especifico, no qual uma roda gira, enquanto a outra permanece totalmente estacionária, assim, o CIR coincide com o ponto de contato da roda estacionária com o solo, e o centro do robô se moverá instantaneamente com metade da velocidade da roda em movimento.

Segundo Siegwart Roland e Nourbakhsh (2004), este comportamento caracteriza um sistema com restrição não holonômica, pois permite que o protótipo realize correções angulares para manter o alinhamento com o alvo, mesmo sem transladar. Embora o modelo cinemático diferencial seja teoricamente simples, sua aplicação em sistemas reais enfrenta o desafio físico das não linearidades.

Para a implementação prática, os sinais resultantes da mixagem devem ter limitadores de saturação, cujo pressuposto é restringir variáveis do sistema de modo a não ultrapassarem o limite físico dos componentes, garantindo que o sistema opere dentro de uma faixa de estabilidade. A técnica de saturação restringe o sinal de controle ao intervalo operacional dos registradores do microcontrolador. Nesse trabalho, operado pela ponte H com PWM em uma resolução de 8 bits, esta operação pode ser descrita pela função de saturação dada por:

$$f_{sat}(u) = \begin{cases} 255, & \text{se } u > 255 \\ u, & \text{se } 0 \leq u \leq 255 \text{ , } \in \mathbb{N} \\ 0, & \text{se } u < 0 \end{cases} \tag{2.22}$$

Para componentes reais Ogata (2010), afirma que linearidade estrita é rara, sendo comum a ocorrência da chamada "zona morta"(Z_m), um intervalo de sinais de entrada que não produz qualquer alteração na saída.

Para regular a movimentação longitudinal no estado Transladar, estabelece-se um intervalo de referência, conforme a Equação (2.23):

$$T = \begin{cases} \text{Avançar}, & \text{se } h_f > h_{max} \\ \text{Recuar}, & \text{se } h_f < h_{min} \end{cases} \tag{2.23}$$

onde h_{max} representa a altura máxima que o alvo pode ficar, e h_{min} a altura mínima permitida. E h_f a altura estimada pelo filtro de Kalman.

Nesse contexto, o emprego desta faixa de histerese cria uma folga operacional que impede a comutação contínua dos motores. Essa faixa de tolerância visual mitiga o fenômeno de *chattering* (comutação rápida de controle), que seria inevitavelmente causado por ruídos pontuais na detecção dos *pixels* ou por oscilações sutis do alvo (DELAI; COELHO, 2012; OGATA, 2010). Para evitar esse efeito de *chattering*, no alinhamento do protótipo foi utilizado um erro mínimo para o inicio do estado Rotacionar.

Da mesma forma, foi implementada uma zona morta (D_b) para o alinhamento do protótipo, no estado Rotacionar, para o mesmo fim, expressa pela Equação (2.24).

$$R = \begin{cases} \text{Rotação Horária}, & \text{se } x_f > D_b \\ \text{Rotação Anti-Horária}, & \text{se } x_f < -D_b \end{cases} \tag{2.24}$$

onde x_f é o erro horizontal estimado pelo filtro de Kalman. Na rotação horária o protótipo rotaciona para a direita, enquanto na rotação anti-horária rotaciona para a esquerda.

Este limitador garante que erros pequenos não sejam processados pelo controlador. Portanto, é essencial mitigar os tremores (oscilações de alta frequência) em torno do ponto de repouso, garantindo que os atuadores não respondam a ruídos insignificantes provenientes da detecção visual. Segundo Ogata (2010), a implementação dessa prática baseia-se na necessidade de prevenir operações excessivamente frequentes do mecanismo de controle.

Devido à não linearidade e às variações de fabricação, como tolerâncias mecânicas, notou-se a necessidade de um fator de correção em um dos motores para que a mesma tensão aplicada produza movimento simétrico. Essa relação foi expressa na Equação (2.25).

$$v_{esq} = v_{dir} \cdot \eta \tag{2.25}$$

onde η representa o coeficiente de compensação estática.

3 METODOLOGIA

Este capitulo abrange os procedimentos metodológicos adotados para o desenvolvimento do robô seguidor. Inicialmente, descreve-se a infraestrutura do protótipo, a especificação dos componentes e a arquitetura de comunicação entre eles via protocolo MQTT. Em seguida, é apresentado o sistema de visão computacional e a implementação do filtro de Kalman. Por fim, explora-se a implementação do controle, bem como a metodologia de verificação e validação experimental para a análise de desempenho. Os algoritmos de visão, atuação e a configuração do ambiente de trabalho, foram mencionados neste capítulo e estão disponíveis no repositório do projeto (ALMEIDA, 2026).

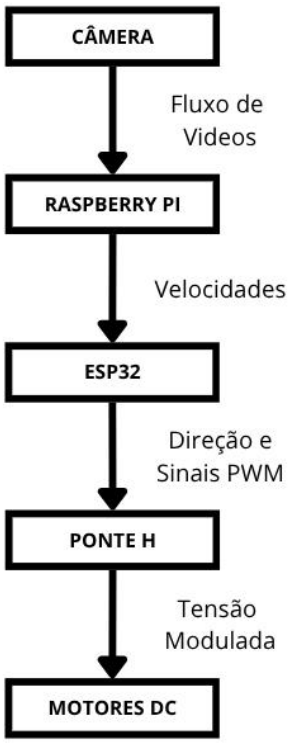
3.1 Infraestrutura de Hardware e Comunicação

Para a concepção do projeto, foram definidos componentes com a capacidade de atender aos requisitos impostos de desenvolver um software de visão computacional que processa imagens, realiza detecção (YOLO11), filtra os dados de saída (Filtro de Kalman) e rastreia objetos; uma arquitetura de rede que realiza comunicação de alta confiabilidade (protocolo MQTT); um sistema de controle para a movimentação do protótipo (FSM); e um circuito eletrônico (Ponte H e atuadores). A Figura 16 mostra o diagrama de blocos funcional simplificado do robô seguidor, cujas justificativas técnicas para a sua solução serão apresentadas a seguir.

Conforme ilustrado na Figura 16, o fluxo operacional inicia-se com a captura de imagens pela câmera. Esse fluxo de vídeo (*stream*) é processado quadro a quadro pela Raspberry Pi, na qual o algoritmo de visão computacional calcula o estado e o desvio horizontal do objeto, além da distância entre os centros do objeto e do quadro. Por meio do protocolo MQTT, o Raspberry Pi atua como Publicador (*Publisher*) e *Broker*, e envia a mensagem para o Subscritor (*Subscriber*), ESP32. Esta última, por sua vez, utiliza essa informação para modular a potência através do erro, enviando-a aos motores via Ponte H.

Para melhor compreensão da arquitetura deste projeto, pode-se fazer uma analogia ao corpo humano, na qual o sistema funciona da seguinte forma: a câmera atua como os olhos (sensores); a Raspberry Pi atua como o cérebro pensante (processamento); o MQTT atua como as sinapses (comunicação); a ESP32 atua como o sistema nervoso periférico (controle e transmissão de comandos), e a Ponte H e os motores atuam como os músculos (atuadores) que reagem ao meio externo.

Figura 16 – Diagrama funcional do protótipo.

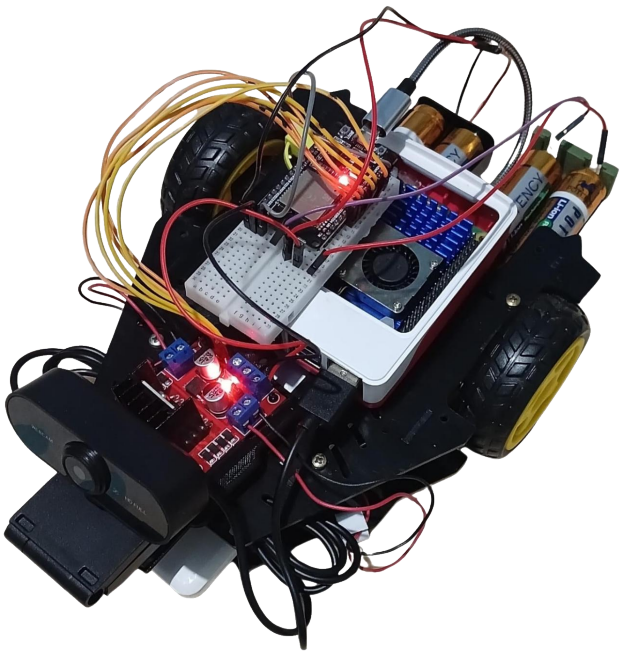


Fonte: O Autor (2026).

3.1.1 Especificação e Justificativa de Hardware

Inicialmente, estabeleceram-se os requisitos do projeto: sensor de imagem, unidade de processamento de alto nível, microcontrolador, driver de potência, atuadores, estrutura mecânica e fontes de alimentação. Prioriza-se a relação entre custo e performance. A Imagem do protótipo montado é apresentada na Figura 17.

Figura 17 – Protótipo Finalizado.



Fonte: O Autor (2026).

Câmera: A escolha do sensor óptico foi uma *WebCam*, que pode ser observada na 18. Optou-se por este componente devido a sua interface nativa USB, que facilita a integração com o Raspberry Pi sem a necessidade de módulos adicionais. A sua resolução possui a densidade de *pixels* (640x480) necessária para que a detecção seja efetuada com precisão.

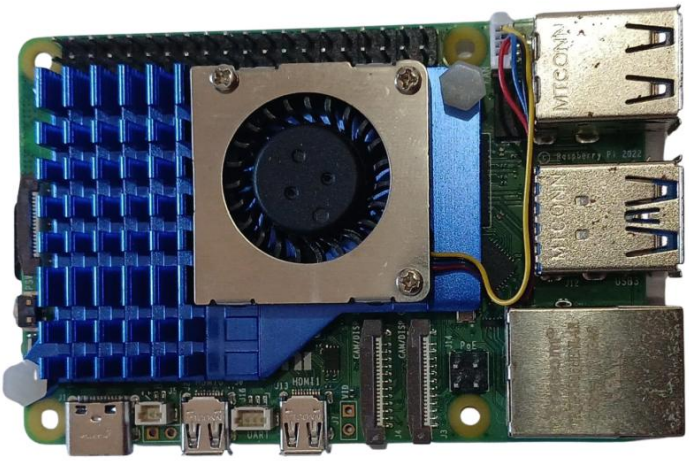
Figura 18 – Módulo de Câmera (640x480).

Fonte: O Autor (2026).

Raspberry Pi 5: O principal componente, a Raspberry Pi 5, Figura 19. Apesar de seu custo maior, atende às finalidades substanciais do protótipo ao executar o algoritmo

de rastreamento, realizar a comunicação via protocolo MQTT, hospedando o *Broker* e operando como Publicador. Com tais requisitos, foram analisadas suas versões e, dentre as opções, a preferência foi pela de 8GB de memória RAM que, devido à sua alta capacidade de processamento e memória RAM, é capaz de manter a estabilidade durante o processamento do fluxo de video, mitigando gargalos de memória.

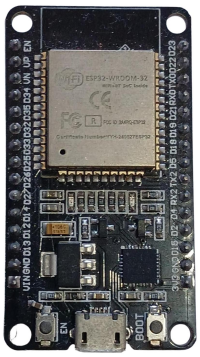
Figura 19 – Raspberry pi 5.



Fonte: O Autor (2026).

ESP32: apresentada na Figura 20, esta placa foi selecionada devido a sua arquitetura dual core e a sua capacidade de conectividade wi-fi nativa. Essas duas especificações tornam a Esp32 uma escolha adequada. Devido aos dois núcleos de processamento, a ESP32 pode estabelecer a conexão MQTT como Subscritor e, simultaneamente, executar o algoritmo de controle e o envio de sinais PWM para os motores de forma determinística.

Figura 20 – Esp32 30 pinos.

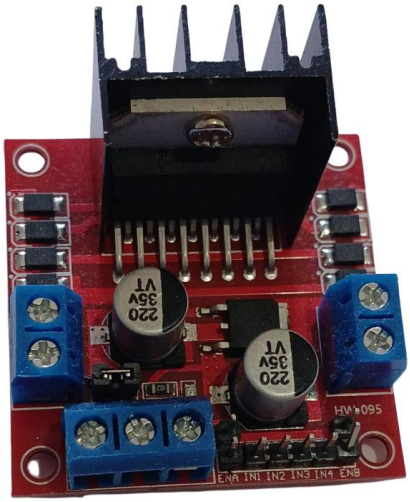


Fonte: O Autor (2026).

Ponte H L298N (Driver de Potência): Observada na Figura 21, sua seleção foi devido a sua capacidade de permitir a inversão de polaridade ao receber sinais PWM, e ter a

compatibilidade para integração à dois motores, controlando direção e velocidade. Além disso, a ponte H suporta a corrente exigida por ambos os motores.

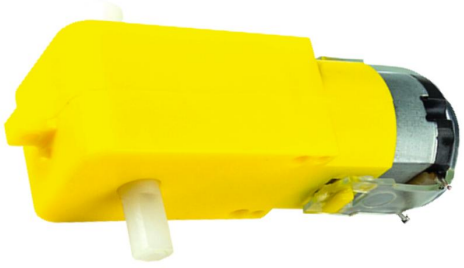
Figura 21 – Ponte H.



Fonte: O Autor (2026).

Motores e Plataforma (Atuadores e Estrutura): Os atuadores do protótipo são dois Motores DC (3-6 V) com caixa de redução e eixo duplo que, segundo seu *datasheet*, possuem torque de 800 gf·cm. Essas especificações conferem capacidade para deslocar a massa total de 800 g do robô. Um dos exemplares pode ser visto na Figura 22.

Figura 22 – Motor DC com Caixa de Redução e Eixo Duplo.



Fonte: O Autor (2026).

Já para a plataforma estrutural do protótipo foi selecionada a FalconV2, observada na Figura 23, ela provê suporte à montagem do projeto, versatilidade e área de acoplamento, o que permite a organização dos componentes de forma a otimizar as conexões do circuito.

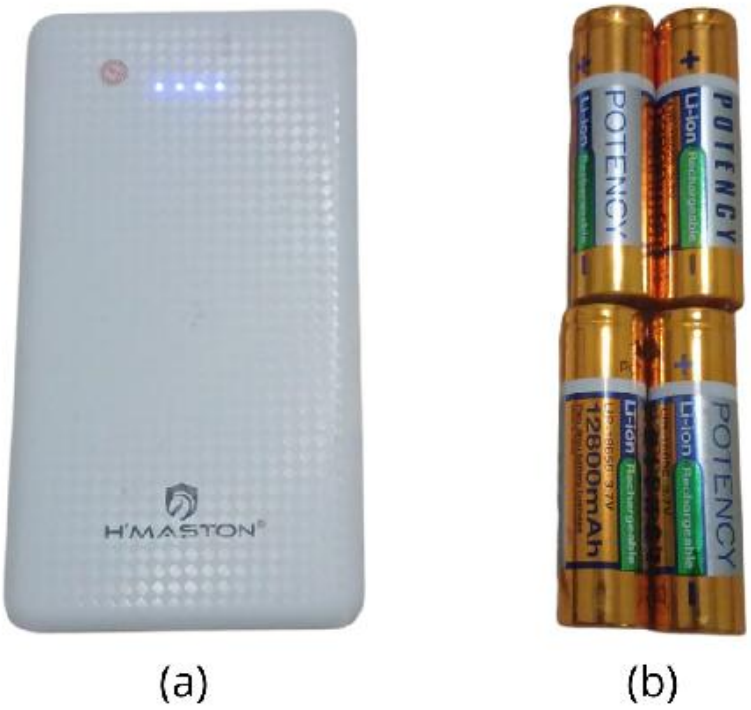
Figura 23 – plataforma robótica falcon v2.

Fonte: O Autor (2026).

Sistema de Alimentação: Para garantir a autonomia do servomecanismo, ilustrada na 24, a alimentação foi dividida em duas partes:

- **Alimentação de Potência:** Selecionou-se um arranjo de quatro baterias Li-ion 18650, em uma configuração 2S2P (duas em série e duas em paralelo). Tal associação fornece uma tensão nominal de 7,4 V, necessária para alimentar os motores DC e gerar um torque que supere a inércia do protótipo, mesmo com a queda de tensão nos componentes de comutação da Ponte H. Outra característica importante dessa configuração é o dobro da capacidade de carga, estendendo assim o tempo de operação.
- **Alimentação Lógica:** Um *Powerbank* de 5000 mAh com saídas USB-C e Micro-USB, utilizadas, respectivamente, para alimentar isoladamente o Raspberry Pi 5 e o ESP32. Essa segmentação de alimentação visa evitar que os ruídos elétricos gerados pelos motores interfiram no processamento lógico, prevenindo o travamento ou reinicialização indesejada da parte lógica do protótipo.

Figura 24 – (a) Powerbank (5000 mAh). (b) Baterias (Li-Ion 18650).



Fonte: O Autor (2026).

A disposição física dos componentes no protótipo foi projetada para garantir a integridade dos sinais lógicos. O espaço interno foi organizado de forma que os componentes de processamento e controle ocupem o setor superior da plataforma. Enquanto os motores e as fontes de alimentação foram colocados no setor inferior. Essa configuração separa os componentes lógicos dos atuadores e fontes, devido a uma maior parcela da massa se concentrar nas fontes, reduz o centro de massa, promovendo ao protótipo maior estabilidade durante as manobras de rastreio.

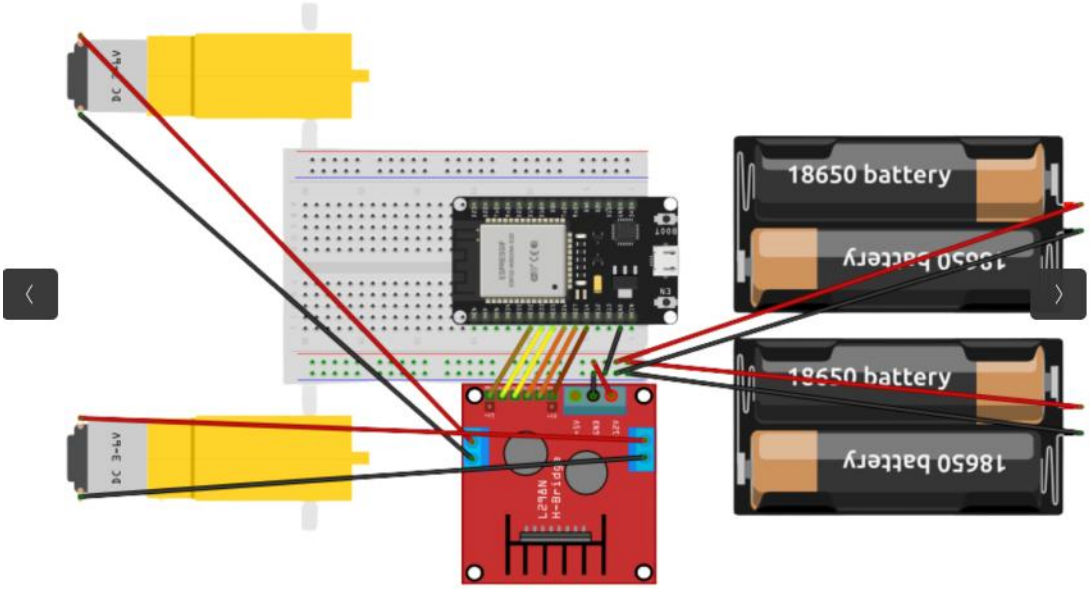
A câmera foi posicionada de maneira a proporcionar um campo de visão livre para o robô, localizada a frente do robô. Assim, foi ajustado para que, mesmo estando próximo ao solo, o sensor óptico tenha a capacidade de monitorar alvos altos sem a necessidade de mudança estrutural na plataforma.

3.1.2 Esquematização do Circuito Elétrico

Para as conexões elétricas do protótipo, optou-se pela segmentação física entre a malha de alta potência (Motores e Ponte H, alimentados pelas baterias 18650) e a malha de baixa potência (ESP32 e Raspberry Pi, alimentados pelo *Powerbank*), de forma a minimizar ruídos provenientes do acionamento dos motores no processamento lógico. Uma característica crítica no sistema elétrico é a unificação dos pontos de referência. Visto que os aterramentos (GND) do ESP32 e da Ponte H foram conectados entre si. Essa configuração elétrica, bem

como a interconexão entre o microcontrolador, *driver* de potência e fontes, é apresentada na Figura 25.

Figura 25 – Esquema do circuito elétrico do servomecanismo.



Fonte: O Autor (2026).

A interface lógica entre a ESP32 e a ponte H L298N é apresentada na Tabela 6. Os pinos de habilitação, ENA e ENB, são responsáveis por controlar a velocidade e a ativação dos atuadores por meio do PWM. Já os pinos de IN1 a IN4 determinam a direção dos motores. A variação do ciclo de trabalho (*duty cycle*) dos pinos de habilitação é comandada pelo sistema de controle, que altera a direção e a tensão média aplicada aos motores, permitindo ajustes finos na velocidade para um rastreamento com maior precisão.

Tabela 6 – Interface lógica entre ESP32 e L298N.

Pino (L298N)	**Pino (ESP32)**	**Constante**	**Função**
IN1	27	IN1	Motor Esquerdo (Avançar)
IN2	26	IN2	Motor Esquerdo (Recuar)
IN3	25	IN3	Motor Direito (Avançar)
IN4	33	IN4	Motor Direito (Recuar)
ENA	14	ENA	Habilitação e PWM do Motor Esquerdo
ENB	32	ENB	Habilitação e PWM do Motor Direito

Fonte: O Autor (2026).

Na alimentação de potência, a Ponte H L298N utiliza transistores bipolares, o que resulta em uma queda de tensão interna variando de 1,4 V a 2 V. A configuração 2S2P das baterias Li-Ion 18650 tem tensão nominal de 7,4 V, o que compensa a queda de tensão e fornece aos motores uma tensão efetiva de 5,4 a 6 V. Na alimentação lógica, o *Powerbank* selecionado

atende às demandas de carga sem quedas de tensão, evitando o *reboot* do Raspberry Pi 5, mesmo com o pico de consumo ao executar o modelo YOLO11n.

3.1.3 Integração e Comunicação MQTT

Para a integração entre os processadores do sistema, Raspberry Pi 5 e ESP32, adotou-se a utilização de um protocolo de comunicação com as seguintes características: leve, eficiente e de alta confiabilidade. Assim, optou-se pelo protocolo MQTT, que realiza a comunicação com baixo *overhead* (sobrecarga de dados) e de latência reduzida.

Para esta estrutura de comunicação, arquitetou-se uma rede local utilizando um roteador dedicado como ponto de acesso externo, sem dependência de internet. Esse modelo de comunicação também permite uma maior flexibilidade durante o desenvolvimento e a depuração, devido ao computador e o sistema se conectarem na mesma rede. Para a configuração da rede local, “rede_pfc”, foram designados endereços reservados de IP estáticos via endereço MAC, descritos na Tabela 7, para que o mapeamento lógico permaneça constante.

Tabela 7 – Mapeamento de endereçamento IP Estático da Rede Local.

Dispositivo	**MAC Address**	**Endereço IP**	**Função na Rede**
Notebook	98:83:89:XX:XX:XX	192.168.0.100	Monitoramento e Debug
ESP32	F8:B3:B7:XX:XX:XX	192.168.0.101	MQTT Subscriber
RP Pi 5	2C:CF:67:XX:XX:XX	192.168.0.102	MQTT Publisher / MQTT Broker

Fonte: O Autor (2026).

No protocolo MQTT, a ESP32 atua como Subscritor e a Raspberry como Publicador e *broker*. Para que o tenha a capacidade de diferenciar as conexões e gerenciá-las, atribuiu-se a cada nó um identificador de cliente (*Client ID*). Para a configuração dos tópicos utilizados na Raspberry Pi, o identificador é `Raspberry_pfc` e o do ESP32 é `ESP32_pfc`.

Outra característica do MQTT é o formato de envio das mensagens, que ocorrem no formato *string* (texto) ou numérico, visando minimizar custo computacional no microcontrolador. Na Tabela 8, podem ser observados os tópicos criados para a comunicação do sistema.

Tabela 8 – Definição de Tópicos MQTT do Projeto.

Tópico	**Direção**	**Dado**	**Finalidade**
pfc/controle	RPi $\rightarrow$ ESP32	Inteiro	Par de velocidades.
pfc/busca	RPi $\rightarrow$ ESP32	Inteiro	Indica quando entrar e sair do modo busca
pfc/config	RPi $\rightarrow$ ESP32	inteiro	Envia parâmetros iniciais para a sintonia.

Fonte: O Autor (2026).

O tópico `pfc/config` foi criado para sintonizar variáveis iniciais, que dinamicamente foi realizada de forma remota, sem a necessidade de novas compilações na ESP32. Já o tópico `pfc/controle` tem a função de enviar o par de velocidades de ambos os motores, transitando entre os estados Parar, Transladar, e Rotacionar. Enquanto o tópico `pfc/busca` envia o comando de ligar e desligar o estado Buscar. A implementação do MQTT na ESP32 foi realizada de forma a tratar as instabilidades de rede e a assincronia dos dados.

Para o envio, com baixa latência, foram selecionados diferentes níveis de QoS para o envio das mensagens. Para as variáveis iniciais de controle por meio do tópico `pfc/config`, que são enviadas logo no início do algoritmo, foi selecionado o nível QoS 1, que consiste em um envio pelo menos uma vez. Já para o envio dos demais, foi utilizado o QoS 0, que consiste em um envio único sem a confirmação, cujo foco é a velocidade de entrega da mensagem, já que o atraso torna esses dados obsoletos. Para garantir o funcionamento do sistema, foram declaradas no próprio algoritmo da ESP32, valores padrão para casos em que há falha na mensagem, o protótipo consiga operar.

Para o tratamento das mensagens recebidas foi implementada a função *callback*, que atua como núcleo de recepção de dados via protocolo MQTT, sendo executada toda vez que a ESP32 recebe uma mensagem no tópico inscrito. Essa função converte o *payload* recebido, para valores referentes a cada mensagem, Filtrando as mensagens por tópicos.

No tópico de busca, a mensagem recebida é convertida para um valor inteiro que define a ativação do estado de busca, no qual o valor "0" desativa o estado Buscar, enquanto o valor "1" ativa-o.

Para o tópico de controle, o sistema converte a mensagem recebida de uma *string* no formato "V_e, V_d"para duas variáveis com valores inteiros. Assim, o algoritmo define a direção e a magnitude da velocidade.

De forma semelhante, o tópico de configuração inicial é recebido e convertido, de uma *string* igual a de velocidade, e dividida em duas variaveis inteiras `ligado_rotacao` e `ligado_busca`, que respectivamente, limitam o tempo que os estados Rotacionar e Buscar ficam acionados no acionamento intermitente. Os valores padrões, em caso de falha no envio das configurações iniciais são de 30 ms para rotação e 750 ms para busca.

Se os dados são recebidos pelo canal de controle, ela extrai as velocidades dos motores e define a direção do movimento; caso cheguem pelo canal de configuração, ela ajusta os parâmetros operacionais de rotação e busca, garantindo que o protótipo responda com baixa latência aos comandos externos de forma dinâmica.

Adicionalmente, o marcador de tempo `last_msg_time` é atualizado, atuando como um sistema de *watchdog* de software operando de maneira a deligar o funcionamento dos motores caso uma nova mensagem não for recebida em um determinado período de tempo. Esse procedimento concede ao sistema a capacidade de cessar o funcionamento dos motores em casos que a comunicação falhe ou a Raspberry Pi trave.

Caso seja identificada uma falha na conexão com o *broker* ou com o sinal Wi-Fi, o

algoritmo da ESP32 inicia uma rotina de reconexão (função *reconnect*). Nessa função, caso a conexão seja perdida, o watchdog de software interrompe o acionamento dos motores por segurança, enquanto o algoritmo continua a monitorar o estado da rede. Ao retomar o sinal, a operação retorna normalmente.

3.2 Rastreamento e Predição

O sistema de rastreio deste trabalho foi executado de forma exclusiva pelo Raspberry Pi 5. Esta unidade é responsável por processar o fluxo de vídeo da câmera, localizar o alvo, calcular o comando e o erro, e enviá-los para o controlador. O modelo utilizado foi o YOLO11n para detecção e o Filtro de Kalman para predição da trajetória do alvo.

Para a implementação destes conceitos, foi desenvolvido um algoritmo em *Python*, cujo fluxo de execução consiste na captura de quadros via *OpenCV*. Assim, cada quadro é processado pela rede neural YOLO11 e as coordenadas da caixa delimitadora são extraídas do alvo, que por sua vez, alimentam o Filtro de Kalman, mitigando ruídos das detecções, proporcionando uma estimativa de estado estável.

3.2.1 Configuração do Ambiente e Bibliotecas

Para a configuração da Raspberry Pi 5, foi efetuada a instalação do sistema operacional Raspberry Pi OS (*Legacy*, 64-bit). Para isso, utilizou-se a interface gráfica *Raspberry Pi Imager*. A justificativa para o uso desta interface é dada pela sua praticidade de se configurar em apenas uma etapa, a conexão WLAN e o protocolo SSH, habilitando o acesso remoto ao terminal do Raspberry Pi 5 via rede local.

Após a conectividade ser estabelecida, realizou-se a configuração do *broker*. Para isso, foi instalado, via terminal, o Eclipse Mosquitto (versão 2.0.18), um dos softwares de código aberto mais utilizados para aplicações de MQTT. Este pacote cria um servidor em segundo plano no SO que opera como *broker* local da rede. Contudo, a comunicação entre Raspberry Pi e ESP32 possui uma baixa latência, não impedindo a operação do protótipo.

O Mosquitto adota uma postura de segurança restritiva, limitando conexões e exigindo autenticação. Para a comunicação via rede local sem fio (WLAN) com a ESP32, é necessária a liberação de acesso. Essa liberação é realizada por meio da adição de duas instruções no arquivo `local.conf`:

- `listener 1883 0.0.0.0`: instrução que direciona o serviço a escutar a porta padrão 1883, permitindo que os dispositivos da rede alcancem o *broker*.
- `allow_anonymous true`: habilita a permissão de troca de mensagens, simplificando o fluxo de dados na fase de validação.

A Raspberry Pi OS possui suporte nativo à linguagem *Python*, foi criado um ambiente virtual denominado “pfc”, o qual confere ao projeto um espaço isolado, organizado e de

reprodutibilidade. Neste ambiente, foram instaladas as dependências críticas para o projeto, como *OpenCV*, *Ultralytics*, *Numpy*, *Filterpy*, *Paho.mqtt.client*.

O *OpenCV* é utilizado para captura e manipulação de imagens, *Ultralytics* é responsável pelo modelo YOLO11n, que por sua vez, tem a função de realizar a detecção, Numpy para tratar as operações de vetores e matrizes, *Filterpy* e utilizado para a implementação do filtro de Kalman e *Paho-mqtt* para gerenciar a comunicação do sistema. Já para o funcionamento lógico no ESP32 foram utilizadas duas bibliotecas: *Wifi.h*, utilizada para gerenciar conexões, modos de operação e verificar *status* de rede; *PubSubClient.h* para implementar o protocolo MQTT no microcontrolador.

No desenvolvimento dos algoritmos de visão e de controle, utilizou-se o *Visual Studio Code* (*VS Code*) para o desenvolvimento da lógica de visão e a filtragem estocástica em *Python*. Enquanto o *IDE Arduino* foi usado para o desenvolvimento do algoritmo executado na ESP32 (em linguagem C++).

Antes da migração do sistema para a Raspberry Pi, a validação dos resultados com a realização de simulações foi realizada pelo computador, simplificando a validação de sintaxe e a depuração do algoritmo.

Uma estratégia utilizada a fim de minimizar o tempo de *upload* do algoritmo para a ESP32, foi enviar os ajustes das variáveis iniciais dos testes diretamente do código em *Python*. Para tal, as variáveis iniciais são publicadas no tópico `pfc/config`. O que permite que o comportamento do sistema seja ajustado durante os testes.

3.2.2 Integração da Rede Neural e Filtragem Estocástica

Para a detecção do alvo, foi utilizado o modelo YOLO11n. Esta versão foi selecionada devido ao equilíbrio entre custo computacional e precisão no rastreamento. O algoritmo foi configurado para otimização do processamento, como um algoritmo para processar exclusivamente a classe de objetos com `ID = 0`, que remete a pessoas. Essa configuração faz com que o modelo descarte objetos que não são interessantes ao objetivo deste trabalho, garantindo o funcionamento para alvos válidos.

De forma alternativa a detecção convencional, foi utilizada a função `model.track`, que por sua vez, não analisa quadros isoladamente, ela localiza o alvo e tenta prever sua posição no próximo quadro com base no histórico de movimento, auxiliando no rastreio. Com o adendo do argumento `persist=True`, o sistema atribui um ID para o alvo, conforme a Figura 26.

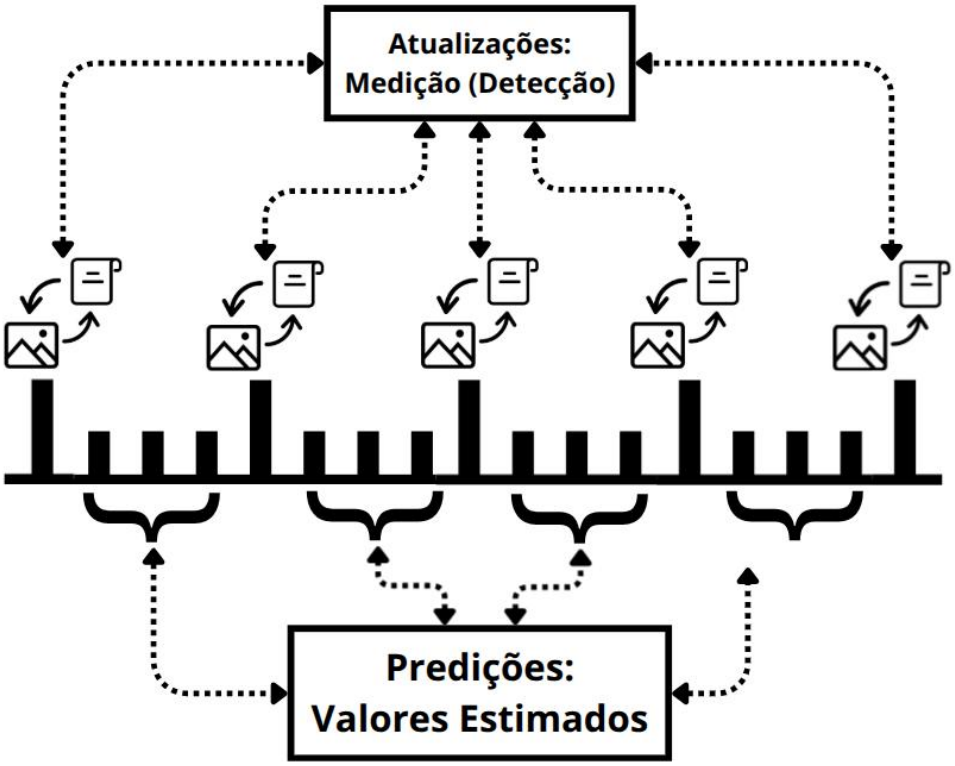
Figura 26 – Teste de detecção de pessoas com YOLO11n.

Fonte: O Autor (2026).
Nota: Imagem de teste gerada pelo autor com o Gemini.

Considerando a alta demanda computacional da rede neural, adotou-se a estratégia de processamento de *multirate* (múltiplas taxas de atualização). Visto que devido a complexidade computacional de redes neurais profundas como a YOLO11n, o algoritmo foi configurado para realizar a inferência da YOLO11n apenas a cada ciclo definido pela variável `FRAMES_PARA_PULAR = 3`, ou seja, a inferência da YOLO11n ocorre em 1 a cada 4 quadros. Contudo, o filtro executa a predição na mesma taxa do fluxo de imagens da câmera, permitindo que o algoritmo de visão computacional envie estimativas de posição e altura para a FSM nos quadros intermediários em que a rede neural está em repouso. Este procedimento reduz o atraso de transporte e garante o controle em todos os quadros.

Outra característica da arquitetura utilizada é como o sistema alterna entre as atualizações baseadas em medições reais (detecções) e as predições (estimativas do filtro) que mantêm o rastreamento ativo nos quadros intermediários. Essa dinâmica de operação é ilustrada na Figura 27.

Figura 27 – Integração entre detecções da YOLO11n e as predições do Filtro de Kalman.



Fonte: O Autor (2026).

Para a dinâmica de filtragem, numa amostra de 4 quadros da câmera, os 3 primeiros são descartados sem que seja executado a detecção pela rede neural, esse procedimento foi implementado devido a limitação do tempo de processamento. Assim, o filtro passa pelo estado de predição para manter a continuidade da trajetória, enquanto no último desses 4 quadros, a detecção é executada e o filtro é atualizado com os resultados da mesma.

A utilização do `model.track` e `persist = True` opera como um suporte para melhorar a confiabilidade dos dados. Se o rastreamento falhar em casos de mudanças de iluminação ou se o alvo estiver obstruído. Assim, sua função principal é fornecer dados sobre quais coordenadas do quadro o sistema deve priorizar para a filtragem estocástica, aumentando a confiabilidade da informação.

Outra técnica utilizada para uma melhor performance foi a utilização da função do parâmetro `imgsz` no objeto model na detecção YOLO, esse parâmetro tem por função redimensionar o fluxo de imagens que do sensor visual, diminuindo seu tamanho, de modo a sacrificar a detecção de objetos pequenos em contrapartida, aumenta o desempenho computacional do sistema, pois ao diminuir a altura e largura por 2, o tamanho (área) da tela é diminuído em 4 vezes.

Após a detecção realizada, o algoritmo fornece as variáveis, que após serem submetidas a um pós processamento, transformando-as nas variáveis de entrada dos controladores. São elas coordenadas da caixa delimitadora, extraídas via atributo `results[0].boxes.xywh`,

`x_pixels` e `h_pixel`, correspondentes respectivamente, à coordenada central horizontal do alvo, e à altura do objeto, ambos valores em *pixels*.

As variáveis normalizadas x_{norm} e h_{norm}, possuem domínio, respectivamente, $[-0,5; 0,5]$ e $[0; 1]$. Essa diferença diz respeito ao referencial utilizado, onde o referencial de x_{norm} é dado como o centro da tela, dividindo a em um lado negativo e outro positivo. Isso simplifica a implementação do estado de rotação: x_{norm} positivo faz o robô virar à direita, e negativo, à esquerda.

Apesar do modelo YOLO11n, indicar coordenadas de detecção a serem priorizadas, esses dados possuem naturalmente ruídos. Para mitigar estes ruídos e aumentar a confiabilidade dos mesmos, foi implementado um filtro de Kalman linear. O filtro de Kalman é aplicado de maneira a processar a coordenada central do alvo em cada quadro, realizando a predição em momentos de falha, assim suavizando a trajetória do alvo.

Assim, foram modelados dois filtros, para x_{norm} e h_{norm} e os vetores de estado bidimensional ($dim_x = 2$), dado pela posição e pela variação no tempo (velocidade), e com uma única dimensão de medição ($dim_z = 1$), associada ao dados medidos e normalizados obtida pela câmera. A implementação da biblioteca *FilterPy*, processa a coordenada central do alvo de forma a suavizar a trajetória.

A matriz de covariância do erro de estimação inicial (P_0) define o grau de incerteza do estado inicial do sistema. Devido a incerteza da posição e velocidade inicial do alvo. No algoritmo, atribuiu-se para P_x e P_h respectivamente os valores 10 e 5 aplicados à matriz identidade. Pois a estimativa inicial da altura apresenta menor incerteza relativa. Essa parametrização força o filtro a atribuir maior peso às primeiras detecções, garantindo uma rápida convergência para a trajetória real do alvo.

A matriz de covariância do ruído de medição (R), representa a variância das medições (detecções do YOLO11n). Nesse presente trabalho, conforme a Equação 2.15, os valores de R foram estimados experimentalmente, por meio da variância das medições. O robô fica em repouso com uma pessoa (alvo) parada a sua frente, no teste foi obtido para R_x e R_h, respectivamente, 0.36×10^{-5} e 59.29×10^{-5}.

Todavia, a matriz de covariância do ruído de processo (Q), quantifica as incertezas do modelo dinâmico, como adotou-se o modelo de movimento com velocidade constante sujeito a aceleração branca Gaussiana. O parâmetro inicial de como a Matriz Q pode ser descrita em função do desvio padrão de aceleração e o tempo discreto dt ajustado empiricamente para controlar a variância do estado estimado. Com valores encontrados de $1,5$ para σ_{ax} e de 5 para σ_{ah}.

O ajuste adequado destas matrizes determinam o comportamento do ganho de Kalman, melhorando a estabilidade do sistema, velocidade de convergência e o rastreamento em casos de oclusões. Essas variáveis normalizadas são filtradas pelo estado `kf_x.x[0]` e `kf_h.x[0]` do filtro de Kalman, assim, tem-se respectivamente, x_f e h_f.

3.3 Controle e Atuação

Nesta seção é apresentada a integração entre o rastreamento e a FSM, bem como o sistema de atuação. Assim, tendo como função converter valores fornecidos pelo algoritmo de visão computacional em sinais de potência para os motores DC.

Quando o objeto é detectado, inicialmente o protótipo adota como alvo a primeira pessoa detectada, nos casos de detecção de mais uma pessoa o alvo é a pessoa identificada com menor ID, em condições normais de aproximação, a maior pessoa representada no quadro.

Em cenários de oclusões, o filtro de Kalman mantém o rastreamento por alguns quadros. Contudo, no caso de perda total da detecção, o robô realiza uma varredura no ambiente girando na direção da última localização do alvo, assim retomando o alvo ou a primeira pessoa encontrada.

3.3.1 Máquina de Estados Finitos

A FSM foi definida por meio do mapeamento da Figura 28, que representa o campo de visão no referencial do robô, o estado de controle para cada posição do alvo.

Na Figura 28, os estados de controle da FSM: Buscar (B), Rotacionar (R), Transladar (T) e Parar (P). No qual divide o estado Transladar em translação de avanço (T_A) e de recuo (T_R), enquanto o estado Rotacionar é segmentado em rotação para esquerda (R_E) e para direita (R_D).

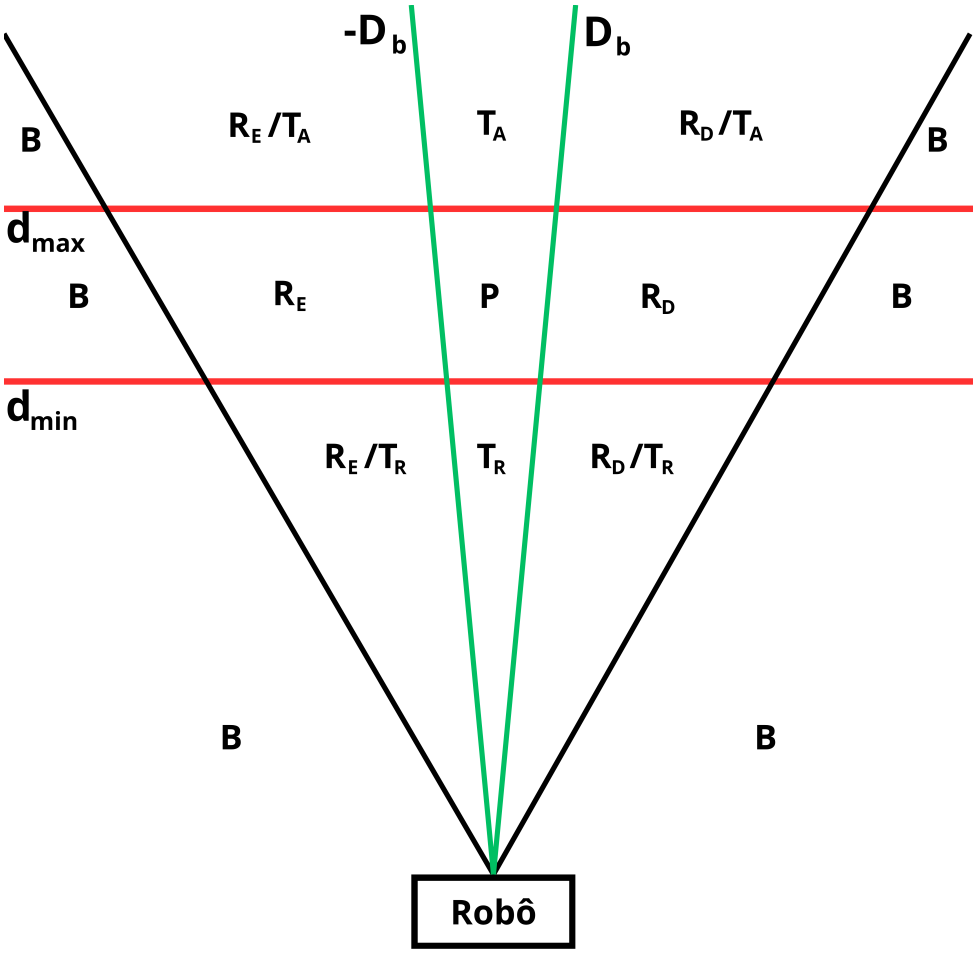
O espaço entre as linhas verdes $\pm\ D_b$, representa a zona morta do estado Rotacionar, enquanto as linhas vermelhas, apresentam os limites da faixa de histerese da distância longitudinal (Estado Transladar), na qual a altura da caixa delimitadora ($h_{filtrado}$) servem como referencia h_{min} e h_{max}, respectivamente, para as distâncias relativas d_{max} e d_{min}. Os valores de D_b e os limites de altura da caixa delimitadora foram selecionados de modo a evitar o efeito de *chattering*, sendo $D_b = 0,1$ (`ERRO_DEADBAND`) e os limites de altura entre $h_{min} = 80\%$ e $h_{max} = 90\%$ da altura máxima da tela.

Logo, a movimentação do protótipo é gerenciada pela FSM que processa as informações x_f e h_f, na qual a prioridade é dada aos estados na seguinte sequência: Buscar, Rotacionar, Transladar e Parar. Dado o Estado são enviados um par de velocidades correspondentes para a ESP32 via MQTT. Para o deslocamento do protótipo, definiu-se uma velocidade base para o sistema de atuação, configurada com um sinal PWM de hardware com frequência de 1000 Hz e *duty cycle* de 82% (*duty cycle* de 82%. No caso, com 256 níveis (0-255) corresponde a um valor de 210).

No estado Parar, o robô se encontra em estado de repouso, permanecendo parado, suas condições de entrada são as variáveis de controle estarem dentro das faixa de tolerância da zona morta e de histerese. A mensagem enviada via MQTT é: $(V_e, V_d) = (0, 0)$.

Enquanto o estado Transladar é acionado quando o alvo se encontra fora do setor desejado entre os dois limiares de altura do alvo. Sendo que neste estágio o comportamento

Figura 28 – Mapeamento dos Estados da FSM.



Fonte: O Autor (2026).

é segmentado em dois movimentos. Conforme a Equação (2.23), quando a altura é maior que h_{max}, o robô afasta-se, a mensagem enviada é: (V_e, V_d) = $(-82\%, -82\%)$. E quando está menor que h_{min} ele se aproxima, a mensagem enviada é: (V_e, V_d) = $(82\%, 82\%)$.

Para os estados que utilizam os movimentos de rotação, Rotacionar e Buscar, foi implementado um acionamento *on-off*, ou acionamento intermitente, controlado por software. Esse procedimento foi adotado para garantir um torque que supere a inércia estática do protótipo, garantindo o funcionamento dos atuadores.

No estado Rotacionar, no qual o robô se alinha com o alvo, x_f é testado conforme a Equação (2.24), se x_f implicar em uma rotação para a direita, logo a mensagem é: (V_e, V_d) = $(82\%, -82\%)$. Em contrapartida se x_f indicar uma rotação para a esquerda, a mensagem é: (V_e, V_d) = $(-82\%, 82\%)$.

Para melhorar a resolução angular, no acionamento intermitente o tempo ligado foi de 30 ms para cada ciclo de 100 ms, evitando oscilações por *overshoot*.

3.3.1.1 Estado Buscar

O estado de Buscar é acionado quando o número de perdas consecutivas for maior do que um limite estabelecido para a busca. Assim, ocorre alteração na variável `busca` de 0 para 1, que ao ser enviada para a ESP32 executa a varredura do ambiente.

Nesse estado, o processo visa encontrar um alvo perdido sem causar borrão de câmera. As velocidades são aplicadas diretamente na ESP32 sem depender de envio, com os mesmos valores do estado Rotacionar. Para a varredura foi utilizado o processo de acionamento intermitente citado no estado Rotacionar, além de outro acionamento que opera com tempo ligado (*ON*) fixado em 750 ms para cada 1000 ms de um ciclo, para uma busca sempre constante.

Este procedimento permite ao robô rotacionar, percorrendo o ambiente de forma suave, e esse repouso maior de 250 ms no primeiro acionamento intermitente garante que a câmera capture novos quadros nítidos e que o Raspberry Pi processe a imagem em busca da reaquisição. Quando o alvo é readquirido, a variável busca tem o valor alterado para 0 novamente.

Contudo, como o alvo foi completamente perdido, foi implementada uma lógica de reaquisição de alvo. Quando o sistema identifica uma nova pessoa, é atribuído um novo track_id a essa pessoa, e o sistema começa a segui-la.

3.4 Validação Experimental

A validação experimental foi realizada por meio de ensaios controlados, em etapas que avaliam a dinâmica operacional do protótipo. O objetivo desta seção é descrever o ambiente de teste e os protocolos de coleta de dados que fundamentam os resultados apresentados no Capitulo 4.

3.4.1 Ambiente de Teste e Equipamentos

Os experimentos foram conduzidos em um ambiente não estruturado e iluminado, utilizando uma pista de testes plana de superfície rígida para garantir um coeficiente de atrito constante.

A infraestrutura de testes baseia-se em uma rede local dedicada, que garante a comunicação bidirecional de baixa latência entre o microcontrolador ESP32 e a unidade central de processamento.

A coleta de dados foi realizada de forma automatizada por meio da adição de uma lógica de coleta de dados no próprio algoritmo de rastreamento executado diretamente na Raspberry Pi 5. Durante os ensaios de rastreio e busca, o algoritmo registra as variáveis críticas do sistema, incluindo as coordenadas do alvo detectadas pelo YOLOv11, as predições do Filtro de Kalman, o tempo do teste, o tempo de cada ciclo e se a busca está ativa ou não, exportando-as para arquivos em formato CSV (*Comma-Separated Values*).

Posteriormente, esses conjuntos de dados foram processados para a geração de gráficos de desempenho, utilizando o *Python* para análise de dados. Essa metodologia permite correlacionar o comportamento dinâmico do robô com a precisão do rastreamento visual, facilitando a análise da eficiência das manobras e recuperação de alvo na busca.

4 RESULTADOS E DISCUSSÕES

Neste capitulo são apresentados os resultados obtidos a partir da implementação e integração entre o sistema de visão computacional, filtragem estocástica e controle. Tais resultados serão segmentados em desempenho do detector de objetos em tempo hábil de operação e o desempenho do hardware.

4.1 Desempenho do Rastreamento Visual

Esta seção destina-se à análise dos resultados obtidos do sistema de visão computacional, promovendo um conjunto de avaliações relativas ao comportamento e desempenho da estimação promovida pela captura das imagens e dos processamentos envolvidos até a estimação de posição e orientação relativas.

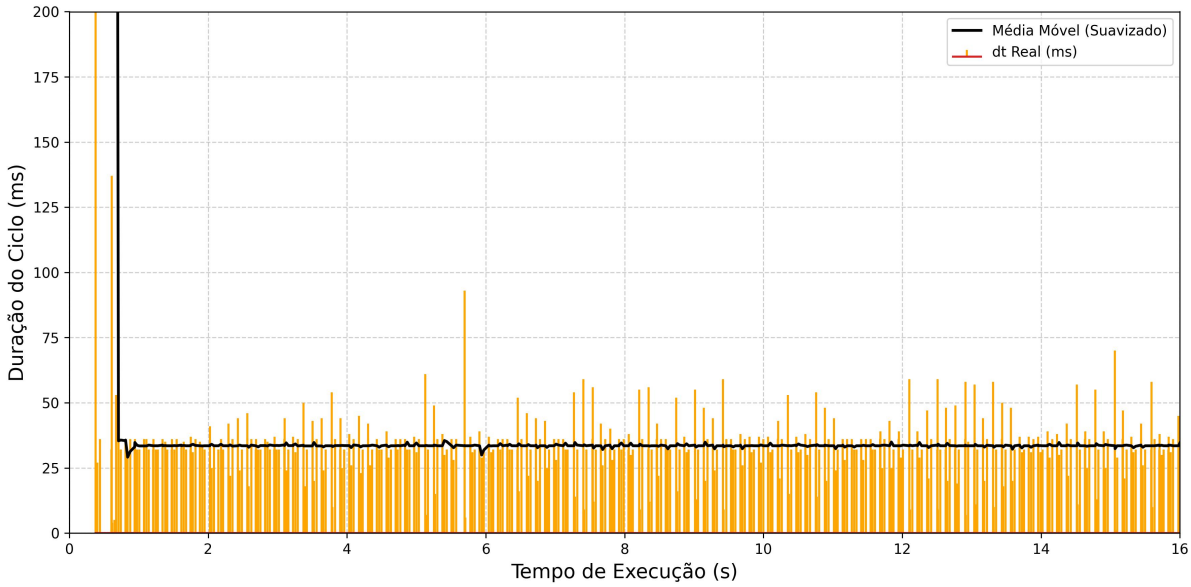
4.1.1 Análise dos tempos de Estimação e Controle

Durante o desenvolvimento, o tamanho das imagens foram variadas para garantir um tempo de processamento que não afetasse o sistema de controle, considerando a operação embarcada no Raspberry Pi 5. De início, foi avaliada a resolução de 640×480 para ser passada para o sistema de detecção pela YOLO11, contudo, ao utilizar essa resolução, o tempo de resposta do sistema não foi satisfatório. Isto posto, a imagem foi redimensionada para 320×240, proporcionando ao sistema a capacidade de operação em tempo hábil.

Com a resolução maior, o tempo de processamento foi de aproximadamente 310 ms por iteração do *loop* principal, no qual ocorre a detecção. Em comparativo, com a diminuição da resolução da tela para 320×240, esse tempo diminui para aproximadamente a 100 ms por iteração, assim, obtendo um processo com mais quadros analisados pela rede neural.

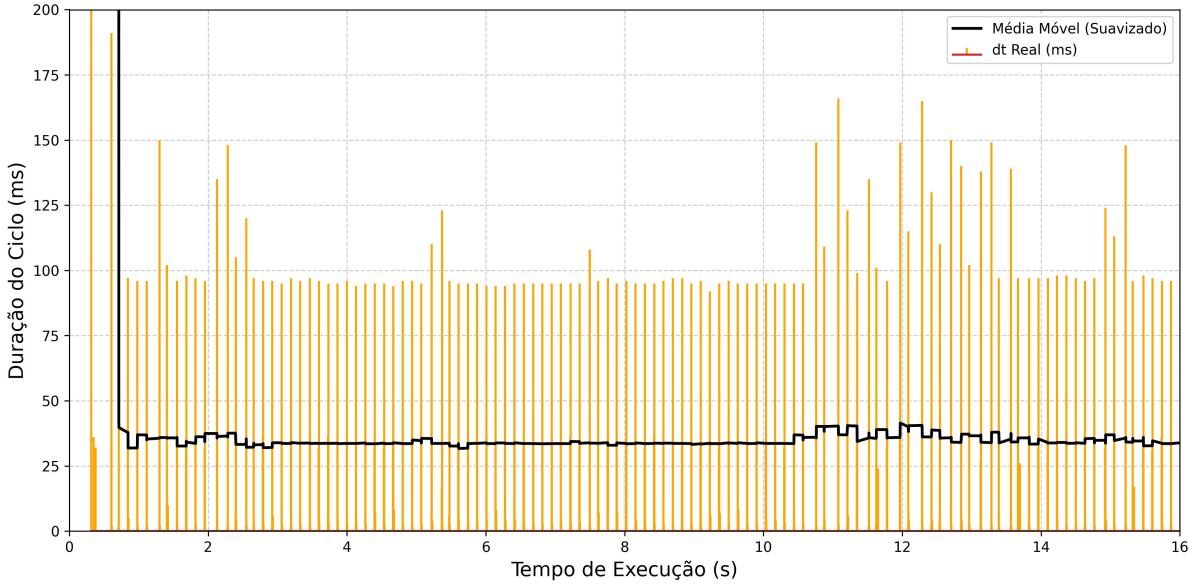
As Figuras 29 e 30 apresentam a variação do tempo necessário para a conclusão de um ciclo (Δt) definido como uma iteração do *loop* principal do algoritmo de rastreio, realizadas respectivamente, no computador e na Raspberry Pi 5, considerando a resolução da imagem de 320×240. O computador utilizado possui as seguintes especificações: processador Intel Core i7 - 7700 HQ (2.80 GHz), 8 GB de memória RAM e sistema operacional Ubuntu 24.04.2 LTS 64-bit.

Figura 29 – Análise do tempo de cada iteração (Δt) no computador.



Nota: a média móvel utilizou uma janela de 16 amostras.
Fonte: O Autor (2026).

Figura 30 – Análise do tempo de cada iteração (Δt) na Raspberry Pi



Nota: a média móvel utilizou uma janela de 16 amostras.
Fonte: O Autor (2026).

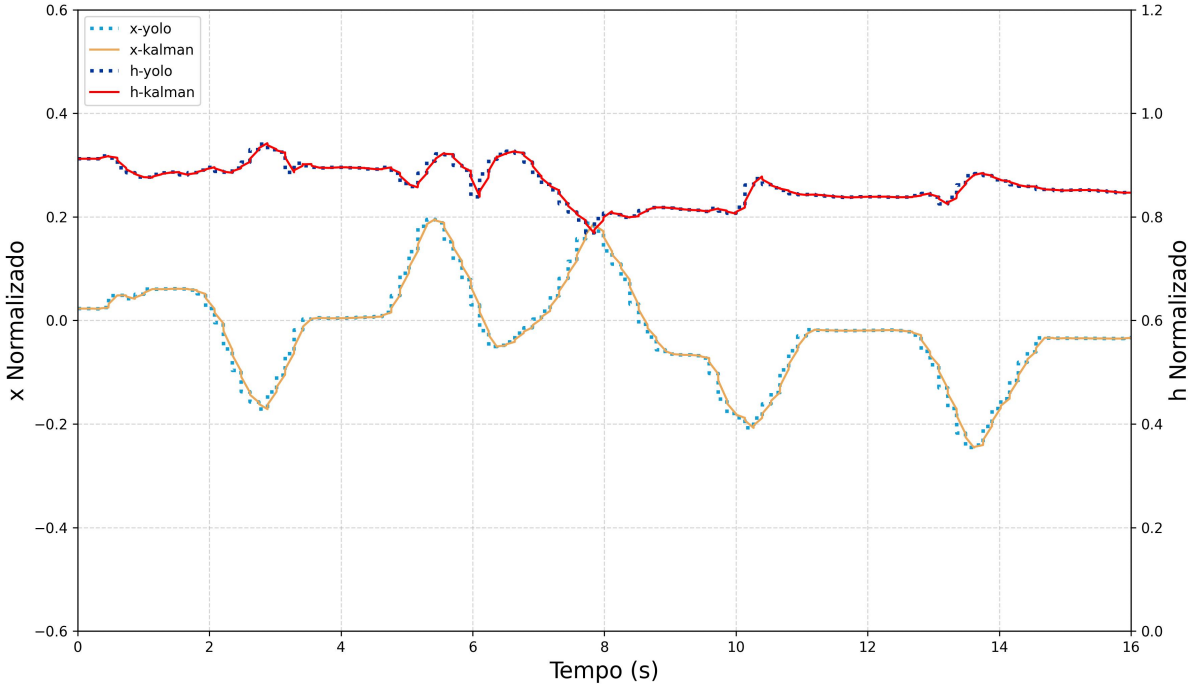
Esses gráficos validam o sistema operando no Raspberry Pi 5 comparado ao desempenho da execução no computador. Nota-se que a média móvel da Raspberry Pi 5 mantém a duração da maior parte dos ciclos abaixo de 100 ms, já o computador, que dispõe de

uma unidade de processamento com maior capacidade, mantém a duração por ciclo aproximadamente a 40 ms na maior parte do tempo. Esses dados mostram que a Raspberry Pi atendeu os requisitos temporais impostos.

4.1.2 Análise de Deslocamento Transiente e Resposta Dinâmica

Para avaliar a dinâmica de resposta do filtro e o tempo de convergência do estimador, foi realizado um teste mostrado na Figura 31, no qual o protótipo estava em repouso (sem o funcionamento dos motores) e o alvo ficou inicialmente centralizado. Assim, realizou-se movimentações laterais de aproximadamente 1 metro em ambas direções, e apresentou-se uma convergência adequada do filtro de Kalman.

Figura 31 – Resposta ao Degrau.



Fonte: O Autor (2026).

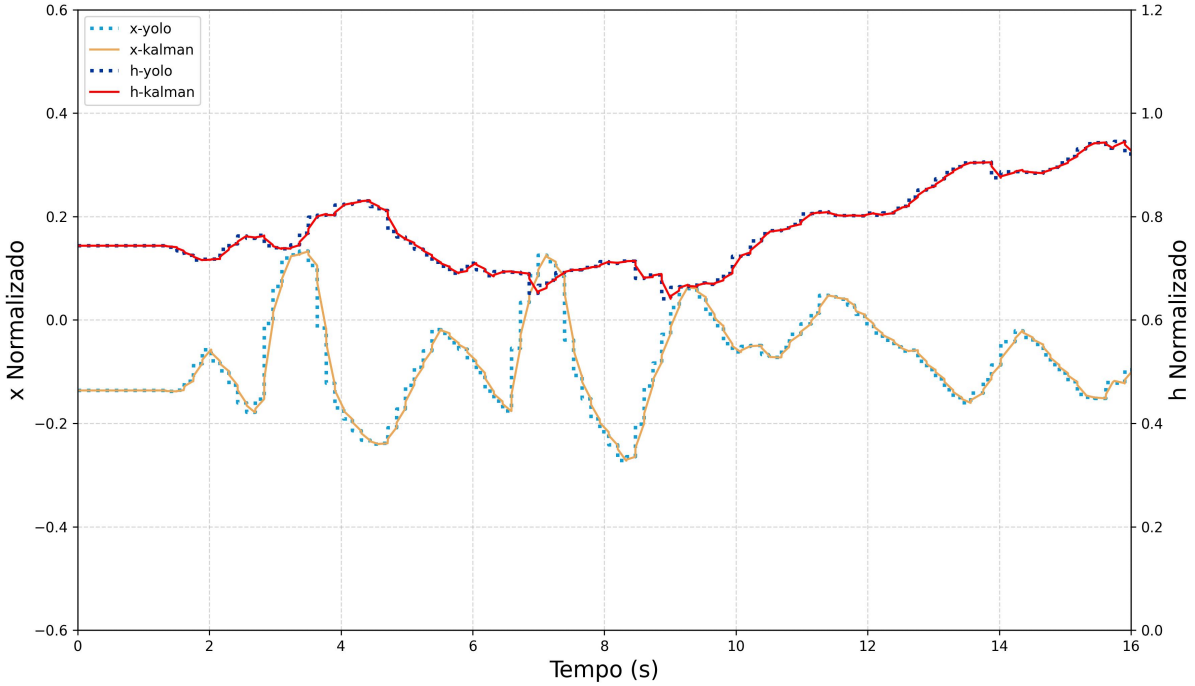
Nesse gráfico pode ser analisado de duas formas, no eixo esquerdo e nas linhas referentes a posição, a correção da orientação utilizando o estado Rotacionar. Enquanto no eixo direito e na linha referente a altura, é exibido a correção de distância por meio do estado Transladar.

4.1.3 Aplicação da filtragem na determinação orientação e distância relativas

O desempenho do sistema de rastreamento é analisado pela sua capacidade de identificar e estimar a trajetória do alvo. As curvas referentes à detecção do YOLO11 em comparação com a curva atenuada da filtragem estocástica podem ser analisadas nas

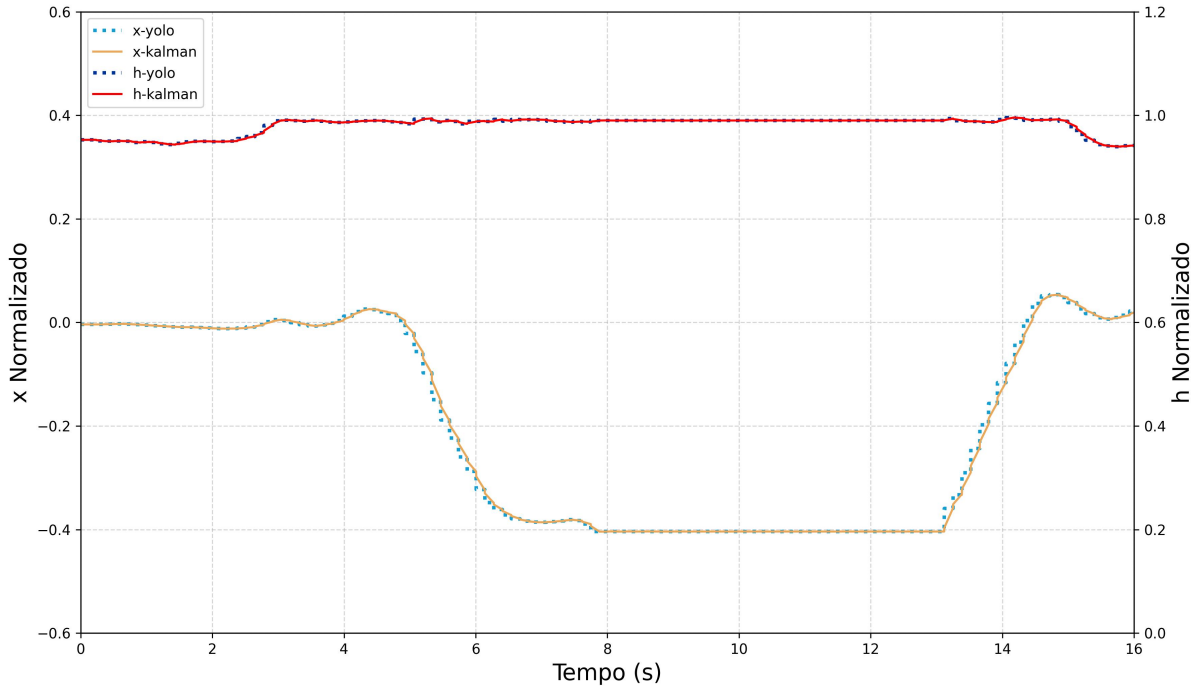
Figuras 32 e 33, representando respectivamente, o comportamento temporal de uma rotina de rastreamento sem perda do alvo e a outra com o alvo sendo perdido no trecho entre 8 e 13 segundos, no qual se comportou como uma reta horizontal, esse fenômeno indica que o sistema entrou no estado Buscar.

Figura 32 – YOLO x Kalman: Teste de Funcionamento



Fonte: O Autor (2026).

Figura 33 – YOLO x Kalman: Teste de Funcionamento com estado Buscar



Fonte: O Autor (2026).

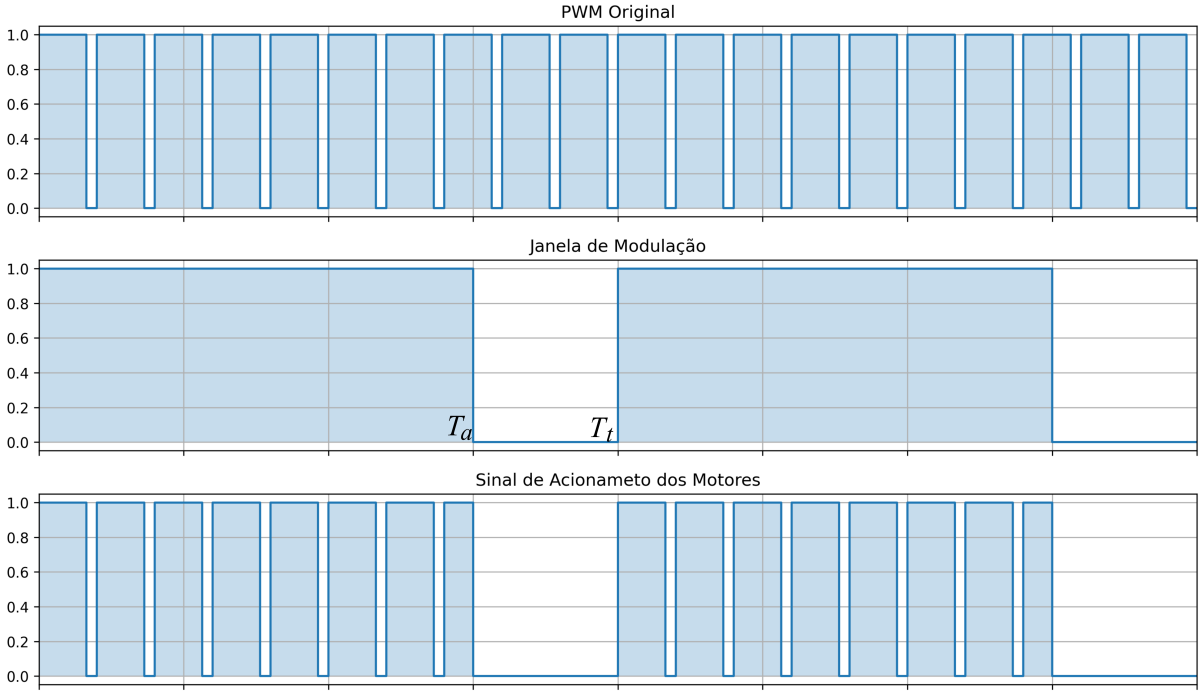
O comportamento do primeiro gráfico (Figura 32) indica que a estimativa do filtro de Kalman acompanha a dinâmica de movimento do robô sujeita a variações de posição do alvo, inclusive com mudanças bruscas de direção. Enquanto no segundo gráfico (Figura 33) é possível analisar a reaquisição do alvo, bem como a correção da orientação (linha laranja).

4.2 Análise do sistema de Hardware

No processo de montagem e nos primeiros testes, verificou-se que o torque de partida inicial não foi suficiente para superar o atrito estático do robô ao utilizar o PWM como sistema de controle (em conjunto com uma ação Proporcional-Derivativa). Tal fator deve-se à massa de aproximadamente 800 g (considerando todo o conjunto), aos efeitos de atrito estático e dinâmico e à baixa capacidade de torque dos motores.

Tal condição exigiu que o PWM operasse acima de 65 % do ciclo de trabalho para que o sistema pudesse vencer a inércia de movimento, resultando em uma faixa limitada de 35% para a atuação de algum controlador. Nesse caso, considerando a estreita faixa de operação do PWM, a inércia do sistema e os atrasos do sistema de visão computacional, tornou-se inviável a definição de um controlador linear para seguir de forma satisfatória o alvo — nesses casos, o robô entrava em estado de oscilação ou perdia muito facilmente a referência. Assim, como alternativa para a implementação do processo de controle, foi estabelecido um sistema duplo de modulação do sinal de controle.

Figura 34 – Sinal de comando PWM aplicado aos motores durante o estado de busca.



Fonte: O Autor (2026).

No processo de acionamento dos motores, foi definido um PWM de hardware fixado com *duty cycle* de 82%, como pode ser visto no primeiro gráfico da Figura 34. Sobre esse sinal, uma nova modulação foi aplicada, comportando-se como um segundo PWM de software, em que foi definida uma janela de modulação do sinal do PWM, que está apresentada no segundo gráfico da Figura 34. Esse sinal foi configurado em função da ação a ser exercida pelo robô, em que foram definidos os tempos T_t como tempo total da janela e T_a como o tempo de acionamento. Por fim, o sinal aplicado ao motor foi definido como o produto destes dois sinal, resultando na forma de onda apresentada no terceiro gráfico da Figura 34. Esta estratégia foi necessária para para vencer os efeitos do atrito estático do sistema e compensar a baixa potência dos motores.

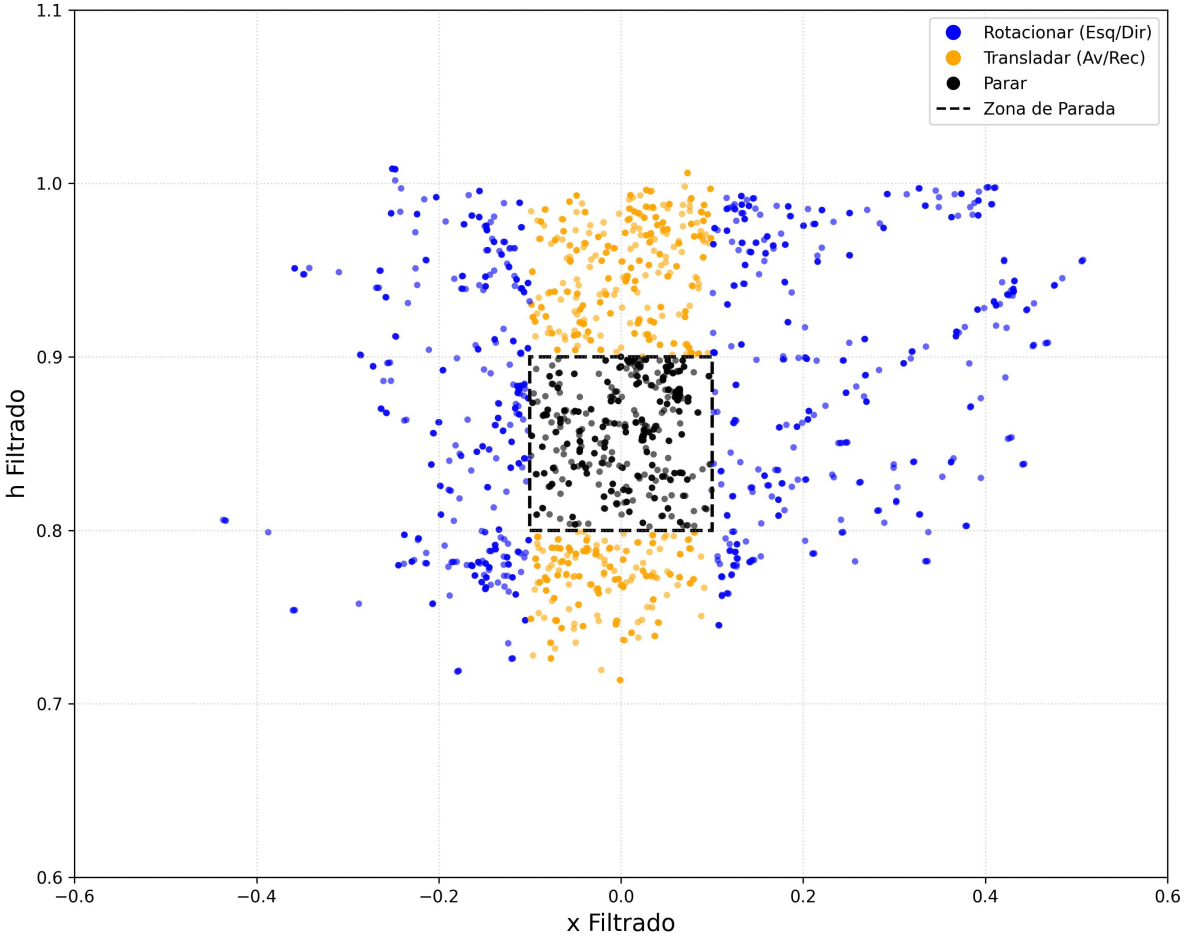
Durante o estado de rotação, os motores operam com janela total de $T_t = 70$ ms e tempo de acionamento $T_a = 30$ ms. Já para o estado de Buscar, foi implementada uma janela de modulação em que $T_t = 1000$ ms e tempo de acionamento $T_a = 750$ ms.

É importante notar que no estado de Busca o sistema permanece parado (Janela de Modulação em nível baixo) durante 250 ms. Este intervalo de tempo é necessário para garantir que o sistema de visão computacional realize a detecção com a câmera estacionada e não causar distorção (*motion blur*).

4.2.1 Controle dos Motores

Para demonstrar o funcionamento dos atuadores, a Figura 35 permite visualizar a transição entre as zonas de decisão do algoritmo de navegação em cada ciclo.

Figura 35 – Mapeamento de Estados

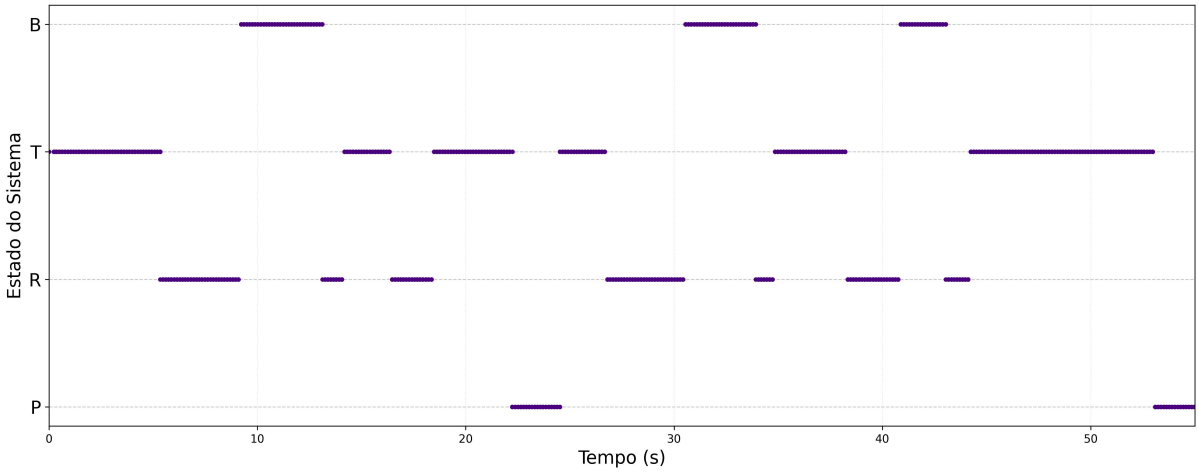


Fonte: O Autor (2026).

Esse mapa ilustra por zonas os estados de operação da FSM, com exceção do estado Buscar. A concentração de pontos no estado Parar foi mostrada na cor preta, enquanto o estado Rotacionar foi exibido na cor azul, indicando que, do lado esquerdo, a rotação é para a esquerda e, do lado direito, a rotação é para a direita. Já a cor laranja indica o estado Transladar, na qual a parte superior indica que a altura da caixa delimitadora é maior, logo o robô recua, e na parte inferior a altura é menor; assim, o robô avança.

A Figura 36 mostra o funcionamento da transição dos estados FSM durante uma rotinas de testes. O sistema transita entre diversos estados, inclusive em momentos de perda, quando o estado atual é Buscar. Nessa linha do tempo, pode ser visto que o sistema readquiriu o alvo nas 3 oportunidades.

Figura 36 – Linha do Tempo dos Estados



Fonte: O Autor (2026).

5 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e a implementação do robô móvel autônomo, que demonstrou ter a capacidade de rastrear pessoas em ambientes não estruturados. A proposta buscou a integração de conceitos como Visão Computacional, Deep Learning, Filtragem estocástica, controle de sistemas e a comunicação, validando a aplicação prática de teorias fundamentais da Engenharia Mecatrônica.

O desenvolvimento deste projeto trouxe alguns desafios, como a otimização do algoritmo de visão computacional para execução na Raspberry Pi, o que acarretou na implementação de estratégias como pular quadros do fluxo de imagens para que a YOLO11 não processe todos os quadros, gerando menos latência no sistema.

A capacidade do robô de interpretar dados visuais e traduzi-los em comandos motores imediatos confirmou a viabilidade de utilizar modelos de one-stage detection para navegação autônoma em dispositivos de baixo custo, fora do eixo estritamente industrial.

Este projeto mostrou que a integração entre IA e robótica móvel está cada vez mais acessível. O desenvolvimento desse projeto trouxe uma grande experiência prática para futuros projetos de engenharia, ao reforçar o estudo multidisciplinar, em diversas áreas do conhecimento que formam um engenheiro mecatrônico.

A experiência adquirida na resolução dos desafios de integração entre o meio físico e digital reforçou a importância do caráter multidisciplinar da engenharia, unindo programação de alto nível e eletrônica de potência.

Um impasse encontrado foi necessário um alto torque para vencer o atrito estático, que acarretou numa baixa margem para a operação de um sistema de controle contínuo, isso se deve a massa dos componentes do protótipo e ao torque dos motores.

Para evolução deste projeto, propõe-se para trabalhos futuros o aprimoramento do protótipo. A principal sugestão é a substituição dos atuadores para motores com maior torque equipados com *encoders* incrementais, impactando na substituição da lógica de controle de Máquina de Estados Finitos por um controlador de malha fechada, como o PID, proporcionando ao sistema um rastreamento mais fluido. Outro ponto de melhoria físico é a substituição do módulo L298n por drivers de potência baseados em tecnologias MOSFET, o que reduziria perdas por queda de tensão.

A calibração da câmera por meio da implementação de modelos de calibração permite corrigir distorções via parâmetros intrínsecos e extrínsecos, proporcionando ao sistema converter a unidade digital (*pixel*) em unidades físicas como distância e ângulo de orientação.

Para melhor a interação e o controle do protótipo, pode ser implementada sistemas de inicialização e/ou finalização do rastreamento com a utilização de gestos de mãos. Por fim, outra sugestão é a substituição das Raspberry Pi e *webcam* por um *smartphone*, integrando sensor visual, algoritmo de visão, algoritmo de controle e comunicação no mesmo dispositivo.

REFERÊNCIAS

AKBAR, J. U. M. et al. A comprehensive review on deep learning assisted computer vision techniques for smart greenhouse agriculture. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 12, p. 4485–4522, 2024. ISSN 2169-3536. Disponível em: https://dx.doi.org/10.1109/ACCESS.2024.3349418.

ALIF, M. A. R. Yolov11 for vehicle detection: Advancements, performance, and applications in intelligent transportation systems. *arXiv preprint arXiv:2410.22898*, 2024.

ALMEIDA, T. J. T. de. *Implementação de robô móvel para rastreio de pessoa com visão computacional*. GitHub, 2026. Projeto de Fim de Curso (Engenharia Mecatrônica) – Universidade Federal de Uberlândia, Uberlândia. Disponível em: https://github.com/thallesj5/UFU_Engenharia_Mecatronica_PFC. Acesso em: 29 abr. 2026.

ALOM, M. Z. et al. *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. arXiv, 2018. Disponível em: https://arxiv.org/abs/1803.01164.

ALOYSIUS, N.; GEETHA, M. A review on deep convolutional neural networks. In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. [S.l.: s.n.], 2017. p. 0588–0592.

AMARAL, H. N.; GASPAROTTO, A. M. S. Inteligência artifical: o uso da robótica indústria 4.0. *Revista Interface Tecnológica*, Interface Tecnologica, v. 18, n. 1, p. 474–486, nov. 2021. ISSN 1807-3980. Disponível em: https://dx.doi.org/10.31510/infa.v18i1.1107.

ATMOKO, R. A.; RIANTINI, R.; HASIN, M. K. IoT real time data acquisition using MQTT protocol. *Journal of Physics: Conference Series*, IOP Publishing, v. 853, p. 012003, maio 2017. ISSN 1742-6596. Disponível em: https://dx.doi.org/10.1088/1742-6596/853/1/012003.

BARELLI, F. *Introdução à visão computacional: Uma abordagem prática com Python e OpenCV*. [S.l.]: Editora Casa do Código, 2018.

BAR-SHALOM, Y.; LI, X.; KIRUBARAJAN, T. *Estimation with Applications to Tracking and Navigation: Theory, Algorithms and Software*. Wiley, 2002. ISBN 9780471221272. Disponível em: https://dx.doi.org/10.1002/0471221279.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. Disponível em: https://opencv.org/.

BROWN, R. G.; HWANG, P. Y. C. *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises*. 3. ed. [S.l.]: John Wiley and Sons (WIE), 1997. ISBN 0-471-12839-2.

DELAI, R. L.; COELHO, A. D. *Visão computacional com a OpenCV–material apostilado e veículo seguidor autônomo*. [S.l.]: Instituto Mauá de Tecnologia, Mauá, 2012.

DINCULEANA, D.; CHENG, X. Vulnerabilities and Limitations of MQTT Protocol Used between IoT Devices. *Applied Sciences*, MDPI AG, v. 9, n. 5, p. 848, fev. 2019. ISSN 2076-3417. Disponível em: https://dx.doi.org/10.3390/app9050848.

FADALI, M. S. *Introduction to random signals, estimation theory, and Kalman filtering*. [S.l.]: Springer Nature, 2024.

FENG, S. et al. A review: state estimation based on hybrid models of kalman filter and neural network. *Systems Science amp; Control Engineering*, Informa UK Limited, v. 11, n. 1, fev. 2023. ISSN 2164-2583. Disponível em: https://dx.doi.org/10.1080/21642583.2023.2173682.

FLECK, L. et al. Redes neurais artificiais: Princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, v. 1, n. 13, p. 47–57, 2016.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: https://www.deeplearningbook.org. Disponível em: https://www.deeplearningbook.org. Acesso em: 18 fev. 2026.

HAYKIN, S. *Redes neurais: princípios e prática*. 2. ed. [S.l.]: Bookman Editora, 2001.

HE, L.-h. et al. Research on object detection and recognition in remote sensing images based on yolov11. *Scientific Reports*, Nature Publishing Group UK London, v. 15, n. 1, p. 14032, 2025.

HE, Z. et al. Comprehensive performance evaluation of yolov11, yolov10, yolov9, yolov8 and yolov5 on object detection of power equipment. In: IEEE. *2025 37th Chinese Control and Decision Conference (CCDC)*. [S.l.], 2025. p. 1281–1286.

HOWSE, J. *OpenCV computer vision with python*. [S.l.]: Packt Publishing Birmingham, UK, 2013. v. 27.

JIANG, P. et al. A review of yolo algorithm developments. *Procedia computer science*, Elsevier, v. 199, p. 1066–1073, 2022.

JOCHER, G.; QIU, J. *Ultralytics YOLO11*. 2024. Disponível em: https://github.com/ultralytics/ultralytics. Acesso em: 18 fev. 2026.

KALMAN, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, v. 82, n. 1, p. 35–45, 1960.

KHANAM, R.; HUSSAIN, M. Yolov11: An overview of the key architectural enhancements. 2024. Disponível em: https://arxiv.org/abs/2410.17725. Acesso em: 18 fev. 2026.

KIM, Y.; BANG, H. Introduction to kalman filter and its applications. In: *Introduction and implementations of the Kalman filter*. [S.l.]: IntechOpen, 2018.

KOTTHAPALLI, M.; RAVIPATI, D.; BHATIA, R. Yolov1 to yolov11: A comprehensive survey of real-time object detection innovations and challenges. *arXiv preprint arXiv:2508.02067*, 2025.

LEE, E. A.; SESHIA, S. A. *Introduction to embedded systems: A cyber-physical systems approach*. [S.l.]: MIT press, 2016.

LEE, Y.-S. et al. Hyperparameter optimization for tomato leaf disease recognition based on yolov11m. *Plants*, MDPI AG, v. 14, n. 5, p. 653, fev. 2025. ISSN 2223-7747. Disponível em: https://dx.doi.org/10.3390/plants14050653.

OGATA, K. *Engenharia de controle moderno. 5ª*. [S.l.]: São Paulo: Pearson, 2010.

ORLANDINI, G. *Desenvolvimento de Aplicativos Baseados em Técnicas de Visão Computacional para Robô Móvel Autônomo*. Dissertação (Dissertação (Mestrado em Ciência da Computação)) — Universidade Metodista de Piracicaba, Piracicaba, 2012.

PEERZADA, P.; LARIK, W. H.; MAHAR, A. A. Dc motor speed control through arduino and l298n motor driver using pid controller. *International Journal of Electrical Engineering & Emerging Technology*, v. 4, n. 2, p. 21–24, 2021.

PUTHIYIDAM, J. J.; JOSEPH, S. Internet of things network performance: Impact of message and client sizes and reliability levels. *ECTI Transactions on Electrical Engineering, Electronics, and Communications*, ECTI, v. 22, n. 1, fev. 2024. ISSN 1685-9545. Disponível em: https://dx.doi.org/10.37936/ecti-eec.2024221.252941.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao processamento digital de imagens. *Rita*, v. 13, n. 2, p. 11–42, 2006.

RASHEED, A. F.; ZARKOOSH, M. Yolov11 optimization for efficient resource utilization. *The Journal of Supercomputing*, Springer, v. 81, n. 9, p. 1–21, 2025.

RASHID, M. H. *Power electronics handbook*. [S.l.]: Butterworth-heinemann, 2017.

SIEGWART ROLAND E NOURBAKHSH, I. R. *Introdução a Robôs Móveis Autônomos*. Londres, Inglaterra: MIT Press, 2004. (Robôs inteligentes e agentes autônomos).

SOHAN, M.; RAM, T. S.; REDDY, C. V. R. A review on yolov8 and its advancements. In: SPRINGER. *International Conference on Data Intelligence and Cognitive Informatics*. [S.l.], 2024. p. 529–545.

SOLOMON, J. *Numerical algorithms: methods for computer vision, machine learning, and graphics*. [S.l.]: CRC press, 2015.

TANENBAUM, A. S. Network protocols. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 13, n. 4, p. 453–489, dez. 1981. ISSN 1557-7341. Disponível em: https://dx.doi.org/10.1145/356859.356864.

TERVEN, J.; CóRDOVA-ESPARZA, D.-M.; ROMERO-GONZáLEZ, J.-A. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, MDPI AG, v. 5, n. 4, p. 1680–1716, nov. 2023. ISSN 2504-4990. Disponível em: https://dx.doi.org/10.3390/make5040083.

UPTON, E.; HALFACREE, G. *Raspberry Pi user guide*. [S.l.]: John Wiley & Sons, 2016.

VOULODIMOS, A. et al. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, Wiley, v. 2018, p. 1–13, 2018. ISSN 1687-5273. Disponível em: https://dx.doi.org/10.1155/2018/7068349.

WANG, B. et al. Kalman filter and its application in data assimilation. *Atmosphere*, v. 14, n. 8, 2023. ISSN 2073-4433. Disponível em: https://www.mdpi.com/2073-4433/14/8/1319.

WOLF, M. *Computers as components: principles of embedded computing system design*. [S.l.]: Elsevier, 2012.

ZHONG, R. Y. et al. Intelligent manufacturing in the context of industry 4.0: A review. *Engineering*, Elsevier BV, v. 3, n. 5, p. 616–630, out. 2017. ISSN 2095-8099. Disponível em: https://dx.doi.org/10.1016/J.ENG.2017.05.015.

APÊNDICE A

Código do sistema embarcado executado na Raspberry Pi 5, disponível em Almeida (2026).

```
# ======================================================
# Lista de Bibliotecas
import cv2
from ultralytics import YOLO
import numpy as np
import time
from filterpy.kalman import KalmanFilter
from filterpy.common import Q_discrete_white_noise
import paho.mqtt.client as mqtt


# ======================================================
# Gráficos
import csv

# Nome do arquivo para o Gráfico 1
arquivo_csv = "dados2.csv"
f = open(arquivo_csv, mode='w', newline='')
writer = csv.writer(f)
writer.writerow(['tempo', 'x_yolo', \
    'x_kalman', 'h_yolo', 'h_kalman', \
    'dt', 'busca'])

x_norm = 0
x_filtrado = 0
h_norm = 0
h_filtrado = 0
tempo_exp = 0

# ======================================================
# Topicos mqtt
mqtt_broker = "localhost"
mqtt_id = "Raspberry_pfc"
mqtt_topico_comando = "pfc/comandos"
mqtt_topico_controle = "pfc/controle"
mqtt_topico_config = "pfc/config"

# ======================================================
# Dimensões da Tela
LARGURA_TELA = 320
ALTURA_TELA = 240
# Variaveis de Controle
h_min = 0.8
```

```
h_max = 0.9
DEADBAND_ERRO = 0.1
v_max = 255
v_esq = 0
v_dir = 0
v_base = 210 #210
vel_rotacao = 30 #30
vel_busca = 30 #30

# ====================================================
# Variaveis de Loop
FRAMES_PARA_PULAR = 3
alvo_id = None
frame_counter = 0
tempo_inferencia = 0
resultados = None


# ====================================================
# Modo Buscar
LIMITE_BUSCA = 40 #40
perdas_consecutivas = 0
busca = 0

# ====================================================
# MQTT
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conectado ao Broker!")
        client.publish(mqtt_topico_config, f"{vel_rotacao},\
        {vel_busca}", retain=True, qos=1)
    else:
        print(f"Falha na conexão, código: {rc}")

# Cliente MQTT
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, \
client_id=mqtt_id)
client.on_connect = on_connect
client.connect(mqtt_broker, 1883, 60)
client.loop_start()

# ====================================================
# Camera
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, LARGURA_TELA)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, ALTURA_TELA)

# Yolo
model = YOLO('yolo11n.pt')
```

```
# ======================================================
# Configuração Filto de Kalman
dt = 0.033

# Posição x
kf_x = KalmanFilter(dim_x=2, dim_z=1)
kf_x.x = np.array([[0.],
                   [0.]])
kf_x.F = np.array([[1., dt],
                   [0., 1.]])
kf_x.H = np.array([[1., 0.]])
kf_x.P = np.eye(2) * 10.
kf_x.R = np.array([[0.36e-5]])
sigma_ax = 1.5
kf_x.Q = sigma_ax**2 * np.array([
    [dt**4/4, dt**3/2],
    [dt**3/2, dt**2]
])

# ======================================================

# Altura h
kf_h = KalmanFilter(dim_x=2, dim_z=1)
kf_h.x = np.array([[0.9],
                   [0.]])
kf_h.F = np.array([[1., dt],
                   [0., 1.]])
kf_h.H = np.array([[1., 0.]])
kf_h.P = np.eye(2) * 5.
kf_h.R = np.array([[59.29e-5]])
sigma_ah = 5
kf_h.Q = sigma_ah**2 * np.array([
    [dt**4/4, dt**3/2],
    [dt**3/2, dt**2]
])

# ======================================================

time.sleep(4)
try:
    if cap.isOpened():
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, LARGURA_TELA)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, ALTURA_TELA)
        width = cap.get(cv2.CAP_PROP_FRAME_WIDTH)
        height = cap.get(cv2.CAP_PROP_FRAME_HEIGHT)
        success, frame = cap.read()
        print(f"Frame width set to: {width}, \
        Frame height set to: {height}")
```

```
tk_0 = time.time()
tk_1 = 0

while cap.isOpened():
    tk_utc = time.time()
    tk = tk_utc - tk_0
    dt = tk - tk_1
    tk_1 = tk
    success, frame = cap.read()
    if not success:
        break
    frame = cv2.resize(frame, (320, 240))

    # Cálculo Dinâmico

    kf_x.F[0, 1] = dt
    kf_x.Q = sigma_ax**2 * np.array([
        [dt**4/4, dt**3/2],
        [dt**3/2, dt**2]
    ])
    kf_x.predict()

    kf_h.F[0, 1] = dt
    kf_h.Q = sigma_ah**2 * np.array([
        [dt**4/4, dt**3/2],
        [dt**3/2, dt**2]
    ])
    kf_h.predict()


    alvo_encontrado_neste_frame = False

    # Processamento YOLO
    if frame_counter % (FRAMES_PARA_PULAR + 1) == 0:
        resultados = model.track(frame, classes=0, \
        persist=True, verbose=False, imgsz=320)

        if resultados[0].boxes.id is not None:
            ids_detectados = resultados[0].boxes.id.int().cpu().tolist()

            if alvo_id is None or (busca == 1):
                alvo_id = ids_detectados[0]

            for i, track_id in enumerate(ids_detectados):
                if track_id == alvo_id:
                    alvo_encontrado_neste_frame = True
                    perdas_consecutivas = 0
                    x_pixel, _, _, h_pixels = resultados[0].boxes.xywh[i]
```

```
                busca = 0

                # Normalização
                x_norm = (x_pixel - LARGURA_TELA / 2) / LARGURA_TELA
                h_norm = float(h_pixels) / ALTURA_TELA
                # Atualização do Filtro de
                kf_x.update(np.array([[x_norm]]))
                kf_h.update(np.array([[h_norm]]))


#Estado Buscar
if not alvo_encontrado_neste_frame and alvo_id is not None:
    perdas_consecutivas += 1
    kf_x.update(np.array([[x_norm]]))
    kf_h.update(np.array([[h_norm]]))
    if perdas_consecutivas > LIMITE_BUSCA:
        busca = 1
        kf_x.update(np.array([[x_norm]]))
        kf_h.update(np.array([[h_norm]]))
        client.publish(mqtt_topico_comando, f"{busca}", qos=0)
        print("\nComando: Buscar")

frame_counter += 1
x_filtrado = kf_x.x[0].item()
h_filtrado = kf_h.x[0].item()

#Salvando itens nos gráficos
writer.writerow([f"{tk:.4f}", f"{x_norm:.4f}", \
f"{x_filtrado:.4f}", f"{h_norm:.4f}", \
f"{h_filtrado:.4f}",f"{dt:.4f}", f"{busca}"])

# FSM
if busca == 0:
    if x_filtrado >= DEADBAND_ERRO:
        #print('Direita!\n')
        v_esq = v_base
        v_dir = -v_base
    elif x_filtrado <= -DEADBAND_ERRO:
        #print('Esquerda!\n')
        v_esq = -v_base
        v_dir = v_base
    elif h_filtrado > h_max:
        #print('Recua')
        v_esq = -v_base
        v_dir = -v_base
    elif h_filtrado <= h_min:
        #print('Avança!')
        v_esq = v_base
        v_dir = v_base
```

```
            else:
                #print('Parado')
                v_esq = 0
                v_dir = 0

            # Enviando ao MQTT
            client.publish(mqtt_topico_controle, f"{v_esq},{v_dir}", qos=0)

        # Enviando ao MQTT
        client.publish(mqtt_topico_comando, f"{busca}", qos=0)
        # VISUALIZAÇÃO
        print(f"tempo:{tempo_exp:.3f}, x_norm: {x_norm:.4f}, \
        x_kalman:{x_filtrado:.4f}, h_norm: {h_norm:.4f}, h_kalman: \
        {h_filtrado:.4f}, dt: {1000 * dt:.0f}, busca: {busca}")

except KeyboardInterrupt:
    print("\n[SISTEMA] Encerrando...")
    client.publish(mqtt_topico_controle, "0,0", qos=1)
    client.disconnect()
    time.sleep(2)
finally:
    cap.release()
    client.loop_stop()
    client.disconnect()
```

APÊNDICE B

Código do sistema embarcado executado na ESP32, disponível em Almeida (2026).

```
//=====================================================================

// Bibliotecas Utilizadas
#include <WiFi.h>
#include <PubSubClient.h>

//=====================================================================

// Configurações de Rede e Tópicos
const char* password = "0507112533";
//Raspberry: 192.168.0.102 Note: 192.168.0.100
const char* mqtt_server = "192.168.0.102";
const char* ssid = "rede_pfc";

const int mqtt_port = 1883;
const char* mqtt_id = "ESP32_pfc";

//Tópicos MQTT
const char* mqtt_topico_controle = "pfc/controle";
const char* mqtt_topico_comando = "pfc/comando";
const char* mqtt_topico_config = "pfc/config";

//=====================================================================

unsigned long last_msg_time = 0;
const unsigned long timeout_mqtt = 20000;

//=====================================================================

// Pinos e PWM
const int ENA = 14; const int IN1 = 27; const int IN2 = 26;
const int IN3 = 25; const int IN4 = 33; const int ENB = 32;
const int motor_esq = 0;
const int motor_dir = 1;
int v_esq = 0;
int v_dir = 0;
int pwm_esq = 0;
int pwm_dir = 0;
unsigned long ms;
const int pwm_frequency = 1000;
const int pwm_resolution = 8;
float comp_dir = 0.98;

//=====================================================================
```

```
// Modo Busca e rotação pura
const int ciclo_burst = 1000;
const int ciclo_rotacao = 100;
const int ligado_burst = 750;
int ligado_rotacao = 30;
int ligado_busca = 30;
const int pwm_busca = 210;
int busca = 0;
unsigned long burst_ms;
int ultima_direcao = 0;

//=====================================================================

WiFiClient espClient;
PubSubClient client(espClient);
void reconnect() {
  static unsigned long lastReconnectAttempt = 0;
  if (millis() - lastReconnectAttempt > 5000) {
    lastReconnectAttempt = millis();
    if (client.connect(mqtt_id)) {
      client.subscribe(mqtt_topico_controle);
      client.subscribe(mqtt_topico_comando);
      client.subscribe(mqtt_topico_config);
    }
  }
}

//=====================================================================

void callback(char* topic, byte* payload, unsigned int length) {

  char msg[length + 1];
  memcpy(msg, payload, length);
  msg[length] = '\0';

  String topicoStr = String(topic);
  last_msg_time = millis(); // Reseta o timeout de segurança

  //Velocidade
  if (topicoStr == mqtt_topico_controle) {
    int v1, v2;
    // Velocidades: V_esq e V_dir
    int res = sscanf(msg, "%d,%d", &v1, &v2);
    if (res == 2) {
      v_esq = v1;
      v_dir = v2;
    }
    if (v_esq > v_dir) {
```

```
        ultima_direcao = 0; // Direita
      }
    else if (v_dir > v_esq) {
      ultima_direcao = 1; // Esquerda
    }
  }

  //Parâmetros de Configuração
  else if (topicoStr == mqtt_topico_config) {
    int r, b;
    //ligado_rotacao e ligado_busca)
    int res = sscanf(msg, "%d,%d", &r, &b);
    if (res == 2) {
      ligado_rotacao = r;
      ligado_busca = b;
    }
  }

//=========================================================================


  // Ativa e Desativa Estado Buscar
  else if (topicoStr == mqtt_topico_comando) {
    busca = atoi(msg);
  }
}

//=========================================================================

// Comandos do Motor
void moveForward() {
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  }
void moveBackward() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  }
void turnLeft() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  }
void turnRight() {
```

```
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  }
void stopMotors() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
  }

//=====================================================================

void setup() {
  Serial.begin(115200);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  ledcSetup(motor_esq, pwm_frequency, pwm_resolution);
  ledcAttachPin(ENA, motor_esq);
  ledcSetup(motor_dir, pwm_frequency, pwm_resolution);
  ledcAttachPin(ENB, motor_dir);


//=====================================================================


  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) delay(500);
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
}

//=====================================================================

void loop() {
  if (!client.connected()) {
    reconnect();
  } else {
    client.loop();
  }

  // Timeout de segurança
  if (millis() - last_msg_time > timeout_mqtt) {
    v_esq = 0; v_dir = 0; busca = 0;
  }
```

```
unsigned long ms = millis();
pwm_esq = 0;
pwm_dir = 0;

// 1. Estado Buscar
if (busca == 1) {
  burst_ms = ms % ciclo_burst;
  if (burst_ms < ligado_burst) {
    if (ultima_direcao == 1) {
      turnLeft();
    } else {
      turnRight();
    }
    if ((burst_ms % ciclo_rotacao) < ligado_busca) {
      pwm_esq = pwm_busca;
      pwm_dir = pwm_busca;
    } else { stopMotors(); }
  } else { stopMotors(); }
}

// 2. Estado Parar
else if (v_esq == 0 && v_dir == 0) {
  stopMotors();
  pwm_esq = 0;
  pwm_dir = 0;
}

// Movivemntação Transladar e Rotacionar
else {
  // Estado Transladar: Avançar e Recuar
  if (v_esq == v_dir) {
    if (v_esq > 0) moveForward();
    else moveBackward();
    pwm_esq = abs(v_esq);
    pwm_dir = abs(v_dir);
  }
  // Estado Rotacionar: Virar Direita/Esquerda
  else {
    if (v_esq < v_dir) turnLeft();
    else turnRight();


    if ((ms % ciclo_rotacao) < ligado_rotacao) {
      pwm_esq = abs(v_esq);
      pwm_dir = abs(v_dir);
    } else {
      stopMotors();
      pwm_esq = 0;
```

```
        pwm_dir = 0;
      }
    }
  }

  // Ajustes finais e PWM
  int final_pwm_esq = constrain(pwm_esq, 0, 255);
  int final_pwm_dir = constrain(round(pwm_dir * comp_dir), 0, 255);

  ledcWrite(motor_esq, final_pwm_esq);
  ledcWrite(motor_dir, final_pwm_dir);
}
```