INSTITUTO FEDERAL DA PARAÍBA
COORDENAÇÃO DO CURSO SUPERIOR DE BACHARELADO EM ENGENHARIA ELÉTRICA

IGOR ALEXANDRE STEFAN

Sistema de Visão Computacional com Ajuste Dinâmico de Câmera e Detecção de Retas para Verificação de Alinhamento

JOÃO PESSOA - PB
2026

IGOR ALEXANDRE STEFAN

Sistema de Visão Computacional com Ajuste Dinâmico de Câmera e Detecção de Retas para Verificação de Alinhamento

Trabalho de conclusão de curso submetido à Coordenação do Curso Superior de Bacharelado em Engenharia Elétrica do Instituto Federal da Paraíba, como parte dos requisitos para a obtenção do grau de engenheiro eletricista.

Orientador: Dr. Ruan Delgado Gomes

JOÃO PESSOA - PB

2026

Dados Internacionais de Catalogação na Publicação – CIP
Biblioteca Nilo Peçanha – IFPB, *Campus* João Pessoa

S816s Stefan, Igor Alexandre.
Sistema de visão computacional com ajuste dinâmico de câmera e detecção de retas para verificação de alinhamento / Igor Alexandre Stefan. – 2026.
62 f. : il.
TCC (Curso Superior de Bacharelado em Engenharia Elétrica) – Instituto Federal de Educação da Paraíba / Unidade Acadêmica de Controle e Processos Industriais / Coordenação do Curso Superior de Bacharelado em Engenharia Elétrica, 2026.
Orientador: Profº Dr. Ruan Delgado Gomes.
1. Processamento de imagens. 2. V4L2. 3. OpenCV. 4. Detecção de bordas (Canny). 5. Transformada de Hough. I. Título.
CDU 004.932(043)

Bibliotecária responsável Lucrecia Camilo de Lima – CRB 15/132

IGOR ALEXANDRE STEFAN

SISTEMA DE VISÃO COMPUTACIONAL COM AJUSTE DINÂMICO DE CÂMERA E DETECÇÃO DE RETAS PARA VERIFICAÇÃO DE ALINHAMENTO

TRABALHO DE CONCLUSÃO DE CURSO submetido a Coordenação do Curso Superior de Bacharelado em Engenharia Elétrica, do Instituto Federal da Paraíba (IFPB), como parte dos requisitos institucionais para a obtenção do Título de **ENGENHEIRO ELETRICISTA.**

Aprovado em 11/02/2026.

Membros da Banca Examinadora:

Prof. Dr. RUAN DELGADO GOMES
Instituto Federal da Paraíba
Orientador

Prof. Dr. OTACÍLIO DE ARAÚJO RAMOS NETO
Instituto Federal da Paraíba
Examinador

Prof. Dr. RAFAEL FRANKLIN ALVES SILVA
Instituto Federal da Paraíba
Examinador

Dedico este trabalho à minha mãe, pelo apoio inabalável e incentivo constante.

RESUMO

Este trabalho apresenta o desenvolvimento de um módulo de inspeção visual para verificação de alinhamento angular em ambiente industrial, integrando aquisição controlável de imagens e processamento interpretável. Em Linux, a captura de imagem é realizada com OpenCV e os parâmetros de câmera (exposição, ganho, balanço de branco, brilho, contraste, etc.) são ajustados dinamicamente via V4L2, com persistência de configurações e salvamento do último frame para rastreabilidade e reprodutibilidade. A arquitetura organiza-se em três blocos: (i) aquisição e ajuste, operados por uma interface web; (ii) processamento, baseado em segmentação por cor em HSV, filtragem por mediana, detecção de bordas (Canny) e detecção de retas (Transformada de Hough); e (iii) exposição do resultado por endpoint HTTP/JSON, com repetição e agregação de execuções válidas para robustez. Um experimento com duas webcams e seis ângulos (0°–45°) avaliou a capacidade de detecção e o desempenho, medindo tempos por etapa e percentis p50/p95 em 300 frames por condição. As etapas do processamento e os custos de tempo são detalhados. Os resultados indicam fluxo de detecção bem-sucedido nas condições testadas e tempo típico por frame estável.

Palavras-chave: Processamento de imagens. V4L2. OpenCV. Detecção de bordas (Canny). Transformada de Hough. API REST.

ABSTRACT

This work presents a computer-vision inspection module to estimate the angular misalignment of a part with respect to fixed reference lines, targeting industrial setups. Under Linux, frames are captured with OpenCV while camera controls (exposure, gain, white balance, brightness, contrast, etc.) are adjusted at runtime through V4L2, with configuration persistence and last-frame saving to support traceability and reproducible debugging. The system is organized into three blocks: (i) acquisition and tuning via a web UI; (ii) image processing using HSV color segmentation, median filtering, Canny edge detection, and Hough line detection to compute the maximum angular deviation between object of interest and reference lines; and (iii) result publication through an HTTP/JSON endpoint that can repeat runs and aggregate valid outputs for robustness. An experiment with two webcams and six nominal angles (0°–45°) recorded intermediate steps and benchmarked processing time per frame (p50/p95) over 300 frames per condition. The pipeline completed detection in the evaluated scenarios and showed stable typical latency.

Keywords: Image processing. V4L2. OpenCV. Edge detection (Canny). Hough transform. REST API.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 ASPECTOS SOBRE A INSPEÇÃO VISUAL INDUSTRIAL

A inspeção visual está presente em uma ampla gama de processos industriais, desde a verificação dimensional e de acabamento superficial até a confirmação de posicionamento e alinhamento de componentes. Historicamente, parte dessas tarefas era realizada por inspeção humana; contudo, requisitos de repetibilidade, velocidade, rastreabilidade e integração com sistemas de automação têm impulsionado o uso de visão computacional (Szeliski, 2010).

Em linhas gerais, sistemas de visão computacional aplicados à indústria recebem imagens de sensores (câmeras) e, a partir delas, extraem informações relevantes para apoiar decisões: aprovar ou reprovar um item, estimar medidas geométricas, identificar desvios e orientar ações corretivas (Szeliski, 2010; Basler AG, 2026d,a).

No chão de fábrica, a preocupação com a qualidade da imagem obtida tem consequências diretas. Um algoritmo que funciona em um ambiente controlado pode falhar quando a cena muda ligeiramente: reflexos, sombras, alterações no espectro de iluminação, vibração mecânica e até mudanças no posicionamento da câmera podem modificar a aparência do *frame* (Henkart, 2022). Além disso, a própria câmera oferece controles (exposição, ganho, balanço de branco, brilho, contraste e nitidez, entre outros) que influenciam ruído, desfoque por movimento, faixa dinâmica e contraste das bordas (Henkart, 2022; Imatest, 2026).

Em diversos processos produtivos, o alinhamento (posição e orientação) de peças e componentes é um requisito de qualidade (National Instruments, 2026; Keyence, 2026). Desvios angulares podem comprometer encaixes e montagens e causar retrabalho, uma vez que peças com orientação incorreta podem avançar na linha e exigir correção em etapas posteriores (Cognex, 2026). Nesse contexto, estimar alinhamento a partir de imagem é valioso por permitir verificação rápida, rastreável e passível de integração com sistemas de controle (National Instruments, 2026; Keyence, 2026).

É necessário que o sistema produza imagens com aparência suficientemente estável e um processamento suficientemente robusto para que a orientação estimada seja consistente entre execuções, dentro de uma tolerância compatível com o processo (Henkart, 2022; Szeliski, 2010).

1.2 CONTEXTO E CENÁRIO DE APLICAÇÃO

A motivação deste trabalho está relacionada a um projeto desenvolvido no Polo de Inovação do IFPB. Nesse contexto, surgiu a necessidade prática de realizar inspeção visual em etapas específicas de um processo produtivo executado por uma máquina de

bancada. Em diferentes pontos do ciclo de produção, tornou-se necessário verificar se a peça permanecia alinhada angularmente antes de permitir a continuidade do processo.

Para viabilizar essa verificação, posicionou-se uma câmera acima do plano de trabalho, a aproximadamente 1,5 m de distância, com suportes e mecanismos de fixação projetados para mitigar vibrações e evitar rotações indesejadas. Além disso, duas retas de referência foram fixadas no campo de visão, representando a direção "base" esperada para comparação.

Além da estimação do desalinhamento, o cenário impôs requisitos de integração. O valor calculado deve ser disponibilizado para consumo por outros componentes do sistema de automação, tipicamente por meio de uma interface de software (por exemplo, um *endpoint* HTTP/JSON).

1.3 SÍNTESE DO PROBLEMA

Dessa forma, o problema central consiste em **estabelecer uma aquisição de imagens controlável e reprodutível** que favoreça algoritmos de detecção de retas e, a partir disso, permita verificar o alinhamento. Complementarmente, é necessário **disponibilizar o resultado do processamento** de modo a viabilizar a integração com sistemas externos.

Para atender a esse objetivo, a abordagem adotada mantém duas preocupações em equilíbrio: (i) tornar a captura controlável e rastreável (para reduzir variabilidade entre *frames*) e (ii) empregar um processamento interpretável, capaz de produzir uma medida numérica de desalinhamento com comportamento estável no cenário estudado.

Portanto, este trabalho descreve o desenvolvimento de um módulo de inspeção visual baseado em software, capaz de: (i) capturar imagens por meio de uma câmera conectada ao Linux; (ii) ajustar dinamicamente parâmetros de aquisição (por exemplo, exposição, balanço de branco, ganho, brilho e contraste) disponibilizados pelo *driver* do dispositivo; (iii) executar rotinas de processamento de imagem para detecção de retas e estimativa do desvio de alinhamento entre um objeto e uma referência; e (iv) disponibilizar o resultado por meio de uma interface HTTP/JSON.

1.4 OBJETIVOS

1.4.1 Objetivo geral

Desenvolver um módulo de visão computacional capaz de ajustar dinamicamente parâmetros de uma câmera via V4L2 e executar algoritmos de detecção de retas para estimar o desvio de alinhamento de um objeto e disponibilizar o resultado por interface HTTP/JSON.

1.4.2 Objetivos específicos

- Implementar uma interface de configuração (frontend) que apresente, de forma organizada, os parâmetros ajustáveis da câmera e seus valores atuais.
- Encapsular a interação com o *hardware* de vídeo em uma camada de abstração, contemplando captura de *frames* e aplicação de configurações iniciais.
- Implementar um servidor Flask que receba requisições para atualizar parâmetros e para executar rotinas de processamento de imagem sob demanda.
- Desenvolver um algoritmo de detecção de retas baseado em segmentação por cor, extração de bordas (Canny) e Transformada de Hough, comparando ângulos de retas do objeto com retas de referência.
- Incorporar mecanismos de reprodutibilidade, como persistência de configurações e salvamento de *frames* para inspeção offline.

1.5 ORGANIZAÇÃO DO TRABALHO

O presente documento está organizado em capítulos que refletem a evolução do raciocínio. Além desta Introdução, o trabalho está organizado da seguinte forma: A Fundamentação Teórica apresenta os conceitos e tecnologias que sustentam o estudo, incluindo aquisição de imagens, processamento com OpenCV, detecção de bordas e retas, o subsistema V4L2 e interface HTTP/JSON. A Metodologia descreve o desenvolvimento do sistema, sua arquitetura em blocos e núcleos, o fluxo de execução e os mecanismos de ajuste, rastreabilidade e integração. Por fim, os capítulos finais apresentam os resultados obtidos no cenário estudado e discutem limitações e possibilidades de evolução, encerrando com as conclusões.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos e tecnologias que sustentam o sistema de inspeção visual proposto. O embasamento é organizado em três eixos: (i) **aquisição e controle de imagem** em Linux, (ii) **processamento digital de imagens** (segmentação, bordas e retas) e (iii) **integração** do resultado por serviços *web*.

A literatura de processamento de imagens destaca que o desempenho de detectores (bordas, contornos, retas, objetos) depende tanto do algoritmo quanto da qualidade do dado de entrada (Gonzalez; Woods, 2008; Szeliski, 2010). Exposição, ganho e balanço de branco influenciam contraste, ruído e distribuição de cores, afetando segmentação, bordas e detecção de retas (Gonzalez; Woods, 2008; Canny, 1986; Duda; Hart, 1972). Por isso, decisões sobre configuração da câmera, registro de parâmetros e salvamento de *frames* são tratadas como parte do embasamento do sistema.

2.1 VISÃO COMPUTACIONAL APLICADA À INSPEÇÃO VISUAL

A inspeção visual automatizada aplica técnicas de visão computacional para verificar conformidade geométrica, identificar desvios e apoiar processos de automação (Szeliski, 2010). Em ambientes industriais, são requisitos típicos:

- **repetibilidade**: respostas consistentes em condições semelhantes;
- **rastreabilidade**: registro de parâmetros e resultados para conferência posterior;
- **robustez**: tolerância a variações moderadas de cena;
- **integração**: disponibilização do resultado para sistemas externos.

Neste trabalho, a inspeção é formulada como uma estimativa de alinhamento: é calculado um coeficiente associado ao desvio angular entre retas do objeto e retas de referência no campo de visão. A formulação se apoia em detecção de bordas, detecção de retas e geometria plana. São adotados métodos clássicos e interpretáveis por favorecerem depuração visual e ajuste incremental em cenários aplicados.

2.2 AQUISIÇÃO DE IMAGENS E CONTROLE DE CÂMERA EM LINUX

A detecção de bordas e estruturas lineares depende da qualidade da imagem de entrada. Variações de iluminação, reflexos, ruído do sensor e ajustes inadequados de exposição, ganho ou balanço de branco reduzem contraste e degradam técnicas como Canny e Hough (Gonzalez; Woods, 2008; Canny, 1986; Duda; Hart, 1972). Exposição longa pode causar desfoque por movimento; ganho elevado tende a amplificar ruído, tornando bordas menos nítidas e medidas de orientação menos estáveis (Lin *et al.*, 2024; Zivid, s.d.).

Além disso, modos automáticos (por exemplo, autoexposição e balanço de branco automático) podem variar entre capturas, alterando brilho e cromaticidade (Basler AG, 2026b,c; Teledyne FLIR, 2026). Em inspeção, essa variabilidade dificulta calibração e comparação de resultados. Assim, o sistema trata **a aquisição como parte do problema**, expondo parâmetros de câmera, registrando configurações e permitindo testes reprodutíveis.

2.2.1 Captura com OpenCV e reprodutibilidade do dado de entrada

O OpenCV oferece rotinas para captura e processamento de imagens e é amplamente utilizado em aplicações de visão computacional (OpenCV Team, 2025).

O salvamento de *frames* cria um dado de entrada reprodutível, útil para depuração e avaliação sem a variabilidade da captura ao vivo. Com o *frame* fixo, alterações em parâmetros do algoritmo (limiares, *kernels*, discretizações) podem ser comparadas de forma consistente (Gonzalez; Woods, 2008).

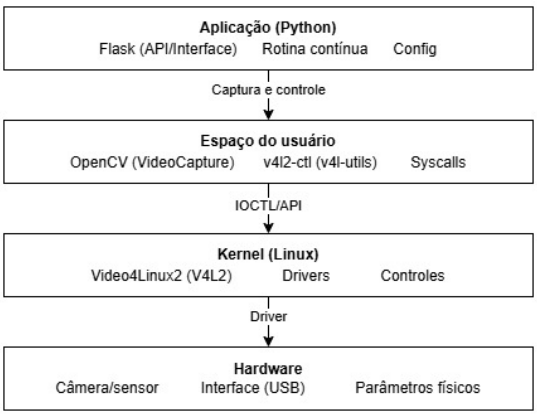
2.2.2 V4L2 e controle de parâmetros no Linux

No Linux, o subsistema Video4Linux2 (V4L2) provê uma API padronizada para dispositivos de vídeo, incluindo *webcams* USB e câmeras compatíveis (The Linux Kernel Developers, 2025). Além de negociar formato, resolução e taxa de quadros, o V4L2 permite acessar controles como exposição, ganho, brilho, contraste, nitidez e balanço de branco.

O V4L2 favorece portabilidade no ecossistema Linux: embora cada câmera ofereça um conjunto próprio de controles, a consulta e o ajuste seguem um padrão comum (The Linux Kernel Developers, 2025). Assim, o sistema trata os controles como um conjunto dinâmico, conforme o dispositivo conectado.

A Figura 1 resume a pilha conceitual de aquisição e controle, relacionando aplicação (Python), ferramentas em espaço de usuário, API V4L2 no *kernel* e *hardware*.

Figura 1 – Pilha conceitual de aquisição e controle de câmera em Linux, destacando o papel do V4L2.



Fonte: Elaborado pelo autor.

Na Figura 1, é mostrado que a aplicação captura *frames* via OpenCV e ajusta parâmetros por utilitários como `v4l2-ctl`. O V4L2 encaminha as chamadas ao *driver*, que aplica o controle no *hardware*.

2.2.2.1 Controles e impacto no *pipeline*

Alguns controles afetam diretamente as etapas posteriores:

- **Exposição e ganho**: influenciam intensidade e relação sinal–ruído. Exposição alta pode causar desfoque e saturação; ganho alto pode amplificar ruído e gerar bordas espúrias (Gonzalez; Woods, 2008).
- **Balanço de branco e saturação**: alteram a distribuição de cores e podem comprometer limiares cromáticos em segmentação (Gonzalez; Woods, 2008).
- **Brilho, contraste e nitidez**: afetam gradientes e, portanto, a resposta de detectores de bordas. Contraste baixo reduz a magnitude do gradiente (Li *et al.*, 2025); nitidez excessiva pode amplificar ruído por realçar componentes de alta frequência (Kheradmand; Milanfar, 2015).

2.2.2.2 Automático vs. calibração reprodutível

Muitas câmeras oferecem controles automáticos (ex.: exposição automática, ganho automático, balanço de branco automático). Em termos práticos, significa que a mesma cena pode ser capturada com aparências diferentes ao longo do tempo, dependendo do algoritmo interno da câmera. Por isso, durante calibração, é comum desabilitar modos automáticos e operar com parâmetros manuais, registrando a configuração aplicada (por exemplo, em JSON). Isso reduz variabilidade entre capturas e torna o processo rastreável e repetível.

2.2.3 Controle de parâmetros com `v4l2-ctl` e rastreabilidade

O utilitário `v4l2-ctl`, do pacote `v4l-utils`, permite consultar e ajustar controles V4L2 pela linha de comando (V4l-utils Project, 2025). Em protótipos, é útil para identificar controles disponíveis e validar ajustes rapidamente, sem implementar chamadas de baixo nível.

No projeto, ajustes são executados por meio de comandos no padrão `v4l2-ctl -c <param>=<valor>` e escrita efetiva dos valores é realizada pelo mecanismo do sistema operacional. A cada alteração, o estado da configuração é registrado, permitindo associar resultados a parâmetros de captura específicos.

2.2.4 Influência da aquisição nos algoritmos

Aquisições com baixo contraste, ruído elevado ou saturação tendem a degradar o mapa de bordas, produzindo contornos ruidosos e fragmentados. Como a Transformada de Hough depende diretamente do conjunto de pixels de borda, esse tipo de degradação pode reduzir votos em picos verdadeiros e, ao mesmo tempo, induzir picos falsos, comprometendo a detecção de retas e, por consequência, as medidas de orientação (Baştan; Bukhari; Breuel, 2017; Guo *et al.*, 2009).

Por outro lado, uma aquisição estável, com referências bem separadas e bordas nítidas, tende a gerar detecções mais consistentes (Gonzalez; Woods, 2008; Szeliski, 2010). Assim, o bloco de aquisição atua como componente de estabilização do *pipeline*, reduzindo falsas detecções associadas a variabilidade da cena.

2.3 REPRESENTAÇÃO DE CORES E SEGMENTAÇÃO

2.3.1 HSV e segmentação por intervalo de cor

Quando a segmentação depende de características de cor, o espaço escolhido influencia diretamente a estabilidade dos limiares. O espaço HSV (*Hue, Saturation, Value*) separa tonalidade (H) e brilho (V), o que costuma facilitar a definição de intervalos menos sensíveis a variações moderadas de iluminação, em comparação com RGB (Gonzalez; Woods, 2008; OpenCV Team, 2025).

Neste trabalho, as retas de referência são segmentadas por intervalos em HSV, produzindo uma máscara binária que restringe o processamento às regiões relevantes do *frame*.

2.3.2 Máscaras binárias e refinamento

Após converter para HSV, define-se um intervalo para H, S e V; pixels dentro do intervalo recebem 1 e os demais 0, gerando uma máscara binária (Gonzalez; Woods, 2008). Essa máscara permite aplicar etapas posteriores apenas na região segmentada, reduzindo custo computacional e a influência de detalhes irrelevantes (Gonzalez; Woods, 2008).

Em cenários ruidosos, operações morfológicas podem refinar a máscara, removendo pontos isolados e preenchendo pequenas falhas (por exemplo, por abertura e fechamento) (Gonzalez; Woods, 2008).

2.4 PRÉ-PROCESSAMENTO E FILTRAGEM

Imagens reais estão sujeitas a ruído introduzido pelo sensor, por condições de iluminação principalmente e compressão quando aplicável. O pré-processamento tem o objetivo de atenuar essas degradações e tornar mais estável a extração de características (Gonza-

lez; Woods, 2008). Para a detecção de bordas e retas, o ruído pode produzir respostas espúrias no mapa de bordas e, consequentemente, aumentar a ocorrência de detecções falsas.

Neste trabalho, emprega-se o filtro de mediana como etapa de suavização. Esse filtro substitui cada pixel pela mediana em uma vizinhança, sendo especialmente eficaz contra ruído impulsivo e, em muitos cenários, preservando bordas melhor do que filtros lineares (Gonzalez; Woods, 2008). Sua aplicação tende a reduzir componentes isolados que confundiriam a detecção de retas, porém o tamanho da vizinhança deve ser ajustado para evitar perda de detalhes finos (Gonzalez; Woods, 2008).

2.5 DETECÇÃO DE BORDAS

Bordas correspondem a variações acentuadas de intensidade ou cor e, em geral, estão associadas aos limites de objetos e regiões (Gonzalez; Woods, 2008; Szeliski, 2010). Métodos baseados em gradientes respondem com maior intensidade onde ocorrem mudanças abruptas. Por isso, um mapa de bordas consistente facilita a detecção de formas ao restringir a análise a um conjunto de pixels candidatos.

O detector de Canny combina suavização, cálculo de gradiente, supressão de não-máximos e histerese para produzir bordas finas e relativamente estáveis (Canny, 1986). Em inspeção, isso beneficia etapas posteriores como a Transformada de Hough.

Em termos conceituais, o método envolve:

1. **Suavização**: reduz ruído antes do cálculo do gradiente.
2. **Gradiente**: estima magnitude e direção de variação, gerando bordas candidatas.
3. **Supressão de não-máximos**: afina as bordas, preservando apenas máximos locais do gradiente.
4. **Histerese**: utiliza dois limiares (alto e baixo) para manter bordas fortes e conectar bordas fracas coerentes.

Na prática, os limiares e o tamanho do filtro devem ser ajustados ao cenário (OpenCV Team, 2025). A visualização conjunta de bordas e retas juntamente ao valor dos parâmetros da câmera e do detector auxiliam no processo de calibração.

2.6 DETECÇÃO DE RETAS COM TRANSFORMADA DE HOUGH

A Transformada de Hough é amplamente utilizada para detectar retas a partir de mapas de bordas. O método mapeia pontos da imagem para um espaço de parâmetros (tipicamente (ρ, θ)), no qual picos no acumulador indicam hipóteses de retas no domínio

da imagem. O método é robusto a ruído moderado e pequenas descontinuidades, pois pontos aproximadamente colineares contribuem para o mesmo pico (Duda; Hart, 1972).

Na prática, a qualidade das retas depende do mapa de bordas e de parâmetros como discretização e limiar de votação. Em cenas com múltiplas bordas, podem surgir várias retas para a mesma estrutura, exigindo seleção e, quando necessário, agrupamento (Szeliski, 2010). Restrições geométricas (por exemplo, faixa de orientação esperada) ajudam a reduzir falsos positivos. Neste trabalho, o objetivo é recuperar hipóteses de retas com orientações estimáveis para comparação com referências.

2.7 GEOMETRIA DO ALINHAMENTO E CÁLCULO DO COEFICIENTE

Do ponto de vista geométrico, uma maneira prática de tratar alinhamento é comparar orientações dominantes observadas no objeto com uma direção de referência. Em cenários industriais, essa referência pode estar associada ao próprio dispositivo, a um gabarito, a marcas no ambiente ou a elementos fixos no campo de visão.

2.7.1 Desvio angular

Com os dados da detecção de retas de referência e retas do objeto, o desalinhamento pode ser expresso pelo desvio angular entre orientações (Szeliski, 2010). No projeto, utiliza-se a **maior diferença absoluta** entre ângulos de referência e ângulos do objeto, o que enfatiza o pior caso. Essa escolha exige mecanismos de robustez para lidar com detecções espúrias, motivando filtragem, agrupamento e repetição.

2.8 ROBUSTEZ, REPETIÇÃO E REPRODUTIBILIDADE

Em operação, sistemas de visão computacional estão sujeitos a variabilidade (iluminação, vibração, ruído e flutuações de foco), o que pode causar falhas pontuais. Para reduzir essa sensibilidade, uma estratégia prática é repetir o processamento e combinar apenas os resultados válidos (por exemplo, por média), descartando execuções que retornem erro (Gonzalez; Woods, 2008). Essa abordagem tende a aumentar a robustez percebida do serviço, ao custo de maior tempo de processamento por requisição; por isso, o número de repetições deve ser definido conforme o compromisso entre latência e confiabilidade exigido no cenário de aplicação.

Para favorecer a reprodutibilidade e a rastreabilidade, é recomendável salvar o último *frame* capturado registrando o dado de entrada associado a cada resultado e permitindo repetir o processamento em condições controladas (Gonzalez; Woods, 2008). Quando o ajuste de parâmetros da câmera faz parte do processo (ex.: exposição, ganho e balanço de branco), o salvamento de *frames* em conjunto com a persistência das configurações

de aquisição cria um histórico comparável, viabilizando avaliar o impacto direto dessas escolhas no mapa de bordas, na detecção de retas e, por fim, no coeficiente de alinhamento.

2.9 SERVIÇOS *WEB* E INTEGRAÇÃO POR HTTP

2.9.1 Flask e API

Flask é um *microframework* em Python utilizado para expor serviços *web* e APIs (Pallets Projects, 2025). *Endpoints* HTTP com retorno em JSON facilitam integração entre componentes heterogêneos e registro de resultados (Pallets Projects, 2025).

No projeto, o servidor fornece rotas tanto para operação humana (interface de configuração) quanto para acionamento sob demanda do processamento. Embora essas rotas coexistam no mesmo servidor por conveniência, distingui-las conceitualmente ajuda a organizar o texto.

2.10 SÍNTESE DO EMBASAMENTO

Os conceitos discutidos fundamentam as escolhas do sistema: controle reprodutível da aquisição via V4L2 (The Linux Kernel Developers, 2025; V4l-utils Project, 2025), uso de OpenCV para captura e processamento (OpenCV Team, 2025), detecção de bordas (Canny) (Canny, 1986), detecção de retas por Hough (Duda; Hart, 1972) e integração via HTTP/JSON (Pallets Projects, 2025). Esses fundamentos orientam as decisões de implementação e a análise apresentada nos capítulos seguintes.

3 METODOLOGIA

A metodologia adotada neste trabalho é de natureza aplicada, com foco na construção de um protótipo funcional capaz de operar com uma câmera conectada ao Linux e executar rotinas de inspeção geométrica baseadas em visão computacional. O desenvolvimento privilegiou modularidade e separação de responsabilidades, organizando o sistema em núcleos que pudessem evoluir de forma relativamente independente.

Do ponto de vista prático, o sistema foi implementado em Python e estruturado em torno de três pilares: (i) uma camada de aplicação e interface (servidor Flask e página *web*) para operação e integração; (ii) uma camada de aquisição e controle de câmera (OpenCV e V4L2) para captura e ajuste de parâmetros; e (iii) um núcleo de processamento de imagem para detecção de retas e cálculo do desvio de alinhamento.

Para tornar a descrição mais clara para o leitor, este capítulo organiza o desenvolvimento em dois níveis complementares:

- **Núcleos** (visão de implementação): componentes do código e suas responsabilidades.
- **Blocos conceituais** (visão de finalidade): agrupamentos de núcleos que, juntos, atendem a um objetivo maior.

Essa dupla visão é importante porque, por conveniência, o mesmo servidor Flask expõe tanto rotas de operação (ajuste de parâmetros de câmera) quanto uma rota de execução sob demanda do algoritmo. Conceitualmente, porém, essas funções pertencem a blocos distintos.

3.1 VISÃO GERAL DA ARQUITETURA

A arquitetura do sistema foi organizada em seis núcleos principais (implementação): (i) interface de configuração (frontend), (ii) aplicação e serviços (servidor Flask), (iii) interface com a câmera (V4L2/OpenCV), (iv) configuração (Config), (v) algoritmos de processamento de imagem. Em uma leitura de alto nível, esses núcleos se agrupam em três blocos conceituais (finalidade), que serão usados como fio condutor da metodologia:

1. **Bloco de aquisição e ajuste da imagem**: reúne frontend, rotas de configuração do Flask, camada de câmera (V4L2/OpenCV) e módulo Config. O objetivo é ajustar a aquisição de forma controlável, observável e reprodutível, favorecendo o desempenho de algoritmos de visão.
2. **Bloco de processamento de imagem**: reúne o núcleo de algoritmos. Embora a estrutura seja expansível, o algoritmo implementado atende a uma situação es-

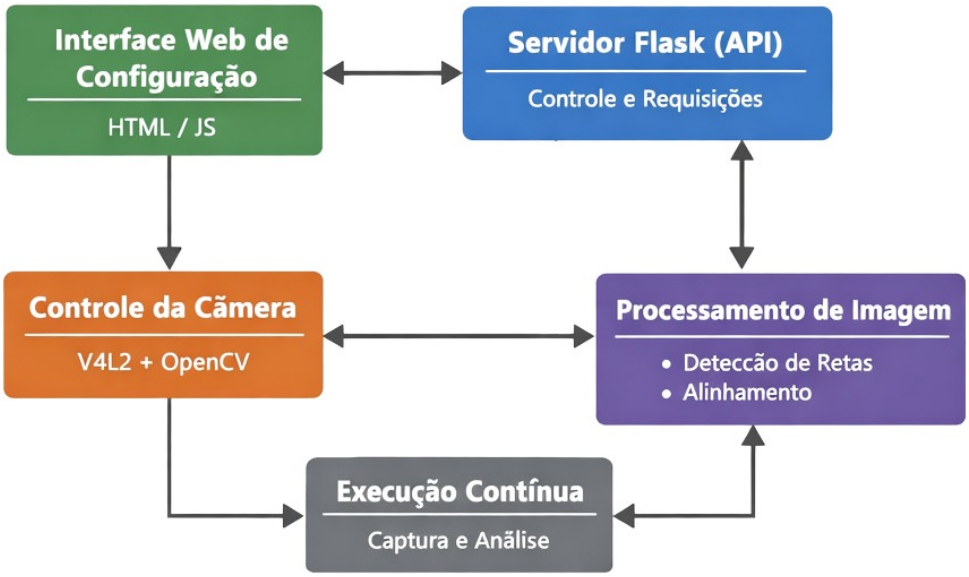
pecífica: estimar alinhamento comparando retas do objeto com retas de referência presentes no campo de visão.

3. **Bloco de exposição e integração**: reúne o *endpoint* de execução sob demanda, permitindo publicar resultados (coeficiente/estado) para consumo por sistemas externos.

3.1.1 Arquitetura em núcleos (visão funcional)

A Figura 2 apresenta a arquitetura em núcleos do sistema, evidenciando as responsabilidades principais e o fluxo de comunicação entre a interface *web*, o servidor de aplicação, a camada de câmera, o núcleo de processamento e os mecanismos de integração externa.

Figura 2 – Arquitetura em núcleos funcionais do sistema de inspeção baseado em visão computacional.



Fonte: Elaborado pelo autor.

Na Figura 2 é possível observar que os núcleos ligados à aquisição (frontend, rotas de configuração, Config e interface com a câmera) formam a base para calibração e reprodutibilidade. Já o núcleo de algoritmos concentra a lógica de inspeção (detecção de retas e alinhamento) e é acionado sob demanda pelo *endpoint* de coeficiente.

3.1.2 Papéis dos processos: servidor e rotina contínua

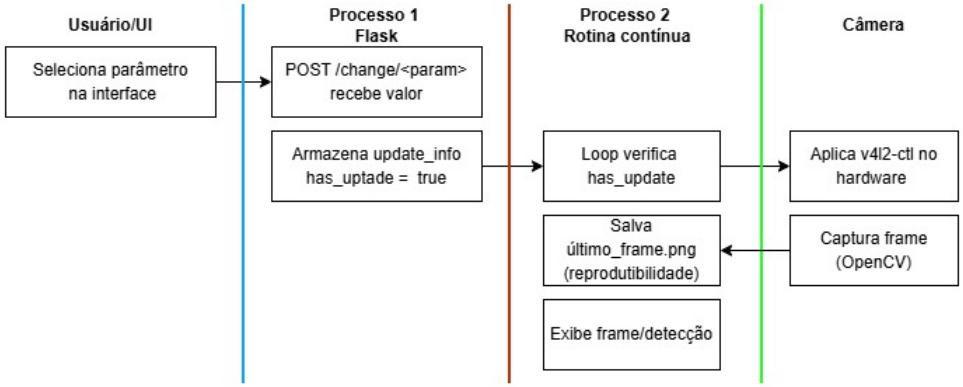
Além dos blocos, a execução do sistema foi concebida com dois processos (no sentido de duas rotinas concorrentes no tempo) que se complementam:

- **Processo 1 (servidor Flask)**: expõe a interface *web* e *endpoints* HTTP. Ele registra solicitações de mudança de parâmetros e solicitações de cálculo sob demanda.

- **Processo 2 (rotina contínua de visualização)**: executa um laço voltado à observação e depuração. Esse laço aplica atualizações pendentes de parâmetros no *hardware*, captura *frames*, salva o último *frame* e (em modo de depuração) exibe a imagem com retas desenhadas.

A Figura 3 sintetiza esse fluxo. O ponto central é que a rota de alteração (`/change`) não bloqueia a requisição aplicando o parâmetro imediatamente; em vez disso, registra a intenção de mudança e sinaliza ao laço contínuo que existe uma atualização pendente. Isso reduz travamentos na interface e centraliza o acesso ao dispositivo de vídeo no laço de visualização.

Figura 3 – Sequência simplificada do ajuste de parâmetros e suporte à visualização contínua (Processos 1 e 2).



Fonte: Elaborado pelo autor.

3.2 BLOCO DE AQUISIÇÃO E AJUSTE DA IMAGEM

O Bloco 1 foi projetado para tornar o ajuste de aquisição controlável e reprodutível, além de oferecer meios práticos para o operador observar, em tempo de execução, o efeito das mudanças de parâmetros na imagem capturada e na detecção de estruturas (bordas e retas). Em termos de núcleos, este bloco inclui o frontend, o servidor Flask (no que diz respeito às rotas de configuração), a camada de câmera (V4L2/OpenCV) e o módulo Config (estado e persistência).

3.2.1 Interface de configuração (frontend)

A interface de configuração foi implementada como uma página HTML contendo uma tabela de propriedades ajustáveis para a câmera conectada. Seu objetivo é atuar como um **facilitador de operação**: em vez de exigir que o operador memorize comandos do sistema ou ajuste parâmetros via terminal, a interface apresenta controles expostos pelo *driver* e permite selecionar valores de forma guiada.

Uma característica relevante é que o frontend não "codifica" uma câmera específica. A lista de propriedades exibida e os valores atuais são obtidos a partir do estado mantido

pela aplicação (módulo `Config`), que por sua vez reflete as capacidades do dispositivo em uso. Assim, ao trocar a câmera, a interface tende a continuar funcional.

Em operação, o frontend envia requisições HTTP para o servidor, contendo o par *parâmetro–valor* desejado. A alteração efetiva no *hardware* não é realizada no navegador: ela é executada pelo *backend* por meio de comandos de sistema (via `v4l2-ctl`). Essa escolha mantém o navegador como uma camada de interação e preserva a responsabilidade de controle do dispositivo no ambiente do servidor.

3.2.2 Servidor de aplicação (Flask) para operação e serviços

O servidor Flask centraliza o controle do sistema e expõe rotas HTTP que atendem tanto à operação humana (via frontend) quanto à integração com outros componentes. As rotas principais relacionadas ao Bloco 1 são:

- `/`: renderiza a página de configuração com a lista de parâmetros e seus valores atuais;
- `/change/<param>`: recebe um `POST` com o parâmetro e o novo valor; registra a solicitação e sinaliza uma atualização pendente para a câmera;
- `/params`: retorna um JSON com o estado atual dos parâmetros mantidos pela aplicação.

Dois aspectos de projeto se destacam nessa camada:

3.2.2.1 Atualizações assíncronas de parâmetros (flags e estrutura de update)

A rota `/change` não altera imediatamente o *hardware*. Em vez disso, o servidor preenche uma estrutura (por exemplo, `update_info`) e ativa uma flag (por exemplo, `has_update`). Essa decisão evita bloquear a requisição HTTP (o que prejudicaria a experiência no frontend) e delega a aplicação do parâmetro para a rotina contínua de visualização, que é quem centraliza o acesso ao dispositivo de vídeo.

Na prática, esse mecanismo também reduz o risco de condições de corrida: a câmera é um recurso compartilhado e, se múltiplas requisições tentarem reconfigurar o *hardware* durante uma captura, pode haver comportamento imprevisível. Ao concentrar a aplicação de parâmetros no laço contínuo, o sistema impõe uma ordem de aplicação e simplifica depuração.

3.2.2.2 Separação entre operação e processamento sob demanda

Embora o servidor exponha uma rota de cálculo sob demanda (`/coeficiente/<tipo>`), ela foi mantida no mesmo processo por conveniência (não criar um segundo servidor apenas para essa finalidade). Conceitualmente, porém, a rota de coeficiente pertence ao Bloco

2 (processamento) e ao Bloco 3 (exposição), enquanto as rotas `/`, `/change` e `/params` pertencem ao Bloco 1 (operação e ajuste de aquisição). Esta distinção é mantida ao longo do texto para situar corretamente o papel de cada componente.

3.2.3 Configuração da câmera e interface V4L2/OpenCV

A interface com a câmera foi encapsulada em uma classe (por exemplo, `Camera`), composta por três elementos principais:

- `Config`: armazena e organiza os parâmetros conhecidos pela aplicação;
- `Capture`: compõe (ou encapsula) a classe `cv2.VideoCapture` para inicializar e realizar captura de *frames* (incluindo, quando aplicável, resolução, FPS e formato);
- `Algoritmo`: executa rotinas de processamento e devolve resultados numéricos (coeficiente/ângulo) ou códigos de erro.

A alteração de parâmetros de controle de câmera foi implementada por comando de sistema com `v4l2-ctl`, seguindo o padrão:

```
sudo v4l2-ctl -d <dispositivo> -c <param>=<valor>
```

Após uma alteração bem-sucedida, o novo estado é atualizado na instância `Config`. Adicionalmente, o sistema pode persistir o estado de configuração em arquivo (por exemplo, `params/ultima_config_camera.json`). Essa persistência tem papel central para rastreabilidade e reprodutibilidade: permite retomar configurações utilizadas anteriormente e documentar quais parâmetros estavam ativos durante um teste específico.

3.2.4 Núcleo de configuração (Config): estado dinâmico e organização de parâmetros

O módulo `Config` foi desenvolvido para representar, em memória, um conjunto de parâmetros de câmera como atributos dinâmicos. Em vez de fixar controles no código, o construtor recebe um conjunto (por exemplo, uma lista de dicionários) e define cada par chave–valor como atributo do objeto. Essa abordagem reforça a estratégia de generalidade: ao trocar a câmera, o sistema continua operando com o novo conjunto de controles, reduzindo alterações estruturais.

Além disso, foi adotado um critério objetivo de organização para melhorar a usabilidade: parâmetros relacionados a modos automáticos (chaves contendo termos como `auto` ou `automatic`) são priorizados no início da lista. A motivação é prática pois certos controles automáticos podem sobrescrever valores manuais, e destacá-los reduz ambiguidades durante calibração (por exemplo, ajustar ganho manual enquanto um modo automático continua ativo).

3.2.5 Rotina de visualização contínua: observação, depuração e reprodutibilidade

O Processo 2 (rotina contínua) existe para permitir a visualização do que a câmera captura e como os parâmetros de aquisição influenciam a imagem e a detectabilidade de retas. Esse processo executa um laço que:

1. Verifica se existe atualização pendente de parâmetros (flag `has_update`) e, se existir, aplica a alteração no *hardware*.
2. Captura um *frame* da câmera e o salva como `ultimo_frame.png`, permitindo inspeção offline e reprodutibilidade de testes.
3. Executa uma rotina de detecção (modo de depuração/visualização), exibindo a imagem com retas desenhadas.

Dessa forma, dois aspectos são observados. O primeiro é operacional: a calibração de parâmetros pode ser realizada rapidamente, com observação das alterações no *frame* e validação de melhorias na segmentação e na detecção. O segundo é metodológico: o salvamento de *frames* cria um registro reutilizável, que pode ser reprocessado para depuração e comparação, sem depender de uma outra captura ao vivo idêntica.

3.3 BLOCO DE PROCESSAMENTO DE IMAGEM

O Bloco 2 corresponde ao núcleo de algoritmos de processamento de imagem. Ele foi estruturado para permitir extensões futuras (múltiplos algoritmos), mas a implementação descrita atende a uma situação específica: detectar retas de referência (por segmentação por cor) e retas do objeto, e então estimar um desvio angular máximo como medida de desalinhamento.

3.3.1 Estrutura do núcleo de algoritmos: classe `Algoritmo` e seleção por tipo

O núcleo de processamento foi modelado por uma classe `Algoritmo`, que oferece um método do tipo `run(tipo, frame)` para selecionar e executar diferentes rotinas. Essa decisão organiza o código e permite evoluir o sistema com novas estratégias sem alterar a interface HTTP: basta associar um novo `tipo` a uma rotina.

Para facilitar a integração, as rotinas se comportam da seguinte maneira: retornam um resultado numérico (quando bem-sucedidas) ou códigos de erro (quando etapas essenciais falham). Esse contrato é útil para que o *endpoint* sob demanda agregue execuções, descarte falhas e reporte ao cliente externo não apenas o valor final, mas também informações de confiabilidade (por exemplo, quantas execuções foram descartadas).

3.3.2 Algoritmo principal: detecção de retas e desvio angular

O algoritmo principal (por exemplo, `algoritmo0`) tem como objetivo estimar o desvio de alinhamento comparando retas do objeto com retas de referência presentes no campo de visão. O fluxo lógico pode ser resumido nas seguintes etapas:

1. Converter o *frame* para o espaço HSV.
2. Segmentar as linhas de referência por intervalo de cor (máscara binária).
3. Filtrar utilizando a mediana
4. Extrair bordas na região segmentada utilizando o método de Canny.
5. Aplicar Transformada de Hough para detectar retas associadas às referências.
6. Converter a representação de retas para pontos na imagem e ajustar para o tamanho do *frame*.
7. Agrupar retas pertencentes à mesma referência e selecionar uma reta representativa (a mais acima no eixo vertical), construindo uma máscara/limite para delimitar a região do objeto.
8. Detectar retas do objeto na região segmentada/delimitada.
9. Calcular ângulos de todas as retas e obter a maior diferença absoluta entre ângulos de referência e ângulos do objeto, produzindo o valor final de desvio.

Do ponto de vista de validação, o modo de depuração (Processo 2) é fundamental: ele permite observar quais retas o algoritmo está detectando e ajustar limiares (segmentação, bordas, Hough) de forma direta, até alcançar comportamento estável para a cena considerada.

3.3.3 Execução sob demanda e agregação de resultados

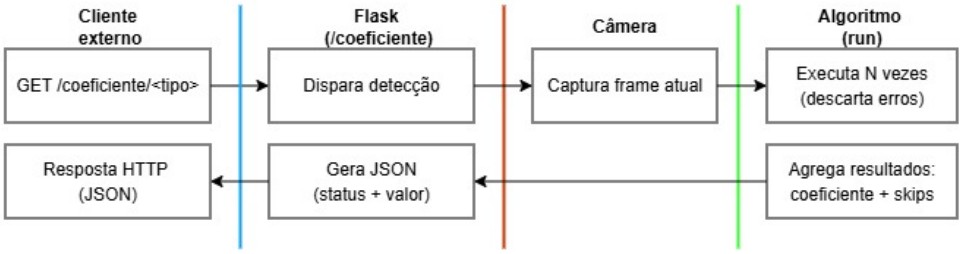
O *endpoint* `/coeficiente/<tipo>` aciona o processamento somente quando solicitado. Ao receber a requisição, o sistema captura um *frame* e executa o algoritmo correspondente. Para aumentar robustez contra variações pontuais e falhas intermitentes, o *endpoint* pode executar o algoritmo múltiplas vezes (um número fixo de repetições) e calcular a média apenas das execuções válidas. Execuções que retornam códigos de erro são contabilizadas como *skips*.

Essa estratégia tem dois efeitos práticos. Primeiro, reduz o impacto de *frames* "ruins" (por exemplo, com ruído momentâneo, reflexo, ou falha na detecção das referências). Segundo, fornece ao cliente externo uma medida indireta de confiabilidade: muitos

skips sugerem que as condições de aquisição ou os parâmetros do algoritmo precisam ser revisados.

A Figura 4 resume a sequência do *endpoint*, destacando captura, repetição, agregação e resposta em JSON.

Figura 4 – Sequência simplificada do *endpoint* `/coeficiente/<tipo>`: captura, repetição, agregação e resposta em JSON.



Fonte: Elaborado pelo autor.

3.4 BLOCO DE EXPOSIÇÃO E INTEGRAÇÃO

O Bloco 3 trata de como o resultado do processamento é disponibilizado para outros sistemas. Em protótipos industriais, esse bloco é essencial: um coeficiente calculado localmente precisa ser exposto de forma padronizada para que possa disparar ações, registrar decisões e integrar-se ao fluxo produtivo.

3.4.1 Disponibilização via HTTP/JSON

O retorno do *endpoint* `/coeficiente/<tipo>` é estruturado em JSON para facilitar integração com clientes heterogêneos. Na prática, essa resposta inclui o valor calculado (coeficiente/desvio) e informações de execução (por exemplo, *skips* e códigos de status). Essa modelagem favorece a criação de consumidores simples (scripts, dashboards, supervisórios) e permite testes automatizados, pois o resultado é um objeto estruturado e fácil de registrar.

3.5 FLUXO DE EXECUÇÃO DO SISTEMA

A execução foi organizada por uma classe orquestradora (por exemplo, `Modulo`). Ao iniciar, o módulo cria uma *thread* para executar o servidor Flask em modo *daemon*. Em paralelo, inicia a rotina contínua descrita anteriormente (Processo 2), que aplica atualizações pendentes, captura *frames*, salva o último *frame* e exibe a depuração.

O laço principal é encerrado por interação do usuário (por exemplo, tecla `q`), finalizando a aplicação. Essa forma de encerramento é apropriada no contexto de protótipo, pois prioriza operação manual durante calibração e testes. Em um cenário de produção, esse mecanismo poderia ser substituído por supervisão de processo, *watchdogs* e inicia-

lização por serviço do sistema, sem alterar a lógica do bloco de aquisição e do bloco de processamento.

3.6 REPRODUTIBILIDADE, RASTREABILIDADE E EXTENSIBILIDADE

O projeto foi concebido para permitir adaptações com o mínimo de alterações estruturais. Algumas decisões intencionais reforçam reprodutibilidade e evolução incremental:

- **Persistência de configurações**: armazenar parâmetros aplicados (por exemplo, em JSON) reduz tempo de calibração e documenta quais ajustes estavam ativos durante uma execução.
- **Salvamento do último *frame***: manter `ultimo_frame.png` permite inspeção offline e reprocessamento do mesmo dado, melhorando comparações entre parametrizações e versões do algoritmo.
- **Generalidade em relação à câmera**: tratar controles como um conjunto dinâmico tende a exigir apenas ajustes de caminhos do dispositivo e revisão de intervalos/limiares, sem reescrever a arquitetura.
- **Extensibilidade de algoritmos**: a seleção por `tipo` permite incluir novas rotinas de detecção e comparação sem alterar a interface HTTP.
- **Separação conceitual por blocos**: mesmo quando componentes convivem no mesmo processo por conveniência (por exemplo, rotas de configuração e de coeficiente), a documentação mantém a separação por finalidade, facilitando manutenção e leitura.

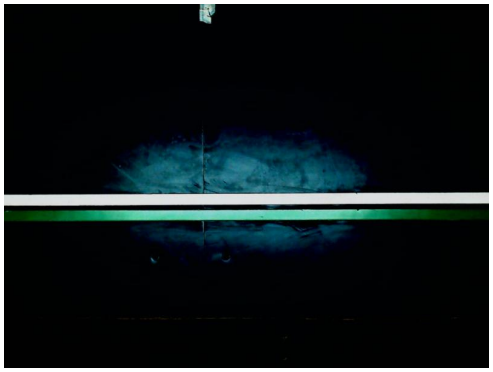
Essas escolhas favorecem evolução incremental, desejável em cenários industriais nos quais condições de cena, *hardware* e requisitos podem mudar ao longo do tempo.

3.7 EXPERIMENTO PARA VALIDAÇÃO

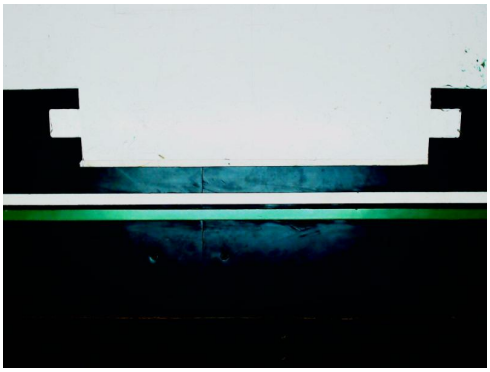
Para demonstrar a execução do software desenvolvido e coletar métricas de desempenho para comparação de resultados, foi realizado um experimento inspirado no exemplo prático vivenciado durante o projeto no Polo de Inovação do IFPB (ver Seção 1.2). Na Figura 5, são mostrados dois *frames* capturados por uma câmera posicionada acima do plano de trabalho. Em ambos, as duas linhas de referência estão fixadas no campo de visão e, no segundo caso (Figura 5b), há um objeto posicionado para inspeção de alinhamento.

Figura 5 – *Frames* capturados na situação de inspeção de alinhamento.

(a) *Frame* apenas com linhas de referência.



(b) *Frame* com linhas de referência e objeto.



Fonte: Elaborado pelo autor.

A cena da Figura 5b serviu como base para a montagem de um cenário experimental controlado.

3.7.1 Condução do experimento

Foram avaliadas diferentes condições de aquisição: utilizaram-se duas câmeras e testaram-se seis ângulos de desalinhamento (0º, 3º, 5º, 15º, 30º e 45º). A escolha desses valores busca contemplar, inicialmente, ângulos próximos do alinhamento esperado (definido pelas linhas de referência) e, em seguida, condições com desalinhamento mais pronunciado. Para cada câmera, ângulo e *frame* capturado, todas as etapas do algoritmo de detecção de retas foram registradas. Adicionalmente, foram coletadas métricas de desempenho para avaliar o tempo de processamento em cada cenário.

3.7.2 Câmeras utilizadas no experimento

Foram utilizadas duas câmeras comerciais, ambas conectadas a um computador com Linux (Ubuntu 22.04) via USB:

- **Câmera 1**: Logitech C920 HD Pro Webcam
- **Câmera 2**: XWF HD1080P Webcam

Na Figura 6, são mostradas as duas câmeras utilizadas.

Figura 6 – Câmeras utilizadas no experimento.

(a) Câmera 1: Logitech C920 HD Pro Webcam.

Fonte: Amazon.

(b) Câmera 2: XWF HD1080P Webcam.

Fonte: Amazon.

A Logitech C920 HD Pro (Figura 6a) é uma *webcam* voltada à videoconferência e à captura em Full HD, oferecendo imagem em 1080p a 30 fps, lente de vidro, correção automática de iluminação e campo de visão diagonal de 78 graus. Além disso, o modelo conta com microfones duplos para captação de áudio estéreo, sendo uma opção consolidada para aplicações de inspeção e monitoramento em que estabilidade e reprodutibilidade na aquisição são desejáveis (Logitech, 2026; Support, 2026).

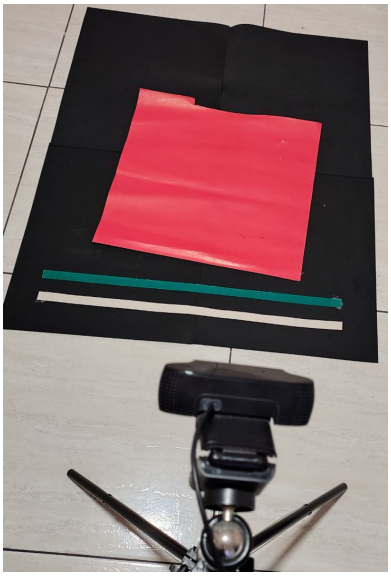
A XWF HD1080P, frequentemente identificada como *XWF-1080P* (Figura 6b), é uma *webcam* de baixo custo voltada ao uso cotidiano (reuniões e videochamadas). Na ausência de uma ficha técnica oficial facilmente verificável do fabricante, resultados de testes práticos reportam operação em 1920×1080 (Full HD), com taxa de quadros observada na faixa de 20 fps em determinados cenários, indicando que seu desempenho pode variar conforme iluminação, *driver* e *pipeline* de captura utilizado (WebcamTests.com, 2025, 2024).

3.7.3 Montagem do cenário

O cenário foi montado utilizando materiais de papelaria: folhas de EVA preto (para o fundo), tiras de cartolina (verde e branca) para as linhas de referência e um pedaço de cartolina vermelha (para o objeto a ser inspecionado). O plano de trabalho foi organizado no chão, e a câmera foi posicionada com o auxílio de um tripé. Na Figura 7, é possível observar o cenário montado para cada câmera.

Figura 7 – Cenário experimental montado para cada câmera.

(a) Cenário para Câmera 1: Logitech C920 HD Pro Webcam.



(b) Cenário para Câmera 2: XWF HD1080P Webcam.

Fonte: Elaborado pelo autor.

3.7.4 Métricas de desempenho

As métricas de desempenho foram obtidas por instrumentação direta do *pipeline*. Em cada *frame*, o tempo de execução de cada etapa foi medido com um temporizador de alta precisão (`time.perf_counter_ns()`) e registrado em milissegundos.

Para cada condição experimental (combinação de câmera e ângulo nominal de desalinhamento), o algoritmo foi executado por $N = 300$ *frames* consecutivos. Antes da coleta, aplicou-se um período de *warm-up* de 10 *frames* descartados, com o objetivo de reduzir efeitos de inicialização (por exemplo, aquecimento de *cache* e alocações internas).

Durante o *benchmark*, recursos que poderiam introduzir sobrecarga e variabilidade — como exibição em tela (*imshow*), salvamento de imagens e mensagens de depuração (*prints*) — foram desativados, mantendo-se apenas a captura e o processamento. Os tempos coletados foram sumarizados por estatísticas descritivas (média e desvio padrão) e por percentis $p50$ (mediana) e $p95$, utilizados para caracterizar, respectivamente, o comportamento típico e a latência de cauda.

Adicionalmente, registrou-se a taxa de sucesso por condição, definida como a proporção de *frames* em que o algoritmo completou todo o fluxo de detecção (identificando pelo menos uma reta de referência e ao menos uma reta do objeto) sem retornar códigos de erro.

4 RESULTADOS

Conforme apresentado no capítulo anterior, foram conduzidos experimentos com duas câmeras: Logitech C920 e XWF HD1080P. Inicialmente, os parâmetros de configuração de cada câmera foram ajustados utilizando o Processo 2 (ver Subseção 3.1.2), com *feedback* contínuo, a fim de facilitar a detecção de retas pelo algoritmo. Em seguida, foram capturadas imagens em diferentes posicionamentos angulares do objeto em relação às linhas de referência (0º, 3º, 5º, 15º, 30º e 45º). Adicionalmente, métricas de *benchmarking* foram coletadas para avaliar o desempenho em termos de tempo de processamento.

Embora tenham sido realizadas medições em seis angulações, neste capítulo são apresentados apenas os resultados para os ângulos de 0º e 30º. Essa escolha se justifica por contemplar, respectivamente, um cenário próximo do alinhamento esperado e um cenário com desalinhamento pronunciado. Os demais resultados para ângulos nominais de 3°, 5°, 15° e 45° (para ambas as câmeras) são apresentados no Apêndice A.

4.1 DETECÇÃO DE RETAS COM A CÂMERA 1 (LOGITECH C920)

Nesta seção são apresentados os resultados obtidos com a câmera Logitech C920. Primeiramente, são descritas as configurações adotadas e, na sequência, os resultados do processamento de imagem para os ângulos de 0º e 30º.

4.1.1 Configurações da câmera

O ajuste dos parâmetros da câmera Logitech C920 foi realizado utilizando *feedback* contínuo por depuração (Processo 2 — ver Subseção 3.1.2). Os valores foram modificados na interface *web*, enquanto o efeito de cada alteração era acompanhado em uma janela de visualização do *frame* capturado. Após os ajustes, e considerando que as cores dos elementos apresentaram boa separabilidade na segmentação, obteve-se o conjunto de parâmetros apresentado na Tabela 1.

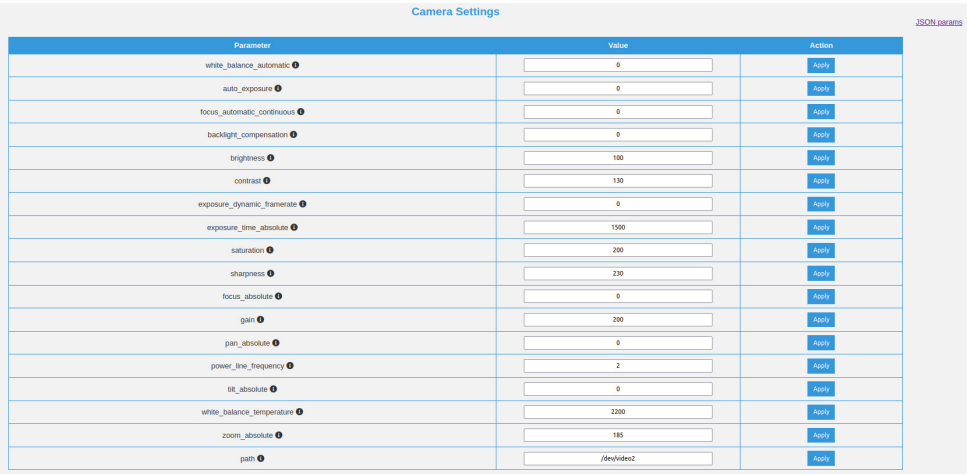
Tabela 1 – Parâmetros de configuração da Câmera Logitech C920.

Parâmetro	**Valor**
`auto_exposure`	0
`backlight_compensation`	0
`brightness`	100
`contrast`	130
`exposure_dynamic_framerate`	0
`exposure_time_absolute`	1500
`focus_absolute`	0
`focus_automatic_continuous`	0
`gain`	200
`pan_absolute`	0
`power_line_frequency`	2
`saturation`	200
`sharpness`	230
`tilt_absolute`	0
`white_balance_automatic`	0
`white_balance_temperature`	2200
`zoom_absolute`	185

Fonte: Elaborado pelo autor.

Como complemento, a Figura 8 apresenta uma captura de tela do *frontend* com esses mesmos parâmetros configurados. Os valores foram escolhidos dentro de intervalos sugeridos por *tooltips*. O botão azul `Apply` na última coluna da tabela é responsável por enviar o comando de alteração por meio do *endpoint* `/change/<param>` (ver Subseção 3.2.2).

Figura 8 – Configurações da câmera Logitech C920 ajustadas no *frontend web*.



Fonte: Elaborado pelo autor.

Na Figura 8, verifica-se que os parâmetros apresentados na Tabela 1 estão refletidos na interface *web*. Observa-se ainda que parâmetros relacionados a modos automáticos são priorizados na listagem (ver Subseção 3.2.4), por exemplo: `auto_exposure`, `white_balance_automatic` e `focus_automatic_continuous`.

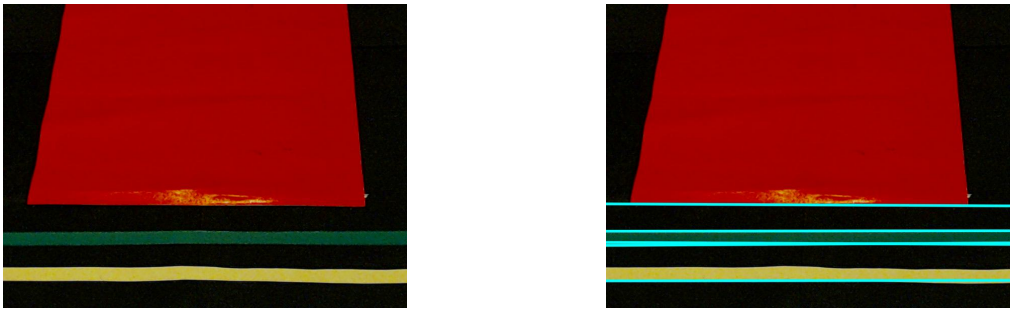
Com os parâmetros ajustados, o algoritmo pôde ser executado de forma estável, conforme apresentado nas subseções a seguir.

4.1.2 Detecção com ângulo nominal de 0º

Neste cenário, simulou-se um objeto posicionado praticamente alinhado à orientação das retas de referência (0º). A Figura 9 apresenta o *frame* de entrada e o resultado final, com as retas detectadas destacadas em ciano.

Figura 9 – Entrada e resultado final (condição nominal: 0°).

(a) Frame original **(b)** Resultado final



Fonte: Elaborado pelo autor.

Pela Figura 9, nota-se que o algoritmo detectou corretamente as retas do objeto e identificou mais de uma reta de referência para cada linha. Para obter esse resultado, o processamento seguiu as etapas descritas na Subseção 3.3.2. A Figura 10 apresenta os principais passos do processamento para a linha de referência verde.

Figura 10 – Processamento da linha de referência verde (condição nominal: 0°).

(a) Máscara (verde) **(b)** Mediana (verde)

(c) Canny (verde) **(d)** Hough e retas detectadas (verde)

Fonte: Elaborado pelo autor.

Na Figura 10, a binarização reduz ruídos e preserva os elementos de interesse, favorecendo a atuação dos filtros subsequentes (mediana e Canny). Em seguida, o processamento é repetido para a linha de referência branca, conforme a Figura 11.

Figura 11 – Processamento da linha de referência branca (condição nominal: 0°).

(a) Máscara (branca) **(b)** Mediana (branca)

(c) Canny (branca) **(d)** Hough e retas detectadas (branca)

Fonte: Elaborado pelo autor.

A Figura 11 evidencia que o mesmo encadeamento de etapas produz resultados similares para a segunda referência. Com as retas detectadas em ambas as linhas, o algoritmo prossegue para a etapa de preparação da imagem destinada à detecção no objeto, mostrada na Figura 12.

Figura 12 – Preparação para detecção do objeto (condição nominal: 0°).

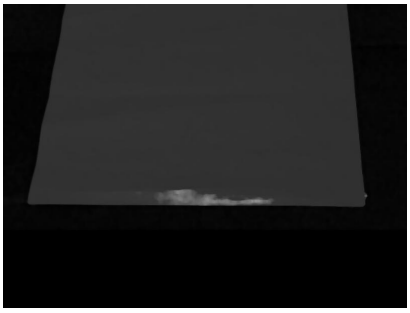
(a) Preparação para processamento do objeto **(b)** Cinza (objeto)

Fonte: Elaborado pelo autor.

Na Figura 12, observa-se que o pré-processamento gera uma imagem em escala de cinza (monocanal), o que favorece a aplicação do limiaramento de Otsu na etapa de detecção de retas do objeto, conforme a Figura 13.

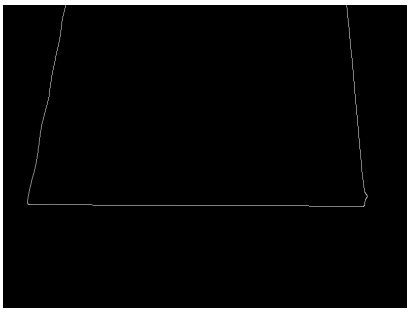
Figura 13 – Processamento das retas do objeto (condição nominal: 0°).

(a) Mediana (objeto)

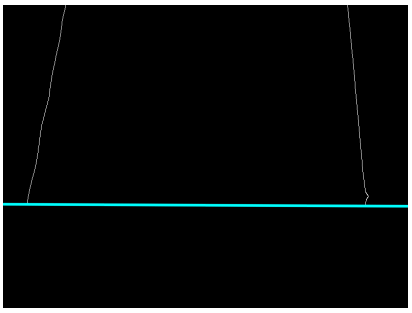


(b) Otsu (objeto)

(c) Canny (objeto)



(d) Hough e retas detectadas (objeto)



Fonte: Elaborado pelo autor.

A Figura 13 mostra que o algoritmo detectou as retas de interesse do objeto, enquanto retas quase perpendiculares foram descartadas, como evidenciado pela Subfigura 13d. O valor estimado para o desalinhamento foi 0,6453°, coerente com a condição nominal, e confirmado pelo retorno do *endpoint* `/coeficiente/0`, mostrado na Figura 14.

Figura 14 – Retorno do *endpoint* `/coeficiente/<tipo>` para o cenário nominal de 0º com a câmera Logitech C920.

```
curl -X GET "http://localhost:5000/coeficiente/0"
{
  "resultado": 0.6453771734098667,
  "skips": 0
}
```

Fonte: Elaborado pelo autor.

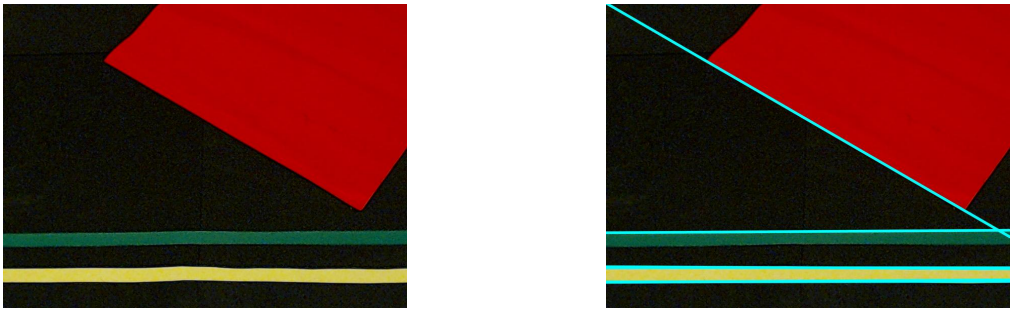
Dessa forma, os resultados indicam que o algoritmo operou conforme esperado no cenário nominal de 0º com a câmera Logitech C920.

4.1.3 Detecção com ângulo nominal de 30º

A detecção seguiu o mesmo encadeamento do cenário de 0º, com a diferença de que o objeto foi posicionado com um ângulo de aproximadamente 30º em relação às linhas de referência. A Figura 15 apresenta o *frame* de entrada e o resultado final, com as retas detectadas destacadas em ciano.

Figura 15 – Entrada e resultado final (condição nominal: 30°).

(a) Frame original **(b)** Resultado final



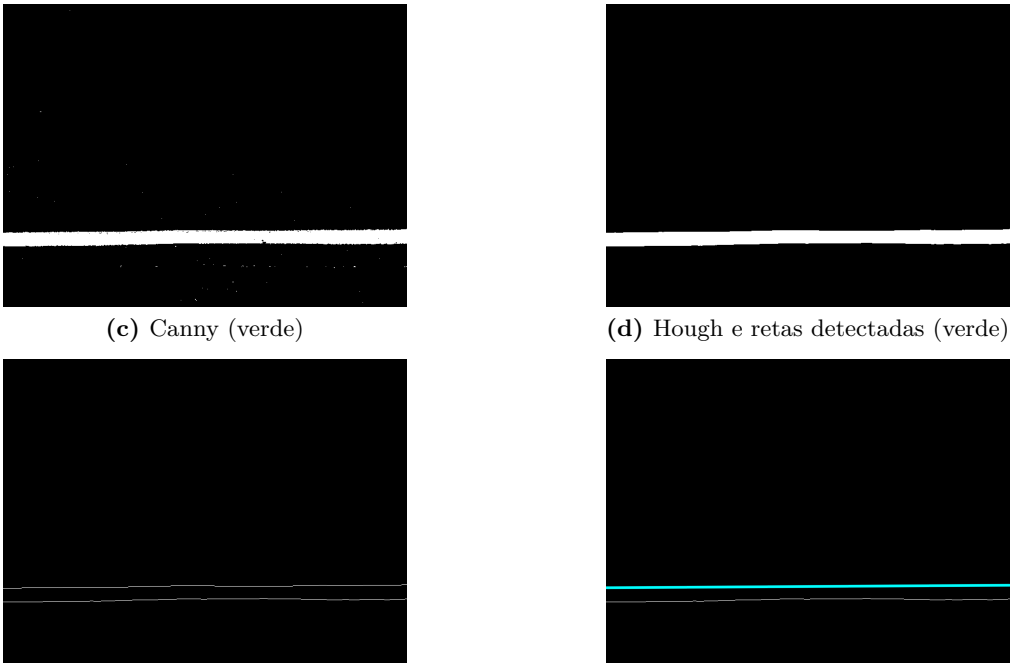
Fonte: Elaborado pelo autor.

Na Figura 15, verifica-se que o algoritmo detectou a reta do objeto. Para as referências, mais de uma reta foi identificada na linha branca, enquanto na linha verde foi detectada apenas a borda superior. O processamento de imagem segue as etapas descritas na Subseção 3.3.2. A Figura 16 apresenta os principais passos para a linha de referência verde.

Figura 16 – Processamento da linha de referência verde (condição nominal: 30°).

(a) Máscara (verde) **(b)** Mediana (verde)

(c) Canny (verde) **(d)** Hough e retas detectadas (verde)



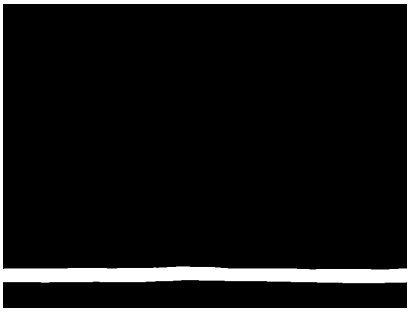
Fonte: Elaborado pelo autor.

Na Figura 16, observa-se que a binarização isola a referência como elemento de interesse. Entretanto, a aplicação dos filtros subsequentes (mediana e Canny) altera parcialmente pixels na borda inferior, o que se relaciona à detecção de reta apenas na borda superior nesse cenário. Em seguida, realiza-se o processamento para a linha de referência branca, conforme a Figura 17.

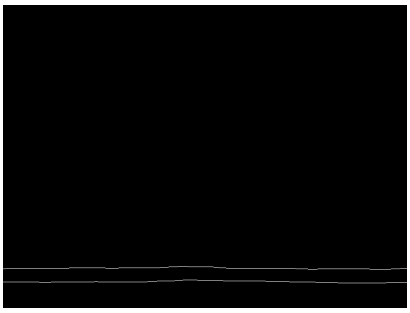
Figura 17 – Processamento da linha de referência branca (condição nominal: 30°).

(a) Máscara (branca)

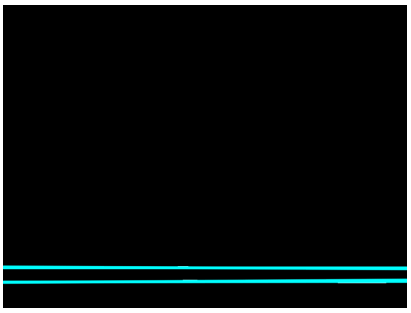
(b) Mediana (branca)



(c) Canny (branca)



(d) Hough e retas detectadas (branca)

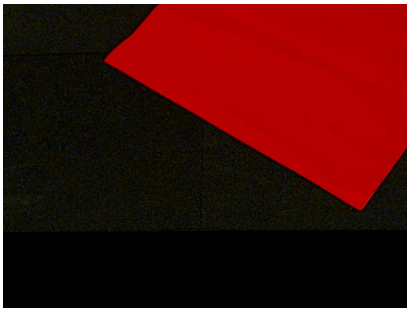


Fonte: Elaborado pelo autor.

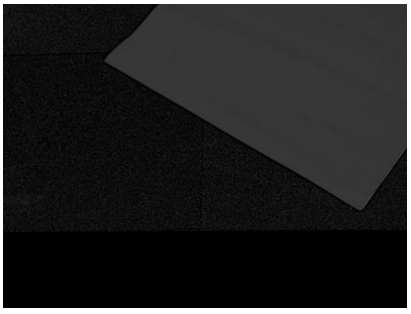
A Figura 17 evidencia que o mesmo procedimento produziu resultados mais estáveis para a referência branca, com detecção em ambas as bordas. Com as referências disponíveis, o algoritmo prossegue para a preparação da imagem para o objeto, conforme a Figura 18.

Figura 18 – Preparação para detecção do objeto (condição nominal: 30°).

(a) Preparação para processamento do objeto



(b) Cinza (objeto)

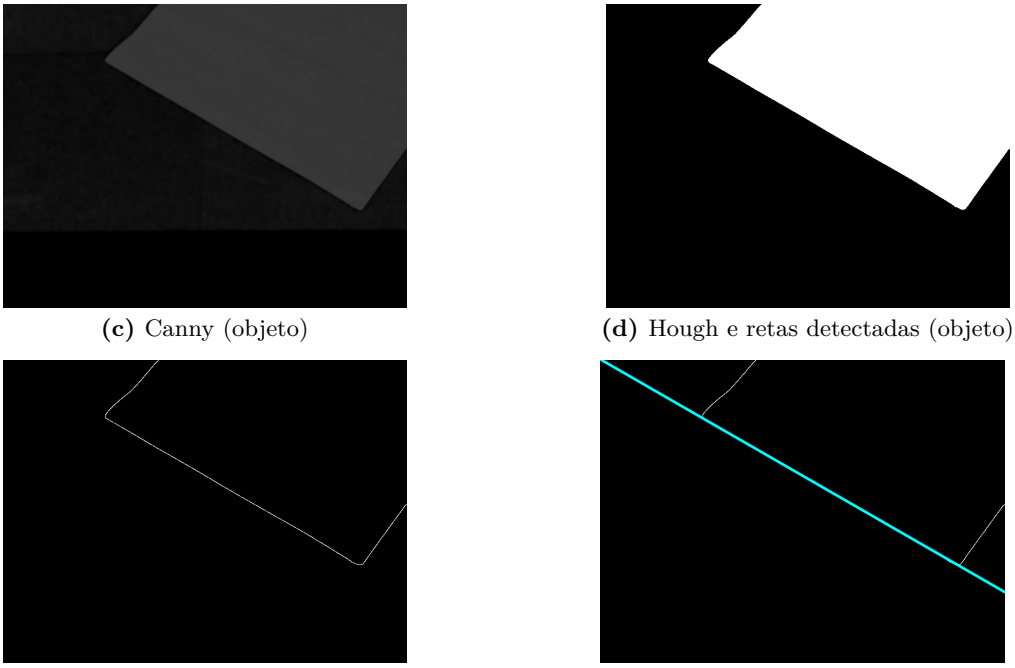


Fonte: Elaborado pelo autor.

Na Figura 18, nota-se novamente a geração de uma imagem em escala de cinza (monocanal), favorecendo o limiaramento de Otsu e a detecção de retas do objeto, procedimento apresentado na Figura 19.

Figura 19 – Processamento das retas do objeto (condição nominal: 30°).

(a) Mediana (objeto) **(b)** Otsu (objeto)



(c) Canny (objeto) **(d)** Hough e retas detectadas (objeto)

Fonte: Elaborado pelo autor.

A Figura 19 mostra que o algoritmo detectou a reta de interesse do objeto, enquanto traços adicionais da borda não foram preservados na detecção, como evidenciado pela Subfigura 19d. O desalinhamento estimado foi de 30,3422°, coerente com a condição nominal, e confirmado pelo retorno do *endpoint* `/coeficiente/0`, apresentado na Figura 20.

Figura 20 – Retorno do *endpoint* `/coeficiente/0` para o cenário nominal de 30º com a câmera Logitech C920.

```
curl -X GET "http://localhost:5000/coeficiente/0"
{
  "resultado": 30.34225075557241,
  "skips": 0
}
```

Fonte: Elaborado pelo autor.

Assim, os resultados indicam que o algoritmo operou conforme esperado também no cenário nominal de 30º com a câmera Logitech C920.

4.2 DETECÇÃO DE RETAS COM A CÂMERA 2 (XWF HD1080P)

Nesta seção são apresentados os resultados obtidos com a câmera XWF HD1080P. Assim como no caso anterior, são mostrados os resultados para os ângulos nominais de 0º e 30º, incluindo (i) o par entrada/saída do *pipeline* e (ii) as principais etapas do processamento para detecção das retas das linhas de referência e do objeto.

4.2.1 Configurações da câmera

O procedimento para obter os parâmetros de uso da câmera para a detecção de retas foi semelhante ao descrito na subseção 4.1.1. Para este caso, os parâmetros definidos são apresentados na Tabela 2.

Tabela 2 – Parâmetros de configuração da Câmera XWF HD1080P.

Parâmetro	Valor
auto_exposure	3
backlight_compensation	0
brightness	-20
contrast	50
exposure_time_absolute	312
focus_absolute	0
focus_automatic_continuous	0
gamma	120
hue	0
power_line_frequency	0
saturation	100
sharpness	0
white_balance_automatic	0
white_balance_temperature	4600

Fonte: Elaborado pelo autor.

Os dados da Tabela 2 são os mesmos retornados pelo *endpoint* de consulta `/params`, mostrados na Figura 21.

Figura 21 – Retorno do *endpoint* `/params` para a câmera XWF HD1080P.

```
curl -X GET "http://localhost:5000/params"
{
  "auto_exposure": 3,
  "backlight_compensation": 0,
  "brightness": -20,
  "contrast": 50,
  "exposure_time_absolute": 312,
  "focus_absolute": 0,
  "focus_automatic_continuous": 0,
  "gamma": 120,
  "hue": 0,
  "path": "/dev/video2",
  "power_line_frequency": 0,
  "saturation": 100,
  "sharpness": 0,
  "white_balance_automatic": 0,
  "white_balance_temperature": 4600
}
```

Fonte: Elaborado pelo autor.

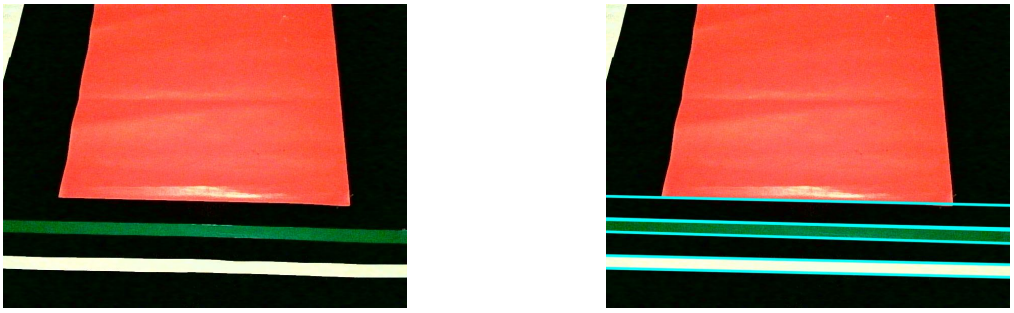
Na sequência, o algoritmo pôde ser executado com as características da imagem favorecendo a detecção dos elementos alvo.

4.2.2 Detecção com ângulo nominal de 0º

Para este cenário, o objeto foi posicionado aproximadamente alinhado às linhas de referência (0º). A Figura 22 apresenta o *frame* de entrada e o resultado final, com as retas detectadas sobrepostas na cor ciano.

Figura 22 – Entrada e resultado final (condição nominal: 0°).

(a) Frame original **(b)** Resultado final



Fonte: Elaborado pelo autor.

Na Figura 22, observa-se que o algoritmo identificou as linhas de referência e estimou o desalinhamento do objeto com um valor baixo, consistente com a condição nominal próxima de 0º.

A Figura 23 detalha as etapas de processamento aplicadas à linha de referência verde. Nota-se que a binarização produz uma máscara bem definida e, após os demais processos, resulta na detecção de retas aproximadamente paralelas que representam as bordas superior e inferior da linha de referência.

Figura 23 – Processamento da linha de referência verde (condição nominal: 0°).

(a) Máscara (verde) **(b)** Mediana (verde)

(c) Canny (verde) **(d)** Hough e retas detectadas (verde)

Fonte: Elaborado pelo autor.

O mesmo comportamento é observado para a linha de referência branca (Figura 24), em que o *pipeline* produz bordas contínuas e a etapa de Hough permite recuperar as retas associadas às arestas da faixa.

Figura 24 – Processamento da linha de referência branca (condição nominal: 0°).

(a) Máscara (branca) **(b)** Mediana (branca)

(c) Canny (branca) **(d)** Hough e retas detectadas (branca)

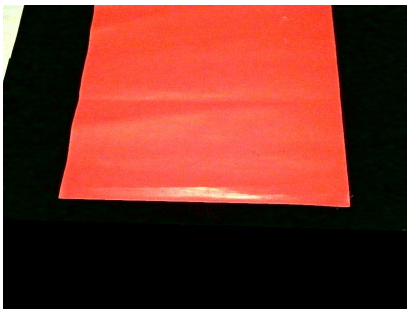
Fonte: Elaborado pelo autor.

Com as retas de referência detectadas, o algoritmo executa a preparação para a detecção no objeto (Figura 25). Nesta etapa, a imagem é reformatada para favorecer a

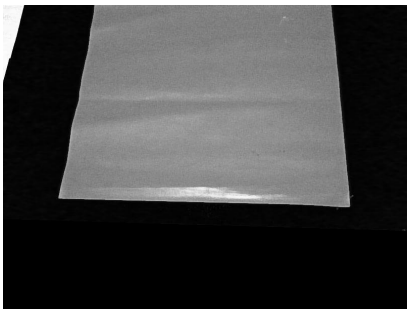
segmentação pelo algoritmo de Otsu.

Figura 25 – Preparação para detecção do objeto (condição nominal: 0°).

(a) Preparação para processamento do objeto



(b) Cinza (objeto)

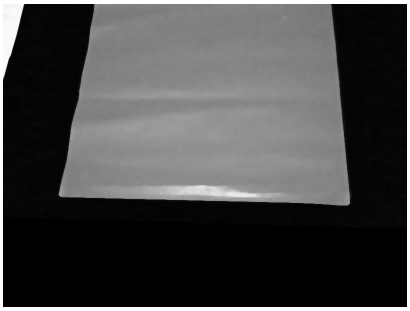


Fonte: Elaborado pelo autor.

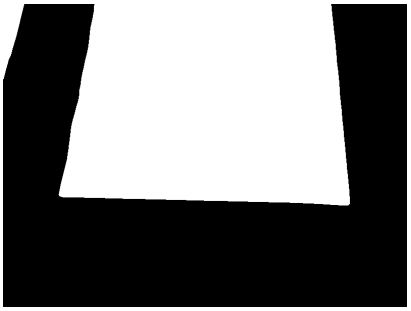
Na Figura 26, observa-se o processamento do objeto: após binarização (Otsu) e Canny, a Transformada de Hough recupera a reta de interesse associada à borda do objeto. As bordas quase perpendiculares (laterais) não dominam a detecção final, pois o algoritmo privilegia a reta com orientação compatível com a inspeção de alinhamento.

Figura 26 – Processamento das retas do objeto (condição nominal: 0°).

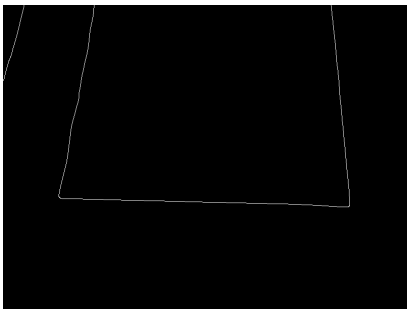
(a) Mediana (objeto)



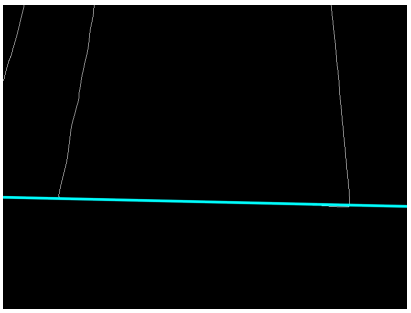
(b) Otsu (objeto)



(c) Canny (objeto)



(d) Hough e retas detectadas (objeto)



Fonte: Elaborado pelo autor.

Por fim, o valor estimado de desalinhamento foi de 0,3583° (próximo de 0º) que é coerente com a montagem do cenário, confirmado pelo retorno do *endpoint* `/coeficiente/0`, mostrado na Figura 27.

Figura 27 – Retorno do *endpoint* `/coeficiente/0` para o cenário nominal de 0º com a câmera XWF HD1080P.

```
curl -X GET "http://localhost:5000/coeficiente/0"
{
  "resultado": 0.3583097691917647,
  "skips": 0
}
```

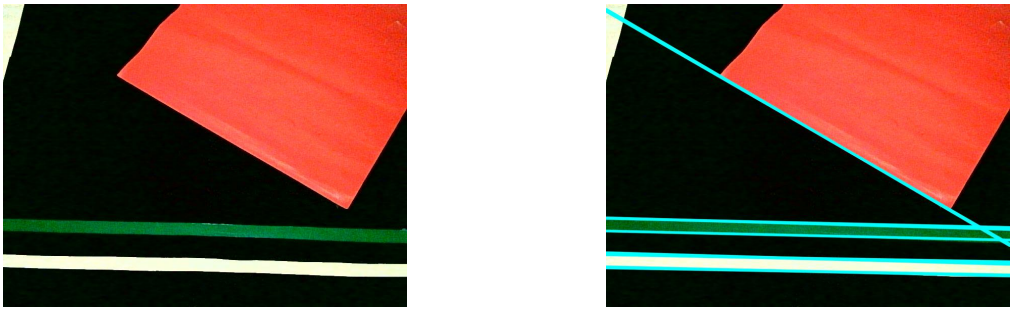
Fonte: Elaborado pelo autor.

4.2.3 Detecção com ângulo nominal de 30º

Neste cenário, o objeto foi posicionado com inclinação aproximada de 30º em relação às linhas de referência. A Figura 28 apresenta o *frame* de entrada e o resultado final do processamento.

Figura 28 – Entrada e resultado final (condição nominal: 30°).

(a) Frame original **(b)** Resultado final



Fonte: Elaborado pelo autor.

Na Figura 28, observa-se que as linhas de referência permanecem detectáveis e que a reta estimada para o objeto apresenta inclinação acentuada, compatível com a condição nominal. O valor numérico apresentado na legenda do par (entrada/saída) indica uma estimativa próxima de 30º, com pequena diferença atribuída à montagem manual do ângulo e a efeitos de perspectiva.

As Figuras 29 e 30 mostram que o processamento das linhas de referência segue o mesmo padrão do caso de 0º: a binarização isola bem as faixas coloridas e, após Canny, a Transformada de Hough evidencia retas associadas às bordas superior e inferior de cada linha de referência, fornecendo a orientação base para comparação.

Figura 29 – Processamento da linha de referência verde (condição nominal: 30°).

(a) Máscara (verde) **(b)** Mediana (verde)

(c) Canny (verde) **(d)** Hough e retas detectadas (verde)

Fonte: Elaborado pelo autor.

Figura 30 – Processamento da linha de referência branca (condição nominal: 30°).

(a) Máscara (branca) **(b)** Mediana (branca)

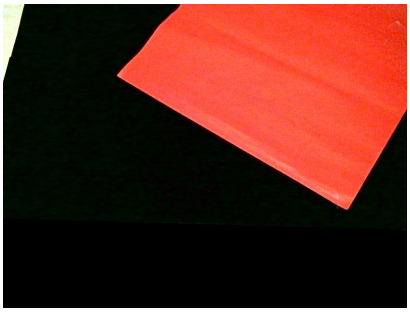
(c) Canny (branca) **(d)** Hough e retas detectadas (branca)

Fonte: Elaborado pelo autor.

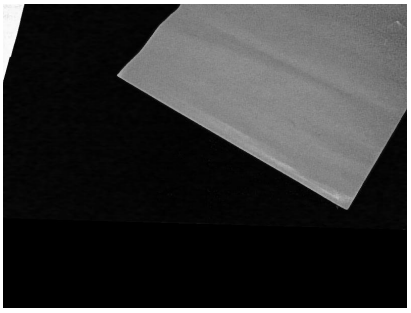
Em seguida, a preparação do recorte/representação do objeto (Figura 31) torna o cenário mais adequado à binarização e à detecção de bordas na etapa específica do objeto.

Figura 31 – Preparação para detecção do objeto (condição nominal: 30°).

(a) Preparação para processamento do objeto



(b) Cinza (objeto)

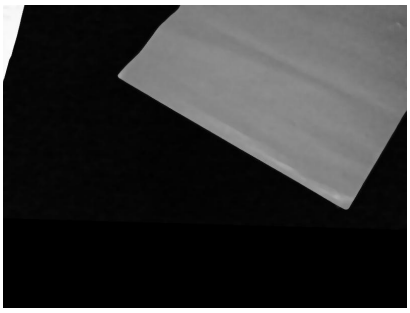


Fonte: Elaborado pelo autor.

Na etapa de detecção das retas do objeto (Figura 32), o resultado do Canny evidencia as bordas do objeto inclinado e a Transformada de Hough recupera a reta dominante com orientação diagonal. O resultado final (sobreposição em ciano) confirma que a reta estimada acompanha a borda do objeto, permitindo calcular o desalinhamento em relação às retas de referência.

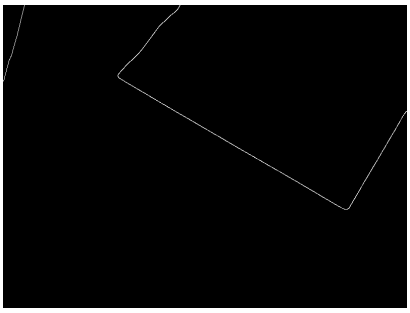
Figura 32 – Processamento das retas do objeto (condição nominal: 30°).

(a) Mediana (objeto)

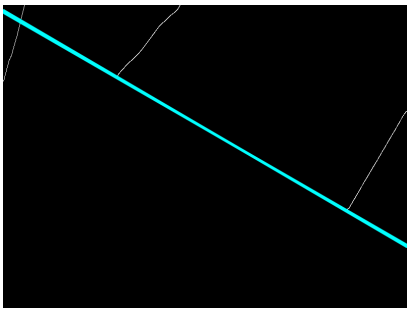


(b) Otsu (objeto)

(c) Canny (objeto)



(d) Hough e retas detectadas (objeto)



Fonte: Elaborado pelo autor.

Por fim, o valor estimado de desalinhamento foi de 29,3550° (próximo de 30º) que é coerente com a montagem do cenário, confirmado pelo retorno do *endpoint* `/coeficiente/0`, mostrado na Figura 33.

Assim, conclui-se que a câmera XWF HD1080P também permitiu a execução consistente do *pipeline* nas condições avaliadas, incluindo um cenário com maior inclinação do objeto (30º), preservando a capacidade de detecção das retas de referência e da reta principal do objeto.

Figura 33 – Retorno do *endpoint* `/coeficiente/0` para o cenário nominal de 30º com a câmera XWF HD1080P.

```
curl -X GET "http://localhost:5000/coeficiente/0"
{
  "resultado": 29.355049869544793,
  "skips": 0
}
```

Fonte: Elaborado pelo autor.

4.3 MÉTRICAS DE DESEMPENHO

Esta seção apresenta os resultados do *benchmark* do *pipeline* para as duas câmeras e para os ângulos avaliados. A análise considera (i) o tempo total por *frame* (`total/frame`) e (ii) o tempo da etapa de detecção do objeto (`det/obj`). As tabelas reportam os percentis p_{50} (mediana) e p_{95} (latência de cauda), além da taxa de sucesso por condição. Os gráficos complementam a interpretação ao mostrar a variação dos tempos em função do ângulo, a comparação entre câmeras e a contribuição relativa das etapas no tempo total (a partir do tempo médio).

4.3.1 Análise geral do tempo de processamento

A Tabela 3 resume os tempos da Câmera 1 (Logitech C920) para cada ângulo, incluindo a taxa de sucesso e a participação média da etapa `det/obj` no tempo total. Observa-se que o tempo típico (p_{50}) permaneceu estável (50,51–55,93 ms), enquanto o p_{95} variou de 54,91 ms a 72,53 ms.

Tabela 3 – Benchmark de tempo por condição — Câmera 1 — C920 Logitech.

Ângulo (°)	p50 total (ms)	p95 total (ms)	p50 proc. obj. (ms)	p95 proc. obj. (ms)	Sucesso (%)	Share proc. obj. (média, %)	FPS @ p95 total
0	55.93	63.62	24.47	30.78	100.0	43.9	15.72
3	53.59	57.59	24.31	25.57	100.0	44.6	17.36
5	55.20	58.27	24.00	25.30	100.0	43.6	17.16
15	55.24	72.53	23.79	30.87	100.0	42.5	13.79
30	52.06	56.42	22.79	23.50	100.0	43.4	17.72
45	50.51	54.91	21.10	22.99	100.0	41.8	18.21

Fonte: Elaborado pelo autor.

A Tabela 4 apresenta o mesmo conjunto de métricas para a Câmera 2 (XWF HD1080P). Nessa câmera, o p_{50} total variou entre 54,24 ms e 60,27 ms, e o p_{95} entre 58,26 ms e 73,65 ms.

Tabela 4 – Benchmark de tempo por condição — Câmera 2 — XWF HD1080P.

Ângulo (°)	p50 total (ms)	p95 total (ms)	p50 proc. obj. (ms)	p95 proc. obj. (ms)	Sucesso (%)	Share proc. obj. (média, %)	FPS @ p95 total
0	58.28	71.25	24.72	30.22	100.0	42.4	14.04
3	60.27	72.06	24.68	30.48	100.0	41.7	13.88
5	57.87	73.65	25.15	31.21	100.0	43.5	13.58
15	54.24	58.26	23.68	24.38	100.0	43.1	17.16
30	56.01	65.55	24.03	27.29	100.0	42.2	15.26
45	56.87	67.09	23.48	26.34	100.0	40.7	14.91

Fonte: Elaborado pelo autor.

Em todas as condições de ambas as câmeras, a taxa de sucesso foi de 100%, indicando que o algoritmo completou o fluxo de detecção (ao menos uma reta de referência e ao menos uma reta do objeto), permitindo o cálculo do desalinhamento. Em termos de composição, a etapa `det/obj` respondeu por mais de 40% do tempo total, caracterizando-a como uma parcela dominante do custo por *frame*.

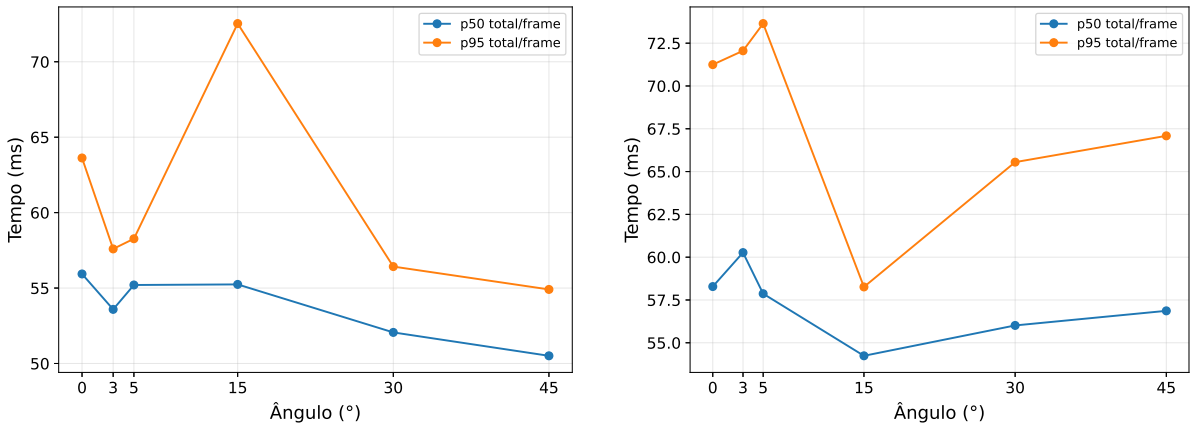
Para facilitar a interpretação prática, valores típicos (p_{50}) na faixa de 50–60 ms correspondem a aproximadamente 17–20 fps, enquanto valores de cauda (p_{95}) próximos de 74 ms implicam quedas pontuais para cerca de 13–14 fps. Assim, ao dimensionar o tempo de resposta de um *endpoint* que estima o desalinhamento a partir de múltiplos *frames*, os percentis ajudam a estimar tanto o comportamento usual quanto a variabilidade em condições menos favoráveis.

4.3.2 Comparações

A Figura 34 apresenta p_{50} e p_{95} do tempo total por *frame* (`total/frame`) em função do ângulo, para cada câmera.

Figura 34 – Tempo total por *frame* (p_{50} e p_{95}) em função do ângulo, para cada câmera.

(a) Câmera 1 (Logitech C920). **(b)** Câmera 2 (XWF 1080p).

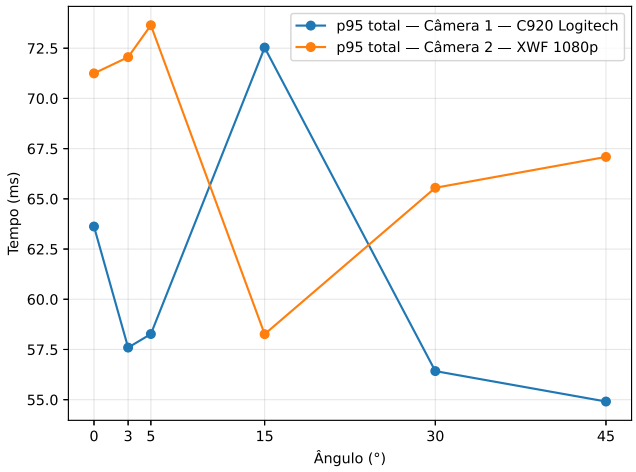


Fonte: Elaborado pelo autor.

Na Câmera 1 (Figura 34a), o p_{95} apresenta um pico em 15°, indicando maior latência de cauda nessa condição, embora o p_{50} permaneça próximo nos demais ângulos. Na Câmera 2 (Figura 34b), o maior p_{95} ocorre em ângulos próximos do alinhamento (0°–5°), com redução acentuada em 15°. Na comparação direta, a Câmera 2 apresenta maior p_{95} em 0°, 3°, 5°, 30° e 45°, enquanto o pior caso da Câmera 1 concentra-se em 15°.

A Figura 35 apresenta uma comparação direta entre câmeras considerando apenas o p_{95} do tempo total por *frame*.

Figura 35 – Comparação entre câmeras: p_{95} do tempo total por *frame* em função do ângulo.



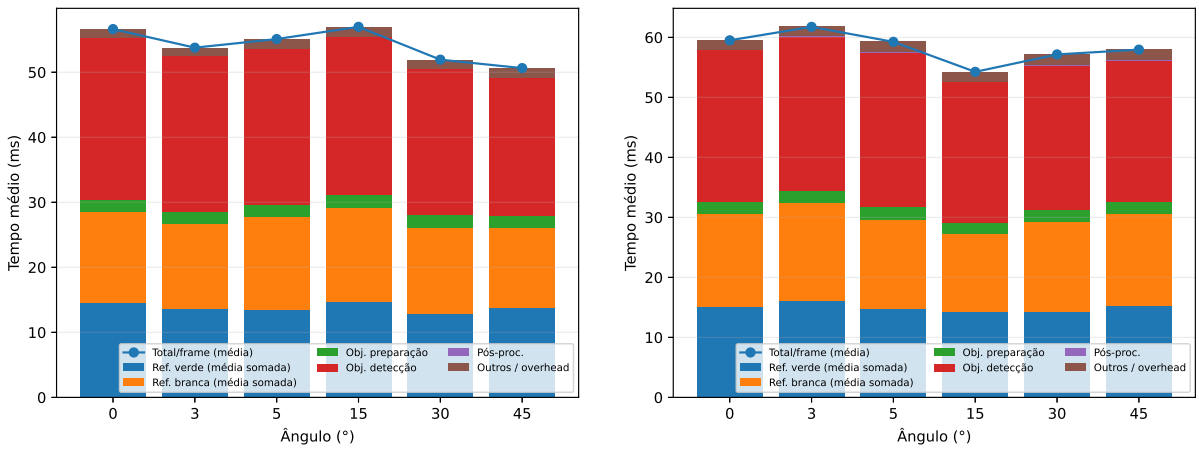
Fonte: Elaborado pelo autor.

Observa-se que a Câmera 2 (XWF) apresenta maior latência de cauda em ângulos próximos ao alinhamento e também em ângulos mais distantes, com uma queda em 15°. Já a Câmera 1 (C920) apresenta o comportamento oposto: um aumento pontual em 15° e valores menores nos demais ângulos, sugerindo maior estabilidade relativa fora dessa condição.

Por fim, a Figura 36 mostra a decomposição do tempo médio por *frame*, agrupando as principais etapas do algoritmo. As categorias incluem: detecção das retas da primeira linha de referência (azul); detecção das retas da segunda linha de referência (laranja); preparação do objeto (verde); detecção das retas do objeto (vermelho); pós-processamento (roxo); e demais operações (conversões, operações matemáticas e manipulação de estruturas de dados) na cor marrom.

Figura 36 – Decomposição do tempo médio por *frame* (somatório de etapas) para cada câmera.

(a) Câmera 1 (Logitech C920). **(b)** Câmera 2 (XWF 1080p).



Fonte: Elaborado pelo autor.

A decomposição evidencia que a proporção entre as etapas é semelhante para as

duas câmeras, com predominância do processamento do objeto (preparação + detecção) no tempo total. As etapas associadas às linhas de referência, por sua vez, tendem a apresentar contribuição total próxima entre si. Como alternativa para reduzir o tempo total por *frame*, uma possibilidade seria remover a detecção de uma das linhas de referência, desde que a robustez do alinhamento permaneça adequada para o cenário de aplicação.

5 DISCUSSÃO

Neste capítulo serão abordados aspectos relacionados aos resultados apresentados: detecção de retas, medida de desalinhamento e métricas de desempenho.

5.1 CONSIDERAÇÕES SOBRE A DETECÇÃO DE RETAS

Com o *frame* capturado favorecido pelos parâmetros de configuração voltados a obter um melhor processamento do algoritmo, era esperado que as retas fossem detectadas. Havia dúvida se as características de cor do *frame* capturado poderiam inviabilizar o algoritmo, dada a diferença de *hardware* entre a Câmera 1 e a Câmera 2. No entanto, a etapa de calibração manual garantiu uma aquisição suficientemente estável para permitir a segmentação confiável dos elementos de interesse.

Na prática, observou-se que os parâmetros de aquisição podem ser ajustados com foco em propriedades relevantes ao processamento (contraste, separação cromática e realce de bordas), e não necessariamente em fidelidade visual da cena. Esse ajuste direcionado favorece as etapas do algoritmo. Como consequência, verificou-se que mesmo uma câmera de menor custo pode ser utilizada, desde que a configuração reduza a sobreposição entre cores do fundo, das linhas de referência e do objeto. Ainda assim, limitações permanecem: variações de iluminação e reflexos podem alterar a resposta de cor e a continuidade das bordas.

Quanto às estimativas angulares, o comportamento observado foi compatível com o princípio do método. Em ângulos próximos do alinhamento, o valor calculado permaneceu próximo de 0°, enquanto inclinações maiores resultaram em estimativas significativamente mais altas. Isso é esperado porque o núcleo do algoritmo consiste em comparar a inclinação (coeficiente angular) das retas detectadas no objeto com a inclinação das retas de referência, isto é, comparar orientações estimadas no plano da imagem.

De forma geral, desde que o *frame* de entrada respeite os pressupostos do *pipeline* — presença de uma ou duas linhas de referência bem segmentadas e do objeto posicionado acima dessas linhas, além de bordas suficientemente definidas — o algoritmo tende a completar o fluxo de detecção e produzir a estimativa de desalinhamento. Esse resultado é corroborado pela taxa de sucesso de 100% apresentada nas Tabelas 3 e 4, indicando que, nas condições experimentais avaliadas, o *pipeline* não retornou códigos de erro e conseguiu detectar retas de referência e do objeto em todos os *frames* medidos.

5.2 CONSIDERAÇÕES SOBRE AS MÉTRICAS DE DESEMPENHO

Do ponto de vista de aplicação, os valores de p_{50} e p_{95} permitem dimensionar expectativas de desempenho: p_{50} reflete o comportamento típico (mais próximo do que se

observa continuamente), enquanto p_{95} captura quedas pontuais de desempenho e é mais adequado para discutir garantias de responsividade.

Os resultados de *benchmark* mostram que o tempo típico por *frame* (p_{50}) permaneceu relativamente estável ao longo dos ângulos para ambas as câmeras, enquanto o p_{95} apresentou variações mais pronunciadas em condições específicas. Essa diferença entre comportamento típico e latência de cauda é relevante: mesmo quando a mediana se mantém estável, episódios pontuais de maior latência impactam a responsividade percebida em aplicações em tempo real.

As oscilações observadas em p_{95} podem estar associadas a variações do próprio conteúdo da cena (por exemplo, mudanças de iluminação, reflexos e textura), que afetam diretamente a densidade e a continuidade de bordas e, por consequência, a estabilidade das etapas de detecção (Canny e Transformada de Hough). Essa interpretação é consistente com a comparação entre câmeras apresentada na Figura 35.

Em termos de comportamento típico, as diferenças entre câmeras foram moderadas: a Câmera 1 apresentou p_{50} total em torno de 55 ms, (ver Tabela 3) enquanto a Câmera 2 se manteve, em média, próxima de 57 ms (ver Tabela 4), embora com maior amplitude (54,24 ms a 60,27 ms).

Esses resultados sugerem que, uma vez que os parâmetros de aquisição estejam ajustados para produzir imagens adequadas à segmentação e à extração de bordas, o custo total por *frame* passa a ser fortemente condicionado pelo processamento realizado no computador (*pipeline* de visão), não importando as diferenças de *hardware* entre as câmeras. Ressalta-se, contudo, que essa leitura se aplica ao tempo medido para processamento do *frame* já capturado; o tempo de captura e entrega do *frame* pelo *driver* não foi incluído no escopo do *benchmark*.

Por fim, a análise da latência de cauda indica que a diferença entre p_{95} e p_{50} variou aproximadamente de 10 ms a 21 ms ao longo das condições avaliadas, com maiores discrepâncias na Câmera 2. Em termos práticos, isso significa que ambas podem apresentar "picos" ocasionais de processamento, mas a Câmera 2 mostrou maior variabilidade temporal. Dessa forma é plausível interpretar que as diferenças de *hardware* entre as câmeras gera maior variabilidade na qualidade do *frame* (ex.: ruído e flutuações de contraste) o que tende a amplificar a instabilidade das etapas sensíveis a bordas, resultando em caudas mais longas.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e a validação de um sistema de inspeção baseado em visão computacional para estimativa de desalinhamento angular, utilizando retas de referência presentes no campo de visão como base de comparação. A adoção de um cenário parcialmente controlado (linhas fixas e objeto posicionado em região conhecida) permitiu explorar um *pipeline* de processamento com técnicas clássicas e interpretáveis, coerentes com aplicações industriais que demandam rastreabilidade, previsibilidade e facilidade de depuração.

A arquitetura proposta organizou o sistema de forma modular, separando aquisição e ajuste de câmera, processamento de imagem e integração externa. Essa separação viabilizou um fluxo não bloqueante de execução, além de permitir uma calibração prática por interface *web* apoiado por Linux via V4L2. Como resultado, foi possível ajustar os parâmetros de aquisição com foco nas características relevantes para o algoritmo (contraste, saturacao e segmentação), o que favoreceu a detecção de bordas por Canny e a detecção de retas pela Transformada de Hough.

A avaliação experimental, conduzida com duas câmeras e múltiplos ângulos nominais, demonstrou que o *pipeline* completou o fluxo de detecção nas condições avaliadas, produzindo saídas numéricas diretamente utilizáveis por sistemas externos via HTTP/JSON. As métricas de desempenho coletadas por instrumentação direta do código mostraram tempos típicos por *frame* estáveis e evidenciaram que a etapa de detecção do objeto concentra a maior parcela do custo computacional.

Foram utilizados métodos clássicos ao invés de abordagens baseadas em *deep learning*. Essas escolhas simplificam a integração e a interpretabilidade, mas limitam a generalização para cenários com maior variabilidade de iluminação, textura e oclusões, bem como exigem adaptações para outros sistemas operacionais e dispositivos.

Como continuidade natural, avanços podem ser feitos em (i) otimizações do processamento do objeto (ii) ensaios com perturbações controladas para quantificar estabilidade; (iii) estudo sistemático do *trade-off* entre resolução, custo computacional e confiabilidade; e (iv) expansão do bloco de integração, por exemplo com um módulo Modbus para comunicação direta com CLPs em ambientes industriais.

Em síntese, o trabalho estabelece uma base técnica e arquitetural sólida e abre espaço para exploração de possibilidades voltadas a maior robustez, desempenho e integração.

REFERÊNCIAS

BASLER AG. **Automated Optical Inspection (AOI)**. Disponível em: https://www.baslerweb.com/en/learning/automated-optical-inspection-aoi/. Acesso em: 26 jan. 2026.

BASLER AG. **Exposure Auto**. Disponível em: https://docs.baslerweb.com/exposure-auto. Acesso em: 15 jan. 2026.

BASLER AG. **Gain Auto**. Disponível em: https://docs.baslerweb.com/gain-auto. Acesso em: 15 jan. 2026.

BASLER AG. **What is Machine Vision?** Disponível em: https://www.baslerweb.com/en/learning/machine-vision/. Acesso em: 26 jan. 2026.

BAŞTAN, Muhammet; BUKHARI, Syed Saqib; BREUEL, Thomas. Active Canny: Edge Detection and Recovery with Open Active Contour Models. **IET Image Processing**, 2017. DOI: 10.1049/iet-ipr.2017.0336.

CANNY, John. A Computational Approach to Edge Detection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, PAMI-8, n. 6, p. 679–698, 1986. DOI: 10.1109/TPAMI.1986.4767851.

COGNEX. **Part and Product Positioning Verification (Part Orientation Verification)**. Disponível em: https://www.cognex.com/en/applications/guidance-and-alignment/part-orientation-verification. Acesso em: 30 jan. 2026.

DUDA, Richard O.; HART, Peter E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. **Communications of the ACM**, v. 15, n. 1, p. 11–15, 1972. DOI: 10.1145/361237.361242.

GONZALEZ, Rafael C.; WOODS, Richard E. **Digital Image Processing**. 3. ed. Upper Saddle River, NJ: Pearson/Prentice Hall, 2008.

GUO, Siyu *et al.* An improved Hough transform voting scheme utilizing surround suppression. **Pattern Recognition Letters**, v. 30, n. 13, p. 1241–1252, 2009. DOI: 10.1016/j.patrec.2009.05.003.

HENKART, Mark. **12+ Image Quality Attributes that Impact Computer Vision**. Commonlands LLC. 2022. Disponível em: https://www.edge-ai-vision.com/wp-content/uploads/2022/05/FW08_Henkart_Commonlands_2022.pdf. Acesso em: 26 jan. 2026.

IMATEST. **Dynamic Range**. Disponível em: https://www.imatest.com/imaging/dynamic-range/. Acesso em: 26 jan. 2026.

KEYENCE. **Positioning/Alignment | Machine Vision Basics**. Disponível em: https://www.keyence.com/ss/products/vision/visionbasics/use/inspection04/. Acesso em: 30 jan. 2026.

KHERADMAND, Amin; MILANFAR, Peyman. Non-Linear Structure-Aware Image Sharpening with Difference of Smoothing Operators. **Frontiers in ICT**, v. 2, 2015. DOI: 10.3389/fict.2015.00022.

LI, D. *et al.* Contrast-Invariant Edge Detection: A Methodological Advance in Medical Image Analysis. **Applied Sciences**, v. 15, n. 963, 2025. DOI: 10.3390/app15020963.

LIN, Hao *et al.* Optimizing Camera Exposure Time for Automotive Applications. **Sensors**, v. 24, n. 16, 2024. DOI: 10.3390/s24165135.

LOGITECH. **C920 HD Pro Webcam**. 2026. Disponível em: https://www.logitech.com/en-eu/shop/p/c920-pro-hd-webcam. Acesso em: 2 fev. 2026.

NATIONAL INSTRUMENTS. **Image Processing with NI Vision Development Module: Alignment**. Disponível em: https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/image-processing-with-ni-vision-development-module.html. Acesso em: 30 jan. 2026.

OPENCV TEAM. **OpenCV Documentation**. 2025. Website. Acesso em: 22 jan. 2026. Disponível em: https://docs.opencv.org/.

PALLETS PROJECTS. **Flask Documentation**. 2025. Website. Acesso em: 22 jan. 2026. Disponível em: https://flask.palletsprojects.com/.

SUPPORT, Logitech. **C920 Technical Specifications**. 2026. Disponível em: https://support.logi.com/hc/de-at/articles/360023307294-C920-Technical-Specifications. Acesso em: 2 fev. 2026.

SZELISKI, Richard. **Computer Vision: Algorithms and Applications**. New York, NY: Springer, 2010. DOI: 10.1007/978-1-84882-935-0.

TELEDYNE FLIR. **Analog Control: Balance White Auto**. Disponível em: https://softwareservices.flir.com/BFS-GE-16S2-BD2/latest/Model/public/AnalogControl.html. Acesso em: 15 jan. 2026.

THE LINUX KERNEL DEVELOPERS. **Video4Linux2 (V4L2) API Documentation**. 2025. Linux kernel documentation. Acesso em: 22 jan. 2026. Disponível em: https://www.kernel.org/doc/html/latest/userspace-api/media/v4l/v4l2.html.

V4L-UTILS PROJECT. **v4l2-ctl(1) — Command line video4linux control tool**. 2025. Project documentation. Acesso em: 22 jan. 2026. Disponível em: `https://www.linuxtv.org/wiki/index.php/V4l-utils`.

WEBCAMTESTS.COM. **Revisão com informações técnicas para "XWF-1080P" (ID 179583)**. 2025. Disponível em: `https://pt.webcamtests.com/reviews/179583`. Acesso em: 2 fev. 2026.

WEBCAMTESTS.COM. **Revisão sobre "XWF-1080P" (ID 169631)**. 2024. Disponível em: `https://pt.webcamtests.com/reviews/169631`. Acesso em: 2 fev. 2026.

ZIVID. **Gain (2D acquisition settings)**. Zivid Knowledge Base. Acesso em: 19 jan. 2026. Disponível em: `https://support.zivid.com/en/latest/reference-articles/settings/2d-settings/2d-acquisition-settings/gain.html`.

APÊNDICE A – OUTROS RESULTADOS

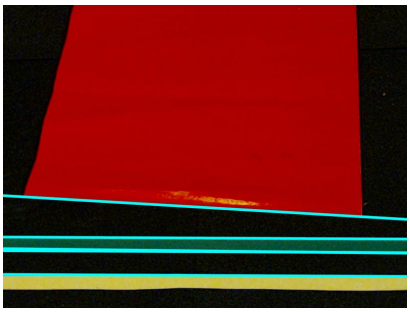
Neste apêndice são apresentados os resultados qualitativos do experimento referentes aos casos de ângulos nominais 3°, 5°, 15° e 45° para as câmeras Logitech C920 e XWF HD 1080p. Por concisão, apresentam-se apenas os resultados finais, uma vez que as etapas intermediárias do processamento, em particular a detecção de retas nas linhas de referência e a detecção de retas do objeto, mostraram-se semelhantes às observadas nos casos detalhados no Capítulo 4.

A.1 CÂMERA LOGITECH C920

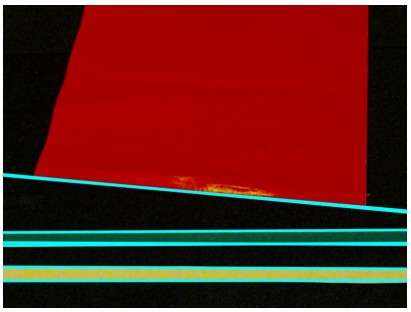
Na Figura 37 são apresentados os resultados do experimento para os casos nominais de 3°, 5°, 15° e 45° da câmera Logitech C920.

Figura 37 – Resultado do experimento para outros casos da câmera Logitech C920.

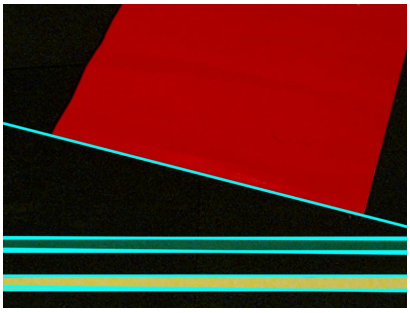
(a) Caso nominal 3°



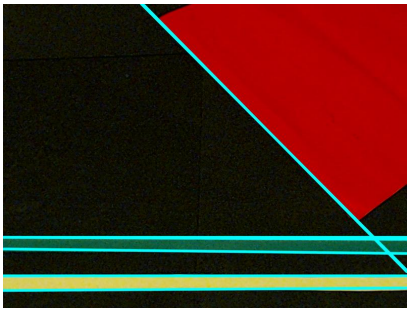
(b) Caso nominal 5°



(c) Caso nominal 15°



(d) Caso nominal 45°



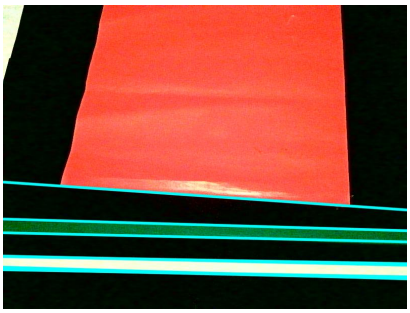
Fonte: Elaborado pelo autor.

A.2 CÂMERA XWF HD 1080P

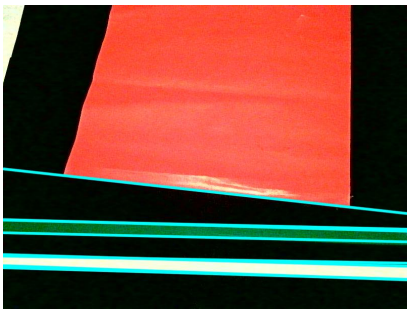
Na Figura 38 são apresentados os resultados do experimento para os casos nominais de 3°, 5°, 15° e 45° da câmera XWF HD 1080p.

Figura 38 – Resultado do experimento para outros casos da câmera XWF HD 1080P.

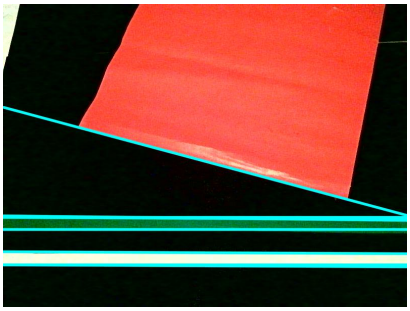
(a) Caso nominal 3°



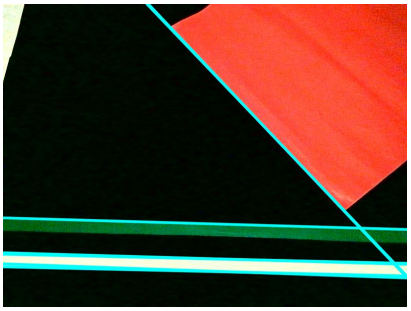
(b) Caso nominal 5°



(c) Caso nominal 15°



(d) Caso nominal 45°



Fonte: Elaborado pelo autor.

APÊNDICE B – CÓDIGO-FONTE DO PROJETO

O código-fonte utilizado neste trabalho está disponível em repositório público no GitHub (`github.com/as-igor/detector-retas-ifpb`).

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus João Pessoa - Código INEP: 25096850
Av. Primeiro de Maio, 720, Jaguaribe, CEP 58015-435, João Pessoa (PB)
CNPJ: 10.783.898/0002-56 - Telefone: (83) 3612.1200

Documento Digitalizado Ostensivo (Público)

Trabalho de Conclusão de Curso

Assunto:	Trabalho de Conclusão de Curso
Assinado por:	Igor Stefan
Tipo do Documento:	Dissertação
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Igor Alexandre Stefan**, **ALUNO (20182610004) DE BACHARELADO EM ENGENHARIA ELÉTRICA - JOÃO PESSOA**, em 13/02/2026 15:21:38.

Este documento foi armazenado no SUAP em 13/02/2026. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1767808
Código de Autenticação: fd45e43d5b