

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

BRUNA CAROLINA ANDRADE

MARCO ANTONIO SORDI COELHO

**DESENVOLVIMENTO DO PROTÓTIPO DE ROBÔ
CORTADOR DE GRAMA**

RELATÓRIO DO PROJETO DA DISCIPLINA DE OFICINA DE INTEGRAÇÃO

PATO BRANCO

2025

1 INTRODUÇÃO

A Robótica Autônoma segundo Siegwart e Nourbakhsh (2004) tem se destacado como uma das principais tendências tecnológicas do século XXI, envolvendo desde processos industriais até aplicações domésticas. O desenvolvimento de sistemas que operam sem intervenção humana, como robôs móveis para serviços, é um campo fundamental para aumentar a eficiência e a constância em tarefas de manutenção, como o corte de grama.

Para que um robô alcance a autonomia, ele requer sistemas de percepção eficazes. A Visão Computacional de acordo com Gonzalez e Woods (2018) é a principal modalidade sensorial empregada, permitindo que a máquina adquira e processe informações do ambiente para tomar decisões de navegação.

No contexto brasileiro, a adoção de soluções robóticas comerciais, como cortadores de grama autônomos, é frequentemente dificultada. Segundo Silva (2011), o custo elevado e a utilização de tecnologias proprietárias restringem o acesso a equipamentos complexos no mercado nacional. Diante disso, este projeto foca na viabilidade de soluções nacionais, que possam ser implementadas com baixo custo.

É neste cenário que o projeto se insere: O protótipo de robô cortador de grama autônomo proposto foi desenvolvido no escopo da disciplina de Oficina de Integração. A solução técnica emprega uma arquitetura de hardware híbrida, que, conforme abordado por Perez *et al.* (2013), é essencial para otimizar o processamento em plataformas de baixo custo. A Raspberry Pi atua como unidade de processamento de alto nível (Visão Computacional) e hospeda a lógica de comunicação para o aplicativo móvel de debugging, enquanto o STM32 Black Pill é dedicado ao controle de baixo nível e determinístico dos atuadores.

O sistema de percepção e navegação do robô baseia-se principalmente em módulos de visão computacional desenvolvidos para atender a requisitos de segurança e eficiência da área de corte. Entre esses módulos, destacam-se a detecção de uma barreira branca perimetral, utilizada como limite virtual de segurança, e o uso de marcadores fiduciais ArUco como referência auxiliar de localização. Adicionalmente, o projeto contempla a classificação visual de grama cortada e não cortada, com o objetivo de otimizar o percurso do robô durante a operação.

No aspecto mecânico, o protótipo utiliza um chassi com tração diferencial, composto por duas rodas motorizadas e uma roda livre, configuração amplamente

adotada em robôs móveis devido à sua simplicidade e mobilidade. O projeto também prevê a utilização de sensores inerciais para auxiliar no controle de orientação e compensar irregularidades do terreno.

Dessa forma, este trabalho tem como objetivo apresentar o desenvolvimento e a validação de uma arquitetura modular para um robô cortador de grama autônomo, explorando a integração entre visão computacional, sistemas embarcados e controle de movimento. Além de contribuir como um exercício prático de integração hardware-software, o projeto estabelece uma base técnica consistente para futuras evoluções, visando a construção de soluções robóticas acessíveis, escaláveis e alinhadas à realidade nacional.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ROBÓTICA AUTÔNOMA E ARQUITETURA DE CONTROLE

A disciplina central para o desenvolvimento do protótipo é a Robótica Móvel Autônoma, cujas plataformas se baseiam em Sistemas Embarcados. Um sistema embarcado pode ser definido como um dispositivo eletrônico microprocessado construído para executar uma ou mais funções específicas.. No contexto robótico, como demonstrado por Igino (2023) em sua aplicação de um controlador de tração, esses sistemas são cruciais para garantir o controle determinístico de atuadores e o tratamento de requisitos de tempo real.

Conforme o Prefácio do livro de Siegwart e Nourbakhsh (2004), a Robótica Móvel é o estudo que busca dar à máquina a capacidade de perceber, raciocinar e agir em um ambiente real sem a intervenção contínua de um operador humano. Mais especificamente, os autores buscam apresentar uma fundação que abrange as camadas mecânica, motora, sensorial, perceptiva e cognitiva da robótica móvel. A complexidade desta área exige a síntese de diversas disciplinas, como cinemática, teoria de controle e inteligência artificial.

O protótipo adota uma Arquitetura de Controle Híbrida, que divide o processamento em dois módulos funcionais, uma estratégia que, conforme Perez *et al.* (2013), é essencial para equilibrar flexibilidade e desempenho em tempo real em plataformas de baixo custo. Os módulos se comunicam via protocolo SPI (Serial Peripheral Interface) e são divididos da seguinte forma:

1. **Módulo Deliberativo (Alto Nível - Raspberry Pi):** Atua como a unidade de processamento central (SBC - Single Board Computer). É responsável pelo processamento de imagens (Visão Computacional), pela lógica de navegação de alto nível e pelo gerenciamento da comunicação com o aplicativo móvel.
2. **Módulo Reativo (Baixo Nível - STM32 Black Pill):** Atua como o microcontrolador dedicado. Sua função é a leitura eficiente de sensores periféricos e o controle preciso e determinístico dos atuadores (motores DC), garantindo a execução das ações em tempo real do sistema propulsor.

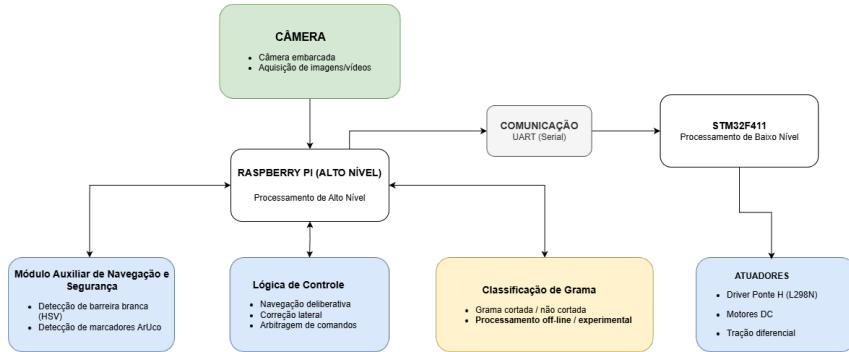


Figura 1: Arquitetura geral do sistema robótico autônomo proposto, destacando os módulos de percepção, controle, comunicação e atuação, bem como a separação entre processamento de alto e baixo nível.

2.2 VISÃO COMPUTACIONAL PARA NAVEGAÇÃO

O processamento digital de imagens, frequentemente denominado Visão Computacional neste contexto de robótica, é um campo fundamental para a autonomia do protótipo.

Segundo Gonzalez e Woods (2018), o processamento pigital de imagens não se limita a sistemas cuja entrada e saída são imagens, abrangendo também técnicas voltadas à extração de atributos e ao reconhecimento de objetos individuais. Essa capacidade de percepção visual constitui o principal sistema sensorial empregado neste trabalho, permitindo que o robô adquira informações do ambiente e tome decisões relacionadas à navegação.

Neste projeto, a visão computacional é aplicada de forma modular. A classificação da grama baseia-se na análise de imagens capturadas pela câmera, enquanto a detecção da barreira branca e dos marcadores ArUco é realizada em tempo real, auxiliando diretamente a navegação e a segurança do robô. Essas informações permitem ao sistema identificar limites físicos e referências visuais para a tomada de decisão.

A biblioteca OpenCV foi adotada como base para o desenvolvimento dos algoritmos de visão, devido à sua ampla utilização acadêmica, robustez e disponibilidade de ferramentas para processamento de imagens em tempo real. Essa escolha permite a implementação eficiente de técnicas de segmentação, detecção de contornos e identificação de marcadores fiduciais, fundamentais para os módulos desenvolvidos neste trabalho.

2.3 DETECÇÃO DE LIMITE POR SEGMENTAÇÃO DE COR

A detecção de limites físicos é fundamental para robôs autônomos que operam em áreas gramadas. A segmentação de cor é uma técnica amplamente empregada para esse propósito, permitindo identificar regiões da imagem com base em informações cromáticas. Em robôs comerciais, a delimitação da área de operação geralmente é realizada por meio de fios de baixa tensão instalados no perímetro. A adoção de visão computacional elimina a necessidade dessa infraestrutura física, reduzindo custos e simplificando a implementação do sistema.

A segmentação por cor é particularmente eficaz quando o objeto de interesse apresenta contraste significativo em relação ao ambiente. No caso deste projeto, o limite da área de corte é representado por uma barreira perimetral branca, cuja identificação é realizada a partir da análise do espaço de cores HSV (Hue, Saturation, Value). Esse modelo é frequentemente preferido em aplicações robóticas por sua maior robustez a variações de iluminação quando comparado ao modelo RGB.

O processo de detecção envolve a limiarização da imagem no espaço HSV, seguida por operações morfológicas destinadas à redução de ruído e à consolidação das regiões detectadas. A partir da máscara binária resultante, são extraídos contornos que permitem determinar a posição da barreira em relação ao robô. Essas informações são então utilizadas como entrada para a lógica de segurança, que pode interromper ou ajustar o movimento do sistema quando o limite é detectado.

Dessa forma, a detecção da barreira perimetral branca atua como um mecanismo de segurança virtual, substituindo sensores físicos tradicionais e contribuindo para a proteção do sistema e do ambiente de operação.

2.4 LOCALIZAÇÃO AUXILIAR POR MARCADORES FIDUCIAIS ARUCO

Marcadores fiduciais são padrões visuais projetados para facilitar a detecção e a estimativa de pose em sistemas de visão computacional. Entre os diversos tipos existentes, os marcadores ArUco destacam-se por sua simplicidade, robustez e ampla adoção em aplicações robóticas e acadêmicas.

Os marcadores ArUco permitem a identificação única de padrões impressos, bem como a estimativa da posição e orientação relativa da câmera em relação ao marcador, a partir de técnicas de visão geométrica. Essa estimativa é realizada por meio da resolução do problema de Perspective-n-Point (PnP),

que relaciona pontos conhecidos no mundo real com suas projeções na imagem capturada.

No presente trabalho os marcadores ArUco foram escolhidos por sua simplicidade computacional e confiabilidade em ambientes controlados, sendo mais adequados a um protótipo acadêmico do que técnicas completas de SLAM. A proposta inicial do sistema previa que a navegação fosse baseada na classificação da grama cortada e não cortada. Contudo, na primeira passagem do robô, essa referência ainda não está disponível. Dessa forma, os marcadores ArUco são utilizados para orientar a navegação inicial, podendo também auxiliar, de forma pontual, na identificação da posição do robô.

2.5 CINEMÁTICA E CONTROLE DE MOTORES EM ROBÔS MÓVEIS

O sistema de locomoção do robô adota tração diferencial, com duas rodas motrizes independentes. Essa configuração permite a execução de movimentos básicos, como avanço, recuo e giros no próprio eixo.

Neste trabalho, o foco esteve na validação do acionamento dos motores e na comunicação serial entre a Raspberry Pi e o microcontrolador STM32, responsável pelo controle de baixo nível. Não foram implementados algoritmos avançados de controle ou sensores adicionais, sendo essas funcionalidades previstas como extensões futuras do sistema.

2.6 CLASSIFICAÇÃO VISUAL DE SUPERFÍCIES NATURAIS

A classificação visual de superfícies naturais, como grama cortada e não cortada, pode ser abordada por meio de técnicas de processamento de imagens e aprendizado de máquina, explorando diferenças de textura, cor e padrão visual. Em aplicações robóticas, esse tipo de classificação pode auxiliar na otimização de trajetórias, evitando passagens redundantes e melhorando a eficiência operacional. Neste projeto, essa funcionalidade é considerada como um módulo auxiliar, desenvolvido de forma independente em outra disciplina, para ser integrada ao sistema principal.

3 MATERIAIS, MÉTODOS E DESENVOLVIMENTO

3.1 MÓDULO DE VISÃO PARA DETECÇÃO DE BARREIRA

O módulo de detecção de limite opera na plataforma de alto nível (Raspberry Pi) e tem como objetivo principal a **segurança reativa**, garantindo que o robô não saia da área de navegação definida pela linha branca perimetral.

3.1.1 Pré-processamento e Segmentação Visual

- Definição da Região de Interesse (ROI):** Para otimizar o tempo de processamento do *frame* de vídeo e focar exclusivamente na área de navegação relevante (próxima ao robô), a análise é restrita à metade inferior da imagem. A ROI é definida pela fatia vertical:

```
frame_roi = frame[height//2 : height, 0 : width]
```

- Segmentação por Limiarização HSV:** A imagem da ROI é convertida para o espaço de cores HSV (*Hue*, *Saturation*, *Value*). A linha branca é isolada utilizando os seguintes limiares calibrados, conforme o script `line_detector.py`:

- Limite Inferior (LOWER_WHITE): $H = 0, S = 0, V = 180$
- Limite Superior (UPPER_WHITE): $H = 180, S = 20, V = 255$

3.1.2 Extração de Dados e Lógica de Controle Reativa

- Extração do Ponto de Decisão:** Os contornos (`cv2.findContours`) são identificados na máscara. O maior contorno (por área) é selecionado. O ponto de decisão para o controle reativo (a distância até a linha) é extraído como:

- Coordenada Vertical (Y): `cy_roi`, correspondente ao **ponto da linha mais próximo do robô (o maior valor de Y do contorno)** dentro da ROI.

- Lógica de Controle Reativa (Sistema de Três Zonas):** A coordenada vertical (`cy_roi`), que representa a distância da linha em pixels, é comparada com limiares de segurança ajustados conforme o código:

- **Zona 3: SEGURO ($Y \leq 235$ px):** O robô avança em velocidade de cruzeiro (None), com a navegação deliberativa (ArUco) no controle.

Tabela 1: Lógica de Controle Reativa de Três Zonas do Módulo de Segurança de Barreira

Zona	Condição	Limiar Calibrado	Comando
Zona 3: SEGURO	$Y \leq 235 \text{ px}$	N/A	None
Zona 2: PERIGO	$Y > 235 \text{ px}$	$\text{LIMIAR_REDUCAO_VEL} = 235 \text{ px}$	'D' (Desacelerar)
Zona 1: CRÍTICO	$Y > 360 \text{ px}$	$\text{LIMIAR_PARADA_CRITICA} = 360 \text{ px}$	'S' (Parar)

- **Zona 2: PERIGO / DESACELERAÇÃO ($Y > 235 \text{ px}$):** O comando 'D' (Desacelerar) é emitido, forçando o robô a reduzir a velocidade.
- **Zona 1: CRÍTICO / PARADA ABSOLUTA ($Y > 360 \text{ px}$):** O comando 'S' (Parar) é emitido. Este comando tem a **prioridade máxima** no sistema de controle.

3.2 MÓDULO DE NAVEGAÇÃO ARUCO

O módulo de navegação ArUco é responsável pela **navegação deliberativa** e pela **correção lateral** do robô. Ele opera com base na detecção de marcadores ArUco, utilizando a Matriz Intrínseca da Câmera (Matriz K) e o algoritmo solvePnP para estimar a pose 3D (distância e desvio lateral) dos marcadores.

3.2.1 Parâmetros de Calibração e Medição

A precisão do sistema de navegação depende diretamente dos seguintes parâmetros físicos e de calibração:

- **Tamanho do Marcador (MARKER_SIZE):** 0.06 metros (6 cm).
- **Distância Alvo (DIST_ALVO):** 0.15 metros (15 cm). É a distância mínima de proximidade que aciona o comando de giro.
- **Matriz Intrínseca (CAM_MATRIX):** Utilizada para a projeção 3D. Os parâmetros de distância focal ($f_x = f_y = 600$) e ponto principal ($c_x = 320, c_y = 240$) são definidos por:

$$K = \begin{pmatrix} 600 & 0 & 320 \\ 0 & 600 & 240 \\ 0 & 0 & 1 \end{pmatrix}$$

Observação: Os valores de calibração, como as distâncias focais da CAM_MATRIX e os limites de desvio ($\text{LIMITE_DESVIO_CM} = 1.0 \text{ cm}$), são definidos para um ambiente

de **simulação** ou **teste simplificado**. Em um ambiente físico real, esses parâmetros requerem calibração detalhada da câmera para garantir precisão métrica.

3.2.2 Estimativa de Pose (Função `calcular_pose_aruco`)

A pose de cada marcador ArUco detectado é calculada através da função `cv2.solvePnP`, que retorna o vetor de rotação (`rvec`) e o vetor de translação (`tvec`) em relação ao frame da câmera.

Os dados de navegação são extraídos de `tvec` da seguinte forma:

1. **Distância Efetiva** (`dist_ponta`): A distância de profundidade (`tvec[2]`) é ajustada subtraindo a distância física da câmera à ponta do robô (`CAMERA_TO_NOTE_FRONT = 0.19 m`).

$$\text{dist_ponta} = \text{tvec}[2] - 0.19$$

2. **Desvio Lateral** (`tx_cm`): O desvio lateral, no eixo X da câmera (`tvec[0]`), é convertido para centímetros. Este valor é crucial para a correção de curso.

$$\text{tx_cm} = \text{tvec}[0] \times 100$$

3.2.3 Lógica de Controle e Prioridade (FSM)

O módulo `logica_planejamento_corte` opera como uma Máquina de Estados Finitos (FSM), executando comandos baseados em prioridade:

3.2.3.1 Correção Lateral (Prioridade Alta)

A prioridade mais alta é dada ao realinhamento do robô.

- **Condição:** $|tx_cm| > \text{LIMITE_DESVIO_CM}$ (onde `LIMITE_DESVIO_CM = 1.0 cm`).
- **Comando:** Emite o comando de correção ('r' para direita, 'l' para esquerda) e ativa o flag `EM_CORRECAO = True`, bloqueando qualquer outra navegação até que o desvio esteja abaixo do limite.

3.2.3.2 Trava de Giro (Prioridade Média)

Após um giro de 180 graus, a flag `AGUARDANDO_NOVO_ARUCO` é ativado para evitar giros repetitivos no mesmo marcador.

- **Ação:** O robô avança ('F') até detectar um novo marcador ($a["id"] \neq \text{ULTIMO_ARUCO_GIRADO}$), momento em que a trava é liberada.

3.2.3.3 Evento de Giro (Prioridade Baixa)

Se o robô está alinhado e a trava está inativa, o robô verifica a distância para acionar o giro de 180 graus.

- **Condição:** $\text{dist_ponta} \leq \text{DIST_ALVO}$ (0.15 m).
- **Comandos e IDs:**
 1. Se $a["id"]$ está em {20, 30}, emite 'L' (Giro Esquerda).
 2. Se $a["id"]$ está em {10, 40}, emite 'R' (Giro Direita).
- **Atualização de Estado:** Em ambos os giros, FAIXA_ATUAL é incrementada e a trava de giro é reativada.

3.2.3.4 Avanço Padrão e Finalização

Se nenhuma das condições de prioridade mais alta for atendida, o comando padrão é avançar ('F').

- **Finalização:** Se $\text{FAIXA_ATUAL} \geq \text{TOTAL_FAIXAS}$ (4), o robô sinaliza a conclusão da varredura.

3.3 MÓDULO DE CONTROLE DE MOTORES E COMUNICAÇÃO SERIAL

O módulo de controle de motores é responsável pela atuação direta sobre o sistema de locomoção do robô, sendo implementado no microcontrolador STM32 Black Pill. Esse módulo deve receber comandos de alto nível provenientes da Raspberry Pi, interpreta-los e gerar os sinais elétricos adequados para o acionamento dos motores de corrente contínua (DC).

A comunicação entre a Raspberry Pi e o microcontrolador é realizada por meio de interface serial UART, utilizando uma taxa de transmissão de 115200 bps. Durante os testes, a comunicação foi validada por meio da ferramenta *picocom*, executada no sistema operacional Linux da Raspberry Pi, permitindo o envio manual de comandos e a visualização das respostas do microcontrolador em tempo real.

Os comandos implementados seguem um protocolo simples baseado em texto, incluindo funcionalidades como ativação e desativação dos motores, ajuste do sinal PWM para controle de velocidade, consulta de estado e visualização de estatísticas de execução. Entre os comandos disponíveis destacam-se `on`, `off`, `pwm <0..100>` e `status`, os quais retornam mensagens de confirmação ou informações de diagnóstico via interface serial.

O controle de potência dos motores é realizado por meio de um driver de ponte H do tipo L298N, responsável por fornecer a corrente necessária aos motores DC e permitir o controle do sentido de rotação. O módulo L298N é alimentado por uma fonte externa e recebe sinais de controle digitais e PWM provenientes do STM32 por meio das entradas IN1, IN2, IN3 e IN4, possibilitando o controle independente de dois motores.

Os motores utilizados são motores DC com caixa de redução, operando na faixa de 4,5 V a 6 V, com relação de redução de 1:48, resultando em uma velocidade aproximada de 200 rpm. Esse tipo de motor é amplamente empregado em robótica móvel devido ao seu bom compromisso entre torque e velocidade, sendo adequado para aplicações de tração diferencial em terrenos planos.

O uso de mensagens de retorno e logs textuais enviados pelo microcontrolador facilitou o processo de depuração e validação do comportamento do sistema, permitindo verificar, por exemplo, o estado atual dos motores e a aplicação correta do sinal PWM.

Esse módulo constitui a camada reativa do sistema de controle do robô, garantindo atuação determinística e em tempo real, em conformidade com a arquitetura híbrida proposta para o protótipo.

Os códigos-fonte desenvolvidos neste trabalho estão disponíveis em um repositório público no GitHub¹.

¹ (https://github.com/BrunaCAndrade/Cone_Prototipo_Robo_Cortador_Grama.git)

4 RESULTADOS

Esta seção apresenta os resultados obtidos a partir dos testes realizados individualmente nos módulos desenvolvidos para o protótipo do robô cortador de grama autônomo. Os experimentos foram conduzidos de forma isolada, com o objetivo de validar o funcionamento básico de cada componente do sistema. Não foram realizados testes de integração entre os módulos no escopo deste trabalho.

Os resultados apresentados a seguir demonstram o comportamento observado em cada módulo, considerando imagens, vídeos, logs e simulações gerados durante o desenvolvimento.

4.1 APLICATIVO DE CAPTURA (CONE)

O aplicativo CONE Data Collector, implementado em FastAPI/Python e acessível via interface web (HTML/JavaScript), foi utilizado como ferramenta de apoio para a captura de imagens e vídeos provenientes da câmera embarcada na Raspberry Pi.

4.1.1 Resultados Operacionais da Interface Web

Durante os testes, a aplicação demonstrou funcionamento estável, cumprindo seus objetivos de suporte à coleta e diagnóstico:

- **Acesso Remoto:** A inicialização do serviço após o *boot* do sistema e o acesso à interface web foram realizados com sucesso por meio de dispositivos externos conectados via Wi-Fi.
- **Funções de Captura:** As funcionalidades de captura de imagens individuais (`/api/photo/single`), sequências de imagens (`/api/photo/sequence`) e gravação de vídeos operaram conforme a especificação do `main.py`. Os arquivos gerados, como a Figura 14 puderam ser armazenados e transferidos para análise posterior.
- **Diagnóstico Integrado:** A visualização de logs (APP, Kernel e System) por meio da interface facilitou o acompanhamento do comportamento do sistema em tempo real, permitindo a identificação e correção de problemas durante a execução dos testes.

A Figura 2 exibe a interface principal do aplicativo, onde os comandos de gravação e as ferramentas de diagnóstico e transferência de dados estão acessíveis.



Figura 2: Interface do aplicativo CONE Data Collector, utilizada para o controle remoto da câmera.

4.2 DETECÇÃO DA BARREIRA BRANCA

O módulo de detecção da barreira perimetral branca, responsável pela segurança reativa do robô, foi avaliado por meio de testes em tempo real, utilizando o fluxo de vídeo proveniente da câmera embarcada na Raspberry Pi. As imagens

foram adquiridas diretamente durante a execução do sistema, sem processamento *off-line* prévio, permitindo a observação contínua do comportamento do algoritmo sob condições operacionais.

4.2.1 Desempenho da Detecção Visual e Segmentação

O algoritmo demonstrou capacidade de identificar regiões correspondentes à barreira perimetral branca por meio de segmentação de cor no espaço HSV. A detecção ocorreu de maneira estável na maioria dos cenários testados, destacando a barreira de forma visualmente consistente no fluxo de vídeo.

- **Estabilidade e Robustez:** A detecção, delimitada pelo contorno amarelo e pelo ponto de decisão vermelho, possibilitou a identificação clara do limite da área de atuação do robô.
- **Impacto da Iluminação:** Observou-se que variações na iluminação do ambiente influenciam diretamente a qualidade da segmentação, exigindo ajustes nos limiares de cor utilizados. Mesmo assim, o método mostrou-se adequado como uma solução de baixo custo computacional para a identificação de barreiras em tempo real.

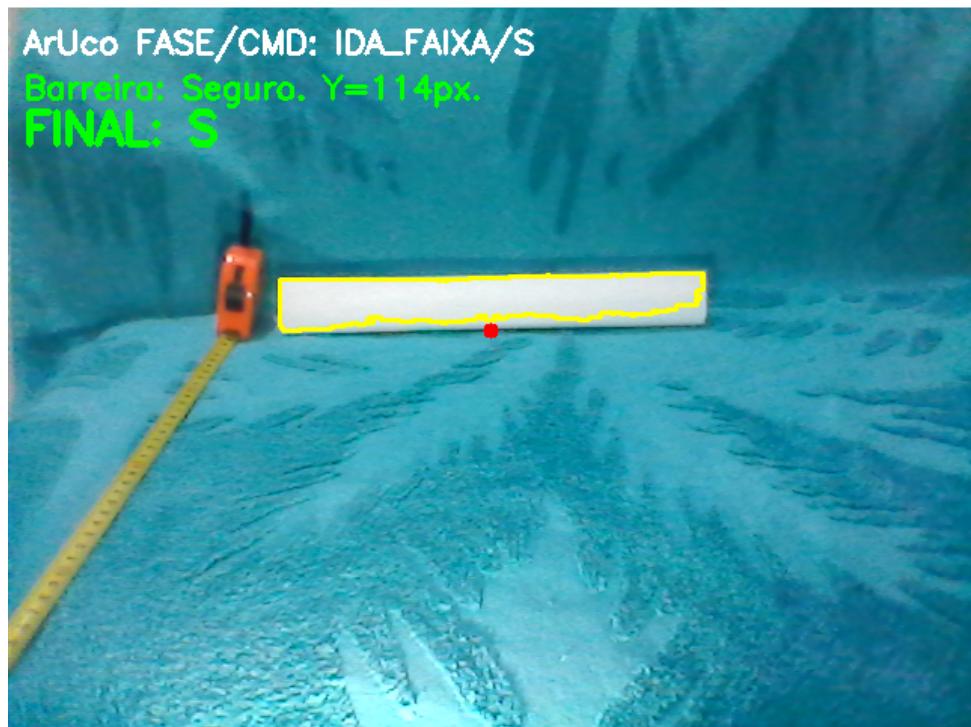


Figura 3: Identificação da barreira.

4.2.2 Avaliação da Lógica de Controle Reativa (Três Zonas)

A Lógica de Controle Reativa atua na prevenção de colisão ou invasão do perímetro ao comparar a coordenada vertical Y da barreira detectada (`cy_roi`) com os limiares definidos na **Tabela 1** da Metodologia. O valor de Y em pixels é a métrica primária para determinar a proximidade da barreira.

4.2.2.1 Análise dos Casos de Teste Visuais e Logs

A eficácia do sistema é confirmada pelos *logs* e *frames* de teste, que demonstram a correta transição entre as zonas de segurança (Seguro $Y \leq 235$ px; Perigo $235 \text{ px} < Y \leq 360$ px; Crítico $Y > 360$ px):

1. **Zona 3 (Seguro):** O robô avança em modo de cruzeiro, com a barreira fora do campo de visão crítico.

- **Log Exemplo:** LIMITE: Seguro. Nenhuma linha perimetral detectada no campo de visão crítico.
- **Teste Visual:** Em Seguro50cm.jpg, a coordenada de decisão é $Y = 114$ px, resultando em FINAL: S (mantém a navegação deliberativa ArUco no controle).

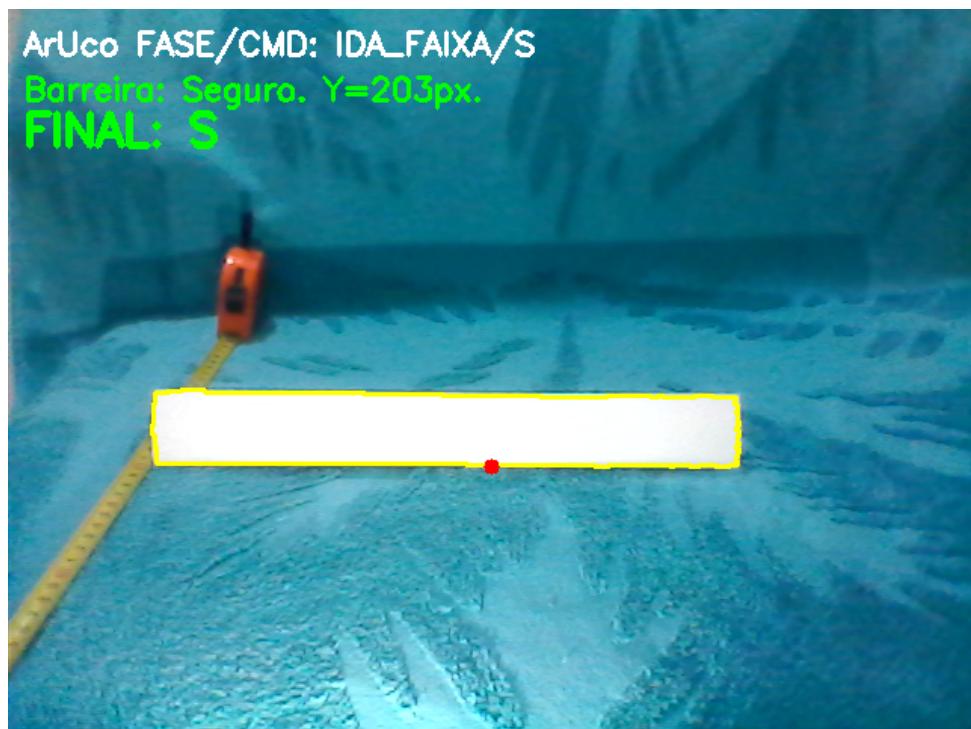


Figura 4: Teste na Zona Segura: A barreira atinge $Y = 203$ px e não é acionado nenhum comando.

2. **Zona 2 (Perigo / Desaceleração):** A barreira cruza o limiar de 235 px, acionando a redução de velocidade.

- **Log Exemplo:** LIMITE: PERIGO! Y=239. Reduzindo velocidade..
- **Teste Visual:** Em Perigo25cm.jpg, a coordenada é $Y = 239$ px, confirmando o acionamento do comando 'D' (Desacelerar). Este comando de segurança assume a prioridade para evitar a invasão da Zona Crítica.

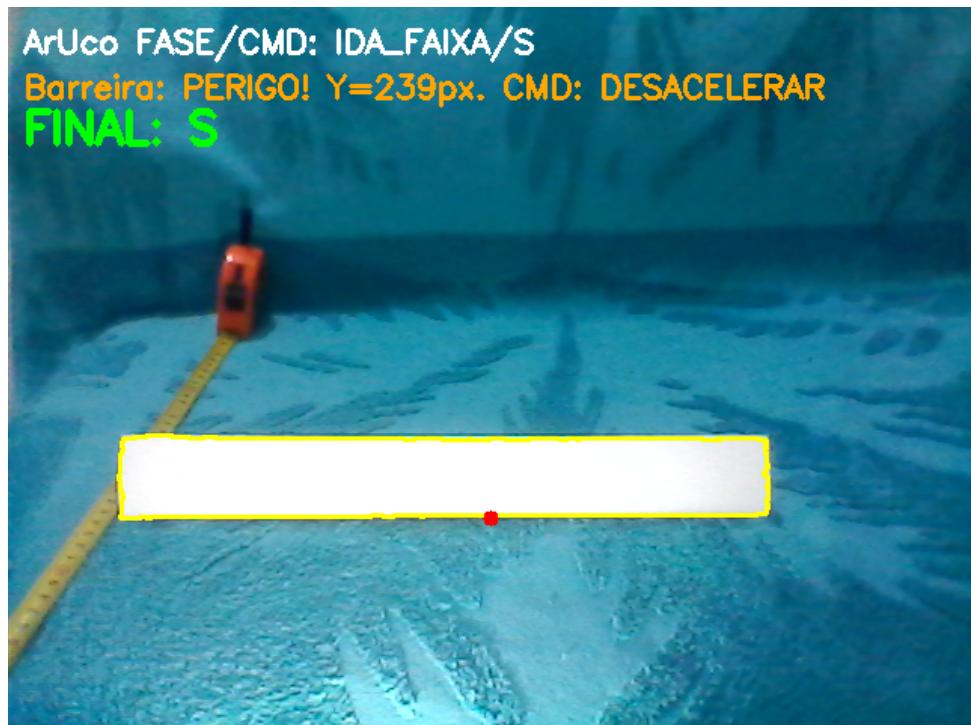


Figura 5: Teste na Zona Perigosa: A barreira atinge $Y = 239$ px e aciona o comando DESACELERAR (D).

3. **Zona 1 (Crítico / Parada Forçada):** A barreira ultrapassa o limiar de 360 px, acionando a parada absoluta do robô.

- **Log Exemplo:** LIMITE: CRITICO! Y=365. PARADA FORÇADA..
- **Teste Visual:** Em Critico10cm.jpg, a coordenada é $Y = 377$ px, resultando no comando 'S' (Parar). Este comando possui a prioridade máxima no sistema de controle.

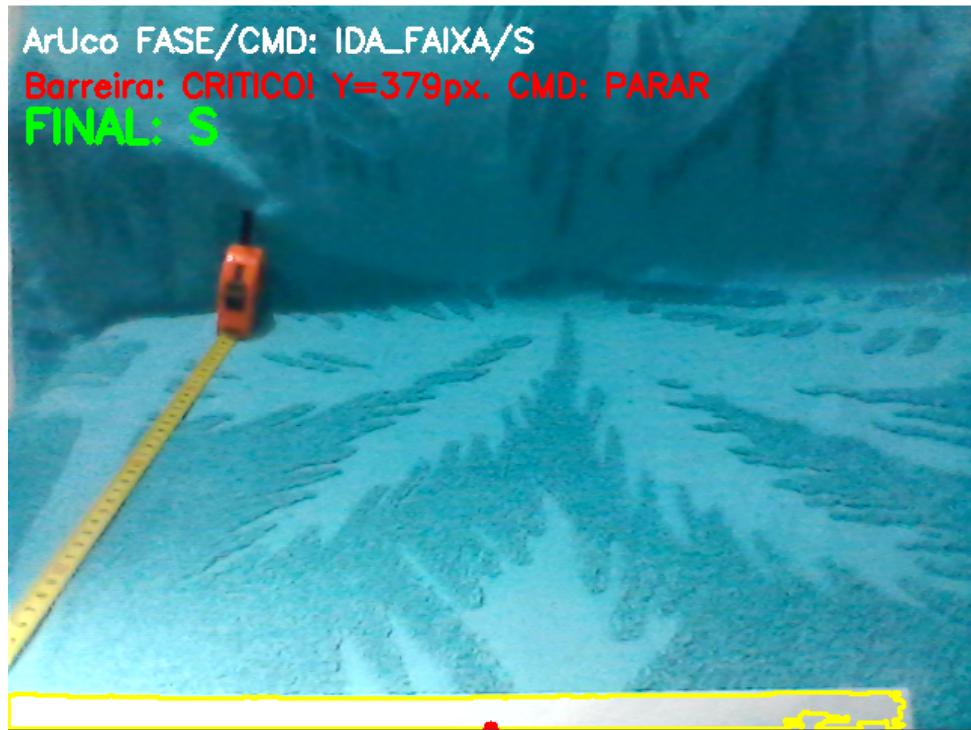


Figura 6: Teste na Zona Crítica: A barreira atinge $Y = 379$ px e aciona o comando **PARAR** (**S**).

O sistema de segurança reativa demonstrou ser eficaz em sua função, garantindo que o comando de segurança seja acionado de forma imediata e escalonada.

4.3 DETECÇÃO DE MARCADORES ARUCO

A detecção de marcadores ArUco foi avaliada em tempo real, utilizando imagens adquiridas diretamente da câmera embarcada durante a execução do sistema. Os testes foram realizados com marcadores posicionados em diferentes orientações e distâncias, permitindo a observação do desempenho do algoritmo em condições dinâmicas.

O sistema foi capaz de identificar corretamente os marcadores presentes no campo de visão, reconhecendo seus respectivos identificadores (IDs) e destacando suas bordas no fluxo de vídeo. A detecção mostrou-se eficiente em ambientes com iluminação adequada e visibilidade direta dos marcadores.

Em situações de encobrimento parcial ou iluminação desfavorável, observou-se redução na taxa de detecção, o que é compatível com as limitações conhecidas dessa técnica. Ainda assim, os resultados indicam que o uso de marcadores ArUco é viável como referência visual em aplicações robóticas.

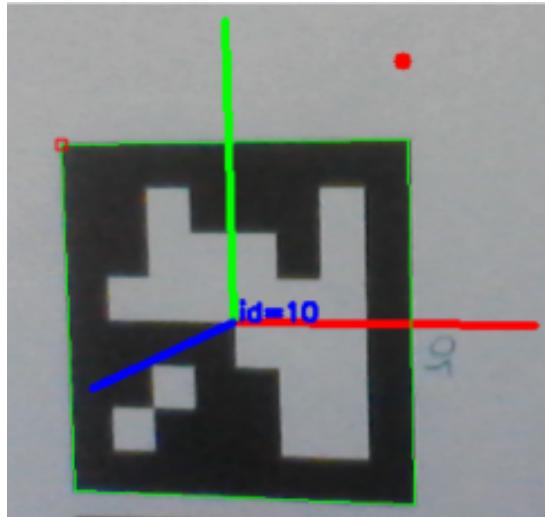


Figura 7: Detecção em tempo real de um marcador ArUco, com identificação do ID e visualização dos eixos de referência utilizados na estimativa de pose.

O módulo de Navegação ArUco foi avaliado por meio da análise dos logs de telemetria durante a varredura da área, focando em duas ações primárias: a correção lateral (realinhamento) e o acionamento dos eventos de giro (fim de faixa). O sistema se baseia na estimativa de pose 3D (distância e desvio lateral) obtida pelo algoritmo cv2.solvePnP.

4.3.1 Validação da Correção Lateral (Prioridade Alta)

A correção lateral é a prioridade máxima e é acionada sempre que o desvio lateral (tx_cm) excede o limiar de $LIMITE_DESVIO_CM = 1.0$ cm.

1. **Correção para a Direita ('r'):** O robô estava à esquerda do marcador (desvio lateral positivo), necessitando mover-se para a direita.

- **Log Visual:** A Figura 8 demonstra o acionamento da correção. O desvio lateral atinge +1.01 cm, excedendo o limiar, e o comando 'r' é emitido. A correção é concluída e o robô retorna ao estado alinhado.
- **Log Exemplo:** WARNING | Entrando em correcao lateral -> DIREITA (desvio 1.01 cm)

```

2025-12-15 01:36:43,185 | INFO | ArUCo 20 | Distancia ponta: 0.19 m | Desvio lateral: 0.7 cm
2025-12-15 01:36:43,245 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: 0.7 cm
2025-12-15 01:36:49,327 | WARNING | Entrando em correcao lateral -> DIREITA (desvio 1.01 cm)
2025-12-15 01:36:55,070 | INFO | Correcao lateral concluida - robo alinhado
2025-12-15 01:36:55,197 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: 0.9 cm
2025-12-15 01:37:01,212 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: 0.7 cm

```

Figura 8: Log de correção lateral para a direita. O sistema detecta desvio de +1.01 cm e realinha o robô.

2. Correção para a Esquerda ('l'): O robô estava à direita do marcador (desvio lateral negativo), necessitando mover-se para a esquerda.

- **Log Visual:** A Figura 9 ilustra o acionamento da correção para a esquerda. O desvio lateral atinge -1.03 cm , e o comando 'l' é emitido, seguido pela confirmação de alinhamento.
- **Log Exemplo:** WARNING | Entrando em correcao lateral -> ESQUERDA (desvio -1.03 cm)

```
2025-12-15 01:37:06,827 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: -0.9 cm
2025-12-15 01:37:06,956 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: -1.0 cm
2025-12-15 01:37:06,957 | WARNING | Entrando em correcao lateral -> ESQUERDA (desvio -1.03 cm)
2025-12-15 01:37:10,028 | INFO | Correcao lateral concluida - robo alinhado
2025-12-15 01:37:10,177 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: -0.9 cm
2025-12-15 01:37:10,304 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: -0.8 cm
2025-12-15 01:37:10,431 | INFO | ArUCo 20 | Distancia ponta: 0.21 m | Desvio lateral: -0.7 cm
□
```

Figura 9: Log de correção lateral para a esquerda. O sistema detecta desvio de -1.03 cm e realinha o robô.

O módulo demonstrou eficácia na detecção e correção de desalinhamentos, garantindo que o robô mantenha a linha de varredura estabelecida.

4.3.2 Validação dos Eventos de Giro (Fim de Faixa)

O evento de giro de 180 graus é acionado quando a **Distância ponta** (dist_ponta) é menor ou igual ao DIST_ALVO = 0.15 m (15 cm). A trava AGUARDANDO_NOVO_ARUCO é ativada para evitar giros múltiplos.

1. Primeiro Giro (Esquerda): Acionado pelo marcador ArUco 20 (indicando giro para a Esquerda).

- **Log Visual:** A Figura 10 exibe o log de acionamento do primeiro giro. A distância atinge 0.15 m, resultando no comando de giro para a ESQUERDA e atualização da posição lateral.
- **Log Exemplo:** CRITICAL | Zona critica no ArUco 20 - giro 180 graus ESQUERDA

2. Segundo Giro (Direita): Acionado pelo marcador ArUco 10 (indicando giro para a Direita).

- **Log Visual:** A Figura 11 mostra o log do segundo giro. Novamente, a distância atinge 0.15 m, acionando o giro para a DIREITA e incrementando a posição lateral.

```

2025-12-15 01:59:19,123 | INFO | ArUCo 20 | Distancia ponta: 0.18 m | Desvio lateral: 0.4 cm
2025-12-15 01:59:19,526 | INFO | ArUCo 20 | Distancia ponta: 0.16 m | Desvio lateral: 0.6 cm
2025-12-15 01:59:19,925 | INFO | ArUCo 20 | Distancia ponta: 0.15 m | Desvio lateral: 0.8 cm
2025-12-15 01:59:19,926 | CRITICAL | Zona critica no ArUCo 20 - giro 180 graus ESQUERDA
2025-12-15 01:59:19,927 | INFO | Posicao lateral estimada = 5 cm

```

Figura 10: Log do primeiro giro (Esquerda). O marcador ArUco 20 aciona o giro de 180 graus ao atingir a distância alvo.

- **Log Exemplo:** CRITICAL | Zona critica no ArUco 10 - giro 180 graus DIREITA

```

2025-12-15 01:47:41,591 | INFO | ArUCo 10 | Distancia ponta: 0.18 m | Desvio lateral: 0.8 cm
2025-12-15 01:47:42,521 | INFO | ArUCo 10 | Distancia ponta: 0.17 m | Desvio lateral: 0.8 cm
2025-12-15 01:47:44,133 | INFO | ArUCo 10 | Distancia ponta: 0.16 m | Desvio lateral: 0.8 cm
2025-12-15 01:47:47,335 | INFO | ArUCo 10 | Distancia ponta: 0.15 m | Desvio lateral: 0.9 cm
2025-12-15 01:47:47,336 | CRITICAL | Zona critica no ArUCo 10 - giro 180 graus DIREITA
2025-12-15 01:47:47,336 | INFO | Posicao lateral estimada = 10 cm

```

Figura 11: Log do segundo giro (Direita). O marcador ArUco 10 aciona o giro de 180 graus ao atingir a distância alvo.

O sequenciamento e acionamento dos comandos de giro demonstraram a execução da lógica de navegação deliberativa (padrão de varredura) do robô.

4.3.3 Finalização e Avanço

O controle de estado FAIXA_ATUAL (varredura) e TOTAL_FAIXAS (4) garante que, após a cobertura de todas as faixas, o robô finalize a varredura.

4.4 LÓGICA DE ARBITRAGEM IMPLEMENTADA

O algoritmo de arbitragem prioriza os comandos de segurança (comando_barreira) sobre os comandos de navegação (comando_aruco), conforme a estrutura:

```

if comando_barreira == "S":          # Prioridade MÁXIMA
    comando_final = "S"
elif comando_barreira == "D":        # Prioridade INTERMEDIÁRIA
    if comando_aruco in ("L", "R"):
        comando_final = comando_aruco
    else:
        comando_final = "D"
comando_final = comando_aruco      # Default (Navegação)

```

4.4.1 Teste Híbrido: Barreira vs. Navegação

A Figura 12 mostra um cenário onde o robô estava navegando ativamente (ArUco detectado) e, ao mesmo tempo, se aproximando da barreira, ativando a Zona de Perigo.

2025-12-15 02:51:09		INFO	aruco	ARUCO 20	dist=0.18m	desvio=-0.5cm
2025-12-15 02:51:09		INFO	aruco	ARUCO 20	dist=0.18m	desvio=-0.5cm
2025-12-15 02:51:09		INFO	aruco	ARUCO 20	dist=0.15m	desvio=-0.5cm
2025-12-15 02:51:10		WARNING	line	LIMITE: PERIGO! Y=276.	Reduzindo velocidade.	
2025-12-15 02:51:15		INFO	aruco	ARUCO 20	dist=0.14m	desvio=-0.4cm
2025-12-15 02:51:15		CRITICAL	aruco	ARUCO 20	-> giro 180	ESQUERDA

Figura 12: Log de Controle Híbrido: Barreira (Limite) acionando a Zona de PERIGO enquanto o ArUco está ativo.

- **ArUco Ativo:** O ArUco 20 é detectado com `dist=0.18m`, indicando navegação ativa.
- **Ação da Barreira:** Em 02:51:10, o limite é detectado em $Y = 276$, excedendo o limiar da Zona 2 ($Y > 235$ px), resultando no `WARNING | LIMITE: PERIGO!` Reduzindo velocidade.
- **Arbitragem e Ação Final:** O comando da barreira ('D' - Desacelerar) prevalece sobre o comando de navegação padrão (assumindo que o ArUco ainda não deu o comando de Giro), garantindo que o robô reduza a velocidade em aproximação ao limite, antes de executar o giro final.
- **Giro Final:** O giro de 180 graus (`CRITICAL — giro 180 ESQUERDA`) é acionado pelo ArUco apenas em 02:51:15 (5 segundos depois), após a desaceleração ser priorizada.

A análise confirma que a lógica de arbitragem do `main_controller.py` funcionou conforme o esperado: o comando de segurança (D) foi aplicado no momento de perigo, mesmo quando o módulo de navegação estava ativo, e o giro final só foi executado quando as condições de segurança permitiram.

4.5 MÓDULO DE CLASSIFICAÇÃO DE GRAMA

O módulo de classificação de grama cortada e não cortada foi desenvolvido como uma contribuição externa no âmbito de outra disciplina, sendo incorporado a este trabalho como apoio conceitual e experimental. O desenvolvimento do algoritmo

foi realizado pelo aluno Lucas, enquanto a captura das imagens e vídeos utilizados na avaliação foi feita por meio da câmera embarcada na Raspberry Pi do protótipo.

Os testes do algoritmo foram realizados de forma independente, utilizando imagens e vídeos previamente capturados, sem integração direta com o sistema de controle ou navegação do robô. O conjunto de dados de treinamento e validação foi composto por um total de 743 imagens, distribuídas em três classes conforme Tabela 2:

Tabela 2: Composição do *dataset* de treinamento e validação.

Classe de Grama	Número de Imagens
Grama Alta	292
Grama Baixa	304
Grama/Obstáculo	147
Total	743

4.5.1 Análise de Desempenho do Treinamento

O treinamento do modelo foi realizado por 20 épocas, com o objetivo de minimizar a função de perda e maximizar a acurácia. O desempenho do modelo no treinamento e na validação é detalhado na Figura 13 (Curvas de Acurácia e Perda).

A análise quantitativa dos logs de treinamento demonstrou um aprendizado consistente e eficaz:

- **Acurácia de Validação (*val_accuracy*):** O modelo atingiu uma acurácia final de 0.9775 (97,75%) na validação.
- **Perda de Validação (*val_loss*):** A perda final de validação foi de 0.1379.

As curvas de desempenho (Figura 13) mostram que a acurácia de validação se manteve consistentemente alta e a perda de validação diminuiu drasticamente, indicando que o modelo possui uma excelente capacidade de generalização e não apresentou *overfitting* (sobreajuste) significativo durante as 20 épocas.

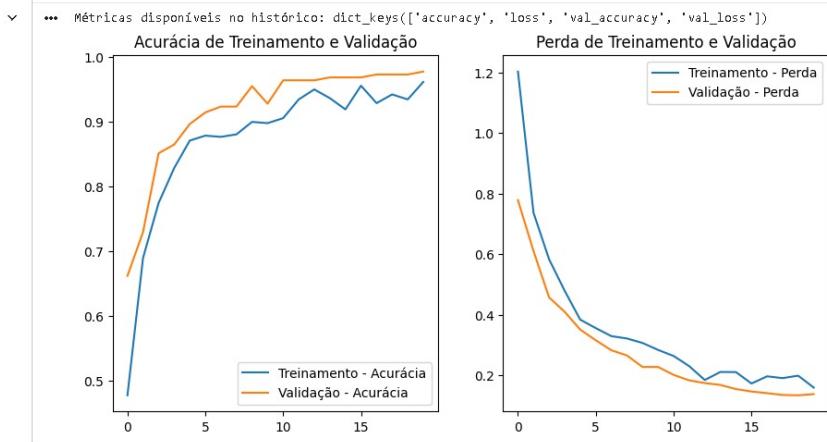


Figura 13: Curvas de Acurácia e Perda para os conjuntos de Treinamento e Validação.

Exemplo de Predição: Em um teste com uma imagem x , o modelo classificou a imagem como *grama_alta* com uma confiança de 57.03%. Apesar de a taxa de confiança ser relativamente baixa, este resultado é promissor e sugere que a acurácia final do modelo pode ser aprimorada com o aumento da diversidade e volume do *dataset*.

Os resultados indicam que o método distingue visualmente regiões de grama cortada e não cortada, com segmentação perceptível entre as classes. No entanto, a avaliação limitou-se aos resultados visuais e ao treinamento do modelo, uma vez que não foram realizados testes quantitativos em tempo real nem utilizado um volume de dados considerado suficiente para máxima eficiência.



Figura 14: Imagem capturada pela Raspberry Pi, classificada 57.03% como grama alta.

5 DISCUSSÃO

A análise dos resultados obtidos neste trabalho evidencia que a arquitetura modular proposta é adequada para o desenvolvimento incremental de sistemas robóticos autônomos. A separação entre módulos de percepção, tomada de decisão e atuação permitiu que cada componente fosse implementado e validado de forma independente.

Os módulos de visão computacional apresentaram desempenho consistente nos testes realizados. A detecção da barreira branca demonstrou-se eficaz como mecanismo de segurança reativa, permitindo identificar o limite operacional em tempo real por meio de segmentação de cor. Apesar de sua sensibilidade a variações de iluminação, a abordagem adotada mostrou-se adequada ao contexto do protótipo, oferecendo uma solução de baixo custo computacional.

De forma complementar, o uso de marcadores fiduciais ArUco mostrou-se viável como referência visual auxiliar para navegação deliberativa. O módulo foi capaz de estimar a pose relativa do robô com precisão suficiente para realizar correções laterais e eventos de giro, possibilitando a execução de um padrão de varredura estruturado. As limitações observadas, como a redução na taxa de detecção em condições de oclusão parcial ou iluminação desfavorável, são inerentes à técnica e reforçam a necessidade de fusão sensorial em versões futuras do sistema.

A lógica de arbitragem implementada desempenhou papel central no comportamento global do robô. A priorização explícita dos comandos de segurança sobre os comandos de navegação garantiu que situações de risco fossem tratadas de forma imediata e determinística. Os testes híbridos demonstraram que, mesmo com a navegação ativa baseada em ArUco, o sistema respondeu corretamente à aproximação da barreira, reduzindo a velocidade ou interrompendo o movimento conforme os limiares definidos.

O módulo de classificação visual de grama, embora não integrado ao controle do robô neste trabalho, apresentou resultados promissores em ambiente controlado, com elevada acurácia no treinamento e validação.

De modo geral, as limitações observadas estão associadas principalmente ao estágio de desenvolvimento do protótipo e às restrições de escopo do projeto, não comprometendo a validade da arquitetura proposta nem os objetivos acadêmicos do trabalho.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um protótipo de robô cortador de grama autônomo baseado em uma arquitetura modular híbrida, integrando processamento de alto nível por meio de visão computacional e controle reativo em tempo real. O projeto foi desenvolvido no contexto da disciplina de Oficina de Integração, consolidando conhecimentos das áreas de sistemas embarcados, robótica móvel e controle de movimento.

Ao longo do desenvolvimento, os principais módulos do sistema foram implementados e avaliados de forma individual e em integrações parciais. Os resultados demonstraram o correto funcionamento dos módulos de percepção visual, da lógica de navegação deliberativa baseada em marcadores ArUco e do mecanismo de segurança reativa por detecção de barreira perimetral. A lógica de arbitragem garantiu comportamento seguro e previsível, assegurando que comandos críticos de segurança prevalecessem sobre comandos de navegação.

Embora a validação completa do sistema em operação contínua e em ambiente externo real não tenha sido realizada, os testes conduzidos confirmaram a viabilidade técnica da arquitetura proposta. As limitações identificadas estiveram relacionadas principalmente a restrições de tempo e à ausência de integração física total, não comprometendo os objetivos acadêmicos do trabalho nem a consistência da solução apresentada.

Como perspectivas para trabalhos futuros, destacam-se a integração de sensores inerciais para fusão sensorial, a realização de testes em ambiente real com diferentes condições de iluminação e terreno, a otimização do controle de movimento por meio de técnicas de controle mais avançadas e a incorporação efetiva do módulo de classificação de grama ao sistema de navegação. Essas extensões têm potencial para aumentar a robustez, a eficiência operacional e o grau de autonomia do robô.

Dessa forma, o projeto cumpriu seu propósito ao demonstrar, de maneira prática, a integração entre hardware e software em um sistema robótico autônomo, estabelecendo uma base técnica consistente para evoluções futuras em contextos acadêmicos ou de pesquisa aplicada.

REFERÊNCIAS

GONZALEZ, Rafael C.; WOODS, Richard E. **Digital Image Processing**. 4th. ed. New York, NY: Pearson, 2018. 1019 p.

IGINO, Wellington Passos. **Ensino de Sistemas Embarcados Baseado em Projeto: Exemplo Aplicado à Robótica**. 117 p. Dissertação (Dissertação de Mestrado) — Universidade Federal da Bahia (UFBA), Salvador, 2023. Disponível em: <<http://www.ppgee.eng.ufba.br/main.php?page=noticiaid=169>>

PEREZ, Anderson Luiz Fernandes; PUNTEL, Fernando Emilio; NANDI, Giann Carlos Spileri; SCHUEROFF, Joildo; TRAMONTIN, Elder Dominghini. Desenvolvimento de um robô explorador para ambientes indoor. **Revista Técnico-Científica do IFSC**, v. 4, n. especial, p. 1–7, 2013. Disponível em: <<https://periodicos.ifsc.edu.br/index.php/rtc/article/view/1268>>.

SIEGWART, Roland; NOURBAKHS, Illah R. **Introduction to Autonomous Mobile Robots**. Cambridge, Massachusetts: The MIT Press, 2004. 321 p.

SILVA, Sérgio Ricardo Xavier da. **Protótipo de um Robô Móvel de Baixo Custo para uso Interdisciplinar em cursos superiores de Engenharia e Computação**. 220 p. Dissertação (Dissertação de Mestrado) — Universidade Federal da Bahia (UFBA), Salvador, Bahia, 2011.