Universidade Federal de Viçosa

Relatório da Tarefa de Trainee do setor de Desenvolvimento de Software da UFVision

Kayo de Melo Lage

Viçosa - MG
23 de dezembro de 2025

Sumário

Capítulo 1

Introdução

O desenvolvimento de Veículos Aéreos Não Tripulados (VANTs) tem avançado significativamente com a integração de sistemas de Visão Computacional, permitindo que aeronaves realizem tarefas complexas de forma autônoma. Este documento detalha o desenvolvimento de um sistema de controle de voo baseado em estímulos visuais cromáticos, realizado como parte do processo de trainee do setor de Desenvolvimento de Software da equipe UFVision.

O projeto consiste na implementação de um pipeline completo que abrange desde a calibração da câmera, a captura e processamento de imagens em tempo real até a conversão desses dados em comandos de navegação via protocolo MAVLink. A solução proposta busca unir robustez no processamento de imagem com uma lógica de controle segura, garantindo que o drone responda de forma precisa a cartões coloridos apresentados à sua câmera.

1.1 Motivação

A motivação primordial deste projeto reside na necessidade de criar interfaces de controle alternativas e intuitivas para sistemas autônomos. Em cenários de competição de VANTs, como os enfrentados pela equipe UFVision, a capacidade de navegar e tomar decisões baseadas em marcadores visuais no ambiente é fundamental.

O desafio técnico de mapear um conjunto limitado de cores (6 tons) para uma gama ampla de movimentos (10 comandos) motivou a criação de uma *Máquina de Estados de Alternância* (*Toggle State Machine*). Esta abordagem não apenas otimiza o uso do espectro de cores disponível, mas também exercita a capacidade de abstração lógica e concorrência de processos, competências vitais para o desenvolvimento de software embarcado em drones de alto desempenho.

1.2 Ferramentas Utilizadas

Para a viabilização deste projeto, foi selecionado um conjunto de ferramentas amplamente utilizadas na indústria e na academia, priorizando a modularidade e a facilidade de integração:

- **Python**: Linguagem base devido ao seu vasto ecossistema de bibliotecas para manipulação de imagens e para controle do drone.
- **OpenCV (Open Source Computer Vision Library)**: Utilizada para a captura de vídeo, conversão de espaços de cor (BGR para HSV) e filtragem morfológica inicial.
- **Scikit-Image**: Empregada para o processamento avançado de máscaras, especificamente na rotulagem e filtragem de componentes conexos por área, garantindo que ruídos de fundo sejam ignorados.
- **ArduPilot SITL (Software In The Loop)**: Simulador de voo que permite testar o código de controle em um ambiente virtual idêntico ao hardware real, mitigando riscos de quedas durante o desenvolvimento.
- **MAVLink & PyMavlink**: O protocolo MAVLink foi utilizado para a comunicação bidirecional entre o script de controle e o drone. A biblioteca `pymavlink` permitiu o envio de comandos de baixo nível, como `SET_POSITION_TARGET_LOCAL_NED`, e a leitura de telemetria em tempo real.
- **Tkinter**: Utilizada para o desenvolvimento de uma interface de emergência (*Kill Switch*), garantindo um mecanismo de pouso imediato operado por thread independente.

Capítulo 2

Metodologias utilizadas

2.1 Abordagens utilizadas para lidar com as cores

O cerne desta seção reside na distinção de cores em ambientes complexos, onde a proximidade cromática exigiu um tratamento diferenciado. Estruturamos um fluxo de processamento robusto a ruídos e variações de iluminação que poderiam comprometer a lógica de controle. Esse processo envolveu o refinamento das máscaras por meio de operações de pós-processamento e a aplicação de limiares (thresholds) de área customizados para cada classe detectada. As subseções a seguir detalham as técnicas e parâmetros utilizados.

2.1.1 Espaço de cor

Embora espaços como RGB, YUV e LAB tenham sido testados, o espaço HSV (Hue, Saturation, Value) foi selecionado por sua capacidade de isolar a informação de cor (matiz) da intensidade luminosa. Isso permitiu definir faixas de threshold mais estáveis para cores críticas como o Marrom e o Preto, cujos valores de brilho costumam oscilar significativamente.

2.1.2 Equalização do brilho da imagem

Para mitigar sombras e reflexos, implementamos o CLAHE (*Contrast Limited Adaptive Histogram Equalization*) [6] no canal V (Value) do HSV. Diferente da equalização global, o CLAHE atua em pequenas regiões (*tiles*), preservando detalhes e garantindo que a detecção de cores escuras (como o Marrom) não fosse prejudicada por pontos de luz excessiva no frame.

2.1.3 Filtragem de Ruído

O pipeline de limpeza de máscaras utiliza três camadas de proteção:

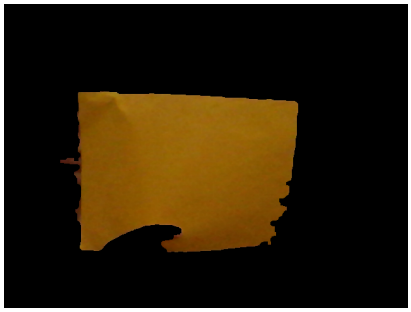
- **Median Blur:** Um kernel de 7×7 para suavizar o frame original antes da binarização.
- **Operações Morfológicas:** Abertura (para eliminar poeira e pixels isolados) e Fechamento (para tapar buracos internos nos objetos detectados).
- **Filtragem por Componentes Conexos (CC):** Foi implementada a remoção de objetos por área utilizando `skimage.measure.label`. Apenas "blobs"com área entre os limites definidos (ex: >30.000 pixels) são considerados válidos, ignorando qualquer interferência de fundo.

2.1.4 Subtração de Máscaras

Devido à proximidade espectral entre certas cores, utilizamos uma lógica de subtração booleana:

- A máscara de **Marrom** é obtida subtraindo-se as máscaras de Vermelho, Azul e Amarelo.
- A máscara de **Preto** sofre a subtração da máscara final de Marrom para evitar confusão entre tons escuros.
- O **Laranja** funciona como um "filtro guloso". A lógica implementada no módulo `vision.py` subtrai ativamente as máscaras das outras cores primárias (Amarelo, Vermelho e Azul) da detecção inicial do Laranja. Isso garante que a máscara final contenha apenas os pixels exclusivos da faixa alaranjada, sem contaminações de matizes adjacentes no espectro HSV.

  A Figura 3.1 ilustra o resultado visual dessa abordagem, demonstrando como o limiar HSV adotado captura várias cores



(a) Laranja capturando Amarelo

(b) Laranja capturando Vermelho

(c) Laranja capturando Azul

(d) Filtro Laranja Isolado

Figura 2.1: Representação da lógica "gulosa"para a cor Laranja.

2.2 Abordagens utilizadas para lidar com a concorrência

O sistema foi desenhado utilizando multi-threading para garantir a segurança e a fluidez do voo:

- **Thread de Visão:** Responsável pela execução ininterrupta do loop de captura e processamento do OpenCV. Os comandos detectados são inseridos em uma `queue.Queue` thread-safe, que serve como ponte de comunicação com o loop principal.
- **Thread de Telemetria:** Realiza a escuta ativa das mensagens `LOCAL_POSITION_NED` via MAVLink. Esta thread atualiza constantemente uma fila circular (*deque*) com as coordenadas (x, y, z) do drone, permitindo que o controlador tenha feedback posicional em tempo real para o cálculo de incrementos.
- **Thread de Emergência:** Opera uma interface gráfica independente em Tkinter. Por rodar em um fluxo separado, ela garante que o comando de pouso imediato (*Kill Switch*) tenha prioridade máxima de execução, sendo

capaz de interromper manobras em curso caso o evento de emergência seja acionado.

2.2.1 Arquitetura Produtor-Consumidor e Sincronização

A interação entre as threads de execução foi estruturada seguindo o padrão de projeto **Produtor-Consumidor**, um problema clássico de computação concorrente formulado originalmente por Edsger W. Dijkstra [1]. Esta arquitetura é essencial para desacoplar a frequência de captura visual da frequência de execução física do drone.

- **Produtor (Thread de Visão):** Esta thread atua produzindo objetos de comando a partir da análise cromática dos frames capturados. Ela alimenta continuamente um buffer compartilhado, independentemente de o drone estar em movimento ou em repouso.
- **Buffer Compartilhado (`command_queue`):** Implementado através de uma `queue.Queue`, este componente atua como o mediador entre as threads. Por ser uma estrutura inerentemente *thread-safe*, ela gerencia internamente os mecanismos de trava (*locks*) e condições de espera, prevenindo condições de corrida (*race conditions*) no acesso aos comandos.
- **Consumidor (Thread Principal):** O loop principal do sistema atua como o consumidor, realizando chamadas bloqueantes à fila através do método `get()`. O consumidor aguarda a disponibilidade de um novo comando para então processar a lógica de estados e os incrementos de posição via MAVLink.

Esta separação garante que o processamento pesado de imagens (OpenCV) não cause latência na thread de controle, permitindo que a telemetria seja lida e processada em paralelo à identificação de novos alvos visuais.

2.2.2 Arquitetura de Execução e Sincronismo

Diferente dos módulos de percepção e telemetria, o **Executor de Movimentos** opera na thread principal do sistema. Esta escolha de design foi feita para garantir o sincronismo das manobras: como a função `move_increments` é bloqueante, o sistema só processa um novo comando da `command_queue` após a conclusão total da trajetória anterior.

Essa abordagem evita a sobreposição de comandos MAVLink e garante que a máquina de estados de alternância só inverta o estado após a confirmação física do deslocamento, utilizando a thread de telemetria apenas como fonte de dados passiva para o controle em malha fechada.

Ou seja, com o objetivo aqui era fazer uma fila de comandos não interrompíveis, não havia uma necessidade da criação de uma thread apenas para os movimentos

2.3 Abordagens utilizadas para lidar com os movimentos do drone

Os movimentos não são enviados como comandos únicos de longa distância, mas sim processados pela função `move_increments`. O deslocamento solicitado é dividido em pequenos passos de no máximo 0,5m. O sistema só envia o próximo passo quando a telemetria confirma que o drone atingiu a posição intermediária anterior, dentro de uma tolerância de 0,1m, garantindo uma trajetória suave e controlada.

Quanto aos movimentos, como foram dadas 6 cores para 10 movimentos possíveis, não seria viável um mapeamento direto de 1 para 1. Nesse sentido, foi escolhido um mapeamento baseado em uma **Máquina de Estados de Alternância** (*Toggle State Machine*).

Nesta abordagem, as cores de navegação funcionam como interruptores que alternam o sentido do movimento a cada nova detecção válida. O sistema armazena o estado atual de cada cor (0 ou 1), invertendo a direção do comando sempre que a mesma cor é apresentada novamente. Isso permite cobrir os dois sentidos (positivo e negativo) dos eixos de translação e rotação utilizando a mesma cor.

As cores críticas (Laranja e Marrom) foram mantidas como comandos absolutos (sem alternância) para garantir a segurança das operações de Pouso e Decolagem. A Tabela 2.3 detalha a lógica implementada:

Cor	**Estado 0 (1ª Detecção)**	**Estado 1 (2ª Detecção)**
Azul	Avançar (+1m)	Retroceder (−1m)
Vermelho	Deslocar p/ Esquerda (+1m)	Deslocar p/ Direita (−1m)
Amarelo	Subir (+1m)	Descer (−1m)
Preto	Girar Horário (+90°)	Girar Anti-horário (−90°)
Laranja	Pousar	Pousar
Marrom	Decolar	Decolar

Tabela 2.1: Mapeamento de Cores e Lógica de Alternância de Estados

Quanto a escolha pela utilização de **Posição Relativa Local**, em detrimento de coordenadas globais (GPS), fundamenta-se na natureza da operação e na necessidade de precisão em curta distância. Visto que os comandos são derivados de estímulos visuais imediatos, é mais robusto e intuitivo comandar o drone para deslocar-se a partir de sua localização atual em vez de calcular coordenadas geográficas absolutas.

Esta abordagem permite que o sistema opere utilizando o referencial `LOCAL_NED` (North-East-Down), onde a origem é o ponto de decolagem. Ao optar por deslocamentos relativos, a função `move_increments` consegue fracionar a trajetória de forma mais estável, minimizando erros acumulados de telemetria e permitindo que o drone mantenha a navegabilidade mesmo em ambientes onde o sinal de GPS possa ser degradado ou inexistente. Além disso, o uso de posições relativas simplifica a lógica da Máquina de Estados, pois cada transição de estado (0 para 1) requer apenas a inversão do sinal do incremento métrico, independentemente da posição absoluta do drone no mapa.

Capítulo 3

Resultados

O sistema foi validado em ambiente de simulação, demonstrando alta precisão na identificação cromática e na execução de trajetórias. A integração entre o processamento de imagem em tempo real e o controle de voo permitiu uma resposta ágil aos estímulos visuais apresentados.

3.1 Eficácia do Processamento de Máscaras

A aplicação do pipeline de visão — composto por equalização CLAHE, desfoque mediano, operações morfológicas e filtragem por componentes conexos — resultou em máscaras binárias extremamente limpas, mesmo em condições de iluminação variável no simulador.

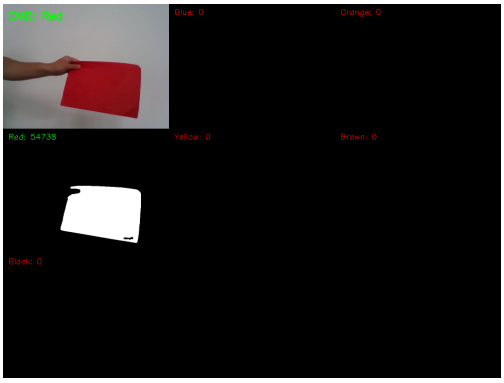
Abaixo, descrevemos o impacto das operações realizadas:

- **Redução de Ruído:** O uso do `BLUR_KERNEL = 7` eliminou cintilações de pixels isolados que poderiam gerar comandos falsos.
- **Operações Morfológicas:** Foi utilizado operações morfológicas de abertura para limpar pequenos ruídos e de fechamento para tornar as máscaras um pouco mais sólidas
- **Filtro de Área:** A configuração do `THRESHOLD_AREA_MIN_REMOVE_CC = 30000` garantiu que apenas o objeto de interesse (a folha colorida) fosse processado, descartando elementos do cenário que possuíam matizes semelhantes.
- **Mosaico de Debug:** A ferramenta de visualização desenvolvida permitiu monitorar simultaneamente o frame original e as seis máscaras, facilitando a calibração dos intervalos HSV em tempo real.

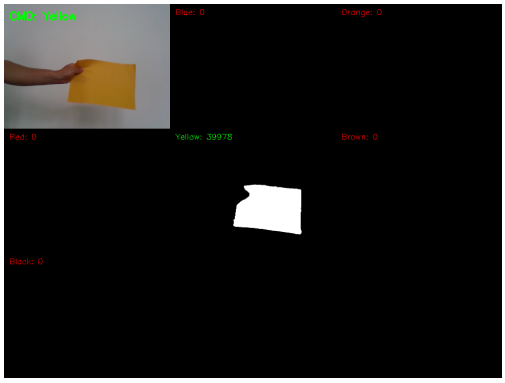
3.1.1 O Filtro "Guloso"e a Separação Cromática

Devido à proximidade dos tons de Laranja, Vermelho e Amarelo no canal *Hue* do espaço HSV, implementou-se uma lógica de subtração "gulosa"no módulo `vision.py`. Esta abordagem é necessária pois o limiar adotado para o Laranja frequentemente captura componentes de matizes adjacentes e opostos, como o Vermelho, Amarelo e Azul.

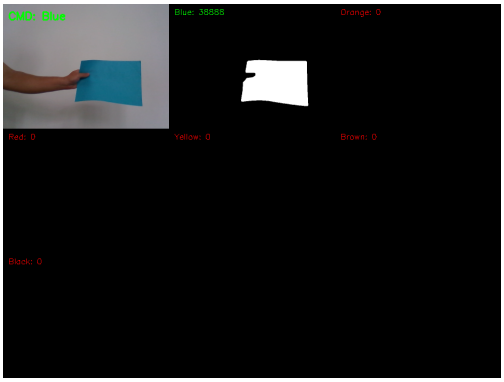
Para resolver este conflito, o sistema processa as máscaras das cores primárias primeiro e, em seguida, as subtrai da máscara bruta do Laranja, garantindo a exclusividade do sinal. A Figura 3.1 demonstra a eficácia desta isolação.



(a) Detecção de Vermelho



(b) Detecção de Amarelo



(c) Detecção de Azul

(d) Laranja isolado (Resultado Final)

Figura 3.1: Resultados da separação cromática e aplicação da lógica de subtração para isolamento do Laranja

3.1.2 Diferenciação de Tons Escuros

A distinção entre o Preto e o Marrom representou um desafio técnico significativo devido à sobreposição de ambos nos espectros de baixa luminosidade (Value) e saturação no espaço HSV. Para mitigar o risco de comandos falsos, o sistema utiliza uma lógica de dependência onde a máscara final de Preto subtrai a máscara resultante do Marrom, assegurando a prioridade da detecção de decolagem.

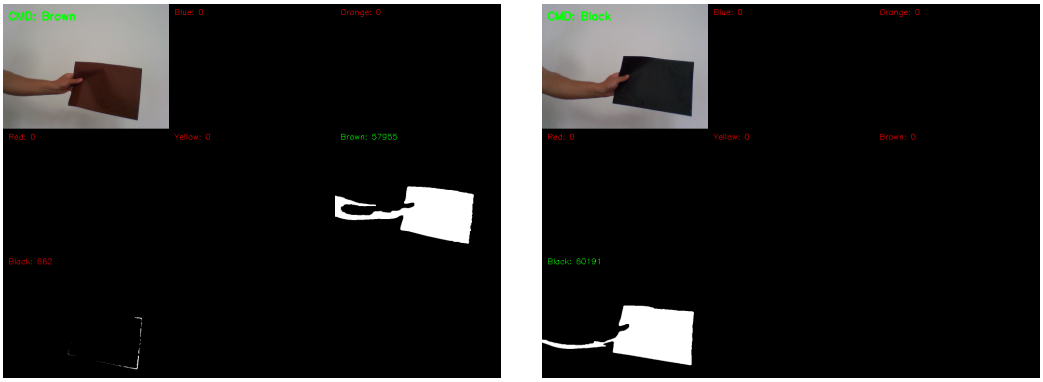
Nesse contexto, o parâmetro `EhPraFazerAbertura` e os thresholds de pixels desse par de cores, funcionam como um mecanismo de robustez configurável de acordo com o ambiente de captura:

- **Processamento do Marrom:** Esta máscara é processada prioritariamente para remover interferências de tons quentes (Vermelho e Amarelo) e frios (Azul). Caso o ambiente de captura possua muitos objetos **pretos** de fundo, recomenda-se configurar `EhPraFazerAbertura=False` para o Marrom. Isso preserva a densidade original da máscara, garantindo que o sinal da decolagem não seja degradado por filtragens excessivas em cenários ruidosos.

- **Processamento do Preto:** A máscara de Preto é processada de forma a permitir a subtração da máscara de Marrom. Conforme documentado, se houver muitos objetos **marrons** no fundo, deve-se utilizar `EhPraFazer Abertura=False` para o Preto. Essa configuração evita a perda de densidade na máscara de Preto após a subtração do Marrom, mantendo a integridade do sinal de rotação (*Yaw*) mesmo em ambientes cromaticamente complexos.

- **Robustez via Threshold:** Em ambos os casos, o ajuste desse parâmetro permite criar máscaras mais "agressivas"ou "conservadoras". Quando a abertura é desativada para manter a densidade, a sensibilidade do sistema é equilibrada através do ajuste dos limites de pixels (`THRESHOLD_BLACK` e `THRESHOLD_BROWN`) no arquivo `settings.py`.

Para o ambiente de captura usado os limiares de ativação definidos no `settings.py` foram estabelecidos em 50.000 pixels para o Marrom e 30.000 pixels para o Preto. Adicionalmente, o sistema utiliza a remoção de componentes conexos com área inferior a 30.000 pixels para ignorar objetos que não correspondam ao alvo de interesse. A Figura 3.2 ilustra o resultado final desta separação.



(a) Máscara do Marrom (b) Máscara do Preto

Figura 3.2: Separação de tons escuros através de subtração mútua e ajustes morfológicos específicos.

3.2 Desempenho da Navegação

O executor de comandos demonstrou estabilidade ao utilizar a técnica de movimentos incrementais de 0,5m. A latência entre a apresentação da cor e o início do movimento foi minimizada pelo uso de threads independentes, mantendo a telemetria sempre atualizada no *deque* para correções instantâneas de posição.

Capítulo 4

Conclusões

4.1 Possíveis melhorias e trabalhos futuros

Embora o sistema tenha atingido os objetivos propostos para o ambiente de simulação, foram identificadas oportunidades de evolução para aumentar a confiabilidade em cenários reais e ruidosos:

- **Robustez de Tons Escuros via Deep Learning:** A detecção baseada puramente em espaços de cor (HSV) e geometria de componentes conexos apresenta limitações significativas em ambientes com sombras projetadas ou fundos texturizados. Um trabalho futuro promissor seria a integração de redes neurais leves, como a *YOLOv8-tiny* ou *MobileNet*, treinadas especificamente para reconhecer os cartões como objetos, tornando a detecção independente da variação cromática do ambiente.

- **Calibração Automática de Thresholds:** Implementar uma rotina de calibração inicial ("*Handshake*") onde o drone, antes da decolagem, analisa o histograma do ambiente e ajusta dinamicamente os valores de `THRESHOLD_BLACK` e `THRESHOLD_BROWN`. Isso permitiria que o sistema se adaptasse a diferentes níveis de iluminância sem a necessidade de alteração manual no código.

- **Filtragem Temporal de Comandos:** Para evitar que ruídos momentâneos disparem comandos falsos, pode-se implementar um filtro de mediana temporal na `command_queue`. Em vez de reagir a um único frame positivo, o sistema exigiria que a cor fosse detectada consistentemente em N frames consecutivos antes de validar o movimento.

- **Suavização de Trajetória com Filtro de Kalman:** A integração de um Filtro de Kalman na leitura das mensagens `LOCAL_POSITION_NED` permitiria uma estimativa de posição mais estável. Isso reduziria as oscilações

causadas por pequenas imprecisões da telemetria durante os cálculos de erro na função `move_increments`.

Capítulo 5

Entregável

Os códigos utilizados neste trabalho, bem como o arquivo `README.md`, instruções de uso e o relatório técnico, estão disponíveis no repositório público do projeto no GitHub [3].

Capítulo 6

Documentação e Materiais de Apoio

Para o desenvolvimento deste projeto, foram consultados os seguintes materiais técnicos e guias fornecidos pela equipe UFVision:

- **SITL e Gazebo:** Guia prático para integração de ambiente de simulação ArduPilot e Gazebo (Documento interno UFVision).
- **Processamento de Imagem:** Documentação oficial da biblioteca OpenCV [5] e Scikit-Image [2].
- **Comunicação MAVLink:** Referência da biblioteca Pymavlink [4].
- **Fundamentos de Concorrência:** Teoria de processos cooperantes [1].

Referências Bibliográficas

[1] Edsger W. Dijkstra. Cooperating sequential processes. *Technological University, Eindhoven, The Netherlands*, 1965. Reprinted in F. Genuys (Ed.), Programming Languages, Academic Press, 1968.

[2] Scikit image contributors. Scikit-image: Image processing in python, 2025. Acessado em: 23 de dezembro de 2025.

[3] Kayo de Melo Lage. *Documentação Técnica UFVision: Módulo de Visão e Controle*, 2025. Scripts: calibra.py, vision.py, settings.py, controller.py e main.py.

[4] ArduPilot Project. Pymavlink documentation, 2025. Acessado em: 23 de dezembro de 2025.

[5] OpenCV team. Opencv library documentation, 2025. Acessado em: 23 de dezembro de 2025.

[6] Karel Zuiderveld. Contrast limited adaptive histogram equalization. In *Graphics gems IV*, pages 474–485. Academic Press Professional, Inc., 1994.

23 de dezembro de 2025

Obrigado pelo Trainee Equipe UFVision

Assinatura:

Kayo de M. Lage
Trainee da equipe de Desenvolvimento de Software