



ESCOLHA DO MÉTODO DE VISÃO COMPUTACIONAL PARA DRONE AUTÔNOMO

Iaroslav Pshenichnikov | iaroslav.p@aluno.ifsc.edu.br¹

Platon Gorinchoy | platon.g@aluno.ifsc.edu.br²

Gabriel Beu Nogueira de Macedo | gabriel.macedo@ifsc.edu.br³

Trabalho de Pesquisa
DT 4 - Inovação Tecnológica, Processos Produtivos e Futuro das Tecnologias

Resumo

A visão computacional permite automatizar a coleta de dados, bem como coletá-los em locais onde o ser humano não consegue chegar. Este trabalho apresenta uma pesquisa sobre diferentes métodos de visão computacional para a leitura do valor de manômetro no âmbito de uma das missões da competição EletroQuad 2026. Como resultado das pesquisas, foi escolhida a rede neural convolucional YOLOv8-Pose, pois ela apresentou distância de busca e precisão satisfatórias, apesar de ser um algoritmo que demanda alto poder de processamento do *hardware* embarcado.

Palavras-chave: Drone; Visão computacional; MobileNet; YOLO; EletroQuad 2026.

Introdução

Uma das missões da competição EletroQuad 2026 consiste em ler os valores de três manômetros fictícios. Para isso, o drone deve ler os valores e transmiti-los à estação de solo de forma autônoma. Dessa forma, é necessário localizar o manômetro na imagem capturada pela câmera, determinar a rotação do próprio manômetro, recortar a área ao redor do centro do manômetro, localizar o ponteiro e avaliar qual valor está sendo apontado.

Devido a restrições como capacidade computacional do computador embarcado, velocidade da conexão com a estação de solo, resolução e ângulo de visão da câmera, foram testados métodos e parâmetros que variam dos mais simples (e menos exigentes em termos de capacidade computacional) aos mais complexos.

¹ Estudante do curso Tecnólogo em Eletrônica Industrial do Instituto Federal de Santa Catarina - Câmpus Florianópolis

² Estudante do curso Técnico Integrado em Mecatrônica do Instituto Federal de Santa Catarina - Câmpus Florianópolis

³ Servidor do Departamento Acadêmico de Eletrotécnica (DAE) do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Fundamentação teórica

A fundamentação teórica deste trabalho aborda diferentes métodos de visão computacional, sendo eles o *Cascade Classifier*, o MobileNet V2, o YOLO e o *Hough Transform*. São apresentados os princípios de funcionamento de cada método, destacando suas características. Essa análise fornece a base conceitual para a comparação e aplicação prática dos algoritmos.

O *Cascade Classifier* é um método de detecção de objetos baseado em *machine learning*, no qual uma função em cascata é treinada a partir de um grande número de imagens positivas e negativas. Destaca-se pelo seu baixo consumo de potência (OpenCV, 2026).

O MobileNet V2 é uma rede neural convolucional (CNN) projetada para classificação de imagens, detecção de objetos e outras tarefas de visão computacional. Foi projetada para ter tamanho reduzido, baixa latência e baixo consumo de energia, tornando-a adequada para inferência no próprio dispositivo e computação em dispositivos com recursos limitados, como sistemas embarcados (Sandler et al., 2018).

O YOLO (*You Only Look Once*) é uma série de sistemas de detecção de objetos em tempo real baseados em redes neurais convolucionais. É uma das estruturas de detecção de objetos mais populares disponíveis na atualidade. O nome refere-se ao fato do algoritmo precisar de apenas uma passagem de propagação para a frente através da rede neural para fazer previsões (Ultralytics, 2026).

O *Hough Transform* é um método de extração de *features* da imagem, que é utilizado para identificar formas básicas em uma imagem, como círculos, linhas e elipses. Ele transfere a representação dessas formas do domínio espacial para o espaço de parâmetros, permitindo uma detecção eficaz mesmo na presença de distorções, como ruído ou oclusão (OpenCV, 2026).

O principal instrumento de integração utilizado foi o ROS (*Robot Operating System*), que é uma coleção de bibliotecas e ferramentas para desenvolvimento de robôs, sendo um *middleware*, que fornece as funcionalidades de um sistema operacional distribuído (Open Robotics, 2026).

Procedimentos metodológicos

A escolha entre os métodos descritos foi realizada de forma iterativa, partindo do que exige menor recurso de processamento para o mais robusto, com cada iteração sendo motivada por uma limitação específica do método anterior. O objetivo era obter a melhor qualidade e velocidade de *inference* em condições de capacidade de processamento limitada.

Inicialmente, foi testado o *Cascade Classifier* para a detecção do manômetro. Este método apresenta a vantagem de exigir baixa capacidade de processamento computacional e prescindir de um conjunto de dados (*dataset*) extenso para o treinamento. Contudo, constatou-se um desempenho insatisfatório na identificação do objeto em distâncias superiores a 1 metro, situação em que a área ocupada pelo manômetro representava menos de 25% da imagem capturada. Além disso, a referida técnica demonstrou-se incapaz de fornecer, de forma intrínseca, os dados referentes à orientação e rotação geométrica do instrumento.

Em seguida foi testado o Mobile Net V2. Essa arquitetura de rede neural tem, potencialmente, um consumo de CPU muito baixo, mas o processo de treinamento está pouco documentado na literatura e o controle de qualidade da inferência é complexo. Não foi possível treinar o modelo de forma rápida e eficaz. Então optou-se por uma arquitetura de modelo mais intensa em processamento, porém mais confiável e versátil.

A arquitetura YOLOv8 foi selecionada como alternativa, respaldada pelos resultados satisfatórios validados em aplicações prévias em competições de robótica. Durante os testes iniciais com esta arquitetura, avaliou-se o modelo YOLOv8-OBB. Entretanto, a fase de treinamento revelou inconsistências e dificuldades técnicas na extração precisa do ângulo de rotação do manômetro. Consequentemente, a solução definitiva adotada para este trabalho consistiu na implementação do modelo YOLOv8-Pose, que permite calcular com precisão a inclinação do manômetro a partir de dois *keypoints*.

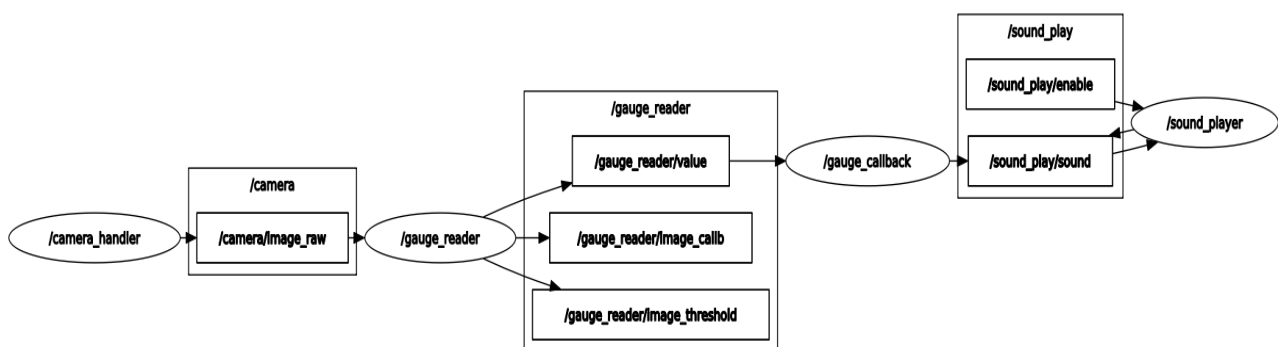
Para reduzir o atraso no processamento no computador de bordo Raspberry Pi 5, foi utilizada a NCNN — um *network inference* de redes neurais otimizado especificamente para ARM (NCNN, 2026). Essa é uma otimização importante, pois mesmo com a carga

máxima do processador do Raspberry Pi 5, a frequência máxima de processamento do fluxo de vídeo estava em 2 Hz, que é considerado um valor baixo.

Depois de identificar o manômetro, ainda é preciso localizar e determinar a rotação do ponteiro do manômetro. Para isso, utiliza-se o *Hough Transform* na área do manômetro, definida pela inferência do modelo. Procura-se então uma linha que comece no centro e termine na borda da área definida pela inferência. Esse método depende muito da qualidade da inferência do modelo. No entanto, para cumprir a missão na competição, o drone ainda precisa voar até os manômetros, ler seus valores e enviá-los para a estação terrestre.

Foi criada uma estrutura de *scripts*, cada um encapsulando uma função específica da missão. O *script* “*gauge_reader*” analisa o fluxo de vídeo usando o YOLOv8 e envia a posição e a orientação do manômetro para o sistema de comunicação por meio de tópicos do ROS. O *script* “*gauge_callback*” funciona como um filtro de integração, se o valor apresentar uma variação inferior a 15% ao longo de alguns segundos, o valor médio desse período é enviado à estação de solo, juntamente com uma solicitação para acionar a sirene. Na estação de solo, o *script* “*sound_player*” está em execução e aguarda uma solicitação do drone. Esta estrutura de nós (*scripts*) do ROS é mostrada na figura 1.

Figura 1 - Estrutura do sistema ROS para esta missão.



Fonte: elaborada pelos autores (2026).

O ROS exigiu baixa capacidade de processamento e permitiu a separação simples das funcionalidades pelos diferentes nós. Nesse aspecto, o ROS realizou o gerenciamento de tarefas de forma satisfatória.

Resultados e discussões

Definida a arquitetura, foram realizados testes para avaliar o desempenho prático sob as restrições reais do *hardware* embarcado da Raspberry Pi 5. Conforme ilustrado na figura 2, o modelo demonstrou capacidade de detectar corretamente o manômetro e estimar sua posição e rotação em tempo real, com boa confiabilidade a curtas distâncias. Com uma resolução da imagem de entrada no modelo de 960×960px, foi possível obter uma detecção estável até aproximadamente 1,2 m, enquanto em distâncias de até 3 m a detecção da posição ainda ocorre, porém com perda de precisão na estimativa angular.

Figura 2 - Funcionamento do modelo após o treinamento.

```
image 1/1 /kaggle/working/dataset/images/val/61432ce4-IMG_20260213_142110_rot0.jpg: 736x960 1 gauge, 12.7ms
Speed: 5.7ms preprocess, 12.7ms inference, 1.8ms postprocess per image at shape (1, 3, 736, 960)
Центр: [ 2034.9 1789]
Стойка: [ 2005.4 2003.7]
Угол манометра: 7.8°
```

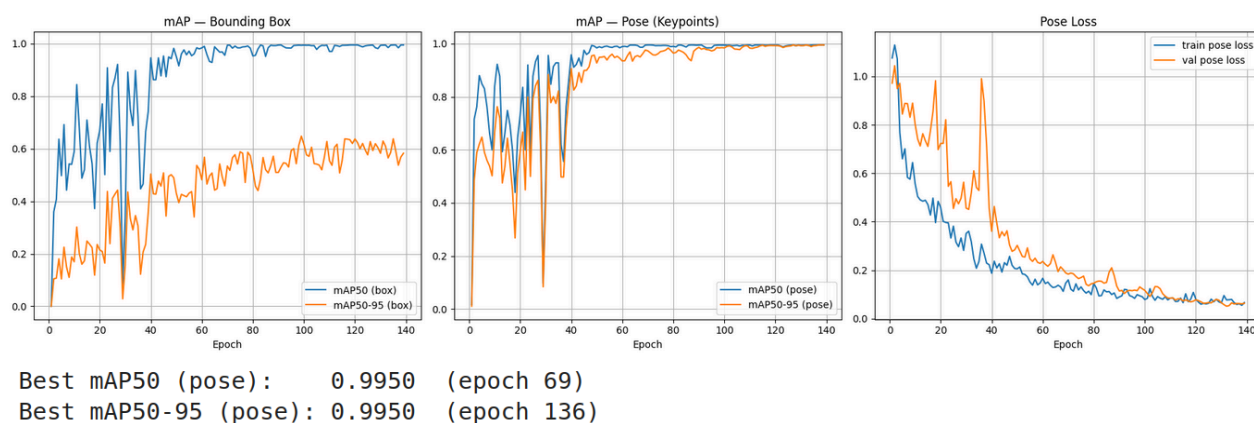


Fonte: elaborada pelos autores (2026).

A figura 3 apresenta os gráficos de precisão média (mAP) e perda durante o treinamento, evidenciando a convergência do modelo e altos valores de desempenho, com mAP50 próximo de 0,995. Apesar desses resultados expressivos em ambiente de

treinamento, na aplicação embarcada observou-se limitação de desempenho computacional, resultando em uma taxa de inferência de aproximadamente 5 Hz, mesmo após otimizações como uso de float16 (em que é feito o arredondamento de números, sacrificando uma precisão insignificante para aumento da velocidade e da economia de memória), ajuste de resolução e redução da carga de processamento. Esses resultados indicam um compromisso entre precisão e desempenho em tempo real no sistema proposto.

Figura 3 - Gráficos de precisão média (mAP).



Fonte: elaborada pelos autores (2026).

Considerações finais

O modelo está pronto para as competições. No entanto, o sucesso da missão dependerá também da lógica geral de execução: antes de ler o manômetro, o drone deve se aproximar dele para obter uma leitura com a melhor resolução possível. A lógica geral está sendo desenvolvida e testada.

A escolha do método de visão computacional não pode ser dissociada da arquitetura geral do sistema: o desempenho do modelo depende do hardware em que será executado. O sucesso da missão depende também da integração do controle de voo via Mavros, biblioteca para a comunicação com o sistema de controle do drone.

Esse sistema se mostrou flexível e prático, embora a curva de aprendizado seja extremamente íngreme. A versão utilizada do ROS, Noetic, é bastante defasada (lançada em 2018), o que causou problemas de incompatibilidade com ferramentas e bibliotecas



atuais. Para trabalhos futuros, está prevista a migração para a versão mais recente do ROS.

Referência ao fomento recebido

Este trabalho foi viabilizado com apoio do Câmpus Florianópolis do IFSC, com a viabilização de espaço físico para as atividades do Laboratório de Drones e destaque para o DAE (Departamento Acadêmico de Eletrotécnica) que fornece a infraestrutura complementar do laboratório. Registra-se também o apoio da Feesc (Fundação Stemmer para Pesquisa, Desenvolvimento e Inovação), por meio do edital “Conexões para Inovar”, que contemplou a equipe de drones do Câmpus Florianópolis nos anos de 2024 a 2026.

Referências

NCNN. ncnn. Disponível em: <<https://github.com/Tencent/ncnn>>. Acesso em: 24 abr. 2026.

OPENCV. **Cascade Classifier**. Disponível em: <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>. Acesso em: 24 abr. 2026.

OPENCV. **Hough Line Transform**. Disponível em: <https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html>. Acesso em: 24 abr. 2026.

OPEN ROBOTICS. **noetic - ROS Wiki**. Disponível em: <<https://wiki.ros.org/noetic>>. Acesso em: 24 abr. 2026.

SANDLER, M. et al. **MobileNetV2: Inverted Residuals and Linear Bottlenecks**. Disponível em: <<https://arxiv.org/abs/1801.04381>>. Acesso em: 24 abr. 2026.

Ultralytics. **YOLOv8**. Disponível em: <<https://github.com/ultralytics/ultralytics>>. Acesso em: 24 abr. 2026.