Tracking de Marcos Fiduciais usando YOLOv8 e Filtro de Kalman para uso em eVTOLs

1st Leonardo Dias
Curso Fundamental
ITA
São José dos Campos, Brasil
leonardo.dias.8730@ga.ita.br

2nd Marcos Sousa
Curso Fundamental
ITA
São José dos Campos, Brasil
marcos.sousa.8842@ga.ita.br

3rd Victor Bastos
Curso Fundamental
ITA
São José dos Campos, Brasil
victor.bastos.8783@ga.ita.br

I. Introdução

O presente projeto concentrou-se na detecção de marcos fiduciais, principalmente aqueles voltados para competições acadêmicas de *eVTOL*, em ambientes dinâmicos. Utilizando a arquitetura da *YOLOv8*, uma *Convolutional Neural Network* (*CNN*) focada em detectção de objetos, o estudo visou aprimorar a capacidade de identificação e localização desses pontos de referência cruciais em imagens capturadas durante o vôo de *eVTOLs*. Devido as limitações presentes na deteção em tempo real, na qual a presença dos marcos não é uniforme em todos os quadros capturados, incorporou-se uma estratégia de otimização baseada no filtro de *Kalman*. Esse filtro, que é uma técnica de estimativa recursiva, foi empregado, juntamente à *YOLOv8*, para melhorar o rastreamento dos marcos fiduciais, mitigando as variações temporais e proporcionando uma trajetória mais precisa.

II. Fudamentaçao Teórica

A. CNNs e a YOLOv8

CNNs são redes neurais baseadas na aplicação de operações de convolução para o aprendizado de padrões e características locais em um *grid*, no presente caso, um tensor que representa um imagem.

Cada camada convolucional consiste em um *kernel* de pesos treináveis que será convoluído com a entrada. As camadas de *pooling* são frequentemente intercaladas entre as camadas convolucionais para reduzir a dimensionalidade dos dados e preservar as características mais importantes, as quais são cruciais para o processo de aprendizado. Além dessas, pode-se ter também as *fully connected layers*, que desempenham tarefas de regressão ou classificação após o *feature extraction*.

A *YOLO* [1] é uma *CNN* projetada para a detecção de objetos em imagens em tempo real. A principal característica da *YOLO* é sua abordagem única para a detecção de objetos em uma única *forward propagation*. Isso pode ser realizado por meio da seguinte estratégia: divisão da imagem em grids, posicionamento de bounding boxes com uma determinada *class probability* associada pelo grid e supressão não-maximal com base em limites de *IoU* (*intersection over union*) entre bounding boxes.

B. Filtro de Kalman

O Filtro de *Kalman* [2] é um algoritmo que utiliza uma série de medidas observadas ao longo do tempo, contendo ruídos estatísticos e outras imprecisões, e produz uma estimativa de variáveis desconhecidas de maneira a minimizar a média do erro quadrático (*MSE*).

Quando aplicado à visão computacional, o Filtro de Kalman processa sequencialmente as informações visuais obtidas (como a posição e a velocidade de um objeto), atualizando suas estimativas sobre o estado desses objetos. Ele faz isso combinando as previsões do modelo com as observações reais, levando em consideração as incertezas tanto nas previsões quanto nas observações. Isso permite que o filtro ofereça uma estimativa mais precisa e suave da posição e do movimento dos objetos ao longo do tempo, o que é crucial em um contexto de rastreamento de objetos, navegação autônoma e sistemas de vigilância.

Dentre as diversas equações do filtro de Kalman, foram utilizadas principalmente, nesse estudo, as listadas abaixo, nas quais está se assumindo um modelo de movimento linear.

1. Modelo de movimento:

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t$$

2. Modelo de observação:

$$\mathbf{z}_t = \mathbf{C}_t\mathbf{x}_t + \mathbf{v}_t$$

$\mathbf{A}_t$, $\mathbf{B}_t$, $\mathbf{C}_t$ são matrizes, $\mathbf{x}_t$ e $\mathbf{u}_t$ são posição e velocidade no instante t, $\mathbf{w}_t$ e $\mathbf{v}_t$ são erros gaussianos com média zero, em que suas matrizes de covariância são respectivamente $\mathbf{Q}_t$ e $\mathbf{R}_t$.

3. Passo de predição:

$$\hat{\mathbf{x}}_t^- = \mathbf{A}_t\hat{\mathbf{x}}_{t-1}^+ + \mathbf{B}_t\mathbf{u}_t$$

$$\mathbf{P}_t^- = \mathbf{A}_t\mathbf{P}_{t-1}^+\mathbf{A}_t^T + \mathbf{Q}_t$$

4. Passo de filtragem:

$$\mathbf{K}_t = \mathbf{P}_t^-\mathbf{C}_t^T(\mathbf{R}_t + \mathbf{C}_t\mathbf{P}_t^-\mathbf{C}_t^T)^{-1}$$

$$\hat{\mathbf{x}}_t^+ = \hat{\mathbf{x}}_t^- + \mathbf{K}_t(\mathbf{z}_t - \mathbf{C}_t\hat{\mathbf{x}}_t^-)$$

$$\mathbf{P}_t^+ = \mathbf{P}_t^- - \mathbf{K}_t\mathbf{C}_t\mathbf{P}_t^-$$

Os expoentes - nas equações acima representam os estados e matrizes antes da filtragem e os expoentes + representam após a filtragem. Nesse sentido, a ideia principal é fazer, sequencialmente, um passo de predição do estado atual, a partir de dados anteriores, e realizar um passo de filtragem, utilizando uma observação atual, para corrigir a predição realizada.

III. Metodologia

- **Seleção e Preparação dos Dados**: Inicialmente, foi realizada a coleta de um conjunto de dados composto por imagens que contêm marcos fiduciais. Foi criado um *dataset* próprio de cerca de 200 imagens. A preparação dos dados envolveu a anotação dos marcos fiduciais para treinamento, validação e teste do modelo *YOLOv8*. Para a separação dos dados entre treinar, validar e testar a proporção do *dataset* utilizado foi de 70%/20%/10%, respectivamente.
- **Implementação e treinamento da *YOLOv8***: A *YOLOv8* foi utilizada para a detecção de marcos fiduciais nas imagens capturadas. O modelo foi treinado, em um notebook, com o dataset preparado, ajustando-o para identificar e localizar eficientemente os marcos fiduciais em diferentes condições de iluminação e ângulos de câmera. Após isso, foi integrado ao código. Tudo isso utilizando a biblioteca *ultralytics* de python Além disso, vale ressaltar que foram utilizadas 25 épocas para o treinamento da rede.
- **Integração do Filtro de Kalman**: Após a detecção dos marcos fiduciais, o Filtro de Kalman foi implementado, seguindo as equações explicadas anteriormente, para rastrear a posição e movimento dos marcos ao longo do tempo e fornecer estimativas da posição do objeto mesmo quando não havia detecção dele.

IV. Implementação

Para a implementação do sistema, foram feitos um código principail e dois códigos secundários. Um dos código secundário foi feito para definir as funções do filtro de Kalman (*Kalman.py*) e o outro para servir de detecção de objetos por meio da YOLO (*YOLOdetector.py*). Já o código principal executa a YOLOv8 em conjunto com câmera do aparelho utilizado (*wKF.py*), gerando as boundary boxes de acordo com as predições do filtro de Kalman.

A. Kalman.py

Nesse código foram definidas as classes do filtro de Kalman. Esse código importa a biblioteca *numpy* do *python* para realizar as operações matriciais necessárias. São definidas as seguints funções na classe *KalmanFilter*:

- *__init__*: Esse é o método construtor da classe, que inicializa uma instância do Filtro de Kalman
- *predict*: Esse método realiza a etapa de predição do Filtro de Kalman. Ele atualiza a estimativa do estado e a covariância do estado com base no modelo de processo.
- *filter*: Esse método é a etapa de correção ou atualização do filtro. Ele ajusta a estimativa do estado com base na nova medição.

Os returns dessa classe são feitos de modo a enviar as coordenadas x e y de centro das boundary boxes.

B. YOLOdetector.py

Esse arquivo funciona para fazer a detecção de objetos a partir da YOLO, usando a biblioteca *ultralytics* que é uma implementação da YOLO para detecção de objetos, além da *numpy*. São definidas as seguintes funções na classe *object_detector*:

- *__init__*: Este é o construtor da classe. Ele inicializa a instância com um modelo YOLO pré-treinado.
- *detect*: Este método é usado para detectar objetos em um quadro (frame) de vídeo ou imagem. Ele retorna uma lista com as coordenadas dos centros dos objetos detectados

C. wKF.py

Esse é o código que une as funcionalidades dos outros dois códigos citados anteriormente, ele utiliza a biblioteca *cv2* advinda do *opencv* para a inicialização e comunicação com a câmera do dispositivo eletrônico utilizado. Ao ser inicializado, o código abrirá uma aba com a imagem da webcam, sendo possível visualizar os objetos detectados pela *YOLO* no terminal, além de poder visualizar em tempo real as *boudanry boxes* da *YOLO* e do filtro de *Kalman*.

V. Resultados e Discussões

O código completo, junto de um vídeo de funcionamento, podem ser acessados em [3]. alguns resultados interessantes, retirados do notebook de treinamento e do vídeo, encontram-se a seguir:

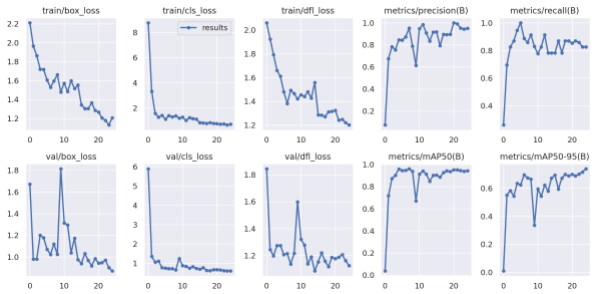

Fig. 1. Métricas de treinamento obtidas em função de épocas.

É possível perceber que as métricas de *train* e *validation* convergiram de forma consistente para o tamanho do *dataset* e o número de épocas utilizado.

Ademais, as bolas vermelha, azul e preta representam, respectivamente, deteccção feita pela *YOLO*, Passo de predição do filtro de *kalman* e passo de filtragem/atualização do filtro de *kalman*. Pode-se perceber que o resultado detectado puramente pela *YOLO* e pelo o filtro convergem bem e acredita-se que, caso a taxa de quadros de detecção fosse maior, os resultados seriam melhores ainda.

As próximas imagens mostram o resultado do passo de predição ocorrendo sozinho após perder a detecção de imagem pela *YOLO*.

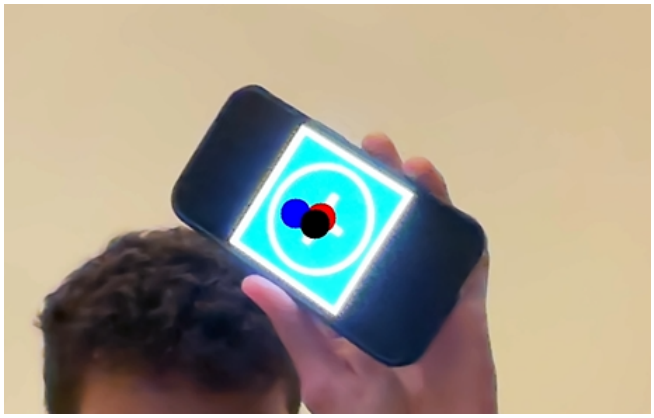

Fig. 2. Quadro de detecção do marco fiducial

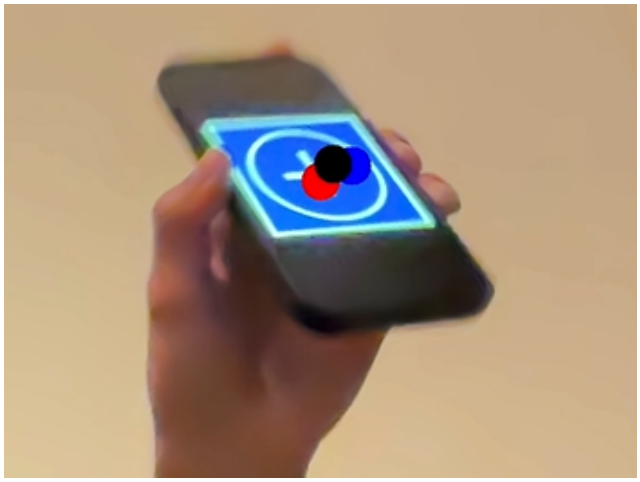

Fig. 3. Quadro de detecção logo antes da imagem sumir

Nesse caso, é interessante ressaltar que o círculo preto se mantém no lugar, pois não há mais detecção, enquanto o azul continua se movendo devido à predição do filtro. Esse resultado é esperado, pois necessita-se disso durante detecção de objetos, principalmente no caso de eVTOLs que precisam encontrar um caminho até o marco fiducial, pois é necessário produzir uma estimativa de posição para o marco quando não se tem mais detecção dele durante alguns quadros para que o sistema consiga alcança-lo de forma mais efetiva.

A seguir há a imagem da detecção ocorrendo em casos mais difíceis. Nesse caso, pode-se perceber que a *YOLO* percebe o objeto mesmo quando a imagem fica de difícil detecção, o que também é muito útil para o caso de eVTOLs que precisam seguir um caminho até o marco, pois é necessário que eles continuem realizando a detecção do objeto mesmo em casos de oclusão, baixa iluminação, entre outros.

VI. Conclusão

Foi possível concluir que *YOLO* apresentou bons resultados para detecção, mesmo com um *dataset* relativamente pequeno, conseguindo detectar o objeto em situações de ângulo e iluminação não tão favoráveis. Além disso, o filtro de *kalman* conseguiu convergir bem para o resultado da *YOLO* e conseguiu realizar o passo de predição de forma adequada à que se é esperada quando se perde o objeto.

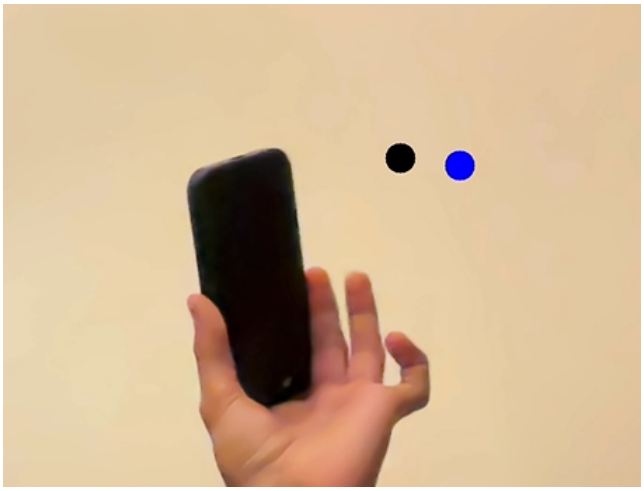

Fig. 4. Quadro de detecção logo após a imagem sumir

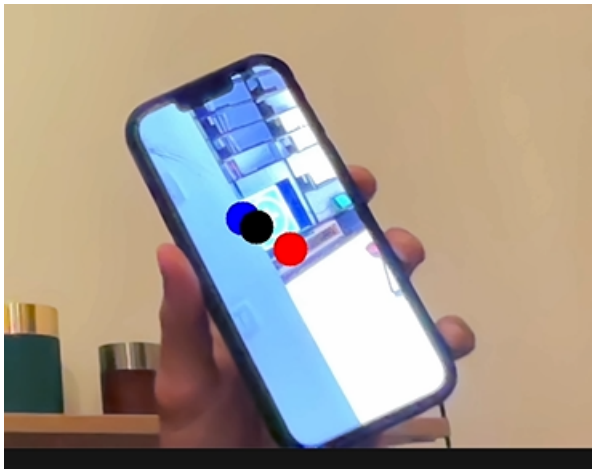

Fig. 5. Frame de detecção do marco fiducial a partir de uma imagem da imagem.

Em geral, acredita-se que os resultados obtidos serão úteis para utilizações reais em eVTOLs. Ademais, espera-se que com um *dataset* mais robusto e uma melhor otimização dos hiperparâmetros do filtro de *kalman* consiga-se encontrar resultados mais consistentes.

References

[1] Ultralytics. (2023). *Ultralytics GitHub Repository*. Disponível em: https://github.com/ultralytics/ultralytics. Acesso em: 14 de dezembro de 2023.

[2] P. Zarchan and H. Musoff, "Fundamentals of Kalman Filtering: A Practical Approach," 4th ed., American Institute of Aeronautics and Astronautics, Reston, VA, 2015, ISBN 978-1624102769.

[3] L. Peres. Exame CM203. Disponível em: https://github.com/leopers/Exame-CM203/tree/main. Acesso em: 14/12/2023.