

# PD6 - Rastreamento Visual

Frederico Guth (18/0081641)  
Tópicos em Sistemas de Computação, ,  
Turma TC - Visão Computacional (PPGI)  
Universidade de Brasília  
Brasília, Brasil  
fredguth@fredguth.com

**Resumo**—Rastreamento visual lida com a localização de objetos em vídeos e tem uma grande variedade de aplicações. Neste projeto, implementamos dois algoritmos para rastrear carros em vídeos de perseguição policial e avaliamos os resultados usando métricas de acurácia e robustez.

**Index Terms**—rastreamento de objetos, rastreamento visual, Filtro de Kalman, Rastreador KCF, perseguição policial de carros

## I. INTRODUÇÃO

Rastreamento visual lida com a localização de objetos em vídeos e tem uma grande variedade de aplicações: interação homem-computador, vigilância, edição de vídeo, controle de braços mecânicos, entre outras [1].

O rastreamento pode ser 2D, quando o objetivo é seguir a posição na imagem do objeto que se move quadro a quadro, ou 3D, quando se quer usar medidas nas imagens para ajustar com 6 graus de liberdade (3 de translação e 3 de rotação) a posição do objeto no espaço 3D.

A natureza do problema em que tanto a imagem do objeto alvo, quanto do ambiente onde se encontra, podem mudar ao longo do tempo e a necessidade de resultados em tempo real, que muitas das aplicações práticas necessitam, tornam esse um problema de grande interesse acadêmico.

### A. Objetivo

O objetivo deste projeto é rastrear carros em vídeos de perseguição policial e avaliar os resultados usando métricas de sobreposição média e robustez, especificadas em [2].

## II. REVISÃO TEÓRICA

### A. Rastreador KCF



		Storage	Bottleneck	Speed
Random Sampling ( $p$ random subwindows)		Features from $p$ subwindows	Learning algorithm (Struct. SVM [4], Boost [3, 6]...)	10 - 25 FPS
Dense Sampling (all subwindows, proposed method)		Features from one image	Fast Fourier Transform	320 FPS

Figura 1: Principais diferenças entre técnicas de rastreamento por detecção tradicionais versus KCF. Velocidade para regiões de 64x64 pixels [3].

Um dos grandes avanços recentes na pesquisa em rastreamento visual foi a adoção de métodos de aprendizado que

fazem "rastreamento por detecção". Dado uma amostra da imagem do alvo, o objetivo é que um modelo classificador aprenda a discriminar entre o alvo e o ambiente em uma fase de treinamento. A cada detecção, uma nova imagem do alvo pode ser usada para melhorar o modelo do classificador [3], [4].

Um desafio é o número praticamente ilimitado de possíveis amostras negativas que podem ser retiradas da imagem, que torna tentador focar na caracterização do objeto de interesse [4]. Isto é o que acontece na maioria dos algoritmos de rastreamento por detecção. Eles adotam uma estratégia de amostragem esparsa, onde cada amostra é uma *cropagem* da imagem com o tamanho do alvo, com alguma translação e/ou rotação a partir deste [3]. Como há muita sobreposição entre as amostras, os modelos contêm muita redundância (vide figura 1) e pouca amostragem de negativos.

A grande contribuição da pesquisa de Henriques *et al* foi constatar que o processo de *cropagem* de imagens levam a uma *matriz circulante*, que pode ser diagonalizada com uma transformada rápida de Fourier (FFT) e que incorpora as informações de todas as potenciais *cropagens*, o que chamam de amostragem densa, sem precisar itera-las com custosos processamentos de matrizes [3], [4].

A regressão linear da formulação que obtiveram é equivalente a um filtro de correlação, usado em diversos outros rastreadores. Mas para regressão por kernel, a formulação obtida, *Kernelized Correlation Filter* (KCF), diferentemente de outras soluções por kernel, tem a mesma complexidade das regressões lineares [4].

O rastreador KCF implementado na OpenCV [5] permite uma avaliação do mesmo em termos de acurácia e robustez. É importante registrar, no entanto, que é uma implementação simples com várias oportunidades de melhoria. Além de uma melhor documentação, a implementação não inclui detecção de falhas do modelo, nem incorpora modelos de incerteza, como por exemplo filtro de Kalman.

### B. Filtro de Kalman

O filtro de Kalman é um algoritmo que usa uma série de medidas observadas ao longo do tempo, contendo ruído estatístico e outras incertezas, e produz estimativas da grandeza medida que tendem a se aproximar mais dos valores reais do que aquelas realizadas a partir de apenas uma medida [1]. Essa técnica foi primeiramente publicada por Rudolf E. Kalman, em

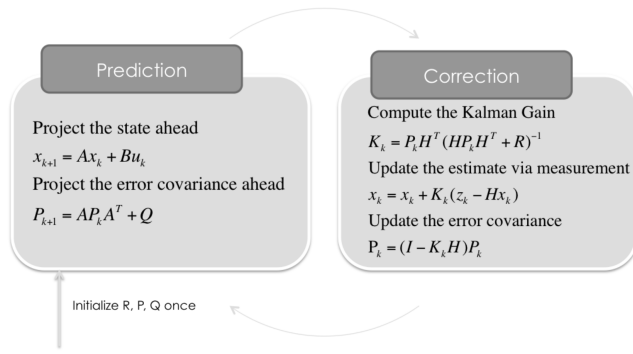


Figura 2: As duas etapas do filtro de Kalman [6]

1960 [7], e tem diversas aplicações práticas, principalmente em sistemas de navegação e controle de veículos aéreos e espaciais, tendo inclusive sido amplamente utilizada no projeto espacial Apollo que colocou o primeiro homem na Lua [8].

O filtro de Kalman tem duas etapas principais: a etapa de predição e a de correção (vide figura 2). A etapa de predição é responsável por estimar o estado futuro, a partir do estado corrente. A etapa de correção melhora as estimativas futuras, a partir de medidas reais do estado corrente.

### III. METODOLOGIA

#### A. Materiais

Foram utilizados:

- computador MacBook Pro (Retina, 13-inch, Early 2015), Processador Intel Core i5 2,7 GHz, 8GB de RAM
- Python 3.6.3 :: Anaconda custom (64-bit)
- OpenCV 3.4.0
- 5 programas python e 1 notebook jupyter especialmente desenvolvidos para o projeto. Todos estão disponíveis no repositório: [git@github.com:fredguth/unb-cv-3183](https://github.com/fredguth/unb-cv-3183).git

Todos os arquivos do projeto estão publicamente disponíveis em [git@github.com:fredguth/unb-cv-3183](https://github.com/fredguth/unb-cv-3183).git

#### B. Visão Geral

Foram implementados dois algoritmos:

- 1) rastreador KCF; e
- 2) rastreador KCF com filtro de Kalman.

#### C. Rastreador KCF

O rastreador KCF foi desenvolvido (r3.py) usando a implementação disponível na biblioteca OpenCV.

- 1) executamos o programa r3.py com 2 vídeos de perseguição;
- 2) em cada execução, guardamos as métricas obtidas;

#### D. Rastreador KCF com filtro de Kalman

Como o algoritmo KCF não inclui um detector de falhas, usamos o filtro de Kalman corrigindo-o com a estimativa do KCF. Com isso, apenas *amortecemos* o resultado do KCF, e melhoramos sua robustez, pois quando este perde o objeto rastreado, reinicializamos com a estimativa do filtro de Kalman.

Obviamente essa não é a melhor forma de aplicar o filtro de Kalman. O ideal seria desenvolver um detector de falhas para o KCF e ajustar o Kalman a partir deste. Na nossa implementação, *trocamos* acurácia por robustez, o que não seria absolutamente necessário em uma implementação com filtro de Kalman com detecção de falhas. De toda forma, acreditamos que o resultado deste experimento pode nos indicar o potencial benefício de uma nova implementação do KCF que incorpore esses itens.

Por último, é importante mencionar também que a implementação do filtro de Kalman na OpenCV, além de muito mal documentada, não permite inicializar o filtro com um estado inicial, sendo sempre inicializado com uma matriz de zeros, o que nos obriga sempre a somar ou subtrair o estado inicial quando se deseja utilizar as funções de predição e correção do filtro.

- 1) executamos o programa r4.py com 2 vídeos de perseguição;
- 2) em cada execução, guardamos as métricas obtidas;

#### E. Métricas

Para avaliar os rastreadores, usamos duas medidas sugeridas por [2].

- 1) sobreposição média:

$$\bar{\phi} = \sum_t \frac{\phi_t}{N}, \quad (1)$$

onde:

$$\phi_t = \frac{R_t^G \cap R_t^T}{R_t^G \cup R_t^T}, \quad (2)$$

sendo  $R_t^G$  a região no *ground-truth*,  $R_t^T$  a região alvo prevista pelo rastreador e  $N$  o número de quadros para os quais temos informação (*ground-truth*).

- 2) robustez:

$$R_S = e^{-S \frac{F_0}{N}}, \quad (3)$$

onde  $F_0$  é o número de falhas em  $N$  quadros e  $S$  é um parâmetro que pode ser interpretado como a probabilidade do rastreador continuar funcionando  $S$  quadros após a última falha [2]. No presente projeto, usamos  $S = 30$ .

#### F. Condições de reinicialização

Os rastreadores rodam enquanto  $\phi > 0$ . Quando a acurácia é zero, uma falha é contabilizada e o rastreador reinicializado com *ground-truth* do quadro.

### IV. RESULTADOS

Nesta seção apresentamos os resultados obtidos. Todos os dados podem ser acessados no repositório do projeto (III-A).

Quantitativamente, os resultados foram interessantes, como se pode ver na Tabela I.

Tabela I: Média da acurácia e robustez

	Video	$\bar{\phi}\%$	$\bar{R}\%$
KCF	1	84.91	56.85
	2	77.85	41.23
Kalman	1	83.19	92.61
	2	68.61	78.07

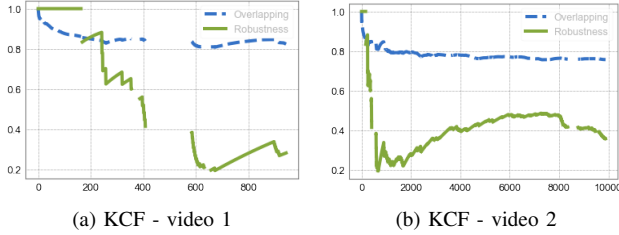


Figura 3: Acurácia e robustez do KCF ao longo do tempo

#### A. Rastreador KCF

Na figura 3 podemos ver o comportamento do rastreador KCF nos dois vídeos ao longo do tempo (quadros). As falhas nos gráficos apontam as regiões para as quais não temos os valores de *ground-truth*. É interessante notar que, nas duas instâncias, a acurácia logo cai para um patamar acima de 80% e não piora. A robustez varia bastante e chega a valores bem baixos em ambos os vídeos, próximo a 20%, mas melhoram até o final.

#### B. Rastreador Kalman

Na figura 4 podemos observar o efeito do filtro de Kalman. A acurácia cai bastante nos quadros com mudança de zoom, mas melhora ao longo do tempo. A robustez, entretanto, nunca cai abaixo de 60%.

#### C. Análise dos resultados

Os resultados obtidos foram dentro do esperado. Com filtro de Kalman, *trocamos* acurácia por robustez, entretanto vale ressaltar que no primeiro vídeo, o filtro de Kalman tem uma acurácia apenas 1,1% pior que o KCF, enquanto a robustez é 62,90% melhor. No vídeo 2, a acurácia com filtro de Kalman cai 11,87%, enquanto a robustez melhora 89,35%.

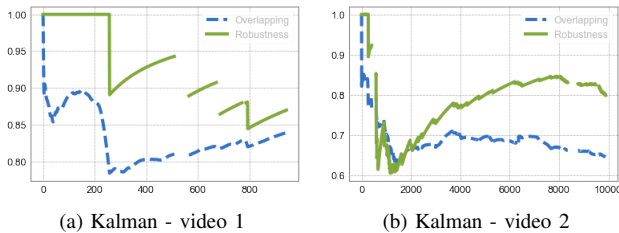


Figura 4: Acurácia e robustez do Filtro de Kalman ao longo do tempo.

Notamos que o *zoom* no vídeo é uma grande fonte de falhas, para minimizar esse problema, uma medida dessa grandeza poderia ser obtida, a partir da largura das faixas dos carros, e incorporada ao filtro de Kalman.

Como já mencionado II-A, o algoritmo KCF implementado na OpenCV não implementa detecção de falhas. Acreditamos que uma implementação que considere falhas e aplique nessa medida o filtro de Kalman pode ter resultados melhores.

## V. DISCUSSÃO E CONCLUSÕES

Neste trabalho, implementamos dois algoritmos para rastreamento visual de objetos em vídeo. O algoritmo KCF apresenta bons resultados de acurácia e robustez. Entretanto, mostramos que seu modelo pode melhorar se incorporar incerteza, o que pode ser feito com um filtro de Kalman. Ao *amortecer* o resultado do KCF com um filtro de Kalman, tivemos resultados de robustez entre 60 e 90% melhores, com perdas de acurácia menores que 12%, esse resultado serve de inspiração para melhorias na implementação do rastreador KCF da OpenCV.

## REFERÊNCIAS

- [1] Wikipedia contributors, “Kalman filter — Wikipedia, the free encyclopedia,” 2018, [Online; acessada 13 de Junho de 2018]. [Online]. Available: [https://en.wikipedia.org/wiki/Kalman\\_filter](https://en.wikipedia.org/wiki/Kalman_filter)
- [2] L. Cehovin, A. Leonardis, and M. Kristan, “Visual object tracking performance measures revisited,” *CoRR*, vol. abs/1502.05803, 2015.
- [3] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “Exploiting the circulant structure of tracking-by-detection with kernels,” in *Computer Vision – ECCV 2012*. Springer Berlin Heidelberg, 2012, pp. 702–715. [Online]. Available: [https://doi.org/10.1007/978-3-642-33765-9\\_50](https://doi.org/10.1007/978-3-642-33765-9_50)
- [4] —, “High-speed tracking with kernelized correlation filters,” *CoRR*, vol. abs/1404.7584, 2014.
- [5] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.
- [6] Balzer, “Kalman — some python implementations of the kalman filter,” 2018, [Online; acessada 13 de Junho de 2018]. [Online]. Available: <https://balzer82.github.io/Kalman/>
- [7] R. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME - Journal of basic Engineering*, vol. 82, pp. 35–45, 01 1960.
- [8] M. Grewal and A. Andrews, “Applications of kalman filtering in aerospace 1960 to the present [historical perspectives],” *IEEE Control Systems Magazine*, vol. 30, no. 3, pp. 69–78, jun 2010. [Online]. Available: <https://doi.org/10.1109/mcs.2010.936465>