UNIVERSIDADE DE SÃO PAULO – ESCOLA POLITÉCNICA

Aplicação em tempo real de *optical flow* utilizando algoritmo de Lucas-Kanade

Andre de O. Botelho*

*POLI-USP, São Paulo, Brazil

e-mail: andre.botelho@usp.br

Abstract – *Este artigo tem como objetivo apresentar aplicação em tempo real do clássico algoritmo de Lucas-Kanade em plataforma embarcadas atuais. Com isso pode ser mostrado que algorítmos clássicos podem ainda ser utilizados uma vez que o poder de processamento atual é muito superior ao da época.*

Palavras-chave: *Lucas-Kanade, optical flow, aplicação em tempo real*

Introdução

A area de visão computacional teve grandes avanços nas decadas de 70 e 80 em termos de eficientização de algoritmos. Em 1981 o algoritmo de *optical flow* proposto por Lucas-Kanade[1] foi um deles.

Este algoritmo foi extremamente inovador e provou-se mais eficiente que as soluções anteriores ([2],[3]). O algoritmo se tornou tão importante que uma série de cursos de graduação e pós pedem para que seus alunos implementem-o e, inclusive, já está disponível em uma série de bibliotecas para visão computacional, OpenCV[4] é um desses exemplos.

Neste artigo eu apresento uma aplicação em tempo real utilizando a plataforma Raspberry Pi e a biblioteca OpenCV (*Open Computer Vision),* linguagem Python para *tracking* de objetos utilizando o algoritmo de Lucas-Kanade [LK].

Com isso pretendo demonstrar que algoritmos clássicos podem ser utilizados mais facilmente hoje em dia e que sua aplicação ainda gera bons resultados e robustos

Materiais e métodos

Para processamento das imagens e apresentação de resultados utilizei a biblioteca OpenCV. Esta é uma das bibliotecas padrões mais usadas pela comunidade (não necessariamente científica) para processamento de imagens e visão computacional. Dentre os muitos algoritmos já implementados por *default* nesta biblioteca está o algoritmo de Lucas-Kanade.

Para o desenvolvimento do software utilizei o software *Thonny Python IDE* (V2.1.16) na plataforma RaspberryPi 3. Esta plataforma conta com 4 CPUs RM Cortex-A53 com clock de 1.2GHz. É uma plataforma potente para seu tamanho, porém bem aquém da tecnologia de ponta disponível hoje em dia.

Esta plataforma foi escolhida por também ser bastante utilizada globalmente e demonstra a facilidade de aplicar algoritmos clássicos em plataformas simples em tempo real.

Foi utilizado um vídeo artificial de uma circunferência branca em fundo preto para analisar a eficiência do *script* e, por fim, um video gravado e processado em tempo real pela própria RaspberryPi.

Em um caso de translação 2D o algoritmo LK quer estimar um vetor h = (u,v) de deslocamento entre duas imagens, como visto abaixo:

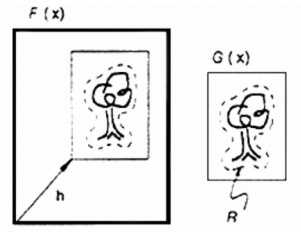

Figura 1: Vetor de Deslocamento h

Este vetor é obtido à partir das suas duas derivadas espaciais (aqui chamadas de f_x e f_y) e a derivada temporal (f_t) segundo a equação:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{(-1)} * \begin{bmatrix} -\sum_i f_{t_i} f_{x_i} \\ -\sum_i f_{t_i} f_{y_i} \end{bmatrix}$$

Esta formula, apesar de parecer simples, é o que demonsta uma robustes no algoritmo. Apesar de ter a inversão de uma matriz nos calculos ela é facilmente calculada por se tratar de uma matrix 2x2.

Este artigo trata de aplicação em tempo real deste algoritmo e, portanto, algumas restrições tem que ser consideradas. A primeira dela é que não necessariamente todos os pontos são interessantes para *tracking*. Segundo que objetos em uma imagem são delimitados por “cantos”, detectar cantos de um objeto sólido e calcular o movimento destes representa o movimento do sólido como um todo. Terceiros: o movimento entre frames pode ser maior que um pixel, algum método deve ser utilizado para tratar esses casos.

A partir da suposição 2 podemos resolver o primeiro problema utilizando o algoritmo proposto por Shi-Tomasi [5]. Neste artigo os autores demonstram a razão e como escolher bons pontos em uma imagem para *tracking*. Esta função está implementada, também, na biblioteca OpenCV.

Uma vez que temos um numero menor de pontos para processar podemos considerar agora o terceiro problema, movimentos que são maiores que um pixel. Para este problema foi utilizado o recurso de pirâmides de uma image. Essa técnica consiste em pegar a imagem original, diminuí-la L vezes, calcular o *optical flow* nestes níveis e ir subindo, sempre levando em conta os níveis mais baixos. Desta maneira, movimentos que são maiores que um pixel no tamanho original podem passar a ser sub-pixel em um dos níveis e, com isso, o algoritmo de LK consegue contabilizá-los. Uma figura que exemplifica isto pode ser visto na Figura 2, uma outra aplicação do mesmo método pode ser visto em [7].

O algoritmo descrito foi implementado em Python utilizando, principalmente, a biblioteca OpenCV e foi baseado em um de seus exemplos [6]. Foi utilizada a camera da Raspberry pi para captura das imagens no video ao vivo, as imagens foram armazenadas no disco interno e o tempo entre elas para estatísticas foram imprimidos na tela enquanto o *script* rodava. Durante o processamento foram traçados os movimentos recentes dos pontos de interesse encontrados e o resultado mostrado em tempo real.

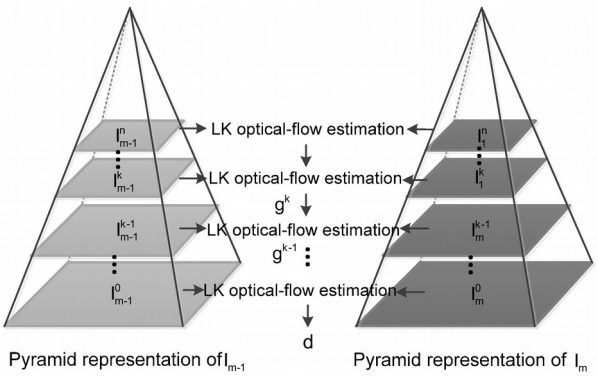

Figura 2: Exemplo do algoritmo de Piramide

Resultados

O primeiro teste foi feito com um vídeo de um objeto branco em movimento pseudo-aleatório em um fundo preto. Exemplos de *frames* do vídeo pode ser visto na Figura 3, o vídeo na íntegra pode ser visto através do link: https://www.youtube.com/watch?v=LmcZc8DPPLY . O resultado do tracking pode ser visto em https://www.youtube.com/watch?v=f2MOtBxYBKo, exemplos de *frames* visto na Figura 4.

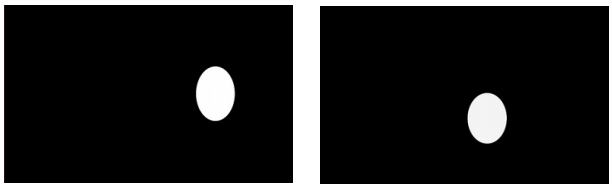

Figura 3: Exemplo de video processado

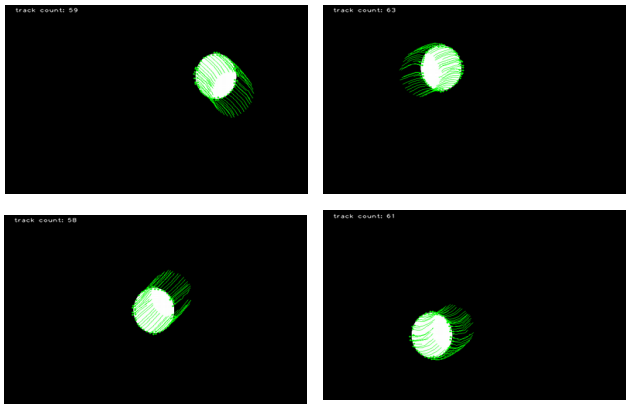

Figura 4: Video processado e *tracking*

Como podemos ver na no video o *tracking* é feito com sucesso e o algoritmo funciona com precisão. Analisando o log, Figura 5, vemos que o *framerate* (quantidade de frames processado por segundo) não é muito satisfatório, cerca de 2 fps (*frames per second*). Isso se deve ao fato de que as imagens originais são muito grandes e muito

pesadas para o hardware em questão, o que faz com que o hardware demore muito para ler as imagens do disco.

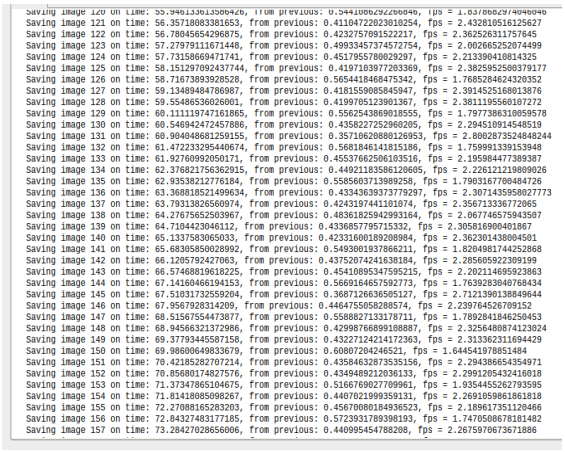

Figura 5: Log do processamento do video

Uma vez processado o video começamos a analisar o processamento do video em tempo real. Pequenas alterações no código tiveram que ser feitas para obtenção da imagem, agora as imagens já são adquiridas no formato (640,480) e, com isso, o processamento fica mais rapido. Na Figura 6 vemos exemplo de imagens processadas e, na Figura 7, o log. Vemo que o *fps* fica próximo de 6 a 7 frames por segundo, com picos de 10. Tal *fps* não é alto para muitas aplicações, porém para algumas, como *tracking* de pessoas e objetos por camera de segurança, é melhor do que muitas marcas no mercado. O vídeo pode ser visto em https://www.youtube.com/watch?v=7l_sMI177ag

Figura 6: Video processado em tempo real

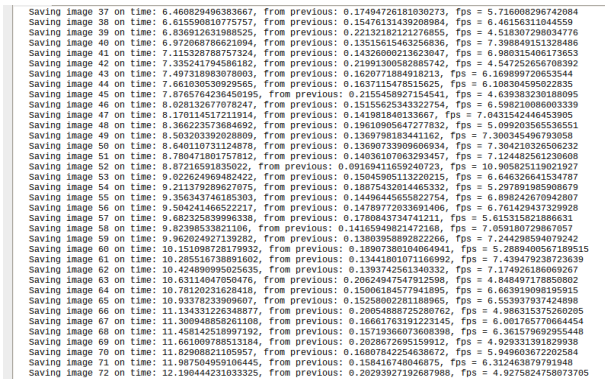

Figura 7: Log do video processado em tempo real

Conclusões

Este artigo teve como objetivo demonstrar o uso do algoritmo de Lucas-Kanade em uma plataforma atual com o intuito de demonstrar que algoritmos tidos como clássicos podem ainda ser usados e consegue performar em plataformas atuais.

A aplicação em tempo real desejada foi de *tracking*, onde pontos de interesse tido como interessantes foram encontrados e seus movimentos estimados em tempo real.

Foi utilizada a biblioteca do OpenCV e a plataforma utilizada foi a Raspberry Pi, ambos são muito utilizados hoje em dia e possuem uma grande comunidade apoiando-os.

O resultado foi mostrado em tempo real, gravados em um vídeo e o log durante o processamento foi capturado. O vídeo em tempo real, cujs imagens já são capturadas em tamanho apropriado, possuíram tempo de processamento compatível com a aplicação desejada. Desta maneira conseguiu-se demonstrar com sucesso a implementação deste algoritmo em plataformas atuais e em tempo real.

Todos os códigos e imagens estão disponíveis no github.com/aobotelho/PTC5750

Referências

[1] B.D. Lucas and T. Kanade, “An Iterative Image Registration Technique with an Application to Stereo Vision”, DARPA Image Understanding Workshop, 1981, pp121–130 (see also IJCAI’81, pp674– 679).(disponível em: https://cseweb.ucsd.edu/classes/sp02/cse252/lucaskanade81.pdf)

[2] Barnea, Daniel I. and Silverman, Harvey F. "A Class of Algorithms for Fast Digital Image

Registration." IEEE Transactions on Computers C-21.2 (February 1972), 179- 186.
[3] Moravec, Hans. P. Visual Mapping by a Robot Rover. Sixth International Joint Conference on Artificial Intelligence, Tokyo, August, 1979, pp. 598-600
[4] https://opencv.org/
[5] J. Shi, C. Tomasi, "*Good Features to Track*", IEEE Conference in Computer Vision and Pattern Recognition, Seattle, June 1994
[6]https://docs.opencv.org/3.3.1/d7/d8b/tutorial_py_lucas_kanade.html
[7] http://www.mdpi.com/2072-666X/6/4/487/htm