USO DO FLUXO ÓPTICO PARA CRIAÇÃO DO HYPERLAPSE

Andressa Stéfany Silva de Oliveira*

*Graduanda em Engenharia de Computação
Universidade do Rio Grande do Norte
Natal, RN, Brasil

Email: astefanysoliveira@gmail.com

Resumo— A partir do estudo de Processamento Digital de Imagens, houve utilização das técnicas de Fluxo Óptico para a estabilização de um vídeo e a aceleração deste, onde o objetivo é a criação do efeito hyperlapse em um vídeo, isto é, a ideia de "colapso de tempo". Esses conceitos foram aplicados com a linguagem C++ utilizando a biblioteca OpenCV.

Palavras-chave— Hyperlapse, Processamento de Imagem, Vídeo, Estabilização, Shi-Tomasi, Lucas-Kanade, Transformação Afim, OpenCV.

1 Introdução

Um dos campos do Processamento Digital de Imagens é o Fluxo Óptico, o qual estuda o deslocamento dos pixels de uma sequência de imagens.

Inúmeros estudos e técnicas tem surgido nesse ramo, como também as aplicações, podendo ser citadas: navegação de robôs móveis, nesse caso o fluxo óptico atuará no rastreio de objetos fazendo com quê o robô não colida em obstáculos presentes no ambiente em que se encontra, outro exemplo é a estabilização de vídeos, como também a recuperação de formas a partir do movimento e compressão de imagens.

Neste trabalho será citado algumas técnicas de detecção de pontos de borda, além de explicar um pouco mais a fundo o funcionamento de dois métodos em específico, os quais foram utilizados na aplicação desenvolvida: hyperlapse.

Como também será explicado todo o código e a estratégia utilizada. Posteriormente, é exposto os resultados obtidos.

1.1 Fluxo Óptico

O Fluxo Óptico proporciona o rastreamento do pixel de uma determinada imagem com relação a outra imagem pré-conhecida, se faz necessário que tenha no mínimo duas imagens para que isso seja possível, em alguns algoritmos, pois há algoritmos que utilizam mais frames no cálculo. Esse movimento poderá surgir a partir da movimentação da câmera ou dos objetos presentes na cena, ademais, supõe-se que o brilho da imagem sofreu uma variância quase imperceptível, ao ponto de não interferir nos algoritmos de rastreamento, observe:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t) \quad (1)$$

sendo I a intensidade do pixel, note que, se for pequeno o deslocamento de um frame para outro, Δx e Δy serão pequenos em um curto espaço de tempo (Δt), logo o brilho do pixel também não mudará drásticamente (FARIA, n.d.). Outro ponto importante é a vizinha do pixel ter sofrido deslocamento similar.

Muitos métodos já foram desenvolvidos para o cálculo do fluxo óptico, podendo ser citados: os agortimos de Horn e Schunck (1981), Lucas e Kanade (1981), Fleet e Jepson (1990), Harris-Shi-Tomasi(1994), Nesi et al. (1995), Grossmann e Santos-Victor (1997) e Lai e Vemuri (1998).

Foi utilizado os métodos de Shi-Tomasi e Lucas Kanade no presente trabalho com o intuito da obtenção dos deslocamentos em x, em y e do ângulo de rotação dos frames de um vídeo e, desse modo, produzir o efeito de hyperlapse no vídeo, através da estabilização dos frames e o aumento de velocidade na exibição de frames por segundo. Esses métodos foram escolhidos pelo fato de poderem ser executados em tempo hábil para uma aplicação de tempo real, diferentemente do algoritmo de Fleet e Jepson, que possuem um custo computacional alto pois é usado muitos quadros simultaneamente (SARCINELLI and CALDEIRA, 2001).

Também é importante informar que o problema de abertura é muito comum nesse processo de rastreamento dos pontos, isso ocorre quando o observador não consegue identificar todos os movimentos presentes na imagem, por exemplo, quando um objeto da imagem está se movendo diagonalmente e o observador só consegue identificar o movimento horizontal. Isso é decorrente de uma janela de pequena abertura para a detecção do deslocamento (BRITO, 2016).

Com relação ao hyperlapse, ele é uma técnica da fotografia que mistura o timelapse com a movimentação da câmera, onde o timelapse é o vídeo acelerado dando o efeito de "lapso de tempo"e a movimentação da câmera utiliza a técnica de varredura de imagem dispondo de um foco na imagem (BEZERRA, 2017).

1.2 Metodologia

Resumidamente, a estratégia utilizada foi capturar um frame, obter os pontos de bordas fortes através do método de Shi-Tomasi e, após a obtenção desses dados, o frame seguinte é capturado para o rastreamento dos pontos anteriormente obtidos, nessa fase é utilizado o método de Lucas Kanade. Agora, sabe-se a localização dos pontos em dois frames distintos, logo, é possível saber o deslocamento em x, y e o ângulo da rotação do segundo frame tendo como referência o primeiro frame. E então, o deslocamento e rotação inversos são aplicado à imagem com o intuito da estabilização do vídeo.

O trabalho foi desenvolvido utilizando a linguagem C++ e a biblioteca OpenCV. As funções mais importante do algoritmo são: goodFeaturesToTrack(), calcOpticalFlowPyrLK(), estimateRigidTransform() e warpAffine(). Abaixo está descrito o que cada função faz e o método por trás dela.

- goodFeaturesToTrack()

Essa função utiliza o método de Shi-Tomasi, que por sua vez teve como base o método de Harris, o qual busca variação no gradiente da imagem. Primeiro, é preciso calcular a intensidade do pixel, o qual é obtido pela seguinte equação:

$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$

Usando expansão de Taylor,

$$E(u,v) = \sum x,y[I(x,y)+uI_x+vI_y-I(x,y)]^2,$$

expandindo e manipulando se chega a

$$E(u,v) = \sum_{x,y} u^2I_x^2 + 2uvI_xI_y + v^2I_y^2,$$

expressando em matriz temos:

$$E(u,v) = \begin{bmatrix} u & v \end{bmatrix} (\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}).$$

Criando uma variável M:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix},$$

a expressão fica:

$$E(u,v) = \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}.$$

A matriz M é usada no cálculo para determinar se uma janela contém possivelmente um canto, esse *score* é calculado através da seguinte equação:

$$R = min(\lambda_1, \lambda_2),$$

onde os λ são os autovalores da matriz M. Esse valor precisa ser maior que um limiar, um λ_{min} (OpenCV[2], 2017). Observe a figura abaixo:

A janela só será considerada um canto quando λ_1 e λ_2 estiverem acima do λ_{min}, ou seja, quando estiver na região verde mostrada na figura 1.

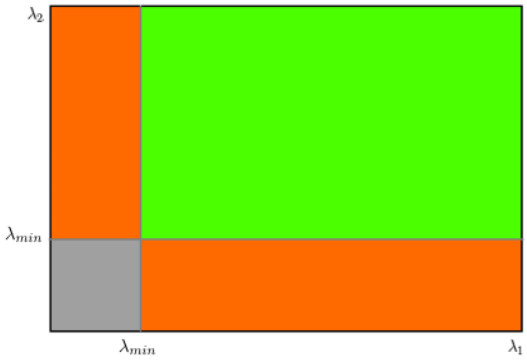


Figura 1: Método Shi-Tomasi

- calcOpticalFlowPyrLK()

Calcula o fluxo óptico para um conjunto de características esparsas usando o método não-iterativo Lucas Kanade, o qual assume que o fluxo é constante em pequenas janelas de tamanho m x m e com $m > 1$ (MARTINS, n.d.). Numerando os pixels de 1 a n chega-se no seguinte conjunto de equações

$$\begin{gathered} I_{x1}V_x + I_{y1}V_y = -I_{t1} \\ I_{x2}V_x + I_{y2}V_y = -I_{t2} \\ \cdots \\ I_{xn}V_x + I_{yn}V_y = -I_{tn} \end{gathered}$$

Notação matricial:

$$\begin{bmatrix} I_{x1} & I_{y1} & I_{z1} \\ I_{x2} & I_{y2} & I_{z2} \\ & \cdots & \\ I_{xn} & I_{yn} & I_{zn} \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \cdots \\ -I_{tn} \end{bmatrix}$$

$$\Rightarrow A\vec{v} = -b$$

Utilizando o método dos mínimos quadrados para a resolução desse sistema ficamos com

$$A^TA\vec{v} = A^T(-b) \text{ ou } \vec{v} = (A^TA)A^T(-b)$$

E então obtemos

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum I_{xi}^2 & \sum I_{xi}I_{yi} \\ \sum I_{xi}I_{yi} & \sum I_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_{xi}I_{ti} \\ \sum I_{yi}I_{ti} \end{bmatrix}$$

Um exemplo visual é a figura 2, é passado como parâmetro a imagem 1, imagem 2 e as bordas da imagem 1, o retorno será as bordas da imagem 2.

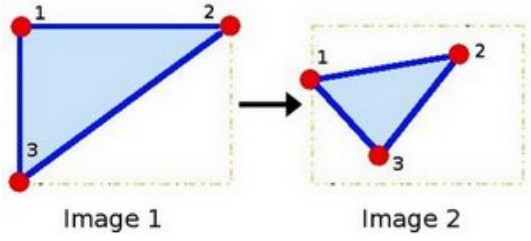


Figura 2: Lucas Kanade

- estimateRigidTransform()

Sendo pré-estabelecido dois vetores de coordenadas de pontos ótimos de borda, essa função retornará uma transformação afim, isto é, retornará o deslocamento em x, y e o ângulo θ, como mostrado nas matrizes abaixo (NghiaHo[1], 2015):

$$\text{R(rotação)}= \begin{bmatrix} cos(\theta) & -\sin(theta) \\ \sin(\theta) & cos(\theta) \end{bmatrix}$$

$$\text{S(escalonamento)}= \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

$$\text{t(translação)}= \begin{bmatrix} tx \\ ty \end{bmatrix}$$

A transformação afim fica

$$T = [RS|t] = \begin{bmatrix} s*cos(\theta) & -s*sin(\theta) & tx \\ s*sin(\theta) & s*cos(\theta) & ty \end{bmatrix}$$

- warpAffine() - (OpenCV[3], 2017)

Basicamente, essa função irá aplicar uma matriz de transformação afim a uma imagem, onde essa transformação afim possuirá deslocamento em x, em y, o escalonamento e a rotação, como mencionado anteriormente. Essa moficação é realizada na imagem através da seguinte equação:

$$\texttt{dst}(x,y) = \texttt{src}(\texttt{M}_{11}x + \texttt{M}_{12}y + \texttt{M}_{13}, \texttt{M}_{21}x + \texttt{M}_{22}y + \texttt{M}_{23})$$

Também é de grande relevância citar que a estabilização do vídeo é mostrada em tempo real para o usuário, pois assim que o tratamento é feito no frame, ele é exibido. Diferentemente da aceleração, que é feita apenas no vídeo resultante. Esse funcionamento será explicado na próxima seção.

1.3 Agoritmo

Nesta seção será mostrado a estrutura do algoritmo desenvolvido.

Primeiro, há a captura de um frame, chamado de *prev_image*, e sua conversão para tons de cinza. Para efeito de visibilidade final do usuário, *prev_image* foi clonado antes da transformação da cor, para que a imagem colorida não seja perdida.

```
1 VideoCapture cap("media/video.mp4");
2 cap>>prev_image;
3
4 prev_color_image = prev_image.clone();
5
6 cvtColor(prev_image, prev_image,
      COLOR_BGR2GRAY);
```

Listing 1: Captura do Primeiro Frame

Em seguida, o código entra em um loop para o tratamento dos seguintes frames. Calcula-se os pontos de borda forte de *prev_image* (*goodFeaturesToTrack*), o próximo frame é capturado e calculado os correspondentes dos pontos encontrados em *prev_image* (*calcOpticalFlowPyrLK*). Assim como no primeiro frame, o segundo também é clonado criando a nova imagem *next_color_image*.

```
1  cap>>next_image;
2
3  goodFeaturesToTrack(prev_image, corners
       [0], 300, 0.01, 20, mask, 3, false,
       0.01);
4  calcOpticalFlowPyrLK(prev_image,
       next_image, corners[0], corners[1],
       status, err, winSize, 3, termcrit, 0,
        0.001);
5
6  for(size_t i=0; i < status.size(); i++) {
7    if(status[i]) { // tira as bordas ruins
8        greatCorners[0].push_back(corners
             [0][i]);
9        greatCorners[1].push_back(corners
             [1][i]);
10   }// fim do if
11 }// fim do for
```

Listing 2: Início do Loop

Observe que existe um loop percorrendo *status* (linha 6), isso se dá porque a função *calcOpticalFlowPyrLK* retorna *1* nessa variável caso o ponto de borda tenha sido encontrado em *next_image*, a função desse for é armazenar apenas os pontos que possuiem correspondentes.

Agora que temos as posições dos pontos de ambos os frames capturados, será usado *estimateRigidTransform* para encontrar a transformação afim. Pode ocorrer dessa matriz não se encontrada, nesse caso ela é substituída por uma matrix identidade. Além disso, a função *estimateRigidTransform* encontra o deslocamento e a rotação que ocorre do primeiro pro segundo argumento passado, então, pensando na transformação inversa, foi passado os pontos do segundo frame como primeiro argumento, e os pontos do primeiro frame como segundo argumento, perceba:

```
1  Mat T = estimateRigidTransform(
       great_corners[1], great_corners[0],
       false);
2
3  if(T.data==NULL){
4    // identidade + coluna de zeros
5    T = Mat(Size(3,2),CV_32FC1, Scalar(0));
6    T(Range(0,2),Range(0,2))= Mat::eye(2,2,
         CV_32FC1);
7  } else {
8    // soma dos deslocamentos e angulo
9    // pois o deslocamento eh com relacao
         ao frame anterior
10   soma_dx += T.at<double>(0,2);
11   soma_dy += T.at<double>(1,2);
12   soma_da += atan2(T.at<double>(1,0), T.
         at<double>(0,0));
13 }
14 // atualiza prev_image
15 prev_image = next_image.clone();
```

Listing 3: Transformação Afim

Caso a matriz de transformação seja encontrada, os valores de deslocamento e rotação estão sendo somados a suas respectivas variáveis que estão acumulando o seu valor ao decorrer dos loops, pois esse deslocamento e rotação que está sendo calculado na matriz T tem como referência o frame anterior.

```
// atualiza deslocamentos e angulo
dx= soma_dx;
dy= soma_dy;
da= soma_da;

// recalcula a matriz T
T.at<double>(0,0) = cos(da);
T.at<double>(0,1) = -sin(da);
T.at<double>(1,0) = sin(da);
T.at<double>(1,1) = cos(da);

T.at<double>(0,2) = dx;
T.at<double>(1,2) = dy;

// aplica a rotacao e translado
warpAffine(next_color_image, image_new, T
    , next_color_image.size());
```

Listing 4: Matriz T e Aplicação

O trecho acima mostra o cálculo da matriz de transformação com os novos valores acumulados e o uso da função *warpAffine*, a qual modifica a imagem *next_color_image*, onde o resultado será retornado em *new_image*. Ademais, *new_image* é recortada com o objetivo de amenizar o efeito que a estabilização causa nas bordas, além de redimensionada com o tamanho inicial do vídeo.

```
// sem estabilizacao
imshow("Sem", image_color);
// com estabilizacao
imshow("Com", image_new);

// limpando
greatCorners[0].clear();
greatCorners[1].clear();
corners[0].clear();
corners[1].clear();
```

Listing 5: Final

Por fim, o frame resultante é mostrado em uma janela e gravado em um vídeo, mais a necessidade de limpar os vetores das posições das bordas. Segue as especificações do vídeo, as quais são colocadas no início do código:

```
VideoWriter video2("final_video.avi",
    CV_FOURCC('M','J','P','G'),60,
    prev_image.size());
```

Listing 6: Vídeo Acelerado

Sendo esse vídeo exibido com 60 frames por segundo (NghiaHo[2], 2014).

1.4 Resultados

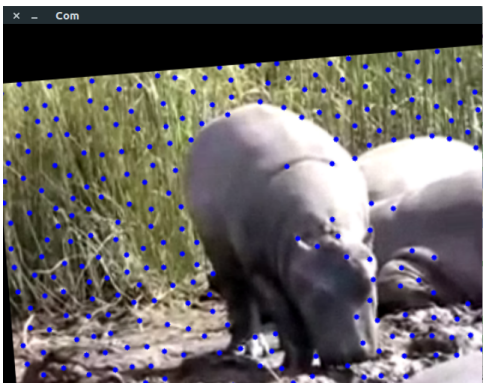
Para exibição dos resultados, foi utilizado o mecanismo de pintar os pontos encontrados em ambos os frames.

(a) Sem Estabilização

(b) Com Estabilização

Figura 3: Resultado 01

(a) Sem Estabilização



(b) Com Estabilização

Figura 4: Resultado 02

As figuras 3 e 4 mostram duas situação em que a aplicação da transformação afim é bem visível, pois é bastante notório a sobra preta na borda das imagem estabilizadas devido ao deslocamento e rotação.

1.5 Dificuldades

Pensar em todo os procedimentos e métodos que poderiam ser utilizados nos frames para tratamento da estabilização foi uma das dificuldades, porém, após algumas pesquisas foi possível entrar em um consenso no que poderia ser usado.

Houveram alguns testes, por exemplo, a aplicação de outras funções para se chegar ao valor do deslocamento e rotação da imagem, podendo ser citadas a função *findHomography()* que encontra a transformação de perspectiva entre dois planos, e *warpPerspective()* que aplicaria a transformada encontrada com *findHomography()*. Mas as funções presentes no algoritmo foram as que responderam melhor à intenção do trabalho.

Porém, o probema da movimentação no eixo z não foi solucionado, isso ocorre quando a câmera se desloca em direção ao objeto principal da cena ou se distancia. Ao rodar o algoritmo proposto nesse projeto, o resultado do tratamento tem um comportamento inesperado o qual não atende às espectativas.

1.6 Conclusão

O algoritmo atendeu bem ao objetivo do trabalho, que no caso é a estabilização e aceleração na visualização dos frames, porém, apenas para um certo tipo de vídeo, o qual a câmera não se movimenta no eixo z, ou seja, não se aproxima e não se distancia do foco da imagem. Posteriormente, esse problema pode ser consertado com um estudo mais aprofundado do fluxo óptico.

Referências

BEZERRA, R. (2017). Hyperlapse você sabe o que é?, http://entrananet.blogspot.com.br/2013/05/hyperlapse-voce-sabe-o-que-e.html. [Online; acessado 30 Nov 2017].

BRITO, A. (2016). Processamento Digital de Imagens - Fluxo Óptico, http://agostinhobritojr.github.io/cursos/pdi/fluxo.pdf. [Online; acessado 22 Oct 2017].

FARIA, A. W. C. (n.d.). Fluxo Óotico, http://homes.dcc.ufba.br/~perfeuge/optical_flow_article.pdf. [Online; acessado 30 Nov 2017].

MARTINS, L. (n.d.). Trabalho Final de Visão Computacional e Realidade Aumentada. Tema: Determinação do Fluxo Óptico em Dados Sísmicos, http://webserver2.tecgraf.puc-rio.br/~mgattass/ra/trb08/LeonardoMartins/trab_final.html#_Método_de_Lucas-Kanade. [Online; acessado 24 Oct 2017].

NghiaHo[1] (2015). Understanding OpenCV CV::estimateRigidTransform, http://nghiaho.com/?p=2208. [Online; acessado 22 Oct 2017].

NghiaHo[2] (2014). Simple Video Stabilization Using OpenCV, http://nghiaho.com/?p=2093. [Online; acessado 22 Oct 2017].

OpenCV[2] (2017). Harris corner detector, https://docs.opencv.org/2.4/doc/tutorials/features2d/trackingmotion/harris_detector/harris_detector.html. [Online; acessado 22 Oct 2017].

OpenCV[3] (2017). Affine Transformations, https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html. [Online; acessado 22 Oct 2017].

SARCINELLI, Mário Filho; SCHNEEBELI, H. A. and CALDEIRA, E. M. d. O. (2001). Cálculo do Fluxo Óotico em Tempo Real e sua Utilização na Navegação de Robôs Móveis, http://fei.edu.br/sbai/SBAI2001/vsbai/artigos/1110.pdf. [Online; acessado 30 Nov 2017].