Visão computacional - 2020/2

Relatório PS4

Tiago Araújo Mendonça

[Código fonte]

Em todas as questões (exceto a 3) é opcional passar um parâmetro na linha de comando para especificar a imagem ou vídeo utilizado, se não for usado, uma imagem padrão é utilizada.

Questão 1

Nesta questão, são usados vídeos como parâmetro de entrada. O algoritmo captura os dois primeiros quadros do vídeo e utiliza-os para obter o *optical flow*.

A implementação foi baseada no pseudocódigo do algoritmo original de Horn-Schunk abaixo, com pequenas modificações.

1: **for** $y = 1$ to N_{rows} **do**
2: **for** $x = 1$ to N_{cols} **do**
3: Compute $I_x(x, y)$, $I_y(x, y)$, and $I_t(x, y)$;
4: Initialize $u(x, y)$ and $v(x, y)$ (in even arrays);
5: **end for**
6: **end for**
7: Select weight factor λ; select $T > 1$; set $n = 1$;
8: **while** $n \leq T$ **do**
9: **for** $y = 1$ to N_{rows} **do**
10: **for** $x = 1$ to N_{cols} {in alternation for even or odd arrays} **do**
11: Compute $\alpha(x, y, n)$;
12: Compute $u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t)$;
13: Compute $v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t)$;
14: **end for**
15: **end for**
16: $n := n + 1$;
17: **end while**

O parâmetro T representa o número de iterações no algoritmo, esse parâmetro foi variado e por final seu valor foi definido como 7. O parâmetro λ que define o impacto da suavidade foi utilizado como 0.1 como sugerido no livro da disciplina.

Algoritmo:

O primeiro passo é obter as matrizes chamadas de fx, fy e ft. Estas matrizes indicam para cada localização (x, y) existente nas imagens t e (t+1) o valor de Ix, Iy e It, estes valores relacionam a imagem t com a imagem (t+1). Armazenar os valores de Ix, Iy e It em matrizes

no início do algoritmo permite otimizar o código, pois esses valores não precisam ser calculados novamente.

Os parâmetros Ix, It, It representam resepectivamente as variações descritas no livro:

$$\cdot \frac{\partial I}{\partial x}(x, y, t) \quad \frac{\partial I}{\partial y}(x, y, t) \quad \frac{\partial I}{\partial t}(x, y, t)$$

O cálculo desses parâmetros foi implementado de duas maneiras diferentes:

1. Utilizando a aproximação originalmente usada no algoritmo de Horn S.:

$$\begin{aligned} I_x(x, y, t) = \frac{1}{4}\big[&I(x+1, y, t) + I(x+1, y, t+1) \\ &+ I(x+1, y+1, t) + I(x+1, y+1, t+1)\big] \\ &- \frac{1}{4}\big[I(x, y, t) + I(x, y, t+1) + I(x, y+1, t) + I(x, y+1, t+1)\big] \end{aligned} \tag{4.24}$$

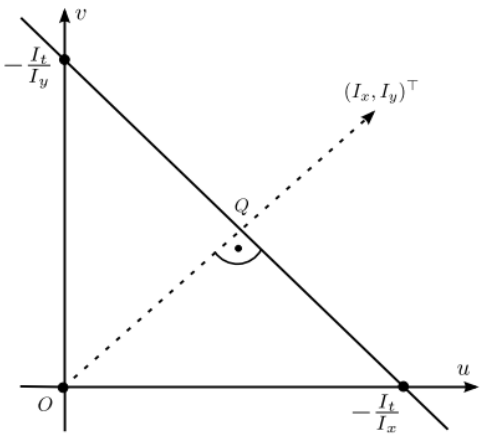
$$\begin{aligned} I_y(x, y, t) = \frac{1}{4}\big[&I(x, y+1, t) + I(x, y+1, t+1) \\ &+ I(x+1, y+1, t) + I(x+1, y+1, t+1)\big] \\ &- \frac{1}{4}\big[I(x, y, t) + I(x, y, t+1) + I(x+1, y, t) + I(x+1, y, t+1)\big] \end{aligned} \tag{4.25}$$

$$\begin{aligned} I_t(x, y, t) = \frac{1}{4}\big[&I(x, y, t+1) + I(x, y+1, t+1) \\ &+ I(x+1, y, t+1) + I(x+1, y+1, t+1)\big] \\ &- \frac{1}{4}\big[I(x, y, t) + I(x, y+1, t) + I(x+1, y, t) + I(x+1, y+1, t)\big] \end{aligned} \tag{4.26}$$

2. Utilizando a aproximação discrete de Sobel, onde:
   Ix = Sx
   Iy = Sy
   It = Imagem_{t-1} - Imagem_t

Após calcular as matrizes fx, fy e ft, o passo seguinte inicializa as matrizes u e v (iteração n=0). Foram implementadas duas formas de inicializar estas matrizes:

1. Inicializar ambas com todos os valores u_{xy} e v_{xy} como 0
2. Obter o ponto Q no plano uv que é o ponto mais próximo da origem considerando a reta representada por $-\bar{I}_t = u \cdot I_x + v \cdot \bar{I}_y$, conforme a imagem abaixo:



O cálculo de Q foi realizado pela projeção do ponto (0,0) na reta

Após estas inicializações, o algoritmo inicia suas iterações de n=1 até n=T. Para cada iteração n, em todas as localizações (x, y) são (re)calculadas as médias $\bar{u}$(x,y) e $\bar{v}$(x,y) para a adjacência de 4 pixels desta localização, e em seguida calculado o valor de alpha:

$$\alpha(x, y, n) = \frac{I_x(x, y)\bar{u}^n_{xy} + I_y(x, y)\bar{v}^n_{xy} + I_t(x, y)}{\lambda^2 + I_x^2(x, y) + I_y^2(x, y)}$$

Este valor alpha é utilizado para calcular os novos valores para as matrizes u e v daquela iteração:

$$u(x, y) = \bar{u} - \alpha(x, y, n) \cdot I_x(x, y, t)\ ;$$
$$v(x, y) = \bar{v} - \alpha(x, y, n) \cdot I_y(x, y, t)\ ;$$

Por fim, é calculada e exibida a média da diferença entre a matriz u da iteração n e a matriz u_anterior da iteração (n-1). O mesmo é feito para a matriz v:

```
media_diferenca_u = np.mean(np.abs(np.subtract(u, u_anterior).flatten()))
media_diferenca_v = np.mean(np.abs(np.subtract(v, v_anterior).flatten()))
```

Estas médias são exibidas no terminal, neste relatório são chamadas de **Ũ** e **Ṽ**.

Análise:

Conforme as implementações discutidas acima, podemos executar o algoritmo de 4 formas diferentes:

1. Inicializar u e v como **0** / Utilizar aproximação de **Horn S.**
2. Inicializar u e v como **ponto Q** / Utilizar aproximação de **Horn S.**
3. Inicializar u e v como **0** / Utilizar aproximação de **Sobel**
4. Inicializar u e v como **ponto Q** / Utilizar aproximação de **Sobel**

Ao executar o programa, para o par de quadros do vídeo passado como parâmetro serão calculados e exibidos os retultados para estas 4 possibilidades (chamadas de PASSOS).

Ao trocar a aproximações de Horn S. pela aproximação Sobel, as médias **Ũ** e **Ṽ** começam maiores nas primeiras iterações e decresem um pouco mais lentamente. Esta aproximação obtem matrizes u e v com mais diferenças entre as iterações. Pelos resultados exibidos, a aproximação de Sobel faz com que mais movimentos sejam detectados.

Variando entre as inicializações com 0 ou Q as médias Ũ e Ṽ não tiveram muito impacto, apenas decresceram um pouco mais rápido. Os resultados também não foram muito diferentes.

As médias Ũ e Ṽ poderiam ser utilizadas como critério de parada ao invés de definir um número fixo de iterações T, quando não há muita variação entre uma iteração e a outra, o algoritmo poderia ser interrompido. O parâmetro T=7 foi utilizado pois gerou uma boa variação entre as matrizes obtidas até a sétima iteração.

Questão 2

Nesta questão, variei os parâmetros sp, cr e L para calcular o meanShift para a imagem da entrada. Utilizei três valores diferentes para cada parâmetro:

1. sp: [5, 12 e 25] conforme o exemplo do enunciado
2. cr: [19, 24, 25] conforme o exemplo do enunciado
3. L: [3, 5 e 7], parâmetros obtidos testando o algoritmo e baseando na descrição do livro para o parâmetro

Realizei três laços de repetição:

1. Variar entre os valores de sp e utilizar os valores do meio para cr e L
2. Variar entre os valores de cr e utilizar os valores do meio para sp e L
3. Variar entre os valores de L e utilizar os valores do meio para cr e sp

Os 9 resultados são exibidos ao final da iteração. Foram utilizadas as funções do OpenCV para o cálculo do meanShift.

Ao variar os valores de sp, a janela espacial usada para a segmentação muda de tamanho. Para valores maiores de sp foram obtidas segmentações maiores pois estas janelas consideram mais pixels.

Ao variar os valores de cr, a janela das cores considerada para a segmentação varia. Valores maiores de cr fazem com que os segmentos obtidos sejam maiores.

Por fim, variando o valor de L, o nível máximo da pirâmide da segmentação é alterado. Quanto maior o L, segmentações maiores são obtidas, a variação deste parâmetro foi a que teve mais impacto nos resultados.

Questão 3

Esta questão recebe como parâmetro o caminho de uma pasta, onde devem estar os quadros utilizados no algoritmo, nomeados como [1.png, 2.png, 3.png, …]. Ao utilizar mais de 6 quadros como entrada, é necessário alterar no algoritmo o número de linhas e

colunasda exibição dos resultados (parâmetros n_colunas e n_linhas). Para cada quadro, são calculados:

1. As bordas da imagem, utilizando a função Canny do OpenCV. Estas bordas são usadas para o cálculo das linhas da imagem.
2. As linhas da imagem, utilizando a função HoughLinesP do OpenCV. Estas linhas são exibidas em vermelho nos resultados
3. Os cantos da imagem, utilizando a função goodFeaturesToTrack do OpenCV. Estes cantos “fortes” são exibidos em verde nos resultados. Estes cantos são necessários para o cálculo dos “subcantos”, que são os cantos refinados pela localização do subpixel.
4. Os subcantos da imagem, utilizando a função cornerSubPix do OpenCV. Estes subcantos são exibidos em azul nos resultados.

Os parâmetros passados para as funções em cada cálculo foram obtidos por experimentação.

A distorção de lente das imagens dificulta a obtenção dos quatro elementos mencionados, pois os cantos são “deslocados” de sua posição original. Existem ferramentas para correção da distorção destas imagens, para que a obtenção dos cantos seja melhor. Outros elementos que dificultam a obtenção dos elementos são ruídos na imagem e variações na iluminação.