Fundamentos de Visão Computacional (CMP197 e INF01030)
Relatório do trabalho 1

José Bruno da Silva Santos (00569622)
Rodrigo Lusa (00216668)

O objetivo deste trabalho é achar a homografia planar dada pela matriz $H_{3\times3}$ que retifique a região da captura digital correspondente à fotografia original impressa para uma região retangular cujas dimensões correspondam às da fotografia, e então faça um recorte (crop) da imagem retificada para recuperar apenas a parte correspondente à fotografia.

O trabalho está dividido em 3 seções. Inicialmente a 1ª seção apresenta as técnicas utilizadas sobre a imagem e na 2ª seção é apresentada a descrição do algoritmo implementado. Por fim, na última seção é apresentado os resultados e discussões do trabalho desenvolvido.

1. Operações sobre a imagem

- Homografia: é uma transformação projetiva planar que mapeia pontos de um plano para outro plano[1]. Este mapeamento linear de pontos pode ser escrito em coordenadas homogêneas, em que H é a matriz de homografia que define o mapeamento de um conjunto de pontos correspondentes entre dois planos.

2. Implementação

- Nosso algoritmo foi implementado utilizando apenas as bibliotecas OpenCV, Numpy e Math.

  O algoritmo basicamente consiste em realizar a leitura da imagem a ser corrigida, criamos um array contendo os quatro pontos referentes aos cantos da imagem. Após a criação do array com os cantos da imagem, temos um array com quatro cantos máximos da imagem de destino, neste ponto também caberia uma implementação automática para a criação deste array, porém em nossa implementação ele ficou inserido manualmente com a resolução da imagem de entrada.

  Após a criação dos arrays, utilizamos a função findHomografy do OpenCV, inserindo como parâmetros os dois arrays criados anteriormente. Feito isso utilizamos a função warpPerspective, também do OpenCV, informando a imagem de entrada e o retorno da função de homografia como parâmetros.

  Ao final, nosso algoritmo escreve uma imagem de saída chamada output.jpg.

  Após este processo realizamos uma métrica quantitativa usando PSNR onde utilizamos as bibliotecas Numpy e Math para a implementação.

  - **findHomografy**: funcionalidade do OpenCV que calcula a homografia entre duas imagens. Como primeiro parâmetro ela recebe **srcPoints** que são as coordenadas dos pontos no plano original, ou seja, uma matriz de células de vetores de 2 elementos {[x,y], ...} que representam os pontos equivalentes na imagem de entrada. Em nossa implementação, estes pontos são as quatro coordenadas dos cantos da foto na imagem, iniciando pelo canto superior

esquerdo e seguindo no sentido horário. O segundo parâmetro é **dstPoints** que são as coordenadas dos pontos no plano alvo, do mesmo tamanho e tipo que srcPoints. Esta função tem como saída uma **Matriz** de homografia H 3x3 e uma matriz **mask** Nx1 que possui mesmo comprimento que os pontos de entrada e indica quais pontos foram realmente usados na melhor computação de H.

- **warpPerspective**: funcionalidade do OpenCv que aplica uma transformação de perspectiva a uma imagem. Recebe como primeiro parâmetro **src** que é a imagem de entrada. Como segundo parâmetro recebe uma **Matriz** que é a matriz transformação M 3x3 resultante da execução da função findHomografy. Esta função tem como saída **dst** que é a imagem de saída que tem o mesmo tamanho de src e é o resultado de nossa implementação.
-

3. Resultados e Discussões

- Resultados Qualitativos: analisando lado a lado a foto obtida a partir do algoritmo de homografia e a foto original, é possível perceber diversas perdas na imagem que obtivemos. O nível de saturação das cores foi afetado, assim deixando a imagem com uma tonalidade de cores mais lavadas, além do alto nível de ruído, que é bastante perceptível quando comparado com as fotos lado a lado. Também notamos que deformações radiais são mantidas e não corrigidas. Gerando assim uma perda na qualidade da foto no geral.
- Resultados Quantitativos: Utilizamos uma métrica PSNR para a avaliação quantitativa que consiste em avaliar duas imagens de entrada, onde uma delas é a imagem ideal sem nenhum ruído e a imagem gerada. Para cálculo do valor de PSNR foram utilizadas as equações abaixo.

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(O(i,j) - D(i,j))^2$$

Onde MSE é o erro quadrático médio e RMSE é a raiz do erro quadrático médio.
Após análise tivemos valores de PSNR em torno de **28** (quanto mais próximo de 100, melhor é o resultado de comparação).

- No ponto em que realizamos a inserção dos cantos da imagem fizemos algumas tentativas para realizar uma identificação automática destes valores, porém sem sucesso. Utilizamos uma função do OpenCV chamada de goodFeaturesToTrack em um primeiro momento e em uma segunda tentativa uma implementação um pouco mais complexa com a função approxPolyDP, também do OpenCV. Ambas funções

foram utilizadas na tentativa de identificarmos polígonos ou formas geométricas na imagem, imaginando que poderíamos encontrar o quadrilátero da foto na qual extrairíamos as coordenadas dos quatro cantos. Nesta segunda tentativa foram realizadas algumas filtragens na imagem, usando funções como cvtColor, para transformá-la em escala de cinza, adaptiveThreshold, medianBlur e findContours. Não obtivemos o resultado esperado.

Acreditamos que esta etapa necessitaria de um filtragem diferente antes da tentativa de identificação dos cantos, porém não conseguimos encontrar.

Referências

[1] Hartley, R. e Zisserman, A. Multiple View Geometry in Computer Vision, second ed. Cambridge University Press, 2004.