Relatório Visão Computacional TA2

Eduardo Mathias e Pedro H. Kochinski
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
ems19@inf.ufpr.br — phks20@inf.ufpr.br

I. Introdução

Este relatório apresenta uma visão geral das atividades realizadas no segundo projeto de Visão Computacional. O atividade realizada foi um projeto visando alcançar a calibração de uma camêra de webcam de baixa qualidade.

II. Descrição do Problema

Algumas câmeras de baixa qualidade introduzem certa distorção significativa em imagens, essas distorções podem ser: radial ou tangencial.

A distorção radial faz com que as linhas retas pareçam curvas. A distorção radial torna-se maior quanto mais distantes os pontos estiverem do centro da imagem. Por exemplo, um tabuleiro de damas como o utilizado nesse projeto pode aparecer com todas as bordas não muito retas e coincidentes. Essa distorção pode ser representada pela fórmula:

$$xdistorted = x(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

$$ydistorted = y(1 + k_1r^2 + k_2r^4 + k_3r^6)$$

A distorção tangencial faz com que algumas áreas da imagem pareçam mais próximas do que o esperado, já que a lente de captura de imagem não está alinhada perfeitamente paralela ao plano de imagem. Essa distorção tangencial pode ser representada como abaixo:

$$xdistorted = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$ydistorted = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

Em resumo, existem cinco parâmetros definidos nessas distorções que são conhecidos como coeficientes de distorção, dados por (k1, k2, p1, p2, k3)

Fora distorções, as câmeras possuem parâmetros intrínsecos e extrínsecos. Os parâmetros extrínsecos correspondem aos vetores de rotação e translação que traduzem as coordenadas de um ponto 3D para um sistema de coordenadas. E, os parâmetros intrínsecos incluem informações como distância focal (fx,fy) e centros ópticos (cx,cy). A distância focal e os centros ópticos podem ser usados para criar uma matriz de câmera, que pode ser usada para remover a distorção. A matriz da câmera é exclusiva, portanto, uma vez calculada, ela pode ser reutilizada em outras imagens tiradas pela mesma câmera. Ela é expressa como uma matriz 3x3:

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Para encontrar esses parâmetros da câmera, precisamos fornecer algumas imagens de amostra de um padrão bem definido (por exemplo, um tabuleiro de damas). Encontramos alguns pontos específicos dos quais já conhecemos as posições relativas (por exemplo, cantos de quadrados no tabuleiro de xadrez). Conhecemos as coordenadas desses pontos no espaço do mundo real e conhecemos as coordenadas na imagem, portanto, podemos resolver os coeficientes de distorção. Para obter melhores resultados, precisamos de pelo menos 10 padrões de teste.

III. Implementação

Temos na pasta fotos do repositório Gitlab, 11 fotos de um tabuleiro de damas em diversas posições, inclinações e distâncias da câmera da webcam sob um fundo vermelho. Veja algumas delas abaixo:

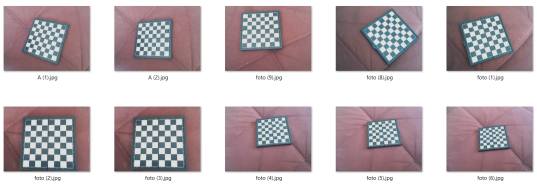

Figura 1. Fotos do tabuleiro

Agora, precisamos achar as bordas que são pontos de reconhecimento dados pela função OpenCV cv.findChessboardCorners(). De primeira, nossas imagens não conseguiram ser identificadas por essa função do OpenCV, por possuir muita informação de cor fora o tabuleiro, por isso, realizamos uma segmentação de cores para obter uma máscara binária, e, tornar apenas o tabuleiro visível.

Assim, conseguimos achar as bordas da imagem com a função acima e desenha-las na imagem com cv.drawChessboardCorners().

Após, acharmos as bordas das 11 imagens, podemos achar a calibração da camera, com a função cv.calibrateCamera() que retorna a matriz da câmera, os coeficientes de distorção, os vetores de rotação e translação.

- Matriz da Camera

$$\begin{bmatrix} 929.7165196 & 0 & 298.91385861 \\ 0 & 876.29371012 & 128.66589241 \\ 0 & 0 & 1 \end{bmatrix}$$

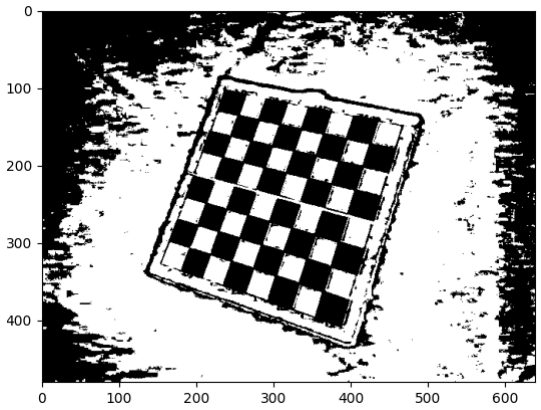

Figura 2. Imagem com tabuleiro e fundo em filtro de bitmask

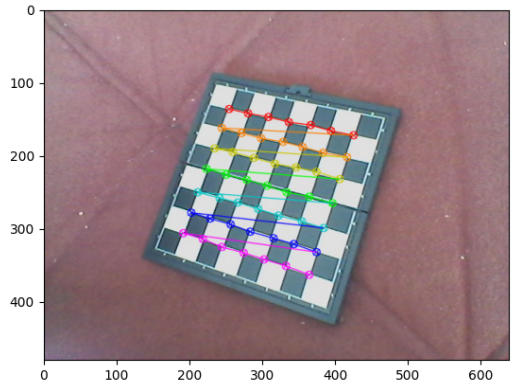

Figura 3. Bordas sob a imagem

- Coeficientes de Distorção: (-4.82689223e-01, 3.57053806e+00, 5.16068491e-02, -3.46339686e-03, -1.04859715e+01)
- Vetores de Rotação: (array([[-0.29768164], [0.4675377], [0.26458306]]), array([[0.08202089], [0.60693056], [1.5104428]]), array([[-0.352463], [0.11342348], [1.26391553]]), array([[-0.21351573], [-0.29522914], [-3.11733399]]), array([[0.1987484], [0.2943834], [3.11168347]]), array([[0.14728753], [0.90688334], [2.91310506]]), array([[-0.43990445], [0.57176604], [1.42059894]]), array([[-0.53743361], [0.6420574], [1.39533824]]), array([[-0.58181259], [0.66985105], [1.36938555]]), array([[-0.33680872], [0.06769409], [0.89471552]]), array([[-0.25071432], [0.12105976], [-0.04167786]]))
- Vetores de Translação: (array([[-1.44155407], [0.2516818], [30.33540186]]), array([[2.3988369], [0.4682143], [28.39994179]]), array([[1.89107066], [-0.49816272], [27.06382942]]), array([[2.62128731], [5.63799128], [22.60332394]]), array([[2.73525738], [5.65294534], [22.55973825]]), array([[5.11970276], [3.94994866], [32.14016964]]), array([[4.7532991], [-0.25010059], [37.2605987]]), array([[4.70497869], [1.7535332], [41.10198462]]), array([[0.95560986], [3.22704535], [41.89384635]]), array([[3.55760617], [-1.65129502], [27.91716334]]), array([[-4.10632199], [-1.31845676], [29.01997774]]))

IV. Realizando a correção da Distorção da Câmera

Com todos os parâmetros definidos, podemos pegar uma imagem e corrigi-la. O OpenCV vem com dois métodos para fazer isso. No entanto, refinaremos a matriz da câmera com base em um parâmetro de escala livre usando cv.getOptimalNewCameraMatrix(). Com parametro alfa=1, todos os pixels serão mantidos com algumas imagens pretas extras. Essa função também retorna uma ROI de imagem que pode ser usada para cortar o resultado. E agora, podemos chamar a função cv.undistort() e teremos uma imagem mais corrigida.

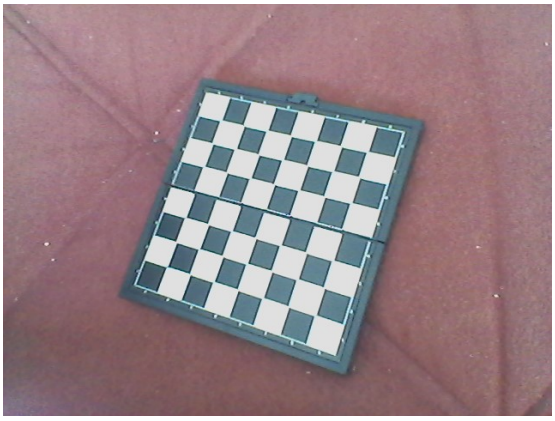

Figura 4. Foto a ser corrigida pelo processo de undistort

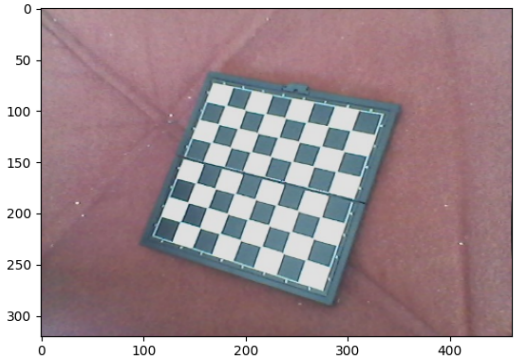

Figura 5. Foto calibrada

V. Conclusão

Neste projeto de Visão Computacional, foram realizadas atividades práticas relacionadas a identificação de bordas e calibração de cameras. Foram exploradas funções do openCV para tais tarefas, fora funções vistas no TA passado (como a segmentação de cores por máscara de bit) para a facilitação de visualização de imagens pela camêra.