Projeto final de Algebra Linear 2022.02

Aluno: Diego Vasconcelos Schardosim de Matos

DRE: 120098723

Tema: Retificação de Imagem (Exemplo do slide de numero 6)

O que é

O propósito do trabalho é conseguirmos transformar uma imagem que está "torta" e corrigir suas perspectiva, igual ao exemplo abaixo

Como

Para resolver este problema, temos que ter em mente a ideia fundamental de transformação de imagens em 2x2, isto é, temos uma imagem origem (input) onde é aplicada uma função/transformação

$$f(T(x,y)) = g(x,y)$$

Algumas das transfomações vistas no curso são rotação e alterar a escala da imagem, essas transformações podem ser representadas por uma matriz T_{2x2}

$$p_2 = Tp_1,$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Porém, para o nosso problema de perspectiva, precisamos transladar pontos da imagem para sua forma "corrigida", porém, com um exemplo simples percebemos que não é possível representar uma translação com uma matriz 2x2

$$x_2 = x_1 + t_x$$
$$y_2 = y_1 + t_y$$

Para isso precisamos usar a definição de **coordenadas homogêneas**

Coordenadas homogêneas

A representação homogênea de um ponto 2D $p = (x, y)$ em um ponto 3D é $\tilde{p} = (\tilde{x}, \tilde{y}, \tilde{z})$. Onde $\tilde{z}$ é uma coordenada $\neq 0$ fictícia usada para normalizar as outras coordenadas. Na prática é toda reta / direção no espaço 'proporcional' a z:

A relação das coordenadas x e y com z é a seguinte:

$$x = \frac{\tilde{x}}{\tilde{z}}$$
$$y = \frac{\tilde{y}}{\tilde{z}}$$

Esses pontos estão relacionados da seguinte forma:

$$p \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{z}x \\ \tilde{z}y \\ \tilde{z} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{p}$$

Podemos assim escrever a matriz de translação da seguinte forma

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

As transformações de escala e rotação também podem ser escritas como uma matriz 3x3

$$Escala \rightarrow \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$Rotação \rightarrow \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Essas transformações possuem uma coisa em comum, que é a última linha sendo 0 0 1, isso as classifica como uma **transformação afim**. Entretanto, se não restringirmos a última linha para esses valores, conseguimos uma maior liberdade na transformação da imagem

Esse tipo de transformação é chamada de **Homografia**. Essa matriz possui 9 valores desconhecidos, porém precisamos no mínimo de 4 pontos de entrada, quanto mais pontos forem fornecidos, mais precisa será o cálculo.

Retificação de Imagem

Com isso em mente podemos voltar ao nosso problema inicial, dado quatro pontos de origem e de destino conseguirmos corrigir sua perspectiva calculando a matriz de homografia entre elas

A matriz de homografia entre as duas imagens é descrita da seguinte forma

$$\begin{bmatrix} x_{dest} \\ y_{dest} \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x_{\tilde{dest}} \\ y_{\tilde{dest}} \\ z_{\tilde{dest}} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix}$$

Como estamos trazendo coordenadas num ponto no espaço (3D) para uma imagem 2D, não podemos esquecer que a coordenada z é 'perdida', entretanto, como explicado anteriormente existe uma relação entre as coordenadas x e y com z

$$x = \frac{\tilde{x}}{\tilde{z}}$$
$$y = \frac{\tilde{y}}{\tilde{z}}$$

No sistema abaixo o sobrescrito (i) refere-se a um dos pontos dados

$$\begin{cases} x_{dest}^{(i)} = \frac{\tilde{x}_{dest}^{(i)}}{\tilde{z}_{dest}^{(i)}} = \frac{h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h13}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}} \\ y_{dest}^{(i)} = \frac{\tilde{y}_{dest}^{(i)}}{\tilde{z}_{dest}^{(i)}} = \frac{h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}} \end{cases} \tag{1}$$

$$x_d^{(i)}(h_{31}x_s + h_{32}y_s + h_{33}) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$
$$y_d^{(i)}(h_{31}x_s + h_{32}y_s + h_{33}) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

Rearranjando para notação matricial

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Lembrando que a matriz da esquerda são pontos conhecidos, o quê queremos descobrir é a matriz dos h's. Como sabemos precisamos no mínimo 4 pontos para calcular a homografia, mas podemos ter mais e 'empilhar' todas numa grande matriz

$$
\underbrace{\begin{bmatrix}
x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\
0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\
0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\
0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)}
\end{bmatrix}}_{A}
\underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}}_{h}
=
\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}
$$

Agora temos um clássico problema de $Ah = 0$ que pode ser resolvido usando mínimos quadrados $A^T Ah = \tilde{x}h$

Um exemplo prático

O progama abaixo (não tão genérico) lê a imagem chamada 'imagem-teste-2.jpg' que acompanha este notebook e pede ao usuário escolher 4 pontos na imagem que é desejado realizar a correção de perspectiva.

Nota 1: Os pontos precisam ser escolhidos, obrigatoriamente, na seguinte ordem: Canto superior esquerdo, canto superior direito, canto inferior equerdo e canto inferior direito.

Nota 2: O programa irá abrir uma janela, caso ela não apareça automaticamente verifique se ela já não está aberta

Caso o usuário deseje testar com outra imagem, coloque-a na mesma pasta deste notebook e altere o nome do arquivo que o progama deve ler.

In [2]:
```
import cv2
import numpy as np
```

In [3]:
```
imageFileName = "imagem-teste-2.jpg";
originalImage = cv2.imread(imageFileName); # Le imagem

"Cria um vetor de zeros 2x4 transposto"
sourcePoints = np.zeros((4, 2), np.float32);
counter = 0;
```

In [3]:
```
"Método usado posteriormente para tratar o clique do mouse"
def mouseClickHandler(event, x, y, flags, params):
    global counter;
    if event == cv2.EVENT_LBUTTONDOWN:
        sourcePoints[counter] = x, y;
        counter = counter +  1;
```

In [4]:
```python
"Desenha circulos onde o usuário Clicou na imagem original"
def drawCircles():
    for point in sourcePoints:
        cv2.circle(originalImage, (int(point[0]), int(point[1])), 3, (0, 255, 0), c
```

In [5]:
```python
def drawOriginalImage():
    cv2.imshow("Imagem Original", originalImage);
    cv2.setMouseCallback("Imagem Original", mouseClickHandler);

    cv2.waitKey(1);
```

In [6]:
```python
"Loop responsável por exibir uma janela com a imagem escolhida e aceitar clique do
while counter != 4:
    drawCircles();
    drawOriginalImage();

cv2.destroyWindow("Imagem Original"); # Destroi janela antiga para evitar problemas
width, height = 350, 350; # Caso sua imagem original esteja numa proporção diferent

destPoints = np.float32([[0,0], [width, 0], [0, height], [width, height]]);
matrix = cv2.getPerspectiveTransform(sourcePoints, destPoints); # Calcula a matriz
imgOutput = cv2.warpPerspective(originalImage, matrix, (width, height)); # Aplica m

cv2.imshow("Imagem Final", imgOutput);
cv2.waitKey(0);
```

Outros casos de uso

Este método de correção de imagem não é o único propósito deste tipo de algoritmo, também podemos "encaixar" uma imagem de nosso interesse dentro de uma determina região em uma outra imagem ou vídeo, como a bandeira de um país numa competição de natação

Referências

- Warp Perspective / Bird View [6] | OpenCV Python Tutorials for Beginners 2020
- Image Stitching | Face Detection
- ENB339 lecture 9: Image geometry and planar homography
- OpenCv Perspective Transformation
- Warp perspective and Transform OpenCV python
- Homography examples using OpenCV (Python / C ++)
- Homework 2: Image Rectification