

Projeto final de Algebra Linear 2022.02

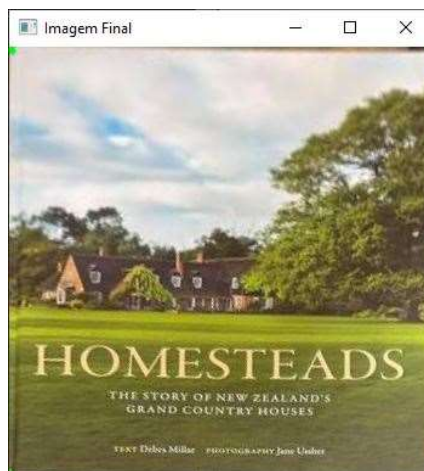
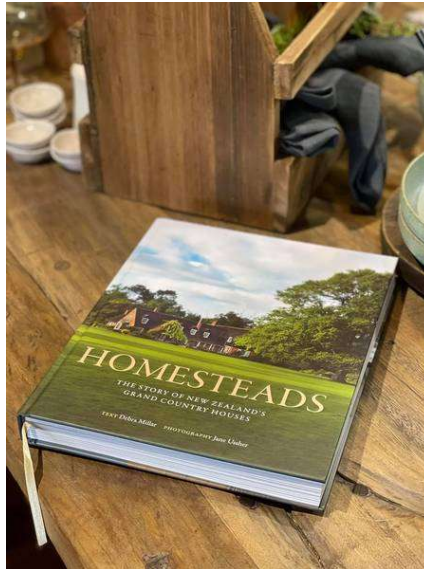
Aluno: Diego Vasconcelos Schardosim de Matos

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 transformações vistas no curso são rotação e alterar a escala da imagem, essas transformações podem ser representadas por uma matriz T 2x2

$$p_2 = T p_1,$$

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

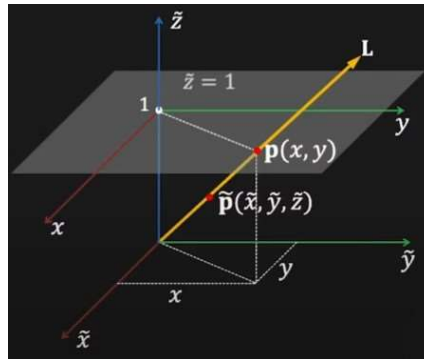
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Porém, para o nosso problema de perspectiva, precisamos transladar pontos da imagem para sua forma "corrigida", porém isso não é possível com uma matriz 2x2, como mostrado abaixo

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 :



Isto pode ser escrito como

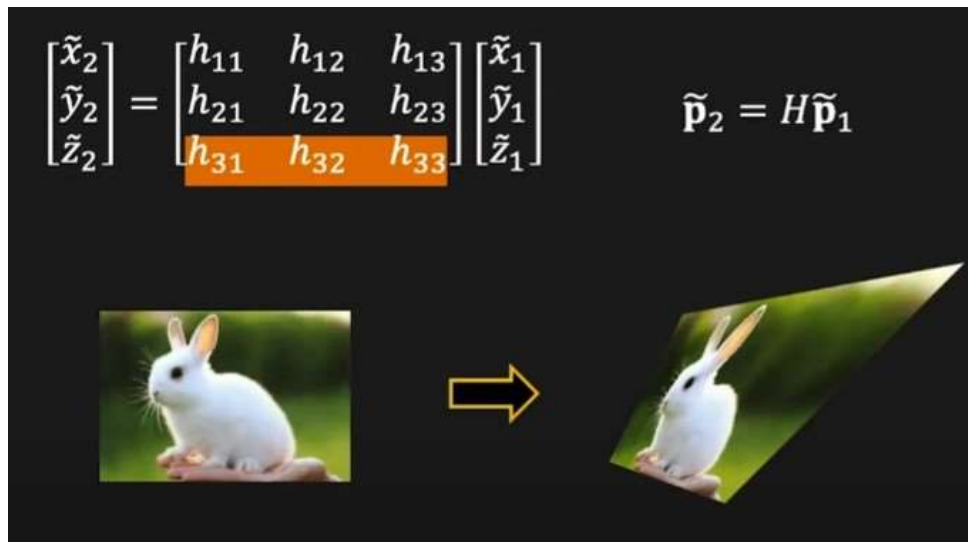
$$p \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{z}x \\ \tilde{z}y \\ \tilde{z} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{p}$$

Abaixo estão as transformações de escala, rotação e translação escritas usando uma matriz 3x3

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

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

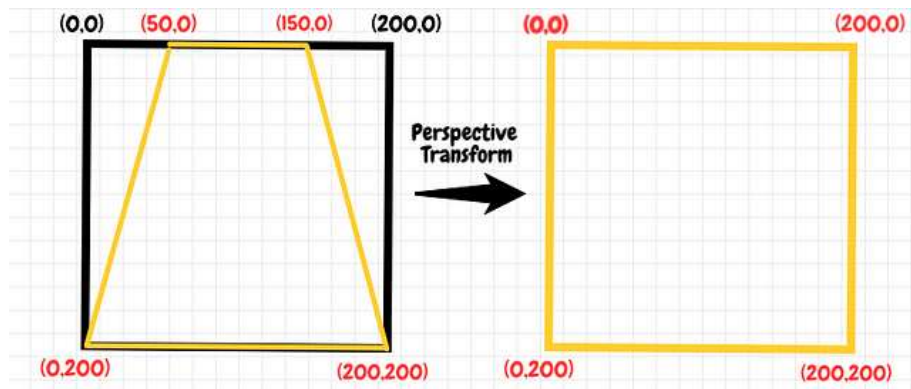
Essas transformações possuem uma coisa em comum, que é a última linha sendo 0 0 1, isso as classifica como uma transformação afim. Entretanto, se não restringirmos a última linha para esses valores, conseguimos uma maior liberdade na transformação da imagem



Esse tipo de transformação é chamada de **Homografia**

Retificação de Imagem

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



A matriz de homografia entre as duas imagens, considerando que a escala 'não muda', é a seguinte:

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

Essa matriz é calculada resolvendo o seguinte sistema linear

$$\begin{cases} x_{dest} \cdot h_{11} + x_{dest} \cdot h_{21} + x_{dest} \cdot h_{31} = x_{src} \\ y_{dest} \cdot h_{12} + y_{dest} \cdot h_{22} + y_{dest} \cdot h_{32} = y_{src} \\ h_{13} + h_{23} + 1 = 1 \end{cases}$$

Um exemplo prático

O programa 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 esquerdo 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 programa 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 mouseClickedHandler(event, x, y, flags, params):
    global counter;
    if event == cv2.EVENT_LBUTTONDOWN:
        sourcePoints[counter] = x, y;
        counter = counter + 1;
```

In [4]:

```
"Desenha círculos 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), cv2.FILLED);
```

In [5]:

```
def drawOriginalImage():
    cv2.imshow("Imagem Original", originalImage);
    cv2.setMouseCallback("Imagem Original", mouseClickedHandler);

    cv2.waitKey(1);
```

In [6]:

```

"Loop responsável por exibir uma janela com a imagem escolhida e aceitar clique do usuário"
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 diferente, altere aqui

destPoints = np.float32([[0,0], [width, 0], [0, height], [width, height]]);
matrix = cv2.getPerspectiveTransform(sourcePoints, destPoints); # Calcula a matriz de homografia mencionada anteriormente
imgOutput = cv2.warpPerspective(originalImage, matrix, (width, height)); # Aplica matriz de transforma na imagem, também é respon

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 \(https://www.youtube.com/watch?v=Tm_7fGoIVGE\)](https://www.youtube.com/watch?v=Tm_7fGoIVGE)
- [Image Stitching | Face Detection \(https://www.youtube.com/playlist?list=PL2zRgk16wsdp8KbDfHKvPYNGF2L-zQASc\)](https://www.youtube.com/playlist?list=PL2zRgk16wsdp8KbDfHKvPYNGF2L-zQASc)
- [ENB339 lecture 9: Image geometry and planar homography \(https://www.youtube.com/watch?v=fVJeJMWZcq8&ab_channel=PeterCorke\)](https://www.youtube.com/watch?v=fVJeJMWZcq8&ab_channel=PeterCorke)
- [OpenCv Perspective Transformation \(https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143\)](https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143)
- [Warp perspective and Transform OpenCV python \(https://thinkinfi.com/warp-perspective-opencv/\)](https://thinkinfi.com/warp-perspective-opencv/)
- [Homography examples using OpenCV \(Python / C ++ \) \(https://learnopencv.com/homography-examples-using-opencv-python-c/\)](https://learnopencv.com/homography-examples-using-opencv-python-c/)
- [Homework 2: Image Rectification \(https://web.ece.ucsb.edu/~manj/ece181bS04/ECE_181b_HW2/rectification.pdf\)](https://web.ece.ucsb.edu/~manj/ece181bS04/ECE_181b_HW2/rectification.pdf)