Relatório Visão Por Computador
Aula Prática 5

Luís Silva, nmec 88888
luisfgbs@ua.pt
Mestrado em Robótica e Sistemas Inteligentes
Universidade de Aveiro

I. Introdução

Este relatório foi redigido no âmbito da UC Visão Por Computador, lecionada pelo professor Paulo Dias e visa encapsular o método de resolução dos exercícios e resultados obtidos

Dezembro 2021

II. Calibração da câmara

Todas as câmaras são diferentes. Mesmo dentro do mesmo modelo, apesar de partilharem componentes como a lente, pequenos erros como desvio da posição do sensor fazem com que seja necessário proceder à calibração da câmara. A calibração é o processo através do qual, utilizando imagens sucessivas de um padrão conhecido em múltiplas posições, se pode calcular o parâmetros intrínsecos e de distorção de uma câmara

A. Chessboard Calibration

Neste caso para calibrar a câmara vamos utilizar um tabuleiro de xadrez. Devemos indicar as dimensões do tabuleiro e preparamos um array de pontos reais e um array que vai guardar os pontos reconhecidos no tabuleiro de xadrez

1) Reconhecer Tabuleiro de Xadrez: Em calibrate.py definimos a função findAndDisplayChessboard(). Esta, numa imagem, procura o padrão de um tabuleiro de dimensões especificadas e, caso encontre, estes serão desenhados sobre a imagem e vão ser devolvidos para que possam ser adicionados ao array de pontos reconhecidos

```
def  FindAndDisplayChessboard(img):
    # Find the chess board corners
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ret, corners = cv2.findChessboardCorners(gray, (board_w,board_h),None)

    # If found, display image with corners
    if ret == True:
        img = cv2.drawChessboardCorners(img, (board_w, board_h), corners, ret)
        cv2.imshow('img',img)
        cv2.waitKey(500)

    return ret, corners
```

Fig. 1. findAndDisplayChessboard()

Fig. 2. Visualização dos cantos encontrados numa imagem

```
for image in images:
    img = cv2.imread(image)
    dimensions=img.shape[:2]
    ret, corners = FindAndDisplayChessboard(img)
    if ret == True:
        objpoints.append(objp)
        imgpoints.append(corners)
```

Fig. 3. Adição dos pontos encontrados ao array img_points

2) Calibração: Tendo o array pontos no mundo real, o array de pontos encontrados nas imagens e as dimensões das imagens podemos então proceder à calibração da imagem. A função calibrateCamera() recebe estes argumentos e calcula a distorção a matriz de parâmetros intrínsecos, rotação e translação da câmara

```
#ret, matriz, distorção, rotacao, translação
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, dimensions, None, None)
```

Fig. 4. Calibrar a câmara

a) Para que não seja necessário repetir este processo múltiplas vezes, vamos guardar estes valores num ficheiro:

```
#save parameters
np.savez("camera.npz" , intrinsics = mtx , distortion = dist)
```

Fig. 5. Guardar parâmetros

III. Projeção pontos 3D na imagem

Tendo a câmara calibrada é agora possível projetar sobre a imagem pontos 3D

A. Projeção nas imagens utilizadas para calibração

Este é o caso mais simples pois para cada uma delas já conhecemos os valores das matrizes de translação e de rotação que foram obtidos juntamente com as matrizes de parâmetros intrínsecos e distorção da câmara durante a calibração. Basta, por isso, definir os pontos que se quer desenhar na imagem e passar toda esta informação à função projectPoints() que irá devolver onde deverão ser representados os pontos desejados. Neste exeercício vamos desenhar um cubo

```
#projeção cubo em imagem
cubo=np.float32([[0,0,0],[0,0,-1], [1,0,-1], [1,0,0],
                 [0,1,0], [0,1,-1], [1,1,-1], [1,1,0]]).reshape(-1,3)
point_projection, jac=cv2.projectPoints(cubo, rvecs[index], tvecs[index], mtx, dist)
```

Fig. 6. projectPoints()

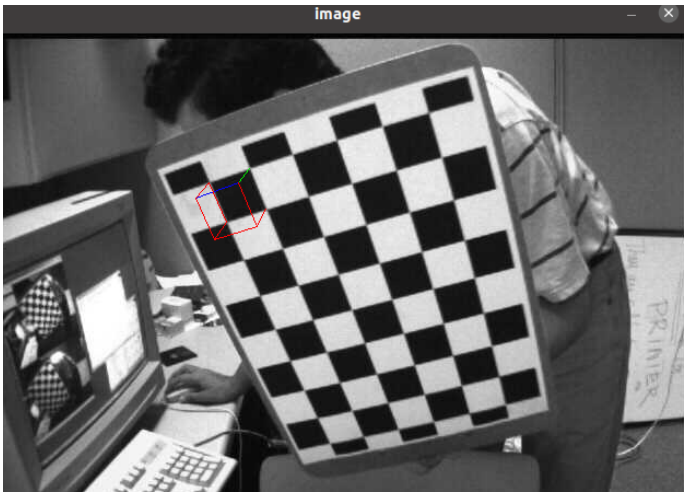

Fig. 7. Desenho de um cubo sobre a imagem na posição (1,1)

B. Projeção em vídeo

Quando pretendemos fazer a projeção de pontos sobre um vídeo temos o desafio acrescido de a cada instante termos uma frame diferente e nova e para a qual não conhecemos os valores de translação e de rotação. A análise deve portanto ser feita frame a frame e recorrendo à função solvePnP().
Em cada frame procuramos na imagem o padrão conhecido, neste caso o tabuleiro de xadrez, usando findChessboard-Corners(). Caso se encontre o padrão, chamamos solvePnP() com as posições dos pontos encontrados e os parâmetros intrínsecos e de distorção da câmara e serão devolvidos os valores de rotação e translação para a frame em estudo e a partir daí poderemos proceder com a projeção como discutido no ponto anterior

```
while True:
    ret, frame= capture.read()

    frame = cv2.flip(frame, 1) # the second arguments value of 1 indicates that w

    ret, corners = cv2.findChessboardCorners(frame, (board_w,board_h),None)

    if ret == True:
        #Como estamos a fazer frame a frame, não há necessidade de criar os array
        ret, rvec, tvec = cv2.solvePnP(objp, corners ,intrinsics, distortion)
```

Fig. 8. Tratamento frame a frame com recurso a solvePnP()