Relatório Visão Por Computador
Aula Prática 6 e 7

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 Stereo

O objetivo é calibrar o setup stereo de câmaras. Para isso vamos usar capturas no mesmo instante e da mesma cena pelas câmaras left e right

A. Deteção de cantos do tabuleiro

Deve-se ter em conta que é necessário que nos certifiquemos que as imagens correspondentes são analisadas em conjunto. Para isso, após carregar as imagens tiradas por cada uma das câmaras, fazemos um sorted() para que estas fiquem ordenadas.

```
# Read images
left_images = sorted(glob.glob('../P5/images/left*.jpg'))
right_images = sorted(glob.glob('../P5/images/right*.jpg'))
```

Fig. 1. Ordenação das imagens

Para cada set de imagens capturadas, fazemos o processo de procurar pelos cantos do tabuleiro de xadrez e, se forem encontrados, adicioná-los ao img_points respetivo. Com isto e sabendo as dimensões das imagens podemos proceder à calibração

B. stereoCalibrate()

Para o processo de calibração stereo são necessárias as matrizes de parâmetros intrínsecos e de distorção de cada uma das câmaras. No entanto, desde que alimentemos a função stereoCalibrate() com os valores apropriados (os arrays de img_points de cada uma das câmaras e o array de obj_points), a função irá calcular estes valores e utilizá-los e por isso podemos passar quatro valores None no lugar das respetivas matrizes de parâmetros intrísecos e de distorção

Esta função vai devolver as respetivas matrizes de parâmetros intrísecos e de distorção, as matrizes de translação e rotação (R e T) que permitem fazer a conversão de pontos de imagens de uma câmara para a da outra, a matriz essencial

```
#Calibração stereo
ret, mtx_left, dist_left, mtx_right, dist_right, R, T,E, F = cv2.stereoCalibrate(
    objpoints, imgpoints_left, imgpoints_right,
    None, None, None, None, dimensions,
    flags=cv2.CALIB_SAME_FOCAL_LENGTH)
```

Fig. 2. Chamada de stereoCalibrate()

(E - mapeia os pontos de uma imagem para a outra tendo conta R e T) e a matriz fundamental(F - mapeia os pixeis de uma imagem para a outra tendo em conta R, T e os parâmetros intrínsecos das câmaras)

Finalmente, guardamos os resultados num ficheiro .npz para que não seja necessário estar sempre a repetir o processo de calibração

```
np.savez("stereoParams.npz",
        intrinsics1 = mtx_left, distortion1 = dist_left,
        intrinsics2 = mtx_right, distortion2 = dist_right,
        R=R, T=T, E=E, F=F)
```

Fig. 3. Guardar os resultados para a posteridade

C. Remover distorção

Aqui transforma-se a imagem para compensar a distorção da lente. Para esta operação usamos os parâmetros guardados no exercício anterior. São visíveis os efeitos sobre a imagem ao remover a distorção

```
left_image_undistorted=cv2.undistort(left_image, intrinsics_left, distortion_left, None)
right_image_undistorted=cv2.undistort(right_image, intrinsics_right, distortion_right, None)
```

Fig. 4. Código de remoção de distorção para as duas imagens selecionadas

D. Linhas Epipolares

Dado um ponto numa das imagens, sabemos que esse mesmo estará representado na outra na outra sobre a linha epipolar correspondente. Numa imagem com a distorção corrigida mas que ainda não tenha sido rectifida, a linha epipolar que contém o ponto (x,y) é dada por

$$ax + by + c = 0$$

Para obter os coeficientes podemos usar a função cv2.computeCorrespondEpilines(). Usando os pontos P0=(0,

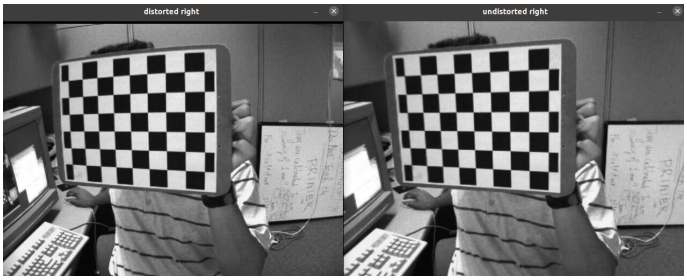

Fig. 5. Comparação imagem original e undistorted da direita

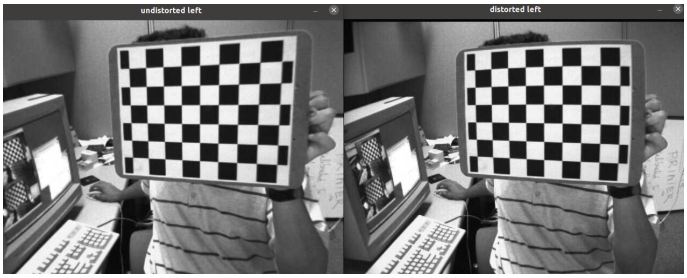

Fig. 6. Comparação imagem original e undistorted da esquerda

y1) e P1=(image_width, y2) para traçar a linha epipolar, basta-nos agora apenas resolver a equação previamente apresentada para conhecermos as ordenadas dos pontos em questão. Daqui resultam as seguintes expressões:

$$y1 = -c/b$$

$$y2 = -(ax + c)/b$$

Tendo reunido toda a informação podemos desenhar a linha epipolar que passa pelo ponto (x,y)

```
if event==cv2.EVENT_LBUTTONDOWN:
    p=np.array([x,y])

    epilineR=cv2.computeCorrespondEpilines(p.reshape(-1,1,2),1,F) #returns coeficient
    epilineR=epilineR.reshape(-1,3)[0]

    color = np.random.randint(0,255,3).tolist()

    start_point_y=int(-epilineR[2]/epilineR[1]) #simpler because x is 0
    end_point_y=int(-((epilineR[0]*width+epilineR[2])/epilineR[1]))

    cv2.line(undistorted_right, (0, start_point_y), (width, end_point_y), color, 1)
    cv2.imshow("undistorted right", undistorted_right)
```

Fig. 7. Código desenha das linhas epipolares

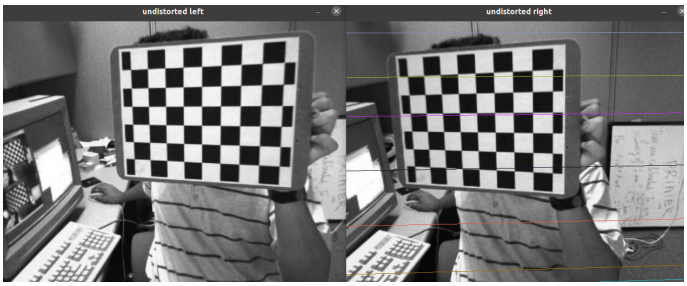

Fig. 8. Linhas epipolares

Visivelmente ao selecionar pontos mais embaixo na imagem obtemos linhas epipolares com declive mais acentuado e selecionar pontos mais acima resulta em linhas epipolares mais próximas da horizontal

E. Rectificação da imagem

Aplicar transformações a um par de imagens stereo para que pontos correspondentes partilhem sempre a ordenada. Daqui resulta que as linhas epipolares passaram a ser linhas horizontais

1) Rectificação: Começamos por preparar matrizes de dimensões apropriadas que irão receber os valores de retorno de stereoRectify R1, R2, P1, P2 e Q e, de seguida, chamamos a função

```
R1=np.zeros(shape=(3,3))
R2=np.zeros(shape=(3,3))
P1=np.zeros(shape=(3,4))
P2=np.zeros(shape=(3,4))
Q=np.zeros(shape=(4,4))

cv2.stereoRectify(intrinsics_left, distortion_left,
                  intrinsics_right, distortion_right,
                  dimensions, R, T, R1, R2, P1, P2, Q,
                  flags=cv2.CALIB_ZERO_DISPARITY, alpha=-1, newImageSize=(0,0))
```

Fig. 9. Uso de stereoRectify()

2) Mapeamento dos pontos: A função initUndistortRectifyMap() utiliza os parâmetros intrínsecos e de distorção da câmara e as matrizes de rotação e projeção respetivas à mesma calculadas com stereoRectify() e produz dois arrays que fazem o mapeamento direto dos pontos da imagem rectificada para a imagem original e vice-versa. Aplicamos estas transformações para obter a imagem retificada

```
map_left_x, map_left_y=cv2.initUndistortRectifyMap(intrinsics_left, distortion_left,
                                                   R1, P1, dimensions, cv2.CV_32FC1)
```

Fig. 10. Mapeamento de pontos

```
remapped_left=cv2.remap(left_image_undistorted, map_left_x, map_left_y, cv2.INTER_LINEAR)
```

Fig. 11. Remap

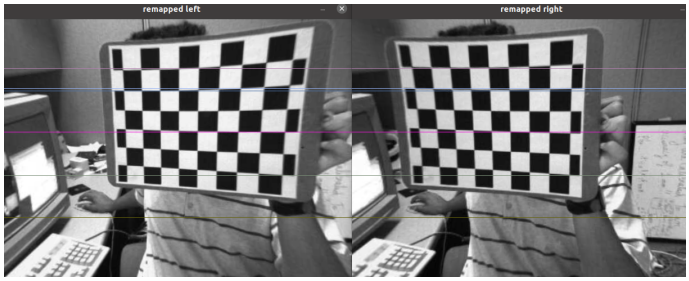

Fig. 12. Imagens retificadas

III. Visão 3D

Podemos utilizar o conhecimento até agora adquirido para criar, visualizar e processar nuvens de pontos com o open3D.

A. StereoBM

Implementa técnicas de block matching para calcular a disparidade entre as duas imagens

```
# Call the constructor for StereoBM
stereo = cv2.StereoSGBM_create(numDisparities =16*5 , blockSize =21)
# Calculate the disparity image
disparity = stereo.compute(remapped_left, remapped_right)
```

Fig. 13. StereoBM

Fig. 14. Generated Disparity Map

B. Reconstrução 3D

Utilizando o mapa de disparidade e a matriz Q calculadas anteriormente em conjunto com a função reprojectImageTo3D() podemos calcular os coordenadas 3D dos pixeis no mapa

C. Visualização da nuvens de pontos

Para uma melhor visualização podemos filtrar os pontos resultantes da reprojeção onde tenha ocorrido uma leitura inválida (+/- inf) e podemos também limitar os pontos num determinado intervalo de profundidades

```
for i in range(coords_3d.shape[0]):
    if np.all(~np.isinf(coords_3d[i])) and coords_3d[i][2]>1 and coords_3d[i][2]<3:
        filtered_coords.append(coords_3d[i])

pcd = o3d.geometry.PointCloud()
pcd.points=o3d.utility.Vector3dVector(filtered_coords)
o3d.visualization.draw_geometries([pcd])
```

Fig. 15. Filtragem dos pontos

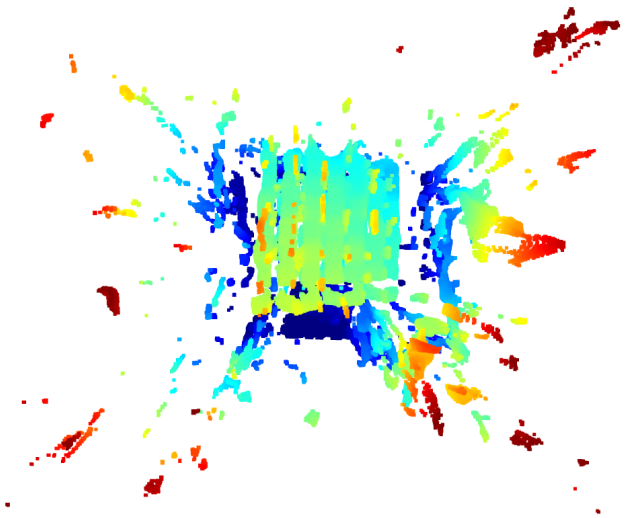

Fig. 16. Visão frontal da nuvem de pontos

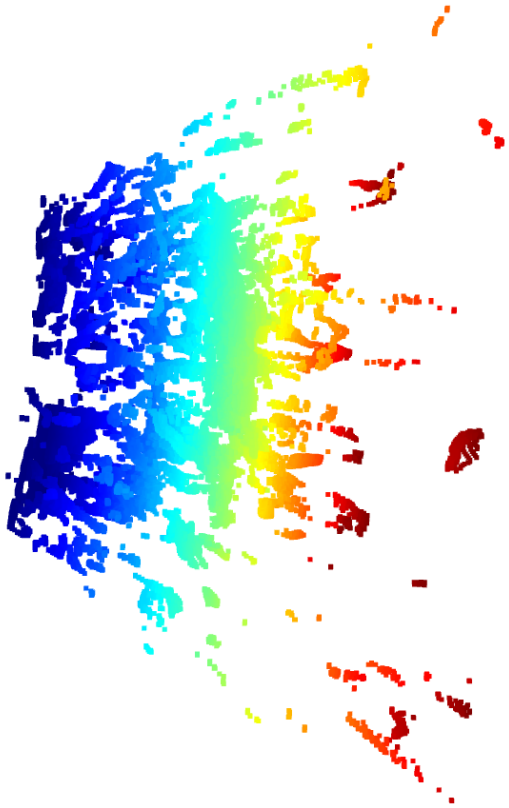

Fig. 17. Visão lateral da nuvem de pontos

D. ICP Alignment

Processo iterativo de alinhamento de duas nuvens de pontos através da aproximação dos pontos das mesmas segundo uma transformação que pretende minimizar a distância entre os pontos até que se atinja, idealmente, o mínimo absoluto

a) Initial Registration: O threshold representa o tamanho da vizinhança que vai ser considerada na aproximação dos pontos. Se não houver nenhuma sobreposição dos pontos das duas nuvens, necessitaremos de definir uma vizinhança vasta para que se possível ao algoritmo fazer a aproximação de pontos e daqui poderá resultar que encontremos um mínimo local e o alinhamento não seja correto. Para que se evite isto, é aconselhável a que se proceda a uma initial registration das duas nuvens, ou seja, através de uma matriz de coordenadas homogéneas 4x4 indicar a translação e rotação indicadas para que as nuvens fiquem mais próximas, haja sobreposição de pontos e, consequentemente, se possa utilizar vinhanças menores. Neste caso, dada a proximidade das duas nuvens, podemos usar a matriz identidade como transformação inicial

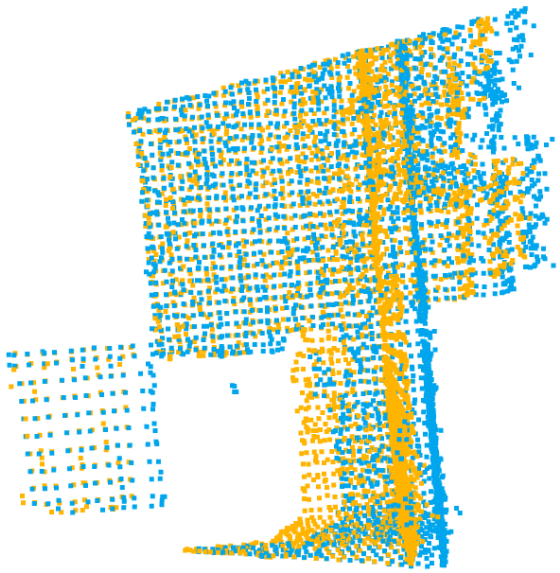

Fig. 18. Nuvens de pontos pré-alinhamento

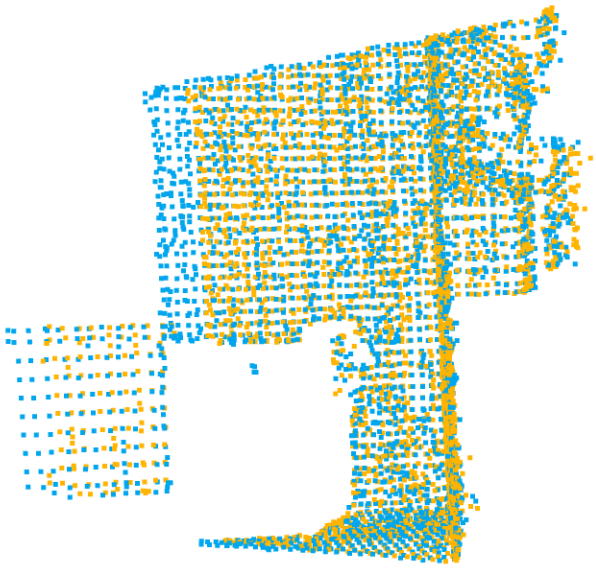

Fig. 19. Nuvens de pontos pós-alinhamento

Fig. 20. Exemplo má initial registration

Fig. 21. Exemplo de mau alinhamento