UFPB-CI-DSC
Disciplina: Visão Computacional
Prof. Augusto de Holanda B. M. Tavares
25 de Junho de 2024

14ª Atividade

Pareamento de *features*

Considere as imagens abaixo:

Figura 1: Objeto.

Figura 2: Cena.

Converta as imagem para a escala de cinza e implemente um código que execute as seguintes tarefas:

1. Realize o pareamento das *features* do objeto com as *features* da cena por força-bruta a partir dos descritores obtidos por algoritmo ORB. Ordene os pareamentos obtidos em ordem crescente de distância e desenhe os 20 pareamentos de menor distância.

2. Repita o item anterior, mas utilizando os descritores obtidos a partir do algoritmo SIFT. Compare este resultado com o resultado do item anterior.

3. Aplique o critério de Lowe com $k = 0.55$ a partir de um pareamento por kNN com $k = 2$ utilizando os descritores ORB por força bruta. Informe quantos pareamentos atenderam ao critério e desenhe estes pareamentos. Compare com o resultado das tarefas anteriores.

4. Aplique o critério de Lowe com $k = 0.55$ a partir de um pareamento por kNN com $k = 2$ utilizando os descritores ORB e utilizando a FLANN. Informe quantos pareamentos atenderam ao critério e desenhe estes pareamentos. Compare com o resultado das tarefas anteriores.

Funções de referência

Algumas funções de referência necessárias para implementar as tarefas acima são descritas abaixo:

```
bf = cv.BFMatcher(distance, crossCheck)
matches = bf.match(des1,des2)
distance_0 = matches[0].distance
```

- Cria um objeto de pareamento por força-bruta bf. O parâmetro distance pode ser definido como $cv.NORM\_L2$ (padrão) para distância L2, $cv.NORM\_L1$ para distância L1 ou $cv.NORM\_HAMMING$ para distância de Hamming, dentre outros. $crossCheck$ pode ser definido em $True$ ou $False$ para habilitar ou desabilitar a verificação de mão-dupla dos pareamentos (habilitado por padrão). A classe do objeto bf possui um método que retorna uma lista de objetos onde cada elemento identifica um pareamento. Cada um destes elementos possui vários atributos, incluído a distância calculada para aquele pareamento específico.

```
matches = bf.knnMatch(des1,des2,k)
```

- Retorna os pareamentos pelo método kNN, isto é, cada pareamento é uma lista de k pareamentos ordenados do melhor para o pior pareamento de acordo com a distância.

```
img3 = cv.drawMatches(img1,kp1,img2,kp2,matches,
                      None,matchesThickness,flags)
# or
cv.drawMatches(img1,kp1,img2,kp2,matches,
               img3,matchesThickness,flags)
```

- Retorna uma imagem que "cola" duas outras imagens e traça linhas de grossura $matchesThickness$ de acordo com os pareamentos especificados em $matches$. O parâmetro $flags$ informa como os pontos-chave serão desenhados, e um valor comum é $cv.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS$.

```
flann = cv.FlannBasedMatcher(index_params,search_params)
```

- Cria um objeto de pareamento utilizando a biblioteca FLANN e configurado de acordo com $index\_params$ e $search\_params$, que são dicionários. Estes parâmetros são variáveis de acordo com a aplicação e o tipo de descritor utilizado, e uma configuração para uso com descritores ORB é fornecida abaixo. O uso do objeto $flann$ obtido para realizar o pareamento por kNN é idêntico ao caso por força-bruta.

```
FLANN_INDEX_LSH = 6
index_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6,
                   key_size = 12,
                   multi_probe_level = 1)
search_params = dict(checks=100)
```