Project 5 Writeup

Nomes

- Caio Cesar Hideo Nakai
- Gabriel Choptian

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.

Introdução

O objetivo deste projeto é criar um mosaico de imagens ou costura, que é uma coleção de pequenas imagens que são alinhadas corretamente para criar uma imagem maior. Especificamente, estimaremos a matriz de homografia, que relaciona pontos em um plano a outro. A matriz de homografia pode ser estimada usando correspondências entre pontos. Para estimar a matriz de homografia, a entrada corresponde a pontos 2d em duas imagens. A matriz de homografia será estimada usando as correspondências entre pontos do SIFT e do RANSAC.

Detalhes de Implementação

O código abaixo é basicamente quase todo o código implementado, o algoritmo seleciona 8 pares de features aleatórios (linha 4 a 11), estima a homografia a partir dos pontos selecionados (linha 14), aplica a homografia estimada (linha 17), computa os inliers através do erro (linha 21), verificando se é menor que o threshold (5, escolhido empiricamente), estes passos são executados várias vezes e no final é mantida a matriz de homografia com a maior quantidade de inliers. Por fim, é recomputada a matriz de homografia com os inliers encontrados (linha 35 a 38).

```
1 % la o externo do RANSAC
2 for i = 1:10000
3   % pega indexes aleatorios dentro do intervalo dos
       matches
4   random_points = randperm(N, sampled_points);
5
```

```
6     % pega os pontos aleatorios dos matches da imagem 1
7     x1_random = x1(random_points);
8     y1_random = y1(random_points);
9     % pega os pontos aleatorios dos matches da imagem 2
10    x2_random = x2(random_points);
11    y2_random = y2(random_points);
12
13    % estima a homografia com a amostra gerada
          randomicamente
14    H_temp = est_homography(x1_random, y1_random, x2_random
          , y2_random);
15
16    % fun  o que aplica a homografia calculada
          anteriormente
17    [x_correspondente, y_correspondente] = apply_homography
          (H_temp, x2, y2);
18
19
20    % calcula o erro e verifica (para todos os pontos), se
          o erro    menor que o threshold definido
21    inliers = ((x_correspondente - x1).^2 +(
          y_correspondente - y1).^2 ) < threshold;
22
23    % verifica se a quantidade de inliers calculada
          maior que a
24    % quantidade de inliers da itera  o anterior
25    if sum(inliers) > size(inlier_ind,1)
26      % salva a melhor a homografia
27      H = H_temp;
28
29      % pega o indice dos inliers
30      inlier_ind = find(inliers);
31
32    end
33  end
34  % recomputa a homografia
35  H = est_homography(x1(inlier_ind), y1(inlier_ind), x2(
        inlier_ind), y2(inlier_ind));
36  [x_correspondente, y_correspondente] = apply_homography(H
        , x2, y2);
37  inliers = ((x_correspondente - x1).^2 +(y_correspondente
        - y1).^2 ) < threshold;
38  inlier_ind = find(inliers);
```

Resultados

As Figuras 1, 2, 3, 4, 5 são as fotos tiradas do fundo da UTF para montar a panorâmica. As Figuras 6, 7, 8, 9 são os matches parciais que mostram a quantidade de inliers. A Figura 10 é o resultado final obtido após a execução do algoritmo desenvolvido.

Figure 1: Foto 1

Figure 2: Foto 2

Figure 3: Foto 3

Figure 4: Foto 4

Figure 5: Foto 5

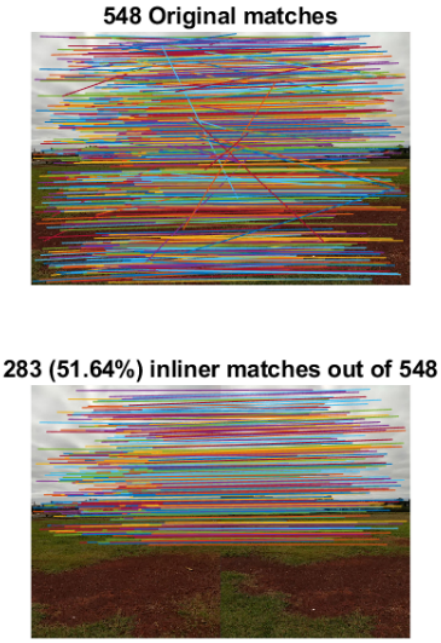

Figure 6: Match 1

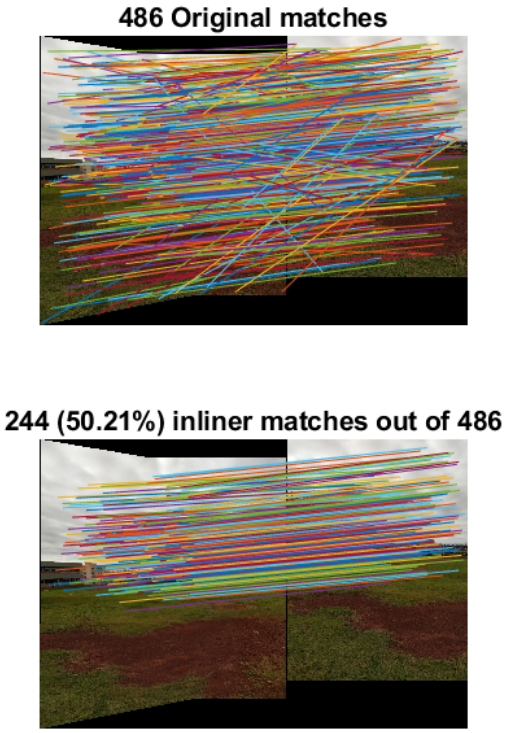

Figure 7: Match 2

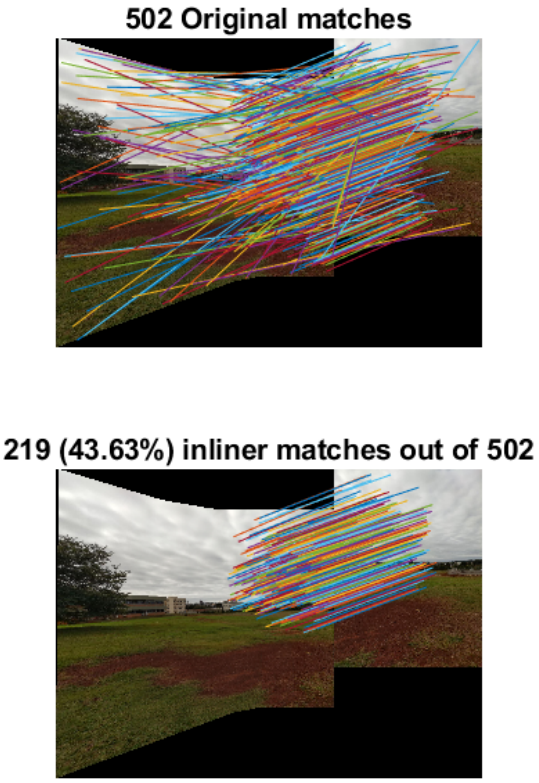

Figure 8: Match 3

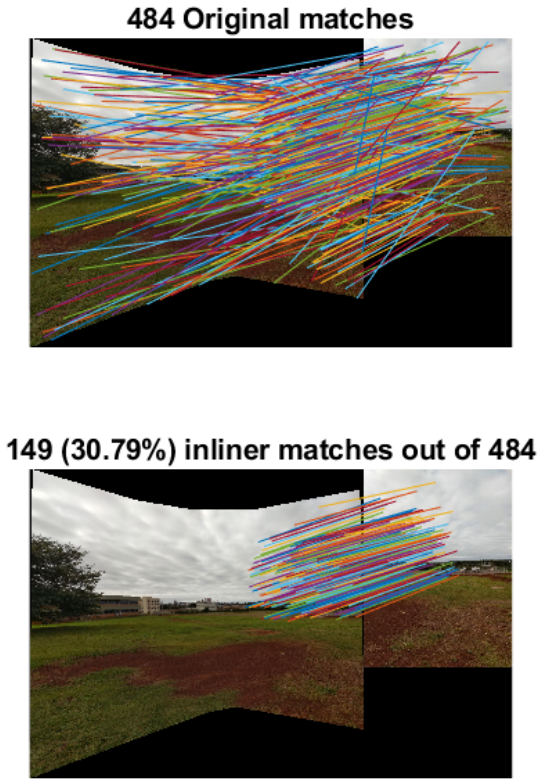

Figure 9: Match 4

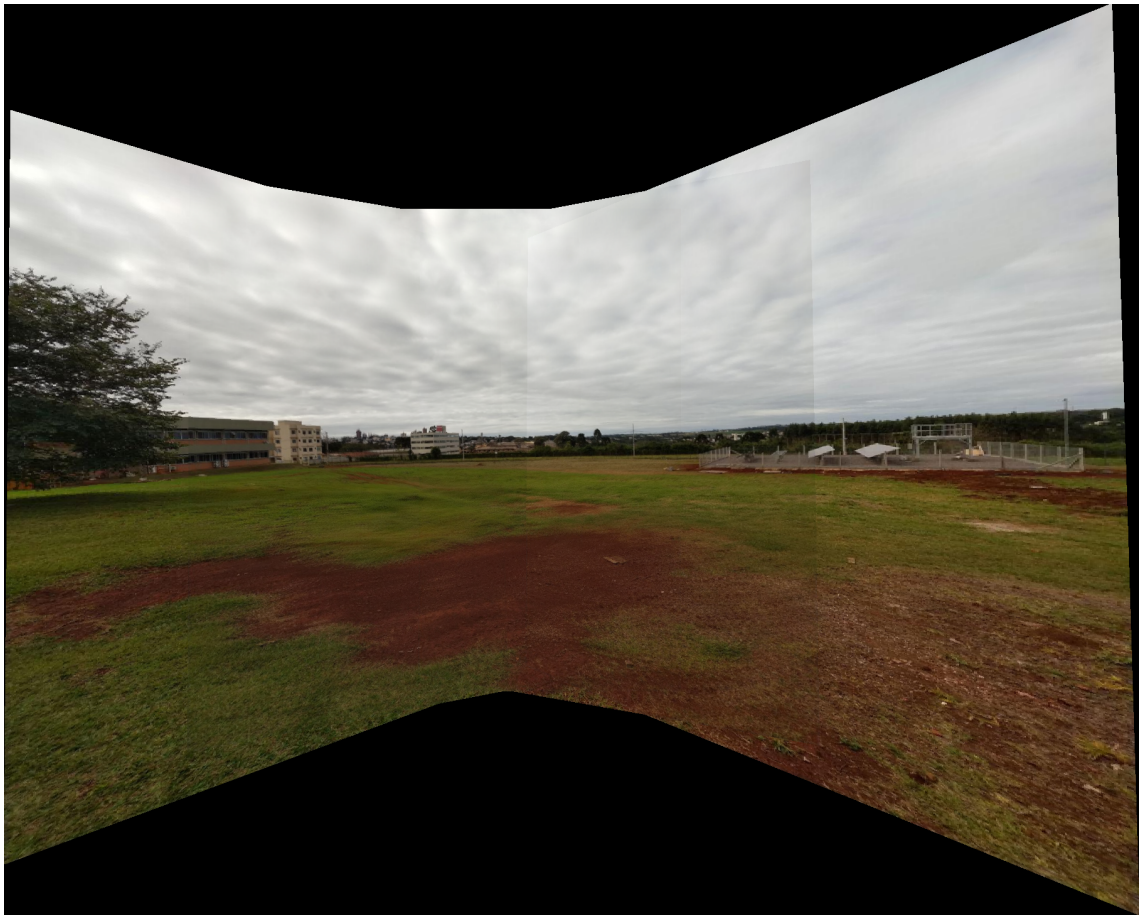

Figure 10: Mosaic: Resultado Final