|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **과목명** | **영상신호처리** | **학기** | **2023 - 1** | **담당교수** | **장주용** |
| **학과** | **전자통신공학과** | **학번** | **2018707072** | **이름** | **이승룡** |
| **과제 4 - Image Stitching** | | | | | |

**1. 과제설명**

“dlt.m” 파일의 DLT 알고리즘을 구현하라.

“im\_warp.m” 파일의 image warping 알고리즘을 구현하라.

“fitHomographyRansac.m” 파일의 RANSAC 기반 homography estimation 알고리즘을 구현하라.

- 매트랩이 제공하는 관련 코드 (예. fitgeotrans(), imwarp() 등을 사용하면 안 됨)

텍스트, 스크린샷, 패턴, 디자인이(가) 표시된 사진

자동 생성된 설명

**2. 이론**

Image stitching은 여러 이미지를 겹치는 영역과 결합하여 합성 이미지를 만드는 데 사용되는 기술이다. 사진, 컴퓨터 비전 및 이미지 처리 응용 프로그램에서 일반적으로 사용되어 파노라마 이미지를 생성하거나 서로 다른 관점에서 촬영한 이미지를 하나의 이미지로 만든다.

Image stitching은 다음과 같은 순서로 수행된다.

1. Feature extraction: 이 단계에서는 각 이미지에서 고유한 특징(엣지, 코너 등)을 추출한다. SIFT(Scale-Invariant Feature Transform) 또는 SURF(Speeded Up Robust Features)와 같은 알고리즘들이 주로 사용된다.
2. Feature Matching: 서로 다른 이미지에서 추출한 특징을 매칭하여 겹치는 영역에서 correspondence point를 찾는다.
3. Estimate Homography: feature matching에서 얻은 데이터를 바탕으로 이미지 간의 변환을 설명해주는 homography 행렬을 추정한다.
4. Image warping: 추정된 Homography를 사용하여 계산된 transformation을 기반으로 정렬되도록 이미지를 변환한다. 이때 왜곡이나 잘못된 정렬을 없애기 위한 변환, 회전, 배율 조정 및 원근 교정과 같은 기하학적 변환이 포함된다.
5. Image blending: 이미지를 정렬한 후 겹치는 영역이 함께 blending 되어 이미지 간에 전환이 매끄러워진다.

**3. 소스코드 설명**

|  |
| --- |
| % Construct the matrix A  % 'A'에 대해 SVD를 수행, 가장 작은 특이값에 해당하는 벡터를 추출.  A = zeros(2\*N, 9);  for i = 1:N  x = p1(1, i);  y = p1(2, i);  u = p2(1, i);  v = p2(2, i);  A(2\*i-1, :) = [-x, -y, -1, 0, 0, 0, x\*u, y\*u, u];  A(2\*i, :) = [0, 0, 0, -x, -y, -1, x\*v, y\*v, v];  end  % Perform Singular Value Decomposition (SVD) of A  [~, ~, V] = svd(A);  % Extract the right singular vector corresponding to the smallest singular value  H = reshape(V(:, end), 3, 3)';  end |

Homography 행렬을 계산하기 위한 Direct linear transform 함수 코드이다. Homography는 3x3행렬로, 미지수가 9개이지만 DoF(자유도)가 8이기 때문에 식 8개, 즉 4개의 매칭 포인트를 이용해서 homgraphy 행렬을 구하게 된다. H는 A를 특이값 분해하여 구할 수 있다.

|  |
| --- |
| % Determine the required number of iterations  %======= IMPLEMENT HERE!! =======%  num\_iterations = ceil(log(1 - Ps) / log(1 - (1 - proportion\_of\_outliers)^s));  %================================%  % Update the number of iterations required if we got a lot of inliers  if 1 - num\_inlier/N < proportion\_of\_outliers  proportion\_of\_outliers = 1 - num\_inlier/N;    %======= IMPLEMENT HERE!! =======%  num\_iterations = ceil(log(1 - Ps) / log(1 - (1  proportion\_of\_outliers)^s));  %================================% |

fitHomographyRansac 코드의 구현 부분이다. 위의 코드는 RANSAC을 처음 진행할 때와 어느정도 진행 후 반복 횟수를 정해준다. 샘플링 된 집합들 중에서 적어도 한 샘플 집합이 모두 inlier이 경우를 나타내는 p와 모델을 fitting하기 위한 최소 point수인 s는 결정된 상태이다. 여기서 outlier ratio를 처음엔 크게 설정하고, 어느 정도 반복 이후 outlier ratio를 inlier수를 이용해 구하고 새로운 N을 설정한다.

|  |
| --- |
| % Compute image 'out'  for i = 1:W\_out  for j = 1:H\_out  % Coordinates of output point  x = (i-1)\*R.PixelExtentInWorldX + R.XWorldLimits(1);  y = (j-1)\*R.PixelExtentInWorldY + R.YWorldLimits(1);  %======= IMPLEMENT HERE!! =======%  % Transform coordinates using the inverse homography  p = Hinv \* [x; y; 1];  % Convert to homogeneous coordinates  p = p ./ p(3);  % Map the point to the input image  x\_in = p(1);  y\_in = p(2);  % Check if the mapped point is within the input image bounds  if x\_in >= 1 && x\_in <= W\_in && y\_in >= 1 && y\_in <= H\_in  % Perform bilinear interpolation  x\_floor = floor(x\_in);  y\_floor = floor(y\_in);  x\_ceil = ceil(x\_in);  y\_ceil = ceil(y\_in);  % Interpolation weights  wx = x\_in - x\_floor;  wy = y\_in - y\_floor;  % Retrieve pixel values from the input image  pixel\_tl = double(in(y\_floor, x\_floor, :));  pixel\_tr = double(in(y\_floor, x\_ceil, :));  pixel\_bl = double(in(y\_ceil, x\_floor, :));  pixel\_br = double(in(y\_ceil, x\_ceil, :));  % Perform bilinear interpolation  pixel\_interp = (1 - wy) \* ((1 - wx) \* pixel\_tl + wx \* pixel\_tr) + ...  wy \* ((1 - wx) \* pixel\_bl + wx \* pixel\_br);  % Assign the interpolated pixel value to the output image  out(j, i, :) = uint8(pixel\_interp);  end  %================================%  end |

Im\_warp 코드의 구현 부분이다. 위에서 구한 homography의 역행렬을 출력 이미지의 각 픽셀에 대해 적용하여 픽셀 좌표를 출력 이미지에서 입력 이미지로 매핑한다.

현재 출력 이미지 픽셀 좌표 (x, y)에 해당하는 입력 이미지 좌표 x\_in 및 y\_in를 계산한다.

매핑된 입력 이미지 좌표가 입력 이미지 범위 내에 있는지 확인하고 이중 선형 보간법을 사용하픽셀 값 (pixel\_interp)을 계산한다. 보간된 픽셀 값을 출력 이미지의 해당 위치에 저장한다.

**4. 실행결과**

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

예술, 패턴, 모티프, 시각 예술이(가) 표시된 사진

자동 생성된 설명예술, 패턴, 모티프, 페인팅이(가) 표시된 사진

자동 생성된 설명

예술, 스크린샷, 패턴, 모티프이(가) 표시된 사진

자동 생성된 설명

실행 결과를 확인해보면 과제 설명 부분의 그림과 동일하게 명령창에 정보들이 출력되고, image stitching한 결과도 동일함을 알 수 있다.

**5. 전체 소스코드**

|  |
| --- |
| % Construct the matrix A  % 'A'에 대해 SVD를 수행, 가장 작은 특이값에 해당하는 벡터를 추출.  A = zeros(2\*N, 9);  for i = 1:N  x = p1(1, i);  y = p1(2, i);  u = p2(1, i);  v = p2(2, i);  A(2\*i-1, :) = [-x, -y, -1, 0, 0, 0, x\*u, y\*u, u];  A(2\*i, :) = [0, 0, 0, -x, -y, -1, x\*v, y\*v, v];  end  % Perform Singular Value Decomposition (SVD) of A  [~, ~, V] = svd(A);  % Extract the right singular vector corresponding to the smallest singular value  H = reshape(V(:, end), 3, 3)';  End  % Determine the required number of iterations  %======= IMPLEMENT HERE!! =======%  num\_iterations = ceil(log(1 - Ps) / log(1 - (1 - proportion\_of\_outliers)^s));  %================================%  % Update the number of iterations required if we got a lot of inliers  if 1 - num\_inlier/N < proportion\_of\_outliers  proportion\_of\_outliers = 1 - num\_inlier/N;    %======= IMPLEMENT HERE!! =======%  num\_iterations = ceil(log(1 - Ps) / log(1 - (1  proportion\_of\_outliers)^s));  %================================%  % Compute image 'out'  for i = 1:W\_out  for j = 1:H\_out  % Coordinates of output point  x = (i-1)\*R.PixelExtentInWorldX + R.XWorldLimits(1);  y = (j-1)\*R.PixelExtentInWorldY + R.YWorldLimits(1);  %======= IMPLEMENT HERE!! =======%  % Transform coordinates using the inverse homography  p = Hinv \* [x; y; 1];  % Convert to homogeneous coordinates  p = p ./ p(3);  % Map the point to the input image  x\_in = p(1);  y\_in = p(2);  % Check if the mapped point is within the input image bounds  if x\_in >= 1 && x\_in <= W\_in && y\_in >= 1 && y\_in <= H\_in  % Perform bilinear interpolation  x\_floor = floor(x\_in);  y\_floor = floor(y\_in);  x\_ceil = ceil(x\_in);  y\_ceil = ceil(y\_in);  % Interpolation weights  wx = x\_in - x\_floor;  wy = y\_in - y\_floor;  % Retrieve pixel values from the input image  pixel\_tl = double(in(y\_floor, x\_floor, :));  pixel\_tr = double(in(y\_floor, x\_ceil, :));  pixel\_bl = double(in(y\_ceil, x\_floor, :));  pixel\_br = double(in(y\_ceil, x\_ceil, :));  % Perform bilinear interpolation  pixel\_interp = (1 - wy) \* ((1 - wx) \* pixel\_tl + wx \* pixel\_tr) + ...  wy \* ((1 - wx) \* pixel\_bl + wx \* pixel\_br);  % Assign the interpolated pixel value to the output image  out(j, i, :) = uint8(pixel\_interp);  end  %================================%  end |