David Murray
1/27/2019

Computer Vision: Homework 5: Panorama

From GitHub Repository: https://github.com/dcyoung/ImageAlign
Functions:

- detect_features
- describe_features
- estimate_homography
- RANSAC_H
- calc_residuals

```
function [ r1, c1, r2, c2 ] = detect_features( grayImg1, grayImg2 )
%use harris corner detector
    sigma = 2;
    thresh = 0.05;From GitHub Repository
    radius = 2;
    disp = 5;
    [~, r1, c1] = harris(grayImg1, sigma, thresh, radius, disp);
    [~, r2, c2] = harris(grayImg2, sigma, thresh, radius, disp);
end
```

Alternatively:

```
% corner1 = corner(I1_gray,'harris');
% r1 = corner1(:,1);
% c1 = corner1(:,2);
% corner2 = corner(I2_gray,'harris');
% r2 = corner2(:,1);
% c2 = corner2(:,2);
```

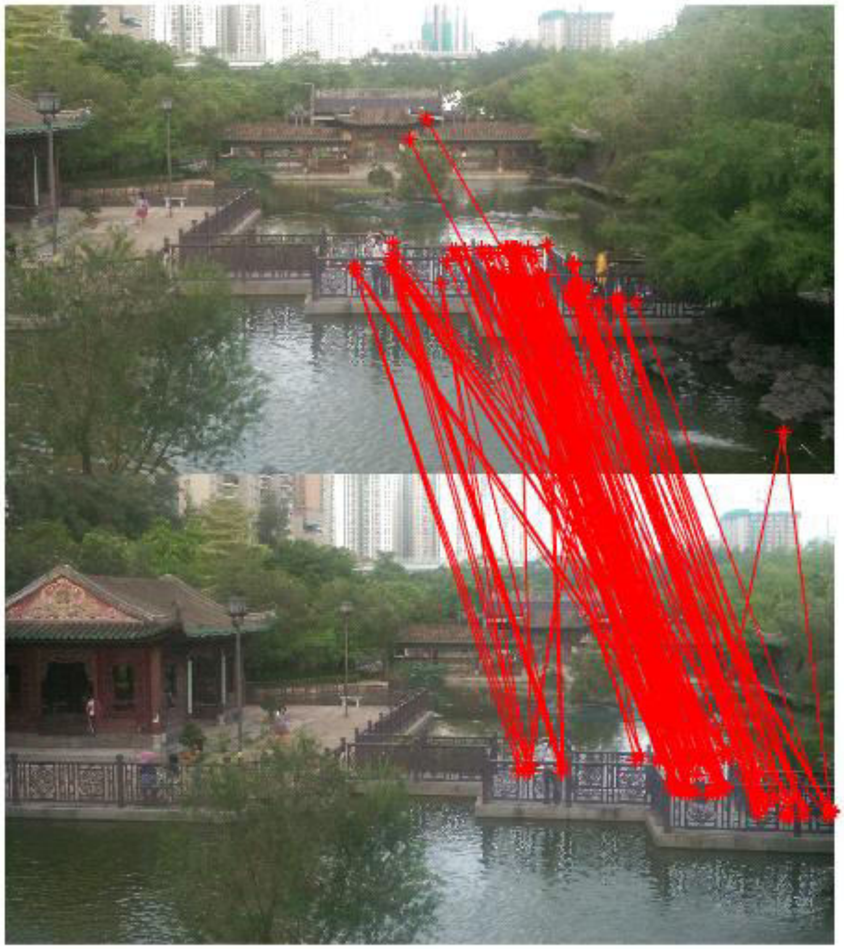
```
function [ featDescriptions ] = describe_features( img, radius, r, c )

    numFeat = length(r); %number of features
    featDescriptions = zeros(numFeat, (2 * radius + 1)^2);

    % matrix with a single 1 in the center and zeros all around it
    padHelper = zeros(2 * radius + 1);
    padHelper(radius + 1, radius + 1) = 1;

    % use the pad Helper matrix to pad the img such that the border values
    % extend out by the radius
    paddedImg = imfilter(img, padHelper, 'replicate', 'full');

    %Extract the neighborhoods around the found features
    for i = 1 : numFeat
        rowRange = r(i) : r(i) + 2 * radius;
        colRange = c(i) : c(i) + 2 * radius;
        neighborhood = paddedImg(rowRange, colRange);
        featDescriptions_test(i,:) = [rowRange colRange];
        flattenedFeatureVec = neighborhood(:);
        featDescriptions(i,:) = flattenedFeatureVec;
    end

    %Normalize all descriptors to have zero mean and unit standard deviation
    featDescriptions_test = zscore(featDescriptions');
```

```matlab
end

function [ H, inlierIndices ] = estimate_homography( img1Feat, img2Feat )
%ESTIMATE_HOMOGRAPHY Summary of this function goes here
%   Detailed explanation goes here

    parameters.numIterations = 150;  %the number of iterations to run
    parameters.subsetSize = 4;  %number of matches to use each iteration
    parameters.inlierDistThreshold = 10;   %the minimum distance for an inlier
    parameters.minInlierRatio = .3;  %minimum inlier ratio required to store a
fitted model

    [H, inlierIndices] = ransac_H(parameters, img1Feat, img2Feat,
@fit_homography, @calc_residuals);

    display('Number of inliers:');
    display(length(inlierIndices));
    display('Average residual for the inliers:')
    display(mean(calc_residuals(H, img1Feat(inlierIndices,:),
img2Feat(inlierIndices,:))));
end

function [ bestFitModel, inlierIndices ] = ransac_H( parameters, x, y, fitModelFxn,
errorFxn )

    [numMatches, ~] = size(x);
    numInliersEachIteration = zeros(parameters.numIterations,1);
    storedModels = {};%zeros(parameters.numIterations,3,3);

    for i = 1 : parameters.numIterations
        %display(['Running ransac Iteration: ', num2str(i)]);

        %select a random subset of points
        subsetIndices = randsample(numMatches, parameters.subsetSize);
        x_subset = x(subsetIndices, :);
        y_subset = y(subsetIndices, :);

        %fit a model to that subset
        model = fitModelFxn(x_subset, y_subset);

        %compute inliers, ie: find all remaining points that are
        %"close" to the model and reject the rest as outliers
        residualErrors = errorFxn(model, x, y);

        %display(['Mean Residual Error: ', num2str(mean(residualErrors))]);
        inlierIndices = find(residualErrors < parameters.inlierDistThreshold);

        %record the number of inliers
        numInliersEachIteration(i) = length(inlierIndices);

        %keep track of any models that generated an acceptable numbers of
        %inliers. This collection can be parsed later to find the best fit
        currentInlierRatio = numInliersEachIteration(i)/numMatches;
        if currentInlierRatio >=  parameters.minInlierRatio
        %if numInliersEachIteration(i) >= max(numInliersEachIteration)
            %re-fit the model using all of the inliers and store it
```

```
            x_inliers = x(inlierIndices, :);
            y_inliers = y(inlierIndices, :);
            storedModels{i} = fitModelFxn(x_inliers, y_inliers);
        end
    end
    %display(storedModels);
    %display(numInliersEachIteration);

    %retrieve the model with the best fit (highest number of inliers)
    bestIteration = find(numInliersEachIteration ==
max(numInliersEachIteration));
    bestIteration = bestIteration(1); %incase there was more than 1 with same value
    bestFitModel = storedModels{bestIteration};

    %recalculate the inlier indices for all points, this was done once before
    %when calculting this model, but it wasn't stored for space reasons.
    %Recalculate it now so that it can be returned to the caller
    residualErrors = errorFxn(bestFitModel, x, y);
    inlierIndices = find(residualErrors < parameters.inlierDistThreshold);
end

function residuals = calc_residuals(F, matches)
    numMatches = size(matches,1);
    L = (F * [matches(:,1:2) ones(numMatches,1)]')'; % transform points from
    % the first image to get epipolar lines in the second image

    % find points on epipolar lines L closest to matches(:,3:4)
    L = L ./ repmat(sqrt(L(:,1).^2 + L(:,2).^2), 1, 3); % rescale the line
    distances = sum(L .* [matches(:,3:4) ones(numMatches,1)],2); %distances from
each pt to its line

    residuals = abs(distances);
end

% MAIN
addpath('C:\Users\Dave\Desktop\Computer Vision\Project 5');
imnames={'IMAG4688.jpg','IMAG4689.jpg'};

I1=imread(imnames{1});
I2=imread(imnames{2});
I1_gray = rgb2gray(I1);
I2_gray = rgb2gray(I2);

% Find row and column locations of corner points in images
[r1, c1, r2, c2] = detect_features(I1_gray, I2_gray);
% corner1 = corner(I1_gray,'harris');
% r1 = corner1(:,1);
% c1 = corner1(:,2);
% corner2 = corner(I2_gray,'harris');
% r2 = corner2(:,1);
% c2 = corner2(:,2);

% Create feature descriptions for each point.  Find corner points in the
% 'neighborhood' radius of each point.
numMatches = 150;
neighborhoodRadius = 20;
```

```
featDescriptions_1 = describe_features(I1_gray, neighborhoodRadius, r1, c1);
featDescriptions_2 = describe_features(I2_gray, neighborhoodRadius, r2, c2);

distances = pdist2(featDescriptions_1, featDescriptions_2);
[~,distance_idx] = sort(distances(:), 'ascend');
BestMatches = distance_idx(1:numMatches);
[~,distance_idx] = sort(distances(:), 'ascend');
[img1_matchedFeature, img2_matchedFeature] =
ind2sub(size(distances),BestMatches);

match_r1 = r1(img1_matchedFeature);
match_c1 = c1(img1_matchedFeature);
match_r2 = r2(img2_matchedFeature);
match_c2 = c2(img2_matchedFeature);
% refined lists of matching points in both pictures
xy1 = [match_c1, match_r1, ones(numMatches,1)]';
xy2 = [match_c2, match_r2, ones(numMatches,1)]';
xx = affine_fit(xy1,xy2);
figure(1)
visualize_match(xy1,xy2,I1,I2);
title('Matching Points without RANSAC')

xy1 = xy1';
xy2 = xy2';
% RANSAC
[H, inlierIndices] = estimate_homography(xy1,xy2);
numMatches = length(inlierIndices);
match_c1_H = match_c1(inlierIndices);
match_c2_H = match_c2(inlierIndices);
match_r1_H = match_r1(inlierIndices);
match_r2_H = match_r2(inlierIndices);
% New Matching Points
xy3 = [match_c1_H, match_r1_H, ones(numMatches,1)]';
xy4 = [match_c2_H, match_r2_H, ones(numMatches,1)]';
xx_ransac = affine_fit(xy3,xy4);
%figure(2)
%visualize_match(xy3,xy4,I1,I2);
%title('Matching Points with RANSAC')

%T = maketform('projective',xx_ransac);
T = projective2d(H);
img1Transformed = imwarp(I1,T);

%figure(3)
%compositeImg1 = stitch(img1Transformed,I2,H);
%imshow(compositeImg1);
%title('Projective Transform')

figure(4)
%[wholeImg,NewImage,offsets]=draw_align_image(H,img1Transformed,I2);
save('DAVID_MURRAY_Project5');
```

Matching Points without RANSAC



Matching Points with RANSAC