Emerson Boyd
EECE 5639
12/12/17

Computer Vision Extra Credit Project - Stereo Images

Abstract

The purpose of this experiment was to find feature correspondences between two images, either through corner detection or by finding SIFT features. We implemented the feature matching between two images using a Harris corner detector, alongside non-max suppression and normalized cross-correlation to get a more sparse set of correspondences. Then, the result of of the correspondence locations are used to calculate the fundamental matrix between the two images. The fundamental matrix is then used to develop an accurate set of inlier correspondences from the correspondences found earlier. Then, the horizontal and vertical disparities between the two images are calculated and displayed to the user. We analyzed the results of our work to determine its effectiveness.

Description of Algorithms

The general correspondence locating is performed in the method “getCorners”. The first algorithm used was the Prewitt edge detection algorithm. Depending on whether or not the edge detector is vertical or horizontal, the filter would search for different edges, however, the algorithm behind them is the same. For both, the filter was a 3x3 matrix, and it calculated the value of the current pixel by taking the row or column of pixels after the current pixel and subtracted the row or column of pixels before the current pixel.

After coming up with horizontal and vertical Prewitt values for each pixel, the C matrix was calculated. The C matrix was a 2x2 matrix consisting of the vertical Prewitt value squared, the horizontal Prewitt value squared, and the vertical and horizontal Prewitt values multiplied together. After creating the C matrix, eigenvalues were calculated using built-in MatLab functions. Then, the determinant of the C matrix was calculated which is the R value for the given pixel. The R values were separated by a given threshold with those above a certain threshold would be given a 1 value, and values below the threshold would be given a 0 value.

The magnitude and orientation of the vertical and horizontal Prewitt values were also calculated. The magnitude was the square root of the horizontal Prewitt squared added to the vertical Prewitt squared. The orientation was the inverse tangent of the horizontal Prewitt divided by the vertical Prewitt. For orientation values that ended up being NaN, the pixel was set to 90.

Finally, the non-maximum suppression was used to come up with the local maxima within a given range of corners. Depending on the orientation of the pixel, different local maxima would be checked around the pixel. For example, if the orientation was 0, then the values to the left and right of the pixel would be used to find the local maxima. If there were values greater than the current pixel value, then the value of the pixel was set to 0, otherwise, the pixel value would stay the same.

The non-maximum suppression was then used to calculate the normalized cross correlation. The normalized cross correlation compared the corners from the non-max suppression in the two separate images to see which ones matched up the best given a range. A 11x11 filter size was used to calculate normalized cross correlation values. The values that ended up with the highest values were used in the correlation values matrix for x and y. The correlation values gave the points that correlated most with the corners. For example, in the corrX matrix, at (10,10), it would give the X value in the second image that correlated the most with the corner in the first image. Combined with the corrY matrix, each corner pixel in the first image had a corner pixel in the second image that correlated the most and would be used for future calculations.

Using the resulting corrX and corrY matrices, the fundamental matrix between the two images was calculated using MATLAB's built-in function "estimateFundamentalMatrix". The function takes point matches in a different form that how corrX and corrY relate, so a function "getMatchedPoints" was created to create point matches in the correct format for the "estimateFundamentalMatrix" function. Once the fundamental matrix was found, we estimated the corner match inliers by performing RANSAC. For each image correspondence, we assumed the property "p2' * F * p1 = 0". We considered all correspondences that did not fall within a certain threshold close to zero as outliers. Then, the list of inlier correspondences was returned from the ransac method.

The disparity maps between the two images are calculated within our "getDisparityMap" function using the "disparity" method built into MATLAB. However, this built-in disparity method only calculates the horizontal disparity between two images (the disparity across columns). So, in order to calculate vertical disparity as well, we had to rotate the images 90 degrees, and then recalculate the disparity between the two images. The values for the disparity matrices were then normalized between 0 and 255 so that they could be shown accurately with "imshow".

A full visualization of algorithms can be seen in the flowchart in Appendix A1.

Experiments, Values of Parameters Used, and Observations

These experiments were run for two different pairs of stereo images. First, using the ideal R threshold and NCC threshold parameters determined in Project 2, correspondences between both images were found and plotted. The optimal R threshold is between 50,000,000

and 100,000,000, which helps determine if a pixel group is a corner or not. The optimal NCC threshold was found to be 1.04, which helps determine if there's a strong correlation between two corners in the two images. The result of this feature correspondence detection is shown below.

Then, as stated before, using the property "p2' * F * p1 = 0", the correspondences were either labelled as "inliers" or "outliers" depending on how close to zero the result of that calculation actually was. it was necessary to find a proper threshold, close to 0, for which to separate the inliers and outliers. This threshold is labelled as the "fThreshold" in the function "calculateNumCorrectPoints". The first fThreshold value tested, 0.002, was found to be sufficient for separating inliers from outliers.

Finally, once the correspondence inliers were determined, the disparity maps of the two images were plotted and are shown below.

"Image Set 1" refers to the "cast" pair of images provided on Blackboard. "Image Set 2" refers to the "Cones" pair of images provided on Blackboard.

Below are the correspondences found in Image Set 1 and Image Set 2 as a result of Harris corner detection, non-max suppression, and NCC.

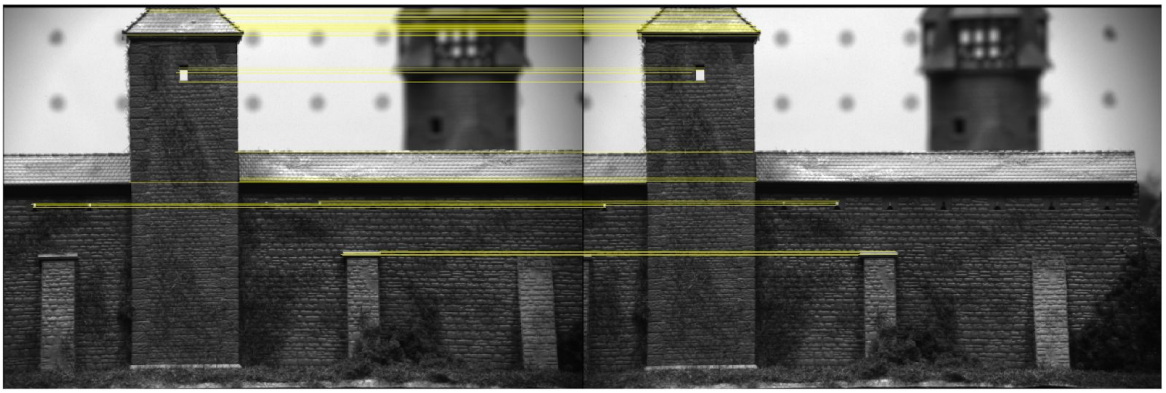
Correspondences from Image Set 1

Correspondences from Image Set 2

In both of the image sets, we see a correlation between corner “features” detected in each image.

Almost all of the correspondences from Image Set 1 appear to be inliers. There are no lines that jump out as being skewed relative to the others. However, in Image Set 2, there are a couple lines in the bottom third of the images that are diagonal and not parallel to the other lines. It is for this reason that we cannot fully trust the results of correspondence detection and need a method to detect correspondence inliers.

Below are the inlier correspondences of Image Set 1 and Image Set 2 using fThreshold value of 0.002.



Inlier Correspondences from Image Set 1

Inlier Correspondences from Image Set 2

In both image sets, we see that all lines match up two features that are actually identical. There are no outlier correspondences remaining. The results from this method demonstrate that we can trust the property “p2’ * F * p1 = 0” to hold true after calculating for the fundamental matrix between two stereo images, and that the property can be used to separate inliers from outliers.

Below are the horizontal and vertical disparity maps of Image Set 1.

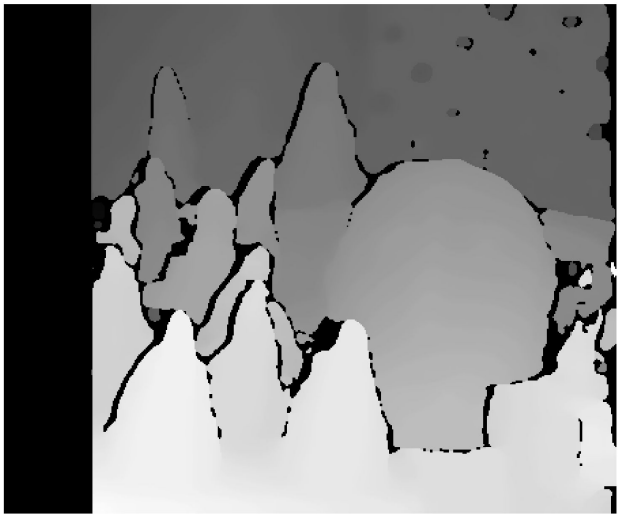
Horizontal Disparity of Image Set 1

The horizontal disparity makes sense when comparing the two images side-by-side. The white patch in the center represents the facade of the castle translated from left to right. It is white (with a value of 255) because it has the greatest noticeable horizontal disparity between the two images. The gray patches in the top right of the image are also parts of the building that didn't translate as much from left to right because they are farther back from the camera than the main facade. The black bar on the left of the disparity image represents the parts of the images that only lie in one of the images, rather than in both. There is no horizontal disparity for this section because there is no correlation of these parts of the images from one image to the next.

Vertical Disparity of Image Set 1

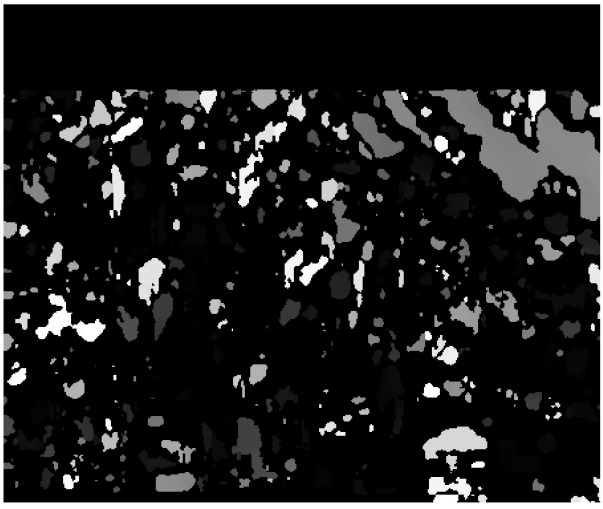
In this image set, we are not very interested in the vertical disparity between the two stereo images. This is because, when looking at the images side-by-side, there is not much visible vertical difference between the two images to the human eye. It is possible that the disparity function returned very small values for the vertical disparity, but upon normalizing the values from 0-255, the colors varied a lot from black to white. This would be an explanation for the seemingly “random” patches of color in the vertical disparity image.

Below are the horizontal and vertical disparity maps of Image Set 2.



Horizontal Disparity of Image Set 2

The parts of the horizontal disparity that are closer to white indicate a larger horizontal disparity, while the parts of the disparity that are closer to black indicate less of a horizontal disparity. There is a gradual gradient from lighter to darker shades of gray as the objects go further into the background. We thus conclude that there is a higher horizontal translation of the objects in the front of the images than there is of the objects in the back of the image, which makes sense. The black bar that appears in the horizontal disparity appears for the same reason as it does in the horizontal disparity of Image Set 1. Because there is a horizontal translation from the first image in Image Set 2 to the second image, the parts of the disparity that cannot be mapped are the sections of the first image that don't appear in the second image. It is for this reason that there is a black bar on the left.



Vertical Disparity of Image Set 2

The vertical disparity between the two images is similar to the vertical disparity in Image Set 1. There are no real points of significance, mostly noise, as there isn't a lot of vertical translation between the two images.

Conclusion

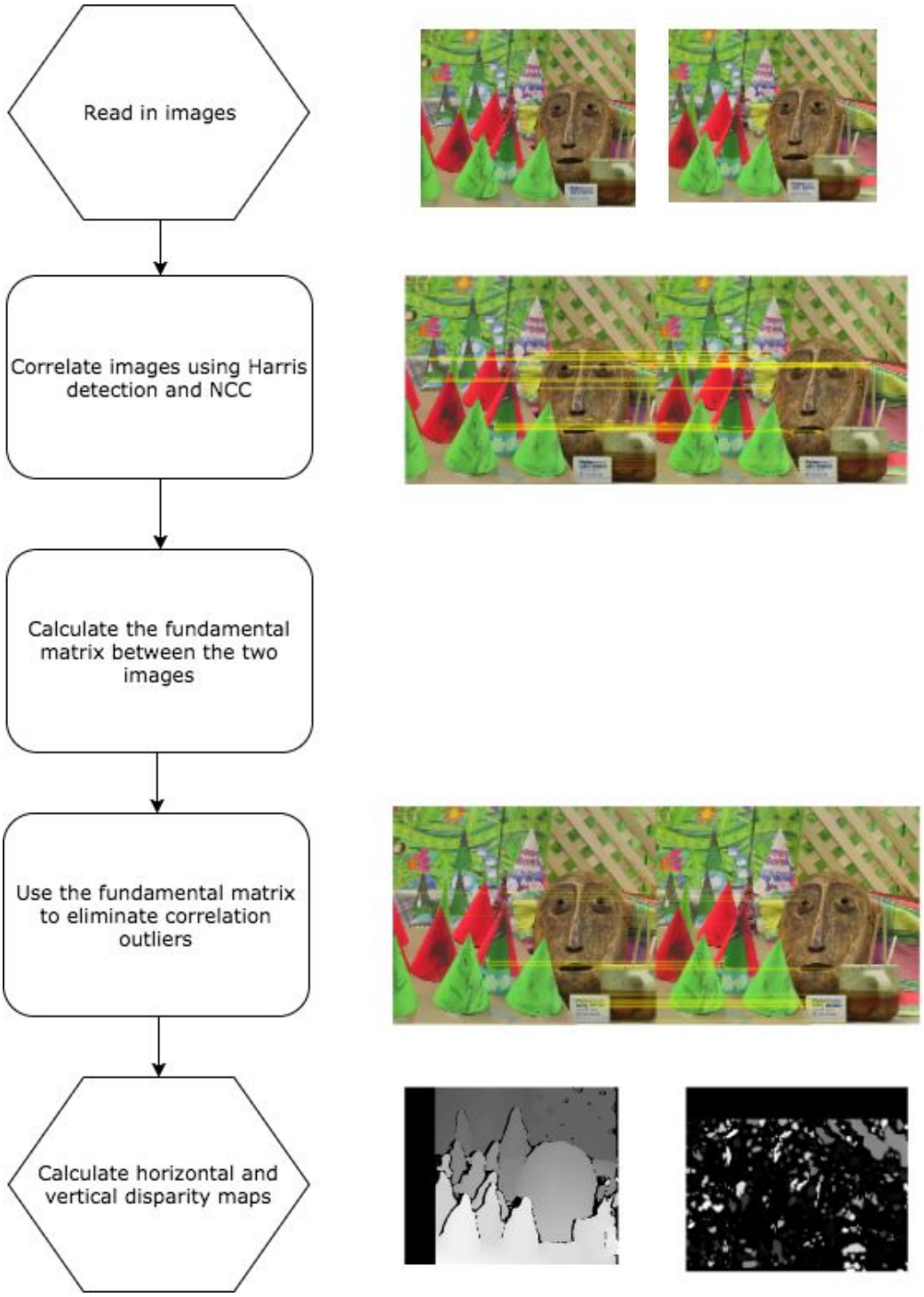
This project is very similar in structure to Project 2, which uses Harris corner detection, non-max suppression and NCC to estimate the homography matrix between two images. However, in Project 2, RANSAC was used to estimate the homography matrix, while a different algorithm was used in this project to estimate the fundamental matrix. RANSAC takes about a minute to compute the homography, as it requires several random samplings of correspondences, while estimating the fundamental matrix only takes a matter of seconds. Therefore, this project is much more viable for applications that require real-time vision processing.

Stereo image disparity maps can be used for many powerful algorithms if the findings from this experiment are applied in greater depth. For example, by analyzing the difference in disparity values, one can determine the depth of the pixels in each image. Depth sensing has many applications in today's modern camera technology. It seems as if all cell phones now use multiple camera sensors to detect image depth and create portrait photos. The backgrounds of photos can be blurred to the user's liking after the image has been taken. This "post-editing" process could act as a future application in MATLAB using the findings from this experiment. One could attempt to take a stereo image pair as input and blur the background while leaving the foreground intact. The amount of depth that defines what the background is could alter, as well, by allowing the user to use a slider to change how far into the background pixels get blurred.

Overall, there are many potential directions one could take when analyzing horizontal and vertical image disparity maps. There are several real-world uses that can be applied using these maps. It has been fun working with algorithms that true computer vision scientists use today.

Appendix

A1. Flow Chart



```
clc
clear all
close all

imageType = 2;

if (imageType == 1)
    folderName = 'Images/cast/';
    fileNamePrefix = 'cast-';
    fileName1 = 'left';
    fileName2 = 'right';
    fileNameExtension = '.jpg';
    rThreshold = 50000000;
    nccThreshold = 1.04;
elseif (imageType == 2)
    folderName = 'Images/Cones/';
    fileNamePrefix = 'Cones_im';
    fileName1 = '2';
    fileName2 = '6';
    fileNameExtension = '.JPG';
    rThreshold = 100000000;
    nccThreshold = 1.04;
else
    error('Cannot select image with the given image type.')
end

fullFileName1 = strcat(folderName, fileNamePrefix, fileName1,
 fileNameExtension);
fullFileName2 = strcat(folderName, fileNamePrefix, fileName2,
 fileNameExtension);
image1 = imread(fullFileName1);
image2 = imread(fullFileName2);

% we need doubles so that we can perform necessary calculations later
rgbImgList(:, :, :, 1) = double(image1);
rgbImgList(:, :, :, 2) = double(image2);
grayImgList(:, :, 1) = double(rgb2gray(image1));
grayImgList(:, :, 2) = double(rgb2gray(image2));

% Harris Corner Detection, Non-Max Suppression and NCC
[corrX, corrY] = getCorners(grayImgList(:,:,1), grayImgList(:,:,2),
 rThreshold, nccThreshold);

plotCorrelations(rgbImgList(:,:,:,1), rgbImgList(:,:,:,2), corrX,
 corrY);

% Calculate the fundamental matrix
[matchedPoints1, matchedPoints2] = getMatchedPoints(corrX, corrY);
F = estimateFundamentalMatrix(matchedPoints1, matchedPoints2);

% Perform RANSAC to find the inliers
sizeImageX = size(grayImgList(:, :, 1), 1);
```

```
sizeImageY = size(grayImgList(:, :, 1), 2);
[inliersX, inliersY] = ransac(F, matchedPoints1, matchedPoints2,
 sizeImageX, sizeImageY);

figure
plotCorrelations(rgbImgList(:,:,:,1), rgbImgList(:,:,:,2), inliersX,
 inliersY);

grayImgList = uint8(grayImgList);
[disparityMapHorizontal, disparityMapVertical] =
 getDisparityMap(grayImgList(:,:,1), grayImgList(:,:,2));

figure
imshow(disparityMapHorizontal, [0 255]);
figure
imshow(disparityMapVertical, [0 255]);
```

Published with MATLAB® R2017a

```
function [corrX, corrY] = getCorners(image1, image2, rThreshold,
 nccThreshold)
    verticalPrewittFilter = [-1 0 1;
                             -1 0 1;
                             -1 0 1];

    horizontalPrewittFilter = [-1 -1 -1;
                               0 0 0;
                               1 1 1];

    % Harris Corner Detection
    verticalPrewitt1 = imfilter(image1,
 verticalPrewittFilter, 'replicate');
    horizontalPrewitt1 = imfilter(image1,
 horizontalPrewittFilter, 'replicate');

    verticalPrewitt2 = imfilter(image2,
 verticalPrewittFilter, 'replicate');
    horizontalPrewitt2 = imfilter(image2,
 horizontalPrewittFilter, 'replicate');

    R1 = CMatrix(verticalPrewitt1, horizontalPrewitt1);
    R2 = CMatrix(verticalPrewitt2, horizontalPrewitt2);

    magnitude1 = sqrt(horizontalPrewitt1.^2 + verticalPrewitt1.^2);
    orientation1 = atand(horizontalPrewitt1./verticalPrewitt1);
    magnitude2 = sqrt(horizontalPrewitt2.^2 + verticalPrewitt2.^2);
    orientation2 = atand(horizontalPrewitt2./verticalPrewitt2);

    R1(R1 < rThreshold) = 0;
    R1(R1 >= rThreshold) = 255;
    orientation1(isnan(orientation1)) = 90; % get rid of bad
 orientation values

    R2(R2 < rThreshold) = 0;
    R2(R2 >= rThreshold) = 255;
    orientation2(isnan(orientation2)) = 90; % get rid of bad
 orientation values

    % Non-Max Suppression
    nonMaxSuppress1 = nonMaxSuppression(orientation1, magnitude1, R1);
    nonMaxSuppress2 = nonMaxSuppression(orientation2, magnitude2, R2);

    % NCC
    [corrX, corrY] = normalizedCrossCorrelation(image1, image2,
 nonMaxSuppress1, nonMaxSuppress2, nccThreshold);
end
```

Published with MATLAB® R2017a

```
function [DetMatrix] = CMatrix(verticalPrewitt, horizontalPrewitt)
    verticalHorizontalPrewitt = verticalPrewitt .* horizontalPrewitt;

    verticalPrewittSquared = verticalPrewitt .* verticalPrewitt;

    horizontalPrewittSquared = horizontalPrewitt .* horizontalPrewitt;

    boxFilterSize = 7;
    boxFilter = ones(boxFilterSize, boxFilterSize)./(boxFilterSize *
 boxFilterSize);

    boxFilterVerticalHorizontalPrewitt =
 imfilter(verticalHorizontalPrewitt, boxFilter);
    boxFilterVerticalPrewitt = imfilter(verticalPrewittSquared,
 boxFilter);
    boxFilterHorizontalPrewitt = imfilter(horizontalPrewittSquared,
 boxFilter);

    sizeMatrix = size(verticalHorizontalPrewitt);
    sizeX = sizeMatrix(1);
    sizeY = sizeMatrix(2);

    DetMatrix = zeros(sizeX, sizeY);
    for i = 1:sizeX
        for j = 1:sizeY
            C = [boxFilterVerticalPrewitt(i, j)
 boxFilterVerticalHorizontalPrewitt(i, j);
                boxFilterVerticalHorizontalPrewitt(i, j)
 boxFilterHorizontalPrewitt(i, j)];
            D = eig(C);
            det = D(1) * D(2);
            trace = D(1) + D(2);
            DetMatrix(i,j) = det - (0.05 * (trace^2));
        end
    end
end
```

Published with MATLAB® R2017a

```
function [outputMatrix] = nonMaxSuppression(orientation, magnitude, R)
    totalSize = size(orientation);
    sizeX = totalSize(1);
    sizeY = totalSize(2);

    outputMatrix = R;

    for i = 2:sizeX-1
        for j = 2:sizeY-1
            angle = closestAngle(orientation(i,j));
            if (angle == 0)
                compareIndex1 = magnitude(i, j-1);
                compareIndex2 = magnitude(i, j+1);
            elseif (angle == 45)
                compareIndex1 = magnitude(i-1, j+1);
                compareIndex2 = magnitude(i+1, j-1);
            elseif (angle == 90)
                compareIndex1 = magnitude(i-1, j);
                compareIndex2 = magnitude(i+1, j);

            elseif (angle == 135)
                compareIndex1 = magnitude(i-1, j-1);
                compareIndex2 = magnitude(i+1, j+1);
            end
            if (or((magnitude(i,j) < compareIndex1),(magnitude(i,j) <
 compareIndex2)))
                outputMatrix(i,j) = 0;
            else
                outputMatrix(i,j) = R(i,j);
            end
        end
    end
end
```

Published with MATLAB® R2017a

```
function [angle] = closestAngle(orientation)
    compareArray = [0 45 90 135];
    if (abs(mod(orientation,180)) >= (180-22.5))
        angle = 0;
    else
        for i = 1:4
            if (abs(mod(orientation,180)-compareArray(i)) <= 22.5)
                angle = compareArray(i);
            end
        end
    end
end
```

Published with MATLAB® R2017a

```
function [corrX, corrY] = normalizedCrossCorrelation(grayImg1,
 grayImg2, nonMaxSuppress1, nonMaxSuppress2, nccThreshold)
    nccBoxFilterSize = 11;

    totalImageSize1 = size(grayImg1);
    imageHeight1 = totalImageSize1(1);
    imageWidth1 = totalImageSize1(2);
    totalImageSize2 = size(grayImg2);
    imageHeight2 = totalImageSize2(1);
    imageWidth2 = totalImageSize2(2);

    nccImg1 = zeros(size(grayImg1));
    nccImg2 = zeros(size(grayImg2));

    halfNccBoxFilterSizeFloored = floor(nccBoxFilterSize / 2);
    xMin = nccBoxFilterSize - halfNccBoxFilterSizeFloored;
    yMin = xMin;
    xMax1 = imageHeight1 - halfNccBoxFilterSizeFloored;
    yMax1 = imageWidth1 - halfNccBoxFilterSizeFloored;
    xMax2 = imageHeight2 - halfNccBoxFilterSizeFloored;
    yMax2 = imageWidth2 - halfNccBoxFilterSizeFloored;

    for x = xMin:xMax1
        for y = yMin:yMax1
            if (nonMaxSuppress1(x, y) == 255)
                xRange = x - halfNccBoxFilterSizeFloored : x +
 halfNccBoxFilterSizeFloored;
                yRange = y - halfNccBoxFilterSizeFloored : y +
 halfNccBoxFilterSizeFloored;
                grayImg1Snippet = grayImg1(xRange, yRange);
                nccImg1(x, y) = norm(grayImg1Snippet);
            end
        end
    end

    for x = xMin:xMax2
        for y = yMin:yMax2
            if (nonMaxSuppress2(x, y) == 255)
                xRange = x - halfNccBoxFilterSizeFloored : x +
 halfNccBoxFilterSizeFloored;
                yRange = y - halfNccBoxFilterSizeFloored : y +
 halfNccBoxFilterSizeFloored;
                grayImg2Snippet = grayImg2(xRange, yRange);
                nccImg2(x, y) = norm(grayImg2Snippet);
            end
        end
    end

    corrX = zeros(size(grayImg1));
    corrY = zeros(size(grayImg1));

    for x1 = xMin:xMax1
```

```
        for y1 = yMin:yMax1
            if (nonMaxSuppress1(x1,y1) == 255)
                x1Range = x1 - halfNccBoxFilterSizeFloored : x1 +
halfNccBoxFilterSizeFloored;
                y1Range = y1 - halfNccBoxFilterSizeFloored : y1 +
halfNccBoxFilterSizeFloored;
                grayImg1Snippet = grayImg1(x1Range, y1Range);

                c = zeros(size(grayImg2));

                for x2 = xMin:xMax2
                    for y2 = yMin:yMax2
                        if (nonMaxSuppress2(x2,y2) == 255)
                            x2Range = x2 -
halfNccBoxFilterSizeFloored : x2 + halfNccBoxFilterSizeFloored;
                            y2Range = y2 -
halfNccBoxFilterSizeFloored : y2 + halfNccBoxFilterSizeFloored;
                            grayImg2Snippet = grayImg2(x2Range,
y2Range);

                            ccPoint = grayImg1Snippet .*
grayImg2Snippet;
                            nccPoint = sum(sum(ccPoint)) /
(nccImg1(x1,y1) * nccImg2(x2,y2));
                            c(x2,y2) = nccPoint;
                        end
                    end
                end

                [cMaxVal, cMaxIndex] = max(c(:));
                if (cMaxVal > nccThreshold)
                    [cMaxX, cMaxY] = ind2sub(size(c), cMaxIndex);
                    corrX(x1,y1) = cMaxX;
                    corrY(x1,y1) = cMaxY;
                end
            end
        end
    end
end
```

Published with MATLAB® R2017a

```
function [] = plotCorrelations(grayImg1, grayImg2, corrX, corrY)

totalImageSize1 = size(grayImg1);
imageHeight1 = totalImageSize1(1);
imageWidth1 = totalImageSize1(2);

grayImgFull = horzcat(grayImg1, grayImg2);

totalImageSizeFull = size(grayImgFull);
imageHeightFull = totalImageSizeFull(1);
imageWidthFull = totalImageSizeFull(2);

imshow(grayImgFull/255)
hold on
for x = 1:imageHeight1
    for y = 1:imageWidth1
        if ((corrX(x,y) ~= 0) && (corrY(x,y) ~= 0))
            img2XCoordinate = imageWidth1 + corrY(x,y);
            line([y img2XCoordinate], [x
 corrX(x,y)], 'Color', 'yellow')
        end
    end
end

axis([1 imageWidthFull 1 imageHeightFull])
daspect([1 1 1])

hold off

end
```

Published with MATLAB® R2017a

```
function [ matchedPoints1, matchedPoints2 ] = getMatchedPoints(corrX,
 corrY)

numMatchedPoints = sum(sum(corrX > 0));
matchedPoints1 = zeros(numMatchedPoints, 2);
matchedPoints2 = zeros(numMatchedPoints, 2);

sizeX = size(corrX, 1);
sizeY = size(corrX, 2);

matchedPointsCount = 0;
for i = 1:sizeX
    for j = 1:sizeY
        if (corrX(i,j) > 0 && corrY(i,j) > 0)
            matchedPointsCount = matchedPointsCount + 1;
            matchedPoints1(matchedPointsCount, :) = [i j];
            matchedPoints2(matchedPointsCount, :) = [corrX(i,j)
 corrY(i,j)];
        end
    end
end

end
```

Published with MATLAB® R2017a

```
function [inliersX, inliersY] = ransac(F, matchedPoints1,
 matchedPoints2, imageSizeX, imageSizeY)

    [inliers1, inliers2, fValue] = calculateNumCorrectPoints(F,
 matchedPoints1, matchedPoints2);

    totalInliersSize = size(inliers1);
    inliersLength = totalInliersSize(1);

    inliersX = zeros(imageSizeX, imageSizeY);
    inliersY = zeros(imageSizeX, imageSizeY);

    for k = 1:inliersLength
        x = inliers1(k, 1);
        y = inliers1(k, 2);

        inliersX(x,y) = inliers2(k,1);
        inliersY(x,y) = inliers2(k,2);
    end
end
```

Published with MATLAB® R2017a

```
function [inliers1, inliers2, fValues] = calculateNumCorrectPoints(F,
 points1, points2)

numPoints = size(points1, 1);
fValues = zeros(numPoints, 1);

fThreshold = .002;

j = 0;
for i = 1:numPoints
    p1 = [points1(i, :) 1];
    p2 = [points2(i, :) 1];

    fVal = p2 * F * p1';
    fValues(i, :) = fVal;

    if (abs(fVal) < fThreshold)
        j = j + 1;
        inliers1(j, :) = points1(i, :);
        inliers2(j, :) = points2(i, :);
    end
end
```

Published with MATLAB® R2017a

```
function [disparityMapHorizontal, disparityMapVertical] =
 getDisparityMap(image1, image2)

    disparityMapHorizontal = disparity(image1,
 image2, 'Method', 'SemiGlobal');

    rotatedImage1 = imrotate(image1, 90);
    rotatedImage2 = imrotate(image2, 90);
    disparityMapVertical = disparity(rotatedImage1,
 rotatedImage2, 'Method', 'SemiGlobal');

    disparityMapVertical = imrotate(disparityMapVertical, -90);

    % normalize the disparity map between 0-255
    disparityMapHorizontal(disparityMapHorizontal < 0) = 0;
    disparityMapVertical(disparityMapVertical < 0) = 0;
    disparityMapHorizontal = normalizeMatrix(disparityMapHorizontal,
 0, 255);
    disparityMapVertical = normalizeMatrix(disparityMapVertical, 0,
 255);
end
```

Published with MATLAB® R2017a

```
function [ normalizedMatrix ] = normalizeMatrix(matrix, normMin,
 normMax)

    matrixMin = min(min(matrix));
    matrixMax = max(max(matrix));
    matrixRange = matrixMax - matrixMin;

    matrix0To1 = (matrix - matrixMin) / matrixRange;

    normRange = normMax - normMin;
    normalizedMatrix = (matrix0To1 * normRange) + normMin;

end
```

Published with MATLAB® R2017a