Open videorectification.m in the Editor **Run in the Command Window**

Image Rectification

Image rectification [2,3] is the process of projecting multiple images onto a common image plane and aligning their coordinate systems. This is a useful procedure in many computer vision applications, especially when merging images to build a composite map from many small images. It is also applicable in stereo vision, as the 2-D stereo correspondence problem is reduced to a 1-D problem when rectified image pairs are used. Image rectification is a prerequisite step for the Stereo Vision Demo.

This demo uses the Video and Image Processing Blockset™ to compute the rectification of two uncalibrated images where the camera intrinsics are unknown. The method implemented follows the approach described in "Quasi-Euclidean Uncalibrated Epipolar Rectification," by L. Irsara and A. Fusiello [1]. The algorithm requires a set of point correspondences between the two images, which are fed into a non-linear optimization to find the best rectifying homographies [2]. Each of these steps is presented in detail below, and more details on the algorithm are available from the references.

This demo requires the Optimization Toolbox to perform the quasi-Euclidean rectification using `LSQNONLIN`.

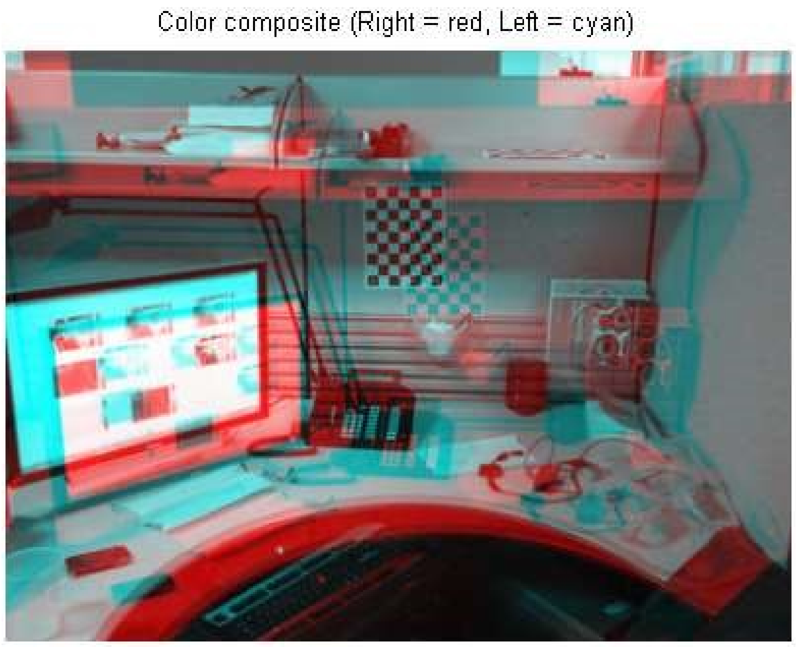
Contents

Step 1. Read stereo image pair

Here we read in two color images of the same scene taken from different positions. We convert them to grayscale since colors are not necessary for the matching process. For this we use the `ImageDataTypeConverter` and the `ColorSpaceConverter` System objects.

Below we show both images side by side, and we produce a color composite to demonstrate the pixel-wise differences between them. There is an obvious offset between the images in both dimensions. The goal of rectification is to transform the images, aligning them such that corresponding features in each image will appear on the same row. The corners of the monitor, for example, are features that would end up on the same row in the rectified images.

```
hIdtc = video.ImageDataTypeConverter();
hCsc = video.ColorSpaceConverter('Conversion','RGB to intensity');
leftI3chan = step(hIdtc,imread('viprectification_deskLeft.png'));
leftI = step(hCsc,leftI3chan);
rightI3chan = step(hIdtc,imread('viprectification_deskRight.png'));
rightI = step(hCsc,rightI3chan);

% Display
figure(1), clf;
montage(cat(4,leftI,rightI),'size',[1 2]);
text(size(leftI,2)/2,0,'Left image',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
text(size(leftI,2)+size(rightI,2)/2,0,'Right image',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');

figure(2), clf;
Composite = cat(3,rightI,leftI,leftI);
imshow(Composite), title('Color composite (Right = red, Left = cyan)');
```



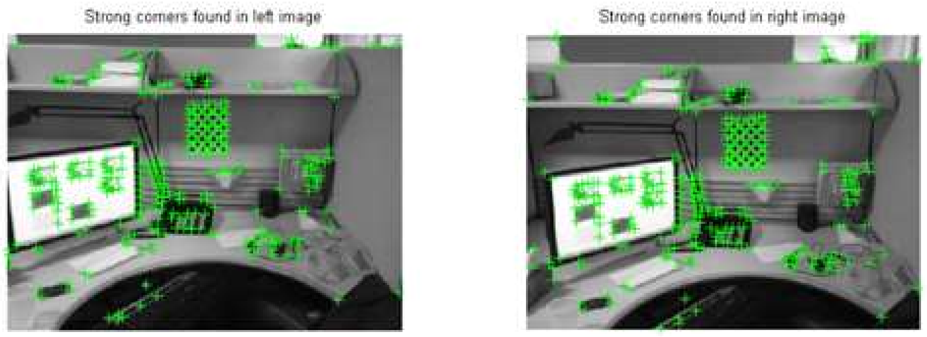
Step 2. Collect salient points from each image

The rectification process requires a set of point correspondences between the two images. To generate these correspondences, we collect points of interest from both images, then choose potential matches between them.

In this step we produce these candidate points for each image. To have the best chance that these points will have correspondences in the other image, we want points around salient image features such as corners. We use the `CornerDetector` System object with its default detection method of Harris corners. The detected points in both images are shown below. Observe how many of them cover the same image features, as desired, such as the corners of the monitor and those of the checkerboard pattern.

```
harris = video.CornerDetector( 'MaximumCornerCount', 1000, ...
                               'CornerThreshold', 1e-4, ...
                               'NeighborhoodSize', [9 9] );
pointsLeft = flipud(step(harris, leftI))+1;
pointsRight = flipud(step(harris, rightI))+1;
% Trim point lists to minimum size.
pointsLeft = pointsLeft(:, 1:find(all(pointsLeft==1,1),1) - 1);
pointsRight = pointsRight(:, 1:find(all(pointsRight==1,1),1) - 1);
```

```
figure(1), clf;
subplot(121), imshow(leftI), hold on;
plot(pointsLeft(1,:),pointsLeft(2,:),'g+');
title('Strong corners found in left image');
subplot(122), imshow(rightI), hold on;
plot(pointsRight(1,:),pointsRight(2,:),'g+');
title('Strong corners found in right image');
```


Strong corners found in left image
Strong corners found in right image

Step 3. Select correspondences between points

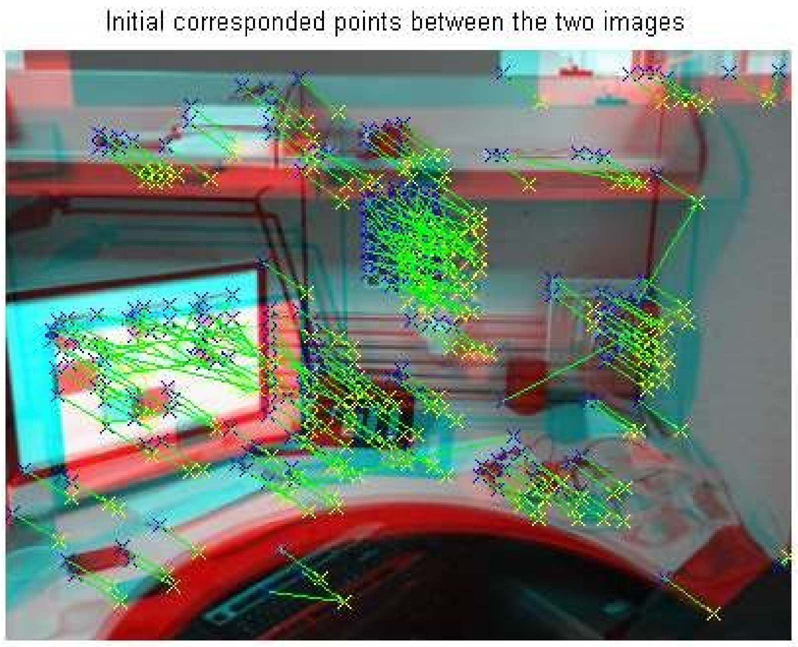
Next we determine correspondences between the left and right image points. For each point, we extract a 9-by-9 block from the image centered around it. We use the sum of squared differences (SSD) between these image regions as the matching cost between points. Each point in the left image is matched with the right-hand-side point with the lowest matching cost. There is no uniqueness constraint, so points from the right image can participate in multiple correspondences.

```
halfBlockSize = 4; % Block half-size.
blockSize = 2*halfBlockSize+1; % Full block size.
[r,c] = size(leftI);
matchThreshold = 0.7;

% Extract features for pointsRight
features = zeros(blockSize^2,size(pointsRight,2), 'single');
for i=1:size(pointsRight,2)
    T = zeros(blockSize, 'single');
    minr = max(1,pointsRight(2,i)-halfBlockSize);
    maxr = min(r,pointsRight(2,i)+halfBlockSize);
    minc = max(1,pointsRight(1,i)-halfBlockSize);
    maxc = min(c,pointsRight(1,i)+halfBlockSize);
    T( halfBlockSize+1-(pointsRight(2,i)-minr):halfBlockSize+1+(maxr-pointsRight(2,i)), ...
       halfBlockSize+1-(pointsRight(1,i)-minc):halfBlockSize+1+(maxc-pointsRight(1,i)) ) =
       rightI( minr:maxr, minc:maxc );
    features(:,i) = T(:);
end

% Loop over pointsLeft, looking for best matches in pointsRight via features.
ix1 = zeros(1,size(pointsLeft,2), 'single');
d = zeros(size(ix1), 'single');
for i=1:size(pointsLeft,2)
    T = zeros(blockSize, 'single');
    minr = max(1,pointsLeft(2,i)-halfBlockSize);
    maxr = min(r,pointsLeft(2,i)+halfBlockSize);
    minc = max(1,pointsLeft(1,i)-halfBlockSize);
    maxc = min(c,pointsLeft(1,i)+halfBlockSize);
    T( halfBlockSize+1-(pointsLeft(2,i)-minr):halfBlockSize+1+(maxr-pointsLeft(2,i)), ...
       halfBlockSize+1-(pointsLeft(1,i)-minc):halfBlockSize+1+(maxc-pointsLeft(1,i)) ) = ..
       leftI( minr:maxr, minc:maxc );
    % Find matching point in pointsRight with features features.
```

```
        [v,ix] = min( sum(bsxfun(@minus,features,T(:)).^2,1) );
        if v < matchThreshold
            ix1(i) = ix;
        end
    end

    % Extract indices of matched points on each side.
    ix2 = nonzeros(ix1);
    ix1 = find(ix1);
    % Create subselected, homogenized point lists.
    pointsLeftH  = [double(pointsLeft(:,ix1));  ones(1,length(ix1), 'single')];
    pointsRightH = [double(pointsRight(:,ix2)); ones(1,length(ix2), 'single')];
```

The image below is another color composite. Points from the left image are shown as blue, and points from the right image are shown as yellow. Green lines are drawn between the points to show the correspondences selected by this alignment procedure. Many of these are correct, but there is also a significant number of outliers.

```
% Display
figure(2), clf;
imshow(cat(3,rightI,leftI,leftI)), hold on;
plot(pointsLeftH(1,:),pointsLeftH(2,:),'x','Color',[0 0 1],'MarkerSize',8);
plot(pointsRightH(1,:),pointsRightH(2,:),'x','Color',[1 1 0],'MarkerSize',8);
plot([pointsLeftH(1,:);pointsRightH(1,:)],...
     [pointsLeftH(2,:);pointsRightH(2,:)],'-','Color',[0 1 0]);
set(gca,'XTick',[],'YTick',[]);
title('Initial corresponded points between the two images');
```



Step 4. Remove outliers using Random Sample Consensus (RANSAC) algorithm

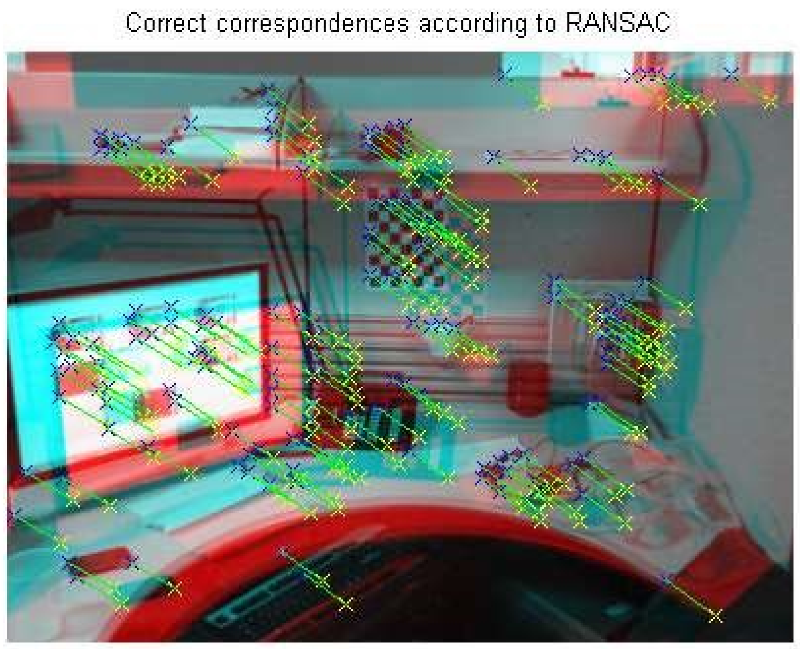
Many of the point correspondences obtained in the previous step are incorrect. We cannot proceed to rectification with so many erroneous point correspondences as above, so we must first discard bad matches (i.e., outliers). A standard way to do this is with the Random Sample Consensus (RANSAC) algorithms [3,4,5].

The RANSAC implementation is contained in the `vipfmatrix_ransac` function, which is also used in the Simulink Stereo Vision demo. In it, we select a random subset of eight correspondences to compute an exact rectification with linear methods. Then we check how many other points qualify as inliers under the resulting rectification. A rectification made from valid point correspondences will qualify many other correspondences as inliers, while one made from invalid points will not. The rectification that passes the most correspondences is assumed to be correct, and we use its inliers as valid correspondences. This procedure is repeated thousands of times to ensure that we achieve a good result with high probability.

```
nInliers = 0;
while nInliers < 100
    [M, x1inliers, ~] = vipfmatrix_ransac(...
        pointsLeftH(1:2,:), pointsRightH(1:2,:), 2500, 0.2);
    nInliers = size(x1inliers, 2);
end
% Recover inlier indices of x1.
[~,inliers] = find(all(bsxfun(@eq, permute(x1inliers,[2 3 1]), ...
    permute(pointsLeftH(1:2,:),[3 2 1]) ),3));
```

The correspondences qualified as inliers are plotted below.

```
figure(2), clf;
imshow(cat(3,rightI,leftI,leftI)), hold on;
plot(pointsLeftH(1,inliers),pointsLeftH(2,inliers),'x','Color',[0 0 1],'MarkerSize',8);
plot(pointsRightH(1,inliers),pointsRightH(2,inliers),'x','Color',[1 1 0],'MarkerSize',8);
plot([pointsLeftH(1,inliers);pointsRightH(1,inliers)],...
     [pointsLeftH(2,inliers);pointsRightH(2,inliers)],'-','Color',[0 1 0]);
set(gca,'XTick',[],'YTick',[]);
title('Correct correspondences according to RANSAC');
```

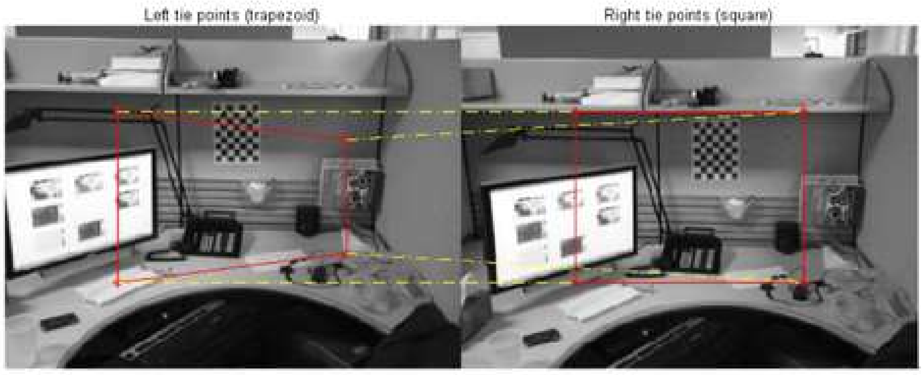


While most incorrect ones have been removed, a few may still remain. But, because of the error metric used in RANSAC, bad correspondences here will actually cause little error in the rectification process, and they will be outweighed by the far greater number of correct correspondences.

Step 5. Example of non-linear optimization for homographies

Before running the rectification procedure of [1] on the above points, we first demonstrate its effects. We use four synthetic point correspondences as shown in the figure below. The points lie in the form of a trapezoid in the left image and a square in the right. Yellow lines are drawn between the points given as correspondences.

```
% Create dummy points.
C1 = [size(leftI,2)*[1 1 3 3]/4;
      size(leftI,1)*[1/4 3/4 1/3 2/3];
      ones(1,4)];
C2 = [size(leftI,2)*[1 1 3 3]/4;
      size(leftI,1)*[1 3 1 3]/4;
      ones(1,4)];

figure(1), clf, hold on, axis ij;
imagesc([leftI rightI]), axis image, colormap gray;
plot(C1(1,:),C1(2,:),'r*');
plot(C1(1,[1 2 4 3 1]), C1(2,[1 2 4 3 1]),'r-');
plot(size(leftI,2)+C2(1,:),C2(2,:),'r*');
plot(size(leftI,2)+C2(1,[1 2 4 3 1]), C2(2,[1 2 4 3 1]),'r-');
plot([C1(1,:); size(leftI,2)+C2(1,:)], ...
     [C1(2,:); C2(2,:)],'y-.');
set(gca,'XTick',[],'YTick',[]);
text(size(leftI,2)/2,0,'Left tie points (trapezoid)',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
text(size(leftI,2)+size(rightI,2)/2,0,'Right tie points (square)',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
```



The rectification procedure transforms the images such that these points will appear on the same rows in both images. The core of the procedure is minimizing the error function defined in `viprectification_lsqnl.m`.

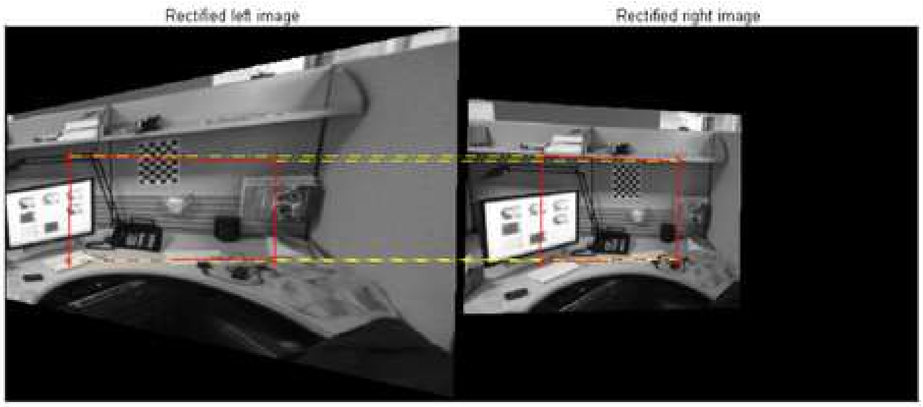
The optimization searches over six parameters: one for the focal length of the cameras, and five angular parameters describing the rotation between the two images. With these five angles, two rotation matrices, RectifiedLeft and RectifiedRight, are assembled for transforming the cameras. These are converted into rectifying homographies, Hleft and Hright, and the `GeometricTransformer` System object is used to rectify the images. Additional work is required to make sure the two rectified images are of the same size and properly aligned, which is handled in `viprectification_rectifywarp.m`.

```
% Set parameters of optimization.
[h,w] = size(leftI);
lambda = 0.1;
options = optimset('Jacobian','off', 'Display','off');
options.MaxFunEvals = 10000;
options.MaxIter = 1000;
bounds = [1; pi/2*ones(5,1)];

% Run optimization to discover rotation between cameras.
% Initialize with slightly random values for focal length parameter and the
% five angles (in radians).
rn = 1;
while rn > 0.0035
    % Loop until derived error is below a threshold.
    start = [2*rand-1.0; 0.2*rand(5,1)-0.1];
```

```
        [m,rn] = lsqnonlin(@(x)viprectification_lsqnl(x,C1,C2,...
            double(size(leftI,2)),double(size(leftI,1)),...
            lambda),start,-bounds,bounds,options);
    end
    % Recover rotation matrices between cameras from m.
    a = (3^m(1))*(w+h)';
    K = [a 0 w/2; 0 a h/2; 0 0 1];
    % Generate rotation matrix from returned angles.
    RectifiedRight = viprectification_rotmat(m(2:4), 'YZX');
    RectifiedLeft = viprectification_rotmat(m(5:6), 'YZ');

    % Compute fundamental matrix and rectifying homographies.
    F = inv(K)'*(RectifiedRight'*[0 0  0
                                  0 0 -1
                                  0 1  0]*RectifiedLeft)/K;
    Hright = (K*RectifiedRight)/K;
    Hleft = (K*RectifiedLeft)/K;
    % Recenter homographies on shifted image center.
    wh2 = [w; h]/2;
    l_cent = Hleft*[wh2;1]; % Reprojected left center.
    del_l = wh2 - l_cent(1:2)/l_cent(3);
    r_cent = Hright*[wh2;1]; % Reprojected right center.
    del_r = wh2 - r_cent(1:2)/r_cent(3);
    del_r(2) = del_l(2);
    Kr = K; % Modify the old intrinsics to construct different new ones.
    Kr(1:2,3) = Kr(1:2,3) + del_r;
    Kl = K;
    Kl(1:2,3) = Kl(1:2,3) + del_l;
    Hright = (Kr*RectifiedRight)/K; % Recompute the homographies.
    Hleft = (Kl*RectifiedLeft)/K;

    [RectifiedLeft,RectifiedRight,off] = ...
        viprectification_rectifywarp(leftI,rightI,Hleft,Hright);
```

The results of the rectification can be seen in the figure below.

```
figure(3), clf;
montage(cat(4,leftI,rightI)), clf, hold on, axis ij;
imagesc([RectifiedLeft RectifiedRight]), axis image, colormap gray;
C1p = Hleft*C1;
C1p = bsxfun(@rdivide,C1p,C1p(3,:));
C2p = Hright*C2;
C2p = bsxfun(@rdivide,C2p,C2p(3,:));
plot(C1p(1,:)+off(1),C1p(2,:)+off(2),'r*');
plot(C1p(1,[1 2 4 3 1])+off(1),C1p(2,[1 2 4 3 1])+off(2),'r-');
plot(size(RectifiedLeft,2)+C2p(1,:)+off(1),C2p(2,:)+off(2),'r*');
plot(size(RectifiedLeft,2)+C2p(1,[1 2 4 3 1])+off(1),...
    C2p(2,[1 2 4 3 1])+off(2),'r-');
plot([C1p(1,:); size(RectifiedLeft,2)+C2p(1,:)]+off(1), ...
     [C1p(2,:); C2p(2,:)]+off(2),'y-.');
set(gca,'XTick',[],'YTick',[]);
text(size(RectifiedLeft,2)/2,0,'Rectified left image',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
text(size(RectifiedLeft,2)+size(RectifiedRight,2)/2,0,...
    'Rectified right image','HorizontalAlignment','center',...
    'VerticalAlignment','bottom');
```



Both images have been heavily warped and scaled, in particular transforming the left-side trapezoid of points into a square. As desired, the lines connecting the point correspondences are now horizontal, meaning they have now been aligned. When we apply this procedure to a larger number of points, it will achieve the same effect while respecting the multiple constraints that all corresponding points must lie on nearly the same rows after rectification. Having a large number of points will make the procedure robust to outliers and avoid rectifications with large distortions.

Step 6. Refine the solution using multiple trials

As seen above, RANSAC may not return perfect results, free of erroneous correspondences. But both RANSAC and the rectification procedure are randomized algorithms, and their results will be different each time they are run. A simple way to improve our solutions is to run the above RANSAC-optimization procedure a fixed number of times and take the best result.

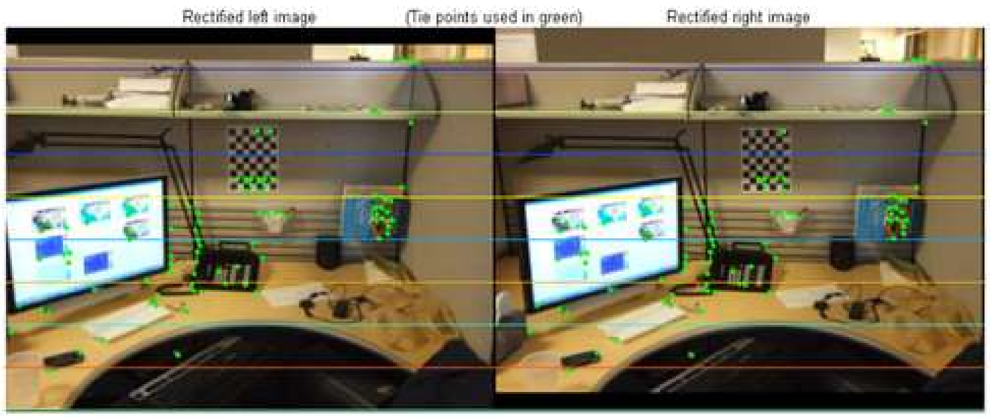
Below we run multiple trials of the combined procedure and take the solution with the least residual error according to the optimization step. The above calculation of rectifying homographies has been folded into a MATLAB function for readability: `viprectification_homographies.m`. As output we achieve the fundamental matrix, F, which relates points to epipolar lines between the two images, and the rectifying homographies.

```
nTrials = 10;
theBest.rn = inf;
for i=1:nTrials
    nInliers = 0;
    while nInliers < 20
        [~,x1inliers,~] = vipfmatrix_ransac(...
            pointsLeftH(1:2,:), pointsRightH(1:2,:), 5000, 4e-3);
        nInliers = size(x1inliers, 2);
    end
    % Recover inlier indices of x1.
    [~,inl] = find(all(bsxfun(@eq, permute(x1inliers,[2 3 1]), ...
        permute(pointsLeftH(1:2,:),[3 2 1]) ),3));

    % Run homography optimization on inliers.
    [Hleft,Hright,F,rn] = viprectification_homographies(...
        pointsLeftH(:,inl),pointsRightH(:,inl),...
        size(leftI,2),size(leftI,1),10,10);

    % Check for best result
    if rn < theBest.rn
        theBest.rn = rn;
        theBest.inl = inl;
        theBest.Hleft = Hleft;
        theBest.Hright = Hright;
        theBest.F = F;
    end
end
inliers = theBest.inl;
Hleft = theBest.Hleft;
```

```
Hright = theBest.Hright;
F = theBest.F;

[RectifiedLeft,RectifiedRight,off] = ...
    viprectification_rectifywarp(leftI3chan,rightI3chan,Hleft,Hright);
```

We plot these rectified images below and the final point correspondences used. We also draw a horizontal grid on the image to allow one to easily trace that features between the two images are now properly aligned along rows.

```
figure(1), clf;
imshow([RectifiedLeft RectifiedRight]), hold on;
X = axis;
ds = 35:35:X(4);
colours = jet(length(ds));
ds = [ds(1:2:end) ds(2:2:end)];
for i=1:length(ds)
    plot([X(1) X(2)],ds(i)*[1 1],'-','Color',colours(i,:));
end

colormap gray;
Pleft = Hleft * pointsLeftH(:,inliers);
Pleft = bsxfun(@rdivide,Pleft,Pleft(3,:));
Pright = Hright * pointsRightH(:,inliers);
Pright = bsxfun(@rdivide,Pright,Pright(3,:));
plot(Pleft(1,:)+off(1),Pleft(2,:)+off(2),'g.');
plot(size(RectifiedLeft,2)+Pright(1,:)+off(1),Pright(2,:)+off(2),'g.');
text(size(RectifiedLeft,2)/2,0,'Rectified left image',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
text(size(RectifiedLeft,2)+size(RectifiedRight,2)/2,0,'Rectified right image',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
text(size(RectifiedLeft,2),0,'(Tie points used in green)',...
    'HorizontalAlignment','center','VerticalAlignment','bottom');
```



References

[1] Irsara, L; Fusiello, A. "Quasi-euclidean uncalibrated epipolar rectification" Rapporto di Ricerca RR 43/2006, Dipartimento di Informatica, Universite di Verona, 2006. http://profs.sci.univr.it/~fusiello/demo/rect

[2] Trucco, E; Verri, A. "Introductory Techniques for 3-D Computer Vision." Prentice Hall, 1998.

[3] Hartley, R; Zisserman, A. "Multiple View Geometry in Computer Vision." Cambridge University Press, 2003.

[4] Hartley, R. "In Defense of the Eight-Point Algorithm." IEEE Transactions on Pattern Analysis and Machine Intelligence, v.19 n.6, June 1997.

[5] Fischler, MA; Bolles, RC. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography." Comm. Of the ACM 24, June 1981.