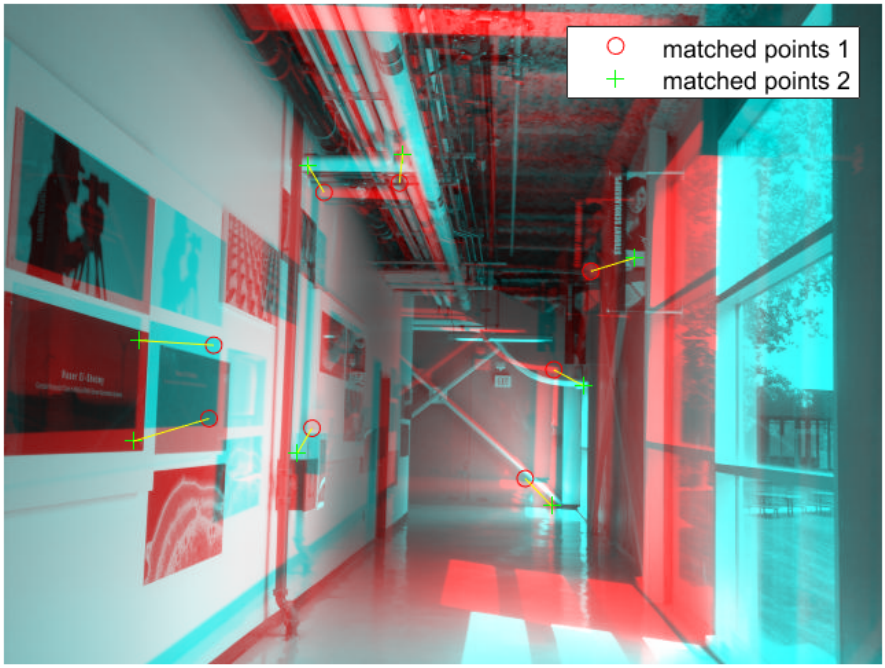
Exercise set: 4 **Student:** Tommaso Carraro **Student number:** 015147592

Exercise set 4 Computer Vision

Question 4.1 E matrix

The first thing that has to be done to compute the essential matrix is to find at least 8 image correspondences between the two provided images. I've done this using MATLAB functions and it is possible to see the 8 best matches between the two provided images in the following image.



The essential matrix operates on image points expressed in normalized coordinates and this means that the points in the image have to be aligned to the camera coordinates. So, we have to normalize the 8 image correspondences using the K matrix. It is possible to compute the K matrix having the calibration file. The formula that has to be used to compute this normalization is: $\hat{x} = K^{-1} * x$, where x is an image point and $\hat{x}$ a camera point. It is possible to use the same calibration matrix for both the images because they have been taken with the same camera. After the normalization it is possible to compute the essential matrix using the 8-point algorithm explained in the slides and having the previously computed 8 image correspondences.

This is my MATLAB code for the computation of the essential matrix using the 8-point algorithm.

```
% finding the 8 best correspondences between the two images
i1 = rgb2gray(imread('EFM1.jpg'));
i2 = rgb2gray(imread('EFM2.jpg'));
points1 = detectSURFFeatures(i1);
points2 = detectSURFFeatures(i2);
[f1, vpts1] = extractFeatures(i1, points1.selectStrongest(25));
[f2, vpts2] = extractFeatures(i2, points2.selectStrongest(25));
indexPairs = matchFeatures(f1, f2) ;
```

```
matchedPoints1 = vpts1(indexPairs(:, 1));
matchedPoints2 = vpts2(indexPairs(:, 2));

% form the K matrix to normalize points
K = [719.302047058490760 0 334.631234942930060;
    0 718.392548175289110 256.166677783686790;
    0 0 1];

% the inverse of the K matrix has to be used to normalize the points
invK = inv(K);

% points normalization with K matrix
x1 = matchedPoints1.Location;
x2 = matchedPoints2.Location;
x1(:,3) = 1;
x2(:,3) = 1;
x1 = invK*x1';
x2 = invK*x2';

% building the constraint matrix
A = [x2(1,:)'.*x1(1,:)' x2(1,:)'.*x1(2,:)' x2(1,:)' ...
    x2(2,:)'.*x1(1,:)' x2(2,:)'.*x1(2,:)' x2(2,:)' ...
    x1(1,:)' x1(2,:)' ones(8,1) ];

[U,D,V] = svd(A);

% Extract fundamental matrix from the column of V
% corresponding to the smallest singular value.
E = reshape(V(:,9),3,3)';

% Enforce rank2 constraint - The constraint of the first two singular
% values must be equal and the third one is zero has to be enforced.
[U,D,V] = svd(E);
E = U*diag([D(1,1) D(2,2) 0])*V';
E
```

After the execution of this code the following essential matrix has been computed:

$$E = \begin{bmatrix} -0.0354 & 0.6921 & -0.0861 \\ -0.7072 & -0.0144 & -0.0146 \\ 0.1080 & 0.0046 & 0.0019 \end{bmatrix}.$$

Question 4.1 F matrix

To compute the fundamental matrix I used the same 8 image correspondences found previously and the normalized version of the 8-point algorithm presented on the slides. To perform the normalization of the matched points we have to force their root-mean-squared distance to be $\sqrt{2}$. To do this it is enough to use the following matrix:

$T = \begin{bmatrix} 2/width & 0 & -1 \\ 0 & 2/height & -1 \\ 0 & 0 & 1 \end{bmatrix}$, where $width$ is the width of the image and $height$ is the height of the image.

After the normalization it is possible to use the 8-point algorithm, but at the end of the computation it is necessary to denormalize the computed fundamental matrix to obtain the correct one.

This is my MATLAB code for the computation of the fundamental matrix using the normalized 8-point algorithm.

```
% finding the 8 best correspondences between the two images
i1 = rgb2gray(imread('EFM1.jpg'));
i2 = rgb2gray(imread('EFM2.jpg'));
points1 = detectSURFFeatures(i1);
points2 = detectSURFFeatures(i2);
[f1, vpts1] = extractFeatures(i1, points1.selectStrongest(25));
[f2, vpts2] = extractFeatures(i2, points2.selectStrongest(25));
indexPairs = matchFeatures(f1, f2) ;
matchedPoints1 = vpts1(indexPairs(:, 1));
matchedPoints2 = vpts2(indexPairs(:, 2));

% normalization of the matched points so that their root-mean-squared
% distance is sqrt(2)
[s1,s2] = size(i1);
T = [2/s2 0 -1;
    0 2/s1 -1;
    0 0 1];
x1 = matchedPoints1.Location;
x2 = matchedPoints2.Location;
x1(:,3) = 1;
x2(:,3) = 1;
x1 = T*x1';
x2 = T*x2';

% building the constraint matrix
A = [x2(1,:)'.*x1(1,:)' x2(1,:)'.*x1(2,:)' x2(1,:)' ...
    x2(2,:)'.*x1(1,:)' x2(2,:)'.*x1(2,:)' x2(2,:)' ...
    x1(1,:)' x1(2,:)' ones(8,1) ];

[U,D,V] = svd(A);

% Extract fundamental matrix from the column of V
% corresponding to the smallest singular value.
F = reshape(V(:,9),3,3)';

% Enforce rank2 constraint - The constraint of the first two singular
% values must be equal and the third one is zero has to be enforced.
[U,D,V] = svd(F);
F = U*diag([D(1,1) D(2,2) 0])*V';

% Denormalize the F matrix
F = T'*F*T;
F
```

After the execution of this code the following fundamental matrix has been computed:

$$F = \begin{bmatrix} -0.0000 & 0.0000 & -0.0027 \\ -0.0000 & -0.0000 & 0.0027 \\ 0.0032 & -0.0027 & -0.1016 \end{bmatrix}.$$

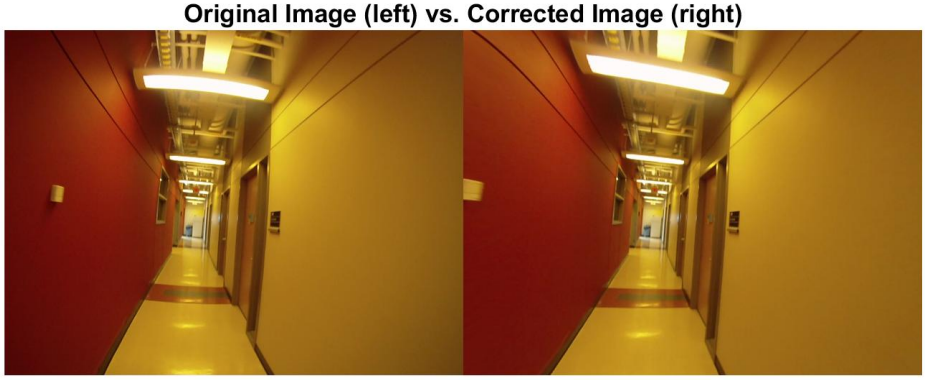
Question 4.2

To solve this exercise it is enough to correct the radial distortion of the two provided images before detecting the lines with the hough transform. To correct the radial distortion I've used the calibration parameters in the calibration file provided and the three distortion coefficients different from zero. To perform this correction I used MATLAB functions and my implementation is provided below.

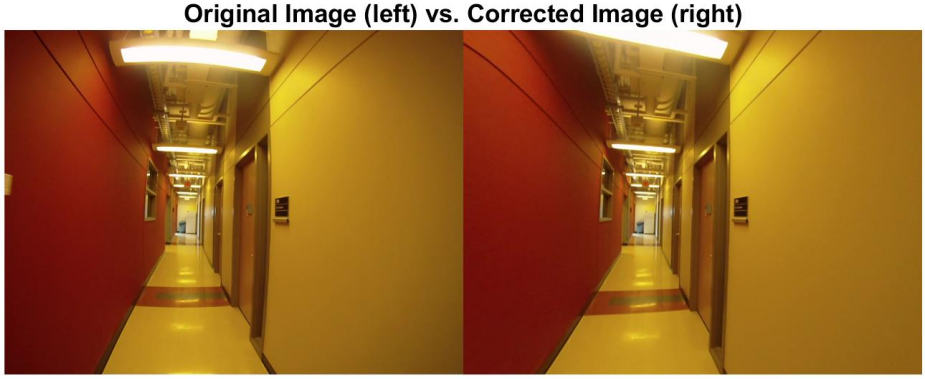
```
% function that corrects the radial distortion of the provided image
function newImg = correctDistorsion(img)
    % distorsion coefficients on the calibration file
    k = [ -0.159950393345315 -0.000685869015729 -0.004896132860374 ];
    imsize = size(img);
    %-- Focal length on the calibration file
    fc = [ 585.850107917267790 586.003198722303180 ];
    %-- Principal point on the calibration file
    cc = [ 664.903569991381570 498.409524449186850 ];
    % correcting radial distorsion with MATLAB function
    IntrinsicMatrix = [fc(1) 0 0; 0 fc(2) 0; cc(1) cc(2) 1];
    radialDistortion = k;
    cameraParams = cameraParameters('IntrinsicMatrix',IntrinsicMatrix,
    'ImageSize', [imsize(1) imsize(2)],
    'RadialDistortion',radialDistortion);
    newImg = undistortImage(img, cameraParams);
end
```

In the following two images I show the results of my correction algorithm on the provided images. It is possible to see that now the images have the correct shapes (the first light is square now).

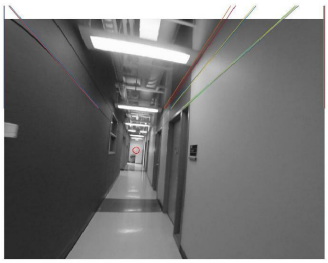
Original Image (left) vs. Corrected Image (right)

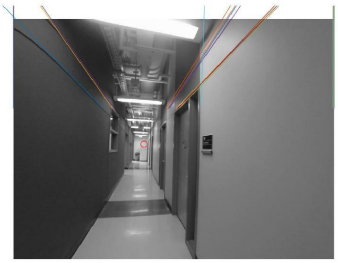


Original Image (left) vs. Corrected Image (right)



After the correction I have computed the hough transform and then the vanishing point locations. In the following two images I show the location of the new vanishing points (second image) compared to the distorted location (first image).





It is possible to observe that the new locations are better than the previous locations, in fact the vanishing points fall more or less in the center of the corridor.