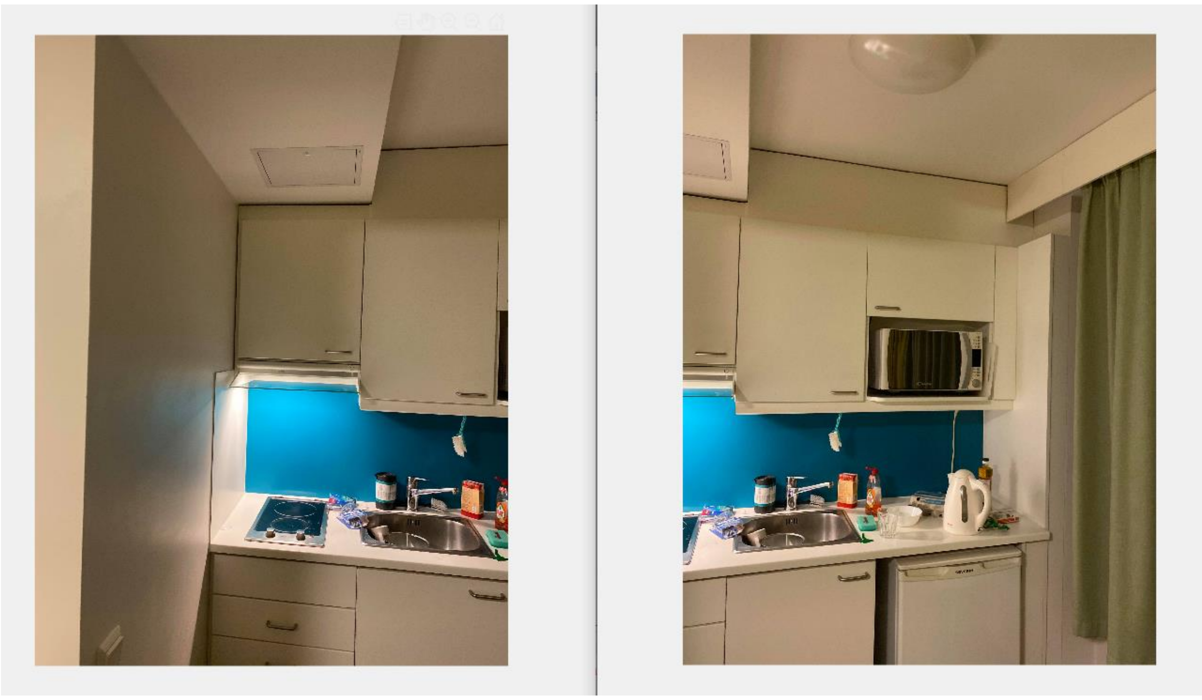
Normalised DLT

Loading the image

```
kitchen1 = imread('kitchen2.jpg');
figure,imshow(kitchen1)
kitchen2 = imread('kitchen1.jpg');
figure,imshow(kitchen2)
```



Grayscaling and finding the points that match with each other from the two images.

```
graykitchen1 = rgb2gray(kitchen1);
graykitchen2 = rgb2gray(kitchen2);
points1 = detectSURFFeatures(graykitchen1)
points2 = detectSURFFeatures(graykitchen2)

[features1,valid_points1] = extractFeatures(graykitchen1,points1);
[features2,valid_points2] = extractFeatures(graykitchen2,points2);
indexPairs = matchFeatures(features1,features2,'MaxRatio',0.1);

%disp(indexPairs)
matchedPoints1 = valid_points1(indexPairs(:,1),:);
```

%disp(matchedPoints1)

matchedPoints2 = valid_points2(indexPairs(:,2),:);

%disp(matchedPoints2)

To get the minimum amount of points needed to do the homography (four points), I randomly selected 4 points from the first image, then I first transposed the points and normalized them with the algorithm given, obtaining normalisedFourPoints at the end.

```
fourPoints1 =

  3×4 single matrix

   1.0e+03 *

    2.4353    2.7005    2.8464    2.3790
    2.9615    3.0575    2.8822    2.8540
    0.0010    0.0010    0.0010    0.0010
```

```
normalisedFourPoints =

   -0.2711    0.1928    0.4479   -0.3696
    0.0397    0.2075   -0.0990   -0.1483
    1.0000    1.0000    1.0000    1.0000
```

```
fourPoints1 = transpose(matchedPoints1([1 3 7 14],:).Location)
fourPoints1(3,:) = 1;
average = mean(fourPoints1,2)
d1 = 0;
for i = 1:size(fourPoints1,2)
  cal = sqrt((fourPoints1(1,i)-average(1,1)).^2+(fourPoints1(2,i)-average(2,1)).^2);
  d1 = d1 + cal;
end
T1 = [sqrt(2)/d1 0 -sqrt(2)*average(1,1)/d1;0 sqrt(2)/d1 -sqrt(2)*average(2,1)/d1; 0 0 1]
normalisedFourPoints = zeros(size(fourPoints1))
for i = 1:size(fourPoints1,2)
  temp = fourPoints1(:,i)
  temp = T1*temp
  normalisedFourPoints(:,i) = temp
end
```

Doing the same thing to the SAME four points matched in the second image:

```
fourPoints2 = transpose(matchedPoints2([1 3 7 14],:).Location)
fourPoints2(3,:) = 1;
average2 = mean(fourPoints2,2)
d2 = 0;
for i = 1:size(fourPoints2,2)
  cal = sqrt((fourPoints2(1,i)-average2(1,1)).^2+(fourPoints1(2,i)-average2(2,1)).^2);
  d2 = d2 + cal;
end
T2 = [sqrt(2)/d2 0 -sqrt(2)*average2(1,1)/d2;0 sqrt(2)/d2 -sqrt(2)*average2(2,1)/d2; 0 0 1]
normalisedFourPoints2 = zeros(size(fourPoints2))
for i = 1:size(fourPoints2,2)
  temp = fourPoints2(:,i)
  temp = T2*temp
  normalisedFourPoints2(:,i) = temp
end
```

After obtaining the normalisedPoints, I used the following function that I implemented called calchomography to find the homography of the points.

```
% Estimate the homography from a set of point correspondences.
function H = calchomography(points1, points2)

  if size(points1, 2) < 4 || size(points2, 2) < 4
    error('no 4 points passed');
  end

  A = [];

  for point = 1:size(points1, 2)
   x = points1(1, point);
   y = points1(2, point);
   u = points2(1, point);
```

```
  v = points2(2, point);
  A = [A; ...
     -x  -y  -1   0   0   0  x * u  y * u  u; ...
      0   0   0  -x  -y  -1  x * v  y * v  v];
end

% Solve A * h = 0 for h.
[U, S, V] = svd(A);
h = V(:,end);
% Reshape vectorised result back into matrix shape.
H = reshape(h, 3, 3)';
% Homogeneous normalisation.
H = H ./ H(3,3);
```

Calling the function above with H and denormalise it, then saving it as mymatrix.mat:

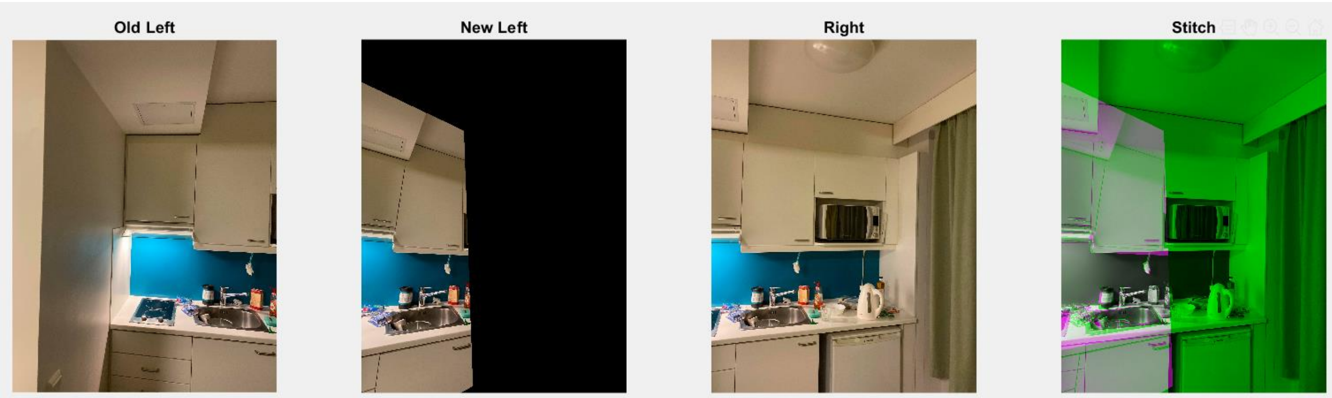
```
H = calchomography(normalisedFourPoints,normalisedFourPoints2);
H = inv(T2)*H*T1  %Denormalising
disp(H)
save ('mymatrix.mat', 'H');
```

To show that the homography we obtained indeed works, I have created a backwardWarping algorithm that allows me to warp the first image to the second image. I didn't include the codes in **week3matlab.m** but it is at **imageAlignment.m** since backward mapping isn't necessary, the result is below, notice the sinks in both images, they line up perfectly.

```
source = im2double(imread('kitchen2.jpg'));
source2 = im2double(imread('kitchen1.jpg'));
newLeft = zeros(size(source));

% The backward warping transformation (rotation + scale about an arbitrary point).
Matrix = load('mymatrix.mat');
M = Matrix.H;
for y = 1:size(newLeft, 1)
  for x = 1:size(newLeft, 2)
```

```
        % Transform source pixel location (round to pixel grid).
        p = [x; y; 1];
        q = inv(M) * p;
        u = round(q(1) / q(3));
        v = round(q(2) / q(3));

       % Check if target pixel falls inside the image domain.
        if (u > 0 && v > 0 && u <= size(source, 2) && v <= size(source, 1))
          % Sample the target pixel colour from the source pixel.
          newLeft(y, x, :) = source(v, u, :);
        end
    end
end
target = imfuse(source2,newLeft);
%overlay
subplot(141); imshow(source); title('Old Left');
subplot(142); imshow(newLeft); title('New Left');
subplot(143); imshow(source2); title('Right');
subplot(144); imshow(target); title('Stitch');
```

Results:



HoughLines detector

Reading and detecting the edge with matlab canny edge detector.

```
corridor = imread('Corridor1.jpg');

grayCorridor = rgb2gray(corridor);

Canny_img = edge(grayCorridor, 'Canny');

[rows, cols] = size(Canny_img);
```

Initialising the hough space (Accumulator) to the size of theta_range and rho_range.

```
theta_maximum = 90;

rho_maximum = hypot(rows,cols) - 1;

theta_range = -theta_maximum:theta_maximum-1;

rho_range = -rho_maximum:rho_maximum;

Hough = zeros(length(rho_range), length(theta_range));
```

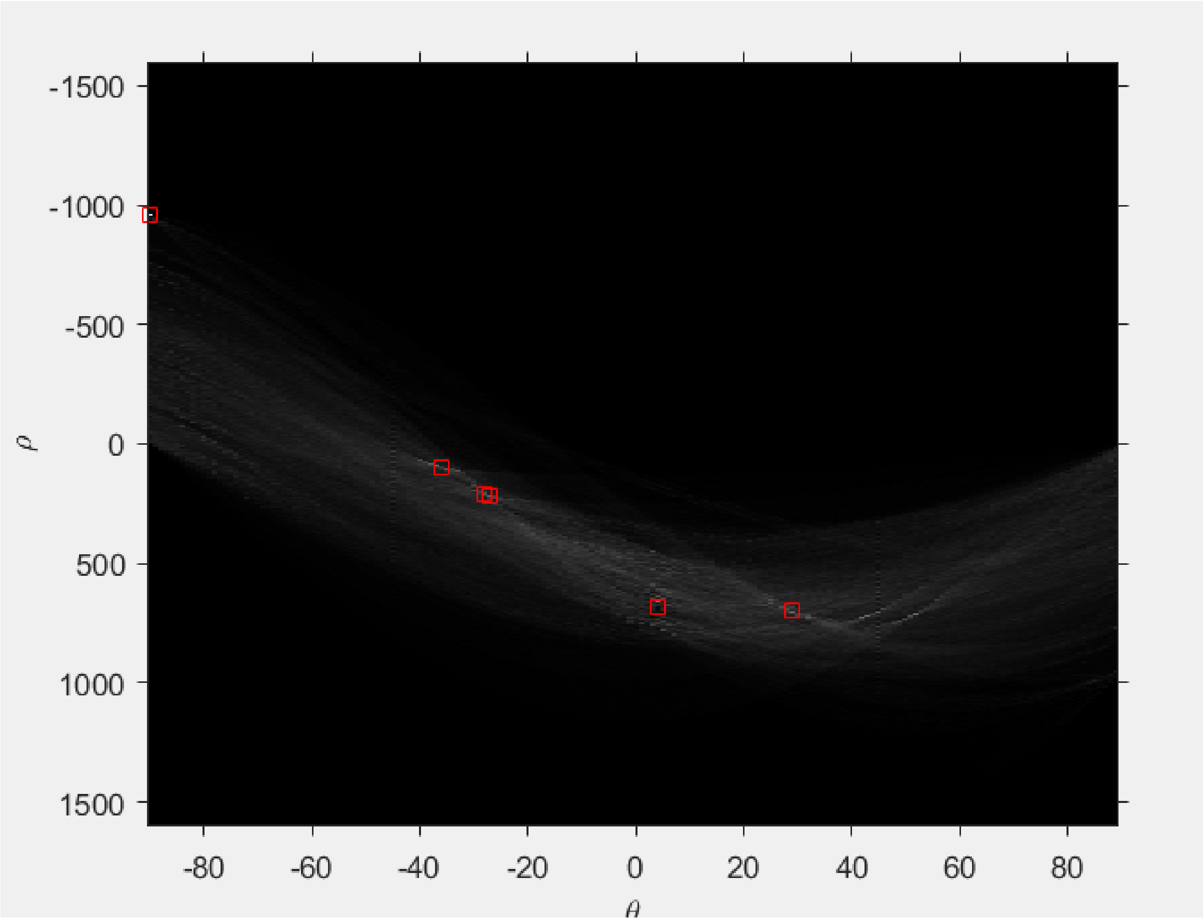
Converting the edge found with r = x*cos(theta) + y*sin(theta).

```
for j = 1:rows

  for i = 1:cols

    if Canny_img(j, i) ==1

      x = i - 1;

      y = j - 1;

      for T = theta_range

        R = round((x * cosd(T)) + (y * sind(T)));

        R_Index = R + rho_maximum + 1;

        T_Index = T + theta_maximum + 1;

        Hough(R_Index, T_Index) = Hough(R_Index, T_Index) + 1;

      end

    end

  end

end
```

Hough peak with thresholding.

```
[X,Y] = find(Hough>0.45*max(Hough(:)))
```

```
imshow(Hough,[],'XData',theta_range,'YData',rho_range,...
        'InitialMagnification','fit');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;

pointsRho = rho_range(X(:));
pointsTheta = theta_range(Y(:));
plot(pointsTheta,pointsRho,'s','color','red');
```



Here is my attempt at dehoughing the peaks to image space. I first get the Rho and Theta pair into a list. I then created a set of binary matrices of the size imageRow x imageColumn x number of rhotheta pair. Then I substitute all the x and y pixel coordinates into the function x*cos(theta) + y*sin(theta) = rho, and if left - right == 0, I would mark the location of the binary matrix(x,y) as 1 and I run this for all the theta pairs against all the x and y value and store them separately in the set of binary matrices mentioned above.

In the second for loop, I just went through all of the set of binary matrices I had obtained and just aggregate them into one binary matrix.

```
for k = 1:length(pointsRho)
  for i = 1:1280
    for j = 1:960
      r = pointsRho(k);
      t = pointsTheta(k);
      threshold = abs(i*cosd(t)+j*sind(t)-r);
      if threshold <0.5
        allLines(j,i,k) = 1;
      end
    end
  end
end
aggregatedImage = zeros(size(Canny_img));

for k = 1:length(pointsRho)
  for j = 1:1280
    for i = 1:960
      if allLines(i,j,k) ==1
        aggregatedImage(i,j) = 1;
      end
    end
  end
end
```

figure,imshow(imfuse(aggregatedImage,corridor))

With houghpeaks = 0.45*max(Hough(:)) we get the following lines