Metric rectification using the images of two circles

perform a metric rectification from two/three images of two circles Be aware that the image dual to the circular points is not estimated accurately, thus we can rely to a different strategy, enforcing ellipses to be circles. A useful reference: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4937883

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Image Analysis and Computer Vision Politecnico di Milano

Luca Magri for comments and suggestions please send an email to luca.magri@polimi.it

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Contents

load the image

```
clear;
close all;
```

```
addpath('E4_aux/');
addpath('E4_data/');
im = imread('E4_data/memphis.jpeg');
im = imresize(im,0.9);
%im = imread('E4_data/sonics.jpeg');
figure; imshow(im);
```

here I have circles...
they should have common points

I can use circles intersections as point in common to use to get e∞, then make affine transform and finally metric rectification
then use geometry to improve

```
do_load_ellipses = 1;
if(do_load_ellipses)
    data = load('memphis_ellipses.mat');
    e1 = data.e1;
    e2 = data.e2;
    e3 = data.e3;
end
```

select ellipses in the image

```
figure;
imshow(im);
hold all;
if(do_load_ellipses)
```

```
    e1 = drawellipse('Center',e1.Center,'SemiAxes',e1.SemiAxes,'RotationAngle',e1.RotationAngle);
    pause
    e2 = drawellipse('Center',e2.Center,'SemiAxes',e2.SemiAxes,'RotationAngle',e2.RotationAngle);
    pause
    e3 = drawellipse('Center',e3.Center,'SemiAxes',e3.SemiAxes,'RotationAngle',e3.RotationAngle);
    pause
else
    e1 = drawellipse();
    pause
    e2 = drawellipse();
    pause
end
if(0)
    save('memphis_ellipses.mat','e1','e2','e3');
end
```

convert ellipses into conic matrices

```
par_geo = [e1.Center, e1.SemiAxes,-e1.RotationAngle]';
par_alg = conic_param_geo2alg(par_geo);
[a1, b1, c1, d1, e1, f1] = deal(par_alg(1),par_alg(2),par_alg(3),par_alg(4),par_alg(5),par_alg(6));
C1=[a1 b1/2 d1/2; b1/2 c1 e1/2; d1/2 e1/2 f1];
% sanity check
im_err = zeros(size(im,1),size(im,2));
for i = 1:size(im,1)
    for j = 1:size(im,2)
        im_err(i,j) = [j,i,1]*C1*[j;i;1];
    end
end

% visual sanity check
lambda = 0.5;
figure;
imshow(lambda*im+(1-lambda)*uint8(255.*(im_err<0)));
```

convert also e2 and e3

```
par_geo = [e2.Center, e2.SemiAxes,-e2.RotationAngle]';
par_alg = conic_param_geo2alg(par_geo);
[a2, b2, c2, d2, e2, f2] = deal(par_alg(1),par_alg(2),par_alg(3),par_alg(4),par_alg(5),par_alg(6));
C2 = [a2 b2/2 d2/2; b2/2 c2 e2/2; d2/2 e2/2 f2];
```

```
par_geo = [e3.Center, e3.SemiAxes,-e3.RotationAngle]';
par_alg = conic_param_geo2alg(par_geo);
[a3, b3, c3, d3, e3, f3] = deal(par_alg(1),par_alg(2),par_alg(3),par_alg(4),par_alg(5),par_alg(6));
C3 = [a3 b3/2 d3/2; b3/2 c3 e3/2; d3/2 e3/2 f3];

% visual check
im_err2 = zeros(size(im,1),size(im,2));
im_err3 = zeros(size(im,1),size(im,2));
for i = 1:size(im,1)
    for j = 1:size(im,2)
        im_err2(i,j) = [j,i,1]*C2*[j;i;1];
        im_err3(i,j) = [j,i,1]*C3*[j;i;1];
    end
end


msk = (im_err <0) + (im_err2<0) + (im_err3<0);
lambda = 0.5;
figure;
imshow(lambda*im + (1-lambda)*uint8(255.*msk));
```

using numerical symbolic solver, then again numerical solution

intersect ellipses

see golub for altenative solutions

```
syms 'x';
syms 'y';

% every conic provide a 2nd degree equation
eq1 = a1*x^2 + b1*x*y + c1*y^2 + d1*x + e1*y + f1;
eq2 = a2*x^2 + b2*x*y + c2*y^2 + d2*x + e2*y + f2;
eq3 = a3*x^2 + b3*x*y + c3*y^2 + d3*x + e3*y + f3;
% solve a system with two conics: this gives a 4th degree equation
eqns = [eq1 == 0, eq2 == 0];
S12 = solve(eqns, [x,y], 'IgnoreAnalyticConstraints',true,'Maxdegree',4);
% hence you get 2 pairs of complex conjugate solution
s1 = [double(S12.x(1)); double(S12.y(1)); 1];
s2 = [double(S12.x(2)); double(S12.y(2)); 1];
s3 = [double(S12.x(3)); double(S12.y(3)); 1];
s4 = [double(S12.x(4)); double(S12.y(4)); 1];
```

find correct solutions... by computing ∞ points

disambiguate between the solutions

if you have other conics in the image you can solve similar systems

```
S13 = solve([eq1 == 0, eq3 == 0], [x,y]);
S23 = solve([eq2 == 0, eq3 == 0], [x,y]);

% and get solutions
p1 = [double(S13.x(1)); double(S13.y(1)); 1];
p2 = [double(S13.x(2)); double(S13.y(2)); 1];
p3 = [double(S13.x(3)); double(S13.y(3)); 1];
p4 = [double(S13.x(4)); double(S13.y(4)); 1];

q1 = [double(S23.x(1)); double(S23.y(1)); 1];
q2 = [double(S23.x(2)); double(S23.y(2)); 1];
q3 = [double(S23.x(3)); double(S23.y(3)); 1];
q4 = [double(S23.x(4));double(S23.y(4));1];


% there should be a pair of solution that is common represent the image of the circular
% points
[s1,s2,s3,s4; p1,p2,p3,p4;q1,q2,q3,q4]
```

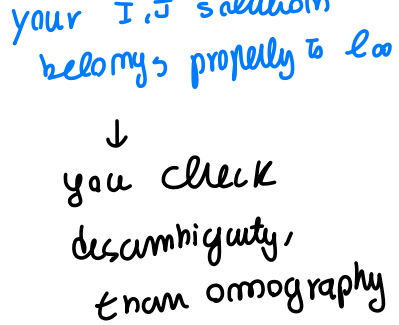
```
ans =

   1.0e+03 *
```

```
   0.9579 - 1.9327i   0.9579 + 1.9327i   0.9944 + 0.0008i   0.9944 - 0.0008i
  -0.5009 + 0.0233i  -0.5009 - 0.0233i   0.2175 + 0.1557i   0.2175 - 0.1557i
   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i
   1.2067 - 0.0238i   1.2067 + 0.0238i   0.7135 + 2.0948i   0.7135 - 2.0948i
   0.2434 - 0.0752i   0.2434 + 0.0752i  -0.5797 - 0.1011i  -0.5797 + 0.1011i
   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i
   0.7852 + 0.0182i   0.7852 - 0.0182i   1.0471 - 1.7145i   1.0471 + 1.7145i
   0.2436 - 0.0755i   0.2436 + 0.0755i  -0.4274 - 0.0290i  -0.4274 + 0.0290i
   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i   0.0010 + 0.0000i
```

disambiguate between the solutions

let's compute the line at infinity. the line at infinity in the correct position is the one from the actual image of circular points compute the line at infinity

```
l_inf_1 = imag(cross(s1,s2));
l_inf_2 = imag(cross(s3,s4));
% also close form solutions are avilable
%l_inf_1 = [imag(s1(2)),-imag(s1(1)), imag(s1(1)*s2(2))];
%l_inf_2 = [imag(s3(2)),-imag(s3(1)), imag(s3(1)*s4(2))];

l_inf_1  = l_inf_1./norm(l_inf_1);
l_inf_2  = l_inf_2./norm(l_inf_2);

% plot the image of the line at infinity
% get the intersection with x=0 and x = size(im,1)
% ax +by +c = 0, x = 0 -> y = -c/b
%           = 0, x = w -> y = -(c+aw)/b
w = size(im,2);
A1 = [0; -l_inf_1(3)/l_inf_1(2);1];
B1 = [w; -(l_inf_1(3)+l_inf_1(1)*w)/l_inf_1(2);1];

A2 = [0; -l_inf_2(3)/l_inf_2(2);1];
B2 = [w; -(l_inf_2(3)+l_inf_2(1)*w)/l_inf_2(2);1];


MSZ = 10;
figure;

hold all;
plot(A1(1),A1(2),'ro','MarkerSize',MSZ);
plot(B1(1),B1(2),'ro','MarkerSize',MSZ);
plot(A2(1),A2(2),'bo','MarkerSize',MSZ);
plot(B2(1),B2(2),'bo','MarkerSize',MSZ);
line([A1(1),B1(1)],[A1(2),B1(2)],'linewidth',4,'Color','r');
line([A2(1),B2(1)],[A2(2),B2(2)],'linewidth',4,'Color','b');
imshow(im);
axis equal;
```



let's also check how the corresponding homographies transform the image

```
H1 = [eye(2),zeros(2,1); l_inf_1(:)'];
H2 = [eye(2),zeros(2,1); l_inf_2(:)'];

tform1 = projective2d(H1');
tform2 = projective2d(H2');
J1 = imwarp(im,tform1);
%J2 = imwarp(im,tform2);
figure;
%subplot(1,2,1);
imshow(J1);
%subplot(1,2,2);
%imshow(J2);
%axis equal;
% the correct pairs of points give rise to an homography that perform a
% correct affine reconstruction.
```

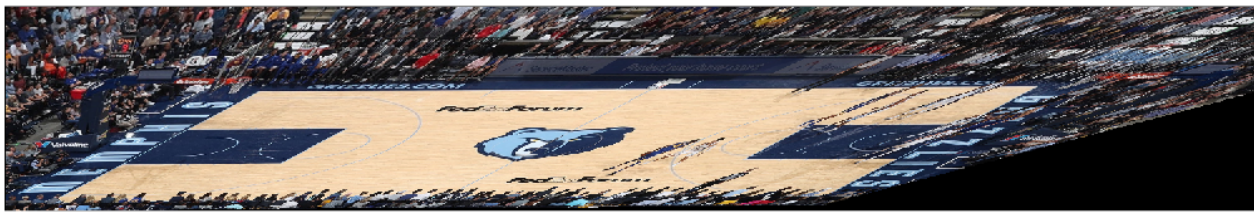
let's try to get a metric transformation

```
% compute the image of the dual conic to principal points
II = s1;%mean([s1,p1,q1],2)
JJ = s2;%mean([s2,p2,q2],2)
imDCCP = II*JJ' + JJ*II';
imDCCP = imDCCP./norm(imDCCP);

C = [eye(2),zeros(2,1);zeros(1,3)]; % canonic conic dual to circular points
```

extract the line at infinity

```
l_inf = null(imDCCP);
% s1 and s2 passes through the line at infinity
s1'*l_inf
s2'*l_inf

H = [eye(2),zeros(2,1); l_inf(:)'];
tform = projective2d(H');
im_aff = imwarp(im,tform);
figure;
imshow(im_aff);
```

```
ans =

  -2.2204e-16 - 1.1102e-16i


ans =

  -2.2204e-16 + 1.1102e-16i
```



it is not possible to compute the rectifying homography

```
[V,D] = eigs(imDCCP)
% sort the eigenvalue
[d,ind] = sort(abs(diag(D)),'descend');
Ds = D(ind,ind);
Vs = V(:,ind);

D
% has a negative eigenvalue :(
```

we have bad rectified solutions...
↳ we can exploit circle!

```
V =

    0.9905    0.1377    0.0000
    0.1377   -0.9905    0.0020
   -0.0003    0.0020    1.0000


D =

   -1.0000         0         0
         0    0.1080         0
         0         0   -0.0000


D =
```

```
   -1.0000         0         0
         0    0.1080         0
         0         0   -0.0000
```

However we have a circle → compute l_∞ from Null($C_\infty^{*\prime}$)

so we can distil metric information let's use the information that the axis of the ellipses should be equal transform the conic in the rectified plane according to the rule C = H^-t C H^-1

```
Q = inv(H)'*C1*inv(H);
Q = Q./Q(3,3);
% sanity check
im_err = zeros(size(im_aff,1),size(im_aff,2));
for i = 1:size(im_err,1)
    for j = 1:size(im_err,2)
        im_err(i,j) = [j,i,1]*Q*[j;i;1];
    end
end
lambda = 0.5;
figure;
imshow(lambda*im_aff+(1-lambda)*uint8(255.*(im_err<0)));
```

↓ we use CIRCLE

convert the conic coefficient to geometric parameters

Ax^2 + Bxy + Cy^2 +Dx + Ey + F = 0

```
par_geo = AtoG([Q(1,1),2*Q(1,2),Q(2,2),2*Q(1,3),2*Q(2,3),Q(3,3)]);
center = par_geo(1:2);
axes = par_geo(3:4);
angle = par_geo(5);
figure;
imshow(im_aff);
hold on;
plot(center(1),center(2),'ro','Markersize',20);
```

Now we can compute the affinity that make the axis of the ellipses to be equal.

↑ force ellipses with scaling transform to have equal axis

The affinity is composed by a rotation a scaling and the inverse rotation

```
alpha = angle;
a = axes(1);
b = axes(2);
% rotation
U = [cos(alpha), -sin(alpha); sin(alpha), cos(alpha)];
% rescaling the axis to be unitary
S = diag([1, a/b]);
K = U*S*U';
A = [K zeros(2,1); zeros(1,2),1];
T = A*H; % the final transformation is the composition between the homography that maps the image of the line at infinity to its canonical positic
```

apply the transformation

```
tform = projective2d(T');
J = imwarp(im,tform);
figure;
imshow(J);
title('Metric rectification.')
```

Metric rectification.

Published with MATLAB® R2021b

people are distorted...
what it OFF the plane
get DISTORTED!

↓

you recover metric
information on PLANE,
not on Z axis

we discuss rectification by

1) Affine rectification := move image l'_∞ to l_∞ canonical

2) Metric reconstruction
+ taking into account Numerically instability

3) stratified approach using l_∞, then constraint

↓

- geometric constraints works well
- Algebraic properties, have non lin syst to account for! so numerical issue (conics eq. non cin) not simple as SVD

Affine measurements

This code demonstrate the estimation of vanishing points and vanishing lines to take affine measurements from an image. Specifically this code allows to estimate the height of an object standing on the ground, provided: - the height of a reference object standingh on the ground, - a vertical vanishing point, - the vanishing line of the ground.

Note that for more accurate results the image should be undirstorted .\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Image Analysis and Computer Vision Politecnico di Milano Luca Magri for comments and suggestions please send an email to luca.magri@polimi.it

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Contents


```
close all % close all the figures
clear % clear the workspace
clc % clear the command window
```

load image

```
im = imread('poli2.jpg');
im = imresize(im,0.6);
figure(1);
imshow(im);
```

draw "vertical" segments in the image

i.e. segments in image that comes from vertical lines drag segments on the image, you can adjust the poistions a little bit, when you are happy of the result press enter.

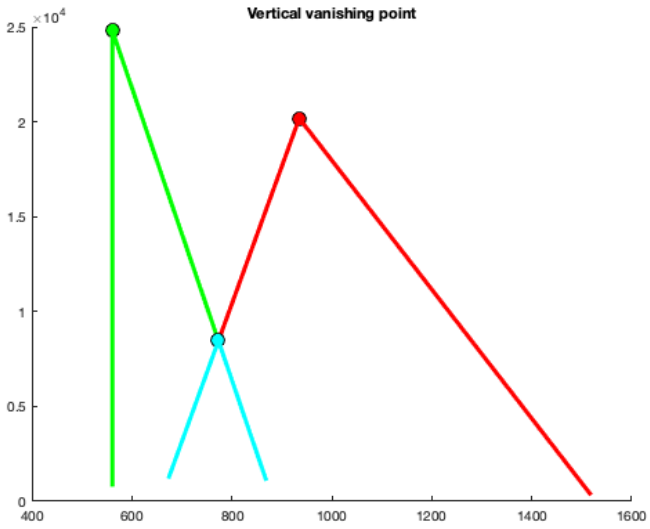
```
numSegments = 4; % the minimum number of segments is two, the higher the better the result
verticalLines = nan(numSegments,3); % store vertical parallel lines
endPoints = nan(2,numSegments); % store the endpoints of the segments for visualization purpose
count = 1;
while(count <=numSegments)
    figure(1);
    title(['Draw ', num2str(numSegments),' vertical segments: step ',num2str(count) ]);
    seg = drawline('Color','b');
    % convert the end points of the segments into the coefficients of
    % the line
    endPoints(:,count) = seg.Position(1,:)';
    % switch to homoegeous coordinates
    a = [seg.Position(1,:)';1];
    b = [seg.Position(2,:)';1];
    % get the parameters of the line
    l = cross(a,b);
    l = l./norm(l); %normalize it, just to remove the scale factor
    verticalLines(count, :) = l';
    count = count +1;
end
fprintf('Press enter to continue\n');
pause
```

```
Press enter to continue
```

Draw 4 vertical segments: step 4

pairwise vanishing points are different because segments are noisy

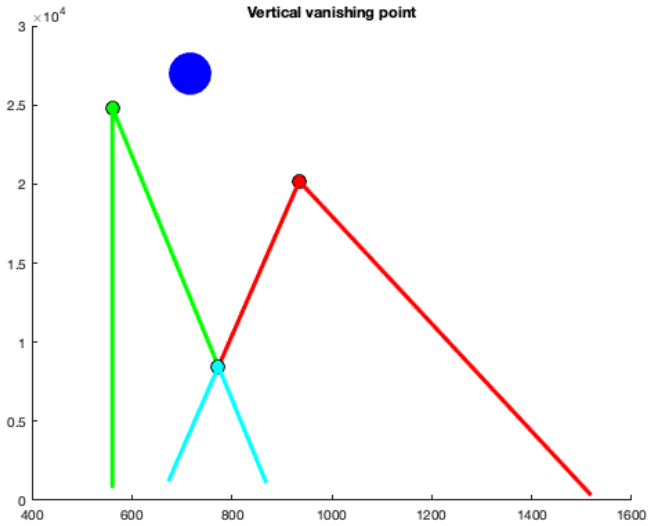
```
% compute vanishing points from pairs of images
v1 = cross(verticalLines(1,:),verticalLines(2,:));
v1 = v1./v1(3);
v2 = cross(verticalLines(3,:),verticalLines(4,:));
v2 = v2./v2(3);
v3 = cross(verticalLines(1,:),verticalLines(4,:));
v3 = v3./v3(3);


figure;
%imshow(im);
hold all;

plot(v1(1),v1(2),'ko','MarkerSize',10,'MarkerFaceColor','r');
line([endPoints(1,1),v1(1)],[endPoints(2,1),v1(2)],'Color','r','LineWidth',3);
line([endPoints(1,2),v1(1)],[endPoints(2,2),v1(2)],'Color','r','LineWidth',3);

plot(v2(1),v2(2),'ko','MarkerSize',10,'MarkerFaceColor','g');
line([endPoints(1,3),v2(1)],[endPoints(2,3),v2(2)],'Color','g','LineWidth',3);
line([endPoints(1,4),v2(1)],[endPoints(2,4),v2(2)],'Color','g','LineWidth',3);

plot(v3(1),v3(2),'ko','MarkerSize',10,'MarkerFaceColor','c');
line([endPoints(1,1),v3(1)],[endPoints(2,1),v3(2)],'Color','c','LineWidth',3);
line([endPoints(1,4),v3(1)],[endPoints(2,4),v3(2)],'Color','c','LineWidth',3);

title('Vertical vanishing point');
% all the vanishing points are different, because they refer to a
% how can we get a unique vanishing point?
```



estimate a unique vertical vanishing point

we can minimize an algebraic error

```
[u,s,v]=svd(verticalLines);
V_vert = v(:,end);
V_vert = V_vert/V_vert(3);


figure(gcf);
hold all;
plot(V_vert(1),V_vert(2),'bo','MarkerSize',30,'MarkerFaceColor','b');

% but this is still suboptimal, it would be better to minimize a geometric
% error
```

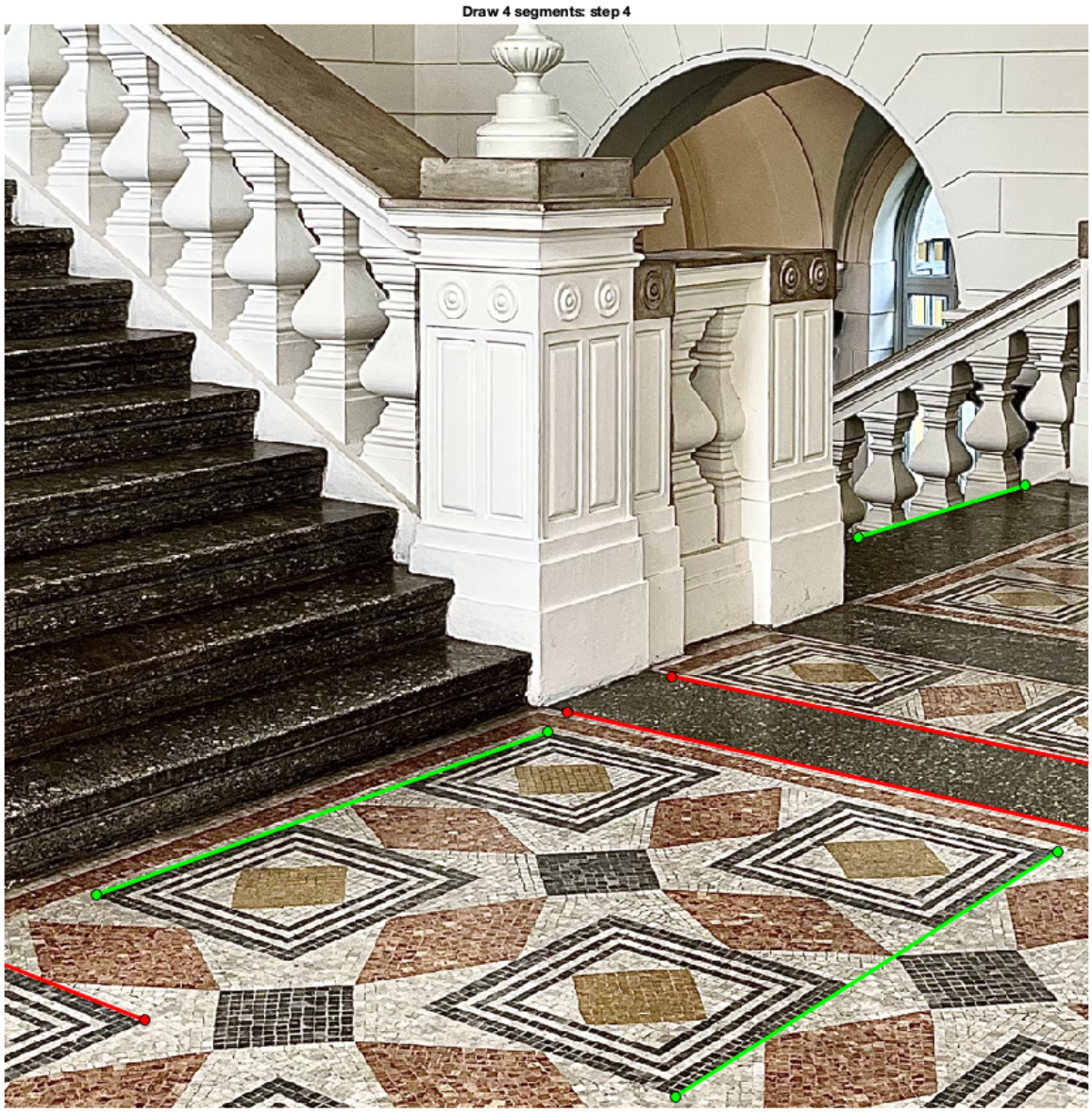


draw segments that comes from parallel lines on the floor

```
f = 2; % number of families of  lines. Two are enough to get the horizon
numSegments = 4;
parallelLines = cell(f,1); % store parallel lines
fprintf(['Draw ', num2str(f) , ' families of parallel segments\n']);
col = 'rgm';
figure;
imshow(im)
for i = 1:f
    count = 1;
    parallelLines{i} = nan(numSegments,3);
    while(count <=numSegments)
        figure(gcf);
        title(['Draw ', num2str(numSegments),' segments: step ',num2str(count) ]);
        seg = drawline('Color',col(i));
```

```
        % convert the end points of the segments into the coefficient of
        % the line
        endPoints = seg.Position;
        % switch to homoegeous coordinates
        a = [endPoints(1,:)';1];
        b = [endPoints(2,:)';1];
        % get the line
        l = cross(a,b);
        l = l./norm(l); %normalize it, just to remove the scale factor
        parallelLines{i}(count, :) = l;
        count = count +1;
    end
    fprintf('Press enter to continue\n');
    pause
end
```

```
Draw 2 families of parallel segments
Press enter to continue
Press enter to continue
```



compute the vanishing points

```
V = nan(3,f);
for i =1:f
    % look for a point that belongs to all the lines in the same family
    % this mean you have to solve a system.
    [u,s,v]=svd(parallelLines{i});
    V(:,i) = v(:,end);
```

```
    V(:,i)= V(:,i)./V(3,i);
end
```

compute the horizon

```
horizon = cross(V(:,1),V(:,2));


figure;
hold all;
for i = 1:f
    plot(V(1,i),V(2,i),'o','Color',col(i),'MarkerSize',20,'MarkerFaceColor',col(i));
end
line(V(1,:),V(2,:),'Color','b','Linewidth',4)
%hold all;
imshow(im);
```

Draw 4 segments: step 4

build the rectification matrix

```
H = [eye(2),zeros(2,1); horizon(:)'];
% we can check that H^-T* imLinfty is the line at infinity in its canonical
% form:
fprintf('The vanishing line is mapped to:\n');
disp(inv(H)'*horizon);
```

```
The vanishing line is mapped to:
     0
     0
     1
```

rectify the image and show the result

```
tform = projective2d(H');
J = imwarp(im,tform);

figure;
imshow(J);
```

```
figure;
imshow(im);
title('select a reference object')
seg1 = drawline('Color','c');
seg2 = drawline('Color','m');

b1 = [seg1.Position(1,:)';1];
t1 = [seg1.Position(2,:)';1];
b2 = [seg2.Position(1,:)';1];
t2 = [seg2.Position(2,:)';1];
```

dispaly the selected points

```
TXT_OFFSET = 20;
figure;
imshow(im);
hold all;
plot(b1(1),b1(2),'c.','MarkerSize',20,'LineWidth',5);
text(b1(1)+TXT_OFFSET,b1(2),'b1','FontSize',30,'Color','c');
plot(t1(1),t1(2),'.','Color','c','MarkerSize',20,'LineWidth',5);
text(t1(1)+TXT_OFFSET,t1(2),'t1','FontSize',30,'Color','c');
plot(b2(1),b2(2),'m.','MarkerSize',20,'LineWidth',5);
text(b2(1)+TXT_OFFSET,b2(2),'b2','FontSize',30,'Color','m');
plot(t2(1),t2(2),'m.','MarkerSize',20,'LineWidth',5);
text(t2(1)+TXT_OFFSET,t2(2),'t2','FontSize',30,'Color','m');
```

compute the relative distance

```
u = cross(cross(b1,b2),horizon);
u = u./u(3);

l2 = cross(V_vert,b2);
t1_tilde = cross(cross(t1,u),l2);
t1_tilde = t1_tilde/t1_tilde(3);

% compute the distance from b2
dist_t1_tilde = norm(t1_tilde-b2);
dist_t2 = norm(t2-b2);
dist_v = norm(V_vert -b2);

% compute 1D projective transfomration mapping the vanishing point to
% infinity
% (0,1)-> (0,1)
% (V_vert,1)->(1,0);
H = [1 0; 1 -dist_v];

H*[dist_v;1]

% compute the distance ratio
d1_d2_ratio = dist_t1_tilde*(dist_v-dist_t2)/(dist_t2*(dist_v-dist_t1_tilde));

%d2 = 1/d1_d2_ratio
lenght1 = 122;
lenght2 =lenght1/d1_d2_ratio
```

```
ans =

   1.0e+04 *

    2.5787
         0


lenght2 =

  124.1234
```

```
TXT_OFFSET = 20;
FNT_SIZE = 30;
c1 = 'w';
c2 = 'm';
c3 = 'b';
figure;
imshow(im);
hold all;
line(V(1,:),V(2,:),'Color',c3,'Linewidth',2);
line([b1(1),t1(1)],[b1(2),t1(2)],'Color',c1,'Linewidth',2);
line([b2(1),t2(1)],[b2(2),t2(2)],'Color',c2,'Linewidth',2);
plot(b1(1),b1(2),'.','Color',c1,'MarkerSize',20,'LineWidth',2);
text(b1(1)+TXT_OFFSET,b1(2),'b1','FontSize',FNT_SIZE,'Color',c1);
plot(t1(1),t1(2),'.','Color',c1,'MarkerSize',20,'LineWidth',5);
text(t1(1)+TXT_OFFSET,t1(2),'t1','FontSize',FNT_SIZE,'Color',c1);
plot(b2(1),b2(2),'.','Color',c2,'MarkerSize',20,'LineWidth',5);
text(b2(1)+TXT_OFFSET,b2(2),'b2','FontSize',FNT_SIZE,'Color',c2);
plot(t2(1),t2(2),'.','Color',c2,'MarkerSize',20,'LineWidth',5);
text(t2(1)-TXT_OFFSET,t2(2)+TXT_OFFSET,'t2','FontSize',30,'Color',c2);
plot(u(1),u(2),'o','Color',c3,'Markersize',20,'MarkerFaceColor',c3)
text(u(1)+TXT_OFFSET,u(2),'u','FontSize',FNT_SIZE,'Color',c3);
plot(t1_tilde(1),t1_tilde(2),'d','Color',c3,'MarkerSize',10,'LineWidth',5,'MarkerFaceColor', c3);
text(t1_tilde(1)+TXT_OFFSET,t1_tilde(2),'t1 tilde','FontSize',30,'Color',c3);
line([b2(1),u(1)],[b2(2),u(2)],'Color',c3,'Linewidth',2);
line([t1_tilde(1),u(1)],[t1_tilde(2),u(2)],'Color',c3,'Linewidth',2);
hold on;
```


```
c1= 'r';
figure;
imshow(im);
hold all;
line([b1(1),t1(1)],[b1(2),t1(2)],'Color',c1,'Linewidth',2);
text(0.5*(b1(1)+t1(1)),0.5*(b1(2)+t1(2)),num2str(lenght1,"%i"),'FontSize',FNT_SIZE,'Color',c1);
line([b2(1),t2(1)],[b2(2),t2(2)],'Color',c1,'Linewidth',2);
text(0.5*(b2(1)+t2(1)),0.5*(b2(2)+t2(2)),num2str(round(lenght2)),'FontSize',FNT_SIZE,'Color',c1);
```

Published with MATLAB® R2021b

Horizon from equally spaced lines and affine rectification

Identify pairs of equispaced parallel planar lines in an image. Compute the vanishing line and use it to perform an affine rectification of the plane.

Reference: Hartely Zisserman 8.23 .\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Image Analysis and Computer Vision Politecnico di Milano

Luca Magri for comments and suggestions please send an email to luca.magri@polimi.it

.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Contents

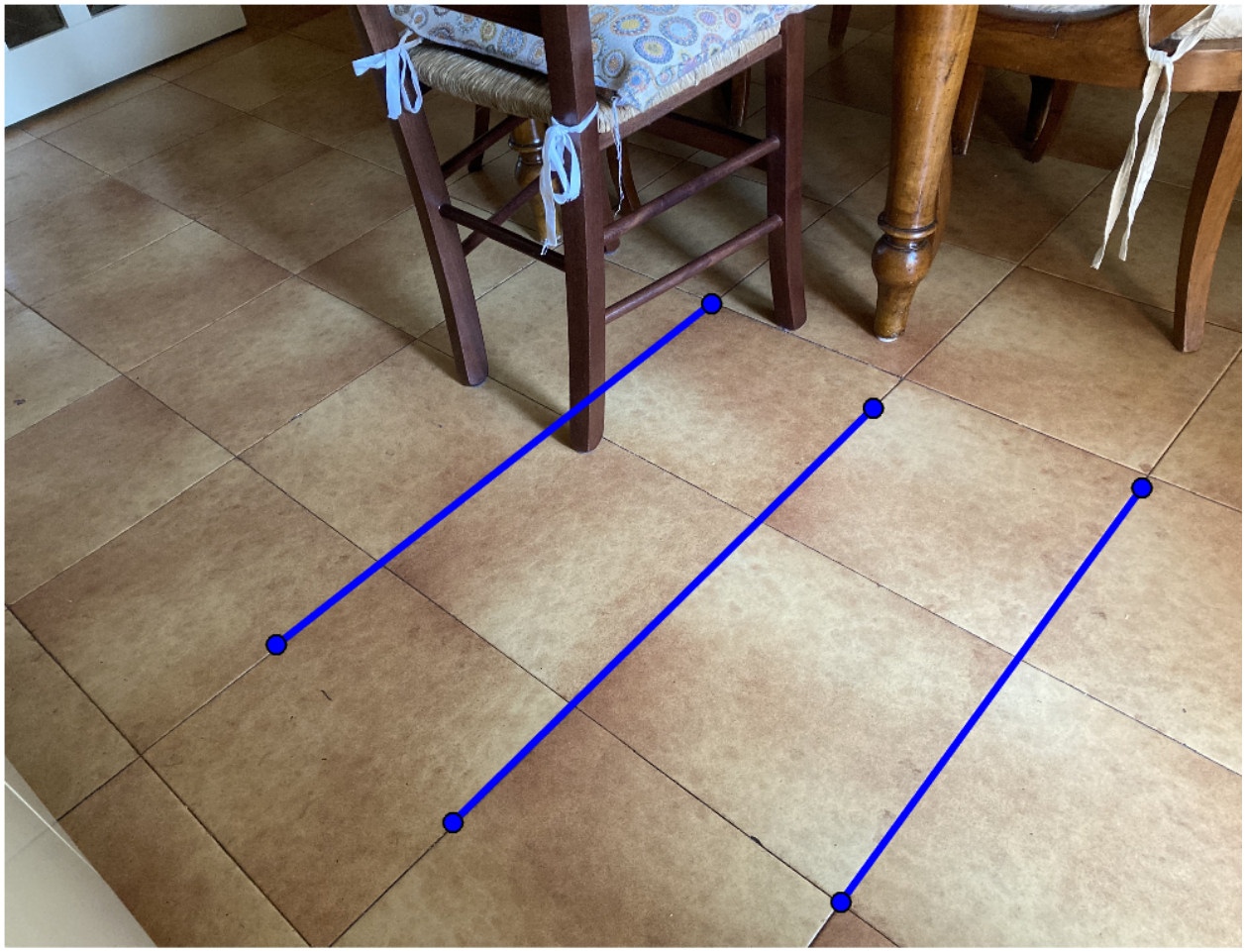

```
clc;
clear;
```

load an image of a plane

```
im = imread("E2_data/floor.jpg");
im = imresize(im,0.5);
```

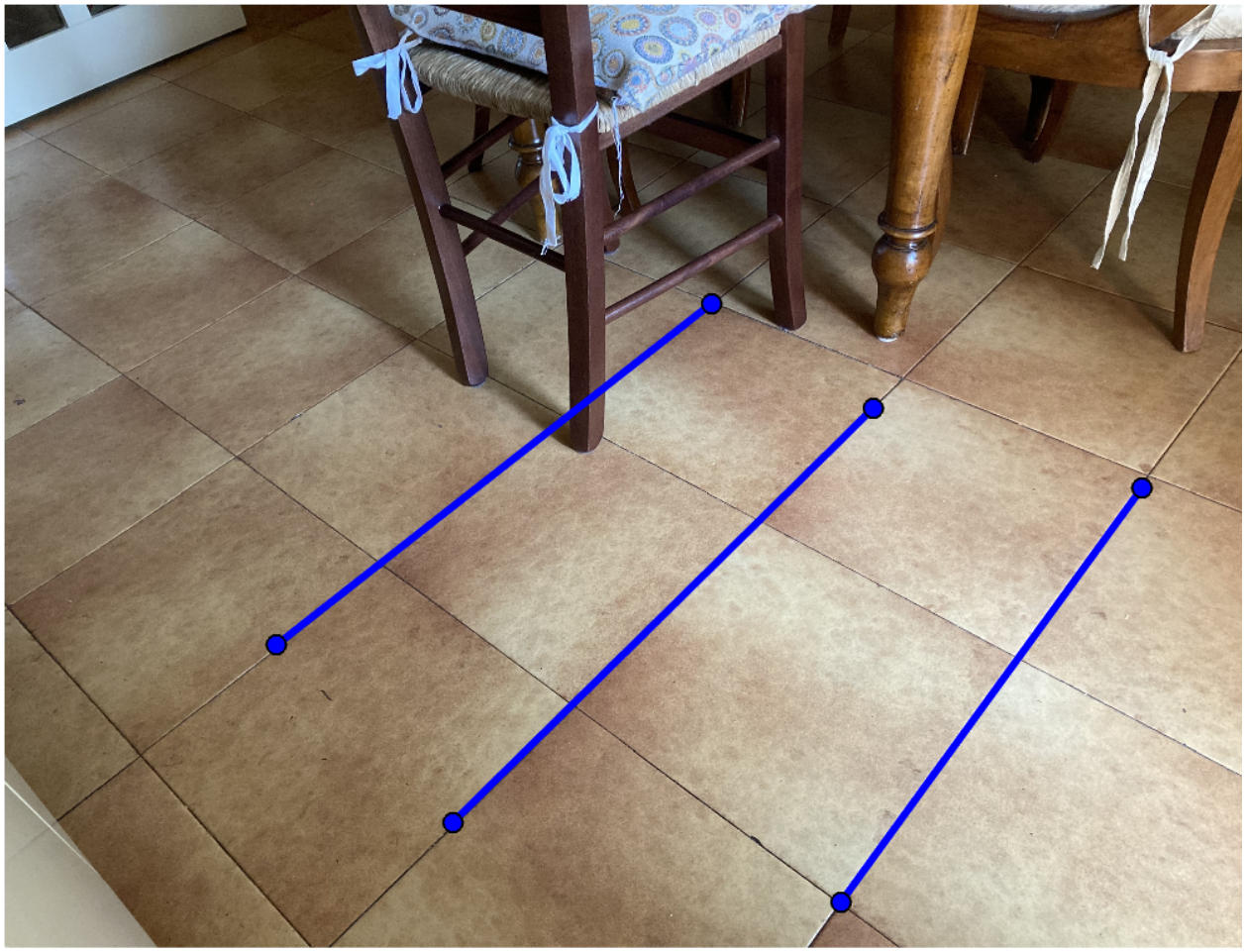
select three lines that are equally spaced in the scene

```
figure; imshow(im);
for i =1:3

    seg{i} = drawline('Color','b');
    % convert the segments to line

    a = [seg{i}.Position(1,:)';1];
    b = [seg{i}.Position(2,:)';1];
    % get the parameters of the line
    lines{i} = cross(a,b);
end
```



compute the horizon

```
l0 = lines{1};
l1 = lines{2};
l2 = lines{3};

l= (cross(l0,l2)'*cross(l1,l2))*l1 +2*(cross(l0,l1)'*cross(l2,l1)*l2);
horizon = l./norm(l);

% identify two points on the line ax+by+c=0
% x = 0 -> y = -c/b
P1 = [0; -horizon(3)/horizon(2)];
% x = size(img,2) -> y = -(ax+c)/b
P2 = [size(im,2); -(horizon(1)*size(im,2)+horizon(3))/horizon(2)];

figure(gcf);
hold on
line([P1(1),P2(1)],[P1(2),P2(2)],'Linewidth',4,'Color','b');
```



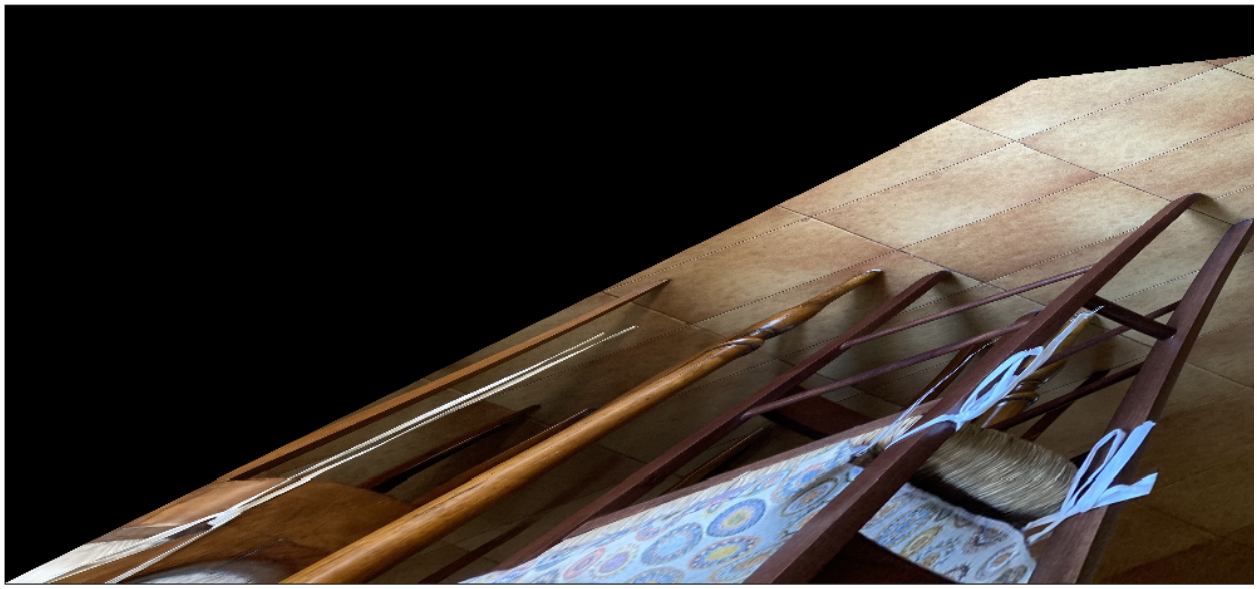
Given the horizon we can rectify the image

```
horizon = horizon./norm(horizon); % super important to regularize the homgraphy
H = [eye(2),zeros(2,1); horizon(:)'];
H = det(H)*H;
% we can check that H^-T* imLinfty is the line at infinity in its canonical
% form:
fprintf('The vanishing line is mapped to:\n');
disp(inv(H)'*horizon);
```

```
The vanishing line is mapped to:
   -0.0000
         0
   -1.0000
```

rectify the image and display it

```
t = maketform( 'projective', H');
J = imtransform(im,t);
figure;
imshow(J);
```



Published with MATLAB® R2021b