ESE 3

RECTIFICATIONS 16/10

we have seen how to estimate homography in application like stitching, CALIBRATION

to estimate camera parameter by using a plane image
↳ recover original plane up to affinity transform,
↓ getting rid of distortion by some method

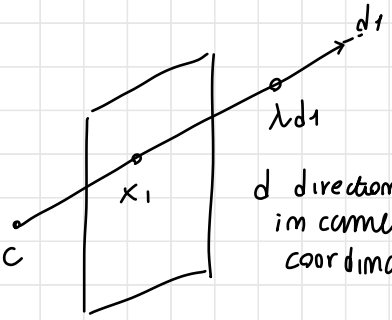
several algebric way to perform Rectification
→ Always consider NUMERICAL INSTANBILITY
you should go with geometry!

What camera CALIBRATION provide?
↳ way to invert image point by pre-multiply by camera matrix inverse

$$x \simeq K[I,0](\lambda d^T, 1)^T \simeq Kd$$

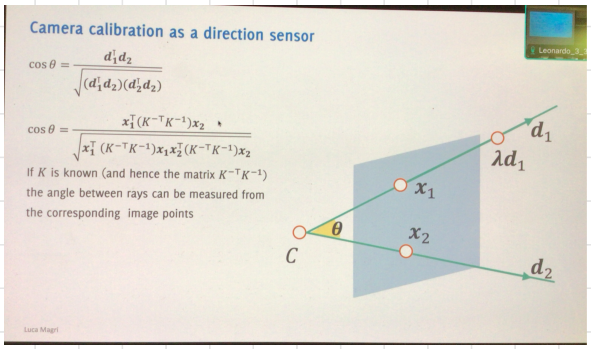
$$d \cong K^{-1}x$$

↑
camera calibration matrix maps point x to ray's direction $d = K^{-1}x$ (in camera frame)



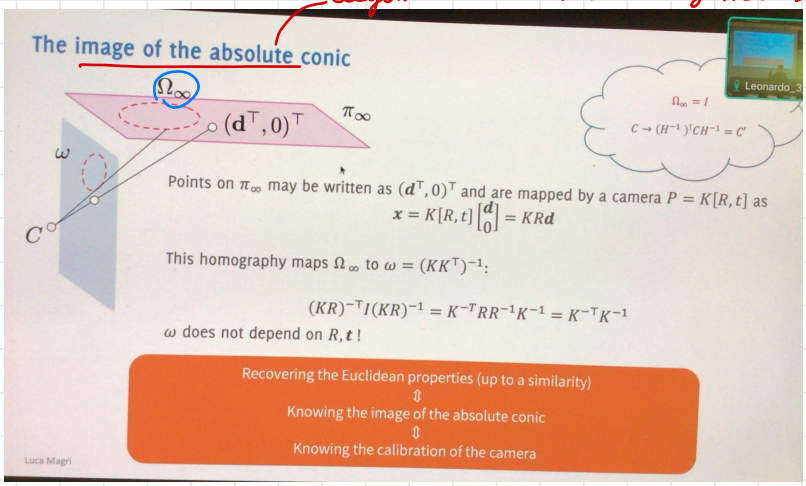
d direction in camera coordinate

than you can use this information to measure angles between optical rays



this MATRIX is built upon calibration matrix K, which play an important role

algebraic device, to recognize some Affine property inside an image



"image of the Absolute comic"

Absolute comic $\simeq I_{3\times3}$ identy $\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$

used to measure angle, INNER product between two vector

$U, V : I$ take $U^\top I V = \cos(\vartheta)$

↑ by define matrix you can define angles

"Absolute" because it is like a comic living in a plane at infinity

"comic" because as 3×3 can be as a CONIC C core

"image" because you are going to map it with your camera → project comic into a plane

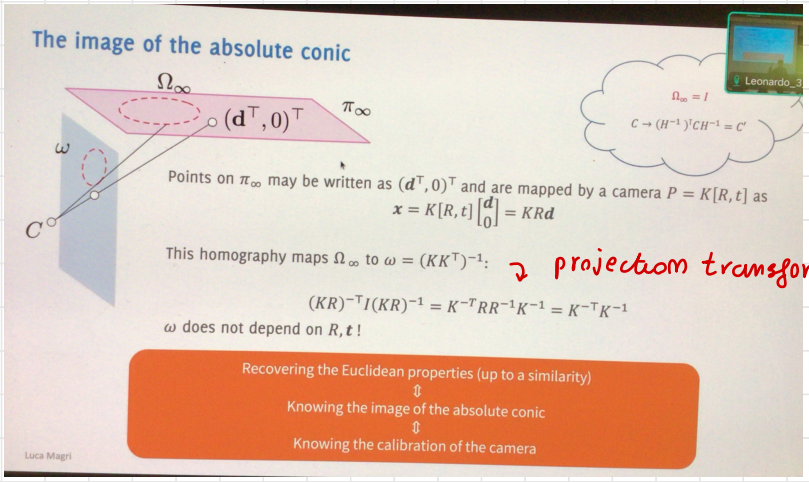
$C \longrightarrow$ map $\longrightarrow$ new C' as picture of Absolute comic!

doing math, can be proven that the Absolute comic Ω_∞, the mapping is an homography (PROJECTION)

↓

you know expression of $\Omega_\infty = I_{3\times3}$

you can then apply projection



↓ projection transform Ω_∞ on $K^{-T} K^{-1}$

NOT depending on ROTATION / TRANSLATION but only on camera calibration

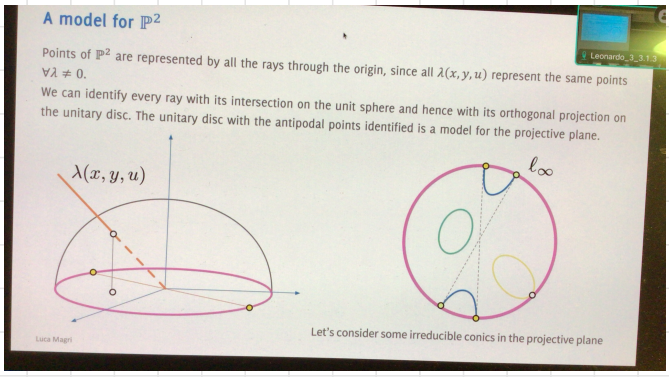
so, the image of absolute conic JUST depends on internal camera parameter

and Recovery camera parameter K, is equivalent to knowing image of Ω_∞, than it is equivalent to recover the euclidean property up to a similarity of your image.

↑

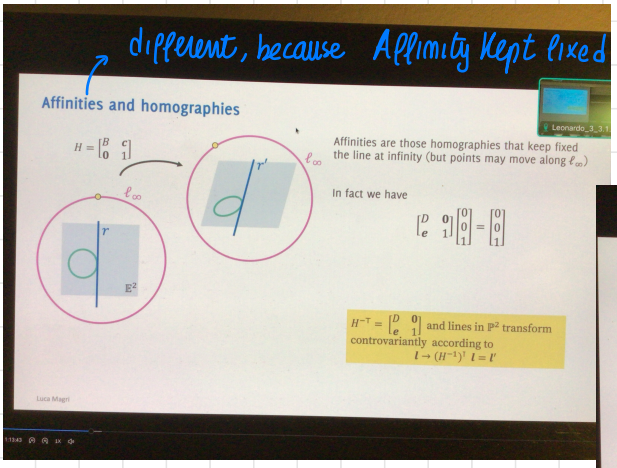
we will work with these elements...

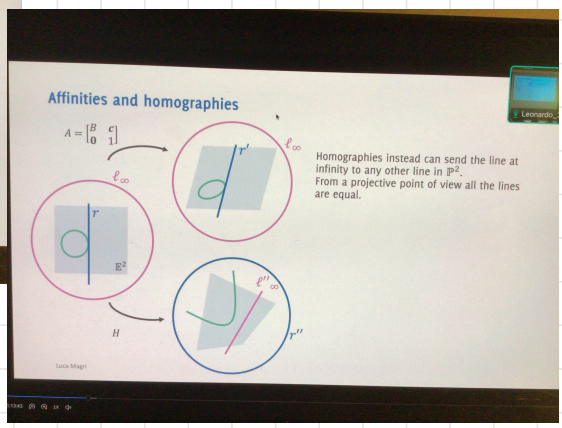
⇒ In our image we look for 2D euclidean invariant to recover everything up to certain transformation



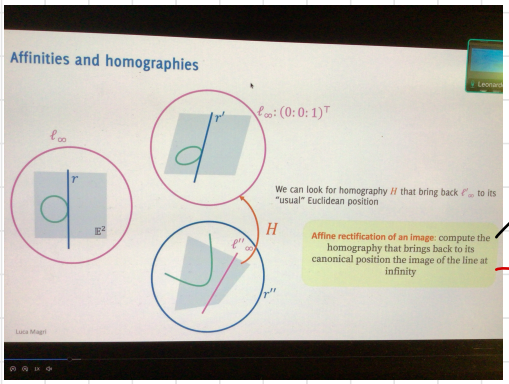
← model of PROJECTIVE plane

different, because Affinity kept fixed the image of ℓ_∞ while homography can bring ℓ_∞ somewhere else in the plane



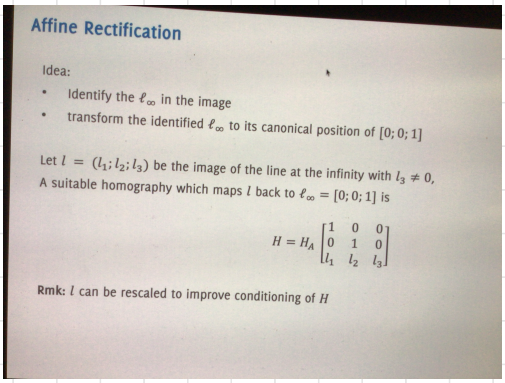


process of AFFINE RECTIFICATION given an image...



take image of ℓ_∞, ℓ'_∞ and bring it back to its canonical position

this can be done

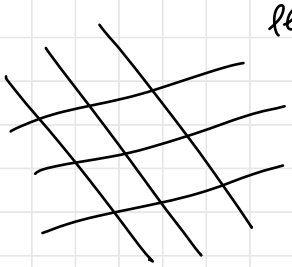


from an image, using PRIOR KNOWLEDGE, like knowing that some lines are parallel

↓

than I use this information to identify image of l_∞

floor (I know lines are parallel)



I identify HORIZON, as e'_∞ ...

once I have e'_∞ I map into l_∞

by putting in the last row of H

the coefficents of e'_∞

Horizon and affine rectification

from pairs of parallel lines.

Identify pairs of parallel lines in the image of a plane and get their vanishing points. Compute the vanishing line and use it to perform an affine rectification of the plane.

Credits: Giacomo Boracchi Edits: Luca Magri, October 2023, for comments and suggestions write 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 four points
% important select the point clockwise or counterclockwise

figure;
imshow(im);
hold on;
[x, y] = getpts();
a = [x(1); y(1); 1];
b = [x(2); y(2); 1];
c = [x(3); y(3); 1];
d = [x(4); y(4); 1];

FNT_SZ = 20;

text(a(1), a(2), 'a', 'FontSize', FNT_SZ, 'Color', 'b')
text(b(1), b(2), 'b', 'FontSize', FNT_SZ, 'Color', 'b')
text(c(1), c(2), 'c', 'FontSize', FNT_SZ, 'Color', 'b')
text(d(1), d(2), 'd', 'FontSize', FNT_SZ, 'Color', 'b')
```

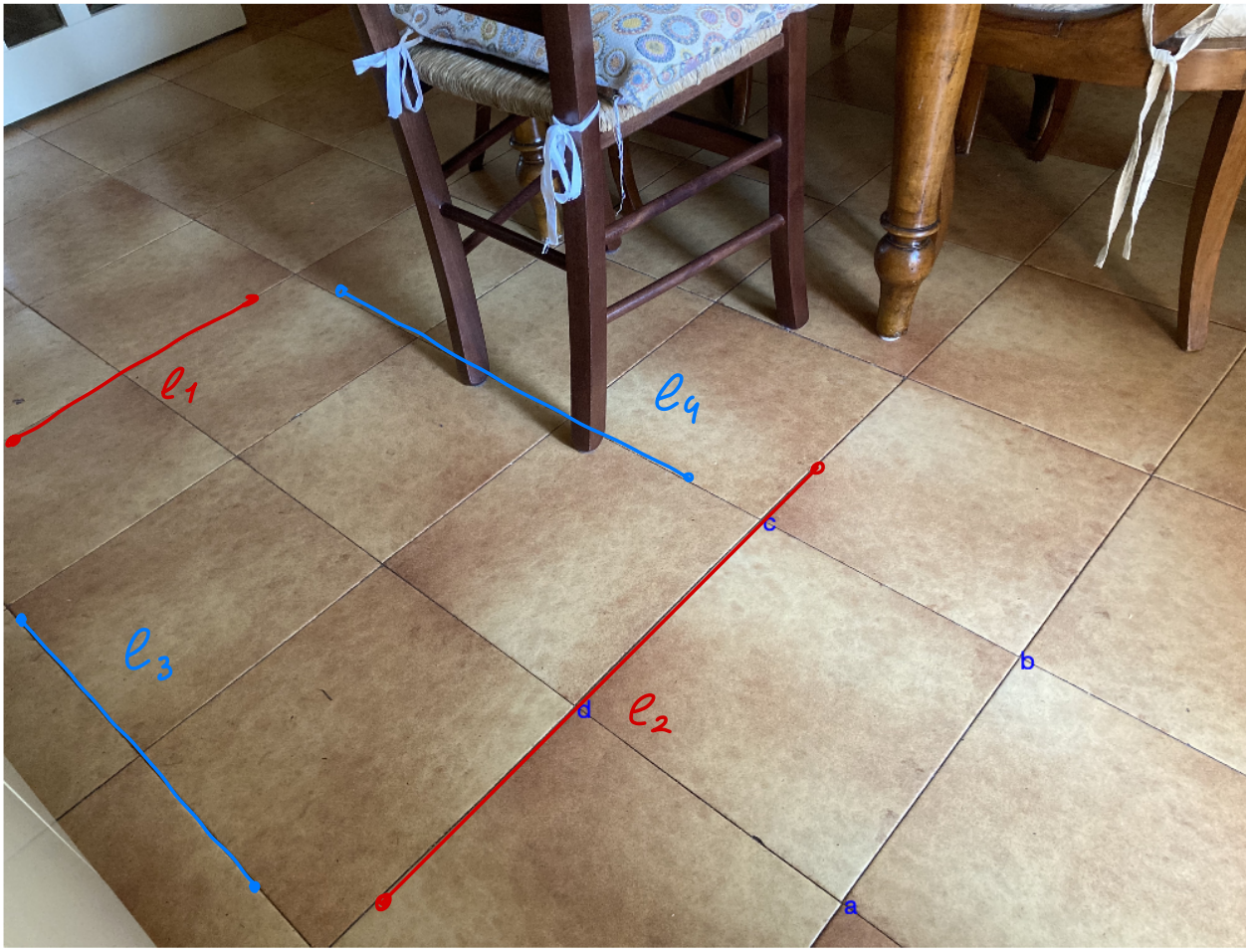
I need to identify two pairs of parallel lines selecting points

↓

better far away lines to have a better conditioned PROBLEM

↓

IF too close to eachother, x'_b tends to be far away, you get big numbers and so INSTABILITY



$\ell_3 \parallel \ell_4$

$\ell_1 \parallel \ell_2$

compute the lines passing through the points

```
lab = cross(a, b);
lbc = cross(b, c);
lcd = cross(c, d);
lda = cross(d, a);
```

intersect lines,

we can get vanishing points from pairs of parallel line in closed form

```
v1 = cross(lab, lcd);
v2 = cross(lbc, lda);

% remember these have to be normalized before plotting them
v1 = v1/v1(3);
v2 = v2/v2(3);
```

Compute the horizon (the vanishing line of the plane)

the horizon is just the line that passes thorugh the vanishing points

plot vanishing points, x'_∞ and so vanishing line e'_∞

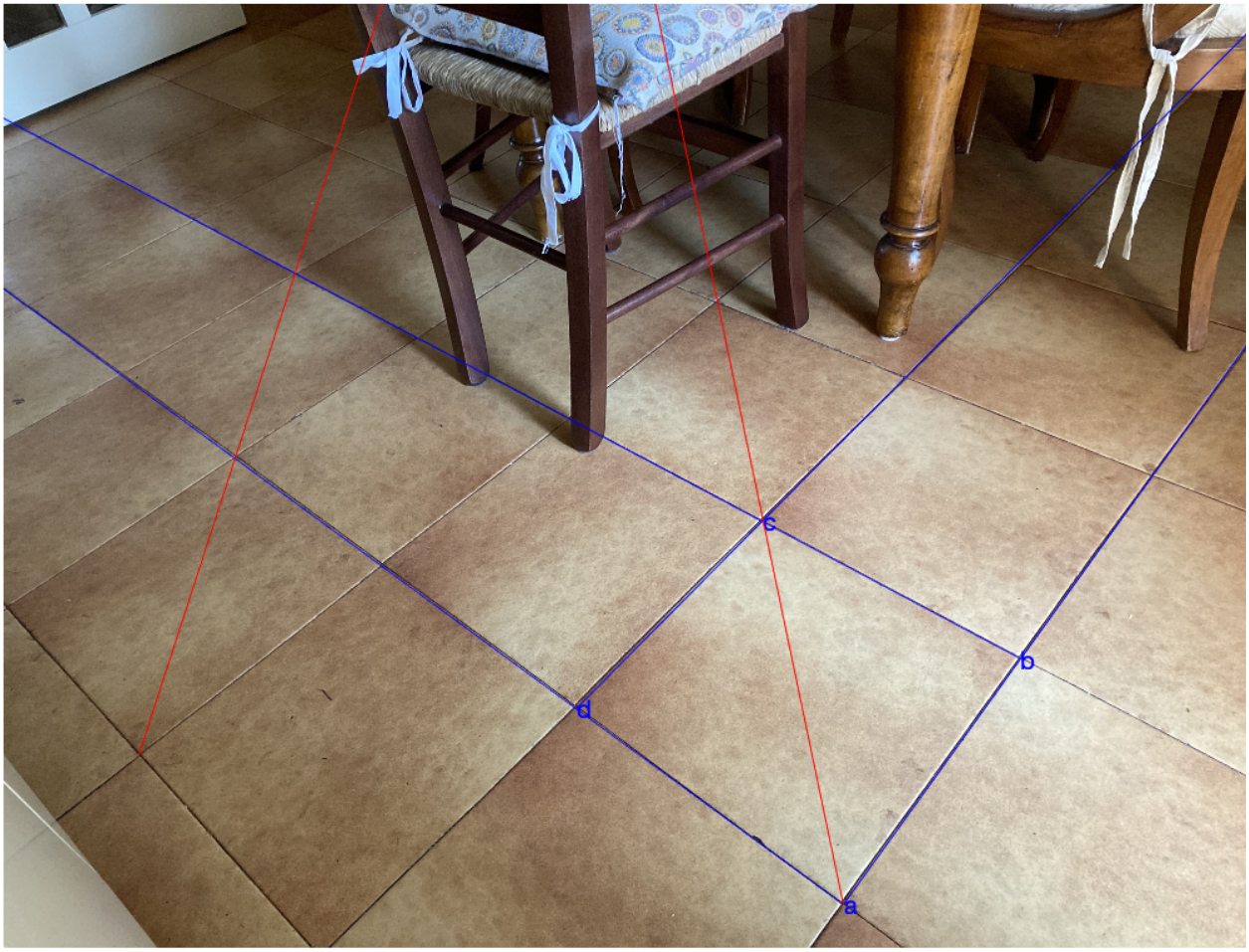
image of e_∞ NO more an infinite line...

```
horizon = cross(v1, v2);


% display the result
plot([a(1), v1( 1)], [a(2), v1(2)], 'b');
plot([d(1), v1(1)], [d(2), v1(2)], 'b');
plot([b(1), v1(1)], [b(2), v1(2)], 'b');
plot([c(1), v1(1)], [c(2), v1(2)], 'b');

plot([a(1), v2(1)], [a(2), v2(2)], 'b');
plot([c(1), v2(1)], [c(2), v2(2)], 'b');
plot([b(1), v2(1)], [b(2), v2(2)], 'b');
plot([d(1), v2(1)], [d(2), v2(2)], 'b');

plot([v1(1), v2(1)], [v1(2), v2(2)], 'b--')
text(v1(1), v1(2), 'v1', 'FontSize', FNT_SZ, 'Color', 'b')
text(v2(1), v2(2), 'v2', 'FontSize', FNT_SZ, 'Color', 'b')

hold off
```

Select another pair of parallel lines


```
disp('Select another pair of parallel lines');
figure(gcf),
hold on;
[x, y] = getpts();
e = [x(1); y(1); 1];
f = [x(2); y(2); 1];
g = [x(3); y(3); 1];
h = [x(4); y(4); 1];


lef = cross(e, f);
lgh = cross(g, h);

v3 = cross(lef, lgh);
v3 = v3 / v3(3);


horizon' * v3
% it is ok to have an higher error, we have fitted the vanishing points
% from noisy estimates, moreover we are not taking into consideration
% radial distortion etc... we are not dealing with a perfect pinhole
% camera...

plot([e(1), v3(1)], [e(2), v3(2)], 'r');
plot([g(1), v3(1)], [g(2), v3(2)], 'r');
text(v3(1), v3(2), 'v3', 'FontSize', FNT_SZ, 'Color', 'b');

% what happens if the camera is not perspective but orthographic?
```

```
ans =

  -3.0317e+05
```



check that e_∞ is mapped by H to its e_∞ initial...

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);
```

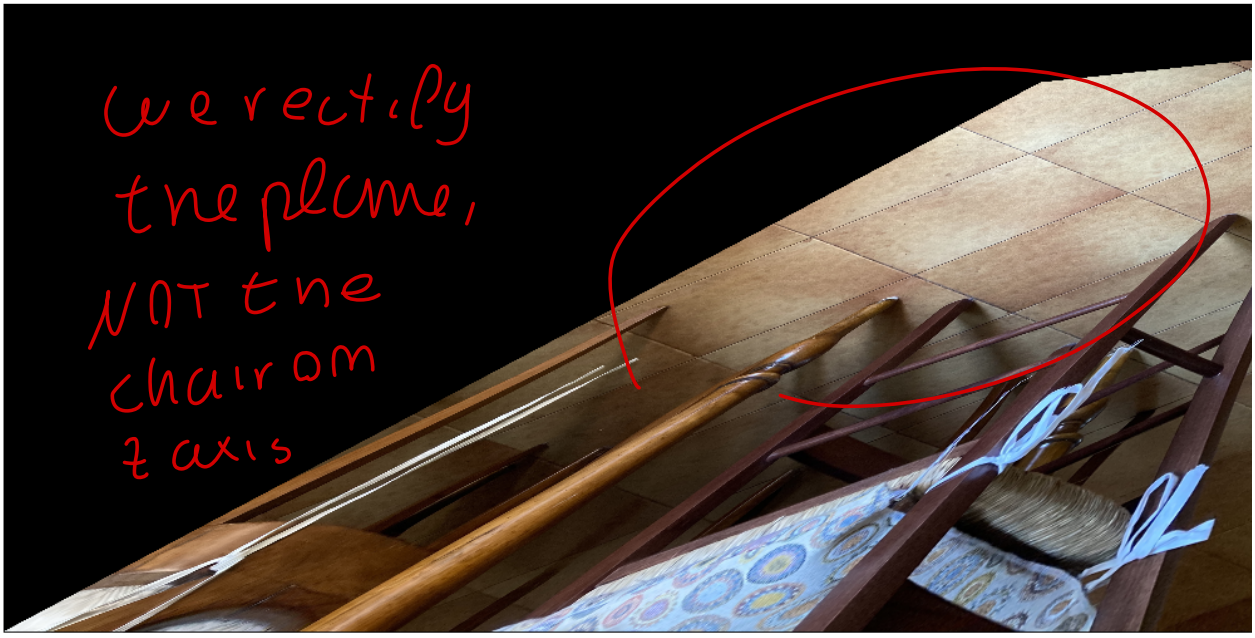
I expect $H^{-1} \cdot$ horizon e_∞ be at e_∞ standard line at inf canonical position

```
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);
% how are the lines in the recrtified images.
% Are the tile of the floor square? Why? Why not?
% What happen to the points that doesn't lie on ground plane?
```

← show the figure...



Published with MATLAB® R2021b

image recovered up to an affinity !

↓

lines parallel on floor are still parallel here... UP to affinity, in fact the ratio of sizes (shape) are NOT equal !

recover info NOT up to similarity !

←

NO similarity information

Rectangle → squares... !

IM SIMILARITY RECONSTRUCTION

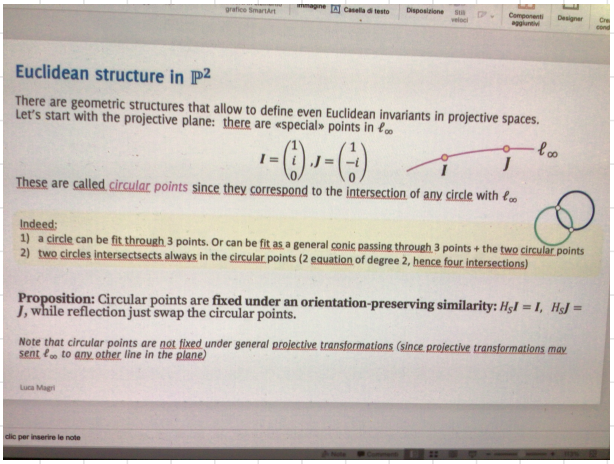
↓ we work with CIRCULAR POINTS, I, J special planar points with canonical position.

↓

they encode similarity information

↑

parallel the vanishing line properties



Euclidean structure in $\mathbb{P}^2$

There are geometric structures that allow to define even Euclidean invariants in projective spaces. Let's start with the projective plane: there are «special» points in ℓ_∞

$$I = \begin{pmatrix} 1 \\ i \\ 0 \end{pmatrix}, J = \begin{pmatrix} 1 \\ -i \\ 0 \end{pmatrix}$$

These are called circular points since they correspond to the intersection of any circle with ℓ_∞

Indeed:
1) a circle can be fit through 3 points. Or can be fit as a general conic passing through 3 points + the two circular points
2) two circles intersectsects always in the circular points (2 equation of degree 2, hence four intersections)

Proposition: Circular points are **fixed under an orientation-preserving similarity:** $H_S I = I$, $H_S J = J$, while reflection just swap the circular points.

Note that circular points are not fixed under general projective transformations (since projective transformations may sent ℓ_∞ to any other line in the plane)

Luca Magri

clic per inserire le note

IN PROJECTIVE PLANE:

ℓ_∞ special position $[0 \;\; 0 \;\; 1]^T$ allow us to recover affine information moving ℓ'_∞ to ℓ_∞.

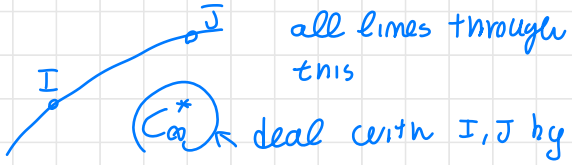
SIMILARLY, having I', J' CIRCULAR POINTS projection encoding metric information

↓

we recover this moving I', J' to I, J by a proper homography

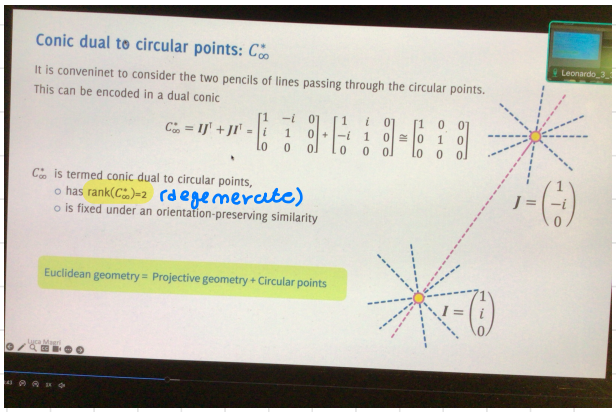
↓

for Algebraic reason, we work with $C^{*\prime}_\infty$, dual conic, as PENCIL of CONICS passing through CIRCULAR POINTS



all lines through this

← deal with I, J by C^*_∞ only!



← Algebraic device to consider I, J jointly

↓

it is a set of lines, pencil of lines through point (in dual space)

↑

dual conic

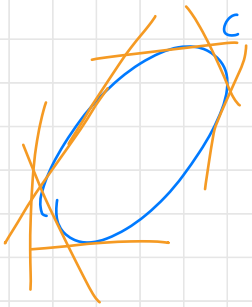
(conic in dual space ~ lines)

NOT full rank! it is degenerate conic and envelop..

↑

so it is a pencil of lines going through I, J

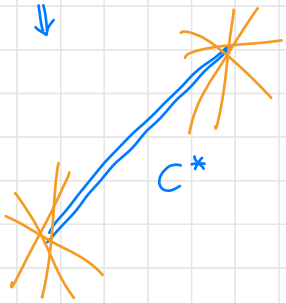
→ way to represent I, J in simple way



consider all the tangent to ellipse...

then squeeze ellipse

(by decreasing one axis)



collapsed ellipse ~ two pencil of lines

DUAL CONIC!

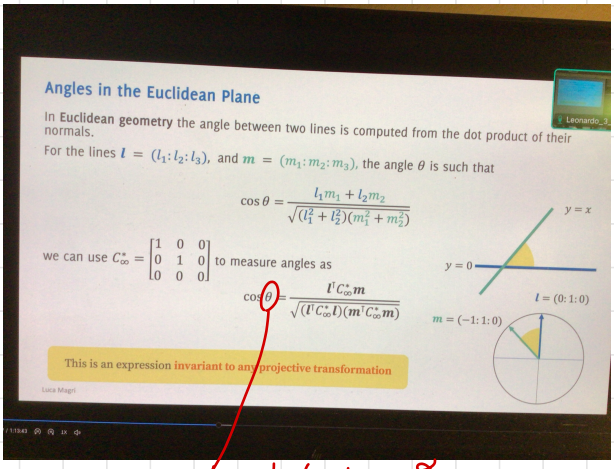
↑

PROJ GEOMETRY + I, J } provide all Euclidean information

↓

We need to identify I', J' in our image

or similarly $C_\infty^{*\prime}$ image of C_∞^*



you need to use θ on real image! from PRIOR KNOWLEDGE

→ Angles between lines are measured by using C_∞^*

INVARIANT under PROJ transform

↓

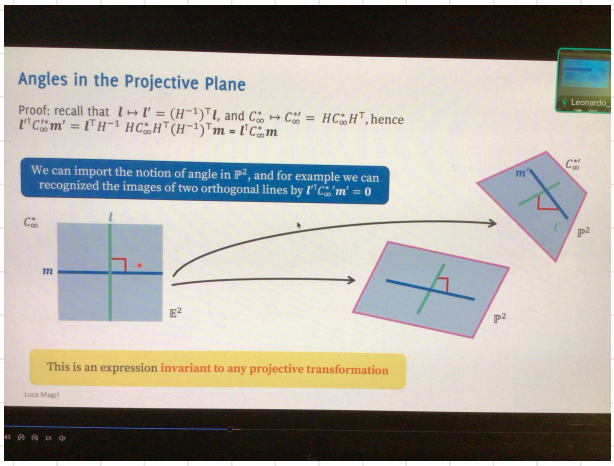
then, if we can measure angle θ in ORIGINAL PLANE

constrain on C_∞^* estimated

↓

from C_∞^* you can find H_{SIM}

you can use $\theta = 90°$ perpendicular lines knowledge to constraint C_∞^*

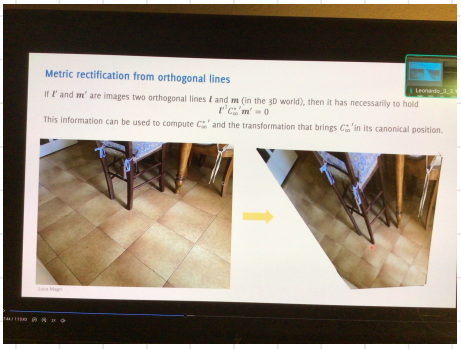


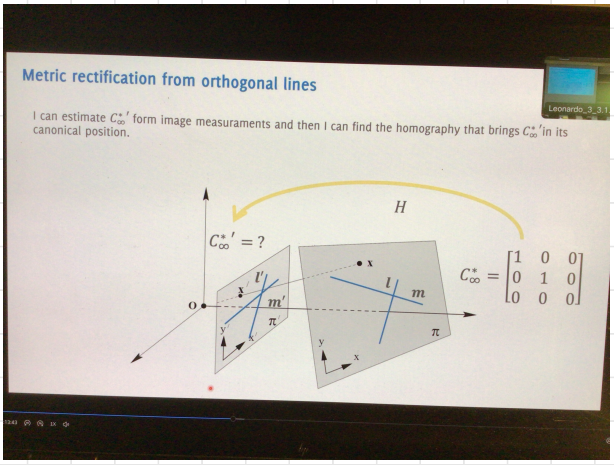
you know two lines are originally ORTHOGONAL

↳ you impose that in new image, since formula is invariant wrt projectivity, IF we use $C_\infty^{*\prime}$ to measure θ, constrained conic

↓

you can move to similarity





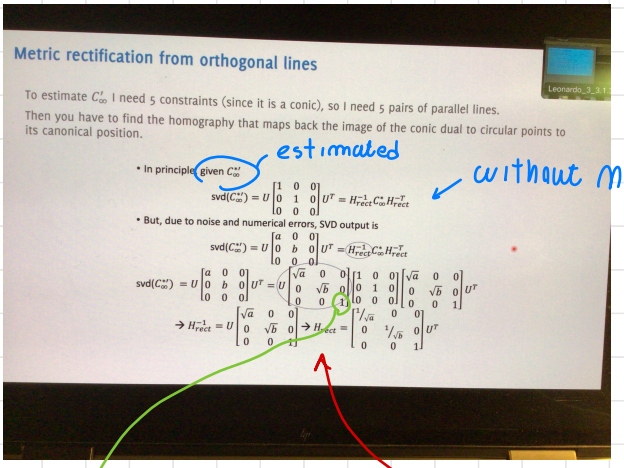
taking lines im image (KNOWN) than force constraint

$C^{*\prime}_\infty$ I estimate

↓

then I need to find H

↓ IM PRACTICE



estimated

without noise

$U = V^T$ because symm SVD

Just using SVD im theory you get homography of rectification

↓

BUT since NOISY! we don't get that SVD are equal to 1 $\forall i \neq 1$

we can just normalize by premultiply by sqrt(b), to find real rectifyer homography

add 1 because it is RANK deficient, we want to make it INVERTIBLE by adding 1

Its enough to get $C^{*\prime}_\infty$ ← its like a CONIC estimation we need 5 constraints

$C_\infty^{*'}$ ← IF I have pairs of parallel / orthogonal lines

↓

I constraint shape of $C_\infty^{*'}$ ⇒ then SVD

↓

then Hrect is found!

Affine rectification from pairs of parrallel lines

perform an affine rectification of an image. Given two pairs of image of parallel lines, the script find the image of the line at infinity as the ones passing through the vanishing points. The homography that maps the line at infinity to its canonical form: (0: 0: 1) rectify the image up to an affinity transformation.

Luca Magri
Politecnico di Milano
2020

Contents

load the image

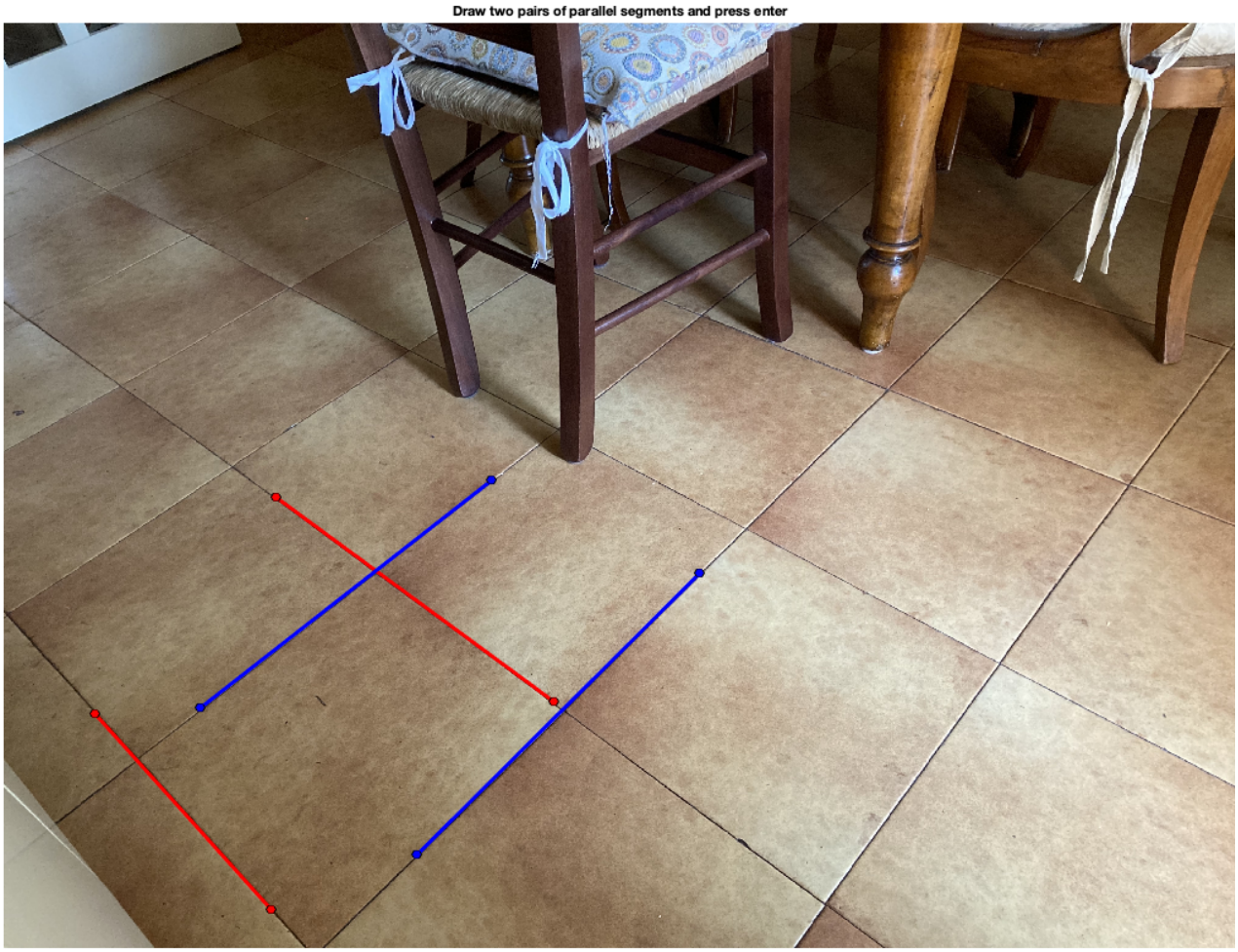
```
clear;
close all;
img = imread('E4_data/img1.JPG');
figure; imshow(img);
title('Draw two pairs of parallel segments and press enter')
```

select two pairs of segments that are image of parallel lines in the scene

```
fprintf('Draw parallel segments\n');

% select a first pair of segments (images of 2 parallel lines)
segment1 = drawline('Color','red');
segment2 = drawline('Color','red');


% select a second pair of segments (images of 2 parallel lines)
segment3 = drawline('Color','blue');
segment4 = drawline('Color','blue');
```

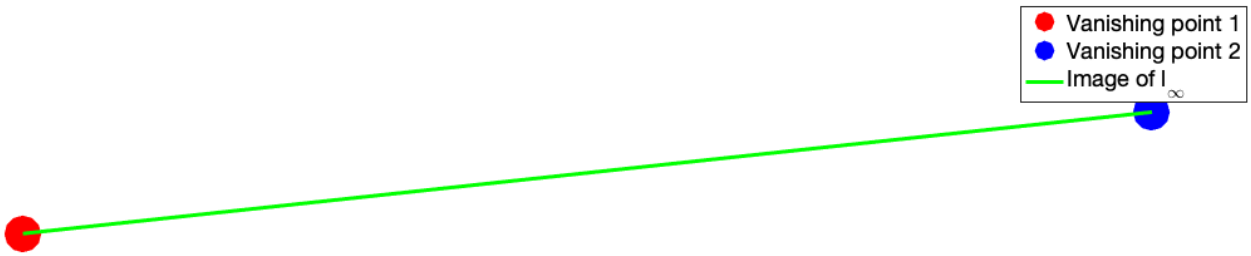
```
fprintf('Press enter to continue\n');
pause
```

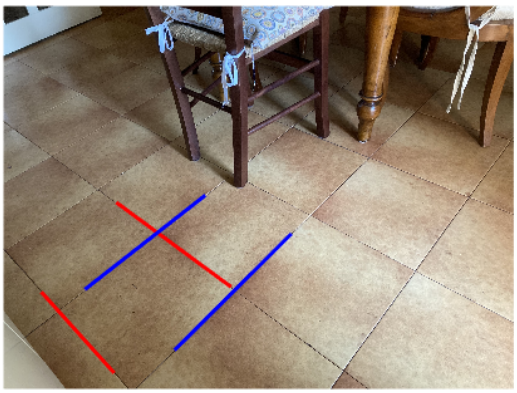
```
Draw parallel segments
Press enter to continue
```



compute the image of the line at infinity

```
l1 = segToLine(segment1.Position);
l2 = segToLine(segment2.Position);

m1 = segToLine(segment3.Position);
m2 = segToLine(segment4.Position);

% compute the vanishing points
L = cross(l1,l2);
L = L./L(3);
M = cross(m1,m2);
M = M./M(3);

% compute the image of the line at infinity
imLinfty = cross(L,M);
imLinfty = imLinfty./(imLinfty(3));

% dispaly the selection
figure;
hold all;
% plot vanishing points
plot(L(1),L(2),'r.','MarkerSize',100);
plot(M(1),M(2),'b.','MarkerSize',100);
imshow(img);
% plot vanishing line
line([L(1),M(1)],[L(2),M(2)],'Color','Green','Linewidth',3);
% plot selected segments
line([segment1.Position(1,1),segment1.Position(2,1)],[segment1.Position(1,2),segment1.Position(2,2)],'Color','red','Linewidth',3);
line([segment2.Position(1,1),segment2.Position(2,1)],[segment2.Position(1,2),segment2.Position(2,2)],'Color','red','Linewidth',3);
line([segment3.Position(1,1),segment3.Position(2,1)],[segment3.Position(1,2),segment3.Position(2,2)],'Color','blue','Linewidth',3);
line([segment4.Position(1,1),segment4.Position(2,1)],[segment4.Position(1,2),segment4.Position(2,2)],'Color','blue','Linewidth',3);
hold off;
legend('Vanishing point 1', 'Vanishing point 2','Image of l_\infty');
set(gca,'FontSize',20)
```





build the rectification matrix

```
H = [eye(2),zeros(2,1); imLinfty(:)'];
% we can check that H^-T* imLinfty is the line at infinity in its canonical
% form:

% compute the rectifying matrix
fprintf('The vanishing line is mapped to:\n');
disp(inv(H)'*imLinfty);
```

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

rectify the image and show the result

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

figure;
imshow(J);
imwrite(J,'images/affRect.JPG');
```

```
Error using imwrite (line 548)
Unable to open file "images/affRect.JPG" for writing.  You might not have write permission.

Error in E4A_affine_rectification (line 89)
imwrite(J,'images/affRect.JPG');
```

```
figure;
imshowpair(img,J,'diff');
```

Published with MATLAB® R2021b

Contents

Metric rectification from orthogonal lines

perform metric rectification of an image. Given at least 5 pairs of image of orthogonal lines, the script finds the image of the dual conic to circular points and computes the homography that brings it back to its canonical form

Luca Magri
Politecnico di Milano
2020

5 constraints are needed to estimate a conic

```
clear;
close all;
img = imread('E4_data/img1.JPG');
```

```
figure;
imshow(img);
numConstraints = 5; %>=5
hold all;
fprintf('Draw 5 pairs of orthogonal segments\n');
count = 1;
A = zeros(numConstraints,6);

% select pairs of orthogonal segments
while (count <=numConstraints)
    figure(gcf);
    title(['Draw ', num2str(numConstraints),' pairs of orthogonal segments: step ',num2str(count) ]);
    col = 'rgbcmykwrgbcmykw';
    segment1 = drawline('Color',col(count));
    segment2 = drawline('Color',col(count));

    l = segToLine(segment1.Position);
    m = segToLine(segment2.Position);

    % each pair of orthogonal lines gives rise to a constraint on the image
    % of the dual conic of principal points imDCCP
    % [l(1)*m(1),0.5*(l(1)*m(2)+l(2)*m(1)),l(2)*m(2),0.5*(l(1)*m(3)+l(3)*m(1))
    %  0.5*(l(2)*m(3)+l(3)*m(2)), l(3)*m(3)]*v = 0
    % store the constraints in a matrix A
    A(count,:) = [l(1)*m(1),0.5*(l(1)*m(2)+l(2)*m(1)),l(2)*m(2),...
        0.5*(l(1)*m(3)+l(3)*m(1)),  0.5*(l(2)*m(3)+l(3)*m(2)), l(3)*m(3)];
    count = count+1;
end
```

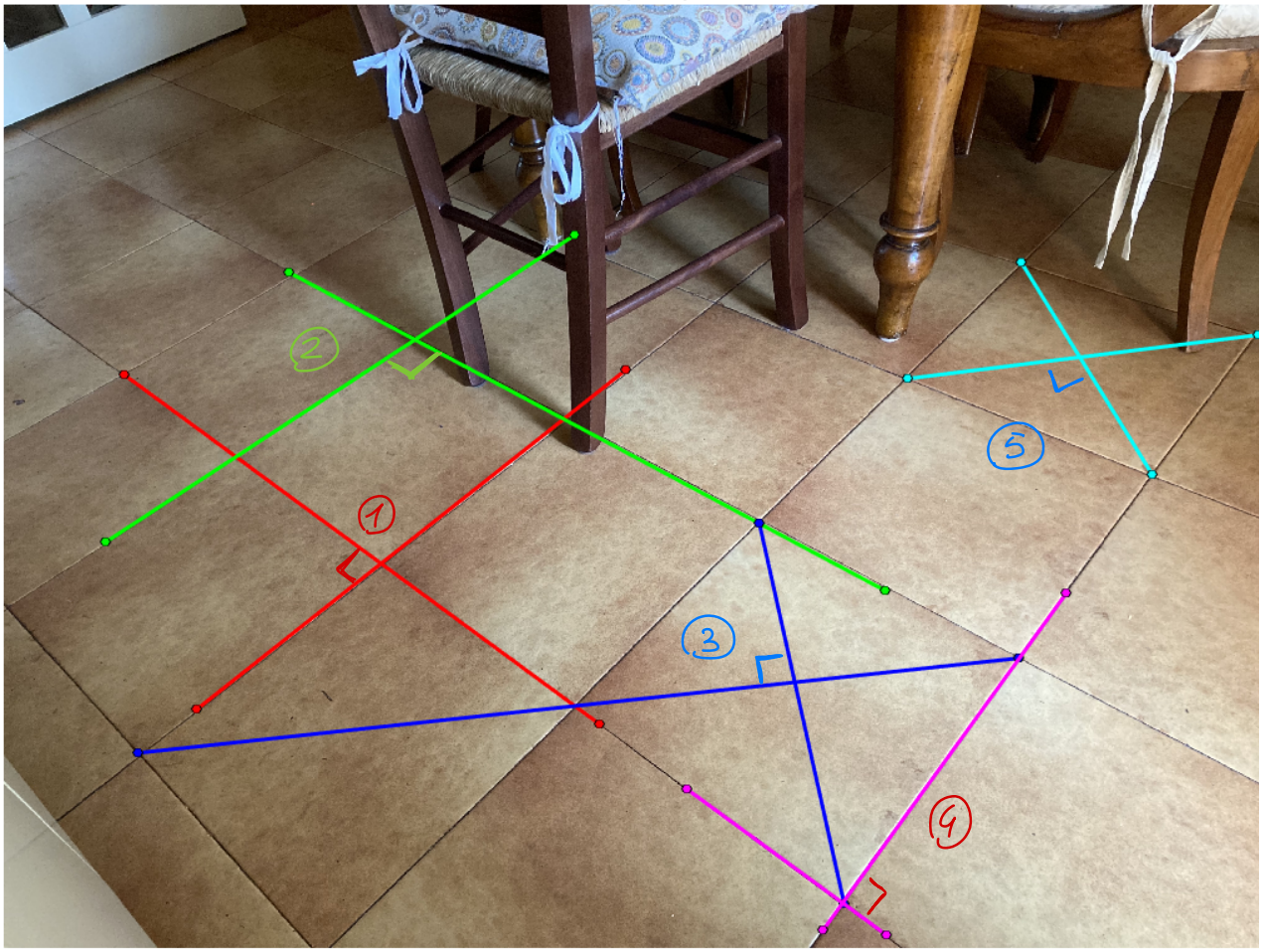
5 are enough, if more than 5 is just more robust...

build coefficient matrix using $\cos(\theta) = 0$

linearize coefficient of matrix as done in CONIC ESTIMATION in DLT

```
Draw 5 pairs of orthogonal segments
```

Draw 5 pairs of orthogonal segments: step 5



↓ after selected parallel lines, estimate conics

Compute the imDCCP image of the dual conic to circular points

```
[~,~,v] = svd(A); %
sol = v(:,end); %sol = (a,b,c,d,e,f)  [a,b/2,d/2; b/2,c,e/2; d/2 e/2 f];
imDCCP = [sol(1)  , sol(2)/2, sol(4)/2;...
    sol(2)/2, sol(3)  , sol(5)/2;...
    sol(4)/2, sol(5)/2  sol(6)];
```

compute the rectifying homography

```
[U,D,V] = svd(imDCCP);
D(3,3) = 1;
A = U*sqrt(D);
```

rectify homography as A^{-1}

↓ see if it is mapped to correct $C_\infty^{*'} \rightarrow C_\infty$ by A

```
C = [eye(2),zeros(2,1);zeros(1,3)];
min(norm(A*C*A' - imDCCP),norm(A*C*A' + imDCCP))

H = inv(A); % rectifying homography
min(norm(H*imDCCP*H'./norm(H*imDCCP*H') - C./norm(C)),norm(H*imDCCP*H'./norm(H*imDCCP*H') + C./norm(C)))
```

I check the difference by norm because of scale ambiguity

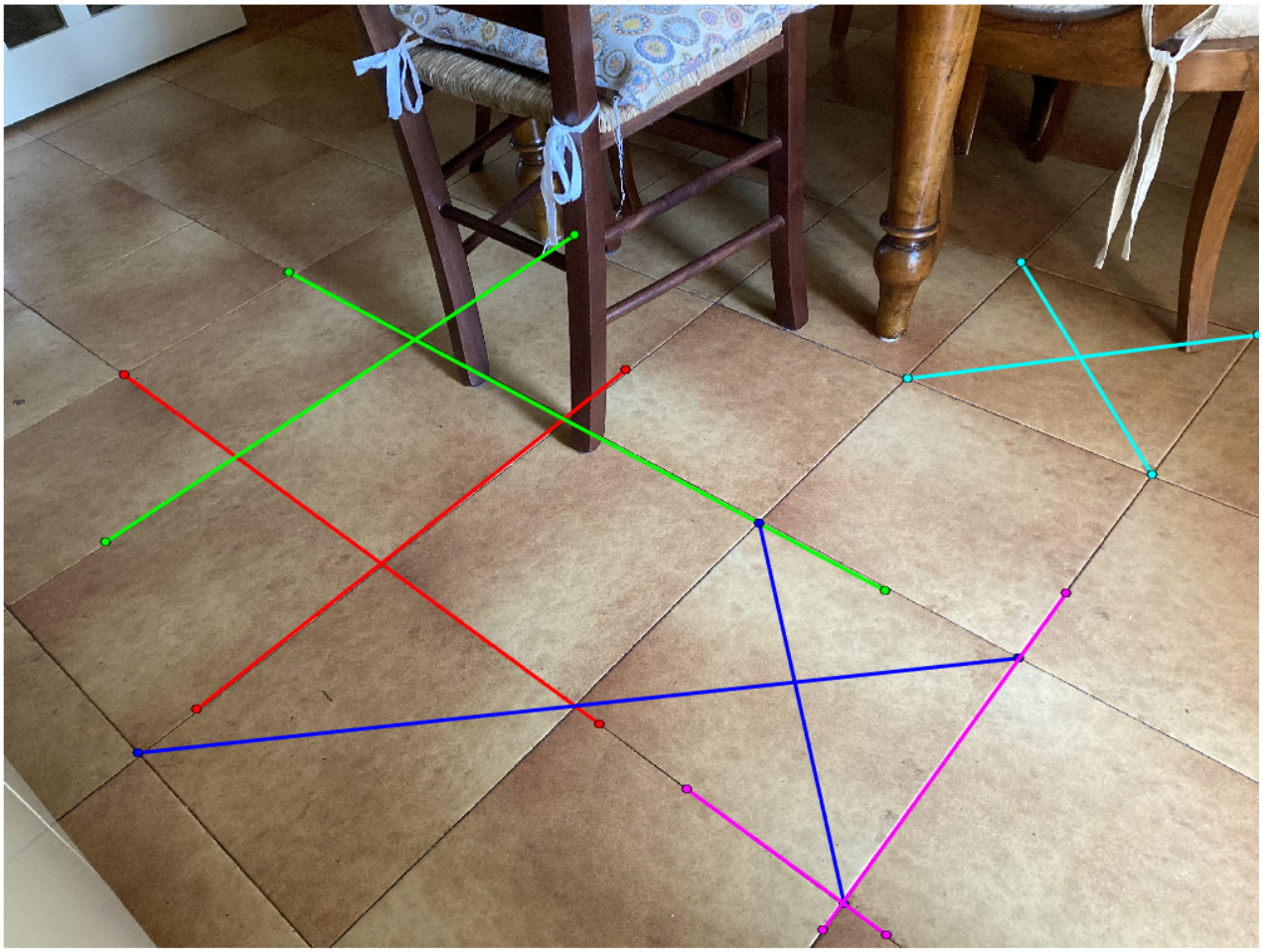
```
ans =

   3.2456e-08

ans =

   3.2456e-08
```

← correct mapping!

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

figure;
imshow(J);
```

see how image looks like

Draw 5 pairs of orthogonal segments: step 5



NOT so good
because of
numeric
instability

[redacted]

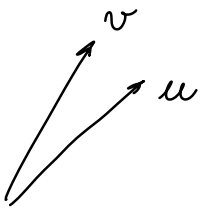
Published with MATLAB® R2021b

to compute the distance between the two matrix in projective space

C_∞^* and $C_\infty^{*'}$ up to scale factor

1) take forbenious norm $|C_\infty^{*'} - C_\infty^*|_{forbenious}$

BUT NOT enough... also due to orientation etc, in space



also $-v$ is close, you need to consider the sign, and chose the one which min the difference considering sign... NOT a unique projective distance definition

we know how to get AFFINE RECONSTRUCTION, we can do in steps, first affine rectification then go to similar (METRIC RECONSTR.) by STRATIFIED APPROACH

↑ another approach for RECTIFICATION

Contents

Stratified metric promotion after affine rectification

perform metric rectification of an image, after affine rectification. Given two pairs of image of orthogonal lines, the script finds the image of the dual conic to circular points and computes the homography that brings it back to its canonical form

you start from image partially affine reconstructed with C_∞'' in canonical position, then just two pairs of segments are enough!

Luca Magri
Politecnico di Milano
2020


```
close all;
clear all;
clc;
imgAffRect = imread('E4_data/affRect.JPG');


figure;
imshow(imgAffRect);
numConstraints = 10;
```

In this case $C_\infty^{*\prime}$ has particular form when line at infinity are canonical an image partially rectified...

the $C_\infty^{*\prime}$ has only 3 unknown up to a scale factor in this case

```
hold all;
fprintf('Draw pairs of orthogonal segments\n');
doAgain = 1;
count = 1;
A = zeros(numConstraints,3);
% select pairs of orthogonal segments
while (count <=numConstraints)
    figure(gcf);
    title('Select pairs of orthogonal segments')
    col = 'rgbcmykwrgbcmykw';
    segment1 = drawline('Color',col(count));
    segment2 = drawline('Color',col(count));

    l = segToLine(segment1.Position);
    m = segToLine(segment2.Position);

    % each pair of orthogonal lines gives rise to a constraint on s
    % [l(1)*m(1),l(1)*m(2)+l(2)*m(1), l(2)*m(2)]*s = 0
    % store the constraints in a matrix A
     A(count,:) = [l(1)*m(1),l(1)*m(2)+l(2)*m(1), l(2)*m(2)];

    count = count+1;
end
```

```
Draw pairs of orthogonal segments
```

$C_\infty^{*'}$ has simplified symm form! im only 2 unknowns (+1 up to scaling factor)

Contents

Stratified metric promotion after affine rectification

perform metric rectification of an image, after affine rectification. Given two pairs of image of orthogonal lines, the script finds the image of the dual conic to circular points and computes the homography that brings it back to its canonical form

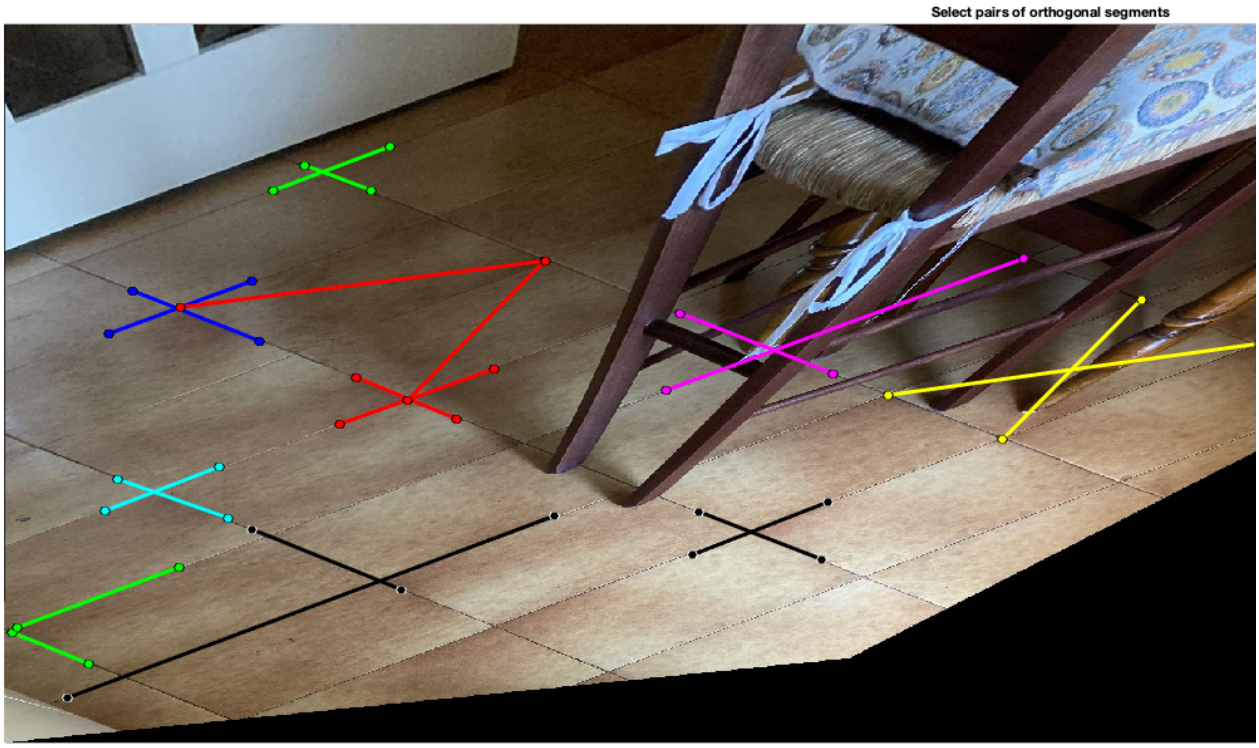
Luca Magri
Politecnico di Milano
2020


```
close all;
clear all;
clc;
imgAffRect = imread('E4_data/affRect.JPG');


figure;
imshow(imgAffRect);
numConstraints = 10;
```

select two constraints, just by 2 orthogonal segments

```
hold all;
fprintf('Draw pairs of orthogonal segments\n');
doAgain = 1;
count = 1;
A = zeros(numConstraints,3);
% select pairs of orthogonal segments
while (count <=numConstraints)
    figure(gcf);
    title('Select pairs of orthogonal segments')
    col = 'rgbcmykwrgbcmykw';
    segment1 = drawline('Color',col(count));
    segment2 = drawline('Color',col(count));

    l = segToLine(segment1.Position);
    m = segToLine(segment2.Position);

    % each pair of orthogonal lines gives rise to a constraint on s
    % [l(1)*m(1),l(1)*m(2)+l(2)*m(1), l(2)*m(2)]*s = 0
    % store the constraints in a matrix A
     A(count,:) = [l(1)*m(1),l(1)*m(2)+l(2)*m(1), l(2)*m(2)];

    count = count+1;
end
```

```
Draw pairs of orthogonal segments
```



2 steps

1) get e' and compute homography affine, then do it as METRIC...

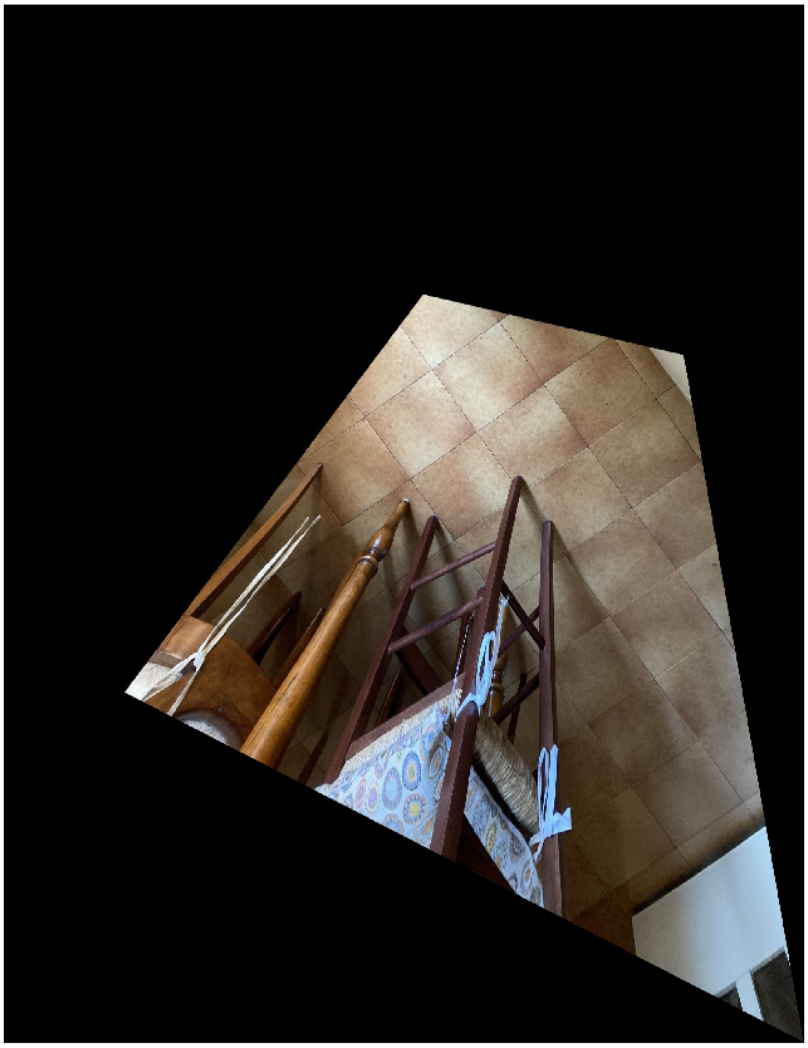
solve the system → JUST in three UNKN system in symm matrix

```
%S = [x(1) x(2); x(2) 1];
[~,~,v] = svd(A);
s = v(:,end); %[s11,s12,s22];
S = [s(1),s(2); s(2),s(3)];
```

compute the rectifying homography then build the 3×3 matrix $C_\infty^{*\prime}$

```
imDCCP = [S,zeros(2,1); zeros(1,3)]; % the image of the circular points
[U,D,V] = svd(S);
A = U*sqrt(D)*V';
H = eye(3);
H(1,1) = A(1,1);
H(1,2) = A(1,2);
H(2,1) = A(2,1);
H(2,2) = A(2,2);

Hrect = inv(H);
Cinfty = [eye(2),zeros(2,1);zeros(1,3)];

tform = projective2d(Hrect');
J = imwarp(imgAffRect,tform);

figure;
imshow(J);
```

and then do as before computing SVD etc...



stratified approach is more Robust than direct one

↓

easy to move horizon to standard position (less instability)

Published with MATLAB® R2021b

use different constraints is better ...

↓

try to promote better conditioning on the image

Metric rectification using the images of one cir

using my image of a circle I can do something BUT I get bad result...

perform a metric rectification from the image of a circle. Using the image dual to circular points leads to poor results, thus we follow a different apporach.p

better to use a different approach

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

Image Analysis and Computer Vision Politecnico di Milano

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

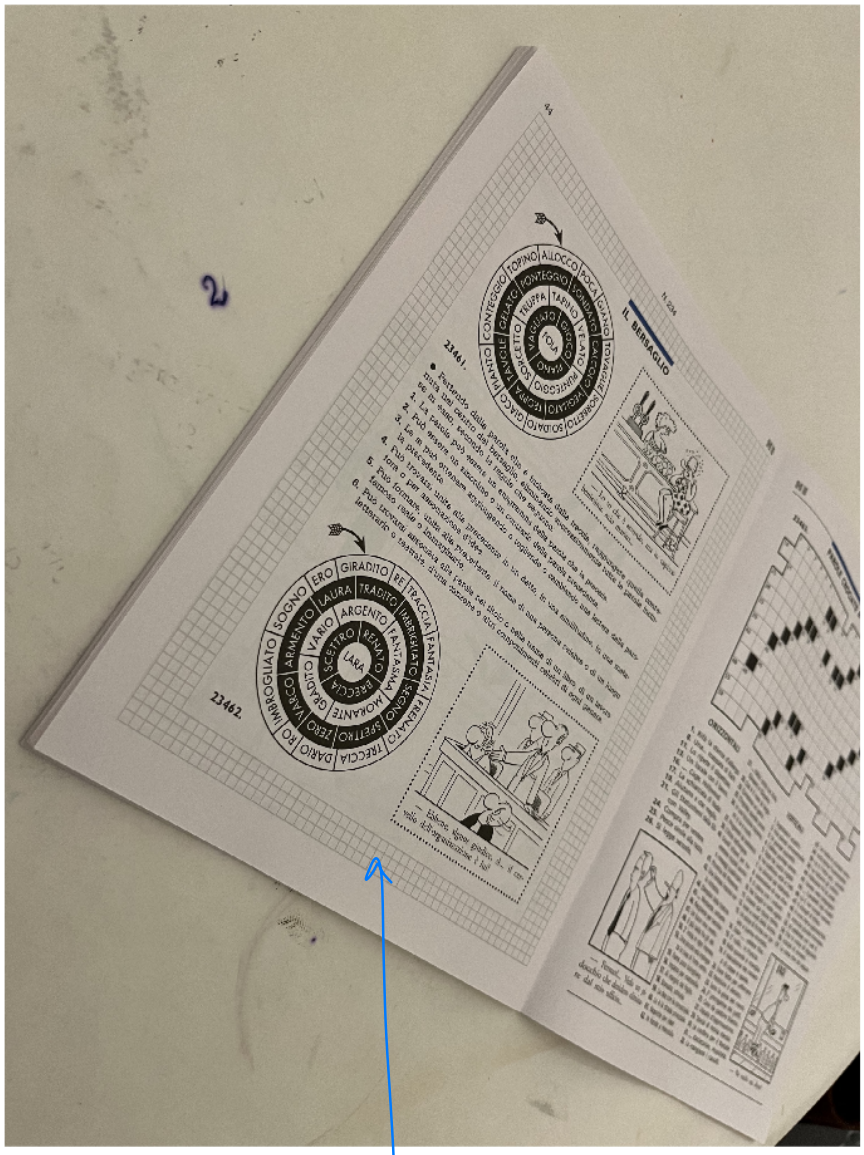
.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Contents

load the image

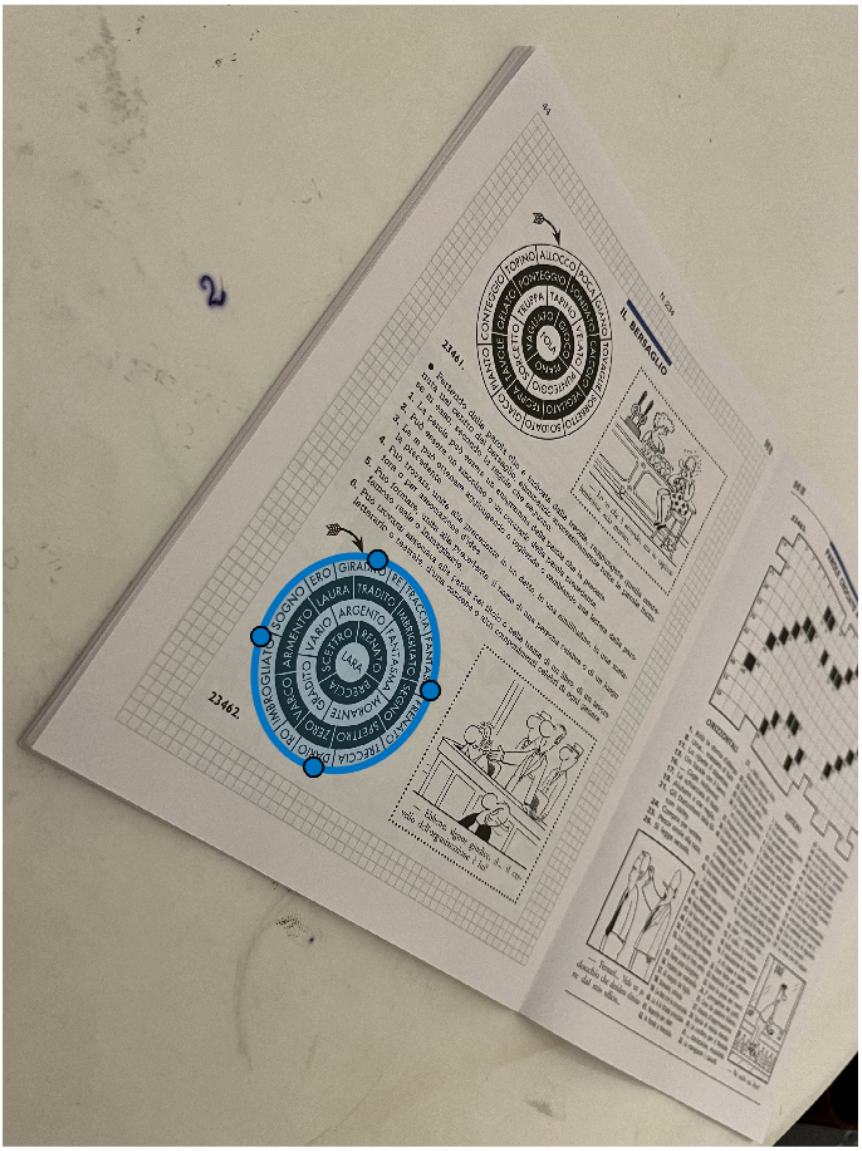
```
clear;
close all;
```

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



I use ellipse information KNOWN! image of circle used to get metric upgrade

```
do_load_annotations = 1;
if(do_load_annotations)
    data = load('bersaglio_annotations.mat');
    ellipse1 = data.e1;
    l1 = data.l1;
    l2 = data.l2;
    m1 = data.m1;
    m2 = data.m2;
end
% select one ellipse in the image
figure;
imshow(im);
hold all;
if(do_load_annotations)
    ellipse1 = drawellipse('Center',ellipse1.Center,'SemiAxes',ellipse1.SemiAxes,'RotationAngle',ellipse1.RotationAngle);
else
    ellipse1 = drawellipse();
end
if(1)
    save('bersaglio_ellipses.mat','ellipse1');
end
```



follow a stratified approach...

select pairs of parallel lines

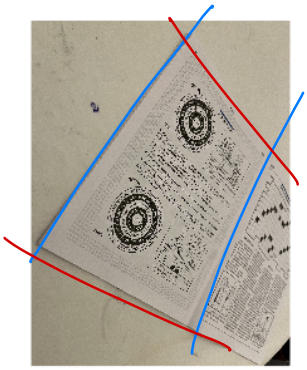
FIRST identify pairs of parallel lines to get vanishing lines

```
figure(gcf);
if(do_load_annotations==0)

    segment1 = drawline('Color','r');
    segment2 = drawline('Color','r');
    l1 = segToLine(segment1.Position);
    l2 = segToLine(segment2.Position);
    segment1 = drawline('Color','b');
    segment2 = drawline('Color','b');
    m1 = segToLine(segment1.Position);
    m2 = segToLine(segment2.Position);
end
```

compute the vanishing points

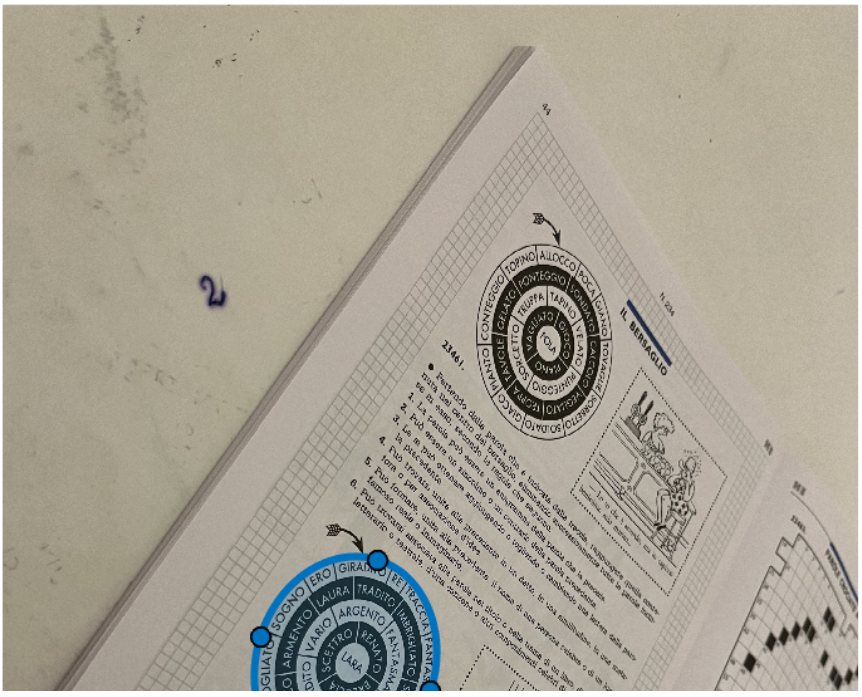
```
L = cross(l1,l2);
L = L./L(3);
M = cross(m1,m2);
M = M./M(3);

% compute the image of the line at infinity
imLinfty = cross(L,M);
imLinfty = imLinfty./(imLinfty(3));

% dispaly the selection
figure;
hold all;
% plot vanishing points
plot(L(1),L(2),'r.','MarkerSize',100);
plot(M(1),M(2),'b.','MarkerSize',100);
imshow(im);
```

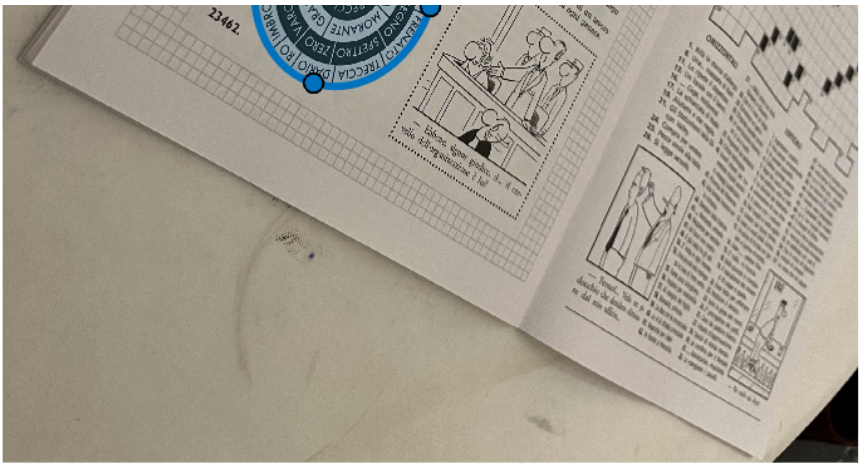
```
% plot vanishing line
line([L(1),M(1)],[L(2),M(2)],'Color','Green','Linewidth',3);
```



ℓ_∞

ℓ_∞ used for affine RECTIFICATION





Affine Rectification

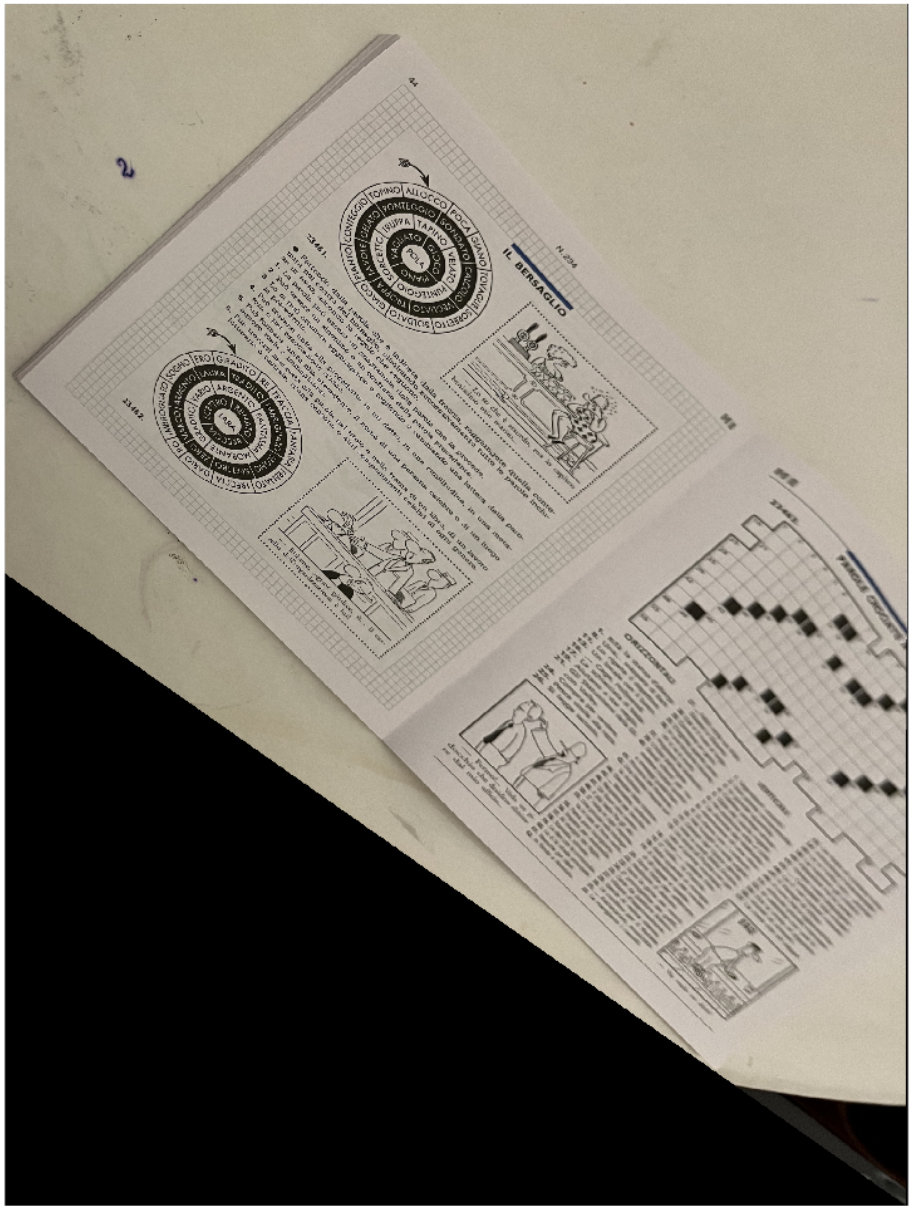
build the rectification matrix

```
H = [eye(2),zeros(2,1); imLinfty(:)'];
% we can check that H^-T* imLinfty is the line at infinity in its canonical
% form:

% compute the rectifying matrix
fprintf('The vanishing line is mapped to:\n');
disp(inv(H)'*imLinfty);
```

```
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);
imwrite(J,'E4_data/bersaglio_aff_rect.jpg');
```



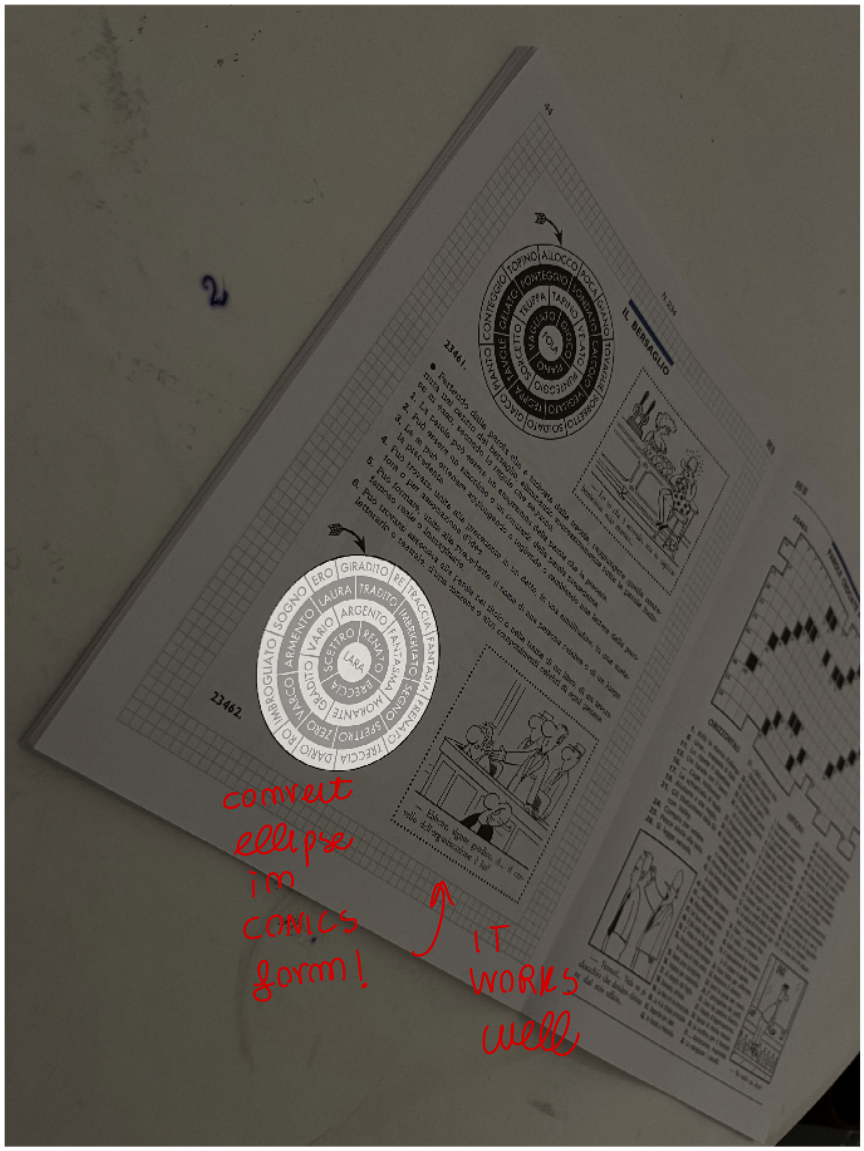
↓ for EUCLIDEAN promotion I identify, by CONVERT ellipses

convert ellipses into conic matrices to solve equation...

```
par_geo = [ellipse1.Center, ellipse1.SemiAxes,-ellipse1.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];
C1 = C1./C1(3,3);
% 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)));
```

having e_∞ and C' image of circle! ↓

it should be enough to get I', J'



I show original image + black image outside ellipse conic

convert ellipse in conics form!

IT WORKS well

intersect C' and e'∞ to get I',J'

↖ projectivity preserve intersection, C' and e'∞ intersect on I',J' image of it, then $C^{*\prime}_{\infty}$ and rectify

$$I'J'^{T} + J'I'^{T} = C^{*\prime}_{\infty}$$

comic dual to circular point

intersect ellipse with the vanishing line

see golub for altenative solutions

(NOT easy in MATLAB!)

↓ it is NUMERICAL

salve polynomical system:

```
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 = imLinfty(1)*x + imLinfty(2)*y+ imLinfty(3);
% solve a system with a line and the conic: this gives a 2th 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];
```

↑ 2nd deg equation solved...

let's check the solution using a direct substitution

express x = alpha y + beta

check correct solution...

```
alpha = -imLinfty(2)/imLinfty(1);
beta = -imLinfty(3)/imLinfty(1);
% substitute in eq2 to get a second degree eq in y
% a1*(alpha y + beta)^2 + b1*(alpha y + beta)*y + c1*y^2 + d1*(alpha y + beta) + e1*y + f1
poli = [a1*alpha*alpha + b1*alpha + c1,... % coefficients of y^2
    2*a1*alpha*beta + b1*beta + d1*alpha + e1,...
    a1*beta*beta + d1*beta + f1];% coefficients of y^2

sol = roots(poli);
y1 = sol(1);
y2 = sol(2);
x1 = alpha*y1 + beta;
```

```
x2 = alpha*y2 + beta;

q1 = [x1;y1;1];
q2 = [x2;y2;1];

disp([s1 s2])
disp([q1 q2])
```

```
   1.0e+03 *

   5.1707 - 0.0074i   5.1707 + 0.0074i
   1.5503 - 4.7445i   1.5503 + 4.7445i
   0.0010 + 0.0000i   0.0010 + 0.0000i

   1.0e+03 *

   5.1707 + 0.0074i   5.1707 - 0.0074i
   1.5503 + 4.7445i   1.5503 - 4.7445i
   0.0010 + 0.0000i   0.0010 + 0.0000i
```

image of the circular points

```
II = s1;
JJ = s2;
```

Conic dual to circular points

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

imDCCP = II*JJ' + JJ*II';
%imDCCP = imDCCP./norm(imDCCP);
```

← $C_\infty^{*\prime}$ computed

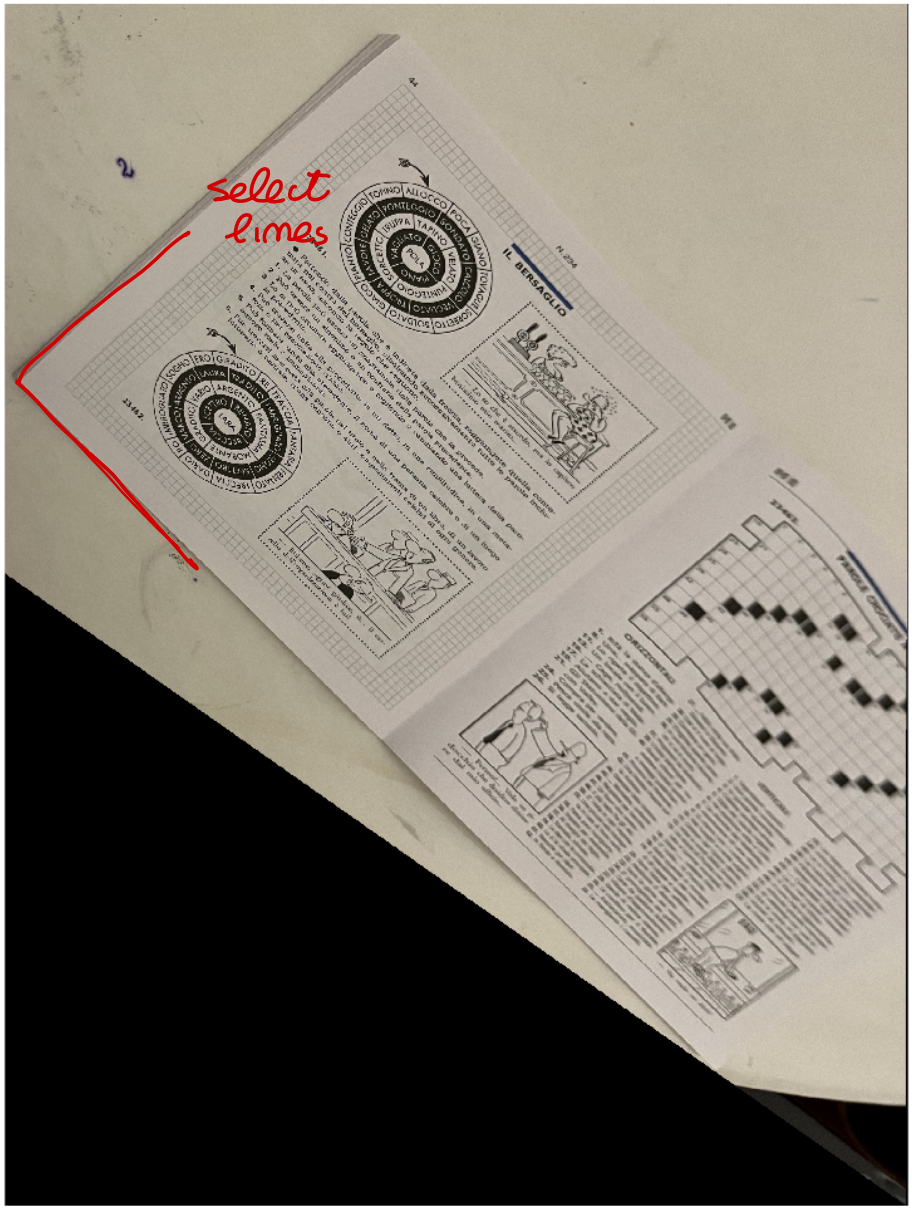
↓ verify Null($C_\infty^{*\prime}$) is ℓ_∞' to check if properly!

extract the line at infinity

```
l_inf = null(imDCCP);
% II and JJ passes through the line at infinity by construction
II'*l_inf
JJ'*l_inf

% they actually coincide with the solution retrieved before
l_inf - imLinfty


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

|| ← rectify image using that estimation of ℓ_∞ (as null space of img dual to circular points)

```
ans =

  -6.1800e-18 - 2.6455e-17i


ans =

  -6.1800e-18 + 2.6455e-17i


ans =

   1.0e-07 *

    0.0000
   -0.0000
   -0.1872
```



being $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ has eig = 1,1,0

↑

eig (conic) are Neg

NOT expected!

NUMERICAL INSTAB. spoil the Result!

IF I want to use $C_\infty^{*\prime}$ to compute θ angles:

compute angles using imDCCP

figure; imshow(im); segment1 = drawline('Color','r'); segment2 = drawline('Color','r'); l = segToLine(segment1.Position); m = segToLine(segment2.Position);

since the conic has one negative eigenvalue this doesn't work! cos_theta = (l'*imDCCP*m)/(sqrt(l'*imDCCP*l)*sqrt(m'*imDCCP*m));

compute the rectifying homography

```
%[U,D,V] = svd(imDCCP);
[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 cannot use it to extract the rectifying
% homography :(
```

force aspect ratio ↩ to be one using KNOWN information

```
V =

   -0.9865   -0.1635   -0.0002
   -0.1635    0.9865    0.0000
   -0.0002   -0.0000    1.0000


D =

   1.0e+07 *
```

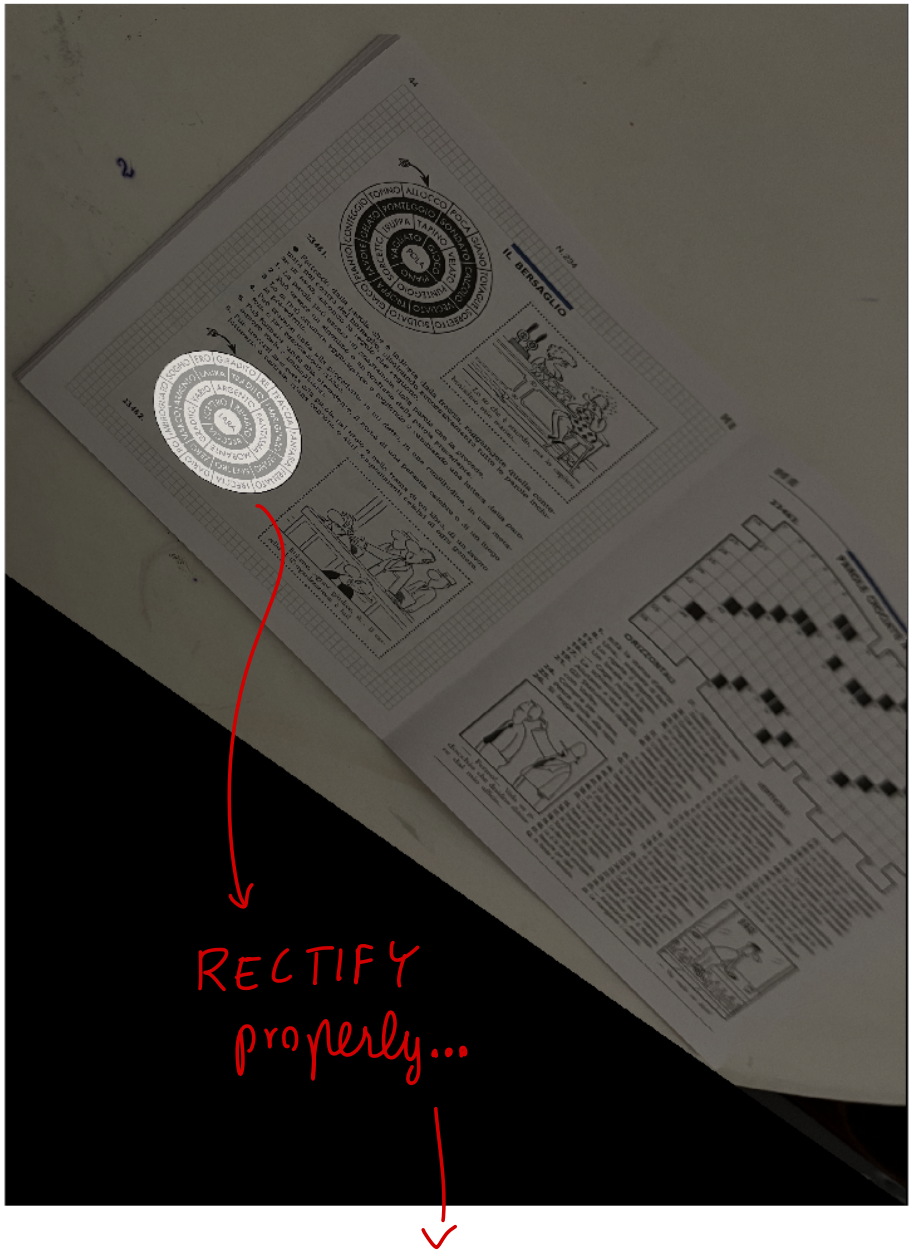
```
    5.6117         0         0
         0   -4.2858         0
         0         0   -0.0000


D =

   1.0e+07 *

    5.6117         0         0
         0   -4.2858         0
         0         0   -0.0000
```

However we have a circle

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_aff = J;
im_err = zeros(size(im_aff,1),size(im_aff,2));
for i = 1:size(im,1)
    for j = 1:size(im,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)));
```

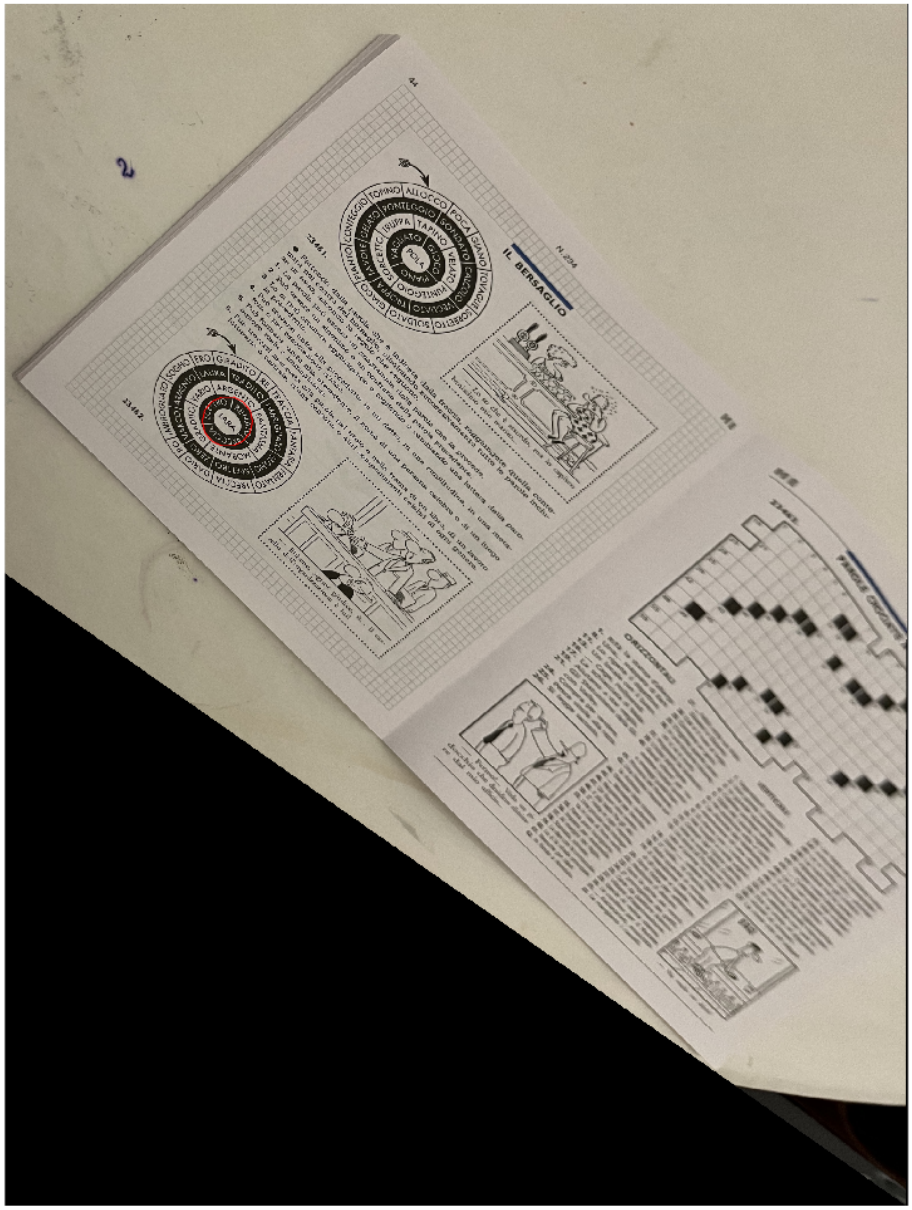
use information
about ellipse axis equal (CIRCLE)



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.

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

Affinity to transform ellipse to a circle... then apply all to 1st image

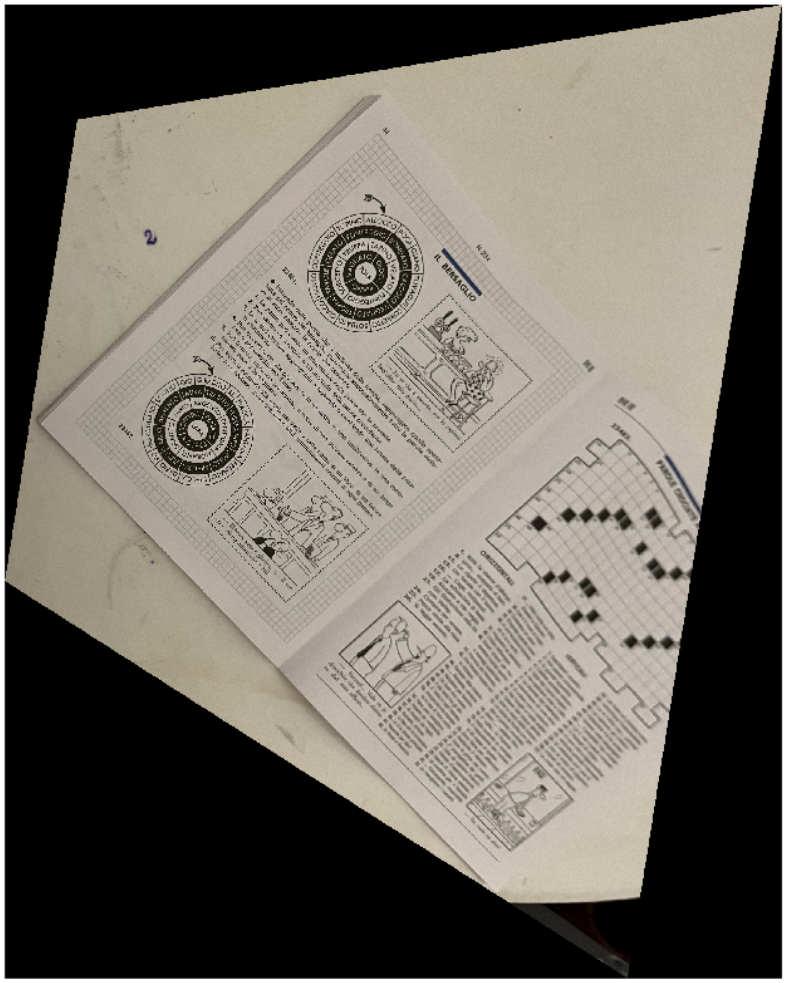
```
alpha = angle;
a = axes(1);
b = axes(2);
% rotation
U = [cos(alpha), -sin(alpha); sin(alpha), cos(alpha)];
% rescaling the axis to be unitary
S = diag([1, a/b]);
K = U*S*U';
A = [K zeros(2,1); zeros(1,2),1];
T = A*H; % the final transformation is the composition between the homography that maps the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

combine homography transformation

apply the transformation

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

Metric rectification.



better result than before! because geometry is more reliable, bypass numerical instability by looking @ geometric information you have on your image

from knowing I have ellipse that is supposed to be CIRCLE, rescale axis of ellipse to be correct

---

Published with MATLAB® R2021b