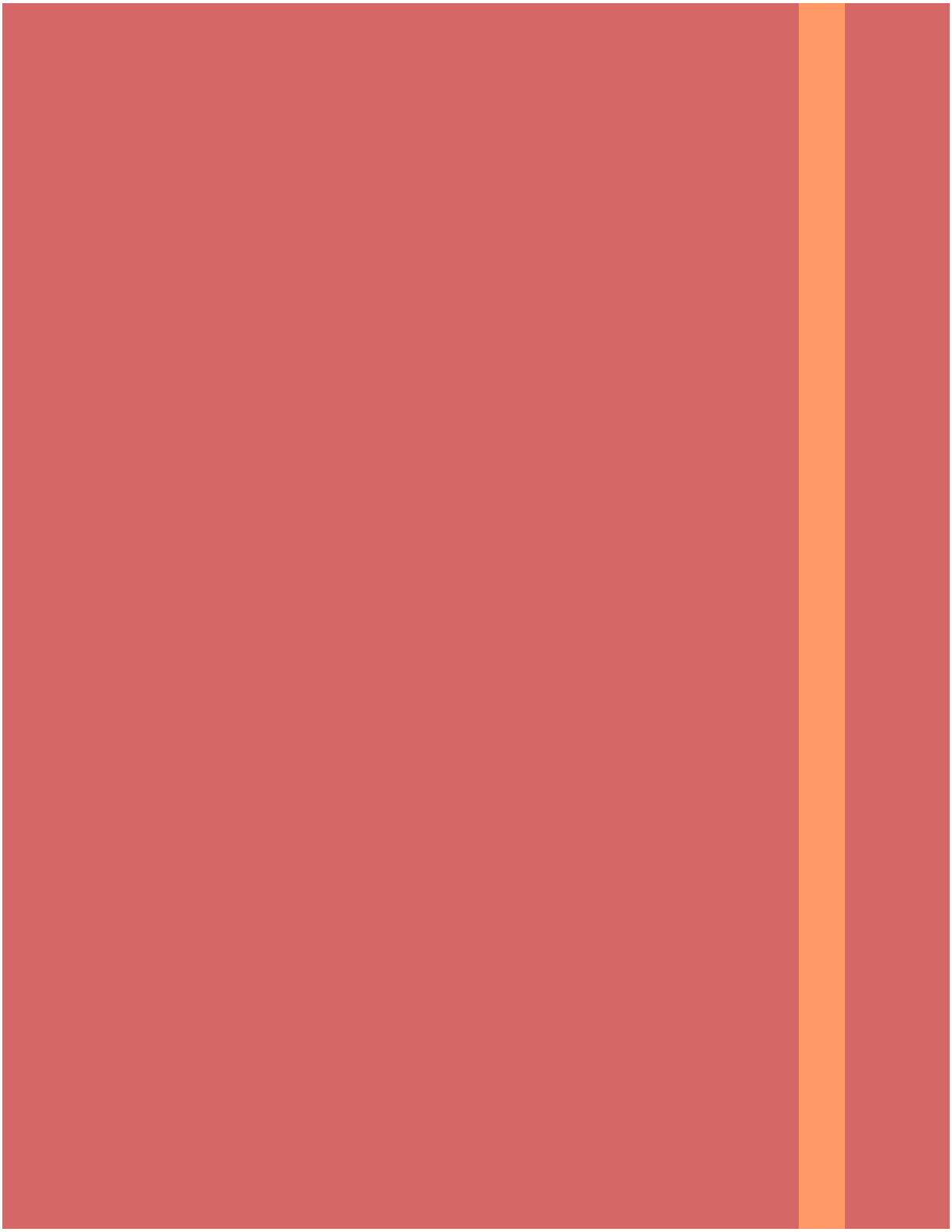


Applications of Basic **DIRECT LINEAR TRANSFORM (DLT)**

↓

from Geometric entities in the PROJECTIVE PLANE, you want to ESTIMATE that

defined up to a scale factor..
to take into account and solve a linear system

how SVD can be used to estimate geometric entities... → a bunch of points in the plane

to estimate a CONIC, you need some POINTS as the unknown of the $x^T C x = 0$

$x :=$ homogeneous coord $\mathbb{R}^3$

C 3×3 → but $C = C^T$ symm

in principle C has 9 DOF BUT being quadratic eq. $C = C^T$ → so $9 - 3 = 6$ DOF and since comic up to scaling factor → 5 DOF

this arguments holds ∀ geometric entity to reconstruct

we need 5 points...

(x_i, y_i, w_i) $i = 1, .., 5$ to define C of comic

↓

as JUST equation of general comics

$ax^2 + bxy + cx + dy + ey^2 = 0$ CARTESIAN

in homogeneous also w in the equation

knowing 5 points $\in C$ and so, the incidence relation! → $\forall x_i$ you write $x_i^T C x_i = 0$

⇒ linear system in which we reconstruct C

$$\begin{cases} \begin{bmatrix} x & y & w \end{bmatrix} \begin{bmatrix} \cdots \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0 \\ \vdots \\ \cdot \\ \cdot \end{cases}$$

Solve the system of equations as homogeneous system

E2C_conic.m

```
%% Playing with conics in homogeneous coordinates
% Image Analysis and Computer Vision
% Politecnico di Milano
%
% Credits: Prof. Giacomo Boracchi
% Edits: Luca Magri.


%% Preparation
% First, we create an empty image and display it (will be our "canvas")
figure(1)
im=zeros(500,500);
imshow(im, []);
hold on;

%%
% Data input: enter five points (then enter)
disp('click 5 points, then enter');
[x, y]=getpts;
scatter(x,y,100,'filled');

%%
% x and y are now column vectors.
x
y
%% Estimation of conic parameters
% using the technique shown in "Multiple View Geometry", page 9.
% this is equivalent to requiring that the conic passes trhough all the six
% points
A = [x.^2 x.*y y.^2 x y ones(size(x))]
% A is 5x6: we find the parameter vector (containing [a b c d e f]) as the
% right null space of A.
% This returns a vector N such that A*N=0.  Note that this expresses the
% fact that the conic passes through all the points we inserted.
N = null(A);
% alternatively you can use the svd
[~,~,V] = svd(A);
N = V(:,end);
%%
% Let's assign the values to variables and build the 3x3 conic matrix
% (which is symmetrical)
cc = N(:, 1);
% change the name of variables
[a b c d e f] = deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
% here is the matrix of the conic: off-diagonal elements must be divided
% by two
C=[a b/2 d/2; b/2 c e/2; d/2 e/2 f]

% Remark: since
% when the poir
% the system ac
% however, thes
```

to click 5 times on the image, 5 points selected

as we check x,y are vectors of point

$x_1\ x_2 .. x_5$
$y_1\ y_2 .. y_5$

to FIT the points, we solve

$AX = 0$ homogeneous eq $X = (a, b, c, d, e, f)$ unknowns!

system of equations with final columns of 1 (because as EUCLIDEAN)

MATRIX build by stacking the various points coefficients (x, y)

6 unknowns, 5 eq. I want Null(A)

fit the conics to this point

$ax^2 + bxy + cy^2 + dx + ey + f \cdot 1$

columns of 1 in homogeneous coordinates

// factorization SVD of A

last columns V is the smallest singular value column, which is a basis of Null space

off diagonal elements divided by 2

rearrange

Condividi Sposta Elimina Dettagli Altro

E2C_conic.m

```
% Remark: since the right null space has dimension one,
% when the points are taken in general position
% the system admits an infinite number of solutions
% however, these can be expressed as lambda * n, where n
% lambda \in R and n = null(A).
% thus, they all corresponds to the same conic.

%% Draw the conic
% We pick every pixel, and compute the incidence relation.
im=zeros(500,500);
for i=1:500
    for j=1:500
        im(i,j)=[j i 1]*C*[j i 1]'; % this is an algebraic error
    end
end
figure; imagesc(im); colormap("spring"); title('Algebraic error');
% keep in mind that for geometric problem the best is to deal with "geometric error"
%%
% Let's draw black for pixels less than zero, white for pixels greater than
% 0.
figure(1)
imshow(im > 0);
scatter(x,y, 100, 'filled');

%% What happen when A is not full rank?
% Instead of taking 5 points take just 4
% N contains in each column a generator of the null space
N = null(A(1 : 4, :));

% then, any vector living in the null space give rise to a conic
% passing throguh those 4 points. Since the nullspace has dimension two
% we can find two conic passing through the four points

%%
% Let's assign the values to variables and build the 3x3 conic matrix
% (which is symmetrical)
% Let's start with a first solution, the first vector of the nullspace
cc = N(:, 1);
% cc = N(:, 1) + N(:, 2); % another example, any linear combination is fine
[a, b, c, d, e, f]=deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
% here is the matrix of the conic
C1_four =[a b/2 d/2; b/2 c e/2; d/2 e/2 f]
% Let's consider a second solution, the second  vector of the nullspace
cc = N(:, 2);
% cc = N(:, 1) + 2*N(:, 2); % another example, any linear combination is fine
[a, b, c, d, e, f]=deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
% here is the matrix of the conic
C2_four =[a b/2 d/2; b/2 c e/2; d/2 e/2 f]

%% Draw the cor
% We pick every
im_four=zeros(5
for i=1:500
```

CONIC is the set of POINT s.t $x^T C x = 0$

take all image pixel and I compute

where if it ~0 error we are in the BORDER

set black pixels with error < 0

taking (x_i, y_i) $i = 1..4$ MULTIPLE SOLUTION
5 unknowns but 4 equations → the dimension of the kernel increase by one dimension!

When we take 4 POINTS, we get multiple solutions!

now I'll take last 2 columns of V in the svd, getting vector space of dimension 2 (2 solutions) and any linear combination of this two is a vector solution of that one cause I'll conic passing through that PENCIL OF CONICS)

① ② I get 2 conics

E2C_conic.m

```
100
101 %% Draw the conic
102 % We pick every pixel, and compute the incidence relation.
103 im_four=zeros(500,500);
104 for i=1:500
105     for j=1:500
106         im_four1(i,j)=[j i 1]*C1_four*[j i 1]'; //first
107         im_four2(i,j)=[j i 1]*C2_four*[j i 1]'; //second
108     end
109 end
110
111 figure(2)
112 subplot(1,2,1)
113 imshow(im_four1>0);
114 hold on;
115 scatter(x(1 : 4), y(1 : 4), 100, 'filled','cyan');
116 scatter(x(5), y(5), 100, 'filled','red');
117 title('1st nullspace col ')
118 subplot(1,2,2)
119 imshow(im_four2>0);
120 hold on
121 scatter(x(1 : 4), y(1 : 4), 100, 'filled','cyan');
122 scatter(x(5), y(5), 100, 'filled','red');
123 title('2n nullspace col ')
124 hold off
125
126
127 %% Any linear combination of N1 and N2 is a solution
128 % we can draw a pencil of conics
129 for lambda = [10, 200, 3000, 100000]
130
131     cc = 1*N(:,1)+ lambda*N(:,2);
132     [a, b, c, d, e, f]=deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
133     C3_four =[a b/2 d/2; b/2 c e/2; d/2 e/2 f];
134     % We pick every pixel, and compute the incidence relation.
135     im_four=zeros(500,500);
136     for i=1:500
137         for j=1:500
138             im_four3(i,j)=[j i 1]*C3_four*[j i 1]';
139         end
140     end
141     figure(3)
142     imshow(im_four3>0);
143     hold on;
144     scatter(x(1 : 4), y(1 : 4), 100, 'filled','cyan');
145     scatter(x(5), y(5), 100, 'filled','red');
146     title('linear combination of nullspace cols ')
147     pause;
148 end
149
150 %% Tangent line
151 % Let's draw th
152 % coordinates i
153 % the line tangent to C at the line x is defined as t = C*x
```

hyperbola ↑ nullspace

ellipses

we take linear combination of the two $C1 + \lambda C2 = C'$

drawing each conics, only if them pass through the 4 points!

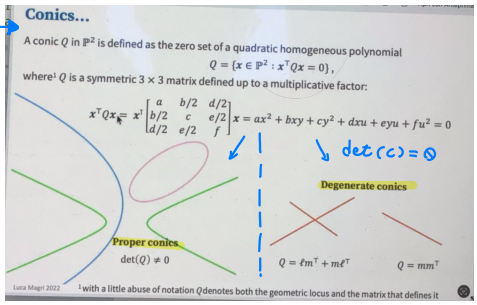
Condividi Sposta Elimina Dettagli Altro

09:54 Dom 1 dic 93%

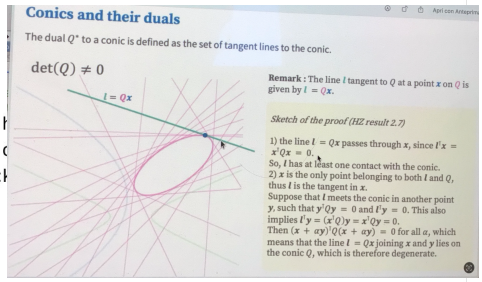
E2C_conic.m

```
125
126
127 %% Any linear combination of N1 and N2 is a solution
128 % we can draw a pencil of conics
129 for lambda = [10, 200, 3000, 100000]
130
131     cc = 1*N(:,1)+ lambda*N(:,2);
132     [a, b, c, d, e, f]=deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
133     C3_four =[a b/2 d/2; b/2 c e/2; d/2 e/2 f];
134     % We pick every pixel, and compute the incidence relation.
135     im_four=zeros(500,500);
136     for i=1:500
137         for j=1:500
138             im_four3(i,j)=[j i 1]*C3_four*[j i 1]';
139         end
140     end
141     figure(3)
142     imshow(im_four3>0);
143     hold on;
144     scatter(x(1 : 4), y(1 : 4), 100, 'filled','cyan');
145     scatter(x(5), y(5), 100, 'filled','red');
146     title('linear combination of nullspace cols ')
147     pause;
148 end
149
150 %% Tangent lines
151 % Let's draw the tangents to our points: the tangent line in homogeneous
152 % coordinates is l
153 % the line tangent to C at the line x is defined as l = C*x
154 % this is the line tangent to the first point
155 l1 = C * [x(1); y(1); 1];
156 % here I compute all the tangent lines simultaneously
157 % stacking all the vectors in the columns of the matrix l
158 l=C*[x y ones(size(x))]'
159 %%
160 % in order to draw it we find its angle,
161 % then plot a line of length 200 around it.
162 % In order to write less lines of code, I'm doing a bit of h
163 % matrices and vectors here: theta will be a column vector c
164 % with a single plot command I'll plot all the lines.  Check
165 % documentation of plot to understand what's happening.
166 theta=atan2(-l(1,:),l(2,:));
167 theta = theta';
168 figure(1), hold on
169 plot([x-cos(theta)*100 x+cos(theta)*100]',...
170     [y-sin(theta)*100 y+sin(theta)*100]','b-','LineWidth',5);
171
172 %%
173 % Let's compute the dual conic and check that all the tangent lines we
174 % computed prev
175 % precision iss
176 Cstar=inv(C);
177 l(:,1)'*Cstar*l
178 l(:,2)'*Cstar*l(:,2)
```

(*) tangent lines.
point ↔ line in dual plane, which satisfy the dual conic equation (QUADRATIC relation) when NON DEGENERATE
so given C, x is easy to find the tangent line



When conic C is NON singular, the tangent line is easy to compute by C x



easy to prove by simple linear algebra

Condividi Sposta Elimina Dettagli Altro

E2C_conic.m

```
125
126
127   %% Any linear combination of N1 and N2 is a solution
128   % we can draw a pencil of conics
129   for lambda = [10, 200, 3000, 100000]
130
131       cc = 1*N(:,1)+ lambda*N(:,2);
132       [a, b, c, d, e, f]=deal(cc(1),cc(2),cc(3),cc(4),cc(5),cc(6));
133       C3_four =[a b/2 d/2; b/2 c e/2; d/2 e/2 f];
134       % We pick every pixel, and compute the incidence relation.
135       im_four=zeros(500,500);
136       for i=1:500
137           for j=1:500
138               im_four3(i,j)=[j i 1]*C3_four*[j i 1]';
139           end
140       end
141       figure(3)
142       imshow(im_four3>0);
143       hold on;
144       scatter(x(1 : 4), y(1 : 4), 100, 'filled','cyan');
145       scatter(x(5), y(5), 100, 'filled','red');
146       title('linear combination of nullspace cols ')
147       pause;
148   end
149
150   %% Tangent lines
151   % Let's draw the tangents to our points: the tangent line in homogeneous
152   % coordinates is l
153   % the line tangent to C at the line x is defined as l = C*x
154
155   % this is the line tangent to the first point
156   l1 = C * [x(1); y(1); 1];
157   % here I compute all the tangent lines simultaneously
158   % stacking all the vectors in the columns of the matrix l
159   l=C*[x y ones(size(x))]'
160   %%
161   % in order to draw it we find its angle,
162   % then plot a line of length 200 around it.
163   % In order to write less lines of code, I'm doing a bit of hacks with
164   % matrices and vectors here: theta will be a column vector of angles, and
165   % with a single plot command I'll plot all the lines.  Check out the
166   % documentation of plot to understand what's happening.
167   theta=atan2(-l(1,:),l(2,:));
168   theta = theta';
169   figure(1), hold on
170   plot([x-cos(theta)*100 x+cos(theta)*100]',...
171       [y-sin(theta)*100 y+sin(theta)*100]','b-','LineWidth',5);
172
173   %% → dual conic and check that all lines pass through it C*
174   % Let's compute the dual conic and check that all the tangent lines we
175   % computed prev
176   % precision iss
177   Cstar=inv(C);
178   l(:,1)'*Cstar*l
```

homogeneous equation, hom coord with w=1, euclidean coordinate

NOT always $= C^{-1}$

$C^* = C^{-1}$ simply (being NON SINGULAR)

than to check if lines satisfy dual conics MUST be $e^T C^* e \cong 0 \ \forall e$ computed before!

Condividi Sposta Elimina Dettagli Altro

← so we just use SVD, solve linear system as our problem...

as homogeneus problem

⇒ A kind of TRANSFORMATION

between images we know is PROJECTIVITY
(HOMOGRAPHY)

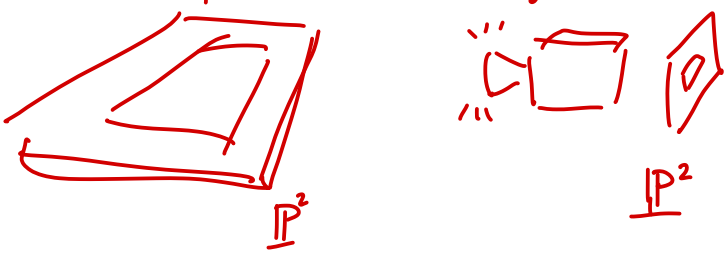
≃ which is as the points in the scene 3D, than I take a picture and a PROJECTION MAPPING

projectivity is defined in a projective space with same dimension

PLANAR PROJECTIVE := 3×3 H invertible mapping planar point nts other

IN SPACE PROJECTIVITY instead we map $\mathbb{P}^d \to \mathbb{P}^d$ by d×d MATRIX s.t $\det(H_d) \neq 0$

- If points lay on same plane



MAPPING
co-planar points

PROJECTION
Restricted to
the plane
↓
PROJECTIVITY

← that has algebraic interpretation...

from camera MATRIX

$$P = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix}$$

↑
3×4 camera MATRIX

↓

mapping $[X\ Y\ Z\ ]$ ⟶ to image

than supposing to take a point on the scene (consider say on same plane)

the PROJECTION MATRIX...

assuming the POINTS are CO-PLANAR, than belongs to $Z=0$ plane! so you get

$X \to x$ mapping to image plane

$$[X\ Y\ 0\ 1] \Rightarrow \underbrace{[P_1\ P_2\ P_4]}_{3\times 3}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$Z=0$

3×3 transformation

↓

projection mapping restricted to plane becomes 3×3 MATRIX (invertible)

→ then PROJECTION Restriction is HOMOGRAPHY maintain 3×3 matrix

than solve the system using SVD to estimate coefficients ↳

Let's estimate HOMOGRAPHY as

- PROJECTIVE entity and the DOF (8) of it!

NOT 9 because of scale factor

each point provide 2 constraint

AT least 4 points to define H ⇒ I can define H by constraints definition

estimate homography coefficients by SVD

Homography estimation via DLT

(Direct Linear Transform)

Estimate an homography via DLT References: Hartley Zisserman, Multiview Geometry. Chapter 4

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

Image Analysis and Computer Vision Politecnico di Milano

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

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

Contents

Let's take two picture of a planar scene

```
im1 = imread('E3_data/ella1.jpg');
im2 = imread('E3_data/ella2.jpg');
im1 = imresize(im1,0.6);
im2 = imresize(im2,0.6);
figure;
subplot(1,2,1);
imshow(im1);
subplot(1,2,2);
imshow(im2);
```

from an almost planar scene → I want to identify corresponding points

Homography transformation

the image coordinates that lie on the same plane of these two images are related by an **homography**: Without loss of generality, we can assume that the point on the plane have coordinates M = [x,y, 0,1], and are mapped via P1 and P2 to m1 = P1*M and m2 =P2*M, where Pi = Ki[Ri,ti] and Ri = [Ri1,Ri2,Ri3]. Having z=0 implies that the coordinates mi = Ki[Ri1,Ri2,ti] [x,y,1] are determined by a 3x3 non singular transformation Hi. The realtion between {m1<->m2} is hence given by H = H1^-1*H2. Our aim is to determine such H.

Identify four pairs of corresponding points

```
m1 = nan(3,4);
m2 = nan(3,4);

figure;
imshow(im1);
hold on;
[x, y] = getpts();
m1(:,1) = [x(1); y(1); 1];
m1(:,2) = [x(2); y(2); 1];
m1(:,3) = [x(3); y(3); 1];
m1(:,4) = [x(4); y(4); 1];

FNT_SZ = 20;
plot(m1(1,1), m1(2,1),'o', 'Color','r');
plot(m1(1,2), m1(2,2),'o','Color', 'r');
plot(m1(1,3), m1(2,3),'o','Color', 'r');
plot(m1(1,4), m1(2,4),'o','Color', 'r');
text(m1(1,1), m1(2,1), 'a', 'FontSize', FNT_SZ, 'Color', 'r');
```

select points on the two images

```
text(m1(1,2), m1(2,2), 'b', 'FontSize', FNT_SZ, 'Color', 'r');
text(m1(1,3), m1(2,3), 'c', 'FontSize', FNT_SZ, 'Color', 'r');
text(m1(1,4), m1(2,4), 'd', 'FontSize', FNT_SZ, 'Color', 'r');
```


```
figure;
imshow(im2);
hold on;
[x, y] = getpts();
m2(:,1) = [x(1); y(1); 1];
m2(:,2) = [x(2); y(2); 1];
m2(:,3) = [x(3); y(3); 1];
m2(:,4) = [x(4); y(4); 1];

FNT_SZ = 20;
plot(m2(1,1), m2(2,1),'o', 'Color','r');
plot(m2(1,2), m2(2,2),'o','Color', 'r');
plot(m2(1,3), m2(2,3),'o','Color', 'r');
plot(m2(1,4), m2(2,4),'o','Color', 'r');
text(m2(1,1), m2(2,1), 'a''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,2), m2(2,2), 'b''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,3), m2(2,3), 'c''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,4), m2(2,4), 'd''', 'FontSize', FNT_SZ, 'Color', 'r');
```

select SAME points on the second image as imgs taken of same planar scene

We get PAIRS of corresponding points in m1 and m2 (handly selected, they can be automatically chosen)


ELLA
AT THE
HOLLYWOOD BOWL
THE IRVING
BERLIN SONGBOOK

to build the equation for homography

Direct linear transform

Hartley & Zisserman Alg 4.2 page 109 in 2nd edition (idea behind)

We want to estimate the homography H = [h1^t; h2^t;h3^t] that maps each mi=[xi;yi;wi] in mi'=[xi';yi';wi']. Since mi and mi' are in homogenous coordinates, mi and mi' represents the same points if exists a lambda different from zero such taht Hmi = lambda*mi'. This condition can be expressed using the cross product as mi'xHmi = 0. This equation be written as yi'*h3'+xi - wi* h2'*xi =0 wi'*h1'*xi - xi'*h3'*xi =0 xi'*h2'*xi - yi'*h1'*xi =0 So for each pairs of corresponding points we have three equations. But only two of the trhee are linear independent. So each correspondence provide 2 constraints. Four points are enough to constraints the 8 d.o.f of H (= 3x3 - 1 to account for the scale).

build system coefficent matrix to solve the projective

```
% Build the  matrix of the linear homogeneous system
% A vec(H) = 0
num_points = 4;
A = zeros(2*num_points,9);
% the rows are the number of independent constraints
% the columns are the number of unknown (9 for a 3x3 matrix)
for i =1:num_points
    p1= m1(:,i);
    p2 = m2(:,i);
    A(2*i-1,:) = [zeros(1,3), p1'*p2(3), - p1'*p2(2)];
    A(2*i,:) = [p2(3)*p1', zeros(1,3), -p1'*p2(1)];
end

% now we have to solve an homogeneous system: svd!
[u,s,v] = svd(A);
% let's have a look at how well the system is conditioned
s = diag(s);
% rough estimate of the nullspace dimension
nullspace_dimension = sum(s < eps * s(1) * 1e3);
if (nullspace_dimension > 1)
  fprintf('Nullspace is too large to accomodate a single solution...\n');
end
h = v(:,end);
H = reshape(h,3,3)';
H = H./norm(H);
% let's factorize this code into a function dlt_homography

%save('ella_homo.mat','H','m1','m2');
```

build coefficient matrix

solve system taking vector of last singular value

I get estimate of the homography

Showing the mapping on the points

```
Hm1 = H*m1;
% dehomogenize points
Hm1 = Hm1./repmat(Hm1(3,:),3,1);
figure;
imshow(im2);
hold all;
MRK_SZ = 300;
scatter(m2(1,:),m2(2,:),MRK_SZ,'ro','filled');
scatter(Hm1(1,:),Hm1(2,:),MRK_SZ,'y+','LineWidth',4);
```

(divide by last coord to dehomogenize)

to check if works well we can try to use it...

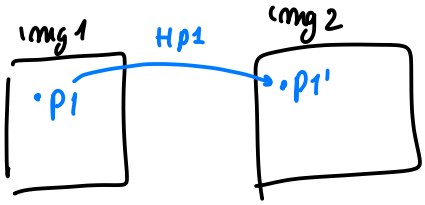
I plot in CARTESIAN

they are mapped in a good way!

↑ Homography through 4 points is easy to estimate!

to build the homography

p1 → p1'



$Hx \approx x'$

almost true... equality holds in PROJECTIVE plane

→ then, $\exists \lambda$: $\lambda x' = Hx$ ← the equality Holds in PROJ space

we says x' and Hx: are PARALLEL to eachother in space

meaning

$x' \times Hx$ { 0 if parallel; max if perpendicular

↑ cross product represent the area of the parallelogram by two vector

we need to use equations to provide Algebraic constraints in H

↳ doing math you find the equations...

$$\begin{cases} y_i^T H_3^T x_i - w_i H_2^T x_i = 0 \\ w_i^T H1^T x_i - x_i^T H_3^T x_i = 0 \\ x_i^T H_2^T x_i - y_i^T H_1^T x_i = 0 \end{cases}$$

than only 2 of this are imporeant, last one is JUST linear combination

Noise by design!

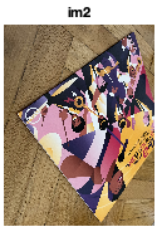
what if I map all the pixels?

Rendering the results

```
t = maketform( 'projective', H');
J = imtransform(im1,t);


figure;
subplot(1,3,1);
imshow(im1);
title('im1')
subplot(1,3,2);
imshow(J);
title('transformed im1')
subplot(1,3,3);
imshow(im2);
title('im2')
```

apply homography to all pixels of first image

reconstruct same view point!



Degenerate configurations - collinearity

what if we take COLINEAR points?

now let's repeat the same procedure but this time we select 4 collinears point

when the 4 points are in the same line, degenerate configuration NOT in cell plane! we don't reconstruct all the plane → Ncell-space too big!

↓

$\tilde{H}$ behaves bad on remain pixels

```
m1 = nan(3,4);
m2 = nan(3,4);

% select points on the first image
figure;
imshow(im1);
hold on;
[x, y] = getpts();
m1(:,1) = [x(1); y(1); 1];
m1(:,2) = [x(2); y(2); 1];
m1(:,3) = [x(3); y(3); 1];
m1(:,4) = [x(4); y(4); 1];

FNT_SZ = 20;
plot(m1(1,1), m1(2,1),'o', 'Color','r');
plot(m1(1,2), m1(2,2),'o','Color', 'r');
plot(m1(1,3), m1(2,3),'o','Color', 'r');
plot(m1(1,4), m1(2,4),'o','Color', 'r');
text(m1(1,1), m1(2,1), 'a', 'FontSize', FNT_SZ, 'Color', 'r');
text(m1(1,2), m1(2,2), 'b', 'FontSize', FNT_SZ, 'Color', 'r');
text(m1(1,3), m1(2,3), 'c', 'FontSize', FNT_SZ, 'Color', 'r');
text(m1(1,4), m1(2,4), 'd', 'FontSize', FNT_SZ, 'Color', 'r');
```

select points on the second image

```
figure;
imshow(im2);
hold on;
[x, y] = getpts();
m2(:,1) = [x(1); y(1); 1];
m2(:,2) = [x(2); y(2); 1];
m2(:,3) = [x(3); y(3); 1];
m2(:,4) = [x(4); y(4); 1];

FNT_SZ = 20;
plot(m2(1,1), m2(2,1),'o', 'Color','r');
plot(m2(1,2), m2(2,2),'o','Color', 'r');
plot(m2(1,3), m2(2,3),'o','Color', 'r');
plot(m2(1,4), m2(2,4),'o','Color', 'r');
text(m2(1,1), m2(2,1), 'a''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,2), m2(2,2), 'b''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,3), m2(2,3), 'c''', 'FontSize', FNT_SZ, 'Color', 'r');
text(m2(1,4), m2(2,4), 'd''', 'FontSize', FNT_SZ, 'Color', 'r');
```


ELLA
AT THE
HOLLYWOOD BOWL
THE IRVING
BERLIN SONGBOOK
d
c
b
a


d'
c'
b
a'

compute the homography

```
[H,A_collinear] = dlt_homography(m1,m2);

Hm1 = H*m1;
% dehomogenize points
Hm1 = Hm1./repmat(Hm1(3,:),3,1);
figure;
imshow(im2);
hold all;
MRK_SZ = 300;
scatter(m2(1,:),m2(2,:),MRK_SZ,'ro','filled');
scatter(Hm1(1,:),Hm1(2,:),MRK_SZ,'y+');

% the condition number of A is much larger than 1, the matrix is sensitive to the inverse calculation.
% let's see what happe to the nullspace of A when points are collinear
cond(A)
cond(A_collinear);
[~,s_collinear,~]=svd(A_collinear);
s_collinear = diag(s_collinear);
figure;
bar([s(end-1),s_collinear(end-1),s(end),s_collinear(end)])
```

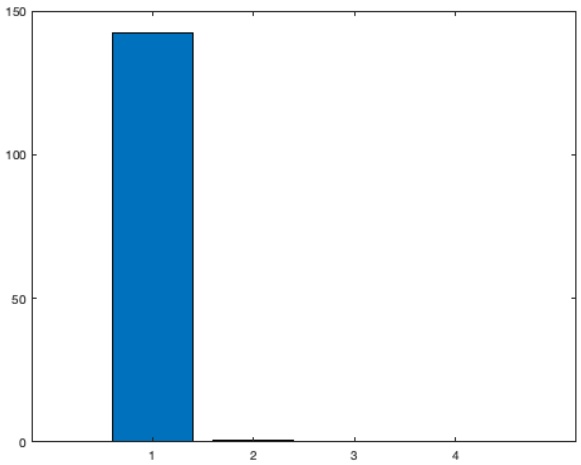
plotting the singular values of SVD decomposition, as we can see, the system is BAD CONDITIONED... 'Big Null space!

```
ans =

   1.3067e+07
```

the transformation compute well points on one BUT bad outside

↓

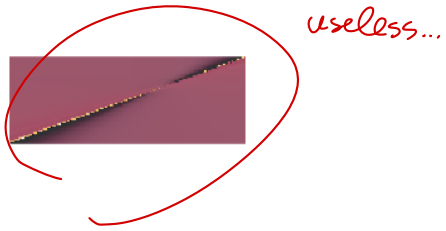
I get a useless transformation =0



let's show the result

```
t = maketform( 'projective', H');
J = imtransform(im1,t);


figure;
subplot(1,2,1);
imshow(im2);
subplot(1,2,2);
imshow(J);
```

```
Warning: The condition number of A is 1604493683560.505.
```



Published with MATLAB® R2021b

Always do computation,

BUT we should understand geometry

to know when computation make sense

seen homography transformation...

Image warping

We now have matlab function that takes (image, H) → output := transform image

Render an image transformed via an homography References: Andrea Fusiello, Visione Computazionale, Chapter 10 on image mosaicing

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

Image Analysis and Computer Vision Politecnico di Milano

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

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

"maketform"
under this transformation, to do this on our own it is easy to do on our own

Contents

from an image we know how to deform it, we want to paint new image distorted
I want to take each pixel and translate it and do for each pixel, translate the whole image

```
clear;
close all;
clc;
```

We are given: - two images im1 and im2 - the homography between im1 and im2 our aim is to synthetize a novel image obtained by superimposing im2 and H(im1). Let's use the backward mapping.

from a slice of the image, apply transform P that multiply by 2 all pixels as a translation With unknown values in the image ... FORWARD MAPPING is feasible BUT not good...
↳ usually backward mapping is used.

```
im1 = imread("E3_data/ella1.jpg");
im2 = imread("E3_data/ella2.jpg");
im1 = imresize(im1,0.6);
im2 = imresize(im2,0.6);
data = load('E3_data/ella_homo.mat');
m1 = data.m1;
m2 = data.m2;
H = data.H;
```

overlay the transformed im1 over im2

let's use bilinear interpolation

from target image, I move pixed by $P' = \frac{1}{2} P$, we check by inverting if inside properly

$x = 2 \xrightarrow{P'} x' = 1$ ... back project pixels to source to find if properly map... start from destination to same!

$x = 4 \rightarrow x' = 2$

loop through all img 2 pixels

if pixel q of im2 I compute pre image of q (inv homography)

h^{-1}

$(q = ph)$

```
canvas = im2;
lambda = 0.7;
for i=1:size(canvas,1)
    for j = 1:size(canvas,2)
        q =[j;i;1];
        % apply the inverse transformation
        p = inv(H)*q; % column is x and row is y
        p = p./p(3); % cartesian coordinate in the plane of the first image
        p = p;
        % if the preimage p lies inside the first image
        if(round(p(1))>0 && round(p(1))<=size(im1,2) && round(p(2))>0 && round(p(2))<=size(im1,1))
            for ch = 1:3
                source_value = im1(round(p(2)),round(p(1)),ch);
                % apply bilinear interpolation
                if(p(1)>1 && p(1)<size(im1,2) && p(2)>1 && p(2)<size(im1,1))
                    y1 = floor(p(2));
                    y2 = ceil(p(2));
                    % interpolate along the x axis
                    x1 = floor(p(1));
                    x2 = ceil(p(1));
                    % at y1
                    v1 = (p(1)-x1)*im1(y1,x1,ch) + (x2-p(1))*im1(y1,x2,ch);
                    % at y2
                    v2 = (p(1)-x1)*im1(y2,x1,ch) + (x2-p(1))*im1(y2,x2,ch);
                    % interpolate along the y axis
                    source_value = (p(2)-y1).*v1 + (y2-p(2)).*v2;
                end
                canvas(i,j,ch) = (1-lambda).*canvas(i,j,ch) + lambda.*source_value;
            end
        end

    end
end
```

↑ cartesian (x_1, x_2, y_1, y_2) → to decide color of pixel, we take bilinear combination

blending colors together

backward mapping to avoid any holes!
Possible because perspective transform (always invertible)

Image warping

Render an image transformed via an homography References: Andrea Fusiello, Visione Computazionale. Chapter 10 on image mosaicing

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

Image Analysis and Computer Vision Politecnico di Milano

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

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

Contents


```
clear;
close all;
clc;
```

We are given: - two images im1 and im2 - the homography between im1 and im2 our aim is to synthetize a novel image obtained by superimposing im2 and H(im1). Let's use the backward mapping.

```
im1 = imread("E3_data/ella1.jpg");
im2 = imread("E3_data/ella2.jpg");
im1 = imresize(im1,0.6);
im2 = imresize(im2,0.6);
data = load('E3_data/ella_homo.mat');
m1 = data.m1;
m2 = data.m2;
H = data.H;
```

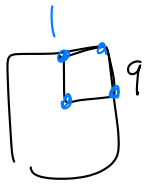
overlay the transformed im1 over im2

let's use bilinear interpolation

blending the colors, weighting linear combinations.

q → p , we get the value of the pixel coordinate...
we take bilinear combination of colors

summ the colors, such that



the color influency most are the closest ones
weighting the intensity by the area of neighbor points

bilinear interpolate

```
canvas = im2;
lambda = 0.7;
for i=1:size(canvas,1)
    for j = 1:size(canvas,2)
        q =[j;i;1];
        % apply the inverse transformation
        p = inv(H)*q; % column is x and row is y
        p = p./p(3); % cartesian coordinate in the plane of the first image
        p = p;
        % if the preimage p lies inside the first image
        if(round(p(1))>0 && round(p(1))<=size(im1,2) && round(p(2))>0 && round(p(2))<=size(im1,1))
            for ch = 1:3
                source_value = im1(round(p(2)),round(p(1)),ch);
                % apply bilinear interpolation
                if(p(1)>1 && p(1)<size(im1,2) && p(2)>1 && p(2)<size(im1,1))
                    y1 = floor(p(2));
                    y2 = ceil(p(2));
                    % interpolate along the x axis
                    x1 = floor(p(1));
                    x2 = ceil(p(1));
                    % at y1
                    v1 = (p(1)-x1)*im1(y1,x1,ch) + (x2-p(1))*im1(y1,x2,ch);
                    % at y2
                    v2 = (p(1)-x1)*im1(y2,x1,ch) + (x2-p(1))*im1(y2,x2,ch);
                    % interpolate along the y axis
                    source_value = (p(2)-y1).*v1 + (y2-p(2)).*v2;
                end
                canvas(i,j,ch) = (1-lambda).*canvas(i,j,ch) + lambda.*source_value;
            end
        end

    end
end
```

{ by definition of bilinear interpolation }

```
figure;
imshow(canvas);
% still some misalagnment are visible
```

a bigger canvas

first we must create a canvas to display our result. The canvas must be large enough to contain both im2 and im1. let's see where the corners of im1 are mapped

```
tl = H*[0;0;1]; % where top-left is mapped
tl = tl./tl(3);
tr = H*[size(im1,2);0;1]; % where top-right is mapped
tr = tr./tr(3);
bl = H*[0;size(im1,1);1]; % where bottom-left is mapped
bl = bl./bl(3);
br = H*[size(im1,2);size(im1,1);1];
br = br./br(3); % where bottom-right is mapped

% here we display the perimeter of im1 via H
figure;
imshow(canvas);
hold all;
plot(tl(1),tl(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
plot(tr(1),tr(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
```

to put one onsideof other, we can just use bigger canvas!
I check where points mapped and then compute Bounding box

← but image reference system needs to be moved!

↳ handle mappings to draw bigger picture

```
plot(bl(1),bl(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
plot(br(1),br(2),'ro','MarkerSize',10,'MarkerFaceColor','r');
line([tl(1),tr(1)],[tl(2),tr(2)],'Color','r','LineWidth',3);
line([tr(1),br(1)],[tr(2),br(2)],'Color','r','LineWidth',3);
line([br(1),bl(1)],[br(2),bl(2)],'Color','r','LineWidth',3);
line([bl(1),tl(1)],[bl(2),tl(2)],'Color','r','LineWidth',3);
```

then backward projection

determine the dimension of the canvas

```
w_max = max([size(im2,2),tl(1),tr(1),bl(1),br(1)]);
w_min = min([[size(im2,2),tl(1),tr(1),bl(1),br(1)]]);
w = ceil(w_max-w_min);
h_max = max([size(im2,1),tl(2),tr(2),bl(2),br(2)]);
h_min = min([[size(im2,1),tl(2),tr(2),bl(2),br(2)]]);
h = ceil(h_max-h_min);
big_canvas = uint8(255.*ones(h,w,3));
big_canvas(-h_min+1:-h_min+size(im2,1), -w_min+1:-w_min+size(im2,2),:) = im2;

figure;
imshow(big_canvas);
```

```
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
```

render the novel image using bilinear interpolation

```
for i=1:size(big_canvas,1)
    for j = 1:size(big_canvas,2)
        % bring [j;i;1] from the reference system of the big_canvas to the
        % reference system of im2
        q =[j+w_min;i+h_min;1];
        % apply the inverse transformation
        p = inv(H)*q; % column is x and row is y
        p = p./p(3); % cartesian coordinate in the plane of the first image
        % if the preimage p lies inside the first image
        if(round(p(1))>0 && round(p(1))<=size(im1,2) && round(p(2))>0 && round(p(2))<=size(im1,1))
            for ch = 1:3
                source_value = im1(round(p(2)),round(p(1)),ch);
```

express in proper reference system

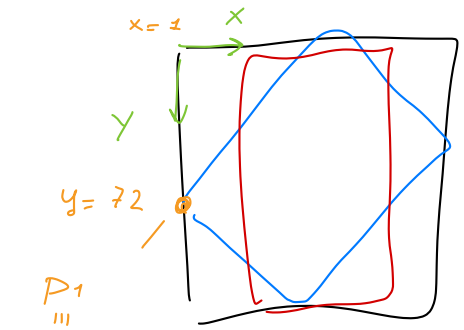
```
                % apply bilinear interpolation
                if(p(1)>1 && p(1)<size(im1,2) && p(2)>1 && p(2)<size(im1,1))
                    y1 = floor(p(2));
                    y2 = ceil(p(2));
                    % interpolate along the x axis
                    x1 = floor(p(1));
                    x2 = ceil(p(1));
                    % at y1
                    v1 = (p(1)-x1)*im1(y1,x1,ch) + (x2-p(1))*im1(y1,x2,ch);
                    % at y2
                    v2 = (p(1)-x1)*im1(y2,x1,ch) + (x2-p(1))*im1(y2,x2,ch);
                    % interpolate along the y axis
                    source_value = (p(2)-y1).*v1 + (y2-p(2)).*v2;
                end
                big_canvas(i,j,ch) = (1-lambda).*big_canvas(i,j,ch) + lambda.*source_value;
            end % end for channels
        end % end if inside the image
    end % end for j
end % end for i
```

```
figure;
imshow(big_canvas);
```

↳ there is an image size problem... so we need to loop to get the pre-image!

↓

the



↳ I take im1 and I see where lay! but H is defined between im1 → im2

you want to reference with offset...

so I need to subtract offset vetor...

then I need to look @ color with bilinear interpolation

↓

the PROBLEM is because homography is NOT defined on Big canvas BUT on 2nd image!

↑

I want to express pixel coord in 2nd image ref

We can draw the full image with a bigger canvas ↘

Published with MATLAB® R2021b

Stiching

Estimate an homography to stich togheter two images acquired by a rotating camera: - Normalization of the points - Direct Linear Transform - Transfer error minimization

References: Hartley Zisserman, Multiview Geometry Szeliski, Computer Vision .\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.\\°//.

Image Analysis and Computer Vision Politecnico di Milano

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

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

Contents

another possible application is IMAGE STITCHING, rotating camera keeping axis fixed! ROTATION around camera

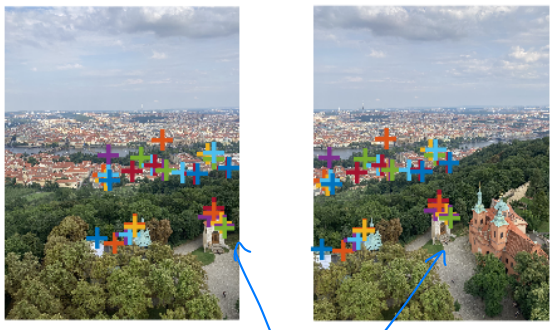
load images

```
clc;
close all;
clear;
```

```
im1 = imread("E3_data/pragaA.jpeg");
im2 = imread("E3_data/pragaB.jpeg");
im1 = imresize(im1,0.6);
im2 = imresize(im2,0.6);



%data = load('E3_data/ella_homo.mat');
%H0 = data.H;
```

get correspondences

```
do_load_correspondences = 1;

if(do_load_correspondences)
    data = load("E3_data/praga_matches.mat");
    m1 = double(data.m1);
    m2 = double(data.m2);

    % add a little bit of noise
    sigma = 0;
    m1(1:2,:)= m1(1:2,:) +sigma.*randn(2,size(m1,2));
    m2(1:2,:)= m2(1:2,:) +sigma.*randn(2,size(m2,2));
end
figure;
subplot(1,2,1);
imshow(im1);
hold all;
for i=1:size(m1,2)
    plot(m1(1,i),m1(2,i),'+','MarkerSize',15,'Linewidth',4);
end
subplot(1,2,2);
imshow(im2);
hold all;
for i=1:size(m2,2)
    plot(m2(1,i),m2(2,i),'+','MarkerSize',15,'LineWidth',4);
end
```



corresponding points taken in automatic way

I want to get a MOSAIC of the two images, as when we take PANORAMA photos..
↓
do this with HOMOGRAPHY

Preconditioning

to improve numerical stability data should be normalized in order to have zero mean and and mean distance to the center equal to sqrt(2). let's perfomr this transformation via a planar transformation Ti that translate the points and rescale them Reference: Hartley, Zisserman

```
% let's incapsulate this in a function normalize2D points
% and let's use it on m2
num_points = size(m1,2);
m1 = m1./repmat(m1(3,:),3,1);

centroid = mean(m1,2);
% distance from the centroid
scale = mean(sqrt(sum((m1-repmat(centroid,1,num_points)).^2,1)));

T1 = diag([sqrt(2)/scale;sqrt(2)/scale;1]);
T1(1:2,3)=-sqrt(2)*centroid(1:2)/scale;
m1_cond = T1*m1;

% let's encode this preconditioning step in a function
[m2_cond,T2] = precondition(m2);
```

in homogeneus coord points are not very balanced!
random points can have very different order, this cause matrix having points of 10^9, 10^{-1}... when solving DLT with so different orders can be solved by PRE-CONDITIONING

compute the homography via DLT

```
H0 = dlt_homography(m1_cond,m2_cond);

% Once we get the homography we have to get rid of the normalization
% process by applying the inverse transformations
H0 = T2\H0*T1;
H0 = H0./norm(H0);
```

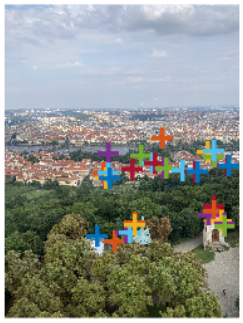
```
lambda = 0.5;
[mosaic_big,offset_x,offset_y] = image_mosaic_big(im1,im2,H0,lambda);
```

```
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
```

compute the errors

```
%H0(3,3) = 1e-9*randn(1,1);

% we consider the symmetric transfer error
hm1 = H0*m1;
hm1 = hm1./repmat(hm1(3,:),3,1);
invhm2 = inv(H0)*m2;
invhm2 = invhm2./repmat(invhm2(3,:),3,1);
transfer_error = 0;
for i = 1:size(m1,2)
    d1 = norm(m1(:,i)-invhm2(:,i));
    d2 = norm(m2(:,i)-hm1(:,i));
    transfer_error = transfer_error + (d1 + d2);
end
disp(transfer_error)
```

```
  113.5701
```



we apply geometry, transforming im1, im2 to make all points around origin in circle of radius 2!
I can do it!
⇓
HARTLEY NORMALIZATION
Change picture shape, putting all in a disc → all same magnitude!

having img1, img2 ⇒ estimate H? BUT imbalance coordinates!

affine transformation easy to invert, done for both ... homography done on this planes

Preconditioning

to improve numerical stability data should be normalized in order to have zero mean and and mean distance to the center equal to sqrt(2). let's perfomr this transformation via a planar transformation Ti that translate the points and rescale them Reference: Hartley, Zisserman

```
% let's incapsulate this in a function normalize2D points
% and let's use it on m2
num_points = size(m1,2);
m1 = m1./repmat(m1(3,:),3,1);

centroid = mean(m1,2);
% distance from the centroid
scale = mean(sqrt(sum((m1-repmat(centroid,1,num_points)).^2,1)));
T1 = diag([sqrt(2)/scale;sqrt(2)/scale;1]);
T1(1:2,3)=-sqrt(2)*centroid(1:2)/scale;
m1_cond = T1*m1;

% let's encode this preconditioning step in a function
[m2_cond,T2] = precondition(m2);
```

[0 0 1] as last = affine transform TRANSLATION & SCALING!

$\begin{bmatrix} \text{diag} & t \\ 0\,0 & 1 \end{bmatrix}$

then final homography $\tilde{H} \cdot N^{-1}$ inverse, and numerical instability

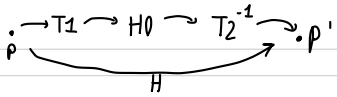
- good practice to normalize put point to get better condition numbers!

compute the homography via DLT

```
H0 = dlt_homography(m1_cond,m2_cond);

% Once we get the homography we have to get rid of the normalization
% process by applying the inverse transformations
H0 = T2\H0*T1;
H0 = H0./norm(H0);
```

final homography as

render the mosaic




```
lambda = 0.5;
[mosaic_big,offset_x,offset_y] = image_mosaic_big(im1,im2,H0,lambda);
```

```
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
```

compute the errors

to check how well homography fit the data we use squared difference between points and projected

```
%H0(3,3) = 1e-9*randn(1,1);

% we consider the symmetric transfer error
hm1 = H0*m1;
hm1 = hm1./repmat(hm1(3,:),3,1);
invhm2 = inv(H0)*m2;
invhm2 = invhm2./repmat(invhm2(3,:),3,1);
transfer_error = 0;
for i = 1:size(m1,2)
    d1 = norm(m1(:,i)-invhm2(:,i));
    d2 = norm(m2(:,i)-hm1(:,i));
    transfer_error = transfer_error + (d1 + d2);
end
disp(transfer_error)
```

```
113.5701
```

— we get error as squared algebraic error

```
cmap = brewermap(size(m1,2),'Paired');
figure;
imshow(mosaic_big);
hold all;
for i=1:size(m1,2)
    plot(m2(1,i)+offset_x,m2(2,i)+offset_y,'o','MarkerSize',25,'Linewidth',4,'Color',cmap(i,:));
    plot(hm1(1,i)+offset_x,hm1(2,i)+offset_y,'+','MarkerSize',25,'Linewidth',4,'Color',cmap(i,:));
end
```

Im fitting problem
↙ ALGEBRAIC ↘ GEOMETRIC
in the bending of line, Algebraic error change a lot, while geometric NOT!
↓
if in HOMOGRAPHY we minimize Algebraic error we get error that depends on how homography lay in space (NOT invariant under geometric transform)
↑
after you compute $\tilde{H}$, estimate then you feed $\tilde{H}$ to optimizer function NON LINEAR which minimize geometric error
↳ IMPROVE a lot!

① DLT to get $\tilde{H}$ minimize Algebraic error: how close are correspondances

iterative minimization ↘ we want to minimize a geometric error! (NOT algebraic)

```
MaxIterations = 1e5;
FunctionTol = 1e-8;
StepTol = 1e-16;

options = optimoptions('lsqnonlin','Algorithm','levenberg-marquardt',...
    'MaxIterations',MaxIterations,...
    'StepTolerance', StepTol, 'FunctionTolerance', FunctionTol,'Display','iter','Diagnostics','on');

fobj = @(h)get_transfer_errors(h,m1,m2);
h0 = H0(:);
h = lsqnonlin(fobj,h0,[],[],options);
H = reshape(h,3,3);
H = H./norm(H);
```

② from $\tilde{H}$, we get geometric error which is more significative (more informative)

solved from a starting point as the estimate $\tilde{H}$

```
   Diagnostic Information

Number of variables: 9

Functions
Objective:                              @(h)get_transfer_errors(h,m1,m2)
```

↑ we wanna minimize that, NON LINEAR.. that require solver!

from $\tilde{H}$ guess + definition of reprojection error

feed both to NON LINEAR SOLVER, that find a better solution H^*

```
Gradient:                            finite-differencing

Number of lower bound constraints:          0
Number of upper bound constraints:          0

Algorithm selected
   levenberg-marquardt


_____________________________________________________________
   End diagnostic information

                                    First-Order                    Norm of
 Iteration  Func-count     Residual       optimality      Lambda           step
     0          10          391.935        4.38e+11         0.01
     1          41                                                 5.70508e-17

Local minimum possible.
lsqnonlin stopped because the relative size of the current step is less than
the value of the step size tolerance.
```

```
lambda = 0.5;
[mosaic_big,offset_x,offset_y] = image_mosaic_big(im1,im2,H,lambda);
```

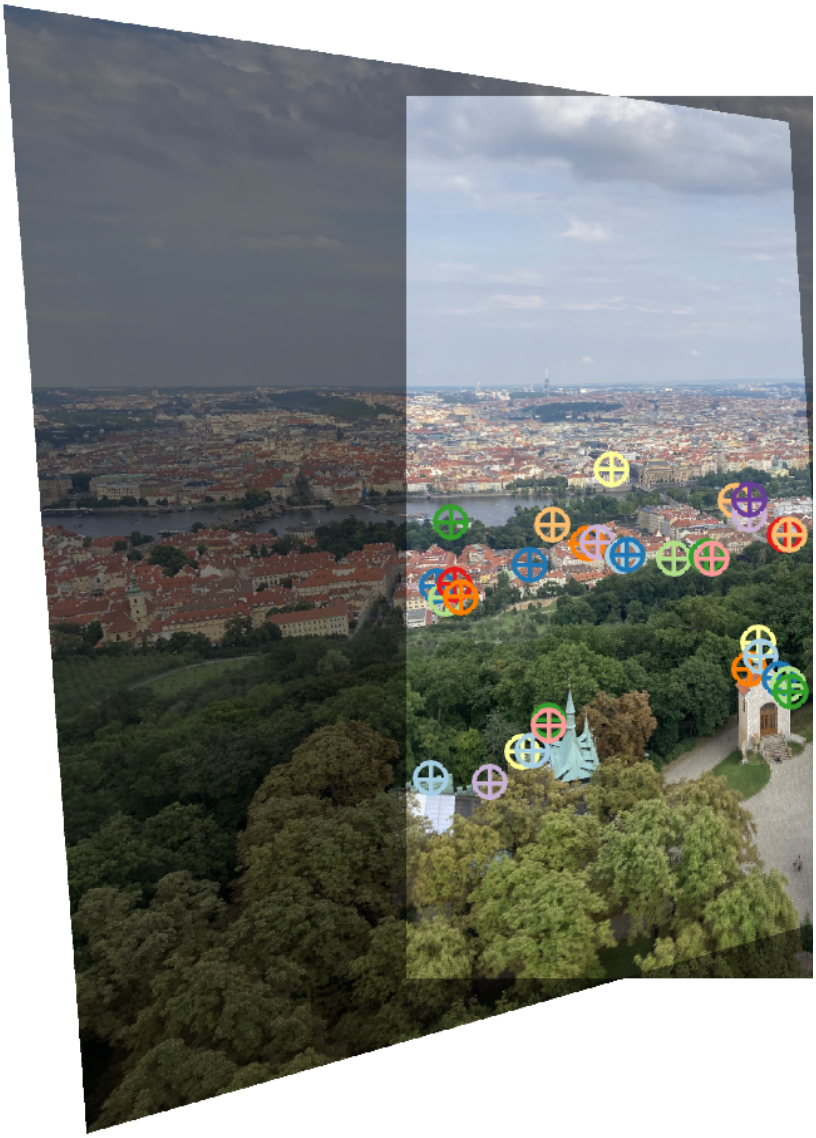
```
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
Warning: Integer operands are required for colon operator when used as index.
```

compute the errors

we consider the symmetric transfer error

```
hm1 = H*m1;
hm1 = hm1./repmat(hm1(3,:),3,1);
invhm2 = inv(H)*m2;
invhm2 = invhm2./repmat(invhm2(3,:),3,1);
transfer_error = 0;
for i = 1:size(m1,2)
    d1 = norm(m1(:,i)-invhm2(:,i));
    d2 = norm(m2(:,i)-hm1(:,i));
    transfer_error = transfer_error + (d1 + d2);
end
disp(transfer_error)
```

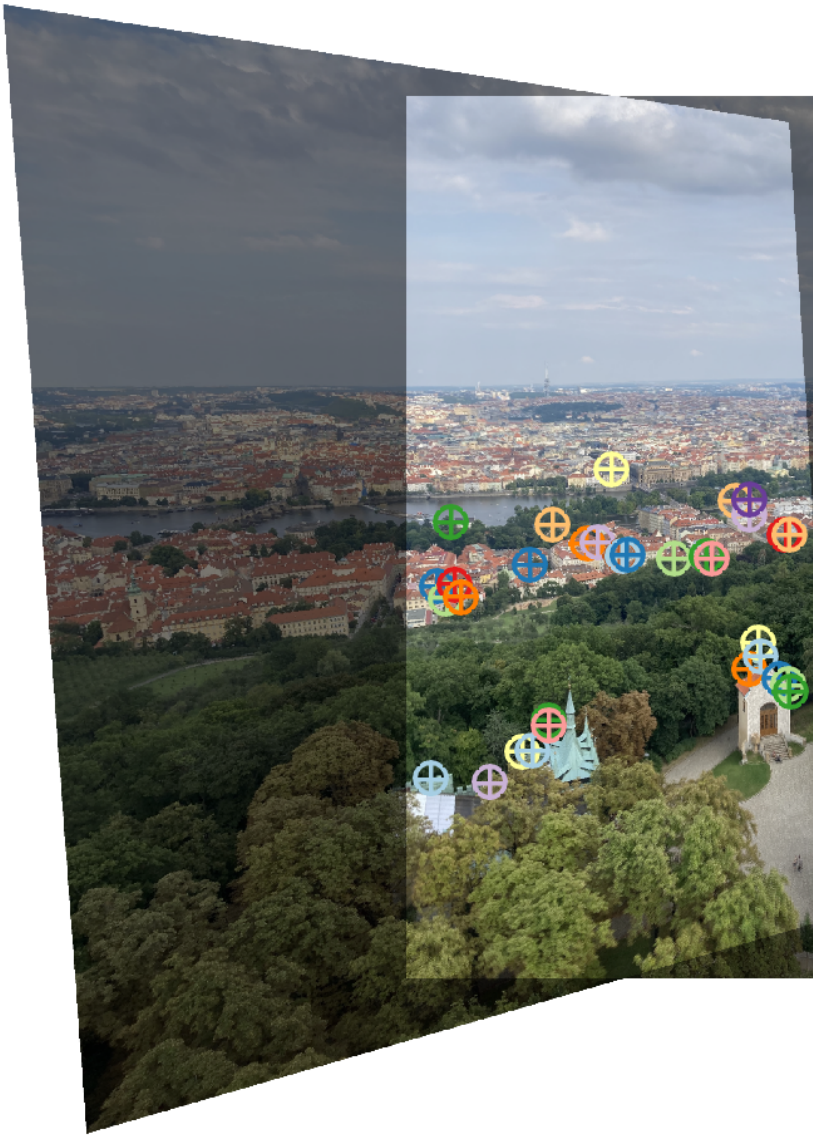
```
  113.5701
```



im new image, error get decreased!

```
cmap = brewermap(size(m1,2),'Paired');
figure;
imshow(mosaic_big);
hold all;
for i=1:size(m1,2)
    plot(m2(1,i)+offset_x,m2(2,i)+offset_y,'o','MarkerSize',25,'Linewidth',4,'Color',cmap(i,:));
    plot(hm1(1,i)+offset_x,hm1(2,i)+offset_y,'+','MarkerSize',25,'Linewidth',4,'Color',cmap(i,:));
end


H./H(3,3)
% Affine homographies have their third row fixed to
% [0, 0, 1] and arise when the image displacements come from a small
% image region or a large focal length lens is being used. When
% estimating affine homographies, the non-linear refinement is omitted: Non-linear refinement is unnecessary
% for them because in that case, the algebraic error equals the
% geometric one.
```

```
ans =

   1.0e+03 *

    0.0014    0.0001   -1.1085
    0.0002    0.0013   -0.2522
    0.0000    0.0000    0.0010
```



Published with MATLAB® R2021b

IF we have multiple images we can combine it
to get full scene mosaic

Rectification

Another application

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

Image Analysis and Computer Vision Politecnico di Milano

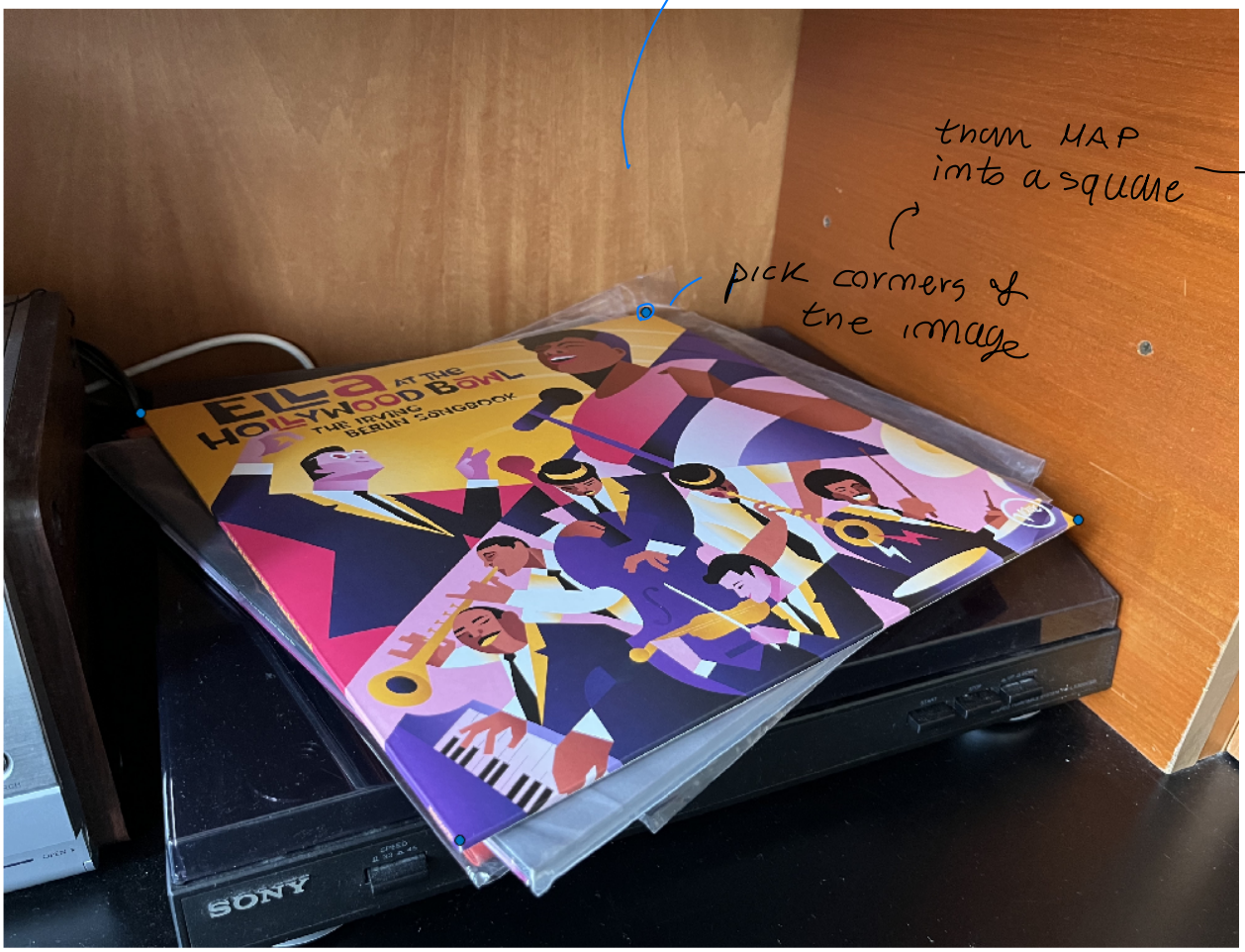
Luca Magri for comments and suggestions please send an email to luca.magri@polimi.it

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

```
clear;
clc;
close all;
```

```
im = imread('E3_data/giradischi.jpg');
figure;
imshow(im);

a = drawpoint();
b = drawpoint();
c = drawpoint();
d = drawpoint();
```

how to have nice pictures it?



then MAP into a square

pick corners of the image

```
m1 = [a.Position;b.Position;c.Position;d.Position];
m1 = [m1'; ones(1,4)];
l = 500;
m2 =[1 l l 1;...
     1 1 l 1;...
     1 1 1 1];
```

} positions of square of dimension 500

```
[m1_normalized,T1] = precondition(m1);
[m2_normalized,T2] = precondition(m2);
H = dlt_homography(m1,m2);
%H = inv(T2)*H*T1;
H = H./norm(H);
```

} normalize all..

```
lambda =1;
[mosaic] = image_mosaic_same(im,uint8(zeros(1,1,3)),H,lambda);
figure; imshow(mosaic)
```

MAP image pick corners to a SQUARE!

render the image as Rectified

(this is what happens when you scan in the phone!)

↓

RECTIFICATION as if parallel img

Published with MATLAB® R2021b

NOTICE: homography is defined up to scale factor ↳ to avoid numerical instability is better to have determinant constrained

It is a nice idea to normalize all by norm(MATRIX)... OR you get too big numbers

Strange results can be caused by BAD numeric computation... NOT normalization, make small entries and divide by Forbenious norm