



Computer Vision 11 – Image alignment and matching 1

doc. dr. Janez Perš
(with contributions by prof. Stanislav Kovačič)

Laboratory for Machine Intelligence
Faculty of Electrical Engineering
University of Ljubljana

Quick recap of the previous lectures

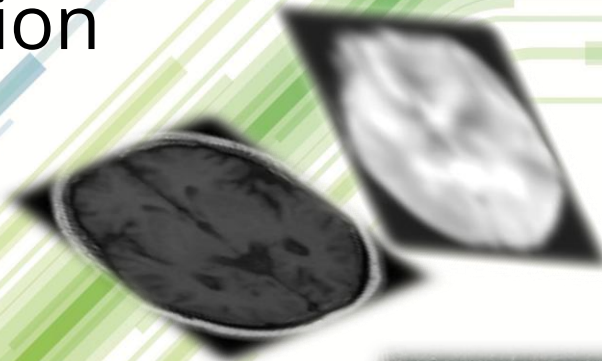
- Image formation
- Color
- Image processing
- Edges, Hough transform
- Segmentation, morphological operations, region boundaries
- Region descriptors, moments

Outline

- Geometric transformations
- Image interpolation
- Least squares
- RANSAC
- Tutorial on image moments
 - If time allows

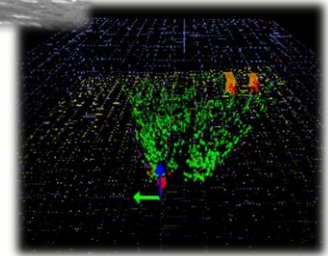
Motivation & uses

- Image registration
 - Medical
- Image stitching
 - Panoramas
- Object tracking
 - Where is object X now?

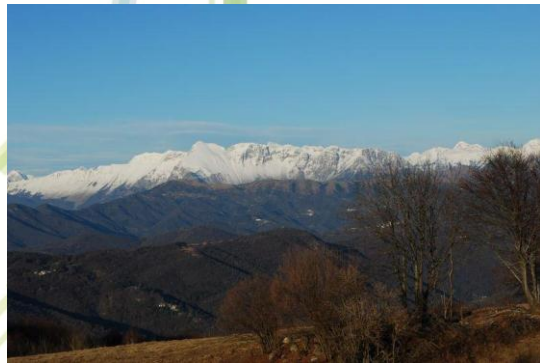


Motivation & uses

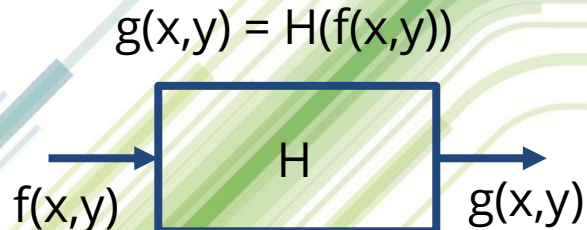
- Stereo matching
 - To obtain 3D data
- Model fitting
 - Robust methods!



Geometric transformations on images

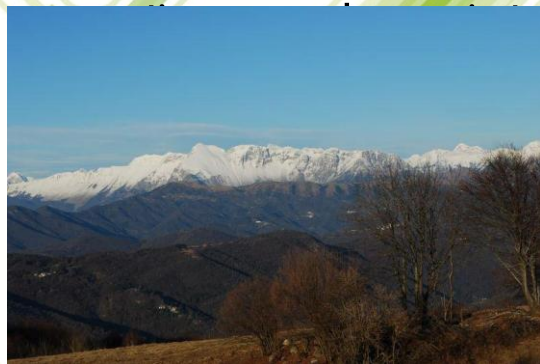


$f(x,y)$

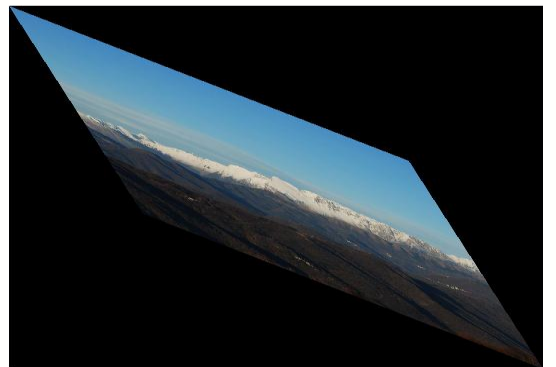
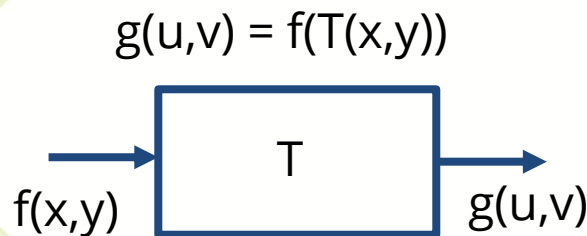


$g(x,y)$

- Image filtering, etc., work on pixel values (image range)
- Geometric image transformations work on pixel positions



$f(x,y)$



$g(u,v)$

Geometric transformations on images

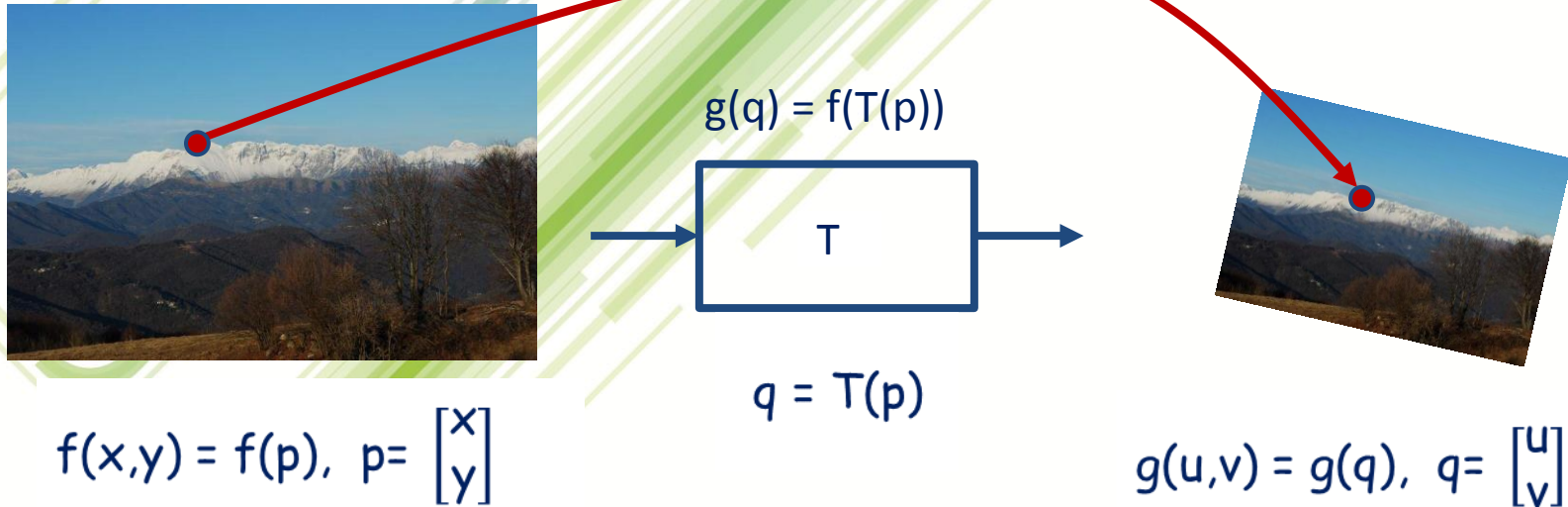
- Uses of geometric image transformations are numerous:
 - geometric correction of images, for
 - size, position, orientation, skew, ...
 - geometric registration of two or more images
 - image to image, model to image, ...
 - image stitching, image mosaics, panorama building.



An example of highly overlapping image stitching

Global vs. local transformations

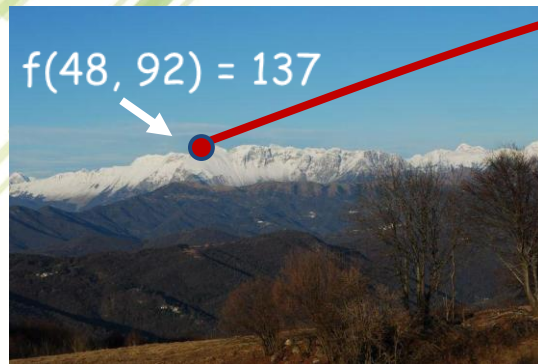
- Global transformations
 - all pixels undergo the same geometric transformation



- Local transformations
 - act only on a particular part (subdomain) of the whole image
 - Could be different for different parts of image.

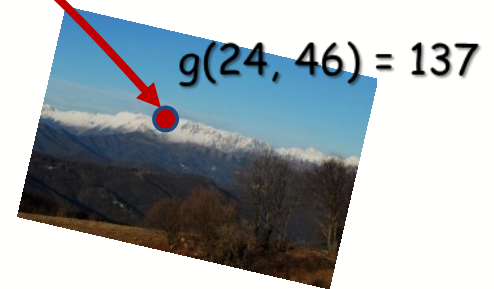
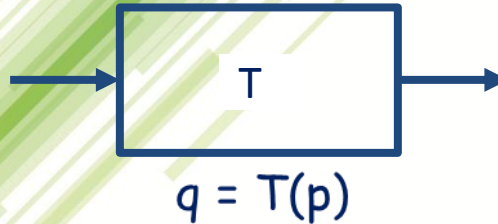
Global linear transformations

- Linear transformations can be written in matrix form!



$$f(x, y) = f(p), \quad p = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$g(q) = f(T(p))$$



Global: all pixels undergo the same geometric transformation

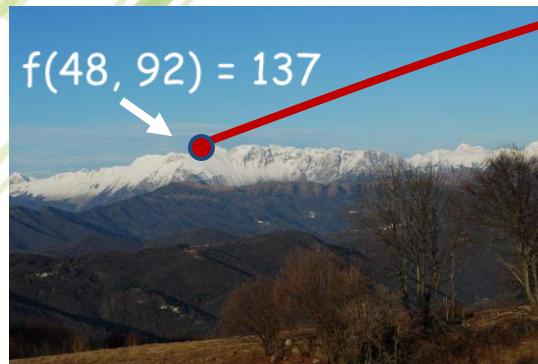
$$\begin{bmatrix} u \\ v \end{bmatrix} = T \begin{bmatrix} x \\ y \end{bmatrix}, \quad g(q) = f(T p)$$

Simple numerical example

$$\begin{bmatrix} 24 \\ 46 \end{bmatrix} = T \begin{bmatrix} 48 \\ 92 \end{bmatrix}, \quad g(24, 46) = f(48, 92)$$

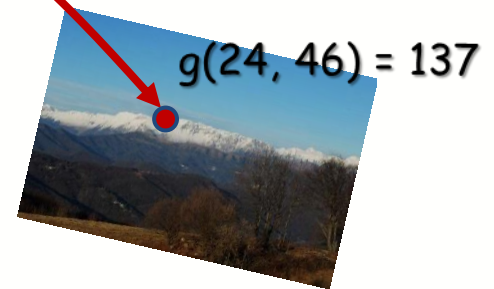
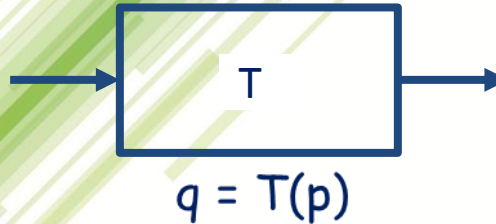
Image origin and 2D coordinates

- Linear transformations can be written in matrix form!



$$f(x, y) = f(p), \quad p = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$g(q) = f(T(p))$$



Global: all pixels undergo the same geometric transformation

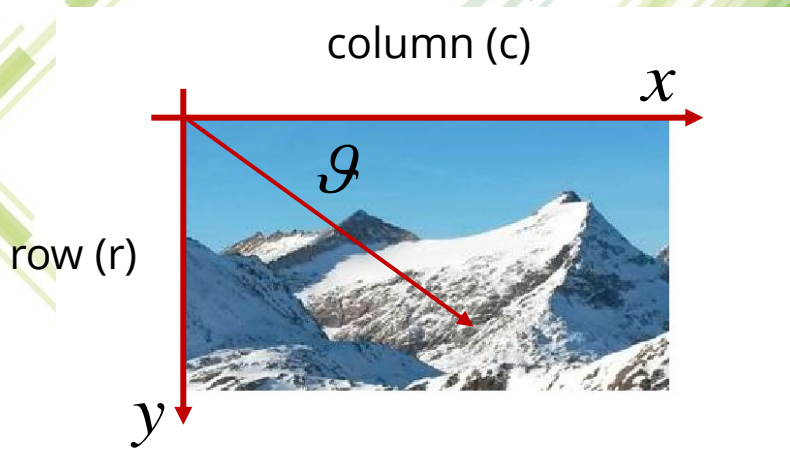
$$\begin{bmatrix} u \\ v \end{bmatrix} = T \begin{bmatrix} x \\ y \end{bmatrix}, \quad g(q) = f(T p)$$

Simple numerical example

$$\begin{bmatrix} 24 \\ 46 \end{bmatrix} = T \begin{bmatrix} 48 \\ 92 \end{bmatrix}, \quad g(24, 46) = f(48, 92)$$

Image origin and 2D coordinates

- We are going to use right-handed coordinate systems all the time.



\mathcal{G} positive angle (rotation)

Remember:

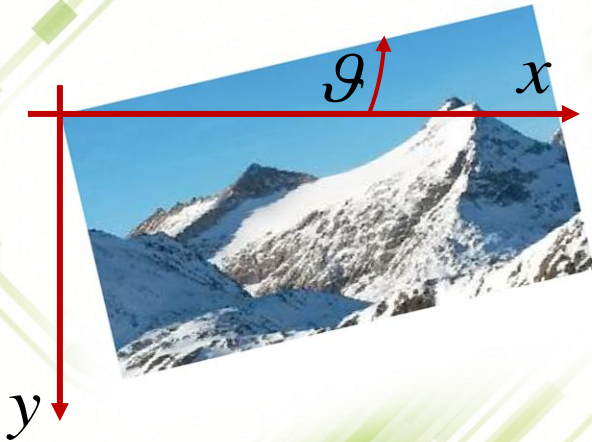
$f(x,y)$, mathematically, and

$I(r, c) = I(y, x)$, in Matlab

And sometimes, the positive direction of rotation will be the opposite.

Linear transformations

- Important:
 - Applying transformation to image is equivalent to applying inverse transformation to the coordinate system.



- For example:
 - Rotating an image in the negative direction is equivalent to rotating coordinates in the positive direction.

Linear transformations

Also keep in mind:

$$\begin{bmatrix} u \\ v \end{bmatrix} = T \begin{bmatrix} x \\ y \end{bmatrix}$$

Where T is a linear transformation (matrix)

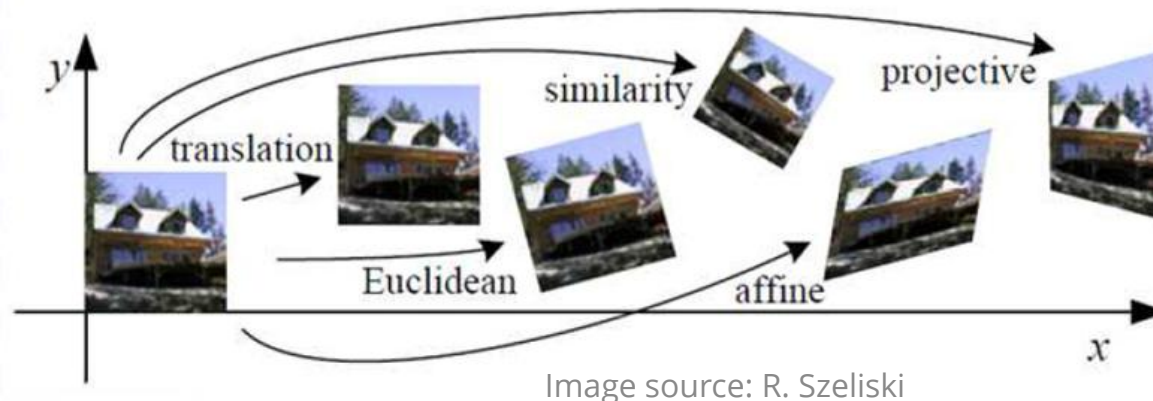
But, due to the rule of transposition:

$$q^T = (Tp)^T = p^T T^T$$

$$\begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} T^T$$

In the literature, and in implementations, you are going to find both forms.

Basic geometric transformations



- Basic types of transformations:
 - translation, rotation
 - rigid (Euclidean)
 - similarity (shape preserving)
 - affine
 - projective
- Not so basic:
 - Curved or Deformable (not shown)

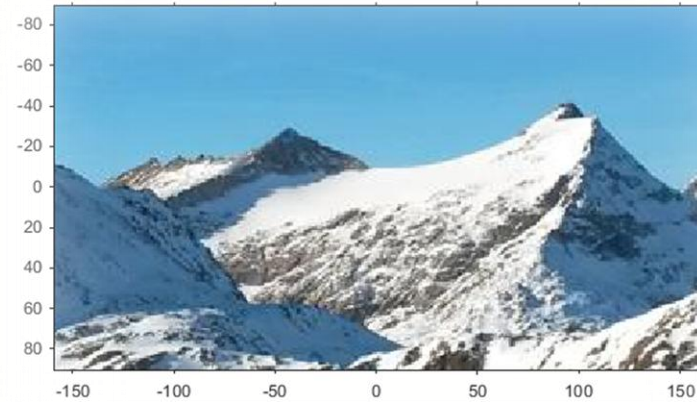
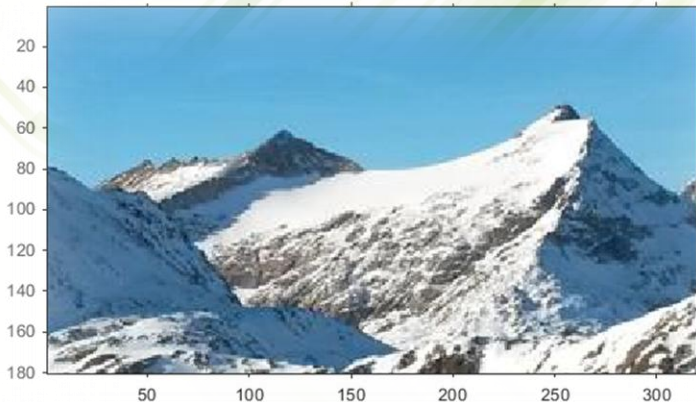
Translation

Translation can be written (mathematically) as vector addition

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

Matlab: `imtranslate`

As an example we translate our image such that the center of the image coincides with the origin of the coordinate system.

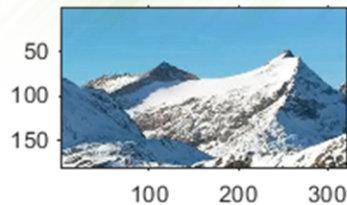


We see no difference in the displayed images, except that the values along coordinates are different.

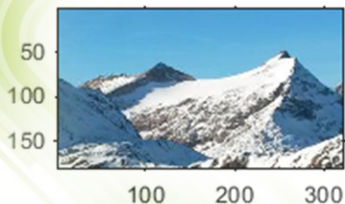
Translation

Nevertheless, in Matlab, what we see on the screen depends on the “Viewing window” through which we observe the image.

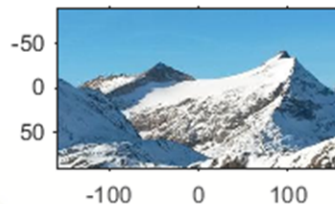
Original



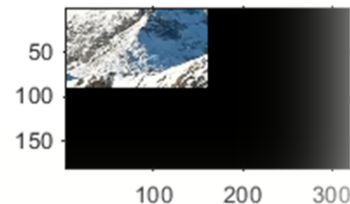
Translated image displayed w.r.t. intrinsic coordinates



Translated image displayed w.r.t. world coordinates



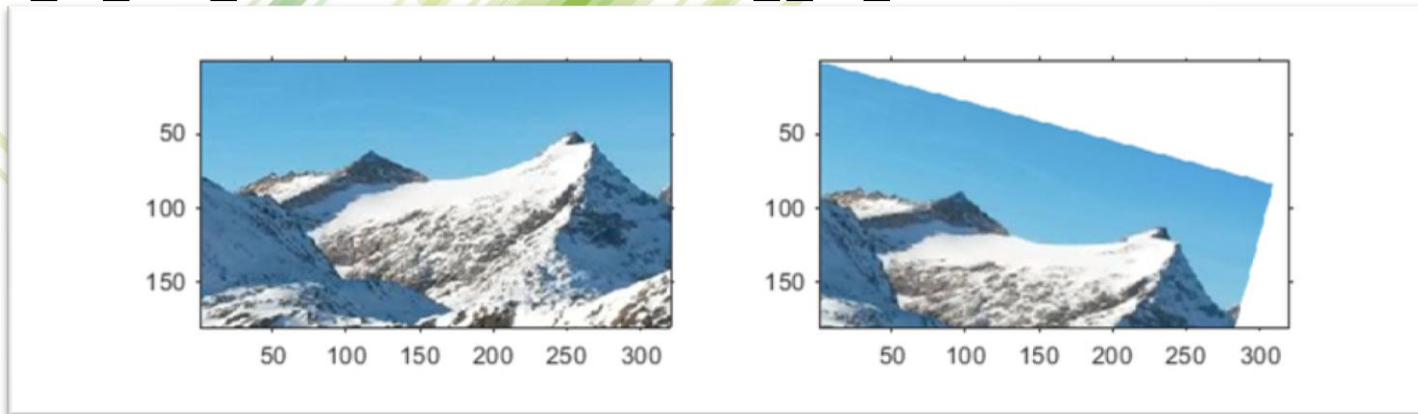
Translated image displayed w.r.t. old intrinsic coordinates



Rotation

- Rotation around the image origin
 - You can derive this by using polar coordinates

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(\mathcal{Q}) & -\sin(\mathcal{Q}) \\ \sin(\mathcal{Q}) & \cos(\mathcal{Q}) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



- Image rotated for 15 degrees clockwise

Translation plus rotation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(\mathcal{Q}) & -\sin(\mathcal{Q}) \\ \sin(\mathcal{Q}) & \cos(\mathcal{Q}) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = R \begin{bmatrix} x \\ y \end{bmatrix} + t$$

It is more convenient to use homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\mathcal{Q}) & -\sin(\mathcal{Q}) & x_t \\ \sin(\mathcal{Q}) & \cos(\mathcal{Q}) & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation plus rotation

Pure translation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pure rotation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\mathcal{J}) & -\sin(\mathcal{J}) & 0 \\ \sin(\mathcal{J}) & \cos(\mathcal{J}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = R \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation plus translation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\mathcal{J}) & -\sin(\mathcal{J}) & x_t \\ \sin(\mathcal{J}) & \cos(\mathcal{J}) & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [T \cdot R] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation plus rotation

- Linear transformations can be chained easily.
- Let' say that, instead about origin, we want to rotate an image around its center (x_c, y_c) .
- Then, we can first translate the image center to the origin,
- rotate the image, and then translate it back.

$$T = \begin{bmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos(\mathcal{Q}) & -\sin(\mathcal{Q}) & 0 \\ \sin(\mathcal{Q}) & \cos(\mathcal{Q}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

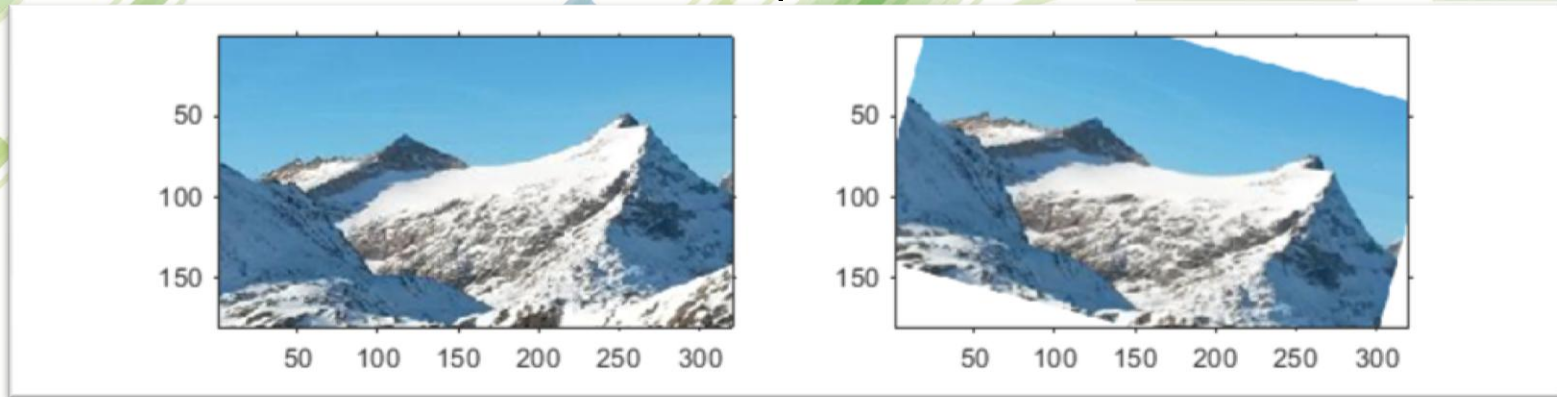
$$G = T^{-1} \cdot R \cdot T$$

- G is therefore composition of three elementary transformations.

Translation plus rotation example

Rotate 15 deg. clockwise around image origin

Matlab: `imrotate(imI,-15,'crop')`:



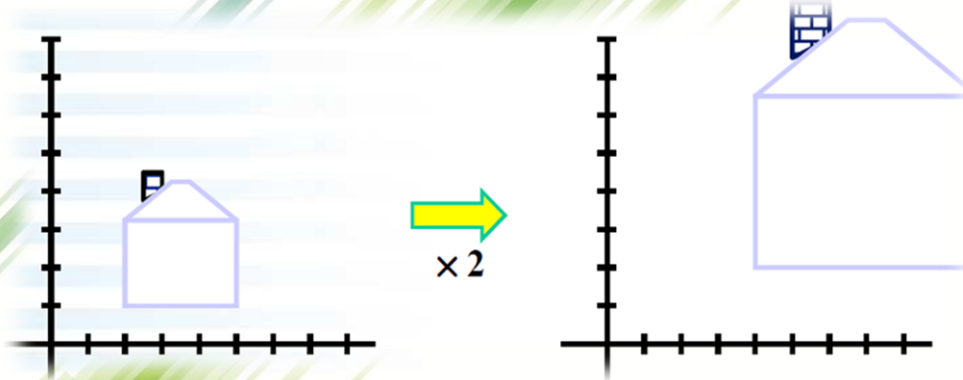
$$T = \begin{bmatrix} 1 & 0 & -160 \\ 0 & 1 & -90 \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} 0.9659 & -0.2588 & 0 \\ 0.2588 & 0.9659 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 160 \\ 0 & 1 & 90 \\ 0 & 0 & 1 \end{bmatrix} \quad G = T^{-1} \cdot R \cdot T = \begin{bmatrix} 0.9659 & -0.2588 & 28.7456 \\ 0.2588 & 0.9659 & -38.3444 \\ 0 & 0 & 1 \end{bmatrix}$$

Matlab:

```
RI = imref2d(size(imI)); % imI is input image
tformG = affine2d(G'); % as required by the affine2d function, see help
imR = imwarp(imI, tformG, 'OutputView',RI, 'FillValue',1);
figure, subplot(1,2,1), imshow(imI), subplot(1,2,2), imshow(imR);
```

Uniform and arbitrary scaling

- Uniform scaling:
 - All coordinates are multiplied with the same scaling factor.
 - Uniform scaling preserves shape, only size is changed.



- Arbitrary scaling (also called stretching):
 - Coordinates are multiplied with different scaling factors.
 - Arbitrary scaling does not preserve shape.

Homogeneous transformations

Translation

$$T = \begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation around origin

$$R = \begin{bmatrix} \cos(\vartheta) & -\sin(\vartheta) & 0 \\ \sin(\vartheta) & \cos(\vartheta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shear (skew)






$$S = \begin{bmatrix} 1 & s_x & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Projective

$$P = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

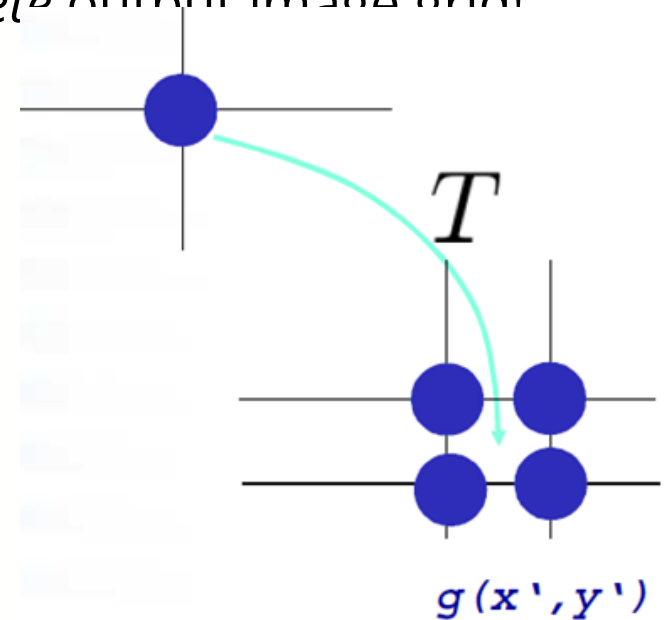
Sometimes, these matrices, excluding projective, are written as 2x3 matrices, with the last row omitted.

Basic 2D transformations - overview

Transformation	Matrix	DoF	Preserves	Symbol
Translacija translation	$[I t]_{2 \times 3}$	2	smer orientation	
Toga (Evklidska) rigid	$[R t]_{2 \times 3}$	3	dolžine distances	
Podobnostna similarity	$[sR t]_{2 \times 3}$	4	koti angles	
Afina affine	$[A]_{2 \times 3}$	6	vzporednost parallelism	
Projektivna projective	$[H]_{3 \times 3}$	8	ravnost straightness	

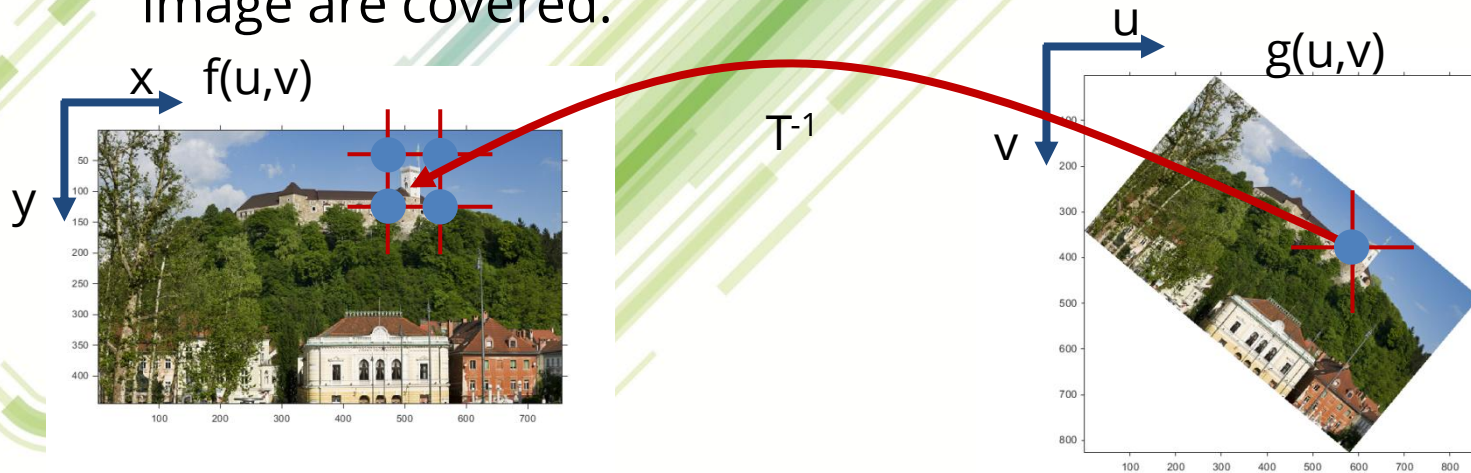
The problem

- When performing arbitrary transformation T that maps image $f(u,v)$ to $g(u,v)$
 - The pixel position after translation is probably *not* a round value!
 - So it does not map nicely to a *discrete* output image grid!
- What to do?
 - Simply rounding the x' and y' to the nearest integer?
- Possible problems:
 - More than one pixel from f maps to the same pixel in g
 - None of pixels in f maps to pixel position in g .



Proper procedure

- Start from output image and apply inverse transformation. Take pixel value from the input image and map it to the output image. Therefore, all pixel positions in the output image are covered.

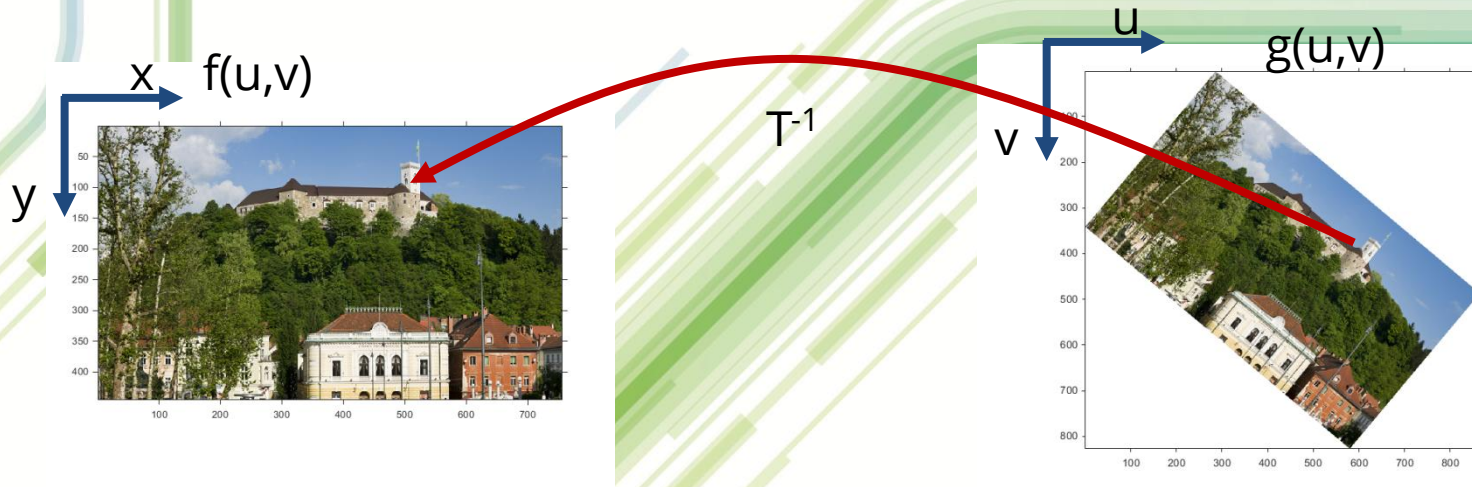


- But now coordinates in the input image are not necessarily on the grid!



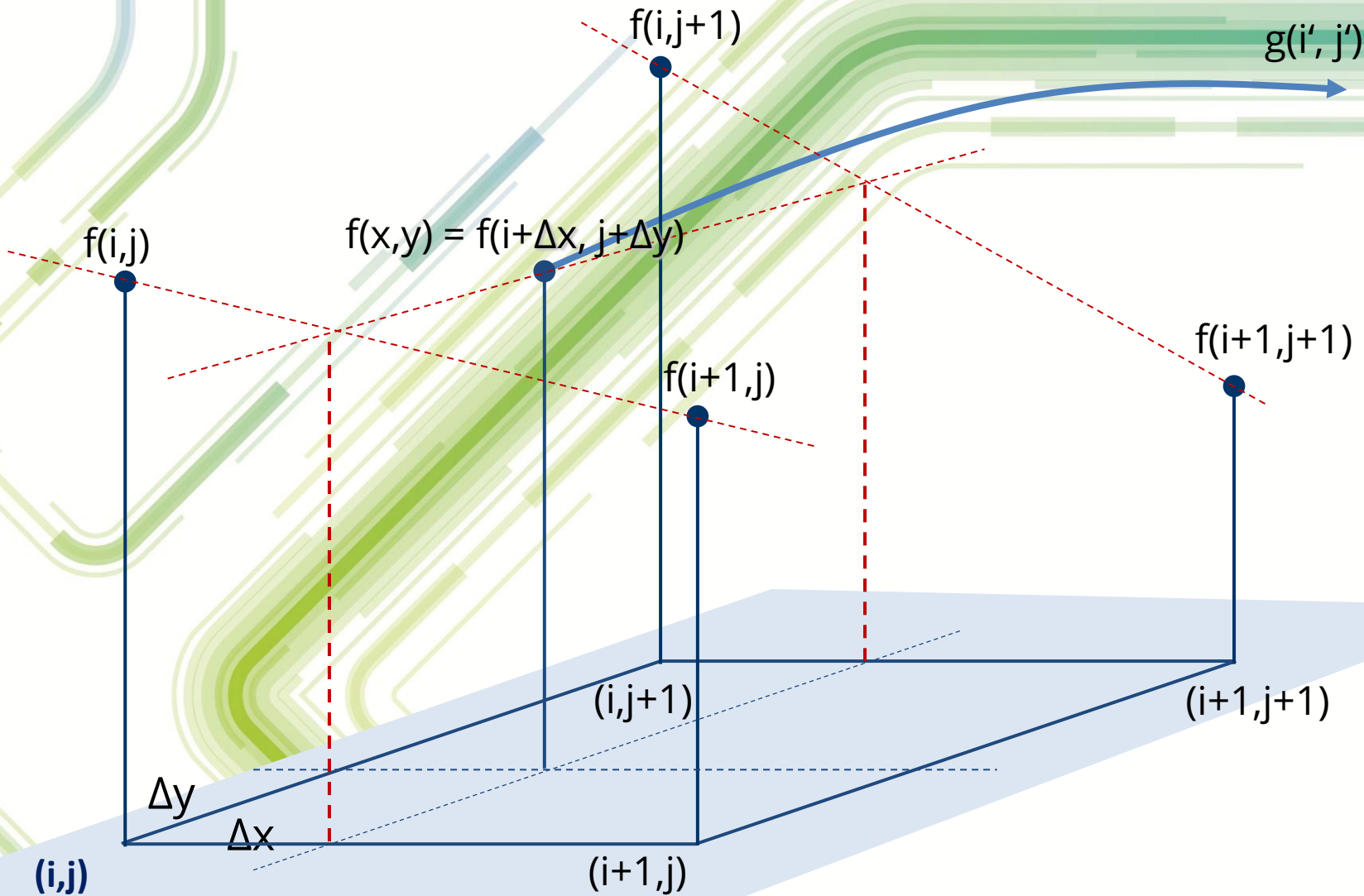
- Therefore we use both *inverse transformation* and *interpolation*!

Interpolation

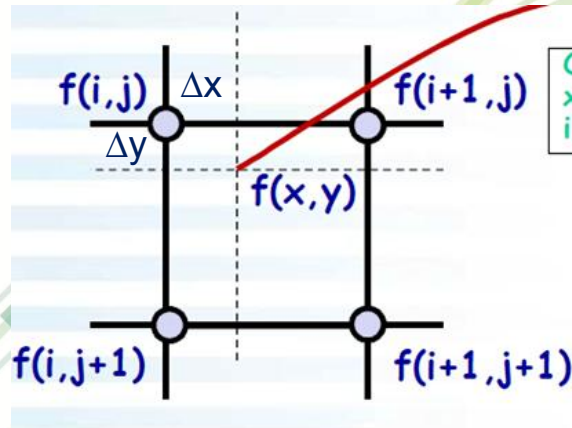


- Interpolation options:
 - Nearest neighbor (zero order, not recommended)
 - Bi-linear (linear in x and y direction, recommended)
 - Bi-cubic
 - Splines (zlepki)
- For many practical applications *bilinear interpolation* is sufficient!

Bilinear interpolation



Bilinear interpolation



- 1. step: Linear interpolation along x

$$f(x,j) = f(i,j) * (1-\Delta x) + f(i+1,j) * \Delta x$$

$$f(x,j+1) = f(i,j+1) * (1-\Delta x) + f(i+1,j+1) * \Delta x$$

- 2. step, linear interpolation along y

$$f(x,y) = f(x,j) * (1-\Delta y) + f(x,j+1) * \Delta y$$

- Final mapping of pixel value

$$g(i',j') = f(x,y)$$

All this in Matlab

imresize, imrotate, imtranslate, imcrop
imtransform
maketform
affine2d, projective2d
imwarp
...

Geometric transformation is frequently applied as the final step.
First, you define transformation, and then you apply it.

Downsampling and upsampling

- Changing the image resolution
 - Downsampling (Downscaling, Decimation) – resolution is lowered
 - Upsampling (Upscaling, Interpolation) - resolution increases
 - Not used very often – with upsampling, the amount of information in the image remains the same, but no. of pixels increases.
- Before downsampling, all frequencies that cannot be represented with lower resolution should be removed
 - Using filtering/smoothing!
 - Otherwise, result may have nasty artifacts.

Fitting and matching – motivation

- We have a model (a reference, a prototype) and data (or target).
- Our task is to ,explain' the data in terms of model.
- We fit, or register, the model onto the data.
- Specifically, we search for the mapping that maps the model onto data.
- Model could be parametric, geometric, pictorial, etc.
- Data could be an image, a set of images, part of image, distinctive image features, etc.
- This is the province of fitting and registration.

Fitting, matching, registration

Fitting line to image points, i.e.
estimate line parameters, m and b

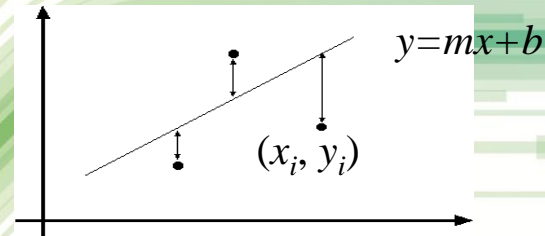


Image matching, estimate transformation parameters to align
(,register') two images, e.g.
estimate homography parameters,
homography matrix.

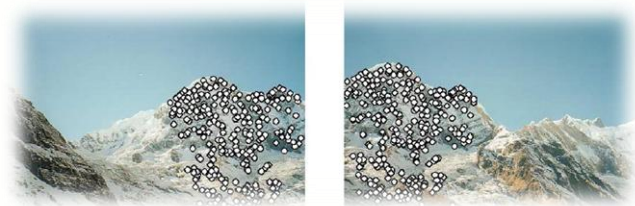
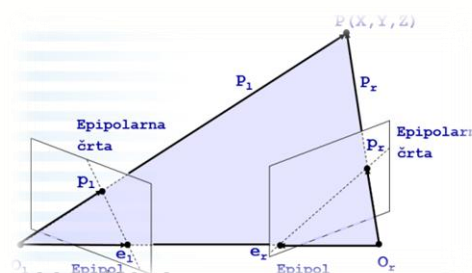


Image source: Brown & Lowe

Determine epipolar geometry, estimate essential matrix and / or
fundamental matrix.



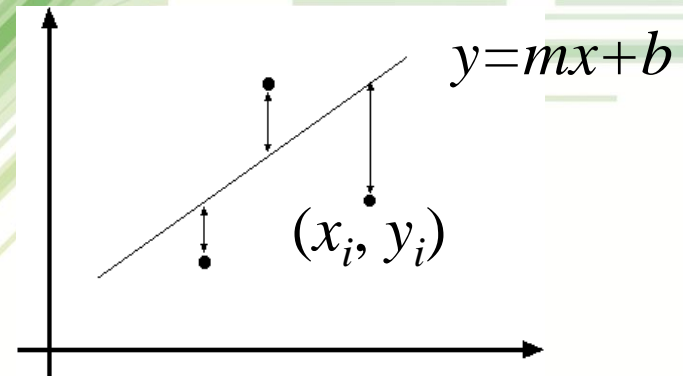
Least squares for line fitting

Measured data: $(x_1, y_1), \dots, (x_n, y_n)$

Model, line equation: $y = f(x) = m x + b$

Goal:

Fit model to data, i.e. find parameters of the line (m, b)



Due to noise (e), equations $y_i = m x_i + b$, $(i = 1, 2, \dots, n)$ simultaneously never hold exactly, $e_i = y_i - m x_i - b$, $(i = 1, 2, \dots, n)$

Thus, find (m, b) that minimize the 'objective' function

$$E(m, b) = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - m x_i - b)^2$$

This is the method known as the method of (linear) least squares

Least squares for lines

$$\min_{m,b} \{E(m,b)\} = \min \left\{ \sum_{i=1}^n e_i^2 \right\} = \min \left\{ \sum_{i=1}^n (y_i - mx_i - b)^2 \right\}$$



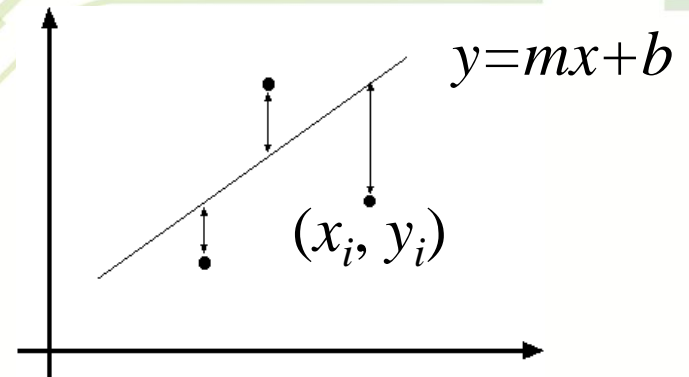
Minimizing, derivative = 0!

$$\frac{\partial E}{\partial m} = -2 \sum_{i=1}^n (y_i - mx_i - b)x_i = 0$$

$$\frac{\partial E}{\partial b} = -2 \sum_{i=1}^n (y_i - mx_i - b)1 = 0$$



$$\begin{aligned} \sum_{i=1}^n x_i^2 m + \sum_{i=1}^n x_i b &= \sum_{i=1}^n y_i x_i \\ \sum_{i=1}^n x_i m + \sum_{i=1}^n 1b &= \sum_{i=1}^n y_i \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i x_i \\ \sum_{i=1}^n y_i \end{bmatrix}$$

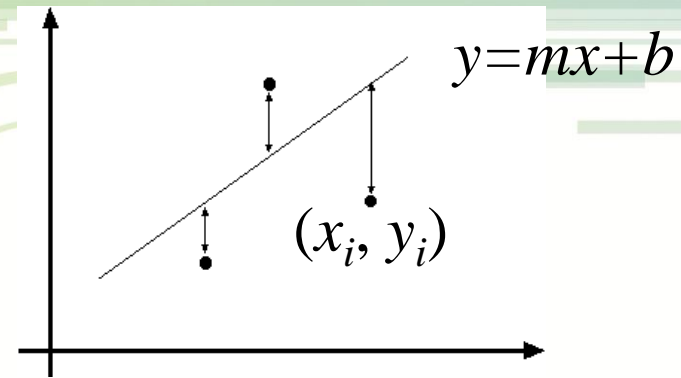


Least squares for lines

$$\begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n y_i x_i \\ \sum_{i=1}^n y_i \end{bmatrix}$$

$$\begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n 1 \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^n y_i x_i \\ \sum_{i=1}^n y_i \end{bmatrix}$$

$$\det \left(\begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n 1 \end{bmatrix} \right) = \sum_{i=1}^n x_i^2 \sum_{i=1}^n 1 - \left(\sum_{i=1}^n x_i \right)^2 \neq 0$$



$$n \sum_{i=1}^n x_i^2 - n^2 \left(\frac{1}{n} \sum_{i=1}^n x_i \right)^2 = n \sum_{i=1}^n x_i^2 - n^2 \bar{x}^2 = n^2 \left(\frac{1}{n} \sum_{i=1}^n x_i^2 - \bar{x}^2 \right)$$

$$n^2 \left(\frac{1}{n} \sum_{i=1}^n x_i^2 - \bar{x}^2 \right) = n^2 (\overline{x^2} - \bar{x}^2)$$

The solution exists unless all x_i are equal!

Linear least squares - reminder

Generally, input data can be ,multidimensional' (of dimension d) and function can be any linear combination of input data

Model: $y = f(x_1, x_2, \dots, x_d) = p_1 x_1 + p_2 x_2 + \dots + p_d x_d$

Or, function (model) can be polynomial (of order r)

Model: $y = f(x) = p_0 + p_1 x^1 + p_2 x^2 + \dots + p_r x^r$

Or, could be multidimensional polynomial function

But, as long as function (model) is *linear in parameters*, we are dealing with *linear least squares*

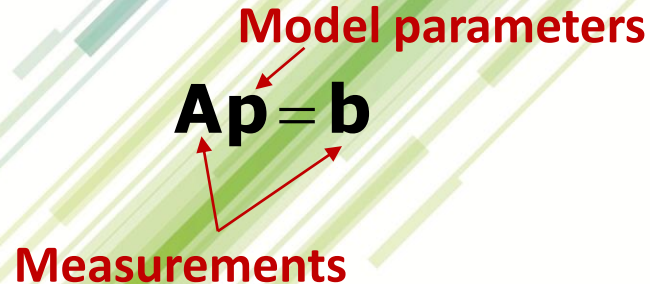
LS Refresher, matrix notation

Generally, we want to find solution of the overdetermined system, more measurements than unknowns, but measurements are corrupted by noise.

$$\mathbf{Ap} = \mathbf{b}$$

Model parameters

Measurements



Exact solution does not exist, thus

$$\mathbf{e} = \mathbf{Ap} - \mathbf{b}$$

We want to find solution that minimizes (mean) square error:

$$E(\mathbf{p}) = \|\mathbf{e}\|^2 = \mathbf{e}^T \mathbf{e} = (\mathbf{Ap} - \mathbf{b})^T (\mathbf{Ap} - \mathbf{b}) = \|\mathbf{Ap} - \mathbf{b}\|^2$$

LS Refresher, matrix notation

$$\begin{aligned} E(\mathbf{p}) &= \|\mathbf{Ap} - \mathbf{b}\|^2 = (\mathbf{Ap} - \mathbf{b})^T (\mathbf{Ap} - \mathbf{b}) = \mathbf{b}^T \mathbf{b} - 2(\mathbf{Ap})^T \mathbf{b} + (\mathbf{Ap})^T (\mathbf{Ap}) \\ &= \mathbf{b}^T \mathbf{b} - 2(\mathbf{Ap})^T \mathbf{b} + \mathbf{p}^T (\mathbf{A}^T \mathbf{A}) \mathbf{p} \end{aligned}$$

$$\frac{dE}{d\mathbf{p}} = 2\mathbf{A}^T \mathbf{Ap} - 2\mathbf{A}^T \mathbf{b} = \mathbf{0}$$

$$(\mathbf{A}^T \mathbf{A}) \mathbf{p} = \mathbf{A}^T \mathbf{b}$$

$$\mathbf{p} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \quad \text{Pseudo inverse}$$

Matlab: `p = A \ b;`

Back to least squares for lines

$$e_1 = x_1 m + 1b - y_1$$

$$e_2 = x_2 m + 1b - y_2$$

.....

$$e_i = x_i m + 1b - y_i$$

.....

$$e_n = x_n m + 1b - y_n$$



$$\begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_i \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_i & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}$$



$$\mathbf{e} = \mathbf{A}\mathbf{p} - \mathbf{b}$$

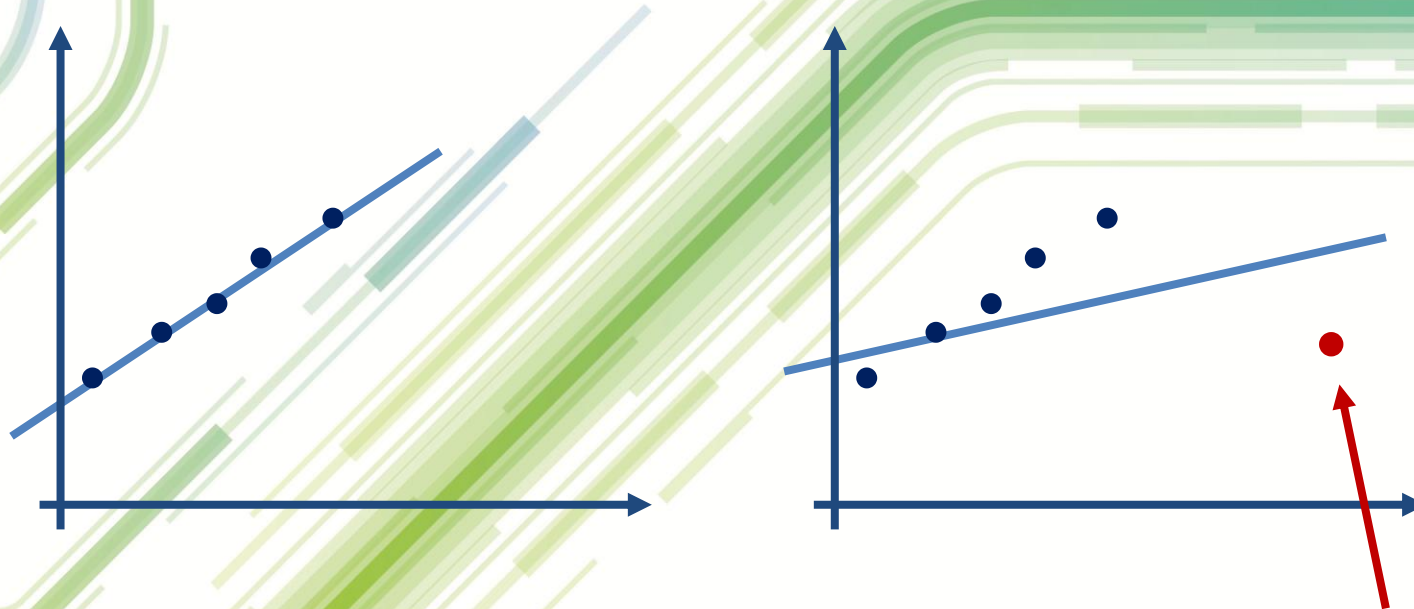
$$E(m, b) = \sum_{i=1}^n e_i^2 = \mathbf{e}^T \mathbf{e} = \|\mathbf{e}\|^2 = \left\| \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{A}\mathbf{p} - \mathbf{b}\|^2$$

$$\mathbf{p} = \begin{bmatrix} m \\ b \end{bmatrix} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$$

Things we will skip...

- Model $y=mx + b$ fails for steep lines
- The distances from line in this case are not perpendicular
- Least squares method is usually employed via SVD (Singular value decomposition)

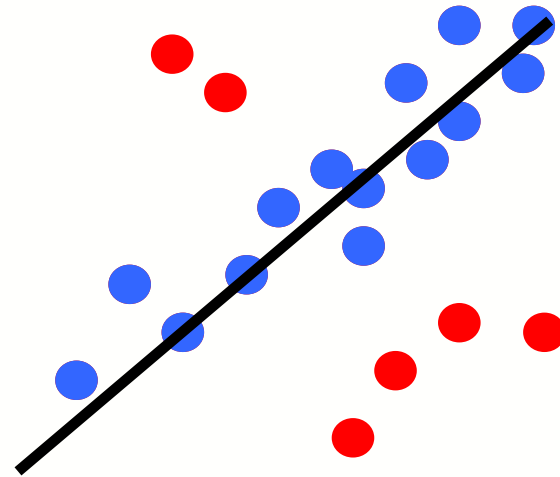
Least squares: robustness to noise



- Least squares fit is very sensitive to gross errors – outliers: LS is *not robust to noise*
- Meaning: single LARGE outlier can heavily influence result
- Solution: detect and separate outliers from inliers automatically

Solution: RANSAC

- **Random Sample Consensus**
 - Fundamentally different from LS
 - It takes the smallest possible sample size to estimate the parameters of the model, rather than as large as possible
 - Fischler & Bolles '81.
- **Illustration:**
 - Blue dots clearly support
 - the line model,
 - while the red ones do not



Solution: RANSAC

repeat {

(1) Sample (randomly) just enough points required to fit the model

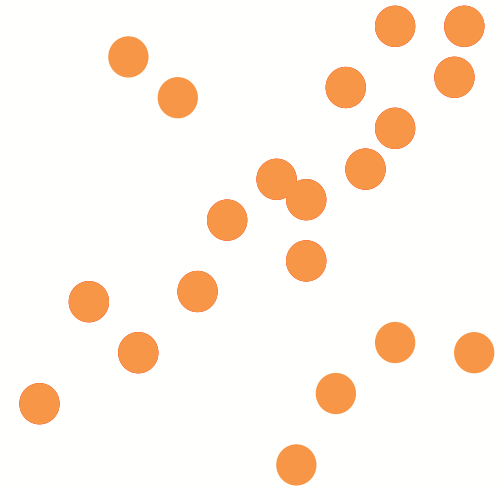
(2) Solve for model parameters using the sampled points only

(3) Evaluate the goodness of fit, find the largest set of points that support the model (inliers').

That is, search for consensus.

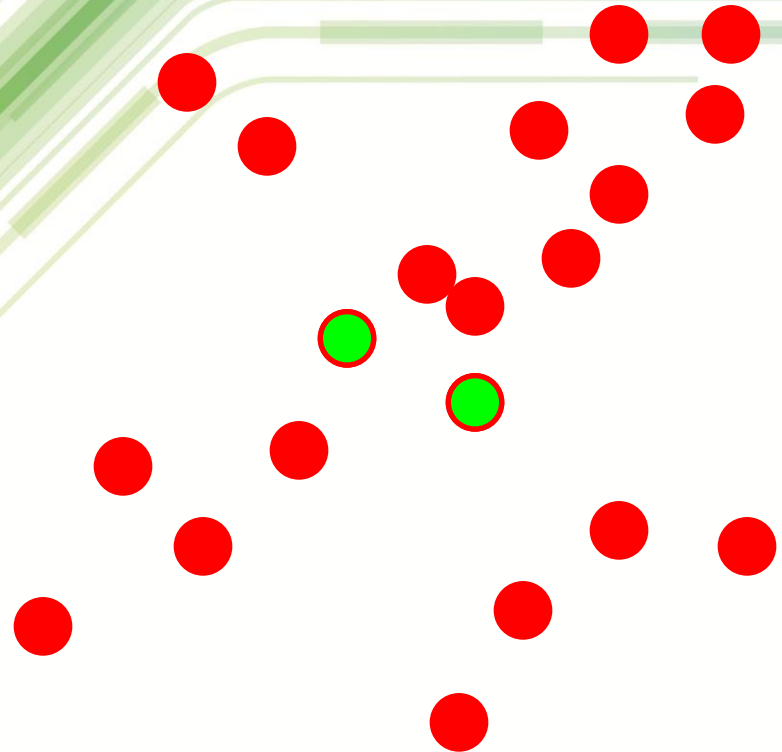
}

until the best model is found with sufficient confidence



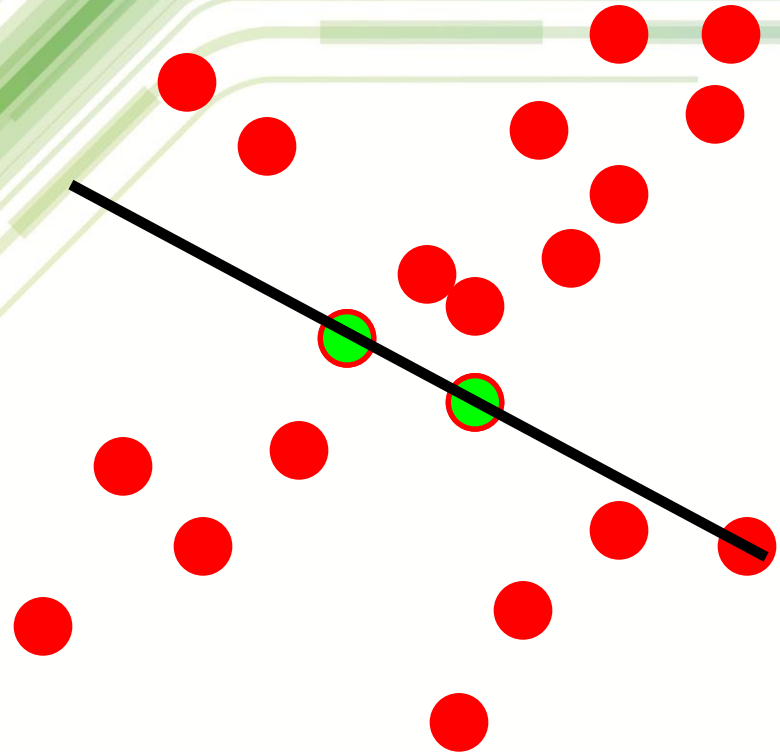
RANSAC- Line fitting example

1. Sample (randomly) the number of points required to fit the model (2 in the case of line)



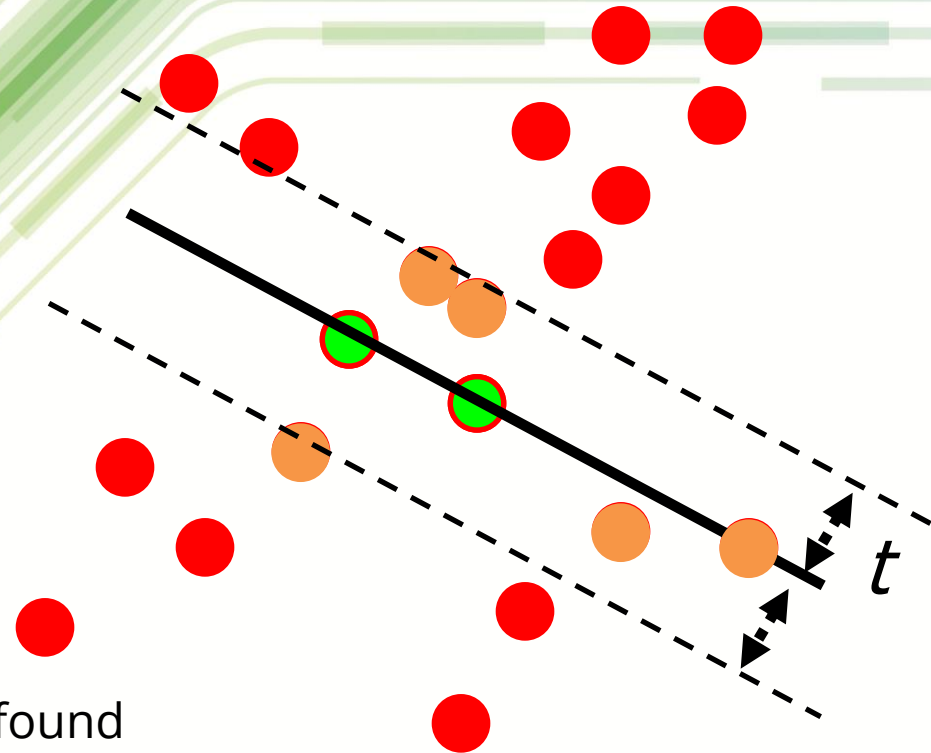
RANSAC – Line fitting example

1. Sample (randomly) the number of points required to fit the model (2 in the case of line)
2. Solve for model parameters



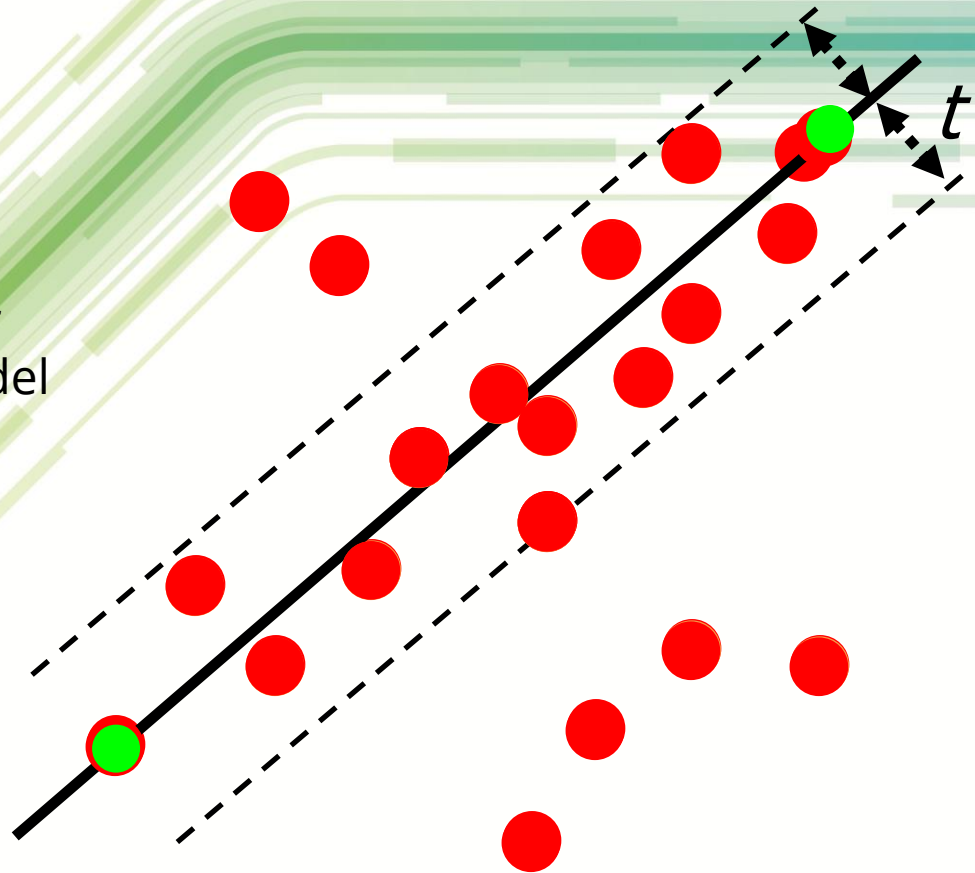
RANSAC - Line fitting example

1. Sample (randomly) the number of points required to fit the model (2 in the case of line)
2. Solve for model parameters
3. Get consensus (5 points in the depicted case)
4. Repeat 1-3 until the best model is found with high confidence



RANSAC - Line fitting example

1. Sample (randomly) the number of points required to fit the model (2 in the case of line)
2. Solve for model parameters
3. Get consensus (12 points now!)
4. Accept the model!



RANSAC – choosing the parameters

- (Initial) number of data points (sample size): **s**
Typically minimum number needed to fit the model
In case of line: 2
- Distance threshold **t**
Choose t such that probability for inlier is large (e.g. 0.95)
Zero-mean Gaussian noise with std σ : $t^2 = 3.84\sigma^2$
- Number of samplings **N** (,iterations')
Choose N such that, with probability p , at least one random sample is free from outliers (e.g. $p = 0.99$)
(for a given outlier ratio: e)

RANSAC – choosing the parameters

$p_i = (1-e)^s$: probability that the sample does not contain outliers

$p_o = 1 - p_i = 1 - (1-e)^s$: probability that sample contains outliers

$p_o^N = (1 - (1-e)^s)^N$: probability that in N trials we never get a sample without outliers

$p = 1 - (1 - (1-e)^s)^N$: probability that in N trials we select at least once a sample without outliers

$$\left(1 - (1 - e)^s\right)^N = 1 - p$$

$$N = \log(1 - p) / \log(1 - (1 - e)^s)$$

Example: $e = 0.5$ (50% of outliers), $s = 2$, $p = 0.99$ and $p = 0.999$

$$N = \log(1 - 0.99) / \log(1 - 0.5^2) = 16$$

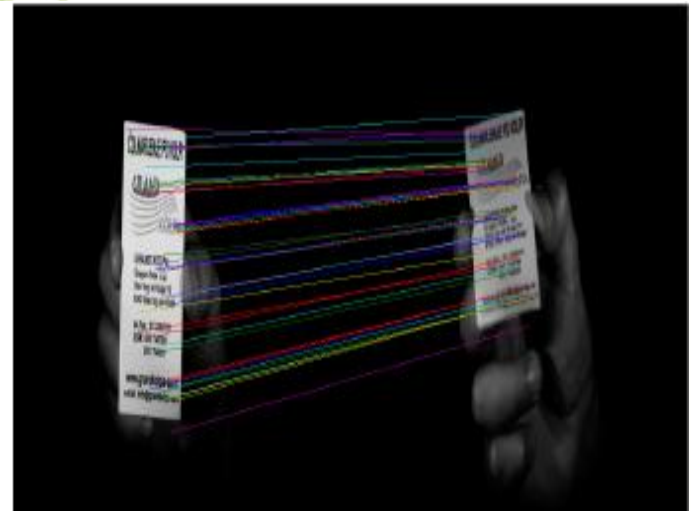
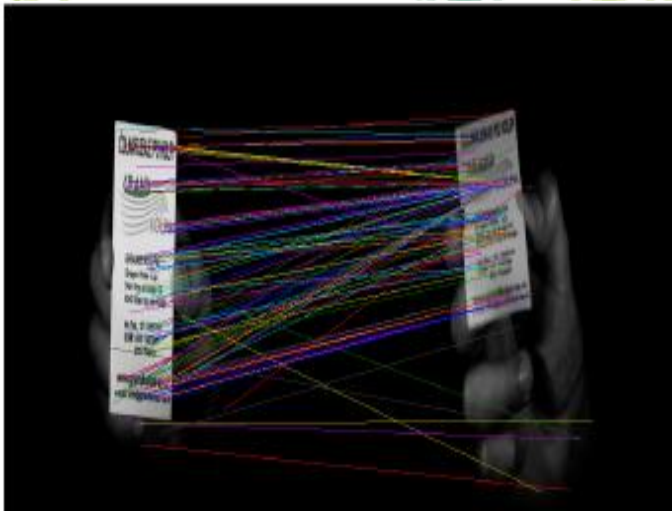
$$N = \log(1 - 0.999) / \log(1 - 0.5^2) = 24$$

RANSAC – an illustrative example

Homography estimation, i.e., plane to plane mapping

First, we detect large number of points (e.g. corners) in both images

Then we estimate transformation parameters



On the left, obviously, there are many false matches present

On the right, the outliers were filtered out and plane to plane transformation (homography) has been estimated

RANSAC – conclusion

- Good
 - Robust to outliers
- Bad
 - Computational time grows quickly with fraction of outliers and number of parameters
- Typical applications
 - Computing a homography (e.g., image stitching)
 - Estimating fundamental matrix (relating two views)
 - All kinds of registration tasks
- In computer vision
 - We can get large amount of data, but not high quality data
 - Therefore RANSAC was quickly adopted!

The background features a series of overlapping, curved lines in various shades of green and blue, creating a sense of depth and movement. The lines are of varying thicknesses and some have a slight gradient, giving them a three-dimensional appearance. They are arranged in a way that suggests a complex, interconnected network or a series of paths.

Questions?