Image processing

Rotem Shalev-Arkushin
rotems7@mail.tau.ac.il

December 2025

Image features

Image features or feature points are used for:

- Image alignment
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- And more…

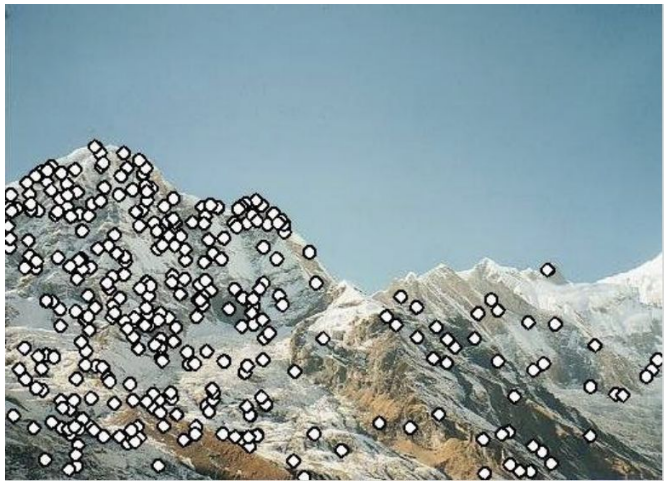
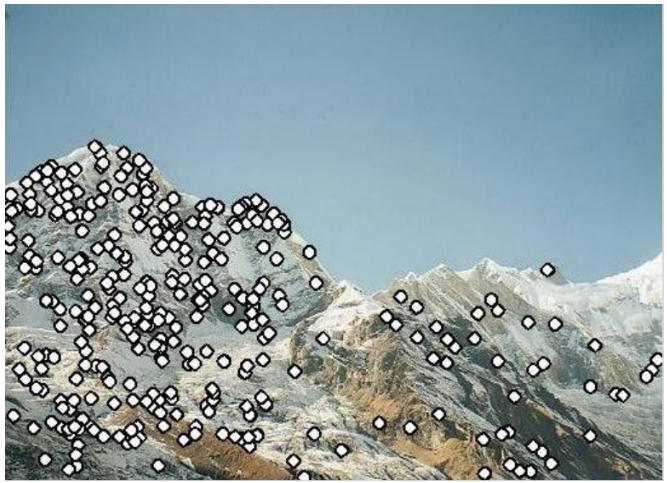


Image features

Image features or feature points are used for:

- **Image alignment**
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- And more…



Panorama exercise

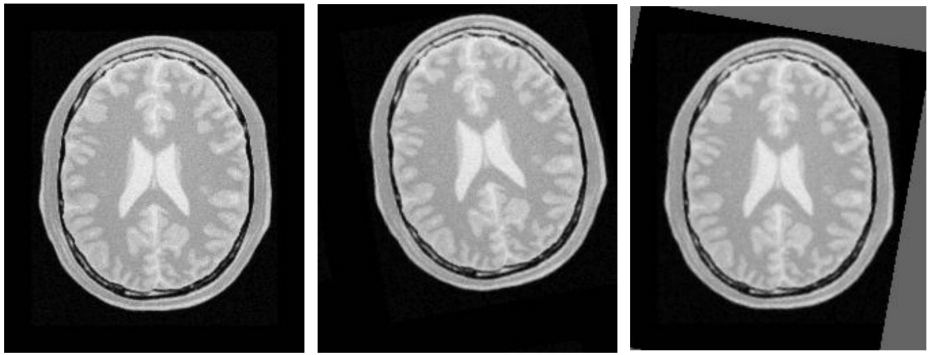
What do we need to know to construct a panorama?

- Image registration and alignment
  - Find unique features in each image (use feature detectors and descriptors)
  - Find matching features between images
  - Align images

- Stitch images together
  - Place aligned images on a shared canvas
  - Blend images together seamlessly

Image Registration

Aligning two or more images of the same scene or object, so that **corresponding points** match. Typically, one image is considered a **reference**, and the other(s) are **transformed** to achieve spatial correspondence.



Images: ITK Registration Guide

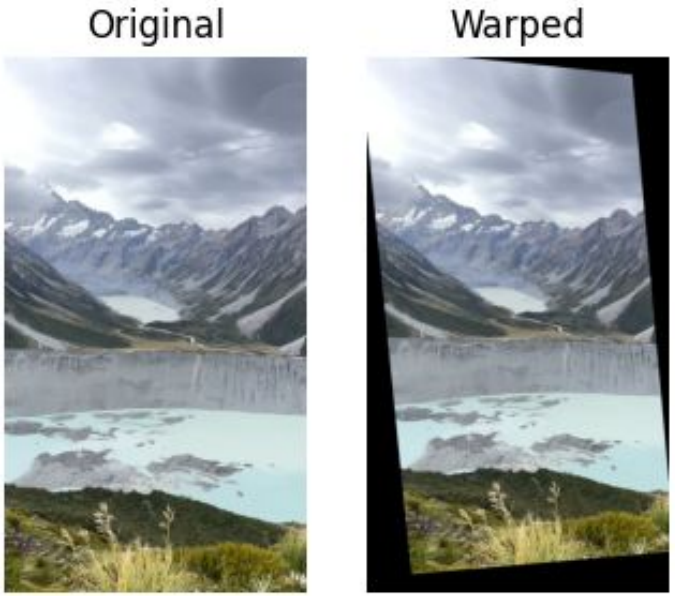
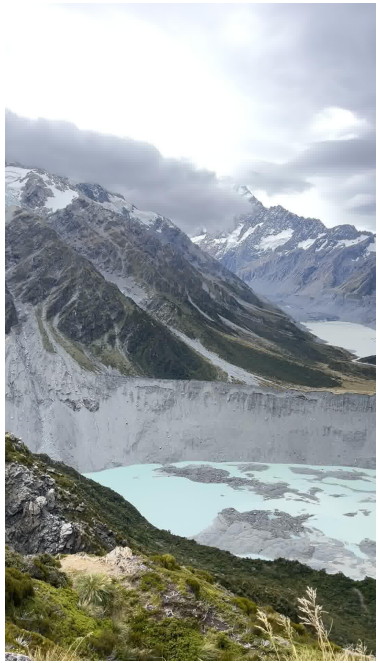


Image features

We want to find image features in each image, that will help us align them.
What are good features?



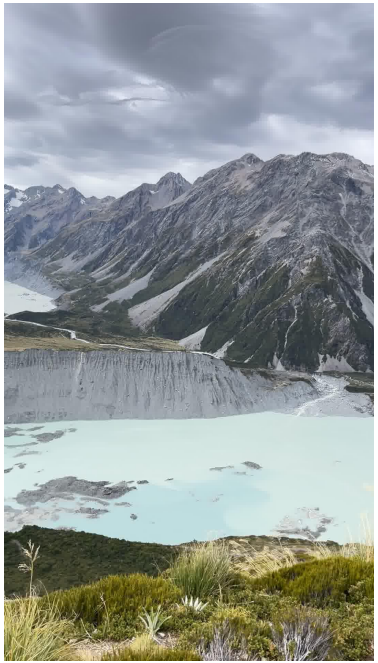


Image features

What are good image features?

- Identifiable – not from smooth areas!

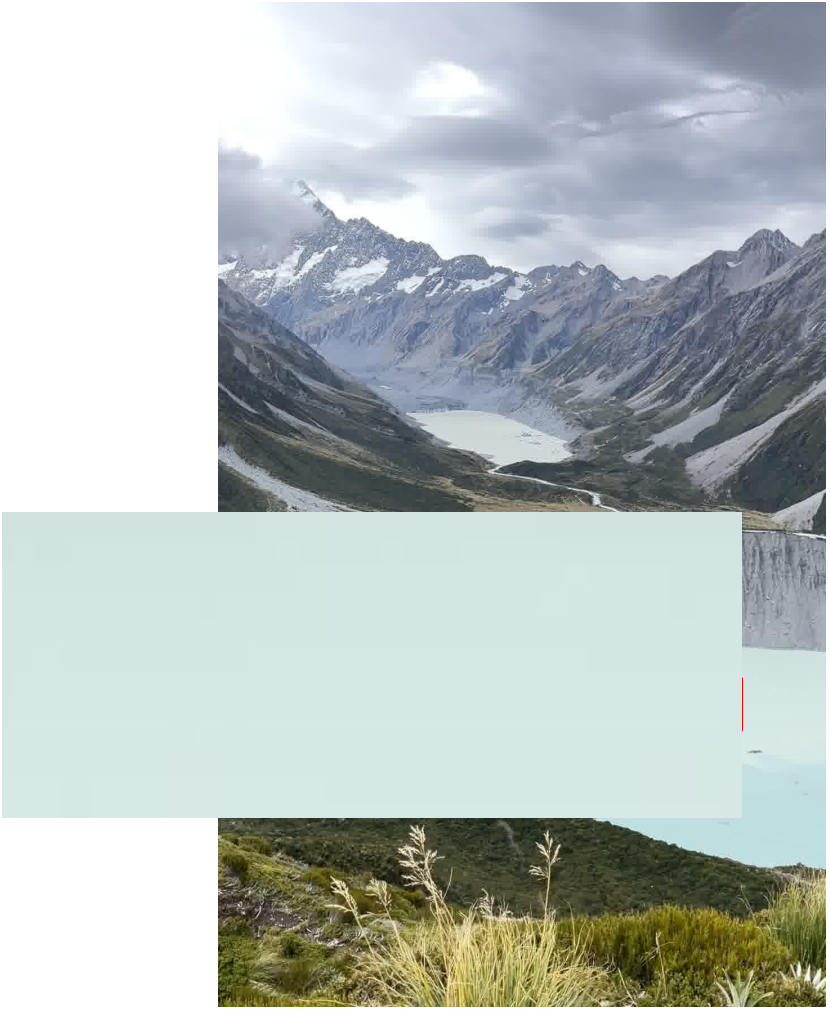


Image features

What are good image features?

- Identifiable – not from smooth areas!
- Unique, not repetitive

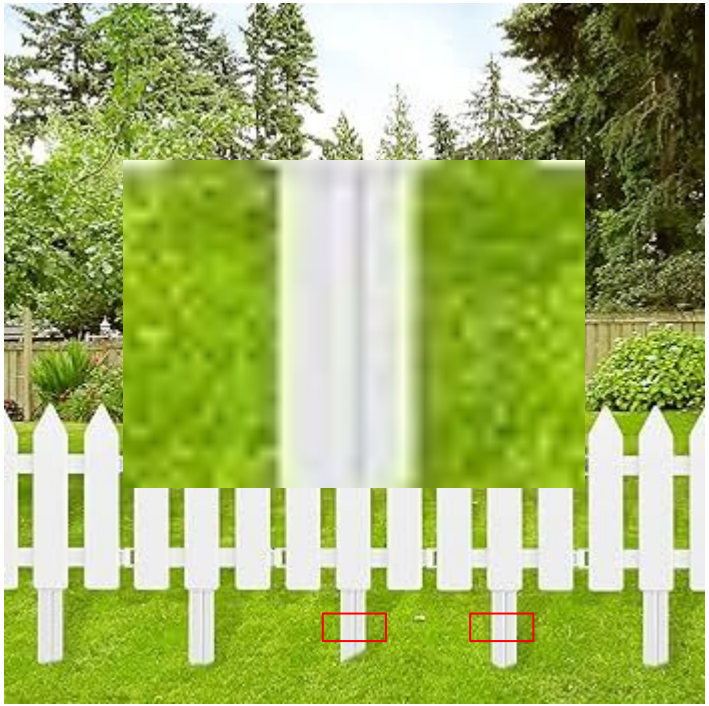


Image features

What are good image features?

- Identifiable – not from smooth areas!
- Unique, not repetitive
- Can be located repeatedly, and exist in both images

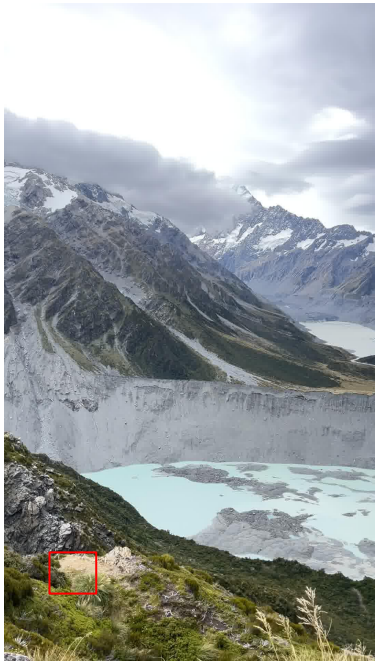


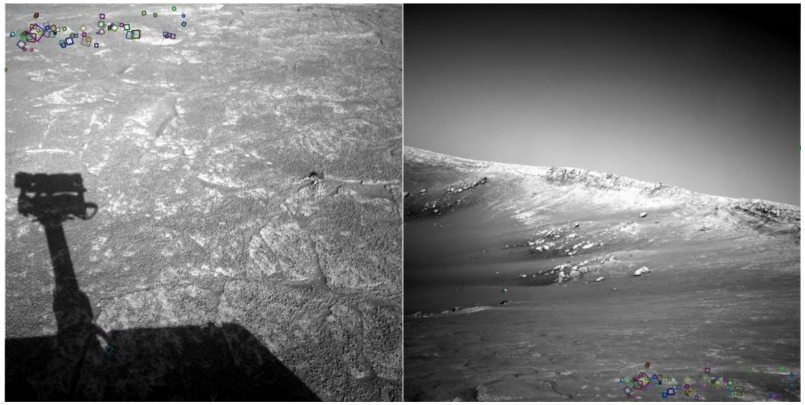
Image features

What are good image features?

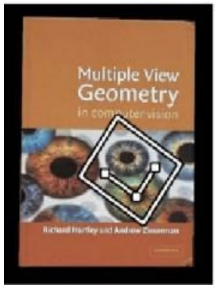
- Identifiable – not from smooth areas!
- Unique, not repetitive
- Can be located repeatedly, and exist in both images

More desired properties:

- Easy to extract
- Invariant to different conditions:
  - Geometric – translation, rotation, scaling, etc.
  - Photometric – illumination, exposure..



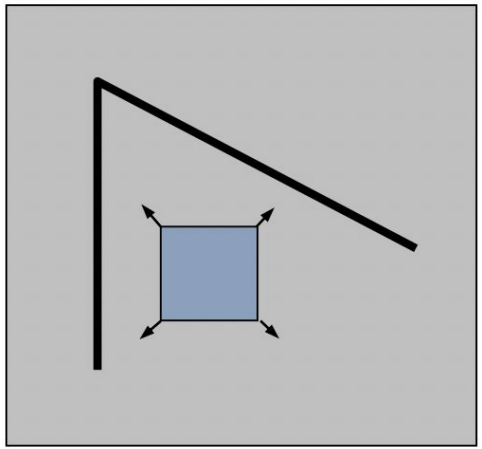
NASA Mars Rover images
with SIFT feature matches



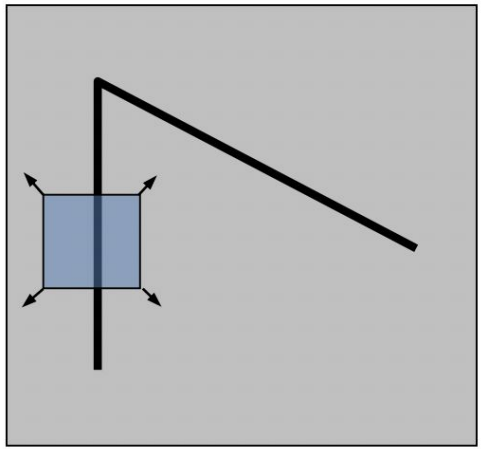
by Diva Sian

by scgbt

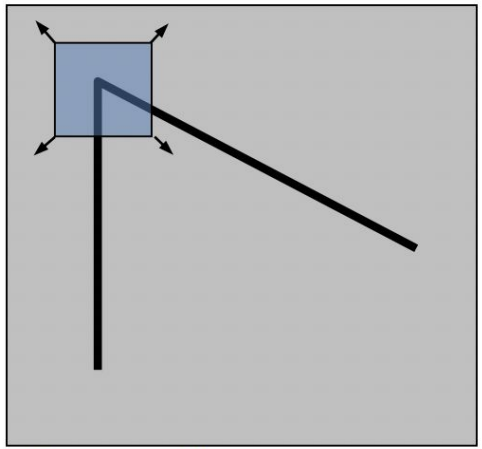
Corners



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

Credit: S. Seitz, D. Frolova, D. Simakov

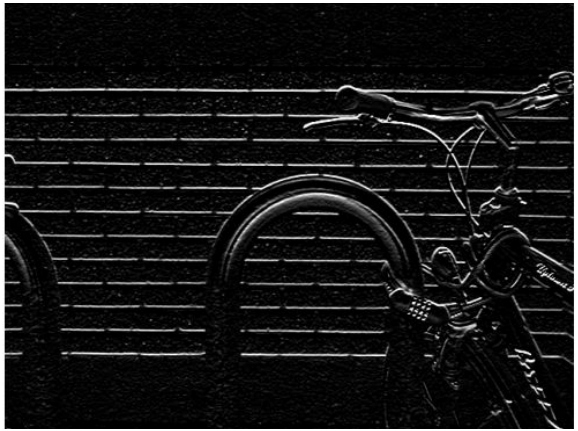
Image features

- Feature point detection
  - Harris corner detector

- Feature descriptors
  - MOPS - Multi-Scale Oriented Patches
  - SIFT - Scale-invariant feature transform

Corners

dy = Horizontal edges



dx = Vertical edges

~dx & dy = Corners

Harris corner detector

“A Combined Corner and Edge Detector” (Chris Harris & Mike Stephens, 1988)

Idea:

Look at a small window around image derivatives of each pixel and shift this window in x and y directions.

- If shifting changes almost nothing → Flat region, not interesting.
- If shifting changes a lot but only in one direction → Edge.
- If shifting changes a lot in all directions → Corner!

Use a score (the Harris response) to keep strong corners and discard edges.

Harris corner detector

More formally – use autocorrelation or SSD (sum of squared differences):

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

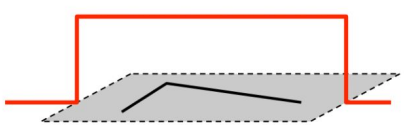
Error function

Window function

Shifted intensity

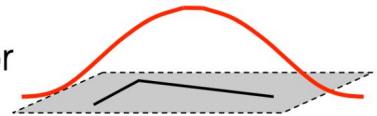
Intensity at pixel (x,y)

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Source: R. Szeliski

Harris corner detector

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$$

Problem: very slow to compute naively for each pixel and each offset.

Solution: use Taylor series expansion.

A function *f* can be represented by an infinite series of its derivatives at a single point *a*:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n.$$

Harris corner detector

A Taylor approximation of $E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I(x,y)}{\partial x}u + \frac{\partial I(x,y)}{\partial y}v = I(x,y) + I_x u + I_y v$$

$$E(u,v) \approx \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

Harris corner detector

A Taylor approximation of $E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I(x,y)}{\partial x}u + \frac{\partial I(x,y)}{\partial y}v = I(x,y) + I_x u + I_y v$$

$$E(u,v) \approx \sum_{x,y} w(x,y)[I_x u + I_y v]^2$$

$$= \sum_{x,y} w(x,y) \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$= \begin{bmatrix} u & v \end{bmatrix} \left[\sum_{x,y} w(x,y) \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} \right] \begin{bmatrix} u \\ v \end{bmatrix}$$

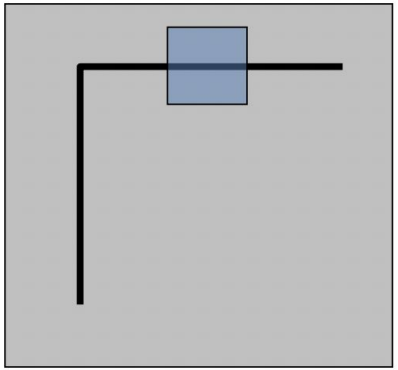
$$= \begin{bmatrix} u & v \end{bmatrix} \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

Harris corner detector

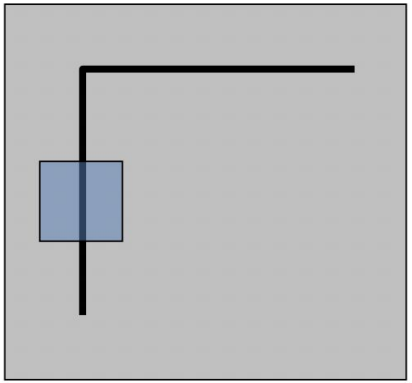
$$[u \quad v] \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$E(u,v) \approx [u \quad v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$



Horizontal edge: $I_x = 0$



Vertical edge: $I_y = 0$

Eigenvalue / Eigenvector reminder

- The **eigenvectors** of a matrix A are the vectors x that satisfy: $Ax = \lambda x$
- λ is the **eigenvalue** corresponding to x.
- The **eigenvalues** are found by solving: $det(A - \lambda I) = 0$

Eigenvalue / Eigenvector reminder

- The **eigenvectors** of a matrix A are the vectors x that satisfy: $Ax = \lambda x$
- λ is the **eigenvalue** corresponding to x.
- The **eigenvalues** are found by solving: $det(A - \lambda I) = 0$

In our case: let λ_1, λ_2 be the eigenvalues of M. For each, we can find the corresponding eigenvectors xi = [ui vi] by: $(M - \lambda_i I)x_i = 0$

Eigenvalue / Eigenvector reminder

- The **eigenvectors** of a matrix A are the vectors x that satisfy: $Ax = \lambda x$
- λ is the **eigenvalue** corresponding to x.
- The **eigenvalues** are found by solving: $det(A - \lambda I) = 0$

In our case: let λ_1, λ_2 be the eigenvalues of M. For each, we can find the corresponding eigenvectors xi = [ui vi] by: $(M - \lambda_i I)x_i = 0$

The smallest and largest eigenvalues and eigenvectors define **shift directions** with the smallest and largest changes in error.
Eigenvectors = direction, **eigenvalues** = amount of increase in that direction.

Back to Harris corner detector

Recall our score function E(u,v).

$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

We want to find areas with change in both directions => both eigenvalues are large.

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

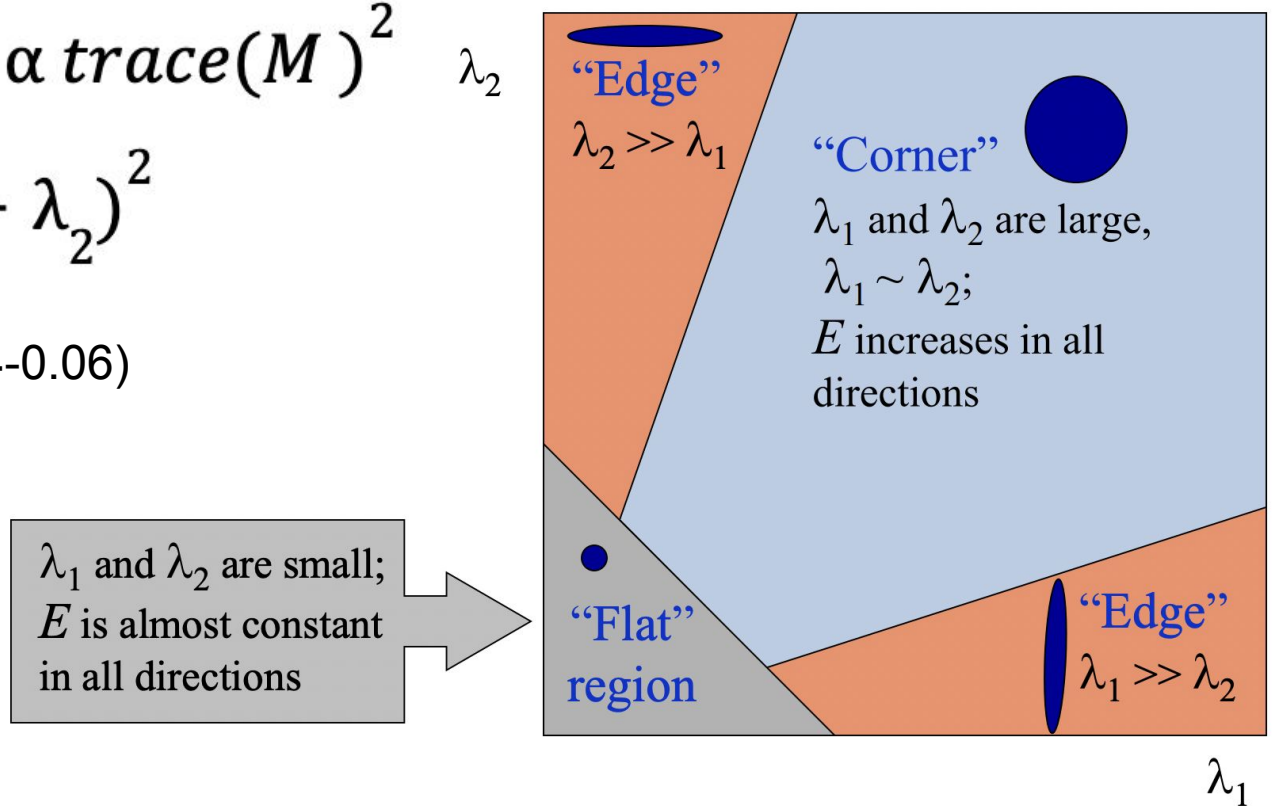
Harris corner detector

$$R = det(M) - \alpha\, trace(M)^2$$

$$= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

- α is a constant (0.04-0.06)

Corner = R > th



Corner detector response variations

- Noble's variation (1989): $R = \frac{det(M)}{trace(M)+\epsilon} = \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2+\epsilon}$

- Shi-Tomashi corner detector (1994): $R = min(\lambda_1, \lambda_2)$

Harris corner detector

Complete algorithm:

1. Compute corner response R by:
   a. calculating the image derivatives and
   b. finding the eigenvalues.

2. Filter points that are below a threshold.

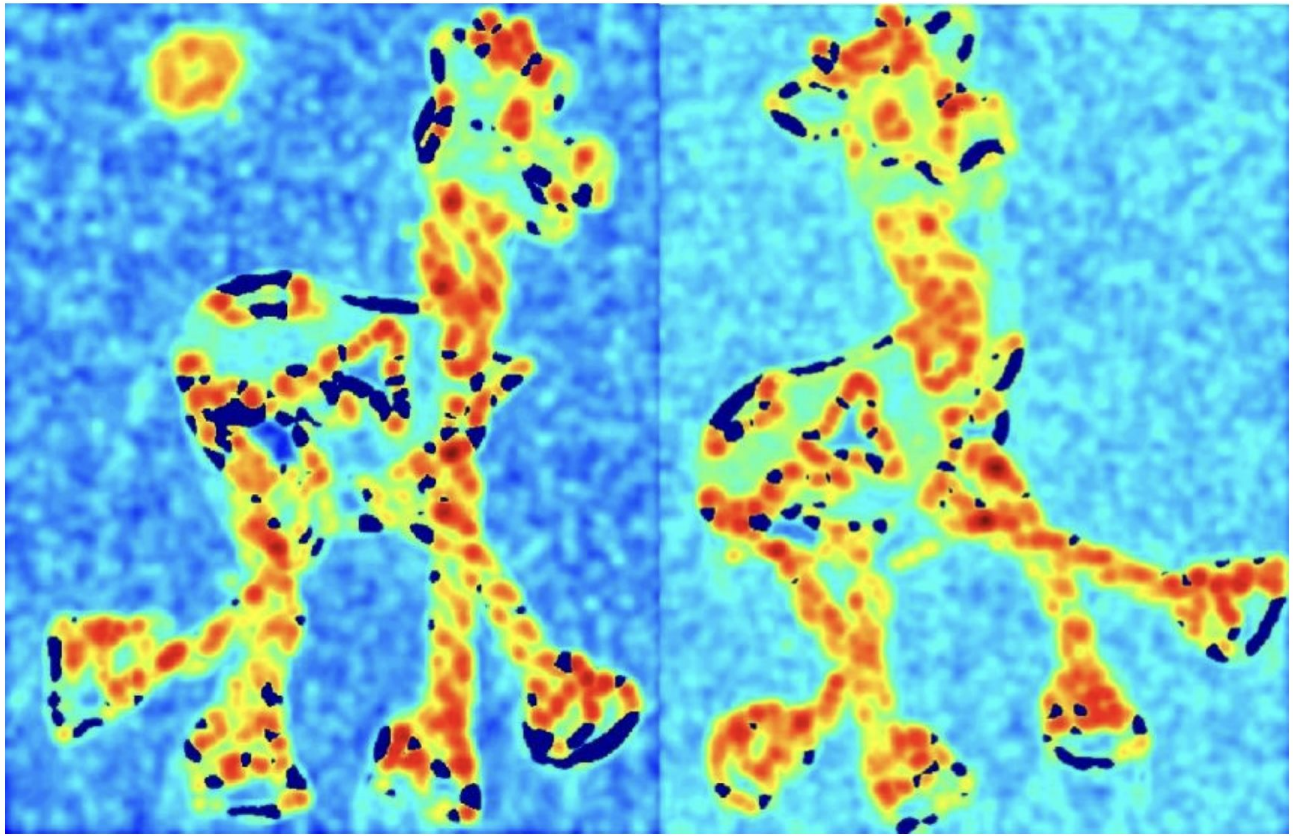
3. Non-maximum suppression: take only local maxima from these points.

Harris corner detector

Example:
Given these 2 images, detect corners.

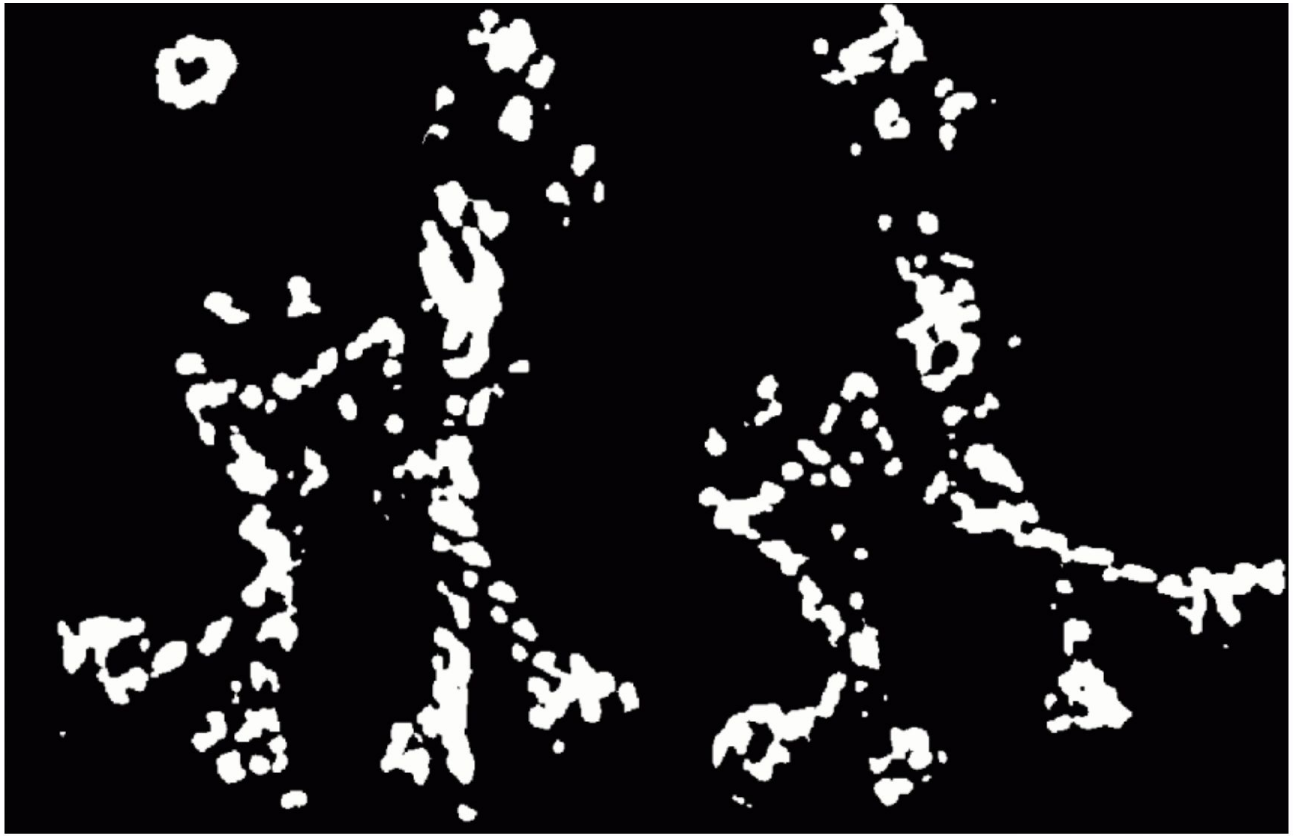
Harris corner detector

Corner response R:
red=high, blue=low



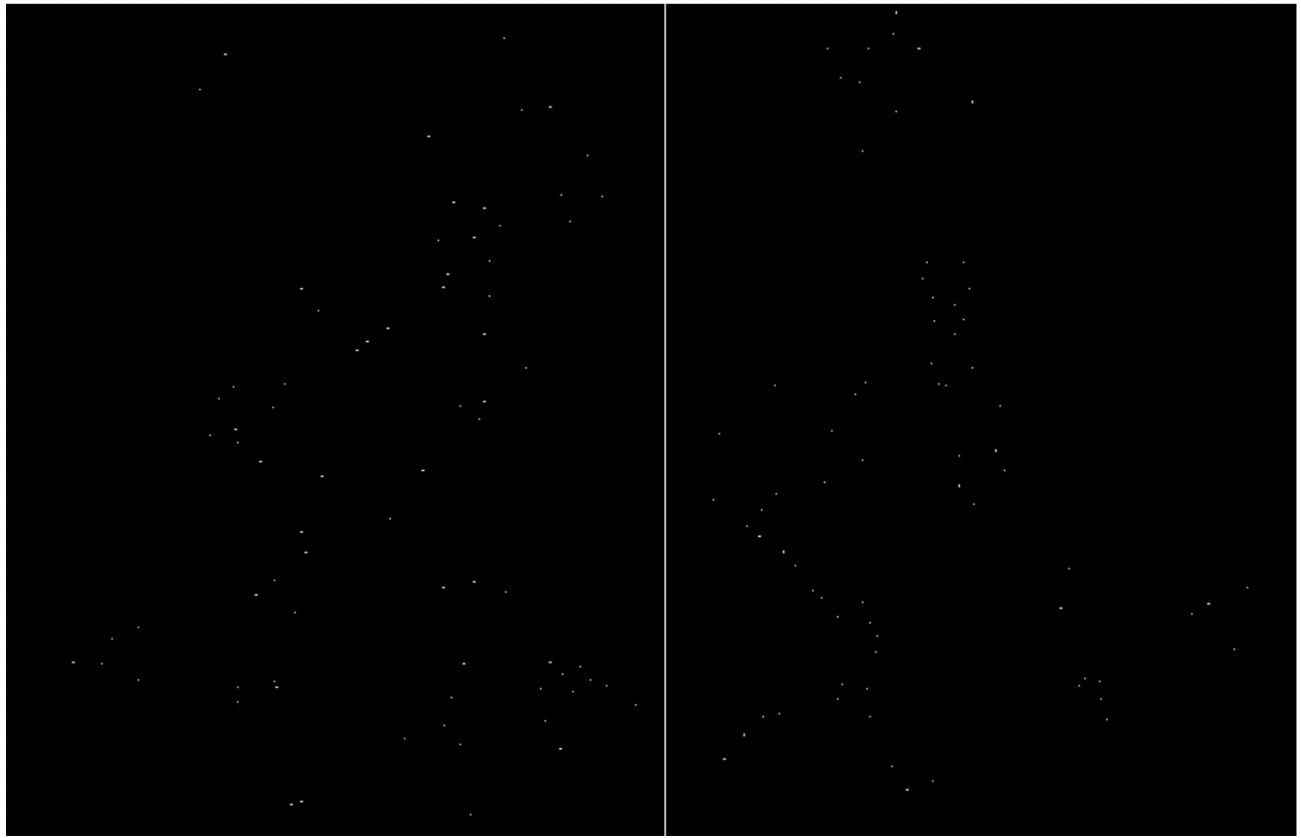
Harris corner detector

Take only points where R > th



Harris corner detector

Take local maxima.



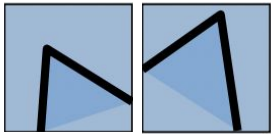
Harris corner detector

Result:
Corners (Harris features) are detected!
Marked in red.

Harris corner detector

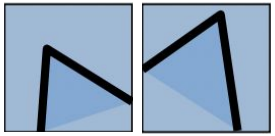
Properties:

- Invariance to rotation: Eigenvalues remain the same.



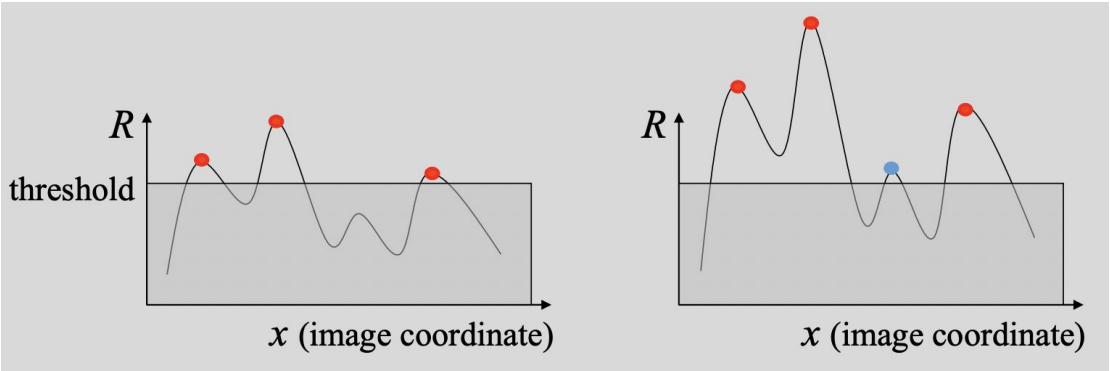
Harris corner detector

Properties:



- Invariance to rotation: Eigenvalues remain the same.

- Partial invariance to change in intensity:
  - Only derivatives are used => invariance to intensity shift I → I + b
  - Not invariant to Intensity scale: I → a*I



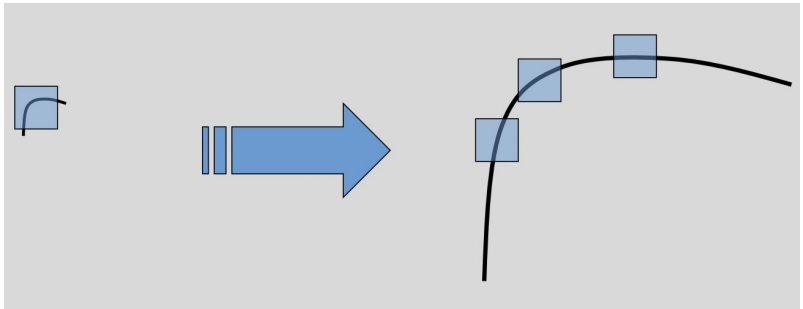
Harris corner detector

Properties:

- Invariance to rotation: Eigenvalues remain the same.

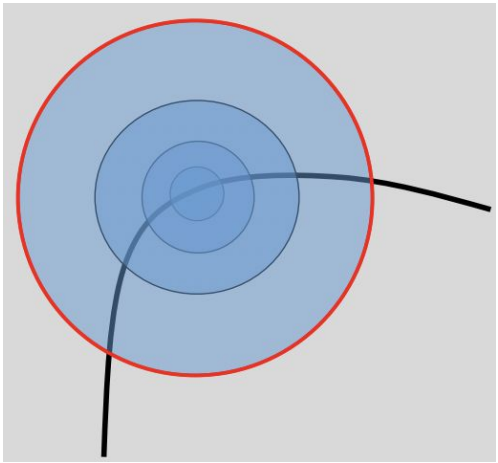
- Partial invariance to change in intensity:
  - Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
  - Not invariant to Intensity scale: $I \rightarrow a*I$

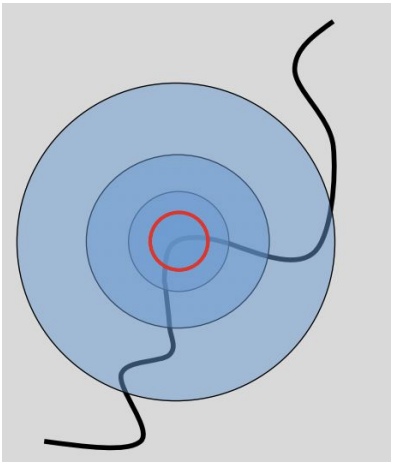
- **Not invariant to geometric scale!**
  On the left scale – window contains the whole **corner**. On the right – all windows will be classified as **edges**.



Scale-Invariant Detection

Idea: look at different windows (e.g. circles) around points, and find local maxima in both position and scale.





Scale-Invariant Detection

Idea: look at different windows (e.g. circles) around points, and find local maxima in both position and scale.

Practically: use **gaussian pyramids** for different scales.

- Compute the feature response at each scale with a fixed window size.
- if local maximum and cross-scale → save scale and feature location.

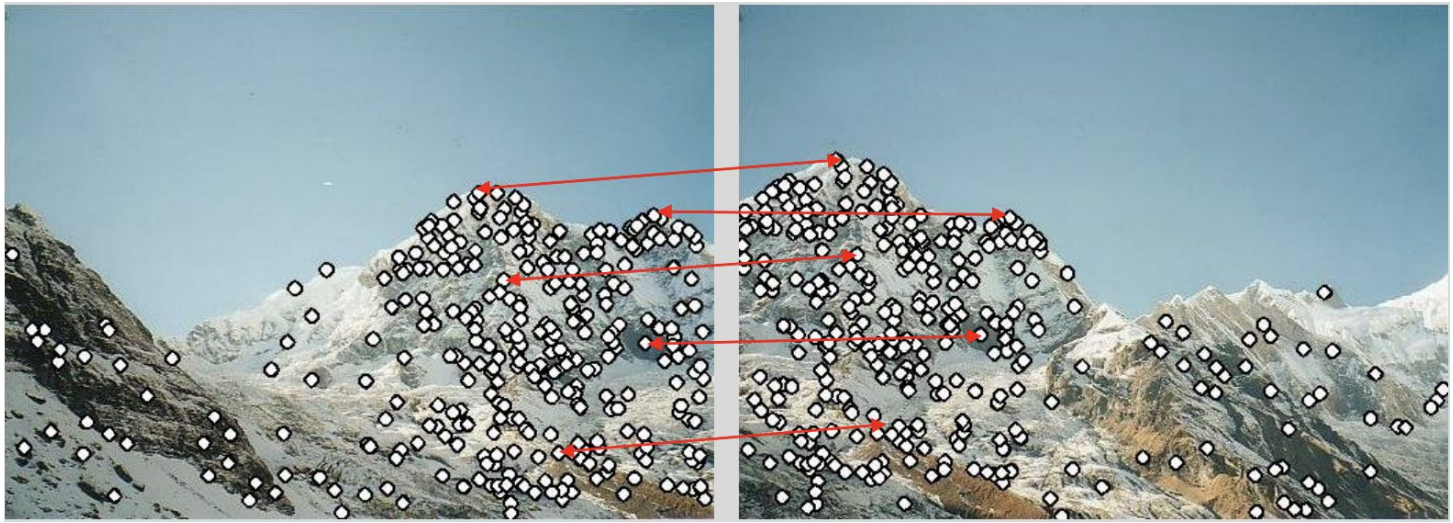
Image features

- ✔ Feature point detection
  - Harris corner detector

- Feature descriptors
  - MOPS - Multi-Scale Oriented Patches
  - SIFT - Scale-invariant feature transform

Feature matching

How can we match between features from two different images?



Feature matching

1. **Find features** in each image, e.g. using Harris corner detector.
2. Define a **feature descriptor.**
3. Define a **similarity measurement** to compare between feature descriptors.
4. Find the feature in img2 that is of minimal distance from each feature in img1.

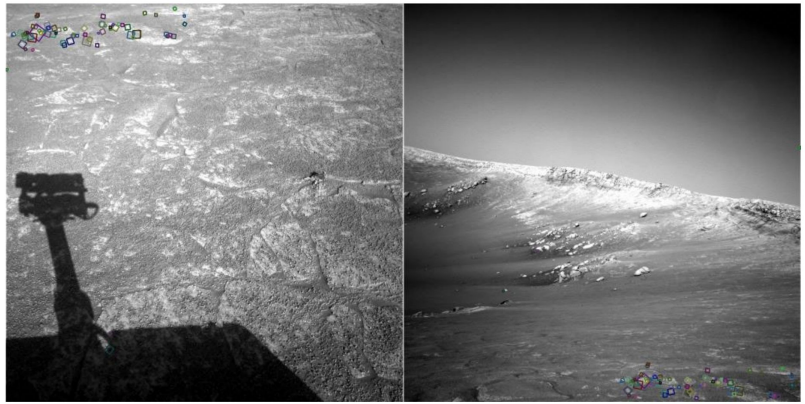
Feature descriptors

Required properties:

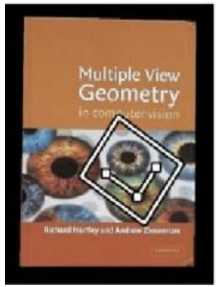
- Unique and distinguishable

Invariant to different conditions:

- Geometric – translation, rotation, scaling, etc.
- Photometric – illumination, exposure..



NASA Mars Rover images
with SIFT feature matches

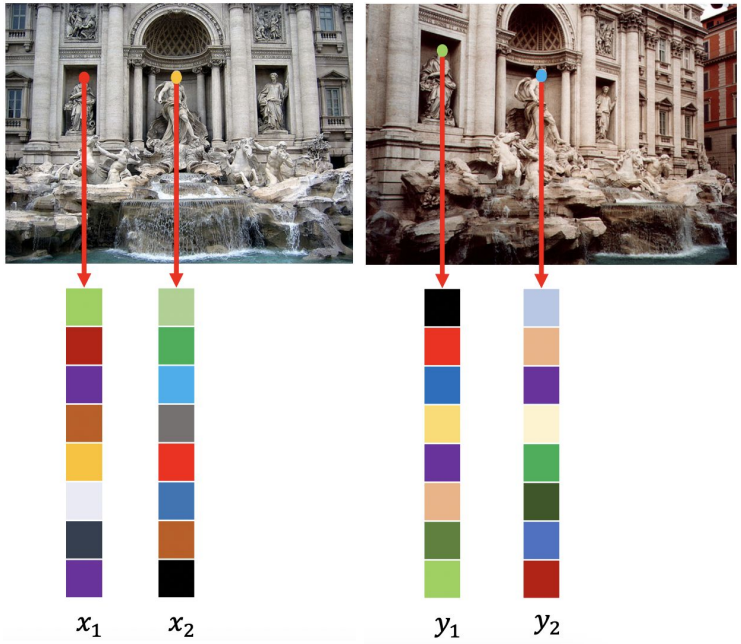


by Diva Sian

by scgbt

Feature descriptors

Simplest descriptor: vector of image pixels in a window.
Similarity: Euclidean distance, cross correlation.



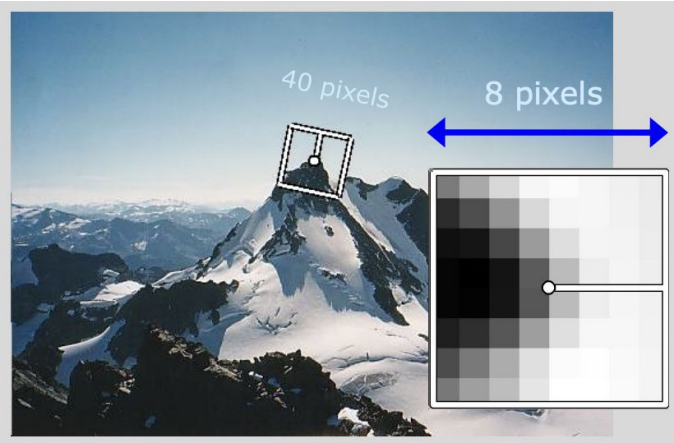
Multi-Scale Oriented Patches (MOPS)

(Multi-Image Matching using Multi-Scale Oriented Patches. Brown, Szeliski, Winder, CVPR’2005)

Idea:
Given a scale-space position feature (x, y, s) e.g. using Harris.
Descriptor vector: 8x8 oriented patch.

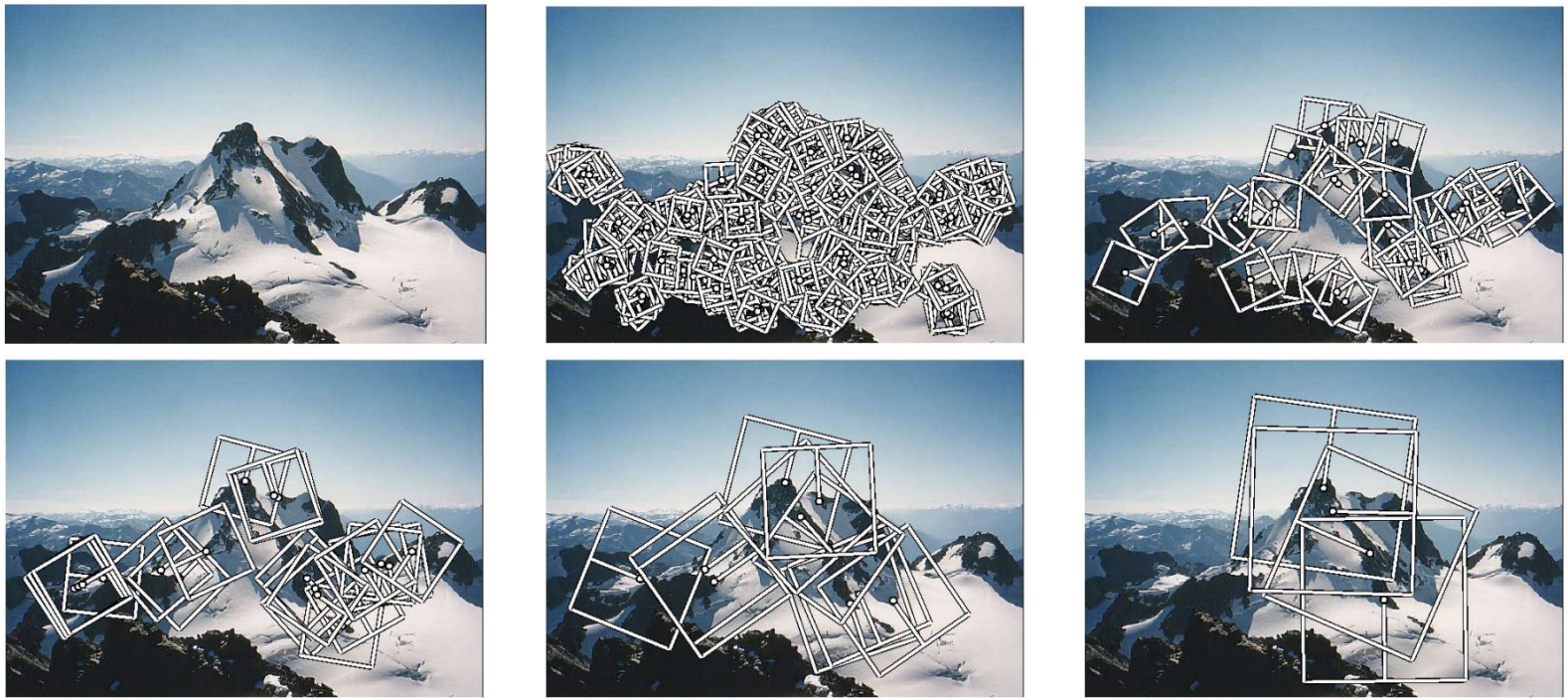
- Take a 40x40 patch, sample every 5 pixel to get a 8x8 patch.
- Normalize by subtracting mean, dividing by std.
- Orient using a blurred local gradient.

Similarity: Euclidean distance (L2) or
normalized cross-correlation



Multi-Scale Oriented Patches (MOPS)

Multi-scale: apply to different pyramid levels.



Multi-Scale Oriented Patches (MOPS)

Properties:

- Invariant to **translation** – local window around a detected keypoint
- Invariant to **rotation** – patch rotated by dominant orientation
- Limited **scale** invariance – descriptor computed at a chosen pyramid level
- Partial **illumination** invariance – patch normalization

Scale-Invariant Feature Transform (SIFT)

(Object Recognition from Local Scale-Invariant Features. David J. Lowe, ICCV 1999)

Both a detector and a descriptor.

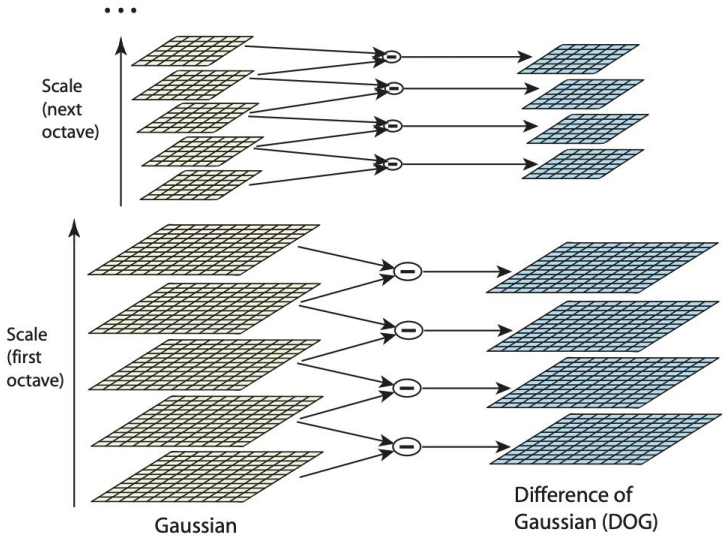
Idea:

- **Detector:** Find local extrema in space and scale
- **Descriptor:** Patch orientation histograms

Scale-Invariant Feature Transform (SIFT)

Detector

- Build a scale-space of the intensity image by progressively blurring the image.
- Compute Differences of Gaussians (DoG) between nearby scales.
- Find DoG local extrema in space and scale → candidate keypoints.
- Refine location, discard low-contrast points and edge-like responses.



Scale-Invariant Feature Transform (SIFT)

Refine location:

- Fit a quadratic (Taylor) model to the DoG response around the extremum.
- Estimate the sub-pixel/sub-scale offset; if the offset is too large, discard the point.

Discard low-contrast points:

- Use the interpolated DoG value at the refined location.
- If its magnitude is below a threshold → the point is noise-sensitive → discard.

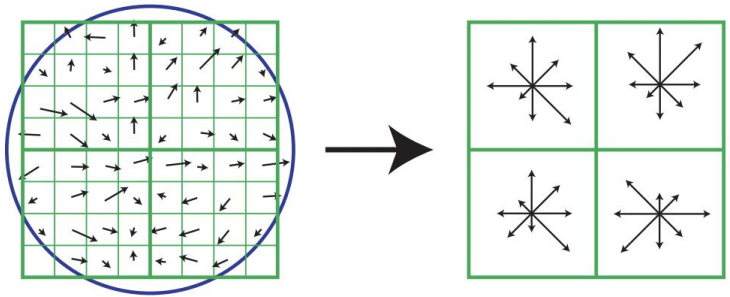
Discard edge-like responses:

- Similarly to harris - if eigenvalues are strong in one direction and weak in the other, it’s an edge response → discard.
- This keeps corner-like blobs and removes elongated edges.

Scale-Invariant Feature Transform (SIFT)

Descriptor

- Assign a dominant orientation from local gradients to each keypoint.
- Take a patch around the keypoint, and align it to its orientation. Patch scale depends on the feature scale. Could be 16x16, 32x32, etc.
- Compute gradient magnitudes and orientations.
- Pool them into **orientation histograms** over a 4×4 grid.
- Concatenate into a 128D vector (16 cells * 8 orientations = 128D descriptor)
- Normalize for illumination robustness.
- Patch similarity: Euclidean distance.



Scale-Invariant Feature Transform (SIFT)

Properties:

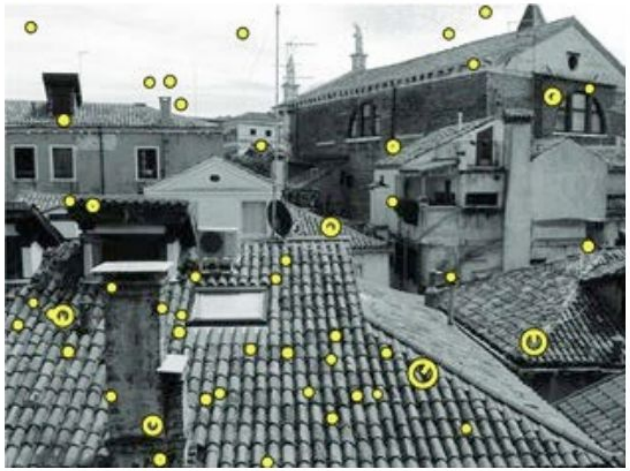
- Invariant to **translation** – **local** orientation histograms
- Invariant to **rotation** – patch rotated by dominant orientation
- Invariant to **scale** – search for maxima in multiple scales
- Invariant to **contrast** – orientation histograms + normalized descriptor
- Still used today when robustness is important (Robotics & SLAM (robust fallback), Cultural heritage, forensics, medical image registration..)

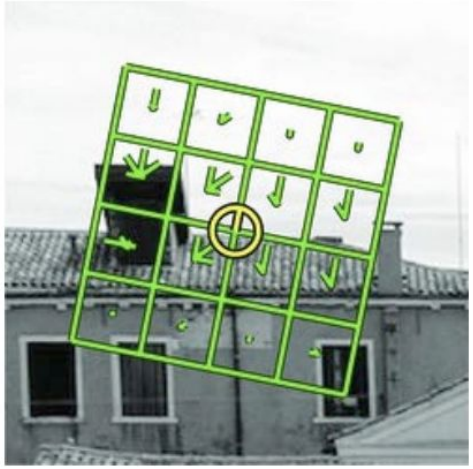
SIFT vs. MOPS

- MOPS is only a descriptor (often uses Harris) while SIFT is both a detector and a descriptor
- MOPS descriptor is a raw pixel patch, while SIFT descriptor uses gradient histograms => SIFT descriptors are more discriminative, especially in repetitive textures
- Both are invariant to translation and rotation
- SIFT is more robust to large scale and illumination changes
- MOPS is faster than SIFT

MOPS is simple and fast but fragile;
SIFT is heavier but more robust and distinctive.

Feature Matching





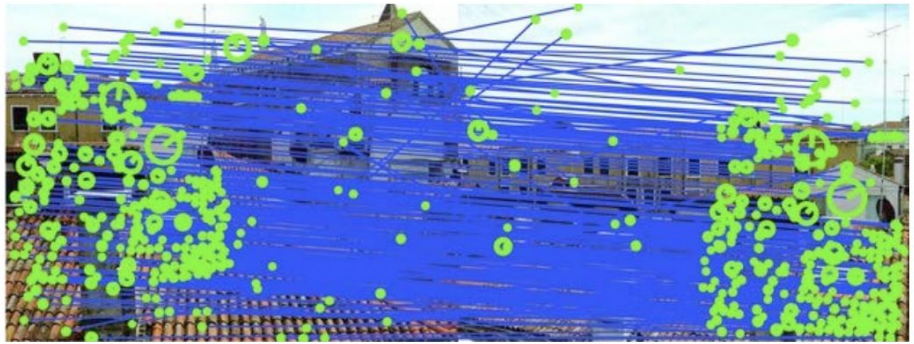
Each image might generate 100's or 1000's of SIFT descriptors

Credit: Kwang Moo Yi

Feature Matching

Goal: Find all correspondences between a pair of images

Means: extract and match all SIFT descriptors from both images

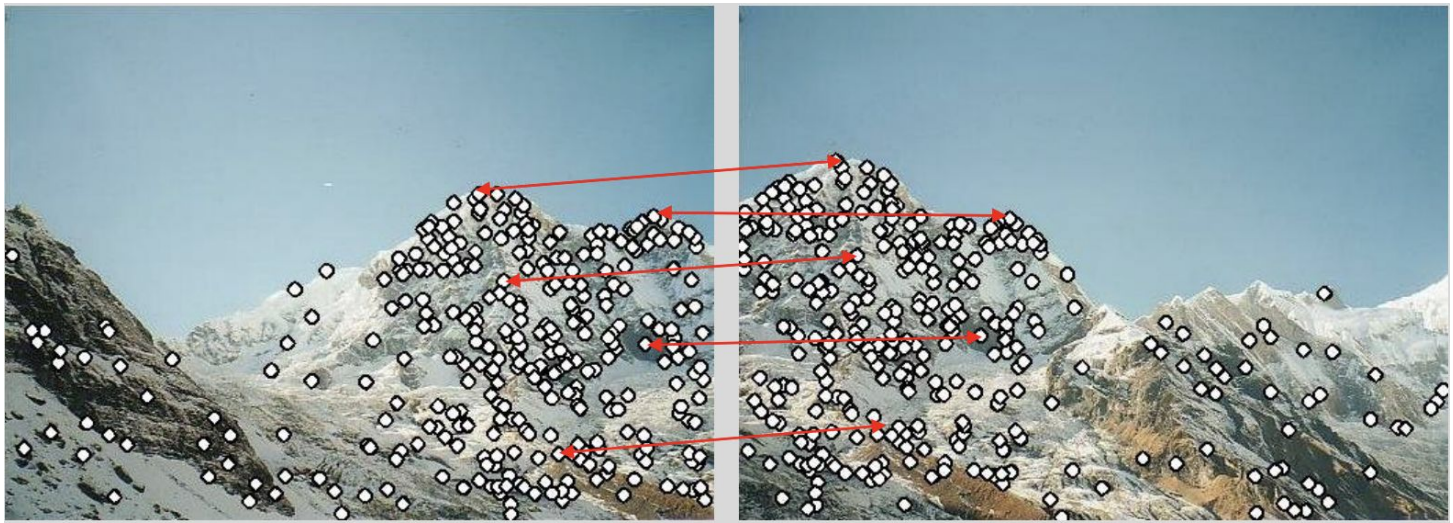


Credit: Kwang Moo Yi

Feature Matching

How to match?

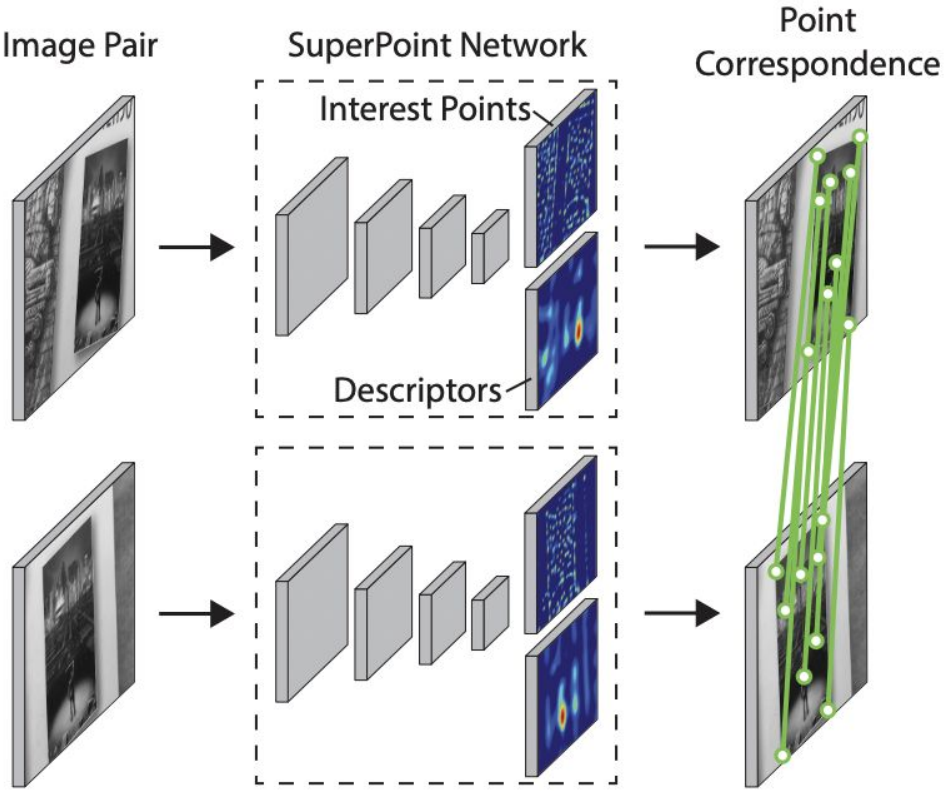
For each keypoint in I1, find nearest neighbor in I2 by euclidean distance between feature descriptor vectors.



Deep feature matching

- **SuperPoint** (CVPR, 2018)

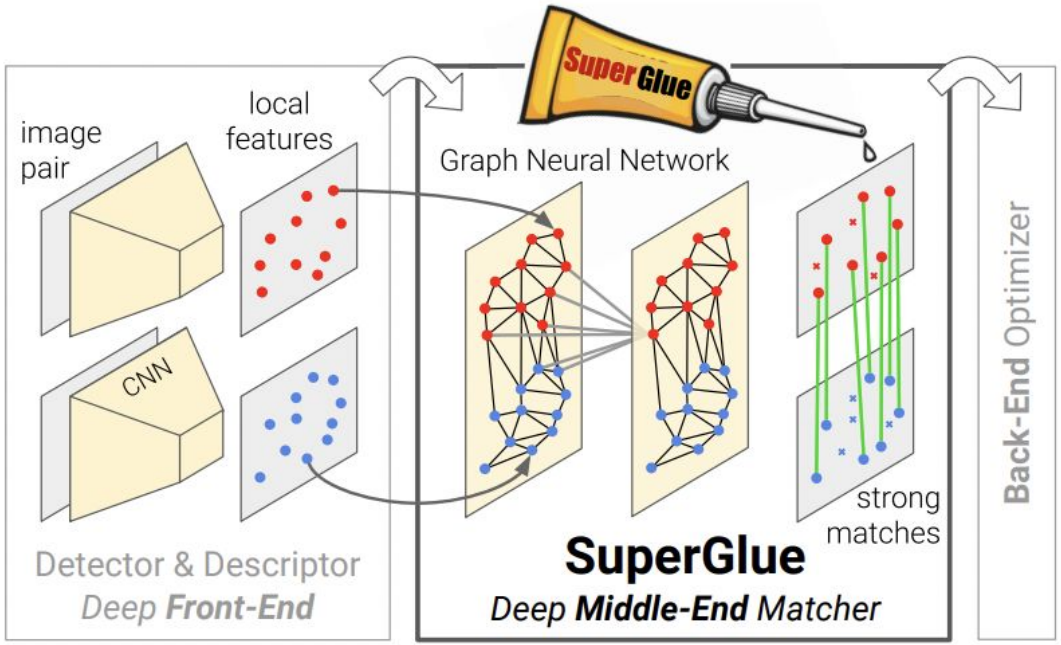
  A fast fully-convolutional neural network that computes SIFT-like 2D interest point locations and descriptors in a single forward pass.



Deep feature matching

- **SuperGlue** (CVPR, 2020)

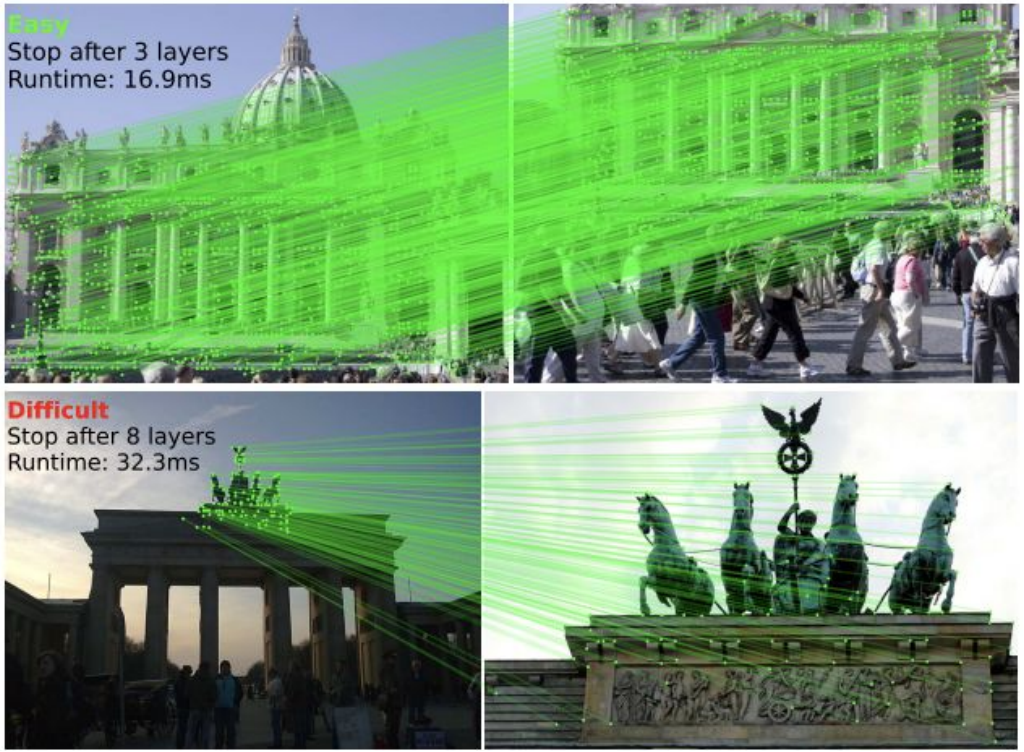
  Uses a graph neural network and attention to solve an assignment optimization problem for pointwise correspondences.



Deep feature matching

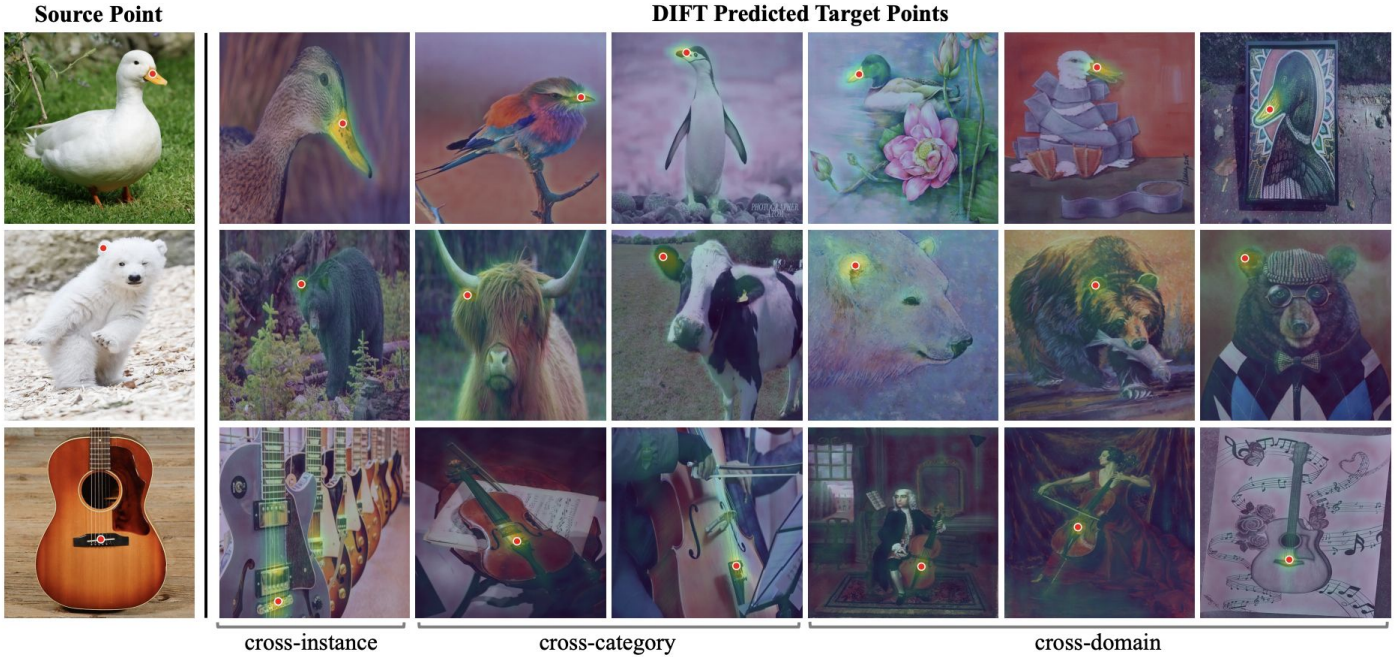
- **LightGlue** (ICCV, 2023)

  A deep neural network that learns to match local features across images using self- and cross-attention, and unmatchable point pruning.



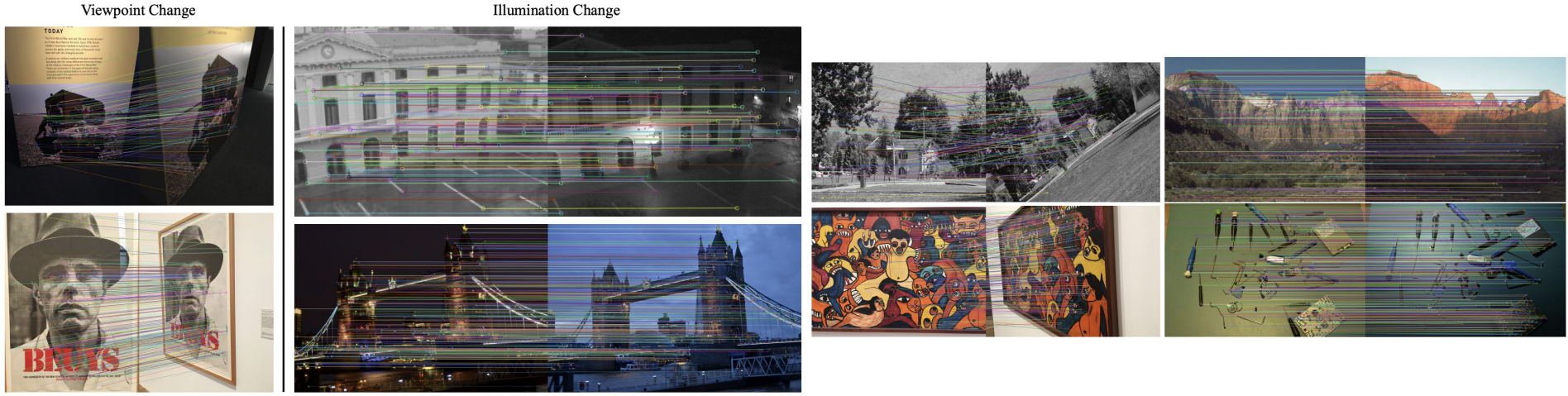
Semantic correspondences

Emergent Correspondence from Image Diffusion (NeurIPS 2023)



Semantic correspondences

Emergent Correspondence from Image Diffusion (NeurIPS 2023)

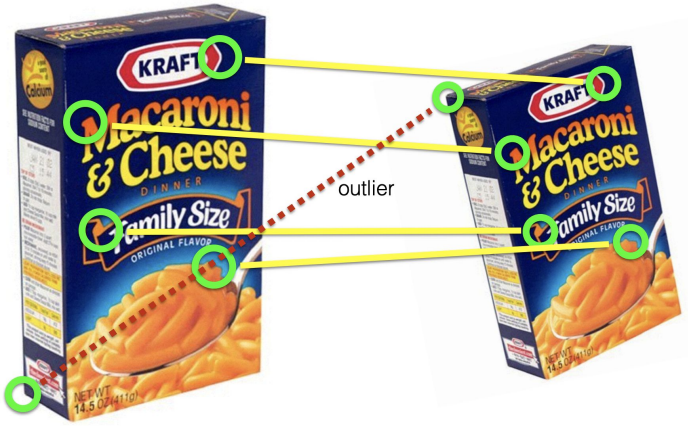


Feature Matching

Matching – For each keypoint in I1, find nearest neighbor in I2 by euclidean distance between feature descriptor vectors.

Matches could be outliers or wrong matches – how can we filter them out?

- Look at 2nd nearest neighbor, if d(1NN) < 0.8*d(2NN) => a good match.
- RANSAC



RANSAC (Random Sample Consensus)

A powerful, iterative algorithm used to estimate parameters for a mathematical model from noisy data containing outliers.

Idea:

- **Sample** random minimal data subsets repeatedly
- **Fit** the mathematical model
- **Count** how many other points support that model (inliers)
- **Select** the most-supported model

RANSAC (Random Sample Consensus)

Why will this work?

All the inliers will agree with each other, while the (hopefully not many) outliers will (hopefully) disagree with each other.

– will work if there are < 50% outliers

“All good matches are alike; every bad match is bad in its own way.”

Tolstoy via Alyosha Efros

RANSAC (Random Sample Consensus)

Example: fit a linear line (y = mx+b), params: (m,b).

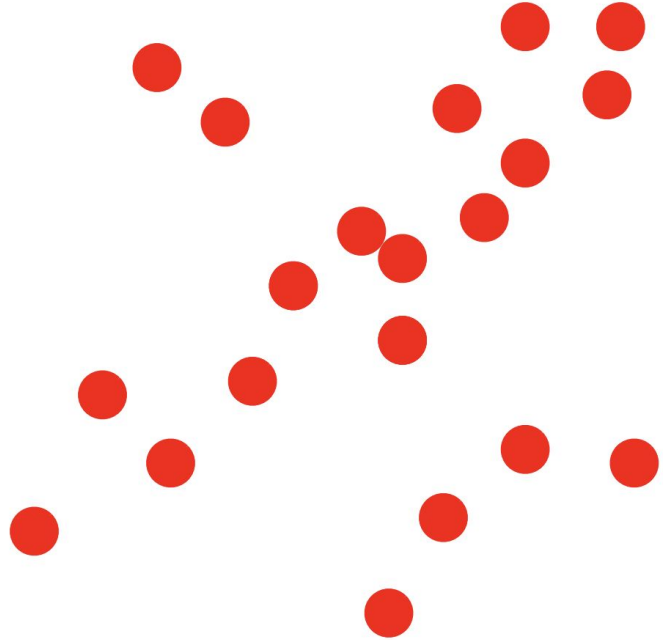
Algorithm:

1. Randomly sample 2 points to fit a linear line

2. Fit the model using the samples

3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until a model is found with high confidence



RANSAC (Random Sample Consensus)

Example: fit a linear line.

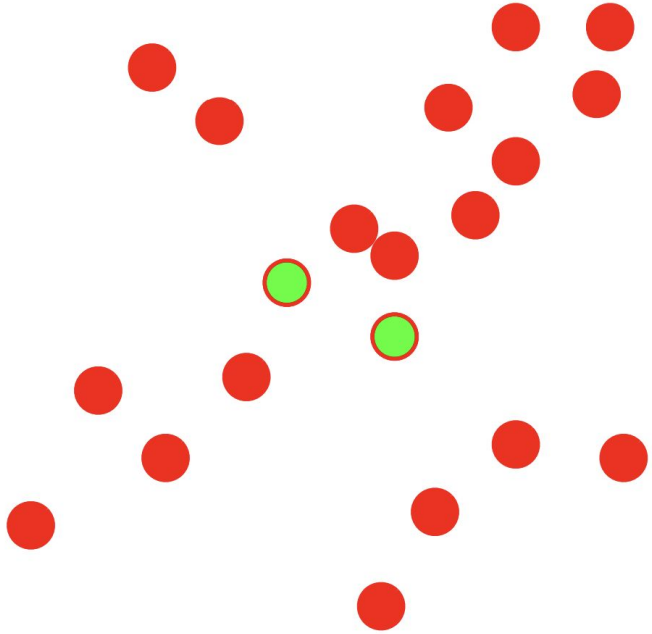
Algorithm:

1. **Randomly sample 2 points to fit a linear line**

2. Fit the model using the samples

3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until a model is found with high confidence



RANSAC (Random Sample Consensus)

Example: fit a linear line.

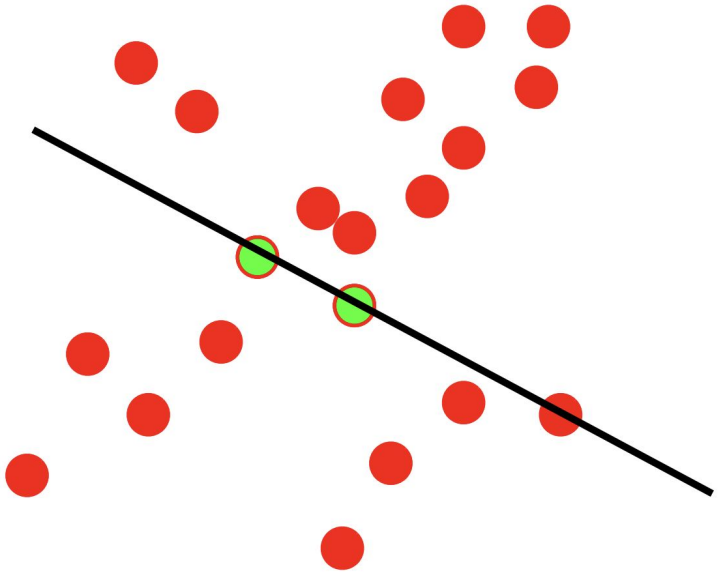
Algorithm:

1. Randomly sample 2 points to fit a linear line

2. **Fit the model using the samples**

3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until a model is found with high confidence



RANSAC (Random Sample Consensus)

Example: fit a linear line.

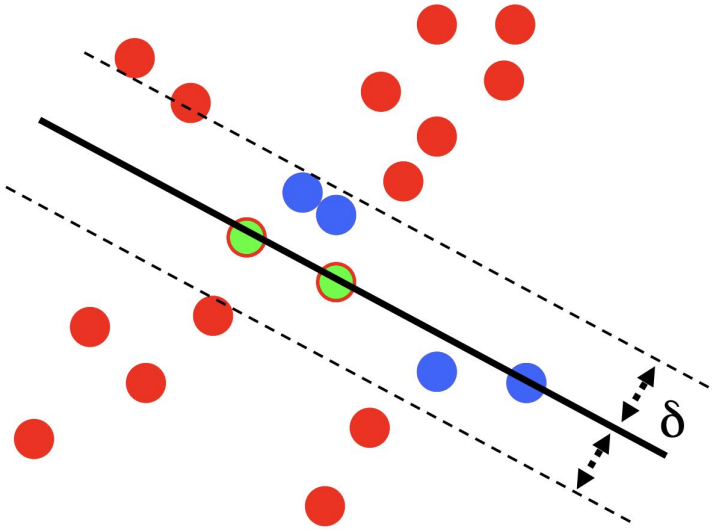
Algorithm:

1. Randomly sample 2 points to fit a linear line

2. Fit the model using the samples

3. **Score by the fraction of inliers within a preset threshold of the model**

Repeat 1-3 until a model is found with high confidence



Ni = 6

RANSAC (Random Sample Consensus)

Example: fit a linear line.

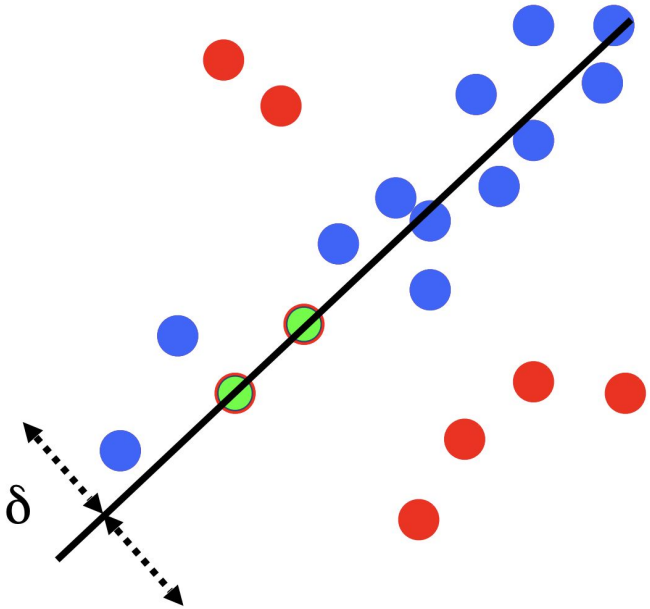
Algorithm:

1. Randomly sample 2 points to fit a linear line

2. Fit the model using the samples

3. Score by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until a model is found with high confidence



Ni = 14

RANSAC (Random Sample Consensus)

Which hyper-parameters to use?

- **Number of sample points s**: the minimum number needed to fit the model.

- **Distance threshold δ**: Choose δ based on the amount of noise we want to allow – a good point with noise should be within the threshold.

- Choose **sample number (repetitions) N** such that with probability p, at least one random sample is free from outliers.

$$N = \frac{\log(1-p)}{\log\left(1-(1-e)^s\right)}$$

	proportion of outliers e						
s	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Reminder - Transformations

Original

Translation

Rotation

Scaling

Affine

Perspective

Reminder - Transformations

Translation: Moves points by (tx,ty).

$$\mathbf{x}' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation: Rotates around a point (cx,cy) by angle θ.

$$\mathbf{x}' = \begin{bmatrix} \cos\theta & -\sin\theta & c_x(1-\cos\theta) + c_y\sin\theta \\ \sin\theta & \cos\theta & c_y(1-\cos\theta) - c_x\sin\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reminder - Transformations

Scaling: Uniform or non-uniform resizing by (sx,sy)

$$\mathbf{x}' = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Affine transformation: Linear transformation + translation.
Preserves parallel lines.

$$\mathbf{x}' = \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How many degrees of freedom?

6 => Minimum points needed for RANSAC:
6 equations/2 per point = 3 points

Reminder - Homogeneous coordinates

- A mapping $R^n -> R^{n+1}$, $e.g.\ (x, y) -> (tx, ty, t)$
  Inverse mapping: (tx,ty,t) → (tx/t,ty/t) = (x,y)

- For $a, b \in \mathbb{R}^3$:
  - a X b = 0 iff a and b are parallel
  - a X b is orthogonal to both a and b

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix}$$

- In Homogeneous coordinates:
  two vectors representing the same 2D point are parallel → their cross product is zero.

Reminder - Transformations

Perspective Transformation (Homography): Maps a plane to another plane in projective space. Can handle perspective distortions.

$$\mathbf{x}' \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$H \sim k \cdot H \quad \text{for any } k \neq 0$$

Reminder - Transformations

Homography in our context: Maps **world coordinates** to **image pixels**.

$$\vec{\mathbf{x}}_{img} \equiv H\vec{\mathbf{x}}_w$$

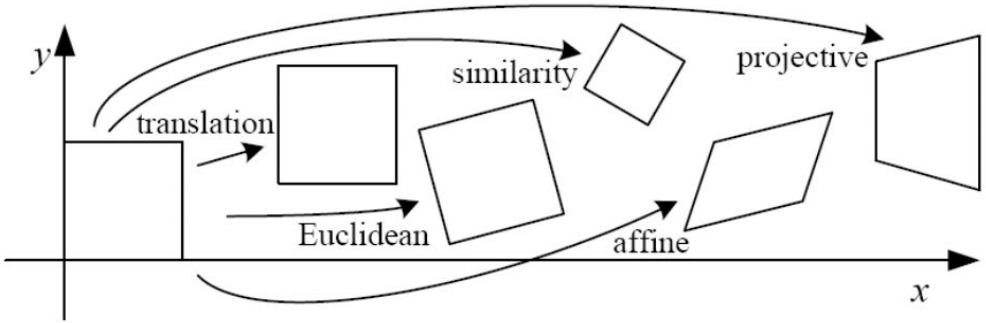
$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$

How many degrees of freedom?

8 => Minimum points needed for RANSAC: 4 points.

$$H \sim k \cdot H \quad \text{for any } k \neq 0$$

Reminder - Transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times3}$	8	straight lines	

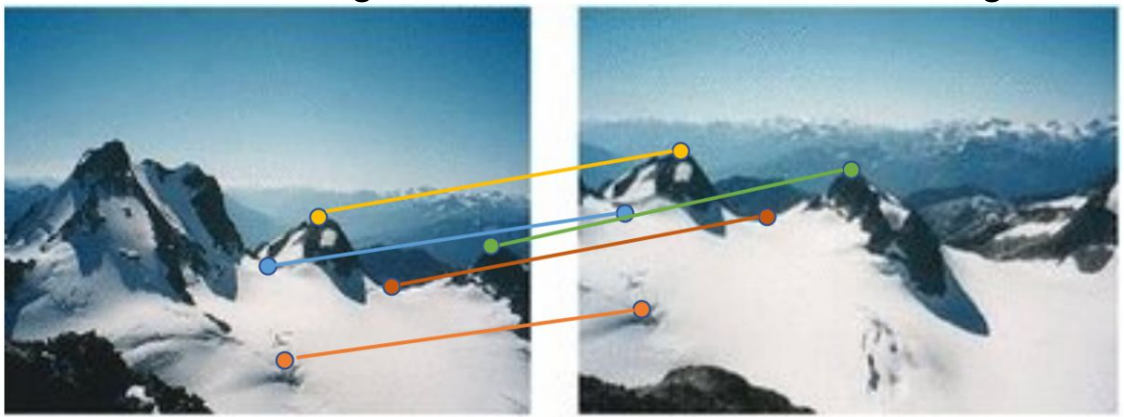
Compute Homography

For each corresponding pair of points (Xd, Xs):

$$\begin{bmatrix} x_d^{(i)} \\ y_d^{(i)} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_s^{(i)} \\ y_s^{(i)} \\ 1 \end{bmatrix}$$

Source Image

Destination Image



Compute Homography

$$x' = \lambda H x$$

$$\mathbf{x}' \times (H\mathbf{x}) = \begin{pmatrix} y'(h_7 x + h_8 y + h_9) - (h_4 x + h_5 y + h_6) \\ (h_1 x + h_2 y + h_3) - x'(h_7 x + h_8 y + h_9) \\ x'(h_4 x + h_5 y + h_6) - y'(h_1 x + h_2 y + h_3) \end{pmatrix}$$

$$x'(h_7 x + h_8 y + h_9) = h_1 x + h_2 y + h_3$$

$$y'(h_7 x + h_8 y + h_9) = h_4 x + h_5 y + h_6$$

Compute Homography

$$x'(h_7x + h_8y + h_9) = h_1x + h_2y + h_3$$

$$y'(h_7x + h_8y + h_9) = h_4x + h_5y + h_6$$

In matrix form:

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

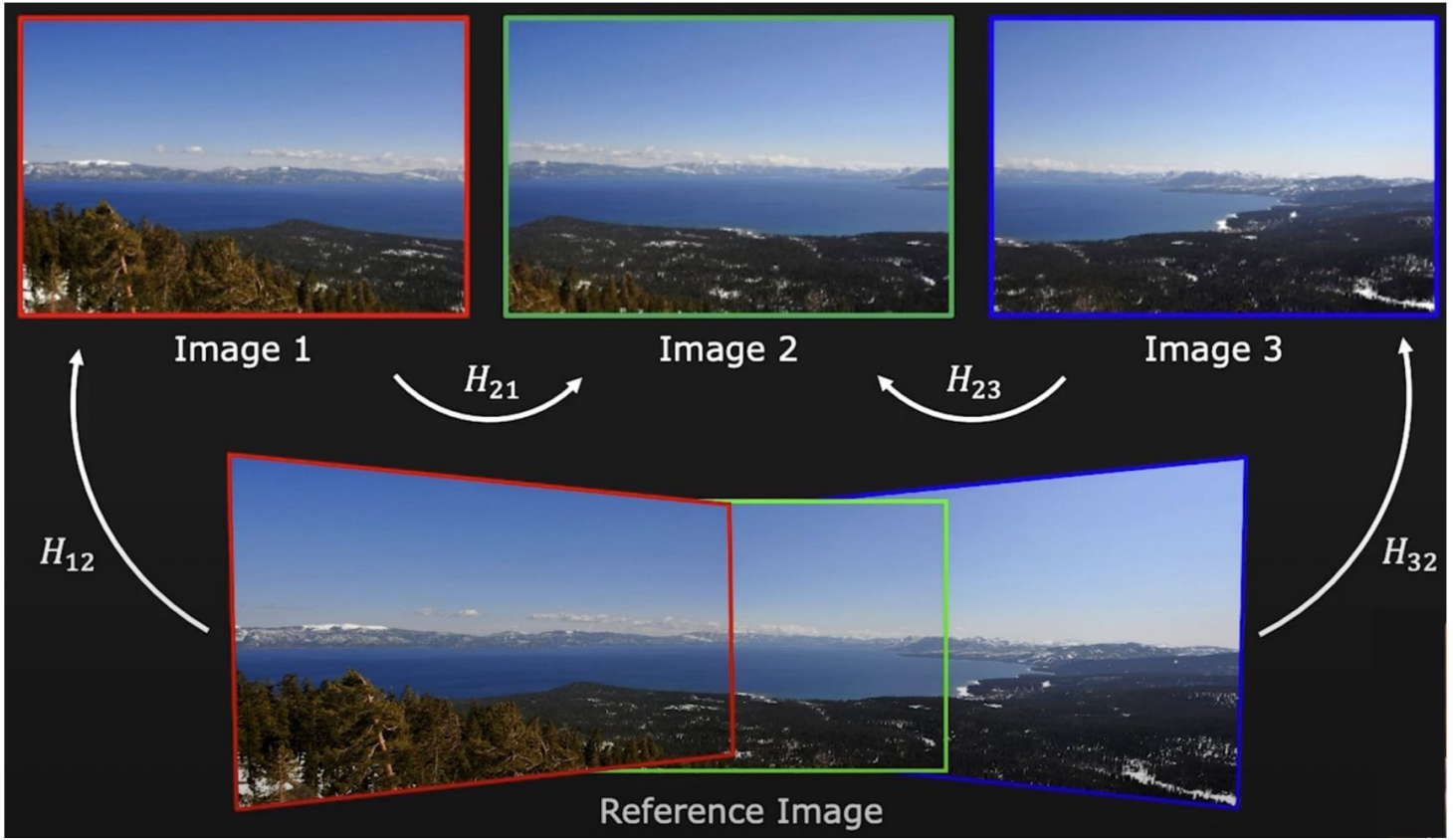
Compute Homography

$$\begin{bmatrix} -x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ & & & & & & \vdots & & \\ -x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\ & & & & & & \vdots & & \\ -x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Use RANSAC for finding Homography

1. Randomly sample 4 corresponding points
2. Fit a homography H using the samples
3. Score by the fraction of inliers within a preset threshold of the model and keep H if the current score is highest
4. Repeat 1-3 until a homography is found with high confidence
5. Recompute H using all inliers

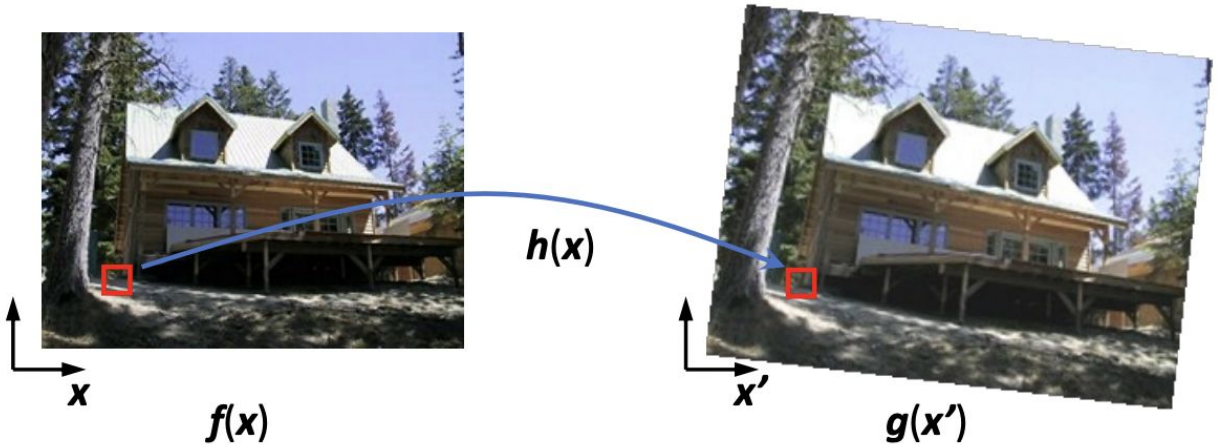
Apply found Homography



Forward Warping

Move each pixel *x* from its location in the original image, into its corresponding location after applying the homography: *x'* = *h(x)*.

What if a pixel lands "between" two pixels?

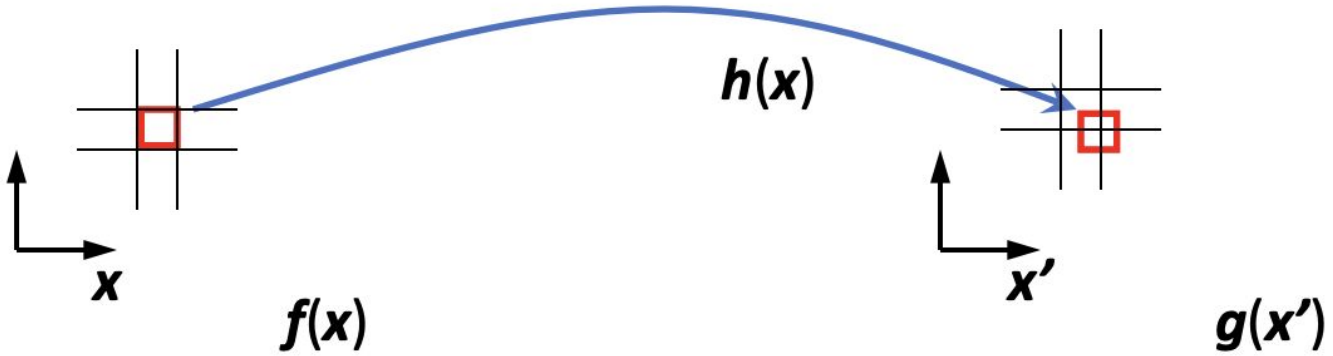


Credit: szeliski

Forward Warping

Move each pixel *x* from its location in the original image, into its corresponding location after applying the homography: *x'* = *h(x)*.

What if a pixel lands "between" two pixels?
add "contribution" to several pixels, normalize later (splatting)



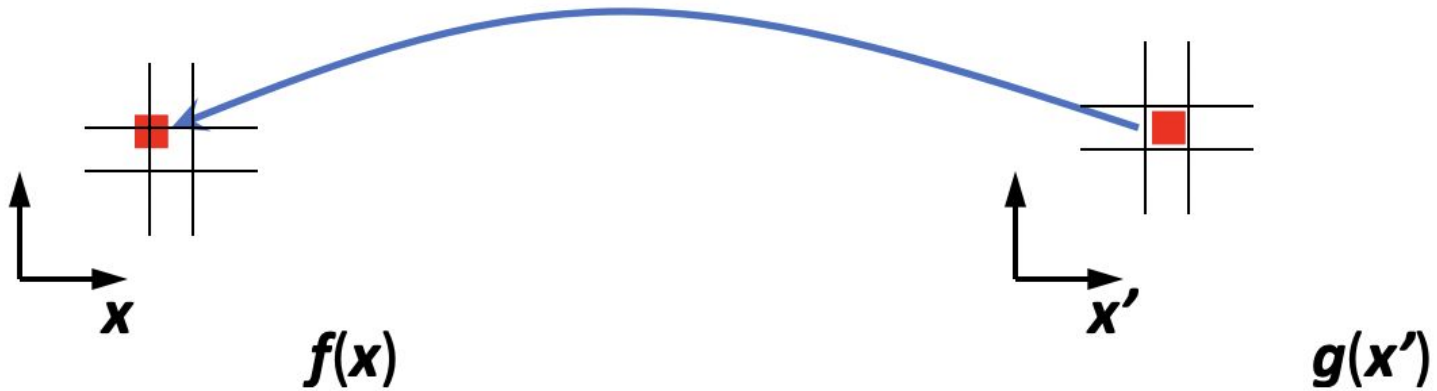
Credit: szeliski

Backward Warping

Get each pixel *x'* from its corresponding location in the original image: *x' = h(x)*.

What if a pixel lands “between” two pixels?

Interpolate color value from relevant source image values.



Credit: szeliski

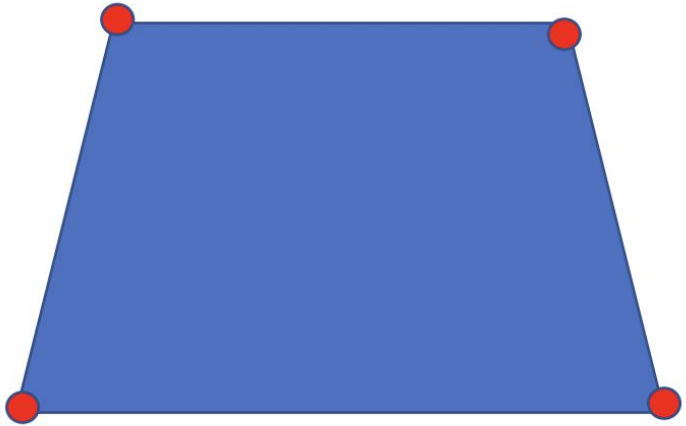
Backward Warping

Find the forward warping of the corners.

Credit: szeliski

Backward Warping

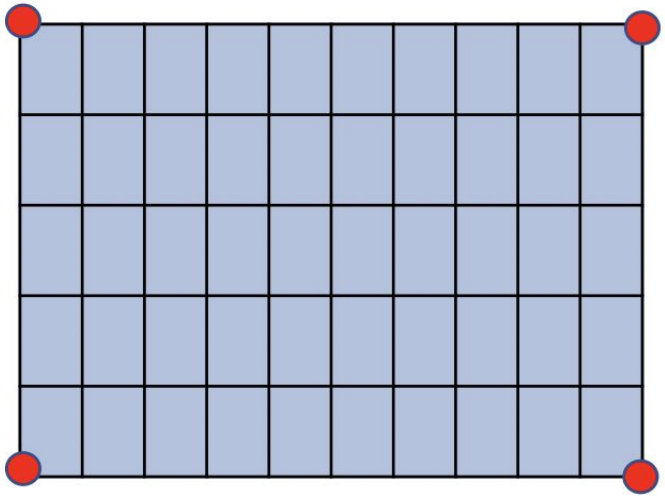
Find the forward warping of the corners.

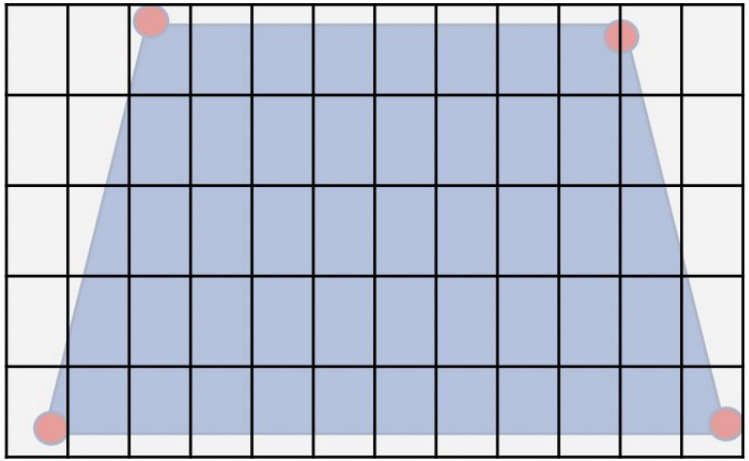


Credit: szeliski

Backward Warping

Set grid mask and pixels to fill

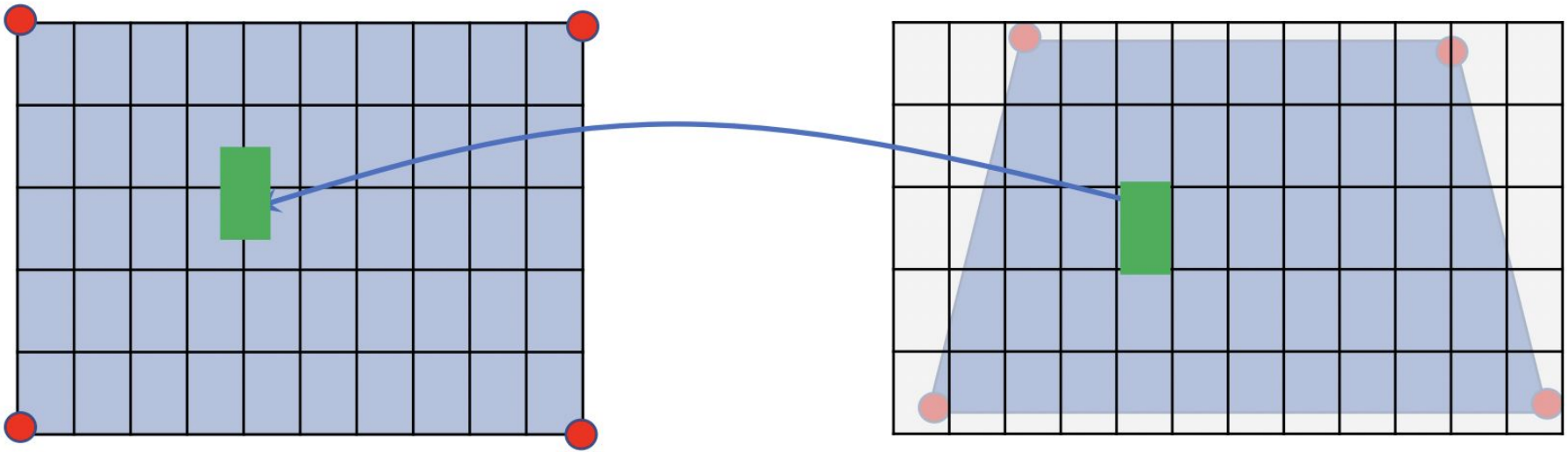




Credit: szeliski

Backward Warping

For each pixel in the mask – use backward warping.



Credit: szeliski