Topic 11:

Feature Detection & Image Matching

- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

The Image Matching Problem

Image I

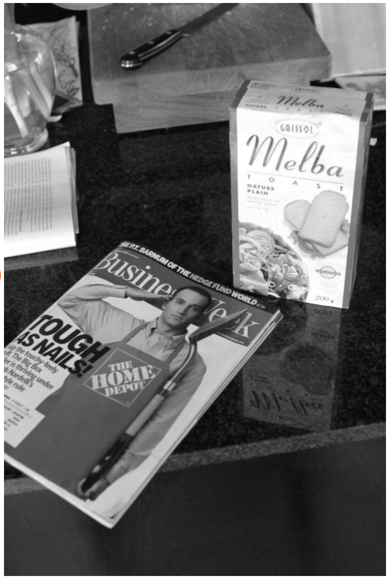
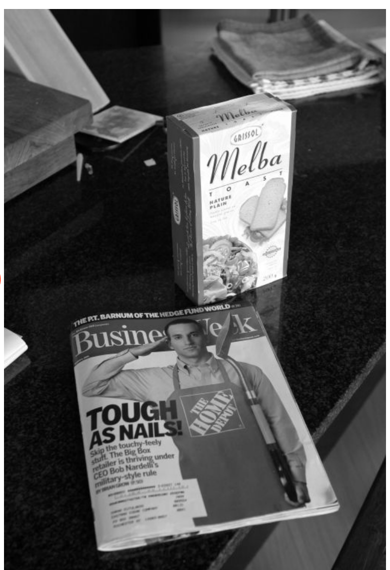


Image I’

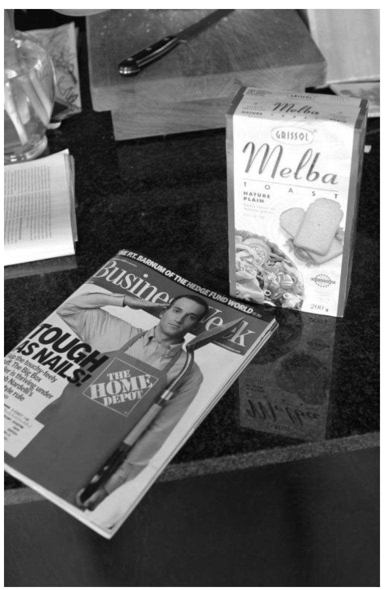


Goal:
Identify “features” or patches in image I that appear in another image, I’

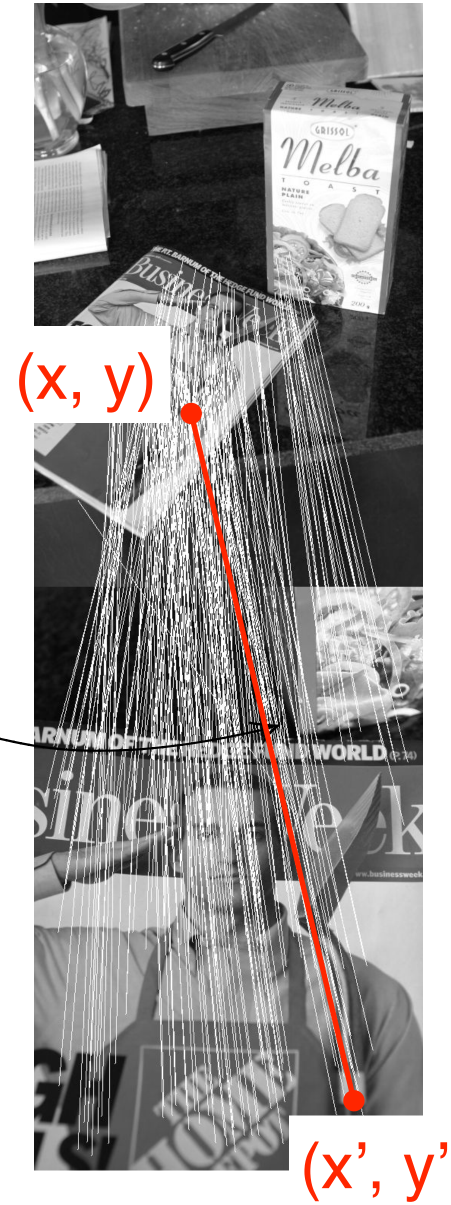
The Image Matching Problem

Image I

Image I’



Indicates a correspondence between location (x, y) in image I and location (x’, y’) in image I’



The Image Matching Problem

Image I

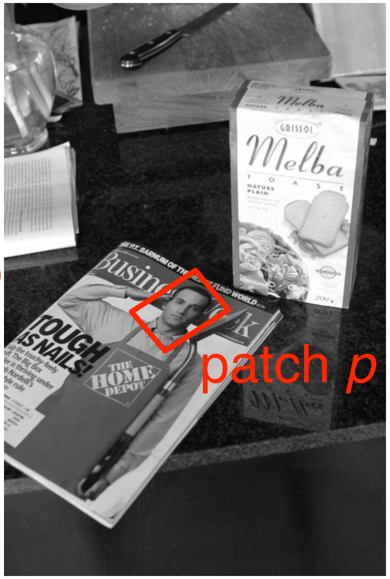
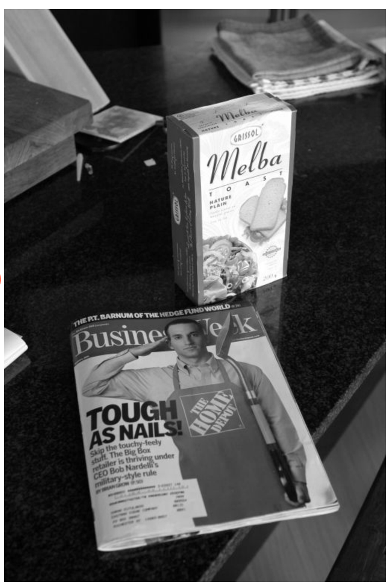


Image I’

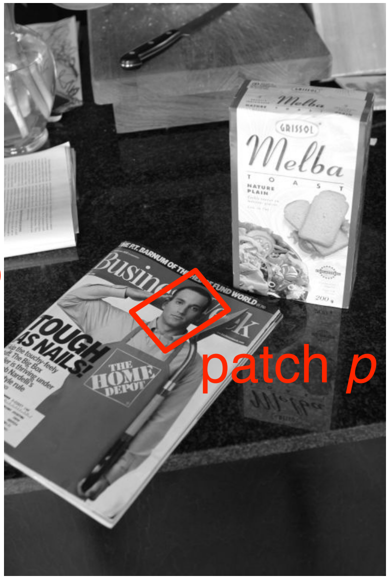


Q:

Is it possible to solve this problem by direct template matching between two images?

The Image Matching Problem

Image I



Q:

Is it possible to solve this problem by direct template matching between two images?

Image I’

The Image Matching Problem

Image I

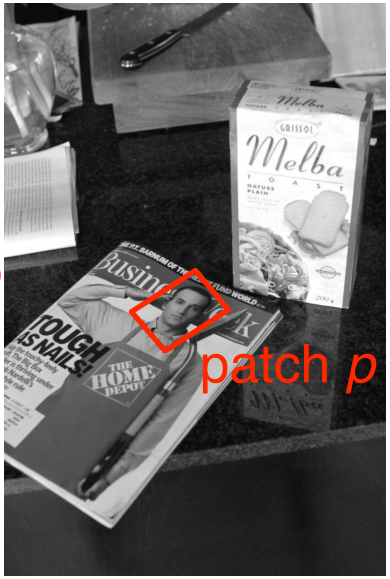


Image I’

Q:

Is it possible to solve this problem by direct template matching between two images?

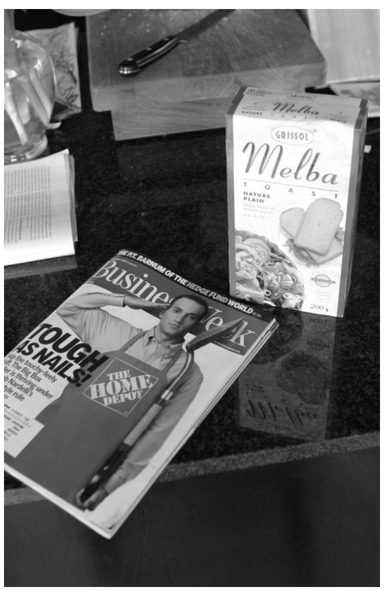
A:

Yes, but it would be impossibly inefficient (ie. must search over all possible pairs of patches)

Feature-Based Image Matching

Image I

Image I'

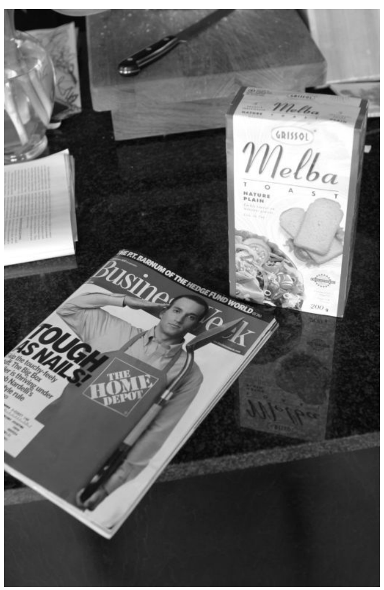


Feature detection & matching

Feature-Based Image Matching

Image I

Image I'



Detect features in I

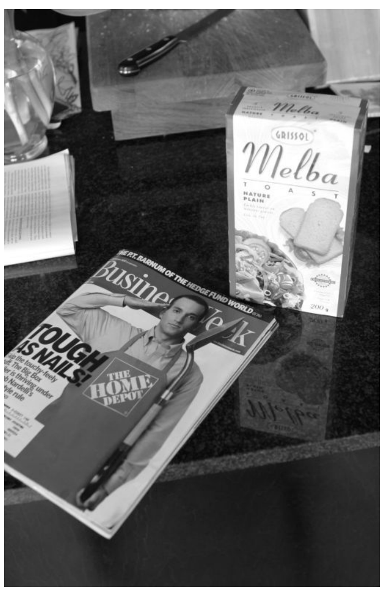
Detect features in I'

Match features across the two images

Errors in Feature-Based Image Matching

Image I

Image I’



In general, some/many of these correspondences may be incorrect.

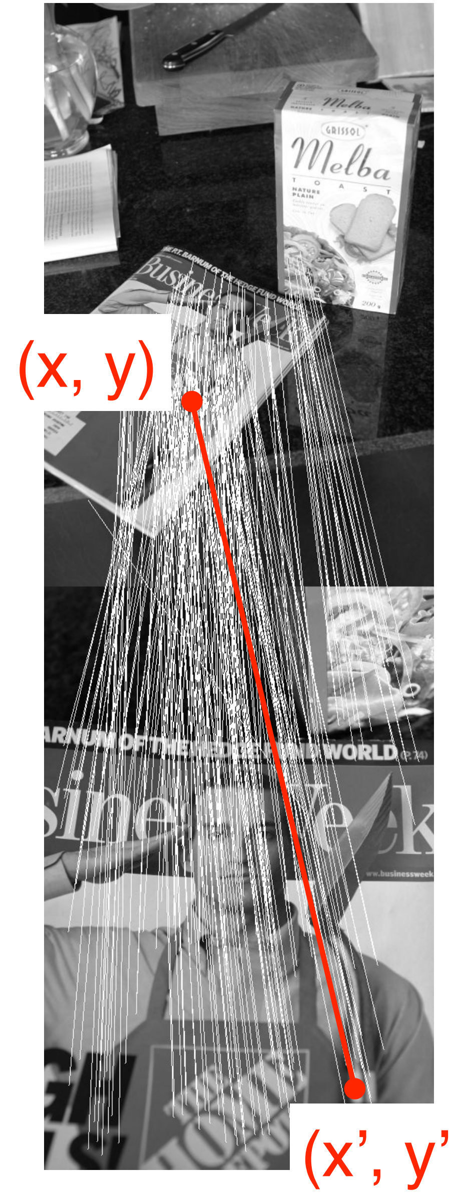
Two types of error:

1. False positive matches

   algorithm returns a correspondence between 2 locations where none exists

2. False negative matches

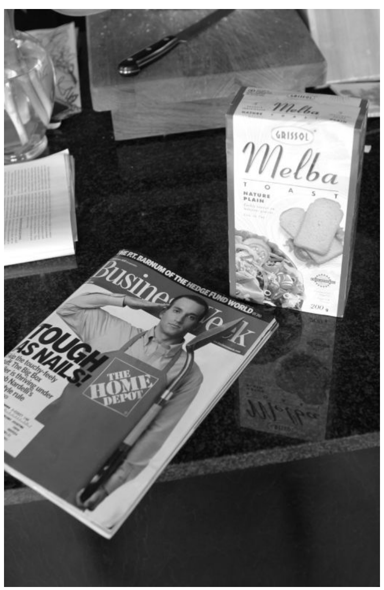
   algorithm fails to detect a correspondence between two instances of the same feature/patch.



Errors in Feature-Based Image Matching

Image I

Image I'



GOAL: minimize false positive and false negatives across a wide range of imaging conditions.

1. False positive matches

   algorithm returns a correspondence between 2 locations where none exists

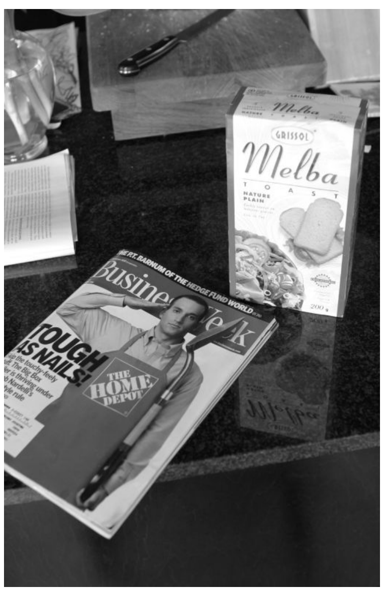
2. False negative matches

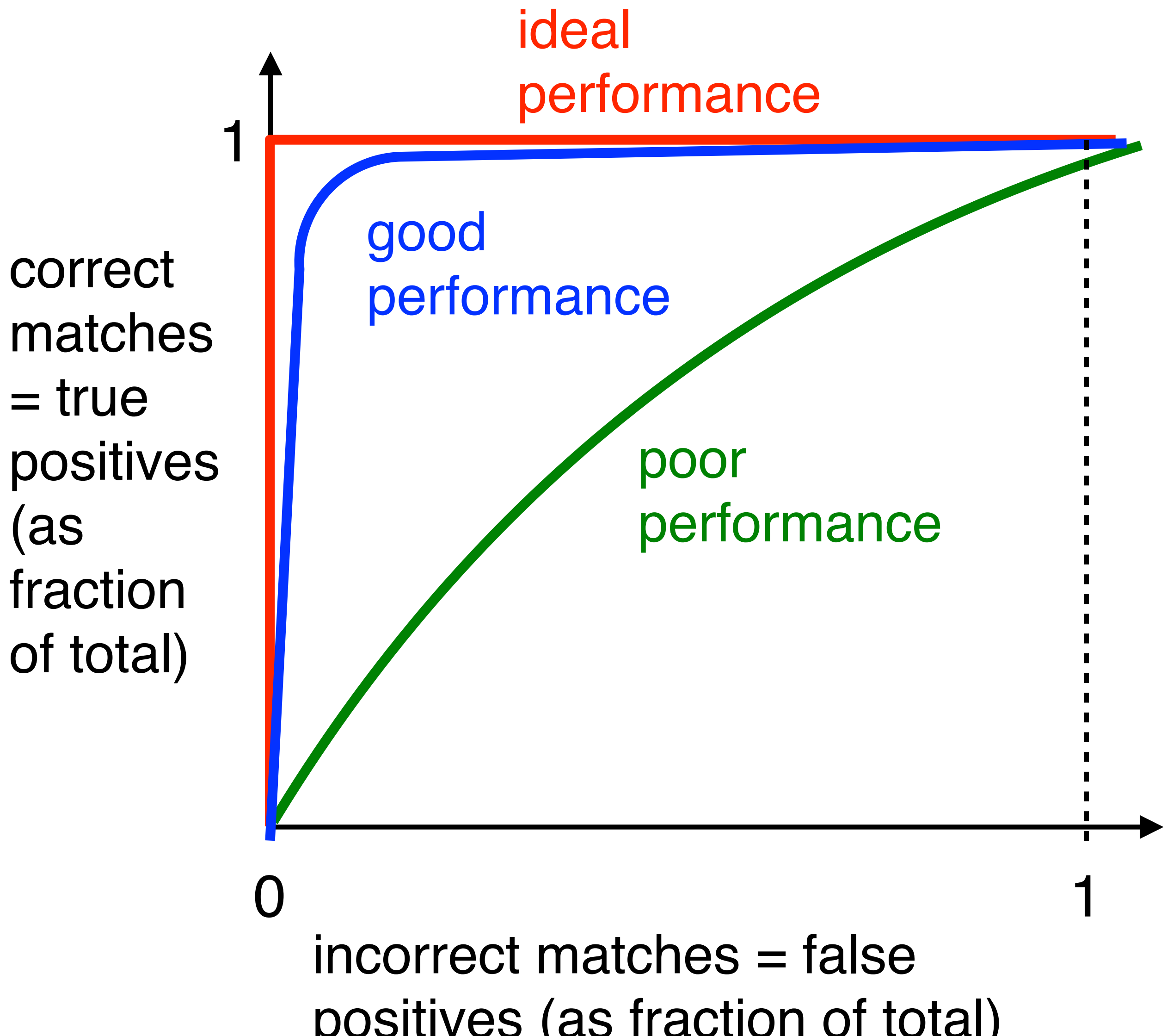
   algorithm fails to detect a correspondence between two instances of the same feature/patch.

Evaluating a Feature Detector's Performance

Image I

Image I'





Feature Matching & Transformation Invariance

Source Image I

?

“Transformed” source images

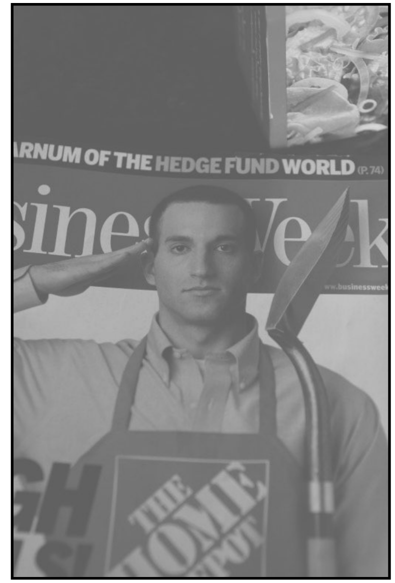
To be most useful, the feature detector & matching algorithm must be insensitive to a wide range of image transformations.

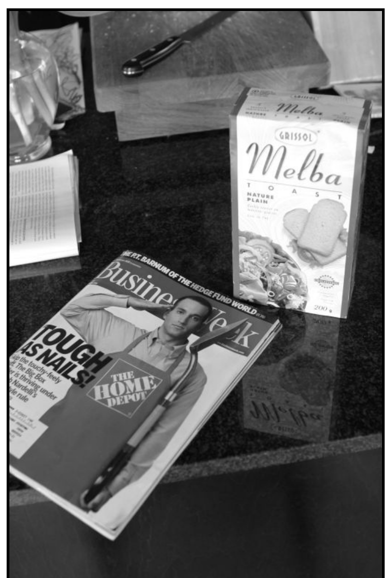
Transformation-Invariant Feature Detectors

Source Image I

?

"Transformed" source images





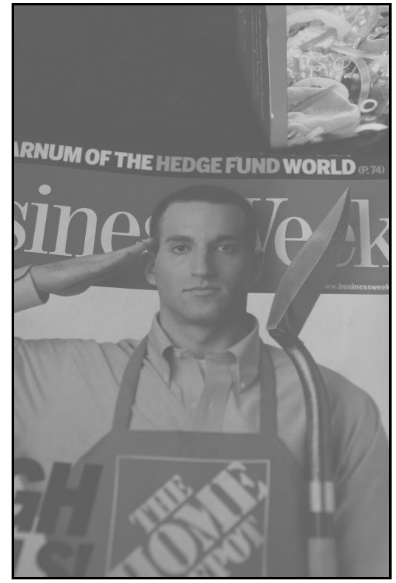
A feature detector is called **invariant** to a certain image transformation if it can reliably detect features in a transformed version of the source image

Transformation-Invariant Feature Detectors

Source Image I

?

“Transformed” source images



Brightness transformation

A feature detector is called **invariant** to a certain image transformation if it can reliably detect features in a transformed version of the source image

Transformation-Invariant Feature Detectors

Source Image I

?

"Transformed" source images

Distortion due to change in viewpoint

A feature detector is called **invariant** to a certain image transformation if it can reliably detect features in a transformed version of the source image

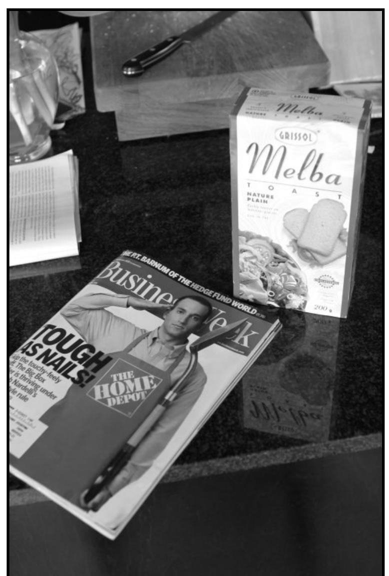
Transformation-Invariant Feature Detectors

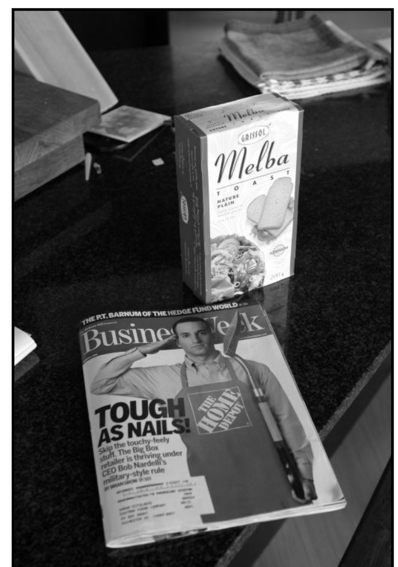
Source Image I

?

"Transformed" source images

A feature detector is called **invariant** to a certain image transformation if it can reliably detect features in a transformed version of the source image





Distortion due to change in viewpoint & magnification (ie. scale)

Topic 11:

Feature Detection & Image Matching

- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

Image Matching Using SIFT Features

SIFT: **S**cale **I**nvariant **F**eature **T**ransform

- Developed by David Lowe in 1999
- One of the most powerful representations for feature detection and matching
- Widely used in applications that range from robotics, to image retrieval & recognition, image stitching, video analysis.

Image Matching Using SIFT Features

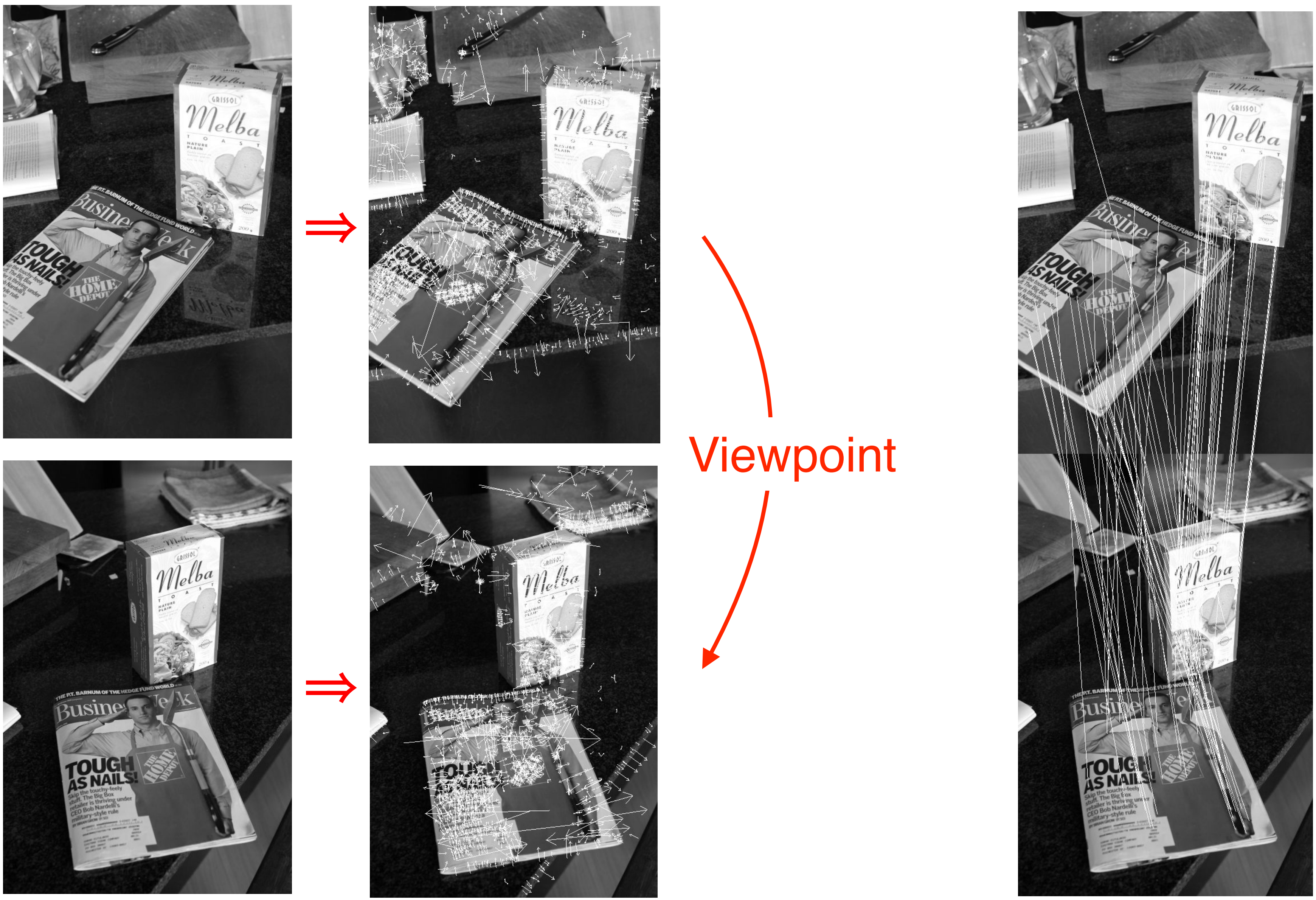
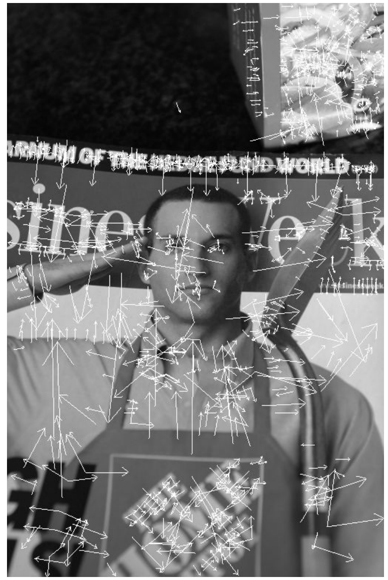
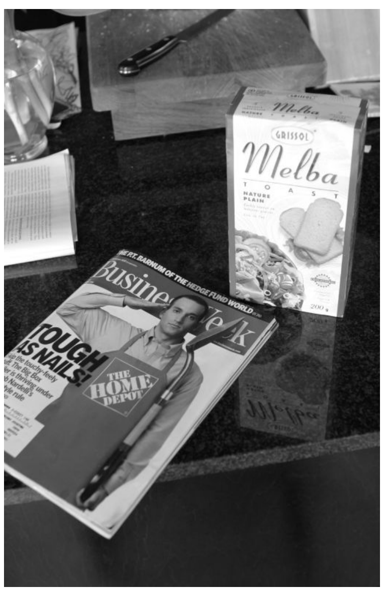


Image Matching Using SIFT Features

⇒





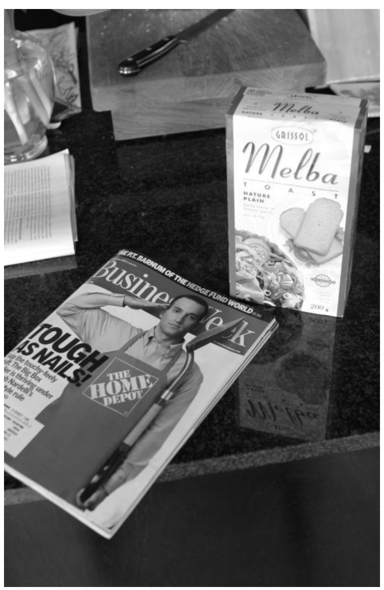
⇒

Scale + Viewpoint

Image Matching Using SIFT Features

Image I

Image I'



Compute SIFT features in I

Compute SIFT features in I'

Match SIFT features across the two images

The SIFT Feature Detection Algorithm

Goal: Represent an image I as a collection of SIFT features that can be identified reliably in other images where the same (or similar) objects are present.

Input:
Image I

Output:
A set of k SIFT **keypoints** $\{p_1, p_2, \ldots, p_k\}$ & **feature vectors** $\{f_1, f_2, \ldots, f_k\}$.

"Source" image I

"Query" image I

The SIFT Feature Detection Algorithm

Goal: Represent an image I as a collection of SIFT features that can be identified reliably in other images where the same (or similar) objects are present.

Input:

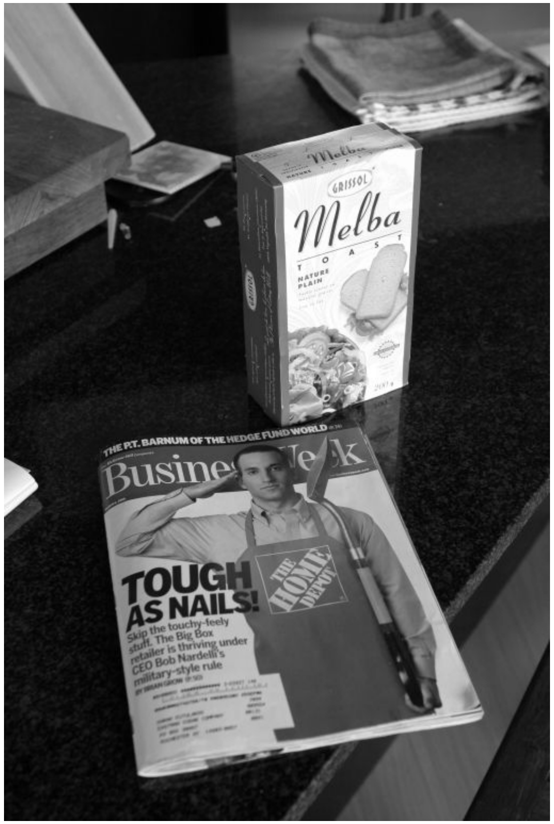
Image I

Output:

A set of *k* SIFT **keypoints** $\{p_1, p_2, \ldots, p_k\}$ & **feature vectors** $\{f_1, f_2, \ldots, f_k\}$.

“Source” image I

“Query” image I



The SIFT Keypoints

Keypoint: A location (x, y) in the source image, with an associated orientation & scale, that is "visually distinct" from its surroundings.

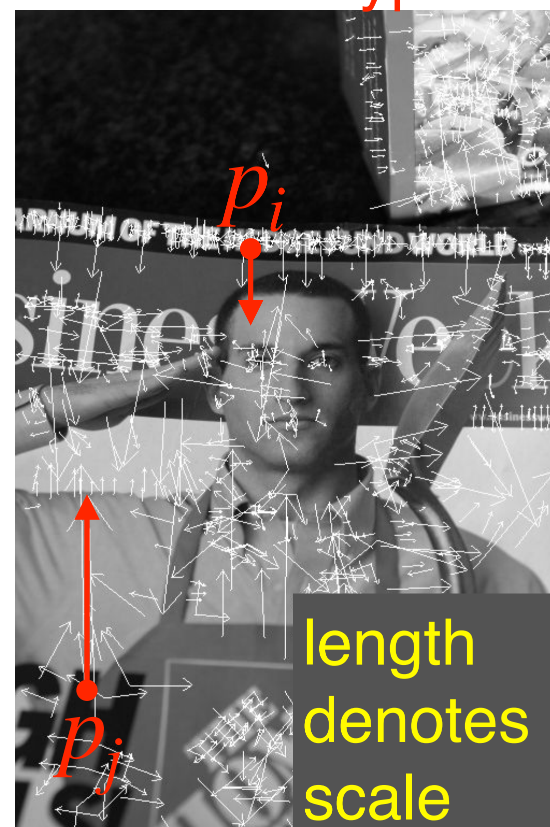
Input:
Image I

Output:
A set of *k* SIFT **keypoints**

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\rho_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

"Source" image I

Detected keypoints



The SIFT Feature Vectors

Feature vector (of a keypoint p_i): A vector of fixed length that represents the image patch centred at p_i.

Input:

Image I

Output:

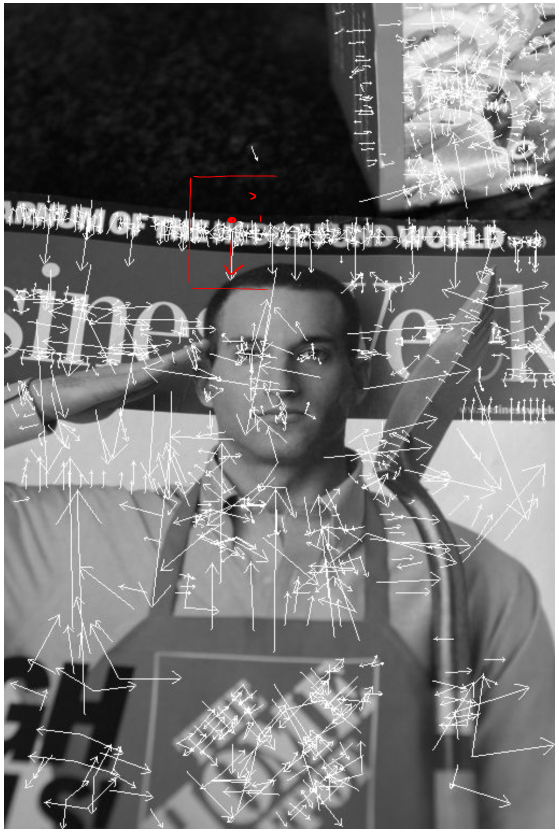
A set of k **keypoints**

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\rho_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

A set of k **feature vectors**

$$f_i = \underbrace{\boxed{}\boxed{}\boxed{\cdots}\boxed{}\boxed{}}_{\text{describes the patch centred at } p_i}$$

Detected keypoints

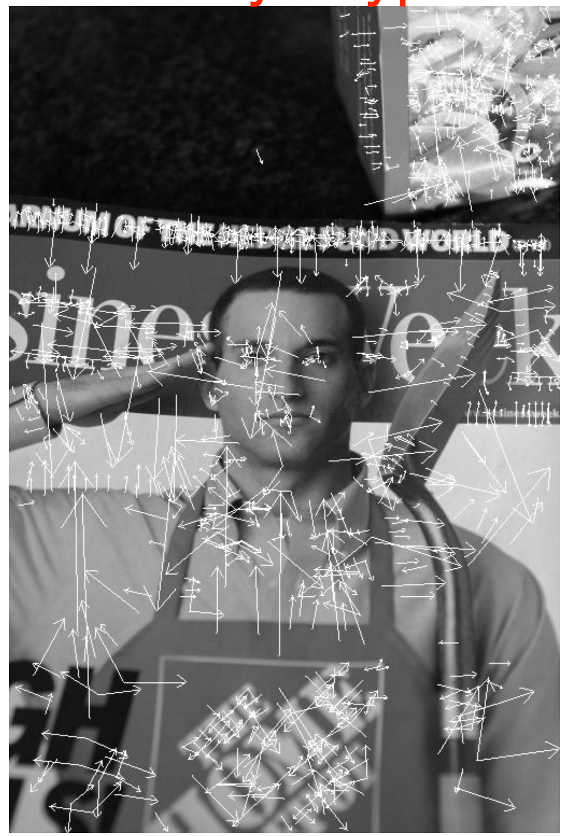


Representing an Image Using SIFT Features: Steps

Source Image I

$\Rightarrow$

1. Identify keypoints



$\Rightarrow$

2. Build feature vectors

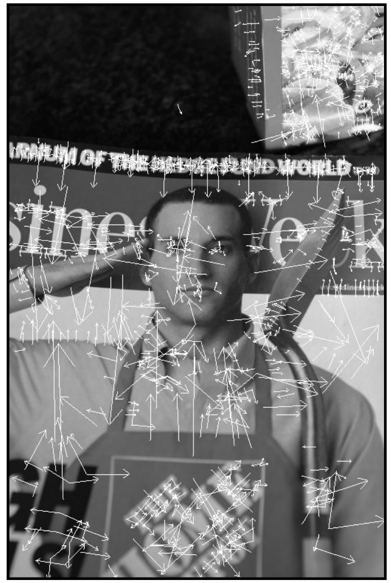
$$f_1 = \boxed{}\boxed{}\boxed{- - -}\boxed{}\boxed{}$$
$$f_2 = \boxed{}\boxed{}\boxed{- - -}\boxed{}\boxed{}$$
$$f_3 = \boxed{}\boxed{}\boxed{- - -}\boxed{}\boxed{}$$
$$\vdots$$
$$f_{k-1} = \boxed{}\boxed{}\boxed{- - -}\boxed{}\boxed{}$$
$$f_k = \boxed{}\boxed{}\boxed{- - -}\boxed{}\boxed{}$$

The number k of detected keypoints depends on I

The dimension of each feature vector is the same for all images.

Matching 2 Images Using SIFT Features

Image I



$f_1 = \square\square\square\square\square\square$

$f_2 = \square\square\square\square\square\square$

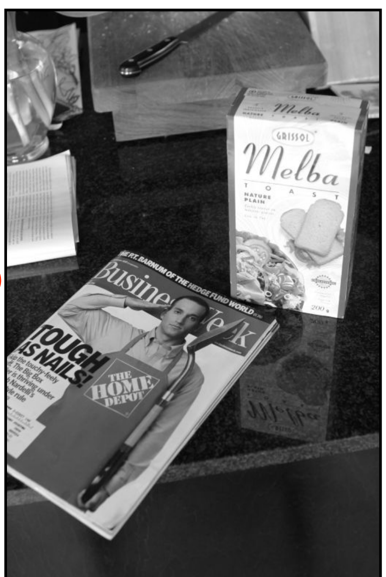
$f_3 = \square\square\square\square\square\square$

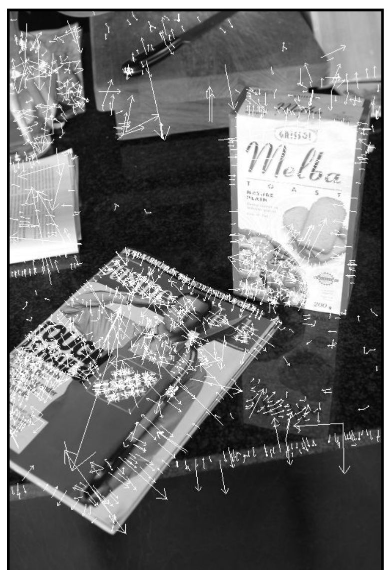
$\vdots$

$f_{k-1} = \square\square\square\square\square\square$

$f_k = \square\square\square\square\square\square$

Image I'





$f'_1 = \square\square\square\square\square\square$

$f'_2 = \square\square\square\square\square\square$

$f'_3 = \square\square\square\square\square\square$

$\vdots$

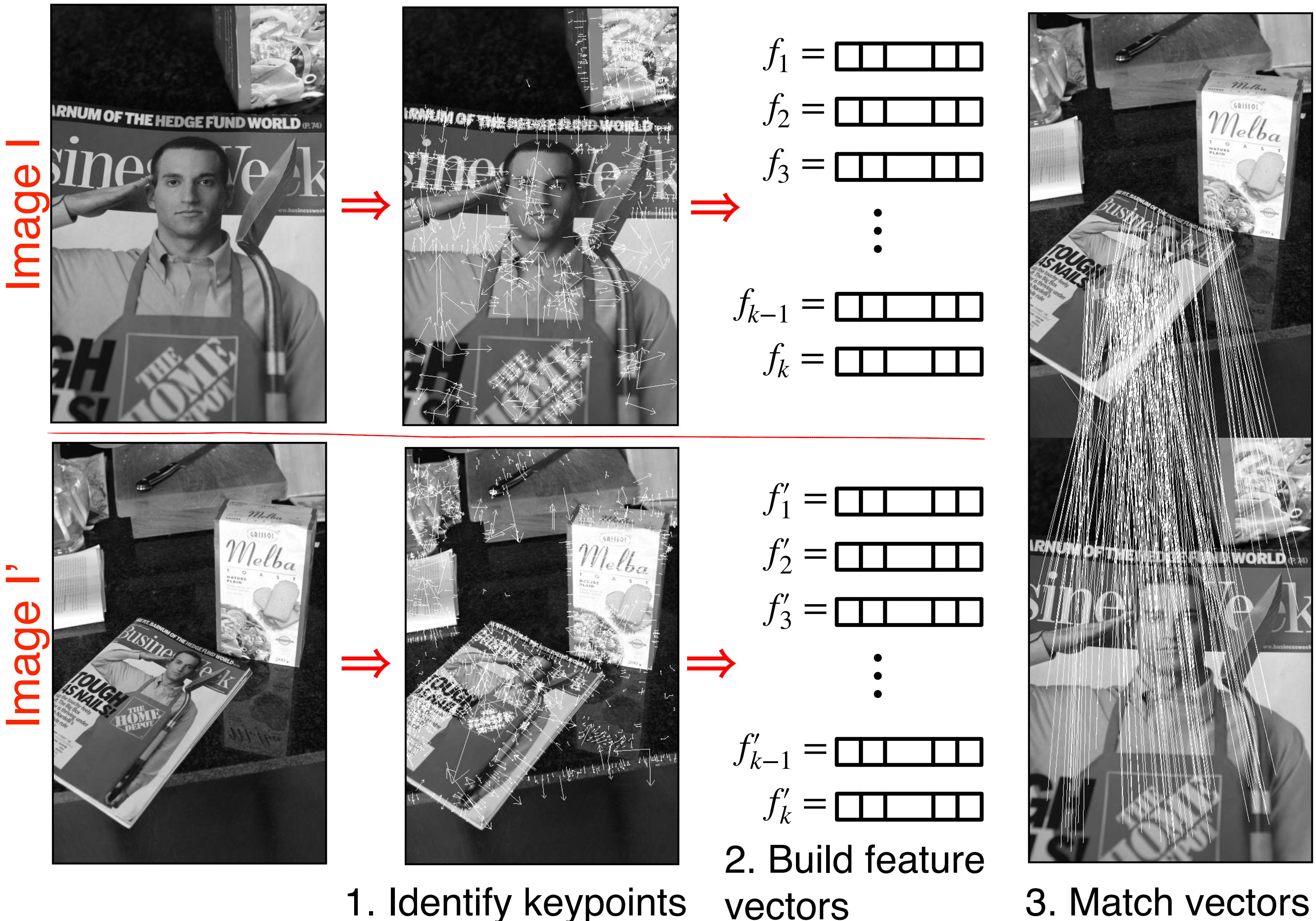
$f'_{k-1} = \square\square\square\square\square\square$

$f'_k = \square\square\square\square\square\square$

3. Match feature vectors in the two sets: $\{f_1, f_2, \ldots, f_k\}$ and $\{f'_1, f'_2, \ldots, f'_k\}$

1. Identify keypoints 2. Build feature vectors

Matching 2 Images Using SIFT Features

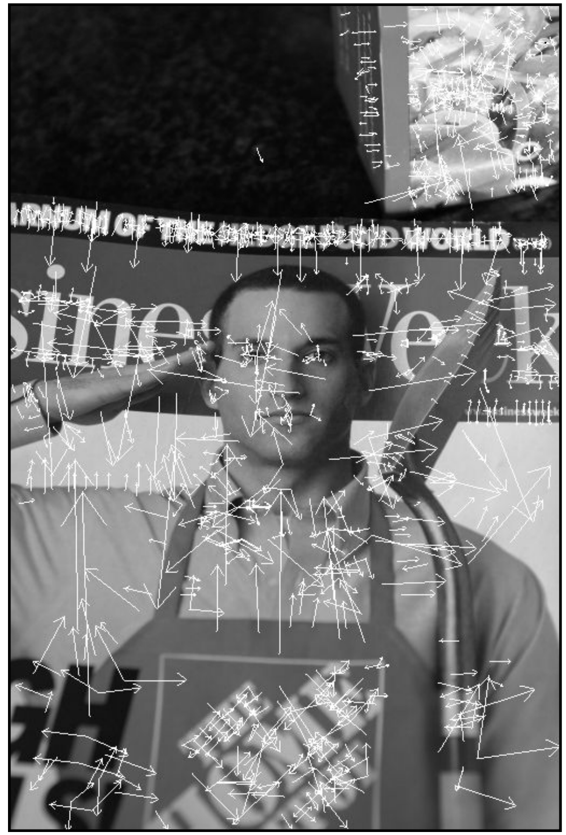


Step 1: Compute a Set of Keypoints

Source image I

⇒

1. Identify keypoints



Goals:

- Identify distinctive image locations
- Assign scale & orientation to each keypoint
- Should be able to detect some keypoint in images that vary in magnification, brightness, etc.

Step 1: Compute a Set of Keypoints

Keypoint: A location (x, y) in the source image, with an associated orientation & scale, that is "visually distinct" from its surroundings.

Input:

Image I

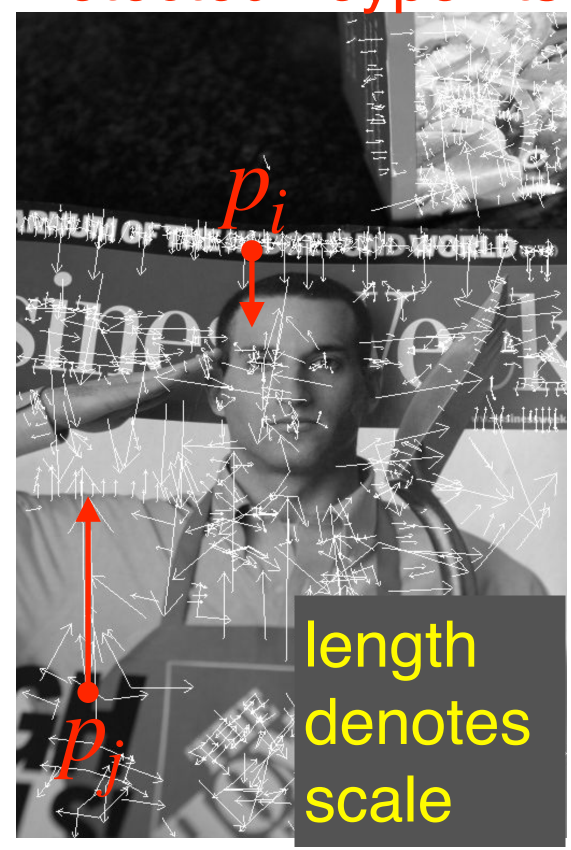
Output:

A set of *k* SIFT **keypoints**

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\rho_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

"Source" image I

Detected keypoints



Topic 11:

Feature Detection & Image Matching

- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

Computing SIFT Keypoints: Basic Steps

Source image I

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Step 1d
Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$

Location refinement

$\Leftarrow$

Step 1e

Prune set of extrema

keypoints = { all remaining (x'_i, y'_i, ρ'_i) }

Extremum pruning

$\Leftarrow$

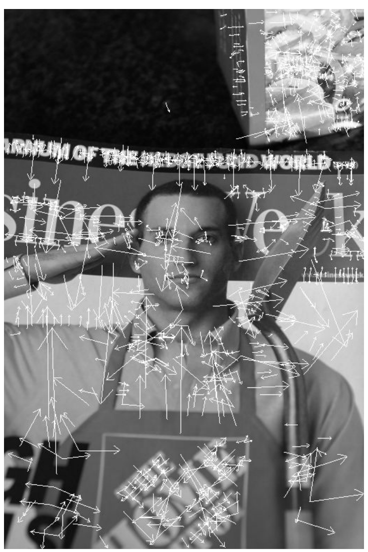
Step 1f

Assign orientation to extrema

$p_i = (x'_i, y'_i, \rho'_i, \theta_i)$

Orientation assign

$\Leftarrow$



Step 1a: Construct a Gauss-like Pyramid

Step 1a: Compute a pyramid of Gauss-filtered images organized into **octaves** of s + 1 images

each image is smoothed by a factor of k more than the image below it

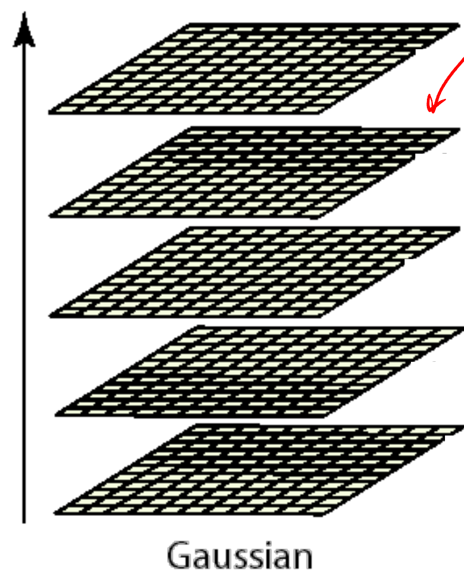
$I_s =$

$\vdots$

$I_2 =$

$I_1 =$

$I_0 =$



Step 1a: Construct a Gauss-like Pyramid

SIFT terminology

An **octave** is a set of Gauss-convolved images, $I_1, \ldots, I_S$ representing a doubling of the scale parameter σ between I_1 and I_S.

each image is smoothed by a factor of k more than the image below it

each octave contains s + 1 images

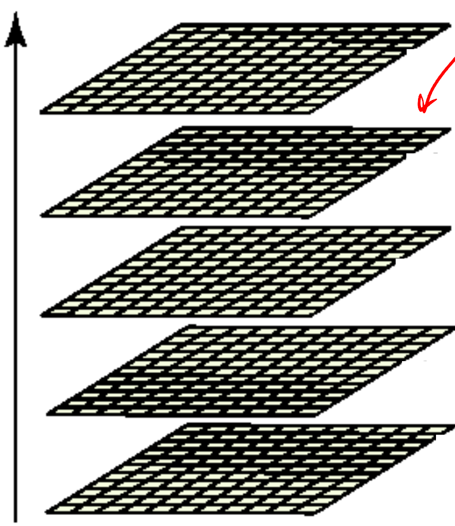
$$I_s = I * G_{k^s \sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

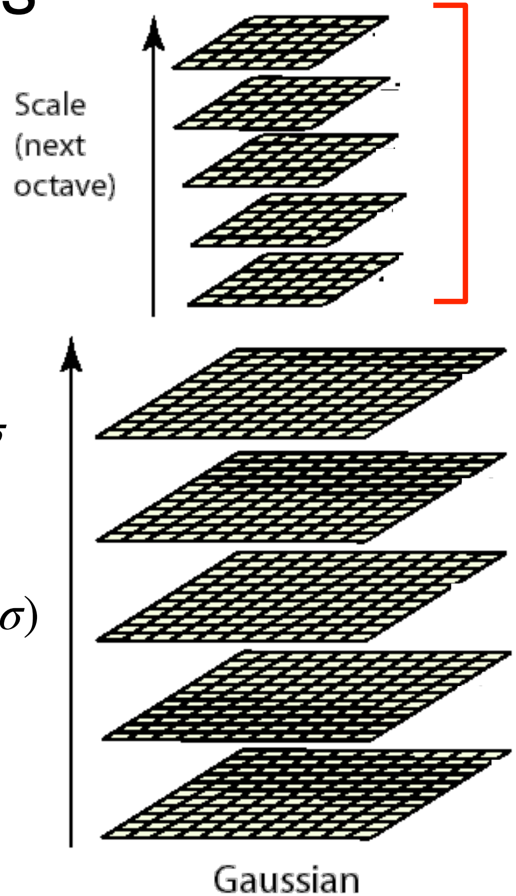
$$I_0 = I * G_{\sigma}$$



Gaussian

Step 1a: Construct a Gauss-like Pyramid

Step 1a: Compute a pyramid of Gauss-filtered images organized into **octaves** of s + 1 images



images in next octave are subsampled and stored at 1/2 resolution of previous octave

each octave contains s + 1 images

$$I_s = I * G_{k^s \sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

$$I_0 = I * G_{\sigma}$$

In practice:

$s = 3$ (3 images/octave)

$\sigma = 1.6$ (for first octave)

Computing SIFT Keypoints: Basic Steps

Source image I

$\Rightarrow$

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

$\Leftarrow$

Extremum pruning

Step 1e

Prune set of extrema

keypoints = { all remaining (x_i', y_i', ρ_i') }

$\Leftarrow$

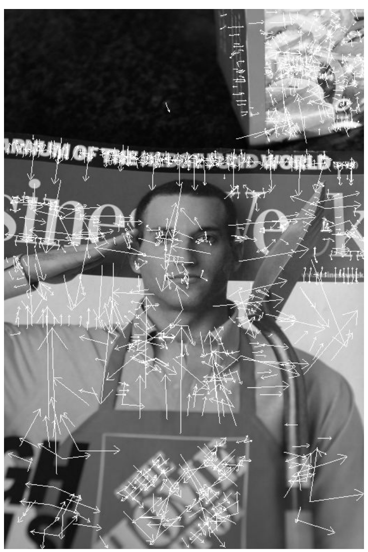
Orientation assign

Step 1f

Assign orientation to extrema

$p_i = (x_i', y_i', \rho_i', \theta_i)$

$\Leftarrow$



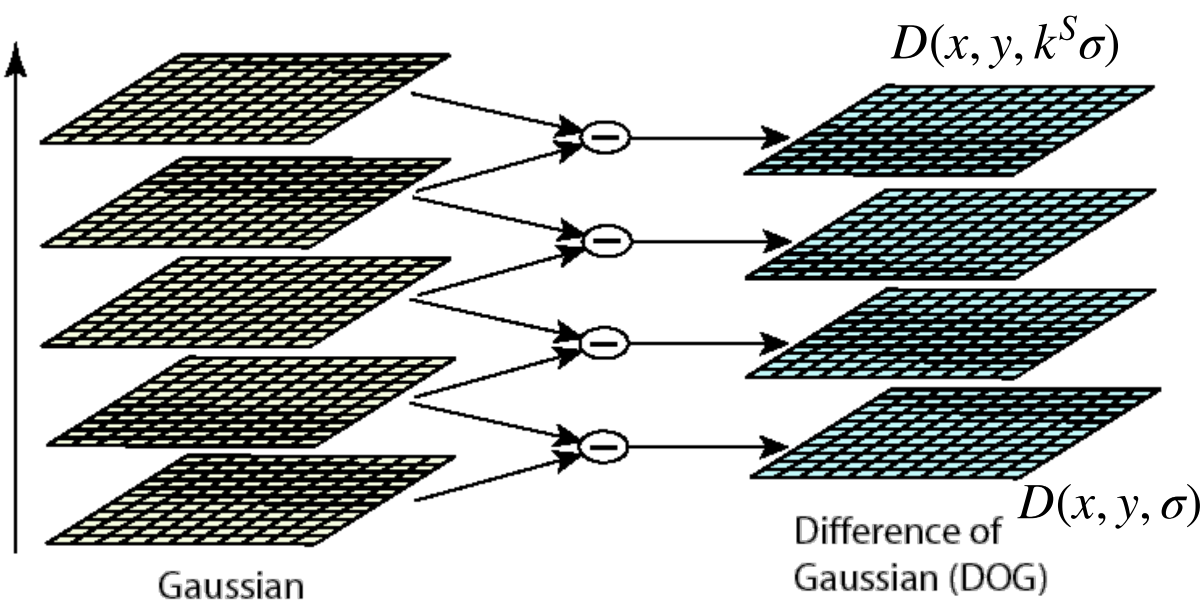
Step 1b: Compute Pyramid of DOG Images

Step 1b: Compute a pyramid of DOG-filtered images

$$\boxed{\begin{gathered} D(x,y,\rho) = I(x,y) * (G(x,y,k\rho) - G(xy,\rho)) \\ \text{for } \rho = \sigma, k\sigma, k^2\sigma, \ldots, k^{S-1}\sigma \end{gathered}}$$

each octave contains s + 1 images

$$\left[\begin{aligned} I_s &= I * G_{k^S\sigma} \\ &\vdots \\ I_2 &= I * G_{k(k\sigma)} \\ I_1 &= I * G_{k\sigma} \\ I_0 &= I * G_{\sigma} \end{aligned}\right.$$



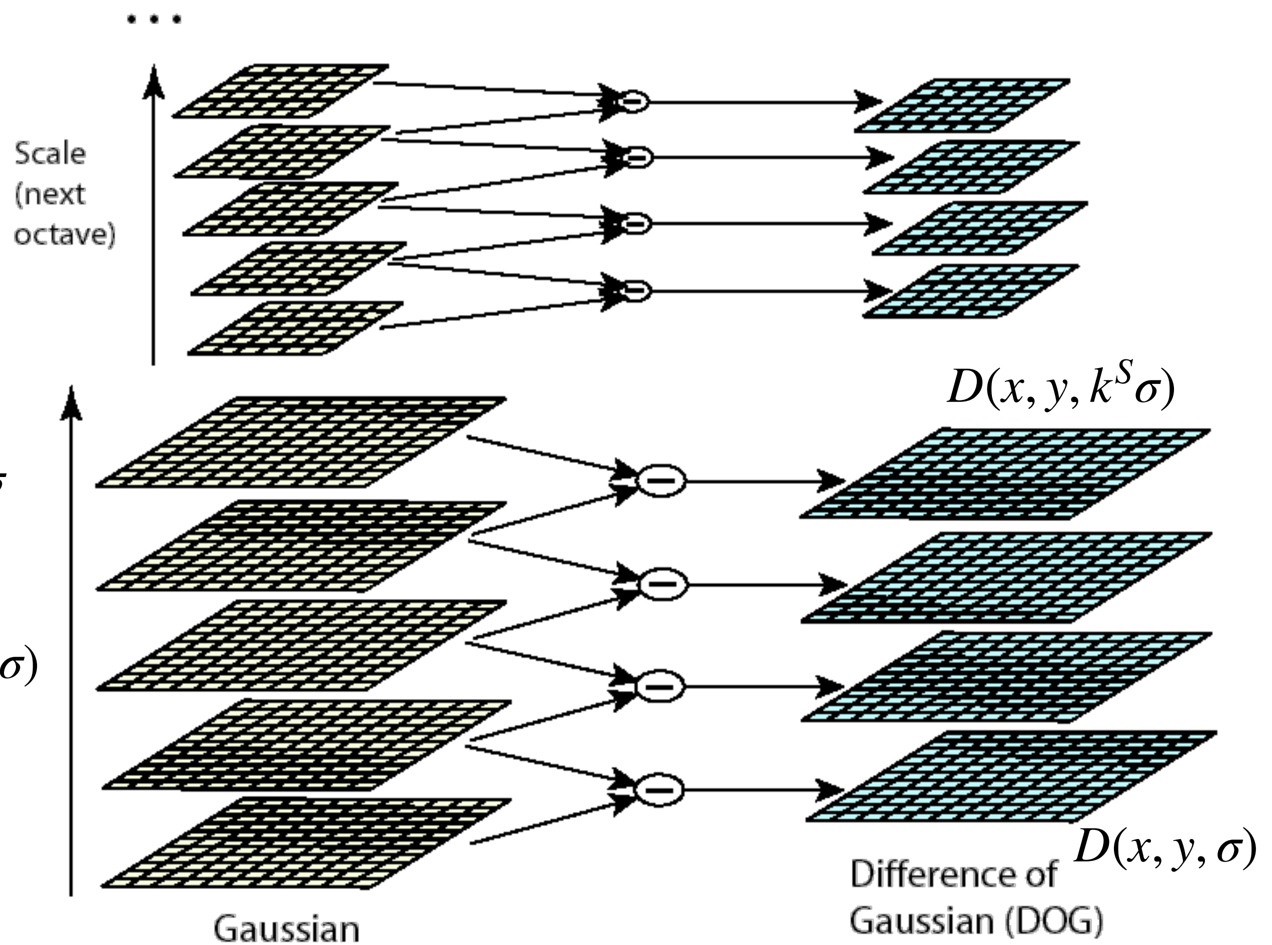
Step 1b: Compute Pyramid of DOG Images

Step 1b: Compute a pyramid of DOG-filtered images

$$D(x,y,\rho) = I(x,y) * (G(x,y,k\rho) - G(xy,\rho))$$
for $\rho = \sigma, k\sigma, k^2\sigma, \ldots, k^{S-1}\sigma$

each octave contains s + 1 images

$$I_s = I * G_{k^S\sigma}$$
$$\vdots$$
$$I_2 = I * G_{k(k\sigma)}$$
$$I_1 = I * G_{k\sigma}$$
$$I_0 = I * G_{\sigma}$$



Reminder: Difference-of-Gaussian Filtering

$$I * G_{\rho}$$

Reminder: Difference-of-Gaussian Filtering

$$I * G_{k\rho}$$

Reminder: Difference-of-Gaussian Filtering

Difference of two Gaussian-smoothed versions of I:

$$I * G_{k\rho} -$$
$$I * G_{\rho} =$$
$$I * (G_{k\rho} - G_{\rho})$$

(just the difference between two Gaussian masks)

But we know that

$$G_{k\rho} - G_{\rho} =$$
$$k\rho(k\rho - \rho)\,\nabla^2 G_{\rho}$$

$$\Downarrow$$

$$\Longrightarrow \quad D(x, y, \rho) = \left[\nabla^2 (I * G_{\rho}) \right] \rho^2 k(k-1)$$

Computing SIFT Keypoints: Basic Steps

Source image I

⇒

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

⇒

DOG pyramid

Step 1b

Build DOG pyramid

⇒

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$

⇐

Extremum pruning

Step 1e

Prune set of extrema

keypoints = { all remaining (x'_i, y'_i, ρ'_i) }

⇐

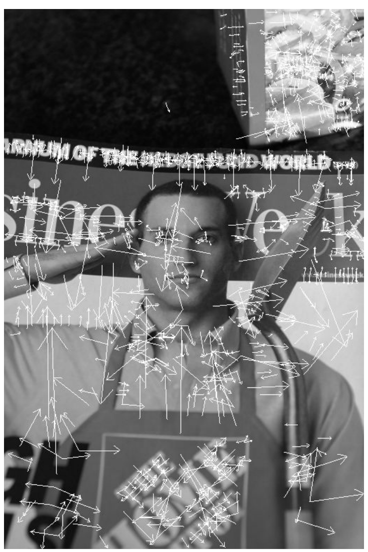
Orientation assign

Step 1f

Assign orientation to extrema

$p_i = (x'_i, y'_i, \rho'_i, \theta_i)$

⇐



Step 1c: Detecting DOG Extrema

Step 1c (Extremum detection): Find all pixels that correspond to extrema of D(x, y, ρ)

Extrema of the image Laplacian are readily distinguishable from their surroundings. There are usually just a few thousand in each pyramid.

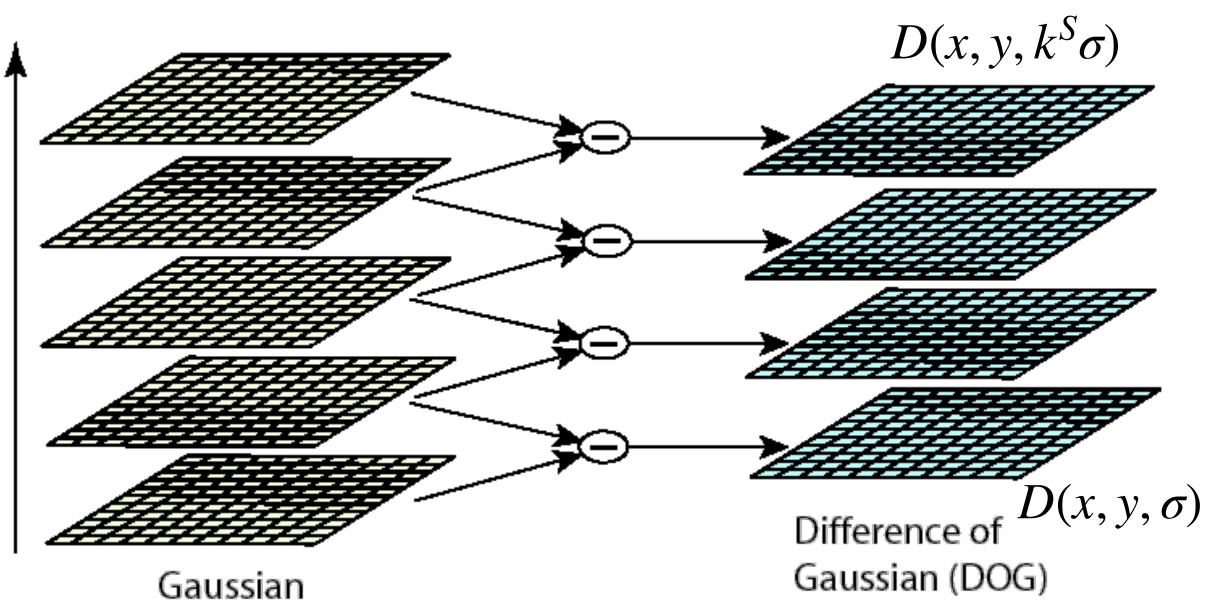
$$I_s = I * G_{k^S\sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

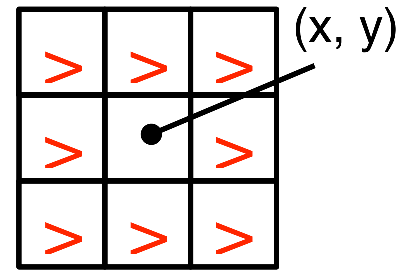
$$I_0 = I * G_{\sigma}$$



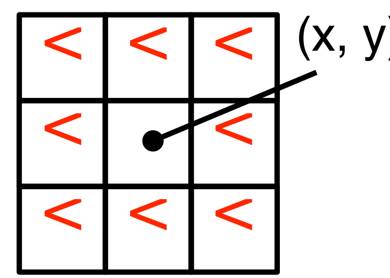
The Difference-Of-Gaussians (DOG) Filter

Finding local extrema in a single image $D(x, y, \rho)$

- minimum at (x, y) if $D(x,y,\rho) <$ all neighbours



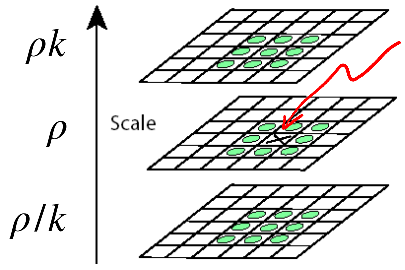
- maximum if $D(x,y,\rho) >$ all neighbours



Step 1c: Detecting DOG Extrema

Step 1c (Extremum detection): Find all pixels that correspond to extrema of $D(x, y, \rho)$

finding extrema in a "stack" of images



must also be < than all neighbours in adjacent scales

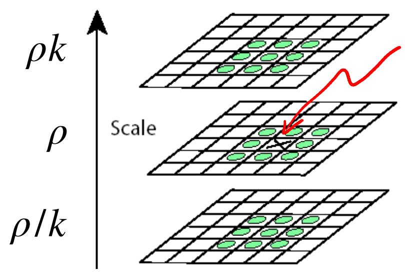
or

must also be > than all neighbours in adjacent scales

Step 1c: Detecting DOG Extrema

Step 1c (Extremum detection): Find all pixels that correspond to extrema of D(x, y, ρ)

finding extrema in a “stack” of images



must also be < than all neighbours in adjacent scales

or

must also be > than all neighbours in adjacent scales

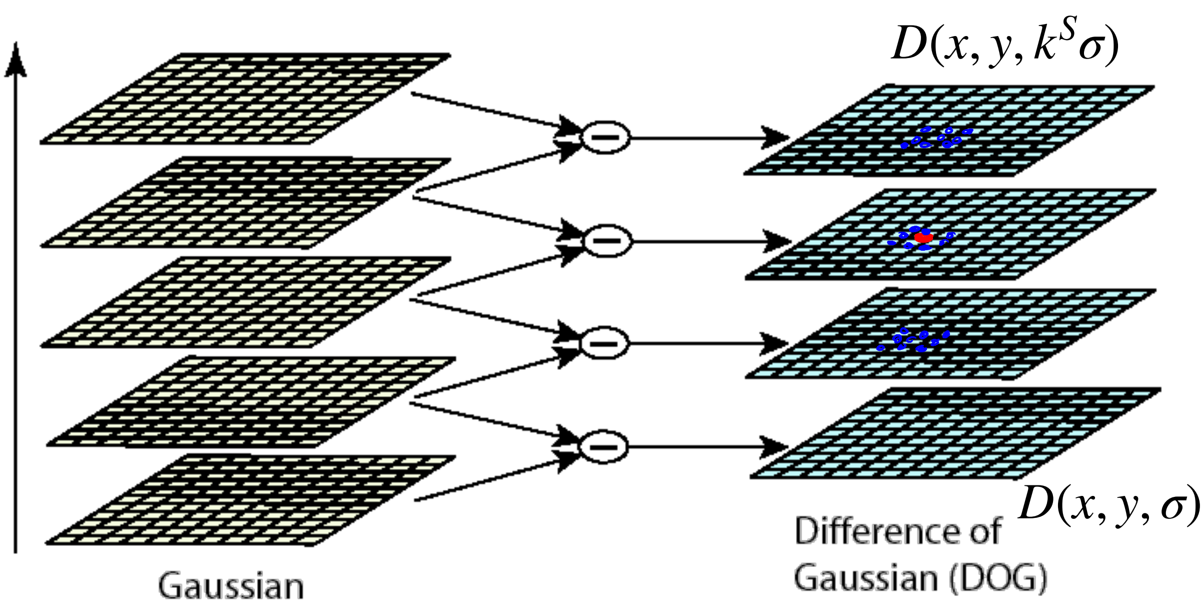
$$I_s = I * G_{k^S \sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

$$I_0 = I * G_{\sigma}$$



Step 1c: Detecting DOG Extrema: Algorithm

Step 1c (Extremum detection): Find all pixels that correspond to extrema of D(x, y, ρ)

For each (x, y, ρ), check whether D(x, y, ρ) is greater than (or smaller than) all of its neighbours in current scale and adjacent scales above & below.

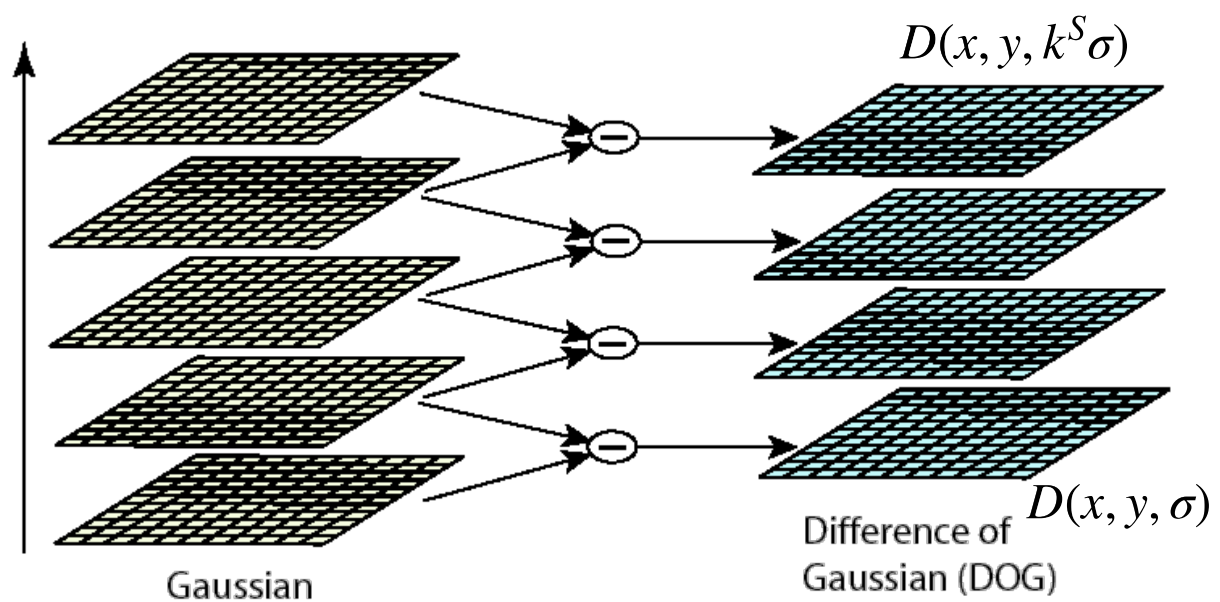
$$I_s = I * G_{k^S\sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

$$I_0 = I * G_{\sigma}$$



Step 1c: SIFT Keypoints = DOG Extrema

Step 1c (Extremum detection): Find all pixels that correspond to extrema of D(x, y, ρ)

An extremum detected at D(x, y, ρ) defines the keypoint (x, y, ρ).

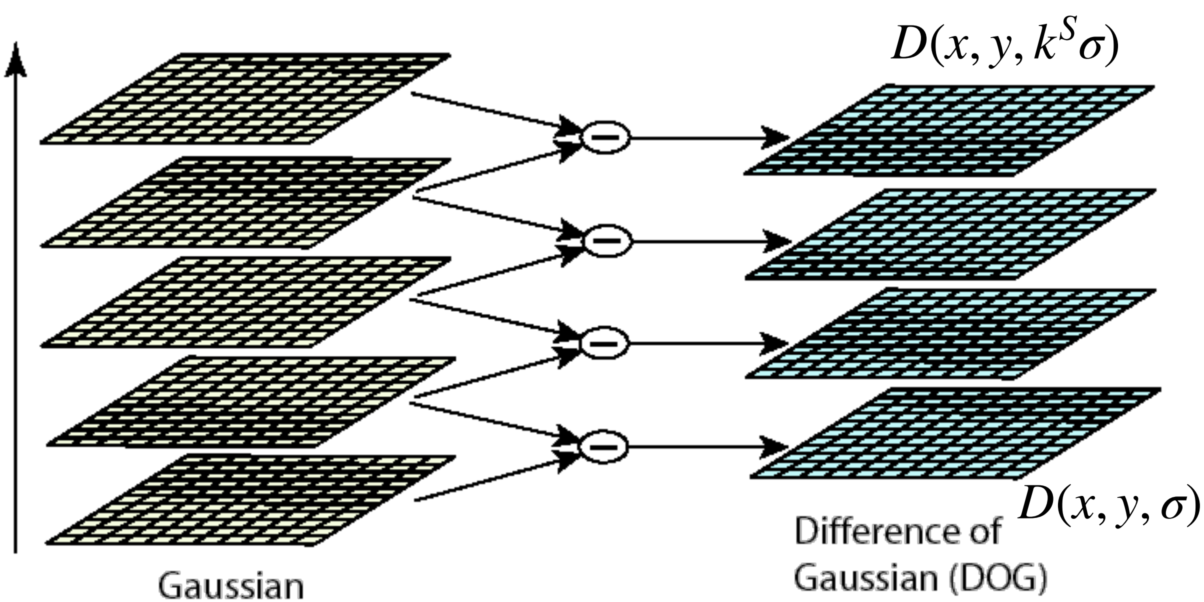
$$I_s = I * G_{k^S\sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

$$I_1 = I * G_{k\sigma}$$

$$I_0 = I * G_{\sigma}$$



Computing SIFT Keypoints: Basic Steps

Source image I

$\Rightarrow$

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$

$\Leftarrow$

Extremum pruning

Step 1e

Prune set of extrema

keypoints = { all remaining (x'_i, y'_i, ρ'_i) }

$\Leftarrow$

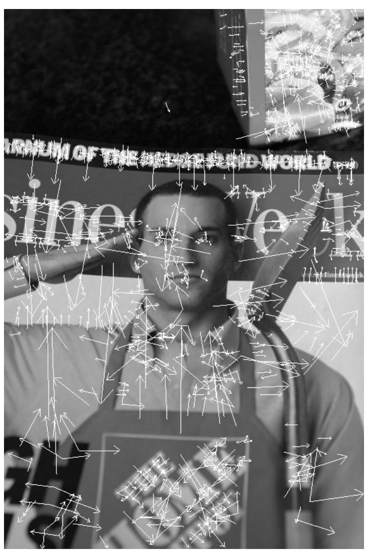
Orientation assign

Step 1f

Assign orientation to extrema

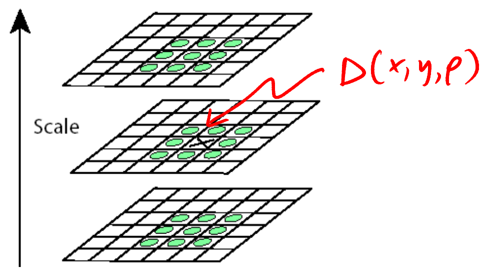
$p_i = (x'_i, y'_i, \rho'_i, \theta_i)$

$\Leftarrow$



Step 1d: Refining Location of Extrema

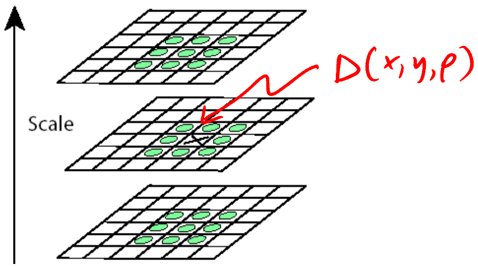
Step 1d (Extremum localization) Refine the location of detected extrema through a quadratic least-squares fit.



- The original SIFT method/paper just uses the pixel location of the extrema as the location of the key point
- Revised method finds the subpixel location interpolated location of the extrema using 2nd order Taylor expansion of D at (x, y, ρ)
- This improves the results when matching significantly

Step 1d: Refining Location of Extrema

1.



2nd order Taylor expansion of *D* at (*x*, *y*, *ρ*):

$$D(\Delta\vec{x}) = D(\vec{x}) + \left(\frac{\partial D}{\partial \vec{x}}\right)^T \cdot \Delta\vec{x}$$
$$+\frac{1}{2}(\Delta\vec{x})^T \cdot \frac{\partial^2 D}{\partial \vec{x}^2} \cdot (\Delta\vec{x})$$

2. Take derivatives with respect to $\Delta\vec{x}$

$$\frac{\partial D}{\partial(\Delta\vec{x})} = \left(\frac{\partial D}{\partial \vec{x}}\right)^T + \left(\frac{\partial^2 D}{\partial \vec{x}^2}\right)(\Delta\vec{x})$$

3. Extremum ⇔ derivative is zero ⇒ solve for $\frac{\partial D}{\partial \Delta\vec{x}} = 0$

$$(\Delta\vec{x}) = \left(\frac{\partial^2 D}{\partial \vec{x}^2}\right)^{-1} \cdot \left(\frac{\partial D}{\partial \vec{x}}\right)$$

Computing SIFT Keypoints: Basic Steps

Source image I

$\Rightarrow$

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

$\Leftarrow$

Extremum pruning

Step 1e

Prune set of extrema

keypoints = { all remaining (x_i', y_i', ρ_i') }

$\Leftarrow$

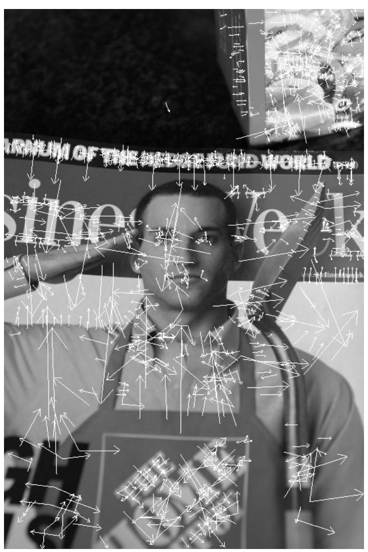
Orientation assign

Step 1f

Assign orientation to extrema

$p_i = (x_i', y_i', \rho_i', \theta_i)$

$\Leftarrow$



Step 1e: Pruning "Insignificant" Extrema

Step 1e (Extremum pruning): Prune all extrema that are weak or that correspond to edges

Condition for detecting a "strong" extremum (x_i', y_i', ρ_i')

$$|D(x_i', y_i', \rho_i')| = \text{large}$$

in practice, > 0.03

assumes image I has pixel intensities in the range [0, 1]

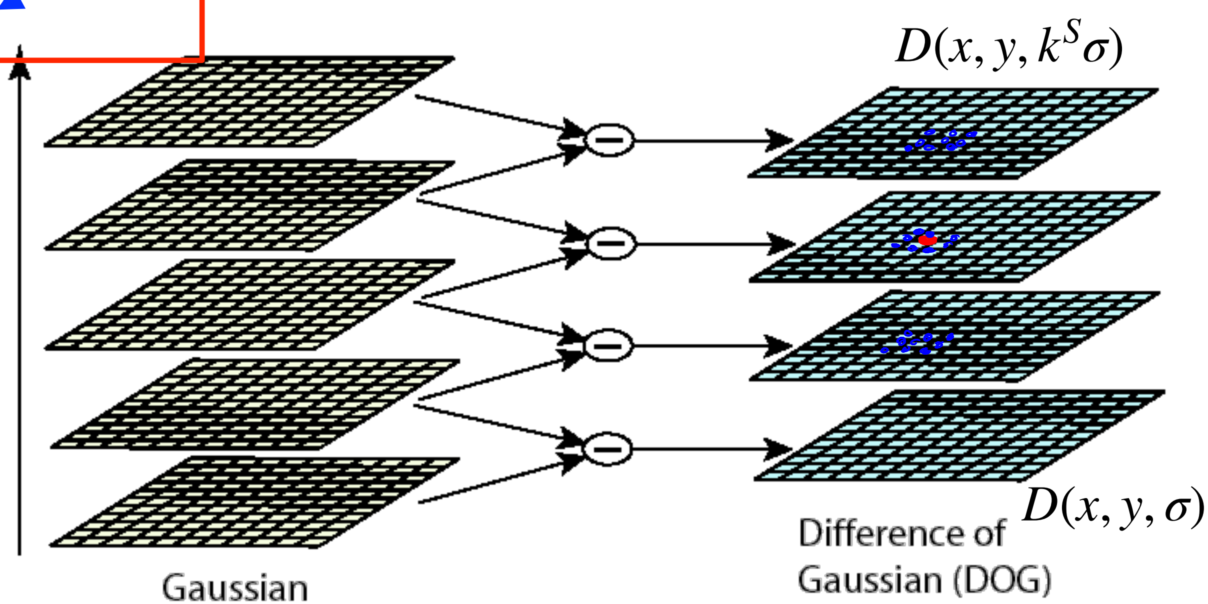
$$I_s = I * G_{k^S\sigma}$$

$$\vdots$$

$$I_2 = I * G_{k(k\sigma)}$$

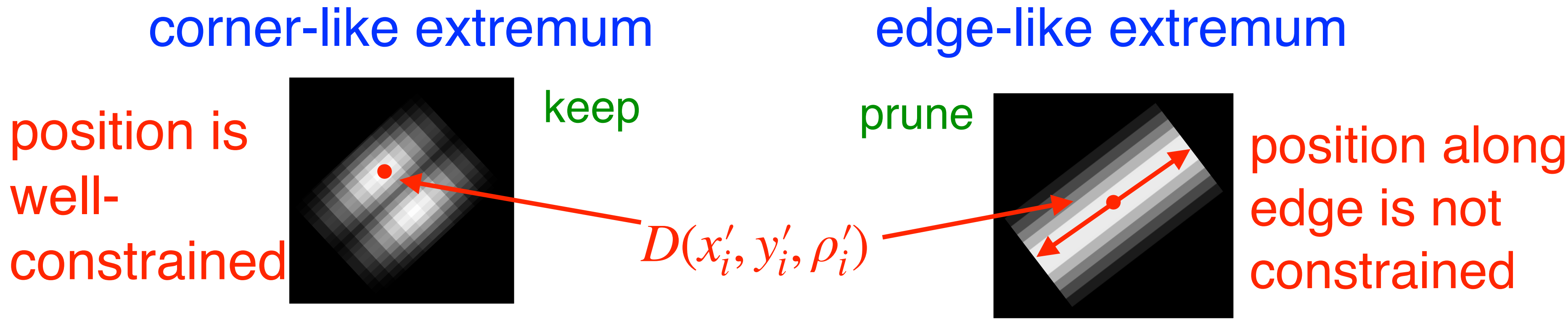
$$I_1 = I * G_{k\sigma}$$

$$I_0 = I * G_{\sigma}$$

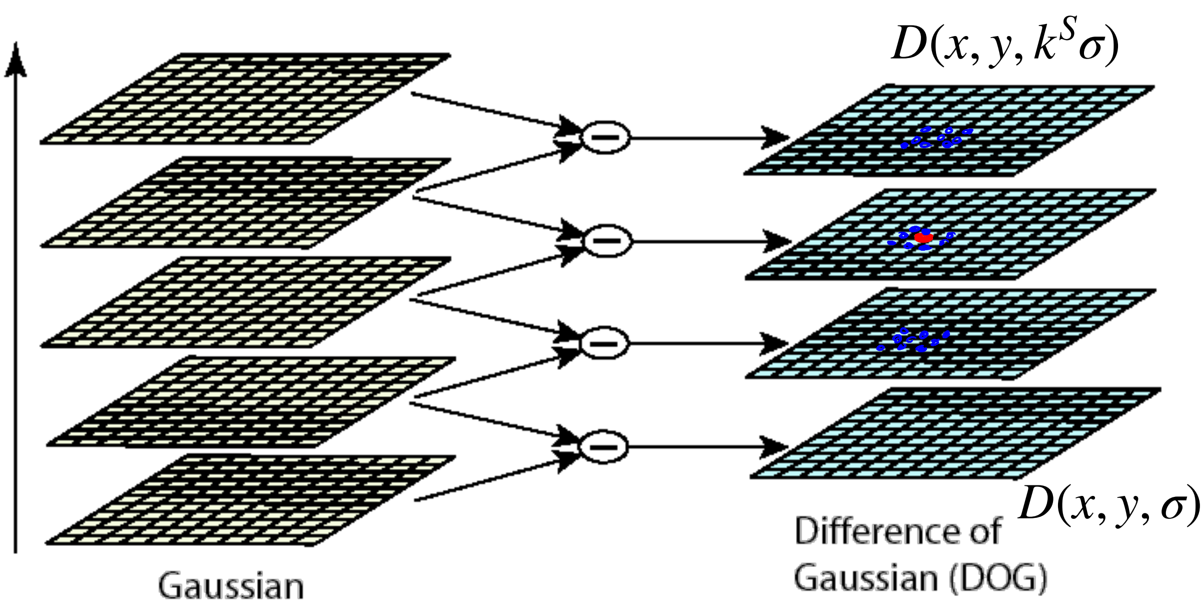


Step 1e: Pruning “Insignificant” Extrema

Step 1e (Extremum pruning): Prune all extrema that are weak or that correspond to edges



$$I_s = I * G_{k^S\sigma}$$
$$\vdots$$
$$I_2 = I * G_{k(k\sigma)}$$
$$I_1 = I * G_{k\sigma}$$
$$I_0 = I * G_{\sigma}$$

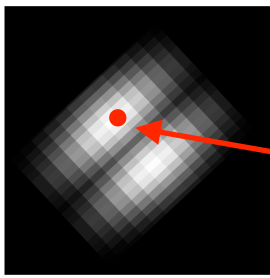


Step 1e: Pruning "Insignificant" Extrema

Step 1e (Extremum pruning): Prune all extrema that are weak or that correspond to edges

corner-like extremum

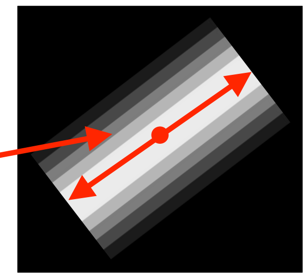
position is well-constrained



keep

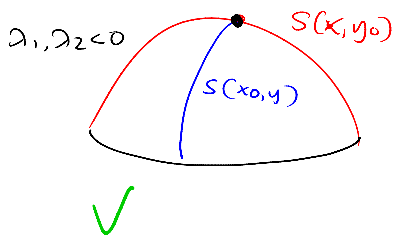
edge-like extremum

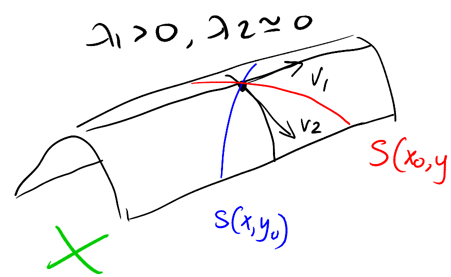
prune



position along edge is not constrained

$D(x_i', y_i', \rho_i')$



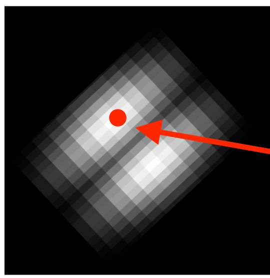


Step 1e: Pruning "Insignificant" Extrema

Step 1e (Extremum pruning): Prune all extrema that are weak or that correspond to edges

corner-like extremum

position is well-constrained

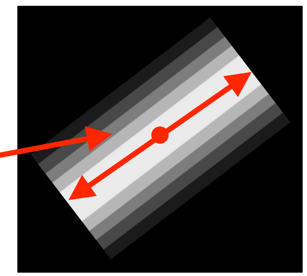


keep

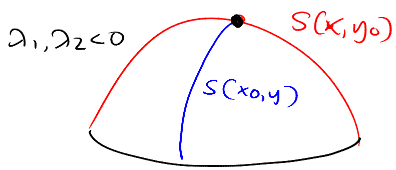
$D(x_i', y_i', \rho_i')$

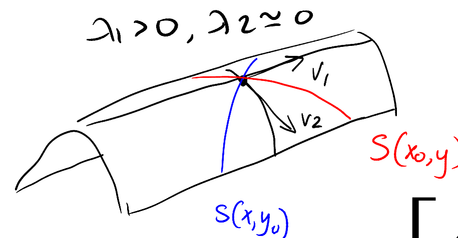
edge-like extremum

prune



position along edge is not constrained





Compute hessian H of

$D(x, y, \rho_i')$ at $(x, y) = (x_i', y_i')$

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}$$

Prune if $\dfrac{\mathrm{Tr}^2(H)}{\mathrm{Det}(H)} > \left(\dfrac{r+1}{r}\right)^2$ where for SIFT $r = 10$

Computing SIFT Keypoints: Basic Steps

Source image I

$\Rightarrow$

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x_i', y_i', \rho_i')$

Location refinement

$\Leftarrow$

Step 1e

Prune set of extrema

keypoints = { all remaining (x_i', y_i', ρ_i') }

Extremum pruning

$\Leftarrow$

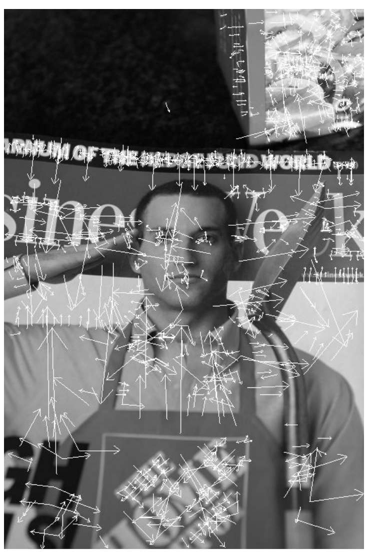
Step 1f

Assign orientation to extrema

$p_i = (x_i', y_i', \rho_i', \theta_i)$

Orientation assign

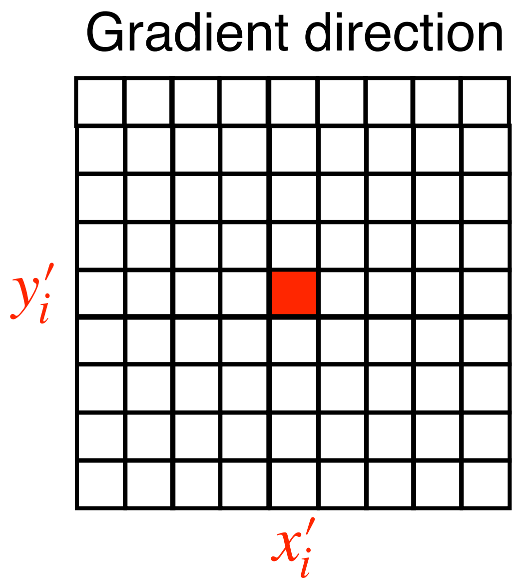
$\Leftarrow$

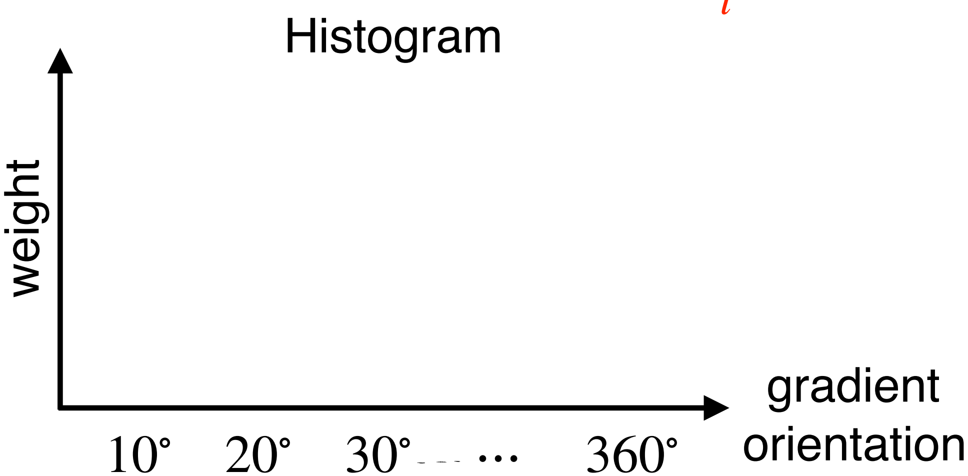


Step 1f: Keypoint Orientation Assignment

Assigning an orientation θ_i to keypoint (x'_i, y'_i, ρ'_i) :

$I * G_{\rho'_i}$





A. Compute smoothed image

B. Compute gradient magnitude & orientation in neighbourhood of (x'_i, y'_i) in $I * G_{\rho'_i}$

C. Compute histogram of orientations

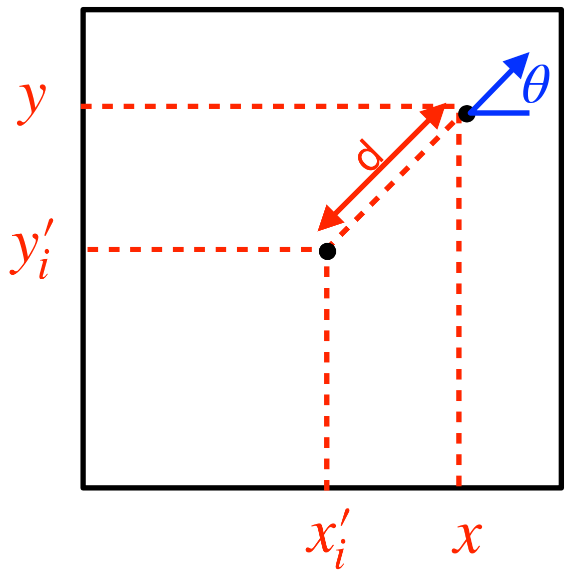
D. Assigned orientation θ_i = highest peak in histogram

Step 1f: Keypoint Orientation Assignment

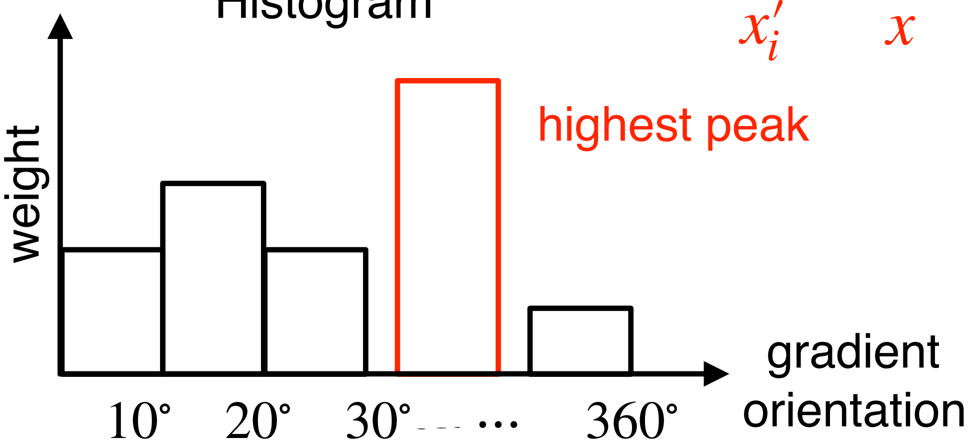
Computing Histogram of Orientations

$I * G_{\rho_i'}$

Gradient direction



Histogram



- Orientations divided into 36 bins (one every 10 degrees).
- Pixel (x, y) contributes to the bin corresponding to the gradient orientation θ at (x, y).
- Contribution to the bin is equal to
  $$|\nabla I(x, y)| \cdot G_{1.5\rho_i'}(d)$$
- Total bin weight = sum of contributions from all pixels

Computing SIFT Keypoints: Basic Steps

Source image I

$\Rightarrow$

Gauss-pyramid

Step 1a

Build pyramid of Gauss-smoothed images

$\Rightarrow$

DOG pyramid

Step 1b

Build DOG pyramid

$\Rightarrow$

DOG extrema

Step 1c

Locate extrema of DOG pyramid

(x_i, y_i, ρ_i)

Location refinement

Step 1d

Refine location of DOG extrema

$(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$

$\Leftarrow$

Extremum pruning

Step 1e

Prune set of extrema

keypoints = { all remaining (x'_i, y'_i, ρ'_i) }

$\Leftarrow$

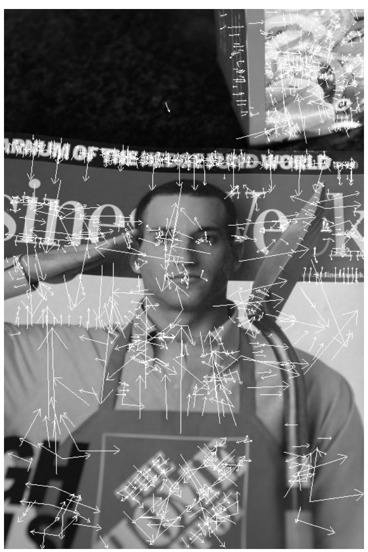
Orientation assign

Step 1f

Assign orientation to extrema

$p_i = (x'_i, y'_i, \rho'_i, \theta_i)$

$\Leftarrow$



Topic 11:

Feature Detection & Image Matching

- Introduction to the image matching problem
- Image matching using SIFT features
- The SIFT feature detector
- The SIFT descriptor

The SIFT Keypoints

Keypoint: A location (x, y) in the source image, with an associated orientation & scale, that is “visually distinct” from its surroundings.

Input:

Image I

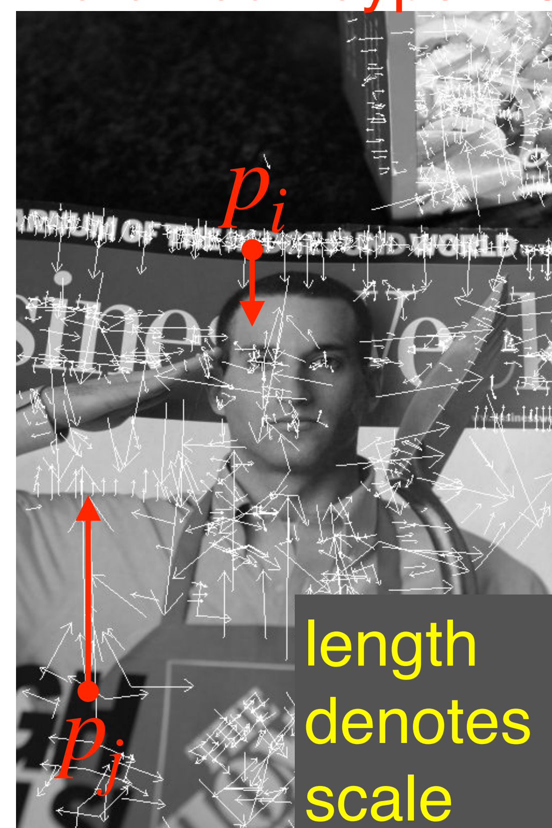
Output:

A set of *k* SIFT **keypoints**

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\rho_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

“Source” image I

Detected keypoints



The SIFT Feature Vectors

Feature vector (of a keypoint p_i): A vector of fixed length that represents the image patch centred at p_i.

Input:

Image I

Output:

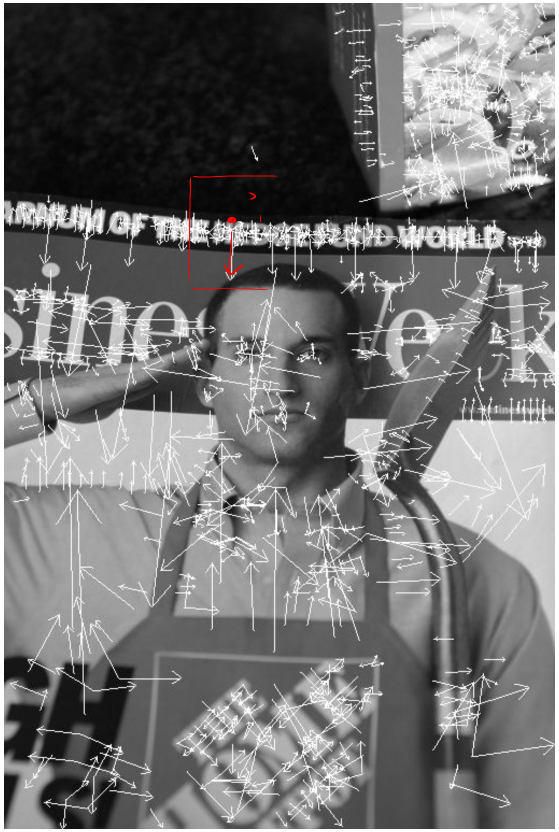
A set of k **keypoints**

$$p_i = (\underbrace{x_i, y_i}_{\text{location}}, \underbrace{\rho_i}_{\text{scale}}, \underbrace{\theta_i}_{\text{orientation}})$$

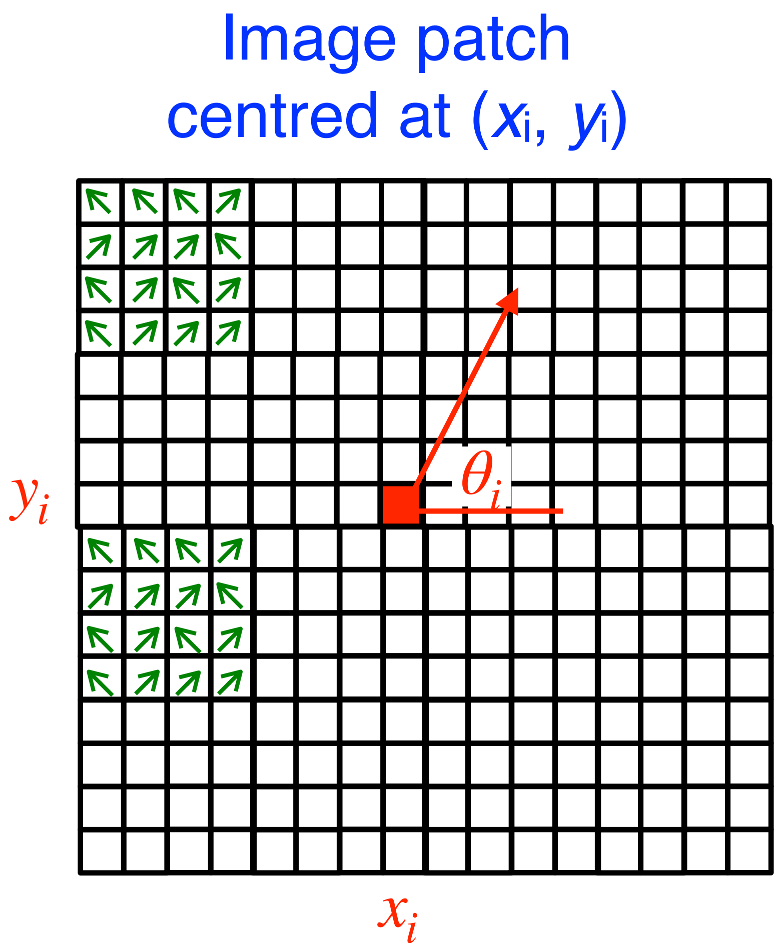
A set of k **feature vectors**

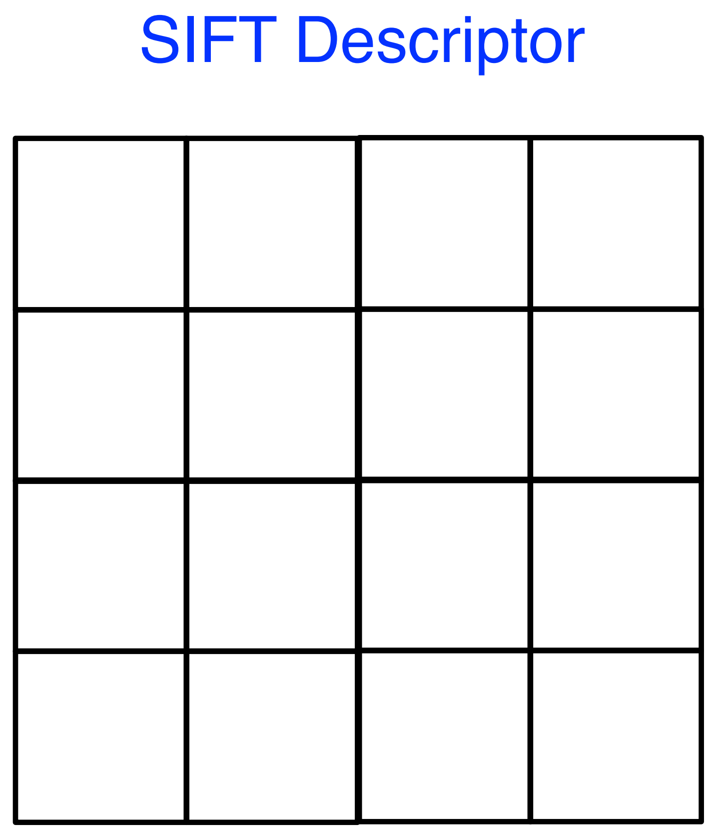
$$f_i = \underbrace{\boxed{}\boxed{}\boxed{\cdots}\boxed{}\boxed{}}_{\text{describes the patch centred at } p_i}$$

Detected keypoints



Building the SIFT Descriptor



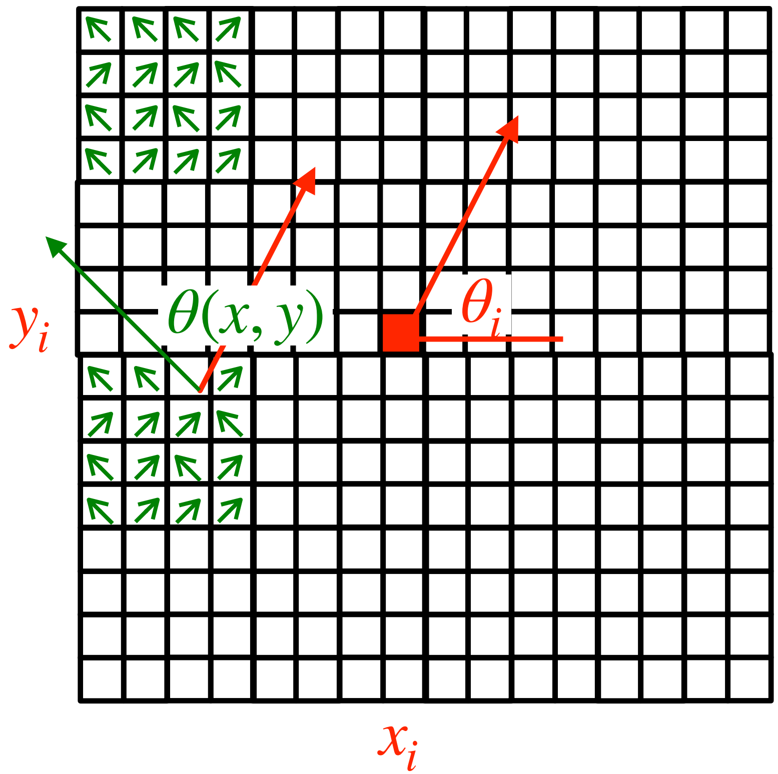


1. Compute gradients in 16 x 16 pixel patch of image $I * G_{\sigma_i}$ centred at (x_i, y_i)

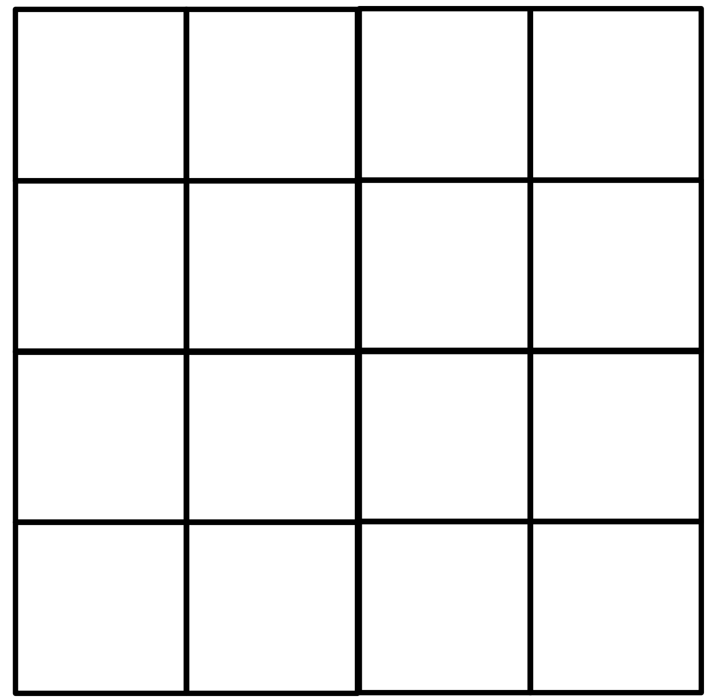
Gaussian-smoothed image at scale of the keypoint

Building the SIFT Descriptor

Image patch centred at (x_i, y_i)



SIFT Descriptor



1. Compute gradients in 16 x 16 pixel patch of image $I * G_{\sigma_i}$ centred at (x_i, y_i)

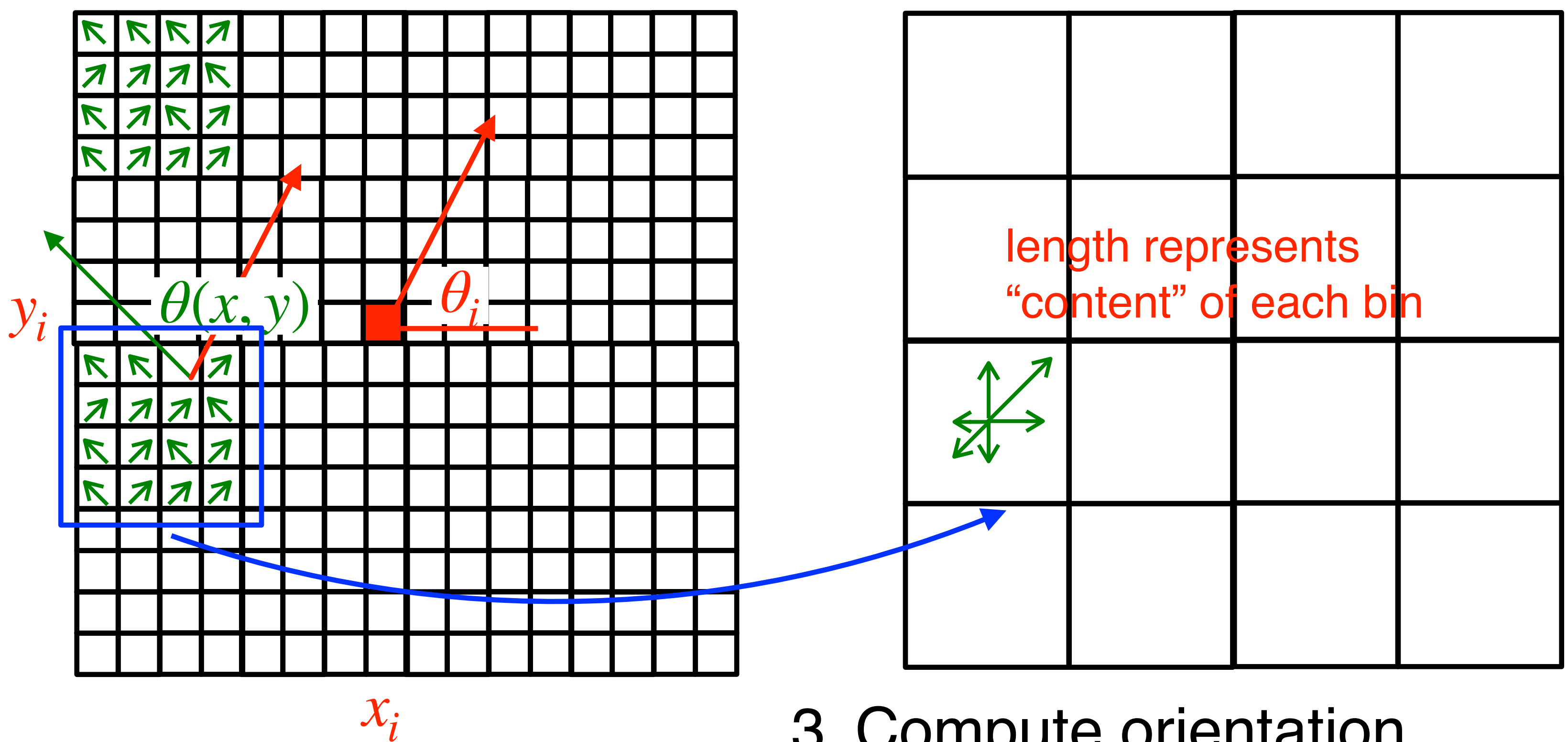
2. Compute gradient orientation relative to keypoint orientation

$$\theta(x, y) = \tan^{-1}\left[\frac{\partial(I * G_{\sigma_i})}{\partial y} \Big/ \frac{\partial(I * G_{\sigma_i})}{\partial x}\right] - \theta_i$$

Building the SIFT Descriptor

Image patch centred at (x_i, y_i)

SIFT Descriptor

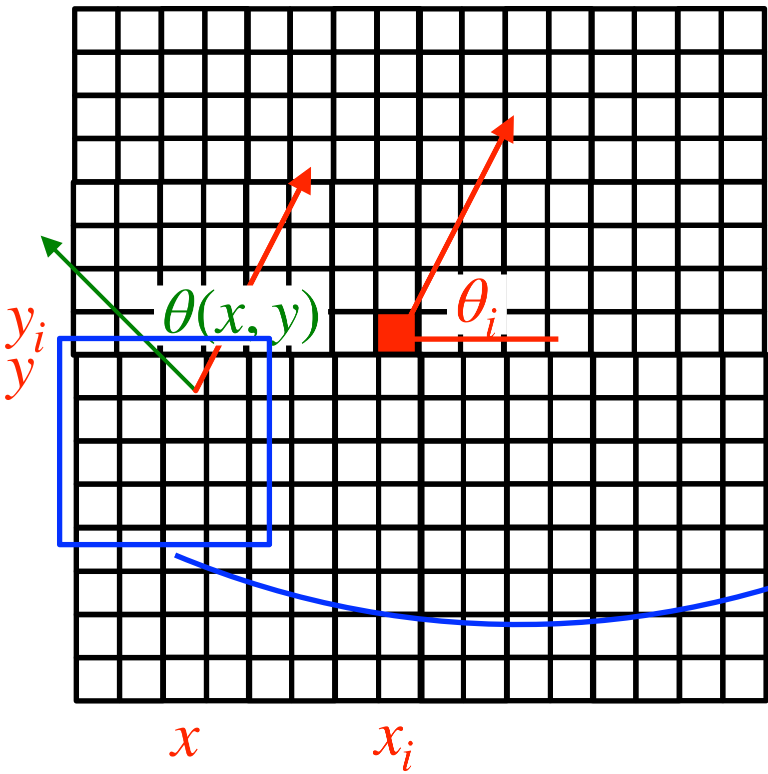


1. Compute gradients
2. Compute relative gradient orientations
3. Compute orientation histogram of each 4x4 pixel block

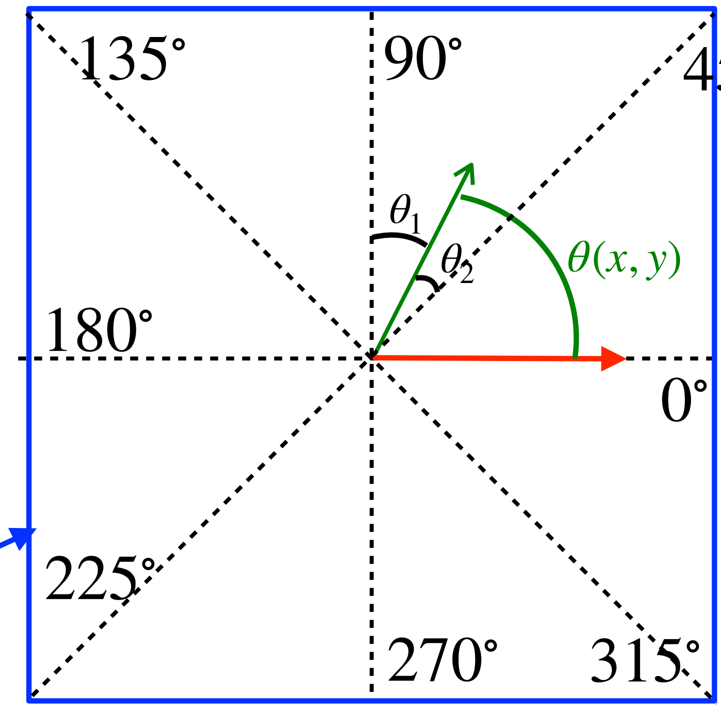
Histogram contains 8 bins, each covering 45°

The 4x4 Orientation Histogram

Image patch centred at (x_i, y_i)



The Orientation Histogram of a 4x4 pixel block



Weight of (x, y) :

$$w = G_{\frac{\sigma_i}{2}}\left(\|(x - x_i, y - y_i)\|\right)$$

⇒ pixels closer to keypoint centre have higher weight

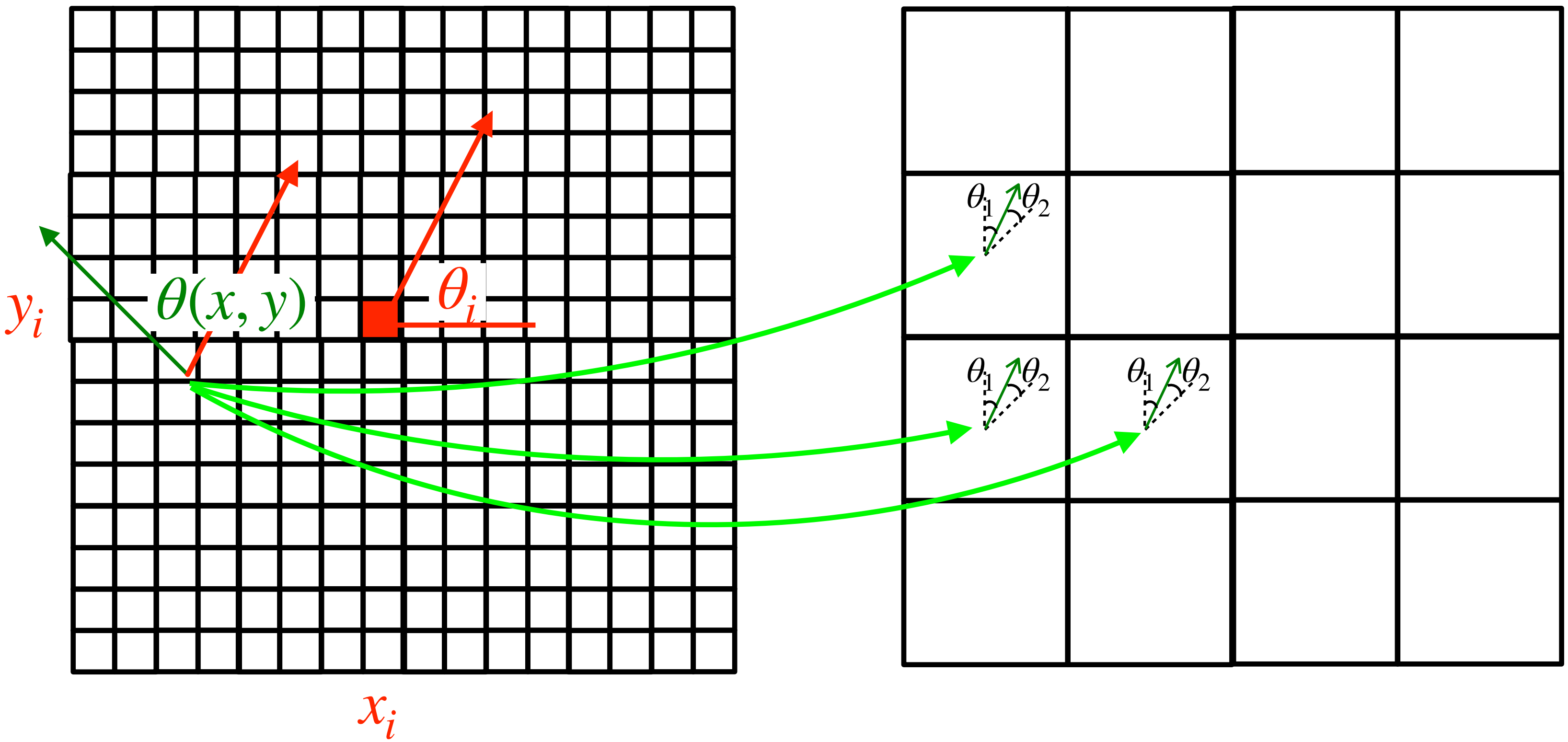
Total contribution of (x, y) to the orientation histograms:

$$C(x, y) = w \cdot \underbrace{\|\nabla I * G_{\sigma_i}(x, y)\|}_{\text{gradient magnitude of smoothed image}}$$

Building the SIFT Descriptor

Image patch
centred at (x_i, y_i)

SIFT Descriptor



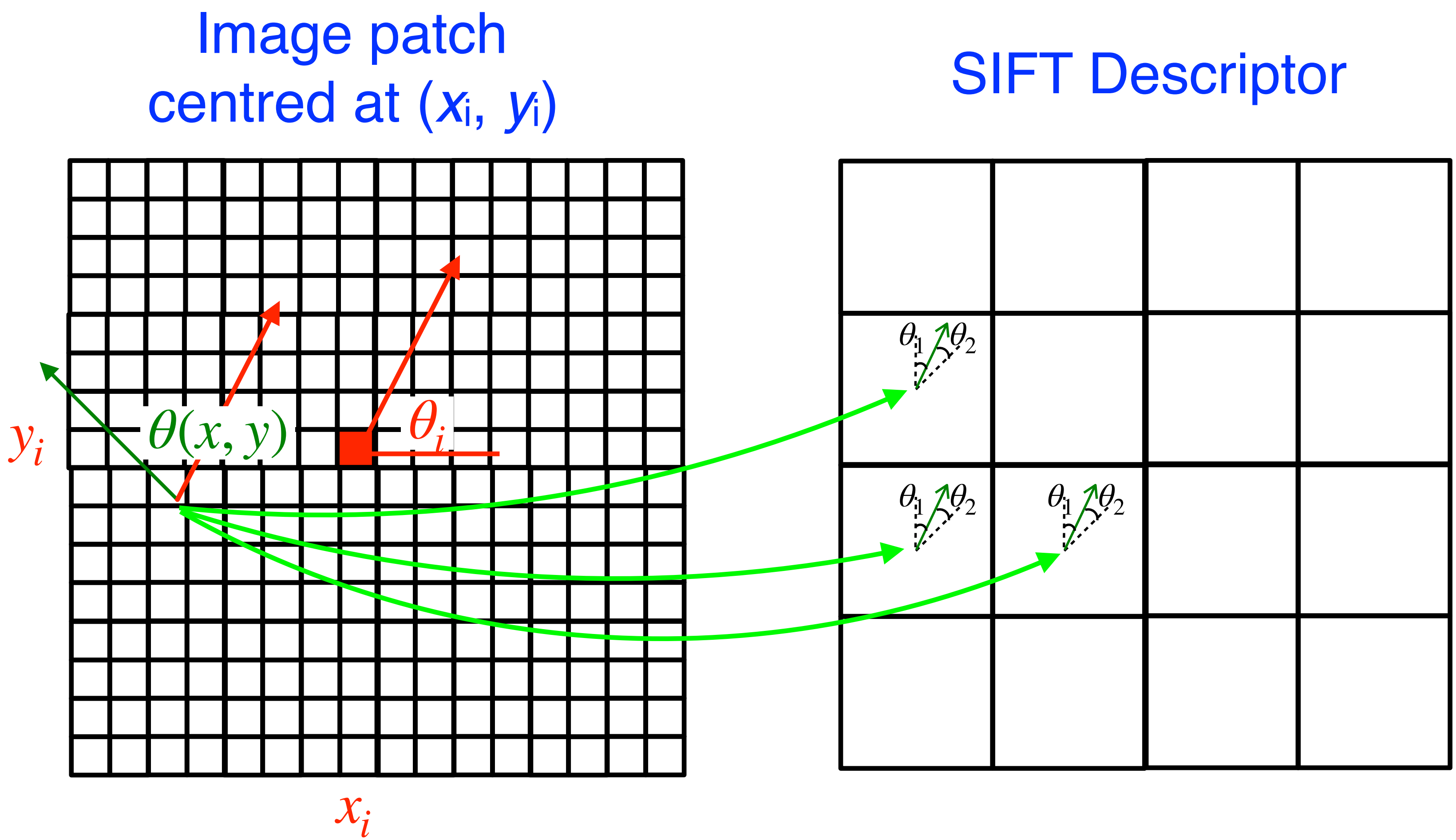
Contribution spread across 2 closest orientations & 3 closest histograms

⇒ no abrupt changes in histogram if keypoint centre displaced by 3 - 4 pixels

Total contribution of (x, y) to the orientation histograms:

$$C(x, y) = w \cdot \|\nabla I * G_{\sigma_i}(x, y)\|$$

Building the SIFT Descriptor



Contribution spread across 2 closest orientations & 3 closest histograms

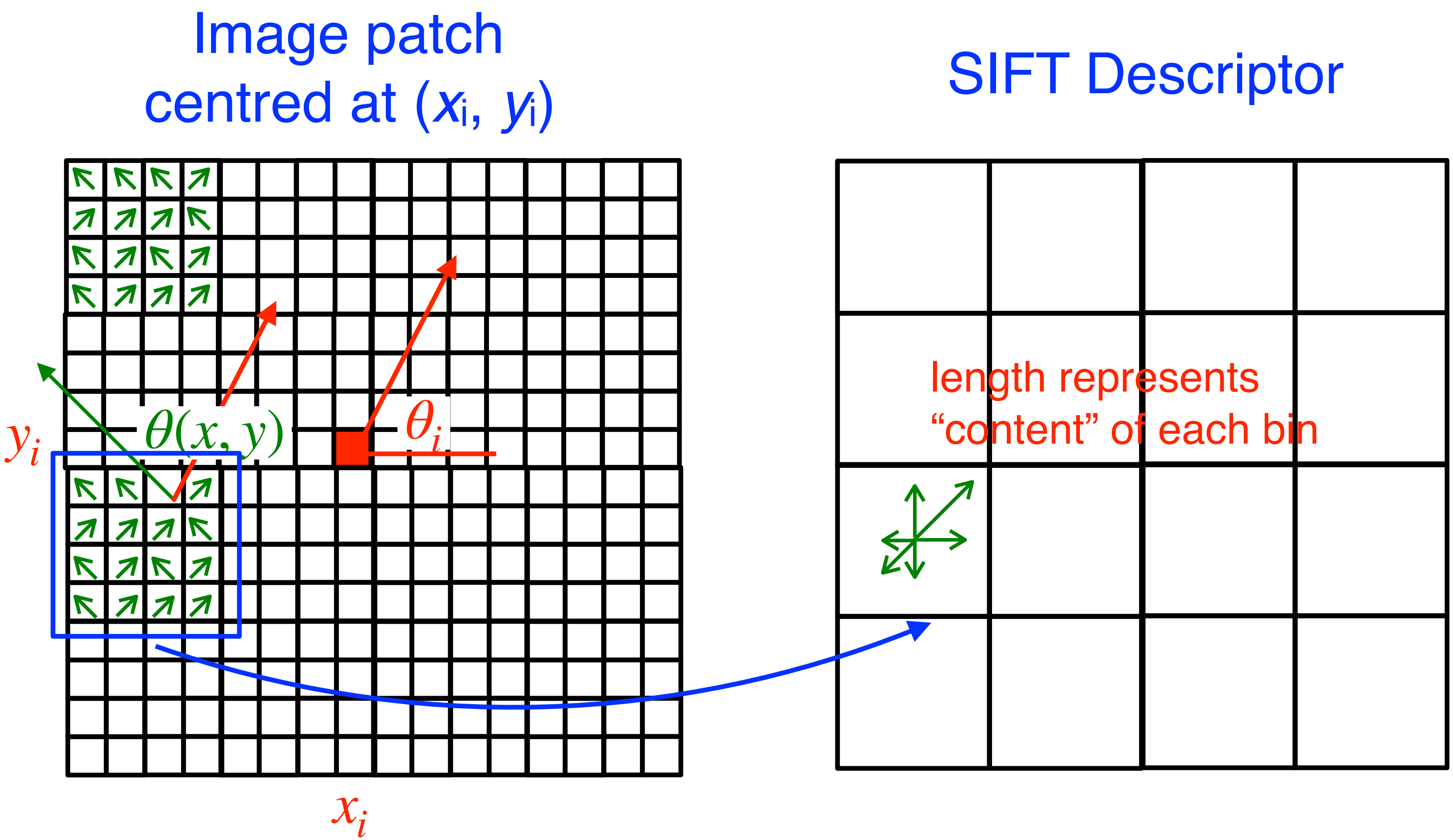
$\Rightarrow$ no abrupt changes in histogram if keypoint centre displaced by 3 - 4 pixels

Example: fraction allocated to

orientation 1: $\dfrac{\theta_1}{\theta_1 + \theta_2}$

orientation 2: $\dfrac{\theta_2}{\theta_1 + \theta_2}$

Building the SIFT Descriptor



1. Compute gradients
2. Compute relative gradient orientations
3. Define an “accumulator” variable for each of the 8 orientations in each of the 16 histograms (128 total).
4. For each pixel, calculate the pixel’s contribution to each accumulator variable.

Converting SIFT Descriptors to 128-dim Vectors

Post processing:

1. Normalize f_i :

$$f_i \rightarrow \frac{f_i}{||f_i||}$$

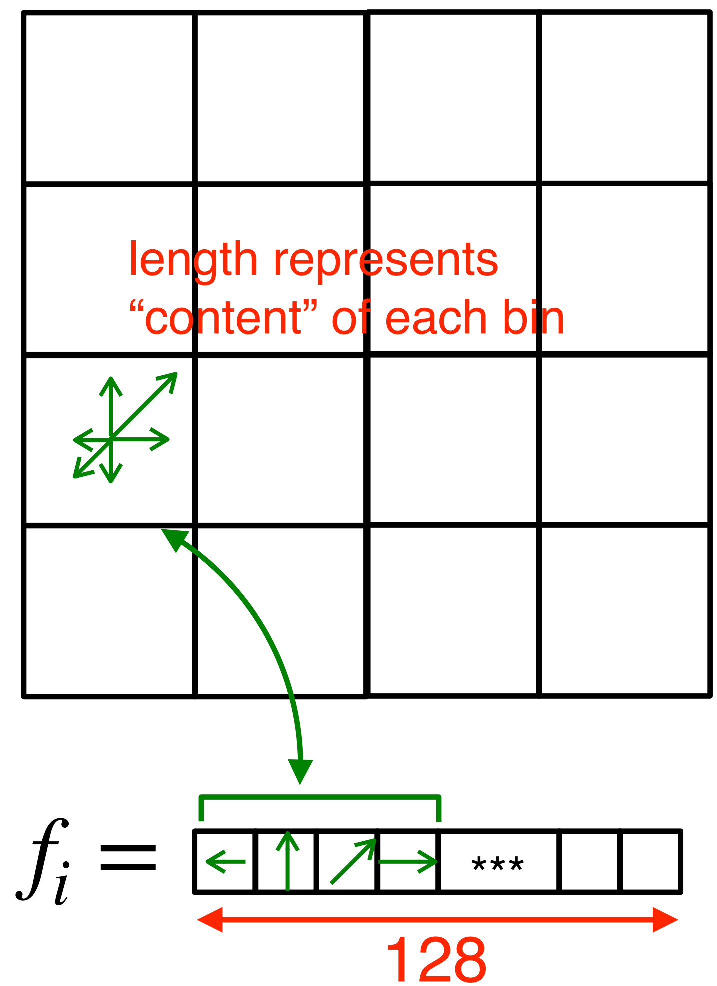
$\Rightarrow$ gives invariance to linear lighting variations across images, ie. when matching image I and image aI + b (because f_i will be the **same** in both images)

2. Clamp f_i :

Clamp all elements of f_i at 0.2

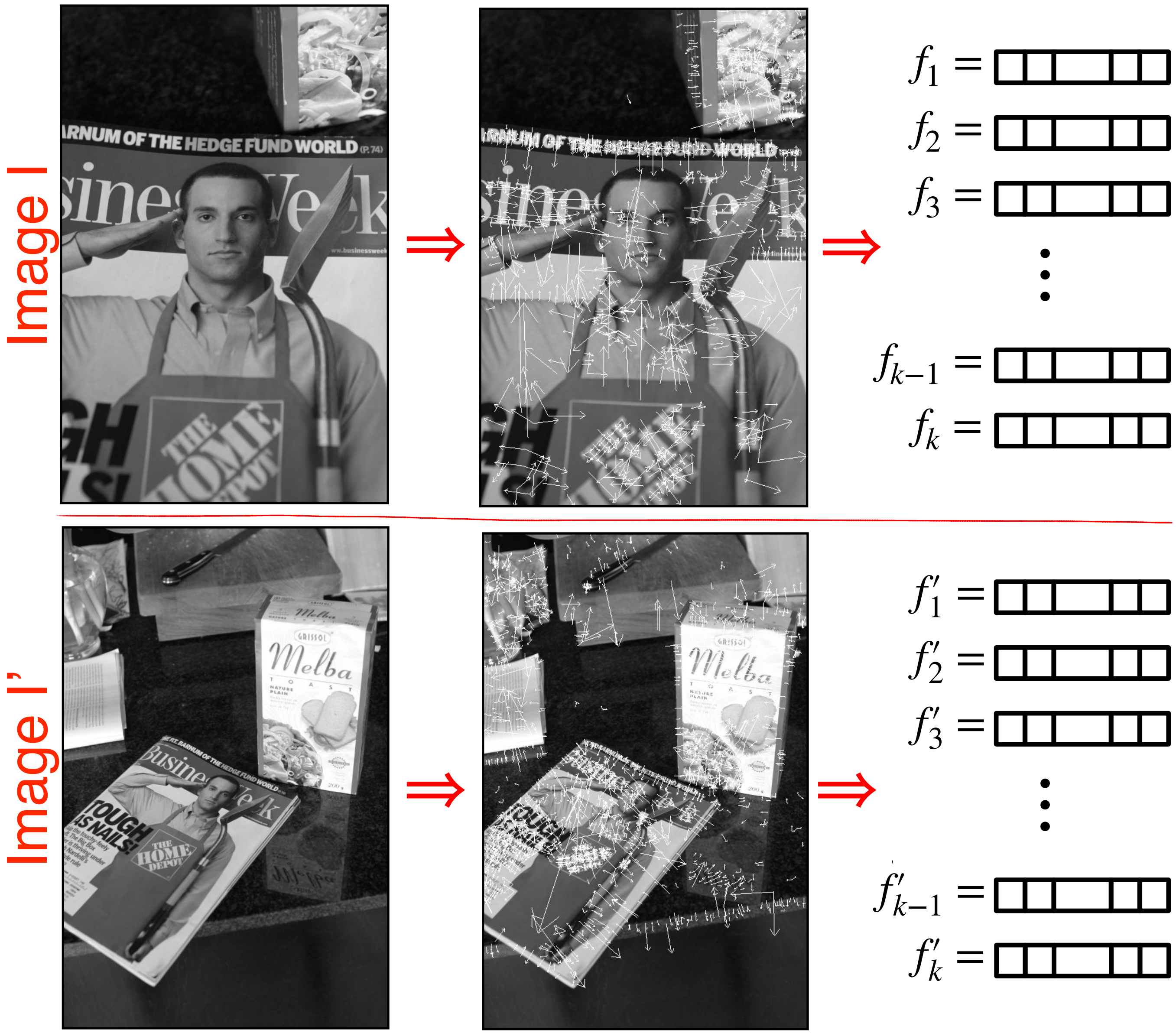
$\Rightarrow$ gives less weight to very large gradient magnitudes

SIFT Descriptor



3. Re-normalize

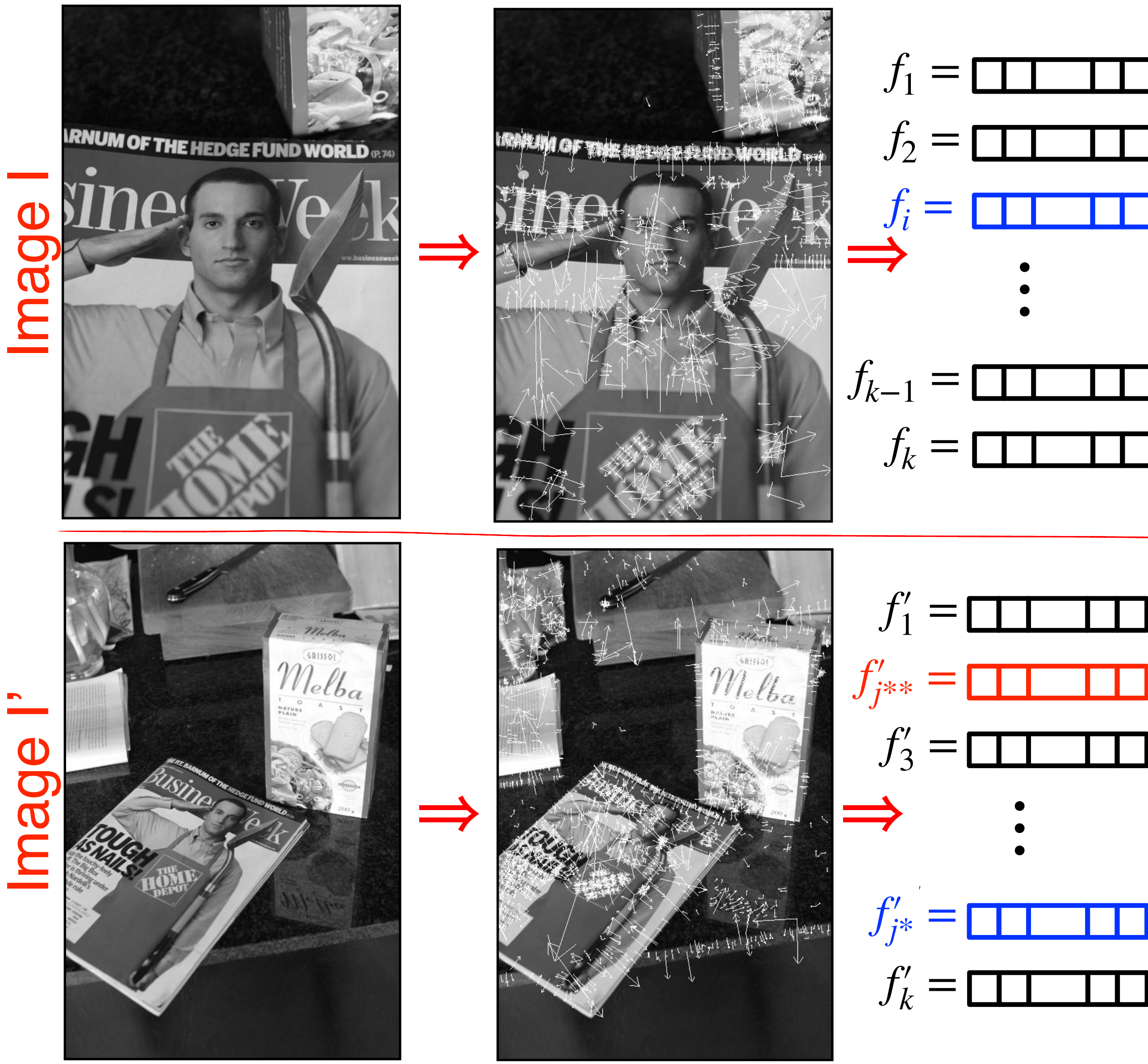
Matching 2 Images Using SIFT Features



3. Match feature vectors in the two sets:
{f_1, f_2, …, f_k}
and
{f'_1, f'_2, …, f'_k}

1. Identify keypoints 2. Build feature vectors

Matching 2 Images Using SIFT Features



1. Identify keypoints 2. Build feature vectors

3. Match f_i

a. Compute $\|f_i - f'_j\|$ for all j

b. Compute fraction

$$\phi = \frac{\|f_i - f'_{j*}\|}{\|f_i - f'_{j**}\|}$$

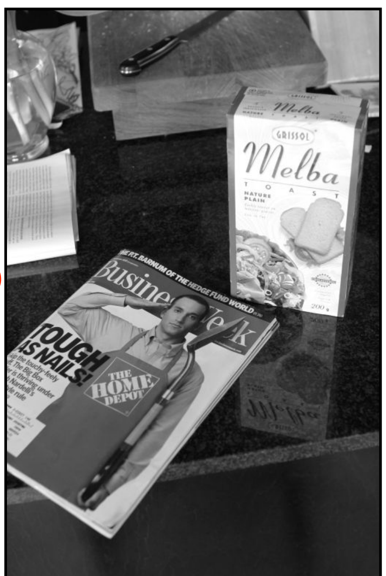
where f'_{j*} is the closest descriptor in I' and f'_{j**} is 2nd-closest.

c. Match f_i to f'_{j*} if $\phi < 0.8$

Matching 2 Images Using SIFT Features

Image I

Image I'



Intuition for matching algorithm:

match established only if it is deemed reliable, ie. if there is only **one** very similar feature in image I'

$f_1 =$

$f_2 =$

$f_i =$

⋮

$f_{k-1} =$

$f_k =$

$f'_1 =$

$f'_{j**} =$

$f'_3 =$

⋮

$f'_{j*} =$

$f'_k =$

3. Match $\mathrm{f_i}$

a. Compute $\|f_i - f'_j\|$ for all j

b. Compute fraction

$$\phi = \frac{\|f_i - f'_{j*}\|}{\|f_i - f'_{j**}\|}$$

where f'_{j*} is the closest descriptor in I' and f'_{j**} is 2nd-closest.

c. Match f_i to f'_{j*} if $\phi < 0.8$

1. Identify keypoints 2. Build feature vectors