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

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

Indicates a correspondence between location \((x, y)\) in image \(I\) and location \((x', y')\) in image \(I'\).
The Image Matching Problem

Q:
Is it possible to solve this problem by direct template matching between two images?
The Image Matching Problem

Q: Is it possible to solve this problem by direct template matching between two images?
The Image Matching Problem

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

A:
Yes, but it would be impossibly inefficient (i.e., must search over all possible pairs of patches).
Feature-Based Image Matching
Feature-Based Image Matching

Detect features in I

Detect features in I’

Match features across the two images
Errors in Feature-Based Image Matching

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

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’**

**Correct matches** = true positives (as fraction of total)

**Incorrect matches** = false positives (as fraction of total)

- **Ideal performance**
- **Good performance**
- **Poor performance**

Graph showing performance metrics for feature detection.
To be most useful, the feature detector & matching algorithm must be insensitive to a wide range of image transformations.
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.
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.

**Brightness transformation**

**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.
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
SIFT: Scale Invariant Feature Transform
• 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
Image Matching Using SIFT Features

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 \( \{f_1, f_2, \ldots, f_k\} \).
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

\[ p_i = (x_i, y_i, \rho_i, \theta_i) \]

- location
- scale
- orientation

“Source” image \(I\)

Detected keypoints

length denotes scale
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
A set of \( k \) feature vectors

\[
p_i = (x_i, y_i, \rho_i, \theta_i)\]

\[
f_i = \ldots\]

location
scale
orientation

describes the patch centred at \( p_i \)

Detected keypoints
Representing an Image Using SIFT Features: Steps

1. Identify keypoints
2. Build feature vectors

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

1. Identify keypoints
2. Build feature vectors
3. Match feature vectors in the two sets: \( \{f_1, f_2, \ldots, f_k\} \) and \( \{f'_1, f'_2, \ldots, f'_k\} \)
Matching 2 Images Using SIFT Features

1. Identify keypoints
2. Build feature vectors
3. Match vectors
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

\[ p_i = (x_i, y_i, \rho_i, \theta_i) \]

“Source” image \(I\)  Detected keypoints

length denotes 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
Computing SIFT Keypoints: Basic Steps

Source image $I$

Gauss-pyramid

Step 1a
Build pyramid of Gauss-smoothed images

Step 1f
Assign orientation to extrema

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

DOG pyramid

Step 1b
Build DOG pyramid

Step 1e
Prune set of extrema

keypoints = \{ all remaining $(x'_i, y'_i, \rho'_i)$ \}

DOG extrema

Step 1c
Locate extrema of DOG pyramid

$(x_i, y_i, \rho_i)$

Step 1d
Refine location of DOG extrema

$(x'_i, y'_i, \rho'_i)$

Location refinement

Orientation assign

Extremum pruning
Step 1a: Construct a Gauss-like Pyramid

Step 1a: Compute a pyramid of Gauss-filtered images organized into \textit{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, $I_1$, ..., $I_S$ representing a doubling of the scale parameter $\sigma$ between $I_1$ and $I_S$.

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

\[
\begin{align*}
I_s &= I \ast G_{k^s\sigma} \\
\vdots \\
I_2 &= I \ast G_{k(k\sigma)} \\
I_1 &= I \ast G_{k\sigma} \\
I_0 &= I \ast G_{\sigma}
\end{align*}
\]
Step 1a: Construct a Gauss-like Pyramid

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

Each octave contains $s + 1$ images.

In practice:
- $s = 3$ (3 images/octave)
- $\sigma = 1.6$ (for first octave)

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

Diagram:
- $I_s = I * G_{k^s\sigma}$
- $I_2 = I * G_{k(k\sigma)}$
- $I_1 = I * G_{k\sigma}$
- $I_0 = I * G_{\sigma}$

Scale (next octave)
Computing SIFT Keypoints: Basic Steps

Source image I

Gauss-pyramid

Step 1a
Build pyramid of Gauss-smoothed images

Step 1b
Build DOG pyramid

Source image I Gauss-pyramid DOG pyramid

Step 1c
Locate extrema of DOG pyramid

Source image I Gauss-pyramid DOG pyramid DOG extrema

Step 1d
Refine location of DOG extrema

Source image I Gauss-pyramid DOG pyramid DOG extrema

Step 1e
Prune set of extrema keypoints = {all remaining $(x'_i, y'_i, \rho'_i)$}

Source image I Gauss-pyramid DOG pyramid DOG extrema

Step 1f
Assign orientation to extrema

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

Source image I Gauss-pyramid DOG pyramid DOG extrema

Extremum pruning  Location refinement  Orientation assign

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

$(x'_i, y'_i, \rho'_i)$
Step 1b: Compute a pyramid of DOG-filtered images

\[ D(x, y, \rho) = I(x, y) \ast (G(x, y, k\rho) - G(xy, \rho)) \]

for \( \rho = \sigma, k\sigma, k^2\sigma, \ldots, k^{s-1}\sigma \)

Each octave contains \( s + 1 \) images.
Step 1b: Compute Pyramid of DOG Images

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

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

\[
I_s = I \ast G_{k^S\sigma}
\]
\[
\vdots
\]
\[
I_2 = I \ast G_{k(k\sigma)}
\]
\[
I_1 = I \ast G_{k\sigma}
\]
\[
I_0 = I \ast G_{\sigma}
\]

Each octave contains \( s + 1 \) images

Difference of Gaussian (DOG)

Scale (next octave)
Reminder: Difference-of-Gaussian Filtering

\[ I \ast G_\rho \]
Reminder: Difference-of-Gaussian Filtering

$I \ast G_{k\rho}$
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

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

$$D(x, y, \rho) = \left[ \nabla^2 (I * G_{\rho}) \right] \rho^2 k(k - 1)$$
Computing SIFT Keypoints: Basic Steps

- **Step 1a**: Build pyramid of Gauss-smoothed images
- **Step 1b**: Build DOG pyramid
- **Step 1c**: Locate extrema of DOG pyramid
- **Step 1d**: Refine location of DOG extrema
- **Step 1e**: Prune set of extrema
- **Step 1f**: Assign orientation to extrema

**Source image I** → **Gauss-pyramid** → **DOG pyramid** → **DOG extrema**

- **Orientation assign**: $p_i = (x'_i, y'_i, \rho'_i, \theta_i)$
- **Extremum pruning**: keypoints = \{ all remaining $(x'_i, y'_i, \rho'_i)$ \}
- **Location refinement**: $(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$
**Step 1c: Detecting DOG Extrema**

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

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

- $I_s = I \ast G_{k^s\sigma}$
- $I_2 = I \ast G_{k(k\sigma)}$
- $I_1 = I \ast G_{k\sigma}$
- $I_0 = I \ast G_\sigma$

\[
D(x, y, k^s\sigma) \quad \text{Gaussian}
\]

\[
D(x, y, k\sigma) \quad \text{Difference of Gaussian (DOG)}
\]
The Difference-Of-Gaussians (DOG) Filter

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

- minimum at $(x, y)$ if $D(x, y, \rho) < \text{all neighbours}$
- maximum if $D(x, y, \rho) > \text{all neighbours}$
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

- $\rho k$ must also be $<$ than all neighbours in adjacent scales
- $\rho$ must also be $>$ than all neighbours in adjacent scales
- $\rho / k$ or
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

$D(x, y, kS\sigma)$

$I_s = I * G_{kS\sigma}$

$\vdots$

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

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

$I_0 = I * G_{\sigma}$

must also be < than all neighbours in adjacent scales

or

must also be > than all neighbours in adjacent scales

Gaussian

Difference of Gaussian (DOG)
Step 1c: Detecting DOG Extrema: Algorithm

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

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

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

Gaussian

$$D(x, y, k^S\sigma)$$

Difference of Gaussian (DOG)
Step 1c: SIFT Keypoints = DOG Extrema

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

An extremum detected at $D(x, y, \rho)$ defines the keypoint $(x, y, \rho)$.

\[
\begin{align*}
I_s &= I \ast G_{k^S\sigma} \\
\vdots & \\
I_2 &= I \ast G_{k(k\sigma)} \\
I_1 &= I \ast G_{k\sigma} \\
I_0 &= I \ast G_{\sigma}
\end{align*}
\]
Computing SIFT Keypoints: Basic Steps

**Source image I**

1. **Step 1a**
   - Build pyramid of Gauss-smoothed images

2. **Step 1b**
   - Build DOG pyramid

3. **Step 1c**
   - Locate extrema of DOG pyramid
   
   $(x_i, y_i, \rho_i)$

4. **Step 1d**
   - Refine location of DOG extrema
   
   $(x_i, y_i, \rho_i) \rightarrow (x'_i, y'_i, \rho'_i)$

5. **Step 1e**
   - Prune set of extrema keypoints
   
   keypoints = \{ all remaining $(x'_i, y'_i, \rho'_i)$ \}

6. **Step 1f**
   - Assign orientation to extrema
   
   $p_i = (x'_i, y'_i, \rho'_i, \theta_i)$

---

**Orientation assign**

**Extremum pruning**

**Location refinement**
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 $(x, y, \rho)$
- This improves the results when matching significantly
Step 1d: Refining Location of Extrema

1. 2nd order Taylor expansion of $D$ at $(x, y, \rho)$:

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

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

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 $\Leftrightarrow$ derivative is zero $\Rightarrow$ solve for $\frac{\partial D}{\partial \Delta \vec{x}} = 0$

$$\left( \frac{\partial \vec{x}}{\partial \vec{x}^2} \right)^{-1} \cdot \left( \frac{\partial D}{\partial \vec{x}} \right) = \left( \frac{\partial^2 D}{\partial \vec{x}^2} \right)(\Delta \vec{x})$$
Computing SIFT Keypoints: Basic Steps

Step 1a
Build pyramid of Gauss-smoothed images

Step 1b
Build DOG pyramid

Step 1c
Locate extrema of DOG pyramid

Step 1d
Refine location of DOG extrema

Step 1e
Prune set of extrema

Step 1f
Assign orientation to extrema

Source image I

Gauss-pyramid

DOG pyramid

DOG extrema

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

pi = (x'_i, y'_i, \rho'_i, \theta_i)

orientation assign  Extremum pruning  Location refinement
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 \((x'_i, y'_i, \rho'_i)\)

\[
|D(x'_i, y'_i, \rho'_i)| = \text{large}
\]

In practice, > 0.03

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

\[
I_s = I \ast G_{k^s \sigma}
\]
\[
\vdots
\]
\[
I_2 = I \ast G_{k(k \sigma)}
\]
\[
I_1 = I \ast G_{k \sigma}
\]
\[
I_0 = I \ast G_{\sigma}
\]

Gaussian

Difference of Gaussian (DOG)

\[
D(x, y, k^s \sigma)
\]

\[
D(x, y, k \sigma)
\]
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**: Position along edge is not constrained. Prune.

Mathematically:

\[
I_s = I \ast G_{k^S \sigma} \\
I_2 = I \ast G_{k(k\sigma)} \\
I_1 = I \ast G_{k\sigma} \\
I_0 = I \ast G_{\sigma}
\]

Diagram: Gaussian (DOG) vs. Difference of Gaussian (DOG)
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**
- Position along edge is not constrained.
- Prune.

$\omega_1, \omega_2 < 0$
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:** Position along edge is not constrained, prune.

**Compute hessian $H$ of** $D(x, y, \rho_i')$ at $(x, y) = (x_i', y_i')$

**Prune if** \[
\frac{\text{Tr}^2(H)}{\text{Det}(H)} > \left( \frac{r + 1}{r} \right)^2
\]

where for SIFT $r = 10$.
### Computing SIFT Keypoints: Basic Steps

<table>
<thead>
<tr>
<th>Source image I</th>
<th>Gauss-pyramid</th>
<th>DOG pyramid</th>
<th>DOG extrema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Step 1a</strong></td>
<td>Build pyramid of Gauss-smoothed images</td>
<td><strong>Step 1b</strong></td>
<td>Build DOG pyramid</td>
</tr>
<tr>
<td><strong>Step 1f</strong></td>
<td>Assign orientation to extrema</td>
<td><strong>Step 1e</strong></td>
<td>Prune set of extrema keypoints = {all remaining ((x'_i, y'_i, \rho'_i))}</td>
</tr>
</tbody>
</table>

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

**Orientation assign** -> **Extremum pruning** -> **Location refinement**
Step 1f: Keypoint Orientation Assignment

Assigning an orientation $\theta_i$ to keypoint $(x'_i, y'_i, \rho'_i)$:

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 $\theta_i = \text{highest peak in histogram}$
Step 1f: Keypoint Orientation Assignment

Computing Histogram of Orientations

- Orientations divided into 36 bins (one every 10 degrees).
- Pixel \((x, y)\) contributes to the bin corresponding to the gradient orientation \(\theta\) 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

### Gradient direction

- \(I \ast G_{\rho_i}'\)
- \(x_i', y_i'\)
- \(d\)
- \(x, y\)
- \(\theta\)
- highest peak
Computing SIFT Keypoints: Basic Steps

Source image $I$

Gauss-pyramid

Step 1a
Build pyramid of Gauss-smoothed images

Step 1f
Assign orientation to extrema

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

DOB pyramid

Step 1b
Build DOG pyramid

Step 1e
Prune set of DOG extrema

keypoints = {
all remaining
$(x'_i, y'_i, \rho'_i)$
}

DOB extrema

Step 1c
Locate extrema of DOG pyramid

$(x_i, y_i, \rho_i)$

Step 1d
Refine location of DOG extrema

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

Orientation assign
Extremum pruning
Location refinement
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

\[ p_i = (x_i, y_i, \rho_i, \theta_i) \]

"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
$\mathbf{p}_i = (x_i, y_i, \rho_i, \theta_i)$

A set of $k$ feature vectors
$\mathbf{f}_i = \ldots$

describes the patch centred at $p_i$
Building the SIFT Descriptor

1. Compute gradients in 16 x 16 pixel patch of Gaussian-smoothed image at scale of the keypoint, centered at \((x_i, y_i)\)

Image patch centered at \((x_i, y_i)\)

SIFT Descriptor

Gaussian-smoothed image at scale of the keypoint
Building the SIFT Descriptor

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

2. Compute gradient orientation relative to keypoint orientation

\[
\theta(x, y) = \tan^{-1} \left[ \frac{\frac{\partial (I \ast G_{\sigma_i})}{\partial y}}{\frac{\partial (I \ast G_{\sigma_i})}{\partial x}} \right] - \theta_i
\]
Building the 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 \((x, y)\):

\[
w = G_{\sigma_i} \left( \| (x - x_i, y - y_i) \| \right)
\]

\(\Rightarrow\) pixels closer to keypoint centre have higher weight

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

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

gradient magnitude of smoothed image
Building the SIFT Descriptor

Image patch centred at \((x_i, y_i)\)

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:
\[ C(x, y) = w \cdot \| \nabla I \ast G_{\sigma_i}(x, y) \| \]
Building the SIFT Descriptor

Image patch centred at \((x_i, y_i)\)

SIFT Descriptor

Contribution spread across 2 closest orientations & 3 closest histograms

\[ \Rightarrow \text{no abrupt changes in histogram if keypoint centre displaced by 3 - 4 pixels} \]

Example: fraction allocated to

orientation 1: \[ \frac{\theta_1}{\theta_1 + \theta_2} \]

orientation 2: \[ \frac{\theta_2}{\theta_1 + \theta_2} \]
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 $f_i$:

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

⇒ gives invariance to linear lighting variations across images, i.e. 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

⇒ gives less weight to very large gradient magnitudes

3. Re-normalize
Matching 2 Images Using SIFT Features

1. Identify keypoints
2. Build feature vectors
3. Match feature vectors in the two sets: \( \{f_1, f_2, \ldots, f_k\} \) and \( \{f'_1, f'_2, \ldots, f'_k\} \)
Matching 2 Images Using SIFT Features

1. Identify keypoints
2. Build feature vectors

\[ f_1 = \ldots \]
\[ f_2 = \ldots \]
\[ f_i = \ldots \]
\[ f_{k-1} = \ldots \]
\[ f_k = \ldots \]

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 \( \phi < 0.8 \)
Intuition for matching algorithm:

match established only if it is deemed reliable, i.e., if there is only one very similar feature in image \( I' \)

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 \( \phi < 0.8 \)