Lesson 6: Image Pyramids, Wavelets and Parametric Transformations

Multi-Resolution Analysis and Global Transformations

AI 879: Machine Vision

Dr. Thao Le

Today's Agenda

1. Introduction & Motivation
2. Image Pyramids
   - Template Matching and Correlation
   - Gaussian Image Pyramids
   - Laplacian Image Pyramids
   - Application: Image Compression
3. Wavelet Transforms
4. Parametric Transformations
5. Homework 6 Discussion
6. Summary & Knowledge Check

Introduction & Motivation

Welcome to Lesson 6

From Single-Scale to Multi-Scale

Lesson 5 Recap:

- Point operators (single pixel); Linear filtering (neighborhood); Frequency analysis (global)
- All at **one scale**

The Challenge

- Objects appear at different sizes
- Need scale-invariant detection
- Efficient multi-resolution analysis
- Image blending and compression

Today's Topics

- Image Pyramids
- Wavelet Transforms
- Parametric Transformations
- Image Blending

Learning Objectives

By the end of this lesson, you will be able to:

1. **Define and construct** image pyramids:
   - Gaussian pyramids
   - Laplacian pyramids
   - Steerable pyramids
2. **Understand** wavelet transforms:
   - Continuous vs. discrete wavelets
   - Multi-resolution decomposition
   - Denoising applications
3. **Apply** parametric transformations:
   - Translation, rotation, scaling
   - Affine transformations
   - Matrix representations
4. **Implement** image blending algorithms
5. **Code** these methods in MATLAB

The Scale Problem

Motivating Example: Object Detection
Problem: Find sunflowers in a field

Single-Scale Approach:

- Create a template (46×44 pixels)
- Use correlation to find matches
- Problem: Only finds sunflowers of similar size!

Multi-Scale Solution:

- Create image pyramid
- Search at each scale
- Find sunflowers of all sizes

Why Not Resize Template?

- More computationally expensive
- Template smaller than image
- Better to downsample large image

Efficiency

Image: 960×594 pixels
Template: 46×44 pixels
Downsampling image is faster!

Applications of Multi-Scale Analysis

Where are image pyramids used?

Computer Vision:

- Object detection
- Face recognition
- Feature matching
- Motion estimation

Image Processing:

- Image blending
- Panorama stitching
- HDR imaging
- Texture synthesis

Compression:

- JPEG 2000
- Progressive images
- Multi-resolution storage
- Efficient transmission

Real-World Impact

Image pyramids are fundamental to modern computer vision systems, from smartphone cameras to autonomous vehicles!

Image Pyramids

What are Image Pyramids?

Definition

Image pyramids are multi-resolution representations of images, where each level represents the image at a different scale.

Structure:

- Base level: Original image (highest resolution)
- Each higher level: Reduced resolution (typically 50%)
- Pyramid gets smaller $\rightarrow$ fewer pixels at top

Three Types:

1. **Gaussian Pyramid:** Smoothed and downsampled versions
2. **Laplacian Pyramid:** Difference images (bandpass filtered)
3. **Steerable Pyramid:** Adds orientation information

The Sunflower Detection Example

Key Observation: Sunflowers at different distances require different scales for detection

Normalized Cross-Correlation

How do we find objects in an image?

Normalized Cross-Correlation:

$$c(u,v) = \frac{\sum_{x,y}[I(x,y) - \bar{I}] \cdot [T(x-u, y-v) - \bar{T}]}{\sqrt{\sum_{x,y}[I(x,y) - \bar{I}]^2 \cdot \sum_{x,y}[T(x-u, y-v) - \bar{T}]^2}}$$

where:

- I = input image
- T = template
- $\bar{I}, \bar{T}$ = means
- $c(u,v)$ = correlation at position (u,v)

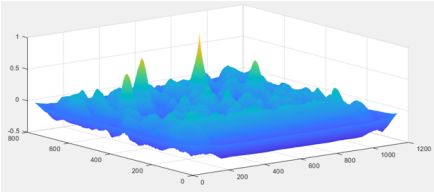
Properties:

- Values range: $[-1, 1]$
- $c = 1$: Perfect match
- $c \approx 0$: No correlation
- Invariant to brightness changes

Template Matching: MATLAB Code

```
I = im2gray(imread('L06 sunflower.png'));
T = im2gray(imread('L06 sunflower Template.png'));
c = normxcorr2(T, I); % Calculate normalized cross-correlation
% Visualize correlation surface
figure;
surf(c);
shading flat;
title('Correlation Surface');
% Find local maxima above threshold and display
threshold = 0.5;
[yvals, xvals] = find(c > threshold * max(c(:)) & ...
islocalmax(c, 1) & islocalmax(c, 2));
figure; imshow(I);
for i = 1:length(yvals)
  yoffSet = yvals(i) - size(T,1);
  xoffSet = xvals(i) - size(T,2);
  rectangle('Position', [xoffSet, yoffSet, ...
  size(T,2), size(T,1)], 'EdgeColor', [1 0 0]);
end
```

Correlation Results



Observation:

- 4 correlation peaks found (0.51-1.0); Each peak corresponds to a detected sunflower
- One false detection near the edge (proximity issue)

Gaussian Pyramid Construction

Algorithm:

1. Start with original image (Level 0)
2. For each subsequent level:
   - Apply Gaussian blur (smooth)
   - Downsample by factor of 2 (reduce size)
3. Repeat until desired number of levels

Why Gaussian Blur?

- Prevents aliasing
- Removes high frequencies before downsampling
- Satisfies Nyquist-Shannon sampling theorem

Key Properties:

- Each level is smoothed version of previous
- Default: downsample by factor of 2
- Pyramid height determined by image size

Gaussian Pyramid: MATLAB Implementation

```
I = imread('L06 sunflower.png');
% Create Gaussian pyramid with 4 levels
level1 = I;  % Original
level2 = impyramid(level1, 'reduce');  % 50% size
level3 = impyramid(level2, 'reduce');  % 25% size
level4 = impyramid(level3, 'reduce');  % 12.5% size
% Display pyramid
figure;
subplot(2,2,1); imshow(level1); title('Level 1 (Original)');
subplot(2,2,2); imshow(level2); title('Level 2 (50%)');
subplot(2,2,3); imshow(level3); title('Level 3 (25%)');
subplot(2,2,4); imshow(level4); title('Level 4 (12.5%)');
% Alternative: Use cell array
pyramid = cell(1, 4);
pyramid{1} = I;
for k = 2:4
  pyramid{k} = impyramid(pyramid{k-1}, 'reduce');
end
```

Gaussian Pyramid Visualization

Level 1 (Original)

Level 2 (50%)

Level 3 (25%)

Level 4 (12.5%)

The impyramid Function

MATLAB's impyramid:

```
J = impyramid(I, direction)
```

Parameters:

- `I`: Input image
- `direction`:
  - `'reduce'`: Smooth and downsample (default factor: 2)
  - `'expand'`: Upsample and smooth

What it does internally:

1. Applies Gaussian filter (5×5 kernel)
2. For 'reduce': Downsamples by keeping every other pixel
3. For 'expand': Upsamples by inserting zeros, then filters

Important

`impyramid` automatically handles the Gaussian smoothing. Don't blur before calling it!

Multi-Scale Object Detection

```
I = im2gray(imread('L06 sunflower.png'));
T = im2gray(imread('L06 sunflower Template.png'));
T = imgaussfilt(T, 2);
T = imresize(T, 0.5, 'nearest');  % Smaller template

figure; imshow(I);
for iter = 1:3  % 3 pyramid levels
  if iter == 1
    I_tmp = imgaussfilt(I, 2);  % Smooth original
  else
    I_tmp = impyramid(I_tmp, 'reduce');  % Downsample
  end

  % Find matches at this scale
  c = normxcorr2(T, I_tmp);
  [yvals, xvals] = find(c > 0.5 & islocalmax(c,1) & islocalmax(c,2));

  % Draw rectangles (scaled back to original size)
  for x = 1:length(yvals)
    yoffSet = iter * (yvals(x) - size(T,1));
    xoffSet = iter * (xvals(x) - size(T,2));
    rectangle('Position', [xoffSet, yoffSet, ...
    iter*size(T,2), iter*size(T,1)], ...
    'EdgeColor', [1 0 0], 'LineWidth', 2);
  end
end
```

Multi-Scale Detection Results

From Gaussian to Laplacian

Limitation of Gaussian Pyramids:

- Only store smoothed versions
- Lose high-frequency information
- Inefficient for reconstruction

Laplacian Pyramid Solution:

- Store **differences** between levels
- Capture information lost during downsampling
- Enable perfect reconstruction
- Useful for compression and blending

Construction:

$$L_i = G_i - \text{upsample}(G_{i+1})$$

where:

- L_i = Laplacian at level i
- G_i = Gaussian at level i
- Except top level: $L_n = G_n$

Laplacian Pyramid: Construction Code

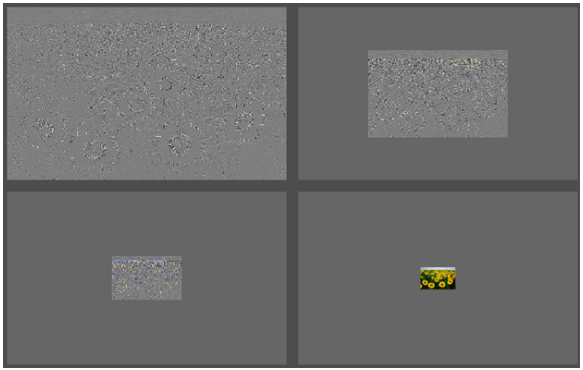
```
function lapp = laplacianPyramid(gaussian_pyramid)
  num_levels = numel(gaussian_pyramid);
  lapp = cell(1, num_levels);
  % Process all levels except the last
  for k = 1:(num_levels - 1)
    A = gaussian_pyramid{k};
    B = imresize(gaussian_pyramid{k+1}, 2, 'bilinear');
    % Crop B to match A's size (handle size mismatch)
    [M, N, ~] = size(A);
    lapp{k} = A - B(1:M, 1:N, :);
  end
  % Top level is just the Gaussian
  lapp{num_levels} = gaussian_pyramid{num_levels};
end

```

Key Steps:

- Expand (upsample) next coarser level
- Subtract from current level
- Store difference (the Laplacian)

Laplacian Pyramid Visualization



Note: Laplacian images appear mostly gray because differences are small

Reconstruction from Laplacian Pyramid

Perfect Reconstruction:

Starting from the top (coarsest level), work down:

$$G_i = L_i + \text{upsample}(G_{i+1})$$

```
function out = reconstructFromLaplacian(lapp)
  num_levels = numel(lapp);
  out = lapp{end};  % Start with top level

  % Work backwards through pyramid
  for k = (num_levels - 1) : -1 : 1
    % Upsample current reconstruction
    out = imresize(out, 2, 'bilinear');

    % Add Laplacian at this level
    g = lapp{k};
    [M, N, ~] = size(g);
    out = out(1:M, 1:N, :) + g;
  end
end
```

Result: Exactly reconstructs the original image (within numerical precision)

Laplacian Pyramids for Compression

Idea: Quantize Laplacian coefficients to reduce file size

Why it works:

- Most Laplacian values are small (near zero)
- Can round to fewer decimal places
- Loses some detail but maintains structure

Simple Quantization:

```
function lapp_quantized = roundPixelValues(lapp)
  for k = 1:(numel(lapp) - 1)
    % Round to 1 decimal place
    lapp_quantized{k} = round(lapp{k}, 1);
  end
  % Keep top level unchanged
  lapp_quantized{numel(lapp)} = lapp{end};
end

```

Compression Results

Key Result: 36% size reduction with minimal visual quality loss!

Complete Laplacian Compression Example

```
% Read image and create Gaussian pyramid
I = imread('L06 sunflower.png');
mrp = multiresolutionPyramid(I, 4);

% Create Laplacian pyramid
lapp = laplacianPyramid(mrp);

% Quantize (compress)
lapp_quantized = roundPixelValues(lapp);

% Reconstruct
out = reconstructFromLaplacianPyramid(lapp_quantized);

% Compare file sizes
imwrite(I, 'original.png');
imwrite(out, 'compressed.png');
info_orig = dir('original.png');
info_comp = dir('compressed.png');
reduction = (1 - info_comp.bytes/info_orig.bytes) * 100;
fprintf('Size reduction: %.1f%%\n', reduction);

% Display
subplot(2,1,1); imshow(I); title('Original');
subplot(2,1,2); imshow(out); title('Compressed & Reconstructed');

```

Wavelet Transforms

Why Wavelets?

Limitations of Fourier Transform:

- Global frequency analysis
- No time/spatial localization
- Fixed resolution across frequencies

Wavelets Advantages:

- Time-frequency localization
- Multi-resolution analysis
- Better for non-stationary signals
- Efficient for compression

Analogy

Fourier: Decomposes signal into sine waves
Wavelets: Decomposes signal into scaled and shifted wavelets

What is a Wavelet?

Definition

A **wavelet** is a rapidly decaying, wave-like oscillation with zero mean.

Properties:

- Localized in both time and frequency
- Different wavelets for different applications
- Orthogonal or biorthogonal

Continuous Wavelet Transform (CWT):

$$W(a,b) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi^* \left(\frac{t-b}{a} \right) dt$$

where:

- a = scale parameter
- b = translation parameter
- ψ = mother wavelet

Discrete Wavelet Transform (DWT)

For Images:

2D DWT Process:

1. Apply 1D DWT to each row
2. Apply 1D DWT to each column
3. Results in 4 subbands:
   - LL: Low-low (approximation)
   - LH: Low-high (horizontal details)
   - HL: High-low (vertical details)
   - HH: High-high (diagonal details)

Multi-Level Decomposition:

- Recursively decompose LL subband
- Creates multi-resolution representation
- Similar to pyramid structure

Wavelet Families

Common Wavelets in MATLAB:

Family	Type	Examples
Daubechies	Orthogonal	db1 (Haar), db2, ..., db45
Symlets	Orthogonal	sym2, sym4, ..., sym45
Coiflets	Orthogonal	coif1, coif2, ..., coif5
BiorSplines	Biorthogonal	bior1.1, bior2.2, bior3.3
Fejér-Korovkin	Orthogonal	fk4, fk6, fk8, fk14, fk22

Choosing a Wavelet:

- Daubechies: General purpose, good compression
- Symlets: More symmetric than Daubechies
- Biorthogonal: Perfect reconstruction, linear phase
- Application-specific: Experiment to find best

Application: Image Denoising

Wavelet Denoising Principle:

1. **Decompose:** Apply DWT to noisy image
2. **Threshold:** Remove small wavelet coefficients (noise)
3. **Reconstruct:** Apply inverse DWT

Why it works:

- Signal energy concentrated in large coefficients
- Noise spread across all coefficients
- Thresholding removes noise while preserving signal

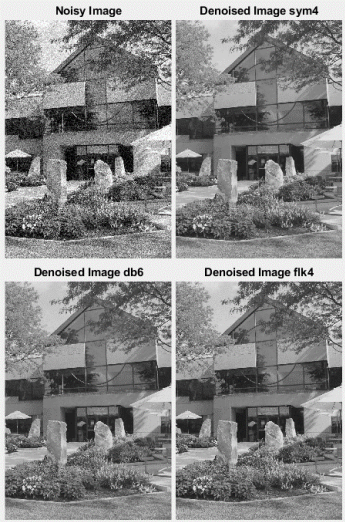
MATLAB Function:

- `wdenoise2`: 2D wavelet denoising
- Automatic threshold selection
- Multiple wavelet options

Wavelet Denoising: MATLAB Code

```
I = imread('L01 greatvalley.jpg');
J = imnoise(I, 'gaussian');  % Add Gaussian noise
% Denoise with different wavelets
out1 = wdenoise2(J);  % Default: sym4, level 7
out2 = wdenoise2(J, 4, Wavelet="db6");   % Level 4, db6
out3 = wdenoise2(J, 3, Wavelet="fk14");  % Level 3, fk14
% Display results
figure;
subplot(2,2,1); imshow(J); title('Noisy Image');
subplot(2,2,2); imshow(out1, []); title('sym4 (default)');
subplot(2,2,3); imshow(out2, []); title('db6, level 4');
subplot(2,2,4); imshow(out3, []); title('fk14, level 3');
% Compare PSNR
psnr1 = psnr(out1, I);
psnr2 = psnr(out2, I);
psnr3 = psnr(out3, I);
fprintf('PSNR: sym4=%.2f, db6=%.2f, fk14=%.2f\n', ...
psnr1, psnr2, psnr3);
```

Wavelet Denoising Results



Decomposition Levels

Choosing the Number of Levels:

Maximum levels:

$$L_{\max} = \lfloor \log_2(\min(M, N)) \rfloor$$

where $M \times N$ is image size

For Great Valley image (960×594):

- Maximum: $\lfloor \log_2(594) \rfloor = 9$
- Default: 7 (automatically chosen by MATLAB)
- Higher levels: More smoothing
- Lower levels: More detail preservation

Trade-offs:

- More levels: Better denoising, possible oversmoothing
- Fewer levels: More details, possible residual noise
- Optimal level depends on noise level and image content

Wavelets vs. Pyramids

Aspect	Pyramids	Wavelets
Decomposition	Isotropic (no direction)	Directional (H, V, D)
Reconstruction	Approximate (Gaussian) or exact (Laplacian)	Exact (orthogonal)
Compression	Good for images	Excellent (JPEG 2000)
Computation	Simple, fast	More complex
Applications	Blending, detection	Denoising, compression
Flexibility	Limited wavelet choice	Many wavelet families

When to Use Which?

Pyramids: Real-time processing, object detection, blending
Wavelets: Compression, denoising, feature extraction

Parametric Transformations

What are Parametric Transformations?

Definition

Parametric transformations apply global deformations to images, controlled by a small number of parameters.

Key Characteristic:

- Apply to the **entire image**
- Defined by **matrix multiplication**
- Preserve certain properties (lines, parallelism, etc.)

Common Types:

- Translation (2 parameters)
- Rotation (1 parameter)
- Scaling (2 parameters)
- Shear (2 parameters)
- Reflection (1 parameter)

Homogeneous Coordinates

Why 3×3 matrices for 2D transformations?

Homogeneous coordinates:

$$\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Advantages:

- Allows translation as matrix multiplication
- Can combine transformations by matrix multiplication
- Uniform representation for all transformations

General Form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation

Move image by (t_x, t_y):

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Effect:

- $x' = x + t_x$
- $y' = y + t_y$
- Shifts entire image

Parameters:

- t_x: horizontal displacement
- t_y: vertical displacement
- Can be positive or negative

Scaling

Scale by factors (s_x, s_y):

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

Effect:

- $x' = s_x \cdot x$
- $y' = s_y \cdot y$
- Scales about origin

Special Cases:

- $s_x = s_y$: Uniform scaling
- $s_x = s_y > 1$: Enlargement
- $s_x = s_y < 1$: Reduction
- $s_x \neq s_y$: Non-uniform (aspect ratio change)

Rotation

Rotate by angle θ about origin:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Effect:

- $x' = x\cos\theta - y\sin\theta$
- $y' = x\sin\theta + y\cos\theta$
- Counterclockwise rotation

Properties:

- Preserves distances
- Preserves angles
- Orthogonal matrix: $R^T = R^{-1}$

Shear

Shear transformations:

Horizontal shear:

$$\text{Shear}_x = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Vertical shear:

$$\text{Shear}_y = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Effect:

- Skews the image
- Preserves areas
- Creates parallelogram from rectangle

Reflection

Mirror transformations:

Vertical reflection (flip up-down):

$$\text{Reflect}_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Horizontal reflection (flip left-right):

$$\text{Reflect}_h = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflection about arbitrary line (angle ϕ):

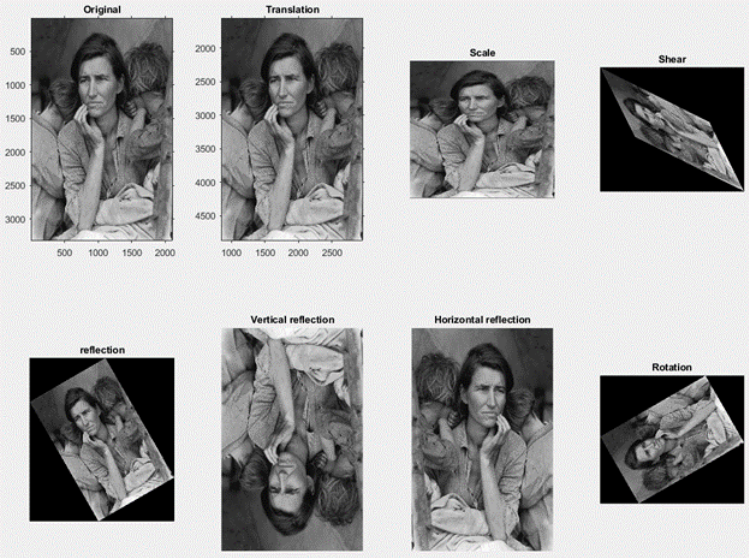
$$\text{Reflect}_\phi = \begin{bmatrix} \cos(2\phi) & \sin(2\phi) & 0 \\ \sin(2\phi) & -\cos(2\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

MATLAB Implementation: affine2d

```
% Load image
I = imread('L01 Migrant Mother.png');
I = im2double(I);
% Translation
T = [1 0 0; 0 1 0; 850 1550 1];
tform = affine2d(T);
J1 = imwarp(I, tform);
% Scaling
T = [0.5 0 0; 0 0.3 0; 0 0 1];
tform = affine2d(T);
J2 = imwarp(I, tform);
% Rotation (30 degrees)
theta = 30;
T = [cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0; 0 0 1];
tform = affine2d(T);
J3 = imwarp(I, tform);
% Shear
T = [1 2 0; 2 1 0; 0 0 1];
tform = affine2d(T);
J4 = imwarp(I, tform);
% Display all
subplot(2,3,1); imshow(I); title('Original');
subplot(2,3,2); imshow(J1); title('Translation');
subplot(2,3,3); imshow(J2); title('Scale');
subplot(2,3,4); imshow(J3); title('Rotation');
subplot(2,3,5); imshow(J4); title('Shear');

```

Parametric Transformation Results



Homework 6 Discussion

Homework 6: Image Pyramids

Objective: Demonstrate understanding of Gaussian and Laplacian pyramids

Task 1: Modify Gaussian Pyramid Code
Make the following modifications and discuss results:

- **A.** Increase input image size by 150%
- **B.** Keep template size at 50% of original
- **C.** Increase iterations to 5
- **D.** Replace `impyramid` with `imresize` at factor 0.8

Task 2: Image Blending

- Implement blending algorithm (Szeliski 3.5.5)
- Choose any two images
- Create smooth composite

Summary & Knowledge Check

Lesson Summary

Key Concepts Covered:

1. **Image Pyramids:**
   - Gaussian: Multi-resolution representation
   - Laplacian: Bandpass decomposition
   - Applications: Detection, compression, blending
2. **Wavelet Transforms:**
   - Time-frequency localization
   - Multi-resolution analysis
   - Applications: Denoising, compression
3. **Parametric Transformations:**
   - Affine transformations (6 DOF)
   - Matrix representations
   - Combining transformations
4. **Applications:**
   - Scale-invariant object detection
   - Seamless image blending
   - Image denoising

Knowledge Check - Question 1

Question: What is the main advantage of using a Gaussian pyramid for object detection compared to a single-scale approach?

A) It makes the image look better
B) It allows detection of objects at multiple scales
C) It reduces image noise
D) It increases image resolution

Knowledge Check - Question 1

Question: What is the main advantage of using a Gaussian pyramid for object detection compared to a single-scale approach?

A) It makes the image look better
B) It allows detection of objects at multiple scales
C) It reduces image noise
D) It increases image resolution

Answer

B) It allows detection of objects at multiple scales

Gaussian pyramids create versions of the image at different resolutions, enabling detection of objects regardless of their size in the image. A single-scale template would only match objects of a specific size.

Knowledge Check - Question 2

Question: In a Laplacian pyramid, what does each level (except the top) represent?

A) A smoothed version of the image
B) The difference between adjacent Gaussian pyramid levels
C) The average of two pyramid levels
D) Random noise

Knowledge Check - Question 2

Question: In a Laplacian pyramid, what does each level (except the top) represent?

A) A smoothed version of the image
B) The difference between adjacent Gaussian pyramid levels
C) The average of two pyramid levels
D) Random noise

Answer

B) The difference between adjacent Gaussian pyramid levels

Each Laplacian level stores the difference between a Gaussian level and an upsampled version of the next coarser level: $L_i = G_i - \text{upsample}(G_{i+1})$. This captures bandpass-filtered information.

Knowledge Check - Question 3

Question: Why is it important to apply Gaussian blur before downsampling an image?

A) To make the image prettier
B) To prevent aliasing artifacts
C) To increase computation speed
D) To enhance edges

Knowledge Check - Question 3

Question: Why is it important to apply Gaussian blur before downsampling an image?

A) To make the image prettier
B) To prevent aliasing artifacts
C) To increase computation speed
D) To enhance edges

Answer

B) To prevent aliasing artifacts

Gaussian blurring acts as a low-pass filter, removing high frequencies before downsampling. This prevents aliasing (high frequencies appearing as low frequencies), following the Nyquist-Shannon sampling theorem.

Knowledge Check - Question 4

Question: What is the key advantage of wavelets over Fourier transforms for image analysis?

A) Wavelets are always faster
B) Wavelets provide time-frequency localization
C) Wavelets only work in 1D
D) Wavelets require less memory

Knowledge Check - Question 4

Question: What is the key advantage of wavelets over Fourier transforms for image analysis?

A) Wavelets are always faster
B) Wavelets provide time-frequency localization
C) Wavelets only work in 1D
D) Wavelets require less memory

Answer

B) Wavelets provide time-frequency localization
Unlike Fourier transforms which provide only frequency information, wavelets decompose signals with both temporal/spatial and frequency localization. This makes them better for analyzing non-stationary signals and localized features.

Knowledge Check - Question 5

Question: Which transformation preserves parallel lines?

A) Only translation
B) Only rotation
C) All affine transformations
D) No transformations preserve parallel lines

Knowledge Check - Question 5

Question: Which transformation preserves parallel lines?

A) Only translation
B) Only rotation
C) All affine transformations
D) No transformations preserve parallel lines

Answer

C) All affine transformations

Affine transformations (including translation, rotation, scaling, and shear) preserve parallelism. Lines that are parallel before an affine transformation remain parallel after. This is one of the defining properties of affine transformations.

Knowledge Check - Question 6

Question: In the `impyramid('reduce')` function, by what factor is the image downsampled?

A) 0.5 (half the size)
B) 0.75 (three-quarters)
C) 0.8 (80%)
D) 0.25 (quarter size)

Knowledge Check - Question 6

Question: In the `impyramid('reduce')` function, by what factor is the image downsampled?

A) 0.5 (half the size)
B) 0.75 (three-quarters)
C) 0.8 (80%)
D) 0.25 (quarter size)

Answer

A) 0.5 (half the size)
The `impyramid('reduce')` function downsamples by a factor of 2 in each dimension, resulting in an image that is 50% the width and 50% the height (25% the total pixels) of the original. It also applies Gaussian smoothing automatically.

Thank You!

Good Luck with Homework 6!

See You Next Time!

Next Lesson: Points and Matches