Artificial Vision (VIAR25/26)

Lab 2: Image Representations, Processing, and Filtering

Prof. David Olivieri

UVigo

September 12, 2025

Section 1

Overview and Objectives

Lab 2 Overview: From Theory to Practice

Bridge Mathematical Foundations to Real Implementation

Today you'll implement the core algorithms from Lecture 2, gaining deep understanding through hands-on coding and performance analysis.

What You'll Build Today

- **Sampling & Aliasing**: Create aliasing demonstrations and anti-aliasing filters
- **Color Conversion**: Manual RGB → HSV with mathematical precision
- **Enhancement**: Histogram equalization and adaptive techniques
- **Filtering**: Direct, separable, and FFT-based convolution with benchmarking
- **Edge Detection**: Complete 5-stage Canny implementation
- **Advanced Methods**: Morphology, non-linear filters, modern CNN analysis

Learning Philosophy

Implement first, optimize second: Understanding the mathematics through code builds intuition that lasts beyond any specific library or framework.

Lab Structure and Methodology

Technical Stack

- **Core**: `NumPy`, `SciPy`, `OpenCV`
- **Deep Learning**: `PyTorch`
- **Visualization**: `Matplotlib`
- **Performance**: time profiling

Code Organization

- 8 main classes, each focused on specific concepts
- Methods with `TODO` for your implementation
- Helper functions and test data provided
- Comprehensive comparison with library implementations

Assessment Approach

- **Implementation Quality**: correctness and efficiency
- **Analysis Depth**: understanding of trade-offs
- **Visualization**: clear presentation of results
- **Performance Study**: benchmarking and optimization

Deliverables

- Completed Python notebook with all implementations
- Comprehensive results visualization
- Performance analysis report
- Classical vs. modern techniques comparison

Section 2

Detailed Exercises

Exercise 1: Sampling and Aliasing Demonstration

Core Implementation Tasks

- **Sine Grating Generation**: create parametric sinusoidal patterns
- **Aliasing Simulation**: demonstrate frequency folding with controlled downsampling
- **Anti-aliasing**: implement Gaussian pre-filtering with optimal σ selection
- **FFT Analysis**: frequency-domain visualization of aliasing effects

Mathematical Focus

- Nyquist–Shannon theorem validation
- Aliased frequency calculation
- Filter design trade-offs

Expected Insights

- Visual impact of aliasing
- Effectiveness of anti-aliasing
- Parameter sensitivity analysis

Challenge Question

For a 0.3 cycles/pixel grating downsampled by factor 3, predict the aliased frequency and verify experimentally. Explain any discrepancies.

Exercise 2: Color Space Conversion Mathematics

Implementation Requirements

- **Manual RGB→HSV**: follow exact algorithm from lecture slides
- **Precision Validation**: compare with `OpenCV` implementation
- **Edge Case Handling**: zero saturation, zero value scenarios
- **Range Mapping**: proper scaling to `OpenCV`'s $H \in [0, 179]$ hue range

Algorithm Implementation Steps

1. Normalize RGB values to $[0, 1]$ range.
2. Compute $V = \max(R, G, B)$ and $\delta = V - \min(R, G, B)$.
3. Calculate $S = \delta / V$ (handle $V = 0$ case).
4. Determine H based on which channel achieves maximum.
5. Map to `OpenCV` ranges: $H \in [0, 179]$, $S, V \in [0, 255]$.

Analysis Tasks

- Quantify differences between implementations
- Identify sources of numerical errors
- Evaluate robustness to different image types

Exercise 3: Image Enhancement Techniques

Histogram Equalization

- Implement complete CDF transformation
- Handle edge case: $c_{\min}$ normalization
- Compare with `cv2.equalizeHist`
- Analyze histogram redistribution

CLAHE Implementation

- Understand contextual adaptation
- Implement clip limiting concept
- Compare local vs. global enhancement
- Analyze noise amplification trade-offs

Mathematical Foundation

$$T[k] = \mathrm{round}\left(\frac{c[k] - c_{\min}}{N - c_{\min}} \times 255\right), \quad \text{with } N = H \cdot W$$

Performance Metrics

- Contrast improvement measurement
- Edge preservation analysis
- Computational efficiency comparison

Critical Analysis

When does global histogram equalization fail? How does CLAHE address these limitations? Provide quantitative evidence.

Exercise 4: Convolution Performance Engineering

Three Implementation Paradigms

- **Direct Convolution**: sliding window with proper boundary handling
- **Separable Convolution**: exploit rank-1 kernel decomposition
- **FFT Convolution**: frequency-domain multiplication with padding

Complexity Analysis

- Direct: $O(HW \cdot K^2)$
- Separable: $O(HW \cdot 2K)$
- FFT: $O(HW \log(HW))$

Benchmarking Matrix

- Image sizes: $64^2, 128^2, 256^2, 512^2$
- Kernel sizes: $3 \times 3, 5 \times 5, 7 \times 7, 15 \times 15$
- Compare with `OpenCV` optimized implementations

Critical Questions

- Where is the crossover point for FFT efficiency?
- How does separability impact performance?
- What are the memory vs. speed trade-offs?

Real-world Implications

- Mobile vs. server deployment
- Real-time processing constraints
- Energy consumption considerations

Exercise 5: Canny Edge Detection – The Complete Pipeline

Five-Stage Implementation

1. **Gaussian Smoothing**: scale-space parameter selection
2. **Gradient Computation**: Sobel operators with magnitude/direction
3. **Non-Maximum Suppression**: directional thinning algorithm
4. **Double Thresholding**: strong/weak edge classification
5. **Hysteresis Tracking**: connectivity-based edge linking

Implementation Challenges

- Gradient direction quantization
- Efficient NMS neighbor indexing
- BFS/DFS for hysteresis tracking
- Parameter sensitivity analysis

Validation Strategy

- Compare with `cv2.Canny`
- Analyze intermediate stage outputs
- Study parameter impact on results
- Measure edge connectivity quality

Deep Understanding Goal

Why does Canny's multi-stage approach achieve superior results compared to simple gradient thresholding? Demonstrate with your implementation.

Exercise 6 & 7: Advanced Filtering and Morphology

Non-Linear Filtering

- **Median Filter**: order statistics for impulse noise
- **Bilateral Filter**: edge-preserving smoothing
- Compare edge preservation capabilities
- Analyze computational complexity

Key Insights

- When linear filtering fails
- Robustness to different noise types
- Parameter tuning strategies

Mathematical Morphology

- **Basic Operations**: erosion, dilation
- **Composite Operations**: opening, closing
- Structuring element design
- Shape analysis applications

Set Theory in Practice

- Binary image processing
- Noise removal vs. shape preservation
- Connectivity analysis

Integration Challenge

Design a complete pipeline for binary image cleanup: noise removal $\rightarrow$ shape enhancement $\rightarrow$ feature extraction. Justify each processing step.

Exercise 8: Modern Deep Learning Perspectives

Bridging Classical and Contemporary Approaches

- **CNN Filter Visualization**: extract and analyze first-layer filters from ResNet
- **Learned vs. Hand-crafted**: compare performance on edge detection tasks
- **Feature Evolution**: study hierarchical representation learning
- **Transfer Learning**: analyze how pre-trained filters generalize

Technical Implementation

- `PyTorch` model loading and analysis
- Filter weight extraction and normalization
- Learnable convolution layer creation
- Performance comparison frameworks

Analytical Questions

- Do learned filters rediscover classical operators?
- How does task-specific training shape filters?
- What can we learn from CNN representations?

Future-Forward Thinking

How might the principles you've implemented today evolve in the next generation of computer vision systems? Consider efficiency, interpretability, and robustness.

Section 3

Practical Information

Implementation Guidelines and Best Practices

Code Quality Standards

- Follow `NumPy` vectorization principles
- Handle edge cases and boundary conditions
- Include comprehensive error checking
- Write clear, documented functions
- Use appropriate data types (`uint8`, `float32`)

Performance Optimization

- Profile your implementations
- Compare with library functions
- Identify computational bottlenecks
- Consider memory usage patterns

Debugging Strategies

- Start with small test cases
- Visualize intermediate results
- Check mathematical formulations
- Validate against known outputs
- Use `assert` statements for sanity checks

Documentation Requirements

- Comment complex mathematical operations
- Explain parameter choices
- Document assumptions and limitations
- Include performance measurements

Testing Philosophy

Every algorithm should be tested with: synthetic data (known ground truth), edge cases (empty images, single pixels), and real-world examples (various image types).

Assessment Criteria and Expectations

Technical Implementation (40%)

- **Correctness**: algorithms produce expected results
- **Efficiency**: reasonable computational complexity
- **Robustness**: handles edge cases gracefully
- **Code Quality**: clean, well-structured, documented

Analysis and Understanding (35%)

- **Performance Analysis**: thorough benchmarking and comparison
- **Parameter Studies**: understanding of algorithm sensitivity
- **Trade-off Analysis**: speed vs. accuracy, memory vs. quality
- **Critical Evaluation**: strengths and limitations assessment

Presentation and Communication (25%)

- **Visualization Quality**: clear, informative plots and comparisons
- **Report Clarity**: well-structured analysis and conclusions
- **Technical Writing**: precise mathematical and algorithmic descriptions
- **Innovation**: creative extensions or improvements to basic algorithms

Timeline and Deliverables

Lab Session Structure

- **Week 1**: Exercises 1–4 (Foundations)
- **Week 2**: Exercises 5–8 (Advanced Techniques)
- **Week 3**: Integration, optimization, and report writing

Milestone Checkpoints

- End of Week 1: core algorithms working
- Mid-Week 2: advanced methods implemented
- End of Week 2: performance analysis complete

Final Deliverables

- Complete Jupyter notebook with all implementations
- Comprehensive visualization suite
- Technical report (8–12 pages)
- Performance benchmark summary
- Classical vs. modern comparison study

Submission Details

- Due: [Insert Date]
- Format: GitHub repository or ZIP archive
- Include: code, report, visualizations, data

Success Strategy

Start early, test frequently, document thoroughly. Focus on understanding rather than just completion. The goal is deep comprehension, not just working code.

Resources and Support

Technical Resources

- **Documentation**: `NumPy`, `OpenCV`, `PyTorch` official docs
- **Reference Implementations**: `SciPy`, `scikit-image` for comparison
- **Datasets**: provided test images and synthetic data generators
- **Code Templates**: skeleton classes with method signatures

Academic References

- Lecture slides and expanded chapter text
- Szeliski: *Computer Vision: Algorithms and Applications*
- Gonzalez & Woods: *Digital Image Processing*
- Original papers: Canny (1986), Tomasi & Manduchi (1998), etc.

Getting Help

- **Office Hours**: [Day/Time] for technical questions
- **Forum**: course discussion board for collaboration
- **TA Support**: available for debugging and conceptual help
- **Peer Collaboration**: encouraged for understanding, not code sharing

Final Thoughts: From Implementation to Innovation

The Journey Ahead

Today's implementations are tomorrow's building blocks. The algorithms you code by hand will deepen your understanding of both classical foundations and modern innovations.

Key Learning Objectives

- **Mathematical Intuition**: feel the math through code
- **Performance Awareness**: understand computational trade-offs
- **Quality Assessment**: develop critical evaluation skills
- **Innovation Preparation**: build foundation for advanced methods

Beyond the Lab

These implementations connect to cutting-edge research: neural architecture search, efficient convolutions, learned image processing, and interpretable AI. Your foundation today enables innovation tomorrow.

Let's begin building the future of computer vision!

Questions before we start coding?