COMPUTER VISION

FUNDAMENTALS

Owner	Brira Abid
Submission	The Knowledge Engine
Date	12/7/2025

Lecture #1: How Robots See

- All computer vision systems begin with pixel data
- Understanding this foundation is essential for robotics perception

Lecture #2: Edge Detection - Finding Shapes

- Edges = places where pixel values change significantly
- Kernels = small number arrays that slide across images to detect changes

INTRODUCTION

Welcome to this AI-Native Textbook

This interactive guide teaches computer vision fundamentals through hands-on experiments. Each chapter includes:
✅ Theory explanations
✅ Interactive Colab notebooks
✅ AI-assisted learning prompts
✅ Real-world applications

How to Use This Book:

1. Read each chapter explanation
2. Click the Colab links to run code
3. Experiment with variables
4. Ask AI assistants questions
5. Try the challenges!

Perfect for beginners interested in robotics and AI.

Chapter 1: How Robots See - The Digital Eye

Interactive Lesson 1

In this interactive notebook, you'll discover:
1. How images are just grids of numbers
2. What "pixels" really mean to a computer
3. Try modifying the code yourself!

NoteBook Explanation

👁️ My First Discovery: Pixels are Numbers

When you open the Colab notebook and run the code, you'll experience something fundamental about computer vision:

We humans see images as objects, colors, and familiar shapes - a cat, a tree, a smiling face. But computers? They see something completely different.

For a computer, every picture is just a matrix of numbers. Each tiny dot (pixel) gets assigned a numerical value. Black isn't "black" to a robot - it's the number 0. White isn't "white" - it's 255. Every color, every shade, every detail in between is just another number in a massive grid.

🎨 From Color to Grayscale: Simplifying Vision

Most basic computer vision starts with grayscale because it's simpler to process:

- 0 = Pure black (the darkest possible)
- 255 = Pure white (the brightest possible)
- 1-254 = 254 shades of gray in between

Think of it like this: As the numbers increase from 0 to 255, the pixels transition from midnight black → dark gray → medium gray → light gray → bright white.

🔢 Interactive Learning: See the Numbers Become an Image

In our Colab notebook, you're seeing this transformation in real-time:

1. A 5×5 number matrix (raw data the computer sees)
2. A visual representation (what humans understand)

The magic happens when the computer translates between these two views. It's like watching a paint-by-numbers kit come to life!

🧪 Your Turn to Experiment!

The best way to learn is by doing. Try these modifications in the Colab code:

```
# TRY CHANGING THESE NUMBERS:

tiny_image = np.array([

    [0, 0, 0, 0, 0],      # All black row

    [255, 255, 255, 255, 255],  # All white row
```

```
    [128, 128, 128, 128, 128],  # All gray row
    [0, 64, 128, 192, 255],     # Gradient from black to white
    [255, 192, 128, 64, 0]      # Reverse gradient
])
```

Experiment Ideas:

1. Make a checkerboard pattern (alternate 0 and 255)
2. Create a diagonal line of white across black
3. Try numbers above 255 - what happens?
4. What if you use negative numbers?

🤔 Reflection Questions:

1. Before running the code: Did you expect images to be stored this way?
2. After seeing the results: How does this change your understanding of "seeing" for machines?
3. Real-world connection: Where have you seen grayscale images used in everyday life? (Hint: medical scans, old photographs, security cameras)

Chapter 2:
Edge Detection - How Robots Find Shapes

Learning Objectives:
- Understand what "edges" mean in computer vision
- Learn how robots detect boundaries by comparing pixels
- See edge detection in action with interactive experiments
- Connect edge detection to real-world robotics applications

The Big Idea: Change Detection

My Aha Moment:

Edges aren't "lines" to computers - they're places where pixel values CHANGE dramatically.

PART 1: What Is an "Edge" Really?

The Human Perspective vs. The Computer Perspective

When we look at the world, we see edges as clear lines and boundaries. We recognize the outline of a book on a table, the frame of a doorway, or the silhouette of a person against a wall. Our brains have learned since childhood to identify these visual separations.

But here's the fascinating truth: Computers don't see this way at all.

How Robots Actually "See" Edges

For a robot or any computer vision system, there are no "lines" in the traditional sense. Instead:

1. Everything is numbers - Remember from Chapter 1 that images are just grids of pixel values (0-255 for grayscale)
2. Edges = Significant changes in these numbers - When pixel values change dramatically from one position to the next
3. It's all about comparison - A computer asks: "How different is this pixel from its neighbor?"

The Simple Mathematics Behind the Magic

Let's make this concrete with numbers:

- Situation A (No edge): Pixel 1 = 100, Pixel 2 = 105
  Difference = 5 → Too small! No edge here.
- Situation B (Edge!): Pixel 1 = 50, Pixel 2 = 200
  Difference = 150 → Big change! This is an edge.

Edges aren't just between black and white. They can be between any two different values:

- Dark gray (50) to medium gray (150) = Edge
- Light gray (200) to white (255) = Edge

- Medium gray (120) to slightly lighter gray (130) = Maybe, maybe not (depends on our threshold/significance)

What Counts as "Significant"?

We use something called a threshold to decide what counts as an edge:

- High threshold (e.g., 100): Only very strong changes are edges
- Low threshold (e.g., 20): Even small changes are considered edges

This threshold is like setting the sensitivity of our edge detector. Just like you might adjust the sensitivity of a motion sensor, we can adjust how sensitive our edge detection is.

> Explore it more in Collab : Hackathon1_ch2.ipynb

PART 2: The Simple Math - Finding Edges Between Neighbors

The Core Idea: Look Left, Look Right, Compare

Edge detection at its simplest is just comparing neighboring pixels. That's it! No complex formulas, no magic - just asking one simple question:

"How different is this pixel from the one next to it?"

Let's Walk Through Your Example Step by Step

Remember the rectangle image from our experiment? Let's look at what **happens at the boundaries:**

At Pixel Position 19 → 20:

```
The change: 0 → 255
Difference: 255 - 0 = 255
Is this big? ABSOLUTELY!
This is an EDGE.
```

At Pixel Position 79 → 80:

```
The change: 255 → 0
Difference: 0 - 255 = -255
Is this big? ABSOLUTELY!
This is ALSO an EDGE.
```

Notice something interesting? Both transitions are edges, but they give us different signs:

- Position 19→20: Difference = +255 (positive)
- Position 79→80: Difference = -255 (negative)

What the sign tells us:

- Positive difference → Going from DARK to BRIGHT
- Negative difference → Going from BRIGHT to DARK

PART 3: The Kernel - The Robot's Edge-Finding Tool

What IS a Kernel?

Think of a kernel as a tiny magnifying glass with special rules that the computer slides across an image. It's not magic - it's just a small set of numbers that tells the computer how to look at pixels and what to do with them.

Simple Analogy:

Imagine you're checking if a surface is slanted. You might:

1. Put your left hand down
2. Put your right hand down
3. Compare the heights

The kernel `[1, 0, -1]` does exactly this for pixels:

- Left hand = ×1
- Ignore the middle = ×0

- Right hand = ×(-1)

Why Do We Need Kernels?

1. Consistency - Same rules applied everywhere
2. Flexibility - Different kernels find different things
3. Precision - Can detect edges at specific orientations

In example we have

Kernel: [1, 0, -1]

What it means: "Take LEFT pixel, ignore MIDDLE, subtract RIGHT pixel"

Step 1: Align pixels

our pixels: [0, 0, 0, 255, 255, 255]

Positions: 0 1 2 3 4 5

Step 2: Multiply Pixels

Pixel at position 2 is (0) so, × 1 = 0

Pixel at position 3 (255) × 0 = 0

Pixel at position 4 (255) × (-1) = -255

Step 3: Add them up

0 + 0 + (-255) = -255

Step 4: Interpret the result

-255 = A STRONG EDGE!

What Does -255 Mean?

LEFT pixel is DARKER than RIGHT pixel means we're going: DARK (left) → BRIGHT (right)

PART 4: Visualizing Edge Detection

Now, we're creating a visual map of where our computer detects edges. Think of it as:

1. The computer analyzes the image pixel by pixel
2. Wherever it finds a significant change (using our kernel `[1, 0, -1]`)
3. It marks that location as an "edge point"
4. We create a new image showing ONLY these edge points

From Math to Visualization

We're translating numerical calculations into something visually meaningful:

```
Numerical result: -255, 0, 150, -200, etc.

Visual representation: Bright pixel, Dark pixel, etc.
```

We logically expect two vertical edge lines:

1. Line 1: Where black becomes white (left side)
2. Line 2: Where white becomes black (right side)

The Surprising Result

But when we run our edge detection and visualize it, we see FOUR bright vertical lines instead of two!

This creates an important question:

"If the computer is detecting edges where pixel values change dramatically, and we only have two changes (black→white and white→black), why do we see FOUR lines in our visualization?"

Let's Walk Through the Math

Let's look at the LEFT edge of the rectangle:

```
Pixels across the left edge:

Position: ...  18   19   20   21   ...

Values:   ...   0    0   255  255  ...

              Black Black White White
```

What happens when our kernel `[1, 0, -1]` slides across:

At Position 18: (kernel covers pixels 17, 18, 19)

```
Pixels: [0, 0, 0]  (all black)

Calculation: (0×1) + (0×0) + (0×-1) = 0

Result: 0 (no edge)
```

At Position 19: (kernel covers pixels 18, 19, 20)

```
Pixels: [0, 0, 255]

Calculation: (0×1) + (0×0) + (255×-1) = -255

Result: -255 (EDGE DETECTED!) ← First line
```

Double Detection!

Our single edge (Black→White) gets detected TWICE:

1. At position 19: Kernel straddles the edge `[0, 0, 255]`
2. At position 20: Kernel also straddles the edge `[0, 255, 255]`

Same Thing Happens on the RIGHT Edge.

Why Don't We See This as a Problem?

Actually, in real edge detection, we usually clean this up by:

1. Looking for peaks (maximum values) instead of all detections
2. Using non-maximum suppression (keeping only the strongest response)

PART 5: Interactive Experiment - Play with Edge Detection!

Your Turn to Take Control!

Now comes the most exciting part - you get to be the scientist! Up until now, we've been exploring pre-set examples. But true learning happens when you experiment, play, and discover for yourself.

What You Can Experiment With:

In this interactive section, you'll find three key variables you can adjust:

1. Circle Properties:

- `circle_radius`: Make the circle bigger or smaller
- `circle_brightness`: Change how bright the circle appears
- `background_value`: Alter the darkness of the background

2. Shape Switching:

- `shape_choice`: Try different shapes! Change it to:
  - `"circle"` for round edges
  - `"triangle"` for sharp corners
  - `"rectangle"` for straight edges

The Goal: Discover Through Doing

Don't just change numbers randomly - ask questions and test them:

Question 1: *"What happens to edge detection when the circle is very faint against the background?"*
→ Try making `circle_brightness` close to `background_value`

Question 2: *"How does edge detection handle sharp corners vs smooth curves?"*
→ Compare `"triangle"` with `"circle"`

What to Watch For:

As you make changes, pay attention to:

1. Edge Strength: Do edges become brighter or fainter?
2. Edge Continuity: Are edges broken or continuous?
3. False Positives: Do unexpected edges appear?
4. Shape Recognition: Can you still recognize the shape from just the edges?

Real-World Applications - Edge Detection in Action

From Theory to Reality

Now that you understand how edge detection works, let's explore where this technology actually changes our world. Edge detection isn't just a classroom exercise - it's a fundamental tool that powers amazing technologies around us.

Self-Driving Cars

What they do: Detect lane markings, identify other vehicles, recognize pedestrians
How edges help: Finding the boundaries between road and not-road, detecting vehicle outlines

Your Smartphone

Face Unlock: Finding facial features (eyes, nose, mouth) by detecting edges
Photo Editing: Auto-enhancing photos by identifying subject boundaries
Document Scanning: Detecting paper edges to crop documents perfectly

Medical Imaging

Tumor Detection: Finding boundaries between healthy and unhealthy tissue
Bone Fracture Analysis: Detecting edges of bone cracks in X-rays
Organ Segmentation: Outlining organs in CT or MRI scans

The Big Picture

Every time a computer "sees" something in the real world, edge detection is often the first step. It's like giving machines the ability to find outlines, which is the foundation for all higher-level understanding.

Challenges for You - Take It Further!

Ready for More?

If you've enjoyed this journey into edge detection, here are some challenges to test and expand your skills. Try these in the Colab notebook!

1. The Disappearing Circle: Make the circle brightness equal to the background. What happens to the edges?
2. Noise Test: Add random "noise" (small random numbers) to your image. Do false edges appear?
3. Shape Classifier: Can you write code that distinguishes circles from triangles using only their edge patterns?