CS 5330: Pattern Recognition and Computer Vision

Northeastern University

Lab 8: Edge Detection

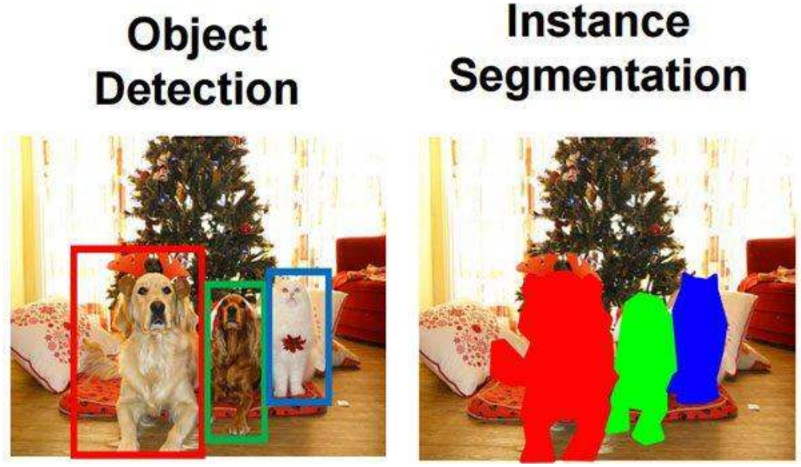
* Contributed by Fall 2024 TAs: Byunghyun Ko, Yihan Wang and Taiwei Cui

Edge Detection

1. Intro to Edge Detection
2. Sobel Edge Detection
3. Canny Edge Detection
4. Code Examples

Introduction to Edge Detection

- Edge detection: process of identifying and locating sharp discontinuities in an image – these reflect significant alterations in pixel intensity, which often correspond to object boundaries
- Application of edge detection
  - Object recognition
  - Image segmentation
  - Pattern recognition



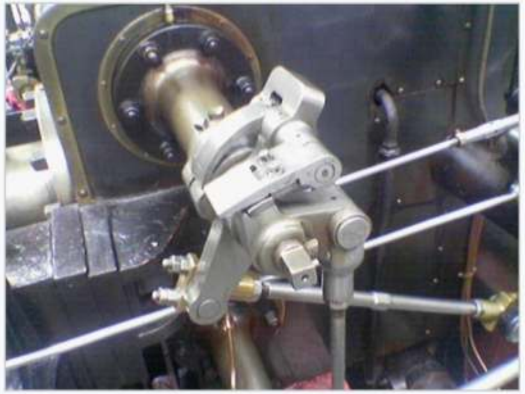
https://pyimagesearch.com/2018/11/26/instance-segmentation-with-opencv/

Sobel Edge Detection (Sobel Operator)

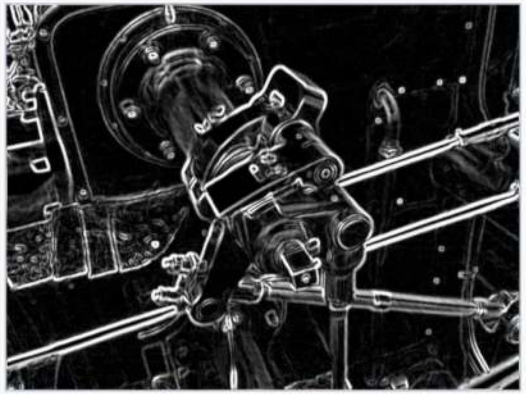
- Sobel Operator: uses a pair of convolution kernels to calculate the gradient in horizontal and vertical directions.
- Highlights changes in intensity, providing both direction and magnitude of the edges.
- Why use Sobel?
  - **Efficient**: Sobel is computationally efficient and is good for detecting edges in simple images.
  - **Edge Direction**: It provides information not only about where the edges are but also about their orientation (horizontal, vertical, or diagonal).

https://en.wikipedia.org/wiki/Sobel_operator

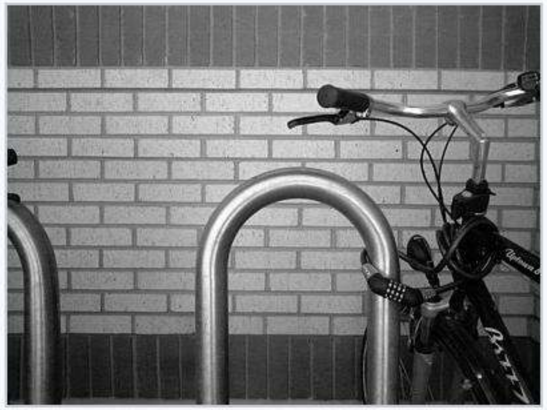
Sobel Edge Detection (Sobel Operator)



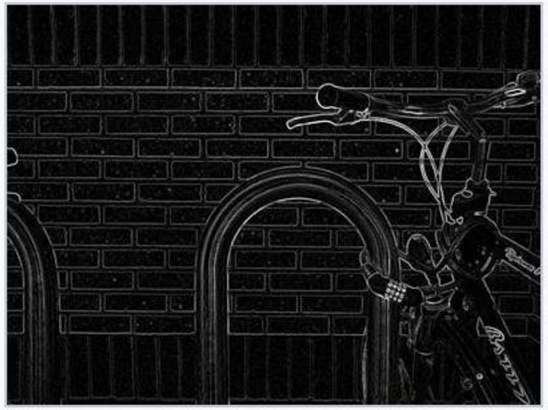
A color picture of an engine



The Sobel operator applied to that image



Grayscale test image of brick wall and bike rack



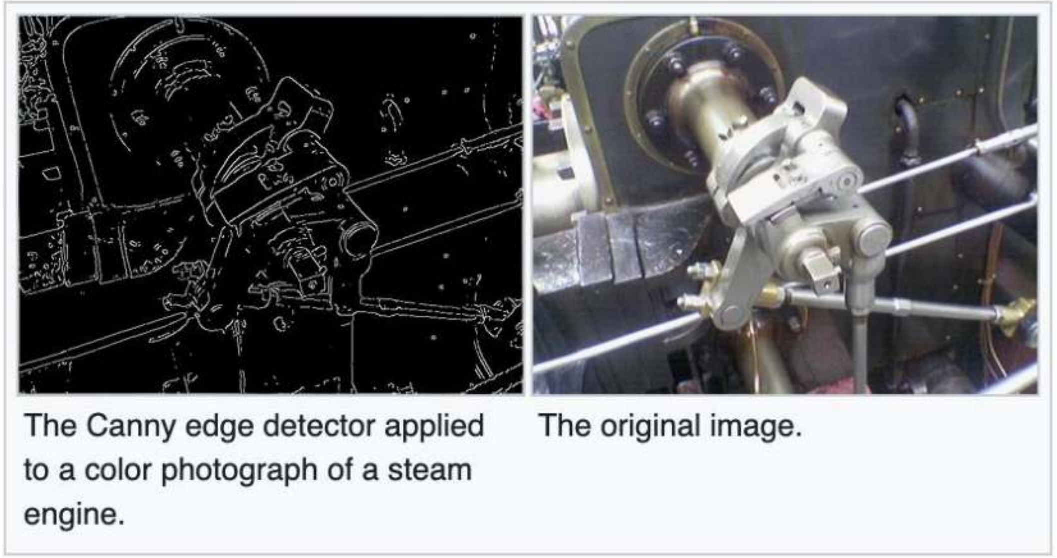
Normalized gradient magnitude from Sobel–Feldman operator

https://en.wikipedia.org/wiki/Sobel_operator

Canny Edge Detection

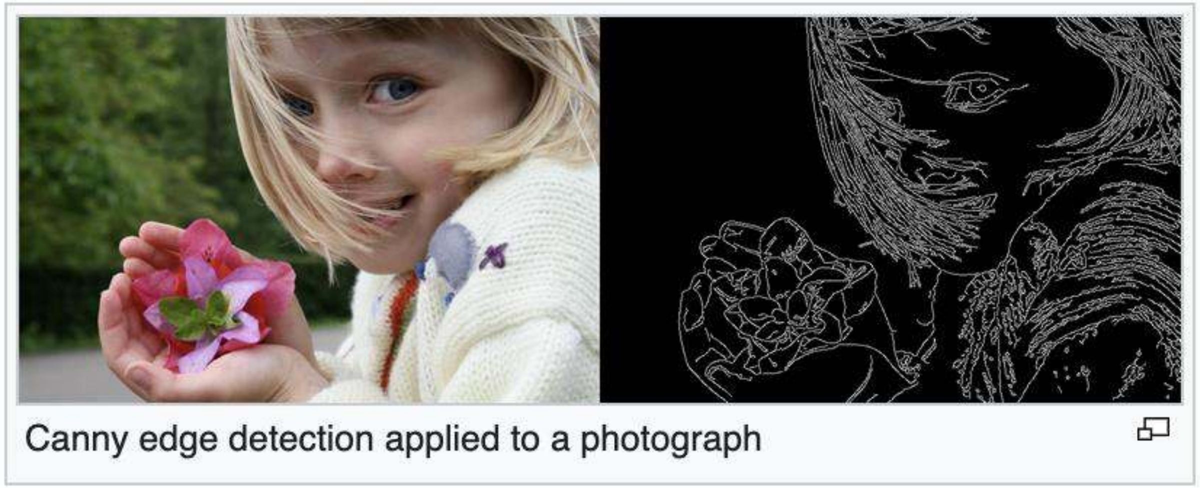
- Canny Edge Detection: multi-step algorithm to detect a wide range of edges in images.
- More refined than Sobel and commonly used in practice.
- Steps involved: Noise reduction, gradient calculation, non-maximum suppression, double threshold, edge tracking by hysteresis.
- Why use canny?
  - **Accuracy**: Canny is very precise in detecting edges and reducing noise.
  - **Multi-step Process**: Each step (smoothing, gradient, non-maximum suppression, thresholding) refines the edge detection process, leading to more robust results.

Canny Edge Detection



The Canny edge detector applied to a color photograph of a steam engine.

The original image.



Canny edge detection applied to a photograph

https://en.wikipedia.org/wiki/Canny_edge_detector

Sobel vs Canny

Feature	**Sobel**	**Canny**
Gradient Calculation	Calculates intensity gradient	Calculates gradient, applies multiple filters
Edge Detection	Simple and fast	More accurate and robust
Noise Sensitivity	Sensitive to noise	Uses Gaussian filtering to reduce noise
Edge Quality	Produces thicker edges	Produces thin, well-defined edges
Control	No thresholds	User-defined thresholds for fine control

Sobel Edge Detection Example

Code

```
# Load an image
image = cv2.imread('input_image.jpg', 0)

# Apply Sobel filter
sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)  # Horizontal edges
sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)  # Vertical edges

# Combine Sobel X and Y
sobel_combined = cv2.sqrt(sobel_x**2 + sobel_y**2)

# Display result
cv2.imshow("Sobel Edge Detection", sobel_combined)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.Sobel(src, ddepth, dx, dy, ksize=3):

- **src**: Source image
- **ddepth**: Desired depth of the destination image
- **dx**: Order of the derivative x
- **dy**: Order of the derivative y
- **ksize**: Kernel size

sobel_combined = cv2.sqrt(sobel_x2 + sobel_y**2):**

- Combine the results of the horizontal and vertical edge detection to calculate the overall edge magnitude.

Canny Edge Detection Example

Code

```
# Load an image
image = cv2.imread('input_image.jpg', 0)

# Apply Canny edge detection
edges = cv2.Canny(image, 100, 200)

# Display result
cv2.imshow("Canny Edge Detection", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.Canny(image, threshold1, threshold2):

- **image**: Source image
- **threshold1**: Lower threshold for weak edges (keep edges if connected to strong ones).
- **threshold2**: Upper threshold for strong edges (keep all pixels with gradients above this value).

Tip on choosing threshold values in Canny

1. Start by experimenting with a **high threshold2** (strong edge threshold) and a **moderate threshold1**. Gradually lower threshold1 to capture finer edges without increasing noise too much.
2. Adjust thresholds based on the **lighting** and **contrast** of the image. For low-contrast images, lowering both thresholds may be necessary to detect the edges clearly.

Summary

- **Sobel edge detection** is simple and fast, good for detecting both horizontal and vertical edges, but produces thicker edges and is sensitive to noise.
- **Canny edge detection** is more advanced as it combines noise reduction, gradient calculation, non-maximum suppression, etc. to detect strong and weak edges. It gives thinner, well-defined edges but requires “experimenting” with two thresholds.
- **Sobel** is efficient for simple images with clear edges, while **Canny** excels in detecting fine, detailed edges (such as noisy or complex images).
- However, with all the work that goes behind **Canny**, it can take more processing time than **Sobel**.