LE/ESSE 2220 Algorithmic and Computational Methods

Lab 6: Feature Detection and LED Matrix Visualization

(Fall 2025-2026)

Z. ARJMANDI (ZAHRARJ@YORKU.CA)

YORK U

Review

Categories of Feature Detectors

Type	Methods
Edge Detectors	Sobel, Prewitt, Laplacian, Canny
Corner Detectors	Harris, FAST
Blob Detectors	LoG, DoG, DoH
Scale-Invariant Feature Detectors	SIFT, SURF
Fast / Binary Features	FAST, BRIEF, ORB

Common Edge Detectors

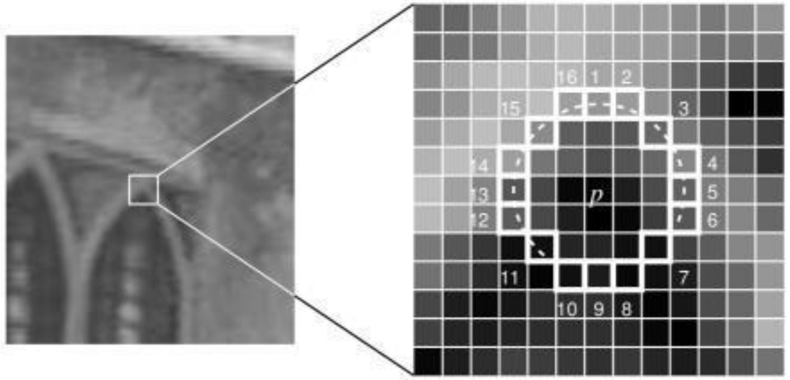
- Edges mark regions where intensity changes sharply.

Original Sobel Canny

Method	Concept	Notes	OpenCV Function
Sobel	Gradient-based (1st derivative)	Gives direction + strength of edges	cv2.Sobel()
Laplacian	2nd derivative	Highlights rapid changes (edges both sides)	cv2.Laplacian()
Canny	Multi-stage process	Best general-purpose edge detector	cv2.Canny()

Common Corner Detectors

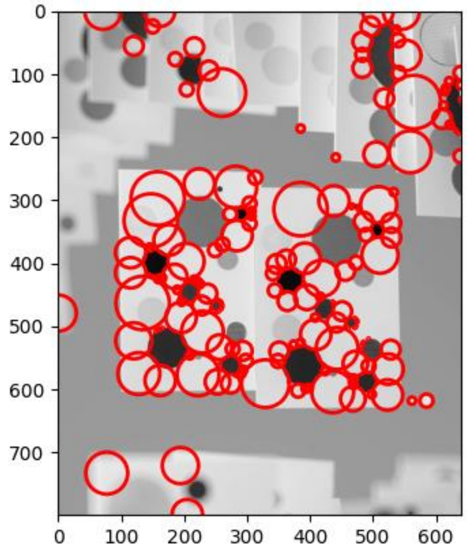
- A **corner** is a point where **gradients in both directions (x and y)** change significantly.



Method	Concept	Notes	OpenCV Function
Harris Corner Detector	Based on **intensity changes in both x and y directions**	Detects corners where gradients vary strongly in all directions. Sensitive to the k parameter (typically 0.04–0.06).	cv2.cornerHarris()
FAST (Features from Accelerated Segment Test)	Compares a pixel's intensity with 16 surrounding pixels in a circle. A corner is detected if several consecutive pixels are brighter or darker.	Extremely fast; ideal for real-time use. Forms the basis of ORB.	cv2.FastFeatureDetector_create()

Common Blob Detectors

- Finds **regions (not just points or lines)** in an image that are **brighter or darker than their surroundings,** basically, *"blobs"* of consistent intensity.

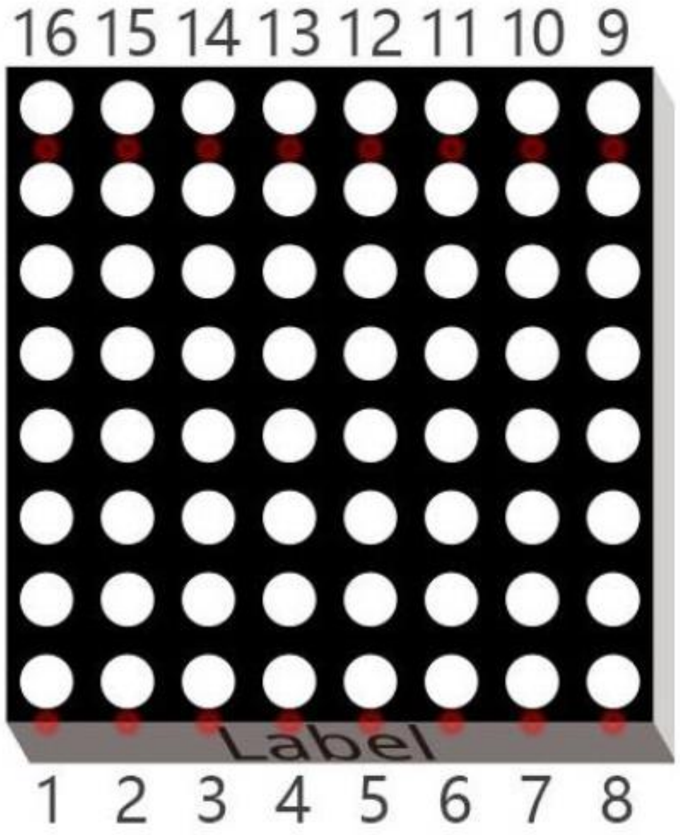


Method	Notes	OpenCV Function
DoG (Difference of Gaussians)	Approximates LoG but faster. Used internally by SIFT for scale-invariant blob detection.	Part of cv2.SIFT_create()
SimpleBlobDetector	Easy to use; customizable filters (area, circularity, color, etc.).	cv2.SimpleBlobDetector_create()

Lab 5

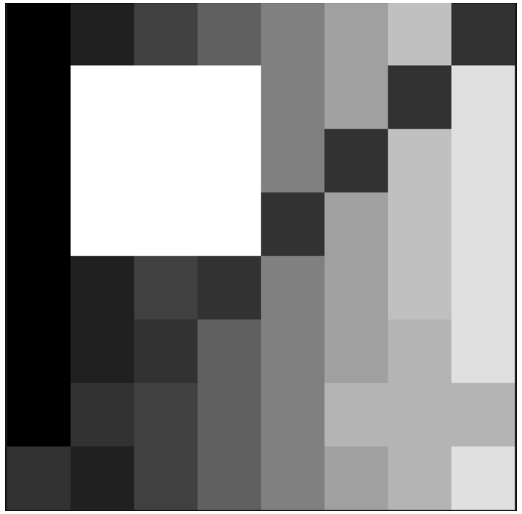
Lab 6: Feature Detection and LED Matrix Visualization

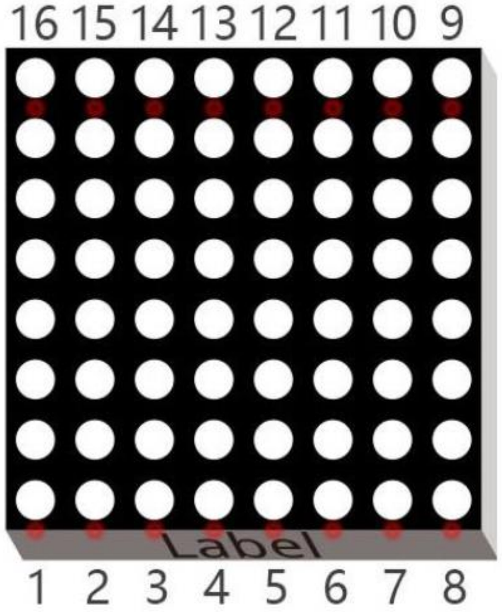
LED Matrix



Lab 6: Feature Detection and LED Matrix Visualization

- An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).





Lab 6: Feature Detection and LED Matrix Visualization

- If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.

1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	1	0
1	0	1	0	0	1	0	1
1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

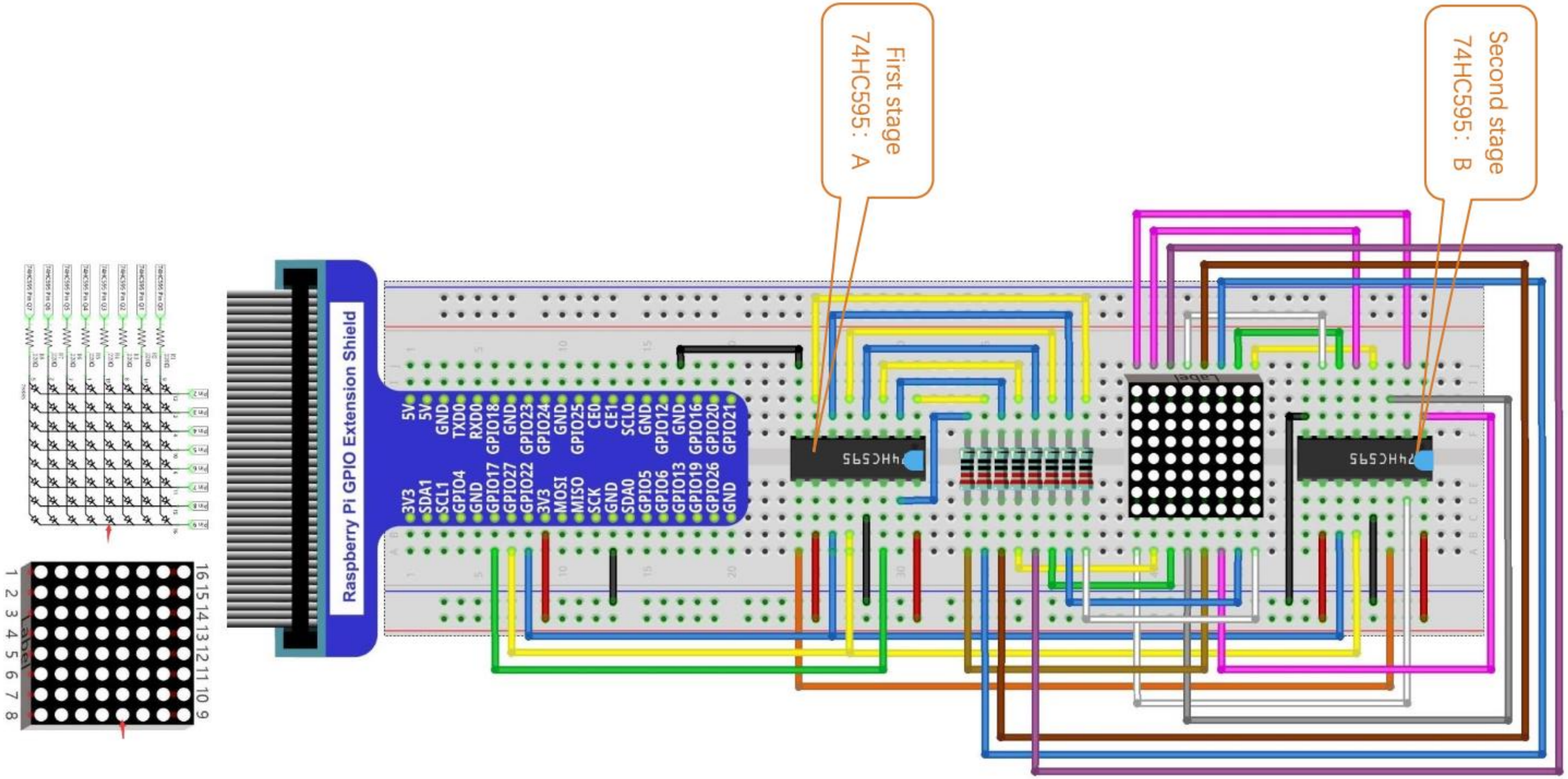
Base-16 numbering system

Column	Binary	Hexadecimal
1	0001 1100	0x1c
2	0010 0010	0x22
3	0101 0001	0x51
4	0100 0101	0x45
5	0100 0101	0x45
6	0101 0001	0x51
7	0010 0010	0x22
8	0001 1100	0x1c

```
pic = [0x1c, 0x22, 0x51, 0x45, 0x45, 0x51, 0x22, 0x1c]# data of smiling face
```

Lab 6: Feature Detection and LED Matrix Visualization

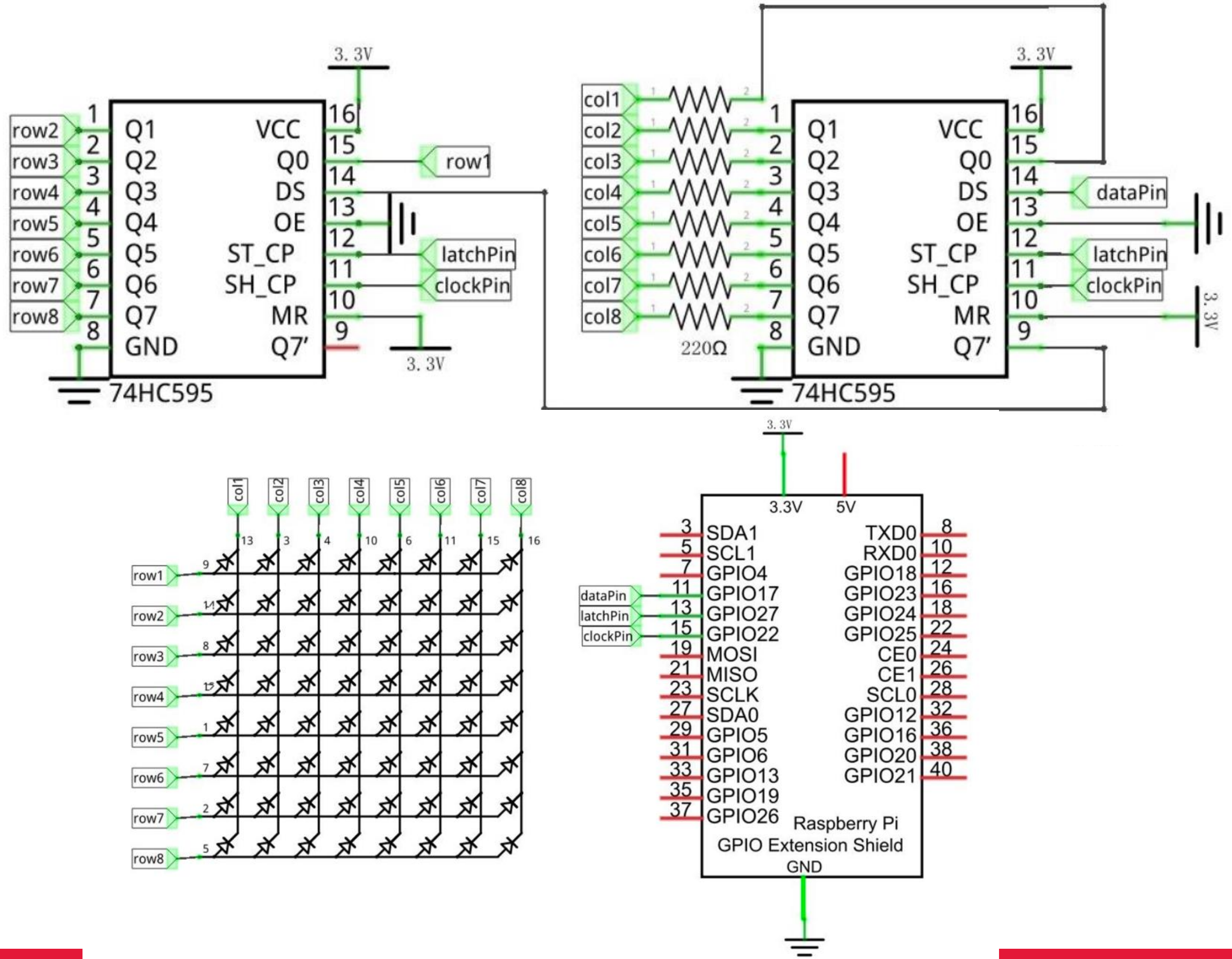
- Build the Circuit



YORK U

Lab 6: Feature Detection and LED Matrix Visualization

- For more details on this circuit, please refer to Tutorial Chapter 21 (available on E-Class).



Lab 6: Steps

- **1- Test the existing code**
  - Run the provided code (LEDMatrix.py) on the Raspberry Pi.
  - Observe the 8×8 LED matrix showing the smiling face.

- **2- Image Processing in OpenCV**
  - **On windows:** Write a new Python file (e.g., image_to_led.py).
  - **Load the small 8×8 grayscale image (Download from E-Class > Lab6 > lab6_8x8_gray.png)**

- **3- Apply the following detectors:**

  You may use the example code provided on (eClass> InClass10) or any reference code from the official OpenCV documentation

  - **Edges:** Canny
  - **Edges:** Sobel

Lab 6: Steps

- **4- Convert the result to binary (8×8)**
  - Threshold the image so pixels are 0 or 1:

```
_, binary = cv2.threshold(img, 127, 1, cv2.THRESH_BINARY)
```

- **5- Convert each row to hexadecimal**
  - Hexadecimal is a base-16 numbering system:

```
hex_values = []
for row in binary:
    bits = ''.join(str(int(b)) for b in row)
    hex_values.append(hex(int(bits, 2)))
```

- **6- Display on the LED Matrix**
  - Take the 8 generated hex values from the OpenCV output.
  - Replace the pic array in LEDMatrix.py with your new values:

```
pic = [0x__, 0x__, 0x__, 0x__, 0x__, 0x__, 0x__, 0x__]
```

YORK U

Report Format (short, personal, verifiable)

Setup (2)

- Attach a **clear photo of your Raspberry Pi circuit** showing the LED matrix and wiring.
- Copy and paste your **final Python code** for:
  - The image-processing part (OpenCV)
  - The LED matrix display code (with your modified pic = [...])
- Include **meaningful comments** in your code explaining:

Observations (5)

- Include **screenshots of your output images** showing the results of each edge detector: **Sobel and Canny (2)**
- Paste the **hexadecimal output** generated for your chosen image (the list of 8 hex values). (1)
- Include a **photo of your LED matrix** displaying the detected features.(2)

Analysis (3)

- What is the main difference between the Sobel, Laplacian, and Canny edge detectors?
- Which edge detector produced the clearest or most meaningful output on your 8×8 image? Why?
- How does the small 8×8 image size affect the accuracy or visibility of detected edges?
- When viewing your LED display, did the orientation match your OpenCV image? If not, explain why.
- How does increasing or decreasing the Canny thresholds affect the detected edges?
- How might this technique (feature extraction + LED visualization) be useful in real embedded or robotic systems?