Intelligence System:

Fundamentals of Computer Vision

NORTON UNIVERSITY

Intelligence System Development

2024 – 2025

Y4E1 – DCS – NU

By: SEK SOCHEAT

Advisor to DCS and Lecturer

Mobile: 017 879 967

Email: socheat.sek@gmail.com

Table of Contents

1. Introduction to Computer Vision

1.1. Definition and Applications

1.2. Overview of the Vision Pipeline

1.3. Fundamental Concepts

• Activity

2. Feature Detection and Extraction

2.1. Edge Detection

2.2. Key-points and Descriptors

2.3. Histogram of Oriented Gradients (HOG)

• Activity

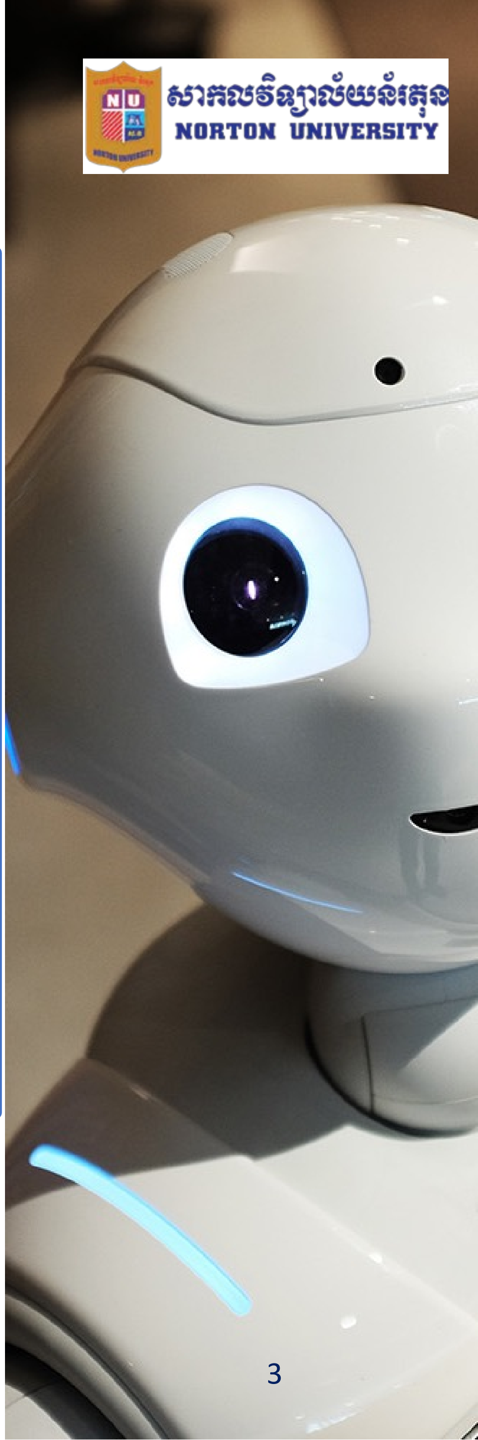
3. Homework

1. Introduction to Computer Vision

1.1. Definition and Applications

- **Computer Vision (CV)** is the field of artificial intelligence and computer science that enables machines to interpret and make decisions based on visual data (images and videos). It aims to simulate the human visual system's capabilities.

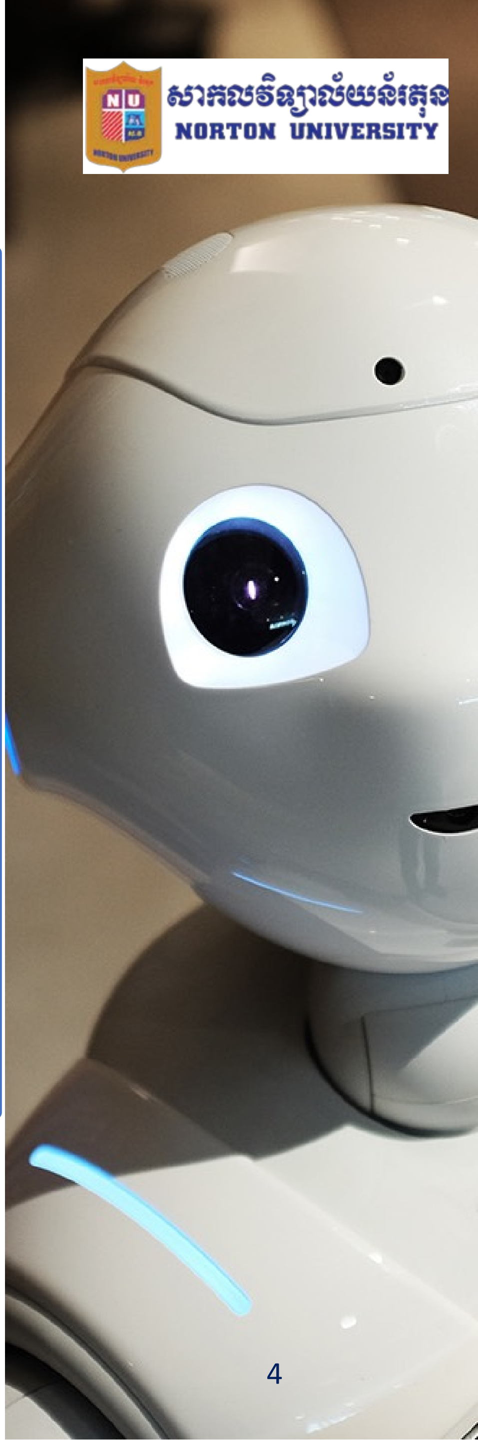
- **Applications** include:
  - Object recognition and detection (e.g., identifying faces in photos).
  - Autonomous vehicles (e.g., recognizing lanes and obstacles).
  - Medical imaging (e.g., detecting tumors in X-rays).
  - Retail (e.g., cashier-less checkouts).
  - Agriculture (e.g., crop health monitoring).
  - Augmented Reality (e.g., overlaying objects in real-world environments).



1. Introduction to Computer Vision

1.2. Overview of the Vision Pipeline

1. **Image Acquisition**:
   - Capturing images or video frames using cameras or sensors.
   - Formats include grayscale, RGB (color), or more complex formats like hyperspectral.
2. **Preprocessing**:
   - Improving image quality or extracting regions of interest:
     - Noise reduction (e.g., Gaussian blur).
     - Contrast enhancement.
     - Geometric corrections.
3. **Feature Extraction**:
   - Identifying key components or patterns:
     - Edge detection (e.g., using Sobel or Canny algorithms).
     - Keypoints and descriptors (e.g., SIFT, ORB).



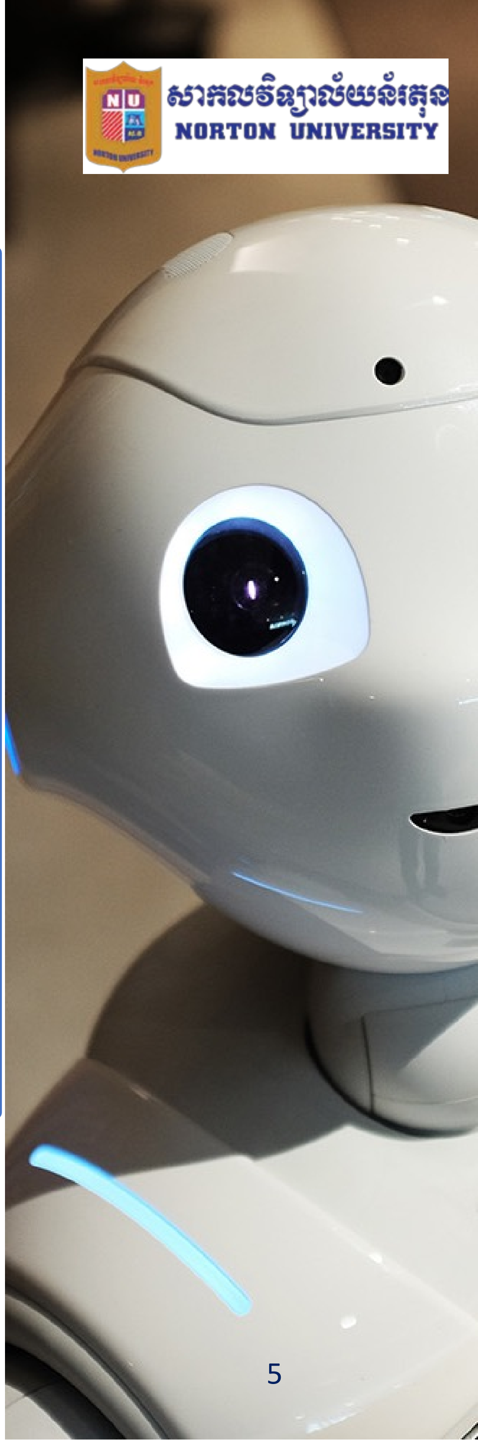
1. Introduction to Computer Vision

1.3. Fundamental Concepts

1. **Pixels**:
    - The smallest units of an image, arranged in a grid. Each pixel holds intensity (grayscale) or color information.

2. **Color Spaces**:
    - Representations of color in images:
        - **RGB** (Red, Green, Blue) – common in screens.
        - **HSV** (Hue, Saturation, Value) – useful for color-based segmentation.
        - **Grayscale** – single intensity value per pixel.

3. **Basic Transformations**:
    - **Grayscale Conversion**: Converts color images to shades of gray, reducing complexity.
    - **Thresholding**: Converts images to binary by setting a pixel intensity cutoff.
        - Example: Otsu's method for adaptive thresholding.



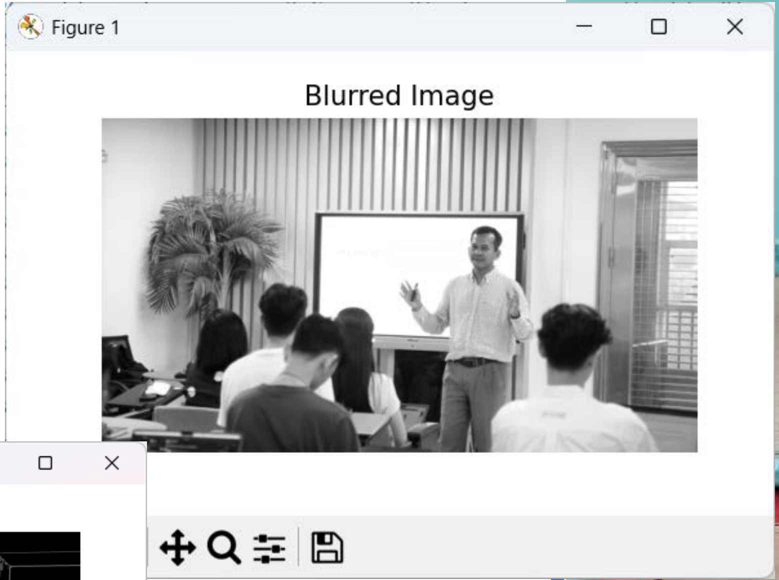
Activity: Convert an Image to Grayscale

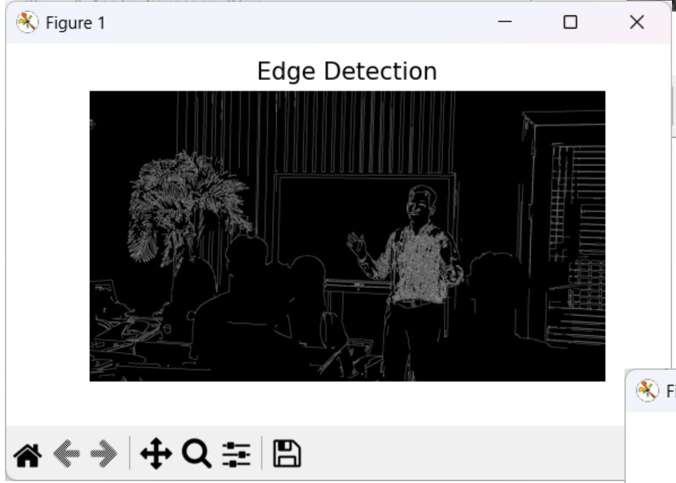
conv_image2grayscal.py > ...

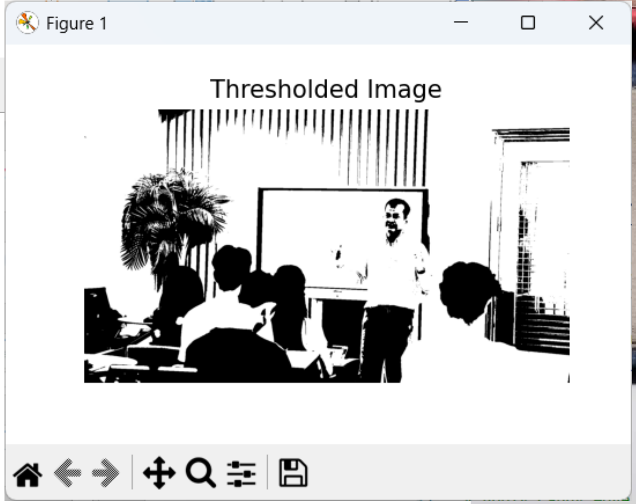
```python
import cv2  # OpenCV library
import matplotlib.pyplot as plt

# Load an image from file
image = cv2.imread('imgs/Sek_Socheat.jpg')  # Replace 'imgs/Sek_Socheat.jpg' with your image file path

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the original and grayscale images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct color display
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Grayscale Image')
plt.imshow(gray_image, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Activity: Convert an Image to Grayscale

```python

# Apply Gaussian Blur
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

# Display the blurred image
plt.imshow(blurred_image, cmap='gray')
plt.title('Blurred Image')
plt.axis('off')
plt.show()


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

# Display the edges
plt.imshow(edges, cmap='gray')
plt.title('Edge Detection')
plt.axis('off')
plt.show()

# Apply binary thresholding
_, binary_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY)

# Display the thresholded image
plt.imshow(binary_image, cmap='gray')
plt.title('Thresholded Image')
plt.axis('off')
plt.show()



cv2.imwrite('grayscale_image.jpg', gray_image)
```







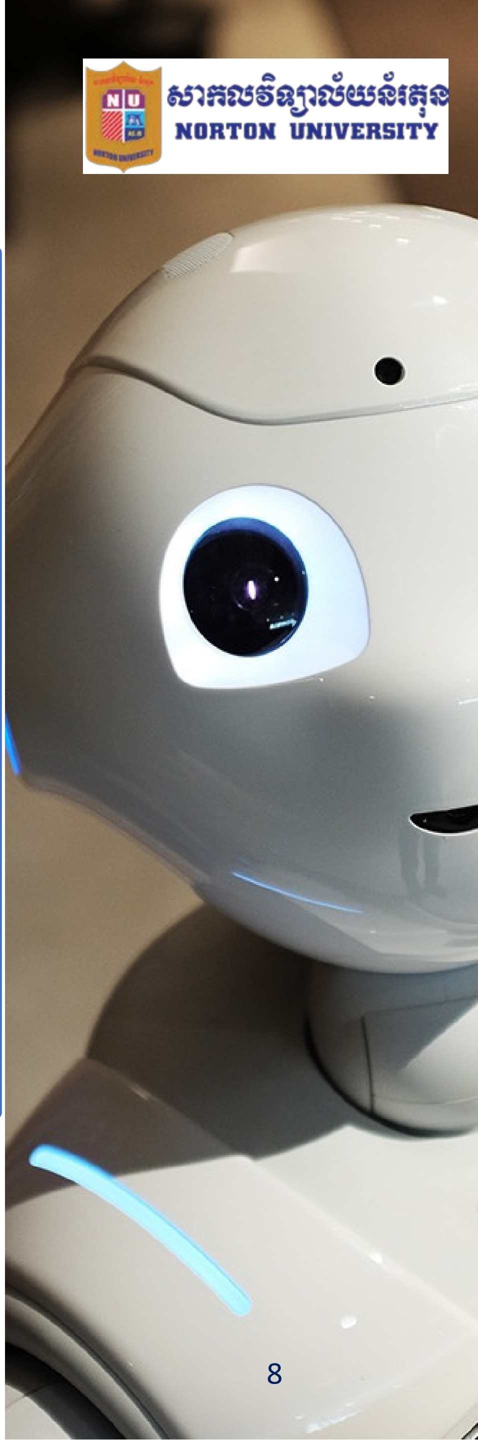
2. Feature Detection and Extraction

2.1. Edge Detection

Feature detection and extraction are essential in computer vision for identifying and describing distinctive elements of an image. These techniques are widely used in applications such as object recognition, image stitching, and motion detection.

A. Sobel Edge Detection

- Sobel operator detects edges using first-order derivatives of the image.
- It computes gradients in the x and y directions.



2. Feature Detection and Extraction

2.1. Edge Detection

B. Canny Edge Detection

- A multi-stage edge detector that uses gradient magnitude and direction, along with thresholding and non-maximum suppression.

C. SIFT (Scale-Invariant Feature Transform)

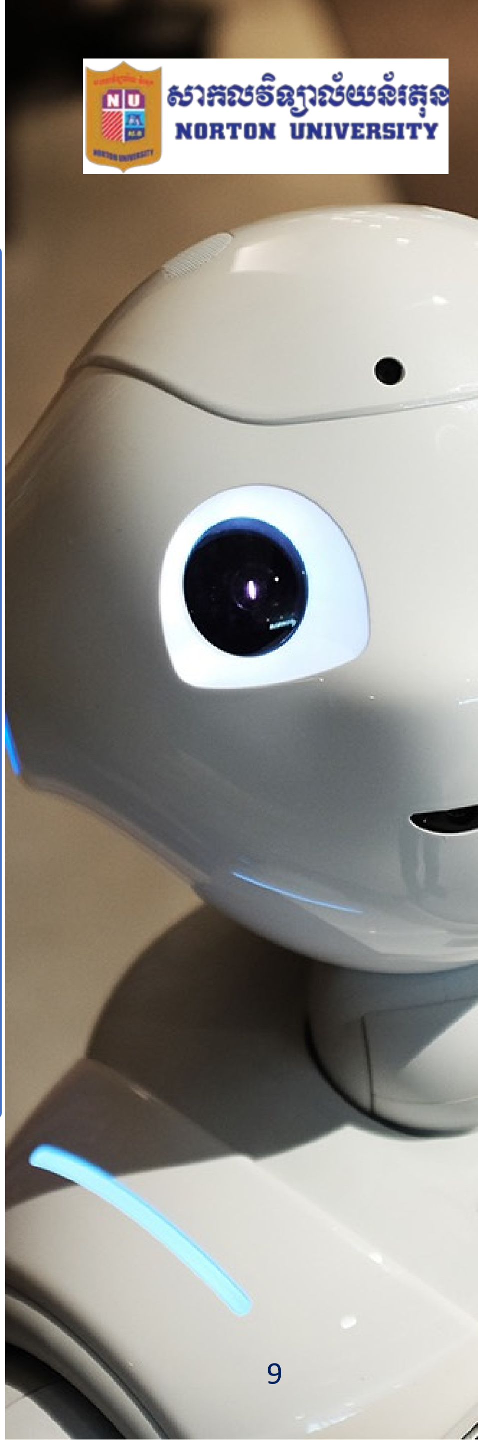
- Detects keypoints and computes descriptors invariant to scale and rotation.

D. SURF (Speeded-Up Robust Features)

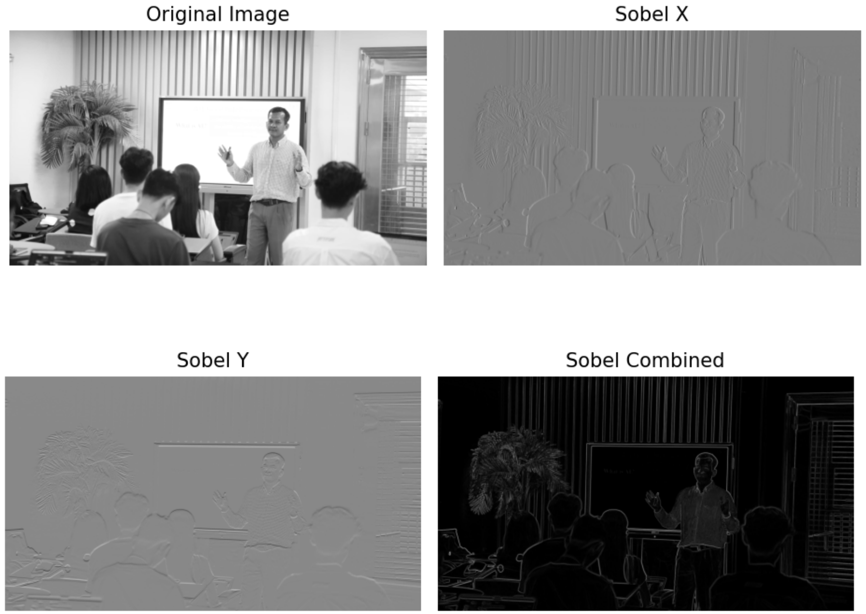
- Similar to SIFT but faster. Requires OpenCV's contrib module.

E. ORB (Oriented FAST and Rotated BRIEF)

- A computationally efficient alternative to SIFT and SURF.

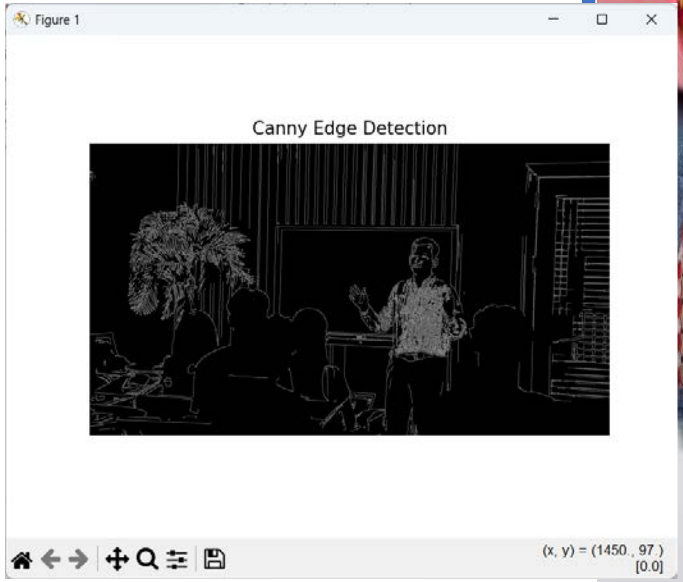


Activity: Sobel Edge Detection



sobel_edge_detection.py > ...

```python
import cv2
import matplotlib.pyplot as plt

# Load an image
image = cv2.imread('imgs/Sek_Socheat.jpg', cv2.IMREAD_GRAYSCALE)

# 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
sobel_combined = cv2.magnitude(sobel_x, sobel_y)  # Combine the two gradients

# Display results
plt.figure(figsize=(15, 5))
titles = ['Original Image', 'Sobel X', 'Sobel Y', 'Sobel Combined']
images = [image, sobel_x, sobel_y, sobel_combined]

for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')

plt.tight_layout()
plt.show()

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

# Display result
plt.imshow(edges, cmap='gray')
plt.title('Canny Edge Detection')
plt.axis('off')
plt.show()
```



Activity: Sobel Edge Detection


```python

# Create a SIFT detector
sift = cv2.SIFT_create()

# Detect and compute keypoints and descriptors
keypoints, descriptors = sift.detectAndCompute(image, None)

# Draw keypoints on the image
image_sift = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(image_sift, cmap='gray')
plt.title('SIFT Keypoints')
plt.axis('off')
plt.show()

# Create an ORB detector
orb = cv2.ORB_create()

# Detect keypoints
keypoints_orb = orb.detect(image, None)

# Compute descriptors
keypoints_orb, descriptors_orb = orb.compute(image, keypoints_orb)

# Draw keypoints on the image
image_orb = cv2.drawKeypoints(image, keypoints_orb, None)

# Display the image with keypoints
plt.imshow(image_orb, cmap='gray')
plt.title('ORB Keypoints')
plt.axis('off')
plt.show()
```

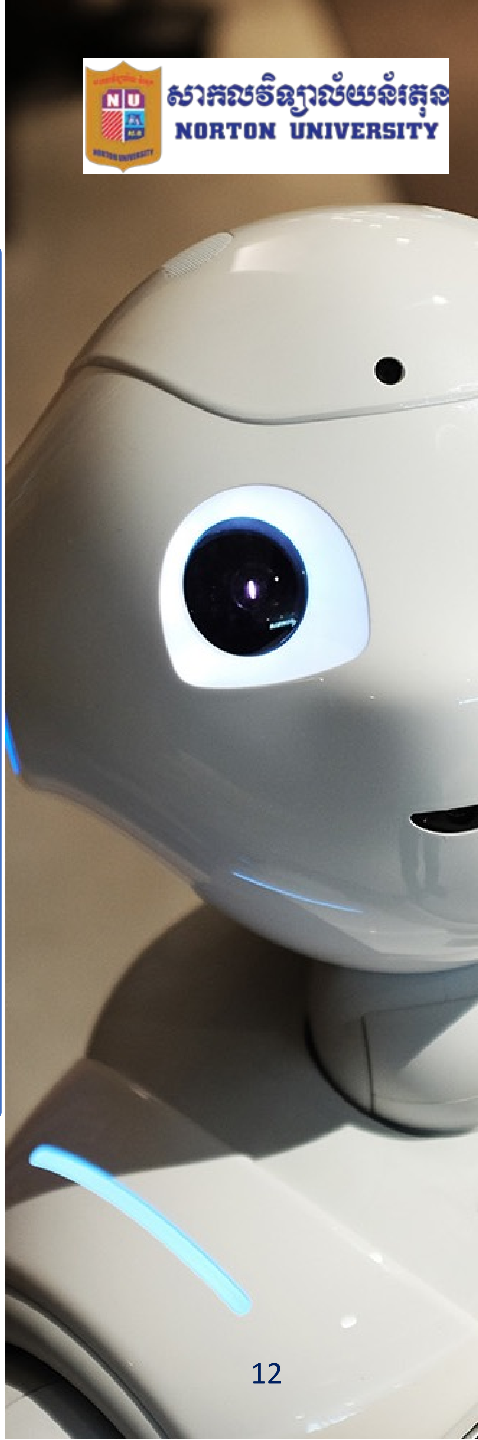
2. Feature Detection and Extraction

2.3. Histogram of Oriented Gradients (HOG)

HOG is used to describe the structure and appearance of an object in an image based on gradient orientations.

Steps:

1. Divide the image into small regions (cells).
2. Compute the gradient orientation histogram for each cell.
3. Normalize the histograms across overlapping blocks.
4. Use the descriptor for tasks like object detection.



Activity: Sobel HOG Detection

sobel_hog_detection.py > ...

```python
# sobel_hog_detection.py
import cv2
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import exposure

def process_image_with_sobel_and_hog(image_path):
    # Load the image in grayscale
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Check if the image is loaded successfully
    if image is None:
        raise FileNotFoundError(f"Image at path '{image_path}' not found or cannot be read.")

    # Apply Sobel filters
    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
    sobel_combined = cv2.magnitude(sobel_x, sobel_y)  # Combine gradients

    # Compute HOG features
    hog_features, hog_image = hog(
        image,
        orientations=9,  # Number of gradient orientation bins
        pixels_per_cell=(8, 8),  # Size of a cell (in pixels)
        cells_per_block=(2, 2),  # Number of cells in each block
        visualize=True,  # Return a visual representation
        block_norm='L2-Hys'  # Block normalization method
    )

```

សាកលវិទ្យាល័យន័រតុន NORTON UNIVERSITY

Example: Sobel HOG Detection

```python

    # Improve the visual appearance of HOG image
    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

    # Display the results
    plt.figure(figsize=(15, 8))
    titles = ['Original Image', 'Sobel Combined', 'HOG Image']
    images = [image, sobel_combined, hog_image_rescaled]

    for i in range(3):
        plt.subplot(1, 3, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(titles[i])
        plt.axis('off')

    plt.tight_layout()
    plt.show()

    return hog_features

# Entry point
if __name__ == "__main__":
    # Provide the path to your image file
    image_path = 'imgs/Sek_Socheat.jpg'  # Replace with the path to your image
    try:
        hog_features = process_image_with_sobel_and_hog(image_path)
        print("HOG features extracted successfully.")
    except Exception as e:
        print(f"Error: {e}")
```

Original Image

Sobel Combined

HOG Image

4. Homework

1. What is computer vision? and list two applications where it plays a crucial role.

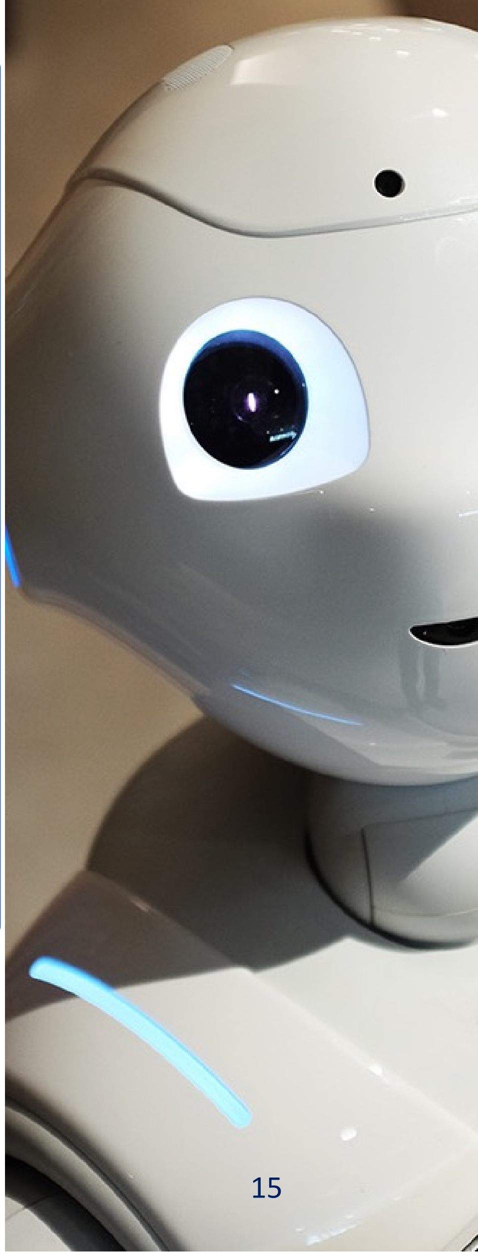
2. What are the main stages of a computer vision pipeline?

3. Why is convolution important in computer vision, and where is it used?

4. How does the Canny edge detection algorithm identify edges in an image?

5. What is the purpose of key-points in computer vision, and how are they typically used?

6. Explain the role of normalization in the computation of Histogram of Oriented Gradients (HOG) features.



Thank you