In [26]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

In [27]:
```
def compute_integral_image(image):
    """
    Compute the integral image (summed area table) of the input image.
    """
    height, width = image.shape
    integral_image = np.zeros((height, width), dtype=np.int32)

    for y in range(height):
        for x in range(width):
            integral_image[y, x] = image[y, x]
            if y > 0:
                integral_image[y, x] += integral_image[y - 1, x]
            if x > 0:
                integral_image[y, x] += integral_image[y, x - 1]
            if y > 0 and x > 0:
                integral_image[y, x] -= integral_image[y - 1, x - 1]

    return integral_image
```

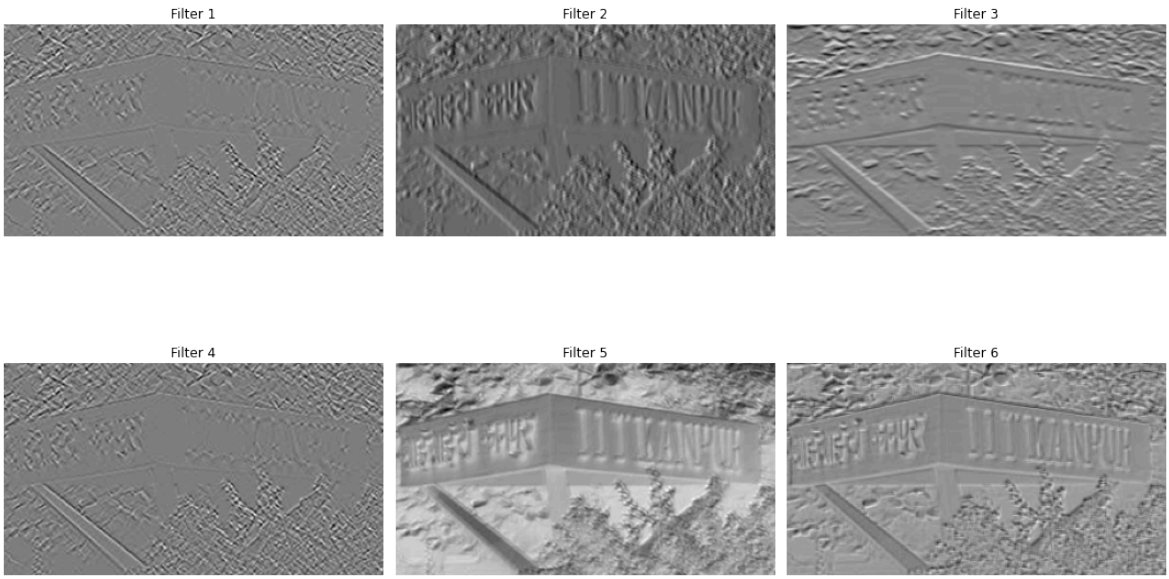
In [33]:
```
def sum_region(integral_img, top_left_corner, bottom_right_corner):
    """
    Compute the sum of the region in the integral image defined by the top-left
    """
    start_col, start_row = top_left_corner  # Top-left corner (start_col, start_
    end_col, end_row = bottom_right_corner  # Bottom-right corner (end_col, end_

    # Integral image formula to compute the sum of the rectangle
    region_sum = integral_img[end_row, end_col]
    if start_row > 0:
        region_sum -= integral_img[start_row - 1, end_col]
    if start_col > 0:
        region_sum -= integral_img[end_row, start_col - 1]
    if start_row > 0 and start_col > 0:
        region_sum += integral_img[start_row - 1, start_col - 1]

    return region_sum
```

In [35]:
```
def apply_custom_filter(integral_img, filter_kernels):
    """
    Apply a custom filter to the integral image using predefined filter kernels.
    """
    filtered_img = np.zeros((integral_img.shape[0] - 3, integral_img.shape[1] -

    # Slide the 4x4 filter over the image
    for row in range(filtered_img.shape[0]):
        for col in range(filtered_img.shape[1]):
            # Sum of the 4x4 region
            filter_sum = sum_region(integral_img, (col, row), (col + 3, row + 3)

            for kernel in filter_kernels:
                kernel = np.array(kernel)
                filter_sum -= 2 * sum_region(integral_img, (col + kernel[0, 0],
```

```
            filtered_img[row, col] = filter_sum

    return filtered_img
```

In [37]:

```
# Load a grayscale image
img_path = "hw2_iitk.png"  # Replace with your image path
img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

# Compute the integral image
integral_img = compute_integral_image(img)

orig_kernels = [[[-1,-1,1,1],[-1,-1,1,1],[1,1,-1,-1],[1,1,-1,-1]],[[-1,-1,-1,-1]
         [[1,1,-1,-1],[1,1,-1,-1],[-1,-1,1,1],[-1,-1,1,1]],[[-1,-1,-1,-1],[-1,-1

filter_kernels = [
    [[[0,0],[1,1]],[[2,2],[3,3]]],
    [[[0,0],[1,3]]],
    [[[0,0],[3,1]]],
    [[[0,2],[1,3]],[[2,0],[3,1]]],
    [[[0,0],[3,1]],[[0,2],[1,3]]],
    [[[1,1],[3,3]]]
]

filter_labels = ["Filter 1", "Filter 2", "Filter 3", "Filter 4", "Filter 5", "Fi
filtered_imgs = [apply_custom_filter(integral_img, f) for f in filter_kernels]

# Plot the filtered images in a 2x3 grid
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
for i, ax in enumerate(axes.flat):
    ax.imshow(filtered_imgs[i], cmap='gray')
    ax.set_title(filter_labels[i])
    ax.axis("off")

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



In [ ]: