Matthew McCaughan
CS 334
March 23rd, 2025
Assignment 3: Filters

Problem 1: Textbook Problems

Exercise 5.1:

The “custom filter” in Adobe Photoshop is not strictly a linear filter because it includes features such as larger kernels, as well as scale and offset modifiers that can take the input and produce a non-linear result given the input pixels. These features allow the “custom filter” to perform more than just linear filtering.

Exercise 5.2:

To find the maximum and minimum possible results from the linear filter
H =
[-1 -2 0]
[-2 0 2]
[0 2 1]

For the maximum, We want the positive coefficients to have the most positive values and negative coefficients with values of 0.
For the minimum, we want the positive coefficients to have values of 0 and the negative coefficients to have the most positive values.

Maximum:
H =
[-1 -2 0]   [0 0 0]
[-2 0 2] x [0 0 255]   = 255 *5 = 1275
[0 2 1]   [0 255 255]

Minimum:

H =
[-1 -2 0]   [255 255 0]
[-2 0 2] x [255 0   0]   = 255 *-5 = -1275
[0 2 1]   [0   0   0]

With no clamping, a resulting value has a maximum magnitude of 1275, with a maximum of 1275 and a minimum of -1275.

Exercise 5.9:
For a k-dimensional filter, A non-separable filter requires k^2 operations per pixel, but using separable x/y filters can reduce this to 2*k operations per pixel. As the size of the filter increases, the speed gain increases.

5x5:
5^2 vs 10 = 2.5x less operations

11x11:
11^2 vs 22 = 5.5x less operations

25x25:
25^2 vs 50 = 12.5x less operations

51x51:
51^2 vs 102 = 25.5x less operations

Exercise 6.1.
Calculate (manually) the gradient and the Laplacian
(using the discrete approximations in Eqn. (6.2) and Eqn. (6.32),
respectively) for the following "image":
I =
[14 10 19 16 14 12]
[18 9 11 12 10 19]
[9 14 15 26 13 6]
[21 27 17 17 19 16]
[11 18 18 19 16 14]
[16 10 13 7 22 21]

Discrete Approximations in terms of x and in terms of y:

df/dx(u) ≈ f(u+1) − f(u−1)/ (u+1) − (u−1) = **I(x+1) - I(x-1) /2**.
df/dy(u) ≈ f(u+1) − f(u−1)/ (u+1) − (u−1) = **I(y+1) − f(y−1) /2** .

For values along the edge, I just take the value itself for one of the components.

I_x =
[-2 2.5 3 -2.5 -2 -1]
[-4.5 -3.5 1.5 0.5 3.5 4.5]
[2.5 3 6 -1 -10 -3.5]
[3 -2 -5 1 -0.5 -1.5]
[3.5 2.5 0.5 -3 -2.5 -1]

[-3 -1.5 -1.5   4.5 7   -0.5]

I_y =
[2 -0.5 -4 -2 -2 3.5]
[-2.5 2 -2 5 -0.5 -3]
[1.5 9 3 2.5 4.5 -1.5]
[1 2 1.5 -3.5 1.5 4]
[-2.5 -8.5 -2 -5 1.5 2.5]
[2.5 -4 -2.5 -6 3 3.5]

For applying the Laplacian filter
H^L =
[0 1 0]
[1 -4 1]
[0 1 0],
I'll take the margins from within the image, leaving a 4x4 matrix:
H^L =
[17 11 15 18]
[4   8 -47 9]
[-38 9 12 -14]
[-6 -5 -18]

Problem 1.5
Describe the expected effects of these linear filters with the following kernels on an image (assume no normalization):

A:
[0 0 0]
[0 0 1]
[0 0 0]
This kernel would set the center pixel to the pixel to the right, this kernel would essentially shift the image one pixel to the left.

B:
[0 0 0]
[0 2 0]
[0 0 0]
This kernel would double the pixel intensity of every center pixel in the image, producing a doubly "brighter" image.

C:
[0 0 ⅓]
[0 ⅓ 0]
[⅓ 0 0]

This kernel takes the averages of one bottom-left to top-right diagonal and sets this value to the center pixel. This will smoothen the image in this diagonal direction.

Problem 2: Highlights

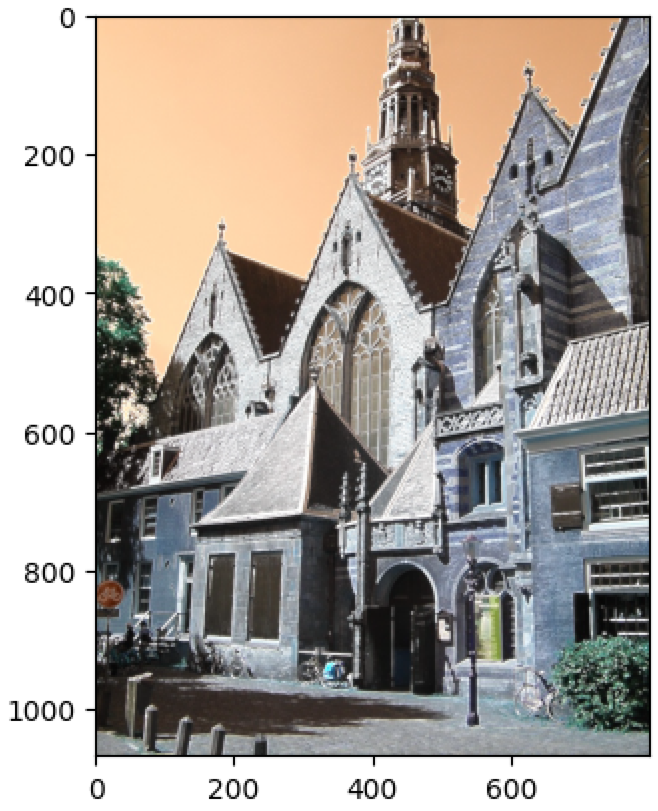
Part 1

```
import numpy as np
import matplotlib.pyplot as plt
import PIL
imG = cv2.imread("/content/ireland-03gray.tif", cv2.IMREAD_GRAYSCALE)

# Using Sobel operations for edge detection
Xgrad = cv2.Sobel(imG, cv2.CV_64F, 1, 0, ksize=3)
Ygrad = cv2.Sobel(imG, cv2.CV_64F, 0, 1, ksize=3)

# calculate gradient magnitude
Magnitude = np.sqrt(Xgrad**2 + Ygrad**2)
NormalizedMagnitude = (Magnitude / Magnitude.max()) * 255

#https://www.geeksforgeeks.org/opencv-alpha-blending-and-masking-of-images
/
alpha = 0.95
EdgeimG = cv2.addWeighted(img.astype(np.float64), 1.0,
NormalizedMagnitude, alpha, 0)
EdgeimG = np.clip(EdgeimG, 0, 255).astype(np.uint8)

#plt.imshow(imG, cmap='gray')

plt.imshow(EdgeimG, cmap='gray')
```

Part 2

```
imG1 = cv2.imread("/content/Amsterdam.JPG", cv2.COLOR_BGR2RGB)
#plt.imshow(imG1)
grayimG1 = np.mean(imG1,axis=2).astype(np.uint8)
#plt.imshow(grayimG1)

# Using Sobel operations for edge detection
Xgrad = cv2.Sobel(grayimG1, cv2.CV_64F, 1, 0, ksize=3)
Ygrad = cv2.Sobel(grayimG1, cv2.CV_64F, 0, 1, ksize=3)

# calculate gradient magnitude
Magnitude = np.sqrt(Xgrad**2 + Ygrad**2)
NormalizedMagnitude = (Magnitude / Magnitude.max()) * 255

#https://www.geeksforgeeks.org/opencv-alpha-blending-and-masking-of-images
/
ResultImG1 = np.zeros_like(imG1)
#print(NormalizedMagnitude_3D.shape)
#print(imG1.shape)
```

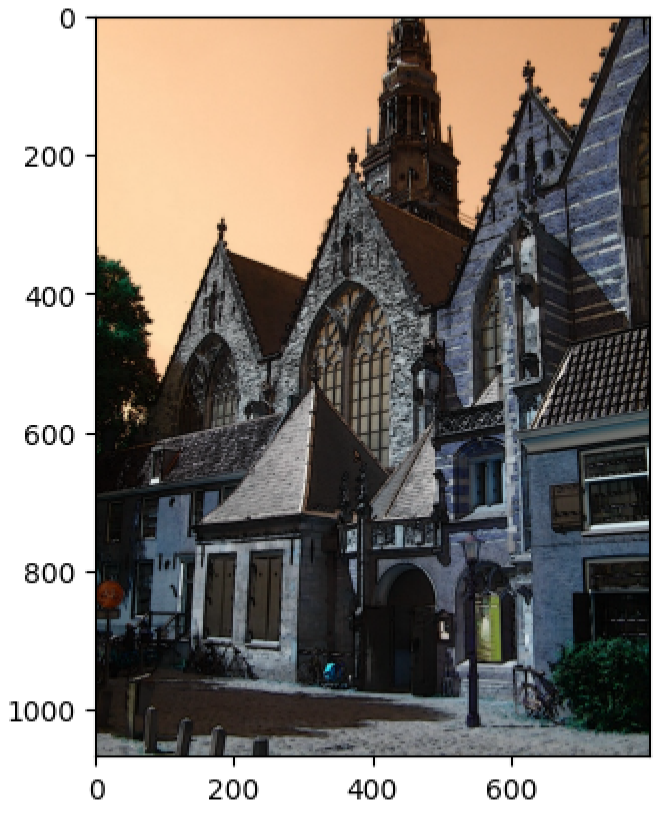
```
#print(ResultImG1.shape)
for i in range(3):
  ResultImG1[:, :, i] = np.clip(imG1[:, :, i] + NormalizedMagnitude, 0,
255)

plt.imshow(ResultImG1)
```



Part 3

```
imG1 = cv2.imread("/content/Amsterdam.JPG", cv2.COLOR_BGR2RGB)
#plt.imshow(imG1)
grayimG1 = np.mean(imG1,axis=2).astype(np.uint8)
#plt.imshow(grayimG1)

# Using Sobel operations for edge detection
Xgrad = cv2.Sobel(grayimG1, cv2.CV_64F, 1, 0, ksize=3)
Ygrad = cv2.Sobel(grayimG1, cv2.CV_64F, 0, 1, ksize=3)

# calculate gradient magnitude
Magnitude = np.sqrt(Xgrad**2 + Ygrad**2)
NormalizedMagnitude = (Magnitude / Magnitude.max()) * 255
```

```
#https://www.geeksforgeeks.org/opencv-alpha-blending-and-masking-of-images
/
ResultImG1 = np.zeros_like(imG1)
#print(NormalizedMagnitude_3D.shape)
#print(imG1.shape)
#print(ResultImG1.shape)
for i in range(3):
  # For dark highlights, subtract normalized magnitude from image.
  ResultImG1[:, :, i] = np.clip(imG1[:, :, i] - NormalizedMagnitude, 0,
255)

plt.imshow(ResultImG1)
```

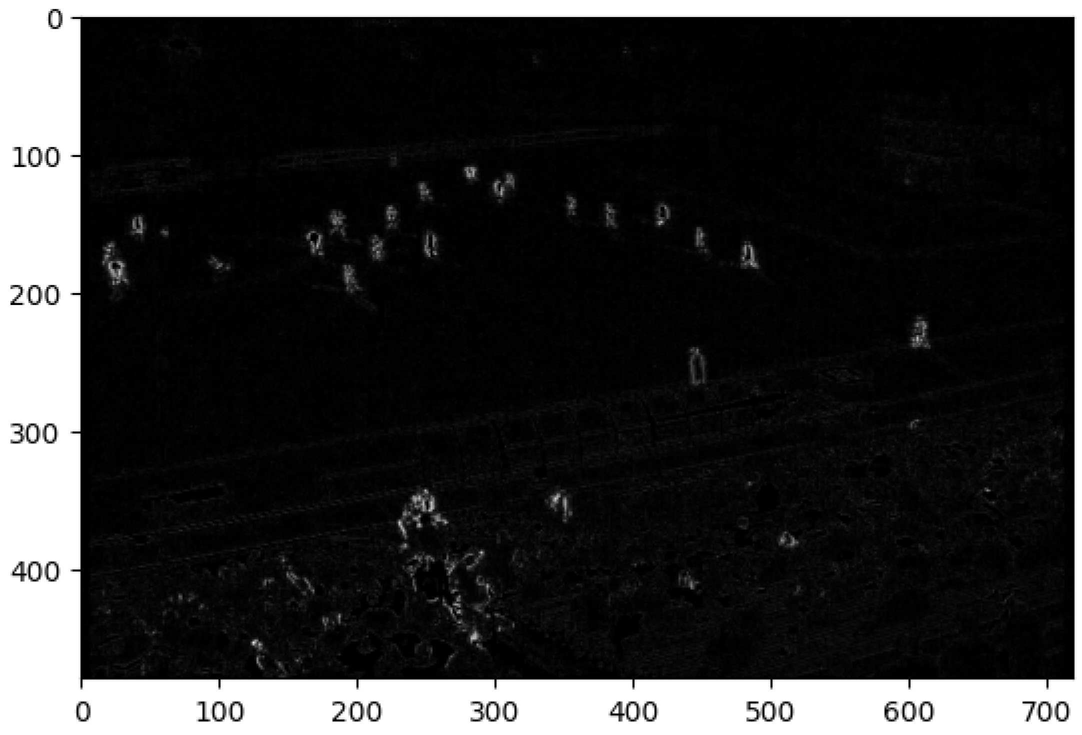


Problem 3: Point Operations

Part 1

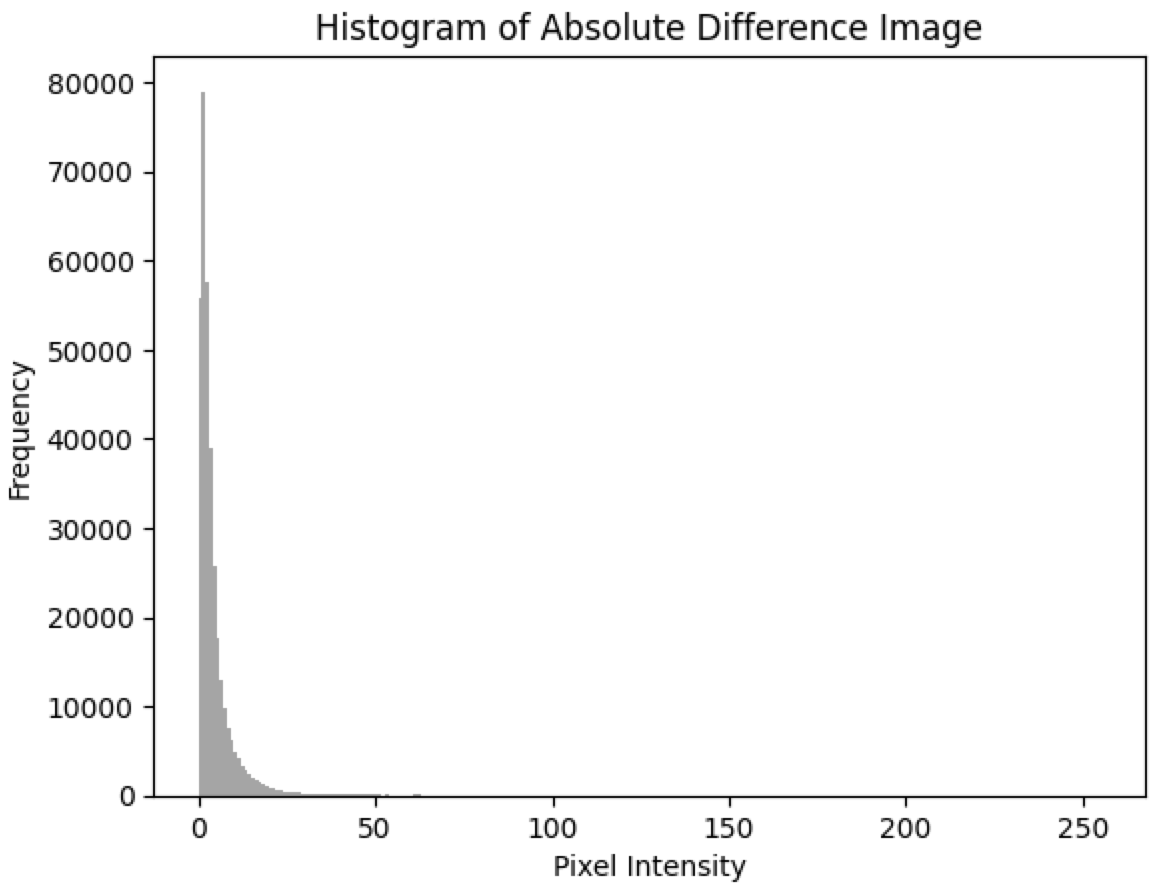
```
I1 = cv2.imread('/content/soccer1.bmp', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('/content/soccer2.bmp', cv2.IMREAD_GRAYSCALE)
#plt.imshow(I2, cmap='gray')
```

```
difference = cv2.absdiff(I1, I2)
plt.imshow(difference, cmap='gray')
```



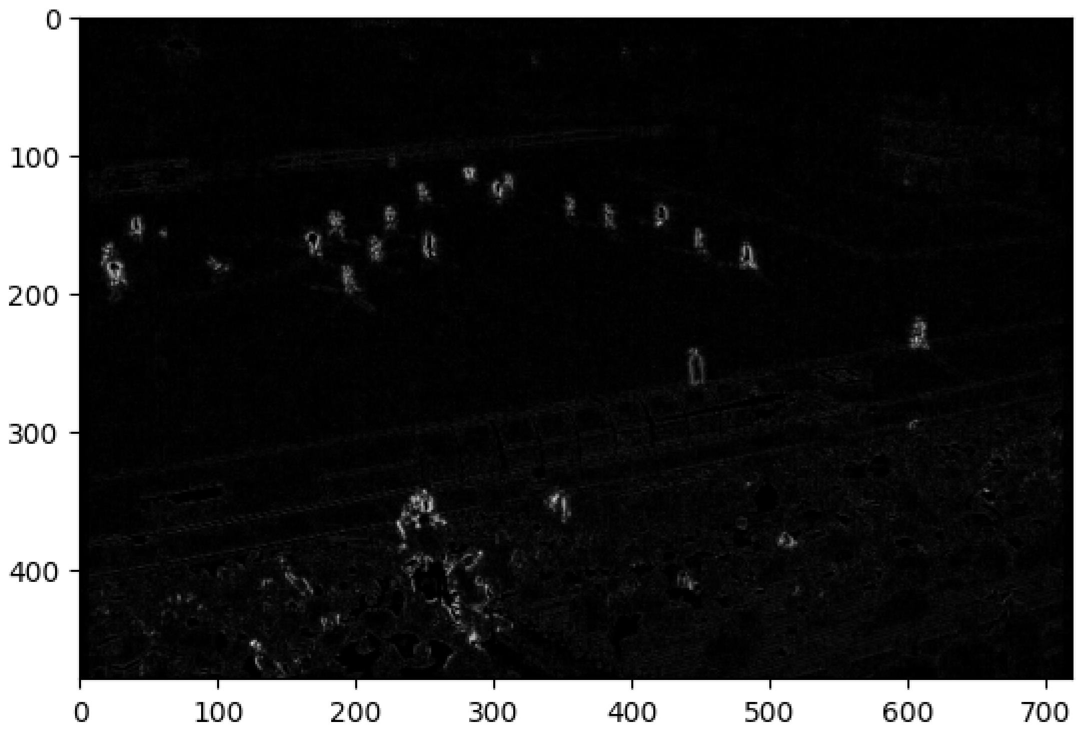
Part 2

```
plt.hist(difference.ravel(), bins=256, range=(0, 255), color='gray',
alpha=0.7)
plt.title('Histogram of Absolute Difference Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
```



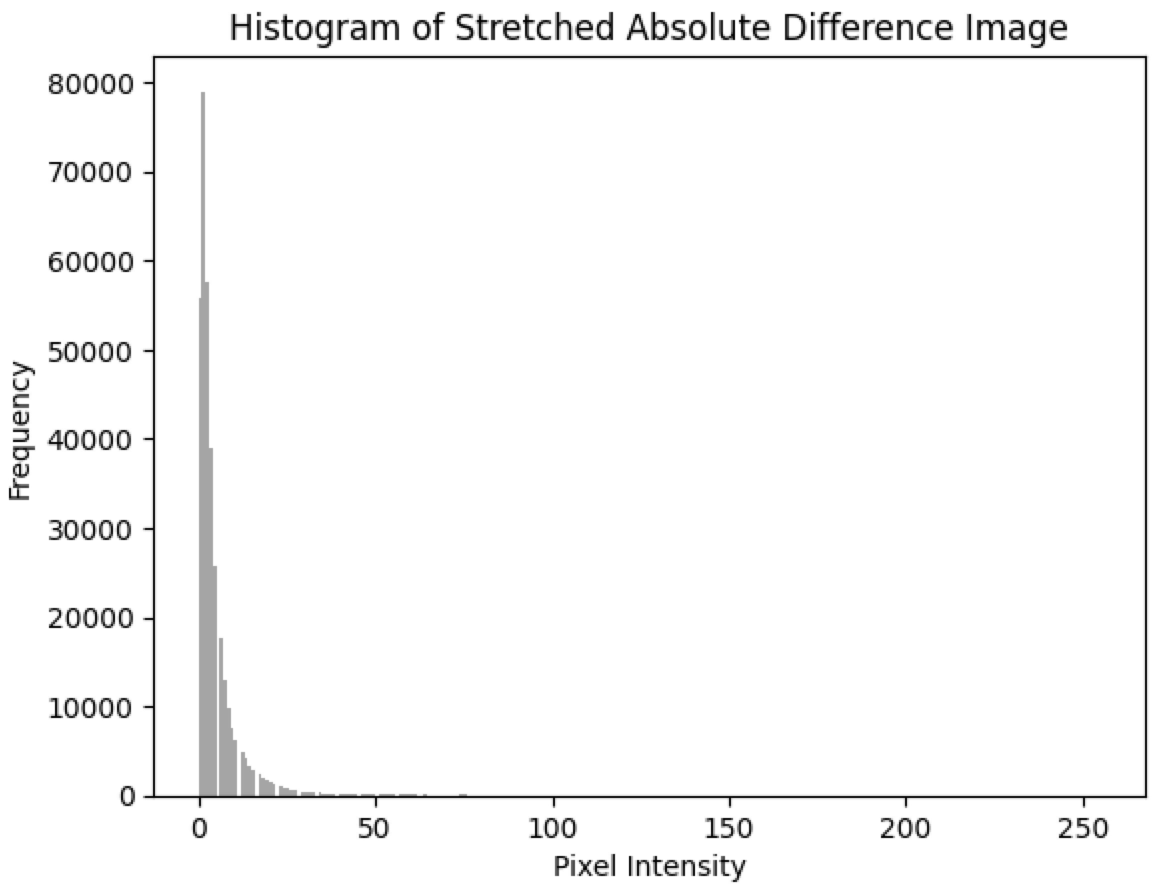
Part 3

```
MIN = np.min(difference)
MAX = np.max(difference)

StretchedDifference = np.uint8((difference - MIN) / (MAX - MIN) * 255)

plt.imshow(StretchedDifference, cmap='gray')
```



Part 3 Histogram:

```
plt.hist(StretchedDifference.ravel(), bins=256, range=(0, 255),
color='gray', alpha=0.7)
plt.title('Histogram of Stretched Absolute Difference Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
```



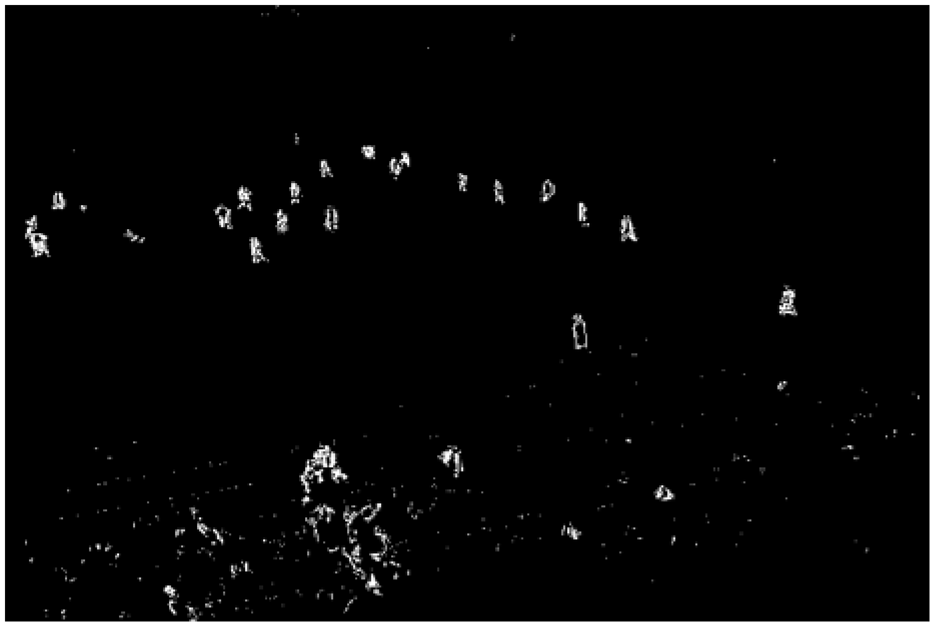
Part 4:

```
#Calculate Percentage Threshold from p% largest values:
ImgToArray = difference.ravel()
SortedImageArray = np.sort(ImgToArray)


p = 0.01
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]

# Produce mask for threshold
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

#plt.imshow(mask, cmap='gray')

plt.figure(figsize=(6, 4))
plt.imshow(mask, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 0.1
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

plt.figure(figsize=(6, 4))
plt.imshow(mask, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 1
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

plt.figure(figsize=(6, 4))
plt.imshow(mask, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 2
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

plt.figure(figsize=(6, 4))
plt.imshow(mask, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')
```

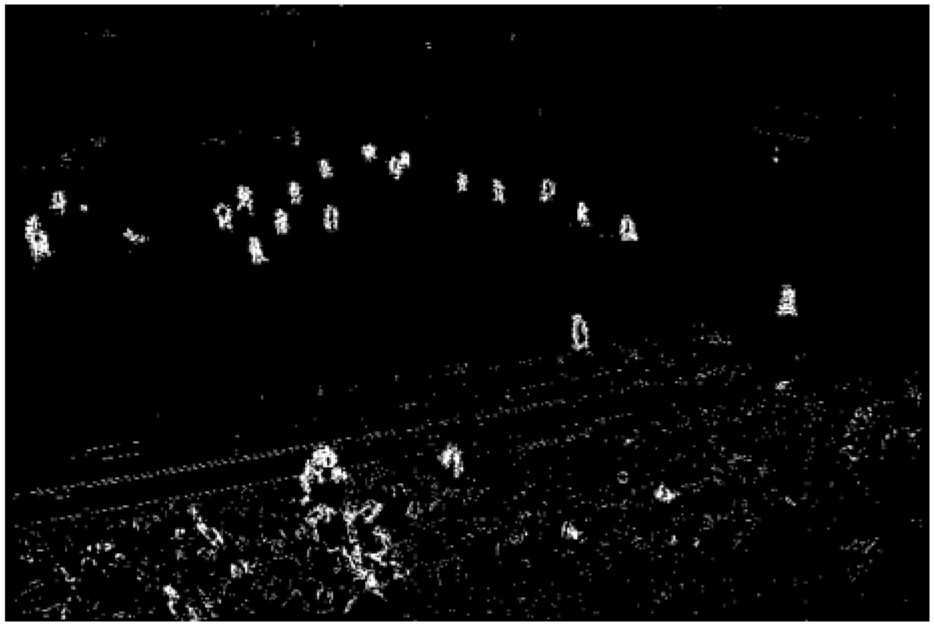
Mask for p=0.01%

Mask for p=0.1%

Mask for p=1%



Mask for p=2%



Part 5: Moving Pixels Visualization

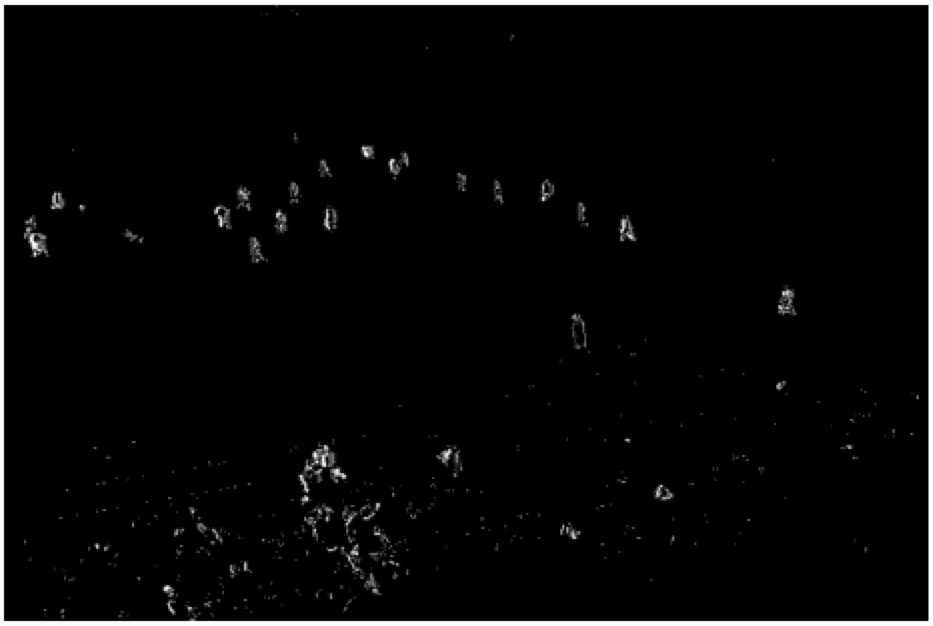
```
#Calculate Percentage Threshold from p% largest values:
ImgToArray = difference.ravel()
SortedImageArray = np.sort(ImgToArray)
```

```python
p = 0.01
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]

# Produce mask for threshold
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

Template = np.zeros_like(I2)
Template[mask==1] = I2[mask==1]

#plt.imshow(mask, cmap='gray')

plt.figure(figsize=(6, 4))
plt.imshow(Template, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 0.1
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

Template = np.zeros_like(I2)
Template[mask==1] = I2[mask==1]
plt.figure(figsize=(6, 4))
plt.imshow(Template, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 1
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

Template = np.zeros_like(I2)
Template[mask==1] = I2[mask==1]
plt.figure(figsize=(6, 4))
plt.imshow(Template, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')

p = 2
ThresholdValue = int(len(SortedImageArray) * (1 - p / 100))
threshold = SortedImageArray[ThresholdValue]
mask = np.where(difference >= threshold, 1, 0).astype(np.uint8)

Template = np.zeros_like(I2)
Template[mask==1] = I2[mask==1]
plt.figure(figsize=(6, 4))
plt.imshow(Template, cmap='gray')
plt.title(f'Mask for p={p}%')
plt.axis('off')
```

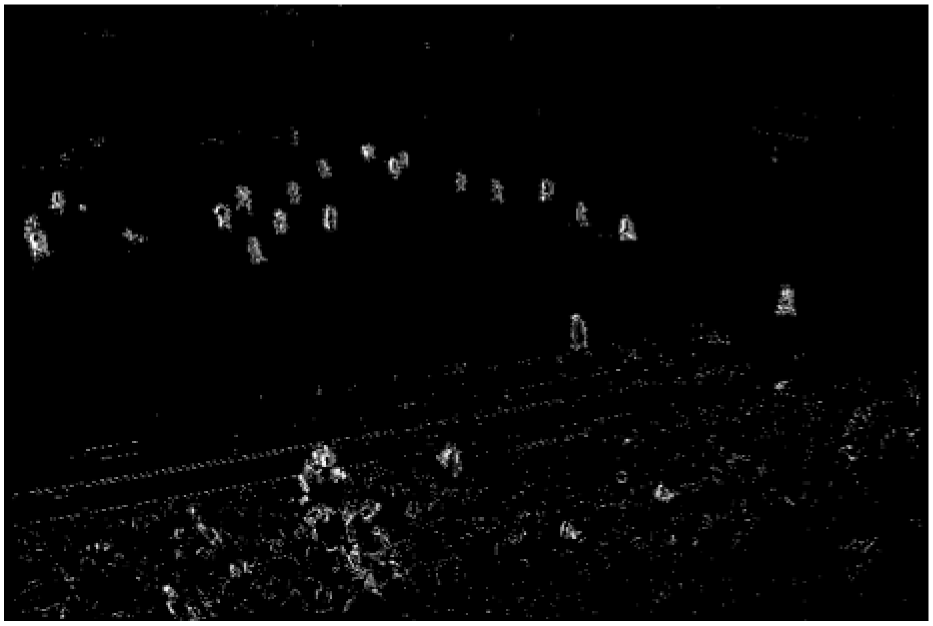
Mask for p=0.01%

Mask for p=0.1%

Mask for p=1%



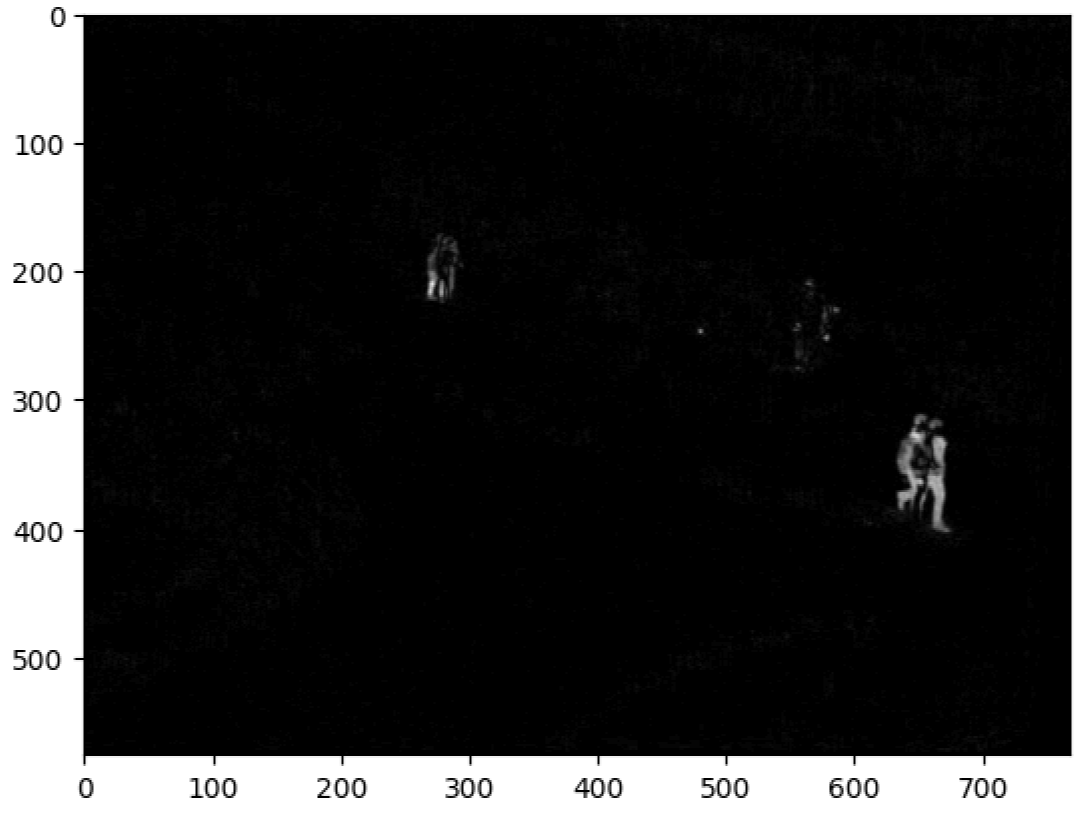
Mask for p=2%



Running code on parking lot scene:

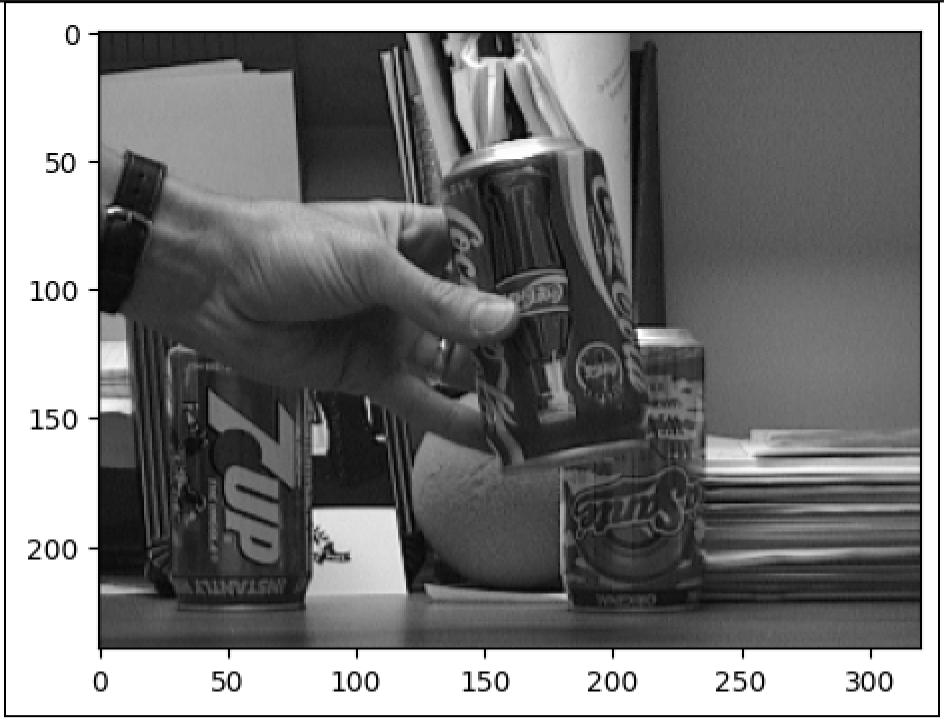
```
I1 = cv2.imread('/content/1015.jpg', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('/content/1020.jpg', cv2.IMREAD_GRAYSCALE)
#plt.imshow(I2, cmap='gray')
```

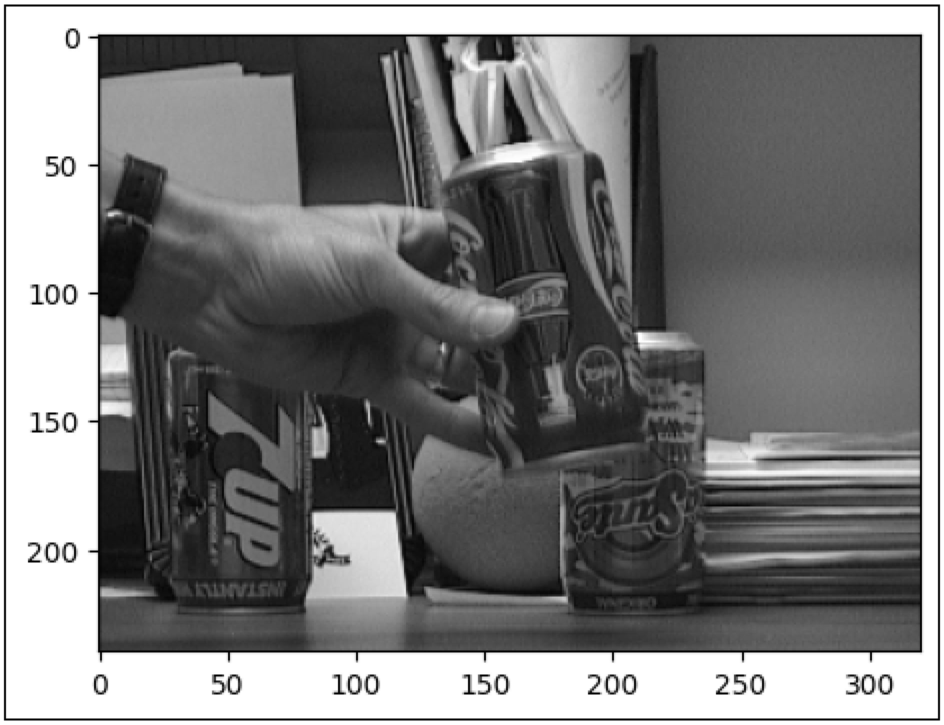
```
difference = cv2.absdiff(I1, I2)
plt.imshow(difference, cmap='gray')
MIN = np.min(difference)
MAX = np.max(difference)

StretchedDifference = np.uint8((difference - MIN) / (MAX - MIN) * 255)

plt.imshow(StretchedDifference, cmap='gray')
```



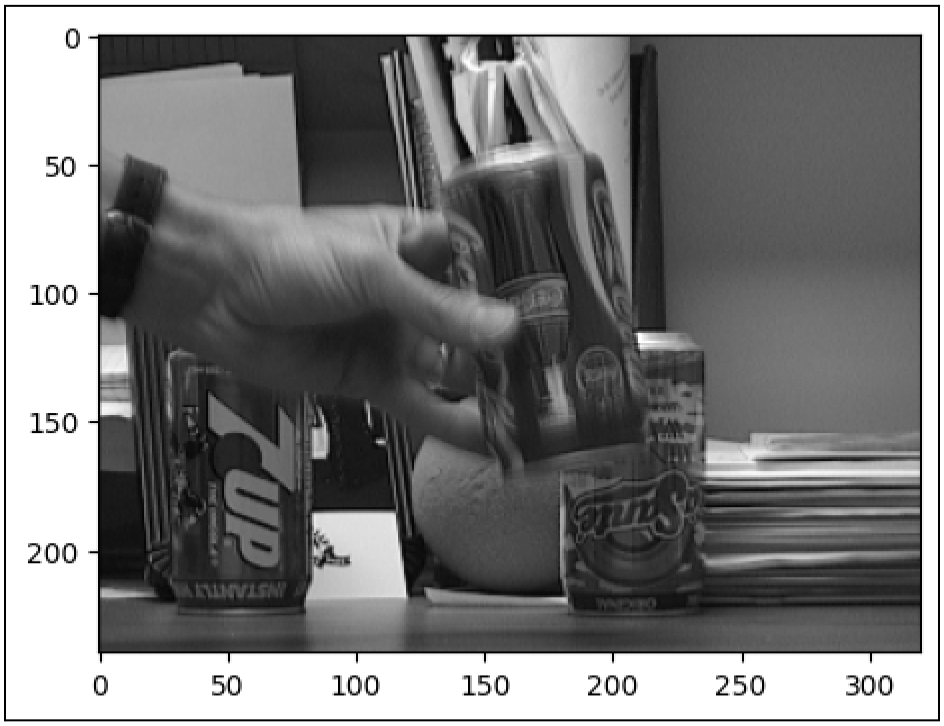
Problem 4: Recursive Blending

```
mages =['/content/100.tif','/content/101.tif','/content/102.tif',
         '/content/103.tif','/content/104.tif', '/content/105.tif',
         '/content/106.tif', '/content/107.tif', '/content/108.tif',
         '/content/109.tif','/content/110.tif']

loaded_images = [cv2.imread(image_path) for image_path in images]

def recursive_blending(images, alpha):
    B = images[0].astype(np.float32)
    # recursive blending
    for i in range(1, len(images)):
        B = alpha * B + (1 - alpha) * images[i].astype(np.float32)
    # Convert B back to uint8 for displaying
    B = np.clip(B, 0, 255).astype(np.uint8)
    return B
plt.figure(figsize=(6, 4))
plt.imshow(recursive_blending(loaded_images, 0.1), cmap='gray')
plt.figure(figsize=(6, 4))
plt.imshow(recursive_blending(loaded_images, 0.2), cmap='gray')
plt.figure(figsize=(6, 4))
plt.imshow(recursive_blending(loaded_images, 0.5), cmap='gray')
plt.figure(figsize=(6, 4))
plt.imshow(recursive_blending(loaded_images, 0.8), cmap='gray')
```

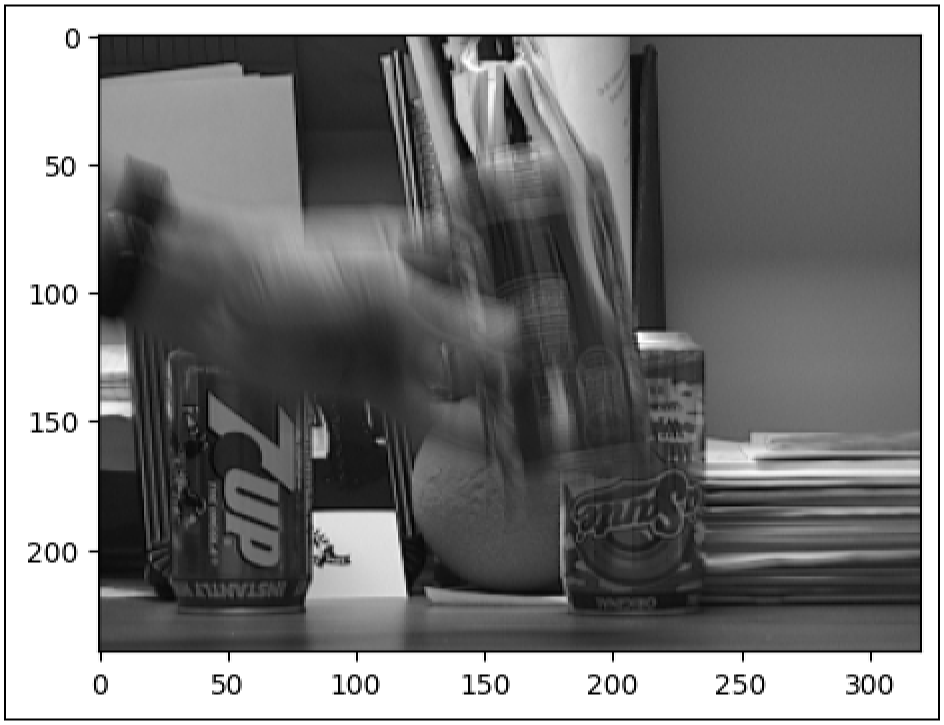




0
50
100
150
200
0
50
100
150
200
250
300
7UP
INSTANTLY

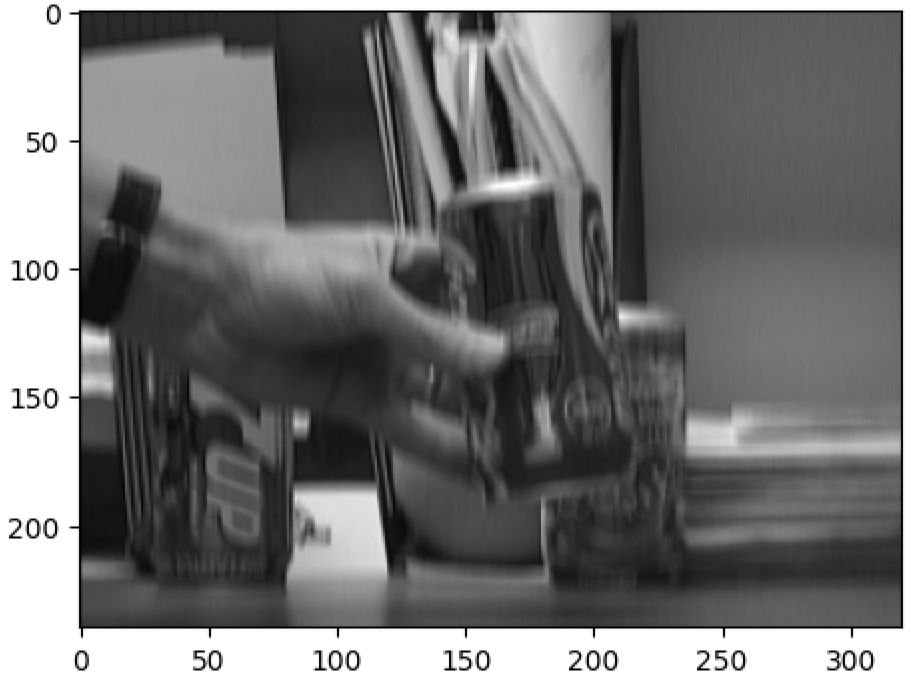


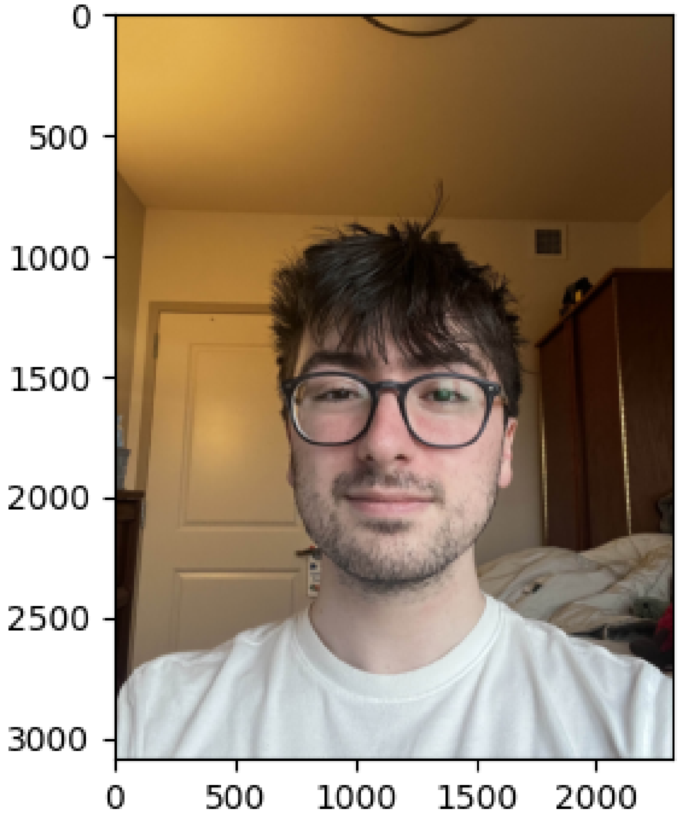
0
50
100
150
200
0
50
100
150
200
250
300
7UP
INSTANTLY



Problem 5: Linear Blur

```
Filter = [1/7,1/7,1/7,1/7,1/7,1/7,1/7]
Kernal = np.array(Filter)
image = cv2.imread('/content/100.tif')
blurred_image = cv2.filter2D(image,-1, Kernal)
plt.figure(figsize=(6, 4))
plt.imshow(blurred_image, cmap='gray')

image = cv2.imread('/content/self.jpg')
#had to change the color channels
colorchange = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
blurred_image = cv2.filter2D(rgb,-1, Kernal)
plt.figure(figsize=(6, 4))
plt.imshow(blurred_image, cmap='gray')
```


0
50
100
150
200
0
50
100
150
200
250
300


0
500
1000
1500
2000
2500
3000
0
500
1000
1500
2000