Kunnskap for en bedre verden

Department of Engineering Cybernetics

TTK4255 - Robotic Vision

Homework 1: Image processing

Zahra Parvinashtiani

31.01.2024

Table of Contents

List of Figures

1 Part 1: Theory questions

Robotic Vision HW1

Thursday, 25 January 2024 14:03

Part 1_

1.
$$\begin{bmatrix} -0.5 & -1.0 & -0.5 \\ -1.0 & +7.0 & -1.0 \\ -0.5 & -1.0 & -0.5 \end{bmatrix}$$

$|7| >$ absolute of every neg value in the matrix

$\Rightarrow$ sharpening kernel

effects:

1. positive center: boosts pixel value at the current location.
2. negative surrounding values: reduce the contributions of the neighboring pixels.

this makes the edeges within the image more pronounced.
So we have a sharper image $\rightarrow$ focus & clarity $\uparrow$

sum of neg values $= -4.0$ center $= 7.$ $\rightarrow 7 - 4 = 3 \rightarrow$ scalling factor?? to normalize

without normalization $\rightarrow$ saturation.

2.

a. $g(h) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{h^2}{2\sigma^2}\right) < \epsilon \rightarrow \exp\left(-\frac{h^2}{2\sigma^2}\right) < \epsilon\sqrt{2\pi}\,\sigma$

$\xrightarrow{Ln} -\frac{h^2}{2\sigma^2} < \ln\left(\epsilon\sqrt{2\pi}\,\sigma\right) \rightarrow h^2 > -2\sigma^2 \ln\left(\epsilon\sqrt{2\pi}\,\sigma\right)$

$\xrightarrow{root}$ h is half_width $\rightarrow$ positive $\rightarrow h > \sqrt{-2\sigma^2 \ln\left(\epsilon\sqrt{2\pi}\,\sigma\right)}$

b. $h > \sqrt{-2(3)^2 \ln\left(\frac{1}{256}\sqrt{2\pi}(3)\right)} \rightarrow h > 7.97 \rightarrow$ integer $\rightarrow h = 8$

Kernel size $= 2h+1 \rightarrow$ for centeral pixels. $\rightarrow$ full kernel size $= 2\times 8 + 1 = \boxed{17}$

New Section 1 Page 1

Figure 1: Answers for theory part.

2 Part 2: Basics

All the code for part 2 is in task2.py.

2.1 Part 2.1

```
## Task 2.1:

image = Image.open( "../data/grass.jpg" )
image.load()
img = np.asarray( image, dtype="int32" )

print("The height is: " + str(img.shape[0]) + " and the width is: " + str(img.shape[1]))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\X1 Yoga G8\Desktop\TTK4255 Robotic Vision\homework1\python>  c:; cd 'c:\Users\X1 Yoga G
K4255 Robotic Vision\homework1\python'; & 'C:\Users\X1 Yoga G8\AppData\Local\Programs\Python\Pythor
xe' 'c:\Users\X1 Yoga G8\.vscode\extensions\ms-python.python-2023.22.1\pythonFiles\lib\python\debug
./..\debugpy\launcher' '3148' '--' 'c:\Users\X1 Yoga G8\Desktop\TTK4255 Robotic Vision\homework1\py
y'
The height is: 720 and the width is: 1280
PS C:\Users\X1 Yoga G8\Desktop\TTK4255 Robotic Vision\homework1\python>
```

Figure 2: Code and the result for the task 2.1.

2.2 Part 2.2

Based on the plots and also RGB(red, green, blue) se second channel (channel[1]) is for green.

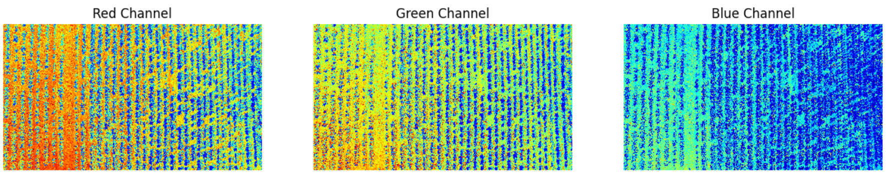

Figure 3: The result for the task 2.2.

2.3 Part 2.3

With adjusting threshold we can get the binary image below for threshold = 132.

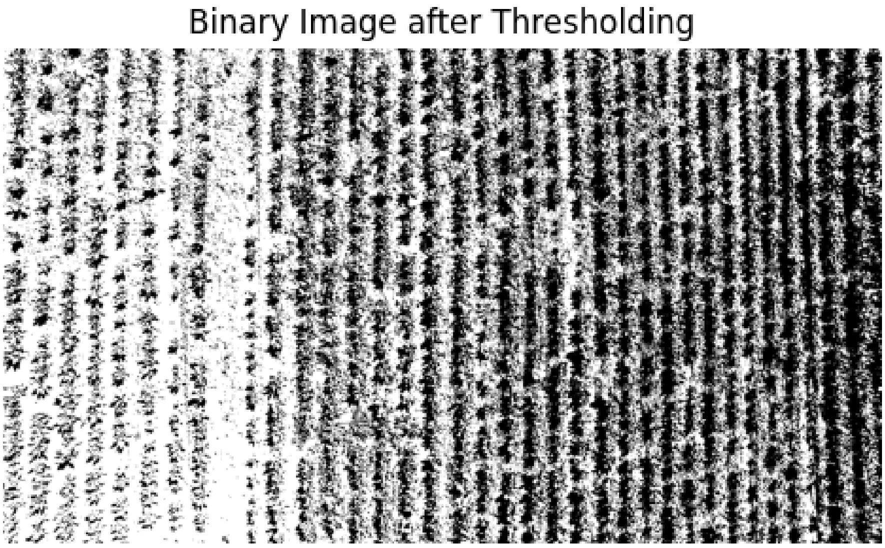

Figure 4: The result for the task 2.3.

2.4 Part 2.4

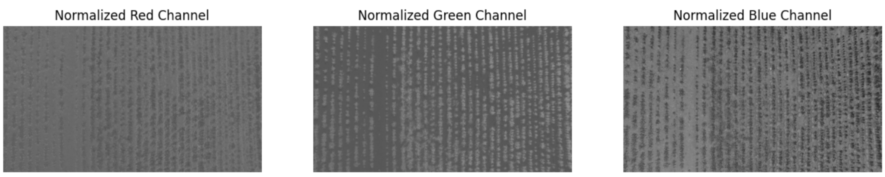

Figure 5: The result for the task 2.4.

2.5 Part 2.5

The image below is for threshold = 0.4.

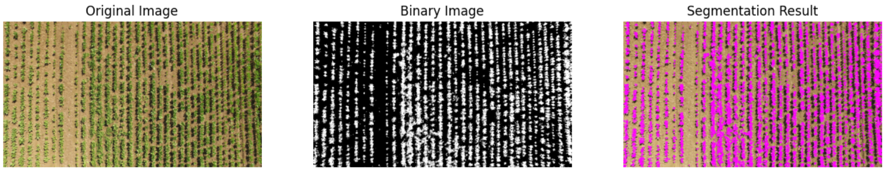

Figure 6: The result for the task 2.5.

3 Part 3: Edge detection

3.1 Part 3.1 - 3.4

The code for part 3.1 - part 3.4 is in common.py.

3.2 Part 3.5

The code for this part is in task3.py, sigma = 3.5 and threshold = 0.01.

- Sigma: This parameter defines the amount of blurring. A higher sigma value means more blurring, which can help reduce noise but might also cause true edges to be less sharp. For detailed images with fine edges, a smaller sigma is generally preferred.
- Threshold: This value determines the sensitivity of the edge detection. A lower threshold may detect more edges, including noise and finer details, while a higher threshold may only detect the most prominent edges.

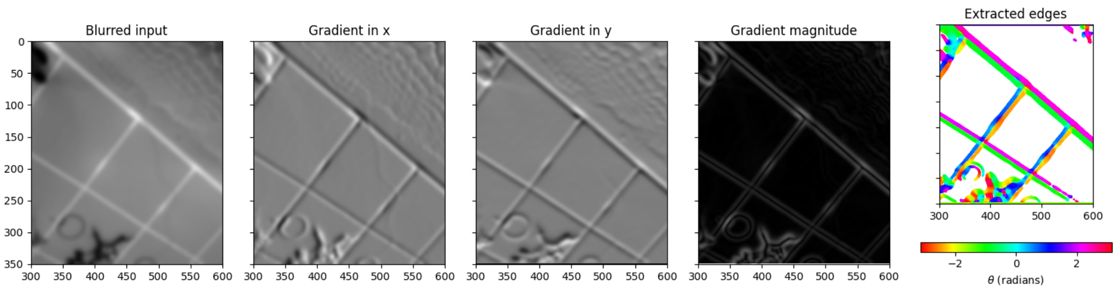

Figure 7: The result for the task 3.

Appendix

A Task 2 code

```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt


## Task 2.1:

image = Image.open( "../data/grass.jpg" )
image.load()
img = np.asarray( image, dtype="int32" )

print("The height is: " + str(img.shape[0]) + " and the width is: " +
↪   str(img.shape[1]))

## Task 2.2:
red_channel = img[:, :, 0]
green_channel = img[:, :, 1]
blue_channel = img[:, :, 2]

# Plot the channels
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(red_channel, cmap='jet', interpolation='none')
axes[0].set_title('Red Channel')
axes[0].axis('off')

axes[1].imshow(green_channel, cmap='jet', interpolation='none')
axes[1].set_title('Green Channel')
axes[1].axis('off')

axes[2].imshow(blue_channel, cmap='jet', interpolation='none')
axes[2].set_title('Blue Channel')
axes[2].axis('off')

plt.savefig('task2.2.png')
#plt.show()

## Task 2.3:

threshold = 132  #adjust this based on the image

# Apply thresholding
binary_image = green_channel > threshold

#Display the result
plt.figure(figsize=(6, 6))
plt.imshow(binary_image, cmap='gray')
plt.title('Binary Image after Thresholding')
plt.axis('off')
plt.savefig('task2.3.png')
#plt.show()
```

```python
## Task 2.4:
epsilon = 1e-8
sum_rgb = np.sum(img, axis=2) + epsilon

# Calculate normalized RGB values
r_norm = img[:, :, 0] / sum_rgb
g_norm = img[:, :, 1] / sum_rgb
b_norm = img[:, :, 2] / sum_rgb

# Plot the normalized channels
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(r_norm, cmap='gray')
axes[0].set_title('Normalized Red Channel')
axes[0].axis('off')

axes[1].imshow(g_norm, cmap='gray')
axes[1].set_title('Normalized Green Channel')
axes[1].axis('off')

axes[2].imshow(b_norm, cmap='gray')
axes[2].set_title('Normalized Blue Channel')
axes[2].axis('off')

plt.savefig('task2.4.png')
# plt.show()

## Task 2.5:

threshold = 0.4  # Threshold

binary_image_norm = g_norm > threshold

# Replace above-threshold pixels with magenta in the original image
magenta = [255, 0, 255]  # RGB value for magenta
segmented_image = np.where(binary_image_norm[..., None], magenta,
↪  img).astype(np.uint8)

# Plotting the results
plt.figure(figsize=(15, 5))

# Original image
plt.subplot(1, 3, 1)
plt.imshow(img.astype(np.uint8))
plt.title('Original Image')
plt.axis('off')

# Binary image
plt.subplot(1, 3, 2)
plt.imshow(binary_image_norm, cmap='gray')
plt.title('Binary Image')
plt.axis('off')

# Segmented image with magenta
plt.subplot(1, 3, 3)
plt.imshow(segmented_image)
plt.title('Segmentation Result')
plt.axis('off')
plt.savefig('task2.5.png')
```

```
# plt.show()
```

⋮

B Common code

```
import numpy as np

from scipy.ndimage import gaussian_filter

def rgb_to_gray(I):
    """
    Converts a HxWx3 RGB image to a HxW grayscale image as
    described in the text.
    """
    return np.mean(I, axis=2) # Placeholder

def central_difference(I):
    """
    Computes the gradient in the x and y direction using
    a central difference filter, and returns the resulting
    gradient images (Ix, Iy) and the gradient magnitude Im.
    """
    kernel = np.array([0.5, 0, -0.5])

    Ix = np.zeros_like(I)
    Iy = np.zeros_like(I)

    # Convolve the kernel with each row for horizontal gradient
    for i in range(I.shape[0]):
        Ix[i, :] = np.convolve(I[i, :], kernel, mode='same')

    # Convolve the kernel with each column for vertical gradient
    for i in range(I.shape[1]):
        Iy[:, i] = np.convolve(I[:, i], kernel, mode='same')

    # Compute the gradient magnitude
    Im = np.sqrt(Ix**2 + Iy**2)

    return Ix, Iy, Im

def gaussian(I, sigma):
    """
    Applies a 2-D Gaussian blur with standard deviation sigma to
    a grayscale image I.
    """

    # Hint: The size of the kernel should depend on sigma. A common
    # choice is to make the half-width be 3 standard deviations. The
    # total kernel width is then 2*np.ceil(3*sigma) + 1.

    result = gaussian_filter(I, sigma=sigma)
    return result

def extract_edges(Ix, Iy, Im, threshold):
    """
    Returns the x, y coordinates of pixels whose gradient
```

```
    magnitude is greater than the threshold. Also, returns
    the angle of the image gradient at each extracted edge.
    """
    # Get the indices of the pixels that are above the threshold
    y_coords, x_coords = np.nonzero(Im > threshold)

    # Compute the gradient orientation for each edge pixel
    angles = np.arctan2(Iy[y_coords, x_coords], Ix[y_coords, x_coords])

    return x_coords, y_coords, angles
```

⋮

C Task 3 code

```
import numpy as np
import matplotlib.pyplot as plt
from common import *

threshold = 0.01 # todo: choose an appropriate value
sigma     = 3.5 # todo: choose an appropriate value
filename  = '../data/grid.jpg'

I_rgb      = plt.imread(filename)
I_rgb      = I_rgb/255.0
I_gray     = rgb_to_gray(I_rgb)
I_blur     = gaussian(I_gray, sigma)
Ix, Iy, Im = central_difference(I_blur)
x,y,theta  = extract_edges(Ix, Iy, Im, threshold)
print(x,y,theta)
fig, axes = plt.subplots(1,5,figsize=[15,4], sharey='row')
plt.set_cmap('gray')
axes[0].imshow(I_blur)
axes[1].imshow(Ix, vmin=-0.05, vmax=0.05)
axes[2].imshow(Iy, vmin=-0.05, vmax=0.05)
axes[3].imshow(Im, vmin=+0.00, vmax=0.10, interpolation='bilinear')
edges = axes[4].scatter(x, y, s=1, c=theta, cmap='hsv')
fig.colorbar(edges, ax=axes[4], orientation='horizontal', label='$\\theta$
↪  (radians)')
for a in axes:
    a.set_xlim([300, 600])
    a.set_ylim([I_rgb.shape[0], 0])
    a.set_aspect('equal')
axes[0].set_title('Blurred input')
axes[1].set_title('Gradient in x')
axes[2].set_title('Gradient in y')
axes[3].set_title('Gradient magnitude')
axes[4].set_title('Extracted edges')
plt.tight_layout()
plt.savefig('out_edges.png') # Uncomment to save figure to working directory
plt.show()
```