Marks

out of
104
Without Answers

Final Exam

National Taiwan Normal University

Computer Vision

Examiners: 包傑奇
Date: 20th December 2023
Time: 9:30 – 12:00
Exam Room: Discord https://discord.gg/CuanKnp5GM

Notes:

- Attempt all questions.
- This is an *open* book and *open* Internet examination. Use of books, notes, laptops, and computers **with Internet connectivity** is *permitted*.
- This exam must be **your own** work. Communication with others via messenger apps, email, phone is strictly **forbidden**.
- Show your work to receive full marks. You must show your reasoning, intermediate steps/calculations to reach the answer.
- Some of the questions may not be solvable, that is it may be impossible to calculate the requested information. In this case, say so in your answer and explain why.
- Submit your exam by printing it to a pdf file (File -> Print or Ctrl-P in most browsers) and then sending the pdf file to the instructor in a private chat message.

Name: 鄭翔元
Student ID: 612K0020C

Submit

Python Programming

This section covers topics in python programming for robot vision.

[1] Opcodes and Program

Given are two list of numbers `opcodes` and `program`.

Implement a function `count_opcodes` that returns the number of times a program executed an opcode listed in `opcodes`.

```
count_opcodes([15,67,65,68,12,30,39,63,88,34,16,69,15,44,88],[87,48,38,97]) => 0
count_opcodes([11,56,21,30,67,53,40,77,77,5,8,26,94],[79,57,40,41,90]) => 1
count_opcodes([43,98,72,10,98,57,23,74,80,94],[98,21,32,20,5,83]) => 2
```

10 marks

```
def count_opcodes( program, opcodes ):
```

```
def count_opcodes(program: list, opcodes: list) -> int:
  """
  Count the number of opcodes in the program
  :param program: The program
  :param opcodes: The opcodes
  :return: The number of opcodes
  """
  count = 0
  for elem in opcodes:
    for code in opcodes:
      if elem == code:
        count += 1
        break
  return count
```

[2] Worst-case Runtime Performance (`program`)

The program `count_opcodes` is called with parameter `program` (length 3000) and parameter `opcode` (length 500). The executionn takes 1.5890 seconds.

6 marks

What is the expected runtime of the `count_opcodes` program if the length of parameter `program` is changed to 6000 numbers.

Expected Worst-case Runtime: 3.1780

There are 2 for loop in the code...
so it would run program length x opcode length as it cost time.
let len(n) as the time that running for n numbers,
then len(3000) x len(500) = 1.5890 sec
-> len(6000) x len(500) = 3.1780 sec

[3] Worst-case Runtime Performance (`opcode`)

The program `count_opcodes` is called with parameter `program` (length 3000) and parameter `opcode` (length 500). The executionn takes 1.5890 seconds.

7 marks

What is the expected runtime of the `count_opcodes` program if the length of parameter `program` is changed to 6000 numbers **and** the length of the parameter `opcodes` is changed to 2500 numbers.

Expected Worst-case Runtime: 15.8900

Same equation as above:
len(3000) x len(500) = 1.5890 sec
len(6000) x len(2500) = 15.890 sec

Edge Detection

This topic includes an investigation of various edge detection alogirthms.

[4] Steganography

The Boeing 747 from Beijing had just landed at the Frankfurt airport. You watched the diaspora of passengers leave the jetway and spill out over the arrival area: parents trying to coral their children, moving just for fun after being stuck in a metal tube for 13 hours, businessman in their standard issue blue suits moving on autopilot, and well-dressed under 30's moving to the duty free shops.

You paid no attention to the usual travellers and focused on finding Agent X, who was brining top secret information about the plans of Evil Doer, the criminal mastermind.

You notice Agent X and push forward against the stream of passengers to welcome your old friend. Suddenly, you hear the distinct "Plop" of a gun with silencer. Next Agent X collapses into your arms, a small red spot appearing on her chest and starting to spread.

Agent X grips your arms, and with her dying breat she says: "The code is in the image." and quickly presses a USB stick into your hands.

"Hang in there! Don't give up!. Who did this? Where is the key?"

Agent X responds: "Don't worry - we will always have Paris. Secrets, sprinkled like snow unseen, the edges will guide the way," she whispers with her dying breath.

Fighting back the tears, you leave your friend behind and rush to the office with the picture.

"Evil Doer is going to pay for this! I must find the secret." you repeat to yourself like a new-age mantra.

Agent X was an expert in steganography - the encoding of secret information in plain sight such as in an image. After a quick search you find an image on the usb stick that cost Agent X her life. You can recover part of the program that was used to encrypt the image and find the following code.

```
text_img = cv.putText( text_img, secret, (x,y), 1, 1, (255,255,0), 5 )
new_img = alpha * text_img + (1 - alpha) * img
```

Unfortunately, the part of the program that assigned values to `secret`, `x`, and `y` has been destroyed.

Save inoocent lives and implement a program to extract the secret message from the image.

Click on the image below to download it as png file.

[5] Merge Images

First you want to understand how the code modifies an image.

Given the grayscale image and secret message below, show the resulting image after merge.

Original Image

33	152	19	153
162	62	165	78
46	127	153	13

Mask

64	9	133	45
135	30	150	9
11	131	107	160

5 marks

Assuming that alpha = 1.60, calculate the resulting output image.

	0	1	2	3
0	82.6	-76.8	201.4	-19.8
1	118.8	10.8	141	-32.4
2	-10	133.4	79.4	248.2

Enter your code here:

```
import numpy as np

ori = np.array([[33, 152, 19, 153], [162, 62, 165, 78], [46, 127, 153, 13]])
mask = np.array([[64, 9, 133, 45], [135, 30, 150, 9], [11, 131, 107, 160]])

# let new = alpha * mask + (1 - alpha) * ori
alpha = 1.6

new = alpha * mask + (1 - alpha) * ori

print(new)
```

[6] Extract the Secret Key

You realize that a text message may have been added to the image by Agent X.

Based on your analysis implement a program to extract the secret key.

15 marks

Secret Code (Numbers only) 3220249

Enter your code below:

```
import cv2 as cv
import numpy as np

# Find the secret text by edge detection
ori_img = cv.imread("taiwan2.png")

# Find the edge in 2 ways
ori_img = cv.cvtColor(ori_img, cv.COLOR_BGR2GRAY)
edge1 = cv.Canny(ori_img, 0, 0, apertureSize=3)
edge2 = cv.Canny(ori_img, 0, 100, apertureSize=3)
```

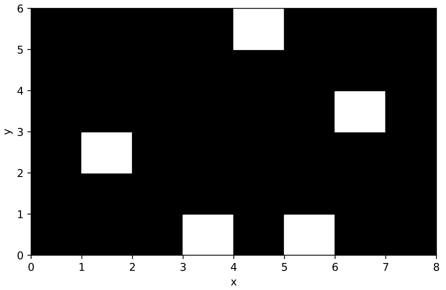
```
# Let edge1 - edge2
edge = edge1 - edge2

# Show the edge
cv.imshow("edge1", edge1)
cv.waitKey(0)
cv.imshow("edge", edge)
cv.waitKey(0)
cv.destroyAllWindows()
```

Hough Transform and Line Detection

This topic covers algorithm, implementation, and issues in Hough transforms and line detection.

[7] Hough Transform

Given is the following edge map of an image as a gray-scale bitmap. The pixel values range from 0 (black) to 255 (white).



Show the resulting Hough space for this input image in the figure below. Use the line-intercept model for lines y=a*x+b with a between -2 to 2 in steps of 1 and b between 0 to 6 in steps of 1. Ignore points that lie outside of the given Hough space.

11 marks

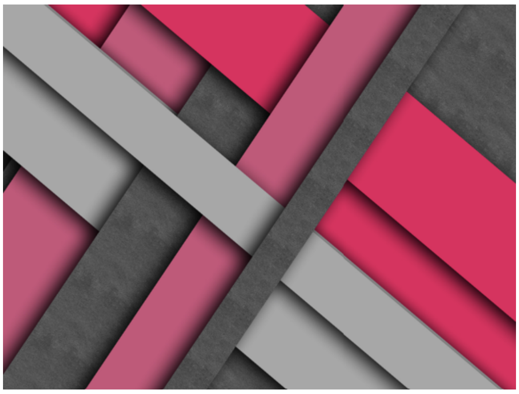
Hough Space:

	-2	-1	0	1	2
0	0	0	0	1	0
1	0	0	1	0	0
2	0	0	0	0	1
3	0	0	1	0	0
4	1	2	1	2	1
5	0	0	1	0	0
6	0	0	1	0	0

```
import numpy as np
import cv2 as cv

imgArray = np.array(
  [
    [0, 0, 0, 0, 255, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 255, 0],
    [0, 255, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 255, 0, 255, 0, 0],
  ]
)

# Let y=ax+b, for range a=[-2,2], b=[0,6]
a = np.linspace(-2, 2, 5)
b = np.linspace(0, 6, 7)

hough = np.zeros((len(b), len(a)))

for adx, a_val in enumerate(a):
  for bdx, b_val in enumerate(b):
    for x, y in zip(*np.where(imgArray == 255)):
      if y == a_val * x + b_val:
        hough[bdx, adx] += 1

print(hough)
```

[8] Hough Line Algorithm

Given is the image below. You can click on the image to download it..



In this question, you will investigate the effect of blurring on the number of lines found in the image using the Hough transform.

Use the OpenCV library to implement the following program.

1. Read the image into your program,
2. apply the Canny edge detector with parameters min=150 and max=500,
3. apply the HoughLines algorithm with a resolution of 5.0 deg for theta. and a resolution of 3.0 for rho. Set the threshold parameter to 100.

15 marks

Given the input image and algorithm as described above, the OpenCV HoughLines algorithm returns 97 lines.

B *I* U ~~S~~

```
import cv2 as cv
import numpy as np

# 1. Read the image into your program,
img = cv.imread("abstract_lines.png")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# 2. apply the Canny edge detector with parameters min=150 and max=500,
edges = cv.Canny(img, 150, 500, apertureSize=3)
# 3. apply the HoughLines algorithm with a resolution of 5.0 deg for theta. and a resolution of 3.0 for rho. Set the threshold parameter to 100.
lines = cv.HoughLines(edges, 3.0, np.pi / 180 / 5.0, 100)

# Print the number of lines detected
print(len(lines))

# Draw the lines on the image
for line in lines:
  rho, theta = line[0]
  a = np.cos(theta)
  b = np.sin(theta)
  x0 = rho * a
  y0 = rho * b
  x1 = int(x0 + 1000 * -b)
  y1 = int(y0 + 1000 * a)
  x2 = int(x0 - 1000 * -b)
  y2 = int(y0 - 1000 * a)
  cv.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Show the image
cv.imshow("lines", img)
cv.waitKey(0)
cv.destroyAllWindows()
```

[9] Hough Lines and Vertical Lines

Extend your solution to the previous question to count the number of vertical lines, that is lines with theta being between 30.0 deg. and 60.0 deg. in the image.

5 marks

There are 36 lines with theta between 30.0 deg. and 60.0 deg.

B *I* U ~~S~~

```
import cv2 as cv
import numpy as np

# 1. Read the image into your program,
img = cv.imread("abstract_lines.png")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# 2. apply the Canny edge detector with parameters min=150 and max=500,
edges = cv.Canny(img, 150, 500, apertureSize=3)
# 3. apply the HoughLines algorithm with a resolution of 5.0 deg for theta. and a resolution of 3.0 for rho. Set the threshold parameter to 100.
lines = cv.HoughLines(edges, 3.0, np.pi / 180 / 5.0, 100)

# Only preserve the line that degree is between 30 and 60
lines = [line for line in lines if 30 < line[0][1] / np.pi * 180 < 60]

# Print the number of lines detected
print(len(lines))

# Draw the lines on the image
for line in lines:
  rho, theta = line[0]
  a = np.cos(theta)
  b = np.sin(theta)
  x0 = rho * a
  y0 = rho * b
  x1 = int(x0 + 1000 * -b)
  y1 = int(y0 + 1000 * a)
  x2 = int(x0 - 1000 * -b)
  y2 = int(y0 - 1000 * a)
  cv.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Show the image
cv.imshow("lines", img)
cv.waitKey(0)
cv.destroyAllWindows()
```

[10] Gaussian Blur and Hough Lines

Use the Gaussian blur algorithm of the OpenCV library. Use a kernel size of 3 and a sigma of 0.

5 marks

Given the input image and algorithm as described above, the OpenCV HoughLines algorithm after application of a blurring algorithm returns 46 lines.

B *I* U ~~S~~

```
import cv2 as cv
import numpy as np

# 1. Read the image into your program,
img = cv.imread("abstract_lines.png")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# ! Apply Gaussian Blur Before Canny Edge Detection
img = cv.GaussianBlur(img, (3, 3), 0)

# 2. apply the Canny edge detector with parameters min=150 and max=500,
edges = cv.Canny(img, 150, 500, apertureSize=3)
# 3. apply the HoughLines algorithm with a resolution of 5.0 deg for theta. and a resolution of 3.0 for rho. Set the threshold parameter to 100.
lines = cv.HoughLines(edges, 3.0, np.pi / 180 / 5.0, 100)

# Print the number of lines detected
print(len(lines))

# Draw the lines on the image
for line in lines:
  rho, theta = line[0]
  a = np.cos(theta)
  b = np.sin(theta)
  x0 = rho * a
  y0 = rho * b
  x1 = int(x0 + 1000 * -b)
  y1 = int(y0 + 1000 * a)
  x2 = int(x0 - 1000 * -b)
  y2 = int(y0 - 1000 * a)
  cv.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

# Show the image
cv.imshow("lines", img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Histogram of Oriented Gradient (HOG) Algorithm

This section covers topics using the Histogram of Oriented Gradient (HOG) algorithm.

[11] Sobel Edge Detection

Given is the following image.

	Block Col 0	**Block Col 1**	**Block Col 2**
Block Row 0	240 232 81 201 181 44 168 13 233	55 154 202 28 220 238 24 0 55	139 188 199 175 225 74 77 43 203
Block Row 1	100 55 132 195 175 23 101 183 140	101 144 176 79 15 55 149 108 150	175 158 76 145 13 54 215 32 98
Block Row 2	103 202 194 246 186 180 213 48 217	130 151 250 141 32 48 160 2 16	124 161 218 211 144 232 189 67 132

6 marks

Using the Sobel edge detector, calculate the **gradient direction** and **gradient magnitude** of the center pixel (pixel[4, 4]=15).

Gradient direction: 123.69006752597979 degrees

Gradient magnitude: 21.633307652783937

B I U S

```
import cv2 as cv
import numpy as np

# Image Array
img = np.array(
  [
    [240, 232, 81, 55, 154, 202, 139, 188, 199],
    [201, 188, 44, 28, 220, 238, 175, 225, 74],
    [168, 13, 233, 24, 0, 55, 77, 43, 203],
    [100, 55, 132, 101, 144, 176, 175, 158, 76],
    [195, 175, 23, 79, 15, 55, 145, 13, 54],
    [101, 183, 140, 149, 108, 150, 215, 32, 98],
    [103, 202, 194, 130, 151, 250, 124, 161, 218],
    [246, 186, 180, 141, 32, 48, 211, 144, 232],
    [213, 48, 217, 160, 2, 16, 189, 67, 132],
  ]
)

# The Center Pixel Value
center = img[4, 4]
print("Center Pixel Value:", center)

# Calculate the Gradient Direction and Magnitude of the Center Pixel
grad_x = np.gradient(img, axis=1)
grad_y = np.gradient(img, axis=0)
magnitude = np.sqrt(grad_x**2 + grad_y**2)
direction = np.arctan2(grad_y, grad_x)
center_pixel_direction = direction[4, 4]
center_pixel_magnitude = magnitude[4, 4]
# Turn the Direction into 0 to 180 Degrees
center_pixel_direction = np.abs(center_pixel_direction) * 180 / np.pi
print("Center Pixel Direction:", center_pixel_direction)
print("Center Pixel Magnitude:", center_pixel_magnitude)
```

[12] HOG Feature Descriptor

Using one block (i.e., a 3 by 3 neighborhood), calculate the **HOG feature descriptor** (9 weighted angle values for 20 degree buckets between 0 to 180 degrees) of the center pixel.

Use the absolute of the angle to map it to a direction between 0 to 180 degrees.

Calculate the HOG feature vector without normalization for this question.

6 marks

HOG feature descriptor (without normalization)

{table}

B I U S

```
❯ python3 task9.py
HOG Feature Vector (without normalization):
[0 1 0 1 0 1 5 1 0]

<Code: task9.py>
import numpy as np


def _gradient(image):
  # Use Sobel operator to get gradient
  grad_x = np.gradient(image, axis=1)
  grad_y = np.gradient(image, axis=0)
  # Get magnitude and direction
  magnitude = np.sqrt(grad_x**2 + grad_y**2)
  direction = np.arctan2(grad_y, grad_x)
  return magnitude, direction


def _hogDesript(ditTan):
  bins = np.arange(0, 180, 20) # Set bins
  ditTan = np.abs(ditTan) * 180 / np.pi # Turn radian to degree
  # Calculate weights
  weights, _ = np.histogram(ditTan, bins=bins)
  return weights


def getBlockHog(image):
  mag, dir = _gradient(image) # Get magnitude and direction
  blockDirection = []
  # Get direction of each block
  for i in range(1, 8, 3):
    for j in range(1, 8, 3):
      blockDirection.append(dir[i, j])
  # Calculate HOG feature vector
  hog_descriptor = _hogDesript(blockDirection)
  return hog_descriptor


# 定義圖像
img = np.array(
  [
    [240, 232, 81, 55, 154, 202, 139, 188, 199],
    [201, 188, 44, 28, 220, 238, 175, 225, 74],
    [168, 13, 233, 24, 0, 55, 77, 43, 203],
    [100, 55, 132, 101, 144, 176, 175, 158, 76],
    [195, 175, 23, 79, 15, 55, 145, 13, 54],
    [101, 183, 140, 149, 108, 150, 215, 32, 98],
    [103, 202, 194, 130, 151, 250, 124, 161, 218],
    [246, 186, 180, 141, 32, 48, 211, 144, 232],
    [213, 48, 217, 160, 2, 16, 189, 67, 132],
  ]
)

# Get HOG feature vector
hogFeat = getBlockHog(img)
```

```
# Print the HOG feature vector
print("HOG Feature Vector (without normalization):")
print(hogFeat)
```

[13] Normalized HOG Feature Descriptor

Normalize the HOG feature detector using a 3 by 3 block neighborhood, that is use a total of 9 blocks to normalize the HOG feature desriptor.

13 marks

HOG feature descriptor (with normalization)

Angle 0:20	0
Angle 20:40	0.11111111
Angle 40:60	0
Angle 60:80	0.11111111
Angle 80:100	0
Angle 100:120	0.11111111
Angle 120:140	0.55555556
Angle 140:160	0.11111111
Angle 160:180	0

```
import numpy as np


def _gradient(image):
  # Use Sobel operator to get gradient
  grad_x = np.gradient(image, axis=1)
  grad_y = np.gradient(image, axis=0)
  # Get magnitude and direction
  magnitude = np.sqrt(grad_x**2 + grad_y**2)
  direction = np.arctan2(grad_y, grad_x)
  return magnitude, direction


def _hogDesript(ditTan):
  bins = np.arange(0, 180, 20) # Set bins
  ditTan = np.abs(ditTan) * 180 / np.pi # Turn radian to degree
  # Calculate weights
  weights, _ = np.histogram(ditTan, bins=bins)
  return weights


def getBlockHog(image):
  mag, dir = _gradient(image) # Get magnitude and direction
  blockDirection = []
  # Get direction of each block
  for i in range(1, 8, 3):
    for j in range(1, 8, 3):
      blockDirection.append(dir[i, j])
  # Calculate HOG feature vector
  hog_descriptor = _hogDesript(blockDirection)
  return hog_descriptor


# 定義圖像
img = np.array(
  [
    [240, 232, 81, 55, 154, 202, 139, 188, 199],
    [201, 188, 44, 28, 220, 238, 175, 225, 74],
    [168, 13, 233, 24, 0, 55, 77, 43, 203],
    [100, 55, 132, 101, 144, 176, 175, 158, 76],
    [195, 175, 23, 79, 15, 55, 145, 13, 54],
    [101, 183, 140, 149, 108, 150, 215, 32, 98],
    [103, 202, 194, 130, 151, 250, 124, 161, 218],
    [246, 186, 180, 141, 32, 48, 211, 144, 232],
    [213, 48, 217, 160, 2, 16, 189, 67, 132],
  ]
)

# Get HOG feature vector
hogFeat = getBlockHog(img)

# Print the HOG feature vector
print("HOG Feature Vector (without normalization):")
print(hogFeat)

# Normalize HOG feature vector
hogFeat = hogFeat / 9

# Print the normalized HOG feature vector
print("HOG Feature Vector (with normalization):")
print(hogFeat)
```