ENIN 880CA Assignment #1

Elyar Zavvari

November 12, 2021

Students should read chapters 12 and 13 of the book: Robotics, Vision, and Control by Peter Corke (1st Edition). Students should also watch the following videos on MasterClasses under "computer vision" in QUT Robot Academy: Image Processing (10 lessons), Spatial Operators (11 lessons), and Feature Extraction (9 lessons). Students are allowed to use the following toolboxes/software libraries for doing exercise #1:

- Machine Vision Toolbox in Matlab by Peter Corke
- Image Processing Toolbox in Matlab by Mathworks
- Computer Vision Toolbox in Matlab by Mathworks
- OpenCV

1 Basics

Write a program that grabs a frame from your own camera and displays it. Add the following effects to the image: (1) negative image, (2) thresh-holding, (3) edge filtering.

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The original picture name is face and the resulted photos from code are saved in this folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "basics.py" in zipped file. The codes are as follows:*

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image #
image = cv2.imread('photos/face.jpeg', 0)

# Basics #

# Inverting the image #
inverted = np.invert(image)

# applying threshhold #
ret,thresh = cv2.threshold(image, 120, 255, cv2.THRESH_BINARY)

```

```
# applying edge detection #
edge = cv2.Canny(image,80,200)

# saving the imaged in photo/ directory #
cv2.imwrite('photos/inverted.jpeg', inverted)
cv2.imwrite('photos/thresh.jpeg', thresh)
cv2.imwrite('photos/edge.jpeg', edge)

# plotting the images resulted in basic #
f = plt.figure("basics")

f.add_subplot(3,2,1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Picture")

f.add_subplot(3,2,2)
plt.imshow(inverted, cmap="gray", vmin=0, vmax=255)
plt.title("Negative Picture")

f.add_subplot(3,2,3)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Picture")

f.add_subplot(3,2,4)
plt.imshow(thresh, cmap="gray", vmin=0, vmax=255)
plt.title("Threshhold Picture")

f.add_subplot(3,2,5)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Picture")

f.add_subplot(3,2,6)
plt.imshow(edge, cmap="gray", vmin=0, vmax=255)
plt.title("Edge Detected Picture")

plt.savefig('plots/basics-plot.png')

plt.show()
```

2 Convolution

Grab a frame of your own photo, and apply smoothing by: (1) using a 21 x 21 uniform kernel and a Gaussian kernel. Can you observe the ringing artefact in the former?

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The original picture name is face and the resulted photos from code are saved in this folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "convolution.py" in zipped file. The codes are as follows:*

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt

```

```
# Load the image #
image = cv2.imread('photos/face.jpeg', 0)

# Convolution #

# creating kernel matrix #
kernel = np.ones((21, 21), np.float32) / (21**2)

# convoluting the image withh kernel #
img_uniform = cv2.filter2D(image, -1, kernel)

#applying Gaussian blur on image #
img_gaussian = cv2.GaussianBlur(image, (21,21), 0)

# saving image in photos folder #
cv2.imwrite('photos/uniform.jpeg', img_uniform)
cv2.imwrite('photos/guassian.jpeg', img_gaussian)

# show the image #
f2 = plt.figure("convolution")

f2.add_subplot(2,2,1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Picture")

f2.add_subplot(2,2,2)
plt.imshow(cv2.cvtColor(img_uniform, cv2.COLOR_BGR2RGB))
plt.title("Blurred Picture with uniform kernel")

f2.add_subplot(2,2,3)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Picture")

f2.add_subplot(2,2,4)
plt.imshow(cv2.cvtColor(img_gaussian, cv2.COLOR_BGR2RGB))
plt.title("Blurred Picture with guassian kernel")

plt.savefig('plots/convolution-plot.png')
plt.show()
```

3 Template Matching

Select your left eye in a photo of you as a template (the template should have odd dimensions). Find the best match and where does it occur? What is the similarity to your other eye?

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The original picture name is face and the resulted photos from code are saved in this folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "template-matching.py" in zipped file. The codes are as follows:*

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
4
5  # Load the image #
6  image = cv2.imread('photos/face.jpeg', 0)
7
8  # Template Matching #
9  # Read the main image #
10 img_rgb = cv2.imread("photos/face.jpeg")
11 img = cv2.imread("photos/face.jpeg")
12
13 # Convert it to grayscale #
14 img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
15
16 # Read the template #
17 template = cv2.imread("photos/template.jpeg", 0)
18
19 # Store width and height of template in w and h #
20 w, h = template.shape[::-1]
21
22 # Perform match operations #
23 res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
24
25 # Specify a threshold #
26 threshold = 0.8
27
28 # Store the coordinates of matched area in a numpy array #
29 loc = np.where( res >= threshold)
30
31 # Draw a rectangle around the matched region #
32 for pt in zip(*loc[::-1]):
33     cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)
34
35 # Show the final image with the matched area #
36 # Show the image #
37 f3 = plt.figure("template-match")
38
39 f3.add_subplot(1,2,1)
40 plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
41 plt.title("Original Picture")
42
43 f3.add_subplot(1,2,2)
44 plt.imshow(cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB))
45 plt.title("Gaussian Blurred Picture")
46
47 plt.savefig('plots/template-match-plot.png')
48 plt.show()
```

4 Blobs

Create an image of an object with several holes in it (you could draw it on a piece of paper and take a picture, export it from a drawing program). Determine the outer, inner and total boundaries of the object.

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The resulted photos from code are saved in this*

folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "blobs.py" in zipped file. The codes are as follows:

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image #
image = cv2.imread('photos/face.jpeg', 0)


# blobs #
# Read image #
img = cv2.imread("photos/blob.png", cv2.IMREAD_GRAYSCALE)

# Setup SimpleBlobDetector parameters #
params = cv2.SimpleBlobDetector_Params()

# Filter by Convexity #
params.filterByConvexity = True
params.minConvexity = 0.0

# Filter by Circularity #
params.filterByCircularity = True
params.minCircularity = 0.5

# Set up the detector with default parameters #
detector = cv2.SimpleBlobDetector_create(params)

# Detect blobs #
keypoints = detector.detect(img)

# Draw detected blobs as red circles #
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS #
# ensures the size of the circle corresponds to the size of blob #
img_with_keypoints = cv2.drawKeypoints(img, keypoints, np.array([]),
    (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

f4 = plt.figure("blob")
plt.imshow(img_with_keypoints,  cmap="gray", vmin=0, vmax=255)
plt.title("Edges of Picture")

plt.savefig('plots/blob-plot.png')
plt.show()
```

5 Hough Transform

Apply the Hough Transform to one of your own images.

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The resulted photos from code are saved in this folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "hough.py" in zipped file. The codes are as follows:*

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image #
image = cv2.imread('photos/face.jpeg', 0)

# Hough Transform #
# Read image #
img = cv2.imread("photos/highway.jpeg")

# Convert the image to gray-scale #
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Find the edges in the image using canny detector #
edges = cv2.Canny(gray, 50, 200)

# Detect points that form a line #
# lines = cv2.HoughLinesP(edges, 1, np.pi/180, #
# max_slider=50, minLineLength=10, maxLineGap=250) #

# Draw lines on the image #
#for line in lines: #
#    x1, y1, x2, y2 = line[0] #
#    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3) #

minLineLength = 10
maxLineGap = 50
#lines = cv2.HoughLinesP(edges, 1, np.pi/150, 200,#
#        minLineLength,maxLineGap) #
#lines = cv2.HoughLinesP(edges, 1, #
# np.pi/180, 20, minLineLength=5, maxLineGap=100) #
lines = cv2.HoughLines(edges,1,np.pi/180,230)
for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

# Show result #
f6 = plt.figure("hough")
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Edges of Picture")

plt.savefig('plots/hough-plot.png')

plt.show()
```

6 Corner Detection

Take a picture of your desk and apply Harris, Nobel, and Shi-Tomas corner detectors on that and compare the results.

Solution: *This program is written in python by using OpenCV libraries. You'll find the resulted pictures in the "photos" folder. The resulted photos from code are saved in this folder. Moreover, the plots are saved in a folder named "plots". The python code related to this problem is named "corner.py" in zipped file. The codes are as follows:*

```
1
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image #
image = cv2.imread('photos/face.jpeg', 0)

# corner detection #
# loading image #
image = cv2.imread('photos/desk.jpg')

# convert the input image into grayscale #
img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# convert the data type #
img_gray = np.float32(img_gray)

# implementing cv2.cornerHarris method #
hcd_img = cv2.cornerHarris(img_gray, 5, 5, 0.08)

# marking dilated corners #
hcd_img = cv2.dilate(hcd_img, None)

# reverting back to the original image #
image[hcd_img > 0.01 * hcd_img.max()]=[0, 0, 255]

# Show result #
f7 = plt.figure("corner")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Corners detected with Harris Algorithm")
plt.savefig('plots/corner-plot.png')
plt.show()
```