Features

شیوا رادمنش

اطلاعات گزارش

تاریخ: ۱۳۹۹/۱۱/۵

واژگان کلیدی:

تبدیل هندسی

گوشه یاب Harris

فیچر

matching

چکیده

در این گزارش در بخش اول به روش محاسبه‌ی تبدیل هندسی با استفاده از match کردن فیچرهای دو تصویر و سپس بازسازی تصویر اولیه به وسیله‌ی اعمال ماتریس تبدیل به تصویر پرداخته شده است. در بخش دوم به چگونگی پیاده سازی الگوریتم Harris Feature Detection و بررسی نتایج آن پرداخته می‌شود.

۱- مقدمه

نوشتار حاضر در بخش اول به بررسی نحوه‌ی بدست آوردن تبدیلات هندسی با استفاده از feature matching با الگوریتم ORB و نهایتا بازسازی تصویر با استفاده از ماتریس تبدیل بدست آمده پرداخته شده است. در بخش دوم به بررسی الگوریتم Harris و نحوه‌ی پیاده سازی آن و همچنین اعمال آن بر یک تصویر پرداخته شده است. تمام پیاده سازی‌های این تمرین با استفاده از کتابخانه‌های opencv و scikit-image پایتون انجام شده است.

۲- شرح تکنیکال

۱-۲- سوال ۷.۱.۱

در این بخش به بازسازی تصاویر Attack2 (تصاویری که که از نظر هندسی تغییر کرده اند بدون اینکه image filtering به آنها اعمال شده باشد) به کمک تصویر halftone (تصویری که کیفیت آن کاهش پیدا کرده اما هندسه‌ی آن دچار تغییر نشده) و تصویر نظیر در Attack1 (تصاویری که علاوه بر تبدیل هندسی به آنها فیلترهای پردازش تصویر هم به آنها اعمال شده). بدین منظور ابتدا با استفاده تصویر halftone و Attack 1، تبدیل هندسی مورد نظر را بدست آورده سپس آن تبدیل را به تصویر نظیر در Attack 2 اعمال می‌کنیم تا تصویر اصلی بدست بیاید.

برای بدست آوردن تبدیل هندسی اعمال شده با استفاده از تصویر halftone و تصویر مورد نظر در Attaack 1 به شرح زیر عمل می‌کنیم:

- ابتدا هر دو تصویر را به تصاویر grayscale تبدیل کرده.
- حال فیچرهای مهم هر دو تصویر را پیدا کرده و description نقاط را بدست آورده، در اینجا برای detect و descript نقاط مهم هر تصویر از الگوریتم ORB که توسط کتابخانه‌ی opencv پیاده سازی شده است، استفاده شده است که هم نقاط مهم و هم description آنها را در اختیار ما قرار می‌دهد.

الگوریتم ORB: این الگوریتم تلفیقی از یک الگوریتم detector به نام FAST و یک الگوریتم descriptor به نام BRIEF می‌باشد. روش FAST یک دایره با شعاع ۱۶ پیکسل در اطراف هر پیکسل در نظر می‌گیرد. و نقاطی از دایره که intensity آنها از یک threshold خاصی از intensity مرکز دایره کمتر یا بیشتر است را مشخص می‌کند. نقاط گوشه با تعداد پیکسل‌های تاریکتر و یا روشن‌تر مشخص شده در اطراف آنها شناسایی می‌شوند. در الگوریتم ORB پس از اعمال FAST از Harris Corner Detector برای انتخاب n گوشه‌ی برتر استفاده می‌شود. BRIEF یک descriptor جنرال است که می‌تواند با detector های مختلف ترکیب شود. این روش در برابر تبدیلات photometric و geometric مقاوم است.

- حال key point های بدست آمده‌ی هر دو تصویر در مرحله‌ی قبل را به هم match می‌کنیم. برای matching از روش brute Force استفاده شده است، به این صورت که تمام فیچرهای descriptor اول با فیچرهای descriptor دوم مقایسه می شوند و به هر مقایسه یک مقدار فاصله داده می‌شود و بهترین نتیجه مطابقت در نظر گرفته می‌شود. در این از معیار hamming به عنوان معیار فاصله استفاده شده است.
- حال match های پیدا شده را بر اساس معیار فاصله‌ی آنها مرتب کرده و ۹۰ درصد برتر را انتخاب کرده(match های نویز حذف می‌شود)
- با استفاده از match های انجام شده، تبدیل هموگرافی را بدست آورده.(هموگرافی تبدیلی است (ماتریسی 3×3) که نقاط

موجود در یک تصویر را به نقاط مربوطه در تصویر دیگر ترسیم می کند.)

حال می‌توان تبدیل بدست آمده را به تصویر نظیر در Attack2 اعمال کرد.

۲-۲- سوال ۷.۲.۱

در این بخش به نحوه‌ی پیاده سازی الگوریتم Harris Feature Detection پرداخته شده است.

ایده‌ی اصلی این الگوریتم این است که گوشه‌ها که نقاط مهمی در تصویر هستند، با جابه‌جایی در هر یک از جهات تغییرات قابل توجهی دارند. مراحل این الگوریتم به شرح زیر است:

- با استفاده از فیلتر sobel مشتق افقی و عمودی را بدست آورده. I_x مشتق افقی و I_y می‌باشد.
- با استفاده از I_x و I_y که در مرحله‌ی قبل بدست آمده ${I_x}^2$ و ${I_y}^2$ و I_xI_y را محاسبه کرده.
- به ${I_x}^2$ و ${I_y}^2$ و I_xI_y فیلتر گاوسی اعمال کرده.
- با استفاده از مقادیر بدست آمده ماتریس M را به شکل زیر می‌سازیم.

$$M = \sum w(x,y)\begin{bmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y \end{bmatrix}$$

- اگر هر دو مقادیر ویژه‌ی ماتریس M (λ_1 و λ_2)در یک نقطه، مقدار بزرگی داشته باشند به این معنا است که تغییرات در همه‌ی جهات در آن نقطه می‌باشد. بر این اساس معیاری به نام R به شکل زیر برای هر نقطه تعریف شده است.

$$R = det(M) - \alpha(trace(M))^2$$
$$det(M) = \lambda_1\lambda_2$$
$$trace(M) = \lambda_1 + \lambda_2$$

مقدار α در اینجا ۰.۰۴ در نظر گرفته شده است.

- برای R یک مقدار آستانه در نظر گرفته که اگر مقدار R از مقدار آن آستانه بزرگتر باشد، آن نقطه گوشه در نظر گرفته می‌شود.
- در مرحله‌ی آخر non maximum suppression انجام داده به این صورت که اگر در اطراف یک نقطه چند فیچر(گوشه) بدست آمده قوی ترین گوشه که در واقع مقدار R آن بیشتر است را انتخاب می‌کنیم(به همسایگی ۲۵ پیکسل).

۳- نتایج

۳-۱- سوال ۷.۱.۱

در جدول زیر نتایج حاصل از اعمال تبدیل هندسی به دست آمده بر تصاویر Attack2 با استفاده از دو معیار MSE (خطای میانگین مربعات) و SSIM (شباهت شاخص ساختاری که محدوده‌ی آن بین ۰ و ۱ می‌باشد) با تصویر اصلی مقایسه شده و همچنین تعداد match های استفاده شده برای بدست آوردن ماتریس تبدیل، گزارش شده است.

Type 1 (Histeq)			Type 2 (sharpen)			Type 3 (Gaussfilt)			Type 4 (Bilafilt)			Mean			STD		
SSIM	MSE	MP	SSIM	MSE	MP	SSIM	MSE	MP	SSIM	MSE	MP	SSIM	MSE	MP	SSIM	MSE	MP
0.822	297.6	2232	0.858	153.9	1703	0.445	7691.6	359	0.882	93.05	1330	0.752	2059	1406	0.178	3252.8	684.2

شکل۱- تصویر اصلی

شکل۲- تصویر halftone

۳-۱-۱ تصویر type1

به تصویر Attack 1، در اینجا histogram equalization اعمال شده است.

شکل۳- تصویر Attack1- type1

شکل۴- تصویر Attack2-type1

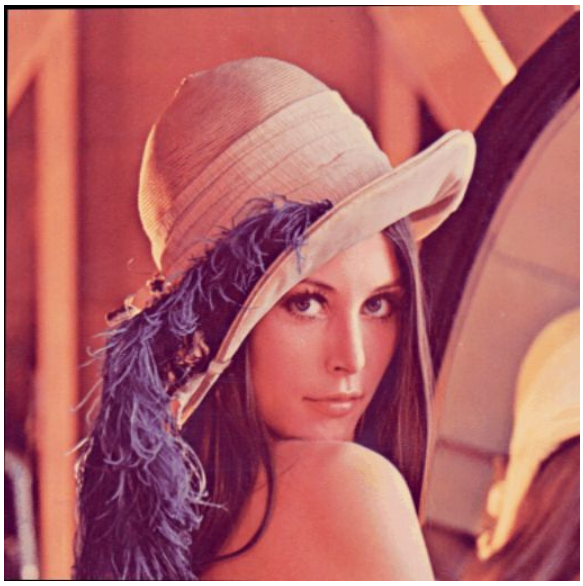
شکل۵- تصویر بازسازی شده از اعمال تبدیل هندسی بر تصویر Attack2-type1

۲-۱-۳- تصویر type2

به تصویر Attack 1، در اینجا فیلتر sharpening اعمال شده است.

شکل۶- تصویر Attack1- type2

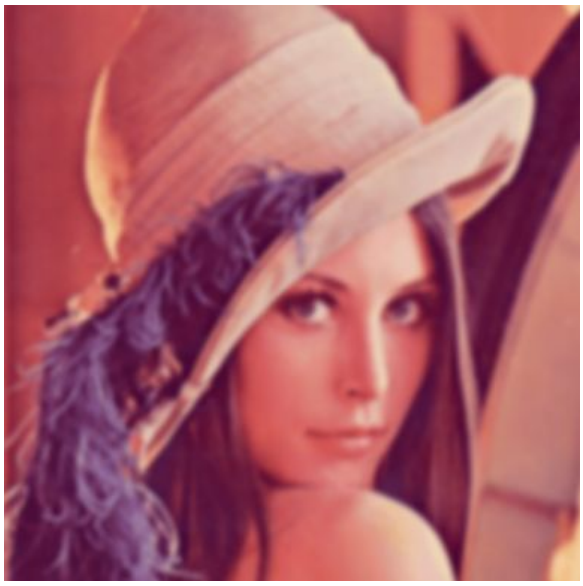
شکل۷- تصویر Attack2-type2



شکل۸- تصویر بازسازی شده از اعمال تبدیل هندسی بر تصویر Attack2-type2

۳-۱-۳- تصویر type3

به تصویر Attack 1، در اینجا فیلتر gaussian اعمال شده است.



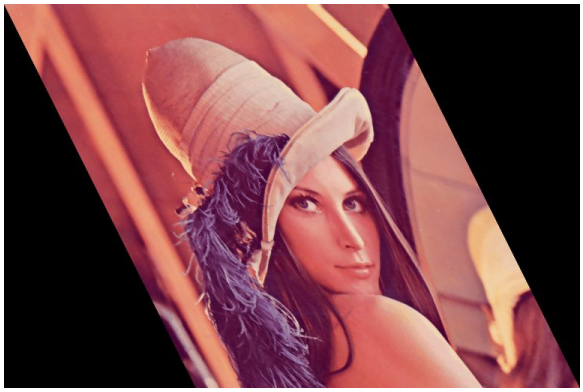
شکل۹- تصویر Attack1- type3

شکل۱۰- تصویر Attack2-type3

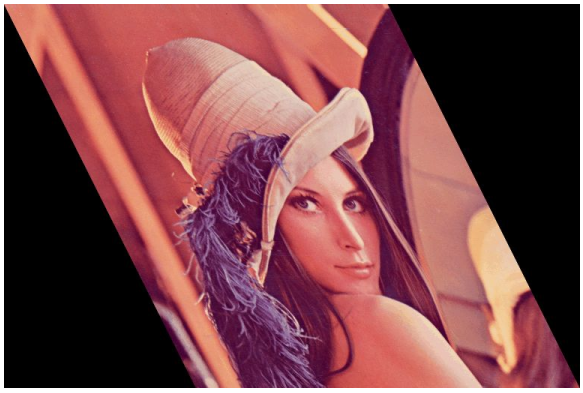
شکل۱۱- تصویر بازسازی شده از اعمال تبدیل هندسی بر تصویر Attack2-type3

۴-۱-۳- تصویر type4

به تصویر Attack 1، در اینجا فیلتر bilateral اعمال شده است.



شکل۱۲- تصویر Attack1- type4



شکل۱۳- تصویر Attack2-type4

شکل۱۴- تصویر بازسازی شده از اعمال تبدیل هندسی بر تصویر Attack2-type4

بر اساس نتایج ارائه شده مشاهده می‌شود که تصویر بازسازی شده از روی تبدیلی که از تصویری که بر آن فیلتر bilateral شده بود، به دست آمده نسبت به سایر تصاویر بازسازی شده هم از نظر بصری و هم از نظر معیار MSE و SSIM به تصویر اصلی شبیه تر بوده. همچنین تصویر بازسازی شده از تصویر type3 که به تصویر Attack1 آن فیلتر gaussian اعمال شده بود، هم از نظر بصری و هم از نظر معیار MSE و SSIM تفاوت بیشتری با تصویر اصلی دارد.

۲-۳- سوال ۷.۲.۱

در ادامه نتایج اعمال الگوریتم Harris Corner Detection به تصویر building قابل مشاهده است. انتخاب نقاط قبل و بعد از non maximum suppression بررسی شده است.

شکل۵- تصویر building پس از اعمال الگوریتم harris بدون اعمال non maximum suppression

شکل۶- تصویر building پس از اعمال الگوریتم harris بعد از اعمال non maximum suppression

maximum suppression از بین فیچرهای مجاور(در اینجا یک همسایگی به شعاع ۲۵ پیکسل) قوی ترین فیچر بر اساس معیار R انتخاب می‌شود و بقیه کنار گذاشته می‌شوند. گوشه‌های این تصویر شامل نقاط گوشه در ساختمان‌ها و همچنین برگ‌های درخت می‌باشند که به خوبی تشخیص داده شده اند.

مشاهده می‌شود که این الگوریتم به خوبی نقاط گوشه را تشخیص داده است و پس از اعمال non

منابع

[1] Digital Image Processing, Rafael C. Gonzalez, Rechard E. Wood

[2] Learning OpenCV 3 Computer Vision with Python - PACKT publishing - Joe Minichino, JOs

[3] https://www.geeksforgeeks.org/image-registration-using-opencv-python/

[4] https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html

Appendix

7.1.1

```
import cv2
import numpy as np
from skimage.metrics import mean_squared_error as compare_mse
from skimage.metrics import structural_similarity  as compare_ssim


def image_reconstruction(attack1_color, halftone_color, attack2_color):
   # Convert to grayscale.
   img1 = cv2.cvtColor(attack1_color, cv2.COLOR_BGR2GRAY)
   img2 = cv2.cvtColor(halftone_color, cv2.COLOR_BGR2GRAY)
   height, width = img2.shape
```

```
    # Create ORB detector with 5000 features.
    orb_detector = cv2.ORB_create(5000)

    # Find keypoints and descriptors.
    # The first arg is the image, second arg is the mask
    #  (which is not reqiured in this case).
    kp1, d1 = orb_detector.detectAndCompute(img1, None)
    kp2, d2 = orb_detector.detectAndCompute(img2, None)

    # Match features between the two images.
    # We create a Brute Force matcher with
    # Hamming distance as measurement mode.
    matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    # Match the two sets of descriptors.
    matches = matcher.match(d1, d2)

    # Sort matches on the basis of their Hamming distance.
    matches.sort(key=lambda x: x.distance)

    # Take the top 90 % matches forward.
    # matches = matches[:int(len(matches) * 100)]
    no_of_matches = len(matches)

    # Define empty matrices of shape no_of_matches * 2.
    p1 = np.zeros((no_of_matches, 2))
    p2 = np.zeros((no_of_matches, 2))

    for i in range(len(matches)):
        p1[i, :] = kp1[matches[i].queryIdx].pt
        p2[i, :] = kp2[matches[i].trainIdx].pt

    # Find the homography matrix.
    homography, mask = cv2.findHomography(p1, p2, cv2.RANSAC)

    # Use this matrix to transform the
    # colored image wrt the reference image.
    transformed_img = cv2.warpPerspective(attack2_color,
                                          homography, (width, height))

    # Save the output.
    # cv2.imwrite(f'images/outputs/{img_name}.bmp', transformed_img)

    return transformed_img, len(matches)
```

```
if __name__ == "__main__":
   for i in range(1, 5):

       img_name = "{}".format(i)
       original_index = i

       attack1 = cv2.imread(f"images/Attack 1/{img_name}.bmp")  # Image to be
aligned.
       halftone = cv2.imread("images/Reference.bmp")  # Reference image.
       attack2 = cv2.imread(f"images/Attack 2/{img_name}.bmp")
       original = cv2.imread("images/Original.bmp")

       transformed_image, match_count = image_reconstruction(attack1, halftone,
attack2)
       cv2.imwrite(f'images/outputs/7_1_1/{i}.bmp', transformed_image)

       transformed_gray = cv2.cvtColor(transformed_image, cv2.COLOR_BGR2GRAY)
       original_gray = cv2.cvtColor(original, cv2.COLOR_BGR2GRAY)

       ssim_score, ssim_dif = compare_ssim(original_gray, transformed_gray,
full=True)
       mse_score = compare_mse(original_gray, transformed_gray)

       with open("result.txt", 'a') as file:
           file.write("\n------------------\n{}.bmp\nSSIM = {}\nMSE = {}\nMatch
Point = {}\n".format(i,ssim_score, mse_score, match_count))
```

7.2.1

```
import numpy as np
import cv2

# Kernel operation using input operator of size 3*3
def GetSobel(image, Sobel, width, height):
   # Initialize the matrix
   I_d = np.zeros((width, height), np.float32)

   # For every pixel in the image
   for rows in range(width):
```

```
        for cols in range(height):
            # Run the Sobel kernel for each pixel
            if rows >= 1 or rows <= width-2 and cols >= 1 or cols <= height-2:
                for ind in range(3):
                    for ite in range(3):
                        I_d[rows][cols] += Sobel[ind][ite] * image[rows - ind -
1][cols - ite - 1]
            else:
                I_d[rows][cols] = image[rows][cols]

    return I_d


# Method implements the Harris Corner Detection algorithm
def HarrisCornerDetection(image):

    # The two Sobel operators - for x and y direction
    SobelX = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    SobelY = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    w, h = image.shape

    # X and Y derivative of image using Sobel operator
    ImgX = GetSobel(image, SobelX, w, h)
    ImgY = GetSobel(image, SobelY, w, h)

    # # Eliminate the negative values
    # There are multiple ways this can be done
    # 1. Off setting with a positive value (commented out below)
    # 2. Setting negative values to Zero (commented out)
    # 3. Multiply by -1 (implemented below, found most reliable method)
    # ImgX += 128.0
    # ImgY += 128.0
    for ind1 in range(w):
        for ind2 in range(h):
            if ImgY[ind1][ind2] < 0:
                ImgY[ind1][ind2] *= -1
                # ImgY[ind1][ind2] = 0
            if ImgX[ind1][ind2] < 0:
                ImgX[ind1][ind2] *= -1
                # ImgX[ind1][ind2] = 0

    # # Display the output results after Sobel operations
    # cv2.imshow("SobelX", ImgX)
    # cv2.imshow("SobelY", ImgY)
```

```
    ImgX_2 = np.square(ImgX)
    ImgY_2 = np.square(ImgY)

    ImgXY = np.multiply(ImgX, ImgY)
    ImgYX = np.multiply(ImgY, ImgX)

    #Use Gaussian Blur
    Sigma = 1.4
    kernelsize = (3, 3)

    ImgX_2 = cv2.GaussianBlur(ImgX_2, kernelsize, Sigma)
    ImgY_2 = cv2.GaussianBlur(ImgY_2, kernelsize, Sigma)
    ImgXY = cv2.GaussianBlur(ImgXY, kernelsize, Sigma)
    ImgYX = cv2.GaussianBlur(ImgYX, kernelsize, Sigma)
    # print(ImgXY.shape, ImgYX.shape)

    alpha = 0.04
    R = np.zeros((w, h), np.float32)
    # For every pixel find the corner strength
    for row in range(w):
        for col in range(h):
            M_bar = np.array([[ImgX_2[row][col], ImgXY[row][col]],
[ImgYX[row][col], ImgY_2[row][col]]])
            R[row][col] = np.linalg.det(M_bar) - (alpha *
np.square(np.trace(M_bar)))
    return R


#### Main Program ####
firstimagename = "images/Building.jpg"

# Get the first image
firstimage = cv2.imread(firstimagename, cv2.IMREAD_GRAYSCALE)
w, h = firstimage.shape

# Covert image to color to draw colored circles on it
bgr = cv2.cvtColor(firstimage, cv2.COLOR_GRAY2RGB)

# Corner detection
R = HarrisCornerDetection(firstimage)

# Empirical Parameter
# This parameter will need tuning based on the use-case
CornerStrengthThreshold = 80000000
```

```
# Plot detected corners on image
radius = 5
color = (255, 0, 255)  # Green
thickness = 2

PointList = []
# Look for Corner strengths above the threshold
for row in range(w):
   for col in range(h):
       if R[row][col] > CornerStrengthThreshold:
           # print(R[row][col])
           max = R[row][col]

           # Local non-maxima suppression
           skip = False
           for nrow in range(25):
               for ncol in range(25):
                   if row + nrow - 2 < w and col + ncol - 2 < h:
                       if R[row + nrow - 2][col + ncol - 2] > max:
                           skip = True
                           break

           if not skip:
               # Point is expressed in x, y which is col, row
               cv2.circle(bgr, (col, row), radius, color, thickness)
               PointList.append((row, col))

# Display image indicating corners and save it
#cv2.imshow("Corners", bgr)
outname = "output/7_2_1/" + str(CornerStrengthThreshold) + ".png"
cv2.imwrite(outname, bgr)

# cv2.waitKey(0)
# cv2.destroyAllWindows()
```