تمرین سری ۳ - سوال ۱

محمدجواد شریعتی 96100414

بخش ۱:

در ابتدا سعی کردم خطها را به صورت اتوماتیک و با کمک Canny Edge Detection و Hough Transform بدست آورم ولی در کلاستر کردن اتوماتیک آن‌ها ناموفق بودم و نمی‌توانستم ۳ دسته ای که مدنظرم هست را به درستی انتخاب کنم به همین دلیل خطها را دستی انتخاب کردم. تلاشی که برای اتوماتیک کردن خطها کردم چنین بود که نتیجه نداد:

```
# automatic
# src_gray_image = cv2.cvtColor(src_img, cv2.COLOR_BGR2GRAY)
# sigma = 1
# edges = feature.canny(src_gray_image, sigma)
# edges2 = np.vectorize(lambda x: 255 if x else 0)(edges)
# cv2.imwrite("out/edges.jpg", edges2)
#
# lines = transform.probabilistic_hough_line(edges, threshold=650, line_length=100,
#                                            line_gap=10)
# sum = 0
# for line in lines:
#     if line[0][0] == line[1][0] or 0.3 <= (line[1][1] - line[0][1]) / (line[1][0] - line[0][0]) <= 10 or \
#             -0.2 >= (line[1][1] - line[0][1]) / (line[1][0] - line[0][0]) >= -10:
#         continue
#
#     eq = line_eq(line[0], line[1])
#     if 28 <= eq[0] <= 32:
#         cv2.line(src_img, line[0], line[1], color=(0, 0, 255), thickness=3)
#     elif 18 <= eq[0] <= 22:
#         cv2.line(src_img, line[0], line[1], color=(0, 255, 0), thickness=3)
#     elif -0.1 <= eq[0] <= 0.1:
#         cv2.line(src_img, line[0], line[1], color=(255, 0, 0), thickness=3)
#     else:
#         cv2.line(src_img, line[0], line[1], color=(255, 255, 255), thickness=3)
#         print(eq[0])
#         sum += 1
#
# print(sum)
# cv2.imwrite("out/lines12.jpg", src_img)

# kmeans = KMeans(n_clusters=3, random_state=1).fit(points)
```

در ادامه خطها را دستی انتخاب کردم و در تابع‌های axis_lines این خطها را مشخص کردم، مثلا بدین شکل:

```python
# TODO read lines from file if not automate
def z_axis_lines(img_copy):
    lines = []

    p0 = (1463, 720)
    p1 = (1506, 2000)
    line = line_eq(p0, p1)
    lines.append(line)
    cv2.line(img_copy, p0, p1, color=(0, 0, 255), thickness=3)

    p0 = (1414, 720)
    p1 = (1456, 2000)
    line = line_eq(p0, p1)
    lines.append(line)
    cv2.line(img_copy, p0, p1, color=(0, 0, 255), thickness=3)

    p0 = (1135, 520)
    p1 = (1187, 2100)
    line = line_eq(p0, p1)
    lines.append(line)
    cv2.line(img_copy, p0, p1, color=(0, 0, 255), thickness=3)
```

درنهایت خط‌های انتخاب شده بدین صورت شدند:

که خطوط قرمز در راستای محور Z ، خط‌های سبز در راستای X و خط‌های آبی در راستای محور Y قرار دارند.

پس از آن برای هر یک از محورها Vanishing Point آن را محاسبه کردم و خط افق را هم بدست آوردم.

```
img_copy = copyof(src_image)

# X axis
x_lines = x_axis_lines(img_copy)
V_x = calculate_vanishing_point(x_lines)
print("V_x = ", V_x)

# Y axis
y_lines = y_axis_lines(img_copy)
V_y = calculate_vanishing_point(y_lines)
print("V_y = ", V_y)

# Z axis
z_lines = z_axis_lines(img_copy)
V_z = calculate_vanishing_point(z_lines)
print("V_z = ", V_z)

h = line_eq(V_x, V_y)
print("h: ", "a=", h[0], ", b=", 1, ", c =", h[1], ",   a^2+b^2= ", 1 + h[0] ** 2)
```

که تابع line_eq هم به سادگی معادله خط را بدست می‌آورد.

```
# y = ax + b
def line_eq(p1, p2):
    a = (p2[1] - p1[1]) / (p2[0] - p1[0])
    b = p1[1] - (a * p1[0])
    return a, b
```

که vanishing point ها و خط افق بدین شکل هستند:

```
V_x =  (8249.610434476259, 2402.5586765492417)
V_y =  (-28644.19538099924, 4191.266510918929)
V_z =  (-2407.080058669275, -112670.81249853193)
h:  a= -0.04848260554402862 , b= 1 , c = 2802.5212851358565 ,   a^2+b^2=  1.002350563040338
```

پس معادله خط افق به فرمت خواسته شده به صورت زیر است:

$$-0.04848x + y + 2802.52128 = 0$$

برای بدست آوردن Vanishing Point خطهایی که رسم کردم بدین شکل عمل کردم:

برای هر جفت خط از خطهای یک دسته (یک محور مختصات) تقاطع آن دو خط را بدست می‌آورم و مجموعه تمام این تقاطعها را به MeanShift می‌دهم تا کلاسترهای مختلف را بدست آورد.

کلاستری که بیشترین تعداد نقطه را دارد (لیبل ۰) را درنظر می‌گیرم و برای بدست آوردن vanishing point از بین نقاط این کلاستر بین تمام نقاط آن میانگین میگیرم و این نقطه را به عنوان Vanishing Point نهایی این دسته (محور مختصات) خروجی می‌دهم.

```
def calculate_vanishing_point(lines):
    intersects = []
    i = 0
    while i < len(lines):
        j = i + 1
        while j < len(lines):
            intersects.append(line_intersection(lines[i], lines[j]))
            j += 1
        i += 1

    points = np.array(intersects)
    mean_shift = MeanShift().fit(points)

    # calculate mean of the biggest cluster
    count = 0
    sum_x = 0
    sum_y = 0
    for i in range(len(points)):
        if mean_shift.labels_[i] == 0:  # label 0 is biggest cluster
            count += 1
            sum_x += points[i][0]
            sum_y += points[i][1]

    vp = (sum_x / count, sum_y / count)
    return vp


def line_intersection(l1, l2):
    a1, b1 = l1
    a2, b2 = l2

    x = (b2 - b1) / (a1 - a2)
    y = (a1 * x) + b1
    return x, y
```

درنهایت هم ریزالت های خواسته شده را می‌سازم:

```
x_coordinates = [V_x[0], V_y[0], V_z[0]]
y_coordinates = [V_x[1], V_y[1], V_z[1]]

plt.imshow(cv2.cvtColor(src_image, cv2.COLOR_BGR2RGB))
plt.axline(V_x, V_y)
plt.xlim([0, width])
plt.ylim([height, 0])
plt.savefig("out/res01.jpg")

plt.clf()

plt.imshow(cv2.cvtColor(src_image, cv2.COLOR_BGR2RGB))
plt.scatter(x_coordinates, y_coordinates)
plt.annotate("Vx", V_x)
plt.annotate("Vy", V_y)
plt.annotate("Vz", V_z)
plt.axline(V_x, V_y)
plt.savefig("out/res02.jpg")
```

بخش ۲:

```
V_x = (8249.610434476259, 2402.5586765492417)
V_y = (-28644.19538099924, 4191.266510918929)
V_z = (-2407.080058669275, -112670.81249853193)

a1, b1 = V_x
a2, b2 = V_y
a3, b3 = V_z

p_y = ((a1 * (a2 - a3)) + (b1 * (b2 - b3))) - (((a2 - a3) * a2 * (a1 - a3)) / (a1 - a3)) - (((a2 - a3) * b2 * (b1 - b3)) / (a1 - a3))
p_y = p_y / ((b2 - b3) - (((a2 - a3) * (b1 - b3)) / (a1 - a3)))
p_x = ((a2 * (a1 - a3)) + (b2 * (b1 - b3)) - ((b1 - b3) * p_y)) / (a1 - a3)
principal_point = (p_x, p_y)
print("principal point: ", principal_point)

focal_length = math.sqrt((-(p_x ** 2)) + (-(p_y ** 2)) + ((a1 + a2) * p_x) + ((b1 + b2) * p_y) - ((a1 * a2) + (b1 * b2)))
print("focal length: ", focal_length, "\n")
```

برای بدست آوردن principal point و فاصله کانونی دوربین، باتوجه به vanishing point هایی که در بخش قبلی بدست آوردم و اینکه هرسه آنها متناهی هستند، از فرمول‌های اسلاید زیر استفاده کردم و با آنها مقادیر خواسته شده را بدست آوردم.

3 finite vanishing points:

$$\mathbf{v}_1 = \begin{bmatrix} a_1 \\ b_1 \\ 1 \end{bmatrix} \quad \mathbf{v}_2 = \begin{bmatrix} a_2 \\ b_2 \\ 1 \end{bmatrix} \quad \mathbf{v}_3 = \begin{bmatrix} a_3 \\ b_3 \\ 1 \end{bmatrix}$$

$$\begin{cases} eq1: & -f^2 - p_x^2 - p_y^2 + (a_1 + a_2)p_x + (b_1 + b_2)p_y = a_1a_2 + b_1b_2 \\ eq2: & -f^2 - p_x^2 - p_y^2 + (a_1 + a_3)p_x + (b_1 + b_3)p_y = a_1a_3 + b_1b_3 \\ eq3: & -f^2 - p_x^2 - p_y^2 + (a_2 + a_3)p_x + (b_2 + b_3)p_y = a_2a_3 + b_2b_3 \end{cases}$$

$$\begin{cases} eq1 - eq3: & (a_1 - a_3)p_x + (b_1 - b_3)p_y = a_2(a_1 - a_3) + b_2(b_1 - b_3) \\ eq1 - eq2: & (a_2 - a_3)p_x + (b_2 - b_3)p_y = a_1(a_2 - a_3) + b_1(b_2 - b_3) \end{cases}$$

$$eq1: \quad f^2 = -p_x^2 - p_y^2 + (a_1 + a_2)p_x + (b_1 + b_2)p_y - (a_1a_2 + b_1b_2)$$

که مقادیر خواسته شده چنین بدست آمد:

```
principal point:  (3116.0985282919273, 1250.0159096160994)
focal length:  12635.344467840621
```

برای بدست آوردن ماتریس کالیبراسیون دوربین هم کافی است مقادیر بدست آمده را جایگذاری کنیم:

```
Calibration Matrix:
[[1.26353445e+04 0.00000000e+00 3.11609853e+03]
 [0.00000000e+00 1.26353445e+04 1.25001591e+03]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
```

در مرحله بعد، چون حالا دوربین ما کالیبره شده می‌توانیم ماتریس دوران R را بدست آوریم. این درواقع ماتریسی است که دستگاه مختصات جهان را به دستگاه مختصات دوربین منطبق می‌کند (از لحاظ دوران-نه جابه‌جایی). برای این‌کار از اسلاید زیر کمک گرفتم:

When the camera is calibrated:

$$\lambda_i \mathbf{v}_i = \mathbf{K}\,[\mathbf{R} \mid \mathbf{t}] \begin{bmatrix} \mathbf{e}_i \\ 0 \end{bmatrix} = \mathbf{K}\,\mathbf{R}\,\mathbf{e}_i$$

$$\lambda_i\ \mathbf{K}^{-1}\,\mathbf{v}_i = \mathbf{R}\,\mathbf{e}_i = \mathbf{r}_i$$

$$\mathbf{r}_i^t\ \ \mathbf{r}_i = 1 \Longrightarrow \lambda_i^2\ \mathbf{v}_i^t\ \mathbf{K}^{-t}\ \mathbf{K}^{-1}\ \mathbf{v}_i = 1$$

$$\lambda_i = \frac{1}{\sqrt{\mathbf{v}_i^t\ \mathbf{B}\ \mathbf{v}_i}}$$

$$\mathbf{R} = \mathbf{K}^{-1}[\lambda_1\mathbf{v}_1 \quad \lambda_2\mathbf{v}_2 \quad \lambda_3\mathbf{v}_3]$$

ابتدا ماتریس معروف B را بدست آوردم و سپس با محاسبه لانداها R را بدست آوردم:

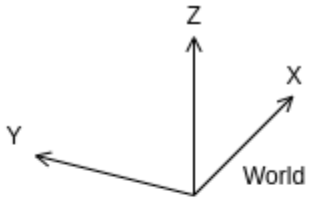
```python
# Calculation Rotation Matrix
B = (1 / (focal_length ** 2)) * np.array([
    [1, 0, -p_x],
    [0, 1, -p_y],
    [-p_x, -p_y, (focal_length ** 2) + (p_x ** 2) + (p_y ** 2)]
])

v0 = np.array([
    [V_x[0]],
    [V_x[1]],
    [1]
])
v1 = np.array([
    [V_y[0]],
    [V_y[1]],
    [1]
])
v2 = np.array([
    [V_z[0]],
    [V_z[1]],
    [1]
])
v = [v0, v1, v2]

lambdas = []
for i in range(3):
    lambdas.append(1 / math.sqrt(v[i].T @ B @ v[i]))

# World-to-Camera Rotation Matrix
R = np.linalg.inv(K) @ np.concatenate((lambdas[0] * v[0], lambdas[1] * v[1], lambdas[2] * v[2]), axis=1)
print("Rotation Matrix R:")
print(R, "\n")
```

که این ماتریس بدین شکل است:

```
Rotation Matrix R:
[[ 0.37506554 -0.92574793 -0.04813127]
 [ 0.08420728  0.08573147 -0.99275337]
 [ 0.92316574  0.36829457  0.11010963]]
```

حال قصد داریم دورانی را بدست آوریم که دوربین را تراز و عمود بر زمین کند. R که تا الان بدست آوردیم دستگاه مختصات جهان را به دستگاه مختصات دوربین (که تراز و عمود نیست) می‌برد. بدست آوردن R2 به طوری که دستگاه مختصات جهان را به دستگاه مختصات دوربین ببرد (که تراز و عمود است) سخت نیست. درواقع دوران یک دستگاه مختصات است تا به طور دیگری قرار بگیرد. اگر دستگاه مختصات جهان را مثل همان شکل موجود در صورت سوالات در نظر بگیریم و دستگاه مختصات دوربین را هم به شکل زیر (مشابه آنچه که در جلسات ابتدایی دیدیم) درنظر بگیریم، ماتریس R2 که درواقع دستگاه مختصات جهان را به دستگاه مختصات دوربین (اگر تراز و صاف باشد) می‌برد به این شکل خواهد شد:

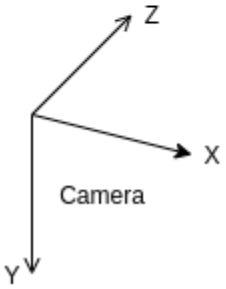
```
# World-to-Good-Camera Rotation Matrix
R2 = np.array([
    [0, -1, 0],
    [0, 0, -1],
    [1, 0, 0]
])
```



(مثلا محور x جهان، می‌شود محور Z دوربین، این ستون اول R2 را می‌سازد و به همین ترتیب)

حال ما دنبال ماتریس دورانی هستیم که دوربین ما را صاف کند. درواقع دنبال ماتریسی مثل R3 که در R ضرب شود و به ما R2 را بدهد(R2 ماتریسی است که تراز و عمود است)

پس می‌توان اینطور نوشت:



$$R2 = R3 * R \Rightarrow R2 * R^{-1} = R3$$

و بدین ترتیب ماتریس R3 را بدست می‌اوریم که چنین است:

```
R3: Y: -22.055264885089898 Z: 2.7587845052519064 X: 6.329000672777569
```

که نشان می‌دهد دوربین ما برای اینکه تراز و عمود شود، چه دوران‌هایی نیاز دارد. طبق خواسته سوال، برای اینکه دوربین تراز و عمود شود باید حول محور Z حدود ۲.۷۵ درجه و حول محور X حدود ۶.۳۲ درجه بچرخد.

```
# World-to-Camera Rotation Matrix
R = np.linalg.inv(K) @ np.concatenate((lambdas[0] * v[0], lambdas[1] * v[1], lambdas[2] * v[2]), axis=1)
print("Rotation Matrix R:")
print(R, "\n")

# World-to-Good-Camera Rotation Matrix
R2 = np.array([
    [0, -1, 0],
    [0, 0, -1],
    [1, 0, 0]
])

# Camera-to-Good-Camera Rotation Matrix
R3 = R2 @ np.linalg.inv(R)

R3 = Rotation.from_matrix(R3).as_euler('YZX', degrees=True)
print("R3: Y:", R3[0], "Z:", R3[1], "X:", R3[2])
```

بخش ۳:

ماتریس R3 که در قسمت قبل بدست آوردیم را می‌خواهیم روی تصویر اعمال کنیم. چون در سوال فقط تغییر حول محورهای Z و X خواسته شده است پس فقط Z و X را دوران می‌دهیم:

```
R = Rotation.from_euler('YZX', [0, 2.7587845052519064, 6.329000672777569], degrees=True).as_matrix()
```

هموگرافی خواسته شده را به راحتی می‌توان به شکل

$$H = K * R * K^{-1}$$

نوشت و بدست آورد:

```
H = K @ R @ np.linalg.inv(K)
print("Homography Matrix: ")
print(H, "\n")
```

که ماتریس هموگرافی ما بدین شکل می‌شود:

```
Homography Matrix:
[[ 9.98841019e-01 -2.06514395e-02  7.74758400e+01]
 [ 4.81312699e-02  1.00365916e+00 -1.55344745e+03]
 [-3.43228369e-23  8.72452647e-06  9.82999488e-01]]
```

درنهایت هم تصویر را مختصات چهارگوشه تصویر را بعد از warp بدست می‌آوریم و سپس تصویر را وارپ و ذخیره می‌کنیم:

```
min_x, min_y, max_x, max_y = get_projection_of_vertices(width, height, H)
max_x = int(max_x - min_x)
max_y = int(max_y - min_y)

T = np.array([
    [1, 0, -min_x],
    [0, 1, -min_y],
    [0, 0, 1]
])

warped_img = cv2.warpPerspective(src_image, T @ H, (max_x, max_y))
cv2.imwrite("out/res04.jpg", warped_img)

# with horizon line
cv2.line(src_image, (0, int(h[1])), (width, int(h[0] * width + h[1])), color=(0, 0, 255), thickness=3)
warped_img_horizon = cv2.warpPerspective(src_image, T @ H, (max_x, max_y))

cv2.imwrite("out/res04-horizon-line.jpg", warped_img_horizon)
```

اگر خط افق را هم رسم کنیم و همراه تصویر warp کنیم می‌بینیم که موازی محور x می‌شود. هم چنین تغییرات حول محور x هم با کمی دقت مشخص است: