تمرین سری ۲ - سوال ۱

محمدجواد شریعتی 96100414

لینک گوگل‌درایو: Google Drive

به منظور خوانایی بیشتر و تمیزی کد، گام های اصلی برنامه (تابع main) در فایل Q*.py قرار دارد و تابع هایی که پیاده سازی کردم در فایل utils.py قرار دارد. ابتدا تابع main و گام های اصلی برنامه را توضیح میدهم و درانتها در مورد هریک از تابع های پیاده سازی شده در فایل utils.py توضیح خواهم داد.

تابع main بدین صورت است:

```
def main():
    # Read video and save it frame by frame in a temporary folder
    # if not os.cut.exists('tmp'):
    #     os.mkdir('tmp')
    # save_frames('../resources/video.mp4', FRAME_NUMBERS, 'tmp')

    part1()
    part2()
    part3()
    part4()
    part5()
    part6()
    part7()
    part8()

    # Remove temporary folder
    # os.rmdir('tmp')
```

که در ابتدا یک پوشه tmp میسازد که در روند اجرا دیتاهایی که موقتی هستند در آن ریخته می‌شود. سپس هرکدام از بخش‌های این سوال یک تابع جدا دارد که صدا زده می‌شود. درنتیجه می‌توان هرکدام از بخش‌ها را جدا از بقیه اجرا کرد(با کامنت کردن بقیه). در انتها پوشه tmp می‌تواند حذف شود. من پوشه tmp را هم در drive قرار دارم تا درصورتی که خواستید کدها را اجرا کنید بتوانید سریع‌تر این کار را بکنید و هر بخش زمان کمتری بگیرد.

بخش ۱:

```python
def part1():
    frame270 = cv2.imread('tmp/frame270.jpg')
    frame450 = cv2.imread('tmp/frame450.jpg')

    # Find Homography matrix from 270 to 450
    H = find_homography_RANSAC(src=frame270, dst=frame450)
    H_inverse = np.linalg.inv(H)

    # Draw rectangle on frame450
    p1, p2 = (400, 300), (1200, 1000)
    rect450 = cv2.rectangle(copyof(frame450), p1, p2, (0, 0, 255), 2)
    cv2.imwrite('out/res01-450-rect.jpg', rect450)

    # Calculate projection of above rectangle and draw it on frame270
    p1 = get_point_projection(np.array([[400], [300], [1]]), H_inverse)
    p2 = get_point_projection(np.array([[1200], [300], [1]]), H_inverse)
    p3 = get_point_projection(np.array([[1200], [1000], [1]]), H_inverse)
    p4 = get_point_projection(np.array([[400], [1000], [1]]), H_inverse)

    rect270 = copyof(frame270)
    cv2.line(rect270, (int(p1[0]), int(p1[1])), (int(p2[0]), int(p2[1])), color=(0, 0, 255), thickness=2)
    cv2.line(rect270, (int(p2[0]), int(p2[1])), (int(p3[0]), int(p3[1])), color=(0, 0, 255), thickness=2)
    cv2.line(rect270, (int(p3[0]), int(p3[1])), (int(p4[0]), int(p4[1])), color=(0, 0, 255), thickness=2)
    cv2.line(rect270, (int(p4[0]), int(p4[1])), (int(p1[0]), int(p1[1])), color=(0, 0, 255), thickness=2)
    cv2.imwrite('out/res02-270-rect.jpg', rect270)

    src_warped, src_mask, ref_warped, ref_mask, panorama_shape = warp(reference=frame450, src=frame270)
    panorama = create_simple_panorama(src_warped, src_mask, ref_warped, ref_mask, panorama_shape)
    cv2.imwrite('out/res03-270-450-panorama.jpg', panorama)
```

در این بخش ابتدا با تابع find_homography_RANSAC که در utils پیاده سازی شده است تابع هموگرافی بین ۲۷۰ و ۴۵۰ را بدست می‌آورم. این تابع دقیقا همان کدی است که در تمرین قبلی زدم. که در ابتدا با استفاده از SIFT نقاط interest رو بدست می‌آورم و descriptor های آن‌ها را محاسبه می‌کنم. سپس با استفاده از BFMatcher نقاط corresponding بین دو عکس را بدست می‌آورم و سپس بین شبیه ترین و دومین شبیه ترین یک ratio test با ضریب 0.9 اعمال می‌کنم. در نهایت با match point های نهایی و با کمک RANSAC آماده هموگرافی بین دو عکس را بدست می‌آورم. (کد در صفحه بعد)

در ادامه یک مستطیل در فریم ۴۵۰ می‌کشم و projection راس‌های آن را تحت وارون H بدست آمده بدست می‌آورم. و با کمک این نقاط بدست آمده، مستطیل خواسته شده را در فریم ۲۷۰ می‌کشم. درنهایت با استفاده از تابع warp که خودم در utils.py پیاده سازی کردم فریم ۲۷۰ را به ۴۵۰ میبرم که در ادامه آن‌را توضیح خواهم داد. سپس یک تابع create_simple_panorama هست در utils.py که صرفا دو عکس را کنار هم می‌گذارد و یک پانوراما خروجی می‌دهد.

تابع find_homography_RANSAC برای پیدا کردن ماتریس هموگرافی بین دو عکس:

```python
def find_homography_RANSAC(src, dst):
    # RGB -> GRAY
    im1_gray = cv2.cvtColor(dst, cv2.COLOR_BGR2GRAY)
    im2_gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    # Create SIFT detector
    sift = cv2.SIFT_create()

    # Find interest points
    interest_points_1 = sift.detect(im1_gray, None)
    interest_points_2 = sift.detect(im2_gray, None)

    # Compute Descriptors
    _, descriptor1 = sift.compute(im1_gray, interest_points_1)
    _, descriptor2 = sift.compute(im2_gray, interest_points_2)

    # Use BFMatcher to match points between two images
    bf_matcher = cv2.BFMatcher()
    match_points = bf_matcher.knnMatch(descriptor1, descriptor2, k=2)

    # Ratio between most similar and second most similar should less than 0.9
    final_match_points = []
    for m, n in match_points:
        if m.distance / n.distance < 0.90:
            final_match_points.append(m)

    # Find Homography
    # Get Src(im2) and Dst(im1) match points for cv2.findHomography function
    src_points = []
    dst_points = []
    for match in final_match_points:
        src_points.append(interest_points_2[match.trainIdx].pt)
        dst_points.append(interest_points_1[match.queryIdx].pt)
    src_points = np.float32(src_points).reshape(-1, 1, 2)
    dst_points = np.float32(dst_points).reshape(-1, 1, 2)

    # Run RANSAC algorithm
    H, _ = cv2.findHomography(src_points, dst_points, cv2.RANSAC, 150.0)
    return H
```

تابع warp:

```python
def warp(reference, src):
    height, width = reference.shape[:2]

    # Find Homography matrix from src to ref
    H = find_homography_RANSAC(src=src, dst=reference)

    # Get minimum X and minimum Y of warped coordinates
    src_warped_min_x, src_warped_min_y, src_warped_max_x, src_warped_max_y = get_projection_of_vertices(width, height, homography=H)

    trans_x, trans_y = 0.0, 0.0
    if src_warped_min_x < 0:
        trans_x = -src_warped_min_x
    if src_warped_min_y < 0:
        trans_y = -src_warped_min_y
    T = np.array([
        [1, 0, trans_x],
        [0, 1, trans_y],
        [0, 0, 1]
    ])

    ref_warped_min_x, ref_warped_min_y, ref_warped_max_x, ref_warped_max_y = get_projection_of_vertices(width, height, T)

    # calculate width and height of panorama image
    panorama_width = int(max(src_warped_max_x + trans_x, ref_warped_max_x)) + 1
    panorama_height = int(max(src_warped_max_y + trans_y, ref_warped_max_y)) + 1
    panorama_shape = (panorama_height, panorama_width)

    # Warp src using T@H
    src_warped = cv2.warpPerspective(src, T @ H, (panorama_width, panorama_height))

    # Warp ref using T
    ref_warped = cv2.warpPerspective(reference, T, (panorama_width, panorama_height))

    src_mask = create_mask(src_warped)
    ref_mask = create_mask(ref_warped)

    return src_warped, src_mask, ref_warped, ref_mask, panorama_shape
```

این تابع که در بخش‌های دیگر هم مورد استفاده قرار می‌گیرد، در ابتدا با استفاده از find_homography_RANSAC توضیح داده شده در قبل ماتریس هموگرافی بین دو عکس را بدست می‌آورد.سپس حساب می‌کند که اگر src بخواهد به صفحه ref برود (source & reference) مختصات چهار گوشه آن چه خواهد شد. این کار برای این انجام می‌شود که اگر بخشی از عکس در ناحیه منفی قرار گرفت، با کمک یک هموگرافی T (ماتریس انتقال درواقع) نقطه شروع آن را 0,0 لحاظ کنیم. پس لازم است عکس ref هم طبق همین هموگرافی warp شود که مشاهده می‌شود مختصات نقاط آن سپس تحت T بدست آمده است. حال می‌توان اندازه عکس panorama حاصل را بدست آورد و درنهایت هم با کمک cv2.warpPerspective هردو عکس را warp می‌کنم. همچنین یک mask از روی آن‌ها با کمک create_mask میسازم که مشخص می‌کند هرکدام چه ناحیه از عکس panorama را به خود اختصاص خواهد داد.

```python
def create_mask(image):
    return np.vectorize(lambda x, y, z: 0 if x == 0 and y == 0 and z == 0 else 1)(image[:, :, 0], image[:, :, 1], image[:, :, 2])
```

تابع create_simple_panorama صرفا دو عکس را کنار هم میگذارد.

```python
def create_simple_panorama(src_warped, src_mask, ref_warped, ref_mask, panorama_shape):
    panorama = np.zeros((panorama_shape[0], panorama_shape[1], 3))

    for i in range(panorama_shape[1]):
        for j in range(panorama_shape[0]):
            try:
                if src_mask[j, i] == 1 and ref_mask[j, i] != 1:
                    panorama[j, i, :] = src_warped[j, i, :]
                else:
                    panorama[j, i, :] = ref_warped[j, i, :]
            except IndexError:
                pass  # left black
    return panorama
```

بخش ۲:

```
# H_270 = find_homography_RANSAC(src=frame270, dst=frame450)
# H_630 = find_homography_RANSAC(src=frame630, dst=frame450)
# H_90_270 = find_homography_RANSAC(src=frame90, dst=frame270)
# H_90 = H_90_270 @ H_270
# H_810_630 = find_homography_RANSAC(src=frame810, dst=frame630)
# H_810 = H_810_630 @ H_630
#
# write_matrix_to_file(H_90, 'frame90')
# write_matrix_to_file(H_270, 'frame270')
# write_matrix_to_file(H_630, 'frame630')
# write_matrix_to_file(H_810, 'frame810')

H_90 = read_matrix_from_file('tmp/homo/frame90')
H_270 = read_matrix_from_file('tmp/homo/frame270')
H_630 = read_matrix_from_file('tmp/homo/frame630')
H_810 = read_matrix_from_file('tmp/homo/frame810')
```

ابتدا هموگرافی‌ها را بدست می‌آورم و آن‌ها را داخل فایل save می‌کنم و در دفعات بعدی از فایل میخونم. برای اینکه زمان کمتری بگیره.

```
# Calculate width and height of panorama
min_x_90, min_y_90, max_x_90, max_y_90 = get_projection_of_vertices(width, height, homography=H_90)
min_x_270, min_y_270, max_x_270, max_y_270 = get_projection_of_vertices(width, height, homography=H_270)
min_x_630, min_y_630, max_x_630, max_y_630 = get_projection_of_vertices(width, height, homography=H_630)
min_x_810, min_y_810, max_x_810, max_y_810 = get_projection_of_vertices(width, height, homography=H_810)

min_x = min(min_x_90, min_x_270, min_x_630, min_x_810)
min_y = min(min_y_90, min_y_270, min_y_630, min_y_810)

panorama_width = int(max(max_x_90, max_x_270, max_x_630, max_x_810))
if min_x < 0:
    panorama_width += -int(min_x)
if panorama_width % 4 != 0:
    panorama_width += (4 - (panorama_width % 4))
panorama_height = int(max(max_y_90, max_y_270, max_y_630, max_y_810))
if min_y < 0:
    panorama_height += -int(min_y)
if panorama_height % 4 != 0:
    panorama_height = panorama_height + (4 - (panorama_height % 4))

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

سپس حساب می‌کنم که هرکدام از frame ها اگر warp شود چه ناحیه ای را اشغال می‌کند. درنتیجه یک کمترین و بیشترین x,y برای هرکدام بدست می‌آورم. کمترین کمترین x ها و کمترین کمترین y ها را بدست می‌آورم. برای اینکه مثل بخش قبل، اگر در ناحیه منفی قرار دارد با یک ماتریس انتقال همه را منتقل کنم. همچنین طول و عرض پانوراما نهایی را بدست می‌آورم. چون از هرم لاپلاسین استفاده کنم، خواستم حتما طول و عرض ضریب ۴ باشد برای راحتی بیشتر.(حداکثر ۳پیکسل اضافه می‌شود که مشکلی نخواهد داشت.)

```
warped_90 = cv2.warpPerspective(frame90, T @ H_90, (panorama_width, panorama_height))
warped_270 = cv2.warpPerspective(frame270, T @ H_270, (panorama_width, panorama_height))
warped_450 = cv2.warpPerspective(frame450, T, (panorama_width, panorama_height))
warped_630 = cv2.warpPerspective(frame630, T @ H_630, (panorama_width, panorama_height))
warped_810 = cv2.warpPerspective(frame810, T @ H_810, (panorama_width, panorama_height))

mask_90 = create_mask(warped_90)
mask_270 = create_mask(warped_270)
mask_450 = create_mask(warped_450)
mask_630 = create_mask(warped_630)
mask_810 = create_mask(warped_810)
```

در ادامه هرکدام از عکس‌ها را warp می‌کنم و mask آن‌ها را هم میسازم.

```
# Fill panorama
# Frame 90 and 270
panorama, start_overlap, end_overlap, best_cut = fill_panorama(warped_90, warped_270, mask_90, mask_270)
# Add Frame 450
warped_270 = update(panorama, warped_270, mask_90, mask_270)
panorama, start_overlap, end_overlap, best_cut = fill_panorama(warped_270, warped_450, mask_270, mask_450,
                                                               panorama, start_overlap, end_overlap, best_cut)
# Add Frame 630
warped_450 = update(panorama, warped_450, mask_270, mask_450)
panorama, start_overlap, end_overlap, best_cut = fill_panorama(warped_450, warped_630, mask_450, mask_630,
                                                               panorama, start_overlap, end_overlap, best_cut)
# Add Frame 810
warped_630 = update(panorama, warped_630, mask_450, mask_630)
panorama, start_overlap, end_overlap, best_cut = fill_panorama(warped_630, warped_810, mask_630, mask_810,
                                                               panorama, start_overlap, end_overlap, best_cut)
cv2.imwrite('out/res04-key-frames-panorama.jpg', panorama)
```

درنهایت عکس پانوراما را مرحله به مرحله پر می‌کنم. این کار را با کمک تابع fill_panorama که در utils.py پیاده سازی شده است انجام می‌دهم. ابتدا عکس 90 و 270 را ترکیب می‌کنم و آن‌ها را روی عکس نهایی قرار می‌دهم. سپس برای اضافه کردن فریم ۴۵۰، ابتدا فریم 270 را آپدیت می‌کنم. بدین معنا که نواحی‌ای را که فریم‌های 90 و 270 اشتراک دارند را از عکس پانوراما ساخته شده میگیرم. دلیل این‌کار این است که اشتراک بین 90 و 270 و 450 تهی نیست. در نتیجه ممکن است وقتی می‌خواهم 450 را اضافه کنم، برشی که بین 270 و 450 بدست می‌آورم، با برشی که بین 90 و 270 بدست آورده بودم برخورد کند. درنتیجه مشکل به وجود آید. اما با کمک تابع update ناحیه overlap بین 270 و 90 را آپدیت می‌کنم تا این مشکل پدید نیاید.

```python
def update(panorama, warped_pic, mask1, mask2):
    for i in range(warped_pic.shape[0]):
        for j in range(warped_pic.shape[1]):
            if mask1[i, j] == 1 and mask2[i, j] == 1:
                warped_pic[i, j, :] = panorama[i, j, :]
    return warped_pic
```

تابع fill_panorama بدین صورت است:

```python
def fill_panorama(src_warped, ref_warped, src_mask, ref_mask, panorama=None, prev_start_overlap=None, prev_end_overlap=None, prev_best_cut=None):
    panorama_height, panorama_width = ref_warped.shape[:2]

    gauss_filter = gaussian_filter(size=11, sigma=3.0)
    src_laplacian, ref_laplacian, src_subsamp, ref_subsamp = get_laplacian_pyramid(gauss_filter, src_warped, ref_warped)
```

ابتدا با یک تابع گوسی یک هرم لاپلاسین ۲ لول بدست می‌آورم(توضیح تابع get_laplacian_pyramid بعد از این تابع)

```python
start_overlap, end_overlap = find_overlap_area(src_mask, ref_mask)
subsamp_start_overlap, subsamp_end_overlap = start_overlap // 4, end_overlap // 4
subsamp_best_cut = find_best_cut(src_subsamp, ref_subsamp, subsamp_start_overlap, subsamp_end_overlap)
best_cut = []
for i in range(len(subsamp_best_cut)):
    best_cut.append(subsamp_best_cut[i] * 4)
```

سپس ابتدا و انتهای ناحیه overlap دو عکس را بدست می‌آورم (در راستای محور x)

و برای عکس کوچک شده بهترین برش را بدست می‌آورم. تابع find_best_cut تابعی است که در درس پردازش تصویر (نیمسال ۹۸-۹۹) پیاده سازی کردم تا با کمک dynamic programming بهترین برش بین دو تصویر را در ناحیه overlap آن دو بدست آورد.همچنین این‌برش باید روی laplacian عکس هم اعمال شود که ۴برابر بزرگ‌تر است. سپس تصاویر کوچک‌شده دو عکس را با هم ادغام می‌کنم:

```python
# Layer 2 of Laplacian Pyramid: Subsamp panorama
subsamp_panorama = np.zeros((panorama_height // 4, panorama_width // 4, 3))
subsamp_src_mask = cv2.resize(np.array(src_mask, dtype='float64'), None, fx=0.25, fy=0.25)
subsamp_ref_mask = cv2.resize(np.array(ref_mask, dtype='float64'), None, fx=0.25, fy=0.25)

# Fill subsamp_panorama (non-overlap area)
for i in range(panorama_width // 4):
    for j in range(panorama_height // 4):
        try:
            if subsamp_src_mask[j, i] == 1 and subsamp_ref_mask[j, i] != 1:
                subsamp_panorama[j, i, :] = src_subsamp[j, i, :]
            elif subsamp_ref_mask[j, i] == 1 and subsamp_src_mask[j, i] != 1:
                subsamp_panorama[j, i, :] = ref_subsamp[j, i, :]
        except IndexError:
            pass
# Fill subsamp_panorama (overlap area)
h, w = ref_subsamp.shape[0], subsamp_end_overlap - subsamp_start_overlap
for i in range(h):
    for j in range(w):
        try:
            if j <= subsamp_best_cut[h - i - 1]:
                subsamp_panorama[i, subsamp_start_overlap + j] = src_subsamp[i, subsamp_start_overlap + j, :]
            else:
                subsamp_panorama[i, subsamp_start_overlap + j] = ref_subsamp[i, subsamp_start_overlap + j, :]
        except IndexError:
            pass

blurred_panorama = cv2.resize(subsamp_panorama, None, fx=4, fy=4)
```

همین کار را برای laplacian دو عکس هم انجام می‌دهم:

```
# Layer 1 of Laplacian Pyramid: Laplacian
laplacian = np.zeros((panorama_height, panorama_width, 3))
# Fill laplacian (non-overlap area)
for i in range(blurred_panorama.shape[1]):
    for j in range(blurred_panorama.shape[0]):
        try:
            if src_mask[j, i] == 1 and ref_mask[j, i] != 1:
                laplacian[j, i, :] = src_laplacian[j, i, :]
            elif ref_mask[j, i] == 1 and src_mask[j, i] != 1:
                laplacian[j, i, :] = ref_laplacian[j, i, :]
        except IndexError:
            pass
# Fill laplacian (overlap area)
h, w = ref_warped.shape[0], (end_overlap - start_overlap)
for i in range(h):
    for j in range(w):
        try:
            if j <= best_cut[(h // 4) - (i // 4) - 1]:
                laplacian[i, start_overlap + j] = src_laplacian[i, start_overlap + j, :]
            # elif j == best_cut[(h // 4) - (i // 4) - 1]:
            #     panorama_diff[i, start_overlap + j] = [0, 0, 0]
            else:
                laplacian[i, start_overlap + j] = ref_laplacian[i, start_overlap + j, :]
        except IndexError:
            pass

# Build new panorama
new_panorama = blurred_panorama + laplacian
```

درنهایت پانوراما ساخته شده از جمع blurred_panorama و laplacian بدست می‌آید.

```
if panorama is not None:
    h, w = panorama_height, prev_end_overlap - prev_start_overlap
    for i in range(panorama_height):
        for j in range(panorama_width):
            try:
                if j > prev_best_cut[(h // 4) - (i // 4) - 1]:
                    panorama[i, prev_start_overlap + j] = new_panorama[i, prev_start_overlap + j, :]
            except IndexError:
                pass
else:
    panorama = new_panorama

return panorama, start_overlap, end_overlap, best_cut
```

اگر مرحله اول نباشد(۹۰و۲۷۰) panorama ساخته شده تا آن مرحله هم به تابع pass داده می‌شود. درنتیجه تصویر جدید بدست آمده را روی آن قرار می‌دهد (از برش قبلی به جلو- مثلا وقتی ۴۵۰ را می‌خواهد قرار دهد، از برش ۹۰-۲۷۰ به جلو را با تصویر بدست آمده پر می‌کند)

تابع get_laplacian_pyramid:

```
def get_laplacian_pyramid(gauss_filter, src_warped, ref_warped):
    src_warped = np.array(src_warped, dtype='float64')
    ref_warped = np.array(ref_warped, dtype='float64')

    height, width = ref_warped.shape[:2]
    src_blurred = np.zeros((height, width, 3))
    ref_blurred = np.zeros((height, width, 3))
    for layer in range(3):
        src_blurred[:, :, layer] = signal.convolve2d(src_warped[:, :, layer], gauss_filter, 'same')
        ref_blurred[:, :, layer] = signal.convolve2d(ref_warped[:, :, layer], gauss_filter, 'same')

    src_laplacian = src_warped - src_blurred
    ref_laplacian = ref_warped - ref_blurred

    src_subsamp = cv2.resize(src_blurred, None, fx=0.25, fy=0.25)
    ref_subsamp = cv2.resize(ref_blurred, None, fx=0.25, fy=0.25)

    return src_laplacian, ref_laplacian, src_subsamp, ref_subsamp
```

این تابع ابتدا عکس‌ها را با کمک فیلتر گوس داده شده blur می‌کند. سپس با کمک کردن blur شده عکس از خود عکس، تقریبی از laplacian آن بدست می‌آورد. درانتها هم آن‌ها را ¼ میکند. خروجی آن هم laplacian و عکس کوچک شده است. این تابع درواقع یک هرم ۲ لایه است.

پس به طور کلی روشی که برای این سوال من استفاده کردن این بود که یک هرم لاپلاسین دولایه زدم که ادغام آن‌ها را با کمک بهترین برش dynamic programming بدست آوردم. در نسخه کوچک شده عکس‌ها را با کمک dp ادغام کردم و سپس لاپلاسین آن‌ها را هم با همان برش ادغام کردم. باقی کار قرار دادن هرکدام از فریم‌های warp شده روی تصویر پانوراما بود.

بخش ۳:

```python
def part3():
    frame450 = cv2.imread('tmp/frame450.jpg')
    frame270 = cv2.imread('tmp/frame270.jpg')
    frame630 = cv2.imread('tmp/frame630.jpg')

    height, width = frame450.shape[:2]

    # H_270 = find_homography_RANSAC(src=frame270, dst=frame450)
    # H_630 = find_homography_RANSAC(src=frame630, dst=frame450)
    #
    # frame_num = 1
    # while frame_num <= FRAME_NUMBERS:
    #     frame = cv2.imread('tmp/frame' + str(frame_num) + '.jpg')
    #     if frame_num < 270:
    #         H = find_homography_RANSAC(src=frame, dst=frame270)
    #         H = H @ H_270
    #     elif frame_num > 630:
    #         H = find_homography_RANSAC(src=frame, dst=frame630)
    #         H = H @ H_630
    #     else:
    #         H = find_homography_RANSAC(src=frame, dst=frame450)
    #
    #     write_matrix_to_file(H, 'tmp/homo/frame' + str(frame_num))
    #     frame_num += 1
```

ابتدا هموگرافی تک‌تک فریم‌ها را بدست می‌آورم و در tmp ذخیره می‌کنم. از این به بعد هرزمان لازم شد از فایل می‌خونم به جای محاسبه.

سپس طول و عرض پانوراما حاصل را مثل قبل بدست می‌آورم. ابتدا کمترین و بیشتری x و y را بین همه فریم‌ها بدست می‌آورم و اگر منفی باشند کمترین x و y یک ماتریس انتقال T میسازم که با کمک آن شروع پانوراما به 0,0 منتقل شود. این T را هم در فایل ذخیره می‌کنم.

```
# Calculate width and height of panorama
minimum_x, minimum_y, maximum_x, maximum_y = 0, 0, 0, 0
frame_num = 1
while frame_num <= 900:
    H = read_matrix_from_file('tmp/homo/frame' + str(frame_num))
    min_x, min_y, max_x, max_y = get_projection_of_vertices(width, height, homography=H)
    if min_x < minimum_x:
        minimum_x = min_x
    if min_y < minimum_y:
        minimum_y = min_y
    if max_x > maximum_x:
        maximum_x = max_x
    if max_y > maximum_y:
        maximum_y = max_y
    frame_num += 1

panorama_width = int(maximum_x)
if minimum_x < 0:
    panorama_width += -int(minimum_x)
panorama_height = int(maximum_y)
if minimum_y < 0:
    panorama_height += -int(minimum_y)

T = np.array([
    [1, 0, -minimum_x],
    [0, 1, -minimum_y],
    [0, 0, 1]
])
write_matrix_to_file(T, 'tmp/homo/T')
```

سپس تکتک فریم‌ها را warp می‌کنم با T*H که هرکدام درجای خود قرار بگیرند. این عکس‌ها را در tmp هم ذخیره می‌کنم. درنهایت هم با این فریم‌ها ویدیو خواسته شده را میسازم. (می‌شد هم در ram نگاه داشت اگر مشکل ram نداریم و دوباره از فایل نخواند)

```
frame_num = 1
while frame_num <= 900:
    frame = cv2.imread('tmp/frame' + str(frame_num) + '.jpg')
    H = read_matrix_from_file('tmp/homo/frame' + str(frame_num))
    warped = cv2.warpPerspective(frame, T @ H, (panorama_width, panorama_height))
    cv2.imwrite('tmp/warp/frame' + str(frame_num) + '.jpg', warped)
    frame_num += 1

video = cv2.VideoWriter('out/res05-reference-plane.mp4', cv2.VideoWriter_fourcc(*'mp4v'), FPS, (panorama_width, panorama_height))
frame_num = 1
while frame_num <= FRAME_NUMBERS:
    warped_frame = cv2.imread('tmp/warp/frame' + str(frame_num) + '.jpg')
    video.write(warped_frame)
    frame_num += 1
video.release()
```

بخش ۴:

```python
def part4():
    frame_450 = cv2.imread('tmp/warp/frame450.jpg')
    height, width = frame_450.shape[:2]

    result = np.zeros((height, width, 3))

    PIECE_SIZE = 700  # almost 10GB memory
    for i in range(0, height, PIECE_SIZE):
        for j in range(0, width, PIECE_SIZE):
            print(i, j)
            frames_l0 = []
            frames_l1 = []
            frames_l2 = []
            frame_num = 1
            while frame_num <= FRAME_NUMBERS:
                # print(frame_num)
                warped_frame = cv2.imread('tmp/warp/frame' + str(frame_num) + '.jpg')
                frames_l0.append(crop(warped_frame, i, j, PIECE_SIZE, 0))
                frames_l1.append(crop(warped_frame, i, j, PIECE_SIZE, 1))
                frames_l2.append(crop(warped_frame, i, j, PIECE_SIZE, 2))
                frame_num += 1

            median_l0 = np.apply_along_axis(lambda v: np.median(v[np.nonzero(v)]), 0, frames_l0)
            median_l1 = np.apply_along_axis(lambda v: np.median(v[np.nonzero(v)]), 0, frames_l1)
            median_l2 = np.apply_along_axis(lambda v: np.median(v[np.nonzero(v)]), 0, frames_l2)

            result[i:i + PIECE_SIZE, j:j + PIECE_SIZE, 0] = median_l0
            result[i:i + PIECE_SIZE, j:j + PIECE_SIZE, 1] = median_l1
            result[i:i + PIECE_SIZE, j:j + PIECE_SIZE, 2] = median_l2

            cv2.imwrite('out/res06-background-panorama.jpg', result)
```

در این بخش اگر می‌خواستیم همه عکس‌ها را با هم در ram نگه داریم و روی آن‌ها عملیات انجام دهیم ram سیستم به کلی پر می‌شد. پس قطعه قطعه این کار را انجام دادم. مثلا یک مربع ۷۰۰ پیکسل در ۷۰۰ پیکسل درنظر میگیرم، فریم‌ها را لود می‌کنم و فقط همان مربع خواسته شده را در ram نگه می‌دارم. هرچقدر این عدد کمتر باشد، تعداد دفعاتی که تمامی ۹۰۰ فریم باید لود شوند بالاتر می‌روند و به نوعی برنامه IO-bound می‌شود. عدد ۷۰۰ برای من مناسب بود، تقریبا ۱۰ گیگ رم گرفت که مشکلی نبود و زمان اجرا هم ۲ ساعت بود. می‌توان PIECE_SIZE را بسته به ram تغییر داد. مثلا میتوان روی یک سرور با ram کافی اجرا کرد و اصلا تکه‌تکه نکرد و سریعتر انجام خواهد شد. زیرا خود خواندن ۹۰۰ فریم از دیسک حدود دو دقیقه طول می‌کشید.

درنتیجه عکس رو تکه تکه می‌کنم. برای هر تکه همه ۹۰۰ فریم را از دیسک می‌خوانم ولی فقط تکه مورد نظر را از هر فریم در ram ذخیره می‌کنم. سپس با استفاده از np.median ، میانه رنگ یک پیکسل را بدست می‌آورم(با کمک np.nonzero پیکسل‌های صفرمقدار را لحاظ نمی‌کنم).

بخش ۵:

```python
def part5():
    main_height, main_width = cv2.imread('tmp/frame1.jpg').shape[:2]
    background_panorama = cv2.imread('out/res06-background-panorama.jpg')

    T = read_matrix_from_file('tmp/homo/T')

    frames = []
    frame_num = 1
    while frame_num <= FRAME_NUMBERS:
        H_frame = read_matrix_from_file('tmp/homo/frame' + str(frame_num))
        H = T @ H_frame
        H_inv = np.linalg.inv(H)
        warped = cv2.warpPerspective(background_panorama, H_inv, (main_width, main_height))
        frames.append(warped)
        cv2.imwrite('tmp/background/frame' + str(frame_num) + '.jpg', warped)
        frame_num += 1

    video = cv2.VideoWriter('out/res07-background-video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), FPS, (main_width, main_height))
    for frame in frames:
        video.write(frame)
    video.release()
```

برای تک‌تک فریم‌ها، هموگرافی آن‌ها را که در بخش ۳ بدست آورده بودم و ذخیره کرده بودم، از دیسک می‌خوانم، T را هم از دیسک می‌خوانم (ماتریس انتقال برای اینکه تصویری در منفی‌ها نیفتد). معکوس T*H را بدست می‌آورم و تصویر background که در مرحله قبل بدست آورده بودم را با آن warp می‌کنم. تصویر warp شده هر فریم از نقطه 0,0 شروع می‌شود و به اندازه width و height ویدیو اصلی طول و عرض دارد. پس کافیست یک مستطیل به اندازه ویدیو اصلی از آن جدا کنیم تا تصویر خواسته شده برای هر فریم بدست آید. هرکدام از فریم‌ها را در tmp ذخیره می‌کنم و درنهایت هم ویدیو خواسته شده را میسازم.

بخش ۶:

در این قسمت برای محاسبه pixel های foreground از تابع is_foreground استفاده می‌کنم که اختلاف پیکسل‌های هرلایه را چک می‌کند و اگر مجموع توان دو اختلاف آن‌ها از یک threshold بیشتر بود، چنل قرمز آن را 255 می‌کند. این مقدار threshold را براساس آزمایش و خطا 7000 بدست آوردم. درنهایت ویدیو خواسته شده ساخته می‌شود.

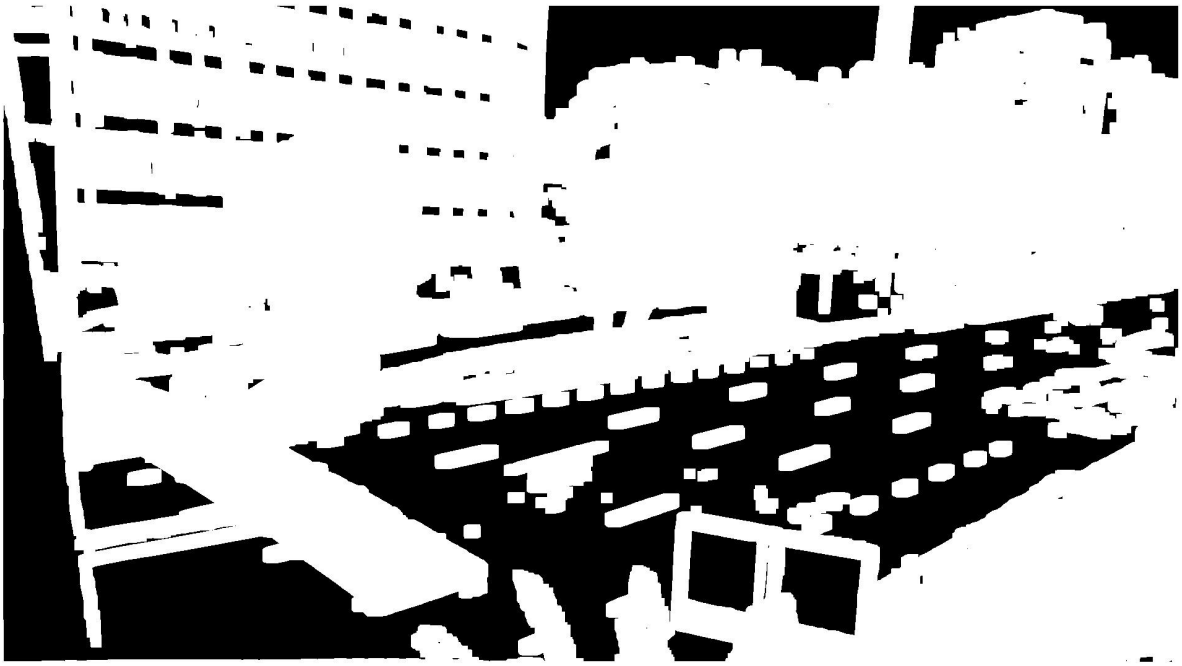
```
def part6():
    main_height, main_width = cv2.imread('tmp/frame1.jpg').shape[:2]
    THRESHOLD = 7000
    frames = []

    frame_num = 1
    while frame_num <= FRAME_NUMBERS:
        frame = np.array(cv2.imread('tmp/frame' + str(frame_num) + '.jpg'), dtype='float64')
        background_frame = cv2.imread('tmp/background/frame' + str(frame_num) + '.jpg')
        background_mask = get_thick_gradient_mask(background_frame)

        def is_foreground(frame_l0, frame_l1, frame_l2, background_frame_l0, background_frame_l1, background_frame_l2, background_mask):
            if background_mask != 255 and math.fabs(frame_l0 - background_frame_l0) ** 2 + math.fabs(frame_l1 - background_frame_l1) ** 2 + \
                    math.fabs(frame_l2 - background_frame_l2) ** 2 > THRESHOLD:
                return frame_l0, frame_l1, 255
            return frame_l0, frame_l1, frame_l2

        foreground_layers = np.vectorize(is_foreground)(frame[:, :, 0], frame[:, :, 1], frame[:, :, 2], background_frame[:, :, 0],
                                                        background_frame[:, :, 1], background_frame[:, :, 2], background_mask)
        foreground_frame = np.zeros(background_frame.shape)
        for layer in range(3):
            foreground_frame[:, :, layer] = foreground_layers[layer]

        foreground_frame = np.array(foreground_frame, dtype='uint8')
        frames.append(foreground_frame)
        cv2.imwrite('tmp/foreground/foreground' + str(frame_num) + '.jpg', foreground_frame)
        frame_num += 1

    video = cv2.VideoWriter('out/res08-foreground-video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), FPS, (main_width, main_height))
    for frame in frames:
        video.write(frame)
    video.release()
```

نویزهایی وجود داشت از جمله اینکه ساختمان‌ها و اجسام ثابت (background) هم جسم متحرک تشخیص داده می‌شدند. دلیل این اتفاق دقیق نبودن هموگرافی بود به طوری که مقداری اختلاف بین background panorama و فریم‌های اصلی وجود داشت. مثلا در اینجا می‌توان مشاهده کرد که مقدار زیادی از background به اشتباه foreground تشخیص داده شده است.

برای حل این مشکل، گرادیان background را محاسبه کردم و سپس از یک threshold عبور دادم و ماسک حاصل را ضخیم کردم:

```
def get_thick_gradient_mask(image):
    gradient = get_gradient(image, filter_size=7, sigma=1.3)
    gradient = np.vectorize(lambda x: 255 if x > 5 else 0)(get_distributed_image(gradient, 'uint8'))
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
    gradient = np.array(gradient, dtype='uint8')
    mask = cv2.dilate(gradient, kernel, iterations=4)
    return mask
```

درواقع جزئیات background را به صورت یک ماسک بدست می‌آورم و کمی ضخیم‌ترش می‌کنم(به خاطر همان عدم انطباق و همچنین اینکه ناحیه بیشتری از ساختمان‌ها را در بربگیرد) . روی این ماسک هم foreground تشخیص نمی‌دهم. برای مثال این ماسک برای فریم ۱ چنین می‌شود:



بخش ۷:

```python
def part7():
    WIDER_WIDTH = 3000  # initial width is 1920 -> increase 1080 pixels, 56%

    main_height, main_width = cv2.imread('tmp/frame1.jpg').shape[:2]
    background_panorama = cv2.imread('out/res06-background-panorama.jpg')
    panorama_height, panorama_width = background_panorama.shape[:2]

    T = read_matrix_from_file('tmp/homo/T')

    frames = []
    frame_num = 1
    while frame_num <= FRAME_NUMBERS:
        H_frame = read_matrix_from_file('tmp/homo/frame' + str(frame_num))
        H = T @ H_frame
        H_inv = np.linalg.inv(H)

        min_x, min_y, max_x, max_y = get_projection_of_vertices(panorama_width, panorama_height, homography=H_inv)
        if frame_num > FRAME_NUMBERS/2 and max_x < WIDER_WIDTH:
            break

        warped = cv2.warpPerspective(background_panorama, H_inv, (WIDER_WIDTH, main_height))
        frames.append(warped)
        cv2.imwrite('tmp/wide/frame' + str(frame_num) + '.jpg', warped)
        frame_num += 1

    video = cv2.VideoWriter('out/res09-background-video-wider.mp4', cv2.VideoWriter_fourcc(*'mp4v'), FPS, (WIDER_WIDTH, main_height))
    for frame in frames:
        video.write(frame)
    video.release()
```

ویدیو حاصل طولش از ۱۹۲۰ به ۳۰۰۰ افزایش یافته است، یعنی ۵۶ درصد افزایش.

برای wide کردن ویدیو، تصویر panorama بدست آمده در بخش۴ را با معکوس هموگرافی فریم‌ها warp می‌کنم. مشابه آنچه در بخش۵ انجام دادم. حال برای اینکه wide تر شود، به جای ۱۹۲۰ پیکسل، ۳۰۰۰ پیکسل را برمیدارم. درنتیجه از همان زاویه دید آن فریم تصویر wideتری بدست می‌آید. زمان ویدیو هم مطابق انتظار کاهش می‌یابد (از ۳۰ ثانیه به ۲۰ ثانیه)

بخش۸:

ایده این روش را از این مقاله گرفته ام.

روشی که من برای حذف کردن لرزش در این سوال استفاده کردم، به نوعی smooth کردن حرکت دوربین و دروان‌های انجام شده است.

جابه‌جایی‌ها و دوران‌های بین فریم‌ها را بدست می‌آورم و برای هرفریم باتوجه به فریم‌های اطرافش، یک جابه‌جایی و دوران جدید بدست می‌آورم که smooth تر است.

الگوریتم بدین صورت است که بین هردو فریم متوالی، جابه‌جایی و دوران را بدست می‌آورم. برای این‌کار، در یک فریم، یک سری interesting points بدست می‌آورم. مشابه آنچه در کلاس هم خواندیم، خوب هست که نقاط corner بدین منظور انتخاب شوند. برای این‌کار می‌توان از SIFT و ... هم استفاده کرد ولی من در اینجا از تابع آماده cv2.goodFeaturesToTrack استفاده می‌کنم. حال لازم است که متناظر این نقاط را در فریم بعدی هم بدست آوریم. برای این کار از تابع cv2.calcOpticalFlowPyrLK استفاده می‌کنم. این تابع یک valid_points هم برمی‌گرداند که به نوعی مشخص می‌کند توانسته متناظر نقطه داده شده را بدست بیاورد یا خیر. در ادامه تنها با نقاطی کار خواهیم کرد که valid باشد.

```
prev_frame = None
prev_frame_good_points = None
frame_num = 1
while frame_num <= FRAME_NUMBERS:
    frame = cv2.imread('tmp/frame' + str(frame_num) + '.jpg')
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    if prev_frame is None:  # frame 1
        prev_frame = gray_frame
        prev_frame_good_points = cv2.goodFeaturesToTrack(gray_frame, maxCorners=200, qualityLevel=0.01, minDistance=30, blockSize=3)
        frame_num += 1
        continue

    good_points, valid_points, _ = cv2.calcOpticalFlowPyrLK(prev_frame, gray_frame, prev_frame_good_points, None)
    # Choose only valid points
    good_points, prev_frame_good_points = get_valid_points(good_points, prev_frame_good_points, valid_points)
```

سپس با کمک تابع cv2.estimateAffinePartial2D رابطه بین این دو سری از نقاط را بدست می‌آوریم و این مقادیر را برای تمام نقاط ذخیره می‌کنم.

```
# Find transformation matrix between 2 frames and separate delta_x, delta_y and delta_angle
transform_matrix = cv2.estimateAffinePartial2D(prev_frame_good_points, good_points)[0]
delta_x = transform_matrix[0, 2]
delta_y = transform_matrix[1, 2]
delta_angle = np.arctan2(transform_matrix[1, 0], transform_matrix[0, 0])

x_transforms.append(delta_x)
y_transforms.append(delta_y)
angle_transforms.append(delta_angle)
```

حال بایستی با استفاده از این تغییرات بدست آمده، trajectory را محاسبه کنیم و سعی کنیم مسیر حرکت دوربین را به نوعی smooth کنیم.

برای این‌کار از cumulative sum استفاده می‌کنیم و روی تغییرات x و y و angle اعمال می‌کنیم:

```
# calculate trajectory
trajectory_x = np.cumsum(x_transforms)
trajectory_y = np.cumsum(y_transforms)
trajectory_angle = np.cumsum(angle_transforms)

# Smooth trajectory
smoothed_trajectory_x = smooth_trajectory(trajectory_x, window_size=SMOOTH_WINDOW)
smoothed_trajectory_y = smooth_trajectory(trajectory_y, window_size=SMOOTH_WINDOW)
smoothed_trajectory_angle = smooth_trajectory(trajectory_angle, window_size=SMOOTH_WINDOW)

# Smooth transformations
smooth_x_transforms = x_transforms + smoothed_trajectory_x - trajectory_x
smooth_y_transforms = y_transforms + smoothed_trajectory_y - trajectory_y
smooth_angle_transforms = angle_transforms + smoothed_trajectory_angle - trajectory_angle
```

پس با کمک تغییرات جابه‌جایی و دوران فریم‌های اطراف، جابه‌جایی و دوران هرفریم را smooth کرده ایم. درواقع برای اینکه ببینم یک فریم قرار است چقدر جابه‌جا شود و دوران پیدا کند، میانگین SMOOTH_WINDOW تا از جابه‌جایی و دوران فریم‌های اطرافش را برای آن درنظر میگیرم.

حال که مقدار smooth شده delta x و delta_ia و delta angle برای هر فریم را بدست آوردیم، یک ماتریس Affine میسازیم و در فریم‌های اصلی ضرب می‌کنیم:

```
frames = []
frame_num = 1
while frame_num < FRAME_NUMBERS:
    frame = cv2.imread('tmp/frame' + str(frame_num) + '.jpg')

    delta_x = smooth_x_transforms[frame_num - 1]
    delta_y = smooth_y_transforms[frame_num - 1]
    delta_angle = smooth_angle_transforms[frame_num - 1]

    affine_matrix = np.array([
        [np.cos(delta_angle), -np.sin(delta_angle), delta_x],
        [np.sin(delta_angle), np.cos(delta_angle), delta_y]
    ])

    warped = cv2.warpAffine(frame, affine_matrix, (width, height))[15:-15, 15:-15, :]
    frames.append(warped)

    frame_num += 1
```

برای اینکه بعضی از فریم‌ها جابه‌جا می‌شوند و/یا دوران پیدا می‌کنند، اطراف تصویر حاشیه های سیاه ظاهر می‌شود. برای اینکه اثر این موضوع کمتر شود، ۱۵ پیکسل از هر طرف عکس کم کردم.

درنهایت هم ویدیو خواسته شده را میسازم.