Xiang Chen
1004704992
November 20, 2020

CSC420 Assignment 3

I have used three grace token for this assignment.

Q1) (20 marks)

1.

We will use convolution on normalized Laplacian of Gaussian operator.

$$LoG_{normalized} = \sigma^2 \nabla G(x, y, \sigma) = \frac{1}{\pi\sigma^2}(\frac{x^2+y^2}{2\sigma^2} - 1)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We want to maximize the response of a black circle with white as background. Black is 0.0 and white is 1.0. We only need to consider the white part, since black is 0.0.

$I(x, y)$ is the image with 0.0 in the the centre black circle and 1.0 elsewhere.

Where V is the region out of circle with $r = \frac{D}{2}$ and $\theta \in [0,2\pi]$. No need to consider black.

$$\sigma^2 \iint_V \frac{1}{\pi\sigma^4}(\frac{r^2}{2\sigma^2} - 1)e^{-\frac{r^2}{2\sigma^2}} r * I(x, y)\, d\theta\, dr$$

$$\rightarrow \sigma^2 \frac{1}{\pi\sigma^4} \iint_V \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, d\theta\, dr$$

$$\rightarrow \sigma^2 * 2\pi * \frac{1}{\pi\sigma^4} \int_r^\infty \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, dr$$

$$\rightarrow \sigma^2 * \frac{2}{\sigma^4} \int_r^\infty \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, dr$$

$$\rightarrow \sigma^2 * (\frac{2}{\sigma^4} \int_r^\infty \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}}\, dr - \frac{2}{\sigma^4} \int_r^\infty e^{-\frac{r^2}{2\sigma^2}} * r\, dr)$$

Let $\frac{1}{2\sigma^2} = a$, we can simplify to the following.

$$\rightarrow \sigma^2 * (8a^2 \int_r^\infty a * r^3 * e^{-a*r^2}\, dr - 8a^2 \int_r^\infty e^{-r^2*a} * r\, dr)$$

$$\rightarrow \sigma^2 * (8a^2 \int_r^\infty a * r^3 * e^{-a*r^2}\, dr + 8a^2 \frac{e^{-ar^2}}{2a} |_{r=\frac{D}{2}}^{r=\infty})$$

$$\rightarrow \sigma^2 * (8a^2 \frac{(ar^2+1)e^{-ar^2}}{2a}|_{r=\frac{D}{2}}^{r=\infty} + 8a^2 \frac{e^{-ar^2}}{2a}|_{r=\frac{D}{2}}^{r=\infty})$$

$$\rightarrow \sigma^2 * (8a^2 \frac{(ar^2+1)e^{-ar^2}}{2a}|_{r=\frac{D}{2}}^{r=\infty} + 8a^2 \frac{e^{-ar^2}}{2a}|_{r=\frac{D}{2}}^{r=\infty})$$

$$\rightarrow \sigma^2 * (-8a^2 \frac{(a\frac{D^2}{4}+1)e^{-a\frac{D^2}{4}}}{2a} - 8a^2 \frac{e^{-a\frac{D^2}{4}}}{2a})$$

$$\rightarrow \sigma^2 * (-4a(a\frac{D^2}{4}+1)e^{-a\frac{D^2}{4}} - 4a\,e^{-a\frac{D^2}{4}})$$

$$\rightarrow -4a * \sigma^2 * ((a\frac{D^2}{4}+1)e^{-a\frac{D^2}{4}} - e^{-a\frac{D^2}{4}})$$

$$\rightarrow -4a * \sigma^2 * (a\frac{D^2}{4}e^{-a\frac{D^2}{4}})$$

$$\rightarrow -4a^2 * \sigma^2 * (\frac{D^2}{4}e^{-a\frac{D^2}{4}})$$

$$\rightarrow -4(\frac{1}{4\sigma^2}\frac{D^2}{4}e^{-\frac{D^2}{8\sigma^2}})$$

$$\rightarrow -\frac{D^2}{4\sigma^2}e^{-\frac{D^2}{8\sigma^2}} \quad (1.1)$$

We take the derivate of (1.1), we can get the following formula.

$$\frac{(8d^2\sigma^2 - d^4)e^{-\frac{d^2}{8\sigma^2}}}{16\sigma^5} \quad (1.2)$$

We want to maximize the above formula, we will set it equal to zero.

It is obviously when $8d^2\sigma^2 - d^4$ is zero, (1.2) is zero.

Therefore, $\sigma = \frac{\sqrt{2}}{4}D$

2.

We will use convolution on normalized Laplacian of Gaussian operator.

$$LoG_{normalized} = \sigma^2 \nabla G(x, y, \sigma) = \frac{1}{\pi\sigma^2}(\frac{x^2+y^2}{2\sigma^2} - 1)e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We want to maximize the response of a white circle with black as background. Black is 0.0 and white is 1.0. We only need to consider the white part, since black is 0.0. In this situation, we

want to minimize the result of convolution, by using the same way above we can get the following.

$I(x, y)$ is the image with 1.0 in the the centre white circle, 0.0 elsewhere.

Where V is the region of circle with $r = \frac{D}{2}$ and $\theta \in [0,2\pi]$.

$$\sigma^2 \iint_V \frac{1}{\pi\sigma^4}\left(\frac{r^2}{2\sigma^2} - 1\right)e^{-\frac{r^2}{2\sigma^2}} r * I(x, y)\, d\theta\, dr$$

$$\rightarrow \sigma^2 \frac{1}{\pi\sigma^4} \iint_V \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, d\theta\, dr$$

$$\rightarrow \sigma^2 * 2\pi * \frac{1}{\pi\sigma^4} \int_0^{\frac{D}{2}} \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, dr$$

$$\rightarrow \sigma^2 * \frac{2}{\sigma^4} \int_0^{\frac{D}{2}} \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}} - e^{-\frac{r^2}{2\sigma^2}} * r\, dr$$

$$\rightarrow \sigma^2 * \left(\frac{2}{\sigma^4} \int_0^{\frac{D}{2}} \frac{r^3}{2\sigma^2} * e^{-\frac{r^2}{2\sigma^2}}\, dr - \frac{2}{\sigma^4} \int_0^{\frac{D}{2}} e^{-\frac{r^2}{2\sigma^2}} * r\, dr\right)$$

The rest calculation is almost the save above.

We can get $\sigma^2 * (8a^2 \frac{(ar^2 + 1)e^{-ar^2}}{2a}|_{r=0}^{r=\frac{D}{2}} + 8a^2 \frac{e^{-ar^2}}{2a}|_{r=0}^{r=\frac{D}{2}})$

We want to find the maximum for

$$\sigma^2 * (8a^2 \frac{(ar^2 + 1)e^{-ar^2}}{2a}|_{r=0}^{r=\frac{D}{2}} + 8a^2 \frac{e^{-ar^2}}{2a}|_{r=0}^{r=\frac{D}{2}}) \quad (2.1)$$

Since we want to find the maximum, we will take differentiate of (2.1), which should only a sign difference with (1.2) and set it to zero.

Then, we can expect we get the same answer with (1.2) $\sigma = \frac{\sqrt{2}}{4}D$.

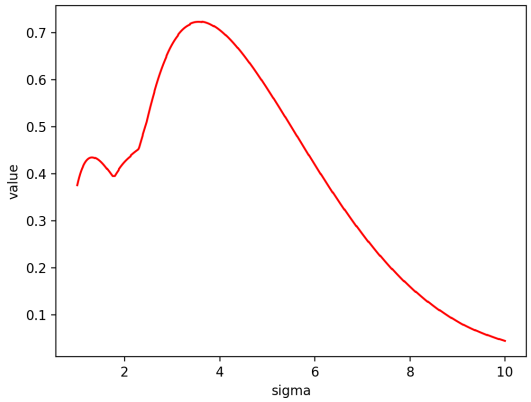
3.

Black square with white background

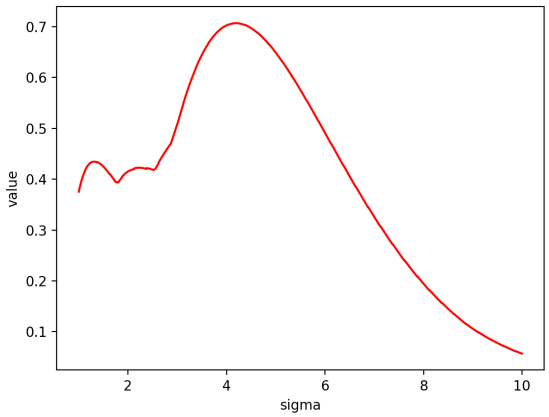
```
def find_max(sigma):
    img = np.ones((20, 20, 1), dtype = np.float64)
    cv.rectangle(img, (6, 6), (14, 14), (0.0), -1)
    LoG = (sigma ** 2) * ndimage.gaussian_laplace(img, sigma = sigma)
    return np.amax(LoG)
```

Here are graphs.

s=8.



s=10.

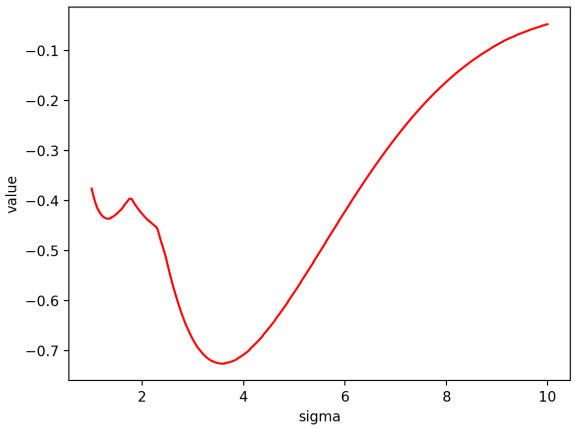


White square with black background.

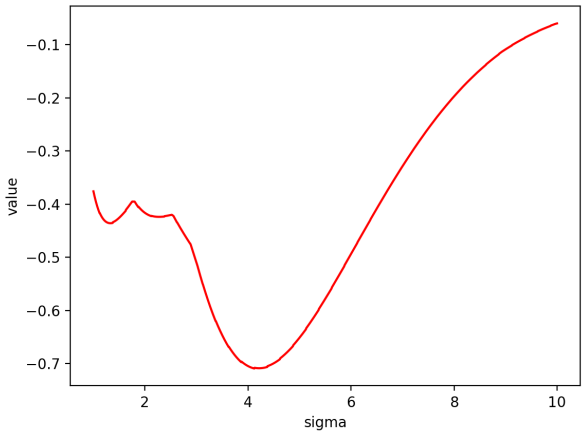
```
def find_maxmag(sigma):
    img = np.zeros((20, 20, 1), dtype = np.float64)
    cv.rectangle(img, (5, 5), (10, 10), (1.0), -1)
    LoG = (sigma ** 2) * ndimage.gaussian_laplace(img, sigma = sigma)
    return np.amin(LoG)
```

Here are graphs.

s=8.



s=10.



From four sample graphs, we can see when $\sigma = 0.42s$, approximately we can get the best response, and I have also tried 20 more times to verify this value.

Q2) (20 marks)

1.

In the class, we have got the formula $S = \frac{log(1-P)}{log(1-p^k)}$. In this question, we have $p = 0.5$ and $P = 0.995$, $k = 3$.

We put the value in to the equation. We can get the following.

$$S = \frac{log(1-0.995)}{log(1-0.5^3)} = 39.67844012 \rightarrow 40$$

Therefore, we will have the minimum trail for 40 to guarantee we have 99.5% chance.

Here is the code for my implementation of RANSAC.

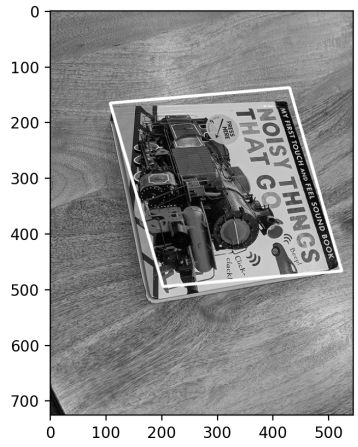
```
def match(path_1, path_2):
    bf = cv2.BFMatcher()
    img1 = cv2.imread(path_1, cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread(path_2, cv2.IMREAD_GRAYSCALE)
    sift = cv2.xfeatures2d.SIFT_create()
    kp1_SIFT, desc1_SIFT = sift.detectAndCompute(img1, None)
    kp2_SIFT, desc2_SIFT = sift.detectAndCompute(img2, None)
    img1_SIFT = cv2.drawKeypoints(img1, kp1_SIFT, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,255,0))
    img2_SIFT = cv2.drawKeypoints(img2, kp1_SIFT, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,255,0))
    kp1 = kp1_SIFT
    kp2 = kp2_SIFT
    desc1 = desc1_SIFT
    desc2 = desc2_SIFT
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(desc1, desc2, k=2)
    good_matches = []
    good_matches_without_list = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append([m])
            good_matches_without_list.append(m)
    precount = 0
    iters = 40
    best_A = None
    P = 0.995
    for i in range(iters):
        random_list = []
        while len(random_list)!=3:
            i = random.randrange(0, len(good_matches))
            if i not in random_list: random_list.append(i)
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches_without_list]).reshape(-1,2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches_without_list]).reshape(-1,2)
# Get three random points from src
        pts1 = np.float32([[src_pts[random_list[0]][0], src_pts[random_list[0]][1]], [src_pts[random_list[1]][0],
            src_pts[random_list[1]][1]], [src_pts[random_list[2]][0], src_pts[random_list[2]][1]]])
# Get three random points from dst
        pts2 = np.float32([[dst_pts[random_list[0]][0], dst_pts[random_list[0]][1]], [dst_pts[random_list[1]][0],
            dst_pts[random_list[1]][1]], [dst_pts[random_list[2]][0], dst_pts[random_list[2]][1]]])
# Compute AffineTransform from src to dst
        A = cv2.getAffineTransform(pts1, pts2)
         threshold = 5
         count = 0
         for j in range(m):
             buff_pt = np.zeros((3,),dtype="float")
             buff_pt[0] = src_pts[j][0]
             buff_pt[1] = src_pts[j][1]
             buff_pt[2] = 1.0
             buff_pt.reshape((3,1))
             new_pt = A.dot(buff_pt.T)
             if (abs(new_pt[0]-dst_pts[j][0]) + abs(new_pt[1]-dst_pts[j][1])) < 5: count += 1
         if count > precount:
             precount = count
             best_A = A
```

Here is the affine matrix.

[[6.73471422e-02, -2.29623825e-01, 4.29007760e+02]

[2.34542934e-01, 1.96092079e-02, 1.38992627e+02]]

Here is the output of affine matrix.



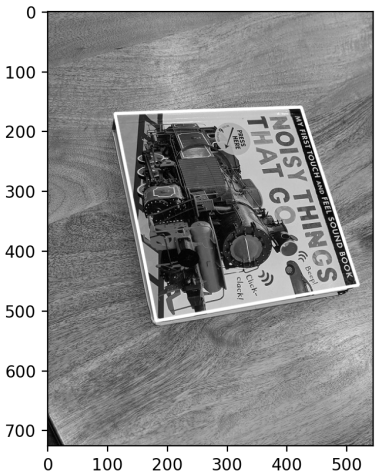
Here is the homogenous matrix.

[[2.95911512e-02 -2.32316776e-01 4.23150679e+02]

[1.77125407e-01 -7.73040189e-03 1.60741585e+02]

[-7.57760118e-05 -8.86119309e-05 1.00000000e+00]]

Here is the output of homogenous matrix.



We can compare the results from two different matrixes and we can see that the image produced by homogenous matrix is more precise than that produced by affine matrix. Although the difference is not so obvious, affine matrix also has done a good job and it also has the advantage for its efficiency since it is 2*3 instead of 3*3.

2.

In this question, we need to dynamically update our iterations when we have a higher percentage of matches than the estimate.

It is the same with the formula we use last question to get our estimation for max iteration.

$1 - p^k$ is probability that we have at least one point is outlier. Assume the new matches we get is $(1 - p^k)^S$ is the probability that S iterations we have at least one point is outlier. The probability to get the correct model is $P = 1 - (1 - p^k)^s$. After simplification, we can get $S = \frac{log(1-P)}{log(1-p^k)}$, where p is the new percentage.

For my implementation, I find we can not have over half matches, which is different from what the question says that at least half of them is matched.

I have try a larger number of iterations and I find the final percentage is around 0.31.

If I use estimation with 0.5, it will never change, which is 40.

I will use 0.25 as my original estimation and update through my new function.

When we have 0.25, we will have a original estimation for

$$S = \frac{log(1-P)}{log(1-p^k)} = \frac{log(1-0.995)}{log(1-0.25^3)} = 190.225 = 191$$

If I use 0.25, the answer is varying. I initialize the original iteration with 999999999.9. Some of them is smaller and some of them is larger than the original estimate, and the majority of the new iteration is smaller than the estimate.

Here are some numerical results of my dynamic version of RANSAC.

new iters: 140

new est: 0.3333333333333333

itertations: 141

new iters: 178

new est: 0.3080357142857143

new iters: 168

new est: 0.31398809523809523

new iters: 136

new est: 0.33630952380952384

itertations: 137

new iters: 268

new est: 0.2693452380952381

new iters: 210

new est: 0.2916666666666667

new iters: 146

new est: 0.3288690476190476

new iters: 131

new est: 0.34077380952380953

itertations: 132

new iters: 220

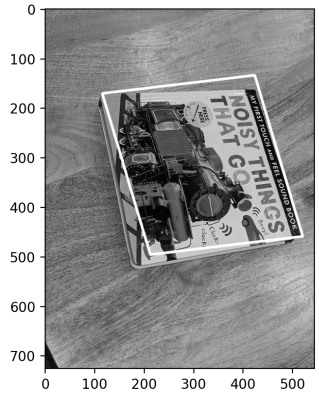
new est: 0.28720238095238093

new iters: 214

new est: 0.29017857142857145

itertations: 215

Here is one image generated by dynamic version of RANSAC.



Here is the code.

```
def dynamtic_map(path_1, path_2, estimate):
    bf = cv2.BFMatcher()
    img1 = cv2.imread(path_1, cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread(path_2, cv2.IMREAD_GRAYSCALE)
    sift = cv2.xfeatures2d.SIFT_create()
    kp1_SIFT, desc1_SIFT = sift.detectAndCompute(img1, None)
    kp2_SIFT, desc2_SIFT = sift.detectAndCompute(img2, None)
    img1_SIFT = cv2.drawKeypoints(img1, kp1_SIFT, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,255,0))
    img2_SIFT = cv2.drawKeypoints(img2, kp1_SIFT, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS, color=(255,255,0))
    kp1 = kp1_SIFT
    kp2 = kp2_SIFT
    desc1 = desc1_SIFT
    desc2 = desc2_SIFT
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(desc1, desc2, k=2)
    good_matches = []
    good_matches_without_list = []
    for m,n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append([m])
            good_matches_without_list.append(m)
    precount = 0
    iters = 999999999
    P = 0.995
    a = 0
    for i in range(iters):
        a += 1
        random_list = []
        while len(random_list)!=3:
            i = random.randrange(0, len(good_matches))
            if i not in random_list: random_list.append(i)
        src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches_without_list]).reshape(-1,2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches_without_list]).reshape(-1,2)
# Get three random points from src
        pts1 = np.float32([[src_pts[random_list[0]][0], src_pts[random_list[0]][1]], [src_pts[random_list[1]][0],
            src_pts[random_list[1]][1]], [src_pts[random_list[2]][0], src_pts[random_list[2]][1]]])
# Get three random points from dst
        pts2 = np.float32([[dst_pts[random_list[0]][0], dst_pts[random_list[0]][1]], [dst_pts[random_list[1]][0],
            dst_pts[random_list[1]][1]], [dst_pts[random_list[2]][0], dst_pts[random_list[2]][1]]])
# Compute AffineTransform from src to dst
        A = cv2.getAffineTransform(pts1, pts2)
        m, n = src_pts.shape
        threshold = 5
        count = 0
        for j in range(m):
            buff_pt = np.array([src_pts[j][0], src_pts[j][1], 1.0])
            new_pt = A.dot(buff_pt.T)
            if (abs(new_pt[0]-dst_pts[j][0]) + abs(new_pt[1]-dst_pts[j][1])) < 5: count += 1
        if count > precount:
            precount = count
            best_A = A
            if (count / m) > estimate:
                iters = int(math.log(1 - P) / math.log(1 - pow(count / m, 3)))
                estimate = (count / m)
                print("new iters:",iters)
                print("new est:", estimate)
        if a > iters: break
    print("itertations:", a)
    h,w = img1.shape
    new_pt_1 = best_A.dot(np.array([0,0,1]).T)
    new_pt_2 = best_A.dot(np.array([w-1,0,1]).T)
    new_pt_3 = best_A.dot(np.array([0,h-1,1]).T)
    new_pt_4 = best_A.dot(np.array([w-1,h-1,1]).T)
    detected_book = np.array([new_pt_1,new_pt_3,new_pt_4,new_pt_2])
    img3 = cv2.polylines(img2,[np.int32(detected_book)], True, 255, 3, cv2.LINE_AA)
    plt.imshow(img3, 'gray')
    plt.show()
    img4 = cv2.drawMatchesKnn(img1,kp1,img3,kp2,good_matches,None,flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
        matchColor=(0,255,0))
    plt.imshow(img4)
    plt.show()
```

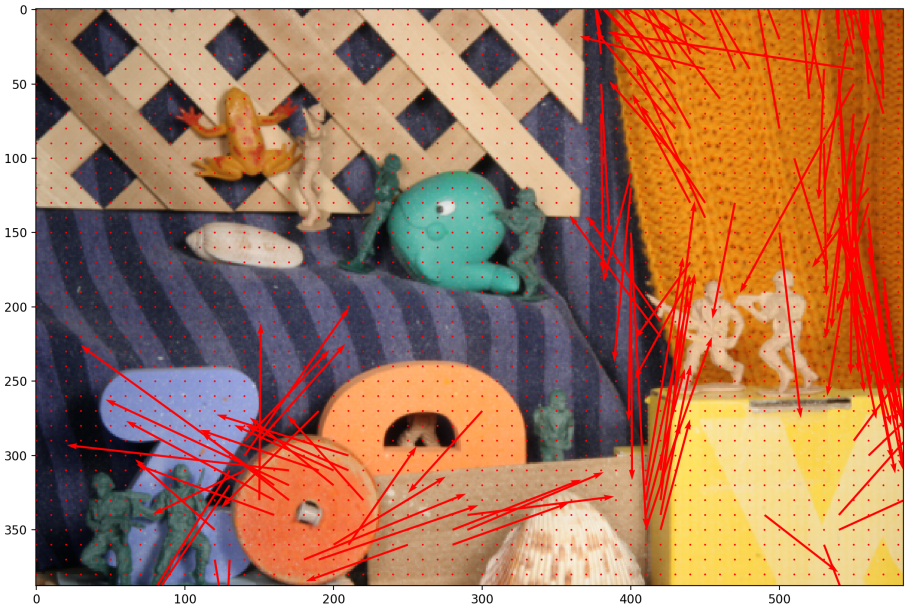
Q3) (50 marks)

I use 1-d Gaussian to blur along the t-axis, when I am trying to compute the window of I_t. I firstly flatten window for $I(t)$ and the window for $I(t+1)$. Then, put them into stack to have a 2-d array with shape (2, n). Then, I apply 1-d Gaussian along the 0-axis to decrease the noise.

Here is my code for optical flow.

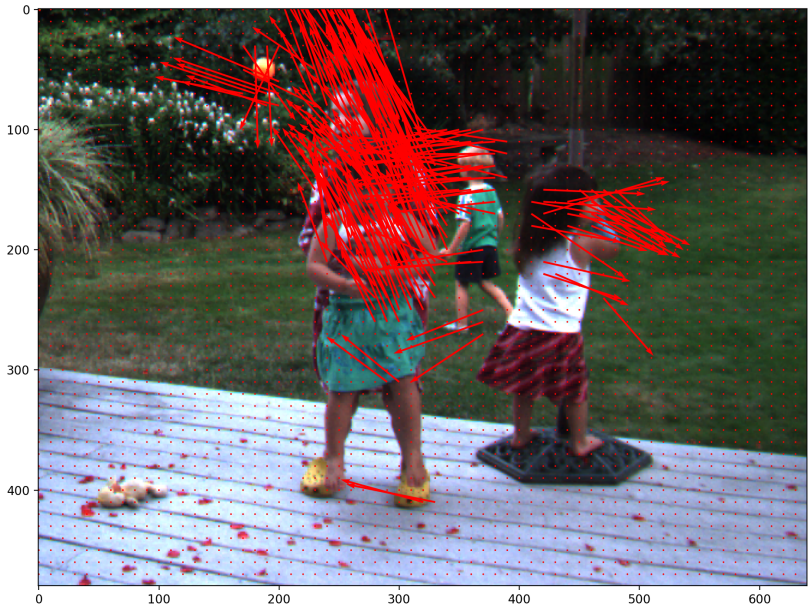
```python
def optical_flow(kernel_size, src_1, src_2):
# Get height and width
    height, width = src_1.shape
# Get blur matrix
    blur_matrix = Gaussian_Blur(9/6, 9)
# Decrease the noise
    src_1 = cv.filter2D(src_1, -1, blur_matrix)
    src_2 = cv.filter2D(src_2, -1, blur_matrix)
# Get sobel operator.
    sobel_x = np.array([[-1,0,1], [-2,0,2], [-1,0,1]])
    sobel_y = np.array([[-1,-2,-1], [0,0,0], [1,2,1]])
# Get gradient X
    I_x = cv.filter2D(src_1, -1, sobel_x)
# Get gradient Y
    I_y = cv.filter2D(src_1, -1, sobel_y)
# Save vector U
    U = np.zeros((height, width), dtype="float")
# Save vector V
    V = np.zeros((height, width), dtype="float")
# Calculate mid of window
    mid =  kernel_size // 2
    for i in range(height):
        for j in range(width):
# Set the original i_buf = i
            i_buf = i
# Set the original j_buf = j
            j_buf = j
# Set maximum loop number
            max_loop = 6
# Calculate left bound of x for the window
            left_height_bound = i - mid
# Calculate right bound of x for the window
            right_height_bound = i + mid + 1
# Calculate left bound of y for the window
            left_width_bound = j - mid
# Calculate right bound of y for the window
            right_width_bound = j + mid + 1
# Cut boundary cases
            if left_height_bound < 0 or left_width_bound < 0 or right_height_bound >= height or right_width_bound >= width: continue
# Set window for gradient x
            window_Ix = I_x[left_height_bound:right_height_bound, left_width_bound:right_width_bound]
# Set window for gradient y
            window_Iy = I_y[left_height_bound:right_height_bound, left_width_bound:right_width_bound]
# Set window for I(t)
            window_I = src_1[left_height_bound:right_height_bound, left_width_bound:right_width_bound]
# Construct Matrix A
            A = np.vstack((window_Ix.flatten(),window_Iy.flatten()))
# Get the transcope of A
            A_T = A.T
# Calculate the inverse of A
            inv_A = None
# Handele exception
            try: inv_A = np.linalg.pinv(A_T)
            except: continue
# Loop until small displacement
            while 1:
# Boundary for I(t+1)
                left_height_bound_t = i_buf - mid
                right_height_bound_t = i_buf + mid + 1
                left_width_bound_t = j_buf - mid
                right_width_bound_t = j_buf + mid + 1
# Cut boundary cases
                if left_height_bound_t < 0 or left_width_bound_t < 0 or right_height_bound_t >= height or right_width_bound_t >= width: break
# Concadinate window_I(t) and window_I(t+1)
                con_src = np.vstack((window_I.flatten(), src_2[left_height_bound_t:right_height_bound_t, left_width_bound_t:right_width_bound_t].flatten()))
# t-axis blur
                con_src = gaussian_filter1d(con_src, sigma = 0.45, axis = 0)
# Construct Matrix b
                window_It = con_src[1] - con_src[0]
# Get the res
                res = np.dot(inv_A, -window_It.T)
# Get u, v displacement
                u, v = res[0], res[1]
# Break when displacement is small or exceed max loop
                if ((abs(u) + abs(v)) < 0.5) or not max_loop: break
# Update j_buf and i_buf
                j_buf += int(u)
                i_buf += int(v)
# Add displacement to U and V
                U[i][j] += u
                V[i][j] += v
# Update upper bound of max_loop
                max_loop -= 1
    return U, V
```

Army (frame 7-8):



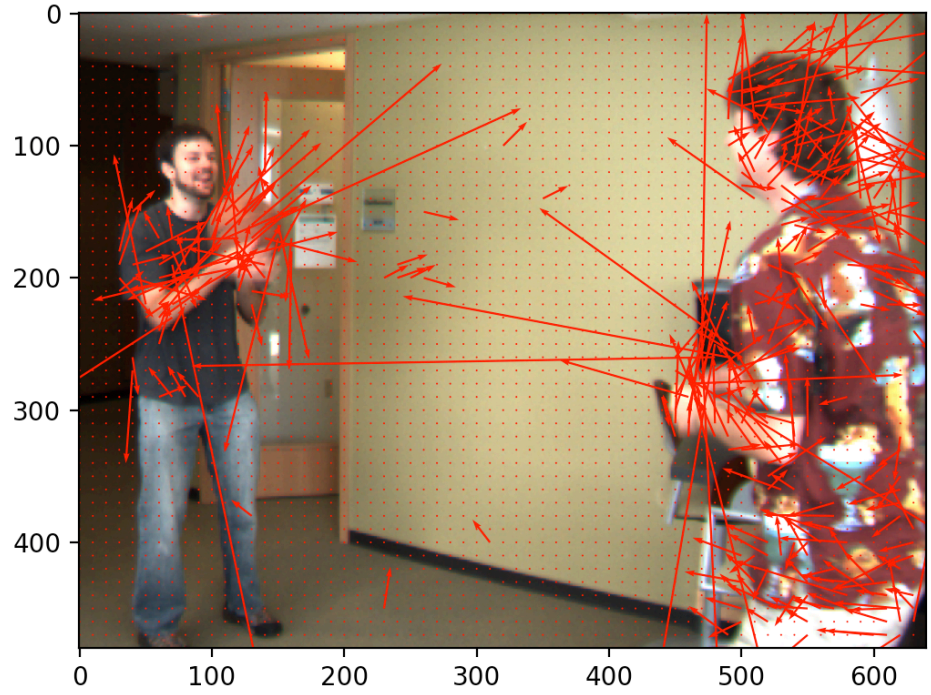
My algorithm actually has a good performance on this image. We can see the round object in the left bottom. It is really clear that it has a rotation trending and our algorithm detects it and put a lot of arrows around it. The direction of these arrows also make sense for rotation.We can notice this algorithm actually cares more about some regions which are high texture. This is the same with what we have learnt in the class. Therefore, my algorithm actually has a good performance on this image.

Backyard (frame 7-8):



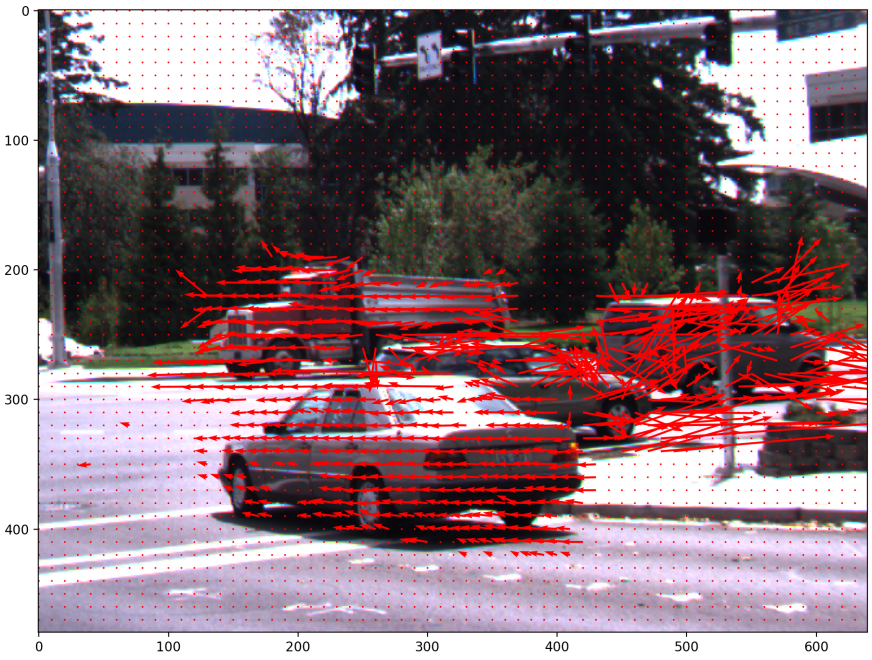
My algorithm actually has a good performance on this image. We can expect the little girl has a trending of going up and the rightest girl has a trending jumping up. The boy are moving forward. Therefore, the main arrows are around the two girls and the boy and arrows also have a good performance on the direction. Therefore, my algorithm actually has a good performance on this image.

Basketball (frame 7-8):



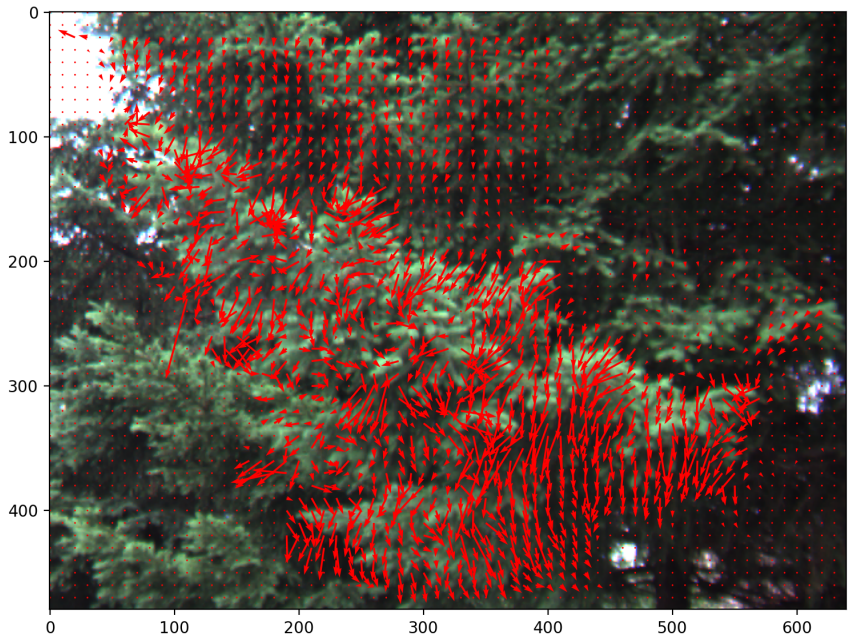
My algorithm actually has a good performance on this image. We can expect this basketball is going up and towards right, which we have arrows are satisfied with it. The man has a trending to throw the ball up, which we have some arrows around the man. Another man wants to use his arm to catch the ball and his body should move up. The arrow on the wall actually display the movement of the shadow of the ball. Therefore, my algorithm actually has a good performance on this image.

Dumptruck (frame 7-8):



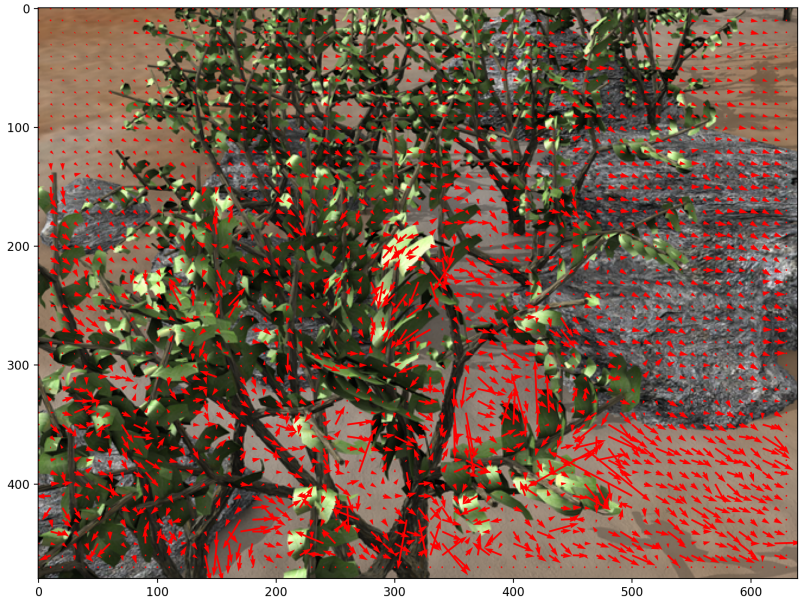
My algorithm actually has good performance on this image. We can expect the cars in the right side is moving to right and we have many big arrows which point to the right. The car in the centre has the trending of going forward, and we have arrow set around it. The arrows on the big truck also make really good sense. Therefore, my algorithm actually has a good performance on this image.

Evergreen (frame 7-8):



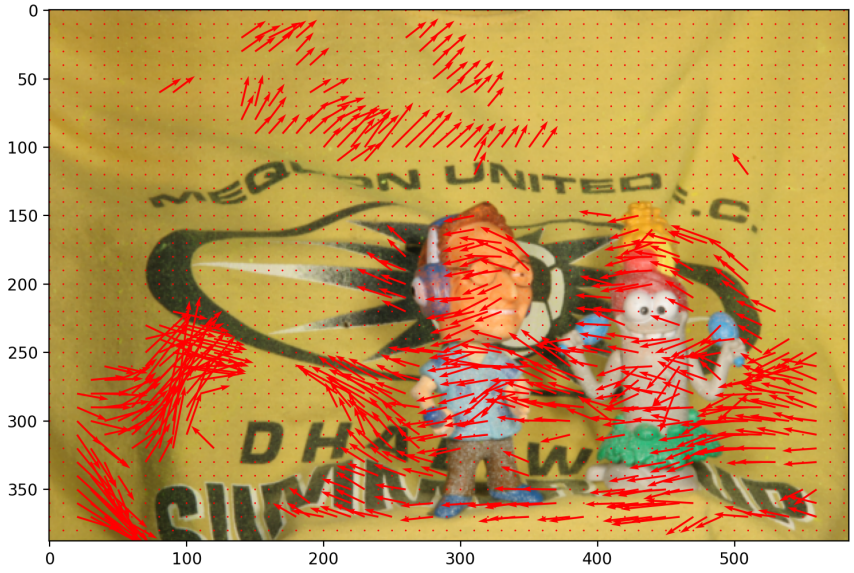
My algorithm actually has a good performance on this image. We can see we have extremely clear motion on this image. All leaves are going down and our arrows have the same direction. The length of each arrows also shows the magnitude of each movement, which also makes sense. Therefore, my algorithm actually has a good performance on this image.

Grove (frame 7-8):



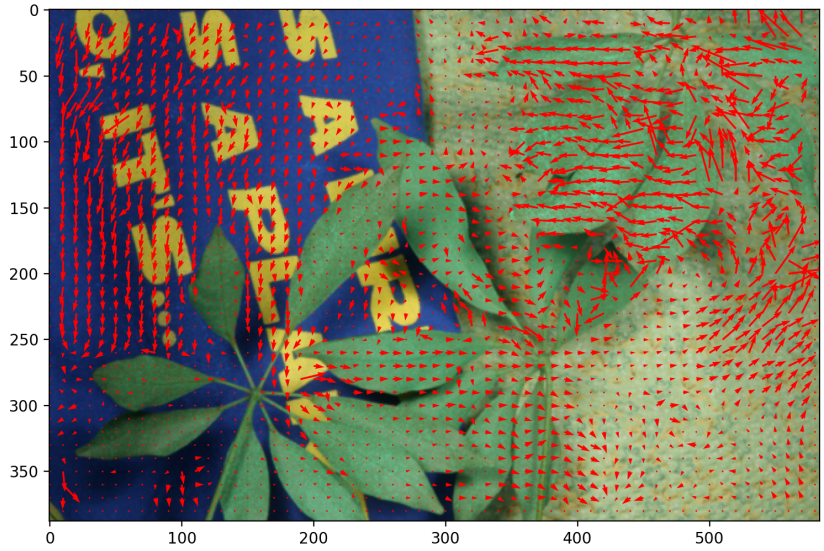
My algorithm actually has a good performance on this image. We can see the movement of all arrows on the image actually have the same gradient on y x-axis, which makes good sense. The length of arrows also show the magnitude of movement. Therefore, my algorithm actually has a good performance on this image.

Mequon (frame 7-8):



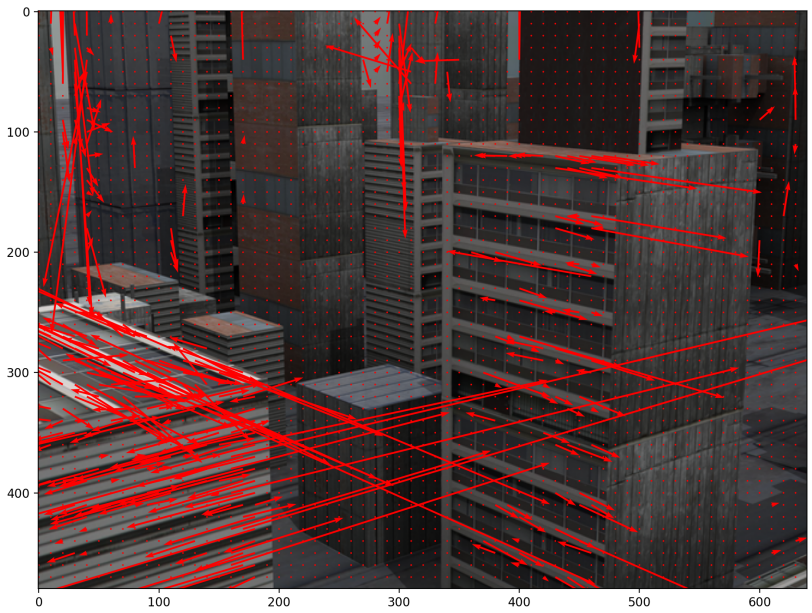
My algorithm actually has a good performance on this image. The result may seem a little bit messy, since we have flow in the image and also objects in the image all have different directions for the movement. We have a really good movement directions and length. Therefore, my algorithm actually has a good performance on this image.

Schefflera (frame 7-8):



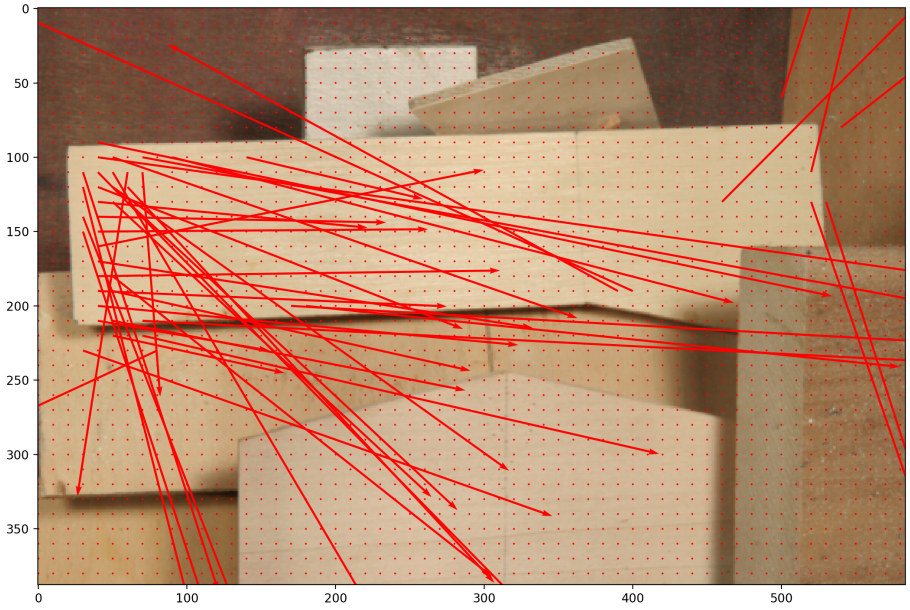
My algorithm actually has a good performance on this image. Each object has a clear direction for movement and a clear magnitude of movement. It is really clean and clear. Therefore, my algorithm actually has a good performance on this image.

Urban (frame 7-8):



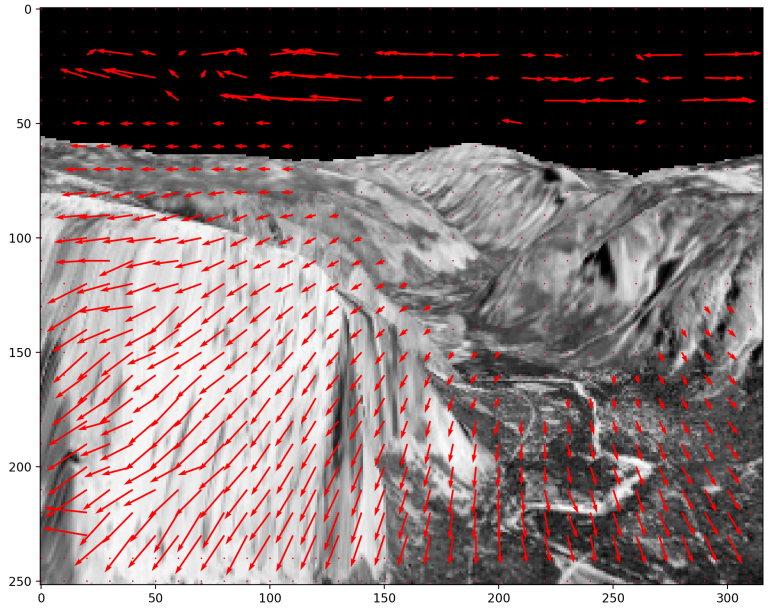
My algorithm actually has a good performance on this image. The result is super clean and all arrows have plausible directions and magnitude if you take a clear see on the continuous frame. Therefore, my algorithm actually has a good performance on this image.

Wooden (frame 7-8):



My algorithm actually has a good performance on this image. The result of this image is really clear we can expect that the wood on the centre will drop down its left side and pull up its right side and it is a rotation which is exactly the same with the real situation. The direction is super clear and the magnitudes also make good sense. Therefore, my algorithm actually has a good performance on this image.

Yosemite (frame 7-8):



My algorithm actually has a good performance on this image. The result is pretty clear and all arrows are clear for their magnitude and direction. All directions are also really clear and make sense. Therefore, my algorithm actually has a good performance on this image.

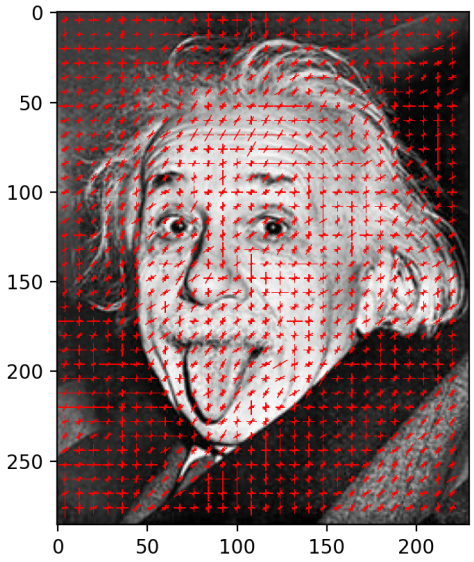
Q4) (60 marks)

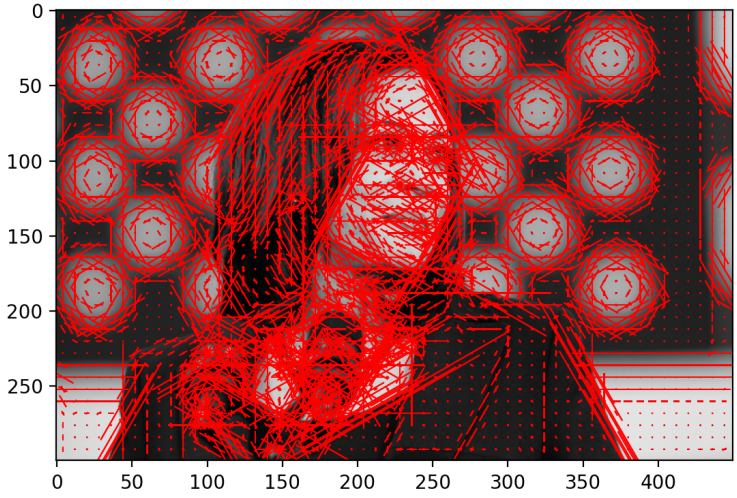
1.

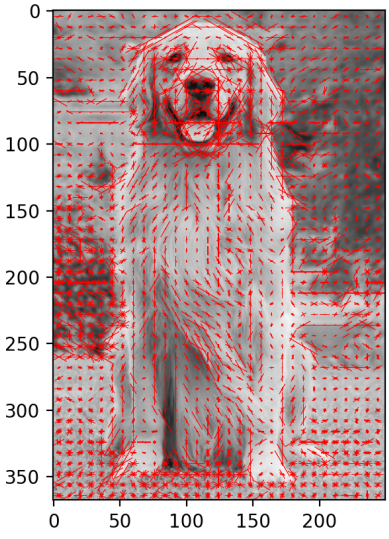
```python
def getGradient(src, threshold):
    m, n = src.shape
    directions = np.zeros((m, n), dtype=float)
    gradients = np.zeros((m, n), dtype=float)
    sobel_x = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    sobel_y = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    grad_x = convolve(src, sobel_x)
    grad_y = convolve(src, sobel_y)
    for i in range(m):
        for j in range(n):
            gradients[i][j] = math.sqrt(grad_x[i][j] * grad_x[i][j] + grad_y[i][j] * grad_y[i][j])
            if gradients[i][j] < threshold: gradients[i][j] = 0
            if not grad_x[i][j]: directions[i][j] = np.pi/2
            else: directions[i][j] = math.atan(grad_y[i][j] / grad_x[i][j])
            directions[i][j] *= (180 / math.pi)
            if directions[i][j] < 0: directions[i][j] += 180
    return gradients, directions
```

2 & 3.

```python
def HOG(src, tao, threshold, block_size, name):
    x, y = src.shape
    gradients, directions = getGradient(src, threshold)
    m = x // tao
    n = y // tao
    histogram = np.zeros((m, n, 6), dtype=float)
    occ = np.zeros((m, n, 6), dtype=int)
    for i in range(m * tao):
        for j in range(n * tao):
            if 15 <= directions[i][j] and directions[i][j] < 45:
                histogram[i // tao][j // tao][1] += gradients[i][j]
                occ[i // tao][j // tao][1] += 1
            elif 45 <= directions[i][j] and directions[i][j] < 75:
                histogram[i // tao][j // tao][2] += gradients[i][j]
                occ[i // tao][j // tao][2] += 1
            elif 75 <= directions[i][j] and directions[i][j] < 105:
                histogram[i // tao][j // tao][3] += gradients[i][j]
                occ[i // tao][j // tao][3] += 1
            elif 105 <= directions[i][j] and directions[i][j] < 135:
                histogram[i // tao][j // tao][4] += gradients[i][j]
                occ[i // tao][j // tao][4] += 1
            elif 135 <= directions[i][j] and directions[i][j] < 165:
                histogram[i // tao][j // tao][5] += gradients[i][j]
                occ[i // tao][j // tao][5] += 1
            else:
                histogram[i // tao][j // tao][0] += gradients[i][j]
                occ[i // tao][j // tao][0] += 1
    src = src.astype('float') / 255.0
    angles_segs = np.arange(0, 180, 180 / 6)
    meshX, meshY = np.meshgrid(np.r_[int(tao / 2) : tao * (y // tao) - 1: tao], np.r_[int(tao /2) : tao * (x // tao) -  1: tao])
    plt.imshow(src, cmap='gray', vmin=0, vmax=1)
```


0
50
100
150
200
250
0
50
100
150
200


0
50
100
150
200
250
0
50
100
150
200
250
300
350
400


0
50
100
150
200
250
300
350
0
50
100
150
200

4.

```
    length = (m-1) * (n-1) * 24
    descriptor = np.zeros((m-1,n-1,24), dtype=float)
    e = 0.001
    for i in range(m-1):
        for j in range(n-1):
            sigma = 0
            for k in range(0,6):
                sigma += histogram[i][j][k] ** 2
                descriptor[i][j][k] = histogram[i][j][k]
            for k in range(0,6):
                sigma += histogram[i+1][j][k] ** 2
                descriptor[i][j][6+k] = histogram[i+1][j][k]
            for k in range(0,6):
                sigma += histogram[i][j+1][k] ** 2
                descriptor[i][j][12+k] = histogram[i][j+1][k]
            for k in range(0,6):
                sigma += histogram[i+1][j+1][k] ** 2
                descriptor[i][j][18+k] = histogram[i+1][j+1][k]
            for k in range(24):
                descriptor[i][j][k] = descriptor[i][j][k] / math.sqrt(sigma + e ** 2)
    fpt = open(name, 'w')
    for i in range(m-1):
        for j in range(n-1):
            for k in range(24):
                fpt.write(str(descriptor[i][j][k])+' ')
            fpt.write('\n')
    fpt.close()
```

Details for descriptors are in files, 1.txt, 2.txt, and 3.txt.

Here are the helper functions and main.

```
def getGreyImge(img):
# Change the rgb value to grey. #
    rgb_weights = [0.2989, 0.5870, 0.1140]
    return np.dot(img[...,:3], rgb_weights)

def main(path, tao, threshold, block_size, name):
    img = plt.imread(path)
    src = None
    if len(img.shape) == 2: src = img
    else: src = getGreyImge(img)
    HOG(src, tao, threshold, block_size, name)


if __name__ == '__main__':
    main("./Q4/1.jpg", 8, 0.01, 2, "1.txt")
    main("./Q4/2.jpg", 8, 0.01, 2, "2.txt")
    main("./Q4/3.jpg", 8, 0.01, 2, "3.txt")
```