

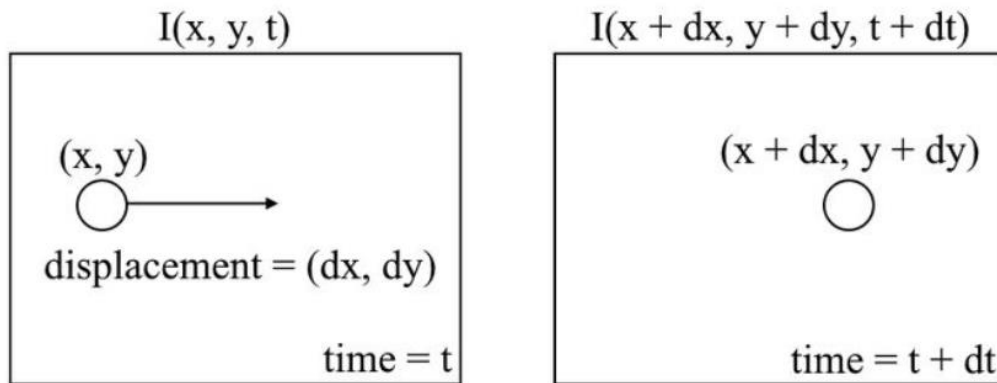
Computer Vision Assignment 5

Optical Flow

By- Apoorva Srivastava (2019702014)

Optical Flow:

Optical flow is the motion of objects between consecutive frames of sequence, caused by the relative movement between the object and camera.



We can express the image intensity (I) as a function of space (x, y) and time (t).

Sparse vs Dense Optical Flow:

Sparse optical flow gives the flow vectors of some "interesting features" (say few pixels depicting the edges or corners of an object) within the frame.

Dense optical flow, which gives the flow vectors of the entire frame (all pixels) upto one flow vector per pixel. As you would've guessed, Dense optical flow has higher accuracy at the cost of being slow/computationally expensive.

Basic Assumptions to form optical flow equation:

Brightness constancy assumption We assume that pixel intensities of an object are constant between consecutive frames.

$$I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$$

Taylor Series Approximation We use Taylor series approximation the of the RHS and remove common terms.

$$I(x + \delta x, y + \delta y, t + \delta t) = I(x, y, t) + \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t + \dots$$

$$\Rightarrow \frac{\partial I}{\partial x} \delta x + \frac{\partial I}{\partial y} \delta y + \frac{\partial I}{\partial t} \delta t = 0$$

Optical flow equation:

We divide above equation by dt to derive the optical flow equation:

$$\frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v + \frac{\partial I}{\partial t} = 0$$

where $u=dx/dt$ and $v=dy/dt$

$dl/dx, dl/dy$ and dl/dt are the image gradients along the horizontal axis, the vertical axis, and time.

We cannot directly solve the optical flow equation for u and v since there is only one equation for two unknown variables.

Lucas Kanade Optical Flow:

Assumptions:

- Two consecutive frames are separated by a small time increment (dt) such that objects are not displaced significantly.
- Spatial coherence constraint : pixel's neighbors have the same (u, v)

If you use a $m \times m$ window , total equations will be $n = m^2$

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

where q_1, q_2, \dots, q_n denote the pixels inside the window.

The set of equations may be represented in the following matrix form where $Av=b$. We now have to solve for two unknowns (V_x and V_y) with n equations, which is over-determined, so we use least squares fitting.

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix} \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix} \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

Algorithm and Code:

Step1: First of all for each pixel of Image the gradient in x direction, y direction and gradient with time are calculated.

```
#Computing Ix,Iy,It at every point on the img I1
Kernel_gradx=np.array([[ -1,1],[ -1,1]])
Kernel_grady=np.array([[ 1,1],[ -1,-1]])
Kernel_gradt=np.array([[ 1,1],[ 1,1]])*.25
#grad_x=convolution2d(im1,Kernel_gradx)
grad_x=convolve2d(img1, Kernel_gradx, boundary='symm', mode='same')
grad_y=convolve2d(img1, Kernel_grady, boundary='symm', mode='same')
grad_t=convolve2d(img1, -Kernel_gradt, boundary='symm', mode='same') +
        convolve2d(img2, Kernel_gradt, boundary='symm', mode='same')
```

Step2: Then at every pixel the square of gradient_x,square of gradient_y, product of gradient_x&gradient_y, product of gradient_x&gradient_t, product of gradient_t&gradient_y were computed for a pixel window of 3x3.

```
#Calculating sum(grad_x^2),sum(grad_y^2),sum(grad_x*grad_y),sum(grad_x*grad_t),sum(grad_t*grad_y)
#at every pixel for a window of 3x3
for i in range(0,h,gap+1):
    for j in range(0,w,gap+1):
        if i-gap<0 or j-gap<0 or i+gap==h or j+gap==w:
            pass
        else:
            grad_x_sq=np.sum(grad_x[i-gap:i+gap+1,j-gap:j+gap+1]**2)
            grad_y_sq=np.sum(grad_y[i-gap:i+gap+1,j-gap:j+gap+1]**2)
            grad_xy=np.sum(np.multiply(grad_x[i-gap:i+gap+1,j-gap:j+gap+1],grad_y[i-gap:i+gap+1,j-gap:j+gap+1]))
            grad_xt=np.sum(np.multiply(grad_x[i-gap:i+gap+1,j-gap:j+gap+1],grad_t[i-gap:i+gap+1,j-gap:j+gap+1]))
            grad_yt=np.sum(np.multiply(grad_y[i-gap:i+gap+1,j-gap:j+gap+1],grad_t[i-gap:i+gap+1,j-gap:j+gap+1]))
            #print(grad_x_sq,grad_y_sq,grad_xy,grad_xt,grad_yt)
```

Step3: From the obtained values Matrices were formed and were put in the equation as

shown-

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \quad v = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \quad b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}; \quad \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix}$$

```
#Forming the matrix
A = np.array([ [ grad_x_sq,grad_xy], [grad_xy, grad_y_sq] ])
b = np.array([ -grad_xt, -grad_yt ])
#print(A)
# if threshold τ is Larger than the smallest eigenvalue of A'A:
U,D,V_T = np.linalg.svd(A.T.dot(A))
#print(np.min(D))
if np.min(D)<thresh:
    u[i-gap:i+gap+1,j-gap:j+gap+1]=0
    v[i-gap:i+gap+1,j-gap:j+gap+1]=0
else:
    o_flow = np.linalg.inv(A.T.dot(A)).dot(A.T).dot(b)
    #o_flow = np.linalg.inv(A).dot(b)
    u[i-gap:i+gap,j-gap:j+gap]=o_flow[0]
    v[i-gap:i+gap,j-gap:j+gap]=o_flow[1]
return(u,v)
```

Step4: As shown in the code snippet above the equation were solved to obtain (u,v) i.e. Optical Flow at each pixel.

Step5: Using the above obtained Optical Flow the flow vectors are drawn over the image frame and also on a black background for better visualization.

```
def FlowMaps(folder,img1,u,v,granularity):
    import cv2
    import numpy as np
    h,w=img1.shape
    flow_map=np.zeros((h,w))
    im=img1
    #Visualization
    for y in range(h):
        for x in range(w):
            if y % granularity == 0 and x % granularity == 0:
                dx = int(u[y, x] )
                dy = int(v[y, x])
                cv2.arrowedLine(im, (x, y), (x + dx, y + dy), (255,255,255))
                cv2.arrowedLine(flow_map, (x, y), (x + dx, y + dy), (255,255,255))
    cv2.imwrite('./'+folder+'/FlowOnImg'+str(granularity)+'.png',im)
    cv2.imwrite('./'+folder+'/FlowMap'+str(granularity)+'.png',flow_map)
    return im, flow_map
```

Step6: After implementation the LucasKanade, Moving Object Detection and Segmentation was implemented. For this the above program was used as a module and optical flow was calculated.

Step7: The Optical Flow calculated was Dense but for object detection the Flow Maps were drawn using optical flow vector for few good features only as shown in the code below:

```
def FlowMaps(folder,img1,img2):
    import cv2
    import numpy as np
    import LucasKanade
    from LucasKanade import Calc_Flow

    #converting to gray_scale
    img1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
    img2=cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
    h,w=img1.shape
    flow_map=np.zeros((h,w))
    u,v=Calc_Flow(folder,img1,img2)
    num_corners=2000
    corners = cv2.goodFeaturesToTrack(img1,num_corners,0.01,1)
    x=[]
    y=[]
    num_corners=len(corners)
    for i in range(num_corners):
        x.append(int(corners[i][0][0]))
        y.append(int(corners[i][0][1]))
    im=img1
    for i,j in zip(x,y):
        dx = int(u[j, i] )
        dy = int(v[j, i])
        cv2.arrowedLine(im, (i, j), (i + dx, j + dy), (255,255,255))
        cv2.arrowedLine(flow_map, (i, j), (i + dx, j + dy), (255,255,255))
    return im, flow_map
```

Step8: From the obtained Flow Map a mask was generated to segment out the object and the mask was employed on the grayscale version of the image to perform segmentation using the following code-

```
def Mask(folder,im):
    import numpy as np
    import cv2
    im=np.asarray(im)
    #im=im[:, :,0]
    h,w=im.shape
    #print(im.shape,h,w)
    for i in range(0,h,6):
        for j in range(0,w,6):
            if i-1==0 or i-3==0 or j-3==0 or i+1==h or j+3==w or i+3==h or j-1==0 or j+1==w or i+2==h or i-2==0 or j-2==0:
                pass
            else:
                window=im[i-3:i+3,j-3:j+3]
                if np.count_nonzero(window)>0:
                    im[i-3:i+3,j-3:j+3]=255
    return im
```

```
def Segment(folder,img,mask):
    import numpy as np
    img=np.asarray(img)
    mask=np.asarray(mask)
    h,w,_=img.shape
    new_img=np.zeros((h,w))
    for i in range(h):
        for j in range(w):
            if mask[i,j]==0:
                new_img[i,j]=0
            else:
                new_img[i,j]=img[i,j,0]
    return(new_img)
```

Step9: From the obtained Segmentation the contour was calculated and the box was drawn around the moving object using the following code-

```
def Box(folder,img1,img):
    import cv2 as cv2
    import numpy as np

    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    image, contours, hierarchy = cv2.findContours(img,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    for item in range(len(contours)):
        cnt = contours[item]
        if len(cnt)>1000:
            x,y,w,h = cv2.boundingRect(cnt)
            cv2.rectangle(img1,(x,y),(x+w,y+h),(255,255,0),1)
    return(img1)
```

Step10: The above code was for 2 images and hence was imported as a module in the program for the implementation of tracking of a moving object throughout the video for its recursive implementation.

```
def Video_track(Video):
    import Video_Image
    from Video_Image import Video2Image
    import cv2
    import numpy as np
    import LucasKanade
    from LucasKanade import Calc_Flow
    import os
    import Detection_Segmentation
    from Detection_Segmentation import FlowMaps,Mask,Segment,Box
    import Image_Video
    from Image_Video import Image2Video

    folder,image_pre,num_frames=Video2Image(Video)
    if not os.path.exists('box_images'):
        os.makedirs('box_images')
    for i in range(1,num_frames,1):
        image1_name='img'+str(i)
        image2_name='img'+str(i+1)
        granularity=2
        img1=cv2.imread('./'+folder+'/'+image1_name+'.jpg')
        img2=cv2.imread('./'+folder+'/'+image1_name+'.jpg')
        im,flow_map=FlowMaps(folder,img1,img2)
        # cv2.imwrite('./flow_on_im.jpg',im)
        # cv2.imwrite('./flow_map.jpg',flow_map)
        mask=Mask(folder,flow_map)
        # cv2.imwrite('./mask.jpg',mask)
        seg=Segment(folder,img1,mask)
        cv2.imwrite('./seg.jpg',seg)
        seg_im=cv2.imread('./seg.jpg')
        boxed_im=Box(folder,img1,seg_im)
        cv2.imwrite('./box_images/img'+str(i)+'.jpg',boxed_im)

    image_array=[]
    Input= './box_images/img'
    Output = './box_images/box_video.mp4'
    fps = 25

    for i in range(1,num_frames-1,1): #Creating array of images to be converted into video
        filename= Input +str(i)+'.jpg'
        img=cv2.imread(filename)
        #print("Loading"+filename)
        image_array.append(img)
        height,width,channels=img.shape
        size=(width,height)
        Image2Video(Output, fps,size,image_array)

if __name__ == "__main__":|
    import cv2
    Video=cv2.VideoCapture(r'C:\Users\Asus\Videos\Antique.mp4')
    Video_track(Video)
```

Analysis and Results:

Hyperparameters: The window size for performing the optical flow, the threshold for the minimum eigen value allowed for A matrix acted as hyperparameters.






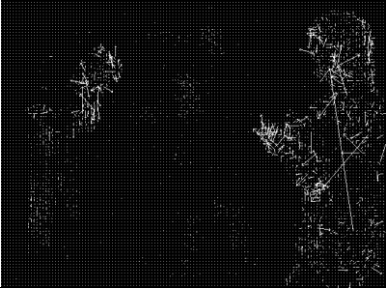



Window Size: The best window size was 3x3 and it was observed that as the window size was increased the density of the optical vector increased and some spurious vectors also got added showing the global flow.

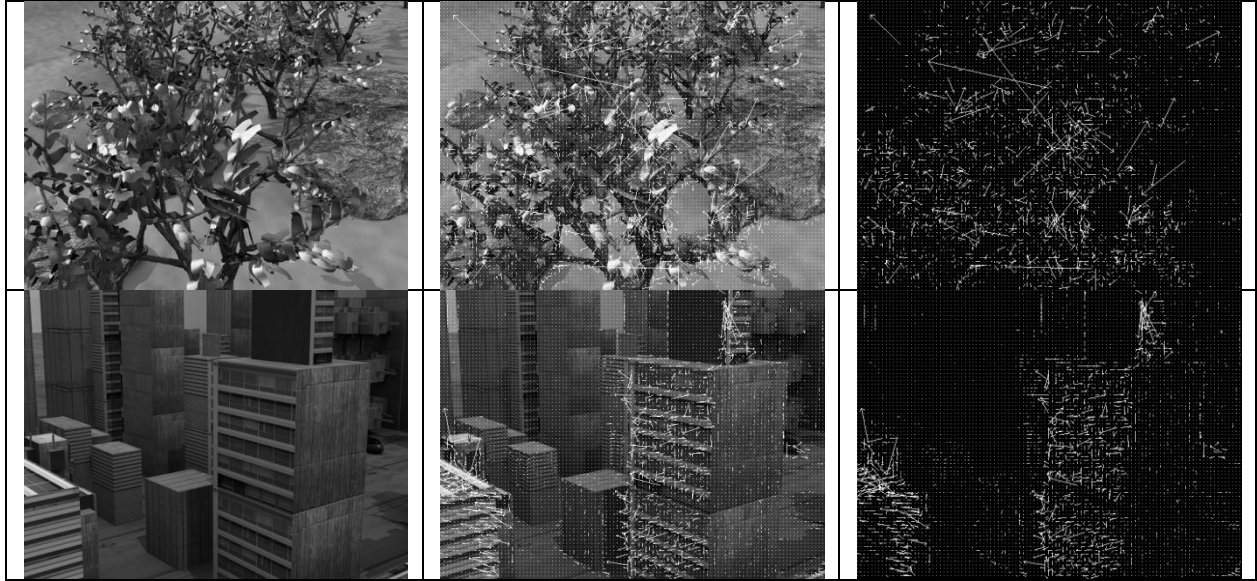
Minimum Threshold for Eigen Value of Matrix A: 0.01 was chosen to be the optimum threshold. Increasing beyond this decreases the number of valid flow vectors and vice-versa. However, it hugely depended on the image that in what range Eigen Values lie.

Granularity: A parameter granularity was introduced to present the flow vectors neatly on the flow map.

All these hyperparameters can be customized in an independent program FlowMap.ipynb

RESULTS on the given Consecutive Images:

Image	Flow_Map_Image	Flow_Map
		
		
		



Results on the image frames provided are shown below, the folder containing all these and many other intermediate results is provided at the link below:- https://iiitaphyd-my.sharepoint.com/:f/g/person/apoorva_srivastava_research_iiit_ac_in/Ejg1MW0fWfBPIJXe8ZUhm0Bb7g8Q075Z7fGP2FkXfK5JQ?e=EMaVwn

Analysis of how does the algorithm works when camera is moving:

When camera moves the algorithm detects the relative motion between the camera and the moving object. However, the motion of static object is detected only in the textured region. In the non-textured static regions the motion of camera has no effect and is considered static but in textured region the camera motion is assigned to this region in opposite direction.

It can be solved if a global camera motion is subtracted from the local motion of the moving object. That is by observing a known static region if we obtain its flow vector due to camera motion and subtract it from all the flow vectors then we can again stabilize the disturbance caused by the camera motion.

Below is the result on a self clicked Video where camera was moving:



Here the box in the center on the hanging objects are correct and depict the correct moving object, however the box appearing on the calendar on the right at the top and on the calendar is due to the motion of camera, as they were static and textured hence still are shown to be moving. While it can be seen that there is no box on the background walls though the camera was moving because they didn't have enough texture to show that small motion.

The link for the clicked video and the video with object tracking is given below:-

https://iiitaphyd-my.sharepoint.com/:f:/g/personal/apoorva_srivastava_research_iiit_ac_in/Ejg1MW0fWfBPIJXe8ZUhm0Bb7g8Q075Z7fGP2FkXfK5JQ?e=EMaVwn

Submitted CodeFiles and their Functions:

1. [LucasKanade.ipynb](#) This is the basic implementation of Dense Optical Flow using Lucas Kanade Algorithm for two images. This file has also been used as a Lucas Kanade Module in other programs.

2. [FlowMap.ipynb](#) This is independent program for getting Flow Map on any two image with customized hyperparameter options.
3. [Detection_Segmentation.ipynb](#) This is the program to draw bounding box around any moving object if 2 consecutive frames are given.
4. [Video_Tracking.ipynb](#) This is the program for drawing bounding box along every moving object throughout the video. It returns a video with the boxes.
5. [Image_Video.ipynb](#) This is the program to convert Images to Video.
6. [Video_Image.ipynb](#) This is the program to convert Video to Images.