Project 5 – Optical Flow

March 6, 2019

Jian Sun
DUID: 873397832

1 CONTENT

2 Introduction

In image processing, optical flow computation technique is a popular method to track motion objects. This project is to implement three methods to build optical flow model and track the motion of running cars. The first method is to extract key points by harris corner detector and then code Brightness Constancy Model to calculate the moved distance. Finally, using arrow line visualizes the change. The second method is to extract key points by harris corner detector and use the Lucas-Kanade (LK) motion estimation method to calculate the moved distance and visualize the optical flow. The third method is to use cv2.goodFeaturesToTrack() function to extract key points and use the Lucas-Kanade (LK) motion estimation method to calculate the moved distance and visualize the optical flow. We will record video for each method and compare them to draw conclusion

3 Data Exploration

The selected dataset is video, slow_traffic_small.mp4.

The frame size of video is 640 X 360. The duration is 31 seconds.

4 Model

4.1 Theory

4.1.1 Harris Corner Detector

Harris Corner Detector is a corner detection operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. It takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45 degree angles, and has been proved to be more accurate in distinguishing between edges and corners. Click here and here for more details. ### Brightness Constancy Model The pixel value doesn't change after motion. With the help of this property, we can calculate the pixel's motion distance. The equations are shown below.

$$I(x,y,t) = I(x+u, y+v, t+1)$$
$$= I(x,y,t) + I_x * u + I_y * v + I_t$$
$$I_x * u + I_y * v + I_t = 0$$
$$\begin{bmatrix} \sum(I_x^2) & \sum(I_x I_y) \\ \sum(I_x I_y) & \sum(I_y^2) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\sum(I_x I_t) \\ -\sum(I_y I_t) \end{bmatrix} \tag{1}$$

Solving equation (1) gets the motion distance u and v. ### Lucas-Kanade (LK) Motion Estimation Method Lucas–Kanade method is a widely used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. It assumes that the flow is essentially constant in a local neighbourhood of the pixel under consideration, and solves the basic optical flow equations for all the pixels in that neighbourhood, by the least squares criterion.

By combining information from several nearby pixels, the Lucas–Kanade method can often resolve the inherent ambiguity of the optical flow equation. It is also less sensitive to image noise than point-wise methods. On the other hand, since it is a purely local method, it cannot provide flow information in the interior of uniform regions of the image. ## Method 1 In this part, we select cv2.cornerHarris( ) to extract key points and then we code Brightness Constancy Model and visualize them.
In meanwhile, we also tried to choose key points among the whole image and track them.

4.2 Method 2.1

In this part, we also extract key points in harris corner detector, then we choose cv2.calcOpticalFlowPyrLK( ) to implement Lucas-Kanade (LK) Motion Estimation Method and record the video.

4.3 Method 2.2

In this part, we select to d ShiTomasi corner detection in cv2.goodFeaturesToTrack(), then we use cv2.calcOpticalFlowPyrLK( ) to implement Lucas-Kanade (LK) Motion Estimation Method and record the video.

4.4 Record Video

For each method, we will save the generated images firstly, then use cv2.VideoWriter( ) to record the video and output it.

5 Conclusion

5.1 Method 1

Please click here to play the video for method 1. This video shows a situation that we only track the key points gotten from harris corner detector.
Please click here to play the video for method 1. This video shows a situation that we track the key points among the whole images.

According to the comparison, I prefer the one that can track the key points among the whole images, because I cannot only track the key points around the motion objects, but also learn that how the other points change when nothing passes this points.

5.2 Method 2.1 vs Method 2.2

Please click here to play the video for method 2.1.
Please click here to play the video for method 2.2.

Method with harris corner detector brings us a clear video. We tracked 25 key points this time. And each one is accurately predicted by the optical flow computation. In meanwhile, method with ShiTomasi corner detector tracks detected points and predicts the objects' motion very good. Generally, I prefer the method with harris corner detector, because the tracked motion here look more naturally, the line is more smooth.

6 Code

6.1 Method 1

```
In [1]: def BGR2RGB(Input):
            output = np.zeros(np.shape(Input));
            output[:,:,0] = Input[:,:,2]
            output[:,:,1] = Input[:,:,1]
            output[:,:,2] = Input[:,:,0]
            output = output.astype('uint8')
            return output
```

6.1.1 Track Key Points Gotten From Harris Corner Detector

```
In [ ]: import os
        import cv2
        import numpy as np
        from scipy.linalg import solve
        import matplotlib.pyplot as plt
        cap = cv2.VideoCapture('slow_traffic_small.mp4')

        ret, old_frame = cap.read()
        old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
        # Create a mask image for drawing purposes
        count=0
        while(True):
```

```
# Take first frame and find corners in it
ret, new_frame = cap.read()
mask = np.zeros_like(old_frame)
new_gray = cv2.cvtColor(new_frame, cv2.COLOR_BGR2GRAY)
M,N = np.shape(old_gray)
tran_img = cv2.cornerHarris(np.float32(old_gray),2,3,0.04)
p_location = np.where(tran_img>0.05*tran_img.max())
rrow = np.shape(p_location)[1]
kp_old = np.zeros([rrow,2])
kp_new = np.zeros([rrow,2])
for i in range(rrow):
    kp_old[i,] = (p_location[0][i],p_location[1][i])

#kp_new = np.zeros([rrow,2])
for i in range(np.shape(kp_old)[0]):
    ## get x and y
    x = int(kp_old[i,0])
    y = int(kp_old[i,1])

    ## start loop from first 3*3 matrix
    if((x-1>=0)&(x+3<=640))&((y-1>=0)&(y+3<=360)):
    #if (y_down-y_up>=5)&(x_right-x_left>=5):
        mat_9 = old_gray[x-1:x+3,y-1:y+3]
        MAT_9 = new_gray[x-1:x+3,y-1:y+3]
        ## calculate Ix Iy Ixy Ix2 Iy2 It
        I_x=[[mat_9[j+1,g]-mat_9[j,g] for g in range(3)]for j in range(3)]
        I_y=[[mat_9[j,g+1]-mat_9[j,g] for g in range(3)]for j in range(3)]
        I_x=np.reshape(I_x,(1,9))
        I_y=np.reshape(I_y,(1,9))
        I_x2=np.sum(np.power(I_x,2))
        I_y2=np.sum(np.power(I_y,2))
        I_xy=np.dot(I_x,np.transpose(I_y))
        I_t=[[MAT_9[j,g]-mat_9[j,g] for g in range(3)]for j in range(3)]
        I_t=np.reshape(I_t,(9,1))

        ## construct linear equation and solve it
        I_xt=np.dot(I_x,I_t)
        I_yt=np.dot(I_y,I_t)
        if (I_x2*I_y2-int(I_xy)*int(I_xy)==0): continue
        else:
            A=((I_x2, int(I_xy)), (int(I_xy), I_y2))
            B=(int(-I_xt),int(-I_yt))
            u,v=np.linalg.solve(A,B)
            '''
            ## prepare for drawing arrow line and avoid out of bound point
            if (x+2*u<0): x_new=int(0)
            elif (x+2*u>640): x_new=int(640)
            else: x_new=round(x+2*u)
```

```
                    if (y+2*v<0): y_new=int(0)
                    elif (y+2*v>360): y_new=int(360)
                    else: y_new=round(y+2*v)
                    '''
                    if (abs(u)<4) or (abs(v)<4):
                        y_new=y+15*round(v)
                        x_new=x+15*round(u)
                    mask=cv2.arrowedLine(np.uint8(mask), (y,x),
                                        (int(y_new),int(x_new)), (0, 0, 255),2)
                else: continue
            count+=1
            ## show each frame with arrow line on each key point
            img = cv2.add(old_frame,mask)
            #cv2.imshow('Frame',img)
            text = ('./1_2/img_%d.png' % (count))
            cv2.imwrite(text,img)
            old_gray = new_gray.copy()
            old_frame = new_frame.copy()
            k = cv2.waitKey(30) & 0xff
            if k == 30: break

        cv2.destroyAllWindows()
        cap.release()
```

6.1.2 Record Video

```
In [50]: import os
         from os.path import isfile, join

         def convert_frames_to_video12(pathOut,fps):
             frame_array = []

             for i in range(913):
                 filename = ('./1_2/img_%d.png' % (i+1))
                 #reading each files
                 img = cv2.imread(filename)
                 height, width, layers = np.shape(img)
                 size = (width,height)
                 #inserting the frames into an image array
                 frame_array.append(img)

             out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX')
                                   , fps, size)

             for i in range(len(frame_array)):
                 # writing to a image array
                 out.write(frame_array[i])
             out.release()
```

```
In [51]: pathOut='optical_flow_12.mp4'
         convert_frames_to_video12(pathOut,18)
```

6.1.3 Track Key Points Among the Whole Image

```
In [ ]: import os
        import cv2
        import numpy as np
        from scipy.linalg import solve
        import matplotlib.pyplot as plt

        cap = cv2.VideoCapture('slow_traffic_small.mp4')
        ret, old_frame = cap.read()
        M,N = np.shape(old_gray)
        old_gray = cv2.cvtColor(np.float32(old_frame), cv2.COLOR_BGR2GRAY)
        count=0

        while(True):
            ret, new_frame = cap.read()
            # Create a mask image for drawing purposes
            mask = np.zeros_like(old_frame)
            new_gray = cv2.cvtColor(np.float32(new_frame), cv2.COLOR_BGR2GRAY)

            for i in range(5,M,15):
                for j in range(5,N,15):
                    ## start loop from first 9*9 matrix
                    if (i+6>M or j+6>N): continue
                    else:
                        mat_9 = old_gray[i-4:i+6,j-4:j+6]
                        MAT_9 = new_gray[i-4:i+6,j-4:j+6]

                        ## calculate Ix Iy Ixy Ix2 Iy2 It
                        I_x=[[mat_9[h+1,g]-mat_9[h,g] for g in range(9)]for h in range(9)]
                        I_y=[[mat_9[h,g+1]-mat_9[h,g] for g in range(9)]for h in range(9)]
                        I_t=[[MAT_9[h,g]-mat_9[h,g] for g in range(9)]for h in range(9)]
                        I_x=np.reshape(I_x,(1,81))
                        I_y=np.reshape(I_y,(1,81))
                        I_x2=np.sum(np.power(I_x,2))
                        I_y2=np.sum(np.power(I_y,2))
                        I_xy=np.dot(I_x,np.transpose(I_y))
                        I_t=np.reshape(I_t,(81,1))

                        ## construct linear equation and solve it
                        I_xt=np.dot(I_x,I_t)
                        I_yt=np.dot(I_y,I_t)
                        if (I_x2*I_y2-int(I_xy)*int(I_xy)==0): continue
                        else:
                            A=((I_x2, int(I_xy)), (int(I_xy), I_y2))
                        B=(int(-I_xt),int(-I_yt))
                        u,v=np.linalg.solve(A,B)
                        if (u>4 or v>4): continue
                        else:

                            x_new=i+3*round(u)
                            y_new=j+3*round(v)
                            mask=cv2.arrowedLine(np.uint8(mask), (j,i),
                                                    (int(y_new),int(x_new)), (0, 0, 255),2)
            count+=1
            ## show each frame with arrow line on each key point
            img = cv2.add(old_frame,mask)
            #cv2.imshow('Frame',img)
            text = ('./1_4/img_%d.png' % (count))
            cv2.imwrite(text,img)
            old_gray = new_gray.copy()
            old_frame = new_frame.copy()
            k = cv2.waitKey(30) & 0xff
            if k == 30: break

        cv2.destroyAllWindows()
        cap.release()
```

6.1.4 Record Video

```
In [42]: import os
         from os.path import isfile, join

         def convert_frames_to_video14(pathOut,fps):
             frame_array = []

             for i in range(913):
                 filename = ('./1_4/img_%d.png' % (i+1))
                 #reading each files
                 img = cv2.imread(filename)
                 height, width, layers = np.shape(img)
                 size = (width,height)
                 #inserting the frames into an image array
                 frame_array.append(img)

             out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'), fps, size)

             for i in range(len(frame_array)):
                 # writing to a image array
                 out.write(frame_array[i])
             out.release()
```

```
In [43]: pathOut='optical_flow14.mp4'
```

```
convert_frames_to_video14(pathOut,20)
```

6.2 Method 2

6.2.1 Key Points Detected in Harris Corner Detector

```
In [ ]: import numpy as np
        import cv2
        cap = cv2.VideoCapture('slow_traffic_small.mp4')
        # params for ShiTomasi corner detection
        feature_params = dict( maxCorners = 200,
                               qualityLevel = 0.3,
                               minDistance = 7,
                               blockSize = 25,
                               useHarrisDetector = 1)
        # Parameters for lucas kanade optical flow
        lk_params = dict( winSize  = (15,15),
                          maxLevel = 2,
                          criteria = (cv2.TERM_CRITERIA_EPS |
                                      cv2.TERM_CRITERIA_COUNT, 10, 0.03))
        # Create some random colors
        color = np.random.randint(0,255,(100,3))
        # Take first frame and find corners in it
        ret, old_frame = cap.read()
        old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
        p0 = cv2.goodFeaturesToTrack(old_gray, mask = None , **feature_params)
        '''
        tran_img = cv2.cornerHarris(np.float32(old_gray),2,3,0.04)
        p_location = np.where(tran_img>0.05*tran_img.max())
        rrow = np.shape(p_location)[1]
        p0 = [[[p_location[1][i]+0.0,p_location[0][i]]] for i in range(rrow)]
        p0=np.stack(p0[0:6],axis=0)
        '''

        # Create a mask image for drawing purposes
        mask = np.zeros_like(old_frame)
        count=0
        while(1):
            ret,frame = cap.read()
            frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            # calculate optical flow
            p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray,
                                    frame_gray, p0, None, **lk_params)
            # Select good points
            good_new = p1[st==1]
            good_old = p0[st==1]
            # draw the tracks
            for i,(new,old) in enumerate(zip(good_new,good_old)):
```

```
            a,b = new.ravel()
            c,d = old.ravel()
            mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
            frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
        img = cv2.add(frame,mask)
        cv2.imshow('frame',img)
        count+=1
        text = ('./2_1/img_%d.jpg' % (count))
        cv2.imwrite(text,img)
        k = cv2.waitKey(30) & 0xff
        if k == 30:
            break
        # Now update the previous frame and previous points
        old_gray = frame_gray.copy()
        p0 = good_new.reshape(-1,1,2)
    cv2.destroyAllWindows()
    cap.release()
```

6.2.2 Record Video

```
In [100]: def convert_frames_to_video2(pathOut,fps):
              frame_array = []

              for i in range(913):
                  filename = ('./2_1/img_%d.jpg' % (i+1))
                  #reading each files
                  img = cv2.imread(filename)
                  height, width, layers = np.shape(img)
                  size = (width,height)
                  #inserting the frames into an image array
                  frame_array.append(img)

              out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'),
                                    fps, size)

              for i in range(len(frame_array)):
                  # writing to a image array
                  out.write(frame_array[i])
              out.release()
```

```
In [101]: pathOut='optical_flow_21.mp4'
          convert_frames_to_video2(pathOut,10)
```

6.2.3 Key Points Detected in cv2.goodFeaturesToTrack()

```
In [ ]: import numpy as np
        import cv2
        cap = cv2.VideoCapture('slow_traffic_small.mp4')
```

```python
# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 200,
                       qualityLevel = 0.3,
                       minDistance = 7,
                       blockSize = 7 )
# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS |
                  cv2.TERM_CRITERIA_COUNT, 10, 0.03))
# Create some random colors
color = np.random.randint(0,255,(100,3))
# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None,
                             **feature_params)
count=0
# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
while(1):
    ret,frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray,
                    frame_gray, p0, None, **lk_params)
    # Select good points
    good_new = p1[st==1]
    good_old = p0[st==1]
    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new,good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        mask = cv2.line(mask, (a,b),(c,d), color[i].tolist(), 2)
        frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
    img = cv2.add(frame,mask)
    cv2.imshow('frame',img)
    count+=1
    text = ('./2_2/img_%d.jpg' % (count))
    cv2.imwrite(text,img)
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break
    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1,1,2)
cv2.destroyAllWindows()
cap.release()
```

6.2.4 Record Video

```
In [42]: def convert_frames_to_video3(pathOut,fps):
             frame_array = []

             #for i in range(len(files)):
             for i in range(913):
                 filename = ('./2_2/img_%d.jpg' % (i+1))
                 #reading each files
                 img = cv2.imread(filename)
                 height, width, layers = np.shape(img)
                 size = (width,height)
                 #inserting the frames into an image array
                 frame_array.append(img)

             out = cv2.VideoWriter(pathOut,cv2.VideoWriter_fourcc(*'DIVX'),
                                   fps, size)

             for i in range(len(frame_array)):
                 # writing to a image array
                 out.write(frame_array[i])
             out.release()
```

```
In [43]: pathOut='optical_flow_22.mp4'
         convert_frames_to_video3(pathOut,10)
```