Hw3 Horn & Schunck OpticalFlow

R07941023 呂彥穎 光電所一年級

Method

1. Create the frame 2 from frame 1

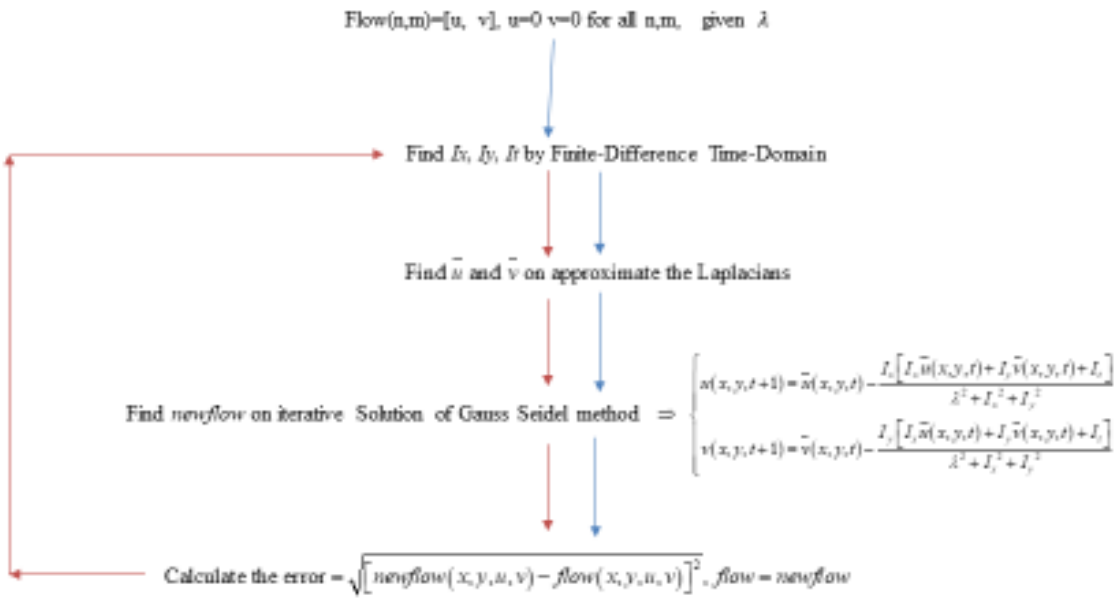
   Frame1 (original frame)

   Frame2 (shift one pixel to the right and downward)

Frame1

Frame2

Experiment



The parameter:

$$\begin{cases} \bar{u}(x,y) = \frac{1}{6}\left[u(x-1,y)+u(x,y+1)+u(x+1,y)+u(x,y-1)\right] \\ \quad +\frac{1}{12}\left[u(x-1,y-1)+u(x-1,y+1)+u(x+1,y+1)+u(x+1,y-1)\right] \\ \bar{v}(x,y) = \frac{1}{6}\left[v(x-1,y)+v(x,y+1)+v(x+1,y)+v(x,y-1)\right] \\ \quad +\frac{1}{12}\left[v(x-1,y-1)+v(x-1,y+1)+v(x+1,y+1)+v(x+1,y-1)\right] \end{cases}$$

$$I_x \approx \frac{1}{4}\begin{bmatrix} I(x+1,y,t)+I(x+1,y,t+1)+I(x+1,y+1,t)+I(x+1,y+1,t+1) \\ -I(x,y,t)-I(x,y,t+1)-I(x,y+1,t)-I(x,y+1,t+1) \end{bmatrix}$$

$$I_y \approx \frac{1}{4}\begin{bmatrix} I(x,y+1,t)+I(x,y+1,t+1)+I(x+1,y+1,t)+I(x+1,y+1,t+1) \\ -I(x,y,t)-I(x,y,t+1)-I(x,y+1,t)-I(x+1,y,t+1) \end{bmatrix}$$

$$I_t \approx \frac{1}{4}\begin{bmatrix} I(x,y,t+1)+I(x,y+1,t+1)+I(x+1,y,t+1)+I(x+1,y+1,t+1) \\ -I(x,y,t)-I(x,y+1,t)-I(x+1,y,t)-I(x+1,y+1,t) \end{bmatrix}$$

$$\Rightarrow \quad \begin{cases} u(x,y,t+1) = \bar{u}(x,y,t) - \dfrac{I_x\left[I_x\bar{u}(x,y,t) + I_y\bar{v}(x,y,t) + I_t\right]}{\lambda^2 + I_x{}^2 + I_y{}^2} \\ v(x,y,t+1) = \bar{v}(x,y,t) - \dfrac{I_y\left[I_x\bar{u}(x,y,t) + I_y\bar{v}(x,y,t) + I_t\right]}{\lambda^2 + I_x{}^2 + I_y{}^2} \end{cases}$$

Result:

λ =0.1

λ =1

λ =10

Validation:

Result – Iteration ($\lambda = 10$)

t = 0

Iteration=0, t = 1

Iteration=1, t = 1

Iteration=2, t = 1

10

Result ($\lambda = 10$)

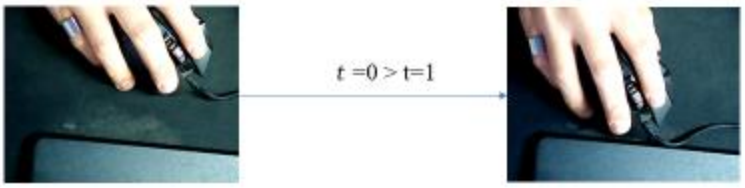
t=0

t=1

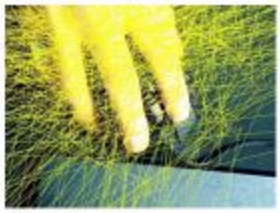
t=2

11

Improvement



t = 1, λ =0.1



t = 1 , λ =1

t = 1 , λ =10

Code (python3)

```
import numpy as np
import cv2
import time

def image(img1, img2):
    step = 10

    # Detect the image (gray or not)
    try:
        prevgray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
        gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    except:
        prevgray = img1
        gray = img2

    # https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html
    ti = time.time()
    # flow = cv2.calcOpticalFlowFarneback(prevgray, gray, None, 0.5, 3, 15, 3, 5, 1.2, 0)  #
Gunnar Farneback
    flow = CalcOpticalFlowHS(prevgray, gray, flow_lambda=0.1, iteration_error=1e-2,
convergence_limit=10)
    tf = time.time()
    print('t = ', tf - ti, )
    # Draw line
    img = gray.copy()
    h, w = gray.shape[:2]
    y, x = np.mgrid[step / 2:h:step, step / 2:w:step].reshape(2, -1).astype(int)
    fx, fy = flow[y, x].T
    lines = np.vstack([x, y, x + fx, y + fy]).T.reshape(-1, 2, 2)
    lines = np.int32(lines)
    line = []
    for l in lines:
        if l[0][0]-l[1][0] > 3 or l[0][1]-l[1][1] > 3:
            line.append(l)

    cv2.polylines(img, line, 0, (0, 255, 255))
    return img

def CalcOpticalFlowHS(prevgray, gray, flow_lambda, iteration_error, convergence_limit):

    boundary_pandding = 0
    img_h, img_w = prevgray.shape[:2]
    pooling_matrix = np.zeros((img_h+1, img_w+1), np.uint8) + boundary_pandding
    pre_matrix = pooling_matrix.copy()
    matrix = pooling_matrix.copy()
    pre_matrix[0:img_h, 0:img_w] = prevgray
    matrix[0:img_h, 0:img_w] = gray
    # cv2.imshow('test', pre_matrix)
    # cv2.waitKey(0)

    flow = np.zeros((img_h, img_w, 2))
    newflow = np.zeros((img_h, img_w, 2))
```

```
    error = iteration_error*2
    count = 1
    while (error > iteration_error):
        for i in range(img_w):
            for j in range(img_h):
                if i > 1 and j > 1 and i < img_w-1 and j < img_h-1:
                    Ex = 1 / 4 * ((int(pre_matrix[j, i+1]) + int(matrix[j, i+1]) +
int(pre_matrix[j+1, i+1]) + int(matrix[j+1, i+1])) - (int(pre_matrix[j, i]) + int(matrix[j, i])
+ int(pre_matrix[j+1, i]) + int(matrix[j+1, i])))
                    Ey = 1 / 4 * ((int(pre_matrix[j+1, i]) + int(matrix[j+1, i]) +
int(pre_matrix[j+1, i+1]) + int(matrix[j+1, i+1])) - (int(pre_matrix[j, i]) + int(matrix[j, i])
+ int(pre_matrix[j, i+1]) + int(matrix[j, i+1])))
                    Et = 1 / 4 * ((int(matrix[j, i]) + int(matrix[j+1, i]) + int(matrix[j, i+1])
+ int(matrix[j+1, i+1])) - (int(pre_matrix[j, i]) + int(pre_matrix[j+1, i]) + int(pre_matrix[j,
i+1]) + int(pre_matrix[j+1, i+1])))
                    u0 = 1/6*(flow[j, i-1][0] + flow[j+1, i][0] + flow[j, i+1][0] + flow[j-1,
i][0]) + 1/12*(flow[j-1, i-1][0] + flow[j+1, i+1][0] + flow[j-1, i+1][0] + flow[j+1, i-1][0])
                    v0 = 1/6*(flow[j, i-1][1] + flow[j+1, i][1] + flow[j, i+1][1] + flow[j-1,
i][1]) + 1/12*(flow[j-1, i-1][1] + flow[j+1, i+1][1] + flow[j-1, i+1][1] + flow[j+1, i-1][1])
                    u = (u0 - Ex*(Ex*u0+Ey*v0+Et)/(1+flow_lambda*(Ex**2+Ey**2)))
                    v = (v0 - Ey*(Ex*u0+Ey*v0+Et)/(1+flow_lambda*(Ex**2+Ey**2)))
                    newflow[j, i] = [u, v]
        error = (newflow-flow).reshape(-1)
        error = np.sqrt(np.sum(np.power(error, 2)))
        if count > convergence_limit:
            break
        flow = newflow
        print(count, error)
        count += 1

    return flow

def make_img2(img1, shiftw, shifth):
    boundary_pooling = 0
    img_h, img_w = img1.shape[:2]
    img2 = np.zeros((img_h, img_w), np.uint8) + boundary_pooling
    img2[shifth:img_h, shiftw:img_w] = img1[0:img_h-shifth, 0:img_w-shiftw]
    return img2

if __name__ == '__main__':
    img1 = cv2.imread("lena.bmp", cv2.IMREAD_GRAYSCALE)  # gray
    img2 = make_img2(img1, shiftw=1, shifth=1)
    Optical_img = image(img1, img2)
    cv2.imshow('Optical_img', Optical_img)
    cv2.imwrite('OpticalFlowHS.jpg', Optical_img)
    cv2.waitKey(0)
```