Advanced Computer Vision – HW3

Optical Flow Estimation

宋體淮, R09921135, Electrical Engineering

1. Description

- Implementing *Horn-Schunck* optical flow estimation
- Synthetically translate lena.bmp one pixel to the right and downward
- Try λ of 0.1, 1, 10.

2. Method

- The *Horn-Schunck* method of estimating optical flow is a method that enforce two constraints, which are brightness constancy and smooth flow field.

- Use the image transformation method to translate lena.bmp to lena_shifted.bmp by moving one pixel to the right and downward.

```python
def translate(img, dx, dy):
    T = np.float32([[1, 0, dx], [0, 1, dy]])
    shifted = cv2.warpAffine(img, T, (img.shape[1], img.shape[0]))
    return shifted
```

- Use *Horn-Schunck* algorithm as below to estimate the optical flow. More detail is referenced from [1].
  1. Precompute image gradients I_x, I_y.
  2. Precompute temporal gradients I_t.
  3. Initialize flow field $u = 0, v = 0$.
  4. While not converged, compute flow field updates for each pixel:

$$\hat{u}_{kl} = \bar{u}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_x$$

$$\hat{v}_{kl} = \bar{v}_{kl} - \frac{I_x \bar{u}_{kl} + I_y \bar{v}_{kl} + I_t}{\lambda^{-1} + I_x^2 + I_y^2} I_y$$

```python
def HornSchunck(img1, img2, lam=1, N_iter=8):
    # Laplacian kernel
    L_kernel = np.array([[1/12, 1/6, 1/12],
                         [1/6,    0, 1/6],
                         [1/12, 1/6, 1/12]], float)

    img1 = img1.astype(np.float32)
    img2 = img2.astype(np.float32)

    # set up initial velocities
    uInitial = np.zeros([img1.shape[0], img1.shape[1]])
    vInitial = np.zeros([img1.shape[0], img1.shape[1]])

    # Set initial value for the flow vectors
    u = uInitial
    v = vInitial

    # calculate derivatives
    [fx, fy, ft] = calDerivatives(img1, img2)

    for _ in range(N_iter):
        # calculate the average of neighborhood velocity
        u_Avg = filter2D(u, L_kernel)
        v_Avg = filter2D(v, L_kernel)
        der = (fx*u_Avg + fy*v_Avg + ft) / (1 + lam*(fx**2 + fy**2))
        u = u_Avg - fx * der
        v = v_Avg - fy * der

    return u, v
```

3. Result

- Iteration = 1:

Figure 1. result for Lambda = 0.1

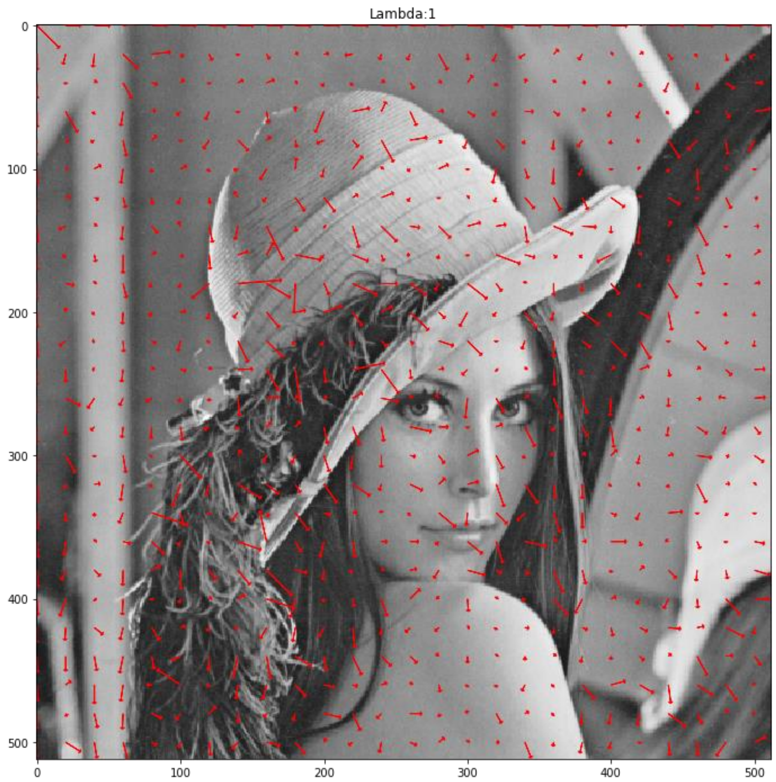

Figure 2. result for Lambda = 1

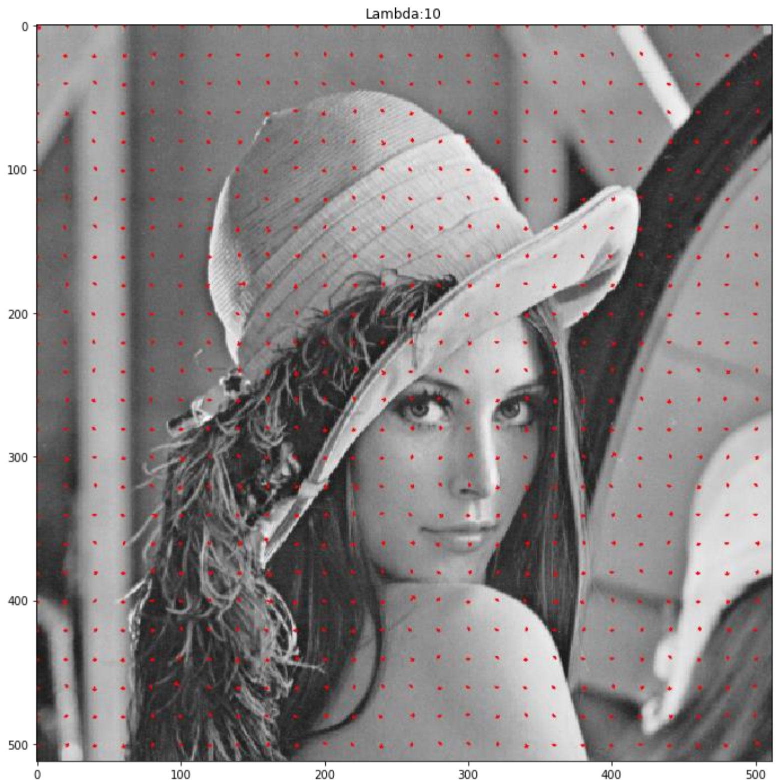

Figure 3. result for Lambda = 10

- Iteration = 4:

Figure 4. result for Lambda = 0.1

Figure 5. result for Lambda = 1

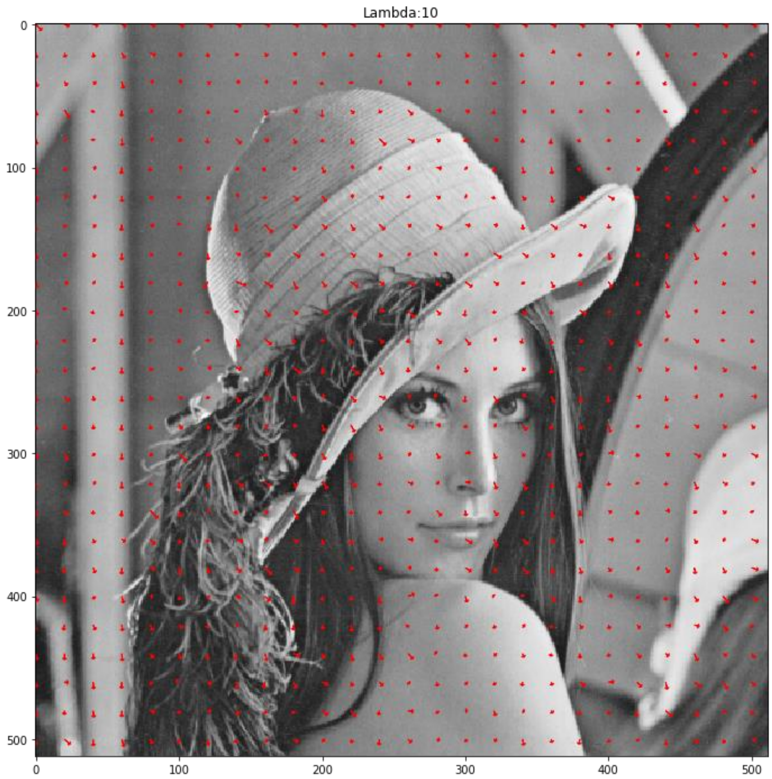

Figure 6. result for Lambda = 10

- Iteration = 16:

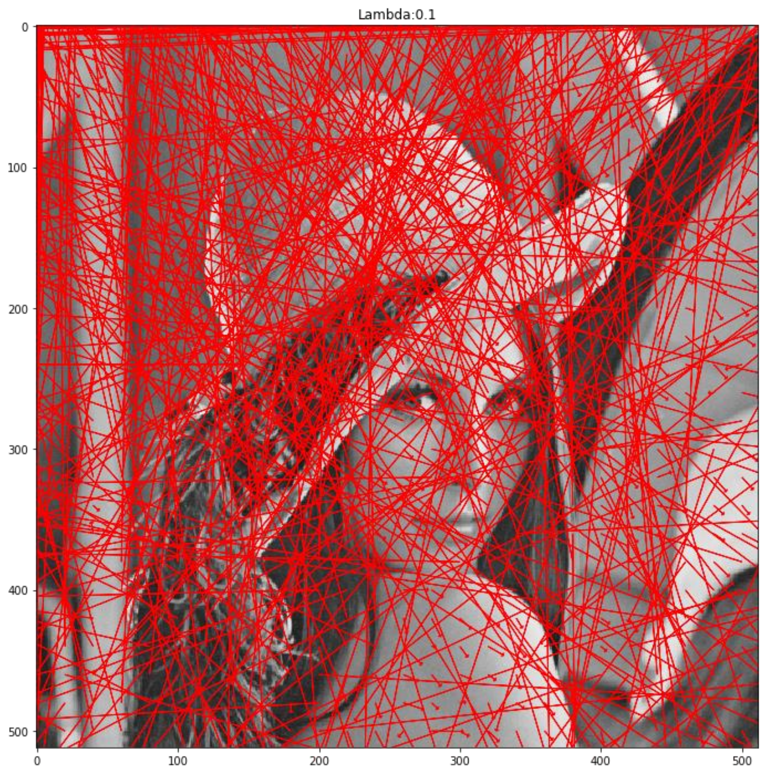

Figure 7. result for Lambda = 0.1

Figure 8. result for Lambda = 1

Figure 9. result for Lambda = 10

- Iteration = 64:

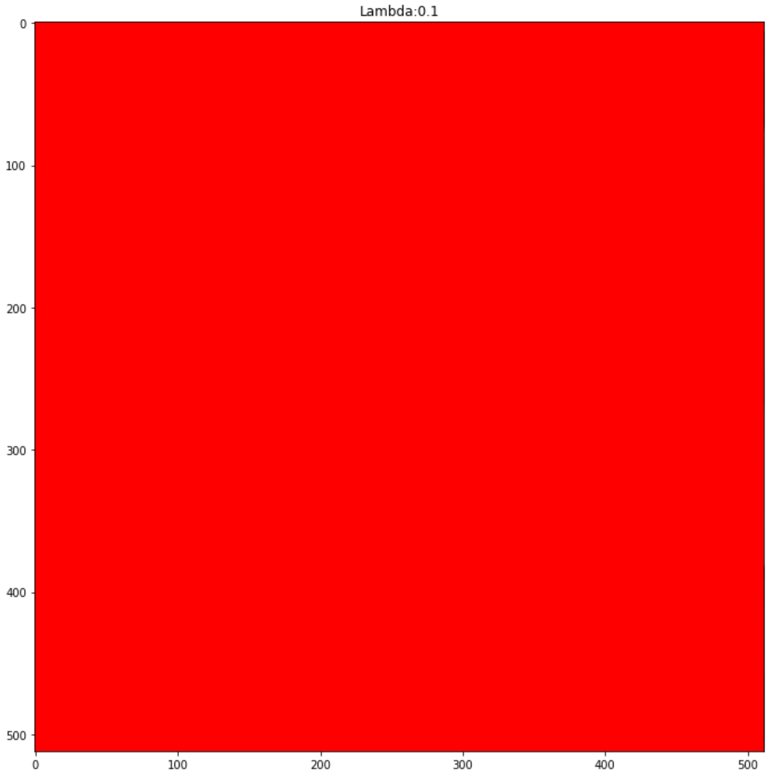

Figure 10. result for Lambda = 0.1

Figure 11. result for Lambda = 1

Figure 11. result for Lambda = 10

- Reference

[1] http://datahacker.rs/013-optical-flow-using-horn-and-schunck-method/