r06946003 湯忠憲

Calculating Optical Flow

Task description:

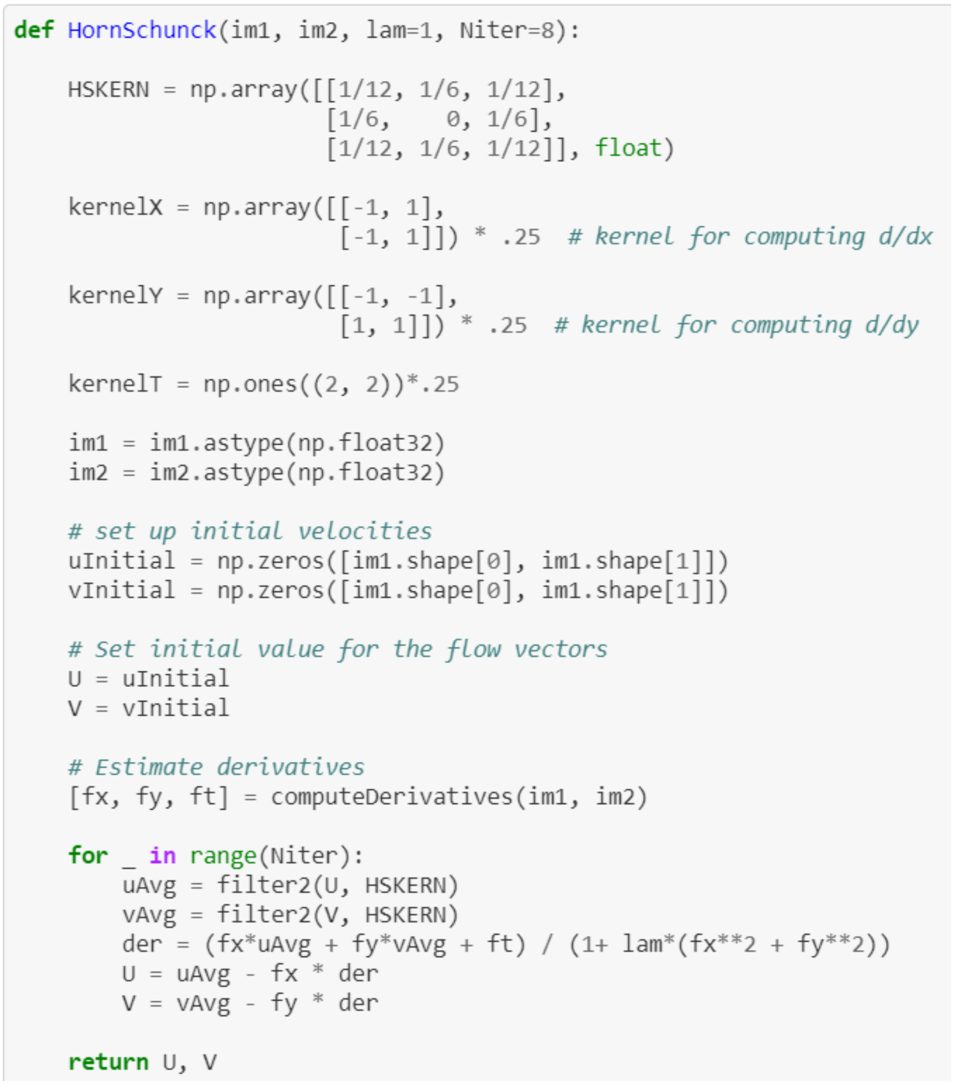
1. Implement Horn & Schunck optical flow estimation.
2. Synthetically translate lena.im one pixel to the right and downward.
3. Try λ of 0.1, 1, 10.

Method:

1. The Horn-Schunck method of estimating optical flow is a global method which introduces a global constraint of smoothness to solve the aperture problem
2. Translate lena.im to lena_shift.im by moving one pixel to the right and downward. Then crop the last row and column of lena_shift.im.
3. Use Horn-Schunck algorithm to find estimate the optical flow. The method with suggested iterative scheme is shown below:

```
HORN-SCHUNCK($I, \lambda$)
$I_x, I_y \leftarrow$ GRADIENT($I$)
for $y \leftarrow 0$ to $height - 1$
for $x \leftarrow 0$ to $width - 1$
        do $u(x,y) \leftarrow 0$
           $v(x,y) \leftarrow 0$
repeat
        do for $y \leftarrow 0$ to $height - 1$
            for $x \leftarrow 0$ to $width - 1$
                do $\bar{u} \leftarrow \frac{1}{4}(u(x-1,y) + u(x+1,y) + u(x,y-1) + u(x,y+1))$
                   $\bar{v} \leftarrow \frac{1}{4}(u(x-1,y) + u(x+1,y) + u(x,y-1) + u(x,y+1))$
                   $\gamma \leftarrow \frac{I_x(x,y)\bar{u} + I_y(x,y)\bar{v} + I_t(x,y)}{1 + \lambda(I_x(x,y)^2 + I_y(x,y)^2)}$
                   $u(x,y) \leftarrow \bar{u} - \gamma I_x(x,y)$
                   $v(x,y) \leftarrow \bar{v} - \gamma I_y(x,y)$
```

Principal code segment:

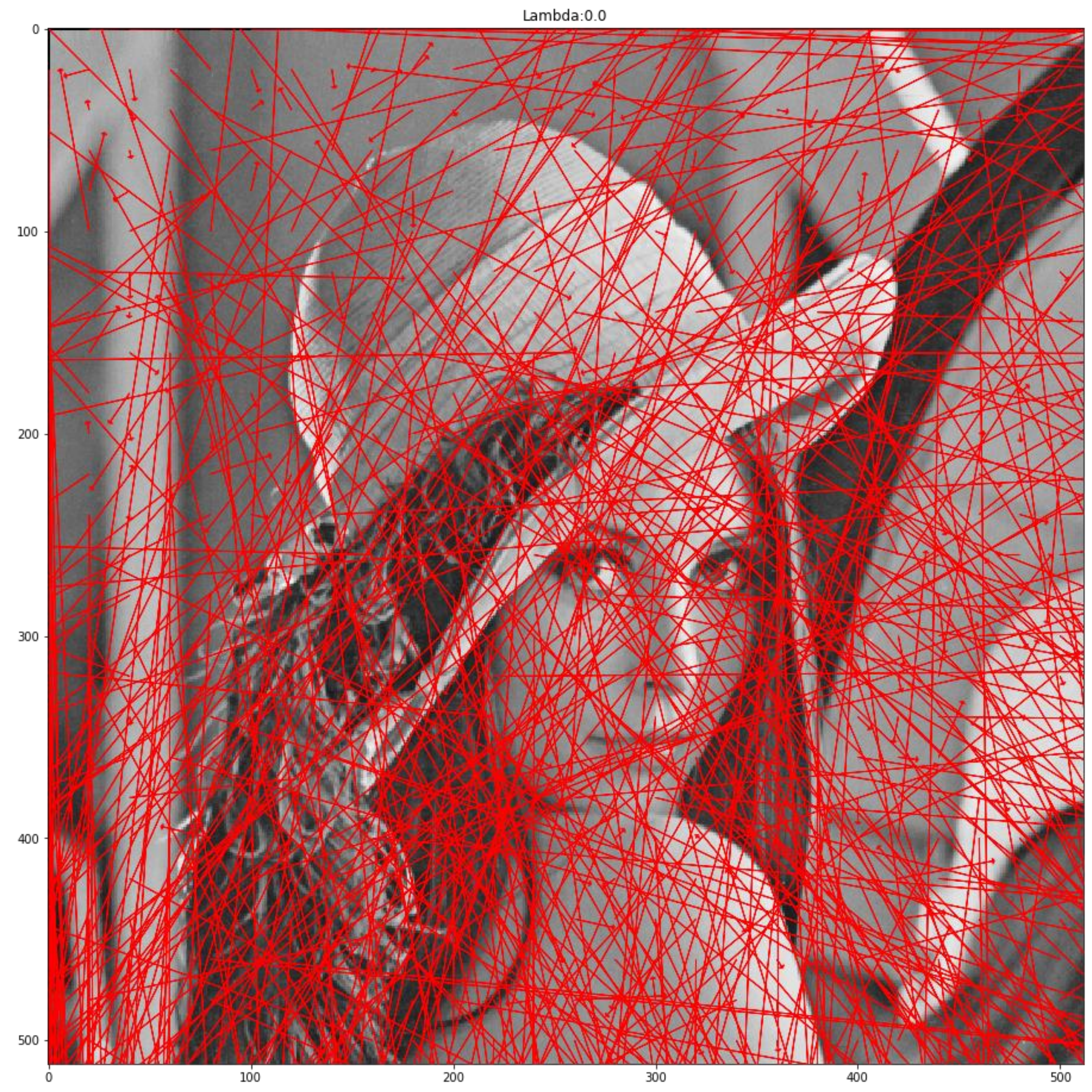


```
def HornSchunck(im1, im2, lam=1, Niter=8):

    HSKERN = np.array([[1/12, 1/6, 1/12],
                       [1/6,    0, 1/6],
                       [1/12, 1/6, 1/12]], float)

    kernelX = np.array([[-1, 1],
                        [-1, 1]]) * .25  # kernel for computing d/dx

    kernelY = np.array([[-1, -1],
                        [1, 1]]) * .25  # kernel for computing d/dy

    kernelT = np.ones((2, 2))*.25

    im1 = im1.astype(np.float32)
    im2 = im2.astype(np.float32)

    # set up initial velocities
    uInitial = np.zeros([im1.shape[0], im1.shape[1]])
    vInitial = np.zeros([im1.shape[0], im1.shape[1]])

    # Set initial value for the flow vectors
    U = uInitial
    V = vInitial

    # Estimate derivatives
    [fx, fy, ft] = computeDerivatives(im1, im2)

    for _ in range(Niter):
        uAvg = filter2(U, HSKERN)
        vAvg = filter2(V, HSKERN)
        der = (fx*uAvg + fy*vAvg + ft) / (1+ lam*(fx**2 + fy**2))
        U = uAvg - fx * der
        V = vAvg - fy * der

    return U, V
```

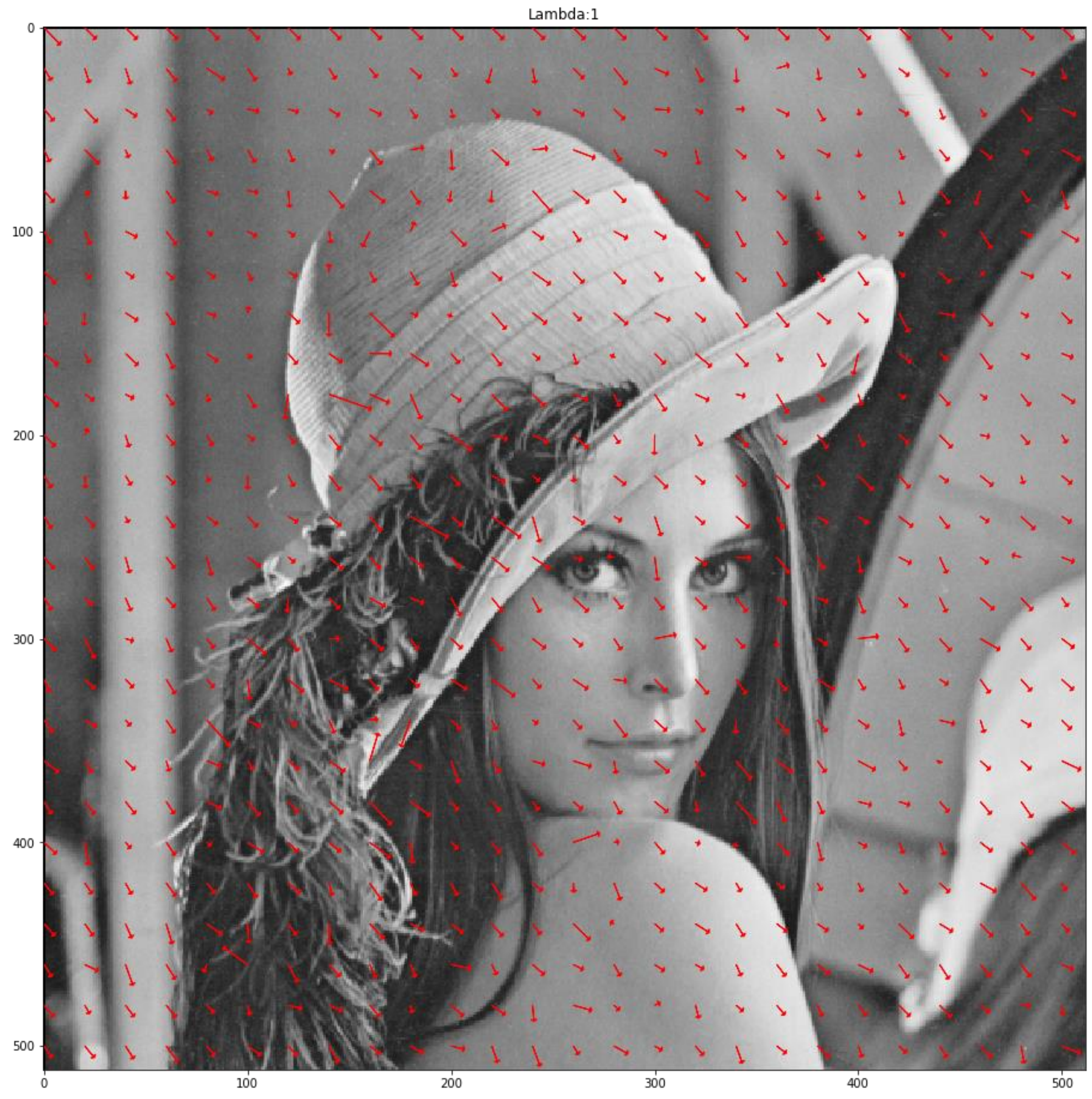
Experiment results:

Various settings of smoothing parameter lambda (λ) which affects the weight of the gradient in the calculation of the motion vector. Here we've tried 0.1, 1 and 10 of it. The results are shown below. We can find out that greater lambda produces (1) greater directional accuracy in the vectors (2) shorter direction vectors, distorting the displacement magnitudes slightly.

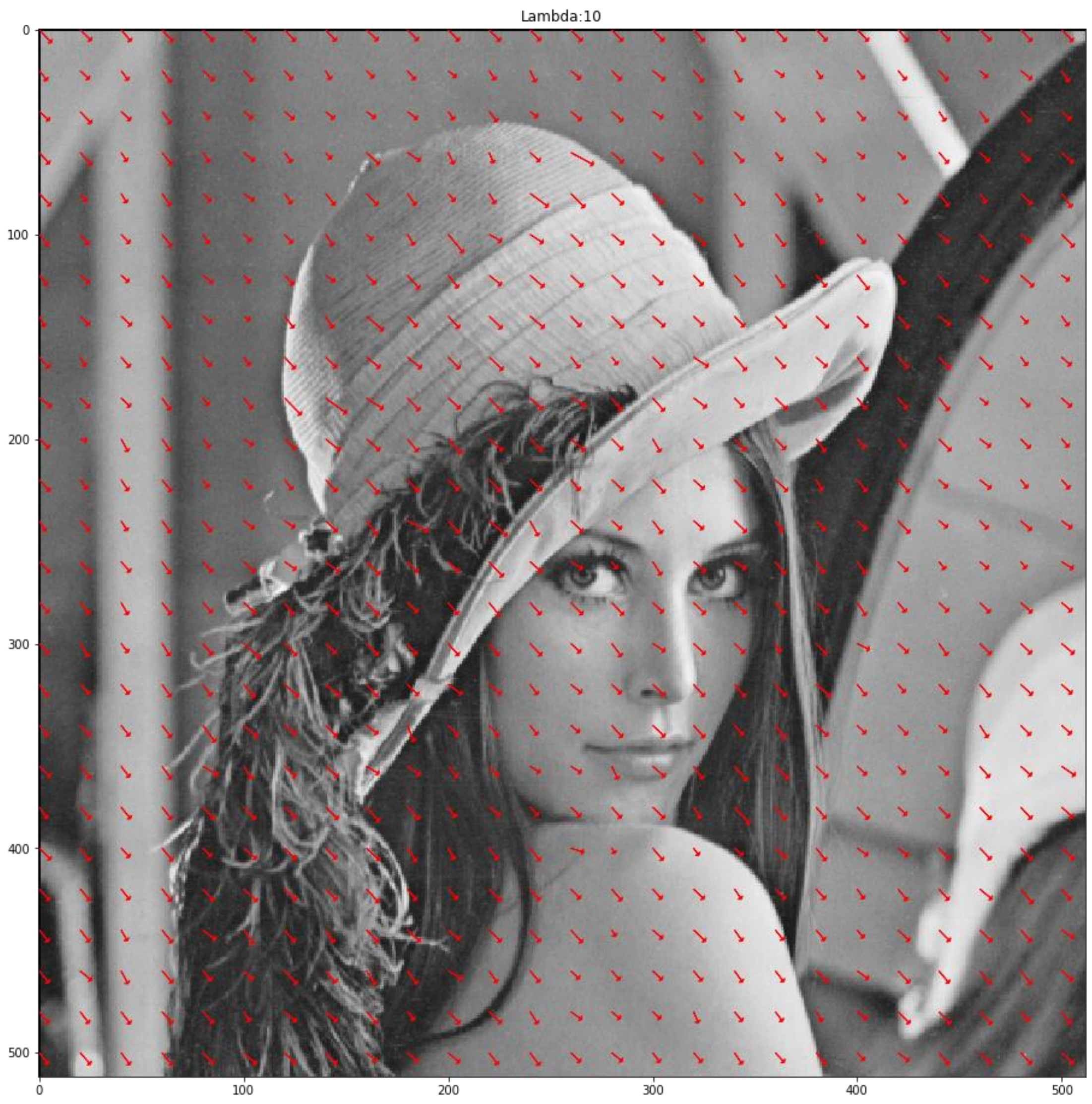
r06946003 湯忠憲



r06946003 湯忠憲



r06946003 湯忠憲


Lambda:10