Exercise 1 Correlation Method

April 11, 2018

** Xiangyi Cheng (xxc283)**

1 Concepts and Background

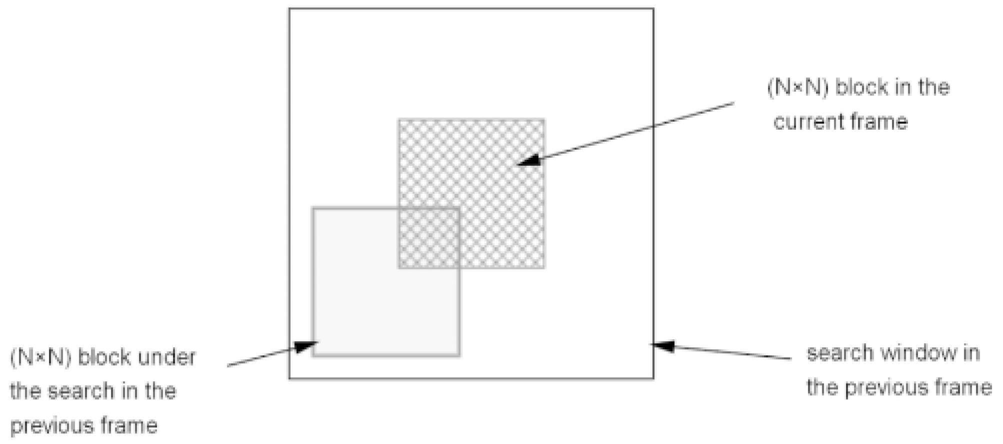
This notebook is created for explaining the basic idea of motion estimation using correlation method. Motion estimation is the process of determining motion vectors that describe the changing from one image to another. The usage of motion estimation is highly critical and broad. Tracking, detecting and identifying objects, computation of 3D structure and position, object and scene segmentation are all the practical applications related to motion estimation. Although the need and market of the motion estimation is giant, motion estimation is still in developing stage and there is no algorithm could ganrantee the high accuracy in every motion. The good side is that there are various algothrim are come up with by researchers in decades and it remains interesting fields for us to dig deeper.

There are numerous motion estimation method such as block-matching algorithm, phase correlation method, gradient method with solving motion gradient constraint equation. This notebook is about to apply the block-matching algorithm with correlation to estimate motion in two images. The basic idea behind this algothrim is to split the first frame into several blocks or grids. Then extract a patch from the previous frame and set a larger area near this patch in the next frame called search region as shown below. Correlations are calculated in each location and check the biggest response in the computation. The maximum response may be the same location in the next frame.

As for correlation, it is a useful concept in statistics. Correlation is any of a broad class of statistical relationships and it most often refers to how close two variables are to having a linear relationship with each other. The population correlation coefficient X,Y between two random variables X and Y with expected values X and Y and standard deviations X and Y is defined as:

$$\rho_{X,Y} = \mathrm{corr}(X,Y) = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

If a series of n measurements of X and Y are obtained as xi and yi for i = 1, 2, ..., n, then the sample correlation coefficient can be used to estimate the population Pearson correlation r between X and Y. The sample correlation coefficient is written as:


(N×N) block in the current frame
(N×N) block under the search in the previous frame
search window in the previous frame

correlation method

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2 \sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

However, a build-in function to compute the correlation between two matrix already exists in python so that the application is easier. With the idea above, we can demonstrate the motion estimation in a simple example as shown below.

2 Implement

To estimate the motion with correlation, several libraries are used and imported.

```
In [1]: from scipy.signal import correlate2d
        import numpy as np
        import matplotlib.pyplot as plt
        from skimage import io
```

Read the image series into the computer. The first frame is named I1 and the next frame is called I2.

```
In [2]: seq1 = {'I1': io.imread('data/image/seq1/frame1.png', as_grey=True),
                'I2': io.imread('data/image/seq1/frame3.png', as_grey=True),
                'U' : np.loadtxt('data/flow/seq1/flow3.u', dtype='double', delimiter=','),
                'V' : np.loadtxt('data/flow/seq1/flow3.v', dtype='double', delimiter=',')}

        rubic = {'I1':io.imread('data/rubic/rubic.0.png', as_grey=True),
                 'I2':io.imread('data/rubic/rubic.5.png', as_grey=True)}

        sphere= {'I1': io.imread('data/sphere/sphere.1.png', as_grey=True),
                 'I2': io.imread('data/sphere/sphere.3.png', as_grey=True)}
```

Define a function called correlation_each_grid() to extract the patch, set the search region and calculate the correlation between two frames. Firstly, some parameters such as image shape are given to the paramters.

Then the patch should be extracted from the first frame which is called I1. Some conditions should be satisfied due to the image got the boundaries and the patch cannot go beyond these boundaries.

The search region is based on the expand size and the patch with several conditions as well.

Finally, the correlation is calculated using correlate2d(region, patch, 'valid') which is a build-in function for computing the corelation between two matrices. The location of the maximum correlation is obtained afterwards by np.unravel_index(correlation.argmax(), correlation.shape).

```
In [7]: def correlation_each_grid(I1, I2, x, y, width, region_expand):
            h= I1.shape[0]
            w= I1.shape[1]

            x = int(x)
            y = int(y)

            half_width = int(np.floor(width/2))


        # patch construction
            if (y-half_width)> 0:
                p_x_up=y-half_width
            else:
                p_x_up=0

            if (y+half_width+1) < h:
                p_x_down=y+half_width+1
            else:
                p_x_down=h

            if (x-half_width)> 0:
                p_y_left=x-half_width
            else:
                p_y_left=0

            if (x+half_width+1) < w:
                p_y_right=x+half_width+1
            else:
                p_y_right=w

            patch = I1[p_x_up:p_x_down, p_y_left:p_y_right]


            # search region
            if y-half_width-region_expand> 0:
                r_x_up=y-half_width-region_expand
        else:
            r_x_up=0

        if y+half_width+1+region_expand < h:
            r_x_down=y+half_width+1+region_expand
        else:
            r_x_down=h

        if x-half_width-region_expand > 0:
            r_y_left = x-half_width-region_expand
        else:
            r_y_left=0

        if x+half_width+1+region_expand < w:
            r_y_right=x+half_width+1+region_expand
        else:
            r_y_right=w

        region = I2[r_x_up:r_x_down, r_y_left:r_y_right]


        # correlation
        correlation = correlate2d(region, patch, 'valid')
        i, j = np.unravel_index(correlation.argmax(), correlation.shape)
        u = j+r_y_left-p_y_left
        v = i+r_x_up-p_x_up

        return (u, v)
```

As shown above, the u and v matrices reflecting the motion is obtained with this function. However, this is only for one block or grid not all grids. Before solving that, to show the optical flow with quiver, a function is defined to draw the vectors on the first frame with the motion vectors to check the results.

```
In [8]: def quiver_drawing(I, x_grid, y_grid, U, V, scale):
            fig, ax = plt.subplots(figsize=(10, 10), dpi=80)
            ax.imshow(I, cmap='gray')
            ax.quiver(x_grid, y_grid, U*scale, V*scale, color='red', angles='xy',
                      scale_units='xy', scale=1)
            ax.set_aspect('equal')
            plt.show()
```

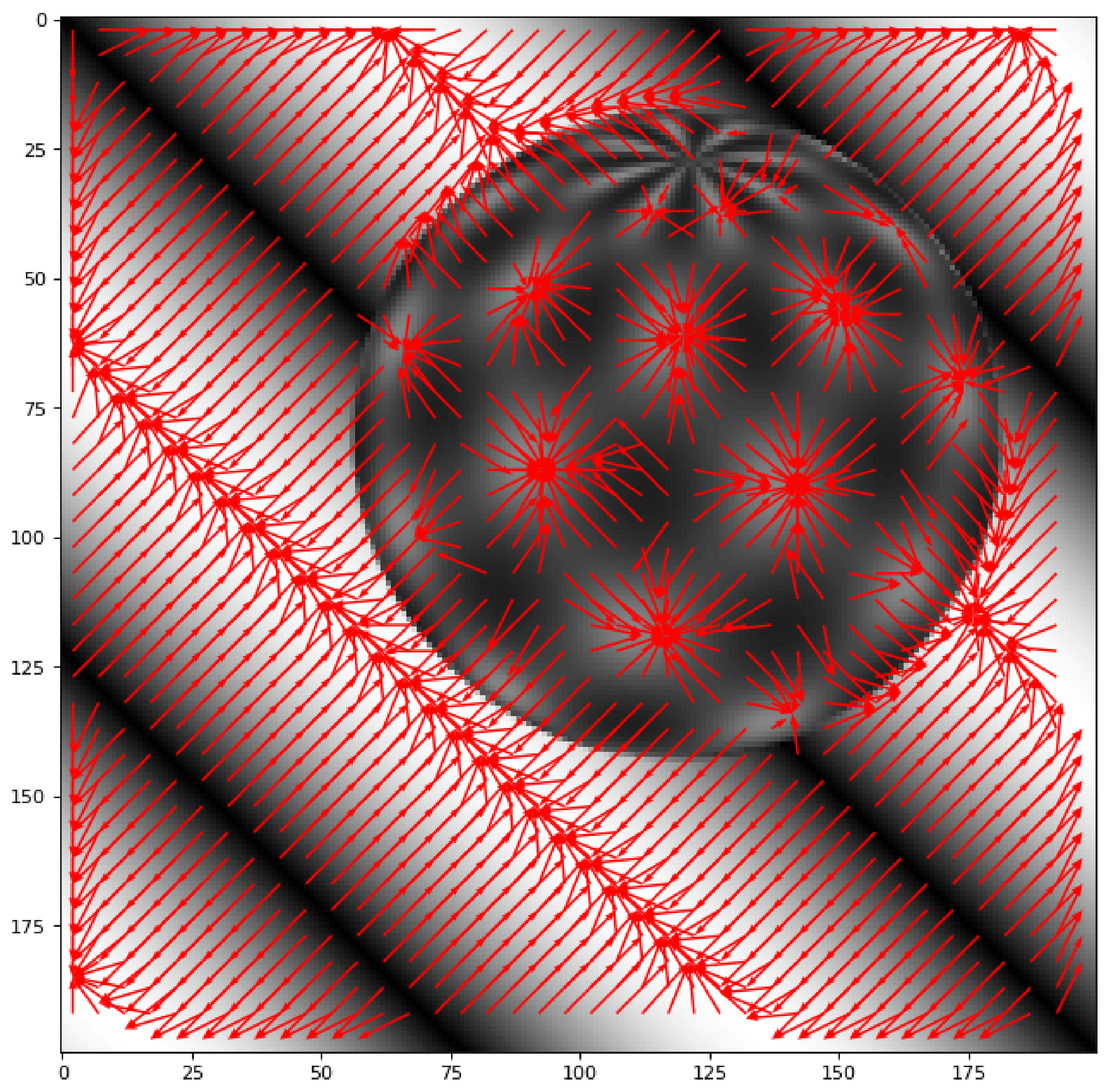
A self-defined function is needed to calculate the motion vectors in different grids. To achieve this goal, each grid is calculated and filled in U and V matrices by iterations shown below.

quiver_drawing() defined above is called to show the motion estimation results which is also called optical flow.

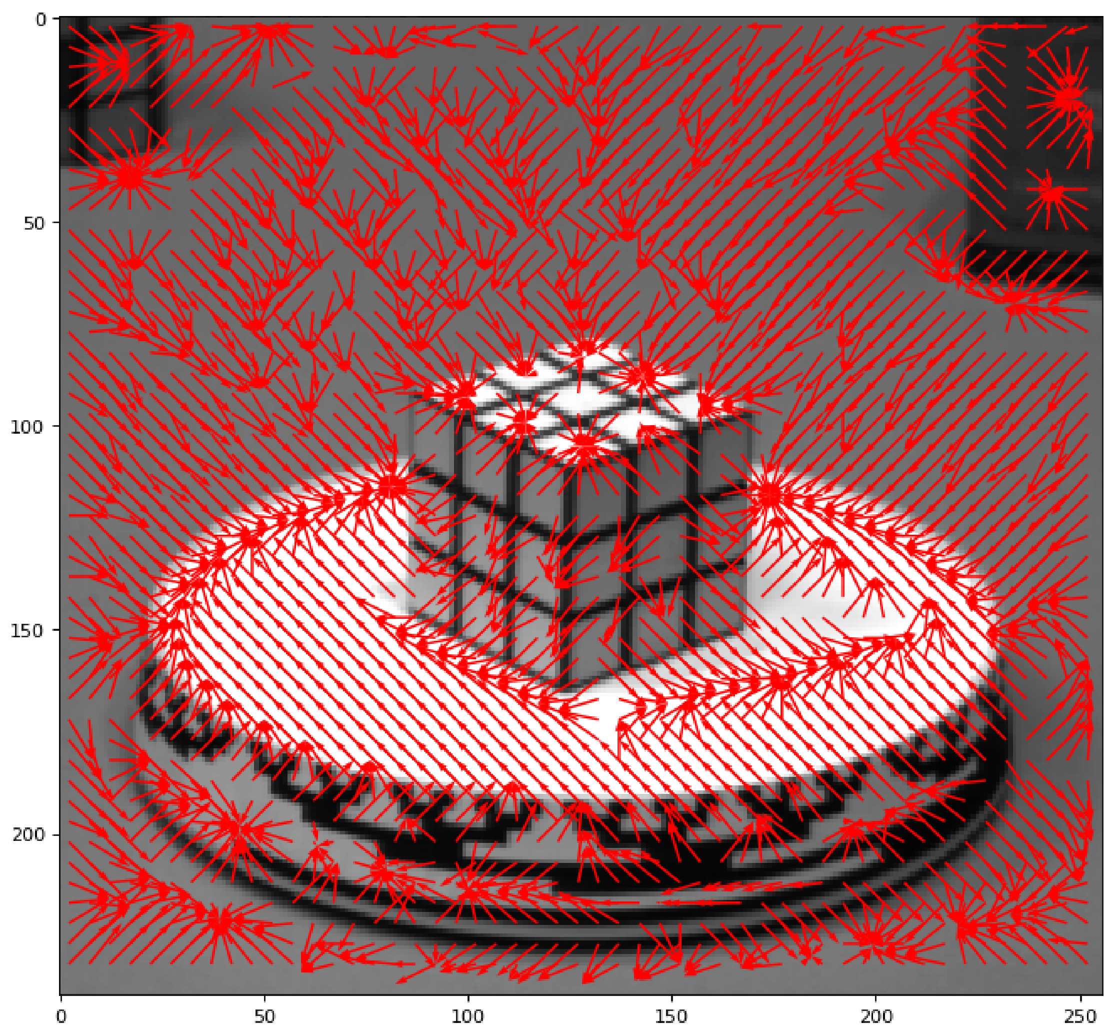
```
In [9]: def optical_flow_estimation(img_series):
            h_img= img_series['I1'].shape[0]
            w_img= img_series['I1'].shape[1]

            grid_size = 5
            width   = 5

            x = np.arange(0, w_img-grid_size, grid_size) + np.floor(grid_size/2);
            y = np.arange(0, h_img-grid_size, grid_size) + np.floor(grid_size/2);
            x_grid, y_grid = np.meshgrid(x,y);

            region_expand = 10

            h_grid = x_grid.shape[0]
            w_grid = x_grid.shape[1]

            U = np.zeros((h_grid, w_grid))
            V = np.zeros((h_grid, w_grid))

            for i in range(0, h_grid):
                for j in range(0, w_grid):
                    u,v =  correlation_each_grid(img_series['I1'], img_series['I2'],
                                                 x_grid[i, j], y_grid[i, j], width,
                                                 region_expand)
                    U[i, j] = u
                    V[i, j] = v


            quiver_drawing(img_series['I1'],x_grid, y_grid, U, V, 1)
```

Call the optical_flow_estimation() with different images and check the result.

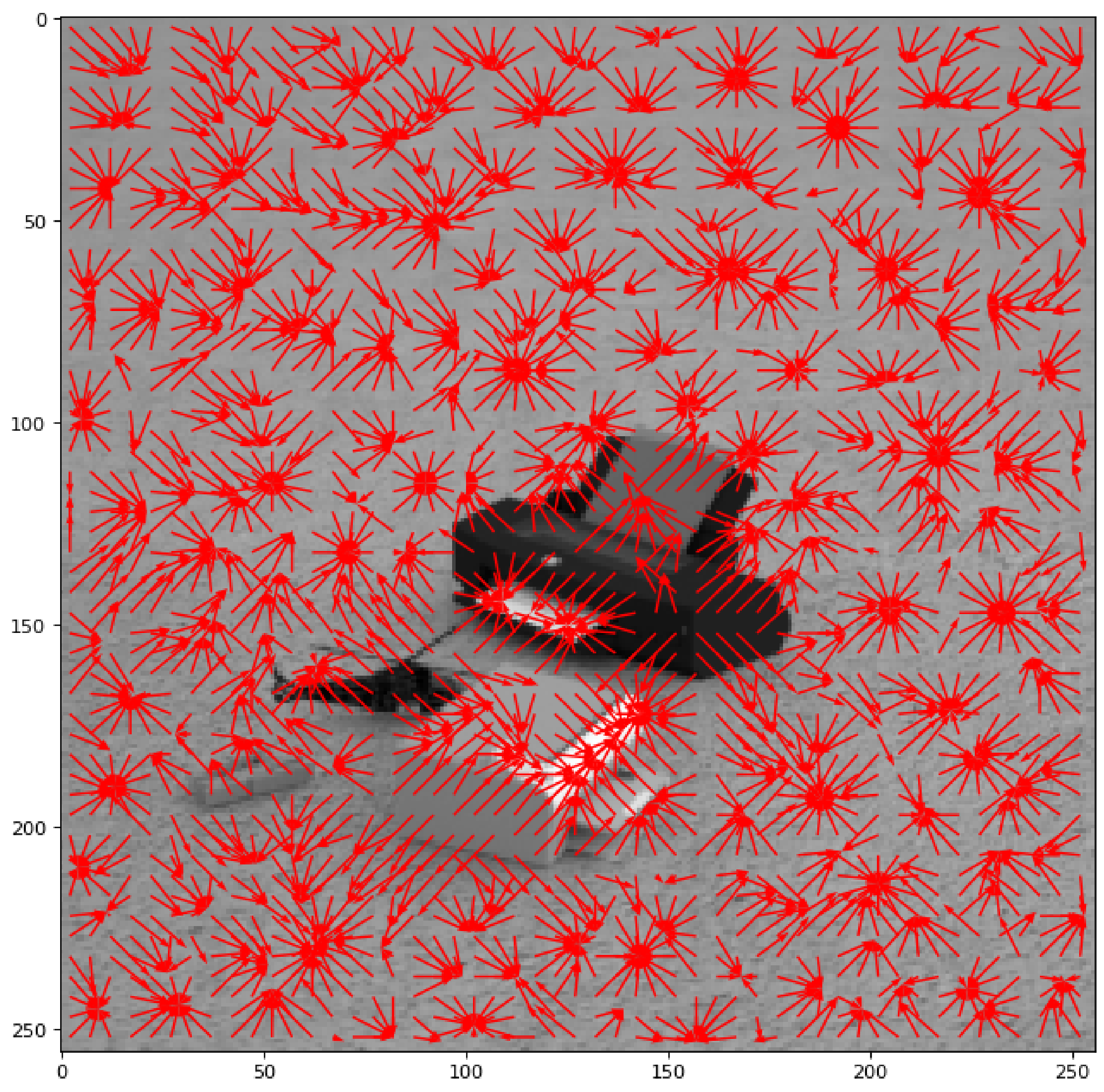
```
In [10]: optical_flow_estimation(sphere)
```



```
In [11]: optical_flow_estimation(rubic)
```



```
In [12]: optical_flow_estimation(seq1)
```



3 Exploration

Another popular method in motion estimation similar to image corelation is called **phase correlation**. Unlike the previous method with calculating the correlation directly based on the image pixel, the phase correlation method measures the movement between two frames from their phases.

To calculate the phase correlation of the grids in respective frames, FFT are first performed on the different frames. And the inverse FFT of the multiplication of one spectrum and the conjugate of the other is their phase correlation. The highest peak is picked out from the phase corelation matrix to estimate the displacement vectors.

To calculate the 2D FFT and inverse FFT, the functions are defined as shown below.

```
In [16]: from scipy.fftpack import dct,idct
```

```
def dct2(image):
    return dct(dct(image.T,norm='ortho').T,norm='ortho')

def idct2(dctmatrix):
    return idct(idct(dctmatrix.T,norm='ortho').T,norm='ortho')
```

Then redefine the correlation_each_grid() function to calculate the correlation based on phase. Basically, apply FFT to the two frames, and use the idea that the inverse FFT of the multiplication of one spectrum and the conjugate of the other is their phase correlation to get the correlation. Then check the highest response of the result.

```
In [17]: def correlation_each_grid(I1, I2, x, y, width, region_expand):
             h= I1.shape[0]
             w= I1.shape[1]

             x = int(x)
             y = int(y)

             half_width = int(np.floor(width/2))

             # patch construction
             if (y-half_width)> 0:
                 p_x_up=y-half_width
             else:
                 p_x_up=0

             if (y+half_width+1) < h:
                 p_x_down=y+half_width+1
             else:
                 p_x_down=h

             if (x-half_width)> 0:
                 p_y_left=x-half_width
             else:
                 p_y_left=0

             if (x+half_width+1) < w:
                 p_y_right=x+half_width+1
             else:
                 p_y_right=w


             dct_img=dct2(I1)
             patch = dct_img[p_x_up:p_x_down, p_y_left:p_y_right]

             # correlation region
             if y-half_width-region_expand> 0:
                 r_x_up=y-half_width+region_expand
```

```
            else:
                r_x_up=0

            if y+half_width+1+region_expand < h:
                r_x_down=y+half_width+1+region_expand
            else:
                r_x_down=h

            if x-half_width-region_expand > 0:
                r_y_left = x-half_width+region_expand
            else:
                r_y_left=0

            if x+half_width+1+region_expand < w:
                r_y_right=x+half_width+1+region_expand
            else:
                r_y_right=w

            dct_region=dct2(I2)
            region = dct_region[r_x_up:r_x_down, r_y_left:r_y_right]

            # correlation
            #correlation = correlate2d(region, patch, 'valid')
            correlation=idct(patch*np.conjugate(region))


            i, j = np.unravel_index(correlation.argmax(), correlation.shape)
            u = j+r_y_left-p_y_left
            v = i+r_x_up-p_x_up

            return (u, v)
```

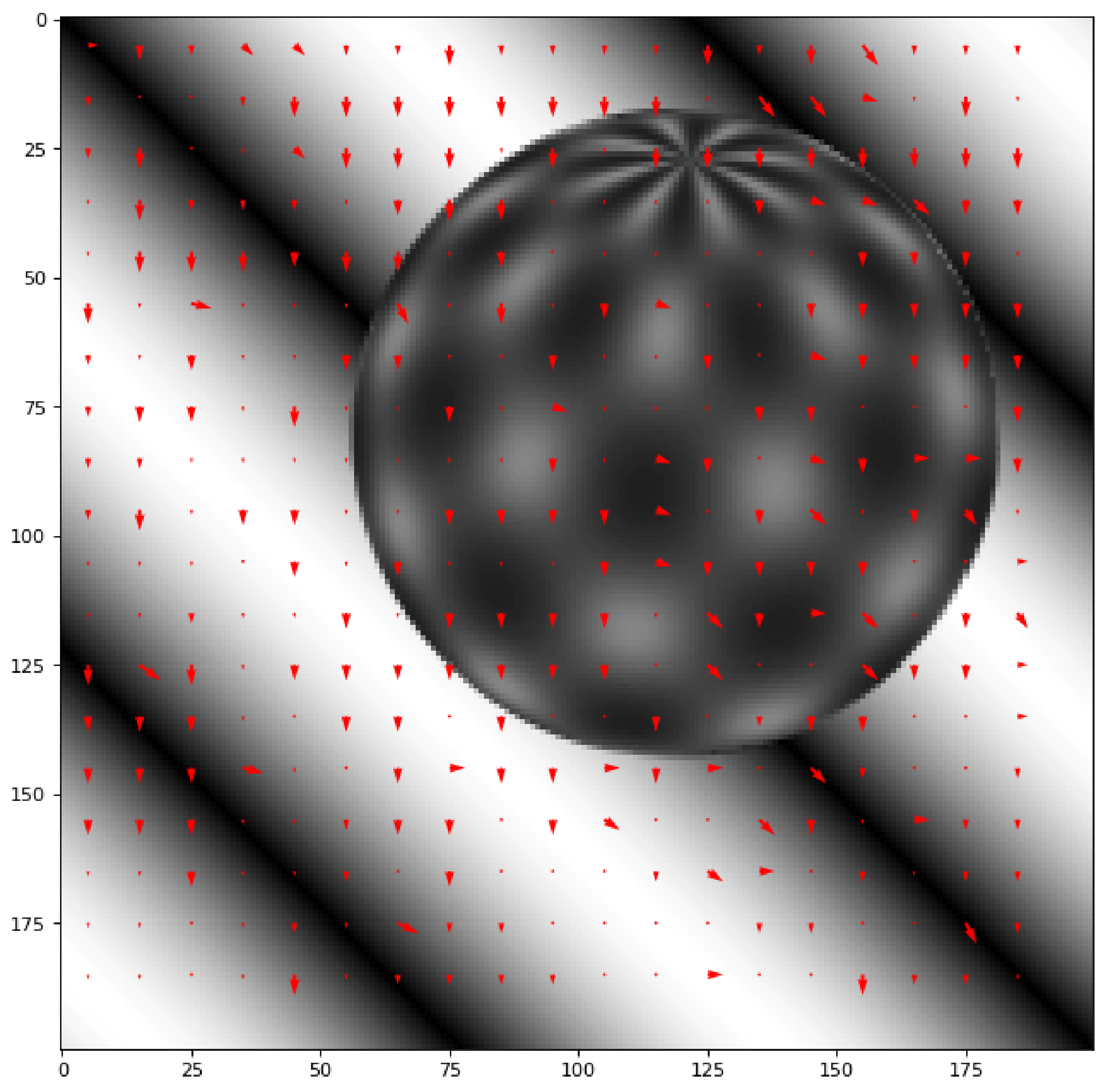
Make the region_expand=0 to ensure patch and region have the same size so that they can be multipiled.

```
In [19]: def optical_flow_estimation(img_series):
             h_img= img_series['I1'].shape[0]
             w_img= img_series['I1'].shape[1]

             grid_size = 10
             width  = 5

             x = np.arange(0, w_img-grid_size, grid_size) + np.floor(grid_size/2);
             y = np.arange(0, h_img-grid_size, grid_size) + np.floor(grid_size/2);
             x_grid, y_grid = np.meshgrid(x,y);

             region_expand = 0
```

```
        h_grid = x_grid.shape[0]
        w_grid = x_grid.shape[1]

        U = np.zeros((h_grid, w_grid))
        V = np.zeros((h_grid, w_grid))

        for i in range(0, h_grid):
            for j in range(0, w_grid):
                u,v =  correlation_each_grid(img_series['I1'],
                                             img_series['I2'],x_grid[i, j], y_grid[i, j],
                                             width, region_expand)
                U[i, j] = u
                V[i, j] = v


        quiver_drawing(img_series['I1'],x_grid, y_grid, U, V, 1)
```

Check the results.

```
In [20]: optical_flow_estimation(sphere)
```

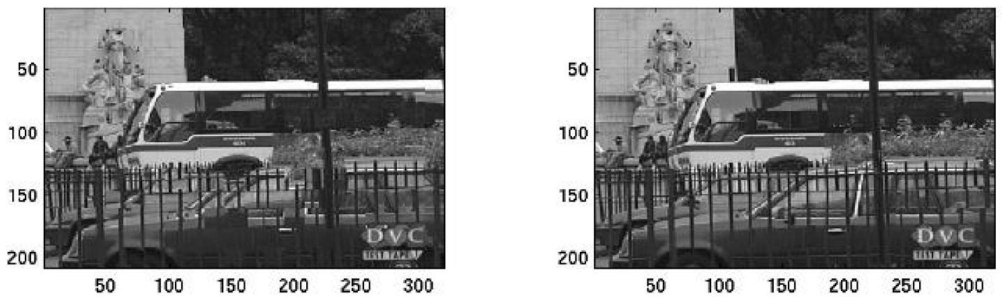


The result seems not really reasonable. However, in one paper called "Phase- Correlation Motion Estimation" written by Yi Liang, the result is robuster than the image correlation. An example he post is to compare these two methods by estimating the bus motion as shown below.

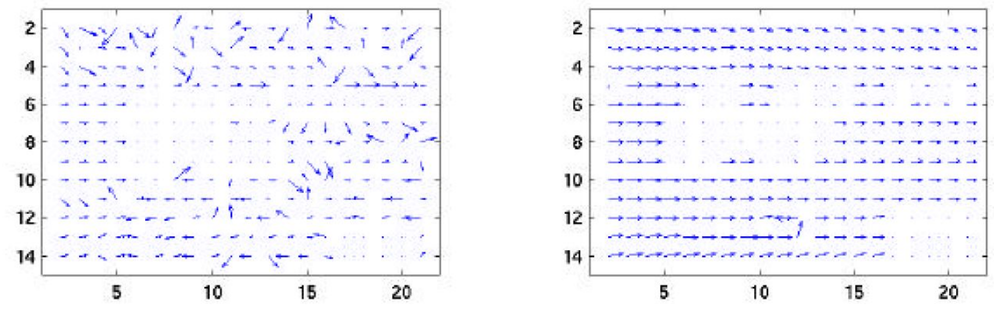
4 Conclusion and Discussion

From the results shown above, we can tell that the correlation method is not reliable in some cases. The optical flow is relatively messy compared to gradient method which will be explained in exercise 2. And the application with phase correlation is not matching the reality as well. After changing the grid in the code, I found that grid size greatly affects the performances.

However, for the bus example within paper "Phase- Correlation Motion Estimation", it demonstrates that phase correlation works better than the image correlation does in the cases of large scale translation motion. Moreover, phase correlation is computationally efficient and it produces much smoother motion field.



bus original images (image correlation vs phase correlation)



bus optical flow field (image correlation vs phase correlation)

5 Reference

https://en.wikipedia.org/wiki/Motion_estimation

https://en.wikipedia.org/wiki/Block-matching_algorithm

https://en.wikipedia.org/wiki/Correlation_and_dependence

Yi Liang, "Phase- Correlation Motion Estimation", EE392J Final Project, Stanford University.