adf

Appendix A

Light-controlled *E. coli*

A.1 Plasmid Construction and Transformation

A.1.1 Plasmid Construction

Here I present the plasmid construction protocol using Gibson assembly. This method can ligate the target gene PR with the plasmid carrier pZE. We first use polymerase chain reaction (PCR) to amplify the gene to sufficient concentration for the ligation. PCR can be easily done with a standard thermo cycler, and all we need to do is to prepare a mixture according to the recipe shown in Table. A.1

The sequence of the primers can be found in Table. **??**, and are synthesized by Eurofins Scientific. After we obtain the more concentrated DNA from PCR, we use Gibson assembly to ligate the plasmid carrier pZE and the target DNA sequence PR. Gibson assembly is done in the following steps:

- prepare the Gibson assembly mix accoring to the Table. A.2.
- add the mix into a 200 μl centrifuge tube, then put the tube into a thermal cycler and run the Gibson assembly program (50-55 ℃ for 1 hour).

Ingredients	Amount (μl)
ddH_2O	37
5X HF buffer (Phusion)	10
dNTP	1
Primer-forward	0.5
Primer-reverse	0.5
Template	small amount of cell culture / 0.5 μl plasmid solution
Phusion polymerase	0.5
Sum	50

Table A.1: PCR recipe using Phusion polymerase

- thaw the chemical competent cells in ice (XL10-Gold, 3-4 min).
- mix cells and plasmid solution for 10 min in ice.
- spread the mix on a selective plate.

Ingredients	Amount (μl)
PCR product	3.3
pZE solution	1.7
2X Gibson assembly mix	5
Sum	10

Table A.2: Gibson assembly recipe.

A.1.2 Extraction and Purification

After growing the chemical competent cells into concentrated liquid cultures, we extract the plasmid from the cells and purify them. We follow the steps in the Zyppy Plasmid Miniprep Kit manual and Zymoclean Gel DNA Recovery Kit manual.

A.1.3 Transformation

Make electrocompetent cells

- Inoculate target cells (*E. coli*, 1 ml) into a test tube and incubate at 37 ℃ for 16-18 hours.
- Prepare a box of ice and put 10% glycerol on it.
- Place the cell culture on ice for 15 minutes (Note: from this step, the cell culture must be on ice all the time).
- Centrifuge at 4000 rpm for 5 min, at 4 ℃ (use the low temperature centrifuge)
- Discard the supernatant, aspirate the residual broth and add 1 ml glycerol. Re-suspend the cells by pipetting up and down.
- Repeat step 4 and 5 for 2 more times (Note: for the last time, add ~ 50 μl glycerol instead of 1 ml).

Electroporation

- Thaw the cell suspensions prepared in the previous step on ice. Place a 1.5 ml centrifuge tube and a 0.1 cm cuvette on ice.
- Add 40 μl cell suspension and 1 μl DNA solution to the 1.5 ml centrifuge tube.
- Set the electroporator mode to Ec1.
- Transfer the mixture of cells and DNA to the cuvette and tap it to make the cell suspension go to the bottom of the cuvette.
- Put the cuvette into the electroporator, and press the pulse button once.

- Remove the cuvette from the electroporator and spread the cell suspension on a selective plate.

A.2 DNA Sequences

A.2.1 Proteorhodopsin (PR)

```
   1 ATGAAGTTGT TGTTGATCTT GGGATCTGTG ATTGCACTTC CGACGTTCGC TGCCGGCGGC
  61 GGGGATTTGG ACGCATCAGA CTACACAGGG GTTTCGTTTT GGCTGGTCAC AGCCGCGCTG
 121 TTAGCGTCCA CCGTCTTTTT TTTCGTTGAA CGGGATAGAG TCTCAGCTAA GTGGAAGACA
 181 TCGTTGACCG TCTCAGGCTT GGTTACCGGC ATTGCCTTCT GGCATTATAT GTACATGCGT
 241 GGTGTCTGGA TTGAAACGGG TGACAGCCCG ACGGTGTTCC GTTATATCGA CTGGCTTTTA
 301 ACCGTTCCCC TTCTGATTTG TGAGTTTTAT TTAATATTGG CGGCAGCAAC GAATGTGGCC
 361 GGTTCACTGT TCAAGAAGCT TCTTGTAGGA AGTTTAGTTA TGTTGGTTTT CGGCTACATG
 421 GGAGAGGCAG GGATAATGGC GGCCTGGCCG GCGTTCATAA TTGGTTGCTT GGCTTGGGTG
 481 TACATGATCT ACGAGCTGTG GGCAGGAGAA GGCAAGTCTG CGTGCAACAC AGCATCGCCA
 541 GCAGTTCAAT CCGCATATAA TACGATGATG TATATAATTA TCTTTGGTTG GGCAATTTAC
 601 CCGGTCGGAT ACTTCACCGG CTATCTTATG GGCGACGGGG GCTCTGCCTT GAACTTGAAT
 661 CTTATATATA ACCTGGCCGA TTTCGTGAAC AAGATTTTGT TTGGACTTAT AATATGGAAC
 721 GTAGCCGTGA AAGAGTCATC GAACGCATAA
```

A.2.2 pZE-PR

```
   1 AGGCGTATCA CGAGGCCCTT TCGTCTTCAC CTCGAGAATT GTGAGCGGAT AACAATTGAC
  61 ATTGTGAGCG GATAACAAGA TACTGAGCAC ATCAGCAGGA CGCACTGACC GAATTCATTA
 121 AAGAGGAGAA AGGTACCATG AAGTTGTTGT TGATCTTGGG ATCTGTGATT GCACTTCCGA
 181 CGTTCGCTGC CGGCGGCGGG GATTTGGACG CATCAGACTA CACAGGGGTT TCGTTTTGGC
 241 TGGTCACAGC CGCGCTGTTA GCGTCCACCG TCTTTTTTTT CGTTGAACGG GATAGAGTCT
```

```
 301 CAGCTAAGTG GAAGACATCG TTGACCGTCT CAGGCTTGGT TACCGGCATT GCCTTCTGGC
 361 ATTATATGTA CATGCGTGGT GTCTGGATTG AAACGGGTGA CAGCCCGACG GTGTTCCGTT
 421 ATATCGACTG GCTTTTAACC GTTCCCCTTC TGATTTGTGA GTTTTATTTA ATATTGGCGG
 481 CAGCAACGAA TGTGGCCGGT TCACTGTTCA AGAAGCTTCT TGTAGGAAGT TTAGTTATGT
 541 TGGTTTTCGG CTACATGGGA GAGGCAGGGA TAATGGCGGC CTGGCCGGCG TTCATAATTG
 601 GTTGCTTGGC TTGGGTGTAC ATGATCTACG AGCTGTGGGC AGGAGAAGGC AAGTCTGCGT
 661 GCAACACAGC ATCGCCAGCA GTTCAATCCG CATATAATAC GATGATGTAT ATAATTATCT
 721 TTGGTTGGGC AATTTACCCG GTCGGATACT TCACCGGCTA TCTTATGGGC GACGGGGGCT
 781 CTGCCTTGAA CTTGAATCTT ATATATAACC TGGCCGATTT CGTGAACAAG ATTTTGTTTG
 841 GACTTATAAT ATGGAACGTA GCCGTGAAAG AGTCATCGAA CGCATAATCT AGAGGCATCA
 901 AATAAAACGA AAGGCTCAGT CGAAAGACTG GGCCTTTCGT TTTATCTGTT GTTTGTCGGT
 961 GAACGCTCTC CTGAGTAGGA CAAATCCGCC GCCCTAGACC TAGGCGTTCG GCTGCGGCGA
1021 GCGGTATCAG CTCACTCAAA GGCGGTAATA CGGTTATCCA CAGAATCAGG GGATAACGCA
1081 GGAAAGAACA TGTGAGCAAA AGGCCAGCAA AAGGCCAGGA ACCGTAAAAA GGCCGCGTTG
1141 CTGGCGTTTT TCCATAGGCT CCGCCCCCCT GACGAGCATC ACAAAAATCG ACGCTCAAGT
1201 CAGAGGTGGC GAAACCCGAC AGGACTATAA AGATACCAGG CGTTTCCCCC TGGAAGCTCC
1261 CTCGTGCGCT CTCCTGTTCC GACCCTGCCG CTTACCGGAT ACCTGTCCGC CTTTCTCCCT
1321 TCGGGAAGCG TGGCGCTTTC TCAATGCTCA CGCTGTAGGT ATCTCAGTTC GGTGTAGGTC
1381 GTTCGCTCCA AGCTGGGCTG TGTGCACGAA CCCCCCGTTC AGCCCGACCG CTGCGCCTTA
1441 TCCGGTAACT ATCGTCTTGA GTCCAACCCG GTAAGACACG ACTTATCGCC ACTGGCAGCA
1501 GCCACTGGTA ACAGGATTAG CAGAGCGAGG TATGTAGGCG GTGCTACAGA GTTCTTGAAG
1561 TGGTGGCCTA ACTACGGCTA CACTAGAAGG ACAGTATTTG GTATCTGCGC TCTGCTGAAG
1621 CCAGTTACCT TCGGAAAAAG AGTTGGTAGC TCTTGATCCG GCAAACAAAC CACCGCTGGT
1681 AGCGGTGGTT TTTTTGTTTG CAAGCAGCAG ATTACGCGCA GAAAAAAAGG ATCTCAAGAA
1741 GATCCTTTGA TCTTTTCTAC GGGGTCTGAC GCTCAGTGGA ACGAAAACTC ACGTTAAGGG
1801 ATTTTGGTCA TGACTAGTGC TTGGATTCTC ACCAATAAAA AACGCCCGGC GGCAACCGAG
```

1861 CGTTCTGAAC AAATCCAGAT GGAGTTCTGA GGTCATTACT GGATCTATCA ACAGGAGTCC

1921 AAGCGAGCTC TCACTGCCCG CTTTCCAGTC GGGAAACCTG TCGTGCCAGC TGCATTAATG

1981 AATCGGCCAA CGCGCGGGGA GAGGCGGTTT GCGTATTGGG CGCCAGGGTG GTTTTTCTTT

2041 TCACCAGTGA GACGGGCAAC AGCTGATTGC CCTTCACCGC CTGGCCCTGA GAGAGTTGCA

2101 GCAAGCGGTC CACGCTGGTT TGCCCCAGCA GGCGAAAATC CTGTTTGATG GTGGTTAACG

2161 GCGGGATATA ACATGAGCTG TCTTCGGTAT CGTCGTATCC CACTACCGAG ATATCCGCAC

2221 CAACGCGCAG CCCGGACTCG GTAATGGCGC GCATTGCGCC CAGCGCCATC TGATCGTTGG

2281 CAACCAGCAT CGCAGTGGGA ACGATGCCCT CATTCAGCAT TTGCATGGTT TGTTGAAAAC

2341 CGGACATGGC ACTCCAGTCG CCTTCCCGTT CCGCTATCGG CTGAATTTGA TTGCGAGTGA

2401 GATATTTATG CCAGCCAGCC AGACGCAGAC GCGCCGAGAC AGAACTTAAT GGGCCCGCTA

2461 ACAGCGCGAT TTGCTGGTGA CCCAATGCGA CCAGATGCTC CACGCCCAGT CGCGTACCGT

2521 CTTCATGGGA GAAAATAATA CTGTTGATGG GTGTCTGGTC AGAGACATCA AGAAATAACG

2581 CCGGAACATT AGTGCAGGCA GCTTCCACAG CAATGGCATC CTGGTCATCC AGCGGATAGT

2641 TAATGATCAG CCCACTGACG CGTTGCGCGA GAAGATTGTG CACCGCCGCT TTACAGGCTT

2701 CGACGCCGCT TCGTTCTACC ATCGACACCA CCACGCTGGC ACCCAGTTGA TCGGCGCGAG

2761 ATTTAATCGC CGCGACAATT TGCGACGGCG CGTGCAGGGC CAGACTGGAG GTGGCAACGC

2821 CAATCAGCAA CGACTGTTTG CCCGCCAGTT GTTGTGCCAC GCGGTTGGGA ATGTAATTCA

2881 GCTCCGCCAT CGCCGCTTCC ACTTTTTCCC GCGTTTTCGC AGAAACGTGG CTGGCCTGGT

2941 TCACCACGCG GGAAACGGTC TGATAAGAGA CACCGGCATA CTCTGCGACA TCGTATAACG

3001 TTACTGGTTT CATGGTATAT CTCCTTCGAG CTCGTAAACT TGGTCTGACA GTTACCAATG

3061 CTTAATCAGT GAGGCACCTA TCTCAGCGAT CTGTCTATTT CGTTCATCCA TAGTTGCCTG

3121 ACTCCCCGTC GTGTAGATAA CTACGATACG GGAGGGCTTA CCATCTGGCC CCAGTGCTGC

3181 AATGATACCG CGAGACCCAC GCTCACCGGC TCCAGATTTA TCAGCAATAA ACCAGCCAGC

3241 CGGAAGGGCC GAGCGCAGAA GTGGTCCTGC AACTTTATCC GCCTCCATCC AGTCTATTAA

3301 TTGTTGCCGG GAAGCTAGAG TAAGTAGTTC GCCAGTTAAT AGTTTGCGCA ACGTTGTTGC

3361 CATTGCTACA GGCATCGTGG TGTCACGCTC GTCGTTTGGT ATGGCTTCAT TCAGCTCCGG

```
3421 TTCCCAACGA TCAAGGCGAG TTACATGATC CCCCATGTTG TGCAAAAAAG CGGTTAGCTC
3481 CTTCGGTCCT CCGATCGTTG TCAGAAGTAA GTTGGCCGCA GTGTTATCAC TCATGGTTAT
3541 GGCAGCACTG CATAATTCTC TTACTGTCAT GCCATCCGTA AGATGCTTTT CTGTGACTGG
3601 TGAGTACTCA ACCAAGTCAT TCTGAGAATA GTGTATGCGG CGACCGAGTT GCTCTTGCCC
3661 GGCGTCAATA CGGGATAATA CCGCGCCACA TAGCAGAACT TTAAAAGTGC TCATCATTGG
3721 AAAACGTTCT TCGGGGCGAA AACTCTCAAG GATCTTACCG CTGTTGAGAT CCAGTTCGAT
3781 GTAACCCACT CGTGCACCCA ACTGATCTTC AGCATCTTTT ACTTTCACCA GCGTTTCTGG
3841 GTGAGCAAAA ACAGGAAGGC AAAATGCCGC AAAAAAGGGA ATAAGGGCGA CACGGAAATG
3901 TTGAATACTC ATACTCTTCC TTTTTCAATA TTATTGAAGC ATTTATCAGG GTTATTGTCT
3961 CATGAGCGGA TACATATTTG AATGTATTTA GAAAAATAAA CAAATAGGGG TTCCGCGCAC
4021 ATTTCCCCGA AAAGTGCCAC CTGACGTCTA AGAAACCATT ATTATCATGA CATTAACCTA
4081 TAAAAAT
```

A.2.3 An Illustration of the Plasmid Structure

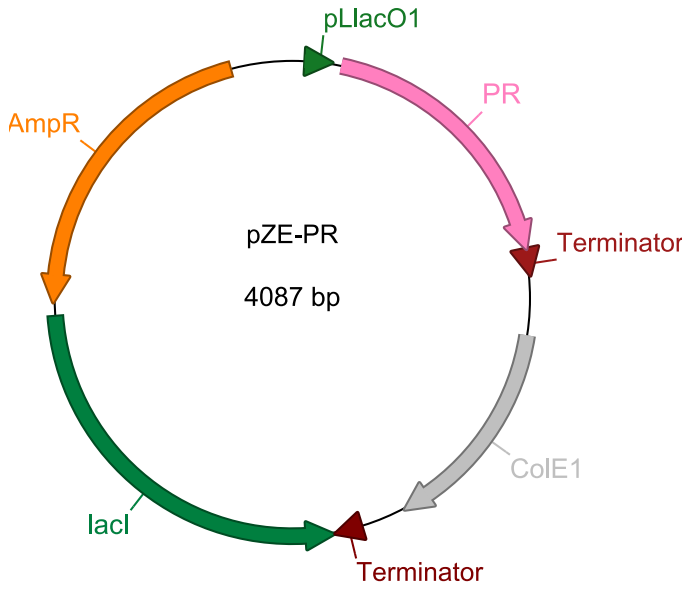

Figure A.1: An illustration of pZE-PR plasmid.

Appendix B

Image Analysis Code

B.1 Code Vectorization

B.1.1 Vectorized Code for Spatial Correlation Function

```
def corrS(X, Y, U, V):
    row, col = X.shape
    vsqrt = (U ** 2 + V ** 2) ** 0.5
    U = U - U.mean()
    V = V - V.mean()
    Ax = U / vsqrt
    Ay = V / vsqrt
    CA = np.ones(X.shape)
    CV = np.ones(X.shape)
    for xin in range(0, col):
        for yin in range(0, row):
            if xin != 0 or yin != 0:
                CA[yin, xin] = (Ax[0:row-yin, 0:col-xin] * Ax[yin:row, xin:col] + Ay[0:r
                CV[yin, xin] = (U[0:row-yin, 0:col-xin] * U[yin:row, xin:col] + V[0:row-
    return CA, CV
```

B.1.2 Non-vectorized Code for Spatial Correlation Function

```
def corrS(X, Y, U, V):
    row, col = X.shape
    vsq = 0
    CA = np.zeros((row, col))
    CV = np.zeros((row, col))
    for i in range(0, row):
        for j in  range(0, col):
            vsq += U[i, j]**2 + V[i, j]**2
    for xin in range(0, col):
        for yin in range(0, row):
            count = 0
            CAt = 0
            CVt = 0
            for i in range(0, col-xin):
                for j in range(0, row-yin):
                    ua = U[j, i]
                    va = V[j, i]
                    ub = U[j+yin, i+xin]
                    vb = V[j+yin, i+xin]
                    CAt += (ua*ub+va*vb)/((ua**2+va**2)*(ub**2+vb**2))**.5
                    CVt += ua*ub + va*vb
                    count += 1
            CA[yin, xin] = CAt / count
            CV[yin, xin] = CVt / vsq
    return CA, CV
```

B.1.3 Performance Comparison

We notice that the vectorized code has two less nested `for` loops compared to the non-vectorized code. As a result, the vectorized one runs much faster for the same task. To quantify this performance difference, we perform the spatial correlation function calculation using both code on the same velocity field, shown in Fig. B.1a. The times

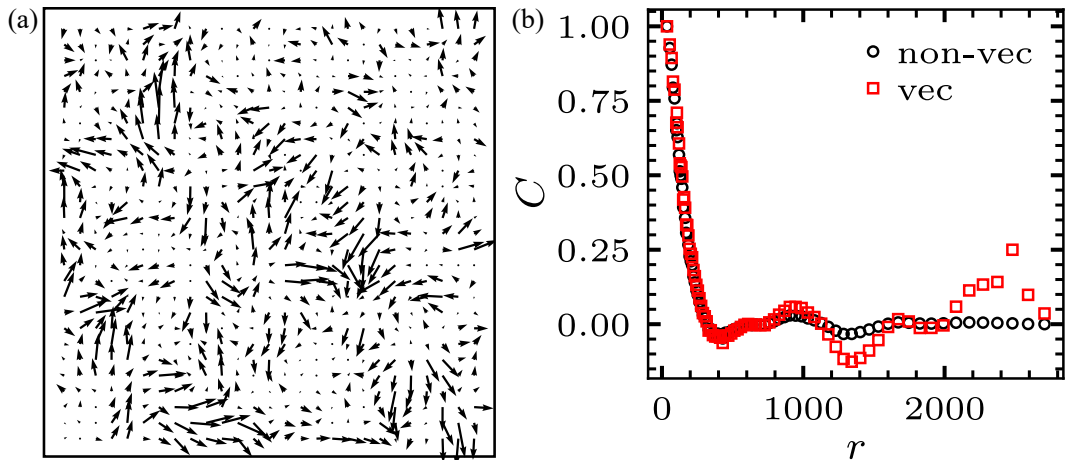

Figure B.1: **Compare the performance of vectorized and non-vectorized code.** (a) Sample velocity field. (b) Velocity correlation functions obtained from the vectorized and non-vectorized code.

taken for the two functions are:

- `Vectorized code:  0.84 s`
- `Non-vectorized code:  52.06 s`

The result is shown in Fig. B.1b. Although in the large r regime, two methods show descrepancies, in the meaningful small r regime, two methods give exactly the same results.

B.2 Energy Spectrum Calculation

The following code is used for calculating the energy spectra in Chap. **??**.

```
1 def compute_energy_density(pivData, d=25*0.33, MPP=0.33):
2     """
3     Compute kinetic energy density in k space from piv data. The unit of the return val
4     where [velocity] is the unit of pivData, and [length] is the unit of sample_spacing
```

```
    Note, the default value of sampling_spacing does not have any significance. It is j
    and should be set with caution when a different magnification and PIV are used.

    Args:
    pivData -- piv data
    d -- sample spacing
    MPP -- microns per pixel

    Returns:
    E -- kinetic energy field in k space
    """

    row = len(pivData.y.drop_duplicates())
    col = len(pivData.x.drop_duplicates())
    U = np.array(pivData.u).reshape((row, col)) * MPP
    V = np.array(pivData.v).reshape((row, col)) * MPP

    u_fft = np.fft.fft2(U) * d * d
    v_fft = np.fft.fft2(V) * d * d

    E = (u_fft * u_fft.conjugate() + v_fft * v_fft.conjugate()) / 2

    return E

def compute_wavenumber_field(shape, d):
    """
    Compute the wave number field Kx and Ky, and magnitude field k.
    Note that this function works for even higher dimensional shape.

    Args:
    shape -- shape of the velocity field and velocity fft field, tuple
    d -- sample spacing. This is the distance between adjacent samples, for example, ve
        The resulting frequency space has the unit which is inverse of the unit of d. Th

    Returns:
    k -- wavenumber magnitude field
    K -- wavenumber fields in given dimensions
    """
```

```
43
44      for num, length in enumerate(shape):
45          kx = np.fft.fftfreq(length, d=d)
46          if num == 0:
47              k = (kx,)
48          else:
49              k += (kx,)
50
51      K = np.meshgrid(*k, indexing='ij')
52
53      for num, k1 in enumerate(K):
54          if num == 0:
55              ksq = k1 ** 2
56          else:
57              ksq += k1 ** 2
58
59      k_mag = ksq ** 0.5 * 2 * np.pi
60
61      return k_mag, K
62
63  def energy_spectrum(pivData, d=25*0.33):
64      """
65      Compute energy spectrum (E vs k) from pivData.
66
67      Args:
68      pivData -- piv data
69      d -- sample spacing. This is the distance between adjacent samples, for example, ve
70          The resulting frequency space has the unit which is inverse of the unit of d. T
71
72      Returns:
73      es -- energy spectrum, DataFrame (k, E)
74      """
75
76      row = len(pivData.y.drop_duplicates())
77      col = len(pivData.x.drop_duplicates())
78
79      E = compute_energy_density(pivData, d) / (row * d * col * d)
80      k, K = compute_wavenumber_field(E.shape, d)
```

```
81
82      ind = np.argsort(k.flatten())
83      k_plot = k.flatten()[ind]
84      E_plot = E.real.flatten()[ind]
85
86      es = pd.DataFrame(data={'k': k_plot, 'E': E_plot})
87
88      return es
```

B.3 Cross-Correlation Based Particle Detecting Method

This is designed for tracking colloidal chains. We show here the basic working principle and the code here. The advantage of this method is that it does not require the feature to be very bright or dark, like traditional particle detecting methods. As long as the features share similar characteristics, i.e. spatial patterns, this method can work well.

B.3.1 Working principle

From raw images, we crop out a single feature which we are looking for as a target mask, as shown in Fig. B.2a inset. Then, we do a 2D correlation between the target mask and an image (Fig. B.2a). The resulting correlation map is shown in Fig. B.2b, where bright pixels stand for high correlation and dark pixels stand for low correlation. The last step is to look for the pixel intensity peaks in the correlation map, as in traditional particle detecting methods.

B.3.2 Code

```
1  import numpy as np
2  from scipy.signal import medfilt2d, convolve2d, fftconvolve
3  from math import exp
4
```

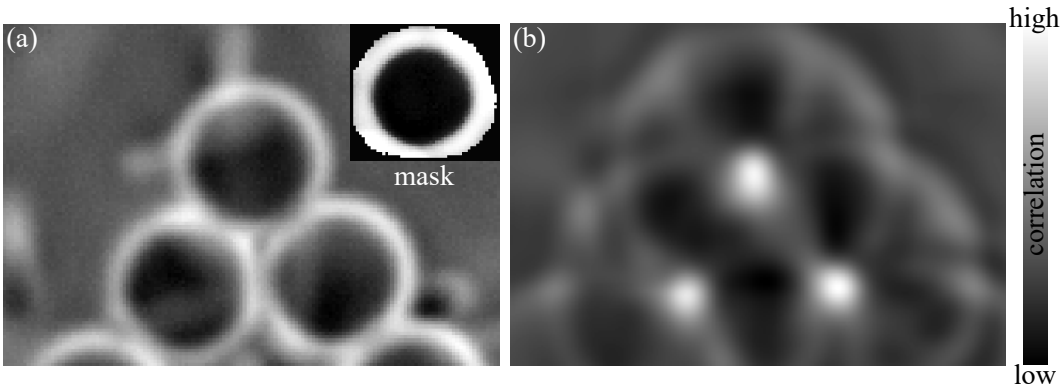

Figure B.2: **Cross-correlation based particle detecting method.** (a) A sample image of colloidal particles. Inset: a mask produced by cropping the raw image and binarize. (b) The cross-correlation map.

```
5  def normxcorr2(template, image, mode="full"):
6      if np.ndim(template) > np.ndim(image) or \
7              len([i for i in range(np.ndim(template)) if template.shape[i] > image.shape
8          print("normxcorr2: TEMPLATE larger than IMG. Arguments may be swapped.")
9      template = template - np.mean(template)
10     image = image - np.mean(image)
11     a1 = np.ones(template.shape)
12     ar = np.flipud(np.fliplr(template))
13     out = fftconvolve(image, ar.conj(), mode=mode)
14     image = fftconvolve(np.square(image), a1, mode=mode) - \
15             np.square(fftconvolve(image, a1, mode=mode)) / (np.prod(template.shape))
16     image[np.where(image < 0)] = 0
17     template = np.sum(np.square(template))
18     out = out / np.sqrt(image * template)
19     out[np.where(np.logical_not(np.isfinite(out)))] = 0
20     return -out
21
22 def matlab_style_gauss2D(shape=(3,3),sigma=0.5):
23     m,n = [(ss-1.)/2. for ss in shape]
24     y,x = np.ogrid[-m:m+1,-n:n+1]
25     h = np.exp( -(x*x + y*y) / (2.*sigma*sigma) )
26     h[ h < np.finfo(h.dtype).eps*h.max() ] = 0
```

```
    sumh = h.sum()
    if sumh != 0:
        h /= sumh
    return h

def FastPeakFind(data):
    if str(data.dtype) != 'float32':
        data = data.astype('float32')
    mf = medfilt2d(data, kernel_size=3)
    mf = mf.astype('float32')
    thres = max(min(np.amax(mf,axis=0)), min(np.amax(mf,axis=1)))
    filt = matlab_style_gauss2D()
    conv = convolve2d(mf, filt, mode='same')
    w_idx = conv > thres
    bw = conv.copy()
    bw[w_idx] = 1
    bw[~w_idx] = 0
    thresholded = np.multiply(bw, conv)
    edg = 3
    shape = data.shape
    idx = np.nonzero(thresholded[edg-1: shape[0]-edg-1, edg-1: shape[1]-edg-1])
    idx = np.transpose(idx)
    cent = []
    for xy in idx:
        x = xy[0]
        y = xy[1]
        if thresholded[x, y] >= thresholded[x-1, y-1] and \
            thresholded[x, y] > thresholded[x-1, y] and \
            thresholded[x, y] >= thresholded[x-1, y+1] and \
            thresholded[x, y] > thresholded[x, y-1] and \
            thresholded[x, y] > thresholded[x, y+1] and \
            thresholded[x, y] >= thresholded[x+1, y-1] and \
            thresholded[x, y] > thresholded[x+1, y] and \
            thresholded[x, y] >= thresholded[x+1, y+1]:
            cent.append(xy)
    cent = np.asarray(cent).transpose()
    return cent

```

```
65 def maxk(array, num_max):
66     array = np.asarray(array)
67     length = array.size
68     array = array.reshape((1, length))
69     idx = np.argsort(array)
70     idx2 = np.flip(idx)
71     return idx2[0, 0:num_max]
72 
73 def gauss1(x,a,x0,sigma):
74     return a*exp(-(x-x0)**2/(2*sigma**2))
75 
76 def track_spheres(img, mask, num_particles):
77     def gauss1(x,a,x0,sigma):
78         return a*exp(-(x-x0)**2/(2*sigma**2))
79     corr = normxcorr2(mask, img, mode='same')
80     cent = FastPeakFind(corr)
81     peaks = corr[cent[0], cent[1]]
82     ind = maxk(peaks, num_particles)
83     max_coor_tmp = cent[:, ind]
84     max_coor = max_coor_tmp.astype('float32')
85     pk_value = peaks[ind]
86     for num in range(0, num_particles):
87         x = max_coor_tmp[0, num]
88         y = max_coor_tmp[1, num]
89         fitx1 = np.asarray(range(x-7, x+8))
90         fity1 = np.asarray(corr[range(x-7, x+8), y])
91         popt, pcov = curve_fit(gauss1, fitx1, fity1, p0=[1, x, 3])
92         max_coor[0, num] = popt[1]
93         fitx2 = np.asarray(range(y-7, y+8))
94         fity2 = np.asarray(corr[x, range(y-7, y+8)])
95         popt,pcov = curve_fit(gauss1, fitx2, fity2, p0=[1, y, 3])
96         max_coor[1, num] = popt[1]
97     return max_coor, pk_value
```

B.4 Fourier Transform Based Orientation Analysis

B.5 Density Fluctuation Calculation