GooseEYE can be used from C++ or as a Python module. Please consider the theory and examples below that describe briefly the different measurements and their interpretation, accompanied by a C++ and a Python example. Since this is a research code, the code itself is the best reference.
2-point probability

1.1 Theory

This measure determines the typical distance over which two points (pixels, voxels, . . . ) are related to each other. This is best understood by considering a binary 2D image, wherein each pixel is either black or white. This is described by the following indicator function, indicating the ‘greyscale’ of a pixel at position $\vec{x}_i$:

$$\mathcal{I}(\vec{x}_i) = \begin{cases} 1 & \text{if } \vec{x}_i \in \text{white} \\ 0 & \text{if } \vec{x}_i \in \text{black} \end{cases}$$

The 2-point probability, $S_2$, is the probability that two points, at a certain distance $\Delta \vec{x}$ are both white. I.e.

$$S_2(\Delta \vec{x}) = P\{\mathcal{I}(\vec{x}) = 1, \mathcal{I}(\vec{x} + \Delta \vec{x}) = 1\}$$

Two limits can directly be identified. If $\Delta \vec{x} = \vec{0}$, $S_2$ is simply the probability that a point is white: the (volume) fraction of white, $\varphi$. I.e.

$$S_2(||\Delta \vec{x}|| = 0) = \varphi$$

The two points are completely uncorrelated if $||\Delta \vec{x}||$ is sufficiently large (i.e. larger than the correlation length). In this case, both a point at $\vec{x}$ and at $\vec{x} + \Delta \vec{x}$ have a probability $\varphi$ to be white, and thus

$$S_2(||\Delta \vec{x}|| \to \infty) = \varphi^2$$

In between these extremes, $S_2$ decays from $\varphi$ towards the asymptotic value of $\varphi^2$.

See also:

1.2 Example

This example is based on a simple, periodic, image comprising circular white inclusions embedded in a black background. The figure shows from left to right: the image, the 2-point probability $S_2$ in two dimensions, and a cross-section of this result in the middle of the region-of-interest along the horizontal axis.

S2.py
S2.cpp

Note: The Python-code can be used for the plotting: The complete code is included in the download. Note that to obtain the same plot one should download and install the matplotlib-styles available in GooseMPL.

Note: All functions make the assumption of the images being periodic. If this assumption is not reasonable be sure to specify the periodic option (that defaults True).

1.2.1 Python

```python
import GooseEYE
import numpy as np

# generate image, extract 'volume-fraction' for plotting
img = GooseEYE.dummy_circles((500, 500))
phi = np.mean(img)

# 2-point probability
S2 = GooseEYE.S2((101, 101), img, img)
```

1.2.2 C++

```cpp
#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) 
   if (!expr) { 
      throw std::runtime_error( 
         std::string(file) + ':' + std::to_string(line) + " : assertion failed (" #expr ") \n\t"); 
   }

int main()
{
   // generate image
   auto I = GooseEYE::dummy_circles((500, 500));
```
1.3 Masked correlation

This function (as most of GooseEYE’s functions) also has the possibility to mask certain pixels (which can be used for example to exclude acquisition artefacts from the measurement). The image’s mask is a binary matrix of exactly the same shape as the image. For each pixel in the mask with value 1, the corresponding pixel in the image is ignored. The normalisation is corrected for the reduced amount of data points, whereby the number of data points is no longer constant over the region-of-interest.

S2_mask.py
S2_mask.cpp

1.4 Ensemble average

To compute the ensemble average of a statistic, one constructs an Ensemble with a certain shape for the region-of-interest, and then adds the result per image to it. Consider the following example.

S2_ensemble.py
S2_ensemble.cpp

Note: An ensemble is used to compute the mean using a selection (ensemble) of relative small measurements. See Wikipedia.

1.4.1 Python

```python
import GooseEYE
import numpy as np

ensemble = GooseEYE.Ensemble((101, 101))
```
for i in range(5):
    img = GooseEYE.dummy_circles((200, 200))
    ensemble.S2(img, img)
S2 = ensemble.result()

1.4.2 C++

#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) \
    if (!(expr)) { \
        throw std::runtime_error( \
            std::string(file) + ':' + std::to_string(line) + \
            " assertion failed (" #expr ") \n	"); \
    }

int main()
{
    GooseEYE::Ensemble ensemble({101, 101});

    for (size_t i = 0; i < 5; ++i) {
        auto I = GooseEYE::dummy_circles({200, 200});
        ensemble.S2(I, I);
    }

    auto S2 = ensemble.result();

    // check against previous versions
    H5Easy::File data("S2_ensemble.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::allclose(S2, H5Easy::load<
        decltype(S2)>(data, "S2")));

    return 0;
}

1.5 Auto-correlation

The the greyscale generalisation of the 2-point probability (for floating-point images (with $0 \leq I(x_i) \leq 1$)) corresponds to:

$$S_2(\Delta \vec{x}) = \frac{1}{N} \sum_{i=1}^{N} I(x_i) I(x_i + \Delta \vec{x}) \equiv I(\vec{x}) \ast I(\vec{x})$$

where the $\ast$ represent the convolution, in this case of $I$ with itself. Along the same arguments as for the 2-point probability, limit values can be obtained. In this case:

$$S_2(\Delta \vec{x} = 0) = \langle I^2 \rangle$$
$$S_2(\Delta \vec{x} \to \infty) = \langle I \rangle^2$$

where the brackets $\langle \ldots \rangle$ denotes the spatial average.
S2_autocorrelation.py
S2_autocorrelation.cpp

See also:
Wikipedia (on correlation in time, while here the correlation is in space)
2-point cluster function

2.1 Theory

If an image consists of isolated clusters (‘islands’ of connected pixels with the same value), the 2-point cluster function can be used to quantify the probability that two points are in the same cluster. It is defined as follows:

\[ C_2(\Delta x) = P\{C(\vec{x}) = C(\vec{x} + \Delta \vec{x}) \neq 0\} \]

whereby \( C \) is an indicator with a unique non-zero index for each cluster.

See also:

2.2 Example

C2.py
C2.cpp

Note: Like for the 2-point correlation, a mask can be used. Similarly, the average can be extended to that of an ensemble of images.

2.2.1 Python
import GooseEYE
import numpy as np

# generate image, extract 'volume-fraction' for plotting
img = GooseEYE.dummy_circles((500, 500))
phi = np.mean(img)

# 2-point probability for comparison
S2 = GooseEYE.S2((101, 101), img, img)

# determine clusters, based on the binary image
C = GooseEYE.clusters(img)

# 2-point cluster function
C2 = GooseEYE.C2((101, 101), C, C)

2.2.2 C++

#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) \
    if (!expr) { \
        throw std::runtime_error( \
            std::string(file) + ':' + std::to_string(line) + " : assertion failed (" #expr " ) \n	"); \
    }

int main()
{
    // generate image
    auto I = GooseEYE::dummy_circles({500, 500});

    // determine clusters, based on the binary image
    auto C = GooseEYE::clusters(I);

    // 2-point cluster function
    auto C2 = GooseEYE::C2({101, 101}, C, C);

    // check against previous versions
    H5Easy::File data("C2.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::all(xt::equal(I, H5Easy::load<decltype(I)>(data, "I"))));
    MYASSERT(xt::all(xt::equal(C, H5Easy::load<decltype(C)>(data, "C"))));
    MYASSERT(xt::allclose(C2, H5Easy::load<decltype(C2)>(data, "C2")));

    return 0;
}
3.1 Theory

The 2-point cluster function has a first order notion of connectedness. To quantify the true connectedness along a path, the lineal path function is used. The lineal path function quantifies the probability that an entire path of pixels connecting $\vec{x}_i$ and $\vec{x}_i + \Delta \vec{x}$ (denoted: $\{\vec{x}_i + \delta\vec{x}_0, \vec{x}_i + \delta\vec{x}_1, ..., \vec{x}_i + \delta\vec{x}_n\}$, where $\delta\vec{x}_0 \equiv \vec{0}$ and $\delta\vec{x}_n \equiv \Delta \vec{x}$) is in the same phase:

$$L(\Delta \vec{x}) = P\{I(\vec{x}) = 1, I(\vec{x} + \delta\vec{x}_1) = 1, ..., I(\vec{x} + \Delta \vec{x}) = 1\}$$

In practice the probability is constructed by starting from each pixel $\vec{x}_i$ for which $I(\vec{x}_i) = 1$ ‘walking’ along a pixel path until the edge of the inclusion is reached at $\vec{x}_i + \delta\vec{x}_j$. The value of $L$ is increased for all the relative positions that have been passed along the path connecting $\vec{0}$ and $\delta\vec{x}_j$. This is then repeated for all possible directions (with each their own path).

Two limit values can again be identified. At zero distance, the volume fraction is again found:

$$L(\Delta \vec{x} = \vec{0}) = \varphi$$

Furthermore it is highly unlikely that a path can be found through the inclusion phase to a relative position very far away. I.e.

$$L(\Delta \vec{x} \to \infty) = 0$$

An important ingredient of the computation of $L$ is thus the choice of the pixel paths. In GooseEYE the paths are constructed between the centre of the region of interest and each of the points on the end of the region of interest. The paths can be computed using different algorithms, illustrated below:

code: pixel_path.py
3.2 Example

L.py
L.cpp

Note: Like for the 2-point correlation, the average can be extended to that of an ensemble of images.

3.2.1 Python

```python
import GooseEYE
import numpy as np

# generate image, extract 'volume-fraction' for plotting
img = GooseEYE.dummy_circles((500, 500))
phi = np.mean(img)

# lineal path function
L = GooseEYE.L((101, 101), img)
```

3.2.2 C++

```cpp
#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>
#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) 
    if (!(expr)) { 
        throw std::runtime_error( 
            std::string(file) + ':' + std::to_string(line) + 
            ": assertion failed (" #expr ") \n\t"; 
    }

int main() 
{
    // generate image
    auto I = GooseEYE::dummy_circles({500, 500});

    // lineal path function
    auto L = GooseEYE::L({101, 101}, I);

    // check against previous versions
    H5Easy::File data("L.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::all(xt::equal(I, H5Easy::load<
        decltype(I)>(data, "I"))));
    MYASSERT(xt::allclose(L, H5Easy::load<
        decltype(L)>(data, "L")));

    return 0;
}
```
CHAPTER 4

Weighted correlation

4.1 Theory

The weighted correlation characterises the average indicator $I$ around high weight factor $W$.

Mathematically the weighted correlation reads

$$P(\Delta \vec{x}) = \frac{\sum_i W(\vec{x}_i) I(\vec{x}_i + \Delta \vec{x})}{\sum_i W(\vec{x}_i)}$$

Additionally pixels can be masked, for instance to ignore $I$ everywhere where $W$ is non-zero. The masked correlation reads

$$P(\Delta \vec{x}) = \frac{\sum_i W(\vec{x}_i) [I(1 - M)](\vec{x}_i + \Delta \vec{x})}{\sum_i W(\vec{x}_i) (1 - M)(\vec{x}_i + \Delta \vec{x})}$$

where all pixels where $M(\vec{x}_i) = 1$ are ignored; all pixels for which $M(\vec{x}_i) = 0$ are considered as normal.

See also:


Note: The notation is short-hand for:

$$P(\Delta \vec{x}) = \frac{\sum_i [W(\vec{x}_i)] [I(\vec{x}_i + \Delta \vec{x})(1 - M)(\vec{x}_i + \Delta \vec{x})]}{\sum_i [W(\vec{x}_i)] [1 - M(\vec{x}_i + \Delta \vec{x})]}$$
4.2 Example

W2.py
W2.cpp

Note: Like for the 2-point correlation, a mask can be used. Similarly, the average can be extended to that of an ensemble of images.

4.2.1 Python

```python
import GooseEYE
import numpy as np

# image + "damage" + correlation
# ------------------------------
# square grid of circles
N = 15
M = 500
row = np.linspace(0, M, N)
col = np.linspace(0, M, N)
row, col = np.meshgrid(row, col, indexing="ij")  # ('indexing' only for comparison with C++ code)
row = row.reshape(-1)
col = col.reshape(-1)
r = float(M) / float(N) / 4.0 * np.ones(N * N)

# random perturbation
row += GooseEYE.random.normal([N * N], 0.0, float(M) / float(N))
col += GooseEYE.random.normal([N * N], 0.0, float(M) / float(N))
r *= GooseEYE.random.random([N * N]) * 2.0 + 0.1

# generate image, extract 'volume-fraction' for plotting
img = GooseEYE.dummy_circles((M, M), np.round(row), np.round(col), np.round(r))
phi = np.mean(img)

# create 'damage' -> right of inclusion
col += 1.1 * r
r *= 0.4
W = GooseEYE.dummy_circles((M, M), np.round(row), np.round(col), np.round(r))
W[img == 1] = 0

# weighted correlation
WI = GooseEYE.W2((101, 101), W, img, fmask=W)

# gray-scale image + correlation
# ------------------------------

# convert to gray-scale image and introduce noise
Igr = np.array(img, copy=True).astype(np.float)
```

(continues on next page)
GooseEYE Documentation

Igr += 0.1 * (2.0 * GooseEYE.random.random(Igr.shape) - 1.0) + 0.1
Igr /= 1.2

# mean intensity (for bounds)
Iav = np.mean(Igr)

# weighted correlation
W1gr = GooseEYE.W2((101, 101), W.astype(np.float), Igr, fmask=W)

4.2.2 C++

#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>
#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line)\
if (!expr) {\
  throw std::runtime_error( \\
    std::string(file) + ':' + std::to_string(line) + \\
    ': assertion failed ("' #expr "\n    "'); \\
  }\
}

int main()
{
  // image + "damage" + correlation
  // ------------------------------

  // square grid of circles
  size_t N = 15;
  size_t M = 500;
  auto row = xt::linspace<double>(0, M, N);
  auto col = xt::linspace<double>(0, M, N);
  xt::xarray<double> rowmat;
  xt::xarray<double> colmat;
  std::tie(rowmat, colmat) = xt::meshgrid(row, col);
  rowmat = xt::ravel(rowmat);
  colmat = xt::ravel(colmat);
  xt::xtensor<double, 1> r = (double)(M) / (double)(N) / 4.0 * xt::ones<double>({N, N});

  // random perturbation
  rowmat += GooseEYE::random::normal({N * N}, 0.0, (double)(M) / (double)(N));
  colmat += GooseEYE::random::normal({N * N}, 0.0, (double)(M) / (double)(N));
  xt::xtensor<double, 1> dr = GooseEYE::random::random({N * N}) * 2.0 + 0.1;
  r = r * dr;

  // generate image
  auto I = GooseEYE::dummy_circles({M, M}, xt::round(rowmat), xt::round(colmat),
  xt::round(r));

  // create 'damage' -> right of inclusion
  colmat += 1.1 * r;
  r *= 0.4;
  auto W = GooseEYE::dummy_circles({M, M}, xt::round(rowmat), xt::round(colmat),
  xt::round(r));

4.2. Example
4.3 Collapse to single point

To calculate the probability of the inclusion directly next to a weight site (i.e. the red circles in the example above and below) the ‘collapsed correlation’ is calculated. The distance to the edge of the site, $\vec{\delta}_i$, is therefore corrected for as follows:

$$ P(\Delta \vec{x}) = \frac{\sum_i W(\vec{x}_i) I(\vec{x}_i + \Delta \vec{x} + \vec{\delta}_i)}{\sum_i W(\vec{x}_i)} $$

Similarly to the above, a mask may be introduced as follows:

$$ P(\Delta \vec{x}) = \frac{\sum_i W(\vec{x}_i) [I(1-M)](\vec{x}_i + \Delta \vec{x} + \vec{\delta}_i)}{\sum_i W(\vec{x}_i) (1-M)(\vec{x}_i + \Delta \vec{x} + \vec{\delta}_i)} $$

See also:


4.3.1 Example

W2c.py
W2c.cpp
Note: Like for the 2-point correlation, a mask can be used. Similarly, the average can be extended to that of an ensemble of images.

4.3.2 Python

```python
import GooseEYE
import numpy as np

# square grid of circles
N = 15
M = 500
row = np.linspace(0, M, N)
col = np.linspace(0, M, N)
row, col = np.meshgrid(row, col, indexing="ij")  # ('indexing' only for comparison,
# with C++ code)
row = row.reshape(-1)
col = col.reshape(-1)
r = float(M) / float(N) / 4.0 * np.ones(N * N)

# random perturbation
row += GooseEYE.random.normal([N * N], 0.0, float(M) / float(N))
col += GooseEYE.random.normal([N * N], 0.0, float(M) / float(N))
r *= GooseEYE.random.random([N * N]) * 2.0 + 0.1

# generate image, extract 'volume-fraction' for plotting
img = GooseEYE.dummy_circles((M, M), np.round(row), np.round(col), np.round(r))
phi = np.mean(img)

# create 'damage' -> right of inclusion
col += 1.1 * r
r *= 0.4
W = GooseEYE.dummy_circles((M, M), np.round(row), np.round(col), np.round(r))
W[img == 1] = 0

# compute individual damage clusters and their centers
Clusters = GooseEYE.Clusters(W)
clusters = Clusters.labels()
centers = Clusters.centers()

# weighted correlation
WI = GooseEYE.W2((101, 101), W, img, fmask=W)

# collapsed weighted correlation
WIc = GooseEYE.W2c((101, 101), clusters, centers, img, fmask=W)
```

4.3.3 C++

```c++
#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>
```

(continues on next page)
# define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)  
#define MYASSERT_IMPL(expr, file, line) \ 
if (!(expr)) { \ 
    throw std::runtime_error( \ 
        std::string(file) + ": assertion failed (" #expr ") \n	"); \ 
}

int main()
{
    // square grid of circles
    size_t N = 15;
    size_t M = 500;
    auto row = xt::linspace<double>(0, M, N);
    auto col = xt::linspace<double>(0, M, N);
    xt::xarray<double> rowmat; \n    xt::xarray<double> colmat;
    std::tie(rowmat, colmat) = xt::meshgrid(row, col);
    rowmat = xt::ravel(rowmat);
    colmat = xt::ravel(colmat);
    xt::xtensor<double, 1> r = (double)(M) / (double)(N) / 4.0 * xt::ones<double>({N, N});

    // random perturbation
    rowmat += GooseEYE::random::normal({N * N}, 0.0, (double)(M) / (double)(N));
    colmat += GooseEYE::random::normal({N * N}, 0.0, (double)(M) / (double)(N));
    xt::xtensor<double, 1> dr = GooseEYE::random::random({N * N}) * 2.0 + 0.1;
    r = r * dr;

    // generate image
    auto I = GooseEYE::dummy_circles({M, M}, xt::round(rowmat), xt::round(colmat), xt::round(r));

    // create 'damage' -> right of inclusion
    colmat += 1.1 * r;
    r *= 0.4;
    auto W = GooseEYE::dummy_circles({M, M}, xt::round(rowmat), xt::round(colmat), xt::round(r));
    W = xt::where(xt::equal(I, 1), 0, W);

    // compute individual damage clusters and their centers
    GooseEYE::Clusters Clusters(W);
    auto clusters = Clusters.labels();
    auto centers = Clusters.centers();

    // weighted correlation
    auto WI = GooseEYE::W2({101, 101}, W, I, W);

    // collapsed weighted correlation
    auto WIC = GooseEYE::W2c({101, 101}, clusters, centers, I, W);

    // check against previous versions
    H5Easy::File data("W2c.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::all(xt::equal(I, H5Easy::load<decltype(I)>(data, "I"))));
    MYASSERT(xt::all(xt::equal(clusters, H5Easy::load<decltype(clusters)>(data, "clusters"))));
    MYASSERT(xt::all(xt::equal(centers, H5Easy::load<decltype(centers)>(data, "centers"))));
}
GooseEYE Documentation

(continued from previous page)

```cpp
MYASSERT(xt::all(xt::equal(W, H5Easy::load< decltype(W) >(data, "W"))));
MYASSERT(xt::allclose(WI, H5Easy::load< decltype(WI) >(data, "WI")));
MYASSERT(xt::allclose(WIc, H5Easy::load< decltype(WIc) >(data, "WIc")));

return 0;
```

4.3. Collapse to single point 19
CHAPTER 5

Height-height correlation

The height-height correlation corresponds to the following:

\[ P(\Delta \vec{x}) = \sqrt{\sum_i (z(\vec{x}_i + \Delta \vec{x}) - z(\vec{x}_i))^2} \]

5.1 Example

heightheight.py
heightheight.cpp

**Note:** Like for the 2-point correlation, a mask can be used. Similarly, the average can be extended to that of an ensemble of images.

5.1.1 Python

```python
import GooseEYE
import numpy as np

L = 2.0 * np.pi
N = 1000
h = L / N
x = np.linspace(0, L, N)
y1 = np.sin(x)
y2 = np.sin(2.0 * x)
```

(continues on next page)
hh1 = GooseEYE.heightheight(roi=[200], f=y1, periodic=True)
hh2 = GooseEYE.heightheight(roi=[200], f=y2, periodic=True)
dx = GooseEYE.distance(roi=[200], h=[h], dim=0)

5.1.2 C++

```cpp
#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) \
  if (!(expr)) { \
    throw std::runtime_error( \
      std::string(file) + ':' + std::to_string(line) + \
      " assertion failed (" #expr ") \n	"); \
  }

int main()
{
  double L = 2.0 * M_PI;
  size_t N = 1000;
  double h = L / (double)N;
  xt::xarray<double> x = xt::linspace<double>({0, L}, N);
  xt::xarray<double> y1 = xt::sin(x);
  xt::xarray<double> y2 = xt::sin(2. * x);
  std::vector<size_t> roi = {200};
  std::vector<double> hv = {h};
  xt::xarray<double> hh1 = GooseEYE::heightheight(roi, y1, true);
  xt::xarray<double> hh2 = GooseEYE::heightheight(roi, y2, true);
  xt::xarray<double> dx = GooseEYE::distance(roi, hv, 0);

  // check against previous versions
  H5Easy::File data("heightheight.h5", H5Easy::File::ReadOnly);
  MYASSERT(xt::allclose(y1, H5Easy::load<decltype(y1)>(data, "y1")));
  MYASSERT(xt::allclose(y2, H5Easy::load<decltype(y2)>(data, "y2")));
  MYASSERT(xt::allclose(hh1, H5Easy::load<decltype(hh1)>(data, "hh1")));
  MYASSERT(xt::allclose(hh2, H5Easy::load<decltype(hh2)>(data, "hh2")));
  MYASSERT(xt::allclose(dx, H5Easy::load<decltype(dx)>(data, "dx")));

  return 0;
}
```

Chapter 5. Height-height correlation
CHAPTER 6

Obtain clusters

6.1 Calculate clusters

Extract clusters (‘islands’ of connected pixels with the same value).

clusters.py
clusters.cpp

6.1.1 Python

```python
import GooseEYE
import numpy as np

# generate image
ing = GooseEYE.dummy_circles((500, 500), periodic=True)

# clusters
clusters = GooseEYE.clusters(img, periodic=False)

# clusters, if the image is periodic
clusters_periodic = GooseEYE.clusters(img, periodic=True)
```

6.1.2 C++
6.2 Calculate clusters and centers

clusters_centers.py
clusters_centers.cpp

6.2.1 Python

```python
import GooseEYE
import numpy as np

# generate image
img = GooseEYE.dummy_circles((500, 500), periodic=True)

# clusters
clusters = GooseEYE.Clusters(img, periodic=False)
labels = clusters.labels()
centers = clusters.center_positions()

# clusters, if the image is periodic
clusters_periodic = GooseEYE.Clusters(img, periodic=True)
labels_periodic = clusters_periodic.labels()
centers_periodic = clusters_periodic.center_positions()
```

6.2.2 C++

```cpp
#include <GooseEYE/GooseEYE.h>
#include <xtensor/xarray.hpp>
#include <xtensor/xio.hpp>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) \
```
if (!(expr)) { \
    throw std::runtime_error( \
        std::string(file) + ' ' + std::to_string(line) + \
        ": assertion failed (" #expr ") \n\n\n\n"}; \
\}

int main()
{
    // generate image
    auto I = GooseEYE::dummy_circles({500, 500}, true);

    // clusters
    GooseEYE::Clusters clusters(I, false);
    auto labels = clusters.labels();
    auto centers = clusters.center_positions();

    // clusters, if the image is periodic
    GooseEYE::Clusters clusters_periodic(I, true);
    auto labels_periodic = clusters_periodic.labels();
    auto centers_periodic = clusters_periodic.center_positions();

    // check against previous versions
    H5Easy::File data("clusters_centers.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::all(xt::equal(I, H5Easy::load<
        decltype(I)>(data, "I"))));
    MYASSERT(xt::all(xt::equal(labels, H5Easy::load<
        decltype(labels)>(data, "labels"))));
    MYASSERT(xt::allclose(centers, H5Easy::load<
        decltype(centers)>(data, "centers")));
    MYASSERT(xt::allclose(centers_periodic, H5Easy::load<
        decltype(centers_periodic)>(data, "centers_periodic")));

    return 0;
}

6.3 Dilate clusters (differently)

clusters_dilate.py
clusters_dilate.cpp

6.3.1 Python

clusters_dilate.py

import GooseEYE
import numpy as np

(continues on next page)
GooseEYE Documentation

# generate image
img = np.zeros((21, 21), dtype="bool")
img[4, 4] = True
img[14, 15] = True
img[15, 15] = True
img[16, 15] = True
img[15, 14] = True
img[15, 16] = True

# clusters
C = GooseEYE.Clusters(img).labels()

# dilate
CD = GooseEYE.dilate(C)

Note: There is an additional example to show the effect of periodicity: clusters_dilate_periodic.py clusters_dilate_periodic.svg

6.3.2 C++

```c++
#include <GooseEYE/GooseEYE.h>
#include <highfive/H5Easy.hpp>

#define MYASSERT(expr) MYASSERT_IMPL(expr, __FILE__, __LINE__)
#define MYASSERT_IMPL(expr, file, line) \
if (!(expr)) { \
    throw std::runtime_error( \
        std::string(file) + ': ' + std::to_string(line) + \
        " assertion failed (" #expr " ) \n    ); \
}

int main()
{
    // generate image
    xt::xarray<int> I = xt::zeros<int>([21, 21]);
    I(4, 4) = 1;
    I(14, 15) = 1;
    I(15, 15) = 1;
    I(16, 15) = 1;
    I(15, 14) = 1;
    I(15, 16) = 1;

    // clusters
    auto C = GooseEYE::Clusters(I).labels();

    // dilate
    auto CD = GooseEYE::dilate(C);

    // check against previous versions
    H5Easy::File data("clusters_dilate.h5", H5Easy::File::ReadOnly);
    MYASSERT(xt::all(xt::equal(I, H5Easy::load< decltype(I) >(data, "I"))));
    MYASSERT(xt::all(xt::equal(C, H5Easy::load< decltype(C) >(data, "C"))));
    MYASSERT(xt::all(xt::equal(CD, H5Easy::load< decltype(CD) >(data, "CD"))));
```

(continues on next page)
6.3. Dilate clusters (differently)
CHAPTER 7

Image manipulation tricks

7.1 Segmentation

7.1.1 Ostu’s method

Note: References
- Implement Otsu’s thresholding in Python without using OpenCV and MATLAB
- Scikit’s skimage.filters.threshold_otsu
- Scikit’s Otsu’s thresholding

source: otsu.py

This example uses Otsu’s method to combine two sources (two different images of the same object) to get an optimal threshold. The idea is to get a weighted average of two images as follows

$$\mathcal{I} = (1 - \alpha)\mathcal{I}_a + \alpha \mathcal{I}_b$$

whereby $\alpha$ is chosen such that the separability, defined by Otsu is maximised.
8.1 v0.6.1

- Using scikit-build, doxygen
- Renaming “test” -> “tests”
- [Python] Fixing setup dependencies

8.2 v0.6.0

- General code-style update (#68)
- Templating all functions (#66)
- [GooseEYE::detail] Deprecating unused functions
- Adding historical tests (#64)
  - Minor code-style updates
  - Separating seed to be manual -> generate the same result on each run unless the seed is set.
  - Adding GooseEYE::random::random and GooseEYE::random::normal to help with reproducible results.
  - Saving/comparing against earlier run examples.
  - Adding numerical tests to check against previous versions
  - [CI] Not all platforms run all examples to save time.
- Switching to templates for ‘clusters’ API (#63)
- Generalizing “pad” (#62)
- Adding “center_of_mass” and “pos2img” (#61)
- Code-simplification (#57)
– Applying pep8-like style, added clang-format style-file.
– Simplifying implementation: using xt::xtensor<T,3> and atleast_3d internally.
– Clarifying comments.
– Adding tests.

• Update CMake style (#55)
CHAPTER 9

Compiling

9.1 Getting GooseEYE

9.1.1 Using conda

The easiest is to use conda to install GooseEYE:

```
conda install -c conda-forge gooseeye
```

9.1.2 From source

Download the package:

```
git checkout https://github.com/tdegeus/GooseEYE.git
cd GooseEYE
```

Install headers, CMake and pkg-config support:

```
cmake .
make install
```

9.2 Compiling

9.2.1 Using CMake

The following structure can be used for your project’s CMakeLists.txt:
find_package(GooseEYE REQUIRED)
add_executable(myexec mymain.cpp)
target_link_libraries(myexec PRIVATE
  GooseEYE
  xtensor::optimize
  xtensor::use_xsimd)

See the documentation of xtensor concerning optimisation.

**Note:** There are additional targets available to expedite your CMakeLists.txt:

- **GooseEYE::assert**: enable GooseEYE assertions by defining GOOSEEYE_ENABLE_ASSERT.
- **GooseEYE::debug**: enable GooseEYE assertions by defining GOOSEEYE_ENABLE_ASSERT and xtensor assertions by defining XTENSOR_ENABLE_ASSERT (slow).
- **GooseEYE::compiler_warnings**: enable compiler warnings (generic).

### 9.2.2 By hand

Presuming that the compiler is `c++`, compile using:

```
c++ -I/path/to/GooseEYE/include ...
```

Note that you have to take care of the `xtensor` dependency, the C++ version, optimisation, enabling `xsimd`, ...

### 9.2.3 Using pkg-config

Find the location of the headers can be automatised using `pkg-config`:

```
pkg-config --cflags GooseEYE
```
Note: GooseEYE is a research code. The best reference is the code itself: this reader just gives an overview and points in the right directions.

10.1 Header-only

Just

```
#include <GooseEYE/GooseEYE.h>
```

Everything is contained in the namespace GooseEYE.

10.2 Ensemble or individual image

There are two modes of using the code:

- Individual image: use individual functions (e.g. GooseFEM::S2(...), GooseFEM::W2(...), etc.)
- Ensemble of images: use the GooseFEM::Ensemble class. See example

The individual functions are simply a wrapper around the GooseFEM::Ensemble class. The general structure for an ensemble of images is as follows:

1. Initialize the ensemble, defining some settings of which the shape of the region-of-interest is mandatory. For example:

```
GooseEYE::Ensemble ensemble({51, 51});
```

2. Compute the statistics by evaluating a sequence of images in the ensemble. For example:
3. Evaluate the result:

```cpp
auto result = ensemble.result();
```

**Note:** The variance around the average can be obtained using

```cpp
ensemble.variance();
```

**Note:** To obtain the raw result and normalisation use:

```cpp
// first moment : x_1 + x_2 + ...
ensemble.data_first();

// second moment: x_1^2 + x_2^2 + ...
ensemble.data_second();

// normalisation (number of measurements)
ensemble.norm();
```

Using the individual images wrapper, all these steps are combined in a single function call with almost the same arguments. The only limitation is the raw data and normalization cannot be accessed.
11.1 Statistics

Note: The functions are available directly in the GooseEYE namespace for individual images, and as member functions of the Ensemble-class.

11.1.1 GooseEYE::mean

The arithmetic mean.

Note: An overload is available to mask certain voxels

See also:

- GooseEYE.h
- Ensemble_mean.hpp

11.1.2 GooseEYE::S2

2-point correlation.

Note: An overload is available to mask certain voxels.

See also:

- GooseEYE.h
- Ensemble_S2.hpp
11.1.3 GooseEYE::C2

2-point cluster function.

Note: An overload is available to mask certain voxels.

See also:
- GooseEYE.h
- Ensemble_C2.hpp
- Theory & Example.

11.1.4 GooseEYE::W2

Weighted 2-point correlation.

Note: An overload is available to mask certain voxels.

See also:
- GooseEYE.h
- Ensemble_W2.hpp
- Theory & Example.

11.1.5 GooseEYE::W2c

Collapsed weighted 2-point correlation.

Note: An overload is available to mask certain voxels.

See also:
- GooseEYE.h
- Ensemble_W2c.hpp
- Theory & Example.

11.1.6 GooseEYE::height

Height-height correlation.

Note: An overload is available to mask certain voxels.

See also:
11.2 Information

11.2.1 GooseEYE::distance

The relative distance of each pixel of the ROI.

See also:

- GooseEYE.h
- GooseEYE.hpp
- *Example.*

11.2.2 GooseEYE::Clusters

Get clusters.

See also:

- GooseEYE.h
- clusters.hpp
- *Example.*

11.3 Generate shape

11.3.1 GooseEYE::dummy_circles

Create a dummy binary images of circles.

See also:

- GooseEYE.h
- dummy_circles.hpp
- *Example.*
CHAPTER 12

Getting GooseEYE

12.1 Using conda

The quickest (but not the most efficient!) is to use conda to install GooseEYE:

```
conda install -c conda-forge python-gooseeye
```

**Warning:** This package does not benefit from xsimd optimisation, as it is not compiled on your hardware. Therefore compiling by hand is recommended.

12.2 From source

Start by installing the dependencies, for example using conda:

```
conda install -c conda-forge pyxtensor xsimd
```

Note that xsimd is optional, but recommended.

**Note:** You can also use:

```
python -m pip install pyxtensor pybind11
```

for use without conda. Note that you install xsimd yourself in such a way that Python can find it in order to use it.

Then, download the package:

```
git checkout https://github.com/tdegeus/GooseEYE.git
cd GooseEYE
```
Install the package using:

```
python -m pip install .
```

**Note:** The following will give more readable output:

```
python setup.py build
python setup.py install
```
The Python interface is a simple wrapper around the C++ classes and functions. In general the Python and C++ codes are almost identical, with as only differences:

- The C++ `xt::xarray` are `numpy.ndarray` in Python.
- The syntax `::` in Python in simply a `.`

For example:

```cpp
GooseEYE::Ensemble ensemble({51, 51});
```

In Python is

```python
ensemble = GooseEYE.Ensemble([51, 51])
```
CHAPTER 14

Indices and tables

- genindex
- modindex
- search