

HoloProcessing 中文文档

Qling

April 24, 2021

Contents

Contents	ii
I 主页	1
II HoloProcessing 中文文档	2
1 Package Features	4
2 Manual Outline	5
III 手册	6
3 标准全息处理	7
3.1 全息图的读取	7
3.2 全息图的再现	7
4 全息降噪算法	9
4.1 空域掩膜法 (SDM)	9
4.2 冗余散斑降噪法 (RSE)	9
4.3 低维重建法 (LDR)	10
5 质量评价指标	11
5.1 图像对比度 (Contrast, C)	11
5.2 等效视数 (Equivalent Number of Looks, ENL)	11
5.3 散斑抑制系数 (Speckle Suppression Index, SSI)	11
5.4 散斑抑制和均值保持指数 (Speckle Suppression and Mean Preservation Index, SMPI)	11
6 函数库	13
6.1 全息处理	13
6.2 全息降噪	15
6.3 评价方法	19

Part I

主页

Part II

HoloProcessing 中文文档

欢迎来到 HoloProcessing 中文文档 ([PDF 版本](#))!

Chapter 1

Package Features

该包主要分三大模块：

- 标准全息处理：
 - 全息的读取
 - 全息的重建（仅实现无透镜傅立叶变换全息的数值重建）
- 全息降噪算法
 - 空域掩膜法（SDM）
 - 冗余散斑降噪法（RSE）
 - 低维重建法（LDR）
- 质量评价指标
 - Contrast
 - ENL
 - SMPI
 - SSI

Chapter 2

Manual Outline

- [标准全息处理](#)
- [全息降噪算法](#)
- [质量评价指标](#)
- [函数库](#)

Part III

手册

Chapter 3

标准全息处理

3.1 全息图的读取

读取全息图，并将其转换为 Float64 类型的二维矩阵

```
| holo = load_holo(path, "xxx.bmp"; convert=true)
```

其中：

- path 是存放全息图的路径
- "xxx.bmp" 是全息图的名称（实验中全息图都是以及 bmp 格式存放的）
- convert 表示是否将其转换为 Float64 的矩阵

3.2 全息图的再现

对全息图实现数值再现（针对无透镜傅立叶变换全息图）

开启多线程

Note

首先需要注意的是，由于 julia 的傅立叶变换实现是默认不开多线程（而 matlab 的傅立叶变换是默认开多线程的，这也是为什么如果直接使用 fft 函数，julia 的性能会比 matlab 差）。因此，需要在建立 P（后面会解释这个 P 是什么）之前开启傅立叶变换的多线程，如下：

```
| FFTW.set_num_threads(Sys.CPU_THREADS)
```

其中，Sys.CPU_THREADS 表示我们 cpu 核心数的最大数量，比如在 12 核 cpu 上，输入 Sys.CPU_THREADS，则显示如下：

```
| julia> Sys.CPU_THREADS  
12
```

高效的傅立叶变换的实现

正常情况下，对图像进行傅立叶变换，其代码如下：

```
fft_img = fft(holo)

# 或者
fft_img = fftshift(fft(fftshit(holo)))
```

其中是否加上 `fftshifts` 其关系不大，`fftshift` 的作用仅仅是对图像进行旋转而已。

在实际的实现中，考虑到会多次执行傅立叶变换的操作。因此，一个更加具备效率的做法是

```
P = plan_fft(holo)
fft_img = P * holo
```

在这里 `P` 是 `FFTW.cFFTWPlan`，表示以后都打算对与 `holo` 同个维度的矩阵进行傅立叶变换。

另外，由于我们知道无透镜傅立叶变换全息图的再现像，其 $+1$ 级和 -1 级都是一样的。因此，为了提高效率和节省空间，我们并不需要重建出完整的图像，而是可以重建出一半即可，这通过改变 `P` 即可做到：

```
P = plan_rfft(holo)
fft_img = P * holo
```

总结

- 总的来说，一个开启了多线程的全息图数值重建代码范例（High Performance）如下：

```
FFTW.set_num_threads(Sys.CPU_THREADS)
Pr = plan_rfft(holo)
# scale 是手动调整的
scale = 1500
re_img = reconst(holo, Pr, scale; nthreads=true)
```

- 如果你坚持要完整的重建像，则范例如下：

```
FFTW.set_num_threads(Sys.CPU_THREADS)
P = plan_fft(holo)
# scale 是手动调整的
scale = 1500
re_img = reconst(holo, P, scale; nthreads=true)
```

- 如果你还需要对图像进行旋转（建议仅在需要观测合适的重建像时使用），则范例如下

```
FFTW.set_num_threads(Sys.CPU_THREADS)
P = plan_fft(holo)
# scale 是手动调整的
scale = 1500
re_img = reconst(holo, P, scale; shift=true, nthreads=true)
```

Note

你可能不确定 `shift` 采用 `true` 还是 `false`，我建议你都试一下，然后用 `imshow` 函数看一下图像的区别。

Chapter 4

全息降噪算法

4.1 空域掩膜法 (SDM)

一个简单的演示案例如下：

```
holo = load_holo(path, "xxx.bmp"; convert=true)
FFTW.set_num_threads(Sys.CPU_THREADS)
P = plan_rfft(holo)^^I# or P = plan_fft(holo)
# Parameter
N = 2
Nx, Ny = size(holo) .÷ N
Dx, Dy = 50, 100
scale = 600

sdm_img = sdm(holo, (Nx, Ny), (Dx, Dy), P, scale)
```

Note

更多的用法，可以通过输入如下：

```
>julia?
help>sdm
```

来获取 `sdm` 函数的更多用法。

4.2 冗余散斑降噪法 (RSE)

一个简单的演示案例如下：

```
holo = load_holo(path, "xxx.bmp"; convert=true)
FFTW.set_num_threads(Sys.CPU_THREADS)
P = plan_rfft(holo)^^I# or P = plan_fft(holo)
# Parameter
N = 2
Nx, Ny = size(holo) .÷ N
Dx, Dy = 50, 100
scale = 600

sdm_img = rse(holo, (Nx, Ny), (Dx, Dy), P, scale)
```

Note

更多的用法，可以通过输入如下：

```
>julia?  
help>rse
```

来获取 `rse` 函数的更多用法.

4.3 低维重建法 (LDR)

一个简单的演示案例如下：

```
holo = load_holo(path, "xxx.bmp"; convert=true)  
FFTW.set_num_threads(Sys.CPU_THREADS)  
  
# Parameter  
N = 2  
P = plan_rfft(similar(holo, size(holo) .÷ N))  
# or P = plan_fft(similar(holo, size(holo) .÷ N))  
  
Dx, Dy = 50, 100  
scale = 600  
  
ldr_img = ldr(holo, N, (Dx, Dy), P, scale)
```

Note

更多的用法，可以通过输入如下：

```
>julia?  
help>ldr
```

来获取 `ldr` 函数的更多用法.

Chapter 5

质量评价指标

5.1 图像对比度 (Contrast, C)

$$C = \frac{\mu_I}{\sigma_I}$$

其中 μ_I 和 σ_I 分别表示图像的平均值及其标准差。

```
| C = contrast(img)
```

5.2 等效视数 (Equivalent Number of Looks, ENL)

$$ENL = \left(\frac{\mu_I}{\sigma_I} \right)^2$$

其中 μ_I 和 σ_I 分别表示图像的平均值及其标准差。ENL 通常用于测量不同的降噪滤波器的性能好坏，当 ENL 值较大时，表明图像比较平滑，这意味着图像的噪点突刺比较少，滤波器的降噪性能较好。

```
| ENL = enl(img)
```

5.3 散斑抑制系数 (Speckle Suppression Index, SSI)

$$SSI = \frac{\sigma_f}{\mu_f} \cdot \frac{\mu_o}{\sigma_o}$$

其中 σ_o 和 μ_o 分别表示原始图像的标准差和均值。类似地， σ_f 和 μ_f 分别是经过降噪滤波器降噪后的图像的标准差和均值。通常来说，图像的均值表示它的信息，而图像的标准差则表示它的噪声严重程度，因此，SSI 越小意味着降噪滤波器的性能越好。

```
| SSI = ssi(noised=noised_img, filtered=filtered_img)
```

5.4 散斑抑制和均值保持指数 (Speckle Suppression and Mean Preservation Index, SMPI)

$$SMPI = (1 + |\mu_f - \mu_o|) \cdot \frac{\sigma_f}{\sigma_o}$$

与 ENL 和 SSI 相比, SMPI 考虑了降噪后的图像和降噪前的图像之间的均值差异。当降噪后的图片均值过于偏离原有的图片均值时, SMPI 的数值的可信度高于 ENL 和 SSI。理论上, 较小的 SMPI 值表示在均值保持和降噪方面, 滤波器具有更好的性能。

```
| SMPI = smpi(noised=noised_img, filtered=filtered_img)
```

Chapter 6

函数库

6.1 全息处理

`HoloProcessing.brightness` - Function.

```
| brightness(c) -> Float64
```

对给定的颜色三通道像素值，求出其亮度并以浮点数的形式输出

Notice

由于实验中记录的全息图都是灰度图像，其三个通道分量颜色都是一致的，故取某一颜色分量作为其亮度即可

Arguments

- c: RGB 类型的像素值

[source](#)

```
| brightness(::AbstractMatrix{T}) -> AbstractMatrix{Float64}
```

对给定的颜色三通道图像，求出其亮度并以浮点数的形式输出

Notice

由于实验中记录的全息图都是灰度图像，其三个通道分量颜色都是一致的，故取某一颜色分量作为其亮度即可

[source](#)

`HoloProcessing.load_holo` - Function.

```
| load_holo(PATH::String, name::String; convert=true) -> AbstractMatrix
```

给定图像路径以及图像的名称(包括后缀)，实现对全息图的载入

Arguments

- PATH: 图像所在路径
- name: 图像的名称，比如 tail.bmp
- convert: 可选参数，默认值是 true。当为 true 时，该函数返回图像的灰度值浮点数矩阵；当为 false 时，该函数直接返回图像

source

`HoloProcessing.normalize` - Function.

```
| normalize(x::AbstractMatrix; nthreads=false) -> AbstractMatrix
```

对矩阵 x 进行归一化，并返回归一化后的矩阵

Arguments

- x : 矩阵
- `nthreads`: 决定是否开多线程来进行归一化，默认为否

Notice

对于不大的矩阵（低于 10000×10000 ），不建议开多线程，这是因为多线程会有额外的内存和时间开销

source

`HoloProcessing.normalize!` - Function.

```
| normalize!(x::AbstractMatrix; nthreads=false) -> AbstractMatrix
```

对矩阵 x 进行原地归一化，即原地修改矩阵并返回修改后的 x

Arguments

- x : 矩阵
- x : 矩阵
- `nthreads`: 决定是否开多线程来进行归一化，默认为否

Notice

对于不大的矩阵（低于 10000×10000 ），不建议开多线程，这是因为多线程会有额外的内存和时间开销

source

`HoloProcessing.color` - Function.

```
| color(x::AbstractMatrix{<:AbstractFloat}) -> AbstractMatrix
```

将浮点数类型矩阵转换为灰度图像

Arguments

- x : 浮点数矩阵

source

`HoloProcessing.reconst` - Function.

```
| reconst(holo::AbstractMatrix, P::FFTW.FFTWPlan, scale::Integer; shift=false) -> AbstractMatrix
```

对无透镜傅立叶全息图进行数值重建

Arguments

- `holo`: 无透镜傅立叶全息图
- `P`: 其类型为 `FFTW.FFTWPlan`, 表示重建采用 `fft` 傅立叶变换
- `scale`: 由于零级像强度过大, 故需要用 `scale` 拉升 ± 1 级像的强度
- `shift`: 是否旋转图像, 默认为否 (除非是想展示完整的全息图, 否则不建议将 `shift` 设为 `true`, 因为其内存和时间开销比较大)
- `nthreads`: 是否开启多线程, 默认为否 (值得注意的是, 该线程与傅立叶变换多线程的开启无关)

source

6.2 全息降噪

`HoloProcessing.make_sub_holo!` - Function.

```
make_sub_holo!(
  tensor::AbstractArray{T,3},
  holo::AbstractMatrix{T},
  windows_size::Tuple{Integer,Integer},
  interval::Tuple{Integer,Integer},
  nums_of_holo::Tuple{Integer,Integer}
) -> AbstractArray{T,3} where T <: AbstractFloat
```

生成子全息图并将其存放到 `tensor` 序列上

Arguments

- `tensor`: 子全息图序列
- `holo`: 原始全息图
- `windows_size`: 掩膜窗口的大小
- `interval`: 窗口偏移量
- `numsofholo`: 子全息图的数量 (与 `tensor` 的第三个维度一致)

Notice

- 当子全息图大小与原始全息图一致时, 则对位于窗口外的地方补零
- 当子全息图大小与掩膜窗口大小一致时, 则抛弃窗口外的数据

source

`HoloProcessing.reconst_tensor!` - Function.

```
reconst_tensor!(
  re_tensor::AbstractArray{T,3},
  holo_tensor::AbstractArray{T,3},
  P::FFTW.FFTWPlan,
  scale::Integer;
  shift = false
) -> AbstractArray{T,3} where T <: AbstractFloat
```

对子全息图序列进行数值重建并存放到 `re_tensor`

Arguments

- `re_tensor`: 子再现像序列
- `holo_tensor`: 子全息图序列
- `P`: 其类型为 `FFTW.FFTWPlan`, 表示重建采用 `fft` 傅立叶变换
- `scale`: 由于零级像强度过大, 故需要用 `scale` 拉升 ± 1 级像的强度
- `shift`: 是否旋转图像, 默认为否 (除非是想展示完整的重建像, 否则不建议将 `shift` 设为 `true`, 因为其内存和时间开销比较大)

source

空域掩膜法实现

`HoloProcessing.sdm` – Function.

```
sdm( holo::AbstractMatrix{T}, windows_size::Tuple{Integer,Integer}, interval::Tuple{Integer,Integer}, P::FFTW.FFTWPlan,
scale::Integer; shift = false ) -> AbstractMatrix where T <: AbstractFloat
```

空域掩膜法的算法实现, 具体可见 doi: 10.1109/JDT.2015.2479646

Arguments

- `holo`: 全息图 (非 RGB 类型而是 `AbstractFloat` 类型)
- `windows_size`: 掩膜窗口大小
- `interval`: 窗口偏移量
- `holo`: 原始全息图
- `P`: 其类型为 `FFTW.FFTWPlan`, 表示重建采用 `fft` 傅立叶变换
- `scale`: 由于零级像强度过大, 故需要用 `scale` 拉升 ± 1 级像的强度
- `shift`: 是否旋转图像, 默认为否 (除非是想展示完整的重建像, 否则不建议将 `shift` 设为 `true`, 因为其内存和时间开销比较大)

source

`HoloProcessing.sdm_core!` – Function.

```
sdm_core!(sdm_img::AbstractMatrix{T}, re_tensor::AbstractArray{T,3}) -> AbstractMatrix where T
↳ <: AbstractFloat
```

空域掩膜法的核心算法, 对子再现像序列进行叠加平均, 并存进 `sdm_img` 中

Arguments

- `sdm_img`: 存放结果的矩阵
- `re_tensor`: 子再现像序列

source

冗余散斑降噪法实现

`HoloProcessing.rse` – Function.

```
rse( holo::AbstractMatrix{T}, windows_size::Tuple{Integer,Integer}, interval::Tuple{Integer,Integer}, P::FFTW.FFTWPlan,
scale::Integer; shift = false ) -> AbstractMatrix{T} where T <: AbstractFloat
```

冗余散斑降噪法的算法实现，具体可见 doi: 10.1364/A0.390500

Arguments

- holo: 全息图（非 RGB 类型而是 AbstractFloat 类型）
- windows_size: 掩膜窗口大小
- interval: 窗口偏移量
- holo: 原始全息图
- P: 其类型为 FFTW.FFTWPlan，表示重建采用 fft 傅立叶变换
- scale: 由于零级像强度过大，故需要用 scale 拉升 ± 1 级像的强度
- shift: 是否旋转图像，默认为否（除非是想展示完整重建像，否则不建议将 shift 设为 true，因为其内存和时间开销比较大）

source

`HoloProcessing.rse_core!` – Function.

```
rse_core!(rse_img::AbstractMatrix{T}, re_tensor::AbstractArray{T,3}) -> AbstractMatrix where T
↳ <: AbstractFloat
```

冗余散斑降噪法的核心算法，对子再现像序列进行去冗余而后叠加平均，并存进 rse_img 中

Arguments

- rse_img: 存放结果的矩阵
- re_tensor: 子再现像序列

source

低维重建法实现

`HoloProcessing.ldr` – Function.

```
ldr(
  holo::AbstractMatrix{T},
  N::Integer,
  interval::Tuple{Integer,Integer},
  P::FFTW.FFTWPlan,
  scale::Integer;
  shift = false
) -> AbstractMatrix{T} where T <: AbstractFloat
```

低维度重建法的算法实现，具体可见 doi: 10.1364/A0.414773

Arguments

- holo: 全息图

- N: 窗口大小的 N 倍就等于 holo 的大小
- interval: 窗口偏移量
- holo: 原始全息图
- P: 其类型为 FFTW.FFTWPlan, 表示重建采用 fft 傅立叶变换
- scale: 由于零级像强度过大, 故需要用 scale 拉升 ± 1 级像的强度
- shift: 是否旋转图像, 默认为否 (除非是想展示完整的重建像, 否则不建议将 shift 设为 true, 因为其内存和时间开销比较大)

source

HoloProcessing.ldr_core! - Function.

```
ldr_core!(ldr_img::AbstractMatrix{T}, re_tensor::AbstractArray{T,3}, N::Integer) ->
↳ AbstractMatrix where T <: AbstractFloat
```

低维重建法的核心降噪算法, 对子再现像序列进行分组平均、聚合、再经过均值滤波器, 最后结果存进 ldr_img 中

Arguments

- ldr_img: 存放结果的矩阵
- re_tensor: 子再现像序列
- N: 分成 N 组

source

HoloProcessing.ldr_denoising! - Function.

```
ldr_denoising!(mean_tensor::AbstractArray{T, 3}, re_tensor::AbstractArray{T,3}, N::Integer) ->
↳ AbstractArray{T, 3} where T <: AbstractFloat
```

低维重建法的核心降噪算法中的降噪步骤, 对子再现像序列 re_tensor 进行分组平均, 其结果放到 mean_tensor 中

Arguments

- mean_tensor: 存放结果的矩阵
- re_tensor: 子再现像序列
- N: 分成 N 组

source

HoloProcessing.ldr_aggregation! - Function.

```
ldr_aggregation!(ldr_img::AbstractArray{T, 3}, mean_tensor::AbstractArray{T,3}, N::Integer) ->
↳ AbstractMatrix where T <: AbstractFloat
```

低维重建法的核心降噪算法中的聚合步骤, 将低维再现像聚合成高维再现像

Arguments

- ldr_img: 存放结果的矩阵
- mean_tensor: 低维再现像序列
- N: 分成 N 组

source

6.3 评价方法

`HoloProcessing.contrast` – Function.

```
| contrast(img::AbstractMatrix) -> AbstractFloat
```

计算图像 `img` 的对比度

[source](#)

`HoloProcessing.ssi` – Function.

```
| ssi(; noised::AbstractMatrix{T}, filtered::AbstractMatrix{T})) -> AbstractFloat where T
```

计算图像 `img` 的 SSI 值

Arguments

- `noised`: 噪声图像
- `filtered`: 降噪后的图像

[source](#)

`HoloProcessing.smpi` – Function.

```
| smpi(; noised::AbstractMatrix{T}, filtered::AbstractMatrix{T})) -> AbstractFloat where T
```

计算图像 `img` 的 SMPI 值

Arguments

- `noised`: 噪声图像
- `filtered`: 降噪后的图像

[source](#)

`HoloProcessing.enl` – Function.

```
| enl(img::AbstractMatrix) -> AbstractFloat
```

计算图像 `img` 的 ENL 值

[source](#)