

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	3
2 Manual Outline	4
III Base	5
3 标准全息处理	6
3.1 全息图的读取	6
3.2 全息图的再现	6
4 全息降噪算法	8
4.1 空域掩膜法 (SDM)	8
4.2 冗余散斑降噪法 (RSE)	8
4.3 低维重建法 (LDR)	9
5 质量评价指标	10
5.1 图像对比度 (Contrast, C)	10
5.2 等效视数 (Equivalent Number of Looks, ENL)	10
5.3 散斑抑制系数 (Speckle Suppression Index, SSI)	10
5.4 散斑抑制和均值保持指数 (Speckle Suppression and Mean Preservation Index, SMPI)	10

Part I

主页

Part II

HoloProcessing 中文文档

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

Base

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)
```