گزارش تمرینات سری سوم

محمد رضیئی

پاسخ سوالات نوشتاری

پاسخ سوال اول و دوم بخش نوشتاری در فایل "MIAP-HW03-MohammadRaziei-98206223-HandWritting.pdf" آورده شده است.

سوال سوم:

ابتدا فرمول اولیه را مشاهده میکنیم:

$$EPI=\frac{\Gamma\left(\Delta s-\overline{\Delta s}\,,\ \widehat{\Delta s}-\overline{\widehat{\Delta s}}\right)}{\sqrt{\Gamma\left(\Delta s-\overline{\Delta s},\Delta s-\overline{\Delta s}\right).\Gamma\left(\widehat{\Delta s}-\overline{\widehat{\Delta s}},\widehat{\Delta s}-\overline{\widehat{\Delta s}}\right)}}$$

$$\Gamma(s_1,s_2)=\sum_{i,j\in ROI} s_1(i,j).s_2(i,j) \quad (9)$$

البته گویا فرمول بالا اشتباه نوشتاری دارد و رادیکال آن کل مخرج را فراگرفته است که از کد EPI.m نیز آن مورد قابل مشاهده است. که علامت Δs نشان دهنده نسخه فیلتر بالاگذر شده ی تصویر به ازای فیلتر لاپلاسین است. که برای تصویر رفرنس بکار میرود و برای تصویر دوم نیز از علامت زاویه ی باز در بالای آن استفاده شده است. و علامت خط بالای آن نیز به معنای میانگین گیری است.

این معیار بسیار قابل شهود است و نیاز به توضیح زیادی ندارد. صرفا توجه داریم که فیلتر لاپلاسین که در لبه ها مقدار بزرگی را اختیار میکند در آشکار سازی لبه کاربرد زیادی دارد و همچنین اگر روابط pearson correlation را یادآور شویم،

$$\rho_{X,Y}=\frac{\mathbb{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X\sigma_Y} \quad \text{(Eq.2)}$$

$$\rho_{X,Y}=\frac{\operatorname{cov}(X,Y)}{\sigma_X\sigma_Y} \quad \text{(Eq.1)}$$

$$r_{xy}=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}} \quad \text{(Eq.3)}$$

متوجه میشویم که رابطه ی بالا صرفا کورولیشن بین لاپلاسین شده ی تصویر مرجع و تصویر مورد بررسی است. فیلتر لاپلاس هم که گفتیم کارش آشکار سازی لبه هاست بنابراین این تکنیک دارد شباهت بین لبه های آشکار شده از تصویر دوم و لبه های تصویر مرجع را نشان می‌دهد. همانگونه که در مقاله نیز ذکر شد که معیار بخاطر مشتق‌گیری(مرتبه دوم) به میزان نویز حساس خواهد بود. باقی مقاله نیز به توضیحات در مورد مسایل متفرقه پرداخته بود.

سوال چهارم:

در این بخش فیلتر trilateralFilter مورد بررسی قرار گرفته است که پوشه Articles مقاله آن آورده شده است.

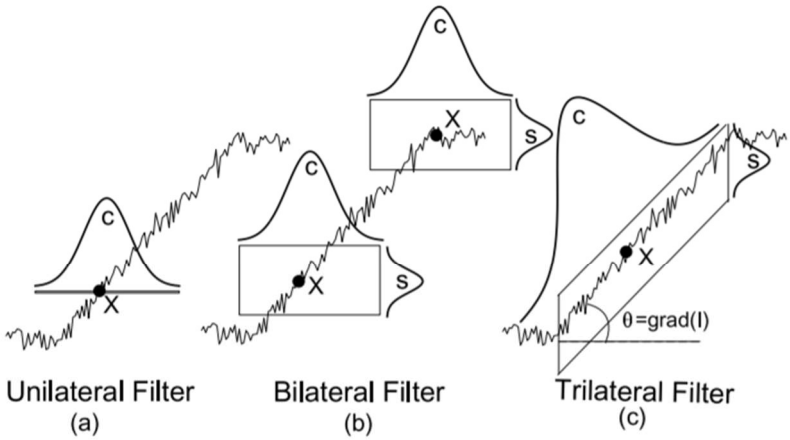
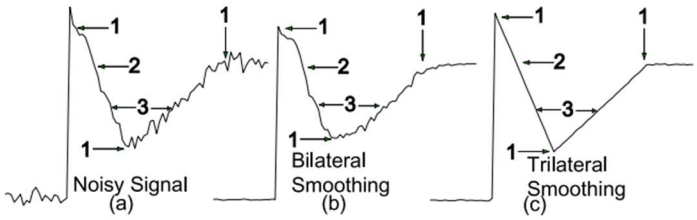


Figure 4: *Unilateral, Bilateral and Trilateral filter windows.*

سه تغییر مهم این فیلتر نسبت به فیلتر bilateral دارد. ایده tilting. S measures در نزدیکی به صفحه ی I(x) و همسایگان وفقی.

این فیلتر در لبه ها بسیار بهتر عمل میکند.



عملیات کاهش نویز بسیار بهتر عمل میکند و لبه های تیز تری را ایجاد میکند. عملیات کاهش نویز آن به صورت زیر محاسبه میشود.

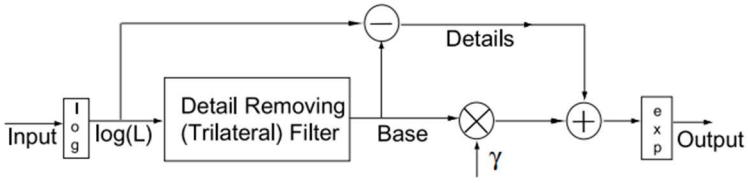


Figure 3: *Contrast reduction method based on edge-preserving filters.*

تفاوت آن با فیلتر bilateral را در محاسبات زیر می‌توان دید.

- Bilateral smooth gradients

$$G_\theta(x)=\frac{1}{k_\theta(x)}\int_{-\infty}^{+\infty}\nabla I_{in}(x+\zeta)c(\zeta)s(\|\nabla I_{in}(x+\zeta)-\nabla I_{in}(x)\|)d\zeta$$

$$k_\theta(x)=\int_{-\infty}^{+\infty}c(\zeta)s(\|\nabla I_{in}(x+\zeta)-\nabla I_{in}(x)\|)d\zeta$$

$$\nabla I_{in}(m,n)\approx(I_{in}(m+1,n)-I_{in}(m,n),I_{in}(m,n+1)-I_{in}(m,n))$$

- Trilateral smooth image
  - Compute approximate value

$$P(x,\zeta)=I_{in}(x)+G_\theta\square\zeta$$

  - Compute $I_\Delta(x,\zeta)$

$$I_\Delta(x,\zeta)=I_{in}(x+\zeta)-p(x,\zeta)$$

  - Filter

$$I_{out}(x)=I_{in}(x)+\frac{1}{k_\Delta(x)}\int_{-\infty}^{+\infty}I_\Delta(x,\zeta)c(\zeta)s(I_\Delta(x,\zeta))f_\theta(x,\zeta)d\zeta$$

$$k_\Delta(x)=\int_{-\infty}^{+\infty}c(\zeta)s(I_\Delta(x,\zeta))f_\theta(x,\zeta)d\zeta$$

$$f_\theta(x,\zeta)=\begin{cases}1 & if\ \|G_\theta(x+\zeta)-G_\theta(x)\|<R\\ 0 & otherwise.\end{cases}$$

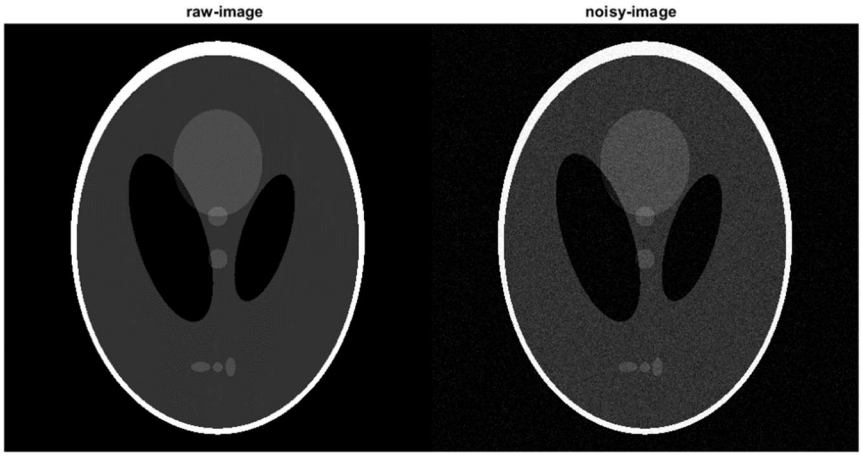
مهمترین علت کاهش نویز به خوبی گفته شده این است که در تخمین overall gradient variability از رابطه ی زیر استفاده کرده است.

$$\sigma_{s\theta}=\beta(\|\max(G_{avg}(\mathbf{x}))-\min(G_{avg}(\mathbf{x}))\|)$$

که تا حد خوبی نسبت به نویز مقاوم بوده است و نویز را کاهش میدهد. همچنین outlier را نیز کم میکند. انتخاب beta نیز به ازای مقادیر 0.1 تا 0.2 بسیار خوب عمل میکند.

پاسخ سوالات شبیه سازی:

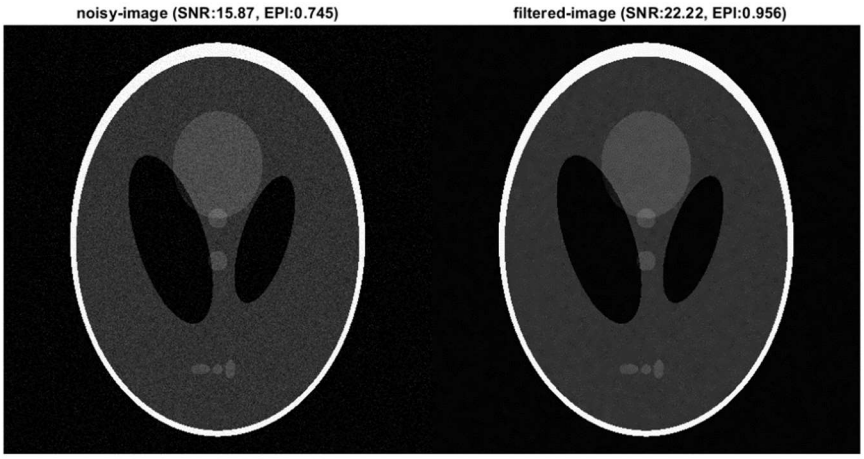
در این سوالات از تصویر گفته شده استفاده شده است که در زیر تصویر آن در مقایسه با نسخه نویزی شده آن وجود دارد:



حال پاسخ سوالات این بخش:

سوال اول:

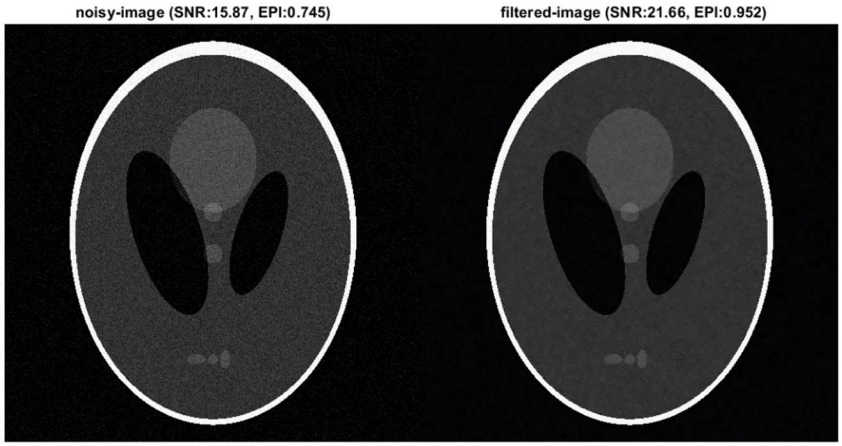
پاسخ قسمت اول که تصویر بالاست و پاسخ دو قسمت بعدی نیز در تصاویر زیر داده شده است. برای پیاده سازی فیلتر nlm و با مفروضات مطلوب به 2 روش انجام شده است. اولین روش استفاده از دستور imnlmfilt خود متلب است و روش دوم نیز استفاده از تابع نوشته شده ی NonLocalMeansFilter است.



تصویر بالا با استفاده از دستور متلب زیر ایجاد شده است.

imnlmfilt(image_noisy, 'ComparisonWindowSize', 3, 'SearchWindowSize', 3*3);

و اما با استفاده از دستور NonLocalMeans که میتوان به صورت زیر نوشت.



که به میزان خوبی هم edge ها حفظ شدند و هم نویز کم شده است.

سوال دوم:

در این سوال که با همان داده های سوال اول شبیه سازی انجام شده است.

قسمت اول: فیلتر BL از حاصل ضرب 2 گوسی به شکل زیر انجام می‌شود. که گوسی اول یک گوسی مکانی است و صرفا به پوزیشن مرتبط است. و گوسی دوم نیز صرفا به روشنایی آن پوزیشن مرتبط است. درصورتی که در نزدیکی لبه حضور داشته باشیم، بخاطر آن که اختلاف روشنایی دو نقطه متعلق به دو طرف لبه زیاد است، ورودی گوسی دوم زیاد میشود و در نتیجه مقدار آن کاهش می یابد و صرفا نقاطی که در یک طرف لبه قرار دارند را فیلتر می‌کند و به این شکل کرنل فیلتر را میبرد. دو پارامتر hx و hg نیز از یکدیگر مستقل هستند که اولی تاثیر فیلتر مکانی را تنظیم می‌کند که این تنظیم به میزان نویز و ناحیه نویزی بستگی دارد و ماکسیمم طول تاثیر

گذاری فیلتر را تنظیم میکند. بنابراین در hx های بالاتر همسایگی های بیشتری مورد پردازش قرار میگیرند و الگوریتم بسیار کند تر میشود. پارامتر hg نیز میزان بریدن کرنل در هنگام رسیدن به لبه را تنظیم میکند و مقدار کمی را میگیرد. در صورتی که لبه ها خیلی قوی باشند میتوان پارامتر hg را افزایش دارد. در حقیقت این پارامتر میزان قوی یا ضعیف بودن لبه را نشان می‌دهد.

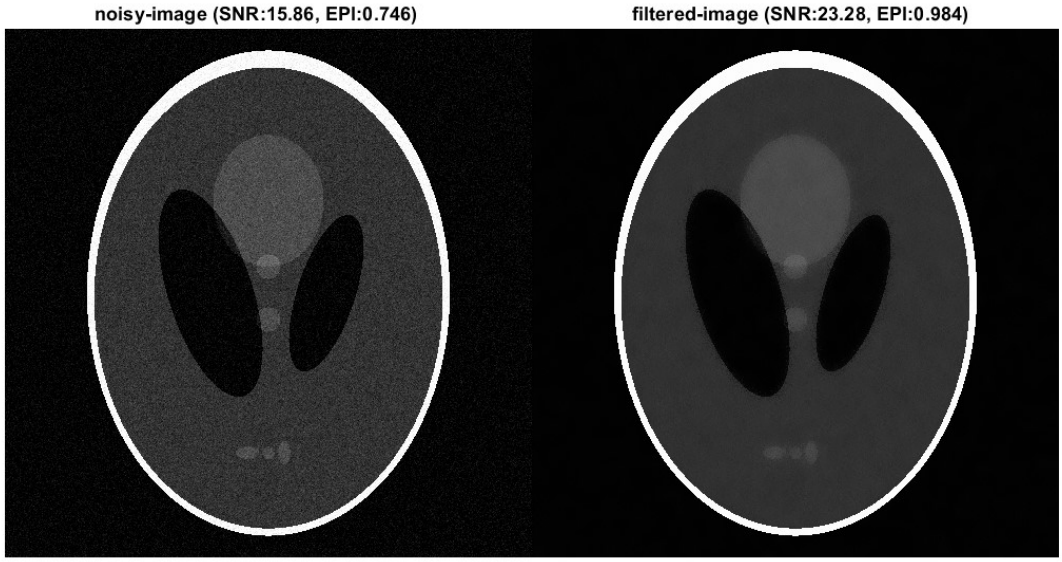
قسمت دوم: این فیلتر به ازای مقادیر مختلف hx, hg فراخوانی شده است. این تابع در فایل BilateralFilter پیاده سازی شده است. گفتنی است که با افزایش hx بسیار کند میشود از این رو پیاده سازی دیگری نیز با استفاده از CUDA در BilateralFilter_cuda انجام شده است که پردازش آن موازی شود.

کد زیر به صورت خودکار اگر cuda ساپورت نمیکرد، روی cpu کار میکند.

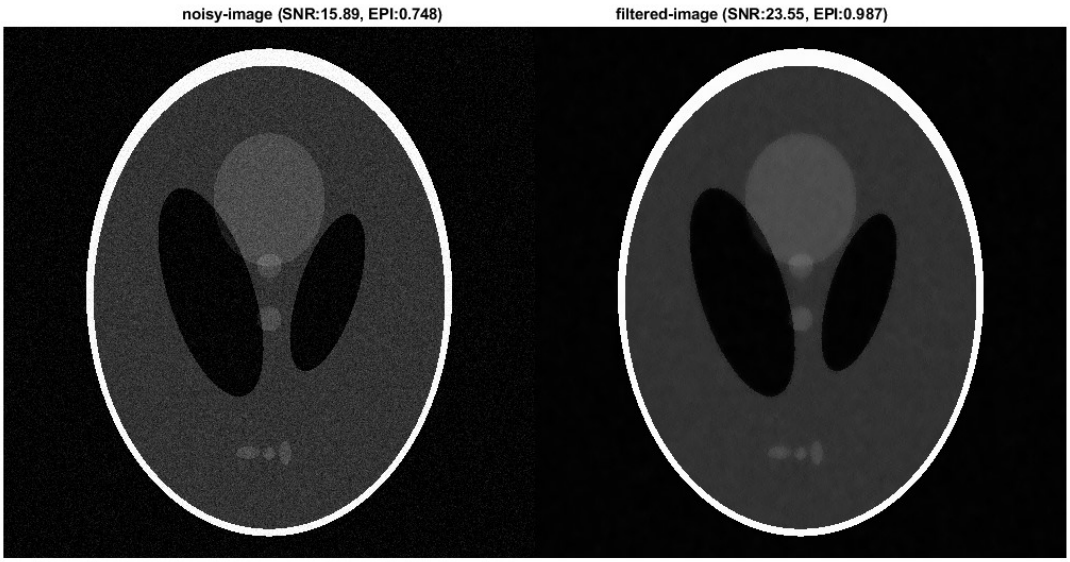
```
try image_filt = BilateralFilter_cuda(image_noisy, hx, hg);
catch,image_filt = BilateralFilter(image_noisy, hx, hg);end
```

فایل های سورس این قسمت در فایل زیپ شده ی codegen.7z قرار گرفته است.

در تصویر زیر به ازای hg=0.1, hx=5 رسم شده است:



همچنین به ازای hx=3, hg=0.1 به صورت زیر است.

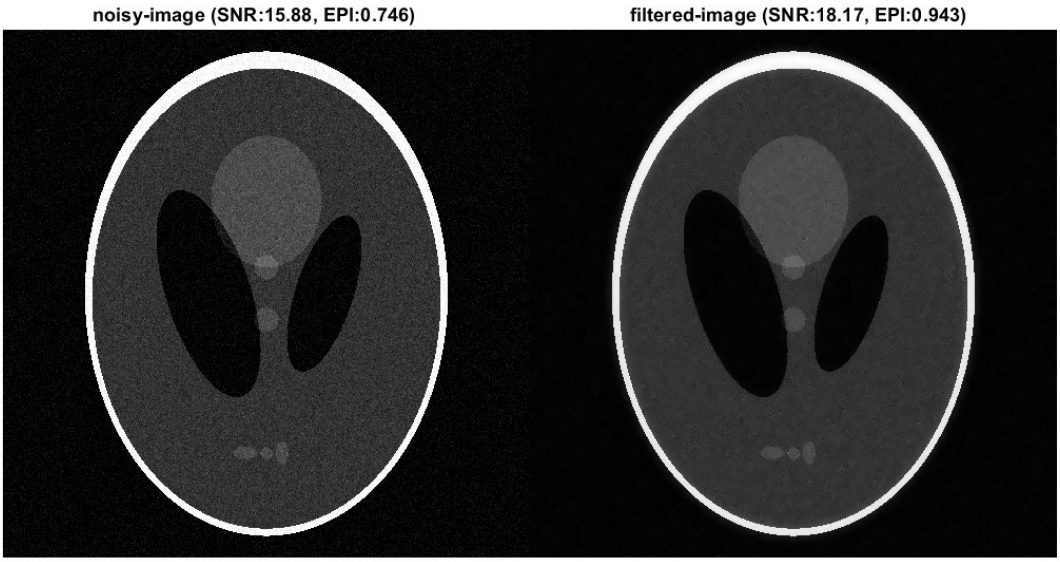


در پوشه Q2 خروجی های دیگر این الگوریتم به ازای برخی مقادیر دیگر hx,hg قرار گرفته است.

توجه: برای اجرا کردن تابع BilateralFilter_cuda لازم است که حتما deviceی که بر روی آن ران میشود، cuda نصب باشد.(این تابع برای موازی کردن پردازش بکار میرود. تا سرعت را افزایش دهد.)

سوال سوم:

پیاده سازی الگوریتم TotalVariation در تابعی به همین نام انجام شده است. هرچند سرعت اجرای این تابع از bilateralFilter بسیار سریعتر بود اما در مجموع بسیار کند عمل میکرد. همچنین در پیاده سازی آن مقداری epsilon به مخرج اضافه شد تا از ناپایداری های عددی که مخرج صفر میشود جلوگیری کند و مقدار محدودی را اختیار کند که این مورد آن کمی تفاوت با فرمولاسیون نسخه اصلی ایجاد می‌نمود.



سوال چهارم:

با توجه به شبیه سازی های انجام شده به جدول زیر میرسیم.

Filtering Method	NonLocalMeans (Q1)	Imnlmfilt (Q1)	BilateralFilter (Q2)	TotalVariation (Q3)
SNR	21.66	22.22	23.55	18.17
EPI	0.952	0.956	0.987	0.943

با توجه به پارامتر های تنظیم شده، بنظر میرسد که فیلتر bilateralFilter هردو کار را بهتر انجام داده است. عیب اصلی و مهم آن حجم پردازشی بالا و کند بودن آن است. فیلتر nonLocalMeans پیاده سازی پیچیده تری را شامل میشد و بسیار سریع تر جواب را میداد. البته سرعت فیلتر bilateral که با CUDA به شکل mex پیاده سازی شده است، بسیار سریع و real-time است. همچنین من در تست های خود اگر شرط سه برابر بودن طول سرچ را برمیداشتم حتی تصاویری با SNR بالاتر یعنی 23.59 را نیز دریافت میکردم. خروجی totalVariation چون شرط مستقیمی روی لبه ها نمیگذارد نسبت به بقیه نیز کمی کمتر شده است. توجه داریم که در bilateral مستقیما کاری میکنیم که روی لبه با سمت دیگر آن مخلوط نشود.(البته نتایج بالا وابسته به پیاده سازی اینجانب است و ممکن است در حالات ایده‌آل برای هرکدام نتایج بهتری دریافت کرد که نتیجه را تغییر دهد.)