به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

تمرین کامپیوتری ۳

پردازش سیگنال های دیجیتال (DSP)

دکتر بدیعی

عرفان پناهی ۸۱۰۱۹۸۳۶۹

نیمسال اول ۰۱-۱۴۰۰

فهرست:

*** فایل مربوط به این تمرین کامپیوتری با نام CA3_ Matlab_810198369.mlx پیوست شده است.

چکیده : هدف از تمرین کامپیوتری ۳

هدف از انجام این تمرین بررسی برخی فیلتر های دیجیتال و یکی از کاربرد های تبدیل کپستروم است. در بخش اول (پردازش صوت) سعی می‌کنیم یک صوت اکویافته و نویز دار را دریافت کرده و در دو مرحله ابتدا اکو را به کمک خاصیت تبدیل کپستروم و سپس نویز را با استفاده از فیلتر های FIR حذف کنیم. برای حذف اکو همانطور که اشاره شد از تبدیل کپستروم استفاده میکنیم که در تمرین کامپیوتری ۲ آنرا مفصل بررسی کردیم. در این تمرین کامپیوتری یکی از خواص آنرا اثبات و به کمک آن اکو را حذف میکنیم. برای حذف و محدود کردن نویز از فیلتر های FIR استفاده کرده و به کمک یکی از این فیلتر ها که برای مسئله ما مناسب تر است نویز را تا حد خوبی کاهش میدهیم. در بخش دوم (پردازش تصویر) با کرنل ها آشنا میشویم. کرنل ها کاربرد های گسترده ای در دستکاری تصاویر دارند که به برخی از آنها در ابتدای بخش دوم اشاره میکنیم. در نهایت و در بخش امتیازی نیز با استفاده از خاصیت یکی از کرنل ها سعی میکنیم برنامه ای بنویسیم تا بتواند گوشه های یک کاغذ را تشخیص داده و یک مستطیل دور کاغذ رسم نماید.

بخش ۱ : پردازش سیگنال صوت

*** فایل صوتی ورودی مربوط به این قسمت با نام input_voice.wav پیوست شده است.

در ورودی این بخش از فایل صوتی **input_voice.wav** با محتوای شعر زیر استفاده می‌کنیم:

چه سرنوشت غم انگیزی که کرم کوچک ابریشم تمام عمر قفس بافت، ولی به فکر پریدن بود

«حسین منزوی»

فرکانس ورودی این صوت مطابق تصویر ۱-۱ برابر $22050Hz$ میباشد. با توجه به اینکه فایل صوتی ورودی ۱۰ ثانیه است، پس تعداد نمونه ها گسسته 220500 تا ($0 \le n < 220500$) خواهد بود.

Name	Value
fs	22050
input	220500x1 double

تصویر ۱-۱ – **فرکانس نمونه برداری و طول فایل صوتی ورودی**

ابتدا با استفاده از رابطه زیر یک اکو به ورودی اعمال می‌کنیم. مقادیر $\alpha = 0.8$ و $\beta = 4410$ در نظر میگیریم. با توجه به فرکانس نمونه برداری میتوانیم بگوییم تأخیر اکو در حوزه پیوسته و چیزی که مشنویم برابر ۰,۲ ثانیه است.

$$y[n] = x[n] + \alpha x[n-\beta] \rightarrow y[n] = x[n] + 0.8x[n-4410]$$

سوال ۱ – فیلتر تک اکو

نوشتن تابع برای محاسبه تبدیل cepstrum

برای نوشتن این تابع یکبار با استفاده از fft() تبدیل فوریه سیگنال ورودی را گرفته و سپس از این تبدیل لگاریتم در مبنای ۱۰ میگیریم و در نهایت با استفاده از ifft() وارون آنرا محاسبه کرده و بصورت خروجی برمیگردانیم. تابع rceps() نیز همین کار را انجام می‌دهد.

در این بخش مطابق توضیحات داده شده در صورت تمرین، ابتدا تبدیل کپستروم صوت نویز و اکودار را رسم میکنیم. همانطور که در تصویر ۲-۱ نیز مشاهده میشود بزرگترین قلّه بعد از نویز های حول صفر در تأخیر اکو یعنی ($\beta + 1$) اتفاق افتاده است و به این ترتیب میتوانیم β را بدست آوریم. با استفاده از روابط زیر میتوانیم این موضوع را اثبات کنیم:

اگر ورودی بدون نویز و بدون اکو را $x[n]$ در نظر بگیریم (مطابق با روابط صفحات ۹۸۱ و ۹۸۲ کتاب) داریم:

$$y[n] = x[n] + \alpha x[n-\beta] \rightarrow Y(e^{j\omega}) = (1 + \alpha e^{-j\beta\omega})X(e^{j\omega})$$

$$|Y(e^{j\omega})| = |X(e^{j\omega})|\sqrt{1 + \alpha^2 + 2\alpha\cos(\beta\omega)}$$

$$\xrightarrow{log} C_y(e^{j\omega}) = C_x(e^{j\omega}) + \frac{1}{2}\log(1 + \alpha^2 + 2\alpha\cos(\beta\omega))$$

با توجه به اینکه تبدیل فوریه گسسته متناوب با دوره تناوب 2π است، پس تبدیل کپستروم متناوب خواهد بود:

$$c_y[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} C_y(e^{j\omega})e^{j\omega n}d\omega = \int_{-\frac{1}{2}}^{\frac{1}{2}} C_y(e^{j2\pi f})e^{j2\pi f n}df$$

$$= \int_{-\frac{1}{2}}^{\frac{1}{2}} \left[C_x(e^{j\omega}) + \frac{1}{2}\log(1+\alpha^2+2\alpha\cos(\beta\omega))\right] e^{j2\pi f n}df$$

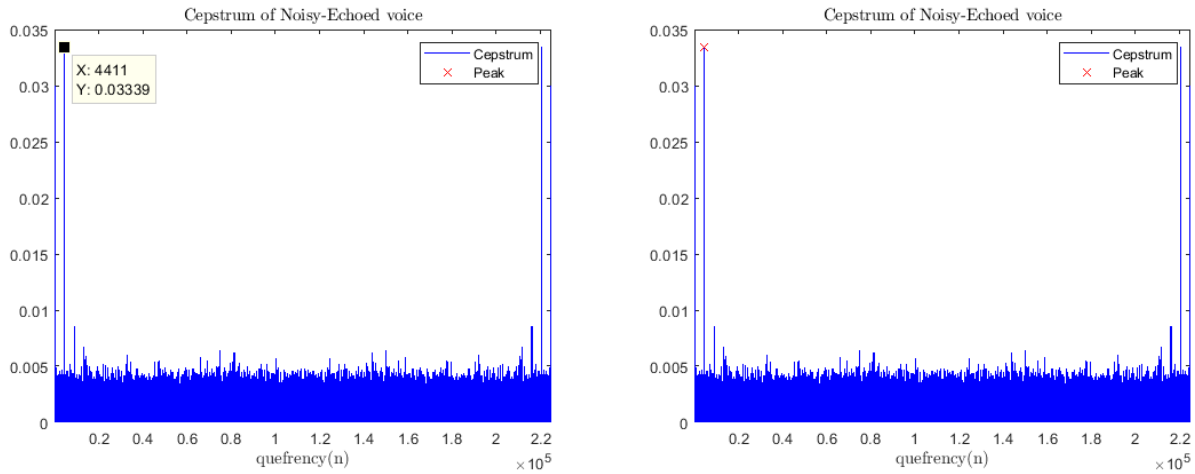
$$= c_x[n] + \sum_{k=1}^{\infty}(-1)^{k+1}\frac{\alpha^k}{2k}(\delta[n+k\beta]+\delta[n-k\beta])$$

عبارت داخل سیگما فقط در ضرایب تأخیر یعنی $n = \pm k\beta$ مقدار دارد. با توجه به اینکه $0 \le \alpha < 1$ است، عبارت سیگما بزرگترین مقدار را در $n = \beta$ و به ازای $k = 1$ دارد.

اگر نویز را در نظر بگیریم با توجه به اینکه نویز خود را در quefrency های حول صفر نشان میدهد، میتوانیم از ۱۰ نمونه اول در quefrency صرف نظر کنیم.

تصویر ۲-۱ اندازه تبدیل کپستروم صوت نویز دار را با صرف نظر از ۱۰ داده اول و آخر نشان میدهد. همانطور که مشاهده میشود پیک در $n = 4411$ معادل با $\beta + 1$ اتفاق می‌افتد. (دلیل اینکه ۱ واحد اضافه شد این است که متلب از اندیس ۱ شماره گذاری را انجام میدهد.)

$$\beta + 1 = 4411 \rightarrow \beta = 4410 \rightarrow Delay = \frac{\beta}{f_s} = \frac{4410}{22050} = 0.2\ sec$$



تصویر۲-۱ – **بدست آوردن تأخیر اکو با استفاده از کپستروم**

* **توضیحات تابع حذف کننده اکو:** این تابع در انتهای فایل متلب با نام CancelEcho() نوشته شده و شامل دو بخش است. یک بخش شامل تشخیص تأخیر اکو که اثبات روابط آن بالا توضیح داده شد و بخش دیگر شامل حذف اکو. در بخش تشخیص

تأخیر اکو (β) از دستور max() و find() برای پیدا کردن نقطه ماکزیمم و اندیس آن استفاده می‌کنیم. سپس در بخش حذف اکو با در دست داشتن پارامتر های تأخیر α و β و با استفاده از دستور filter() که در تمرین کامپیوتری ۱ با آن آشنا شدیم، میتوانیم اکوی ایجاد شده را حذف کنیم. آرگومان های این دستور به صورت زیر خواهد بود.

$$y[n] = x[n] + \alpha x[n-\beta] \rightarrow h[n] = \delta[n] + \alpha\delta[n-\beta]$$

با توجه به اینکه ورودی مجهول و خروجی معلوم است و با یک فیلتر IIR سر و کار داریم آرگومان اول 1 و آرگومان دوم h[n] و آرگومان سوم صوت اکودار خواهد بود.

تصویر ۳-۱ تأخیر خروجی این تابع را نشان میدهد. همانطور که مشاهده میشود محاسبه تأخیر به درستی صورت گرفته است.

```
fprintf('Beta = %d \nReconstructed Beta =  %d \nDelay in time domain = %.1f sec',Beta,BetaRec,Beta/fs);
```

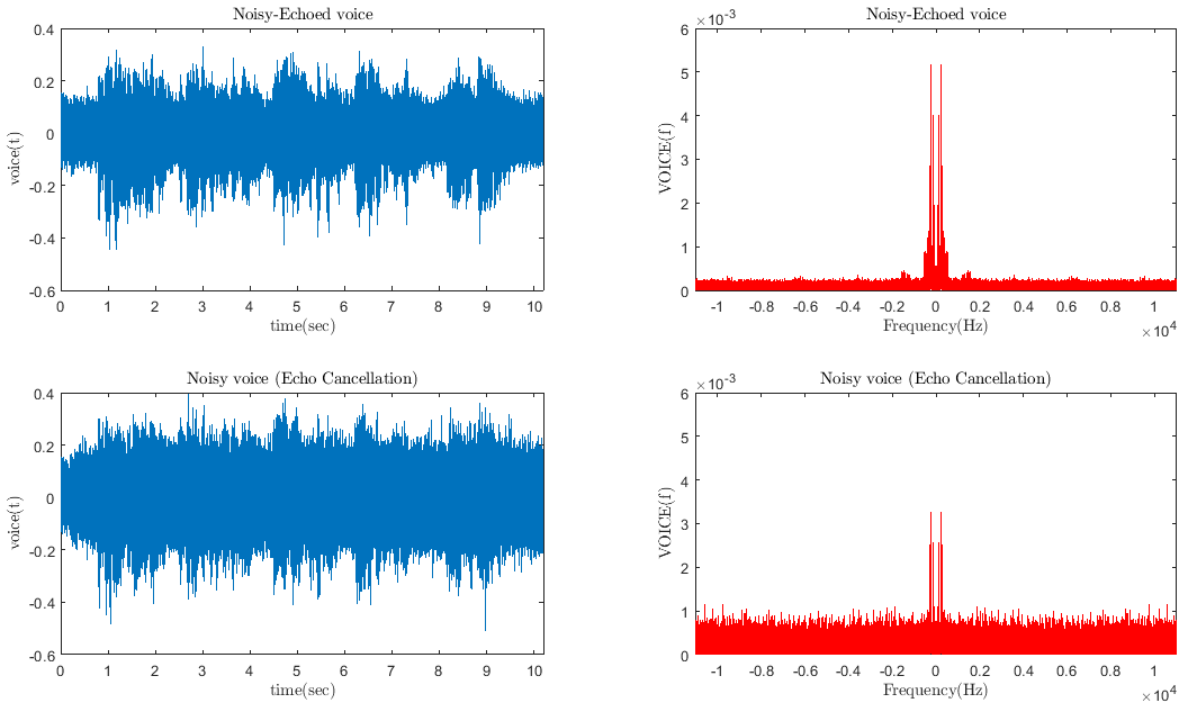
```
Beta = 4410
Reconstructed Beta =  4410
Delay in time domain = 0.2 sec
```

تصویر۳-۱ – خروجی تابع **CancelEcho()** و مقایسه با تأخیر اعمالی اولیه

* **دامنهٔ زمانی و فرکانسی صوت اکودار و خروجی تابع:** با استفاده از دستور fft() و fftshift() تبدیل فوریه سیگنال های اکودار و خروجی تابع را بدست آورده و اندازه آن را مطابق با تصویر ۴-۱ رسم میکنیم.

*** دقت شود که وقتی برای محاسبه تبدیل فوریه پیوسته از دستور **fft(fftshift())** استفاده میکنیم باید خروجی را بر تعداد سمپل ها تقسیم میکنیم تا نرمالیزه شود. (علی رغم اینکه در برخی منابع نوشته شده که باید بر فرکانس نمونه برداری تقسیم شود اما با توجه به اینکه با DFT سر و کار داریم باید طبق روابط آن را بر تعداد نمونه تقسیم کنیم.)

```
Xf = fftshift(fft(x))/length(x);
```



تصویر۴-۱ – دامنهٔ زمانی و فرکانسی صوت اکودار و خروجی تابع

همانطور که در تصویر مشاهده میشود زمانی که اکو وجود دارد در حوزه زمان هر بخش تقریباً(بخاطر وجود نویز) 0.2 ثانیه بعد تکرار میشود. در حوزه فرکانس نیز بخاطر اندازه تبدیل در فرکانس ها مختلف بخاطر حذف اکو تضعیف و کم میشود. همچنین بخاطر اینکه dc سیگنال $1-\alpha$ برابر شده است در فرکانس حول صفر که dc را نشان میدهد حذف اکو را متوجه میشویم. لازم به ذکر است بخاطر فیلتری که استفاده شده علی رغم حذف اکو، نویز تقویت شده و SNR کاهش یافته است.

سوال ۲ – حذف نویز به کمک فیلتر های FIR

* **فرکانس سیگنال صوت انسان و پارامتر های این سیگنال:** در حالت عادی و بدون نویز فرکانس سیگنال گفتار انسان در حدود $300-3400Hz$ می‌باشد. با دخالت نویز این محدوده تغییر پیدا می‌کند و به همین خاطر شاهد تقلیل کیفیت صوت خواهیم بود. با استفاده از فیلتر میتوانیم نویز را محدود کنیم اما اگر بخواهیم همراه با محدود کردن نویز کیفیت صوت اصلی را هم افزایش دهیم نیاز داریم تا شاخص ها فیلتر را به درستی تعیین کنیم. (فیلتر مناسبی را انتخاب کنیم.) به این منظور باید پارامتر های دیگر سیگنال صوت را مشخص سازیم و سپس به کمک آن فیلترمان را طراحی کنیم. پارامتر های دیگر سیگنال صوت انسان به صورت زیر می‌باشد:

Passband Edge: $f_p = 3400Hz$
Stopband Edge: $f_s = 3800Hz$
stopband attenuation is less than 80dB $\equiv A_{stop} \leq -80dB$
passband ripple is less than 3dB $\equiv A_{pass} \leq -3dB$

با توجه به جدول ۱-۱ مشاهده میکنیم که تنها با استفاده از فیلتر Kaiser میتوانیم $A_{stop} \leq -80dB$ را برآورده سازیم.

Table1: Comparison of basic parameters of commonly used window functions

Window Function	Side Lobe Peak Range/dB	Transition Band	Minimum Attenuate in Stopband/dB
Rectangular Window	-13	4π/N	-21
Triangular Window	-25	8π/N	-25
Hanning Window	-31	8π/N	-44
Hamming Window	-41	8π/N	-53
Blackman Window	-57	12π/N	-74
Kaiser Window (α=7.685)	-57	10π/N	-80

Table2 Influence of Kaiser Window Parameters on Filter Performance

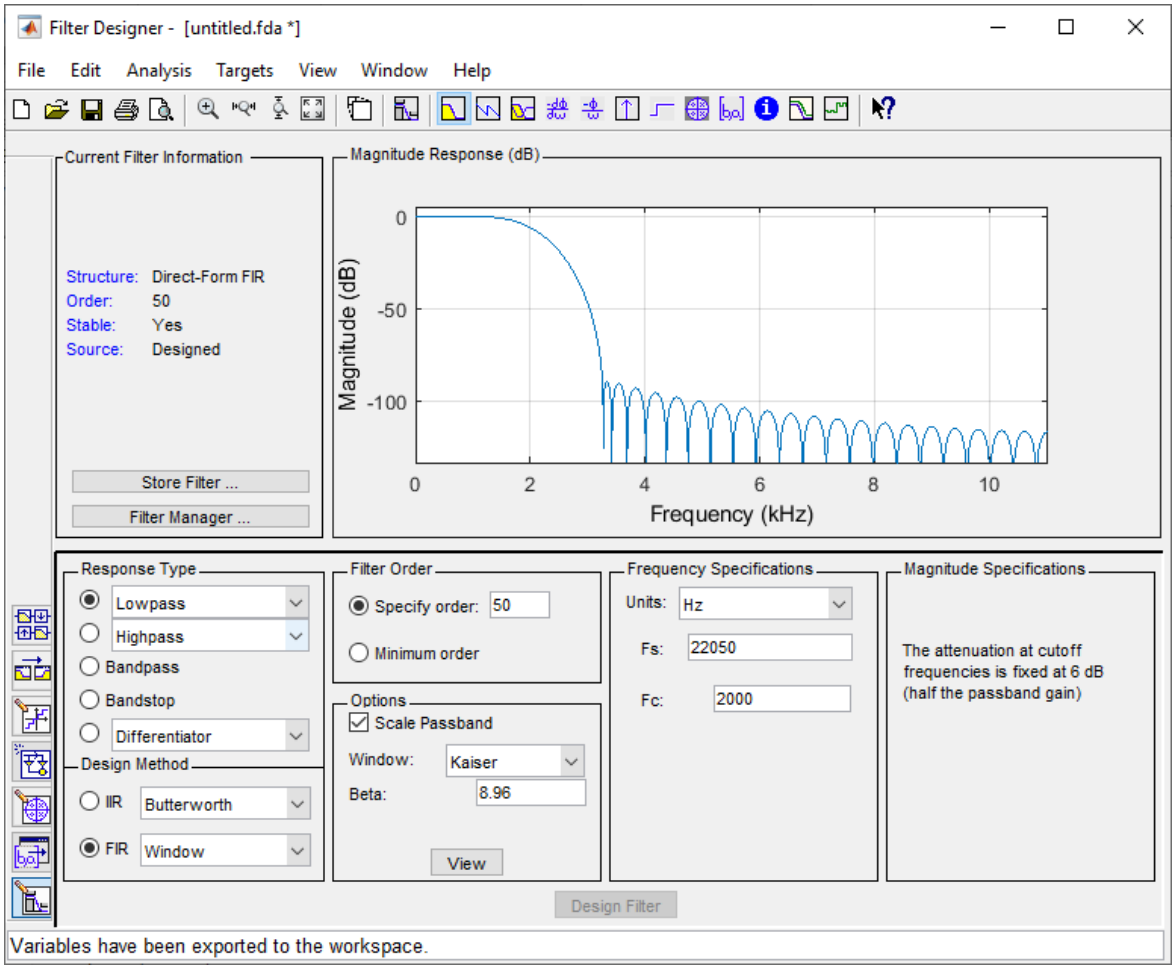
β	Transition Band	Passband Ripple/dB	Minimum Attenuate in Stopband/dB
2.120	3.00π/N	±0.27	-30
3.384	4.46π/N	±0.0868	-40
4.538	5.86π/N	±0.0274	-50
5.658	7.24π/N	±0.00868	-60
6.764	8.64π/N	±0.00275	-70
7.865	10.0π/N	±0.000868	-80
8960	11.4π/N	±0.000275	-90
10.056	12.8π/N	±0.000087	-100

جدول۱-۱ و ۲-۱

مطابق با جدول ۲-۱ مقدار $\beta = 8.96$ را در نظر میگریم تا هر دو خواسته، $A_{stop} \leq -80dB$ و $A_{pass} \leq -3dB$ برآورده شود.

سپس با استفاده از ابزار *fdatool* باقی پارامتر ها را با آزمون و خطا طوری تعیین میکنیم تا باقی مشخصات فیلتر برآورده شود.

ابتدا یک فیلتر پایین‌گذر *low-pass* طراحی میکنیم. مطابق تصویر۵-۱ با انتخاب پارامتر های مناسب دو ویژگی لبه های عبور و توقف فیلتر یعنی $f_p = 3400Hz$ و $f_s = 3800Hz$ را به طور تقریبی برآورده میسازیم.



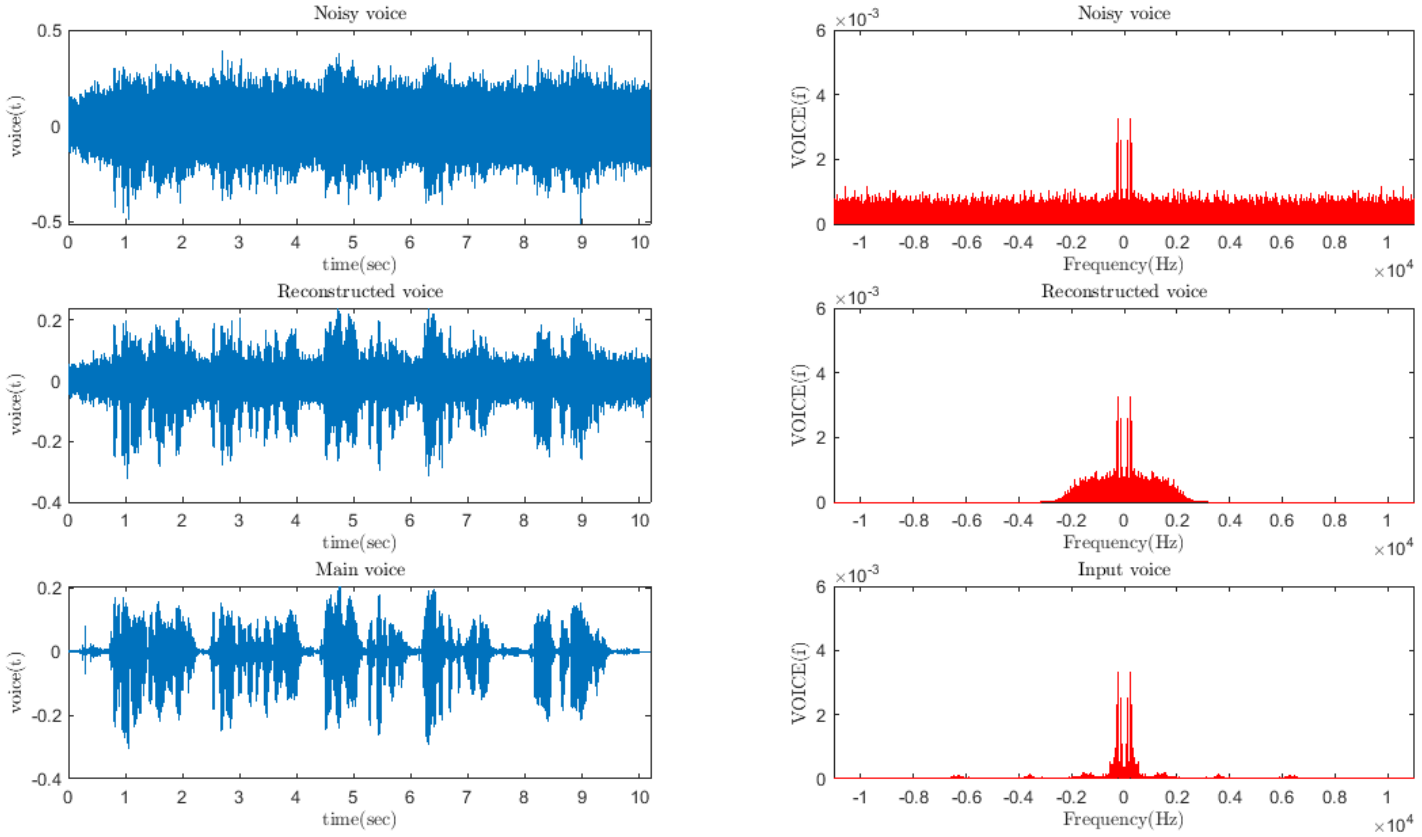
تصویر۵-۱ – **انتخاب پارامتر های مناسب برای برآورده سازی فیلتر پایین گذر دیجیتال**

سپس این فیلتر طراحی شده را در workspace متلب Export کرده و با سیگنال بدست آمده از بخش قبلی کانوالو میکنیم. (با استفاده از دستور conv()). درنهایت وقتی خروجی را میشنویم متوجه محدود شدن نویز میشویم. دقت شود که قرار نیست نویز به طور کامل حذف شود بلکه باید صرفاً نویز را محدود کنیم تا SNR صوت خروجی افزایش یابد.

سپس از سیگنال خروجی تبدیل فوریه گرفته و با دو صوت اولیه و نویز دار مقایسه میکنیم. (تصویر ۶-۱)

مقایسه با سیگنال اولیه: همانطور که مشاهده میشود سیگنال بازیابی شده در دامنه زمانی شباهتی با سیگنال اصلی دارد. اما در حوزه فرکانس همانطور که شاهد هستیم نویز ها به طور کامل حذف نشده و دلیل این امر این است که صوت اصلی پهنای فرکانسی کمی را پوشش داده است و بخاطر همین سیگنال بازیابی شده باز هم نویز دارد. از طرف دیگری برخی فرکانس های بالاتر نیز حذف شده و دلیل ضعیف تر شدن صوت اصلی نیز میتواند همین موضوع باشد.

مقایسه با سیگنال نویز دار: همانطور که انتظار میرفت پهنای باند سیگنال صوتی در حوزه فرکانسی فیلتر شده است. با مقایسه این دو سیگنال میتوانیم به نوع و شکل فیلتر نیز پی ببریم. اگر به حوزه زمان نیز دقت کنیم متوجه کاهش یافتن نویز های خروجی فیلتر میشویم.



تصویر۶-۱ – **مقایسه حوزه زمان و فرکانس سیگنال بازیابی شده با سیگنال های نویز دار و اولیه**

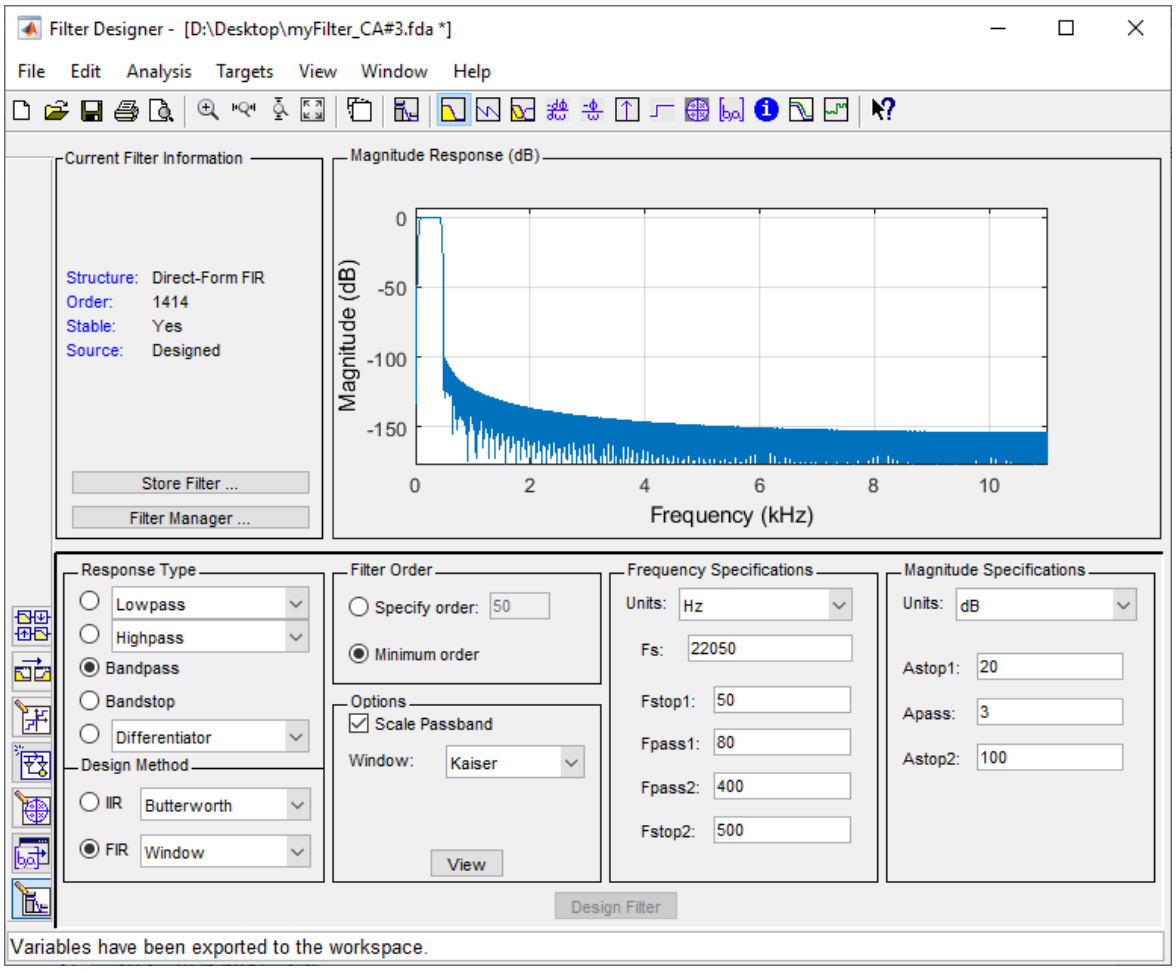
حال میتوانیم یک فیلتر میان گذری طراحی کنیم که فرکانس قطع آن معادل با پهنای بند سیگنال اولیه و فرکانس قطع اول آن طوری باشد که نویز dc حذف شود. به این منظور مطابق تصویر ۷-۱ پارامتر های فیلتر را برای برآورده سازی خواسته های مدنظر تعیین میکنیم.

با شنیدن صوت خروجی متوجه ضعیف شدن بیش از حد آن میشویم. دلیل این امر حذف شدن برخی از فرکانس های اصلی صوت اولیه است.

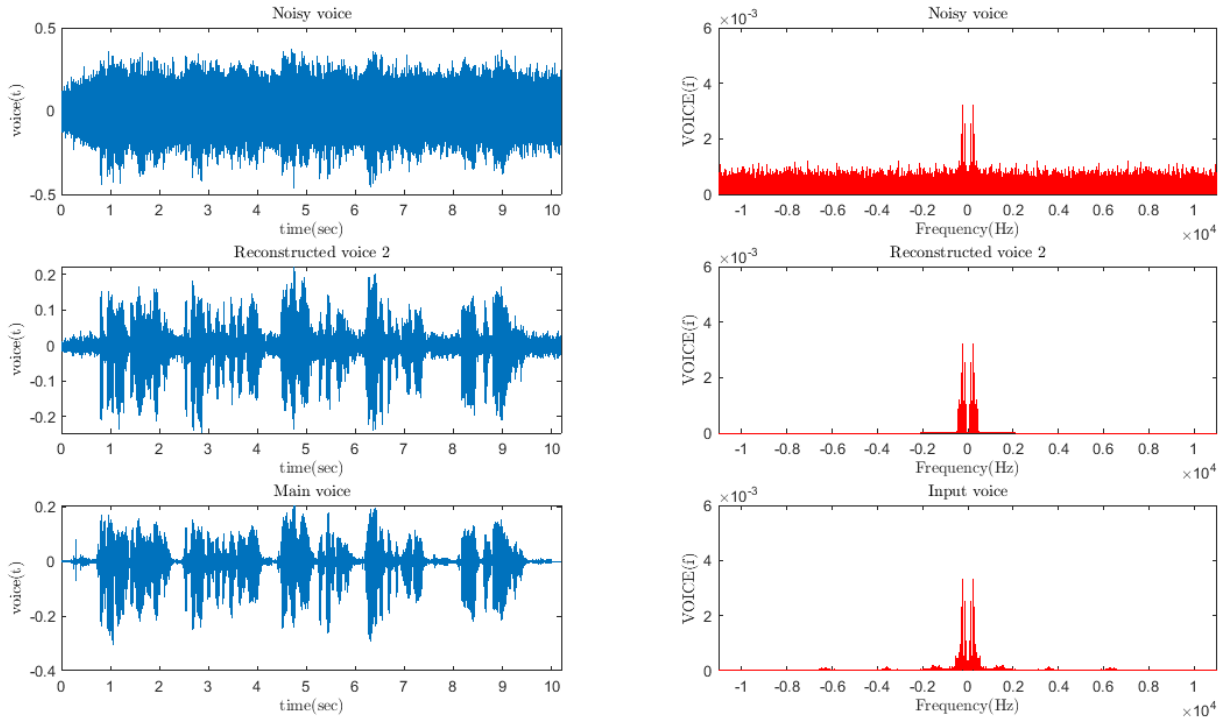
سپس از سیگنال خروجی تبدیل فوریه گرفته و با دو صوت اولیه و نویز دار مقایسه میکنیم. (تصویر ۸-۱)

مشاهده میشود که نویز سیگنال تا حد خوبی حذف شده است اما همانطور که گفته شد در حوزه زمان سیگنال بازیابی شده دارای دامنه کمتری نسبت به سیگنال اولیه است و نشان از تضعیف این سیگنال دارد.

برای جبران سازی این مورد میتوانیم از یک تقریب کننده استفاده کنیم تا دامنه آنرا به حدود دامنه صوت اولیه برساند. در فایل متلب این تمرین این موضوع بررسی شده است. اما یک مشکل دیگر این است که برخی از فرکانس های بالا توسط فیلتر از بین رفته است. یعنی علی رغم اینکه نویزی نداریم ولی برخی اطلاعات نیز به همراه نویز حذف شده است.



تصویر۷-۱ – انتخاب پارامتر های مناسب برای برآورده سازی فیلتر میان گذر دیجیتال



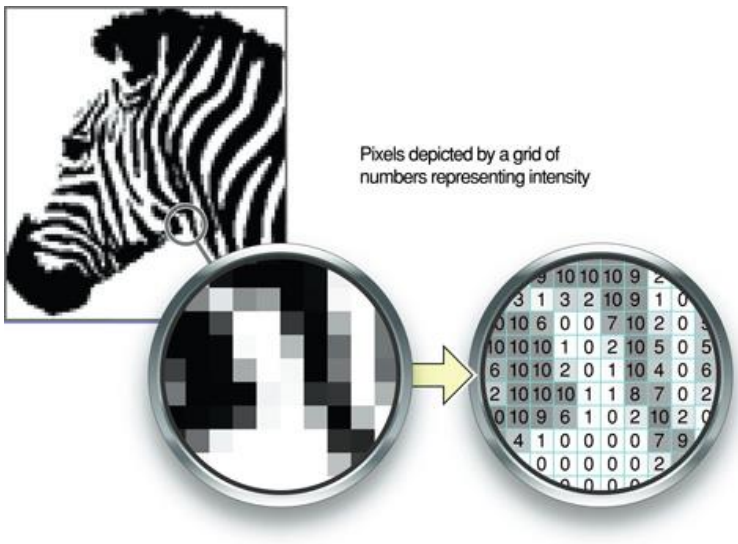
تصویر۸-۱ – مقایسه حوزه زمان و فرکانس سیگنال بازیابی شده با سیگنال های نویز دار و اولیه

بخش ۲ : پردازش تصویر

قسمت ۱ – بررسی کرنل های مختلف و تأثیر آنها

ابتدا درباره کرنل ها توضیح مختصری میدهیم و سپس خواسته مسئله را برآورده میسازیم و کرنل ها را روی یک تصویر خاص اعمال میکنیم.

همانطور که میدانیم یک تصویر سیاه و سفید از یک ماتریس ساخته شده است که عناصر داخل آن رنگ تصویر را نشان میدهد. اگر تصویر رنگی باشد نیاز به ۳ ماتریس داریم که هرکدام رنگ های اصلی را پوشش دهد و با مقدار آنها میتوان رنگ هر پیکسل را متمایز ساخت.



تصویر ۱-۲ – **تصویر یک ماتریس است.**

یک کرنل ماتریسی عموماً مربعی است که با اعمال آن روی تصاویر میتوان کیفیت تصویر را بهبود داد یا استفاده مورد نظر را از خروجی انجام داد. منظور از اعمال روی تصاویر لغزاندن آن روی ماتریس یک تصویر و بعبارت دیگر کانوالو (به صورت دو بعدی) کردن آن در ماتریس یک تصویر است. یک انیمیشن در خصوص طریقه اعمال یک کرنل روی ماتریس یک تصویر با نام **3D_Convolution_Animation.gif** پیوست شده است.

در صورت تمرین کامپیوتری سوم ۸ نوع کرنل معرفی شده است که مختصراً تأثیر هرکدام بر تصویر را بررسی میکنیم.

۱. **فیلتر identity** : کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، پیکسل خروجی از هیچکدام از پیکسل های مجاور اثر نمیگیرد و در نتیجه این کرنل تأثیری روی تصویر اصلی نخواهد داشت.

0	0	0
0	1	0
0	0	0

کرنل identity

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
WindowSize = 3;
Identity = zeros(WindowSize); Identity(2,2) = 1;
```

۲. فیلتر $V-line$: کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، پیکسل های هم ستون تقویت و پیکسل های ستون های مجاور از آن کم میشود. از این کرنل برای تشخیص لبه های عمودی استفاده میشود.

-1	2	-1
-1	2	-1
-1	2	-1

کرنل $V-line$

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
Vline = -1 * ones(WindowSize); Vline(:,2) = 2;
```

۳. فیلتر $H-line$: کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، پیکسل های هم سطر تقویت و پیکسل های سطر های مجاور از آن کم میشود. از این کرنل برای تشخیص لبه های افقی استفاده میشود.

-1	-1	-1
2	2	2
-1	-1	-1

کرنل $H-line$

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
Hline = -1 * ones(WindowSize); Hline(2,:) = 2;
```

۴. فیلتر Moving Average: کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، هر پیکسل تصویر اصلی با میانگین پیکسل های مجاور جایگزین میشود. از این فیلتر برای محو (تار) کردن عکس استفاده میشود.

$$\frac{1}{9} \times \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

کرنل Moving Average

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
MovingAvg = ones(WindowSize)/WindowSize^2;
```

۵. فیلتر Guassian: کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در همه پیکسل ها تضعیف میشود و به همین خاطر شاهد شفاف تر شدن عکس خواهیم بود. این فیلتر عملکردی مشابه فیلتر

Moving Average دارد با این تفاوت که نسبت تضعیف در این فیلتر به گونه ای است که پیکسل های مجاور تأثیر کمتری خواهند داشت. هرچه انحراف معیاری کرنل گوسی بیشتر شود این کرنل بیشتر به Moving Average شبیه میشود. از این فیلتر برای بهبود کیفیت تصاویر نیز استفاده میشود.

0.0113	0.0838	0.0113
0.0838	0.6193	0.0838
0.0113	0.0838	0.0113

کرنل Gaussian

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
Sigma = 0.5;
Gauss = fspecial('gaussian',[WindowSize WindowSize],Sigma);
```

۶. فیلتر Outline : کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، پیکسل اصلی تقویت و پیکسل های مجاور از آن کم میشود و بعبارت دیگر وابستگی دو پیکسل مجاور کمتر میشود و درنتیجه لبه های عکس و جاهایی که شاهد تغییرات ناگهانی رنگ هستیم بیشتر نمایان میشود.

-1	-1	-1
-1	8	-1
-1	-1	-1

کرنل Outline

برای تولید این کرنل از دستور زیر استفاده میکنیم:

```
Outline = -1 * ones(WindowSize); Outline(2,2) = 8;
```

۷. فیلتر Blur : کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، همه پیکسل ها تضعیف میشود و به همین خاطر شاهد شفاف تر شدن عکس خواهیم بود. البته نسبت تضعیف در این فیلتر به گونه ای است که پیکسل های مجاور تأثیر کمتری خواهند داشت.

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

کرنل Blur

برای تولید این کرنل بصورت زیر عمل میکنیم:

```
Blur = [1/16 1/8 1/16;
        0/8  1/2  1/8;
        1/16 1/8 1/16];
```

۸. فیلتر Sharpen : کرنل 3×3 این فیلتر بصورت زیر تعریف میشود. همانطور که مشاهده میشود با کانوالو شدن تصویر در کرنل، هر پیکسل تقویت و ۴ پیکسل مجاور از آن کم میشود و وابستگی به پیکسل های کناری از بین میرود و به همین خاطر در تصویر خروجی شاهد برجسته شدن لبه ها خواهیم بود.

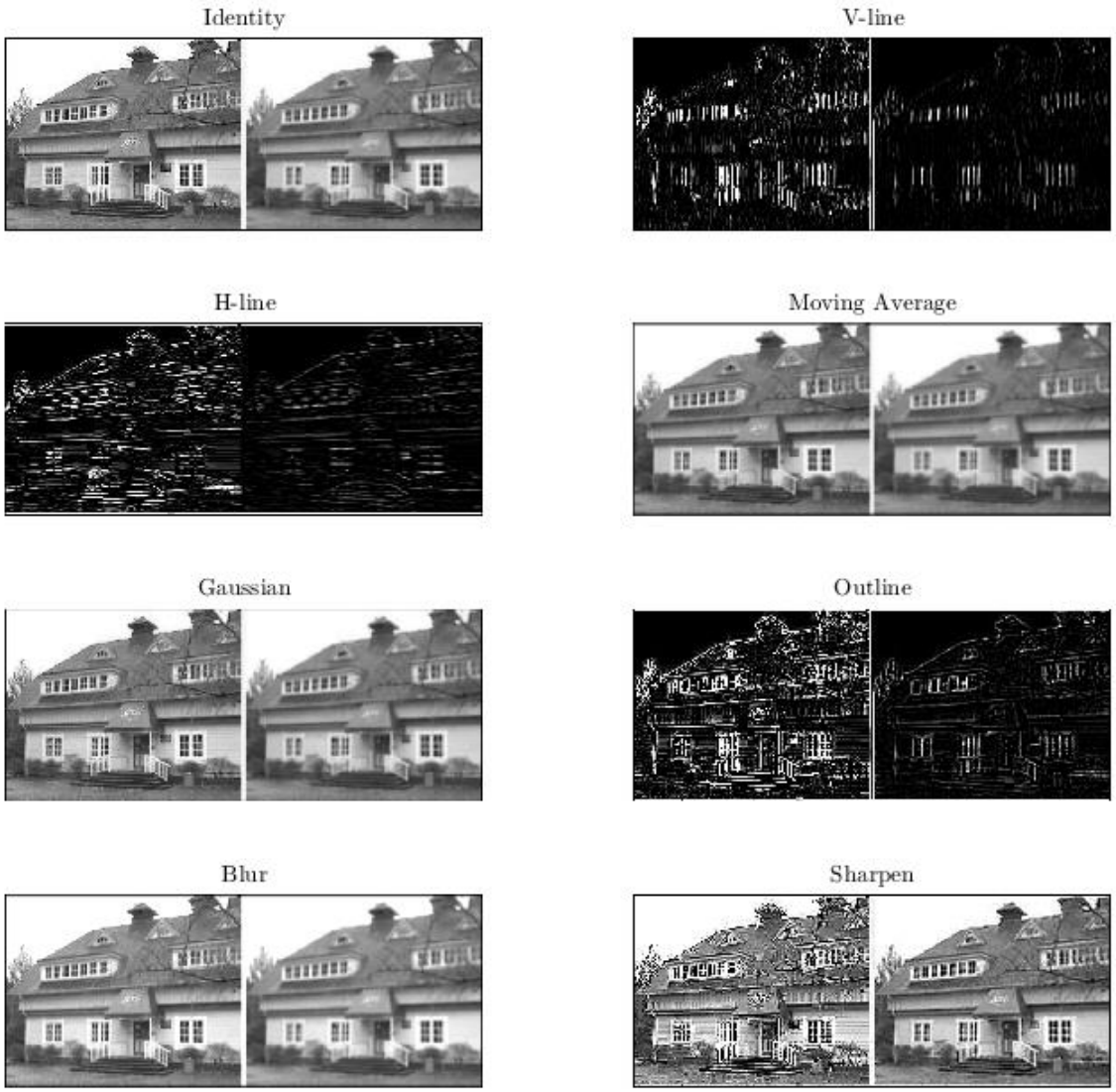
0	-1	0
-1	5	-1
0	-1	0

کرنل Sharpen

برای تولید این کرنل بصورت زیر عمل میکنیم:

```
Sharpen = [0 -1 0;
           -1 5 -1;
           0 -1 0];
```

تصویر ۲-۲ خروجی اعمال هر کرنل روی تصویر اصلی را نشان میدهد.



تصویر۲-۲ – **خروجی اعمال هر کرنل روی تصویر house.jpg**

قسمت ۲ – تغییر اندازه تصویر

تصویر ۲-۳ ، کیفیت اصلی تصویر kobe.jpeg بصورت سیاه و سفید را نشان می‌دهد.

تصویر۳-۲ – **کیفیت اصلی تصویر kobe.jpeg بصورت سیاه و سفید**

با استفاده از دستور imresize() اندازه تصویر را با اسکیل $\frac{1}{5}$ کاهش میدهیم. خروجی در تصویر ۲-۴ نشان داده شده است.

تصویر۴-۲ – **کاهش اندازه تصویر kobe.jpeg با اسکیل $\frac{1}{5}$**

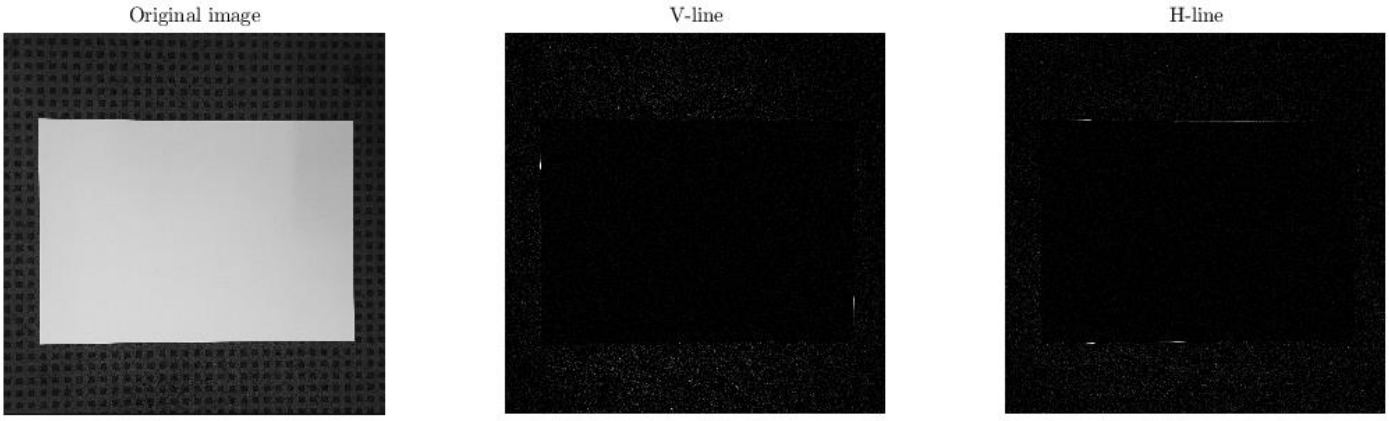
سپس مجدداً با استفاده از دستور imresize() اندازه تصویر را با اسکیل 5 افزایش میدهیم. دلیل این امر کاهش کیفیت تصاویر با یکسان سازی پیکسل های مجاور است. زمانی که تصویر کوچک میشود، نمونه برداری از تصویر اصلی صورت میگیرد و تعداد پیکسل ها کمتر میشود. با بزرگتر شدن تصویر تعداد پیکسل ها افزایش می‌یابد و نیاز داریم مقادیر پیکسل های اضافه شده را اضافه کنیم. برای اینکار علاوه بر اینکه در بزرگسازی از $method = nearest$ برای بازیابی استفاده میکنیم ، از دو فیلتر گوسی و Moving Average نیز برای بازسازی و افزایش کیفیت تصویر استفاده میکنیم.

تصویر ۵-۲ تصاویر بزرگسازی شده و خروجی فیلتر های گوسی و Moving Average را نشان میدهد. همانطور که مشاهده میشود هر دو فیلتر کمی تصویر را شفاف تر میکنند. (خروجی Moving Average کمی تار تر است.)

تصویر۵-۲ – **بازیابی و افزایش کیفیت تصویر**

قسمت امتیازی – مشخص سازی لبه های کاغذ

در این بخش ابتدا با استفاده از کرنل های $H-line$ و $V-line$ لبه های کاغذ را تشخیص میدهیم. تصویر ۶-۲ خروجی این دو فیلتر را نشان میدهد.



تصویر۶-۲ – **تشخیص لبه ها**

ایده مشخص کردن لبه های کاغذ: همانطور که مشاهده میشود لبه های افقی و عمودی به این دو فیلتر با رنگ سفید مشخص میشود. همانطور که در تمرین کامپیوتری۲ بررسی شد، نقاط سفید مقدار بزرگتری دارند و در نتیجه میتوانیم بگوییم اگر از سطر ها و ستون های مجموع بگیریم، لبه های سفید بزرگترین مقدار را دارند. در نتیجه دو لبه افقی و دو لبه عمودی را به همین ترتیب میتوانیم بدست آوریم. محل برخورد این ۴ لبه ، ۴ نقطه گوشه کاغذ را به ما میدهد. در نهایت با پیدا کردن این ۴ نقطه و با استفاده از دستور rectangle() مستطیل لبه های تصویر را رسم می‌کنیم.

تابع نوشته شده به صورت زیر میباشد. در این تابع ماتریس تصویر بعنوان ورودی گرفته شده و مستطیل روی لبه های آن رسم میشود.

```
function ObjLoc(pageRGB)
```

ابتدا تصویر را سیاه و سفید کرده و سپس با استفاده از دو فیلتر$H-line$ و $V-line$ ، تصویر اولیه را فیلتر میکنیم.

```
    page = rgb2gray(pageRGB);
```

```
    WindowSize = 3;
    Hline = -1 * ones(WindowSize); Hline(2,:) = 2;
    Vline = -1 * ones(WindowSize); Vline(:,2) = 2;
    Hline_page = conv2(page,Hline);
    Vline_page = conv2(page,Vline);
```

حال مجموع ستون ها و سطر ها را حساب می‌کنیم:

```
    Hvalues = sum(abs(uint8(Hline_page')));
    Vvalues = sum(abs(uint8(Vline_page)));
```

ماکزیمم سطر و ستون و اندیس آنرا بدست می‌آوریم:

```
    Hmax = maxk(Hvalues(10:length(Hvalues)-10),2);
    Vmax = maxk(Vvalues(10:length(Vvalues)-10),2);
    H(1) = find(Hvalues == Hmax(1)); H(2) = find(Hvalues == Hmax(2));
    V(1) = find(Vvalues == Vmax(1)); V(2) = find(Vvalues == Vmax(2));
    H1 = min(H); H2 = max(H); V1 = min(V); V2 = max(V);
```

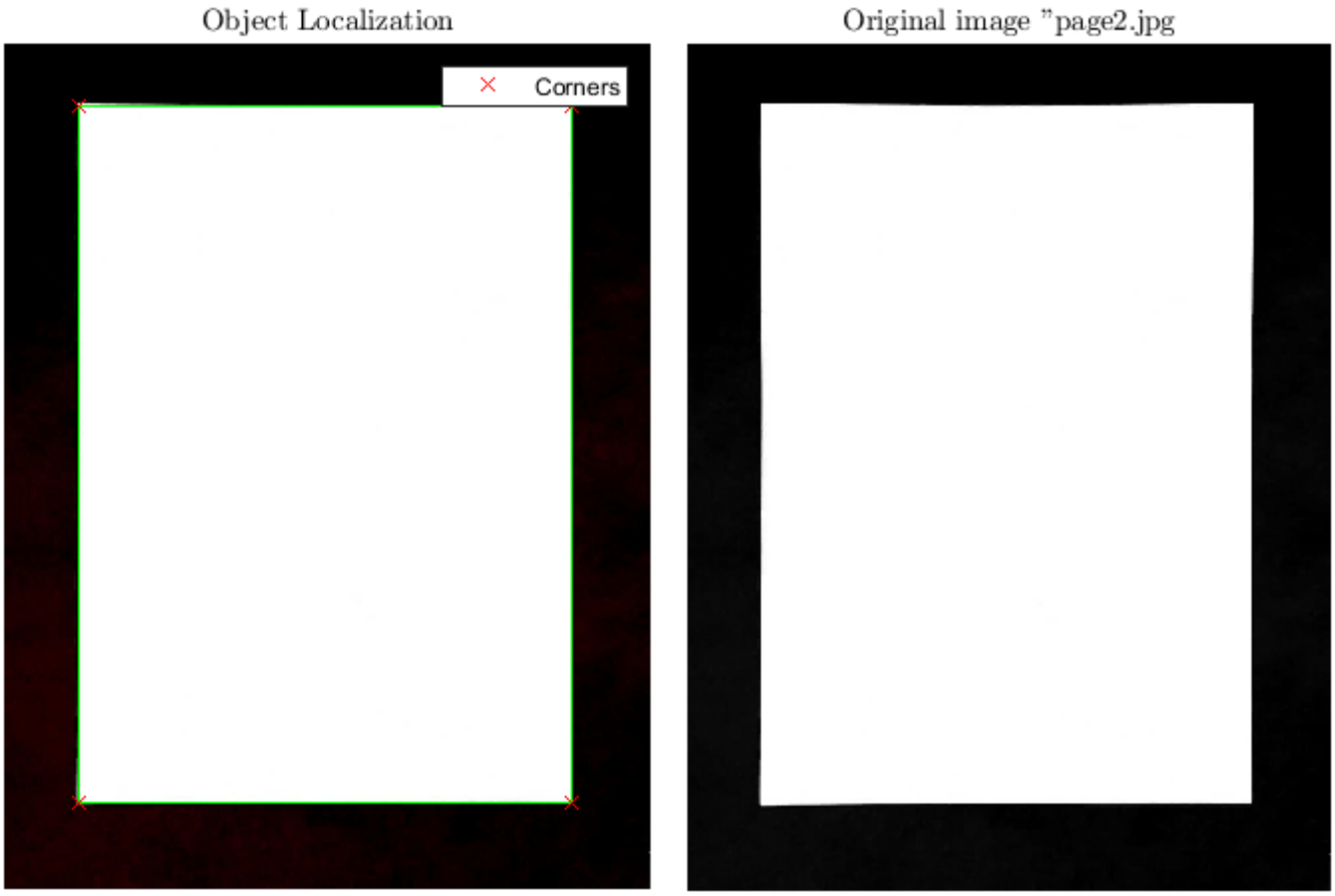
در نهایت با در دست داشت اندیس لبه های تصویر مستطیل روی لبه ها را رسم میکنیم:

```
    imshow(pageRGB);
    hold on
    plot([V1 V2 V1 V2],[H1 H1 H2 H2],'rx');
    rectangle('Position',[V1 H1 V2-V1 H2-H1],'EdgeColor','g');
    title('Object Localization','Interpreter','latex');
    legend('Corners');
    hold off
end
```

تصویر ۷-۲ خروجی ایت تابع را نشان میدهد.

تصویر۷-۲ – **خروجی تابع نوشته شده برای تشخیص لبه های کاغذ تصویر page.jpg**

در پایان نیز از یک تصویر دیگر برای تست کردن تابع نوشته شده استفاده میکنیم. تصویر ۸-۲ تصویر اولیه و نتیجه تابع را نشان میدهد.



تصویر۷-۲ – **خروجی تابع نوشته شده برای تست page2.jpg**