به نام خدا

دانشگاه تهران

دانشکده‌ی مهندسی برق و کامپیوتر

پردازش سیگنال‌های زمان‌گسسته

گزارش پروژه سوم

نام و نام خانوادگی	مریم ریاضی
شماره‌ی دانشجویی	810197518

بخش 1: پردازش سیگنال صوت

سوال1- فیلتر تک اکو

۱)

```
function y = echoDelet(x,fs)
    [c,~] = rceps(x);
    [~,locs] = findpeaks(c,'Threshold',0.2);
    dl = locs(1)-1;
    alpha = 0.8;
    y = filter(1,[1 zeros(1,dl-1) alpha],x);
    beta = dl/fs;
    disp(beta);
    audiowrite('noisy_voice.wav',y,fs);
end
```

شکل ۱: کد تابع حذف کننده اکو

---

2)

شکل 2: مقدار بتا به دست آمده از تابع

شکل 1، کد تابعی است که برای حذف اکو استفاده شده است. در این تابع ابتدا با استفاده از تابع rceps، قسمت حقیقی تبدیل کپستروم سیگنال به دست می آید. با توجه به صورت پروژه، برای یافتن مقدار بتا باید پیک های نمودار تبدیل کپستروم را پیدا کرد و به همین دلیل از findpeaks استفاده شده است. از آنجایی که این پیک در beta+1 است، ابتدا باید یک را از ایندکس این قله کم کرد. برای رسیدن به مقدار اصلی تاخیر اکو باید مقدار dl را بر فرکانس سیگنال تقسیم کرد. علت این تقسیم این است که برای تولید اکو نیز مقدار بتا که بر حسب ثانیه است را در فرکانس ضرب کرده و سپس سینال اصلی را به اندازه مقدار این حاصل ضرب

شیفت دادیم. علت این ضرب هم این است که برای مثال وقتی فرکانس سیگنال 48 کیلوهرتز باشد، برای تولید سیگنالی که0.5ثانیه شیفت خورده است، باید 24000 نمونه در سیگنال جلو برویم. برای حذف این اکو نیز باید سیگنال را از یک فیلتر IIR عبور دهیم.

این فیلتر به طریق زیر به دست می آید:

$$y[n] = x[n] + \alpha x[n - \beta * fs]$$

$$Y(z) = X(z)\left(1 + \alpha z^{-\beta * fs}\right)$$

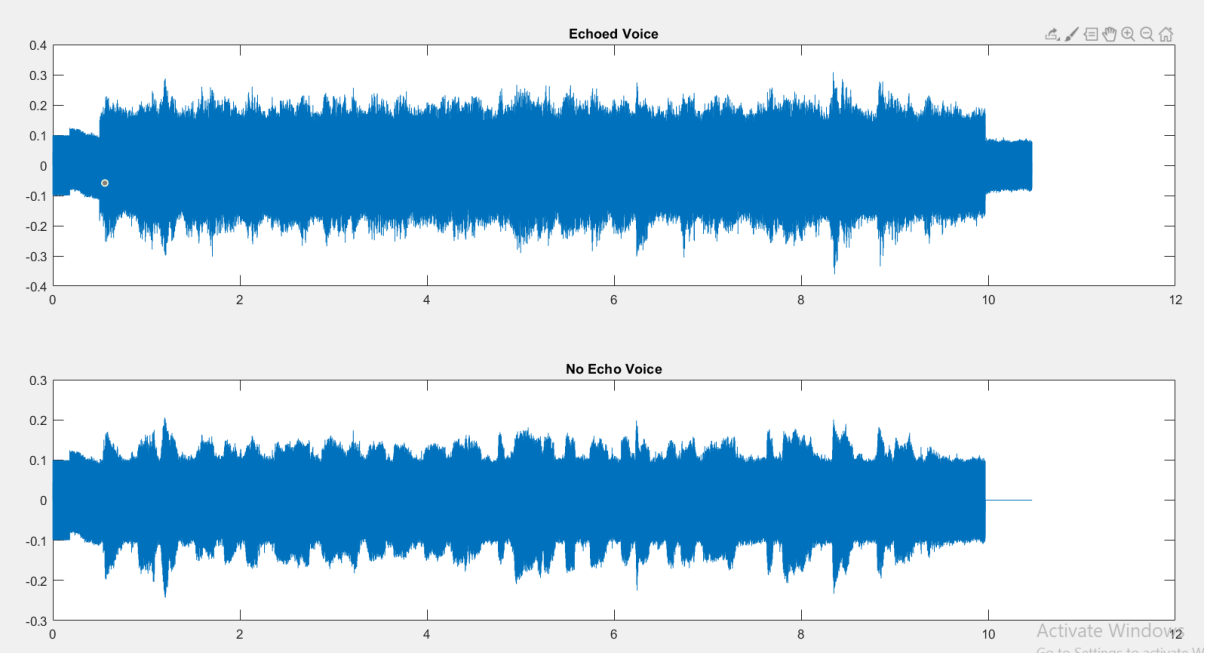
$$H(z) = 1 + \alpha z^{-\beta * fs}$$

برای از بین بردن اکو و رسیدن به سیگنال اصلی که در اینجا سیگنال نویزدار است، باید سیگنال اکو دار را از فیلتری با تابع تبدیل عکس H(z) عبور دهیم. در واقع تابع تبدیل فیلتر در حالت خاصی که $\alpha = 0.8$ و$\beta = 2$ است، به صورت زیر در می آید.

$$H(z) = \frac{1}{1 + 0.8z^{-24000}}$$

مقدار بتایی که برای تولید اکو استفاده کردم و مقدار بتایی که در تابع محاسبه شد، یکسان بود.

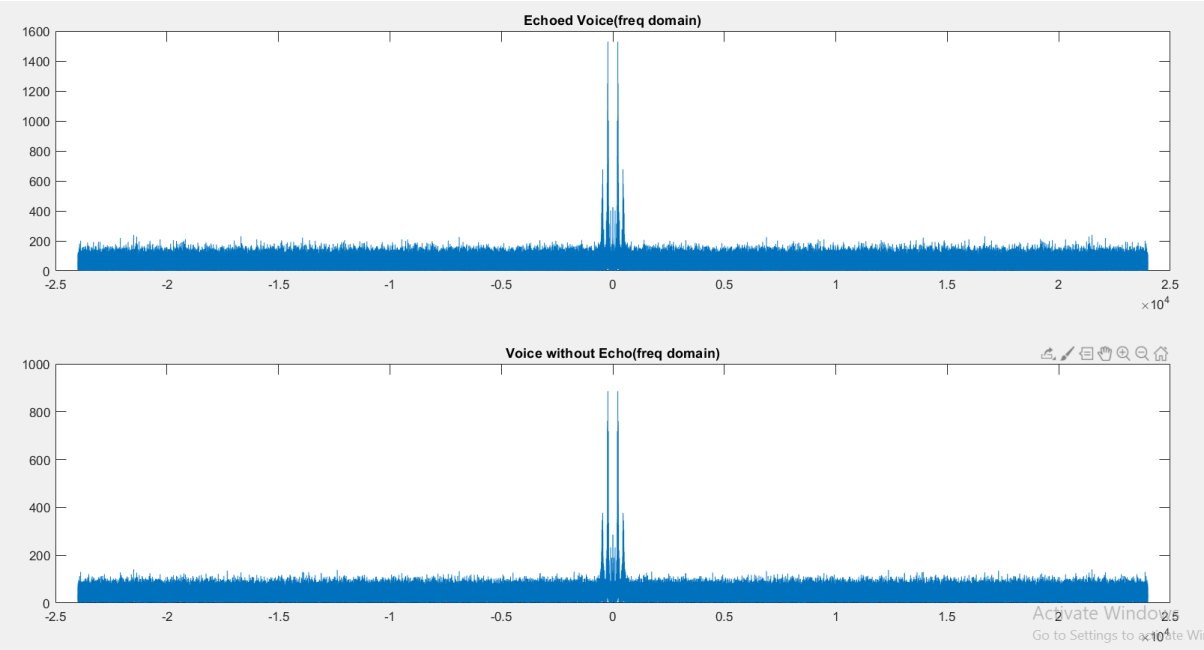
(3



شکل 3: نمودار صوت همراه با اکو و نویز و نمودار صوت بدون اکو

شکل 3 هر دو سیگنال صوت به همراه نویز و اکو و صوت بدون اکو را نشان می دهد. از آنجایی که من بتا را عدد 0.5 انتخاب کردم، در شکل می بینیم که اندازه نمودار بالایی از ثانیه نیم به بعد افزایش پیدا کرده است. این موضوع نیز به وضوح به دلیل اضافه شدن صوت شیفت خورده است که از ثانیه دوم به بعد به مقدار سیگنال اصلی اضافه شده است از طرفی دیگر مدت زمان نمودار بالایی 0.5 ثانیه بیشتر است.

نمودار پایینی که سیگنال صوت بدون اکو را نشان می دهد، اندازه ای یکنواخت دارد که در طول مدت 10 ثانیه تغییر نکرده است. در مدت زمان 10 تا 11.5 ثانیه نیز مقدار این سیگنال صفر است؛ زیرا اکوی اضافه شده، حذف شده است و آن سیگنال اضافه تر وجود ندارد.



شکل 4: صوت همراه با اکو و نویز و صوت بدون اکو در حوزه فرکانس

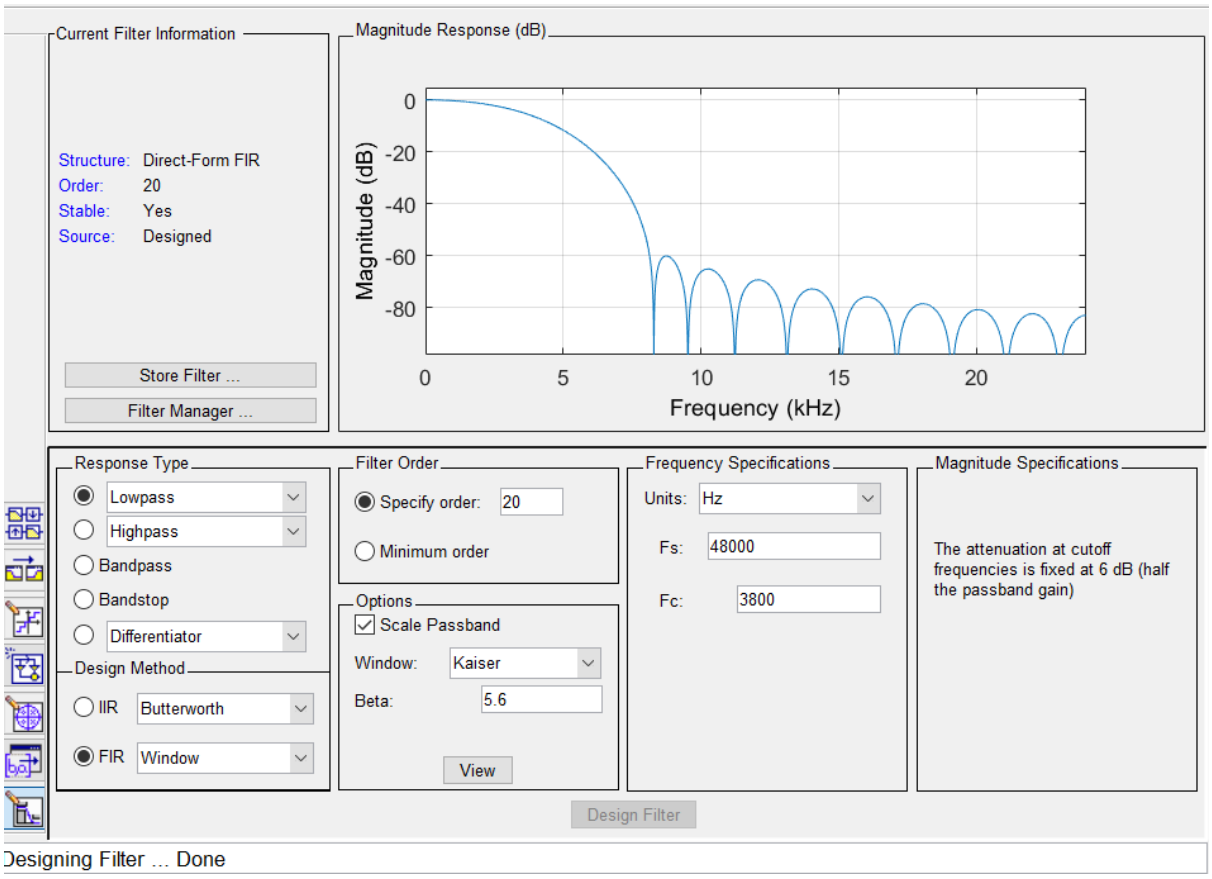
باتوجه به شکل 4 می بینیم که باند فرکانسی صوت با اکو و صوت بدون اکو تفاوتی ندارند و این موضوع از تابع تبدیل حوزه z نیز واضح است. اکو فقط بر روی اندازه تبدیل فوریه صوت تاثیر گذاشته است.

سوال ۲- حذف نویز به کمک فیلتر FIR

۱)

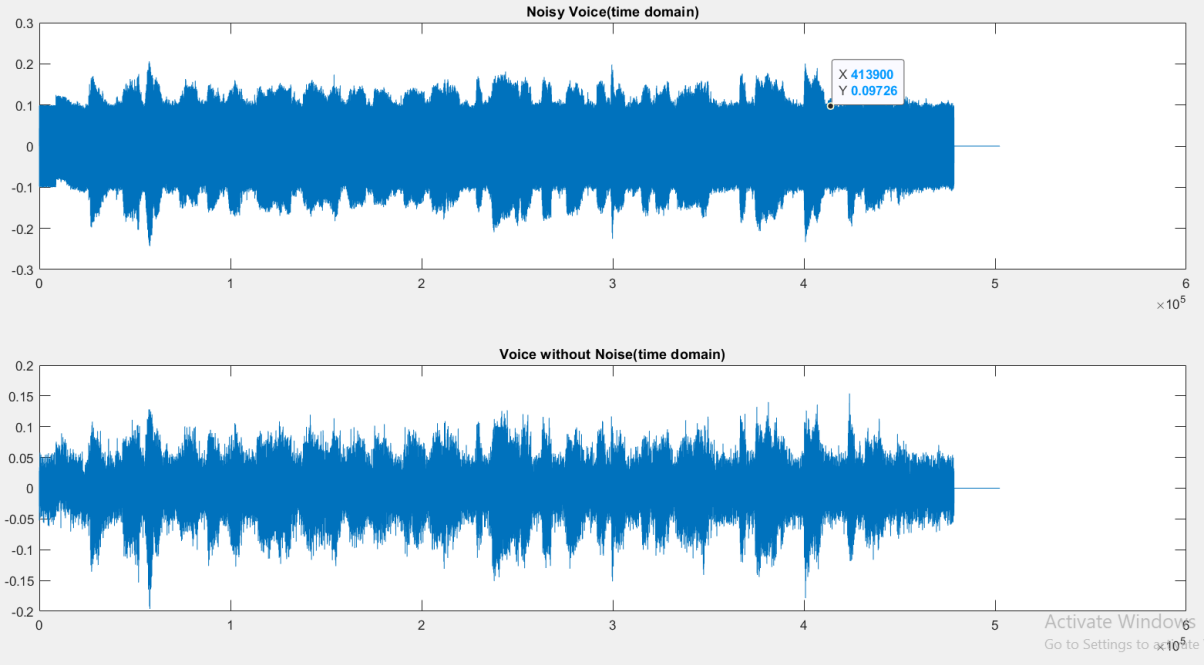
محدوده فرکانسی صوت انسان بین 300 تا 3400 هرتز است. فرکانس قطع فیلتر پایین گذر برای حذف نویز باید 3800 هرتز باشد. تضعیف باند توقف باید کمتر از 80dB و نوسان باند گذر باید کمتر 3dB باشد.

2)

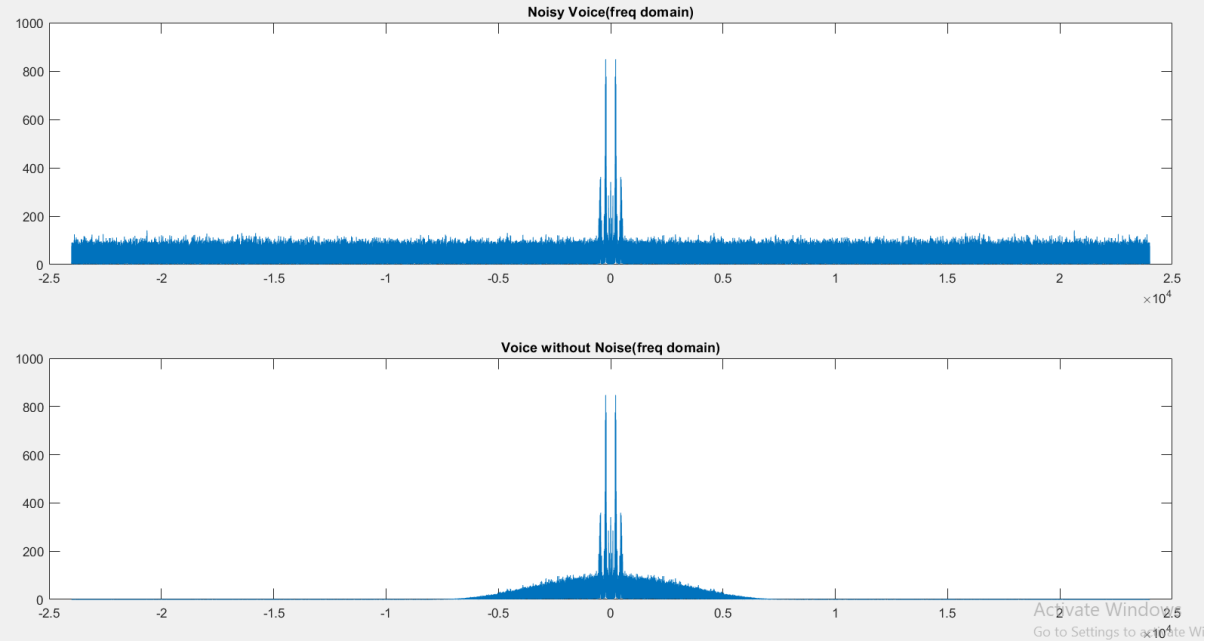


شکل 5: فیلتر طراحی شده

(3



شکل 6: سیگنال صوت با نویز و بدون نویز در حوزه زمان



شکل7: سیگنال صوت با نویز و بدون نویز در حوزه فرکانس

همانطور که در شکل 6 می بینیم، تفاوت سیگنال نویزی و سیگنال بازیابی شده در حوزه زمان در اندازه سیگنال است. در واقع چون نویز را به صورت یک سیگنال تصادفی به سیگنال اصلی اضافه کردیم، باعث می شود که اندازه سیگنال در لحظات مختلف، بیشتر شود.

در حوزه فرکانس نیز با توجه به شکل7 می بینیم که سیگنال نویزدار، باند فرکانسی بزرگتری دارد. عبور فیلتر پایین گذر از سیگنال نویزدار برای بازیابی سیگنال اصلی باعث شده است که این فرکانس های بیشتر و اضافی را حذف کند.

بخش دوم: پردازش تصویر

در ابتدا تصویر داده شده را بارگذاری کردم. در متلب این تصویر مانند یک تصویر رنگی، بعد سوم داشت و ازآنجایی که در ادامه قصد داشتم حاصل کانولوشن تصویر و kernel ها را به دست آورم، تصویر را با استفاده از دستور rgb2gray به صورت یک تصویر سیاه و سفید درآودم.

کاربرد هر kernel:

Sharpening:

کاربرد آن واضح کردن تصویر و برجسته کردن لبه های تصویر است. این kernel شبیه به kernel تشخیص لبه ها کار می کند ولی مقدار مرکزی آن متفاوت است. کنتراست لبه ها با برجسته کردن مناطق روشن و تاریک در لبه ها بیشتر می شود.

Blur:

این kernel برای تار کردن تصویر است. مجموع مقادیر این ماتریس برابر 1 است. از این موضوع می توان نتیجه گرفت که این کرنل هر پیکسل از تصویر را با میانگین پیکسل های اطراف جایگزین می کند. و به همین دلیل باعث افت کیفیت تصویر و محو شدن آن می شود.

Outline:

کار این کرنل همانطور که در شکل نیز مشاهده می شود، این است که خطوط لبه را در تصویر سفید کند و سایر عکس را سیاه کند. با توجه به مقادیر این ماتریس می توان دید که این کرنل در واقع به شکلی تفاوت پیکسل مورد نظر و پیکسل های اطراف را جایگزین پیکسل اصلی می کند.

Gauss:

این کرنل به بهبود کیفیت تصویر کمک می کند. در ماتریس این کرنل، مقدار درایه وسط بیشتر از سایر درایه ها است و به همین دلیل با تمرکز روی مقدار همان پیکسل از عکس و تقویت آن باعث افزایش کیفیت تصویر می شود.

Moving avg:

این کرنل نیز به نوعی دیگر باعث محو شدن تصویر می شود. مجموع مقادیر اعداد این ماتریس برابر یک می شود و این کرنل به گونه ای بین تمام پیکسل های مجاور میانگین می گیرد.

Line H:

این کرنل برای تشخص خطوط افقی استفاده می شود. این کرنل تمام عکس بجز خطوط افقی را سیاه کرده و تا جای ممکن باعث می شود که خطوط افقی سفید و روشن شوند. باتوجه به مقادیر این کرنل می بینیم که این کرنل مقدار پیکسل های افقی مجاور پیکسل مرکزی را در 2 ضرب کرده و سایر پیکسل ها را کم می کند. این کار باعث می شود که خط های افقی تشخیص داده شوند.

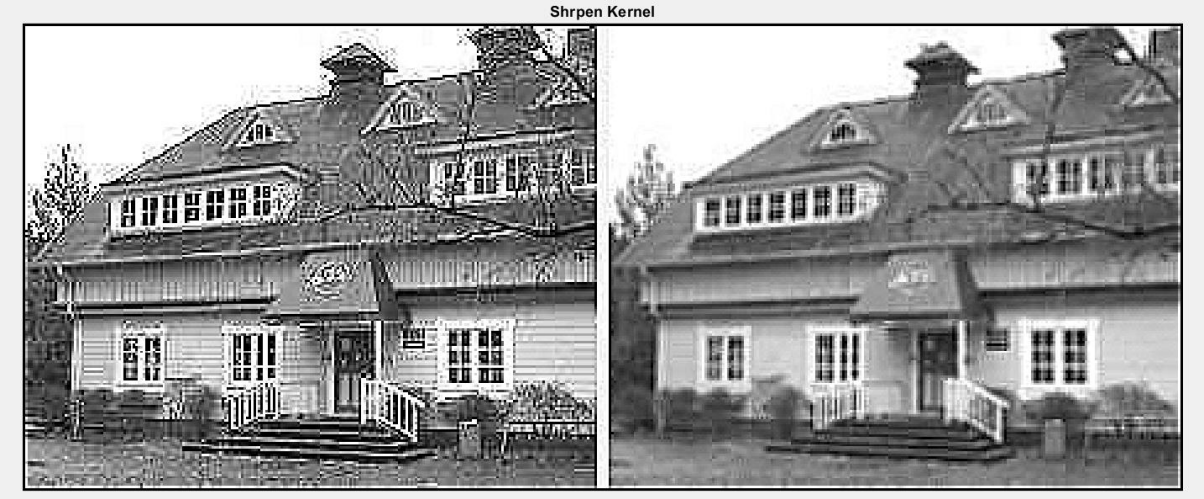
Line V:

این کرنل برعکس کرنل قبلی خطوط عمودی را تشخیص می دهد. این کرنل برعکس کرنل قبلی بر پیکسل های عمودی متمرکز است.

Identity:

همانطور که در مقادیر ماتریس این کرنل مشاهده می شود، فقط مقدار درایه وسط که در واقع مربوط به پیکسل اصلی است، برابر 1 است و باقی پیکسل های اطراف را صفر می کند. این کرنل در واقع همان تصویر اولیه را به عنوان خروجی می دهد.

A) Sharpen Kernel



شکل 8: نتیجه اعمال کرنل sharpening

B) Blur

شکل 9: نتیجه اعمال کرنل blur

C) Outline

شکل 10: نتیجه اعمال کرنل outline

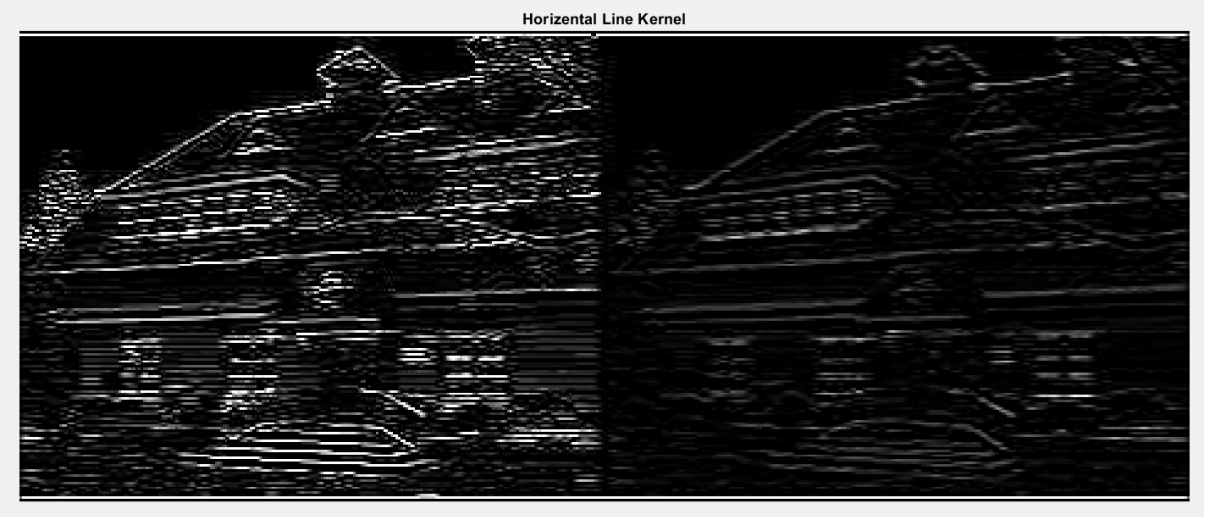
D) Gauss

شکل 11: نتیجه اعمال کرنل gauss

E) Moving Average

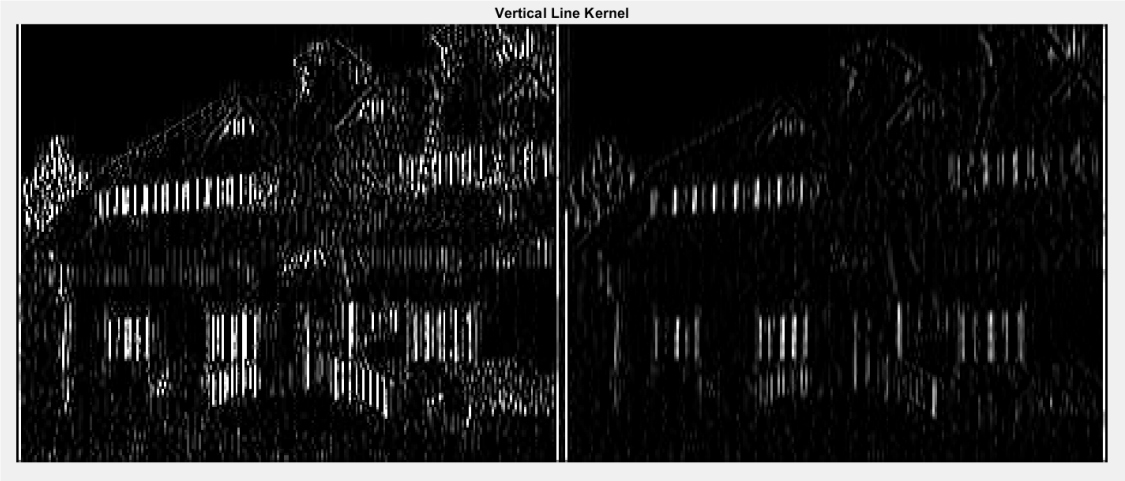
شکل 12: نتیجه اعمال کرنل moving average

F) Line H



شکل 13: نتیجه اعمال کرنل Line H

G) Line V

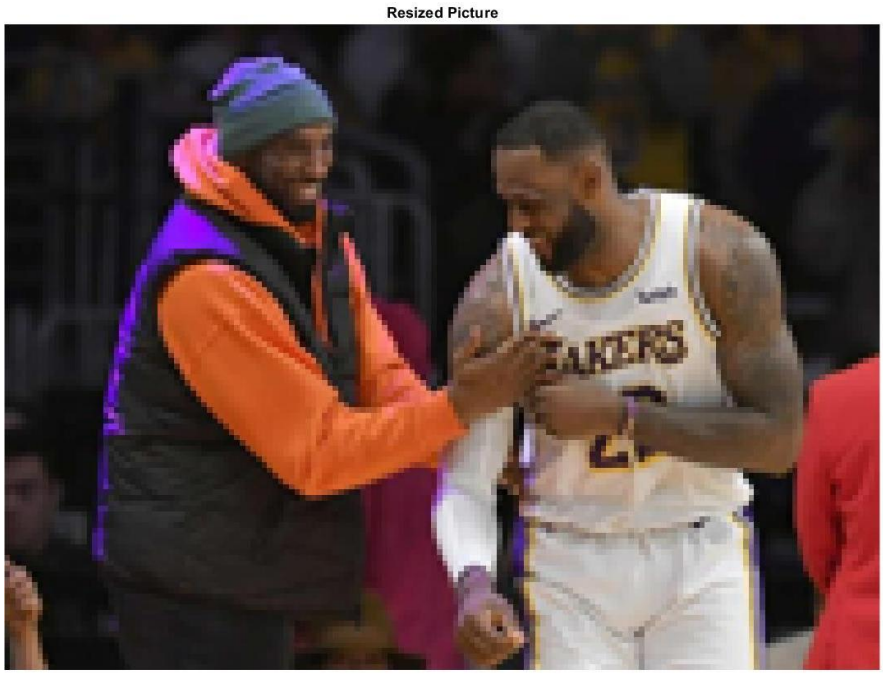


شکل 14: نتیجه اعمال کرنل Line V

H) Identity

شکل 15: نتیجه اعمال کرنل Identity

شکل 16: شکل کوچک شده تصویر اصلی



شکل 17: تصویر حاصل از دو بار تغییر سایز عکس اصلی.

همانطور که در شکل 17 مشخص است، تصویری که پس از دوبار Resizeکردن به دست امده نسبت به تصویر اصلی کیفیت کمتری دارد. علت این موضوع استفاده کردن از مد nearest است؛ زیرا این مد سرعت بیشتری در تغییر اندازه دارد اما کمترین کیفیت را در خروجی می دهد. در هر صورت با تغییر اندازه عکس در متلب تعدادی از پیکسل های عکس از بین می رود و وقتی دوباره ضریب معکوس را استفاده می کنیم به مقدار اصلی پیکسل های تصویر اصلی نمی رسیم و اطلاعات خیلی دقیق بازیابی نمی شوند.

بعد از اینکه اندازه تصویر را کم کردم و دوباره با استفاده مجدد از imresize اندازه آن را به مدار اصلی برگرداندم، مشاهده شد که تصویر حاصل کیفیت کمی دارد. برای بهبود کیفیت عکس ازkernel ها می توان استفاده کرد. Kernel هایی که برای بهبود کیفیت می توانند مورد استفاده قرار بگیرند، gauss & moving average هستند. هر کدام از این دو به روش خود کیفیت را بهتر می کنند به همین دلیل تصاویر حاصل دقیقا شبیه به یکدیگر نیستند و مهم تر از همه اینکه تصاویر حاصل به اندازه تصوصر اول باکیفیت نیستند. لازم به ذکر است که برای اعمال kernel ها مجبور شدم که عکس رنگی را به صورت سیاه و سفید دربیاورم تا بتوانم عمل کانولوشن را روی عکس ها و ماتریس های kernel ها انجام دهم.

شکل 18 :تصویر بهبود یافته با استفاده از Gaussian

شکل 19 :تصویر بهبود یافته با استفاده از moving average

با توجه به دو شکل 18 و 19 می بینیم که بازهم کیفیت به اندازه عکس اصلی نیست. با مقایسه این دو تصویر می توان به طور تقریبی گفت که moving average کیفیت بهتری را نسبت به گوسی ارائه داده.