



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
پردازش سیگنال‌های زمان گسسته

گزارش پروژه ۳

نام و نام خانوادگی	فاطمه صالحی
شماره دانشجویی	۸۱۰۱۹۸۴۲۳

بخش اول: پردازش سیگنال صوت :

سوال ۱- فیلتر تک اکو

کد مربوط به تابع Delay_Finder :

```

1 function [Delay,Delay_place,noisy_sound] = Delay_Finder(ynoisyy,x,Fs,Gain)
2 %-----Finding delay
3 L=length(ynoisyy);
4 dt=1/Fs;
5 t=(0:dt:(L/Fs-dt))';
6 c = rceps(ynoisyy);
7 L_half_c = floor(length(c)/2);
8 half_of_c = c(50:L_half_c,1);
9 Max = max(half_of_c);
10 Delay_place = find(c == Max);
11 Delay = t(Delay_place(1));
12 Delay=Delay(1);
13 %-----Remaking the noisy voice
14 zero1 = zeros(Delay_place(1)-1,1);
15 zero2 = zeros(L-Delay_place(1)+1-length(x));
16 noisyy_sound = noisyy - cat(1,zero1,Gain*x,zero2);
17 %-----Saving the noisy voice
18 audiowrite("noisyy_voice.m4a",noisyy_sound,Fs)
19 end
20

```

Figure 1.1 - Delay_Finder function

ورودی های این تابع، سیگنال نویزی اکو شده، سیگنال بدون نویز و اکو، فرکانس نمونه برداری، و ضریب سیگنال اکو شده(در این مسئله ۰.۸) میباشد.

ابتدا بر اساس طول سیگنال نویزی و فرکانس نمونه برداری بردار زمان را تشکیل داده و سپس با تابع `rceps` تبدیل کپستروم را انجام میدهیم .

بدون در نظر گرفتن ۵۰ داده اولیه خروجی تابع مذکور، پیک را در در نصفه اولیه خروجی بدست می آوریم و سپس با کمک تابع `find` جایگاه مقدار ماکسیمم را پیدا کرده(بتا) و سپس زمان متناظر با آن را نیز پیدا میکنیم. در ادامه با توجه بتا و `Gain` اکو را از سیگنال حذف میکنیم.

```

%% Making echoed noisy voice
[x,Fs] = audioread("Taylor.m4a");
h=[1 zeros(1,Fs/4) 0.8];
y=conv(x(:,1),h);
%sound(y,Fs)
wnoise = 0.009*randn(size(y));
ynoisyy = y + wnoise ;
%sound(ynoisyy,Fs)
audiowrite("noisyy_echoed_voice.m4a",ynoisyy,Fs)
%% Finding Beta using Delay_Finder function
[Delay_time,Beta,noisyy_sound] = Delay_Finder(ynoisyy,x,Fs,0.8);
%sound(noisyy_sound,Fs)

```

Name	Value
Beta	12002
Delay_time	0.2500
Fs	48000
h	1x12002 double
noisyy_sound	483591x1 double
wnoise	483591x1 double
x	471590x1 double
y	483591x1 double
ynoisyy	483591x1 double

Figure 1.2 - Delay_Finder function output

همانطور که در تصویر بالا دیده میشود، زمان تاخیر بدست آمده و ۰.۲۵ و بتا بدست آمده ۱۲۰۰۲ است که یعنی ربع ثانیه تاخیر داشته ایم، که با توجه به اینکه $Fs/4$ صفر بین توابع ضربه در `h` در نظر گرفته شده، این پاسخ صحیح میباشد.

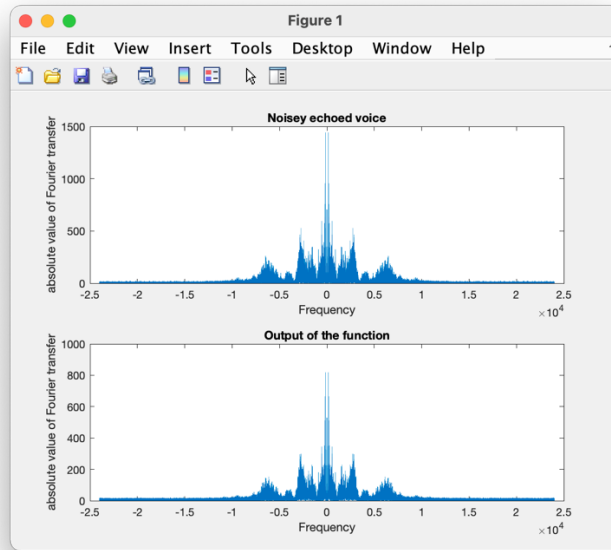


Figure 1.3 – FFT of Noisy echoed voice and FFT of the Output of the function comparison

$$y[n] = x[n] + ax[n - \beta] \xrightarrow{\mathcal{F}} X(w)(1 + \alpha e^{-jw\beta}) \stackrel{||}{\Rightarrow} |X(w)| \times \sqrt{1 + \alpha^2 + 2\alpha \cos(w\beta)}$$

با توجه به محاسبات انجام شده با اکو شدن صدا فرکانس جدیدی به تبدیل فوریه سیگنال اصلی اضافه نمیشود و به طریق مشابه با حذف اکو هم چنین اتفاقی نخواهد افتاد بلکه اندازه خواهد کرد، زیرا با کم کردن اکو، با توجه به اینکه $a=0.8$ است انکار توان یکسری فرکانس ها مشخص 0.64% کاهش پیدا خواهد کرد.

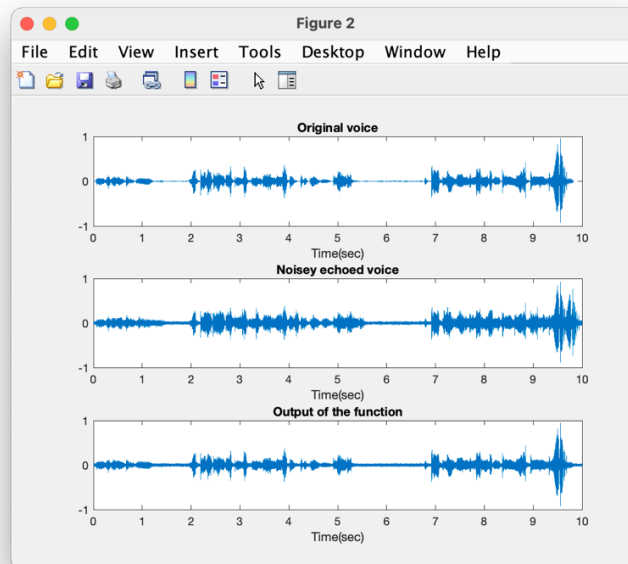


Figure 1.3 – Compared Original, Noisy echoed voice, and Output of the function in time domain

همانطور که در تصویر بالا دیده میشود، خروجی تابع و سیگنال در حوزه زمان شباهت زیادی دارد و تنها تفاوت بین این دو (که شباهت بین خروجی تابع و سیگنال نویزی اکو دار نیز میباشد)، در لحاظاتی که سیگنال اصلی صفر است، خروجی تابع مقادیر کمی را اخذ کرده است که همان نویز سفید گوسی میباشد.

سوال ۲ - حذف نویز به کمک فیلترهای FIR

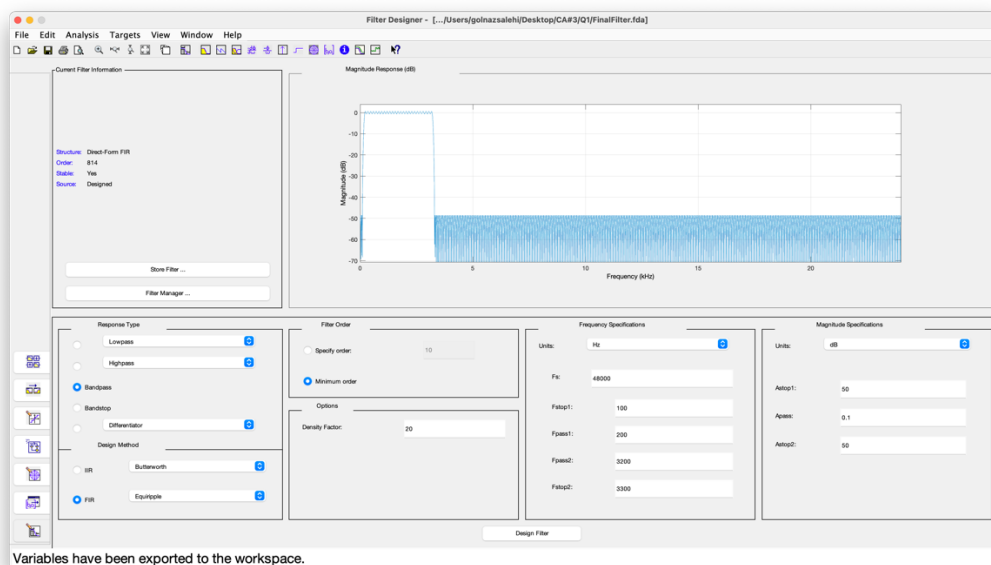


Figure 1.4 – Designed FIR filter

فرکانس های صوت انسان از ۲۰۰ هرتز تا ۳۲۰۰ هرتز را پوشش میدهند، بنابراین اگر مزر باند توقف را ۱۰۰ هرتز و ۳۳۰۰ در نظر بگیریم و همچنین با توجه به مطالعات انجام شده و آزمایش و خطا، اگر حدود نوسان باند گذرا ۰.۱ دسی بل و حدود نوسان باند توقف ۵۰ دسی بل باشد، نویز را حداقل در خارج از پهنای باند صوت حذف خواهد کرد.

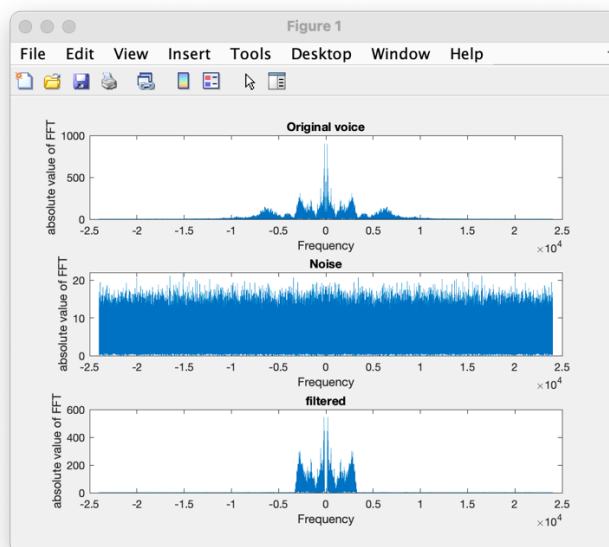


Figure 1.5 – original voice, gaussian noise, filtered voice comparison in frequency domain

با توجه به تصویر بالا، چون نویز سفید گوسی در تمام فرکانس ها مقدار غیر صفر دارد، بنابراین برای اینکه توان نویز در خروجی به طور چشم گیری کاهش یابد باید از یک سر فرکانس های سیگنال اصلی صرف نظر کرد؛ همانطور که دیده میشود این کار انجام شده و در خروجی با اینکه همچنان نویز وجود دارد ولی توان آن کاهش یافته است.

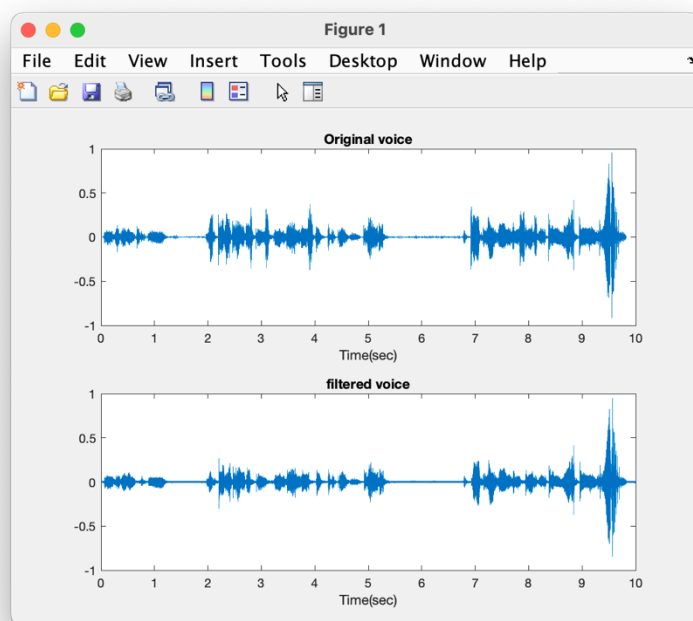


Figure 1.6 – original voice and filtered voice comparison in time domain

همانطور که مشاهده میکنیم، در حوزه زمان سیگنال ها تقریباً شبیه یکدیگر میباشند با این فرق که دامنه سیگنال فیلتر شده اندکی کمتر میباشد و اینکه در لحظاتی که در سیگنال اصلی سکوت بوده است، در سیگنال فیلتر شده نویز داریم که همانطور که اشاره شد حذف نویز به طور کامل امکان پذیر نمیباشد.

بخش ۲: پردازش تصویر :

سوال ۱- کرنل ها

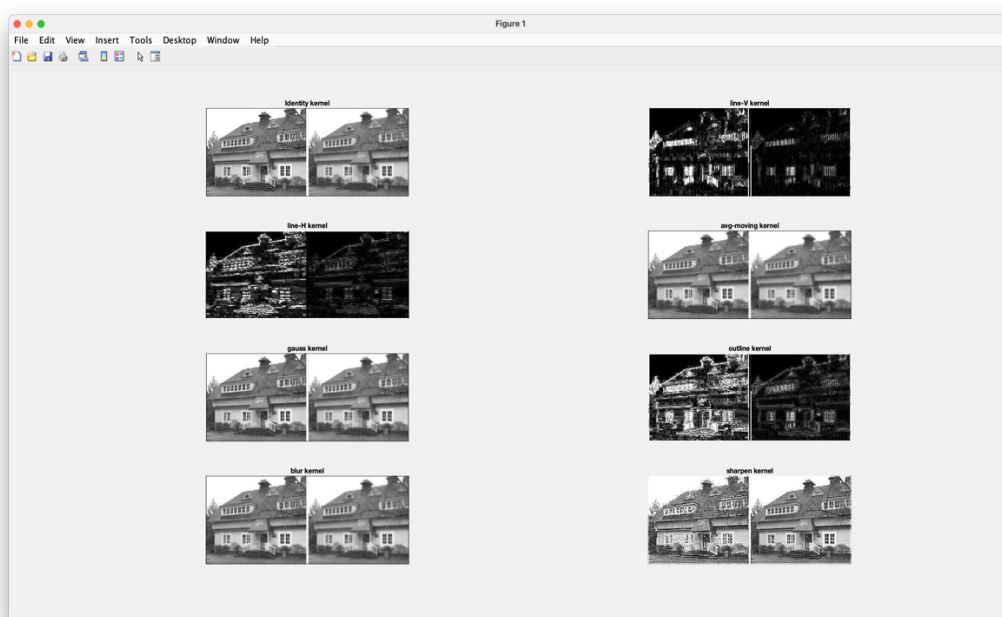


Figure 2.1 – The outputs of applied kernels to house.jpg

Identity kernel: $\begin{bmatrix} 0 & 0 & 0; \\ 0 & 1 & 0; \\ 0 & 0 & 0; \end{bmatrix};$

همانطور که از اسم این کرنل پیداست، این کرنل به ازای هر پیکسل خودش را قرار میدهد و تغییری در تصویر نخواهیم داشت.

Gauss kernel: $\begin{bmatrix} 0.0113 & 0.0838 & 0.0113; \\ 0.0838 & 0.6193 & 0.0838; \\ 0.0113 & 0.0838 & 0.0113; \end{bmatrix};$

این کرنل، از توضیح گوسی ۲ بعدی حاصل شده است و همانطور که دیده میشود به پیکسل مرکزی وزن بیشتری داده شده است بنابراین وقتی که به تصویری این کرنل را اعمال شود، یک میانگین وزن دار از پیکسل مرکزی و همسایه های آن در خروجی خواهیم داشت؛ لازم به ذکر است که در خروجی همچنان مرز اشکال، که از فرکانس بالا برخوردار میباشند، قابل دیدن است.

avg_moving kernel: $\begin{bmatrix} 0.111 & 0.111 & 0.111; \\ 0.111 & 0.111 & 0.111; \\ 0.111 & 0.111 & 0.111; \end{bmatrix};$

این کرنل، به تمام پیکسل ها به یک مقدار وزن میدهد و درواقع در خروجی، به ازای هر پیکسل، میانگینی از خودش و ۸ همسایه اش خواهیم داشت؛ لازم به ذکر است که تفاوت این کرنل و کرنل گوسی در مرز تغییرات میباشد، در این کرنل، شناسایی مرز تغییرات مشکل خواهد بود.

line_v kernel: $\begin{bmatrix} -1 & 2 & -1; \\ -1 & 2 & -1; \\ -1 & 2 & -1; \end{bmatrix};$

با اعمال این کرنل بر تصویر تغییرات عمودی در تصویر را **bold** میشوند. همانطور که مشاهده میشود، ستون وسط مقادیر مثبت و ستون های دیگر مقادیر منفی را اخذ کرده اند، اگر در محدوده ای باشیم که پیکسل ها با هم فرق چشم چیزی نداشته باشند، در خروجی مقداری که پیکسل مرکزی میگیرد صفر خواهد بود، ولی اگر این کرنل در مرز تغییرات قرار بگیرد مقدار پیکسل در خروجی دیگر صفر نخواهد بود بنابراین میتوان مرز را تشخیص داد.

```
line_H kernel: [-1 -1 -1;
                 2  2  2;
                 -1 -1 -1];
```

عملکرد این کرنل مانند کرنل **line_v** میباشد با این فرق که تغییرات افقی را **bold** میکند.

```
outline kernel: [-1 -1 -1;
                  -1 8 -1;
                  -1 -1 -1];
```

عملکرد این کرنل به بدین شکل است که تغییرات عمودی و افقی را نشان میدهد. اگر کرنل در محدوده ای قرار بگیرد که پیکسل ها فرق چندانی با هم نداشته باشند، خروجی صفر یا پیکسل سیاه خواهد بود و اگر لب مرز باشد و درواقع پیکسل مرکزی از همسایه هایش متفاوت باشد آنگاه خروجی نزدیک ۲۵۶ یا پیکسل سفید خواهد بود.

```
blur kernel: [0.0625 0.125 0.0625;
                0.125 0.25 0.125;
                0.0625 0.125 0.0625];
```

این کرنل، با اختصاص دادن وزن بیشتر به پیکسل مرکزی و ۴ پیکسلی که با آن ضلع مشترک دارند باعث مات شدن تصویر خواهد شد.

```
sharpen kernel: [0 -1 0;
                  -1 5 -1;
                  0 -1 0];
```

این کرنل باعث **sharp** شدن تغییرات میشود، زیرا اگر این کرنل در محدوده ای قرار بگیرد که پیکسل ها شبیه هم باشند خروجی سیاه خواهد بود و اگر پیکسل مرکزی با پیکسل های اطرافش متفاوت باشد، آنگاه خروجی سفید خواهد بود.

سوال ۲ - imresize:

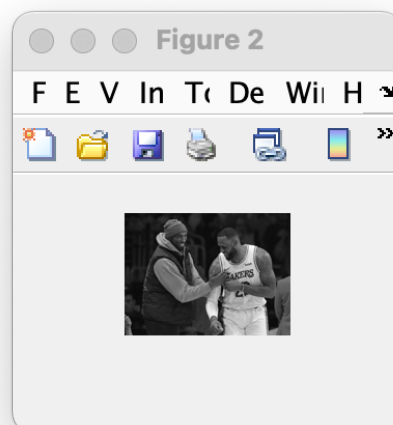


Figure 2.2 - kobe.jpeg scaled by factor 0.2 (Kobe_resize1)

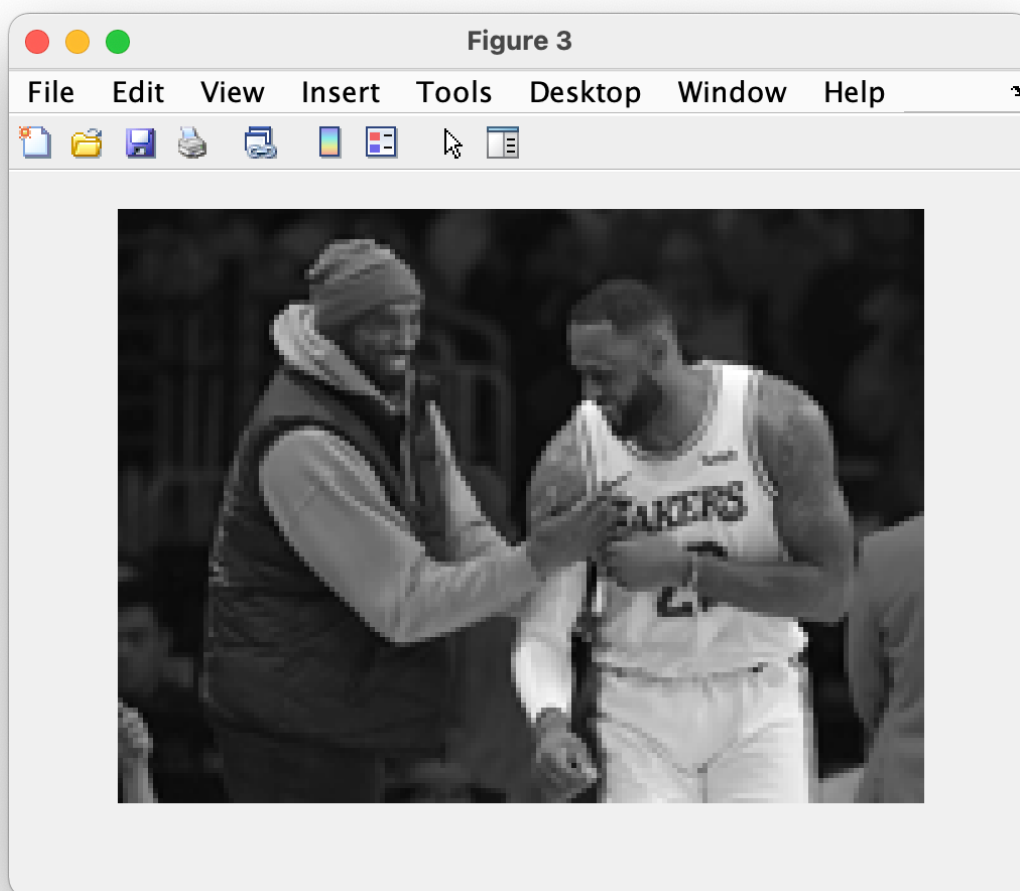


Figure 2.2 – Kobe_resize1 scaled by factor 5 (Kobe_resize2)

وقتی سایز تصویر اصلی را با نسبت $1/5$ تغییر میدهیم، درواقع تصویر را دون سمپل میکنیم (تعداد پیکسل ها در طول $1/5$ و در عرض نیز $1/5$ خواهد شد)؛ و وقتی دوباره آن را به سایز اولیه برمیگردانیم، چون اطلاعات را سابق را نداریم، انتظار میرود که کیفیت تصویر اصلی را نداشته باشیم و درواقع کاری که این تابع انجام میدهد این است که مساحت هر پیکسل را 25 برابر میکند و سپس این پیکسل را به پیکسل های استاندارد تقسیم میکند و ذخیره خواهد کرد.



Figure 2.3 – applied gauss kernel and avg-moving kernel to the resized picture

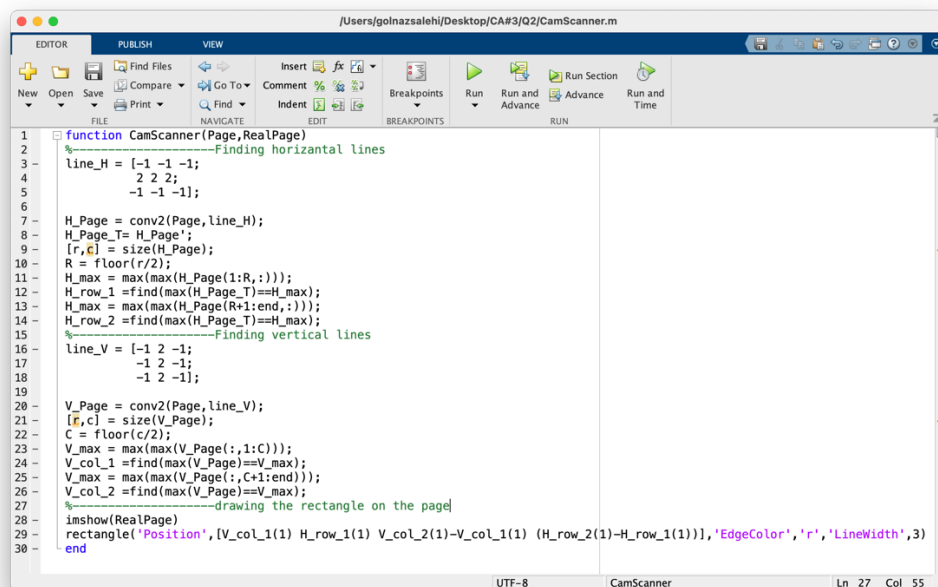
برای اینکه فرق بین عکس مشخص شود، بر قسمت مشخصی از تصویر زوم میکنیم.



Figure 2.4 – zoomed-in version of Figure 2.3

همانطور که دیده میشود، حرف S، به خصوص با فیلتر avg-moving، واضح تر شده است.

بخش ۳- امتیازی:

A screenshot of a MATLAB editor window showing the code for a function named 'CamScanner'. The code is written in a script format and includes comments in Persian. It defines a function that takes 'Page' and 'RealPage' as inputs. The code uses 'conv2' to convolve the page with horizontal and vertical line templates. It then finds the maximum values in the resulting images to determine the bounding box of the page. Finally, it uses 'imshow' to display the original page with a red rectangle indicating the detected page area. The code is as follows:

```
1 function CamScanner(Page,RealPage)
2 %-----Finding horizontal lines
3 line_H = [-1 -1 -1;
4           2 2 2;
5           -1 -1 -1];
6
7 H_Page = conv2(Page,line_H);
8 H_Page_T= H_Page';
9 [r,c] = size(H_Page);
10 R = floor(r/2);
11 H_max = max(max(H_Page(1:R,:)));
12 H_row_1 =find(max(H_Page_T==H_max));
13 H_max = max(max(H_Page(R+1:end,:)));
14 H_row_2 =find(max(H_Page_T==H_max));
15 %-----Finding vertical lines
16 line_V = [-1 2 -1;
17           -1 2 -1;
18           -1 2 -1];
19
20 V_Page = conv2(Page,line_V);
21 [r,c] = size(V_Page);
22 C = floor(c/2);
23 V_max = max(max(V_Page(:,1:C)));
24 V_col_1 =find(max(V_Page==V_max));
25 V_max = max(max(V_Page(:,C+1:end)));
26 V_col_2 =find(max(V_Page==V_max));
27 %-----drawing the rectangle on the page
28 imshow(RealPage)
29 rectangle('Position',[V_col_1(1) H_row_1(1) V_col_2(1)-V_col_1(1) (H_row_2(1)-H_row_1(1))],'EdgeColor','r','LineWidth',3)
30 end
```

Figure 3.1 – CamScanner function

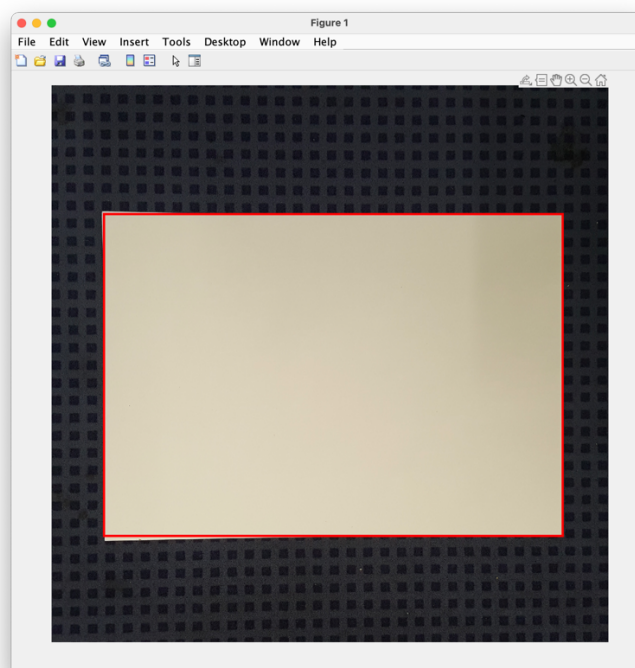


Figure 3.2 – Detected page