به نام خدا

1. قانون افزودن فازوری

1.1. مشخص سازی مقادیر فاز و دامنه و فرکانس

با توجه به شماره دانشجویی درج شده در بالای صفحه و روش گفته شده در صورت سوال، مقادیر فاز و دامنه فرکانس را تعیین می کنیم:

$$\omega_0 = 17$$
$$A_1 = 7 \quad A_2 = 1 \quad A_3 = 6$$
$$\varphi_1 = 17° \quad \varphi_2 = 176° \quad \varphi_3 = 716°$$

با توجه به اینکه می خواهیم زوایا در بازه اصلی باشند، زاویه φ_3 برابر با 356 درجه خواهد شد.

حال فازور های متناظر با این زوایا و دامنه ها، به صورت زیر خواهند شد:

$$Phasor1 = 7\angle 17° \quad Phasor2 = 1\angle 176° \quad Phasor3 = 6\angle 356°$$

با توجه به اینکه فازور یک عدد مختلط است، بخش های حقیقی و موهومی آن به صورت زیر محاسبه می شوند:

$$A\angle\varphi = A\cos\varphi + jA\sin\varphi$$

پس فازور های ما به شکل زیر در می آیند:

$$Phasor1 = 6.694 + j2.047$$
$$Phasor2 = -0.998 + j0.070$$
$$Phasor3 = 5.985 - j0.419$$

بنابراین جمع آن ها به صورت زیر خواهد بود:

$$Final\ Phasor = 11.681 + j1.698$$

حال می دانیم شکل فازوری یک عدد مختلط به فرم $a + jb$ به صورت زیر است:

$$\sqrt{a^2 + b^2}\angle \tan^{-1}\frac{b}{a}$$

پس با تبدیل عدد مختلط نهایی به دست آمده به شکل فازوری آن داریم:

$$Final\ Phasor = 11.804\angle 8.271°$$

با تبدیل زاویه به دست آمده به رادیان، سیگنال نهایی ما برابر خواهد بود با:

$$\mathbf{11.80\cos(17t + 0.14)}$$

2.1. پیاده سازی در متلب

کدی می نویسیم که مقدار فرکانس و مقادیر فاز و دامنه سه فازور را از کاربر دریافت کند و سپس با جمع آن ها، سیگنال کسینوسی نهایی را نشان دهد.

با وارد کردن مقادیر تعیین شده در بخش 1 این سوال، ضابطه سیگنال نهایی را دریافت می کنیم:

$$x(t) = 11.80\cos(17t+0.14)$$

شکل1- ضابطه سیگنال نهایی

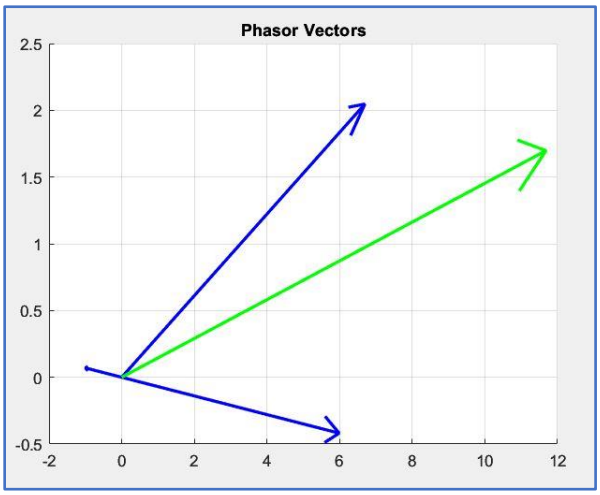
همانطور که دیده می شود، ضابطه به دست آمده کاملا با محاسبات تئوری ما تطابق دارد و کد ما درست کار می کند.

همچنین در کد، یک متغیر تعریف کردیم که مقدار فاز نهایی را برحسب درجه محاسبه و درون خود ذخیره می کند. با مراجعه به workspace و مشاهده مقدار این متغیر، می بینیم که فاز به دست آمده بر حسب درجه نیز با فازی که در محاسبات تئوری بر حسب درجه به دست آوردیم، تقریبا برابر است:

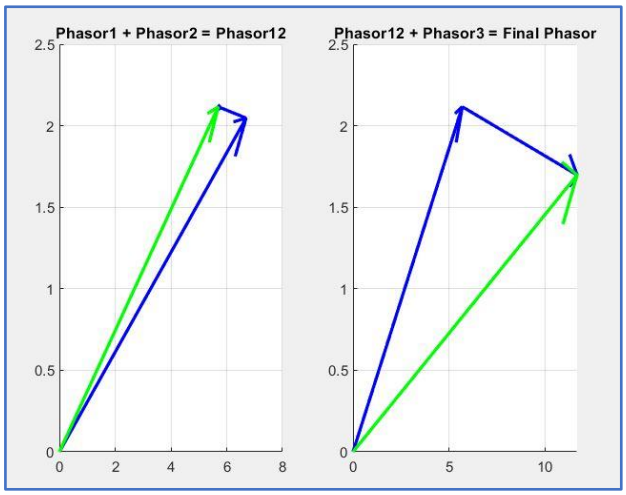
final_phi_degree	8.2693

شکل2- فاز نهایی به دست آمده بر حسب درجه

همچنین با استفاده از دستور quiver، کدی نوشته ایم که یک بار فازور ها را به صورت برداری در صفحه مختلط رسم کند و بار دیگر با رسم این بردار ها پشت سر یکدیگر، قاعده جمع فازوری به روش برداری را نشان دهد.



شکل3- بردار های متناظر با فازور ها



شکل4- جمع فازور ها به روش برداری در دو مرحله

دقت شود که رنگ آبی، فازور های ورودی و رنگ سبز فازور حاصل جمع را نشان می دهد.

---

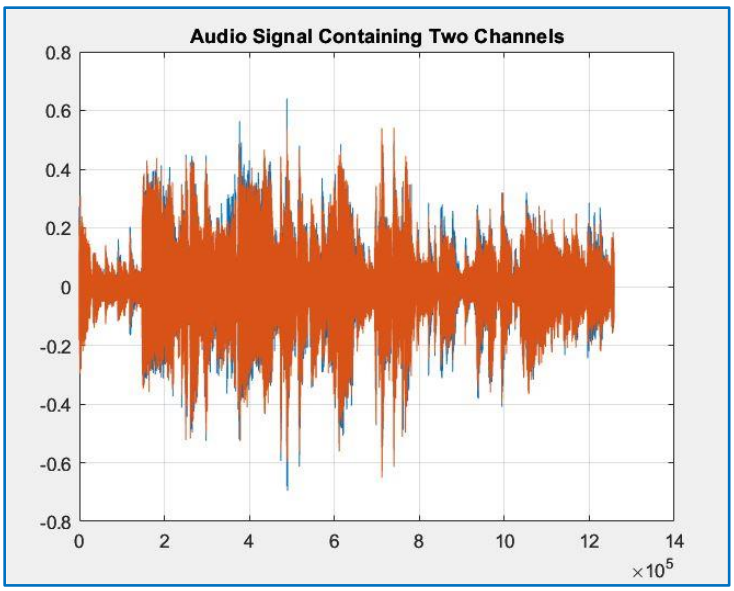
2. مقدمه ای بر پردازش صوت

1.2. بارگذاری فایل صوتی در متلب و اجرا کردن آن

*** نکته : در این سوال قبل از اجرای کد بخش های مختلف، حتما کد بخش 1 اجرا شود.**

ابتدا با استفاده از دستور audioread فایل صوتی را بارگذاری کرده و آن را با فرکانس نمونه برداری 48 kHz اجرا می کنیم.
همچنین با استفاده از دستور plot سیگنال صوتی را رسم می کنیم.

همانطور که در شکل صفحه بعد دیده می شود، این سیگنال، دو کانال مختلف دارد.



شکل5- سیگنال صوتی که شامل دو کانال است

در ادامه از این دو کانال میانگین گرفته و از سیگنال بدست آمده در باقی بخش های سوال استفاده می کنیم.

با اجرای کانال های مختلف سیگنال، تفاوت جزئی‌ای که در می یابیم این است که کانال اول، کمی عمق بیشتر دارد و با کیفیت تر است؛ اما به طور کلی تفاوت این کیفیت بسیار ناچیز و آن چنان فرق خاصی حس نمی شود.

2.2. افزودن اکو به سیگنال صوتی

معادله تفاضلی داده شده در صورت سوال برای توصیف سیستم ایجاد کننده اکو به صورت زیر است:

$$y[n] = x[n] + \alpha x[n - n_d]$$

با گرفتن تبدیل z از این معادله داریم:

$$Y(z) = X(z) + \alpha z^{-n_d} X(z)$$

با فاکتور گرفتن از $X(z)$ و تقسیم معادله بر آن، تابع سیستم به دست می آید:

$$H(z) = 1 + \alpha z^{-n_d}$$

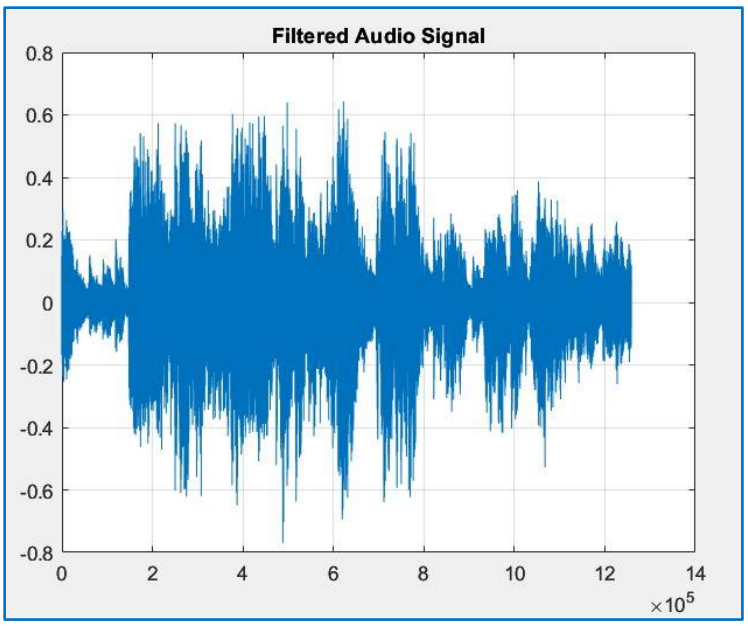
حال با استفاده از دستور filter این تابع سیستم را توصیف کرده و سیگنالی که در اختیارمان است را فیلتر می کنیم. این دستور به ترتیب ،ضرایب صورت و مخرج در تابع سیستم را به صورت یک آرایه دریافت می کند. عضو اول این آرایه، ضریب z^0 و عضو آخر این آرایه ضریب z^{-n_d} است. در نهایت سیگنال مد نظر را دریافت می کند و آن را فیلتر می کند.

با گوش دادن به سیگنال صوتی فیلتر شده، اکوی ایجاد شده با استفاده از تاخیر سیگنال زمانی را در آهنگ حس می کنیم.

سیگنال فیلتر شده را plot کرده ایم و تصویر آن را در صفحه بعد در شکل6 آورده ایم. با مقایسه این سیگنال با سیگنال شکل5، تفاوت حاصل از تاخیر ایجاد شده در سیگنال فیلتر شده، کاملا قابل رویت است.

3.2. بازیابی سیگنال اصلی از روی سیگنال فیلتر شده

در این بخش با استفاده از مفهوم سیستم وارون، اکوی ایجاد شده را از روی سیگنال فیلتر شده برداشته و آن را رفع فیلتر می‌کنیم.



شکل6- سیگنال صوتی فیلتر شده

سیستم وارون، سیستمی است که اگر به صورت کسکود با سیستم اصلی بسته شود، سیگنال خروجی برابر با سیگنال ورودی خواهد شد. به بیان ریاضی، اگر سیستم اصلی را h_1 و سیستم وارون را h_2 بنامیم در این صورت با دادن سیگنال ورودی $x(t)$ به سیستم اصلی، خواهیم داشت:

$$\text{خروجی سیستم اصلی}: y_1(t) = x(t) * h_1(t)$$

$$\text{خروجی نهایی}: y_2(t) = y_1(t) * h_2(t)$$

در نتیجه:

$$y_2(t) = x(t) * h_1(t) * h_2(t)$$

با تبدیل لاپلاس گرفتن از این معادله داریم:

$$Y_2(s) = X(s).H_1(s).H_2(s)$$

حال با توجه به مفهوم سیستم وارون می دانیم که دو سیگنال $x(t)$ و $y_2(t)$ برابر هستند، پس تبدیل لاپلاس آن ها نیز برابر است؛ در نتیجه داریم:

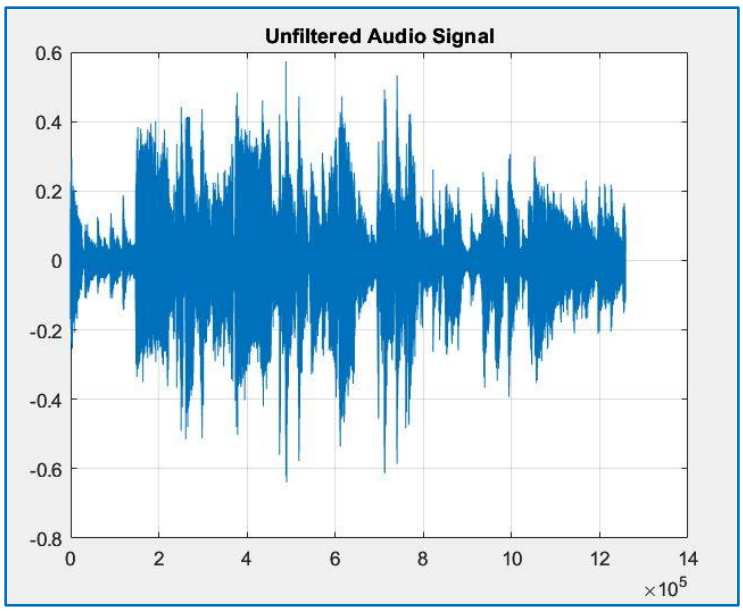
$$1 = H_1(s).H_2(s)$$

$$\boldsymbol{H_2(s) = \frac{1}{H_1(s)}}$$

پس تابع تبدیل سیستم وارون، معکوس تابع تبدیل سیستم اصلی است.

بنابراین برای رفع فیلتر سیگنال به دست آمده در بخش قبل، کافی است آرایه های ضرایب صورت و مخرج تابع تبدیل سیستم اصلی را به صورت جا به جا، به دستور filter بدهیم و آن را روی سیگنال اعمال کنیم.

با گوش کردن به سیگنال صوتی به دست آمده، می بینیم که کاملا رفع فیلتر شده و هیچ اکویی روی آن وجود ندارد.



شکل7- سیگنال صوتی رفع فیلتر شده

4.2. فیلتر کردن و رفع فیلتر با استفاده از یک سیستم پیچیده تر

معادله تفاضلی داده شده برای این سیستم به صورت زیر است:

$$y[n] = x[n] + \alpha x[n - n_d] + \alpha^2 x[n - 2n_d] + \alpha^3 x[n - 3n_d]$$

از این معادله، تبدیل z می گیریم:

$$Y(z) = X(z) + \alpha z^{-n_d} X(z) + \alpha^2 z^{-2n_d} X(z) + \alpha^3 z^{-3n_d} X(z)$$

با فاکتور گرفتن از $X(z)$ و تقسیم معادله بر آن، تابع سیستم به دست می آید:

$$H(z) = 1 + \alpha z^{-n_d} + \alpha^2 z^{-2n_d} + \alpha^3 z^{-3n_d}$$

حال با استفاده از روش توضیح داده شده در بخش 2، این تابع را با استفاده از دستور filter روی سیگنالمان اعمال می کنیم. سپس یک بار دیگر تابع سیستم وارون را روی آن اعمال می کنیم تا سیگنال اصلی بازیابی شود. نمودار زمانی سیگنال اصلی، سیگنال فیلتر شده و سیگنال بازیابی شده را رسم کرده ایم و تصویر آن را در شکل8 در صفحه بعد آورده ایم.

با گوش دادن به سیگنال فیلتر شده، یک نوع اکو را حس می کنیم که از اکوی ایجاد شده در بخش 2 پیشرفته تر است.

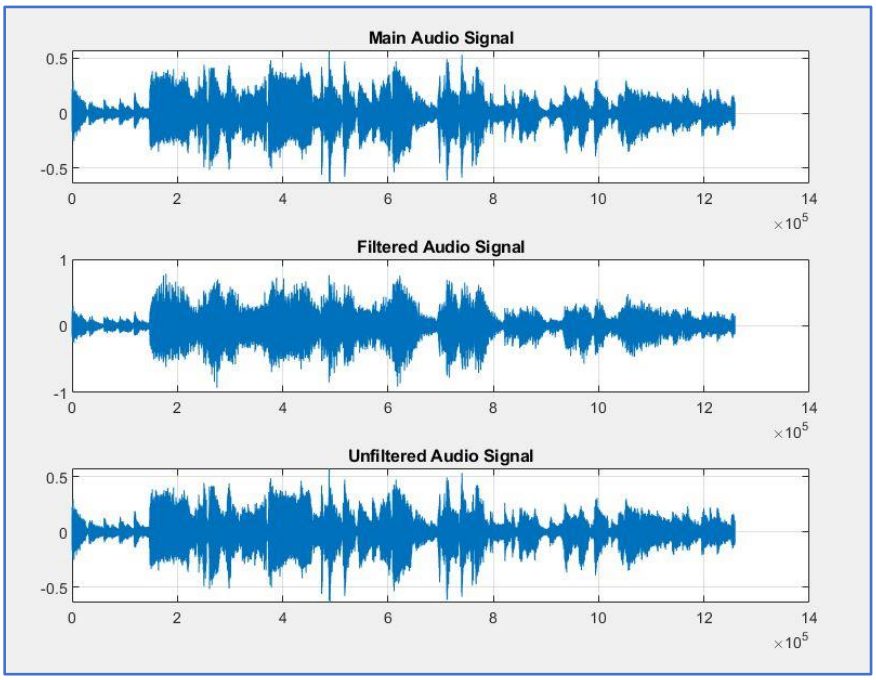
همچنین با گوش دادن به سیگنال بازیابی شده، مشاهده می کنیم که مانند سیگنال اصلی مان است.

5.2. اضافه کردن نویز به سیگنال

ابتدا با دستورات rand و randn نمونه های تصادفی با توزیع یونیفرم و نرمال و هم اندازه با سیگنال صوتی مان درست می کنیم و آن ها را جداگانه به سیگنال اضافه می کنیم.

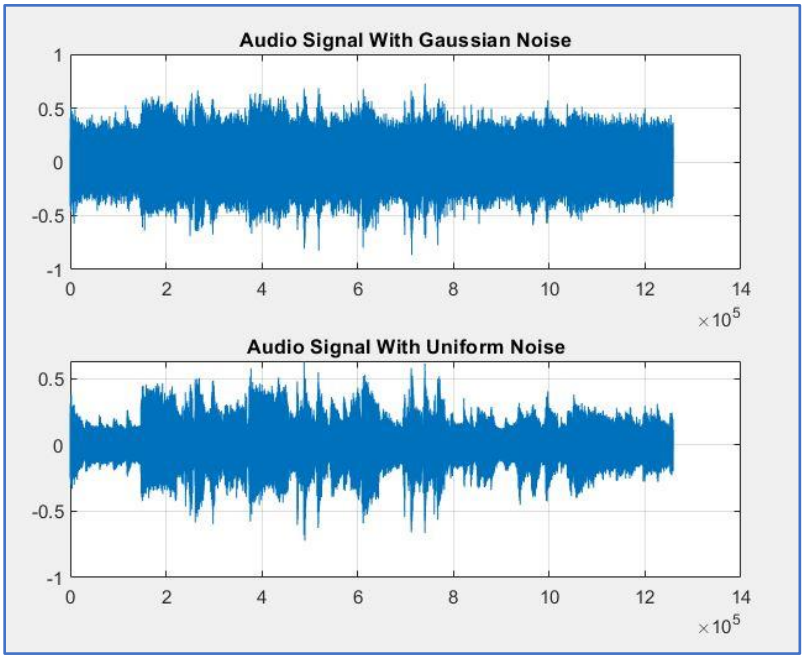
با گوش دادن به دو سیگنال صوتی جدید، می توان دریافت که شدت نویز در سیگنال نویزی شده با استفاده از توزیع نرمال بیش تر است. چرا که این سیگنال واریانس 0.01 دارد، در نتیجه انحراف معیار آن 5 است.

این یعنی بازه اطمینان 95 درصدی تغییرات داده های آن از 0.2- تا 0.2 است؛ اما بازه تغییرات داده های توزیع یونیفرمی که تعریف کردیم از 0.1- تا 0.1 است. پس منطقی است که با گوش دادن به سیگنال نویزی شده با توزیع نرمال، نویز بیشتری احساس کنیم.



شکل8- نمودار زمانی سیگنال اصلی، سیگنال فیلتر شده و سیگنال بازیابی شده در بخش 4 سوال 2

سیگنال های نویزی شده را رسم کرده ایم و در شکل زیر نمایش داده ایم. همانطور که دیده می شود، شدت نویز در سیگنال نویزی شده توسط توزیع نرمال بیش تر است.

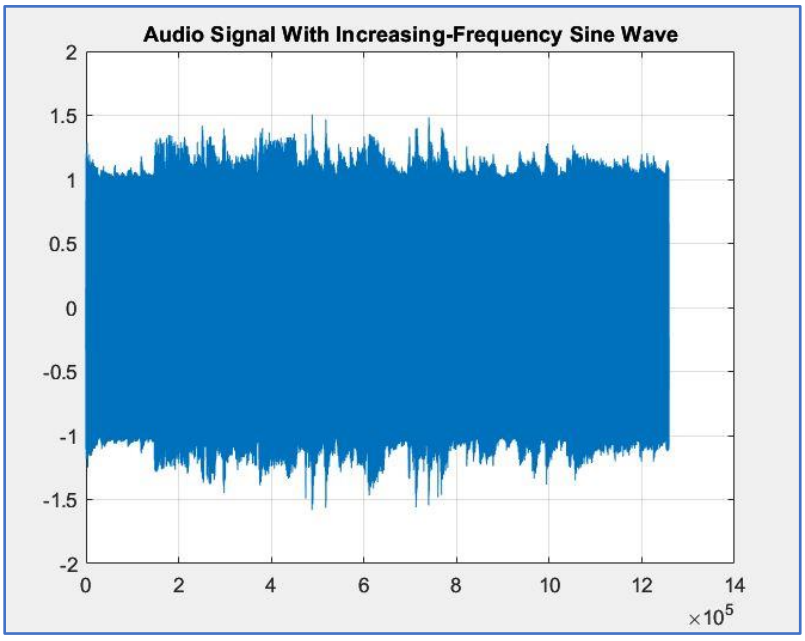


شکل9- سیگنال های نویزی شده با استفاده از توزیع های نرمال و یونیفرم

6.2. اضافه کردن سیگنال سینوسی با فرکانس متغیر به سیگنال اصلی

در این بخش با توجه به فرکانس نمونه برداری، یک موج سینوسی تعریف می کنیم که فرکانس آن به طور خطی در زمان از 1000 کیلوهرتز تا 2000 کیلوهرتز افزایش می یابد.

حال این موج سینوسی را به سیگنال صوتی مان اضافه می کنیم و سیگنال صوتی حاصل را اجرا و ذخیره می کنیم.



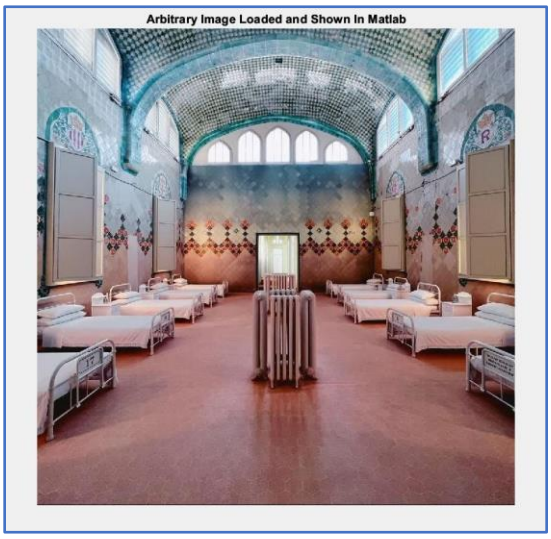
شکل10- سیگنال حاصل پس از جمع با سیگنال سینوسی

---

3. مقدمه ای بر پردازش تصویر

1.3. بارگذاری تصویر دلخواه در متلب و نمایش آن

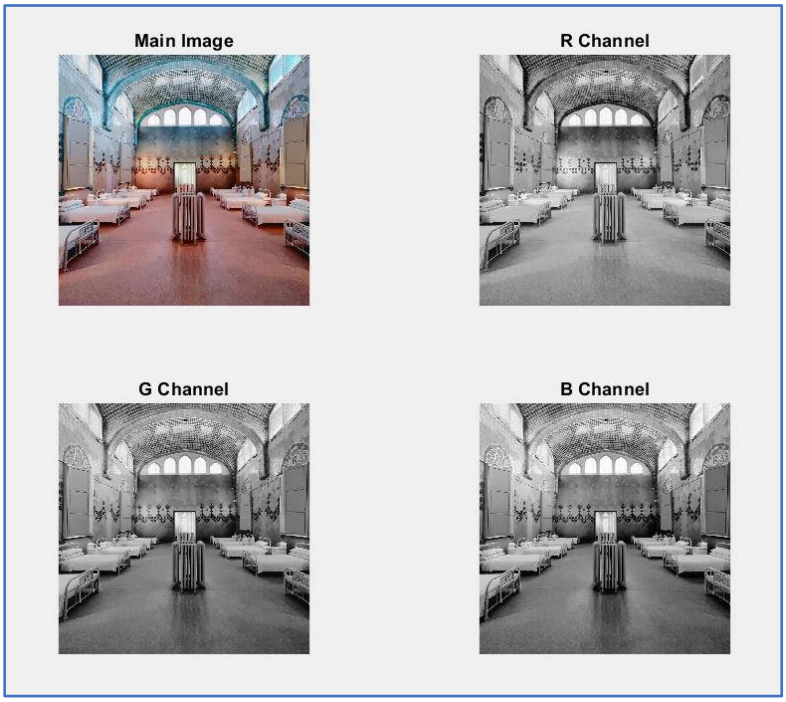
در این بخش با استفاده از دستورات imread و imshow، یک تصویر دلخواه را در متلب بارگذاری کرده و نمایش می دهیم.



شکل11- تصویر نمایش داده شده در متلب

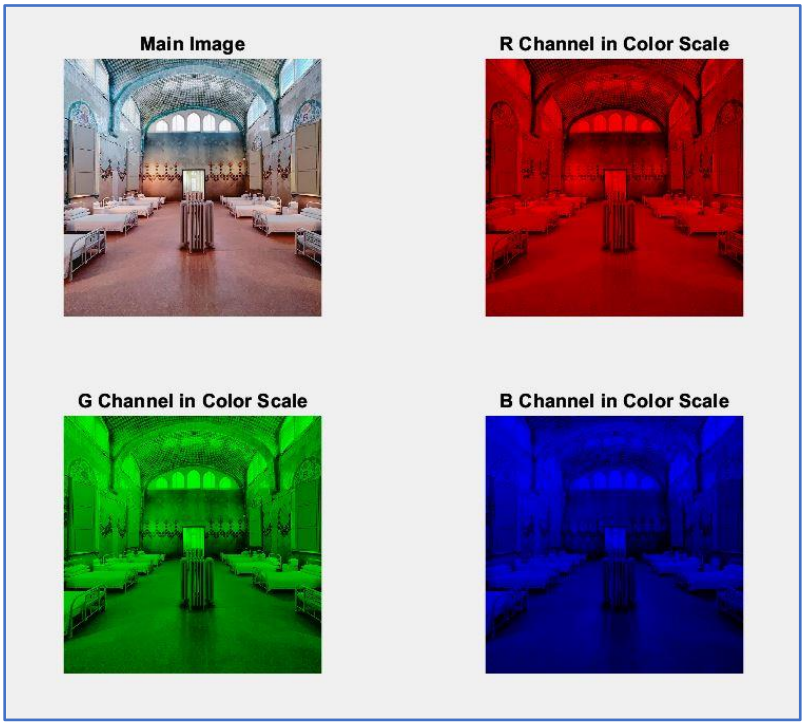
2.3. جدا کردن چنل های RGB

از آنجایی که این تصویر لود شده، یک تانسور بعدی است، کانال های مختلف آن را جدا می کنیم و نمایش می دهیم:



شکل12- نمایش کانال های RGB در مقیاس خاکستری

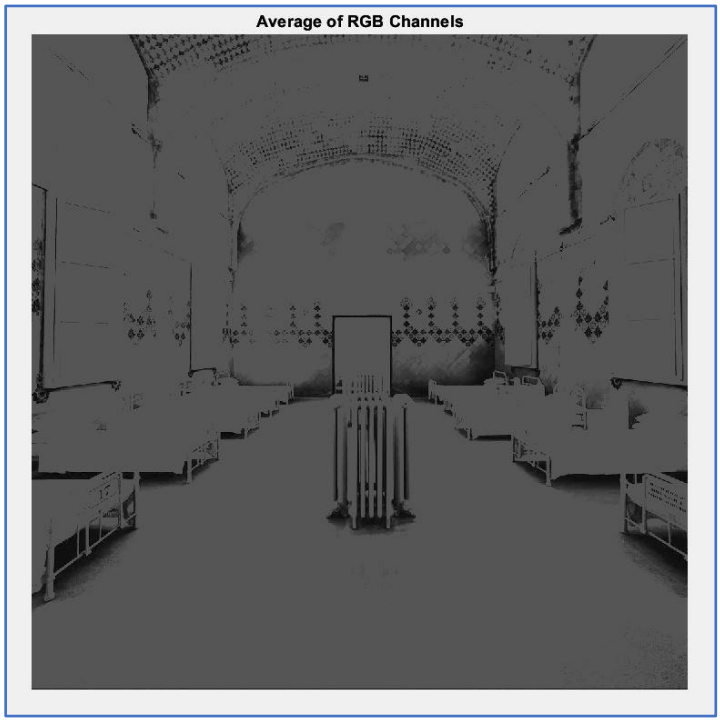
یکبار نیز به طور دلخواه، با استفاده از کد های یافته شده در اینترنت، کانال های RGB تصویر را در مقیاس رنگی نشان می دهیم:



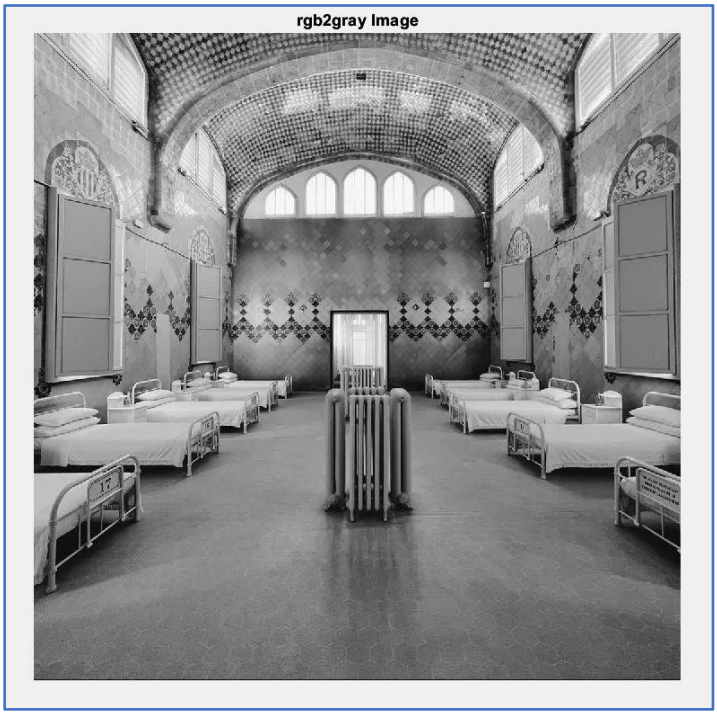
شکل13- نمایش کانال های RGB در مقیاس رنگی

3.3. برآیند گیری از کانال ها و مقایسه با مقیاس خاکستری

حال همانطور که در صورت سوال توضیح داده شده، از کانال های به دست آمده میانگین می گیریم و با نتیجه rgb2gray مقایسه می کنیم:



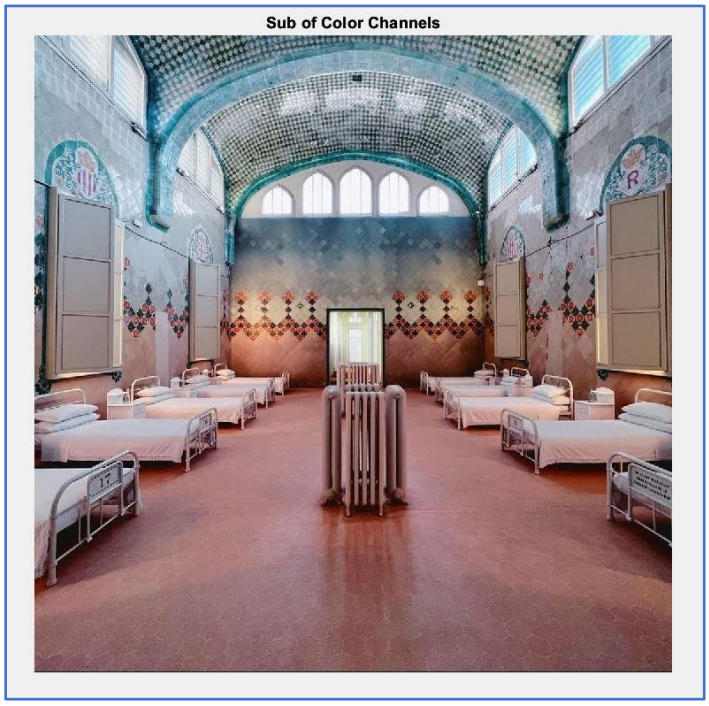
شکل14- تصویر حاصل از میانگین گیری از کانال های RGB



شکل15- تصویر حاصل از دستور rgb2gray

همانطور که مشخص است، تفاوت بسیار زیادی بین دو تصویر به دست آمده دیده می شود. دلیل این امر این است که عملیات rgb2gray صرفا یک میانگین گیری ساده نیست و فرمول بسیار پیچیده تری برای حفظ محتوای عکس دارد.

اکنون به طور دلخواه، کانال های رنگی به دست آمده در بخش قبل را با هم جمع می زنیم و حاصل را نمایش می دهیم:



شکل16- تصویر حاصل از جمع زدن کانال های رنگی

همانطور که می بینیم، تصویر به دست آمده دقیقا با تصویر اصلی ما یکسان است.

4.3. تشخیص لبه های تصویر با استفاده از فیلتر کردن تصویر

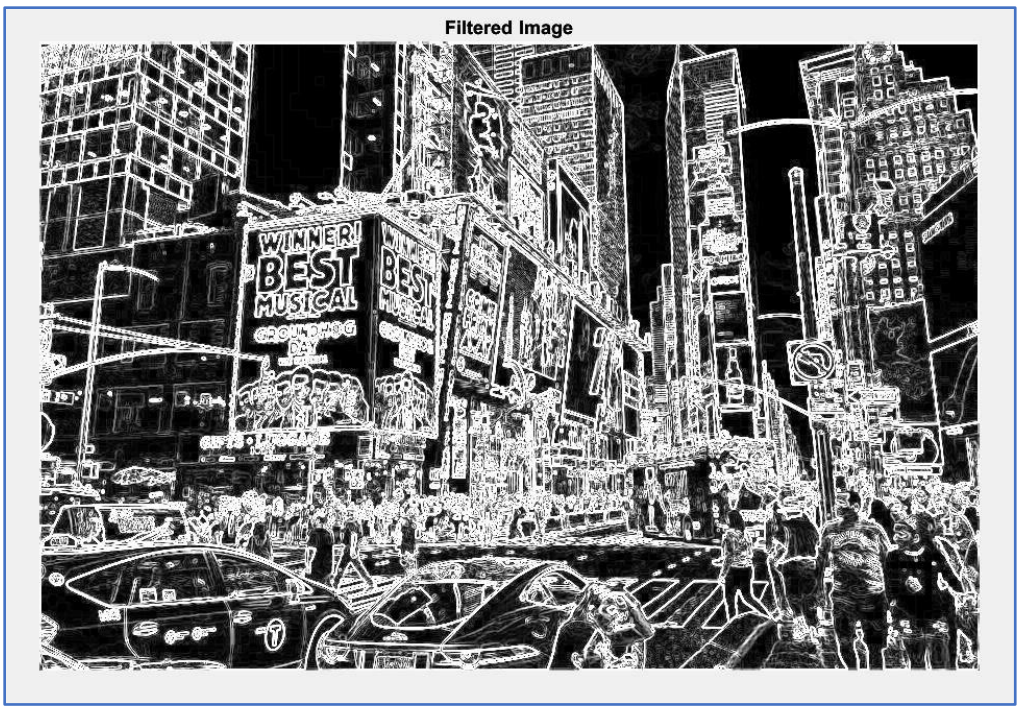
ابتدا تصویر مربوطه را در متلب لود می کنیم و با استفاده از دستور rgb2gray آن را به مقیاس خاکستری می بریم.

حال ماتریس های مربوط به فیلتر در راستای افقی و عمودی را تعریف می کنیم و سپس با دستور conv2 آن ها را با تصویر اصلی کانوالو می کنیم.

نکته ای که در انجام این کار باید در نظر بگیریم، این است که ماتریس مربوطه برای نمایش تصاویر در متلب، از جنس uint8 است. به این معنی که درایه های آن، اعداد صحیح 0 تا 255 را اختیار می کنند. همچنین ماتریس های کرنلی که برای فیلتر کردن تصویر تعریف کرده ایم از جنس double هستند. با کانوالو کردن این کرنل ها با تصویر اصلی، حاصل یک ماتریس از جنس double خواهد بود.

در نتیجه ما باید بعد از مجذور کردن ماتریس های حاصل و جمع کردن آن ها و سپس جذر گرفتن از نتیجه، ماتریس حاصل که از جنس double است را به یک ماتریس از جنس uint8 تبدیل کنیم.

در نهایت تصویر خروجی ما بعد از انجام تمام عملیات ریاضی موردنظر برای فیلتر کردن تصویر، به صورت شکلی که در صفحه بعد آورده شده، خواهد بود. همانطور که دیده می شود، در این تصویر لبه ها با وضوح بسیار بالایی تشخیص داده شده اند.



شکل17- تصویر فیلترشده نهایی

5.3. تشخیص نوع فیلتر

همانطور که در بخش قبل توضیح دادیم، عملکرد این فیلتر تشخیص دادن لبه های موجود در تصویر است.

از آنجایی که لبه ها در تصاویر حاوی فرکانس های بالا هستند در نتیجه می توان گفت که این فیلتر یک فیلتر بالاگذر است.