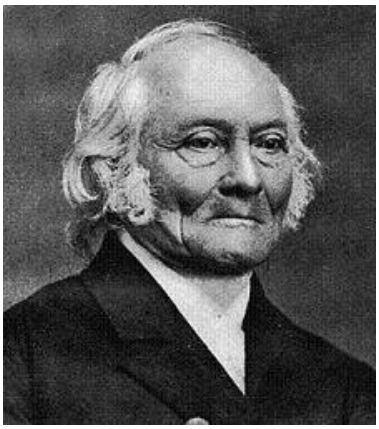
سایکوفیزیک

خوب برای این که این اسم جدید را هضم کنیم، بیایید با یک مثال شروع کنیم: فرض کنید میخواهید کیفتان را برای مدرسه آماده کنید، و باید چهار کتاب ضخمت چند صد صفحه ای را به همراه داشته باشید. نگاه تان می افتد روی کمیک بتمنی که روی تخت تان افتاده، و هنوز فرصت نکرده اید بخوانیدش.با خود میگویید که توی راه وقت دارم بخوانمش، همچنین امروز به قدری کیفم سنگین هست که شاید اصلا حس نکنم که چیزی به آن اضافه شده! به این فکر میکنید که اگر کیفتان خالی بود شاید چنین تصمیمی نمیگرفتید، چون سنگینی اش میتوانست اذیتتان کند.

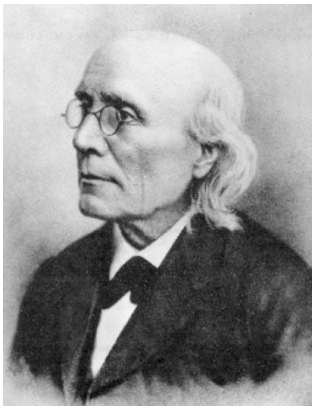


ایشان آقای وبر (Ernst Heinrich Weber) است (نترسید، میگویند قلب مهربانی داشته). آقای وبر هم، مثل شما متوجه این قضیه شده بود، که اگر به وزنه ای، وزنه ای کوچک تر اضافه شود، برای این که فرد بتواند تشخیص دهد که وزن وزنه بیشتر شده، حداقل وزن وزنه کوچکتر، به وزن وزنه بزرگتر بستگی دارد. پس برای این که بتواند این ارتباط را به صورت کمی (عددی) بررسی کند، آزمایشی طراحی کرد.

به نظرتان برای این آزمایش به چه چیز هایی نیاز داریم؟ درست است! وزنه های سنگین مان، که از اینجا وزنه های پایه صدایشان میکنیم(آقای وبر وزنه های 50،100 ... تا 500گرمی، یعنی 10 وزنه تهیه کرد). وزنه های سبک تری هم نیاز داریم، که همان هایی هستند که به سنگین ها اضافه میشوند تا ببینیم که فرد متوجه افزایش وزن شده یا نه (آقای وبر وزنه های 2، 4 تا 10گرمی، یعنی ده وزنه دیگر هم آماده کرد). یک جعبه ای، چیزی هم میخواهیم که این وزنه ها را بگذاریم داخلش و به دست فرد بدهیم، که وزنه هارا نبیند (و این جعبه باید تا جای ممکن سبک باشد تا تاثیری روی وزن نگذارد). و البته تعدادی آدم مهربان، که آزمایش مان را رویشان انجام دهیم.

خوب آقای وبر رفت و با همه این ها برگشت. برای آزمایشش، در هر مرحله یا آزمون (trial)، دو بار جعبه ای به دست راست افراد میداد (به نظرتان چرا نباید دو وزنه را همزمان به دو دست چپ و راست افراد داد؟ تفاوت در حساسیت هر دست فرد؟ باز هم درست است!). یکی از این جعبه ها، فقط وزنه پایه داشت، و دیگری وزنه پایه به همراه وزنه سبک تر.به نظرتان ترتیب دادن وزنه های سنگین و سبک در هرآزمون به فرد چگونه باید باشد؟ آفرین، به صورت تصادفی. چون ممکن است الگویی که انتخاب میکنیم، روی پاسخ های فرد اثربگذارد. پس در هر آزمون، دو بار جعبه ای به فرد میدهیم، و بعد از او میپرسیم که از نظرش، کدام جعبه سنگین تر بود.

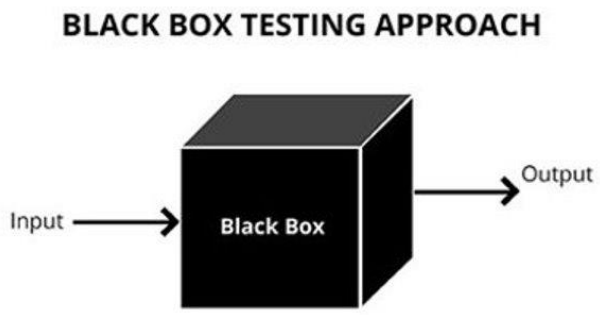
به دلایل متفاوتی، در هر آزمایش خطا های متعددی وجود دارد. برای این که اثر این خطا ها را در نتیجه کلی، تا آنجایی که میتوان کمرنگ تر کرد، باید آزمایش را چندین بار تکرار کنیم. مثلا در مورد آزمایش آقای وبر، ما 100 حالت آزمایش داریم (10 وزنه پایه * 10 وزنه سبک). اگر بخواهیم هر حالت را 10 بار هم تکرار کنیم، میشود 1000 آزمون برای هر فرد (زیاد شد نه؟) ،به دلیل تفاوت های فردی، نمیتوان به پاسخ های یک نفر اعتماد کرد، و باید آزمایش را روی چندین نفر انجام داد (ببینید صبر وبر چقدر بالا بوده).



آقای وبر از آزمایشش نتایج مورد نظر را گرفت، و فرضیه اش تایید شد؛ یعنی دید که وزنی که باید به وزنه پایه اضافه شود تا فرد متوجه تفاوت وزن این دو وزنه شود، نسبتی ثابت است از وزنه پایه. بعدا، انسان دیگری، به نام آقای فخنر (Gustav Theodor Fechner) که چهره اش را در این تصویر میبینید، تحلیل روابط به دست آمده از کار آقای وبر را ادامه داد و رابطه ای کمی پیچیده تر (که رفتار افراد را بهتر پیشبینی میکرد) برای میزان حس شدن محرک پیشنهاد کرد. رابطه او بیان میکند میزان حس شدن یک محرک (sensation)، ضریبی از لوگاریتم طبیعی شدت محرک (intensity of stimulus) است (حالا مهم نیست که لوگاریتم طبیعی چیست. شما لب مطلب را دریابید.) محرک، هر شیئ یا پدیده ای است که بتواند در صورت داشتن شدت مناسب، باعث ایجاد حس یا ادراکی در فرد مورد آزمایش (subject) شود.

خوب، حالا ببینیم حس و ادراک چیست. حس شدن یک محرک (sensation)، یعنی تشخیص داده شدن آن توسط گیرنده های حسی. و ادراک (perception)، به معنای نحوه سازماندهی، برداشت و تجربه ای است که از اطلاعات حسی برای فرد حاصل میشود.

سایکوفیزیک، بررسی علمی رابطه بین محرک و حس یا ادراک است. در سایکوفیزیک، ذهن و یا کارکرد های مغز را جعبه ای سیاه در نظر میگیریم، که با مطالعه رابطه بین ورودی (محرک) و خروجی (ادراک یا حس سابجکت)، میتوانیم به ویژگی های شیئ مرموز درون جعبه پی ببریم.

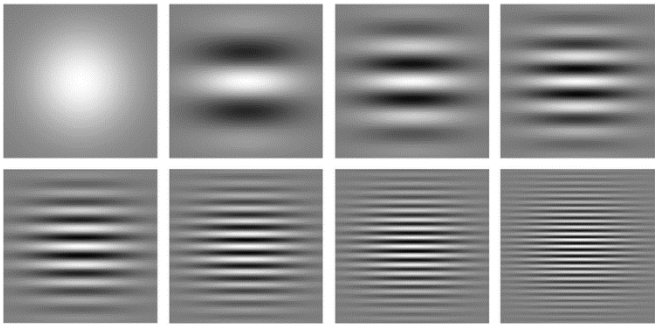


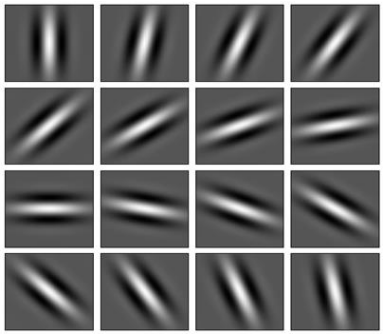
یک آزمایش خوب سایکوفیزیکی، مانند هر آزمایش دیگری، اول با یک سوال شروع میشود، که خوب آن سوال نتیجه مشاهده دقیق است (مانند کار دوست خوبمان آقای وبر). با داشتن سوال خوب، در حقیقت پدیده ای را برای بررسی بیشتر انتخاب میکنیم. با گزینش پدیده، در حقیقت داریم قدم اول مان را برای طراحی آزمایش سایکوفیزیک برمیداریم، یعنی انتخاب **محرک (stimulus)**.

دست مان در این مرحله خیلی باز است! محرک ها میتوانند در حس های مختلف باشند، حتا دو حس را همزمان درگیر کنند، و یا حتا برای گونه هایی به جز انسان ها انتخاب شوند.

مهم است که بتوانیم ویژگی های قابل اندازه گیری محرک مان را بشناسیم. در عمده آزمایش ها، ما تمام شرایط را ثابت درنظر میگیریم، و تنها متغیر مان همان ویژگی از محرک است که میخواهیم بررسی اش کنیم. پس لازم است همه ویژگی های محرک مان را بشناسیم تا بتوانیم همه مواردی که در آزمایش مان مورد بررسی نیستند را ثابت در نظر بگیریم!

به این شکل دقت کنید. نام این محرک ها، gabor است. با توجه به حالات مختلفی که در این تصاویر میبینید، به نظرتان ویژگی های گبور چه میتواند باشد؟





دقیقا! زاویه خطوط مان با افق، تعداد خطوط سیاه و سفید (یا فرکانس)، و میزان شفافیت تصویر (که میتوان با آن سختی تشخیص را تنظیم کرد).

حالا که محرک مان را انتخاب کردیم، باید به این هم فکر کنیم سابجکت هایمان با این محرک چه باید بکنند؟ در حقیقت، باید ببینیم چگونه میتوانیم محرک مان را برای نمونه ها به یک سوال تبدیل کنیم. در این مرحله، در حال طراحی **آزمون** (task) آزمایش مان هستیم!

یک نوع تسکی که میتوان در نظر گرفت اینگونه است که از سابجکت مان بخواهیم زمانی که یک محرک مشخص را تشخیص داد، به ما پاسخ بدهد (مثلا زمانی که در وسط صفحه تماما خاکستری مانیتور، یک دایره سفید مانند شکل ظاهر شد، دکمه ای را در کیبورد فشار دهد). به این نوع تسک، **تشخیص** (detection) میگوییم.

یا مثلا میتوانیم از افراد مورد آزمایش مان بخواهیم که مقدار یک محرک را برایمان تخمین بزنند. البته دراین نوع تسک، ابتدا باید برای سابجکت مان مقدار ها را تعریف کنیم! یعنی مثلا برای همان تسک دایره نورانی وسط صفحه، روشن ترین حالت را 10، کمترین درجه روشنایی را 1، و مقادیر بین این دو را با نسبت عدد دهی کنیم و به سابجکت زمان بدهیم تا مقدار ها را یاد بگیرد، حال همان دایره می آید روی صفحه و از فرد میخواهیم مقداری برای روشنایی آن تخمین بزند. به این نوع تسک ها هم، تسک **تخمین مقدار** (magnitude estimation) میگوییم.

حال ممکن است بخواهیم سابجکت مان دو محرک را با هم مقایسه کند. مثلا در مورد همان دایره ها، مانند تصویر، دو دایره به او نشان بدیم،و از او بخواهیم مشخص کند کدام یک روشن تر است. به این نوع تسک ها هم، تسک **تشخیص** (discrimination) میگوییم.

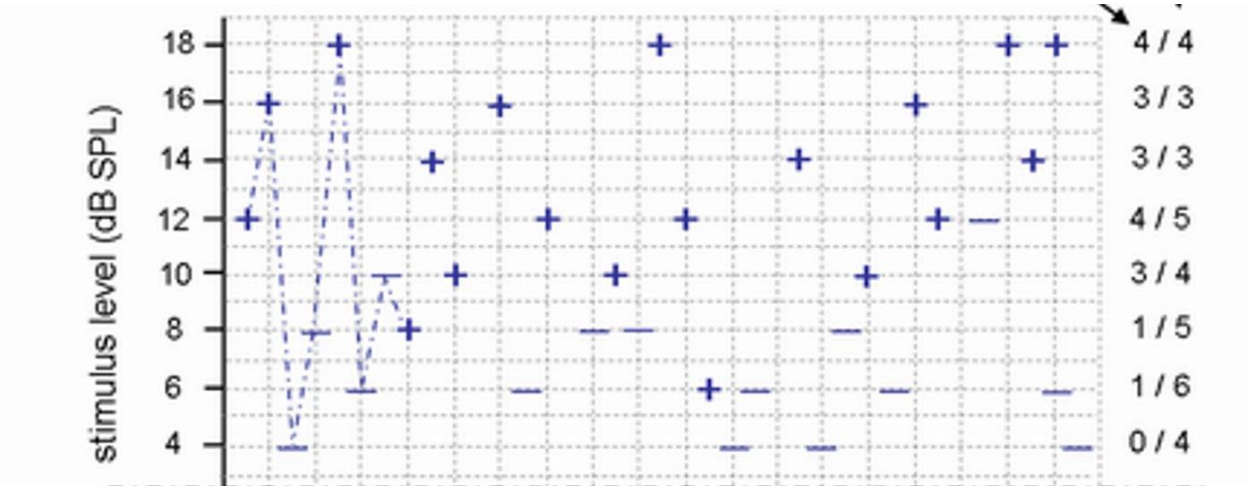
برای طراحی تسک مان میتوانیم صرفا یک سوال بله یا خیر از سابجکت بپرسیم: دایره را دیدی یا نه؟ (مثل تسک تشخیص). یا میتوانیم در تسک تشخیص مان، دو گزینه تصویر راست و تصویر چپ را به فرد بدهیم، و از او بخواهیم حتما یکی از این دو گزینه را انتخاب کند. به این حالت، two alternative forced choice تسک میگوییم. چرا میخواهیم که حتما جواب بدهد؟ چون برای مقایسه سابجکت های مختلف، اگر تعداد سوال هایی که پاسخ داده اند متفاوت باشد، مقایسه پاسخ هایشان سخت میشود، همچنین ممکن است مثلا یک سابجکت تصمیم بگیرد از نیمه آزمایش به بعد هیچ کدام را جواب ندهد، یعنی اثر خستگی روی جواب هایش را مشاهده نخواهیم کرد، پس نمیتوان جواب او را با کسی که تمام سوال ها را پاسخ داده مقایسه نمود.

در طراحی آزمون مان، باید حواس مان به خطاهای که در جریان آزمایش میتواند پیش بیاید باشد، تا بتوانیم آن ها را به حداقل برسانیم. یکی از مواردی که قبلا راجع به آن صحبت کردیم، ثابت بودن عوامل موثر در آزمایش است (عوامل غیر متغیر مان). مثلا محیط سابجکت هایمان باید ثابت باشد، یا تا حد ممکن به هم شبیه باشد (از یکی وسط بازار آزمون نگیریم و دیگری در اتاق ایزوله). حتی در بعضی تسک ها ممکن است عواملی مانند نور محیط روی کار ما اثر بگذارند. باید توجه

کنیم که تسک مان قابل اعتماد باشد، یعنی اگر خواستیم همین آزمون را با گروه نمونه دیگری تکرار کنیم، نتایج مشابه بگیریم. همچنین تسک مان باید قابل تکرار هم باشد، یعنی اگر پژوهنده دیگری خواست کار ما را تکرار کند، به نتایج مشابهی برسد.

حالا که سوال مان را پیدا کردیم، باید ببینیم چگونه باید آن را از سابجکت ها بپرسیم. به این مرحله از طراحی تسک، sampling method میگوییم.

یکی از راه هایی که ممکن است سریع به ذهن برسد، انتخاب ترتیب رندوم برای آزمون هایمان است. اگر اول آزمایش مان، بعد از این که همه سوال هایمان را آماده کردیم، یک بار ترتیبشان را بر بزنیم (randomize کنیم)، و تا آخر با ترتیب به دست آمده پیش برویم، از **روش محرک های ثابت** (method of constant stimuli) استفاده کرده ایم. به شکل زیر توجه کنید:



به نظرتان نوع تسک این آزمایش چه چیز هایی میتواند باشد؟

سوالی که ممکن است برایتان پیش بیاید این است که خوب که چی؟ الان ما پاسخ های درست و غلط فرد را جمع میکنیم که چه بشود؟ و پاسخ ما این خواهد بود که صبر داشته باشید و عجله نکنید. کمی جلو تر میبینیم که با داده های به دست امده از این آزمایش ها، چه میتوان کرد.

همچنین میتوان تنظیم شدت محرک را بر عهده خود سابجکت گذاشت! اسم این دستگاه knob (پیچ تنظیم) است. قضیه از این قرار است که knob، یک پیچ است که فرد میتواند با پیچاندن آن، شدت محرک را تنظیم کند. مثلا میتوانیم بگوییم انقدر صدا را زیاد کن تا بشنوی، یا آنقدر صدا را کم کن که دیگر نشنوی. به این روش که در آن تنظیم شدت محرک در دست خود سابجکت است، **روش تنظیم** (adjustment) میگوییم.

حتی میتوان شدت محرک را، با توجه به پاسخ قبلی خود سابجکت انتخاب کرد.(برخلاف محرک های ثابت، که ترتیب محرک ها در آن ثابت است. فکر کنم اسمش به اندازه کافی گویا بود. خیلی توضیح نمیخواست، نه؟) شکل زیر مربوط به **روش حد ها** (method of limits) است. ببینیم میتوانیم از ساز و کار آن سر در بیاوریم:

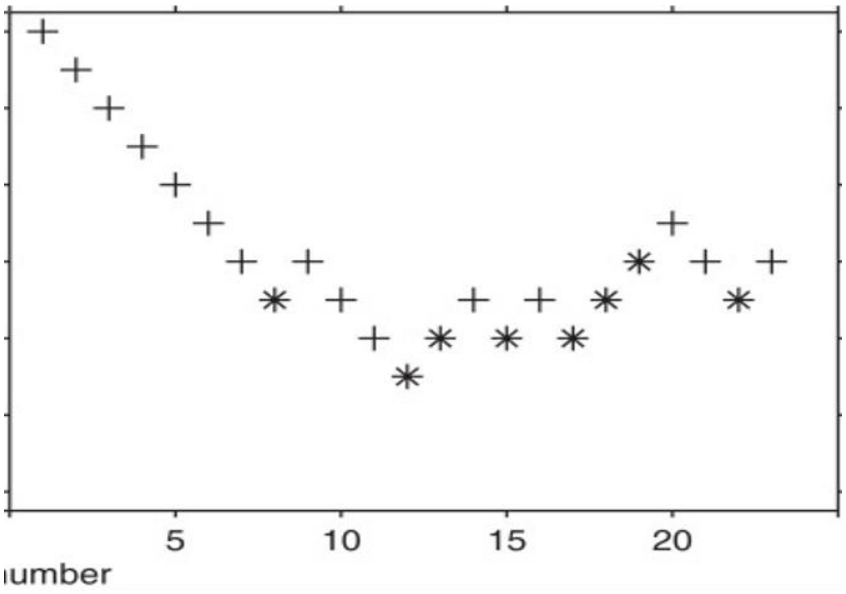
Stimulus Intensity	A	D	A	D	A	D	A	D
9		Y						
8		Y				Y		Y
7		Y		Y		Y		Y
6		Y		Y		Y		Y
5		Y		Y		Y		Y
4	Y	Y		Y		Y		Y
3	N	Y	Y	Y	Y	Y		Y
2	N	N	N	Y	N	N	Y	Y
1	N	N	N	N	N		N	N
0	N		N		N		N	
-1	N		N		N		N	
-2	N		N		N		N	
-3	N		N		N		N	
-4	N		N		N		N	
-5	N		N				N	
-6	N						N	
Transition Points	3.5	2.5	2.5	1.5	2.5	2.5	1.5	1.5

Mean Threshold = 2.25

خوب جریان در این روش اینگونه است که ابتدا، از یکی از اکسترمم های بازه (ماکزیمم شدت مصرک یا مینیمم آن) شروع میکنیم. مثلا اگر از ماکزیمم بازه مان شروع کرده باشیم، سابجکت (با احتمال خیلی خوبی) میتواند مصرک را تشخیص دهد (یا اگر در مورد مقدار تفاوت دو مصرک مختلف است، تمایز). پس از جواب مثبت سابجکت، یک درجه میزان شدت مصرک را کم میکنیم. و این روند را ادامه میدهیم، تا جواب سابجکت تغییر کند (مثلا بگوید که مصرکی تشخیص ندادم). حالا از پایین بازه شروع میکنیم. در این روش، نقاطی برای ما اهمیت دارند که جواب سابجکت عوض شده است.

به نظرتان الگو دار بودن سوال هایمان روی جواب سابجکت چه تاثیری میتواند بگذارد؟

روش مد ها یک نمونه تغییر یافته و کاربردی تر هم دارد. به این روش **روش پلکان (staircase method)** میگوییم:



در این روش هم از یکی از اکسترمم های بازه شروع میکنیم، و تا عوض شدن نحوه پاسخ دهی سابجکت مان شدت محرک را متناسب با پاسخش تغییر میدهیم. اما وقتی جریان پاسخ هایش عوض شد، به جای شروع از ابتدای بازه، از همان نقطه شدت محرک مان را بر خلاف جهت قبلی اش ادامه میدهیم. اینجا هم نقاط تغییر روند پاسخ نمونه (پاکرد ها) برایمان مهم اند.

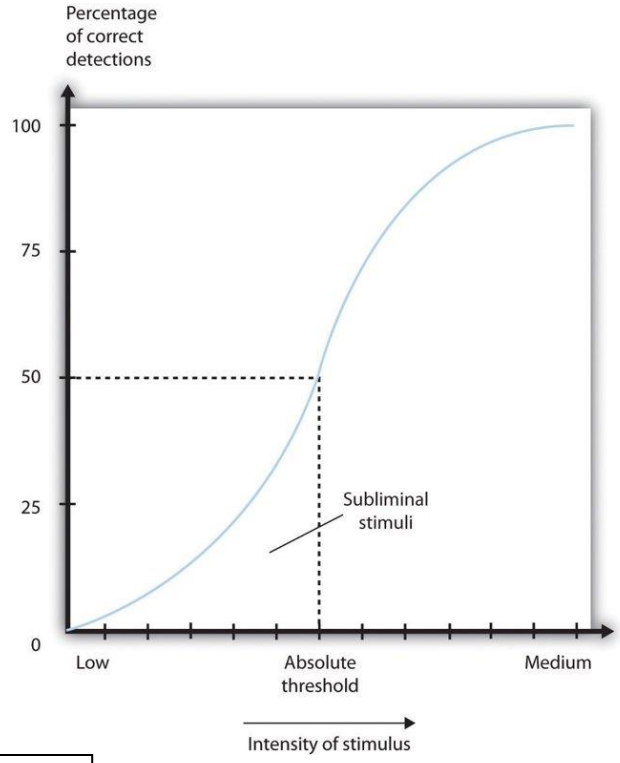
خوب حالابرویم سراغ سوالی که اول صفحه پیش پرسیدید. اطلاعات به دست آمده از آزمایشمان چیست؟ و چگونه میتوان از ان استفاده کرد؟ با **روش های اندازه گیری عملکرد (performance measurement method)**، ما میتوانیم پاسخ های سابجکت مان را بسنجیم. شدت محرک مان را هم که دیدیم چگونه میشد اندازه گیری کرد، سایکوفیزیک هم که رابطه کمّی بین این دو تاست. تمام شد رفت (:

یکی از چیز هایی که میتوان اندازه گیری کرد، **زمان پاسخ دهی (reaction time)** سابجکت هاست. مثلا درمورد تسک تشخیص مان (دایره های سفید وسط صفحه خاکستری را یادتان هست؟) میتوانیم زمانی که طول میکشد تا پس از ظاهر شدن دایره سفید روی صفحه، سابجکت دکمه را فشار دهد را به عنوان reaction time او اندازه گیری کنیم. به نظرتان عواملی مانند تمرین زیاد، خستگی و عوامل محیطی حواس پرت کن چه تاثیری میتوانند روی زمان پاسخ فرد بگذارند؟

یک مورد دیگر، عملکرد، یا درصد پاسخ های صحیح است (performance). بیایید شکل مربوط به روش محرک های ثابت را یکبار دیگر نگاه کنیم: در سمت راست این نمودار، عملکرد نمونه در هر شدتی از محرک که در سمت چپ ذکر شده، نوشته شده.

به نظر شما، بین زمان پاسخ و عملکرد چه رابطه ای وجود دارد؟ اگر روی آزمونی محدودیت زمانی بگذاریم، به نظرتان چه تاثیری روی پاسخ فرد میگذارد؟

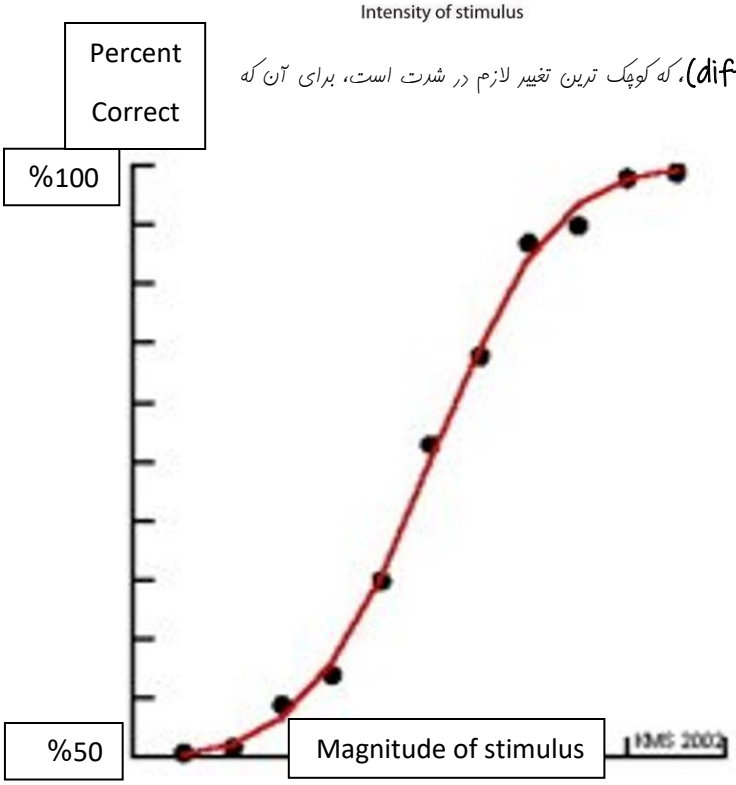
میرسیم به بحث شیرین **حد آستانه ها** (thresholds). حد آستانه مطلق، عبارت است از کوچک ترین شدت محرک، برای آن که در سابجکت ایجاد حس یا ادراک کند. برای مثال، در روش حد ها، میتوان گفت آن نقاطی که پاسخ سابجکت با پاسخ قبلی اش متفاوت بود، و در روش پلکانی، پاکرد ها، نشان دهنده حد آستانه هستند (که اگر تسک مان از نوع تشخیص باشد، میشود حدآستانه مطلق). پس میتوانیم با کمی تحلیل آماری (که برای آن میتوانید از دوستانتان در کارگاه آمار راهنمایی بگیرید) روی آن داده ها، حدآستانه را مشخص کنیم.



به نمودار روبرو توجه کنید. در محور افقی آن، شدت محرک، و در محور عمودی آن، عملکرد نمونه قرار گرفته. میبینیم که با افزایش شدت محرک، احتمال درست پاسخ دادن نمونه هم بیشتر میشود. نقطه عطف نمودار ما، حدآستانه مطلق است، که شدت آن در این نمودار، در نقطه پاسخ دهی 50٪ نمونه ها قرار گرفته است.

به نظرتان با دستگاه پیچ تنظیم چگونه میتوان حدآستانه مطلق را اندازه گیری کرد؟

نوعی دیگر از حدآستانه هم داریم، به نام **حدآستانه تفاوت** (difference threshold, or just noticeable difference)، که کوچک ترین تغییر لازم در شدت است، برای آن که در تجربه حسی فرد، تغییر قابل تشخیصی ایجاد شود. برای تسک های تمایز، میتوان حدآستانه تفاوت را تعریف کرد.

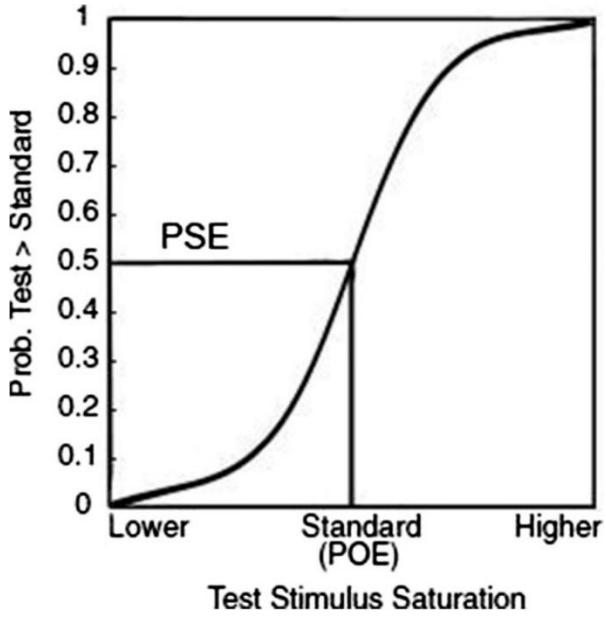


در یک تسک تمایز two alternative forced choice، نمودار روبرو از پاسخ های سابجکت به دست آمده. اگر به نقطه شروع نمودار توجه کنید، میبینید که از 50٪ شروع میشود. در حقیقت این ویژگی نمودار مان، به نوع سوال مان برمیگردد. وقتی دو گزینه به فرد میدهیم، رندوم هم جواب بدهد 50٪ درست انتخاب میکند دیگر! پس نقطه عطف نمودار، که همان حدآستانه تفاوت است، جایی حدود 75٪ عملکرد قرار میگیرد.

راستی به این نمودار هایی که در محور افق آن شدت محرک را داریم و در محور عمود آن یکی از روش های اندازه گیری عملکرد، تابع روان سنجی (psychometric function) میگویند.

یکی دیگر از روش های اندازه گیری عملکرد، **نقطه هم ارزی نسبی** یا **point of subjective equality** است. این نقطه، جایی است که از نظر سابجکت، دو محرک یکسان اند. در برابر آن، نقطه هم ارزی شیئی یا point of objective equality قرار دارد، که در آن در حقیقت دو محرک واقعا یکسان هستند.

در حقیقت، در نقطه هم ارزی نسبی، سابجکت چون نمیتواند بین دو محرک تفاوتی قائل شود، بین آن ها رندوم انتخاب میکند، پس میتوان آن را نقطه 50٪ عملکرد دانست.



طراحی تسک سایکوفیزیک به کمک متلب و سایکتولباکس

خب در این بخش از جزوه می خواهیم به شما پیاده سازی task سایکوفیزیک با متلب(MATLAB) و سایکتولباکس (Psychtoolbox) را بررسی کنیم اما اول نیاز است کمی با نرم افزار متلب و کتاب خانه سایکتولباکس آشنا شوید.

متلب یک زبان برنامه نویسی سطح بالا بر پایه C است که در سال های اخیر محبوبیت زیادی بین محققان و مهندسانی که به برنامه نویسی صرفا به چشم یک ابزار نگاه میکنند پیدا کرده است(گرچه برخی برنامه نویسان حرفه ای هنوز متلب را به دلیل ساده بودن رابط کاربری به عنوان زبان برنامه نویسی قبول ندارند و به چشم نرم افزار به آن نگاه می کنند:D)

به طور کلی می توان عامل محبوبیت متلب را در کتابخانه های فراوان و کاربردی آن خلاصه کرد. یکی از کتابخانه های بسیار کاربردی متلب برای طراحی تسک های روانشناسی و سایکوفیزیک کتابخانه psychtoolbox می باشد که به کمک آن میتوان تسک های دیداری و شنیداری بسیاری را بر پایه Open GL طراحی کرد و به راحتی پاسخ شخص مورد آزمایش را ثبت کرد.

باید به این نکته توجه کنید که سایکتولباکس برنامه مجزایی نیست کتابخانه ای است که باید بر روی متلب خود آن را نصب کنید و انتظار دیدن محیط گرافیکی جدیدی را بعد نصب آن نداشته باشید. تنها اتفاقی که بعد از نصب سایکتولباکس رخ می دهد اضافه شدن تعدادی دستور جدید به متلب است که در ادامه با تعدادی از این دستور ها آشنا می شوید.

شیوه نصب سایکتولباکس بر روی متلب به طور کامل در لینک زیر توضیح داده شده است.

http://highschool.cognsc.ir/psychtoolbox-installation/

* پس از نصب سایکتولباکس حتما با نوشتن دستور PsychtoolboxVersion در command window متلب از درست بودن نصب خود مطمئن شوید(در صورت درست بودن نصب باید عددی به عنوان ورژن نمایش داده شود و در صورت وجود مشکل با ارور مواجه می شوید.

* سایکتولباکس بر روی متلب ورژن قبل 2014 به درستی کار نمیکند. برای استفاده از سایکتولباکس ورژن متلب باید 2014 یا بعد از آن باشد.

خب حال که توانستید سخت ترین قسمت کار یعنی نصب سایکتولباکس را با موفقیت پشت سر بگذارید نوبت شروع قسمت جذاب کار یعنی برنامه نویسی است :))

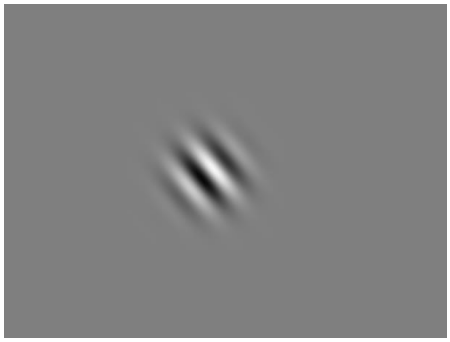
برای نوشتن یک برنامه مخصوصا طراحی یک تسک سایکوفیزیک باید قبل از درگیر شدن با دستورات یا به اصطلاح syntax ها بتوانید تک تک مراحل تسک خود را به ساده ترین شیوه ممکن طراحی کنید تا قابل تفهیم برای کامپیوتر باشد. به این کار الگوریتم نویسی میگویند و می توان گفت مهم ترین و در عین حال جذاب ترین بخش پیاده سازی تسک است.

برای ادامه جزوه ابتدا یک تسک ساده سایکوفیزیک را با هم در نظر میگیریم و الگوریتم و کد آن را قدم به قدم طراحی میکنیم.

تسک برین صورت است که میخواهیم به سابجکت(شخص مورد آزمایش) 16 gabor با 8 زاویه مختلف از 10 تا 80 درجه در 2 حالت متمایل به چپ و متمایل به راست نسبت به افق نشان دهیم و ببینیم در چه زاویه ای سابجکت میتواند جهت gabor را تشخیص دهد. فرض میکنیم از سابجکت خواسته می شود اگر gabor متمایل به راست بود فلش سمت راست(right arrow) و اگر gabor متمایل به چپ بود فلش سمت چپ(left arrow) را فشار دهد.

Gaborمتمایل به راست

Gabor متمایل به چپ



بنابراین سابجکت باید در زاویه های مختلف انحراف gabor نسبت به افق تشخیص دهد gabor به سمت راست متمایل است یا چپ

خب همان طور که واضح است این تسک 3 قسمت اصلی دارد:

1. نشان دادن gabor به سابجکت

2. ثبت پاسخ سابجکت

3. مقایسه پاسخ سابجکت با جهت واقعی gabor و بررسی صحت پاسخ سابجکت و زمان مورد نیاز برای پاسخ دادن سابجکت

حال باید این 3 قسمت را تا حد ممکن ساده کرد و توضیح داد تا برای کامپیوتر قابل فهم شود. همواره به خاطر داشته باشید کامپیوتر نفهم ترین موجودی است که می توانید با آن سر و کار داشته باشید بنابراین تا جایی که می توانید باید تسک را ساده و شفاف سازی کنید.

اولین سوالی که ممکن است برای هر یک از شما پیش بیاید این است که تعداد gabor های متمایز که می تواند به عنوان محرک مورد استفاده قرار بگیرد چند تاست؟

خب همان طور که گفته شد ما میخواهیم gabor های با زاویه 10 تا 80 درجه (یعنی 10و20و30و....و80) درجه را به سابجکت نشان دهیم تا جهت آن را تشخیص دهد بنابراین 8 زاویه مختلف برای gabor داریم و از طرف دیگر هر gabor ممکن است متمایل به چپ یا راست باشد بنابراین 2 حالت نیز برای جهت gabor داریم یعنی تعداد کل محرک های متمایزی که می توانیم به سابجکت نشان دهیم 16=8*2 محرک است.

برای طراحی این تسک نیز منطقی ترین شیوه استفاده از block design است بدین صورت که تعدادی بلوک در نظر گرفته می شود که در هر بلوک هر 16 محرک با ترتیب تصادفی به سابجکت نشان داده می شود بنابراین تعداد بلوک ها نمایانگر تعداد تکرار هر محرک است که هر چه بیشتر باشد نتیجه حاصل ضریب اطمینان بیشتری دارد.

*به این نکته توجه کنید که نمیتوانید به صورت تصادفی تعدادی محرک را به سابجکت نشان دهید. در هر بلوک باید تمام محرک ها به سابجکت نشان داده شوند و فقط ترتیب نشان دادن آن ها در هر مرحله به صورت تصادفی تغییر کند.

* به دلیل پیچیده بودن دستورات اجرا شده توسط سایکتولباکس نمیتوانید در حین نشان دادن محرک حالت آن محرک را بسازید. پیش از آغاز کار سایکتولباکس باید در متلب ماتریسی برای تعریف شرایط هر آزمایش تمام بلوک ها بسازید. این ماتریس به ماتریس شرایط(condition matrix) معروف است.(درباره ماتریس هر چه میخواهید بدانید از عضو حاضر در کارگاه سیگنال بپرسید. تا دلتان بخواهد در کارگاه درباره ماتریس اطلاعات جمع کرده)

ماتریس شرایط برای تسک gabor ما شکلی تقریبا مشابه شکل رو به رو دارد:

Block.NO	Degree	Side
1	40	2
1	60	1
1	10	1
1	50	1
1	20	1
1	50	2
1	70	1
1	60	2
1	20	2
1	70	2
1	80	2
1	10	2
1	30	1
1	30	2
1	80	1
1	40	1

این مثالی برای ماتریس شرایط تسکی با یک بلوک است همان طور که میبینید تمام 16 حالت ممکن با ترتیب تصادفی در آن وجود دارد. اگر بخواهیم تعداد بلوک ها را اضافه کنیم باید ترتیب این 16 ردیف را دوباره رندوم کرده و با شماره بلوک 2 در زیر بلوک 1 قرار دهیم.

به دلیل درگیر نشدن با پیچیدگی ذخیره کردن حروف در ماتریس بهتر است برای نشان دادن جهت راست و چپ کدی تعریف کنید که با آن کد جهت مشخص شود. در این مثال، جهت راست با کد 1 و جهت چپ با کد 2 نمایش داده شده است.

این ماتریس باید به تعداد بلوک های ممکن ساخته شود. واضح است که در این صورت تعداد سطر های ماتریس برابر (تعداد کل حالات*تعداد بلوک ها) خواهد بود و تعداد ستون ها برابر (تعداد شرایط تاثیر گزار +1) خواهد بود که آن 1 به دلیل ستون شماره بلوک است.

ساختن ماتریس شرایط برای یک تسک در حال حاضر جز انتظارات ما از شما نیست ولی به طور کلی ماتریس شرایط را می توان به کمک یک حلقه for تو در تو یا روش های هوشمندانه تر همانند توزیع باینری تولید کرد. در صورت علاقه می توانید درباره آن سرچ کنید و اگر شیوه جدید و ابتکاری برای آن پیدا کردید ما را نیز در جریان بگذارید ولی ساخت ماتریس شرایط از سطح این جزوه فرا تر است و در این جزوه تنها استفاده از آن بررسی می شود.

خب پس نتیجه تا اینجا بدین صورت است که یک ماتریس شرایط داریم که هر سطر از ماتریس شرایط ویژگی های محرک آن آزمایش(trial) را نشان می دهد یعنی بیان میکند در این آزمایش محرک باید با چه زاویه ای و در چه جهتی بیاید

حال باید از خود بپرسیم بعد از نشان دادن محرک چه اتفاقی می افتد؟ آیا محرک برای همیشه روی صفحه می ماند؟ یا پس از زمانی باید از روی صفحه محو شود؟

به طور منطقی اگر میخواهیم ببینیم سابجکت از چه درجه ای از انحراف از سطح افق را می تواند تشخیص دهد باید محرک زمان کوتاهی(به عنوان مثال 500 میلی ثانیه) روی صفحه باشد و سپس برود و پس از آن از سابجکت پرسیده شود جهت محرک به کدام سمت بود.

خب 3 مرحله اصلی تسک یعنی نشان دادن محرک به سابجکت، ثبت جواب سابجکت و بررسی صحت جواب سابجکت و زمان پاسخگویی را به خاطر دارید؟

مرحله اول یعنی نشان دادن محرک به سابجکت را می توان در مراحل زیر توصیف کرد:

1. به دست آوردن زاویه و جهت gabor در آزمایش مورد نظر از روی ماتریس شرایط

2. نشان دادن gabor با شرایط به دست آمده بر روی صفحه نمایش

3. صبر کردن به مدت 500 میلی ثانیه

4. رفتن شکل از روی صفحه نمایش

خب حال که مرحله اول را انجام دادید باید به سراغ مرحله دوم یعنی ثبت پاسخ سابجکت برویم(خبر خوب: این مرحله ساده ترین و کوتاه ترین مرحله است!)

در این مرحله باید صبر کنیم تا سابجکت جواب دهد و ببینید سابجکت کدام دکمه از کیبورد را فشار داد و زمان پاسخگویی او(فاصله زمانی میان رفتن محرک از روی صفحه تا جواب دادن سابجکت) را محاسبه کنیم.

مرحله اول بسیار ساده است. سایکتولباکس دستوری به نام Kbwait به دستورات متلب اضافه می کند. وظیفه این دستور این است که کد متوقف می شود تا سابجکت دکمه ای از کیبورد را فشار دهد و سپس با کمک دستور دیگری به نام KbCheck می توان کد دکمه ای که سابجکت فشار داده را نیز پیدا کرد. یکی دیگر از ویژگی های KbCheck نیز این است که علاوه بر کد دکمه ای که سابجکت فشار داد تابع زمان فشار دادن دکمه را نیز برمیگرداند(تابع زمان به طور کامل در پاراگراف بعد توضیح داده شده)

در متلب دستوری به نام GetSecs وجود دارد. این دستور عددی را به عنوان تابعی از زمان برمیگرداند. خود این عدد معنی و مفهوم خاصی ندارد ولی می توان از آن برای پیدا کردن زمان استفاده کرد.

همان طور که حدس میزنید خروجی تابع GetSecs با زمان متغیر است یعنی اگر الان تابع GetSecs را فراخوانی کنیم و 2 ثانیه بعد دوباره مقدار GetSecs را فراخوانی کنیم خروجی 2 عدد متفاوت خواهد بود(این کار را در متلب امتحان کنید تا به چشم خودتان ببینید!) و دوباره همان طور که پیش بینی میکنید اگر مقدار GetSecs را در یک لحظه بگیرید و درون یک تابع دیگر ذخیره کنید و 2 ثانیه بعد دوباره مقدار GetSecs را بگیرید و مقدار آن را در یک تابع جدید ذخیره کنید اختلاف مقدار این 2 تابع برابر 2 ثانیه خواهد بود در واقع یعنی اختلاف مقدار GetSecs در 2 زمان متفاوت اختلاف زمانی آن دو نقطه را نمایش می دهد.

(قبل از رفتن به ادامه بحث پیشنهاد می شود پاراگراف بالا را چند بار خوب بخوانید تا متوجه شوید زیرا درک کامل آن بسیار مهم است)

خب اگر یادتان باشد دیدیم تابع Kbwait صبر می کند تا سابجکت دکمه ای را روی کیبرد فشار دهد. بنابراین اگر یک بار قبل از Kbwiat با استفاده از GetSecs آن عدد عجیب و غریب نمایانگر زمان را بگیریم و در یک متغیر به نام TimeStart ذخیره کنیم و تابع زمان لحظه پاسخ دادن نیز به کمک KbCheck به دست می آید. می توان با به دست آوردن اختلاف این 2 مقدار زمان پاسخگویی سابجکت یا به اصطلاح reaction time سابجکت را محاسبه کرد.(اگر این قسمت را خیلی خوب متوجه نشدید اشکالی ندارد. فهمیدن طرز کار GetSecs و KbCheck نیاز به درک دستور های سایکتولباکس دارد که هنوز یاد نگرفته اید. پیشنهاد می شود بعد از خواندن ادامه جزوه و هنگام مطالعه کد آپلود شده در github دوباره این بخش را بخوانید. آن موقع کامل متوجه مفهوم این دو دستور خواهید شد)

خب مرحله 2 یعنی ثبت پاسخ سابجکت هم تمام شد(دیدین چقدر راحت بود!) پس در الگوریتم می توان این مرحله را در مراحل زیر توصیف کرد(از آنجایی که در الگوریتم این مراحل پس از نمایش محرک اتفاق می افتد شماره گذاری این مراحل نیز از 5 شروع می شود)

5. ذخیره تابع زمان برای لحظه شروع پاسخگویی سابجکت

6. صبر کردن تا جواب دادن سابجکت

7.به دست آوردن کد دکمه ای از کیبرد که سابجکت فشار داده

8. محاسبه اختلاف تابع زمان در لحظه فشار دادن دکمه کیبرد با تابع زمان شروع پاسخویی سابجکت و به دست آوردن زمان پاسخگویی

خب رسیدیم به مرحله سوم و آخرین مرحله طراحی الگوریتیم.

در این مرحله میخواهیم ببینیم پاسخ سابجکت و زمان پاسخگویی(Reaction Time) که در مرحله قبل محاسبه شد را برای هر ترایال ذخیره کنیم.

خب اگر یادتان باشد اول جزوه گفتیم از سابجکت خواسته می شود اگر جهت gabor متمایل به راست بود فلش راست کیبرد و اگر متمایل به چپ بود فلش سمت چپ کیبرد را فشار دهد. هر دکمه از کیبرد کدی دارد که به کمک متلب می توان کد آن دکمه را پیدا کرد همچنین کد دکمه ای که سابجکت فشار داد را نیز به کمک خروجی Kbwait میتوانیم پیدا کنیم بنابراین می توان به راحتی به کمک متلب گفت سابجکت کدام دکمه را فشار پس می توان گفت(برای اینکه با ذخیره کردن حروف در متلب درگیر نشویم از همان منطق 1 برای سمت راست و 2 برای سمت چپ استفاده می کنیم):

اگر (سابجکت فلش راست را روی کیبرد فشار داد)

در این صورت: جهت = 1

در غیر این صورت اگر (سابجکت فلش چپ را روی کیبرد فشار داد)

در این صورت: جهت = 2

پس از این مرحله نیاز داریم این اطلاعات پاسخ سابجکت و زمان پاسخگویی(Reaction Time) را برای هر آزمایش (ترایال) در جایی ذخیره کنیم و چه جایی بهتر از همان ماتریس شرایط که در اول داشتیم!!!

بنابراین 2 ستون جدید به ماتریس شرایط اضافه می کنیم. مقدار پاسخ سابجکت را در ستون اول و زمان پاسخگویی را در ستون دوم قرار می دهیم.(یعنی ستون چهارم و پنجم ماتریس شرایط در این تسک)

خب پس بیایید یک جمع بندی از الگوریتم بکنیم. الگوریتم تسک ما چنین شکلی دارد:

1. به دست آوردن زاویه و جهت gabor در آزمایش مورد نظر از روی ماتریس شرایط

2. نشان دادن gabor با شرایط به دست آمده بر روی صفحه نمایش

3. صبر کردن به مدت 500 میلی ثانیه

4. رفتن شکل از روی صفحه نمایش

5. ذخیره تابع زمان برای لحظه شروع پاسخگویی سابجکت

6. صبر کردن تا جواب دادن سابجکت

7. به دست آوردن کد دکمه ای از کیبرد که سابجکت فشار داده

8. محاسبه اختلاف تابع زمان در لحظه جدید با تابع زمان شروع پاسخگویی سابجکت و به دست آوردن زمان پاسخگویی(Reaction Time)

9 اگر (سابجکت فلش راست را روی کیبرد فشار داد)

در این صورت: جهت = 1

در غیر این صورت اگر (سابجکت فلش چپ را روی کیبرد فشار داد)

در این صورت: جهت = 2

10. قرار دادن جهت پاسخ و Reaction Time در ستون چهارم و پنجم ماتریس شرایط

خب اگر به این مراحل خوب نگاه کنید میبینید یک جای کار مشکل دارد. آن هم اینکه این که این الگوریتم در حال حاضر فقط برای یک ترایال اجرا می شود و باید به نوعی خودش را به تعداد ترایال ها تکرار کند. برای این کار نیازی به استفاده از مفهومی به عنوان حلقه معین دارید.

حلقه معین که در بیشتر زبان های برنامه نویسی با نام حلقه for شناخته می شود دستورات معینی را به تعداد بار گفته شده انجام می دهد و در حالت کلی به صورت زیر تعریف می شود:

برای i های از 1 تا n:

[دستور ها]

پایان

مقدار i که از آن به عنوان شمارنده حلقه یاد می شود به این صورت عمل می کند که در اولین باری که دستورات درون حلقه اجرا می شود مقدار i برابر 1 است در دومین بار برابر 2 و هر وقت مقدار i و n برابر شود به این معنی است که دستور های درون حلقه n بار تکرار شده اند و حلقه پایان می یابد به طور کلی هر گاه درون حلقه مقدار i برابر عددی مانند a شود بدین معنی است که دستورات درون حلقه a بار تکرار شده اند.

خب حال باید با این مفهوم برخی از مراحل الگوریتم بالا را شفاف سازی کنیم:

*در مرحله 1 برای به دست آوردن زاویه و جهت gabor باید به این شکل عمل کنیم:

زاویه gabor در هر آزمایش در خانه واقع در سطر مربوط به آن آزمایش و ستون دوم ماتریس شرایط قرار دارد. اگر شمارنده حلقه را i فرض کنیم سطر مربوط به هر آزمایش، سطر i ام تعریف می شود(به طور منطقی نیز فرض کنیم هر سطر یک ترایال را توصیف می کند بنابراین سطر i ام، آزمایش i ام را توصیف می کند) و به همین شکل خانه واقع در سطر i ام و ستون سوم جهت ممرک آن آزمایش را توصیف می کند.

*میتوانیم مرحله 9 و 10 را به طور کلی به این شکل خلاصه کنیم:

اگر (سابجکت فلش راست را روی کیبرد فشار داد)

در این صورت: سطر i ام و ستون چهارم ماتریس شرایط = 1

در غیر این صورت اگر (سابجکت فلش چپ را روی کیبرد فشار داد)

در این صورت: سطر i ام و ستون چهارم ماتریس شرایط = 2

سطر i ام و ستون پنجم ماتریس شرایط= Reaction Time

*توجه کنید برای ترایال هایی که سابجکت پاسخ اشتباه داده زمان پاسخگویی ذخیره نمیشود

* فقط یک نکته باقی میماند تا طراحی الگوریتم تمام شود و آن انتخاب n برای تعریف حلقه است یعنی تعداد کل ترایال های ما از کجا معلوم می شود. به طور کلی اندازه ماتریس شرایط بیانگر تعداد آزمایش های ماست. اندازه ماتریس شرایط نیز همان طور که گفته شد از ضرب تعداد حالات در تعداد بلوک ها به دست می آید. ولی از آنجایی که در حال حاضر محاسبه تعداد شرایط را از شما نمیخواهیم و ماتریس شرایط به شما داده می شود می توانید با استفاده از دستور length در متلب طول ماتریس را به دست آورده و به عنوان ممروده حلقه از آن استفاده کنید.

بنابراین نتیجه نهایی طراحی الگوریتم به شکل زیر تعریف می شود:

برای i های از 1 تا تعداد سطر های ماتریس شرایط

1. زاویه محرک = سطر i ام و ستون دوم ماتریس شرایط
2. جهت محرک = سطر i ام و ستون سوم ماتریس شرایط
3. نشان دادن Gabor با شرایط به دست آمده بر روی صفحه نمایش
4. صبر کردن به مدت 500 میلی ثانیه
5. رفتن شکل از روی صفحه نمایش
6. ذخیره تابع زمان برای لحظه شروع پاسخگویی سابجکت
7. صبر کردن تا جواب دادن سابجکت
8. به دست آوردن کد دکمه ای از کیبرد که سابجکت فشار داده
9. محاسبه اختلاف تابع زمان در لحظه جدید با تابع زمان شروع پاسخگویی سابجکت و به دست آوردن زمان پاسخگویی(Reaction Time)
10. اگر (سابجکت فلش راست را روی کیبرد فشار داد)

در این صورت: سطر i ام و ستون چهارم ماتریس شرایط = 1

در غیر این صورت اگر (سابجکت فلش چپ را روی کیبرد فشار داد)

در این صورت: سطر i ام و ستون چهارم ماتریس شرایط = 2

سطر i ام و ستون پنجم ماتریس شرایط= Reaction Time

پایان.

خب تبریک!!! الگوریتم تسک تمام شد. دیدین چقدر شیرین و ساده بود:)))))

و اما قسمت شیرین و جذاب برنامه نویسی:

خب با نوشتن الگوریتم تسک شما عملا 80 درصد کار پیاده سازی تسکی که دیزاین کردین رو انجام دادین ولی در جریانین که هنوز حتی یک خط کد هم نزدین:)

قبل از اینکه وارد مسئله syntax و کد زدن بشیم باید یک سری چیز های اولیه در مورد سایکتولباکس رو مرور کنیم.

سایکتولباکس یک کتابخانه قوی برای طراحی تسک های دیداری و شنیداری حساب میشود که بر پایه هسته قدرتمند Open-GL قرار دارد. اجرای دستورات سایکتولباکس به طور پیشفرض بر روی پردازنده گرافیکی کامپیوتر(GPU) طراحی شده زیرا محاسبات گرافیکی آن برای پردازنده مرکزی(CPU) بیش از حد سنگین است.

تمام دنیای سایکتولباکس به یک دستور و یک کلمه خلاصه می شود: Screen

*یاد آوری: متلب case-sensitive است یعنی به حروف بزرگ و کوچک حساس است. در استفاده از دستور ها و نام گزاری متغیر ها به این نکته توجه کنید.

تقریبا تمام کاربرد سایکتولباکس هنگامی است که نیاز است چیزی بر روی مانیتور نمایش داده شود. سایکتولباکس به کمک دستور OpenScreen پنجره ای را باز می کند که عملا پلی برای ورود به دنیای سایکتولباکس حساب می شود.

برای انجام کار هایی همانند باز کردن Screen، پیدا کردن نقطه وسط صفحه و .... نیاز به دستور هایی است که هر کدی برای هر تسکی بنویسید عملا باید همین دستورات را عینا تکرار کنید. در تمرینات شما نیز این کد های ثابت و پیشفرض به شما داده می شود و برای انجام تمرینات کارگاه نیازی به دانستن شیوه کار و منطق هر کدام از این دستورات ندارید. بعد ها در پروژه های سایکوفیزیک نیز این کد ها را عملا یک بار نوشته و برای هر پروژه دیگری که بخواهید می توانید کپی پیست کنید. زیرا بودن آنها در اول کد اجباری است و همواره نیز به یک صورت تعریف می شوند ولی منطق کار تمام این دستورات پیشفرض در پوشه رفرنس در فایل تمرینات آمده اند. مطالعه اسلاید های این پوشه کاملا اختیاری است ولی برای درک شیوه کار توابع سایکتولباکس به شدت توصیه می شود.

اما با این که نیازی به شیوه کار کد های پایه سایکتولباکس نیست اما باید با ساختار پایه نوشتن OpenWindow آشنا شوید.

```
[wPtr, screenrect] = Screen('OpenWindow',max(Screen('Screens')),[255 255 255] );
```

- wPtr: نشانگر تمام صفحه نمایش(در کشیدن هر شکلی با سایکتولباکس باید از این نشانگر استفاده کنید)
- screenrect: ماتریس مختصات صفحه کشیده شده(از مقدار این مختصات در پیدا کردن نقطه وسط صفحه استفاده می شود)
- OpenWindow: دستوری که به سایکتولباکس میگوید یک صفحه جدید باز کند
- max(Screen('Screens')): مقدار Screen Number را برمیگرداند و نشانگر کد مانیتور است. برای تسک هایی اهمیت دارد که در آن از چند مانیتور استفاده شده است
- [255 255 255]: رنگ

الگوریتم تعریف رنگ در متلب و سایکتولباکس از قاعده RGB پیروی می کند بدین معنی که برای تعریف رنگ صفحه باید یک بردار با 3 مقدار تعریف شود که مقدار اول غلظت رنگ قرمز(Red)، مقدار دوم غلظت رنگ سبز(Green) و مقدار سوم غلظت رنگ آبی(Blue) را برمیگرداند که هر کدام از این مقدار ها می تواند عددی بین 0 تا 255 باشد(نام RGB نیز از 3 حرف اول این رنگ ها می آید) به عنوان مثال [0 0 255] نمایانگر رنگ قرمز است

*[0 0 0] رنگ سیاه و [255 255 255] رنگ سفید را بیان می کنند(پس خودتان فکر کنید که رنگ خاکستری چیست؟ میتوانید حدس خود را در متلب امتحان کنید)

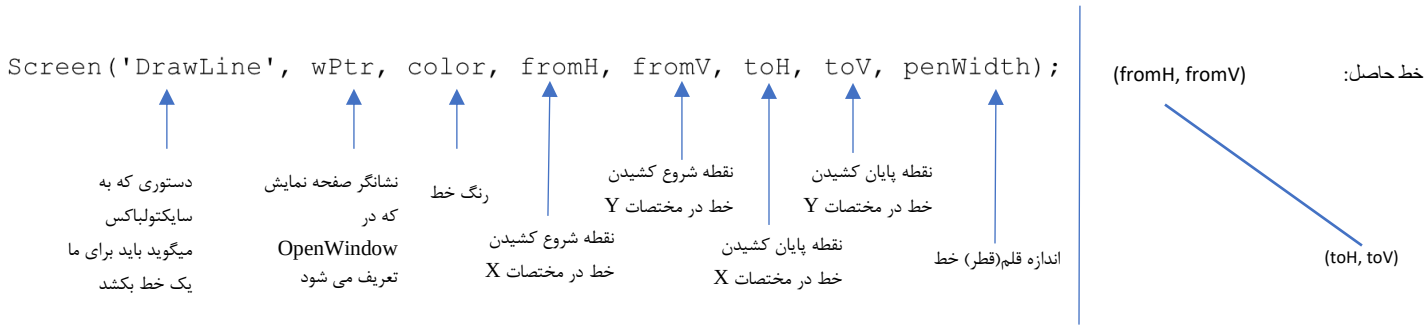
اما یک دستور خاص بین این دستورات است که نیاز است دلیل وجود و شیوه کار آن توضیح داده شود و آن دستوری نیست به جز flip

برای درک کامل دلیل استفاده از flip باید به ساختار فیزیکی مانیتور مراجعه کرد اما به طور کلی می توان گفت مانیتور های ما refresh rate معادل 60Hz دارند یعنی به طور متوسط هر 16ms یک بار تمام پیکسل های مانیتور فرمان جدید را دریافت می کنند و هر پیکسل رنگ جدیدی می گیرد.

این بدین معنی است که اگر فرمان نشان دادن چیزی به مانیتور داده شود و فرمان باقی ماندن آن به مانیتور داده نشود، آن شکل بعد از اولین refresh یعنی 16 میلی ثانیه از روی صفحه نمایش پاک می شود! بیشتر نرم افزار ها این قابلیت را دارند که تا هنگامی که کاربر خود دستور پاک شدن شکل از روی صفحه نمایش را ندهد آن را بر روی مانیتور نگه داشته و اجازه نمی دهند تصویر از روی مانیتور حذف شود اما سایکتولباکس به صورت پیشفرض برای کشیدن شکل یا نمایش عکس چنین قابلیتی ندارد و نیاز به دستوری است که شکل را بر روی صفحه نگه دارد تا شکل کشیده شده بعد از اولین refresh از روی صفحه نرود و تا فرمان کاربر بماند. این دستور در سایکتولباکس flip نام دارد و به کمک Screen('Flip',wPtr) تعریف می شود.

بنابراین استفاده از flip بعد از کشیدن هر شکلی به کمک سایکتولباکس الزامی است زیرا در غیر این صورت شکل کشیده می شود و بعد از 16ms پاک میشود که اغلب نیز تصور می شود مشکلی در کشیدن شکل پیش آمده و پیدا کردن مشکل واقعی و رفع آن زمان زیادی می برد.

یکی از دلایل استفاده از سایکتولباکس توانایی کشیدن شکل های ممرک های مختلف است. در نمونه کد زیر کد کشیدن یک خط در سایکتولباکس نشان داده می شود



کشیدن بقیه شکل ها در سایکتولباکس نیز نیز دقیقا همانند کشیدن خط است

در لیست زیر تعدادی از کاربردی ترین ساختار های کشیدن شکل در سایکتولباکس بیان شده است:

```
Screen('DrawLine', wPtr [,color], fromH, fromV, toH, toV [,penWidth]);
Screen('DrawArc',wPtr,[color],[rect],startAngle,arcAngle)
Screen('FrameArc',wPtr,[color],[rect],startAngle,arcAngle[,penWidth] [,penHeight] [,penMode])
Screen('FillArc',wPtr,[color],[rect],startAngle,arcAngle)
Screen('FillRect', wPtr [,color] [,rect] );
Screen('FrameRect', wPtr [,color] [,rect] [,penWidth]);
Screen('FillOval', wPtr [,color] [,rect] [,perfectUpToMaxDiameter]);
Screen('FrameOval', wPtr [,color] [,rect] [,penWidth] [,penHeight] [,penMode]);
Screen('FramePoly', wPtr [,color], pointList [,penWidth]);
Screen('FillPoly', wPtr [,color], pointList [, isConvex]);
Screen('DrawDots', wPtr, xy [,size] [,color] [,center] [,dot_type]);
Screen('DrawLines', wPtr, xy [,width] [,colors] [,center] [,smooth]);
```

توجه کنید مقادیری که درون ' ' هستند باید عینا همان چیزی که اینجا نوشته شده(با رعایت حروف بزرگ و کوچک) نوشته شود و بقیه متغیر ها باید مقدار بگیرند یعنی مثلا برای رنگ مقداری در سیستم RGB یا برای مختصات عددی در مختصات صفحه نمایش ارائه شود

خب دستور دیگری که باید با آن آشنا شوید دستور imread است که عکس را دریافت می کند و در سایکتولباکس نشان می دهد. به عنوان مثال برای تسکی که طراحی کردیم کشیدن Gabor با سایکتولباکس ممکن است ولی کار راحتی نیست. فرض کنید شما عکس Gabor را در اینترنت پیدا کرده اید و میخواهد به جای کشیدن Gabor از عکس آن استفاده کنید بنابراین نیاز به وارد کردن عکس به محیط متلب و استفاده از آن در سایکتولباکس است. به این شکل:

```
faceData = imread('sadface.jpg');------>jpeg خواندن یک عکس فرض با فرمت
faceTexture = Screen('MakeTexture',wPtr,faceData);------>تبدیل عکس خوانده شده به فرمت قابل نمایش در سایکتولباکس
Screen('DrawTexture',wPtr,faceTexture);------>کشیدن عکس در سایکتولباکس
Screen('Flip',wPtr);------>فلیپ معروف(!) برای باقی ماندن عکس بر روی صفحه
```

خب دیگر تقریبا تمام دستوراتی که ساختار آن ها نیاز به توضیح داشت را یاد گرفته اید. حال بیایید کد تسک Gabor خودمان را پیاده کنیم

*فرض میکنیم تمام gabor ها را به صورت عکس ورودی گرفته ایم و texture آن ها درون 2 آرایه به نام های rightGabor (gabor های متمایل به راست) وleftGabor (gabor های متمایل به چپ) قرار داده ایم که هر یک از این آرایه ها 8 خانه دارد و شامل gabor ها با زاویه خاص است یعنی در خانه اول gabor با زاویه 10 درجه، در خانه دوم gabor با زاویه 20 درجه و ... بنابراین میتوان نتیجه گرفت ممرک با زاویه R درجه در خانه R/10ام آرایه قرار دارد.

دستور آخری هم که باید با آن آشنا شوید دستور WaitSecs است که به میزان مشخصی صبر می کند و سپس کد را ادامه می دهد به عنوان مثال

WaitSecs(0.5) نیم ثانیه معادل 500 میلی ثانیه صبر کرده و سپس کد خط بعد را اجرا می کند

خب هر چیزی که برای دیزاین یک تسک سایکوفیزیک نیاز بود را یاد گرفته اید. الان می توانید با کمی سرچ در گوگل خودتان کد تسک Gabor را به راحتی طراحی کنید ولی ما فایل کامل کد تسک را در لینک زیر به همراه توضیحات فینگیلیش(از آنجایی که کامنت گذاشتن در کد به فارسی سخت است و خواندن کامنت انگلیسی برای هر خط ممکن است برای شما سخت باشد) برایتان قرار داده ایم. ولی پیشنهاد می شود حتما سعی کنید در کنار دیدن آن کد، خودتان نیز کد را دوباره در متلب بنویسید تا در کار با آن روان شوید

لینک کد تسک Gabor:

https://github.com/CNCH-Psychophysics/CNCH-Code

راه های ارتباط:

خب خیلی ممنون که تو این جزوه همراه ما بودین. هر سوالی مرتبط با جزوه و کلا سایکوفیزیک داشته باشین میتونین از طریق ایمیل های زیر با ما در ارتباط باشین.

tara.ghafari@gmail.com (پاسخی برای تمام سوالات شما؛)

soroshsamiei.2000@gmail.com(سایکوفیزیک)

aryan@zoroufi.com (سایکتولباکس و برنامه نویسی)