در این بخش چند تابع پیاده سازی شده که ابتدا توضیحشان میدهیم.

تابع `rotate` :

ورودی ها:
1) یک عکس چهره
2) مختصات چشم سمت چپ در تصویر
3) مختصات چشم سمت راست در تصویر

خروجی ها:
1) یک عکس چهره به صورت صاف پس از دوران مناسب
2) مختصات چشم سمت چپ در تصویر بعد از دوران
3) مختصات چشم سمت راست در تصویر بعد از دوران

عملکرد: هدف این است که چهره را به اندازه‌ای بچرخانیم که چهره کج نباشد. این کار معادل با این است که خط واصل بین دو چشم با محور افقی زاویه‌ی صفر درجه بسازد، آنگاه میتوان گفت چهره صاف است. همچنین علاوه بر این که عکس را دوران میدهیم باید مختصات چشم‌ها بعد از دوران را بدست آوریم. پس ابتدا زاویه‌ی بین خط واصل بین دو چشم را با افق بدست میاوریم که برابر با زاویه‌ی زیر است.

$$\tan\theta = \frac{y_R - y_L}{x_R - x_L} \Rightarrow \theta = \tan^{-1}\frac{y_R - y_L}{x_R - x_L}$$

که صورت کسر برابر با تفاضل مختصات دو چشم در راستای عمودی و مخرج کسر برابر با تفاضل مختصات دو چشم در راستای افقی است ابتدا مختصات چشم ها را تحت زاویه‌ی θ نسبت به مرکز عکس دوران میدهیم. این کار را با استفاده از فرمول زیر انجام میدهیم.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

اکنون برای آنکه ماتریس دوران در جهت ساعتگرد تحت زاویه‌ی θ نسبت به مرکز عکس بدست آید، ابتدا θ را برحسب درجه بدست آورده و از تابع آماده‌ی `cv2.getRotationMatrix2D` استفاده میکنیم و سپس با تابع آماده‌ی `cv2.warpAffine` این دوران را برای کل تصویر انجام میدهیم. تصویر حاصل و مختصات چشم چپ و راست را پس از دوران بعنوان خروجی بازگشت میدهیم.

تابع `scale` :

ورودی ها:
1) دو عکس چهره با مقیاس و ابعاد متفاوت
2) مختصات چشم سمت چپ در هر تصویر
3) مختصات چشم سمت راست در هر تصویر

خروجی ها:
1) دو عکس چهره که هم مقیاس شده اند
2) مختصات چشم سمت چپ در تصویر بعد از هم مقیاس شدن
3) مختصات چشم سمت راست در تصویر بعد از هم مقیاس شدن

عملکرد: هدف این است که چهره های داده شده هم اندازه شوند. این کار معادل با این است که فاصله‌ی بین دو چشم در هر دو تصویر برابر شوند. برای این کار فاصله‌ی بین دو چشم در هر دو عکس داده شده را محاسبه میکنیم. عکسی که دارای فاصله‌ی بین چشم بیشتری بود را بعنوان عکس بزرگ در `imgLarge` نگه میداریم و عکسی که دارای فاصله‌ی بین چشم کمتری بود را عکس بعنوان عکس کوچک در `imgSmall` نگه میداریم. قصد داریم عکس بزرگ تر را تغییر سایز دهیم به گونه‌ای که فاصله‌ی بین چشم ها در عکس بزرگ و عکس کوچک برابر شود. پس نسبت فاصله‌ی چشم ها در عکس بزرگ به فاصله بین چشم ها در عکس کوچک را در `rr` نگه میداریم. این نسبت نشان میدهد که ابعاد عکس بزرگ `rr` برابر ابعاد عکس کوچک است. پس برای هم مقیاس کردن، ابعاد عکس بزرگ را با تابع آماده‌ی `cv2.resize` به نسبت `1/rr` تغییر میدهیم اکنون فاصله‌ی بین چشم ها در هر دو تصویر یکسان شده اند. همچنین علاوه بر این که عکس را تغییر مقیاس میدهیم باید مختصات چشم‌ها بعد از هم مقیاس کردن را نیز بدست آوریم که به وضوح مولفه های مختصات هر دو چشم `1/rr` میشود. اکنون ابعاد عکس کوچک بدون تغییر باقی مانده و ابعاد عکس بزرگ `1/rr` برابر شده. برای آنکه عکس ها و مختصات های جدید چشم ها به همان ترتیبی که ورودی گرفتیم، بعنوان خروجی بازگشت دهیم، از متغیری بنام `firstImageIsLarger` استفاده میکنیم. اگر عکس اول بزرگتر بود این متغیر مقدار `True` و اگر عکس دوم بزرگتر بود مقدار `False` را نگه میدارد. در نهایت اگر عکس اول بزرگتر بود ابتدا اطلاعات عکس بزرگ و سپس عکس کوچک را بعنوان خروجی بازگشت میدهیم و اگر عکس دوم بزرگتر بود ابتدا اطلاعات عکس کوچک و سپس عکس بزرگ را بعنوان خروجی بازگشت میدهیم.

تابع `match` :

ورودی ها:
1) دو عکس چهره
2) مختصات چشم سمت چپ در هر تصویر
3) مختصات چشم سمت راست در هر تصویر

خروجی ها:
1) دو عکس چهره که چشم چپ آنها روی هم منطبق است
2) مختصات چشم سمت چپ در تصویر بعد از انتقال
3) مختصات چشم سمت راست در تصویر بعد از انتقال

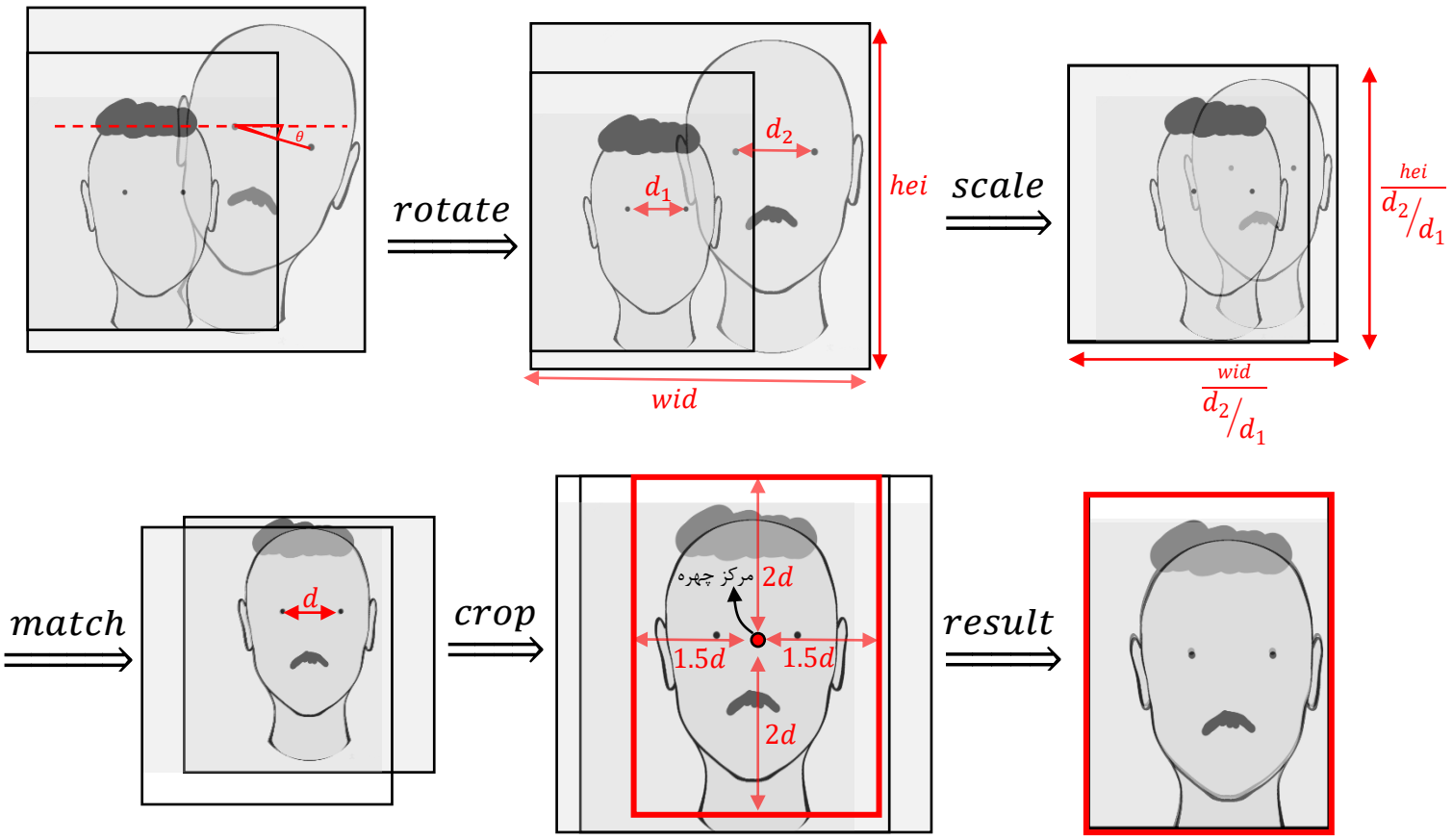
عملکرد: هدف این است که چشم چپ چهره های داده شده روی هم قرار گیرند. برای این کار تفاضل مولفه های چشم چپ در دوم با اول را محاسبه کرده و یک ماتریس انتقال میسازیم که فرم آن به صورت $\begin{bmatrix}1 & 0 & dx\\ 0 & 1 & dy\end{bmatrix}$ است. که به اندازه‌ی dx در راستای افقی و به اندازه‌ی dy در راستای عمودی، انتقال را انجام میدهد. حال عکس دوم را تحت این نگاشت با تابع آماده‌ی cv2.warpAffine انتقال میدهیم. همچنین علاوه بر این که عکس را انتقال میدهیم باید مختصات چشم‌ها را بعد از انتقال بدست آوریم که به وضوح مولفه‌ی افقی و عمودی آن به ترتیب با dx و dy جمع میشود. این کار باید برای هر دو چشم تصویر انتقال یافته انجام شود. اکنون هر دو عکس و مختصات چشم هایشان را بعنوان خروجی بازگشت میدهیم.

تابع getFaces :

ورودی ها:
1) دو عکس چهره
2) مختصات چشم سمت چپ در هردو تصویر
3) مختصات چشم سمت راست در هردو تصویر

خروجی ها:
1) دو عکس چهره که بر هم منطبق هستند

عملکرد: این تابع ابتدا هر دو عکس را مستقلا حول مرکزشان با تابع rotate که بالاتر تعریف شد دوران میدهد تا چهره ها صاف و مستقیم شوند. سپس دو عکس حاصل را با استفاده از تابع scale که بالاتر تعریف شد هم مقیاس میکند. اکنون دو عکس حاصل هم مستقیم و مقیاس اند و فاصله‌ی بین چشم ها در هر دو تصویر یکسان است. پس توقع داریم اگر چشم چپ از چهره ای بر روی چشم چپ چهره‌ی دیگری منطبق بود، آنگاه چشم راست هر دو چهره نیز بر هم منطبق شوند و به همین ترتیب از آنجا که کل اجزای صورت انسان ها تقریبا مقیاس مشابه دارند، تمام دو چهره برهم منطبق میباشد. پس با استفاده از تابع match که بالاتر تعریف شد یکی از عکس ها را انتقال میدهیم تا بر عکس دیگر منطبق شود. اکنون با توجه به ابعاد صورت تصمیم داریم چهره ها را برش دهیم. برای این کار نقطه‌ی میان دو چشم را بعنوان مرکز چهره ها در نظر میگیریم سپس به اندازه‌ی 1.5 برابر فاصله‌ی بین چشم ها از سمت راست و چپ مرکز چهره را حفظ کرده و باقی را حذف میکنیم و به اندازه‌ی 2 برابر فاصله‌ی بین چشم ها از بالا و پایین مرکز چهره را حفظ کرده و باقی را حذف میکنیم. تمام مراحل انجام شده به بیان تصویری در زیر آمده است.



اکنون به توضیح توابعی که برای هیبریدی کردن عکس ها است میپردازیم.

تابع `dftFilter` :

ورودی ها: 1) یک عکس که در حوزه‌ی مکان است

خروجی ها: 1) ماتریس حوزه‌ی فرکانس برای عکس داده شده

عملکرد: ابتدا یک ماتریس با ابعاد عکس داده شده میسازیم که درایه های آن عدد مختلط باشند، سپس هر چنل عکس داده شده را با استفاده از تابع آماده‌ی `fft2` به حوزه‌ی فرکانس میبریم و با تابع آماده‌ی `fftshift` شیفت میدهیم تا ضرایب مطلوب بدست آیند و آن را بعنوان خروجی بازگشت میدهیم.

تابع `idftFilter` :

ورودی ها: 1) یک ماتریس که در حوزه‌ی فرکانس است

خروجی ها: 1) عکس حوزه‌ی مکان برای ماتریس داده شده

عملکرد: ابتدا یک عکس با ابعاد ماتریس داده شده میسازیم، سپس هر چنل ماتریس داده شده را با استفاده از تابع آماده‌ی `ifftshift` به صورت معکوس شیفت میدهیم و با تابع آماده‌ی `ifft2` آن را به حوزه‌ی مکان میبریم و آن را بعنوان خروجی بازگشت میدهیم.

تابع `gaussFilter` :

ورودی ها:
1) یک عکس که به ابعاد آن یک ماتریس گوسی میسازیم
2) یک عدد که برابر با انحراف معیار ماتریس گوسی است
3) نوع فیلتر گوسی که یا highpass و یا lowpass خواهد بود

خروجی ها:
1) ماتریسی که درایه های آن از توزیع گوسی دو بعدی اند

عملکرد: ابتدا یک ماتریس با ابعاد عکس داده شده میسازیم، سپس مرکز این ماتریس را تعیین میکنیم و در هر چنل با استفاده از فرمول زیر داریه های ماتریس را مقدار دهی میکنیم.

$$gaussMatrix[i,j,:] = \begin{cases} 1 - e^{-\frac{(i-i_0)^2+(j-j_0)^2}{2\sigma^2}} & highpass \\ e^{-\frac{(i-i_0)^2+(j-j_0)^2}{2\sigma^2}} & lowpass \end{cases}$$

i_0, j_0: مرکز ماتریس

سپس ماتریس حاصل را بعنوان خروجی بازگشت میدهیم.

تابع `cutoffFilter` :

ورودی ها:
1) یک عکس که به ابعاد آن یک ماتریس برشی میسازیم
2) یک عدد که برابر با آستانه‌ی برش دادن است
3) نوع فیلتر برش که یا highpass و یا lowpass خواهد بود

خروجی ها:
1) ماتریسی که درایه های آن 0 یا 1 اند

عملکرد: ابتدا یک ماتریس با ابعاد عکس داده شده میسازیم، سپس مرکز این ماتریس را تعیین میکنیم و در هر چنل با استفاده از فرمول زیر داریه های ماتریس را مقدار دهی میکنیم.

$$cutoffMatrix[i,j,:] = \begin{cases} \begin{cases} 1 & dist > threshhold \\ 0 & else \end{cases} & highpass \\ \begin{cases} 1 & dist < threshhold \\ 0 & else \end{cases} & lowpass \end{cases}$$

سپس ماتریس حاصل را بعنوان خروجی بازگشت میدهیم

تابع normalize :

ورودی ها: 1) یک ماتریس با درایه های حقیقی

خروجی ها: 1) یک ماتریس که تمام درایه های آن در بازه‌ی 0 تا 255 اند

عملکرد: این تابع برای خلاصه تر شدن کد اصلی است و تنها از تابع آماده‌ی cv2.normalize استفاده میکند و درایه های ماتریس داده شده را به بازه‌ی 0 تا 255 نگاشت میدهد.

حال پس از توضیحات مربوط به توابع پیاده سازی شده به توضیحات روند استفاده از آنها و رسیدن به خروجی مطلوب میپردازیم.

ابتدا عکسی که میخواهیم از نزدیک دیده شود را در I1 و عکسی که میخواهیم از دور دیده شود را در I2 نگه میداریم. همچنین مختصات چشم ها در هر دو تصویر را به صورت دستی و تجربی نگه میداریم. در ادامه تمام متغیر های همنامی که با شماره های 1 و 2 تفکیک شده اند به ترتیب مربوط به عکس های I1 و I2 اند. برای مثال eyeL1 و eyeR1 مختصات چشم های درون تصویر I1 و بطور مشابه eyeL2 و eyeR2 مختصات چشم های درون تصویر I2 میباشد. در ابتدا با استفاده از تابع getFaces که بالاتر تعریف شد، چهره های درون عکس I1 و I2 را بدست میاوریم تا دو چهره‌ی برهم منطبق داشته باشیم. سپس با استفاده از تابع dftFilter عکس ها را به حوزه‌ی فرکانس میبریم. سپس با استفاده از تابع gaussFilter که بالاتر تعریف شد، دو فیلتر گوسی با انحراف معیار های r=20 از نوع highpass و s=10 از نوع lowpass میسازیم و به ترتیب آنها را gauss1 و gauss2 مینامیم. سپس با استفاده از تابع cutoffFilter که بالاتر تعریف شد، دو فیلتر برشی با آستانه های 10 از نوع highpass و 20 از نوع lowpass میسازیم و به ترتیب آنها را cutoff1 و cutoff2 مینامیم. این دو فیلتر برشی به صورت ماسک عمل میکنند پس آنها را در فیلترهای گوسی متناظرشان ضرب درایه به درایه میکنیم، تا فیلتر گوسی برش یافته به ما داده شود و آنها را به ترتیب filter1 و filter2 مینامیم. اکنون هر فیلتر را در تصویر متناظرش ضرب درایه به درایه میکنیم و آنها را به ترتیب dftFiltered1 و dftFiltered2 مینامیم. اکنون باید دو تصویر حاصل را ترکیب کنیم. برای این کار ابتدا یک ماتریس بنام hybridFrequency با درایه های مختلط ساخته و سپس به درایه هایی که ماتریس dftFiltered2 در آن درایه ها برابر صفر باشد، مقدار ماتریس dftFiltered1

را داده و به درایه هایی که ماتریس dftFiltered1 در آن درایه ها برابر صفر باشد، مقدار ماتریس dftFiltered2 را میدهیم. همچنین به درایه هایی که هر دو ماتریس dftFiltered1 و dftFiltered2 در آن درایه ها ناصفر اند (نوار اشتراک)، میانگین وزن دار دو ماتریس با وزن های 1/2 و 1/2 را میدهیم. اکنون عکس هیبریدی ما در حوزه‌ی فرکانس آماده است پس با تابع idftFilter که بالاتر تعریف شد آن را به حوزه مکان برده و یک نسخه‌ی بزرگ ذخیره میکنیم سپس برای عدم کاهش کیفیت، ابتدا عکس را بلور کرده و سپس سایزش را 1/4 کرده و یک نسخه‌ی کوچک ذخیره میکنیم.