
بسم الله الرحمن الرحیم

دانشگاه الزهراء(س)

دانشکده فنی و مهندسی-گروه کامپیوتر

پایان نامه مقطع کارشناسی رشته مهندسی کامپیوتر گرایش نرم‌افزار

ردگیری بی‌درنگ چشم مبتنی بر روش تطبیق الگو

گردآوردنده:

سیده مهسا موسوی

شماره دانشجویی

استاد راهنما:

دکتر رضا عزمی

بهمن ۱۳۹۲

تقديم

چکیده

ردگیری موقعیت چشم در کاربردهای بسیاری از پردازش تصویر همانند تطبیق چهره، ردگیری مسیر دید و تشخیص هویت با شناسایی عنبیه استفاده میگردد. علاوه بر این با تشخیص موقعیت چشم، امکان پیاده سازی روشهای ارتباط مستقیم انسان با ماشین همانند کنترل دستگاه با چشم فراهم میگردد، که خود استفاده زیادی برای ارتباط ماشین با افراد معلول دارد.

برای ردگیری چشم در این پروژه از روشی مبتنی بر تطبیق الگو استفاده میگردد. در تطبیق الگو، کل تصویر توسط یک الگو (که در اینجا چشم شخص است) جاروب شده و بر اساس معیارهای شباهت متفاوت، شبیه‌ترین بخش تصویر به الگوی مورد نظر پیدا میگردد. الگوی مورد نظر مختص به کاربر نیز با به کار بردن کلاسیفایر هار[1] از چند فریم اول تصویر استخراج میگردد .

از آنجا که به کار بردن کلاسیفایر هار عملی زمان‌بر است، این کلاسیفایر فقط در چند فریم اول استفاده شده و پس از پیدا شدن الگو، روش تطبیق الگو استفاده میگردد که به شدت دقیق و سریع است. ترکیب کردن استفاده از کلاسیفایر هار که یک ماجول سطح بالا است با روش تطبیق الگو منجر به دستیابی همزمان به دقت و سرعت در این پروژه میشود. به همین دلیل سیستم طراحی شده قادر است حتی تصاویر با کیفیت پایین همانند تصاویر ضبط شده با وبکم را با دقت مناسب و با سرعت بسیار بالا و به صورت بی درنگ[2] پردازش نماید. الگوریتم این پروژه در زبان C++ پیاده میشود.

کلید واژه ها: ردگیری چشم، تطبیق الگو، کلاسیفایر هار، بی درنگ، کتابخانه OpenCV، Eye Tracking، Pattern Matching، Template Matching، Real time

[1] -Haar Classifier

[2] -Real Time

فهرست مطالب

فهرست اشکال

فصل اول

مقدمه

۱-۱- پردازش تصویر و کاربردهای آن

پردازش تصاویر[۳] امروزه بیشتر به موضوع پردازش تصویر دیجیتال گفته می‌شود که شاخه‌ای از دانش رایانه است که با پردازش سیگنال دیجیتال که نماینده تصاویر برداشته شده با دوربین دیجیتال یا پویش شده توسط پویشگر هستند سر و کار دارد.

پردازش تصاویر دارای دو شاخه عمده ی بهبود تصاویر و بینایی ماشین است. بهبود تصاویر دربرگیرنده ی روشهایی چون استفاده از فیلتر محوکننده و افزایش تضاد برای بهتر کردن کیفیت دیداری تصاویر و اطمینان از نمایش درست آنها در محیط مقصد(مانند چاپگر یا نمایشگر رایانه)است، در حالی که بینایی ماشین به روشهایی می‌پردازد که به کمک آنها می‌توان معنی و محتوای تصاویر را درک کرد تا از آنها در کارهایی چون رباتیک و محور تصاویر استفاده شود.

در معنای خاص آن پردازش تصویر عبارتست از هر نوع پردازش سیگنال که ورودی یک تصویر است مثل عکس یا صحنه‌ای از یک فیلم.خروجی پردازشگر تصویر میتواند یک تصویر یا یک مجموعه از نشانهای ویژه یا متغیرهای مربوط به تصویر باشد.اغلب تکنیک‌های پردازش تصویر شامل برخورد با تصویر به عنوان یک سیگنال دو بعدی و بکاربستن تکنیک‌های استاندارد پردازش سیگنال روی آنها میشود. پردازش تصویر اغلب به پردازش دیجیتالی تصویر اشاره میکند ولی پردازش نوری و آنالوگ تصویر هم وجود دارند.این مقاله در مورد تکنیک‌های کلی است که برای همه آنها به کار میرود. [1]

[۳] -Image processing

شکل ۱-۱: بکارگیری پردازش تصویر در جایگزین کردن مشخصات عنبیه چشم بجای اثر انگشت

عملیات اصلی در پردازش تصویر:

۱) تبدیلات هندسی : همانند تغییر اندازه، چرخش و...
۲) رنگ : همانند تغییر روشنایی، وضوح و یا تغییر فضای رنگ
۳) ترکیب تصاویر : ترکیب دو و یا چند تصویر
۴) فشرده سازی تصویر : کاهش حجم تصویر
۵) قطعه بندی تصویر : تجزیه‌ی تصویر به قطعات با معنی
۶) تفاوت تصاویر : به دست آوردن تفاوت‌های تصویر
۷) میانگین گیری : به دست آوردن تصویر میانگین از دو تصویر

کاربردهای پردازش تصویر :

زمینه های مختلف کاربرد پردازش تصویر عبارتند از صنعت، هواشناسی، شهرسازی، کشاورزی، پزشکی، فناوری های علمی که در ادامه درباره‌ی هرکدام مختصرا بحث شده است.

صنعت :

امروزه کمتر کارخانه‌ی پیشرفته‌ای وجود دارد که بخشی از خط تولید آن توسط برنامه‌های هوشمند بینایی ماشین کنترل نشود. خطای بسیار کم، سرعت زیاد، هزینه‌ی نگهداری بسیار پایین، عدم نیاز به حضور اپراتور ۲۴ ساعته باعث شده که صنایع و کارخانه‌ها به سرعت به سمت پردازش تصویر و بینایی ماشین روی بیاورند.

هواشناسی :

از آنجا که در علم هواشناسی تشخیص و پیش‌بینی آب و هوا اکثرا از طریق تصاویر هوایی و ماهواره ای انجام میگیرد، پردازش تصویر در این علم کاربرد فراوانی دارد و دقت و سرعت پیش بینی آب و هوا و طوفان‌ها را بسیار بالا می‌برد. جبه‌های پرفشار، کم فشار، گردبادها و گرداب‌های به وجود آمده در سطح کره‌ی زمین را می‌توان مشاهده کرد.

شهرسازی :

با مقایسه‌ی عکس‌های مختلف از سال‌های مختلف یک شهر می‌توان میزان گسترش و پیشرفت آن را مشاهده کرد.

کاربرد دیگر پردازش تصویر می‌تواند در کنترل ترافیک باشد. با گرفتن عکس‌های هوایی از زمین ترافیم هر قسمت از شهر مشخص می‌شود.

قبل از ساختن یک شهر می‌توان آن را توسط کامپیوتر شبیه سازی کرد که به صورت دو بعدی از بالا و حتی به صورت سه بعدی از دیدهای مختلف، یک شهرک چطور ممکن است به نظر برسد. تصاویر ماهواره ای که از شهرها گرفته می‌شود، می‌تواند توسط فیلترهای مختلف پردازش تصویر فیلتر شود و اطلاعات مختلفی از آن استخراج شود. به طور مثال اینکه شهر در چه قسمت‌هایی دارای ساختمان‌ها، آب‌ها یا

راه‌های بیش‌تری است و همینطور می‌توان جاده‌هایی که داخل یا خارج از شهر کشیده شده‌اند را تحلیل کرد.

کشاورزی:

این علم در بخش کشاورزی معمولا در دو حالت کاربرد دارد. یکی در پردازش تصاویر گرفته شده از ارتفاعات بالا مثلا از هواپیما و دیگری در پردازش تصویر نزدیک به زمین.

در تصاویر دور به عنوان مثال می‌توان تقسیم بندی اراضی را تحلیل کرد. همچنین می‌توان با مقایسه‌ی تصاویر دریافتی در زمان‌های متفاوت میزان صدمات متوالی وارد به محیط زیست را دید. به عنوان مثال می‌توان برنامه‌ای نوشت که با توجه به محل رودخانه‌ها و نوع خاک مناطق مختلف، به صورت اتوماتیک بهترین نقاط برای کشت محصولات مختلف را تعیین می‌کند.

تصاویر نزدیک هم در ساخت ماشین‌های هرزچین اتوماتیک کاربرد دارد. امروزه ماشین‌های بسیار گران‌قیمت کشاورزی وجود دارند که می‌توانند علف‌های هرز را از گیاهان تشخیص بدهند و به صورت خودکار آن‌ها را نابود کنند.

برای مثال یکی از پروژه های جالب در بخش کشاورزی، تشخیص خودکار گل زعفران برای جداسازی پرچم قرمز رنگ آن بوده است. این پردازش توسط نرم افزار ©Stigma detection انجام گرفته است.

پزشکی :

یکی از مهم‌ترین کاربردهای پردازش تصویر در علم پزشکی است. در جایی که ما نیاز داریم تمام عکس‌ها با نهایت شفافیت و وضوح گرفته شوند زیرا دیدن تمام جزییات لازم است. جراحی‌های ریز[۴] با ایجاد یک

[۴] -Micro Surgery

سوراخ کوچک و فقط دیدن محل جراحی توسط پزشک، از راه دور و توسط بازوهای روباتیک بسیار دقیق انجام می‌شود.

فناوری‌های علمی :

پردازش تصویر در افزایش سرعت پیشرفت‌های علمی تاثیر فوق‌العاده داشته است. اولین و مشخص‌ترین تاثیر آن را می‌توان در علم عکاسی هنر دید. شکار لحظه‌های شگفت‌آوری که در کثری از ثانیه اتفاق می‌افتد، بالا بردن وضوح عکس‌های گرفته شده و ایجاد افکت‌های خیره کننده از دستاوردهای پردازش تصویر است.

این علم در پیشرفت علوم پایه فیزیک، شیمی و مخصوصا تحقیقات فیزیکی و مکانیکی، کمک فراوانی کرده است. به عنوان مثال وظیفه‌ای برای حمل و نقل کالاها در مسی مسیرهای صعب العبور ساخته شده است. قبل از ساخت آن، رفتار چهارپایان در حالت‌های مختلف توسط کامپیوتر تحلیل و عینا به دستگاه آموزش داده شده است. در کل پردازش تصاویر به علت سرعت زیاد آن در ساخت وسایل مکانیکی پر سرعت، کاربرد زیادی دارد. وسیله‌ای وجود دارد که قادر است، توپی که با سرعت بسیار زیاد به سمت پایین می‌آید را مهار کند.

۱-۲- ردگیری چشم[۵] و کاربردهای آن

به ساده ترین بیان، رد گیری چشم به معنی دنبال کردن حرکت چشم انسان است. در چه زمانی نگاه می کنیم؟ به کجا نگاه می کنیم؟ در چه زمان چشم خود را باز و بسته می کنیم و چه مکانی از وب سایت بیشتر توجه ما را به خود جلب می کند؟ با وجود اینکه موضوع دنبال کردن چشم انسان ساده به نظر می رسد اما پیاده سازی آن کار دشواری است.

۵- Eye Tracking

شکل ۲-۲: ردگیری چشم در سیستمهای تشخیص مسیر دید

با ظهور سیستم‌های کوچک، بسیار دقیق و کم هزینه، امروزه ردیابی چشم به سرعت در دستگاه ها و برنامه های کاربردی که هر دو به منظور افزایش تعامل با کامپیوتر و درک رفتار انسان به وجود آمده اند پذیرفته شده است.

کاربردها و مزایای ردگیری چشم :

در دنیای پیشرفته‌ی امروز و با یشرفت روز افزون صنعت و تکنولوژی ردگیری چشم در زمینه‌های متعدد دارای کاربردهای گوناگون می‌باشد.

به طور کلی می‌توان کاربردهای ردگیری چشم را در دو گروه عمده دسته بندی نمود :

- درک و دریافت رفتارهای انسان
- بهبود و ارتقا واسطهای کاربری

رد گیری چشم به منظور درک و دریافت رفتارهای انسان :

در طول قرون اخیر، ردگیری چشم به عنوان ابزاری دقیق و قدرتمند جهت درک و دریافت رفتار انسان شناخته شده است. مغز انسان به صورت خودکار چشم را به سوی اطلاعاتی که در حال پردازش آن است

هدایت می‌کند، بنابراین با مشاهده‌ی آنچه که فرد در حال نظاره‌ی آن است می‌توان به راحتی به اطلاعاتی که مغز در حال پردازش آن‌ها است، دست یافت. شرکت‌های پیشرو در کالاهای مصرفی از ردیابی چشم به منظور بهینه سازی بسته بندی محصول و طراحی قفسه های خرده فروشی استفاده می‌کنند. شرکت‌های تحقیقات بازار و تبلیغات عمده از آن جهت بهینه سازی چاپ و تبلیغات تلویزیون استفاده می‌کنند. شرکت‌های وب از ردگیری چشم جهت بهینه سازی تجربه کاربر آنلاین استفاده کرده و در دانشگاه‌ها نیز برای تحقیق در روان شناسی، عصب شناسی و پزشکی بسیار کاربردی می‌باشد.

شرکت‌های وب از ردگیری چشم جهت بهینه سازی تجربه کاربر آنلاین استفاده کرده و در دانشگاه‌ها نیز برای تحقیق در روان شناسی، عصب شناسی و پزشکی بسیار کاربردی می‌باشد.

انواعی جدید از تشخیص پزشکی نیز ممکن است با ردیابی چشم انجام گیرد.

رد گیری چشم به منظور بهبود و ارتقا واسطهای کاربری :

نگاه کردن روش بسیار کارآمدی جهت اشاره کردن را فراهم می‌آورد، روشی که تمامی انسان‌ها از آن برای تعامل با یکدیگر استفاده می‌کنند. فناوری رد یابی چشم ما را قادر به استفاده از نگاه کردن و اشاره کردن جهت تعامل با کامپیوتر و ماشین آلات ساخته است. این روش، روشی سریع، شهودی و طبیعی می‌باشد.

لازم به ذکر است که در تمامی کاربردها ی مذکور پیدا کردن مکان چشم در صورت بسیار مهم بوده و در اولویت قرار دارد. علاوه بر این رد گیری چشم در سیستم‌های امروزه به صورت بلادرنگ انجام می‌گردد.

در ادامه به بررسی این مطلب و پیاده سازی ان خواهیم پرداخت.

فصل دوم

کلیات سیستم ارائه شده

۲-۱- هدف سیستم

در این پروژه هدف پیاده سازی یک سیستم ردگیری چشم انسان است که قادر به عملکرد بلادرنگ باشد. لزوم بلادرنگ بودن سیستم در عنوان پروژه محدودیتهای زیادی برای سیستم ایجاد میکند. این سیستم لازم است که حداقل بار محاسبات را داشته باشد تا بتواند با پردازنده های عادی و رایج سازگار باشد. هدف بعدی در این سیستم طراحی دقیق آن است تا بتواند با دوربینهای معمولی و کم کیفیت همانند وبکمهای کامپیوترهای همراه به خوبی عمل نماید. این دو ویژگی اصلی یعنی کم بودن بار محاسبات و تطبیق پذیری با دوربینهای معمولی سبب شد که در این پروژه از الگوریتم تطبیق الگو که روشی دقیق و با بار محاسباتی کم است استفاده شود.

۲-۲- طرح کلی سیستم

برای ردگیری چشم انسان روشهای متفاوتی وجود دارد. در کتابخانه OpenCV کلاسیفایری به نام کلاسیفایر هار وجود دارد که با استفاده از دیتابیس غنی که در اختیار دارد، قادر است چشم انسان و یا انسانها را در یک تصویر پیدا کند. بنابرین ساده ترین روشی که برای ردگیری چشم انسان به نظر میرسد استفاده از کلاسیفایر هار و یافتن چشم در تصویر در هر فریم است. [2]

مشکلی که در اینجا وجود دارد این است که این کار بار محاسباتی بسیار بالایی دارد. اولین راه حلی که برای حل این مشکل به نظر میرسد راه حل ساده محدود کردن فضای جستجو است. بدین معنی که به جای اینکه کلاسیفایر هار در کل تصویر به دنبال چشم افراد بگردد، تنها در صورت افراد به دنبال چشم آنها بگردد. با این روش بار محاسبات و دقت سیستم به شکل چشمگیری افزایش می یابد.

بنابرین برای پیاده کردن چنین سیستمی ابتدا لازم است صورتها در تصویر یافته شوند. برای پیدا کردن صورتها نیز میتوان از کلاسیفایر هار استفاده کرد. این کلاسیفایر دیتابیس مربوط به صورتهای مختلف را دارا است که میتوان با استفاده از آن صورتهای مختلف موجود در تصویر را شناسایی کرد.

اما تا اینجای کار هنوز بار محاسبات بالا است. اگر بخواهیم در هر فریم با استفاده از کلاسیفایر هار صورت صورتها را تشخیص دهیم و آنگاه در آن صورتها با استفاده از این کلاسیفایر دنبال چشم بگردیم نیاز به محاسبات گسترده ای داریم. راه حل فرار از این حجم سنگین محاسبات استفاده از روشهای تطبیق الگو است.

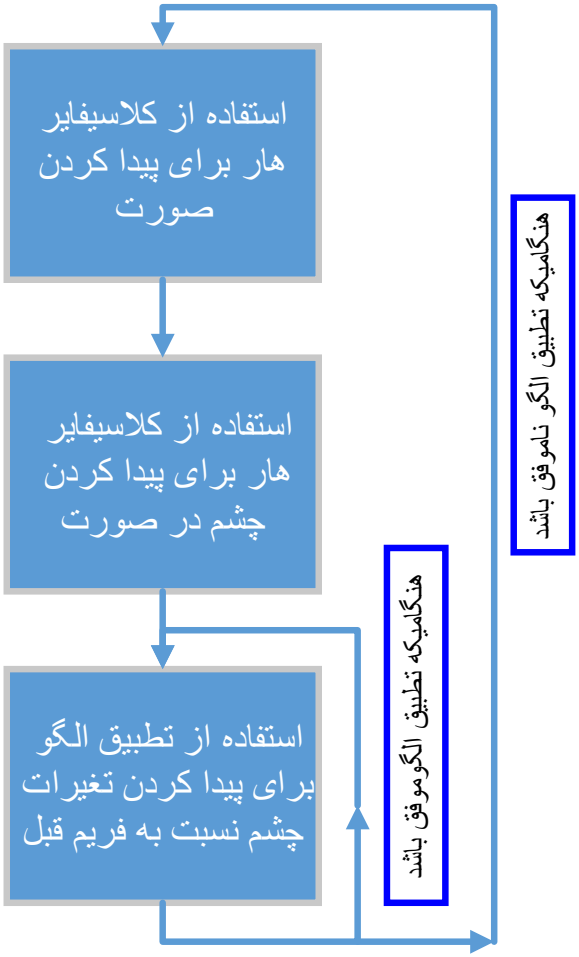
روش تطبیق الگو یکی از روشهای شاخص پردازش تصویر است که به منظور یافتن یک الگوی خاص در تصویر به کار میرود. ویژگی اصلی این روش که موجب محبوبیت آن شده است بار محاسباتی پایین آن است. در این روش بخشی از تصویر به نام "ناحیه مورد نظر"[۶] با استفاده از یک الگو جاروب شده و بخش مشابه با الگوی مورد نظر در آن یافته میشود. [3]

بنابرین در سیستم مورد نظر ما در صورتیکه بتوانیم چشم فرد را در تصویر پیدا کنیم. در فریمهای بعدی میتوان چشم فرد را به عنوان الگو در روش تطبیق الگو بکار برد و با استفاده از آن در فریمهای بعدی چشم را پیدا کرد.

بنابرین روندی که در این پروژه طی میشود این است که ابتدا با استفاده از کلاسیفایر هار صورتها در تصویر پیدا میشود. سپس با استفاده از همین کلاسیفایر چشمها در صورتها پیدا میشوند. در فریمهای بعدی با استفاده از روش تطبیق الگو چشمهای پیدا شده در فریم اول در تصویر یافته میشوند. شمای کلی این عملکرد در شکل ۲-۱ نمایش داده شده است.

لازم به ذکر است که برای کاهش بیشتر بار محاسبات، تطبیق الگو نیز تنها در ناحیه صورت، انجام میگیرد و برای پیدا کردن صورت در فریمهای بعد از روشی هوشمند استفاده میشود که در فصل چهارم توضیح داده خواهد شد.

۶- Region of Interest (ROI)



شکل۲-۱: شمای عملکرد کلی سیستم پیاده سازی شده

علاوه بر آن در این سیستم مکانیزمی مورد نیاز است که در صورتیکه به هر دلیلی از جمله: نویز، عملکرد اشتباه سیستم، خطا در روش تطبیق الگو، وارد شدن یا خارج شدن افراد جدید به تصویر نیاز به بکارگیری مجدد کلاسیفایرها در بخش تشخیص صورت و چشم باشد، برنامه در فریم بعدی به این بخش رفته و مجدداً همانند فریم اول کلاسیفایرها را برای تشخیص صورت و چشم بکارگرفته و در فریمهای بعدی مجدداً برنامه وارد بخش تطبیق الگو میشود. این مکانیزم سبب میشود که از ورود برنامه به حلقه های خطا و باقی ماندن از آن جلوگیری شود. این مکانیزم نیز در این سیستم پیش بینی شده است و پیاده سازی آن در فصل چهارم نشان داده میشود.

۲-۳- ویژگیهای سیستم

برخ از ویژگیهای سیستم طراحی شده بطور خلاصه در زیر آورده شده است:

- بلادرنگ بودن سیستم
- بار محاسباتی پایین
- سازگار با دوربینهای عادی با تفکیک پذیری پایین مثل وبکمهای کامپیوترهای همراه
- قابلیت تشخیص خطا در چرخه تطبیق الگو
- قابلیت تشخیص چشم بیش از یک فرد در تصویر به شکل همزمان
- عدم حساسیت به شدت روشنایی محیط

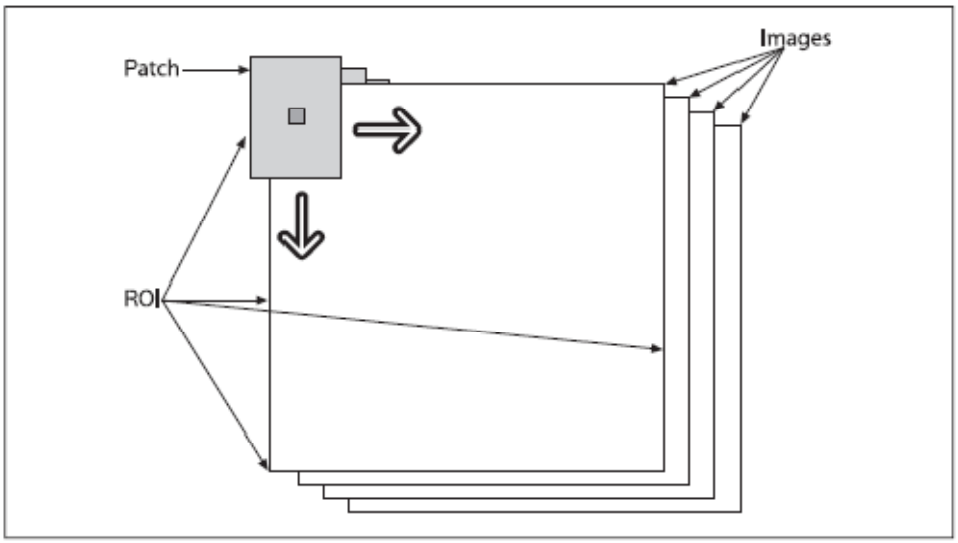
فصل سوم

روش تطبیق الگو

۳-۱- کلیات روش تطبیق الگو

روش تطبیق الگو روشی سودمند و پرکاربرد در پردازش تصویر است که به منظور یافتن یک الگوی خاص در تصویر به کار میرود. ویژگی اصلی این روش که موجب محبوبیت آن شده است بار محاسباتی پایین آن است. همین ویژگی سبب شده است که در این پروژه که به دلیل بلادرنگ بودن سیستم لازم است که بار محاسباتی تا حد امکان کاهش داده شود، از روش تطبیق الگو استفاده شده است.

در این روش بخشی از تصویر به نام "ناحیه مورد نظر"[۷] با استفاده از یک الگو جاروب شده و بخش مشابه با الگوی مورد نظر در آن یافته میشود. شکل ۳-۱ شمای کلی عملکرد اسن روش را نمایش میدهد. همانطور که در این شکل دیده میشود الگوی مورد نظر از سمت چپ بالای ناحیه مورد نظر آغاز به جاروب شدن کرده و تا انتهای محدوده جاروب میگردد. [4]



شکل ۳-۱: شمای کلی عملکرد روش تطبیق الگو

[7] - Region of Interest (ROI)

الگوریتمهای متفاوتی برای پیاده سازی تطبیق الگو وجود دارد که در دقت محاسبات و حجم محاسبات با هم تفاوت دارند. در ابتدایی ترین الگوریتم تطبیق الگو به نام "مجموع مربع تفاضلات" مقایسه تصویر و الگو با استفاده از مجموع مربع اختلاف روشنایی پیکسلها در الگو و ناحیه مورد نظر به دست می آید. هرچه مقدار مجموع مربع تفاضل پیکسلها به عدد صفر نزدیکتر باشد تصویر به الگوی مورد نظر شباهت بیشتری داشته و هرچه این مقدار بزرگتر باشد شباهت کمتر است. رابطه استفاده شده در این روش ابتدایی به شکل زیر است:

$$R_{\text{sq\_diff}}(x,y)=\sum_{x',y'}[T(x',y')-I(x+x',y+y')]^2$$

در رابطه بالا T (x',y') روشنایی پیکسلهای پترن و I (x',y') روشنایی پیکسلهای ناحیه مورد نظر میباشد.

کتابخانه OPEN CV قادر به اجاری تطبیق الگو با روشهای متفاوت میباشد. تابع موجود در این کتابخانه برای پیاده سازی تطبیق الگو تابع cvMatchTemplate() است که با گرفتن آرگومانهای متفاوت، تطبیق الگو را با روشهای متفاوت پیاده سازی میکند. در این پروژه روش تطبیق الگوی استفاده شده روش NCCM است که در ادامه شرح داده میشود.

۳-۲- الگوریتم NCCM

برای بهبود مقایسه و همچنین کاهش بار محاسبات، از روش NCCM استفاده میگردد. این روش دو تفاوت عمده با الگوریتم مربع تفاضلات دارد:

- به جای مجموع مربع تفاضلات از کورلیشن (هم بستگی) پیکسلها نسبت به هم استفاده میگردد. استفاده از کرولیشن نسبت به تفاضل معیار مطلوبتری از شباهت نسبت به روشهای تفاضلی به کاربر ارائه میکند.

- روشنایی ناحیه مورد نظر و الگو نسبت به متوسط روشنایی پیکسلهای آن نرمالیزه میشود. این کار به این دلیل صورت میگیرد که با تغییر روشنایی اندازه گیری شباهت دچار مشکل نگردد.

به همین دلیل در تصاویری که الگو پیچیده گی های بیشتری دارد از این روش استفاده میگردد. به عنوان مثال در شکل ۳-۲ مشاهده میگردد که الگو چشم فرد است که باید در ناحیه مورد نظر یافته شود.

شکل ۳-۲: پیاده سازی تطبیق الگو با استفاده از الگوریتم $NCCM$

از آنجا که چشم افراد متفاوت شباهتهای بسیاری با هم دارد، در صورتی که از روشهای ساده تر در این مورد استفاده گردد ممکن است که در خروجی روش تطبیق الگوچشم فرد دوم به جای چشم فرد اول یافته شود. به همین دلیل در این مثال باید از روشهای دقیقتر مثل روش NCCM استفاده کرد. در این پروژه نیز به دلیل نیاز به دقت بالا از روش NCCM استفاده شده است.

روش NCCM با استفاده از رابطه زیر پیاده سازی میگردد:

رابطه اصلی NCCM

$$R_{ccoeff}(x,y)=\sum_{x',y'}[T'(x',y')\cdot I'(x+x',y+y')]^2$$

نرمالیزه کردن روشنایی الگو

$$T'(x',y')=T(x',y')-\frac{1}{(w\cdot h)\sum_{x'',y''}T(x'',y'')}$$

نرمالیزه کردن روشنایی ROI

$$I'(x+x',y+y')=I(x+x',y+y')-\frac{1}{(w\cdot h)\sum_{x'',y''}I(x+x'',y+y'')}$$

در روابط بالا T’(x’,y’) روشنایی پیکسلهای پترن بعد ار نرمالیزه شدن و I’(x’,y’) روشنایی پیکسلهای مورد نظر پس از نرمالیزه شدن است. مشاهده میگردد که کرولیشن با استفاده از سیگمای شیفت داده شده پیاده شده است. این سیگمای شیفت داده شده با دو حلقه تو در تو به راحتی قابل پیاده سازی است.

فصل چهارم

سیستم ردگیری چشم

پیاده شده در پروژه

۴-۱- ساختار کلی کد ارائه شده

ساختار ارائه شده در این پروژه تلفیقی از استفاده از کلاسیفایرها و استفاده از روش تطبیق الگو است. همانطور که در فصل دوم بیان شد در این پروژه برای پیدا کردن صورت و چشم از کلاسیفایر هار استفاده شده است. استفاده از این کلاسیفایر بار محاسباتی بالایی داشته و به همین دلیل استفاده مستقیم از این کلاسیفایر برای پیاده سازی یک سیستم بیدرنگ مناسب نمی‌باشد. برای حل این مشکل در این پروژه چند ایده اساسی استفاده شده است که در زیر به اختصار آورده شده و در ادامه به تفصیل توضیح داده میشود:

-استفاده از روش تطبیق الگو و محدود کردن استفاده از کلاسیفایرها به هر چند فریم یک بار

- محدود کردن ناحیه جستجوی کلاسیفایر هار برای پیدا کردن چشم به صورت فرد به جای کل تصویر

- محدود کردن ناحیه مورد نطر در تطبیق الگو به صورت فرد به جای کل تصویر

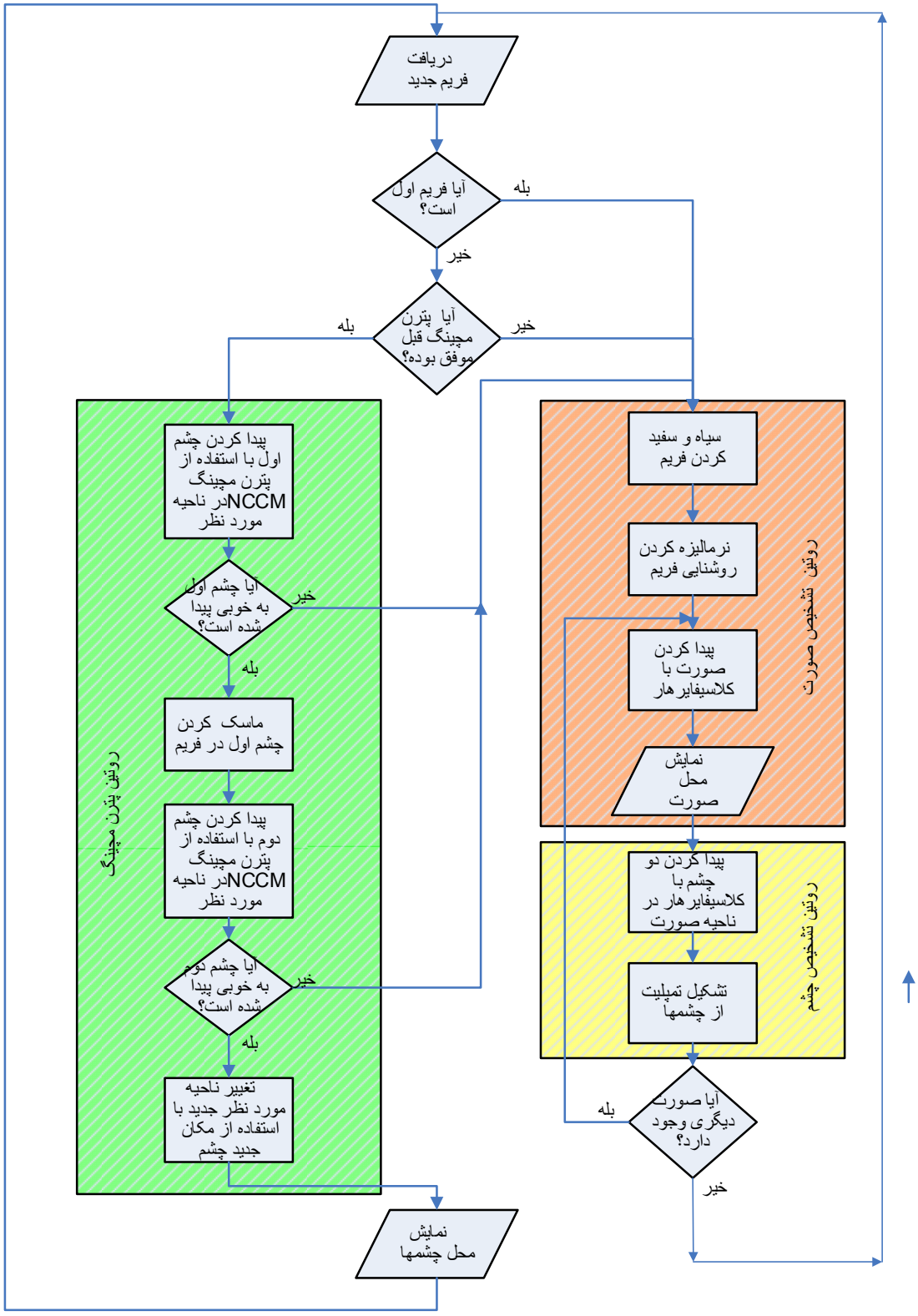
- به روز رسانی هوشمند مکان صورت فرد به جای کاربرد پیاپی کلاسیفایر هار

پیاده سازی ایده های ذکر شده در بالا موجب کاهش چشمگیر حجم عملیات پردازشی در این پروژه شده است.

شکل ۴-۱ فلوچارت ساختار کلی پیاده شده در این پروژه را نمایش میدهد. همانطور که مشاهده میشود در ابتدا یک فریم از وبکم یا دوربین پیشفرض سیستم دریافته شده و عملیات پردازش آغاز میگردد. در صورتی که این فریم اولین فریم دریافتی باشد، سیستم به بخش تشخیص صورت با استفاده از کلاسیفایرها میرود. در این بخش صورت شخص با استفاده از کلاسیفایر هار پیدا شده و مختصات آن به شکل مختصات یک مستطیل ذخیره میگردد. توضیح بیشتر درباره جزئیات تشخیص ضورت در بخش ۴-۲ ارائه میگردد.

پس از تشخیص صورت نوبت به پیدا کردن چشمها با استفاده از کلاسیفایر هار میشود. برای کاهش بار محاسبات محدوده عملکرد کلاسیفایر هار بجای کل تصویر به صورت فرد خلاصه میگردد. این تکنیک علاوه بر کاهش بار محاسبات موجب افزایش چشمگیر دقت عملکرد سیستم نیز میشود. پس از پیدا کردن چشمها، آنها را به عنوان الگو برای استفاده در بخش تطبیق الگو ذخیره میکنیم. در صورتی که چند صورت در فریم موجود باشد عملیات تشخیص صورت و تشخیص چشم برای تک تک آنها انجام شده و پس از اتمام صورتها برنامه آماده دریافت فرم جدید میشود.[5]

در این فریم دیگر از کلاسیفایر هار استفاده نمیشود و به جای این عمل وقتگیر برای فریمهای متوالی برنامه وارد بخش تطبیق الگو میشود. در این بخش مکان چشمها با استفاده از تطبیق الگو و در زمان بسیار کوتاهتر با بار محاسباتی بسیار کمتر پیدا شده و نمایش داده میشود. در صورتیکه به هر دلیلی از جمله: نویز، هملکرد اشتباه سیستم، خطا در روش تطبیق الگو، وارد شدن یا خارج شدن افراد جدید به تصویر نیاز به بکارگیری مجدد کلاسیفایرها در بخش تشخیص صورت و چشم باشد، برنامه در فریم بعدی به این بخش رفته و مجدداً همانند فریم اول کلاسیفایرها را برای تشخیص صورت و چشم بکارگرفته و در فریمهای بعدی مجدداً برنامه وارد بخش تطبیق الگو میشود. این مکانیزم سبب میشود که از ورود برنامه به حلقه های خطا و باقی ماندن از آن جلوگیری شود. مکانیزم هوشمند تشخیص زمان سوئیچ سیستم به بخشتشخیص صورت در قسمت ۴-۴ به تفصیل توضیح داده میشود.[6]



شکل ۴-۱ : فلوچارت سیستم پیاده شده در این پروژه

۴-۲- بخش پیدا کننده صورت

در این بخش ایده اصلی استفاده از کلاسیفایر هار برای تشخیص دادن صورت است. این بخش از برنامه در کد در تابع DetectFace نوشته شده است که به عنوان ورودی ماتریسی که حامل پیکسلهای فریم است را دریافت میکند. در این فصل در هربخش خلاصه ای از کد که مربوط به همان بخش است برای سهولت فهم آورده میشود اما کد موجود در ضمیمه ۱ به طور کامل و همراه با کامنتهای کافی در انتهای پایان نامه موجود میباشد..

نخستین کاری که در این بخش انجام میشود تبدیل فریم دریافت شده از دوربین که رنگی است به یک فریم سیاه سفید که پیکسلها تنها مشخصه روشنایی دارند میباشد. این عمل با استفاده از تابع ()cvtColor که از توابع کتابخانه OpenCV است انجام میگردد. پس از آنکه فریم مورد نظر به فریم سیاه و سفید تبدیل شد نوبت به نرمالیزه کردن آن میرسد.

یکی از مفیدترین کارهایی که به منظور کاهش خطا در بکارگیری کلاسیفایر هار انجام میگردد، نرمالیزه کردن روشنایی تصویر است. این عمل علاوه بر بهبود نتیجه دریافت شده از کلاسیفایر موجب کاهش خطای سیستم در مواجهه با تغییر روشنایی محیط نیز میگردد. از آنجایی که میانگین روشنایی تصویر دریافت شده از وبکمها معمولاً تغیرات زیادی میکند نرمالیزه کردن روشنایی فریم در این پروژه بسیار سودمند است. نرمالیزه کردن روشنایی با استفاده از تابع ()equalizeHist انجام میگردد. اکنون فریم مورد نظر آماده برای پیاده شدن کلاسیفایر هار است.

کلاسیفایر هار با استفاده از تابع ()face_cascade.detectMultiScale پیاده میشود. نکته قابل ذکر در اینجا این است که کلاسیفایر هار تمامی صورتهای موجود در یک تصویر را پیدا میکند. به همین دلیل در نوشتن برنامه به این نکته دقت شده است که خروجی کلاسیفایر هار در برداری از مستطیلها ذخیره میگردد. سپس به ازای هر صورت یک بار روتین بخش پیدا کننده چشم اجرا شده و چشمهای موجود در هر صورت یافته میشوند. در انتهای این بخش در صورتیکه کلاسیفایر هار حداقل یک صورت در تصویر پیدا کرده باشد برنامه ادامه داده

میشود اما در غیر اینصورت پرچم مربوط به بخش ورود به تطبیق الگو غیر فعال شده و برای فریم بعد نیز برنامه وارد همین بخش پیدا کننده صورت میشود.

تابع مربوط به روتین پیدا کردن صورت به طور خلاصه در این بخش آورده شده است:

```
void detectFace(Mat &frame)
{
	std::vector<Rect> faces;
	Mat frame_gray;
	cvtColor(frame, frame_gray, CV_BGR2GRAY);
	equalizeHist(frame_gray, frame_gray);
	face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
	Size(30, 30));
	for (int i = 0; i < faces.size(); i++)
	{
		rectangle(frame, faces[i], Scalar(255, 0, 0),4);
		detectEye(frame, faces[i]);
	}

	if (eyeTemplate1.size() > 0)
	{
		pframe = false;
		cout << "pframe done" << endl;
	}
	else
	{
		pframe = true;
	}
}
```

۴-۳- بخش پیدا کننده چشم

در این بخش نیز از کلاسیفایر هار برای یافتن چشم استفاده میشود. این بخش از برنامه در کد در تابع DetectEye نوشته شده است که به عنوان ورودی ماتریسی که حامل پیکسلهای فریم است و همچنین مستطیل مکان صورت را دریافت میکند. در این بخش نیز ابتدا با تابع با استفاده از تابع cvtColor() فریم رنگی برای پردازش به فریم سیاه و سفید تبدیل میشود. مستطیل مکان صورت که به عنوان ورودی تابع DetectEye از تابع DetectFace گرفته شده است در اینجا به عنوان حوزه عملکرد کلاسیفایر هار برای پیدا کردن چشم تعریف

میگردد. همانطور که توضیح داده شد جستجوی چشم تنها در ناحیه صورت به جای کل تصویر موجب افزایش چشمگیر سرعت و دقت میگردد.

در مرحله بعد کلاسیفایر هار برای پیدا کردن چشم با استفاده از تابع eye_cascade.detectMultiScale() پیاده سازی میگردد. خروجی این تابع چشمهای صورت مورد نظر است. در صورتیکه کلاسیفایر هار در این بخش درست عمل کرده باشد و خروجی آن حاوی دو مستطیل که مختصات چشم را دارند باشد، برنامه مختصات این چشمها را متغیر eyeTemplate1 و eyeTemplate2 به عنوان الگوهای لازم در بخش تطبیق الگو ذخیره میکند.

تابع مربوط به روتین پیدا کردن چشم به طور خلاصه در این بخش آورده شده است:

```
void detectEye(Mat &frame, Rect faceRect)
{
	std::vector<Rect> eyes;
	Mat frame_gray;
	cvtColor(frame, frame_gray, CV_BGR2GRAY);
	Mat faceROI = frame_gray(faceRect);
	imshow("ROI", faceROI);
	eye_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
	Size(5, 5));

	if (eyes.size() > 1)
	{
		Rect shiftedEyeRect1 = Rect(faceRect.x + eyes[0].x, faceRect.y + eyes[0].y,
		eyes[0].width, eyes[0].height);
		Rect shiftedEyeRect2 = Rect(faceRect.x + eyes[1].x, faceRect.y + eyes[1].y,
		eyes[1].width, eyes[1].height);

		searchRegion.push_back(faceRect);
		eyeRegion.push_back(eyes[0]);
		eyeTemplate1.push_back(frame_gray(shiftedEyeRect1));
		eyeTemplate2.push_back(frame_gray(shiftedEyeRect2));
	}
}
```

۴-۴- پیاده سازی تطبیق الگو

این بخش از برنامه که مربوط به پیاده سازی تطبیق الگو است در کد در تابع PatternMatching() نوشته شده است که به عنوان ورودی ماتریسی که حامل پیکسلهای فریم است را دریافت میکند. پترن مچینگ با استفاده از

الگوریتم NCMM با استفاده از تابع matchTemplate() پیاده میشود. آرگومان CV_TM_CCOEFF_NORMED در این تابع دستور میدهد که روش محاسبه تطبیق الگو منطبق بر روش NCCM باشد. پس از تطبیق الگو ناحیه ای که بالاترین تطبیق در آنجا صورت گرفته پیدا میشود. این بخش چشم فرد در فریم جدید خواهد بود و بدین ترتیب بدون استفاده از کلاسیفایر میتوان چشم فرد را پیدا کرد.

پس از اینکه اولین چشم صورت پیدا شد، عملیات برای یافتن چشم دوم آغاز میشود. در اینجا به دلیل آنکه معمولاً دو چشم انسان شباهت بسیاری با هم دارند به منظور آنکه در فرایند چشم دوم به اشتباه مجدداً چشم اول پیدا نشود، چشم اول که در فرایند قبل پیدا شده ماسک میشود، یعنی پیکسلهای موجود در آن ناحیه تماما پاک میشود، و مجداداً فرایند تطبیق الگو این بار بر مبنای الگوی چشم دوم آغاز میگردد.

پس از پیدا کردن چشم دوم، عملی بسیار مهم صورت میگیرد که همان بروز رسانی ناحیه مورد نظر است. در این سیستم برای اینکه با حرکت سر فرد، تطبیق الگو دچار مشکل نشود و به صورت پیاپی نیاز به فراخوانی بخش پیدا کردن صورت نباشد، سیستم به صورت هوشمن حرکت سر فرد را پیش پیدا میکند و ناحیه مورد نظر برای تطبیق الگو را با آن متناظر میکند. این کار بدین صورت انجام میشود که بردار حرکت چشم فرد نسبت به فریم قبلی پیدا شده و به همان اندازه ناحیه مورد نظر در تطبیق الگو (که همان سر فرد است) شیفت داده میشود. این بخش که یکی از مهمترین بخشهای پیاده شده در این سیستم است در انتهای تطبیق الگو قرار داده شده و کمک شایانی به کاهش حجم محاسبات میکند.

بخشهای مهم تابع مربوط به پیاده سازی تطبیق الگو به طور خلاصه در اینجا آورده شده است. برای مشاهده کد کامل تطبیق الگو به ضمیمه ۱ مراجعه شود.

```
void PatternMatching(Mat &frame)
{
	Mat frame_gray;
	cvtColor(frame, frame_gray, CV_BGR2GRAY);
	for (int i = 0; i < eyeTemplate1.size(); i++)
```

```
	{

		// define the template matching result
		Mat tmMat1 = Mat(searchRegion[i].width - eyeTemplate1[i].cols + 1,
		searchRegion[i].height - eyeTemplate1[i].rows + 1, CV_32FC1);
		Mat tmMat2 = Mat(searchRegion[i].width - eyeTemplate2[i].cols + 1,
		searchRegion[i].height - eyeTemplate2[i].rows + 1, CV_32FC1);
		// run template matching
		matchTemplate(frame_gray(searchRegion[i]), eyeTemplate1[i], tmMat1,
		CV_TM_CCOEFF_NORMED);
		// search for the maximum matching point
		double minval1, maxval1;
		Point minloc1, maxloc1;
		minMaxLoc(tmMat1, &minval1, &maxval1, &minloc1, &maxloc1);
		if (maxval1 >= 0.7)
		{
			cout << "matched!" << endl;
			Rect eye = Rect(searchRegion[i].x + maxloc1.x, searchRegion[i].y +
			maxloc1.y, eyeTemplate1[i].rows, eyeTemplate1[i].cols);
			rectangle(frame, eye, Scalar(255, 255, 255), 2);

			Mat frame_masked = frame_gray.clone();
			for (int i = eye.x-5; i < eye.x + eye.width+5; i++)
			{
				for (int j = eye.y-5; j < eye.y + eye.height+5; j++)
				{
					frame_masked.at<uchar>(j, i) = 128;
				}

			}

			// Update search region
			searchRegion[i].x += maxloc1.x - eyeRegion[i].x;
			searchRegion[i].y += maxloc1.y - eyeRegion[i].y;

			if (searchRegion[i].x < 0)  searchRegion[i].x = 0;
			if (searchRegion[i].x >= frame.cols)searchRegion[i].x = frame.cols -
			1;
			if ((searchRegion[i].x + searchRegion[i].width) >= frame.cols)
			searchRegion[i].width = frame.cols - 1 - searchRegion[i].x;
			if (searchRegion[i].y < 0)  searchRegion[i].y = 0;
			if (searchRegion[i].y > frame.rows)searchRegion[i].y = frame.rows -
			1;
			if ((searchRegion[i].y + searchRegion[i].height) >= frame.rows)
			searchRegion[i].height = frame.rows - 1 - searchRegion[i].y;

		}
		else
		{
			cout << "first eye not matched!" << endl;
			pframe = true;
		}
	}
}
```

فصل پنجم

جمع بندی

۵-۱- جمع بندی و نتیجه گیری

طی این پروژه سیستم ردگیری بیدرنگ چشم انسان با استفاده از روش تطبیق الگو پیاده سازی شد. ویژگی اصلی این سیستم بار محاسباتی پایین آن است که موجب گشته است که این سیستم بر روی کامپیوترهای معمولی به راحتی به صورت بیدرنگ کار کند. همچنین دقت بالای این سیستم سبب شده است که با دوربینهای با تفکیک پذیری پایین مثل وبکم نیز سیستم عملکرد مناسبی دارد. برای طراحی این سیستم از تلفیق کلاسفایر هار و روش تطبیق الگو با الگوریتم NCCM استفاده گردد. این سیستم علاوه بر بلادرنگ بودن ویژگیهای دیگری از جمله قابلیت تشخیص همزمان بیش از یک فرد در تصویر و عدم حساسیت به شدت روشنایی محیط را نیز دارا میباشد. پیاده سازی این سیستم در با استفاده از زبان C++ انجام گردید.

۵-۲- ارائه پیشنهادات

در طی انجام این پروژه به موارد متعددی برخورد کردیم که جای کار بیشتر دارد. برخی از آنها به عنوان پیشنهادات در پایان آورده میشود:

- پیاده سازی روش تطبیق الگو با استفاده از روشهای دقیقتر و جدیدتر
- بازبینی در مکانیزم هوشمند بازگشت از تطبیق الگو به بکارگیری کلاسیفایرها
- استفاده از فیلترهای متفاوت در فریمهای اولیه

منابع و مرا جع

[1] Current and future use of image processing, Alen L. Mc Roberts, International Symposium on the Forensic Application of Digital Image Processing, 1986.

[2] FACIAL FEATURE DETECTION USING HAAR CLASSIFIERS, Phillip Ian Wilson, JCSC 21, 4 (April 2006)

[3] Template Matching Based Object Recognition With Unknown Geometric Parameters, Roger M. Dufour, Eric L. Miller, and Nikolas P. Galatsanos, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 11, NO. 12, DECEMBER 2002.

[4] Learning OpenCV, Gary Bradski, Adrian Kaehler, O'Reily, 2008, First Edition.

[5] Eye Tracking by Template Matching using an Automatic Codebook Generation Scheme, M.J.T. Reinders.

[6] Eyes Tracking in a Video Sequence based-on Haar-like Features and Skin Color, Qi Han, Qiong Li, Xiongdong Huang, Tong Zhou, Xiamu Niu.

ضمیمه ۱: کد C++ سیستم پیاده شده

کد کامل سیستم پیاده شده در زیر آورده شده است. لازم به یادآوری است که وجود فایلهای کتابخانه های فراخوانی در کنار فایل سورس برنامه برای اجرای برنامه الزامی است.

```
// EyeTracking.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
// Add required OpenCV include files here
#include "opencv2\core\core.hpp"                //always required
#include "opencv2\highgui\highgui.hpp"          //for displaying images in windows
#include "opencv2\objdetect\objdetect.hpp"      // for object detection,including the
classifiers
#include "opencv2\imgproc\imgproc.hpp"          //general image processing functions


// so that we won't need to use cv::*** in the code
using namespace cv;
using namespace std;

/*Global Variables*/
String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eyes_cascade_name = "haarcascade_eye_tree_eyeglasses.xml";
String smile_cascade_name = "haarcascade_smile.xml";
CascadeClassifier face_cascade;
CascadeClassifier eye_cascade;
CascadeClassifier smile_cascade;
bool pframe = true;
vector<Rect> searchRegion;
vector<Rect> eyeRegion;
vector<Mat> eyeTemplate1;
vector<Mat> eyeTemplate2;

/*Function prototypes*/
void detectFace(Mat &frame);
void detectEye(Mat &frame, Rect faceRect);
void PatternMatching(Mat &frame);

int _tmain(int argc, _TCHAR* argv[])
{
	///*Read and display an image for 1000 msec*/
	//// read an image
	//Mat image = imread("1.jpg");
	//// create a window
	//namedWindow("My image");
	////show the image
	//imshow("My image", image);
	////wait for keypress for x milliseconds (zero for unlimited delay, until
keypress)
```

```
//waitKey(1000);

///*Capture frames from the default capturing device*/
////create a capture object for default device
//VideoCapture cap(0);
////create a Mat to store each incoming frame
//Mat frame;
//// each iteration of while loop takes 50 msec
//// stop condition: press 'q'
//while (waitKey(1) != 'q')
//{
//     // capture a new frame
//     cap >> frame;
//     // display
//     imshow("Frame", frame);
//}

/*Eye Detection*/
// 1. Load face and eye cascade classifiers files
// the output of face_cascade.load is true, if the files are loaded correctly
if (!face_cascade.load(face_cascade_name))
{
       printf("Error loading face cascade xml\n");
       return -1;
}
if (!eye_cascade.load(eyes_cascade_name))
{
       printf("Error loading eye cascade xml\n");
       return -1;
}

// 2. Main loop of face detection
//create a capture object for default device
VideoCapture cap(0);

// the output of isOpened is true, if the device driver is loaded successfully
if (!cap.isOpened())
{
       printf("Error opening the default capturing device\n");
       return -1;
}

//create a Mat to store each incoming frame
Mat frame;

// Initialize searchRegion and eyeTemplate
searchRegion.clear();
eyeRegion.clear();
eyeTemplate1.clear();
eyeTemplate2.clear();

// each iteration of while loop takes at least 1 msec
// stop condition: press 'q'
while (waitKey(1) != 'q')
{
       // capture a new frame
       cap >> frame;
       // check the new frame. skip the iteration if the new frame is null
```

```
		// empty routine returns true if the new frame is corrupted
		if (frame.empty())
		{
			printf("Frame skipped\n");
			continue;
		}

		//
		if (pframe == true)
		{
			// Template matching has failed, use cascade classifier

			// Reset the number of active faces
			searchRegion.clear();
			eyeTemplate1.clear();
			eyeTemplate2.clear();

			// detect faces
			detectFace(frame);
		}
		else
		{
			// use template matching
			PatternMatching(frame);
		}

		// display
		imshow("Frame", frame);
	}

	waitKey(0);


	return 0;
}

// function for implementing pattern matching
void PatternMatching(Mat &frame)
{
	// convert the colored frame to gray and store it in frame_gray
	Mat frame_gray;
	cvtColor(frame, frame_gray, CV_BGR2GRAY);

	for (int i = 0; i < eyeTemplate1.size(); i++)
	//for (int i = 0; i < 1; i++)
	{
		//
		stringstream name;
		name<<"template"<< (i + 1);
		imshow(name.str(), eyeTemplate1[i]);
		rectangle(frame, searchRegion[i], Scalar(0, 0, 0), 1);
		// define the template matching result
		Mat tmMat1 = Mat(searchRegion[i].width - eyeTemplate1[i].cols + 1,
searchRegion[i].height - eyeTemplate1[i].rows + 1, CV_32FC1);
		Mat tmMat2 = Mat(searchRegion[i].width - eyeTemplate2[i].cols + 1,
searchRegion[i].height - eyeTemplate2[i].rows + 1, CV_32FC1);
		// run template matching
```

```
		matchTemplate(frame_gray(searchRegion[i]), eyeTemplate1[i], tmMat1,
CV_TM_CCOEFF_NORMED);
		// search for the maximum matching point
		double minval1, maxval1;
		Point minloc1, maxloc1;
		minMaxLoc(tmMat1, &minval1, &maxval1, &minloc1, &maxloc1);
		if (maxval1 >= 0.7)
		{
			cout << "matched!" << endl;
			Rect eye = Rect(searchRegion[i].x + maxloc1.x, searchRegion[i].y +
maxloc1.y, eyeTemplate1[i].rows, eyeTemplate1[i].cols);
			// Draw rectangle
			rectangle(frame, eye, Scalar(255, 255, 255), 2);

			// find the other eye by masking the found one
			// make a copy of the search region
			Mat frame_masked = frame_gray.clone();
			for (int i = eye.x-5; i < eye.x + eye.width+5; i++)
			{
				for (int j = eye.y-5; j < eye.y + eye.height+5; j++)
				{
					frame_masked.at<uchar>(j, i) = 128;
				}

			}
			imshow("frame_masked", frame_masked);
			// run template matching
			matchTemplate(frame_masked(searchRegion[i]), eyeTemplate2[i],
tmMat2, CV_TM_CCOEFF_NORMED);
			// search for the maximum matching point
			double minval2, maxval2;
			Point minloc2, maxloc2;
			minMaxLoc(tmMat2, &minval2, &maxval2, &minloc2, &maxloc2);
			if (maxval2 >= 0.7)
			{
				Rect eye = Rect(searchRegion[i].x + maxloc2.x,
searchRegion[i].y + maxloc2.y, eyeTemplate2[i].rows, eyeTemplate2[i].cols);
				// Draw rectangle
				rectangle(frame, eye, Scalar(0, 255, 255), 2);
			}
			else
			{
				// Set pframe to use the classifiers
				cout << "second eye not matched!" << endl;
				pframe = true;
			}

			// Update search region
			searchRegion[i].x += maxloc1.x - eyeRegion[i].x;
			searchRegion[i].y += maxloc1.y - eyeRegion[i].y;

			if (searchRegion[i].x < 0)  searchRegion[i].x = 0;
			if (searchRegion[i].x >= frame.cols)     searchRegion[i].x =
frame.cols - 1;
			if ((searchRegion[i].x + searchRegion[i].width) >= frame.cols)
	searchRegion[i].width = frame.cols - 1 - searchRegion[i].x;
			if (searchRegion[i].y < 0)  searchRegion[i].y = 0;
```

```
                    if (searchRegion[i].y > frame.rows)      searchRegion[i].y =
frame.rows - 1;
                    if ((searchRegion[i].y + searchRegion[i].height) >= frame.rows)
       searchRegion[i].height = frame.rows - 1 - searchRegion[i].y;

              }
              else
              {
                    // Set pframe to use the classifiers
                    cout << "first eye not matched!" << endl;
                    pframe = true;
              }
       }
}

// function for detect faces in a new frame
void detectFace(Mat &frame)
{
       // create a vector of Rects to store the recognized faces for a single frame
       std::vector<Rect> faces;
       // create a Mat to store the gray version of the captured frame
       Mat frame_gray;
       // convert the colored frame to gray and store it in frame_gray
       cvtColor(frame, frame_gray, CV_BGR2GRAY);
       // Normalize the intensity of the frame
       equalizeHist(frame_gray, frame_gray);

       //Detect faces using cascade classifier
       //detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor =
1.1, int minNeighbors = 3, int flags = 0,Size minSize = Size(), Size maxSize = Size())
       //image – Matrix of the type CV_8U containing an image where objects are detected
       //objects – Vector of rectangles where each rectangle contains the detected
object.
       //scaleFactor – Parameter specifying how much the image size is reduced at each
image scale.
       //minNeighbors – Parameter specifying how many neighbors each candidate rectangle
should have to retain it.
       //flags – Parameter with the same meaning for an old cascade as in the function
       //cvHaarDetectObjects. It is not used for a new cascade.
       //minSize – Minimum possible object size.Objects smaller than that are ignored.
       //maxSize – Maximum possible object size.Objects larger than that are ignored.
       face_cascade.detectMultiScale(frame_gray, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
Size(30, 30));

       //for each recognized face, do the followings
       for (int i = 0; i < faces.size(); i++)
       {
              // Draw an ellipse
              //Point center(faces[i].x + faces[i].width*0.5, faces[i].y +
faces[i].height*0.5);
              //ellipse(frame, center, Size(faces[i].width*0.5, faces[i].height*0.5), 0,
0, 360, Scalar(255, 0, 255), 4, 8, 0);

              // Draw a rectangle
              rectangle(frame, faces[i], Scalar(255, 0, 0),4);

              // use cascade classifier to find the eye patterns in the i-th face
rectangle region
```

```
			detectEye(frame, faces[i]);
		}

		// End of pframe routine, check the number of detected eyes
		if (eyeTemplate1.size() > 0)
		{
			// Set the flag to proceed to pattern matching
			pframe = false;
			cout << "pframe done" << endl;
		}
		else
		{
			// No eye detected. repeat the pframe routine
			pframe = true;
		}

}

// function for detect eyes using cascade classifier
void detectEye(Mat &frame, Rect faceRect)
{
		// create a vector of Rects to store the recognized eyes for a single face
		std::vector<Rect> eyes;
		// create a Mat to store the gray version of the frame
		Mat frame_gray;
		// convert the colored frame to gray and store it in frame_gray
		cvtColor(frame, frame_gray, CV_BGR2GRAY);

		// define a region of interest (ROI) using the rectangle of the face
		Mat faceROI = frame_gray(faceRect);
		// display the face ROI
		imshow("ROI", faceROI);

		//Detect eyes using cascade classifier
		//detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor =
1.1, int minNeighbors = 3, int flags = 0,Size minSize = Size(), Size maxSize = Size())
		//image – Matrix of the type CV_8U containing an image where objects are detected
		//objects – Vector of rectangles where each rectangle contains the detected
object.
		//scaleFactor – Parameter specifying how much the image size is reduced at each
image scale.
		//minNeighbors – Parameter specifying how many neighbors each candidate rectangle
should have to retain it.
		//flags – Parameter with the same meaning for an old cascade as in the function
		//cvHaarDetectObjects. It is not used for a new cascade.
		//minSize – Minimum possible object size.Objects smaller than that are ignored.
		//maxSize – Maximum possible object size.Objects larger than that are ignored.
		eye_cascade.detectMultiScale(faceROI, eyes, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
Size(5, 5));

		if (eyes.size() > 1)
		{
			// Create the shifted eye rect
			Rect shiftedEyeRect1 = Rect(faceRect.x + eyes[0].x, faceRect.y + eyes[0].y,
eyes[0].width, eyes[0].height);
			Rect shiftedEyeRect2 = Rect(faceRect.x + eyes[1].x, faceRect.y + eyes[1].y,
eyes[1].width, eyes[1].height);
```

```
            // add the face rect and eye tamplate to the searchRegions
            searchRegion.push_back(faceRect);
            eyeRegion.push_back(eyes[0]);
            eyeTemplate1.push_back(frame_gray(shiftedEyeRect1));
            eyeTemplate2.push_back(frame_gray(shiftedEyeRect2));
      }

      //for each recognized eye, do the followings
      //for (int i = 0; i < eyes.size(); i++)
      //{
      //     // Create the shifted eye rect
      //     Rect shiftedEyeRect = Rect(faceRect.x + eyes[i].x, faceRect.y + eyes[i].y,
eyes[i].width, eyes[i].height);
      //     // Draw a rectangle
      //     rectangle(frame, shiftedEyeRect, Scalar(255, 255, 255), 2);
      //}

}
```