نشریه‌ی علمی شهر ایمن
JOURNAL OF RESILIENT CITY
(JRC)

ردگیری بلادرنگ ویدئویی اشیا با استفاده از الگوریتم Kernel Base Object Tracking اصلاح‌شده با قابلیت شناسایی دقیق‌تر در محصولات امنیتی و نظارتی

مجید شاکری*[1]

۱- گروه برق، واحد شهریار، دانشگاه آزاد اسلامی، شهریار، ایران

دریافت دست‌نوشته : ۱۳۹۹/۱۰/۱۲ پذیرش دست‌نوشته: ۱۴۰۰/۰۲/۱۳

واژگان کلیدی

بینایی ماشین

ردگیری بلادرنگ

ویدیویی الگوریتم

Kernel Base Object Tracking

الگوریتم *Normalized Cross Correlation*

چکیده

یکی از زمینه‌های کاربرد بینایی ماشین، ردیابی شیء متحرک می‌باشد. ردیابی شیء متحرک به صورت بلادرنگ فرآیندی است که طی آن موقعیت یک شیء متحرک در هر لحظه از زمان تعیین می‌شود. بدین منظور در این مقاله با یک رویکرد عملی که منجر به تولید محصول و با استفاده از کتابخانه *OpenCV* در محیط برنامه‌نویسی *Visual C++* می‌شود، تلاش شده است تا یک شیء را در فریم‌های متوالی و با استفاده از یک دوربین *Pan Tilt Zoom* که دوربینی با یک پلتفرم دو درجه آزادی است، به صورت بلادرنگ ردگیری شود. لذا با توجه به این که هدف از این تحقیق پیاده‌سازی فرمول‌های ریاضی و مطالب ارائه‌شده به کدهایی است که به راحتی قابل استفاده در محصولات نظارتی و امنیتی باشند به بررسی یک الگوریتم مبتنی بر هیستوگرام به نام *Kernel Base Object Tracking* پرداخته و عملکرد آن با الگوریتم پرکاربرد *Normalized Cross Correlation* که مبتنی بر ضریب همبستگی آماری می‌باشد، مقایسه شده است. و مشخص گردید که عملکرد الگوریتم *Kernel Base* برای نشان دادن شیء مورد تعقیب با محیط پس‌زمینه کاملاً متمایز بسیار بهتر از الگوریتم *Normalized* شده می‌باشد.

۱- پیش‌گفتار

پیاده‌سازی عملی و بلادرنگ الگوریتم‌های بینایی ماشین یکی از چالش‌برانگیزترین و با اهمیت‌ترین مباحث در این حوزه می‌باشد که گاهی اصل موضوع و حتی مباحث ریاضی آن را نیز تحت شعاع خود قرار می‌دهد. پیاده‌سازی الگوریتم‌های بینایی ماشین به صورت بلادرنگ از دو منظر بهینه‌سازی می‌گردد که اولی نرم‌افزاری و دومی سخت‌افزاری می‌باشد. در بعد نرم‌افزاری، بهینه‌سازی و به‌کارگیری الگوریتم‌های خلاقانه راهکاری اساسی است و در مقالات مختلف به این موضوع پرداخته می‌شود؛ ولی در بعد سخت‌افزاری، معماری سخت‌افزار و بهینگی آن برای پردازش سریع داده‌ها حائز اهمیت است که خود وابستگی زیادی به تعداد واحدهای محاسبه و منطق پردازشگر دارد. آنچه که در این مقاله ارزیابی شده است، بررسی و مقایسه‌ی دو الگوریتم پرکاربرد دنبال کردن یا همان *track* از منظر نرم‌افزاری و بهینگی هر چه بیش‌تر آنها و در نهایت مقایسه‌ی چگونگی عملکرد آنها به روی دنبال کردن اجسام مختلف می‌باشد. سؤالی که مطرح است کدام الگوریتم برای نشان دادن شیء مورد تعقیب با محیط پس‌زمینه‌ی کاملاً متمایز بسیار بهتر عمل می‌کند؟ هدف از ارائه‌ی این مقاله صرفاً رویکردی عمل‌گرایانه به یکی از مباحث پرکاربرد پردازش تصویر یعنی تعقیب اشیا در تصاویر ویدیویی می‌باشد که بتوان از آن در جهت تولید محصولات پرکاربرد نظارتی امنیتی استفاده نمود. بدین جهت سعی شده فرمول‌های ریاضی موجود در مقالات ارائه شده به

* نویسنده مسؤل: دانشگاه آزاد اسلامی شهریار، شهریار، ایران؛ رایانامه: **majid.sh@gmail.com**

کدهایی تبدیل گردند که به راحتی قابل استفاده در یک محصول نظارتی، امنیتی باشد.

۲- مبانی نظری پژوهش

امروزه حوزه‌های کار با تصویر بسیار وسیع است ولی عموماً محدوده های مورد توجه در چهار زمینه‌ی بهبود کیفیت ظاهری *(Enhancement)*، بازسازی تصاویر مختل شده *(Restoration)*، فشردگی و رمزگذاری تصویر *(Compression and Coding)* و درک تصویر ماشین *(Mashine)*، متمرکز می‌گردند.

در *Comaniciu, D. & et al, (2003)* یک رویکرد جدید نسبت به نمایش هدف و محلی سازی جزء مرکزی در ردیابی بصری اشیا ارائه شده است. بازنمایی‌های هدف مبتنی بر هیستوگرام ویژگی‌ها، توسط ماسک فضایی با یک کرنل ایزوتروپیک تنظیم می‌شود. در روش ردیابی ارائه شده، هم ادغام با فیلترهای متحرک و هم تکنیک‌های اتصال داده مورد بحث قرار گرفته‌است. همچنین بهره‌برداری از اطلاعات پس‌زمینه، ردیابی کالمن با استفاده از مدل‌های حرکتی و چهره‌ی مخاطب توصیف شده است.

مقاله‌ی *Elgammal, A. & et al, (2003)* استفاده از تبدیل گاوس فوری *(FGT)* به سرعت تکنیک‌های تخمین چگالی کرنل را مورد بررسی قرار داده است. تبدیل گاوس سریع می‌تواند برای بسیاری از مشکلات بینایی مفید باشد که در آن از تخمین چگالی کرنل استفاده می‌شود. در *Chang, C. & Ansari, R. (2005)* یک فیلتر ذرات جدید، که فیلتر ذرات کرنل *(KPF)* می‌باشد، برای دنبال کردن تصویر در توالی تصویر پیشنهاد شده است. در *KPF* از کرنل‌ها برای ارزیابی مداوم تابع چگالی پسین استفاده می‌شود. ذرات براساس اطلاعات شیب‌دار تخمین زده شده از تخمین چگالی کرنل پسین صورت می‌گیرند.

در یک سناریوی ردیابی، عادی نیست که اشیا با اشکال پیچیده‌ای را مشاهده کنید که مقیاس و جهت‌گیری آنها به طور دائمی به دلیل حرکت دوربین و جسم تغییر می‌کند. در مقاله *Yilmaz, A. (2007)*، یک روش ردیابی شیء براساس تغییر میانگین هسته‌ی نامتقارن ارائه شده که در آن مقیاس و جهت‌گیری هسته به صورت انطباقی بسته به مشاهدات در هر تکرار تغییر می‌کند. روش پیشنهادی، ردیابی متوسط تغییرات متداول را که در مختصات تصویر انجام می‌شود، از طریق شمول مقیاس و جهت‌گیری به‌عنوان ابعاد اضافی به کار می‌برد و به طور همزمان همه‌ی ناشناخته‌ها را در تعداد کمی از میانگین تکرار تغییر می‌دهد.

در *Venkatesh Babua, R. & Pérez, P. (2007)* یک روش جدید برای ردیابی اشیا با ترکیب دو ردیاب شناخته شده، همچنین تفاوت‌های مجموع مربعات (*SSD*) و ردیاب متوسط (*MS*) مبتنی بر رنگ، پیشنهاد شده است. در ترکیب پیشنهادی، دو ردیاب، با غلبه بر معایب‌شان، یکدیگر را تکمیل می‌کنند. تغییر مدل سریع در ردیاب *SSD* توسط ماژول ردیاب *MS* غلبه می‌شود، درحالیکه ناتوانی ردیاب *MS* در برخورد با جابجایی‌های بزرگ توسط ماژول *SSD* رفع می‌شود.

در مقاله‌ی *Porikli, F. , (2005)* یک الگوریتم ردیابی شیء برای ویدیو با فریم پایین ارائه شده که در آن اشیا حرکت سریع دارند. ردیابی متداول انتقال میانگین*(Mean-Shift)* در صورت جابه‌جایی یک شیء بزرگ است و مناطق آن بین فریم‌های متوالی هم‌پوشانی ندارند. راه حل این مشکل با استفاده از چند هسته مرکزی در مناطق حرکتی بالا ارائه شده است. علاوه بر آن، ویژگی‌های هم‌گرایی انتقال میانگین را با ادغام شباهت‌های پس زمینه و الگو، در مکانیزم به روز رسانی تکراری بهبود بخشیده است.

در *Jeyakara, J. & VenkateshBabu, R. (2008)* یک الگوریتم ردیابی ارائه شده است که نقاط ضعف ردیابی مبتنی بر هیستوگرام رنگ جهانی را برطرف می‌نماید. برای این منظور ردیابی بر اساس رنگ‌های قابل اعتماد، با جدا کردن جسم از پس‌زمینه آن، ترکیب شده است.

در مقاله‌ی *Wesley, E. S. & Hairong, Q. (2004)* الگوریتمی برای محاسبه‌ی سریع همبستگی متقابل نرمال (*NCC*) و کاربرد آن به مسئله‌ی تطبیق الگو ارائه شده است. با توجه به یک الگوی t که موقعیت آن در یک تصویر f مشخص می‌شود. ایده‌ی اساسی الگوریتم، نشان دادن الگویی است که برای آن همبستگی متقابل نرمال به عنوان مجموع توابع پایه مستطیلی محاسبه می‌شود.

در این مقاله با توجه به ماهیت آن که در رده‌ی درک تصویر توسط ماشین گنجانده می‌شود، دو روش مختلف در این

فرآیند با یکدیگر مقایسه می‌شود و در روش اصلی که عنوان مقاله نیز می‌باشد، اصلاحاتی لحاظ شده است.

کتابخانه *OpenCV* در این تحقیق ابزار اصلی ما در امر پیاده سازی‌ها می‌با شد. به جرئت می‌توان گفت این کتابخانه توانسته رقبای خود در امر پردازش تصویر نظیر جعبه‌ابزار پردازش تصویر نرم‌افزار *Matlab* را به راحتی پشت سر بگذارد؛ زیرا دامنه‌ی پوشش توابع آن به مراتب بیش‌تر از *Matlab* می‌باشد، ضمن این‌که کلیه‌ی توابع آن در زبان برنامه‌نویسی *C* به بهینه‌ترین شکل ممکنه طراحی گردیده است.

۳-مفهوم شناسی و متغیرهای تحقیق

۳-۱- بیان الگوریتم ***Normalized Cross Correlation***:

الگوریتم *Normalized Cross Correlation* یا همان ضریب همبستگی آماری با فرمول زیر به فرم گسسته بیان می‌گردد:

$$NCC = \sum_{k=1}^{N} \frac{(x_k - \mu_x)(y_k - \mu_y)}{\sigma_x \sigma_y}$$

در این فرمول $\boldsymbol{\mu_x}$ و $\boldsymbol{\mu_y}$ میانگین دو نمونه داده است که قرار است فی مابین آنها این ضریب همبستگی محاسبه گردد. همچنین $\boldsymbol{\sigma_x}$ و $\boldsymbol{\sigma_y}$ واریانس این دو نمونه داده و N نیز تعداد داده‌های موجود در دو نمونه می‌باشد. الگوریتم در فرم گسسته و دوبُعدی در حقیقت بیانگر شباهت آماری دو ماتریس عددی هم‌اندازه می‌باشد که یکی از ساده‌ترین ولی پرکاربردترین الگوریتم‌های پردازش تصویر در حوزه‌ی بینایی ماشین و تعقیب اجسام در فریم‌های متوالی ویدیویی می‌باشد. این شباهت آماری در تئوری آمار و احتمالات به نام ضریب همبستگی آماری شناخته می‌شود که هرچه این دو نمونه داده از حیث آماری به هم بیشتر شباهت داشته باشند این ضریب به یک نزدیک‌تر می‌باشد ولی در صورت عدم تطابق، این ضریب به صفر میل می‌کند. قبل از بررسی الگوریتم فوق باید اذعان داشت که در پیاده‌سازی این الگوریتم از توابع کتابخانه *OpenCV* استفاده گردیده است. هم‌اکنون به این مسئله می‌پردازیم که به چه شکل فرمول ریاضی فوق را پیاده‌سازی نماییم و باید توجه داشت که داده‌های ما در حقیقت پیکسل‌های فریم‌های تصویر هستند که در ادامه به تفصیل بیان گردیده است.

ابتدا هر فریم از کارت کپچر به صورت *RGB* خوانده می‌شود؛ سپس این فریم در لحظه t در حافظه ذخیره گردیده و به یک فریم خاکستری‌گونه (*Scale Gary*) تبدیل می‌گردد؛ سپس کاربر با استفاده از ماوس یک شیء را انتخاب می‌کند که با این کار ماتریس داده‌ای از پیکسل‌های شیء تحت تعقیب در حافظه‌ی *RAM* ذخیره می‌شود. سپس این ماتریس دو بعدی در فریم‌های بعدی برای محاسبه این ضریب همبستگی آماری به کار می‌رود، بدین شکل که در فریم‌های بعد از t با لغزاندن این ماتریس بر روی ماتریس بزرگ‌تر یعنی ماتریس دو بعدی خاکستری‌گونه، کل تصویر و محاسبه‌ی ضریب همبستگی در سطح زیر لغزش مکان جدید شیء محاسبه می‌گردد. به عبارت دیگر در مکانی که این ضریب نزیک‌تر به یک باشد، آن مکان، موقعیت جدید شیء در فریم کنونی می‌باشد و پنجره‌ی هدف به دور آن کشیده می‌شود.

نکته‌ی بعدی که در پیاده‌سازی این الگوریتم حائز اهمیت است، آن است که از آن‌جایی که الگوریتم فوق دارای بار محاسباتی زیادی برای پردازشگر می‌باشد، به جای لغزاندن ماتریس شیء بر روی کل فریم تصویر این ماتریس بر روی یک ناحیه *ROI* (*Region of Interset*) از فریم کلی لغزانده می‌شود، که این ناحیه *ROI* مکانش بر روی فریم تصویر موقعیت قبلی شیء و اندازه‌اش دو برابر یا سه برابر اندازه‌ی ماتریس شیء بسته به انتخاب برنامه‌نویس می‌باشد.

اگر بخواهیم به زبان ساده‌تر مسئله را بیان کنیم آن است که در فریمt یک ماتریس با مقادیر عددی پیکسل‌های شیء مد نظر کاربر برای تعقیب از فریم مذکور جدا شده و در حافظه نگهداری می‌شود؛ سپس در فریم بعدی یعنی $t+1$ این ماتریس شیء را بر روی ناحیه‌ی *ROI* بر روی فریم $t+1$ لغزانده و به ازای هر لغزش یک ضریب شباهت محاسبه می‌شود که مکانی که این ضریب حداکثر باشد در حقیقت موقعیت جدید شیء در فریم $t+1$ می‌باشد و همین فرآیند برای فریم‌های بعدی نیز تکرار می‌شود.

۳-۱-۱- کنترل *PtzCamera* با اســـتفاده از الگوریتم *Normalized Cross Correlation*:

هدف از این بخش آن است که همیشه شیء تحت تعقیب را در مرکز تصویر نگه داریم. این کار با استفاده از دوربینی محقق می‌شود که حرکتش توسط دو موتور در راستاهای افقی (*Pan*) و عمودی (*Tilt*) میسر گردد؛ اصطلاحاً دوربین دارای دو درجه آزادی باشد. به این نوع سامانه‌ها *PtzCamera* *(Pan Tilt Zoom Camera)* گفته می‌شود که با استفاده از یک الگوریتم ردیابی و محاسبه‌ی فاصله‌ی شیء تحت تعقیب از مرکز فریم در دو راستای عمودی و افقی و با یک تناسب ساده می‌توانیم فرامین لازم برای موتورها را برای جبران این فاصله با مرکز صادر نماییم و در حقیقت حرکت شیء را با دوربینی تعقیب نماییم که همواره در مرکز تصویر باقی بماند. در ادامه به بررسی برخی از ادوات برای پیاده سازی این بخش پروژه می‌پردازیم.

۳-۱-۲- دوربین *PtzCamera*:

دوربین مورد اســـتفاده در این پروژه باید از نوع *PTZ* باشـــد. این دوربین ها دارای پلتفرمی با دو درجه آزادی موتور برای حرکت در جهت چپ و راســـت و بالا و پایین می‌باشـــند. دوربین مورد اســـتفاده در این پروژه مـدل-*HCV* ۸۵۰□□□۱□۸ ســـاخت شـــرکت *HITRON SYSTEMS* و دارای پروتکل *Pelco-P* می‌باشد که در شکل ۱ نشان داده شده است. ویژگی این پروتکل این است که دوربین را در دو محور با استفاده از فرمان کنترلی سرعت، کنترل می‌نماید.

شکل۱: دوربین *dome* با قابلیت *pan & tilt*

۳-۱-۳- مبدل *RS232* به *RS485*

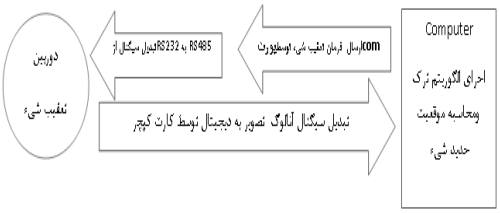
به دلیل این‌که سکوی دوربین‌های *PTZ* با سیگنال ۴۸۵ کار می‌کند، ما از یک مبدل *RS232* به *RS485* استفاده کرده‌ایم تا بتوان برای ردگیری شـــیء متحرک، فرامین مربوط به حرکت موتورها را از طریق پورت ســـریال به دوربین اعمال نمود. همچنین برای عدم نویزپذیری، از سیگنال‌های تفاضلی نظیر استاندارد ۴۸۵ استفاده می شود و برای انتقال سیگنال به فواصـــل دور می‌توان از آن اســـتفاده نمود. در این حالت علاوه بر یک زوجِ به‌هم‌تابیده برای دو ســـیگنال $+d$ و $-d$ از یک سیم زمین نیز استفاده می‌گردد.

۳-۱-۴- مبدل تصویر آنالوگ به دیجیتال

برای تبدیل ویدئو دریافتی از ســـکوی دوربین چرخان به یک تصـــویر دیجیتال مناســـب برای پردازش تصـــویر، از این قطعه استفاده نموده‌ایم. ویدئو استفاده‌شده از نوع آنالوگ با فرمت *PAL* (*Phase Alternated Line*) می‌باشد. در شکل ۲ این مبدّل نمایش داده شده است:

شکل ۲: مبدل تصویر آنالوگ به دیجیتال مورد استفاده در پروژه

در نهایت بلوک دیاگرام کلی پیاده سازی سخت‌افزاری پروژه به صورت زیر می‌باشد.



شکل ۳: بلوک دیاگرام کلی پروژه

۳-۲- بررسی الگوریتم *Kernel Base Object Tracking*

الگوریتم *Kernel Base Object Tracking* برای اولین بار توسط *Dorin Comaniciu* و *Peter Meer* ارائه گردید، که یک بیان ریاضی از تعقیب اشیا به صورت بلادرنگ در تصاویر ویدیویی می‌باشد و بر اساس ناحیه‌ی جسم مورد تعقیب عمل می‌کند و با تشکیل نمودار هیستوگرام وزن‌دار از ناحیه‌ی هدف مورد تعقیب در فریم‌های متوالی بر اساس تعیین جهت حرکت هدف و بر اساس مشتق تابع *kernel* که روی ناحیه‌ی هدف تشکیل می‌شود، عمل می‌نماید. در ادامه، قبل از بیان الگوریتم ابتدا یک سری مفاهیم اولیه را تعریف می‌نماییم که در جهت فهم بهتر الگوریتم بسیار مؤثر است. این مفاهیم، در حقیقت مفاهیمی می‌باشند که در مقالات آمده است و در قسمت مراجع به صورت ریاضی بیان گردیده که ما در این طرح به آنها رنگ و بوی پردازش تصویر داده‌ایم.

۳-۲-۱- نمودار هیستوگرام

نمودار هیستوگرام را می‌توانیم برای هر تصویر یا فریم ویدیوئی تشکیل دهیم که بیان می‌کند احتمال هر رنگ در تصاویر رنگی یا هر شدت نوری در تصاویر خاکستری‌گونه چه مقدار می‌باشد. برای درک بهتر این موضوع، به مثال زیر توجه کنید. فرض کنیم در یک تصویر ساده حداکثر m سطح رنگ داریم و ابعاد تصویر برابر با w در عرض و h در ارتفاع می‌باشد. حال، شروع به شمارش پیکسل‌ها در هر کدام از m سطح موجود می‌نماییم که به طور مثال شمارش را انجام داده‌ایم.

سطح رنگ	۰	۱	۲	....	*m-1*
تعداد	*N0*	*N1*	*N2*	....	*Nm-۱*

شکل ۶- نمایش تعداد هر سطر

شکل ۶ تعداد هر سطح رنگ را نمایش می‌دهد، برای این که احتمال هر کدام از سطوح را در فریم مربوطه به دست آوریم، باید تعداد هر کدام از سطوح را تقسیم به تعداد کل پیکسل‌های فریم نماییم. در نتیجه، شکل ۷ حاصل می‌شود.

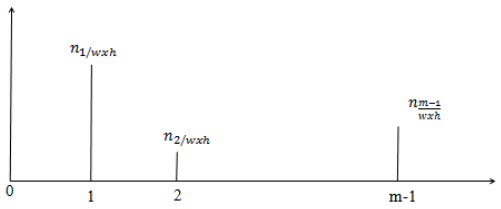
سطح رنگ	تعداد
۰	N0/(w*h)
1	N1/(w*h)
2	N2/(w*h)
...	...
m-1	Nm-1/(w*h)

شکل ۷- نمایش تعداد در سطح رنگ

که رابطه‌ی زیر متناسب با جدول بالا بدیهی می‌باشد:

$$\frac{n_o}{w_{xh}} + \frac{n_1}{w_{xh}} + \cdots \ \ldots + \frac{n_{m-1}}{w_{xh}} = 1$$

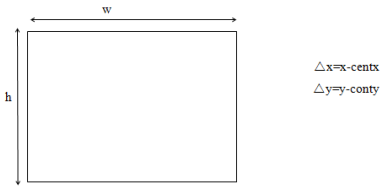
حال با توجه به جدول احتمال هر سطح رنگی در فریم می‌توانیم یک نمودار میله‌ای را که همان نمودار هیستوگرام است، از جدول احتمالات به دست آوریم. (شکل ۸)



شکل ۸- نمودار هیستوگرام

۳-۲-۲- نمودار هیستوگرام وزن‌دار

در این نمودار، دیگر همه‌ی پیکسل‌ها در هر فریم وزن ثابتی در نمودار هیستوگرام به خود نمی‌گیرد، بلکه ابتدا سطح رنگ پیکسل استخراج شده سپس به جای اضافه نمودن یک واحد به نمودار هیستوگرام، یک وزن متناسب با جایگاه پیکسل در فریم به نمودار هیستوگرام اضافه می‌گردد. راهکار تشکیل این وزن در ادامه آمده است.



شکل ۹- نمودار هیستوگرام وزن‌دار

در شـــکل ۹ عرض و ارتفاع تصـــویر به ترتیب w و h می‌باشد که پیکسل در موقعیت (y و x) دارای سطح رنگی برابر با $Pix\ (x, y)$ می‌باشد و مختصات مرکز تصویر برابر با $(centx\ \ centy)$ می‌باشد و فاصله‌ی پیکسل مد نظر از مرکز تصویر در راستای محور عرضی برابر با Δx و در راستای محور ارتفاع برابر Δy می‌باشد. حال این فواصل را نسبت به عرض و ارتفاع تصویر نرمال می‌نماییم و این بدان معناست که Δx را بر $w/۲$ و Δy را بر $h/۲$ تقسیم می‌نماییم:

$$Xnormal = \frac{x-contx}{x/z} = \frac{\triangle x}{w/z}$$
$$Xnormal = \frac{y-conty}{h/z} = \frac{\triangle y}{h/z}$$

در ادامه، فاصله‌ی نرمال شده پیکسل مورد نظر تا مرکز تصویر متناسب با فرمول زیر به دست می‌آید.

$$dist = \sqrt{x2normal + y2normal}$$

که بدیهی است فاصله‌ی نرمال شده‌ی بالا بین صفر و یک می‌باشد؛ یعنی هر چه پیکسل به مرکز تصویر نزدیک‌تر باشد، مقدارش به صفر نزدیک‌تر می‌شود و بر عکس هر چه به حواشی تصویر نزدیک‌تر گردد، به یک نزدیک‌تر می‌گردد. پس از بیان مفاهیم بالا، زمان آن رسیده که نحوه‌ی تشکیل نمودار هیستوگرام وزن‌دار را تشریح نماییم که برای این منظور یک قاعده در نظر می‌گیریم و این قاعده‌ی کلی آن اســـت که با اســـتفاده از یک تابع *kernel* طوری عمل نماییم که پیکسل‌هایی که به مرکز تصویر نزدیک‌تر باشند، وزن بالاتر و آنهایی که از مرکز دورتر باشـــند، وزن کم‌تری می‌گیرند؛ پس در نتیجه با مقدار dist به دســـت آمده از فرمول‌هایی بالا می توانیم به تابع *kernel* زیر را رامحاسبه کنیم:

$$kernel = ۱ - dist$$

در فرمول بالا مشـــاهده می‌کنیم که قاعده‌ی مدنظر رعایت شده است؛ یعنی تابع *kernel* متناسب با مقدار *dist* برای پیکسل‌هایی که نزدیک‌تر به مرکز هستند، وزن بزرگ‌تر و برای آنهایی که دورتر از مرکز تصـــویر هســـتند، وزن کوچک‌تری تولید می‌کنند. حال، این وزن به میله‌ی مربوط مســـاوی با ســـطح رنگ پیکســـل اضـــافه می‌گردد که روند الگوریتم فوق برای تک تک پیکسل‌های تصویر اجرا می‌شود. هر کدام از میله های نمودار هیســـتوگرام متناســـب با جایگاه‌های پیکســـل‌های مربوطه وزن‌های متناســـب را به صـــورت انباشـــتگی می‌گیرند و در پایان برای آن‌که نمودار هیســـتوگرام رنگ و بوی احتمال به خود بگیرد، مجموع تمام وزن‌های پیکســـل‌ها را به دســـت آورده و تک‌تک وزن‌های موجود بر روی هر میله هیستوگرام را تقسیم بر این مجموع وزن‌ها می‌نماییم. در بدنه‌ی اصـلی برنامه، تابع *evalHist* که الگوریتم بالا را محاسبه می‌نماید، بیان گردیده است.

۳-۲-۳- بیان کلی الگوریتم *Centroid*

الگوریتم *Centroid* یا الگوریتم محاســـبه‌ی مرکز ثقل گروهی، مشـخص از پیکســـل‌های تصـــویر می‌باشـد. به طور مثال، فرض کنیم می‌خواهیم مرکز ثقل پیکســـل‌هایی از یک تصـــویر را در یک محوطه‌ی مســـتطیلی محاســـبه کنیم که شاخصه‌ی این پیکسل‌ها شدت رنگ آنها از یک مقدار آستانه مانند *th* بالاتر است. حال می‌توانیم با یک پیمایش ساده در عرض و ارتفاعِ این ناحیه‌ی مســـتطیلی پیکســـل‌هایی را که اندازه‌ی آنها از *th* بیش‌تر باشد، برچسب یک و آنهایی را که کم‌تر یا مساوی با *th* باشند، برچسب صفر بگذاریم. سپس با یک میانگین‌گیری در راستای x و y پیکسل‌های با برچسب یک موقعیت پیکسل مرکز ثقل به دست می‌آید.

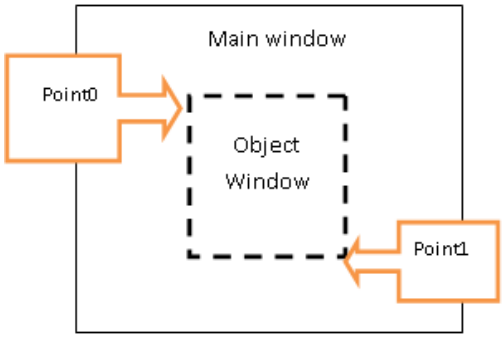
$$X_{centroid} = \frac{x_{.}+x_{۱}+\cdots+x_{n-۱}}{n}$$
$$Y_{centroid} = \frac{yx_{.}+yx_{۱}+\cdots+yx_{n-۱}}{n}$$

در فرمول بالا، عدد n بیان‌کننده‌ی تعداد پیک سل‌هایی ا ست که دارای مقادیر رنگ بالاتر از مقدار آ ستانه th بوده‌اند که بر این اســـاس با میانگیری در دو محور x و y از جایگاه این پیکســـل‌ها می‌توان جایگاه پیکســـل مرکز ثقل را در این ناحیه‌ی مستطیلی به دست آورد.

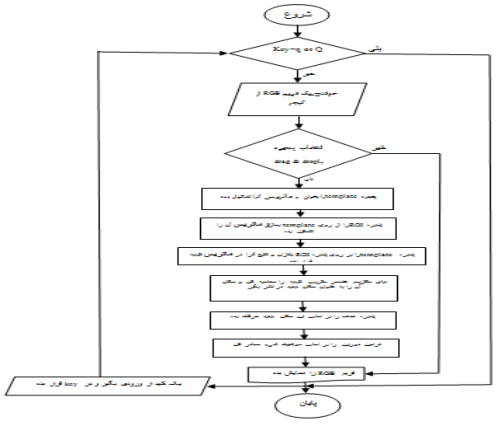
۴- روش شناسی

۴-۱- نحوه‌ی عملکرد محصـــول پیشـــنـ هادی و فلوچارت کلی:

در این پروژه، ابتدا قســـمتی از تصـــویر یک شـــیء (*Sub Image*) که در م یدان د ید دوربین *PTZ* قرار دارد، به وســـیله‌ی ماوس و به وســـیله‌ی عمل *drag&drop* انتخاب می شود؛ (شکل۴) سپس شیء به حرکت درمی‌آید و سعی می‌کند تا خود را از میدان دید دوربین دور ســـازد. در این حالت، نرم‌افزار با ا ستفاده از الگوریتم *normalized cross correlation* ابتدا اطلا عات تصـــویر دیجی تال را که به صـــورت رنگی اســـت، به خاکســـتری‌گونه تبدیل و ســـپس پردازش می‌کند و فرامین ســـر یال با پروتکل *Pelco_P* جهت فر مان دادن به دو اســـتپر موتور مربوط به پلتفرم دوربین جهت حرکت دوربین به طرفین و بالا و پایین صادر و به وســـیله پورت ســـریال به مبدل *RS232* به *RS485* منتقل و تبدیل کرده و در انتها به دوربین اعمال می‌نماید و دوربین به سمت شیء حرکت کرده و مانع خروج شیء از دید خود می شود و به این ترتیب شی پویا ردیابی می‌گردد که فلوچارت این کار در شکل ۵ دیده می‌شود.



شکل ۴: ایجاد یک *sub image* به وسیله عمل drag & drop



شکل ۵- فلوچارت کلی تعقیب با استفاده از الگوریتم *Normalized Cross Correlation*

۴-۲- بیان سورس برنامه

همان‌طور که در روند کلی این پروژه تا به حال مشاهده گردیده است، این گزارش یک گزارش عملی از یک فرمول ریاضی تا یک محصول اولیه‌ی نظارتی - امنیتی است که در بسیاری از پروژه‌های آتی مرتبط با صنعت قابل استناد می‌باشد. در ادامه با بیان سورس آن، این فرآیند را کامل می‌نماییم.

پیش از بیان سورس، به بعضی از تعاریف زیر که به فهم هر چه بهتر سورس برنامه و انطباق آن با مفاهیم ذکر شده در قسمت‌های قبلی کمک می‌کند، توجه فرمایید:

- *Mouse Handler*: تعریف تابع عملیات ماوس بروی پنجره نمایش ویدئو.
- *Check Border*: تابعی برای چک کردن عدم خروج از حدود ماتریس فریم.
- *CHitron Contro lClass*: کلاس کنترلی دوربین *PTZ*.
- *Get Rect SubPix*: یکی از توابع پرکاربرد *OpenCV* برای جدا نمودن قطعه‌ای کوچک‌تر از ماتریس بزرگ‌تر می‌باشد که از آن برای جدا نمودن ماتریس‌های *ObjectMat* (ماتریس عددی از پیکسل‌های شیء تحت تعقیب) و ماتریس *ROIMat* (ماتریس ناحیه جستجو برای ماتریس *ObjectMat*) از *GrayFrame* (ماتریس خاکستری‌گونه فریم تصویر) به کار می‌رود.

- *Match template*: قلب برنامه این تابع است؛ تابعی از کتابخانه *OpenCV* که وظیفه‌ی لغزاندن ماتریس *ObjectMat* بر روی ماتریس *ROIMat* را به عهده دارد و با هر لغزش یک ضریب همبستگی آماری محاسبه می‌نماید؛ و کلیه‌ی این ضرایب در ماتریسی به نام *ResultMat* ذخیره می‌شود. این تابع همان‌طوری که از اسمش مشخص است، برای ارزیابی شباهت دو نمونه‌ی داده‌ای یا همان *template* به کار می‌رود و قادر است با الگوریتم‌های متفاوتی این کار را انجام دهد که با مقداردهی یکی از آرگومان‌های آن با مقدار *CV_TM_CCOEFF_NORMED* در حقیقت آن را ملزم به اجرا با الگوریتم *Normalized Cross Correlation* نموده‌ایم.

minMaxLoc: این تابع یک ماتریس را دریافت کرده و در آن مکان و مقادیر مینیمم و ماکزیمم را پیدا می‌نماید که در حقیقت مکان ماکزیمم مقدار در ماتریس *ResultMat* محاسبه‌شده در بالا، مکان جدید شیء را مشخص می‌نماید.

```
#include"stdafx. h"
#include"VideoObjectTracking. h"
#ifdef _DEBUG
#definenew DEBUG_NEW
#endif
// The one and only application object
CWinApp theApp;
usingnamespace std;
```

تعریف نقطه *point0* و *point1* و فونت در محیط *OpenCV*

```
CvFont font;
CvPoint point0;
CvPoint point1;
bool drag = 0;
bool movement = 0;
bool trackenable = 0;
bool selecttarget = 0;
```

تعریف تابع ماکرویی تعیین مینیمم و ماکزیمم

```
(a, b) (((a) < (b))? (a): (b)) min #define
(a, b) (((a) > (b))? (a): (b)) max #define
int VideoWidth;
int VideoHeight;
```

تابع چک کردن حاشیه های تصویر

```
void CheckBorder (int &validRangeLeft,
&validRangeTop,  int&validRangeRight, int
int&validRangeDown)
{
validRangeLeft =max(0 ,validRangeLeft);
validRangeTop = max(0 ,validRangeTop );
validRangeRight =min(VideoWidth-1 ,
validRangeRight)
validRangeDown = min(VideoHeight-1 ,
validRangeDown);
}
```

انتخاب*object window* به وسیله عمل *Drag & Drap*

```
(intevent, int x, int y,  mouseHandler void Static
int flags, void *param)
{
/* user press left button */
(event == CV_EVENT_LBUTTONDOWN  If
&& !drag)
 {
point0 = cvPoint(x, y);
drag = 1;
movement=0;
 }
/* user drag the mouse */
If (event == CV_EVENT_MOUSEMOVE &&
drag)
 {
point1 = cvPoint(x, y);
movement =1;
 }
/* user release left button */
If (event == CV_EVENT_LBUTTONUP &&
drag)
 {
drag = 0;
movement =0;
trackenable = 1;
selecttarget =0;
 }
If (event ==
CV_EVENT_RBUTTONUP)
 {
drag = 0;
movement =0;
trackenable = 0;
selecttarget =0;
 }
}
// TODO: code your application's behavior here
```

کلاس کنترلی دوربین *PTZ*

```
CHitronControlClass CameraConrolObj;
```

تعریف دریافت تصویر رنگی از کپچر و ایجاد ماتریس برای تصویر رنگی

```
cv::VideoCapture Cap(0);
cv::Mat RGBFrame;
```

```
cv::Mat GrayFrame;
```

ایجاد پنجره‌ی نمایش تصویر

```
cv::namedWindow("Correlation Video
Window");
```

فراخوانی تابع مدیریت موس برروی پنجره

```
v::setMouseCallback("Correlation Video
Window", mouseHandler, NULL);
CvFont font;
int hscale=1. 0;
int vscale=1. 0;
int LineWidth=1;
```

تعیین فونت خط به کار رفته درروی تصویر رنگی

```
cvInitFont(&font,CV_FONT_HERSHEY_SIMPL
EX|CV_FONT_ITALIC,hscale,vscale,0,LineWidt
h);
int ObjectHeight;
int ObjectWidth;
int ObjectCentX;
        int ObjectCentY;
cv::Mat ObjectMat;
cv::Mat ROIMat;
cv::Ma ResultMat;
int ROI_WINDOW_COEFICENT =2;
intObject_window_left,Object_window_
right,Object_window_top,Object_window_do
wn;
intROI_window_left,ROI_window_right
,ROI_window_top,ROI_window_down;
int VideoCenterX;
int VideoCenterY;
int key=0;
```

یک حلقه‌ی بی‌نهایت تا زمانی که کلید q یا Q زده شود، اعمال داخل حلقه اجرا می‌گردد.

```
while (key!='q'&&key!='Q')
 {
```

دریافت *RGB frame* از کارت کپچر

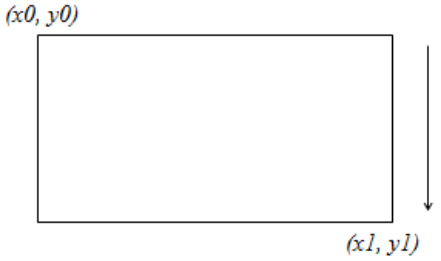
```
Cap >> RGBFrame;
VideoHeight = RGBFrame.
rows;
VideoWidth = RGBFrame.
cols;
```

تابع تبدیل *RGB frame* به *gray level*

```
cv::cvtColor(RGBFrame,GrayFrame,CV_RGB2
GRAY);
(drag&&movement) if
cv::rectangle(RGBFrame,point0,point1,
CV_RGB(255,255,0),3,8,0);
if (trackenable)
{ if (!selecttarget)
{ int temp;
 If (point0. y>point1. y)
{temp = point0. y;
 point0. y = point1. y;
 point1. y = temp;
        }
ObjectWidth = (point1. x-
point0. x);
ObjectHeight= (point1. y-
point0. y);
ObjectCentX=(point1. x+point0. x)/2;
ObjectCentY = (point1. y+point0. y)/2;
ObjectCentY=ROI_window_top+max_loc.
y+ObjectHeight/2;
CameraConrolObj. tiltspeed= -
((float)(ObjectCentY-
VideoCenterY)/(float)(VideoCenterY))*63;
Object_window_top=ObjectCentY-
ObjectHeight/2;
CheckBorder(Object_window_left,Object_windo
w_right,Object_window_top
   ,Object_window_down);
else{ if ((key=='8')){
CameraConrolObj. tiltspeed = -10;
CameraConrolObj. panspeed =0;
CameraConrolObj.
CameraControlFuc();
 }
If ((key=='2')) {
CameraConrolObj. tiltspeed = 10;
CameraConrolObj. panspeed =0;
CameraConrolObj.
CameraControlFuc();
}
If ((key=='5')) {
CameraConrolObj. tiltspeed = 0;
CameraConrolObj. panspeed =0;
CameraConrolObj. CameraControlFuc();
}
If ((key=='4')) {
CameraConrolObj. tiltspeed = 0;
CameraConrolObj. panspeed =-10;
CameraConrolObj.
CameraControlFuc();
}
If ((key=='6')) {
CameraConrolObj. tiltspeed = 0;
CameraConrolObj. panspeed =10;
CameraConrolObj.
CameraControlFuc();
}
        }
```

```
float  qu[NUMBINS];
float  pu[NUMBINS];
float*  deriv_kernel_array;
float*  weight_array;
void  evalHist(float* Ptr);
inlinefloat kernelFunc(float);
inlinefloat deriveKernelFunc(float);
void  updateWeight();
void  computeDisplacement(int& dy,int& dx);
void  findNewTargetPos();
float  bhattacharyyaCoe();
void CheckBorder(int&validRangeLeft,
int&validRangeRight,
int&validRangeTop,int&validRangeDown);
KernelBaseTargetTracking(void);
void GetFrame(cv::Mat rgbFrame);
void GetTrackObject(CvPoint point0,CvPoint
point1);
void Track(cv::Mat rgbFrame);
};
```

شروع الگوریتم در حقیقت با اجرای فاز اول صورت می‌پذیرد که در آن یک قطعه از تصویر که همان هدف است، انتخاب می‌شود؛ ولی در این الگوریتم چیزی که از این هدف در حافظه ذخیره می‌شود، نمودار هیستوگرام وزن‌دار آن قطعه می‌باشد که با استفاده از تابع *evalHist* محاسبه می‌شود. تابع ذکر شده در ادامه *GetTrackObject* می‌باشد که هنگام انتخاب هدف تحت تعقیب توسط ماوس و عمل *drag & drop* فراخوانی می‌شود. این تابع چهار آرگومان ورودی دارد که به ترتیب *x0* و *y0* گوشه سمت چپ و بالایی مستطیل و *x1* و *y1* گوشه سمت راست و پایینی این مستطیل انتخابی توسط کاربر می‌باشد. (شکل ۱۰)



شکل ۱۰- مستطیل انتخابی کاربر

```
cv::putText(RGBFrame,"Correlation",cvPoint(5
۰,۳۰),۲,۱. ۱,cvScalar(0,0,0),1);

key = cv::waitKey(40);
 }
 }
}
else {
// TODO: change error code to suit your needs
T_t_printf(("FatalError:GetModuleHandleFaile
d\n"));
nRetCode = 1;
}
return nRetCode;
```

۴-۳- بیان کلی الگوریتم *Kernel Base Object Trading*

این الگوریتم نیز مانند الگوریتم *Normalized cross correlation* به دو فاز تقسیم‌بندی می‌شود.

فاز اول، انتخاب یک قطعه از فریم کنونی *t*، توسط کاربر و با استفاده از موس و *drag & drop* و نگه داشتن این قطعه در حافظه جهت تعقیب در فریم‌های بعدی. فاز دوم، انطباق قطعه‌ی به دست آمده در فاز قبلی با قطعه‌ی دارای بیش‌ترین شباهت در فریم‌های بعد از فریم*t*، یعنی *(t + 1 , t+ 2 , ... t + n)*. با توجه به دو فاز بالا هر کدام از آنها را متناسب با الگوریتم *kernel* طراحی می‌نماییم.

```
class KernelBaseTargetTracking
{
private:
int VideoWidth;
int VideoHeight;
int ObjectCenterX;
int ObjectCenterY;
int ObjectHeight;
int ObjectWidth;
int object_half_x ;
int object_half_y ;
int obj_window_left;
int obj_window_top;
int obj_window_right;
    int obj_window_down;
cv::Mat RGBFrame;
      cv::Mat GrayFrame;
```

تابع *GetTrackObject*

```
void
KernelBaseTargetTracking::GetTrackObject(int
x0,int y0, int x1,int y1)
{
evalHist (&qu[0],x0,y0,x1,y1);
}
```

در تابع*GetTrackObject*، تابع *evalHist* صدا زده می‌شود که این تابع نیز با آرگومان‌های*qu* و دو گوشه‌ی مستطیل صدا زده می‌شود که در آن *qu* نام بافری است از نوع *float* که تعداد سلول‌هایش برابر تعداد رنگ‌های به‌کاررفته در فریم‌های تصویر است که با متغیر *NUMBINS* بیان شده است که هیستوگرام وزن‌دار هدف مطلوب انتخابی توسط کاربر در آن محاسبه و در حافظه نگهداری می‌شود. در ادامه تابعی که در برنامه اصلی الگوریتم *kernel base object tracking* اقدام به محاسبه این هیستوگرام می‌نماید بیان می‌کنیم.

تابع *evalHist* با دریافت آرگومانی از نوع بافر با نام *Hist* و از نوع *float* اقدام به محاسبه هیستوگرام وزن‌دار برای یک ناحیه از تصویر می‌کند که در پایین، برنامه مربوط به محاسبه هیستوگرام وزن‌دار را مشاهده می‌کنیم.

```
void KernelBaseTargetTracking::Evalhist(float*
Hist,int x0,int y0,int x1,int y1)
{
int halfw=(x1-x0)/2;
int halfh=(y1-y0)/2;
intObjCentX =(x1+x0)/2;
intObjCentY =(y1+y0)/2;
        float dx,dy,dist,norm_dx,norm_dy;
memset(&Hist[0],0,NUMBINS*sizeof(float));
float total = 0;
for (int y =y0; y <y1; y++)
{
for (int x = x0; x <x1 ; x++)
{
dx = (float)(x-ObjCentX);
dy = (float)(y-ObjCentY);
norm_dx = (dx/ (float)(halfw));
norm_dy = (dy/ (float)(halfh));
dist = sqrt(norm_dx * norm_dx + norm_dy*
norm_dy);
uchar pixel = GrayFrame. at<uchar>(x, y);
if (dist<1. 0){
Hist[pixel] += kernel_func(dist);
deriv_kernel_array[y*VideoWidth+x]=
deriv_kernel_func(dist);
total += Hist[pixel];
}
 }
        }
for (int i = 0; i < NUMBINS; i++)Hist[i]= total;
}
```

در برنامه‌ی بالا مشاهده می‌کنیم که دو حلقه *for* تو در تو پیکسل‌های تصویر را پیمایش می‌کنند، ولی باید مد نظر داشته باشیم که این پیمایش در یک مستطیل اطراف هدف در فریم اصلی می‌باشد؛ پس حدود پیمایش پنجره (*y0 , x0*) گوشه سمت چپ بالا پنجره و (*y1 , x1*) گوشه‌ی مقابلش یعنی سمت راست و پایین آن می‌باشد که با توجه به این مقادیر، *ObjCentX* و *ObjCentY* مرکز مستطیل هدف و همچنین مقادیر *halfw* و *halfh* به ترتیب برابر با نصف عرض و ارتفاع تصویر محاسبه می‌گردد.

فاز دوم الگوریتم *kernel Base object tracking*، فراخوانی تابع *TrackObject* می‌باشد که برای فریم‌های بعد از فریم *t* بیان می‌گردد. تابع *TrackObject* موقعیت جدید هدف را در هر فریم محاسبه نموده و یک مستطیل به دور هدف در هر فریم می‌کشد و فریم را نمایش می‌دهد. همان‌طور که در متن تابع در ادامه می‌بینیم، تابعی که در داخل *TrackObject* فراخوانی می‌شود *findNewTargetPos* می‌باشد که این تابع *computeDisplacement* را صدا می‌زند و تابع *computeDisplacement* نیز تابع *updateWeight*را صدا می‌زند و در آخر تابع *updateWeight* نیز تابع *evalHist* را صدا می‌زند. روند بالا نشان می‌دهد که تابع *evalHist* باید برای تمام فریم‌ها صدا زده شود یعنی در هر فریم باید نمودار هیستوگرام وزن‌دار مجدداً محاسبه گردد. به متن تابع *updateWeight* توجه فرمایید.

```
void KernelBaseTargetTracking:updateWeight()
{
evalhist(pu, ObjectCenterX - object_half_x,
  ObjectCenterY - object_half_y,
```

```
  ObjectCenterX + object_half_x,
  ObjectCenterY + object_half_y);
float weight[NUMBINS];
for (int i = 0; i < NUMBINS; i++)
{
weight[i] = sqrt(qu[i]/pu[i]);
else
weight[i] = 0;
}
obj_window_left=max(0,ObjectCenterX-
object_half_x);
obj_window_top=max(0,ObjectCenterY-
object_half_y);
obj_window_down=min(VideoHeight-1 ,
ObjectCenterY + object_half_y);
for(int
y=obj_window_top;y<=obj_window_down;
y++)
for(int
x=obj_window_left;x<=obj_window_right ;
x++)
weight_array[y*VideoWidth+x]=
weight[GrayFrame. data[y*VideoWidth+x]];
}
```

همان‌طور که می‌بینید داخل تابع *updateWeight* تابع *evalHist* با آرگومان *pu* صدا زده شده است که نمودار هیستوگرام وزن‌دار را در هر فریم در موقعیت جدید هدف محاسبه می‌کند.

پس آرگومان‌های بعدی تابع متناسب با این موقعیت جدید مقداردهی می‌شود و بعد از محاسبه هیستوگرام وزن‌دار جدید باید تک‌تک میله‌های هیستوگرام وزن‌دار اولیه یا همان *qu* تقسیم بر میله‌های هیستوگرام جدید *pu* گردد و پس از آن جذر گرفته شود و حاصل آن آرایه‌ای می‌شود با نام *weights* که همان وزن‌های مورد نیاز برای تعقیب هدف یعنی به ازای هر رنگ یک وزن به دست می‌آید. مثلاً اگر *weights* یعنی وزن رنگ صفر برابر ۰. ۷ می‌شود و *weights* یعنی وزن رنگ ۱۵ برابر با ۰. ۱۴ می‌باشد، این وزن‌ها را جایگزین مقدار رنگ‌ها در مستطیل هدف می‌نماییم که در تابع فوق با دو حلقه *for* تو در تو قابل تشخیص می‌باشد.

تابع *updateWeights* داخل تابع *compute Displacement* صدا زده می‌شود که این تابع مقدار جابه‌جایی هدف را نسبت به موقعیت قبلش محاسبه می‌کند این تابع از یک نوع الگوریتم *Centroid* خاص متناسب با مشتق تابع *kernel* روی ناحیه مستطیلی شکل استفاده می‌کند.

```
void
KernelBaseTargetTracking::computeDisplaceme
nt(int& dy,int& dx)
{
updateWeight();
float weight_sum = 0;
float x_sum = 0;
float y_sum = 0;
float temp;
for(int
y=obj_window_top;y<=obj_window_down;
y++)
for(int
x=obj_window_left;x<=obj_window_right;
x++)
{
temp=
weight_array[y*VideoWidth+x]*deriv_kernel_a
rray[y*VideoWidth+x];
weight_sum += temp;
x_sum += (x-ObjectCenterX)*temp;
y_sum += (y-ObjectCenterY)*temp;
}
dx = (int)floor(x_sum/weight_sum);
dy = (int)floor(y_sum/weight_sum);
}
```

در این تابع، ابتدا تابع *updateWeights* صدا زده می‌شود و مقادیر وزن‌های هر رنگ به جای مقدار آنها در ناحیه‌ی کنونی مستطیلی هدف قرار می‌گیرد؛ سپس در دو حلقه‌ی for تو در تو بر روی مستطیل هدف پیمایش شده و حاصل‌ضرب هر کدام از مقادیر وزنی پیکسل‌ها در مشتق تابع *kernel* که هنگام *evalHist* محاسبه گردید و برابر با یک است، ضرب می‌شود؛ بعد از آن، این حاصل‌ضرب در اختلاف فاصله‌ی هر پیکسل با مرکز ناحیه‌ی مستطیلی ضرب می‌گردد و پس از برنامه به صورت شیءگرا و در قالب کلاس *Kernel Base Target*

Tracking که نحوه‌ی تعریف آن، متغیرها و توابع عضو آن در زیر آمده است، می‌باشد.

اتمام حلقه‌های *for*، تقسیم بر مجموع وزن‌ها می‌گردد که بدین‌سان الگوریتم *Centroid* متناسب با تابع *kernel* محاسبه می‌گردد. اگر بخواهیم نسبت به مطلب بالا یک حس شهودی پیدا کنیم، باید اذعان داشت وقتی ما ناحیه‌ی مستطیلی‌شکل هدف را با یک تابع *kernel* بیان می‌کنیم باید در فریم‌های متوالی با مشتق گرفتن از این تابع *kernel* و ضرب آن در وزن‌های ناحیه‌ی مستطیلی‌شکل و به‌روزرسانی مرکز ثقل هدف، در حقیقت اقدام به تعقیب هدف محصورشده در ناحیه‌ی مستطیلی نماییم. خروجی تابع *computeDisplacement* مقادیر *dx* و *dy* می‌باشند که مقدار جابه‌جایی مرکز هدف هستند. تابع *computeDisplacement* داخل تابع نهایی *findNowTargetPos* صدا زده می‌شود که در ادامه بدنه‌ی تابع *findNewTargetPos* را مشاهده می‌نمایید.

```
void
KernelBaseTargetTracking::findNewTargetPos()
{
int loopcount = 0;
int dx=0,dy=0;
int pdx=0,pdy=0;
int maxindex;
while(1)
{
pdx = dx;
pdy = dy;
computedisplacement(dy,dx);
ObjectCenterX += dx;
ObjectCenterY += dy;
loopcount++;
if (((dx==0)&&(dy==0)) || (loopcount>20) ||
((pdx+dx==0) && (pdy+dy==0))) break;
}
obj_window_left=ObjectCenterX-ObjectWidth;
obj_window_top=ObjectCenterY-
ObjectHeight;
obj_window_right=ObjectCenterX+
ObjectWidth;
obj_window_down=ObjectCenterY+
ObjectHeight;
}
```

همان‌طور که مشاهده می‌کنید، این تابع از یک حلقه‌ی *while* تشکیل شده است که در داخل آن تابع *computeDisplacement* صدا زده می‌شود و این حلقه با توجه به مقادیر خروجی *dx* , *dy* تصمیم‌گیری می‌کند که آیا در فریم جاری به کارش که همان به‌روزرسانی موقعیت هدف است، ادامه دهد یا خیر. در حقیقت اگر مقادیر *dx* , *dy* یا همان مقدار جابه‌جایی برابر با صفر بود، به منزله‌ی آن است که در فریم جاری موقعیت جدید هدف به طور دقیق محاسبه گردیده است و پس از آن باید مستطیل به سمت هدف در فریم جاری کشیده شود تا کاربر بتواند به‌روزرسانی موقعیت هدف را مشاهده نماید و اگر لازم باشد، متناسب با موقعیت جدید هدف فرامین کنترلی به موتورهای دوربین اعمال گردد تا هدف را مرکز تصویر نگه دارد.

۵- یافته‌های تحقیق: انجام تست‌های عملی

در این قسمت هر دو الگوریتم را در سه مورد متفاوت مورد ارزیابی قرار می دهیم و نتایج را بررسی می‌نماییم.

۵-۱- تست اول: تعقیب عنوان کتاب گرافیک رایانه‌ای

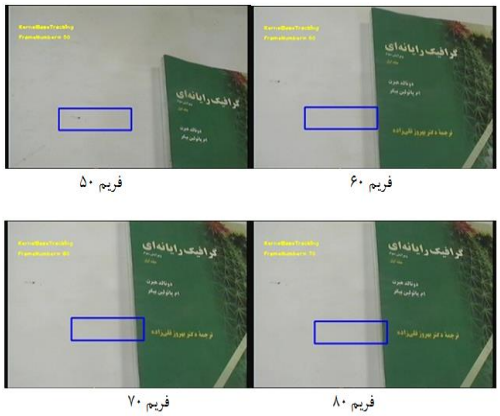
تست اول، تعقیب عنوان کتاب گرافیک رایانه‌ای است که در اولین مورد تعقیب با استفاده از یک *webcam* بر روی جلد کتاب انجام می‌گیرد و نوشته‌ی عنوان کتاب تحت تعقیب می‌باشد. توالی تصاویر زیر که ۱۰ فریم به ۱۰ فریم نمایش داده شده است، عملکرد الگوریتم‌ها را به خوبی نمایان می‌نماید.

۵-۱-۱- عملکرد الگوریتم *Normalized Cross Correlation* به روی جلد کتاب گرافیک رایانه ای

شکل ۱۱- فریم‌های تست عملکرد الگوریتم *Normalized Cross Correlation* به روی جلد کتاب گرافیک رایانه‌ای

نتیجه عملکرد، عالی در محیط‌های پیچیده می‌باشد.

۵-۲-۱- عملکرد الگوریتم *Kernel Base Object Tracking* به روی جلد کتاب گرافیک رایانه ای



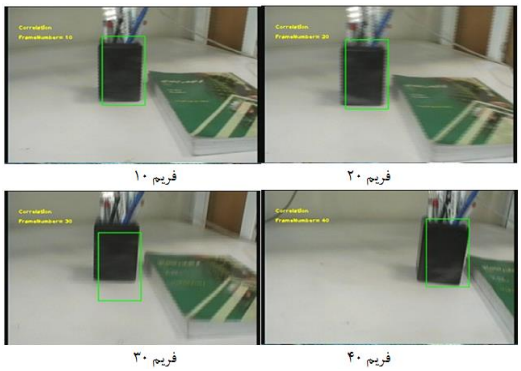
شکل ۱۲- فریم‌های تست عملکرد الگوریتم *Kernel Base Object Tracking* به روی جلد کتاب گرافیک رایانه‌ای

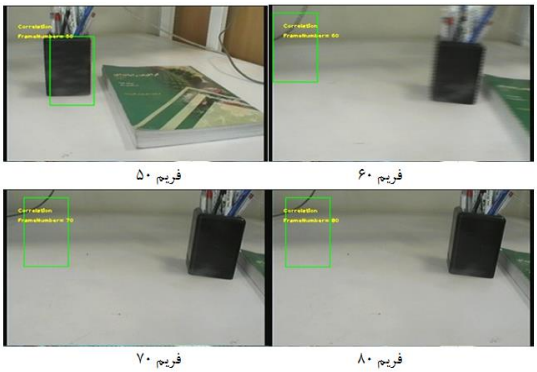
همان‌طور که در تصاویر بالا مشاهده می‌شود، متن تعقیب‌شده دارای مرزبندی مشخصی با محیط پس‌زمینه نمی‌باشد و در این‌جا عملکرد الگوریتم *Normalized Cross Correlation* از الگوریتم *Kernel Base Object Tracking* بسیار بهتر است و الگوریتم *Kernel Base Object Tracking* عملاً از فریم چهلم به بعد، هدف تحت تعقیب را گم نموده است.

۵-۲- تست دوم: تعقیب جامدادی

در این تست، برعکسِ تست بالا، هدف نسبت به محیط تمایز آشکار داشته و دارای مرزبندی مشخص می‌باشد.

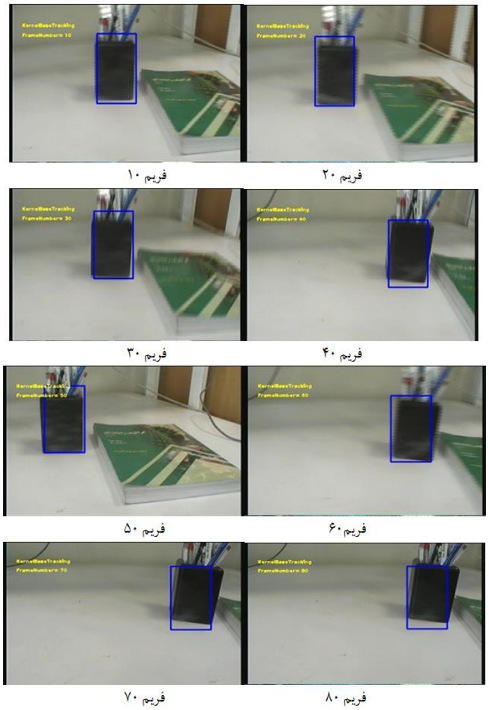
۵-۲-۱- عملکرد الگوریتم *Normalized Cross Correlation* به روی جامدادی





شکل ۱۳- فریم‌های تست عملکرد الگوریتم *Normalized Cross Correlation* به روی جامدادی

۵-۲-۲- بررسی الگوریتم *Kernel Base Object Tracking* به روی جامدادی



شکل ۱۴- فریم‌های تست عملکرد الگوریتم Kernel Base Object Tracking به روی جامدادی

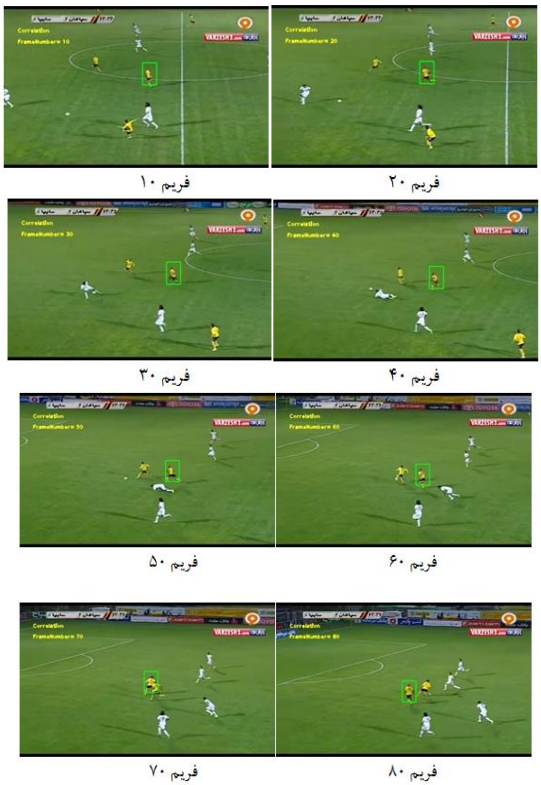
در تست بالا هر دو الگوریتم با چالشی جدید مواجه شده‌اند که همان‌طور که مشاهده شده، به دلیل تمایز کامل جامدادی از محیط پس‌زمینه الگوریتم *kernel Base Object Tracking* عملکرد موفق‌تری نسبت به *Normalized Cross Correlation* داشته است.

۵-۱- تست سوم: بازیکن فوتبال

این تست، برخلاف دو تست بالا، بر روی یک فیلم صورت گرفته و فریم‌های تصویر به‌ازای خوانده شدن از کارت کپچر و یا *WebCam*، از روی یک فایل ویدیویی با فرمت *AVI* در حافظه کپچر شده که عملکرد دو الگوریتم در این مورد نیز تست شده است.

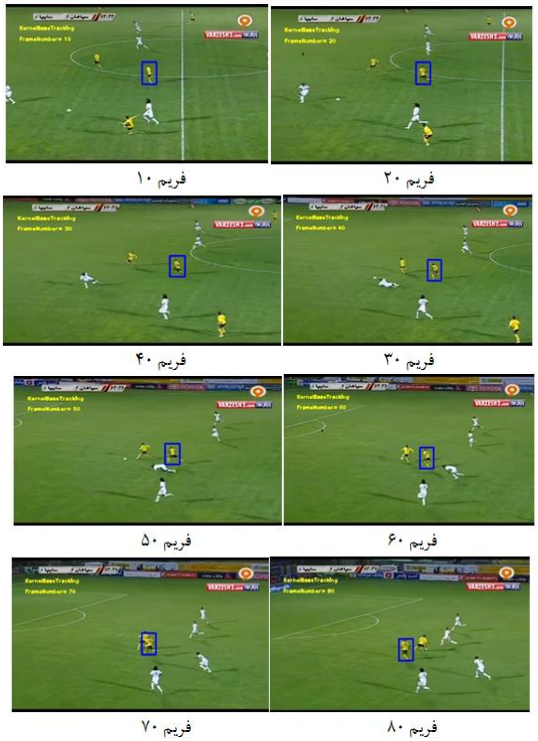
۵-۱-۳- عملکرد الگوریتم *Normalized Cross Correlation* به روی مهاجم زردپوش

در تصاویر زیر نحوه‌ی عملکرد الگوریتم *Normalized Cross Correlation* را در تعقیب مهاجم زردپوش سپاهان در بین حدود ۱۰۰ فریم متوالی را نشان می‌دهد.



شکل ۱۵- فریم‌های تست عملکرد الگوریتم *Normalized Cross Correlation* به روی مهاجم زردپوش

۵-۳-۲- عملکرد الگوریتم *Kernel Base Object Tracking* به روی مهاجم زردپوش



شکل ۱۵- فریم‌های تست عملکرد الگوریتم *Kernel Base Object Tracking* به روی مهاجم زردپوش

در شکل‌های ۱۶ و ۱۷ نتایج حاصل از اعمال الگوریتم‌های مذکور بر روی سه هدف تعیین‌شده نمایش داده شده‌اند.

	فریم ۱۰	فریم ۲۰	فریم ۳۰	فریم ۴۰	فریم ۵۰	فریم ۶۰	فریم ۷۰	فریم ۸۰
جلد کتاب	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
جامدادی	Yes	Yes	Yes	Yes	Yes	No	No	No
بازیکن	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

شکل ۱۶- جدول عملکرد الگوریتم *Normalized Cross Correlation*

	فریم ۱۰	فریم ۲۰	فریم ۳۰	فریم ۴۰	فریم ۵۰	فریم ۶۰	فریم ۷۰	فریم ۸۰
جلد کتاب	Yes	Yes	Yes	No	No	No	No	No
جامدادی	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
بازیکن	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

شکل ۱۷- جدول عملکرد الگوریتم *Kernel Base Object Tracking*

۶- نتیجه‌گیری

در این تحقیق، فرمول‌های ریاضی و مطالب ارائه‌شده به کدهایی که به راحتی قابل استفاده در محصولات نظارتی و امنیتی باشند، تبدیل و اجرا گردید. با توجه به ماهیت متفاوت دو الگوریتم می‌توان متوجه شد که کاربرد این دو الگوریتم با یکدیگر متفاوت می‌باشد که برای الگوریتم *Normalized Cross Correlation* با توجه به ضریب همبستگی آماری داده‌ها که محاسبه می‌شود، می‌توان متوجه شد که لزومی ندارد که شیء مورد تعقیب از محیط پس‌زمینه متفاوت باشد؛ زیرا در این الگوریتم برای جستجو از یک رابطه‌ی آماری استفاده می‌گردد که این رابطه به ماهیت و شکل شیء تحت تعقیب وابستگی ندارد و در عوض در محیط‌هایی که شیء مورد تعقیب با محیط پس‌زمینه کاملاً متمایز باشد، این الگوریتم عملکرد ضعیف‌تری از خود نشان می‌دهد.

۷- پیشنهادات

GPU یک مدار اختصاصی طراحی‌شده برای سرعت بخشیدن به نمایش تصویر خروجی، بر روی صفحه نمایش است. *GPU* وسیله‌ای است شامل تعدادی عملگر ابتدایی گرافیکی، که باعث می‌شود نسبت به پردازشگر مرکزی یا همان *CPU*، در خلق تصاویر بر صفحه‌ی نمایشگر، بسیار سریع‌تر عمل کند. در واقع *GPU* همانند *CPU* است ولی وظیفه‌ی اصلی آن، پردازش اطلاعات مرتبط با تصاویر است.

تلفن‌های هوشمند جدید به چیپ‌ستهایی مجهز شده‌اند که هر کدام می‌تواند وظایف مختلفی را با توجه به برنامه‌نویسی که برای آنها درنظر گرفته شده است، انجام دهند و *GPU* بخش مهمی از این چیپ‌ستها می‌باشد که مخصوصاً در اجرای بازی‌ها و خروجی تصویر باکیفیّت آنها بر روی صفحه نمایش، بسیار مؤثر است. در واقع، پردازش تصاویر گرافیکی و نمایش بهتر جزئیات، بر عهده‌ی *GPU* است.

با توجه به این‌که استفاده از *Gpu* در پردازش موازی داده‌های تصویری می‌تواند سرعت پردازش را بالاتر ببرد، می‌توان دو الگوریتم *Normalized Cross Correlation* و *Kernel Base Object Tracking* را با استفاده از *gpu*های *nvidia* و *sdk*های ارائه‌شده توسط این شرکت موازی‌سازی نمود.

۸- مراجع:

[1] Klette, R. (2014). An Introduction into Theory and Algorithms, Springer-Verlag Concise Computer Vision.

[2]Comaniciu, D. & et al, (2003). Kernel-based Object Tracking, IEEE Trans. Pattern Anal. Machine Intell.

[3] Elgammal, A. & et al, (2003). Efficient Kernel Density Estimation Using the Fast Gauss Transform with Applications to Color Modeling and Tracking, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[4]Chang, C. & Ansari, R. (2005). Kernel Particle Filter for Visual Tracking, IEEE Signal Processing Letters.

[5] Yilmaz, A. (2007). Object Tracking by Asymmetric Kernel Mean Shift with Automatic Scale and Orientation Selection, IEEE Conference on Computer Vision and Pattern Recognition.

[6]Caseiro, R. & et al, (2015). High-Speed Tracking with Kernelized Correlation Filters, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[7]Venkatesh Babua, R. & Pérez, P. (2007). Robust Tracking With Motion Estimation And Local Kernel-Based Color Modeling, Image and Vision Computing.

[8]Wu, Yi & et al, (2013). Online Object Tracking: A Benchmark, IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[9] Porikli, F. , (2005). Multi-Kernel Object Tracking, IEEE International Conference on Multimedia and Expo.

[10] Jeyakara, J. & VenkateshBabu, R. (2008). Robust Object Tracking with Background-Weighted Local Kernels, Computer Vision and Image Understanding.

[11] Wesley, E. S. & Hairong, Q. (2004). Machine Vision, Cambridge University Press.

[12] Briechle, K. Template Maching using Fast Normalized Cross Corralation, Institude of Automatic Control Engineering ,Technical University Munchen, Germany.

[13] The OpenCV Reference Manual, Release 2. 4. 3.

نشریه‌ی علمی شهر ایمن
JOURNAL OF RESILIENT CITY
(JRC)

Real-time video tracking of objects using modified Kernel Base Object Tracking algorithm with more accurate detection capability in security and surveillance products

Majid Shakeri[*1];

١- Electrical Department, Shahriar Branch, Islamic Azad University, Shahriar, Tehran, Iran (Corresponding Author)

Abstract:

One of the fields of visual observation is the moving object tracking machine .Detection of a moving object is a real-time process in which the position of an animated object is determined at any time .For this purpose, in this article, we tried to work with a practical approach and result in the production of the product and using the OpenCV library in the Visual C ++ programming environment to try to create an object in successive frames using a Pan Tilt Zoom camera, a camera with a platform Two degrees of freedom are tracked promptly .Also, in this paper, we investigate a histogram based on the Kernel Base Object Tracking algorithm and its performance is compared with the Normalized Cross Correlation algorithm based on the statistical correlation coefficient .The purpose of this article is to translate mathematical formulas and materials into codes that are easy to use in surveillance and security.

Key Words: Machine Vision, Real-time video tracking, Kernel Base Object Tracking Algorithm

Corresponding Author: Shahriar Branch, Islamic Azad University, Shahriar, Tehran, Iran majid.sh@gmail.com