علی آراسته ۹۶۱۰۱۱۶۵

دید کامپیوتری – پروژه

در این پروژه یکی از روش‌های موجود برای خلاصه‌سازی ویدئو (Video Synopsis) پیاده‌سازی شده است. مبنای این روش خلاصه سازی ویدیو به صورت مکانی و زمانی است به طوری که در نهایت باعث Dense تر شدن اشیا در تصویر و کوتاه شدن زمان کلی ویدیو می‌شود. در این روش ابتدا اشیا متحرک را از پس زمینه جدا می‌کنیم. سپس این اشیا را در طول وجود خود در ویدیو Track می‌کنیم تا Tube های زمانی مکانی از آنها بسازیم. در نهایت این Tube ها را در ویدیو خلاصه شده‌ای که میخواهیم آن را تولید کنیم طوری قرار می‌دهیم که زمان و همپوشانی میان آن‌ها کمینه شود. (یک Trade-off بین زمان و همپوشانی داریم. یعنی اگر مثلا همپوشانی را در نظر نگیریم طول ویدیو نهایی برابر با طول بلندترین Tube می‌شود.)

در این گزارش نحوه پیاده‌سازی هر یک از موارد ذکر شده، مشکلات موجود در هر قسمت و چگونگی رسیدن به نتیجه نهایی را بررسی می‌کنیم. لازم به ذکر است در در کدهای ارسالی سه فایل py. وجود دارد و کد اصلی در فایل project.py قرار گرفته است. در این فایل برای track کردن اشیا از دو فایل دیگر یعنی tracker.py و kalman_filter.py استفاده شده است. حال به بررسی دقیق قسمت‌های مختلف کد می‌پردازیم.

در ابتدا با خواندن ویدئو و میانگین گرفتن از تعداد محدودی از فریم‌ها (فریم‌هایی که مضرب ۱۰ هستند)، تصویر background ویدئو را بدست می‌آوریم. سپس grayScale این تصویر را تولید می‌کنیم تا از آن در جدا کردن اشیا از background استفاده کنیم. علت این است که اعمال فیلترهای مختلف در فرمت grayScale، به مراتب نتیجه بهتری تولید می‌کند. حال دوباره از ابتدا ویدئو را می‌خوانیم. برای هر فریم ابتدا grayScale آن فریم را بدست می‌آوریم و سپس با استفاده از تابع cv2.absdiff، برای هر پیکسل، اختلاف میان intensity دو تصویر grayFrame و grayBackground را محاسبه می‌کنیم. چون ممکن است intensity بعضی از پیکسل‌های مربوط به background در یک فریم با intensity پیکسل‌های منتاظر در background دقیقا یکی نباشند، با قرار دادن مقدار آستانه‌ای مناسبی، دقت تشخیص پیکسل‌های متفاوت را افزایش می‌دهیم. با این کار یک mask به صورت ۰ یا ۲۵۵ بدست می‌آید. در ادامه با اعمال فیلترهای closing و opening با اندازه مناسب، نویز موجود در mask را از بین می‌بریم. همچنین با اعمال فیلترهای dilation و erosion یکی قبل و دیگری بعد از فیلترهای closing و opening پیوستگی و انسجام mask را افزایش می‌هیم. در آخر نیز با اعمال یک مقدار آستانه‌ای، mask ای به صورت ۰ یا ۲۵۵ تولید می‌کنیم. حال نوبت به تشخیص اشیا می‌رسد. برای این کار از تابع cv2.findContours بهره می‌گیریم. برای افزایش دقت با اعمال محدودیت روی مساحت contour های شناسایی شده توسط تابع، contour های مناسب را گزینش می‌کنیم. برای نحوه ذخیره کردن contour ها، روش‌های مختلفی را بررسی کردیم. در ابتدا از مستطیل محیطی contour استفاده کردیم، سپس convex hull را امتحان کردیم، اما در نهایت بهترین نتیجه مربوط به خود contour بود. البته لازم به ذکر است استفاده از خود contour، باعث شد نتوانیم از تابع‌های آماده موجود در openCV برای track کردن اشیای شناسایی شده استفاده کنیم و لذا مجبور شدیم الگوریتمی برای track کردن اشیا با استفاده از خود contour پیاده‌سازی کنیم که این موضوع مشکلات زیادی را به همراه داشت. در ادامه به شرح نحوه عملکرد این الگوریتم می‌پردازیم. این الگوریتم مرکز contour های شناسایی شده در هر فریم را به عنوان ورودی دریافت می‌کند و با استفاده از آن اشیای شناسایی شده در فریم‌های قبلی را در فریم فعلی track می‌کنیم.

برای این کار از کلاس tracker استفاده می‌کنیم که کد مربوط به آن در فایل tracker.py موجود است. سازنده این کلاس سه متغیر برای ایجاد شی دریافت می‌کند. متغیر اول dist_thresh مقدار آستانه را برای حداکثر فاصله میان مرکز contour های مربوط به یک شی در دو فریم متوالی تعیین می‌کند. متغیر دوم max_frames_to_skip مقدار آستانه را برای حداکثر تعداد فریم‌هایی که یک track بدون تشخیص contour جدید، فعال می‌ماند، تعیین می‌کند. متغیر سوم min_frames_count مقدار آستانه را برای حداقل تعداد فریم‌های یک track تعیین می‌کند. برای تولید ویدئو خلاصه شده از مقادیر (۱۰۰، ۲۵، ۲۵۰) استفاده کردیم. این کلاس دارای دو تابع است. تابع اول Update اشیای شناسایی شده در فریم‌های قبلی را در فریم فعلی track می‌کند. برای یک ماتریس N * M تولید می‌کند که در آن N تعداد track های فعال از فریم‌های قبلی و M تعداد contour های شناسایی شده در فریم فعلی است. برای هر یک از درایه‌های ماتریس فاصله اقلیدسی میان آخرین مرکز track سر i ام و مرکز contour شناسایی شده j ام محاسبه می‌شود. سپس با استفاده از تابع linear_sum_assignment موجود در پکیج scipy.optimize بهترین تخصیص صورت می‌گیرد. در ادامه مقدار آستانه dist_thresh اعمال می‌شود تا از تخصیص‌های غلط جلوگیری شود. اگر در یک مرحله به یک track فعال، contour جدیدی اختصاص نیابد، متغیر skipped_frames آن، یک واحد افزایش می‌یابد و اگر متغیر skipped_frames یک track فعال از مقدار آستانه max_frames_to_skip بیشتر شود، آن track، غیرفعال می‌شود. اگر تعداد فریم‌های یک track در هنگام غیر فعال شدن از مقدار آستانه min_frames_count بیشتر باشد، ذخیره می‌شود. در هر فریم اگر هر یک از contour های شناسایی شده، به هیچ یک از track های فعال از فریم‌های قبلی اختصاص نیابد، track جدیدی برای آن تولید می‌شود. سپس برای هر یک از contour های شناسایی شده در فریم فعلی، mask ای ایجاد می‌شود و در track مربوط به آن contour ذخیره می‌شود. این mask تصویری با ابعاد فریم‌های ویدئو است که در آن intensity پیکسل‌هایی که درون contour نبوده اند، صفر شده است. در آخر مرکز آن دسته از track هایی که یک contour جدید از فریم فعلی را به خود اختصاص داده اند، به روز رسانی می‌شود. برای این کار فیلتر kalman استفاده می‌شود. کد مربوط به پیاده‌سازی این فیلتر در فایل kalman_filter.py موجود است. برخی از قسمت‌های امتیازی نیز در این تابع پیاده‌سازی شده است که در ادامه آن‌ها را توضیح می‌دهیم. تابع دوم finish پس از خوانده شدن آخرین فریم فراخوانی می‌شود. وظیفه این تابع ذخیره کردن track هایی است که پس خوانده شدن آخرین فریم، فعال بوده اند.

حال نوبت به تولید ویدئو خلاصه شده می‌رسد. برای این کار ابتدا ماکسیمم تعداد فریم‌های tube های ایجاد شده را، محاسبه می‌کنیم. روش‌های مختلفی را برای تولید ویدئو خلاصه شده از روی این tube ها، مورد آزمایش قرار دادیم. در ابتدا بدون توجه به تعداد فریم‌های tube ها، محل شروع همه آن‌ها را ابتدای ویدئو قرار دادیم. نتیجه ویدئویی با طولی برابر بلندترین tube بود که در آن همپوشانی بین tube ها زیاد بود. برای کاهش این امر زمان شروع هر tube را بر اساس تعداد فریم‌های آن تعیین کردیم. نتیجه به مراتب بهتر بود.

موارد امتیازی:

تشخیص مسیر حرکت و ترسیم آن:

برای این کار به دقت بالاتری احتیاج داشتیم، لذا باید از tube هایی که دقت بیشتری داشتند، استفاده می‌کردیم. برای بدست آوردن این tube ها، از مقادیر (۵۰، ۵، ۲۰) استفاده کردیم. در ابتدا اندازه و زاویه خطی را که از وصل کردن ابتدا و انتهای یک

tube به یکدیگر بدست می‌آمد، محاسبه کردیم. سپس با استفاده از تابع cv2.kmean، زاویه‌های tube ها را به دو دسته cluster کردیم. در آخر برای هریک از دو دسته، خطی با زاویه مرکز دسته به طول بلندتری tube دسته از محل شروع آن tube رسم کردیم.

تشخیص رنگ خودرو:

برای این کار در هر فریم و برای هریک از contour های شناسایی شده، میانگین هر یک از کانال‌های قرمز، سبز و آبی را محاسبه کردیم. برای این کار با استفاده از mask تولید شده برای contour، تعداد پیکسل‌های موجود در آن و به وسیله آن، میانگین رنگ پیکسل‌های contour را بدست آوردیم. در آخر این رنگ را به صورت دایره‌ای در مرکز contour نمایش دادیم.

شمارش تعداد خودروها:

برای این کار از کد نوشته شده برای تشخیص مسیر حرکت استفاده کردیم و با شمارش تعداد tube های هر مسیر، تقریبی از تعداد ماشین‌هایی را که در طول ویدئو از مسیر یکسانی عبور می‌کنند، محاسبه کردیم.

محاسبه و نمایش کمیتی مربوط به سرعت هر خودرو:

برای این کار از متوسط‌گیری پویا روی مقدار جابه‌جایی پیکسلی هر خودرو در دو فریم متوالی استفاده کردیم. در این کار به هریک از مقدارهای فعلی و قبلی ضریب ۰/۵ را نسبت دادیم.

کمینه کردن همپوشانی بین اشیا مختلف:

برای این کار از یک الگوریتم تصادفی استفاده کردیم. در ابتدا برای هر tube، اختلاف میان تعداد فریم‌های آن و ماکسیمم تعداد فریم‌های tube های ایجاد شده را محاسبه کردیم. سپس با استفاده از یک توزیع یکنواخت برای هر tube، عددی بین صفر تا عدد بدست آمده در قسمت قبل، تعیین کردیم. در آخر این اعداد را برای تعیین زمان شروع هر tube مورد استفاده قرار دادیم.

خروجی مربوط به دو بخش تشخیص مسیر حرکت و ترسیم آن و شمارش تعداد خودروها:

لازم به ذکر است ویدئو خلاصه شده شامل موارد امتیازی به همراه گزارش ارسال شده است.