دانشکده مکانیک، برق و کامپیوتر

گروه هوش مصنوعی و رباتیکز

دوره ی OpenCV

پردازش ویدیو و حرکت

تهیه و تنظیم :

زهرا عبادی

استاد راهنما :

دکتر عباس کوچاری

تابستان 1402

فهرست مطالب صفحه

1. VideoWriter:

```
# Open the video file for reading
cap = cv2.VideoCapture("drive/MyDrive/pedestrians.avi")

output_file  = "drive/MyDrive/output_video.avi"
fourcc = cv2.VideoWriter_fourcc(*'XVID')

fps = 40
seconds = 2
num_frames = fps * seconds

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

out = cv2.VideoWriter(output_file,  fourcc, fps,(frame_width, frame_height))

for i in range(num_frames):
   ret, frame = cap.read()

   if not ret:
      break

   out.write(frame)
```

توضیح کد این بخش:

- **"output_file = "drive/MyDrive/output_video.avi**: این خط نام و مکان فایل ویدیوی خروجی را که ایجاد می‌شود، مشخص می‌کند.
- **fourcc = cv2.VideoWriter_fourcc(*'XVID'):** در اینجا، fourcc به "Four Character Code" یا کد چهار حرفی اشاره دارد، که برای مشخص کردن کدک ویدیویی که برای رمزگذاری ویدیو استفاده می‌شود. در این خط، به 'XVID' تنظیم شده است که یک کدک معمولی برای فرمت ویدیوی AVI (Audio Video Interleave) است. نحوه استفاده از 'XVID'*برای باز کردن حروف'X' ، 'V'، 'I'و 'D' به عنوان آرگومان‌های جداگانه برای VideoWriter_fourccاست. شما می‌توانید این کدک را به کدک دیگری تغییر دهید اگر ترجیح می‌دهید. جایگزین‌های معمول شامل 'MJPG' برای Motion-JPEG و 'H264' برای فشرده‌سازی H.264 هستند.
- **fps = 40 و seconds = 2** : این خطوط نرخ فریم در ثانیه و مدت زمان (به صورت ثانیه) ویدیوی خروجی را مشخص می‌کنند.
- **num_frames = fps * seconds**: این خط تعداد کل فریم‌های مورد نیاز برای ویدیوی خروجی را بر اساس نرخ فریم و مدت زمان مشخص می‌کند.

- **frame_width = int(cap.get(3))** و **frame_height = int(cap.get(4))**: این خطوط عرض و ارتفاع فریم‌های ویدیو را از شی VideoCapture (cap) با استفاده از متد get دریافت می‌کنند cap.get(3) به عرض (CV_CAP_PROP_FRAME_WIDTH) و cap.get(4) به ارتفاع (CV_CAP_PROP_FRAME_HEIGH) فریم‌های ویدیو اشاره دارند.
- **out = cv2.VideoWriter(output_file, fourcc, fps, (frame_width, frame_height))**: این خط یک شی VideoWriter به نام out ایجاد می‌کند تا ویدیوی خروجی را ذخیره کند. این شی به موارد زیر نیاز دارد:
  - **output_file**: نام فایل ویدیو خروجی.
  - **fourcc**: کدک ویدیویی که برای رمزگذاری ویدیو استفاده می‌شود، که قبلاً تعیین شده است مثلاً **('XVID')**
  - **fps**: تعداد فریم‌ها در هر ثانیه (فریم در ثانیه) ویدیو خروجی.
  - **(frame_width, frame_height)**: عرض و ارتفاع فریم‌های ویدیو خروجی. این مقادیر از شی VideoCapture گرفته می‌شوند تا ویدیو خروجی دارای ابعاد مشابه فریم‌های ورودی باشد.
- **for i in range(num_frames)**: برای تعداد فریم‌های مشخص شده اجرا می‌شود.
- **ret, frame = cap.read()**: این خط یک فریم از ویدیو ورودی می‌خواند ret .یک بولین است که نشان می‌دهد آیا یک فریم با موفقیت خوانده شده است یا نه و frame حاوی داده تصویر است.
- **if not ret: break**: این خط بررسی می‌کند که آیا یک فریم با موفقیت خوانده شده است یا نه (به عنوان مثال، اگر انتهای ویدیو رسیده شود) و در صورت عدم موفقیت، از حلقه خارج می‌شود.
- **frame = cv2.resize(frame, (frame_width, frame_height))**: این خط فریم را بازبازسازی می‌کند تا ابعاد آن با ابعاد ویدیوی خروجی همخوانی کند.
- **out.write(frame)**: این خط فریم را به ویدیوی خروجی می‌نویسد.

به طور خلاصه، این خطوط کد مشخص می‌کنند که نام فایل ویدیو خروجی، کدک، نرخ فریم و ابعاد فریم‌ها چگونه تعریف می‌شوند و سپس یک شی **VideoWriter** ایجاد می‌شود که برای نوشتن فریم‌های پردازش شده به فایل ویدیو خروجی با تنظیمات مشخص‌شده استفاده می‌شود.

2. Implementing a basic background subtractor:

برای پیاده‌سازی کاهش پس‌زمینه پایه، از رویکرد زیر استفاده میکنیم :

- از دوربین، فریم‌ها را ضبط کنید.
- 9 فریم را رد کنید تا دوربین به مدتی فرصت داشته باشد که تنظیم خودکار نوردهی خود را برای تطابق با شرایط نور در صحنه انجام دهد.
- فریم دهم را بگیرید، آن را به سیاه و سفید تبدیل کنید، آن را تار کنید و از این تصویر تار به عنوان تصویر مرجع پس‌زمینه استفاده کنید.
- برای هر فریم پی‌ازپی، فریم را تار کنید، به سیاه و سفید تبدیل کنید و تفاوت مطلق بین این فریم تار و تصویر مرجع پس‌زمینه را محاسبه کنید. روی تصویر تفاوت‌ها، آستانه‌گذاری، انعطاف و تشخیص لبه(thresholding, smoothing, and contour detection) انجام دهید. مستطیل‌های محدودکننده مشخصات اصلی را رسم و نمایش دهید.

توضیحات کد :

10 فریم از دوربین را ضبط مبکنیم.

```
for i in range(10):
    success, frame = cap.read()
    if not success:
        exit(1)
```

اگر قادر به ضبط 10 فریم نبودیم، خارج می‌شویم. در غیر اینصورت، به تبدیل فریم دهم به سیاه و سفید و تار می‌پردازیم :

```
gray_background = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray_background = cv2.GaussianBlur(gray_background, (BLUR_RADIUS, BLUR_RADIUS), 0)
```

در این مرحله، تصویر مرجع پس‌زمینه را داریم:

حالا به ضبط تصاویر بیشتر ادامه می دهیم که ممکن است در آنها حرکت تشخیص داده شود. پردازش هر فریم با تبدیل به سیاه و سفید و اعمال عملیات تار کردن گوسی شروع می‌شود :

```
gray_background = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray_background = cv2.GaussianBlur(gray_background, (BLUR_RADIUS, BLUR_RADIUS), 0)
```

حالا می‌توانیم نسخه تار و سیاه و سفید فریم فعلی را با نسخه تار و سیاه و سفید تصویر پس‌زمینه مقایسه کنیم. به‌طور خاص، از تابع cv2.absdiff برای پیدا کردن مقدار مطلق (یا مقدار) تفاوت بین این دو تصویر استفاده خواهیم کرد. سپس، یک آستانه را برای به دست آوردن یک تصویر سیاه و سفید خالص اعمال کرده و عملیات مورفولوژی برای نرم‌کردن تصویر آستانه‌ای انجام می دهیم. کد مربوط به این بخش به شرح زیر است :

```
diff = cv2.absdiff(gray_background, gray_frame)
_, thresh = cv2.threshold(diff, 40, 255, cv2.THRESH_BINARY)
cv2.erode(thresh, erode_kernel, thresh, iterations=2)
cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)
```

فرسایش (cv2.erode()) : `cv2.erode(thresh, erode_kernel, thresh, iterations=2)`

فرسایش یک عملیات مورفولوژیک است که برای فرسودن مرزهای اشیاء جلوگیری کننده در تصویر دودویی به کار می‌رود. این عملیات با حرکت یک عنصر سازگاری (که توسط erode_kernel تعریف شده است) روی تصویر دودویی انجام می‌شود و هر پیکسل در همسایگی عنصر سازگاری با کمینه مقدار پیکسل جایگزین می‌شود.

پارامترها:

- **thresh:** تصویر دودویی که فرسایش روی آن اعمال می‌شود.
- **erode_kernel:** عنصر سازگاری مورد استفاده برای فرسایش. این عنصر شکل و اندازه محله‌ای که برای عملیات استفاده می‌شود را تعریف می‌کند.
- **iterations:** تعداد بارهایی که عملیات فرسایش انجام می‌شود.

در کد شما، شما دو بار (iterations=2) فرسایش روی تصویر دودویی thresh انجام می‌دهید. این عملیات می‌تواند به کاهش نویز در تصویر دودویی کمک کند و همچنین برای جداسازی اشیاءی که به یکدیگر نزدیک هستند، مورد استفاده قرار گیرد.

گسترش (cv2.dilate()): `cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)`

گسترش یک عملیات مورفولوژیک دیگر است که برای گسترش مرزهای اشیاء جلوگیری‌کننده در تصویر دودویی به کار می‌رود. این عملیات با حرکت یک عنصر سازگاری (که توسط dilate_kernel تعریف شده است) روی تصویر دودویی انجام می‌شود و هر پیکسل در همسایگی عنصر سازگاری با بیشینه مقدار پیکسل جایگزین می‌شود.

پارامترها:

- **thresh**: تصویر دودویی که گسترش روی آن اعمال می‌شود.
- **dilate_kernel**: عنصر سازگاری مورد استفاده برای گسترش. این عنصر شکل و اندازه محله‌ای که برای عملیات استفاده می‌شود را تعریف می‌کند.
- **iterations**: تعداد بارهایی که عملیات گسترش انجام می‌شود.

در کد شما، شما دو بار (iterations=2) گسترش روی تصویر دودویی thresh انجام می‌دهید. گسترش می‌تواند به اتصال اجزاء اشیاء نزدیک به یکدیگر و دیدن بهتر آنها کمک کند. این عملیات اغلب با فرسایش ترکیب می‌شود تا عملیات‌هایی مانند بسته کردن (ترکیبی از گسترش و فرسایش) برای وظایفی مانند تشخیص اشیاء انجام شود.

ترکیب فرسایش و گسترش به عنوان "بازکردن مورفولوژیک" شناخته می‌شود هنگامی که فرسایش قبل از گسترش انجام شود و "بسته کردن مورفولوژیک" شناخته می‌شود هنگامی که گسترش قبل از فرسایش انجام شود. این عملیات‌ها ابزارهای مهمی در پیش‌پردازش و تحلیل تصویر برای وظایفی مانند تشخیص اشیاء و تقسیم‌بندی تصویر هستند.

حال اگر تکنیک ما خوب کار کرده باشد، تصویر آستانه‌ای ما باید قسمت های سفید را در هر جایی که یک شیء در حال حرکت است، داشته باشد. حالا می‌خواهیم مرزهای قسمت های سفید را پیدا کنیم و جعبه‌های محدودکننده را دور آنها بکشیم. به عنوان یک وسیله‌ی بیشتر برای فیلتر کردن تغییرات کوچک که احتمالاً اشیاء واقعی نیستند، می‌توانیم یک آستانه بر اساس مساحت مرز تعریف کنیم. اگر مرز خیلی کوچک باشد، به نتیجه می‌رسیم که یک شیء حرکتی واقعی نیست. (البته، تعریف از کوچکی خیلی ممکن است به تفکیک دقت دوربین شما و برنامه شما بستگی داشته باشد؛ در برخی شرایط، ممکن است نخواهید این آزمون را اصلاً اعمال کنید.) کد برای تشخیص مرزها و کشیدن جعبه‌های محدودکننده به شرح زیر است :

```
contours, hier = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for c in contours:
    if cv2.contourArea(c) > 4000:
        x, y, w, h = cv2.boundingRect(c)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
```

- **cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE**: در این خط کد، توابع OpenCV برای پیدا کردن لبه‌های اشیاء در تصویر دودویی thresh استفاده می‌شوند.
  - **thresh**: تصویر دودویی که لبه‌های اشیاء در آن جستجو می‌شود.
  - **cv2.RETR_EXTERNAL**: نوع بازیابی اطلاعات از لبه‌ها. در اینجا، فقط لبه‌های خارجی اشیاء استخراج می‌شوند.

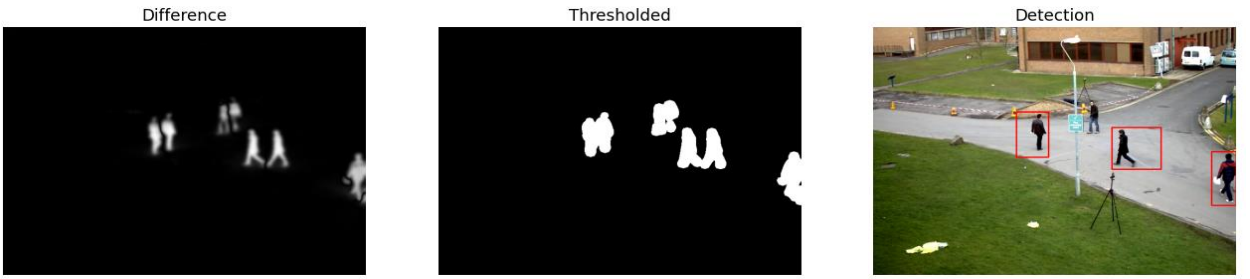
- **cv2.CHAIN_APPROX_SIMPLE**: نوع فرآیند ساده‌سازی که برای تقریب لبه‌های اشیاء استفاده می‌شود.

- **for c in contours**: این حلقه for برای پیمایش تمام اشیاء یا لبه‌های پیدا شده در تصویر ایجاد می‌شود.
- **if cv2.contourArea(c) > 4000**: در اینجا، بررسی می‌شود که آیا مساحت اشیاء (لبه‌ها) بزرگ‌تر از ٤٠٠٠ پیکسل مربع است یا خیر. اگر مساحت بیشتر از این مقدار باشد، به این معناست که این اشیاء بزرگ‌تر و مهم‌تر هستند.
- **x, y, w, h = cv2.boundingRect(c)**: این خط کد اطلاعات مربوط به مستطیل محدب (bounding rectangle) اطراف اشیاء را به دست می‌آورد **x** و **y** مختصات نقطه‌ی بالا و چپ مستطیل و w و h عرض و ارتفاع آن را نشان می‌دهند.
- **cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 255, 0), 2)**: در این خط کد، یک مستطیل رنگی به تصویر اصلی frame اضافه می‌شود. مختصات ابتدایی و انتهایی مستطیل با (x, y) و (x+w, y+h) تعیین می‌شوند. در اینجا، رنگ مستطیل (255, 255, 0) به صورت RGB تعیین شده است و ضخامت خطوط مستطیل با 2 مشخص می‌شود.

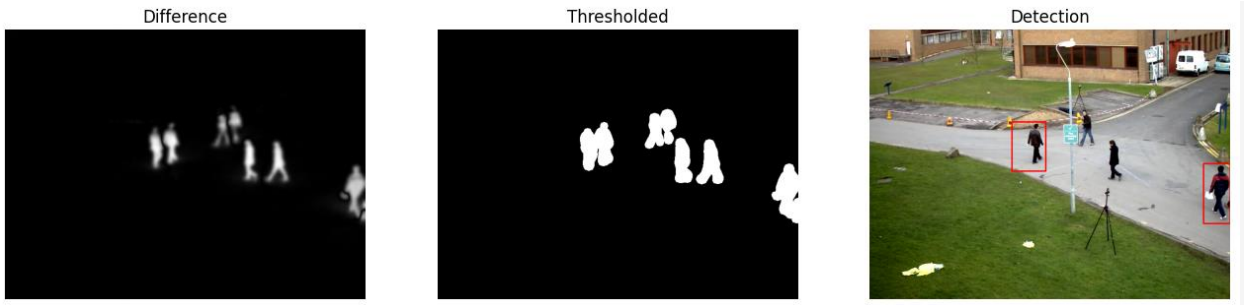
به این ترتیب، لبه‌های اشیاء را در تصویر دودویی شناسایی کرده و مستطیل‌های محدب (مرتبط با هر شیء) را به تصویر اصلی اضافه می‌کند. این روند برای اشیاء با مساحت بزرگ‌تر از ٤٠٠٠ پیکسل مربع انجام می‌شود.

تصویر تفاوت، تصویر آستانه‌گذاری شده و نتیجه تشخیص با مستطیل‌های محدودکننده به شکل زیر است :

Frame 4:



Frame 5:



3. Using a MOG background subtractor:

cv2.createBackgroundSubtractorMOG2 یک تابع در کتابخانه OpenCV است که یک نمونه از تفکیک‌کننده پس‌زمینه مختلط گوسی (MOG) ایجاد می‌کند. این الگوریتم بر اساس مدل‌سازی پس‌زمینه به عنوان یک مخلوط از توزیع‌های گوسی چندگانه استوار است. به خصوص در شرایطی که نورپردازی نسبتاً ثابت است و پس‌زمینه را می‌توان به خوبی با ترکیبی از توزیع‌های گوسی تخمین زد، بسیار موثر است.

در openCV، دو پیاده‌سازی از حذف‌کننده پس‌زمینه MOG وجود دارد که به نام cv2.BackgroundSubtractorMOG و cv2.BackgroundSubtractorMOG2 نام‌گذاری شده‌اند. نسخه دوم، پیاده‌سازی بهبود یافته‌تری است که از تشخیص سایه نیز پشتیبانی می‌کند، بنابراین ما از آن استفاده خواهیم کرد.

```
bg_subtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=16, detectShadows=True)
```

در زیر توضیحات پارامترهای مهم آن آمده است:

- **history:** این پارامتر تعداد فریم‌های قبلی را که برای ساختن مدل پس‌زمینه استفاده می‌شود تنظیم می‌کند. مقدار بالاتر باعث می‌شود مدل به تغییرات در صحنه واکنش پیدا کند.
- **varThreshold:** این یک آستانه بر فاصله مهالانوبیس مربع شده بین پیکسل و حالت است. این به تصمیم‌گیری در مورد این که یک پیکسل بخشی از پس‌زمینه است یا پیش‌زمینه کمک می‌کند.
- **detectShadows:** این پارامتر نشان می‌دهد آیا الگوریتم باید سایه‌ها را تشخیص دهد یا خیر. اگر به **True** تنظیم شود، سایه‌ها را در ماسک پیش‌زمینه علامت گذاری می‌کند. سایه‌ها به عنوان بخشی از پیش‌زمینه در نظر گرفته می‌شوند. اگر به **False** تنظیم شود، سایه‌ها علامت گذاری نخواهند شد.

به عنوان نقطه شروع، دستورالعمل حذف پس‌زمینه اصلی ما را از بخش Implementing a basic background subtractor بگیرید. ما تغییرات زیر را در آن اعمال خواهیم کرد:

1. مدل حذف پس‌زمینه اصلی ما را با یک حذف‌کننده پس‌زمینه MOG جایگزین کنید.
2. به عنوان ورودی، از یک فایل ویدیو به جای دوربین استفاده کنید.
3. استفاده از بلور گوسی را حذف کنید.
4. پارامترهای استفاده شده در مراحل آستانه‌گذاری، مورفولوژی، و تحلیل کانتور را تنظیم کنید.

این تغییرات تأثیری روی چندین خط کد دارند که در سراسر اسکریپت پخش شده‌اند. نزدیک به ابتدای اسکریپت، ما حذف‌کننده پس‌زمینه MOG را مقداردهی اولیه کرده و اندازه کرنل‌های مورفولوژی را تغییر خواهیم داد، همانطور که در بلوک کد زیر به صورت پررنگ نشان داده شده است:

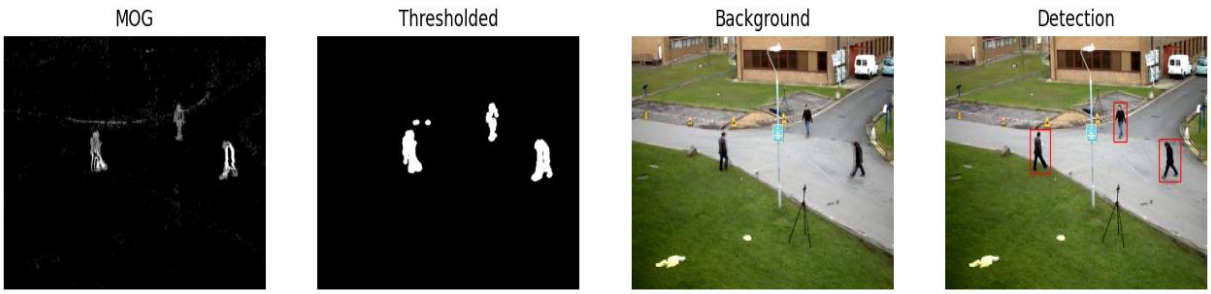
```
# Create a background subtractor using the MOG2 method with shadow detection enabled.
bg_subtractor = cv2.createBackgroundSubtractorMOG2(detectShadows=True)
# Define the kernel sizes for erosion and dilation
erode_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
dilate_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
```

توجه داشته باشید که OpenCV یک تابع به نام cv2.createBackgroundSubtractorMOG2 ارائه داده است تا یک نمونه از cv2.BackgroundSubtractorMOG2ایجاد کند. این تابع یک پارامتر به نام detectShadows را قبول می‌کند که ما آن را به Trueتنظیم می‌کنیم تا مناطق سایه به عنوان سایه تشخیص داده شوند و به عنوان قسمتی از پیش‌زمینه علامت‌گذاری نشوند.

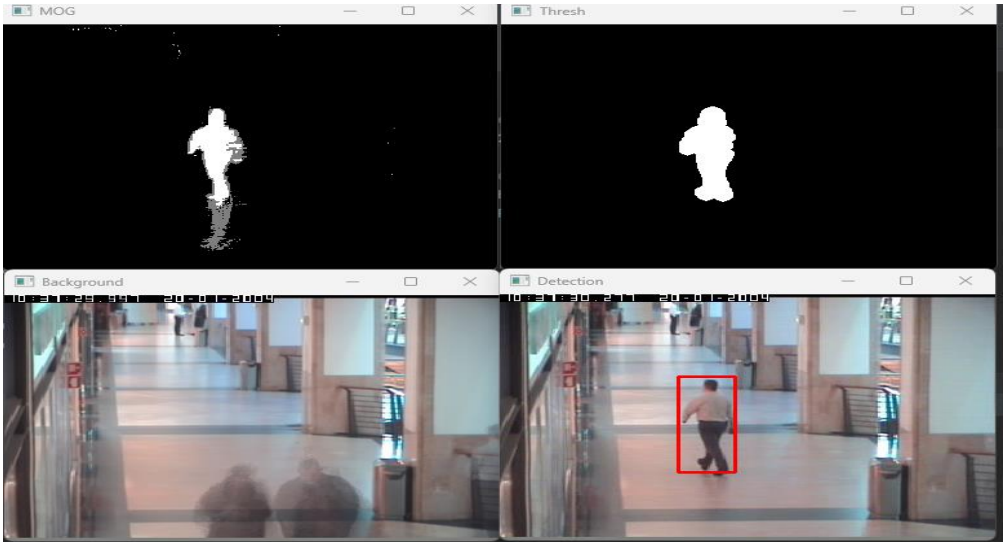
تغییرات باقی‌مانده، از جمله استفاده از حذف‌کننده پس‌زمینه MOG برای به دست آوردن ماسک پیش‌زمینه/سایه/پس‌زمینه، در بلوک کد زیر به صورت پررنگ نشان داده شده است:

```
# Apply background subtraction to get the foreground mask
fg_mask = bg_subtractor.apply(frame)
# Apply thresholding to create a binary image
_, thresh = cv2.threshold(fg_mask, 244, 255, cv2.THRESH_BINARY)
# Apply morphological erosion and dilation to smoothen the thresholded image
cv2.erode(thresh, erode_kernel, thresh, iterations=2)
cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)
# Find contours of objects in the thresholded image
contours, hier = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# Loop through detected contours and draw bounding rectangles for large ones
for c in contours:
    if cv2.contourArea(c) > 1000:
        x, y, w, h = cv2.boundingRect(c)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
```

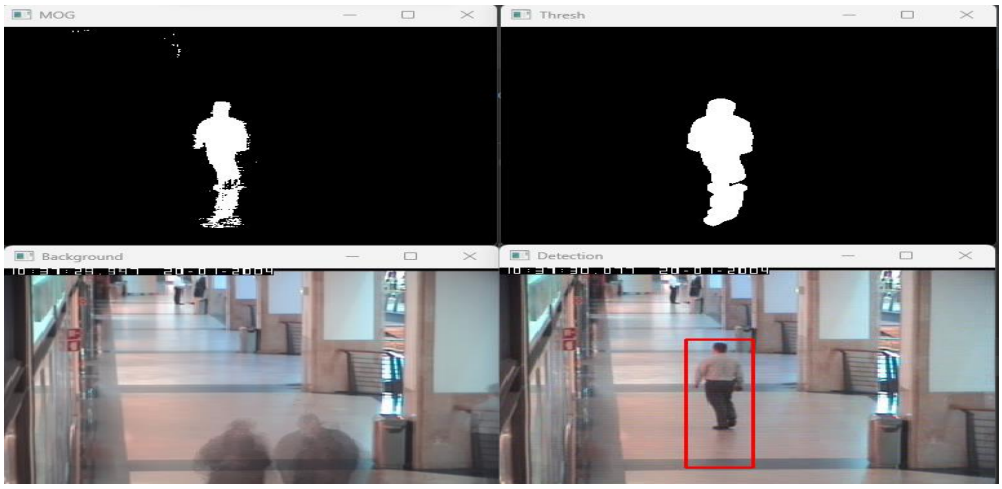
وقتی یک فریم به متد apply حذف‌کننده پس‌زمینه ارسال می‌شود، حذف‌کننده مدل داخلی خود از پس‌زمینه را به‌روز کرده و سپس یک ماسک برمی‌گرداند. ماسک برای بخش‌های پیش‌زمینه سفید (255)، برای بخش‌های سایه خاکستری (127) و برای بخش‌های پس‌زمینه سیاه (0) است. برای اهداف ما، ما سایه‌ها را به عنوان پس‌زمینه در نظر می‌گیریم، بنابراین ما یک آستانه تقریباً سفید (244) روی ماسک اعمال می‌کنیم.ماسک MOG ، تصویرآستانه‌گذاری شده، تصویر پس زمینهBackground و نتیجه تشخیص با مستطیل‌های محدودکننده به شکل زیر است : (Frame 2)



ماسک MOG (تصویر بالا سمت چپ)، تصویر آستانه‌گذاری شده(تصویر بالا سمت راست) و تصویر پس زمینهBackground(تصویر پایین سمت چپ) و نتیجه تشخیص (تصویر پایین پایین سمت راست) به شکل زیر است :



برای مقایسه، اگر ما تشخیص سایه‌ها را با تنظیم detectShadows=False غیرفعال کنیم، نتایجی شبیه به مجموعه تصاویر زیر خواهیم داشت:



این صحنه شامل نه تنها سایه‌ها بلکه همچنین انعکاس‌ها نیز است، به دلیل کف و دیواری که براق هستند. زمانی که تشخیص سایه‌ها فعال است، ما می‌توانیم از یک آستانه استفاده کنیم تا سایه‌ها و انعکاسات را از ماسک حذف کنیم و مستطیل دقیقی اطراف مرد در هال را داشته باشیم. با این وجود، زمانی که تشخیص سایه‌ها غیرفعال است، دو تشخیص داریم که هر دوی آن‌ها احتمالاً دقیق نیستند. یک تشخیص مرد همراه با سایه و انعکاس او در زمین می‌پوشاند. تشخیص دوم انعکاس مرد روی دیوار را می‌پوشاند. این تشخیص‌ها، به طور قابل ادعا، تشخیصات نادرست هستند زیرا سایه و انعکاس‌های مرد در واقع اشیاء متحرک نیستند، حتی اگر نماینده‌های تصویری از یک شیء متحرک باشند.

4. Using a KNN background subtractor

با تغییر فقط پنج خط از کد در اسکریپت تفکیک پس‌زمینه MOG ما، می‌توانیم از یک الگوریتم مختلف برای تفکیک پس‌زمینه، پارامترهای مورفولوژی متفاوت و یک ویدیو ورودی متفاوت استفاده کنیم.

فقط با جایگزینی cv2.createBackgroundSubtractorMOG2 با cv2.createBackgroundSubtractorKNN می‌توانیم از یک تفکیک‌کننده پس‌زمینه براساس خوشه‌بندی KNN به جای خوشه‌بندی MOG استفاده کنیم:

```
bg_subtractor = cv2.createBackgroundSubtractorKNN(detectShadows=True)
```

cv2.createBackgroundSubtractorKNN یک تابع در کتابخانه OpenCV است که یک نمونه از تفکیک‌کننده پس‌زمینه براساس الگوریتم همسایگان نزدیک (K-Nearest Neighbors یا KNN) ایجاد می‌کند. این الگوریتم تفکیک پس‌زمینه بر اساس یک شکل از خوشه‌بندی به نام همسایگان نزدیک انجام می‌شود که پیکسل‌ها را بر اساس شباهت آن‌ها با پیکسل‌های همسایه به دو دسته پس‌زمینه و پیش‌زمینه تقسیم می‌کند.

توجه کنید که با اینکه الگوریتم تغییر کرده است، پارامتر detectShadows همچنان پشتیبانی می‌شود. علاوه بر این، متد apply نیز همچنان پشتیبانی می‌شود، بنابراین ما نیازی به تغییر هیچ چیز مرتبط با استفاده از تفکیک‌کننده پس‌زمینه در ادامه اسکریپت نداریم.

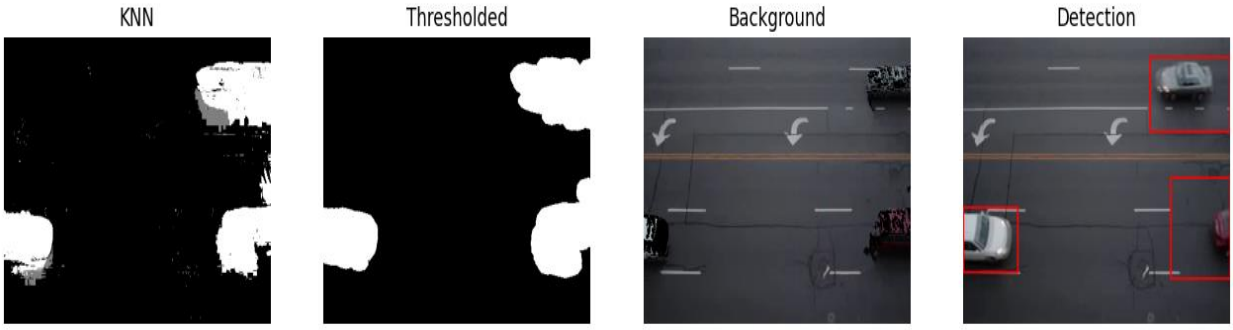
بیاد داشته باشید که cv2.createBackgroundSubtractorMOG2 یک نمونه جدید از کلاس cv2.createBackgroundSubtractorKNN برمی‌گرداند. به مانند آن، cv2.createBackgroundSubtractorKNN یک نمونه جدید از کلاس cv2.createBackgroundSubtractorKNN برمی‌گرداند. هر دوی این کلاس‌ها زیرکلاس‌های cv2.createBackgroundSubtractor هستند که متدهای مشترکی مانند apply را تعریف می‌کنند.

با این تغییرات زیر، می‌توانیم از هسته‌های مورفولوژی استفاده کنیم که کمی بهتر به یک شیء با طول افقی مناسب می‌شوند (در این حالت، یک خودرو)، و همچنین می‌توانیم از یک ویدیو ترافیک به عنوان ورودی استفاده کنیم:

```
# Define the kernel sizes for erosion and dilation
erode_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 5))
dilate_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17, 11))


# Open the video file for reading
cap = cv2.VideoCapture("drive/MyDrive/traffic.flv")
```

ماسک KNN، تصویرآستانه‌گذاری شده، تصویر پس زمینهBackground و نتیجه تشخیص با مستطیل‌های محدودکننده به شکل زیر است : (Frame 6)



تفکیک‌کننده پس‌زمینهKNN ، همراه با قابلیت تفکیک بین اشیاء و سایه‌ها، در اینجا خیلی خوب کار کرده است. تمامی خودروها به صورت جداگانه شناسایی شده‌اند؛ حتی اگر برخی از خودروها به هم نزدیک باشند، به یک شناسایی ترکیبی تبدیل نشده‌اند. به طور کلی، این یک نتیجه تشخیص مفید است که ممکن است به ما اجازه بدهد تا تعداد خودروهای حرکت کننده در هر خط را بشماریم.

5. Using GMG and other background subtractor

cv2.bgsegm.createBackgroundSubtractorGMG() یک تابع در کتابخانه OpenCV است که یک نمونه از تفکیک‌کننده پس‌زمینه مبتنی بر حرکت سراسری (Global Motion-based Background Subtraction-GMG) ایجاد می‌کند. این الگوریتم بر اساس مفهوم حرکت سراسری است و به خصوص در شرایطی موثر است که دوربین ثابت است یا حرکت سراسری مشخصی در صحنه وجود دارد.

```
bg_subtractor = cv2.bgsegm.createBackgroundSubtractorGMG(initializationFrames=120,decisionThreshold=0.7)
```

در زیر توضیحات پارامترهای مهم آن آمده است:

- **initializationFrames:** این پارامتر تعداد فریم‌های اولیه را که برای مقداردهی اولیه مدل پس‌زمینه استفاده می‌شود را تنظیم می‌کند. این به الگوریتم امکان می‌دهد تا به صحنه اولیه واکنش نشان دهد.
- **decisionThreshold:** این یک آستانه استفاده شده برای تصمیم‌گیری در الگوریتم است. این بر تحمیلی پس‌زمینه تاثیر می‌گذارد.
- **smoothingRadius:** این پارامتر شعاعی را برای هموار کردن مدل پس‌زمینه تنظیم می‌کند. این کنترل می‌کند که چه میزان هموار سازی زمانی بر روی مدل پس‌زمینه اعمال شود.

الگوریتم GMG به خصوص برای شرایطی مناسب است که پس‌زمینه ثابت است و اشیاء در صحنه یک الگوی حرکت سراسری واضح دارند. این به طور معمول در برنامه‌های مراقبت و نظارت مورد استفاده قرار می‌گیرد.

توجه کنید کاست‌کننده پس‌زمینه GMG از تشخیص سایه پشتیبانی نمی‌کند. اگر برنامه شما نیاز به تشخیص سایه دارد، ممکن است می‌خواهید از یک روش متفاوت استفاده کنید یا راه حل‌های سفارشی را بررسی کنید.

ماسک GMG، تصویرآستانه‌گذاری شده و نتیجه تشخیص با مستطیل‌های محدودکننده به شکل زیر است :

تعدادی کاست‌کننده پس‌زمینه بیشتر در ماژول cv2.bgsegm این‌ها می‌توانند با استفاده از توابع زیر ایجاد شوند:

- cv2.bgsegm.createBackgroundSubtractorCNT
- cv2.bgsegm.createBackgroundSubtractorGMG
- cv2.bgsegm.createBackgroundSubtractorGSOC
- cv2.bgsegm.createBackgroundSubtractorLSBP
- cv2.bgsegm.createBackgroundSubtractorMOG
- cv2.bgsegm.createSyntheticSequenceGenerator

این توابع پارامتر detectShadows را پشتیبانی نمی‌کنند و یک کاست‌کننده پس‌زمینه بدون پشتیبانی از تشخیص سایه ایجاد می کنند. با این حال، تمامی کاست‌کننده‌های پس‌زمینه از متد apply پشتیبانی می‌کنند.