



دانشکده مکانیک، برق و کامپیوتر  
گروه هوش مصنوعی و رباتیکز

## دوره ی OpenCV

### پردازش ویدیو و حرکت

تهیه و تنظیم :

زهره عبادی

استاد راهنما :

دکتر عباس کوچاری

3.....	1. VideoWriter
5.....	2. Implementing a basic background subtractor
9.....	3. Using a MOG background subtractor
12.....	4. Using a KNN background subtractor
14.....	5. Using GMG and other background subtractor

## 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\_fourcc** است. 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.
- **seconds = 2 و fps = 40**: 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 `cap.get(4)` به ارتفاع `(CV_CAP_PROP_FRAME_WIDTH)` و `(CV_CAP_PROP_FRAME_HEIGHT)` 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 بازسازی 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 مشخص شده 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 پی‌آپی، 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 میکنیم.

```
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:

Gray background



حالا به ضبط تصاویر بیشتر ادامه می‌دهیم 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 می‌دهیم. 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 اشیاء 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 جلوگیری کننده 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 قسمت‌های سفید را در هر جایی که یک شیء در حال حرکت است، داشته باشد. حالا می‌خواهیم مرزهای قسمت‌های 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.

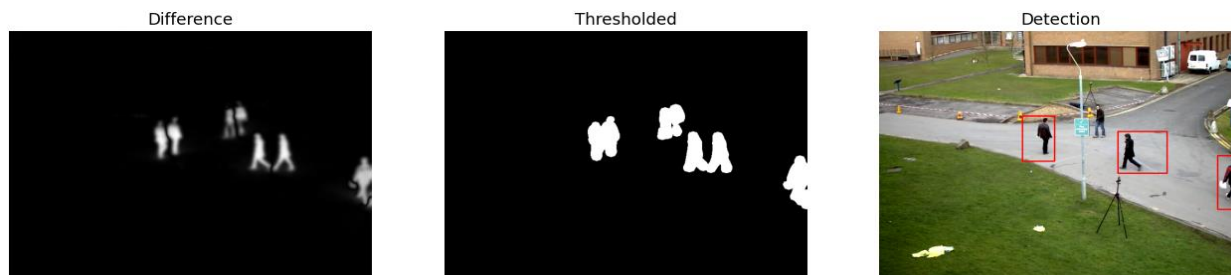
- **thresh**: تصویر دودویی که لبه‌های اشیاء در آن جستجو می‌شود.
- **cv2.RETR\_EXTERNAL**: نوع بازیابی اطلاعات از لبه‌ها. در اینجا، فقط لبه‌های خارجی اشیاء استخراج می‌شوند.

▪ **cv2.CHAIN\_APPROX\_SIMPLE**: نوع فرآیند ساده‌سازی که برای تقریب لبه‌های اشیاء استفاده می‌شود.

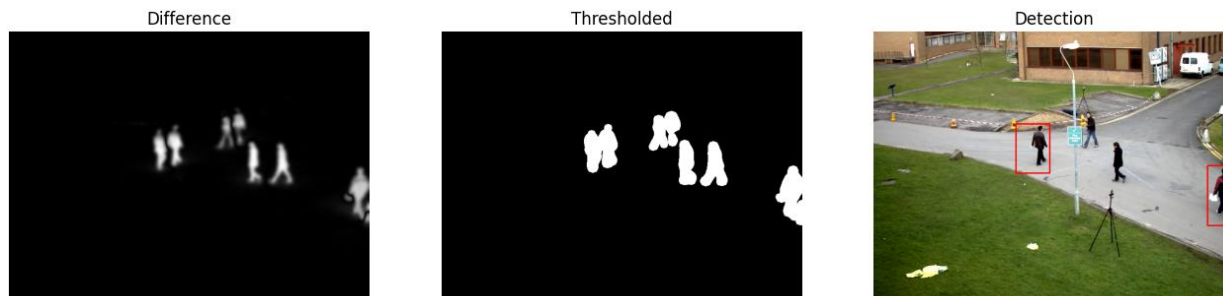
- **for c in contours**: این حلقه for برای پیمایش تمام اشیاء یا لبه‌های پیدا شده در تصویر ایجاد می‌شود.
  - **if cv2.contourArea(c) > 4000**: در اینجا، بررسی می‌شود که آیا مساحت اشیاء (لبه‌ها) بزرگ‌تر از ۴۰۰۰ پیکسل مربع است یا خیر. اگر مساحت بیشتر از این مقدار باشد، به این معناست که این اشیاء بزرگ‌تر و مهم‌تر هستند.
  - **x, y, w, h = cv2.boundingRect(c)**: این خط کد اطلاعات مربوط به مستطیل محذب (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 **(0, 255, 255)** 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 محذب (مرتبط با هر شیء) را به تصویر اصلی اضافه می‌کند. این روند برای اشیاء با مساحت بزرگ‌تر از ۴۰۰۰ 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

بسیاری از تفریق‌کننده‌های پس‌زمینه به دلیل تکنیک‌های خوشه‌بندی آماری نام‌گذاری شده‌اند که برای رویکرد یادگیری ماشینی خود به آنها تکیه می‌کنند. بنابراین، ما با بررسی یک تفریق‌کننده پس‌زمینه بر اساس تکنیک خوشه‌بندی MOG (Mixture of Gaussians) شروع می‌کنیم.

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.

به عنوان نقطه شروع، دستورالعمل حذف پس‌زمینه اصلی ما را از بخش Implementing a basic background subtractor بگیرد. 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 اسکرپت پخش شده‌اند. نزدیک به ابتدای اسکرپت، 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.BackgroundSubtractorMOG2 ایجاد کند. این تابع یک پارامتر به نام detectShadows را قبول می‌کند که ما آن را به True تنظیم 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:

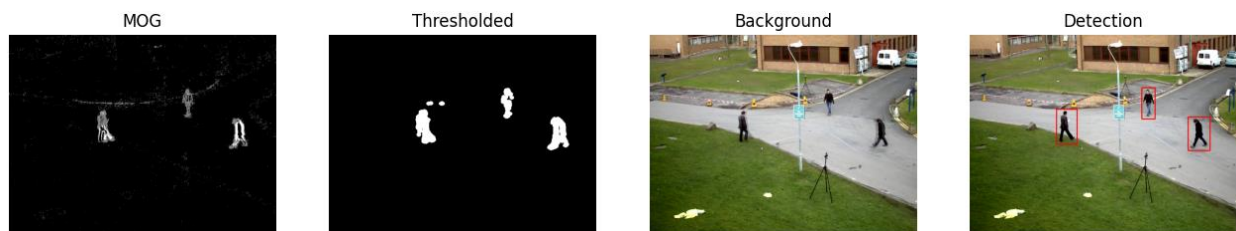
```
# Apply background subtraction to get the foreground mask  
fg_mask = bg_subtractor.apply(frame)  
# Apply thresholding to create a binary image  
_, thresh = cv2.threshold(fg_mask, 244, 255, cv2.THRESH_BINARY)  
# Apply morphological erosion and dilation to smoothen the thresholded image
```

```

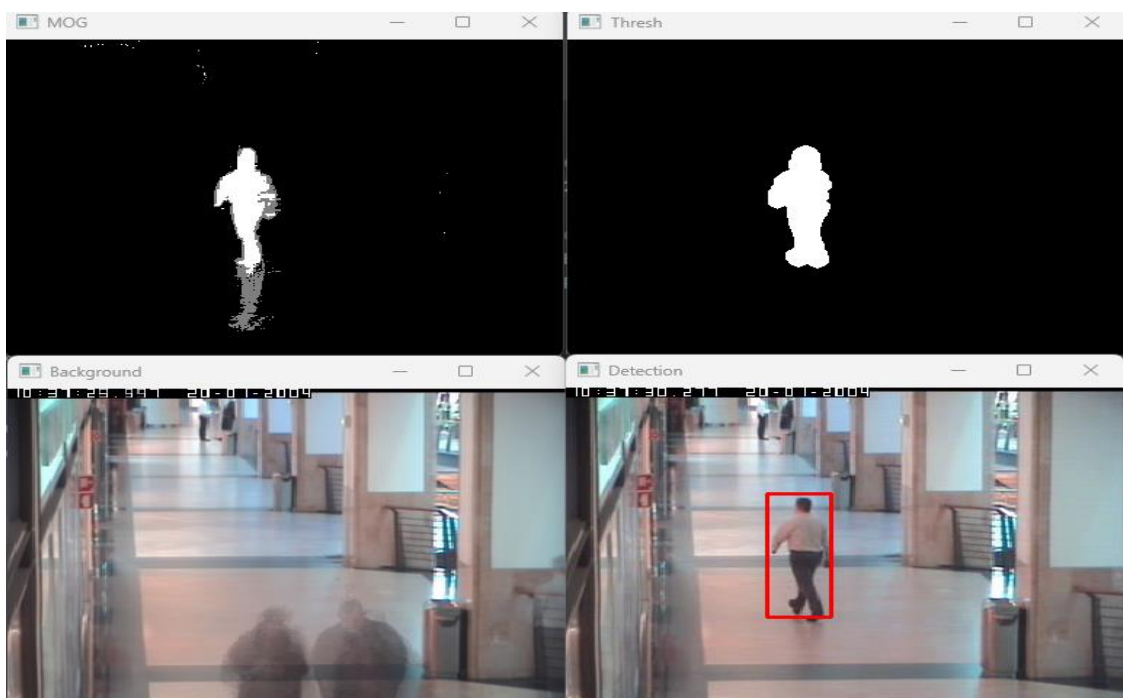
cv2.erode(thresh, erode_kernel, thresh, iterations=2)
cv2.dilate(thresh, dilate_kernel, thresh, iterations=2)
# Find contours of objects in the thresholded image
contours, hier = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
# Loop through detected contours and draw bounding rectangles 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)

```

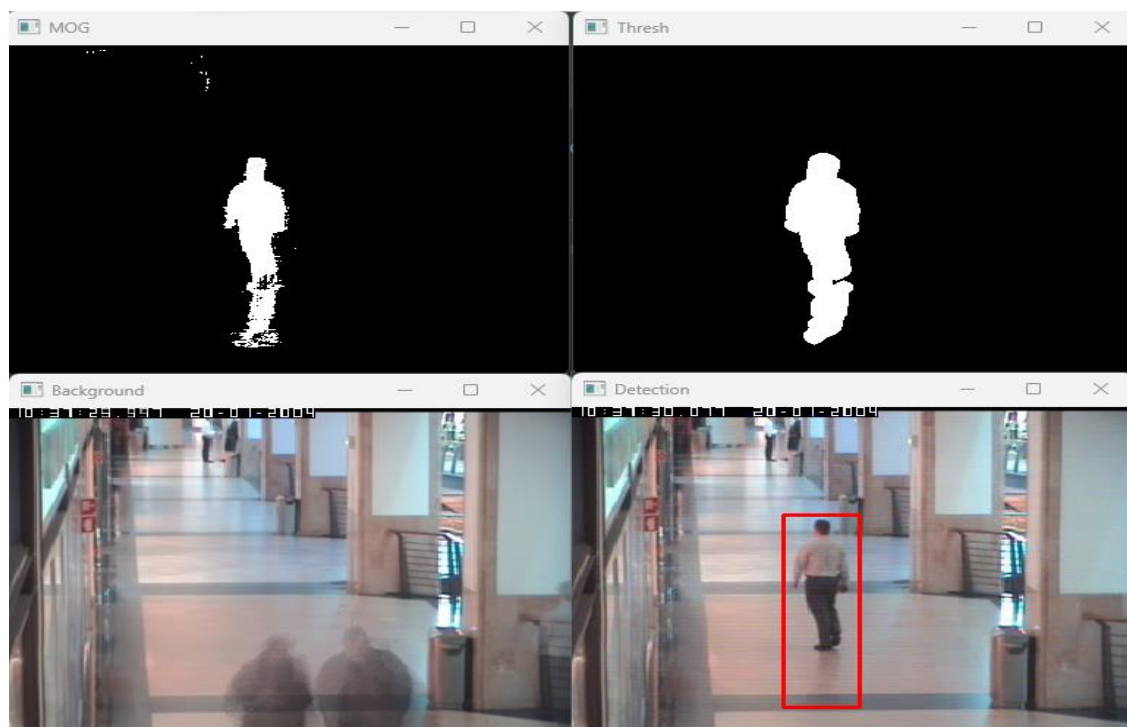
وقتی یک فریم به متد apply حذف‌کننده پس‌زمینه ارسال می‌شود، حذف‌کننده مدل داخلی خود از پس‌زمینه را به‌روز کرده و سپس یک ماسک برمی‌گرداند. ماسک برای بخش‌های پیش‌زمینه سفید (255)، برای بخش‌های سایه خاکستری (127) و برای بخش‌های پس‌زمینه سیاه (0) است. برای اهداف ما، ما سایه‌ها را به عنوان پس‌زمینه در نظر می‌گیریم، بنابراین ما یک آستانه تقریباً سفید (244) روی ماسک اعمال می‌کنیم. ماسک MOG، تصویر آستانه‌گذاری شده، تصویر پس‌زمینه Background 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 پس‌زمینه Background (تصویر پایین سمت چپ) و نتیجه تشخیص (تصویر 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 حال 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.

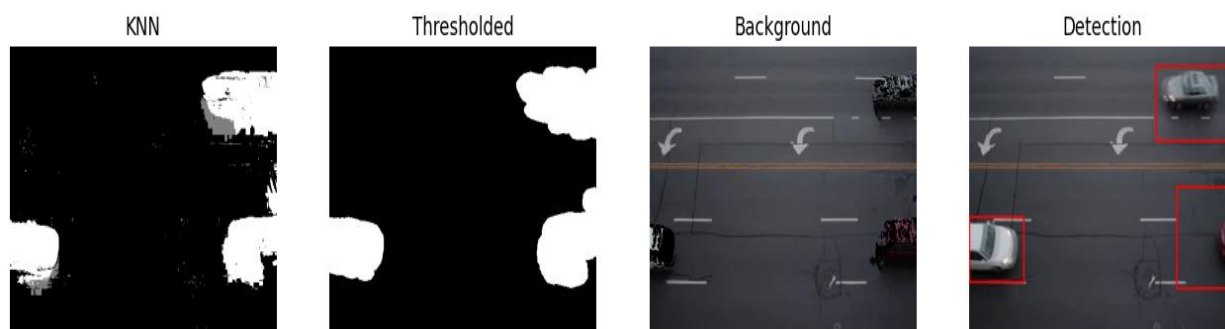
بیاد داشته باشید که `cv2.createBackgroundSubtractorMOG2` یک نمونه جدید از کلاس `cv2.createBackgroundSubtractorKNN` برمی‌گرداند. به مانند آن، `cv2.createBackgroundSubtractorKNN` یک نمونه جدید از کلاس `cv2.createBackgroundSubtractorKNN` برمی‌گرداند. هر دوی این کلاس‌ها زیرکلاس‌های `cv2.createBackgroundSubtractor` 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، تصویر آستانه‌گذاری شده، تصویر پس زمینه Background و نتیجه تشخیص با مستطیل‌های محدودکننده به شکل زیر است : (Frame 6)



تفکیک‌کننده پس‌زمینه KNN، 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 حرکت‌کننده در هر خط را بشماریم.

**Using GMG and other background subtractor .5**