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

- از کتاب :

- 3 Joseph Howse Learning OpenCV 4 Computer Vision with Python 3, 3rd Edition by Joe Minichino -chapter8

1. VideoCapture:

cv2.VideoCapture یک کلاس در کتابخانه OpenCV است که برای گرفتن جریان‌های ویدیویی از منابع مختلف مانند فایل‌های ویدیویی، دنباله‌های تصاویر یا ورودی‌های دوربین زنده استفاده می‌شود. این کلاس واسطی فراهم می‌کند تا با دستگاه‌های ورودی ویدیو کار کنید و امکان خواندن فریم‌ها از آنها را فراهم می‌کند.

در ادامه توضیحات ابتدایی در مورد cv2.VideoCapture آمده است:

1. مقداردهی اولیه:

برای استفاده از cv2.VideoCapture ، ابتدا باید نمونه‌ای از این کلاس ایجاد کنید. این کار با فراهم کردن منبع به عنوان ورودی انجام می‌شود. منبع می‌تواند:

- یک شاخص دوربین (معمولاً از 0 شروع شده و به دوربین پیش‌فرض سیستم شما ارتباط دارد).

```
cap1 = cv2.VideoCapture(0)
```

- یک فایل ویدیو باشد که با ارائه مسیر فایل اشاره می‌شود.

```
cap2 = cv2.VideoCapture(r"D:\OpenCV_Course\videos\traffic.flv")
```

- یک URL از جریان ویدیویی دوربین IP.

```
cap3 =
cv2.VideoCapture("https://github.com/ZahraEk/OpenCVCourse/raw/main/videos/pedestrians.avi")
```

2. خواندن فریم‌ها:

پس از مقداردهی اولیه cv2.VideoCapture، می‌توانید شروع به خواندن فریم‌ها از آن کنید. این کار با استفاده از متد cap.read() انجام می‌شود. این متد دو مقدار را برمی‌گرداند:

- یک بولین (ret) که نشان دهنده این است که آیا یک فریم با موفقیت خوانده شده است یا خیر.
- فریم واقعی به عنوان یک آرایه NumPy..

می‌توانید این متد را در یک حلقه فراخوانی کنید تا به صورت مداوم فریم‌ها را از منبع بخوانید.

```
while True:
   ret, frame = cap.read()
```

3. آزادسازی منابع:

پس از اتمام گرفتن ویدیو، مهم است که منابع مرتبط با cv2.VideoCapture را آزاد کنید. این با استفاده از متد `cap.release()` انجام می‌شود. عدم آزادسازی صحیح منابع ممکن است منجر به مشکلاتی شود، به ویژه اگر با چندین جریان ویدیویی کار می‌کنید.

4. کنترل خطا:

مهم است که در هنگام کار با cv2.VideoCapture از کنترل خطا استفاده کنید. اگر منبع نتواست باز شود، برای ret مقدار False را برمی‌گرداند. شما باید این مورد را بررسی کنید تا اطمینان حاصل کنید که جریان ویدیویی با موفقیت مقداردهی اولیه شده است.

```
if not ret:
    print("Can't receive frame (stream end?). Exiting ...")
    break
```

5. عملیات‌های اضافی:

می‌توانید عملیات مختلفی را بر روی فریم‌های خوانده شده انجام دهید، مانند پردازش آنها، نمایش آنها، ذخیره آنها به عنوان تصاویر یا ویدیوها و غیره.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

2. VideoWriter:

```
# Define the output video file name and codec
output_file1 = r"D:\OpenCV_Course\videos\output_flip.avi"
output_file2 = r"D:\OpenCV_Course\videos\output_rectangle.avi"
fourcc = cv2.VideoWriter_fourcc(*'XVID')

# Get the frame dimensions
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Create a VideoWriter object to save the video
out1 = cv2.VideoWriter(output_file1, fourcc, 20.0,(frame_width, frame_height))
out2 = cv2.VideoWriter(output_file2, fourcc, 40.0,(frame_width, frame_height))

while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    # Flip the video vertically
    flip = cv2.flip(frame, 0)

    # Flip the video horizontally
    #flip = cv2.flip(frame, 1)

    # Flip the video both vertically and horizontally
    #flip= cv2.flip(frame, -1)

    # adding rectangle on each frame
    cv2.rectangle(frame, (200, 200), (400, 400), (255, 0, 0), 2)

    #Convert the flip video to grayscale
    #gray = cv2.cvtColor(flip, cv2.COLOR_BGR2GRAY)
```

```
# write the flipped frame
out1.write(flip)
out2.write(frame)
```

توضیح کد این بخش:

- output_file1 = r"D:\OpenCV_Course\videos\output_flip.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، 'XVID' به AVI (Audio Video Interleave) تنظیم شده است که یک کدک معمولی برای فرمت ویدیوی 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.
- frame_width = int(cap.get(3))
- 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.

- **out1 = cv2.VideoWriter(output_file1, fourcc, 20.0,(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.

- 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:
```
    print("Can't receive frame (stream end?). Exiting ...")
```

```
break
```

این کد یک شرط را بررسی می‌کند تا مطمئن شود که فریم به درستی خوانده شده است. اگر ret برابر با False باشد، این به معنی این است که فریمی به درستی خوانده نشده است، ممکن است به دلیل پایان جریان ویدیو (stream end) یا دلایل دیگر. در این صورت، برنامه پیامی چاپ می‌کند که اعلام می‌کند فریمی دریافت نشده است و سپس از حلقه خارج شده و برنامه به پایان می‌رسد.

- `flip = cv2.flip(frame, 0)` : این خط فریم را به صورت عمودی معکوس می‌کند. تابع cv2.flip() دو آرگومان می‌گیرد - فریمی که باید معکوس شود (در این حالت frame است) و یک پرچم که نوع معکوس را مشخص می‌کند. 0 معکوسی عمودی را نشان می‌دهد.
- `#flip = cv2.flip(frame, 1)`
- `#flip= cv2.flip(frame, -1)` :

دو خط بعدی (که به صورت کامنت زده شده‌اند) نشان می‌دهند که چگونه می‌توان انعکاس افقی (1) و هم افقی و هم عمودی (-1) را انجام داد.

- `cv2.rectangle(frame, (200, 200), (400, 400), (255, 0, 0), 2)`: این یک مستطیل روی فریم اصلی می کشد.

آرگومان‌های تابع (start_point, end_point, color, thickness) هستند. در این حالت، این یک مستطیل آبی با ضخامت 2 پیکسل می‌کشد که از نقطه (200، 200) تا (400، 400) است.

- `#gray = cv2.cvtColor(flip, cv2.COLOR_BGR2GRAY)` : این خط کد، فیلم معکوس شده عمودی (که در متغیر flip قرار دارد) را به تصویر سطح خاکستری تبدیل می‌کند. این عمل به وسیله تابع cv2.cvtColor() انجام می‌شود.
- `out1.write(flip)` : این خط فریم معکوس شده عمودی را به یک خروجی ویدیو (out1) می‌نویسد.
- `out2.write(frame)` : این خط فریم اصلی را به یک خروجی ویدیو دیگر (out2) 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.

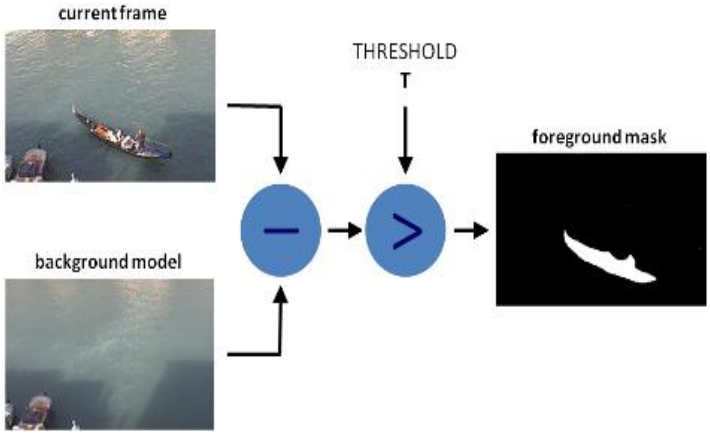
3. Implementing a basic background subtractor:

کاهش پس‌زمینه (BS) یک تکنیک متداول و گسترده برای تولید یک ماسک پیش زمینه (به نام تصویر دودویی که حاوی پیکسل‌های متعلق به اشیاء در حال حرکت در صحنه است) با استفاده از دوربین‌های استاتیک است. همانطور که از نامش پیداست، BS ماسک پیش زمینه را با انجام یک عملیات تفریق بین فریم فعلی و یک مدل پس‌زمینه محاسبه می‌کند، که شامل بخش استاتیک صحنه یا به عبارت دقیق‌تر هر چیزی که با توجه به ویژگی‌های صحنه مشاهده‌شده می‌تواند به عنوان پس‌زمینه در نظر گرفته شود.

مدل‌سازی پس‌زمینه از دو مرحله اصلی تشکیل شده است :

1. مقداردهی اولیه پس زمینه (Background Initialization)
2. به‌روزرسانی پس‌زمینه(Background Update)

در مرحله اول، مدل اولیه پس‌زمینه محاسبه می‌شود، در حالی که در مرحله دوم، این مدل به‌روزرسانی می‌شود تا با تغییرات ممکن در صحنه سازگار شود.



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

1. از دوربین، فریم‌ها را ضبط کنید.
2. 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.
3. 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.
4. 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 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:

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_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray_frame = cv2.GaussianBlur(gray_frame, (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)
```

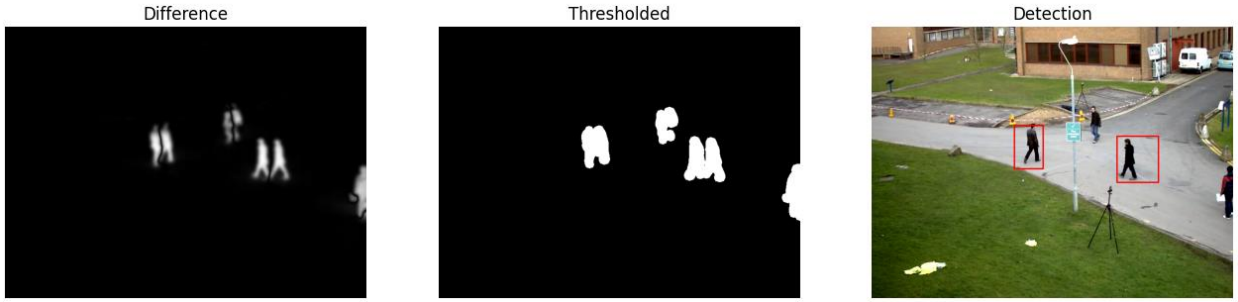
- **cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE**: در این خط کد، توابع OpenCV برای پیدا کردن لبه‌های اشیاء در تصویر دودویی thresh استفاده می‌شوند.
  - **thresh**: تصویر دودویی که لبه‌های اشیاء در آن جستجو می‌شود.
  - **cv2.RETR_EXTERNAL**: نوع بازیابی اطلاعات از لبه‌ها. در اینجا،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

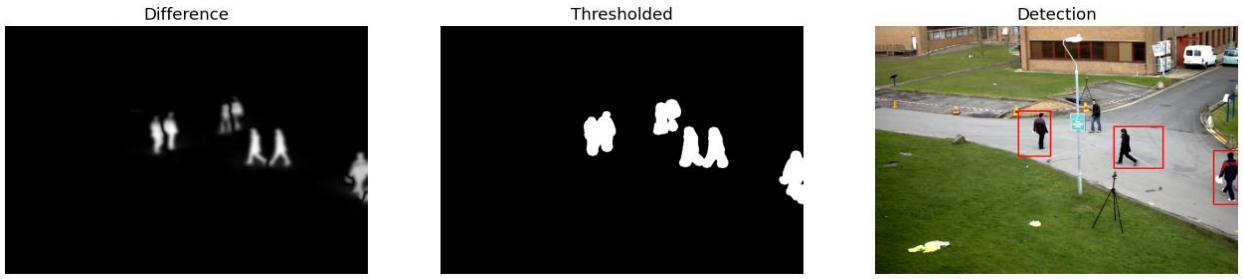
به این ترتیب، لبه‌های اشیاء را در تصویر دودویی شناسایی کرده و مستطیل‌های محدب (مرتبط با هر شیء) را به تصویر اصلی اضافه می‌کند. این روند برای اشیاء با مساحت بزرگ‌تر از ٤٠٠٠ پیکسل مربع انجام می‌شود.

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

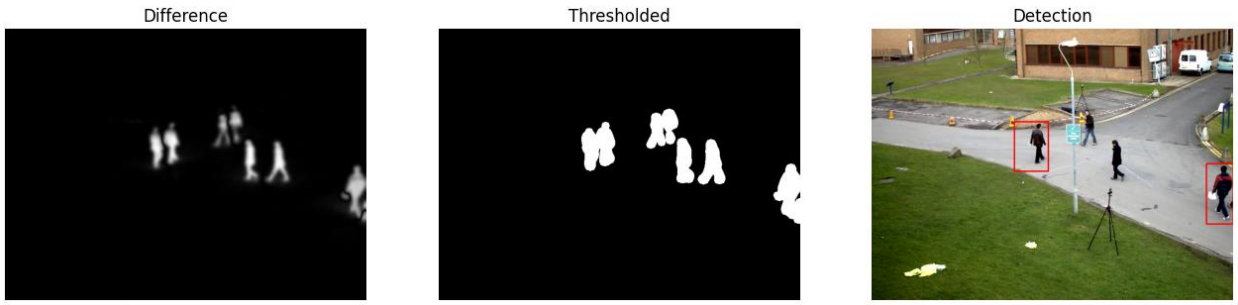
: Frame 3



: Frame 4



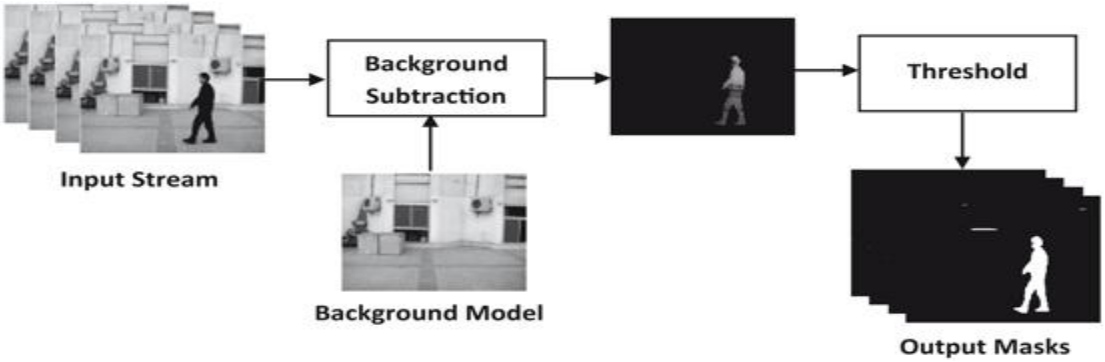
:Frame 5



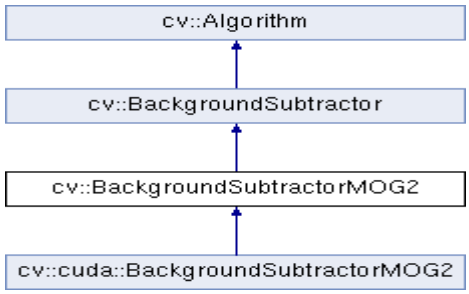
4. Using a MOG background subtractor:

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.

به طور خلاصه توضیح داده شد، این رویکرد با مدل‌سازی هر پیکسل به عنوان مجموع گاوسیان وزن‌دار شروع می‌شود که در آن وزن سهم هر گاوسی را مشخص می‌کند. شهود پشت داشتن چندین گاوسی به جای یک این است که یک پیکسل می تواند اشیاء زیادی را نشان دهد (مثلاً دانه های برف و یک ساختمان پشت سر). به عنوان مثال، هنگامی که ما یک گوسی با وزن بزرگ و یک انحراف معیار کوچک به دست می آوریم، به این معنی است که شی توصیف شده اغلب ظاهر می شود و بین فریم ها تغییر نمی کند و بنابراین احتمالاً بخشی از پس زمینه است. و این نحوه عملکرد الگوریتم است. هر پیکسل ورودی با مدل های موجود بررسی می شود. در صورت تطابق، وزن، میانگین و انحراف استاندارد مدل خود را به روز می کنیم و اگر وزن تقسیم بر انحراف استاندارد بزرگ باشد، پیکسل را به عنوان پس زمینه در غیر این صورت به عنوان پیش زمینه طبقه بندی می کنیم.



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))
```

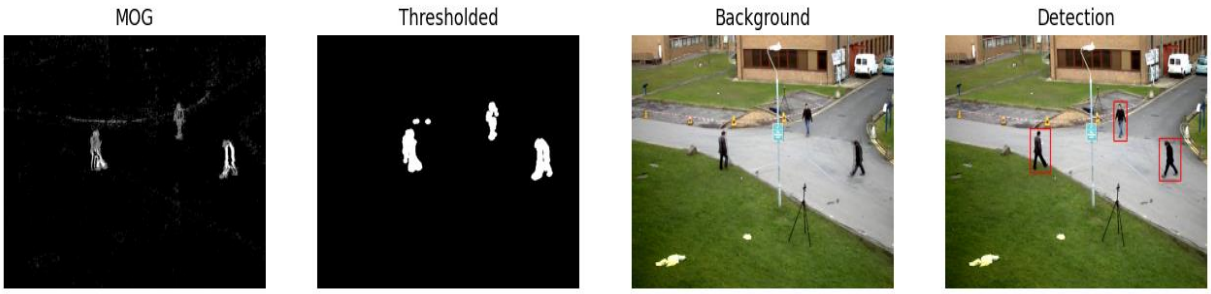
توجه داشته باشید که OpenCV یک تابع به نام cv2.createBackgroundSubtractorMOG2 ارائه داده است تا یک نمونه از cv2.BackgroundSubtractorMOG2ایجاد کند. این تابع یک پارامتر به نام detectShadows را قبول می‌کند که ما آن را به Trueتنظیم می‌کنیم تا مناطق سایه به عنوان سایه تشخیص داده شوند و به عنوان قسمتی از پیش‌زمینه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

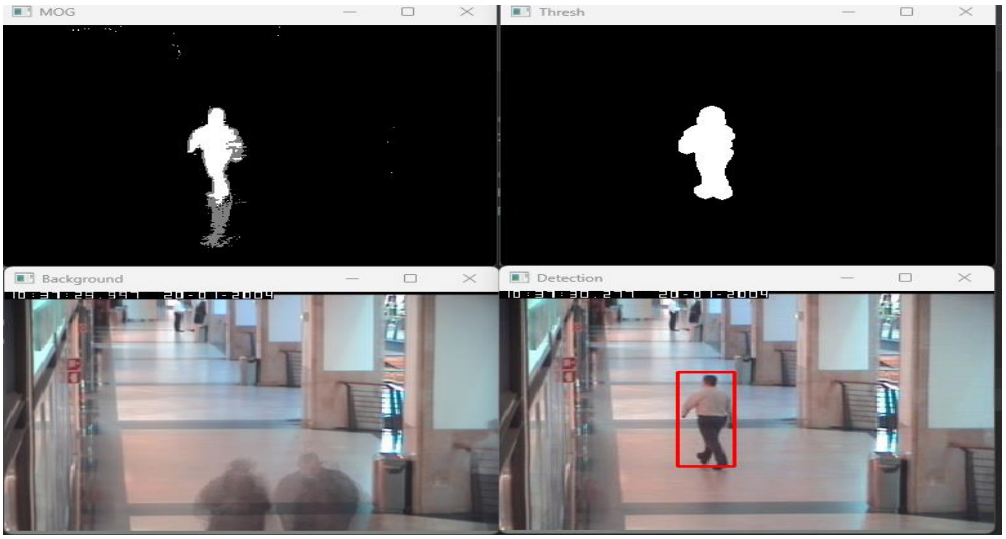
```
# Apply background subtraction to get the foreground mask
fg_mask = bg_subtractor.apply(frame)
```

```
# 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)
```

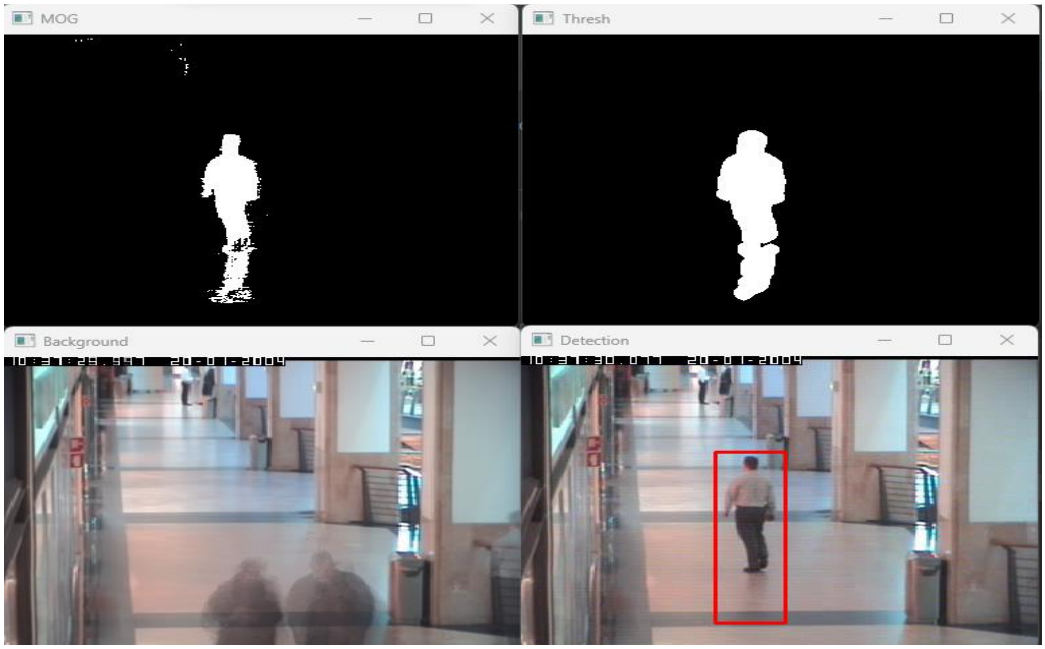
وقتی یک فریم به متد apply حذف‌کننده پس‌زمینه ارسال می‌شود، حذف‌کننده مدل داخلی خود از پس‌زمینه را به‌روز کرده و سپس یک ماسک برمی‌گرداند. ماسک برای بخش‌های پیش‌زمینه سفید (255)، برای بخش‌های سایه خاکستری (127) و برای بخش‌های پس‌زمینه سیاه (0) است. برای اهداف ما، ما سایه‌ها را به عنوان پس‌زمینه در نظر 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.

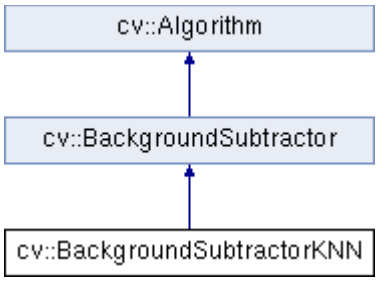
5. Using a KNN background subtractor

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 با مقایسه هر پیکسل در فریم فعلی با K نزدیکترین همسایه‌اش در مدل پس‌زمینه کار می‌کند که به صورت پویا به‌روزرسانی می‌شود.



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.

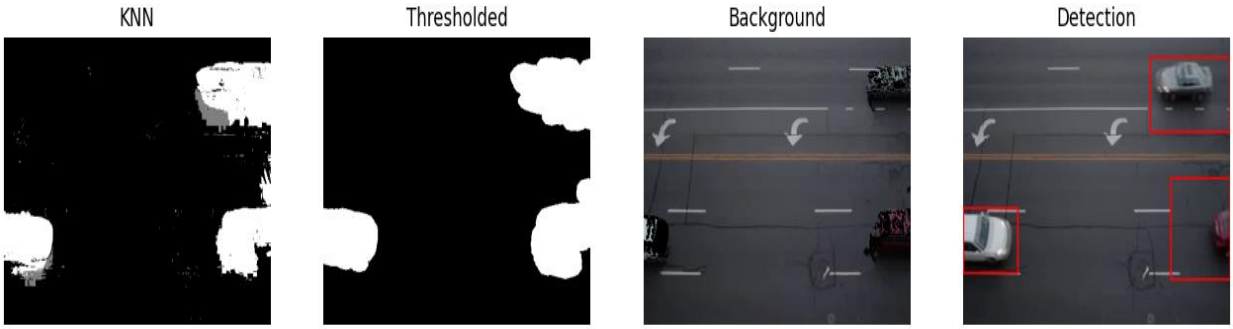
بیاد داشته باشید که cv2.createBackgroundSubtractorMOG2 یک نمونه جدید از کلاس cv2.createBackgroundSubtractorMOG2 برمی‌گرداند. به مانند آن، cv2.createBackgroundSubtractorKNN یک نمونه جدید از کلاس cv2.createBackgroundSubtractorKNN برمی‌گرداند. هر دوی این کلاس‌ها زیرکلاس‌های cv2.createBackgroundSubtractor 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)

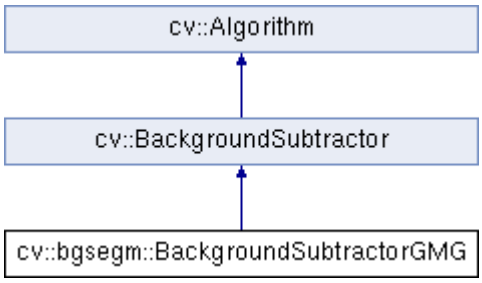


تفکیک‌کننده پس‌زمینهKNN ، همراه با قابلیت تفکیک بین اشیاء و سایه‌ها، در اینجا خیلی خوب کار کرده است. تمامی خودروها به صورت جداگانه شناسایی شده‌اند؛ حتی اگر برخی از خودروها به هم نزدیک باشند، به یک شناسایی ترکیبی تبدیل نشده‌اند. به طور کلی، این یک نتیجه تشخیص مفید است که ممکن است به ما اجازه بدهد تا تعداد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

6. 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.

این الگوریتم تخمین تصویر پس‌زمینه آماری و تقسیم‌بندی بیزین پیکسل به پیکسل را ترکیب می‌کند.از تعداد اولیه‌ای از فریم‌ها (به طور پیش‌فرض 120 فریم) برای مدل‌سازی پس‌زمینه استفاده می‌کند. از الگوریتم تقسیم‌بندی احتمالاتی پیش‌زمینه استفاده می‌کند که با استنباط بیزین، اشیاء ممکن در پیش‌زمینه را شناسایی می‌کند. این تخمینات تطبیقی هستند؛ مشاهدات جدید بیشترین وزن را دارند تا با نور متغیر سازگار شوند. چندین عملیات فیلترینگ مورفولوژیک مانند بسته شدن و باز شدن برای حذف نویز غیرمطلوب انجام می‌شود. در ابتدای تعدادی از اولین فریم‌ها، یک پنجره سیاه خواهید داشت.




```
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.

7. Tracking_colorful_objects:

ما دیدیم که کاشف پس زمینه (background subtraction) می‌تواند یک تکنیک موثر برای شناسایی اشیاء در حال حرکت باشد؛ اما ما می‌دانیم که این تکنیک محدودیت‌های ذاتی دارد. به طور خاص، این فرض می‌کند که پس‌زمینه فعلی می‌تواند بر اساس فریم‌های گذشته پیش‌بینی شود. این فرض شکننده است. به عنوان مثال، اگر دوربین حرکت کند، مدل پس‌زمینه کاملاً منسوخ می‌شود. بنابراین، در یک سیستم ردیابی قوی، ساختن نوعی مدل از اشیاء پیش زمینه به جای پس زمینه مهم است.

برای ردیابی اشیاء، نیازهای ما کمی متفاوت هستند. اگر ما داریم خودروها را ردیابی می‌کنیم، ما نیاز داریم که برای هر خودرو در صحنه یک مدل متفاوت داشته باشیم تا خودروی قرمز و خودروی آبی با هم قاطی نشوند. ما می‌خواهیم حرکت هر خودرو را به صورت جداگانه ردیابی کنیم. هنگامی که ما یک شیء متحرک را شناسایی کردیم (توسط background subtraction یا روش های دیگر)، می‌خواهیم شیء را به گونه‌ای توصیف کنیم که آن را از سایر اشیاء در حرکت متمایز کند. که به این ترتیب، ما می‌توانیم به شناسایی و ردیابی شیء ادامه دهیم، حتی اگر با یک شیء دیگر تلاقی کند.

یک هیستوگرام رنگ ممکن است به عنوان یک توصیف منحصر به فرد عمل کند. به طور اساسی، هیستوگرام رنگ یک تخمین از توزیع احتمال رنگ پیکسل‌ها در شیء است. به عنوان مثال، هیستوگرام ممکن است نشان دهد که هر پیکسل در شیء احتمال جسم به احتمال 10٪ آبی است. این هیستوگرام بر اساس رنگ‌های واقعی مشاهده شده در ناحیه شیء در یک تصویر مرجع است. به عنوان مثال، تصویر مرجع می‌تواند فریم ویدیویی باشد که در آن ابتدا شیء در حرکت را تشخیص دادیم

نسبت به دیگر روش‌های توصیف یک شیء، هیستوگرام رنگ برخی ویژگی‌هایی دارد که به ویژه در زمینه ردیابی حرکت جذاب هستند. هیستوگرام به عنوان یک جدول جستجو عمل می کند که مستقیماً مقادیر پیکسل را به احتمالات نگاشت می کند، بنابراین ما را قادر می سازد از هر پیکسل به عنوان یک ویژگی با هزینه محاسباتی کم استفاده کنیم.به این ترتیب، ، ما می توانیم ردیابی را با وضوح فضایی بسیار خوب در زمان واقعی انجام دهیم. برای یافتن محتمل ترین مکان شی ای که در حال ردیابی آن هستیم، فقط باید ناحیه مورد نظر را پیدا کنیم که در آن مقادیر پیکسل با حداکثر احتمال، مطابق با هیستوگرام نقشه برداری می شود.

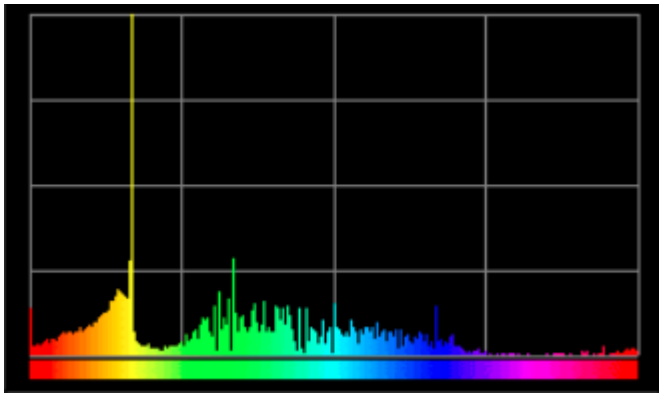
به طور طبیعی، این رویکرد توسط الگوریتمی به نام **MeanShift** استفاده می شود:

برای هر فریم در یک ویدیو، الگوریتم MeanShift با محاسبه یک مرکز بر اساس مقادیر احتمال در مستطیل ردیابی فعلی، انتقال مرکز مستطیل به این مرکز، محاسبه مجدد مرکز بر اساس مقادیر مستطیل جدید، و جابجایی مجدد مستطیل، ردیابی را به صورت مکرر انجام می‌دهد. ، و غیره. این روند تا زمانی ادامه می یابد که همگرایی حاصل شود (به این معنی که مرکز حرکت متوقف می شود یا تقریباً از حرکت باز می ماند) یا تا زمانی که به حداکثر تعداد تکرار برسد. اساساً، MeanShift یک الگوریتم خوشه‌بندی است که کاربردهایی دارد که خارج از دید حوزه بینایی ماشین می‌باشد.

Calculating and back-projecting color histograms:

برای محاسبه هیستوگرام رنگ ، OpenCV تابعی به نام cv2.calcHist ارائه می دهد. برای اعمال هیستوگرام به عنوان جدول جستجو، OpenCV تابع دیگری به نام cv2.calcBackProject را ارائه می دهد. عملیات دوم به نام بازتاب هیستوگرام است و تصویر داده شده را بر اساس هیستوگرام داده شده به یک نقشه احتمال تبدیل می‌کند. اجازه دهید ابتدا خروجی این دو تابع را تجسم کنیم و سپس پارامترهای آنها را بررسی کنیم.

یک هیستوگرام می تواند از هر مدل رنگی مانند آبی–سبز–قرمز (BGR)، مقدار اشباع رنگ (HSV) یا مقیاس خاکستری استفاده کند. برای نمونه‌های خود، ما از هیستوگرام‌های کانال رنگ اصلی (H) از مدل رنگی HSV استفاده خواهیم کرد. نمودار زیر یک نمایش از هیستوگرام رنگ سطوح نوری (Hue) است:



در محور x این نمودار، رنگ(Hue) را داریم و در محورy ، احتمال تخمینی رنگ یا به عبارت دیگر، نسبت پیکسل‌های تصویر را داریم که رنگ داده شده را دارند. نمودار بر اساس رنگ کدگذاری شده است. از چپ به راست، طرح از طریق رنگ‌های چرخه رنگ پیش می‌رود: قرمز، زرد، سبز، فیروزه‌ای، آبی، سرخابی، و در نهایت به قرمز. به نظر می رسد این هیستوگرام خاص یک شی را نشان می دهد که مقدار زیادی رنگ زرد در آن وجود دارد.

OpenCV مقادیر Hue را با بازه ای از 0 تا 179 نشان می دهد. برخی دیگر از سیستم ها از محدوده 0 تا 359 (مانند درجات یک دایره) یا از 0 تا 255 استفاده می کنند.

در تفسیر هیستوگرام های رنگی به دلیل خالص بودن مقداری احتیاط لازم است.پیکسل های سیاه و سفید خالص رنگ معنی داری ندارند. با این حال، رنگ آنها معمولاً به صورت 0 (قرمز) نشان داده می شود.

هنگامی که از cv2.calcHist برای تولید هیستوگرام رنگی استفاده می کنیم، آرایه 1 بعدی را برمی گرداند که از نظر مفهومی مشابه نمودار قبلی است. از طرف دیگر، بسته به پارامترهایی که ارائه می‌کنیم، می‌توانیم از cv2.calcHist برای تولید هیستوگرام یک کانال مختلف یا دو کانال به طور همزمان استفاده کنیم. در مورد دوم، cv2.calcHist یک آرایه دو بعدی را برمی گرداند .هنگامی که یک هیستوگرام داشته باشیم، می‌توانیم هیستوگرام را روی هر تصویری بازتاب دهیم.

cv2.calcBackProject یک تصویر پشت سر هم در قالب یک تصویر 8 بیتی در مقیاس خاکستری تولید می کند، با مقادیر پیکسل که به طور بالقوه از 0 (نشان دهنده احتمال کم) تا 255 (نشان دهنده احتمال بالا) است، بسته به اینکه چگونه مقادیر را مقیاس بندی کنیم. به عنوان مثال، جفت عکس زیر را در نظر بگیرید که یک پروجکشن به عقب و سپس تجسم نتیجه ردیابی MeanShift را نشان می دهد.

8. cv2.calcHist:

cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]]) -> hist

توضیحات پارامترها:

- **images**: تصویر ورودی (یا تصاویر). می‌تواند تصویر واحد یا یک لیست از تصاویر باشد.
- **channels**: فهرست کانال‌های رنگی که می‌خواهید هیستوگرام آن‌ها را بسازید. برای مثال، [0] برای کانال خاکستری و

  برای کانال‌های **BGR** به شکل زیر است:

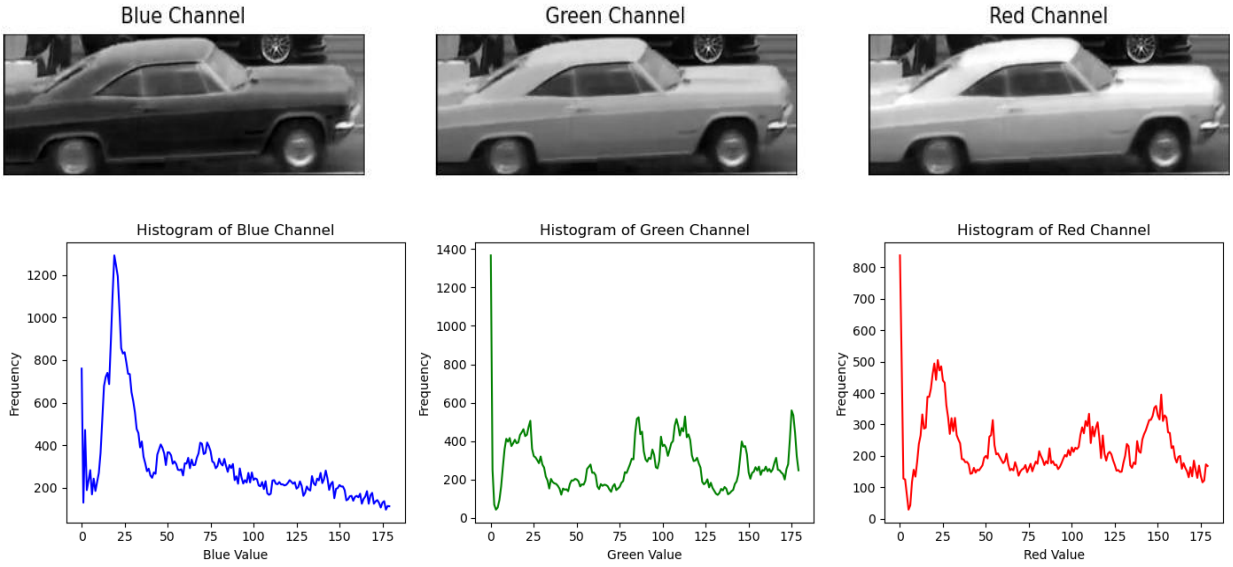
  - [0] برای محاسبه هیستوگرام کانال آبی (Blue)
  - [1] برای محاسبه هیستوگرام کانال سبز (Green)
  - [2] برای محاسبه هیستوگرام کانال قرمز (Red)

  و برای کانال های **HSV** به شکل زیر است:

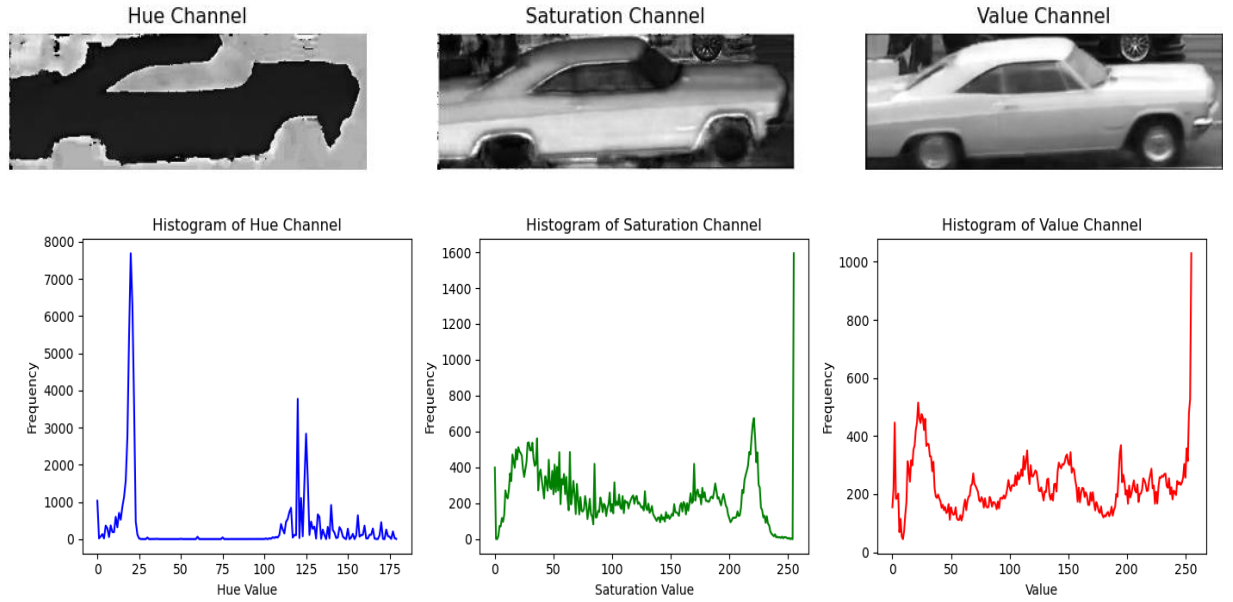
  - [0] برای محاسبه هیستوگرام کانال فام_رنگ (Hue)
  - [1] برای محاسبه هیستوگرام کانال شدت(Saturation)
  - [2] برای محاسبه هیستوگرام کانال روشنایی (Value)
- **mask**: یک ماسک برای مشخص کردن ناحیه‌ای از تصویر که در محاسبه هیستوگرام شرکت می‌کند. (اختیاری)
- **histSize**: تعداد بازه‌ها یا سطوح هیستوگرام برای هر کانال.
- **ranges**: محدوده هر بازه در هر کانال. برای مثال، [0, 256] برای کانال‌های BGR .
- **hist**: آرایه نتیجه هیستوگرام (اختیاری)
- **accumulate**: یک پرچم که مشخص می‌کند آیا هیستوگرام قبلی را افزایش دهد یا نه. (اختیاری)

با استفاده از این پارامترها، ما می‌توانیم هیستوگرام‌های مختلف را محاسبه کنیم و از آنها برای بازتاب هیستوگرام استفاده کنیم. **(hist)**

برای کانال‌های **BGR** به شکل زیر است:



و برای کانال های **HSV** به شکل زیر است:



9. cv2.calcBackProject :

cv2.calcBackProject(images, channels, hist, ranges, scale[, dst]) -> dst

- **images**: تصویر ورودی (یا تصاویر). می‌تواند تصویر واحد یا یک لیست از تصاویر باشد.
- **channels**: این پارامتر لیستی از اندیس‌های کانال‌های استفاده شده برای محاسبه بازپخش (back projection) است. به عنوان مثال، channels=[0] به این معنی است که فقط کانال اول (کانال با اندیس 0) استفاده می‌شود.
- **hist**: این پارامتر هیستوگرامی است که می‌خواهیم بازپخش کنیم. باید یک هیستوگرام یک بعدی یا دو بعدی باشد.
- **ranges**: این پارامتر محدوده مقادیر برای هر بعد است. این یک لیست از محدوده‌ها برای هر کانال است.
- **scale**: این پارامتر یک عامل مقیاس است که بر روی بازپخش اعمال می‌شود. این تنظیم کننده شدت کلی خروجی است.
- **dst**: این مقصد خروجی اختیاری است. اگر مشخص نشود، یک آرایه جدید ایجاد می‌شود.

تابع تصویر بازپخش شده را برمی‌گرداند. **(dst)**

back projection یک تکنیک پردازش تصویر است که برای شناسایی نواحی در یک تصویر که احتمالاً شامل یک ویژگی یا شیء خاص هستند، استفاده می‌شود. این تکنیک این کار را با مقایسه توزیع رنگ یا شدت پیکسل‌ها در یک تصویر با یک هیستوگرام مرجع یا مدل پیش‌تعریف شده انجام می‌دهد.

در پردازش ویدیو برای شناسایی اشیاء در حال حرکت، back projection می‌تواند یک ابزار قدرتمند باشد. در زیر نحوه کاربرد آن توضیح داده شده است:

1. **تولید هیستوگرام مدل:**
   - ابتدا، چند فریم از ویدیو در شرایط ثابت ضبط می‌شود. این فریم‌ها را به یک فضای رنگ مناسب برای برنامه ی شما تبدیل می‌کنید (معمولاً HSV) .
   - یک هیستوگرام ایجاد می‌کنید که نماینده توزیع رنگ پس‌زمینه ثابت است.
2. **مرحله ی مقدماتی:**
   - یک منطقه مورد نظر (region of interest-ROI) را در فریم ویدیو تعریف می‌کنید که می‌خواهید اشیاء در حال حرکت را شناسایی کنید.
   - ROI را به همان فضای رنگ تبدیل می‌کنید. برای مثال، HSV که برای هیستوگرام مدل استفاده شده است.
   - هیستوگرام ROI را محاسبه و آن را نرمال‌سازی می‌کنید.

3. پردازش فریم به فریم:

- برای هر فریم بعدی در جریان ویدیو:
  - فریم را به همان فضای رنگ که برای هیستوگرام مدل استفاده شده است تبدیل می‌کنید.
  - با استفاده از هیستوگرام مدل و هیستوگرامROI ، پس‌پراکنش فریم را محاسبه می‌کنید.
  - یک آستانه بندی بر روی تصویر پس‌پراکنش اعمال می‌کنید تا نواحی را که احتمال حضور ویژگی بالا است روشن سازید.
  - اختیاری، از عملیات مورفولوژی مانند تراکم و افرایش برای بهبود نتیجه و حذف نویز استفاده می‌کنید.

4. شناسایی اشیاء:

- تصویر پس‌پراکنش آستانه‌ای حالا نواحی را نشان می‌دهد که شیء در حرکت تشخیص داده شده است.
- می‌توانید از تکنیک‌هایی مانند شناسایی لبه‌ها استفاده کنید تا اشیاء را در تصویر آستانه‌ای شناسایی کنید.

5. پیگیری و پردازش پس از شناسایی (اختیاری):

- اگر نیاز باشد، می‌توانید الگوریتم‌های پیگیری اشیاء را پیاده سازی کنید تا اشیاء شناسایی شده را در طول فریم‌ها پیگیری کنید.
- از مراحل پردازش پس از شناسایی مانند فیلترینگ استفاده کرده و دقت شناسایی را افزایش دهید.

6. نمایش یا اقدام (اختیاری):

- حالا می‌توانید براساس اشیاء در حال حرکت شناسایی شده، اقداماتی انجام دهید. به عنوان مثال، می‌توانید چهارچوب‌های محدودیتی را حول آن‌ها بکشید یا رویدادهای خاصی را آغاز کنید.

این فرآیند از Back Projection برای مقایسه توزیع رنگ هر فریم با توزیع مورد انتظار از هیستوگرام مدل استفاده می کند. این یک روش قوی برای تشخیص اجسام متحرک در جریان های ویدئویی است و به طور گسترده در برنامه هایی مانند نظارت، تشخیص حرکت و ردیابی اشیا استفاده می شود.

- تفسیر ویدیو پس‌پراکنش:

نواحی روشن‌تر در ویدیو Back Projection نشان می‌دهند که احتمالاً بیشترین احتمال تعلق به ویژگی مورد نظر را دارند.نواحی تاریک‌تر نشانگر احتمال کمتری هستند.

: Mouse Callback Function (draw_rectangle)

کد زیر از کتابخانه OpenCV استفاده می‌کند تا امکان برش برنامه‌ای را در یک فریم ویدیو ارائه کند. این به کاربر اجازه می‌دهد تا با کلیک و کشیدن، یک منطقه مورد نظر (ROI) در یک فریم ویدیو تعریف کند و آن را به عنوان یک تصویر فوق العاده ذخیره کنید.

```
# initialize the list of reference points and boolean indicating
# whether cropping is being performed or not
refPt = []
sel_rect_endpoint = []
cropping = False

def click_and_crop(event, x, y, flags, param):
    # grab references to the global variables
    global refPt, cropping
    # if the left mouse button was clicked, record the starting
    # (x, y) coordinates and indicate that cropping is being performed
    if event == cv2.EVENT_LBUTTONDOWN:
        refPt = [(x, y)]
        cropping = True
    # check to see if the left mouse button was released
    elif event == cv2.EVENT_LBUTTONUP:
        # record the ending (x, y) coordinates and indicate that
```

```
    # the cropping operation is finished
    refPt.append((x, y))
    cropping = False
    # draw a rectangle around the region of interest
    cv2.rectangle(frame, refPt[0], refPt[1], (0, 0, 0), 2)
    cv2.imshow('frame', frame)
```

- refPtو sel_rect_endpoint : دو لیست هستند که برای ذخیره نقاط مرجع برای ناحیه مورد نظر و نقاط انتهای مستطیل انتخاب شده استفاده می‌شوند.
- cropping: یک متغیر بولین است که نشان می‌دهد آیا عملیات برش در حال انجام است یا خیر.
- click_and_crop(event, x, y, flags, param): یک تابع بازخوانی است که هرگاه یک رویداد ماوس رخ می‌دهد فراخوانی می‌شود. این تابع منطق برش ناحیه مورد نظر را اجرا می‌کند.
- وقتی دکمه چپ ماوس کلیک می‌شود(cv2.EVENT_LBUTTONDOWN) ، این نقطه شروع(x ، y)را ثبت کرده و cropping را بر روی True می‌گذارد.
- وقتی دکمه چپ ماوس رها می‌شود(cv2.EVENT_LBUTTONUP) ، نقاط پایانی(x ، y)را ثبت کرده، croppingرا بر روی False می‌گذارد و یک مستطیل دور ناحیه انتخابی رسم می‌کند.

```
clone = frame.copy()
cv2.namedWindow('frame')
cv2.setMouseCallback('frame', click_and_crop)
```

- cloneیک کپی از فریم فعلی است که برای تنظیم مجدد ناحیه برش اگر کلید 'r' فشرده شود استفاده می شود.
- cv2.namedWindow('frame'): یک پنجره برای نمایش فریم ویدیو ایجاد می‌کند.
- cv2.setMouseCallback('frame', click_and_crop): تابع فراخوانی ماوس را تنظیم می‌کند، بنابراین هرگاه رویدادی از ماوس در پنجره با نام 'frame' رخ دهد، تابع click_and_crop فراخوانی می‌شود.

```
while True:
    # display the image and wait for a keypress
    cv2.imshow('frame', frame)
    key = cv2.waitKey(1) & 0xFF
    # if the 'r' key is pressed, reset the cropping region
    if key == ord('r'):
```

```
        frame = clone.copy()
    # if the 's' key is pressed, save the cropped region
    elif key == ord('s'):
        if len(refPt) == 2:
            roi = clone[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
            cv2.imwrite('roi.jpg', roi)
            print("Cropped ROI saved as 'roi.jpg'.")
    # if the 'q' key is pressed, break from the loop
    elif key == ord('q'):
        break
```

- cv2.waitKey(1) & 0xFF: منتظر رویداد کلید می‌ماند (با تأخیر ۱ میلی ثانیه). عملگر بیتی اطمینان حاصل می‌کند که فقط ۸ بیت کم‌اندازه‌یاز لحاظ می‌شوند.
- درون حلقه، چندین رویداد کلید بررسی می‌شود:

1. 'r': اگر کلید 'r' فشرده شود، ناحیه برش با کپی کردن فریم اصلی از clone تنظیم مجدد می‌شود.
2. 's': اگر کلید 's' فشرده شود و دو نقطه مرجع وجود داشته باشد، ناحیه مورد نظر برش می‌شود، به عنوان 'roi.jpg' ذخیره می‌شود و پیامی چاپ می‌شود.
3. 'q': اگر کلید 'q' فشرده شود، از حلقه خارج شده و برنامه به پایان می‌رسد.

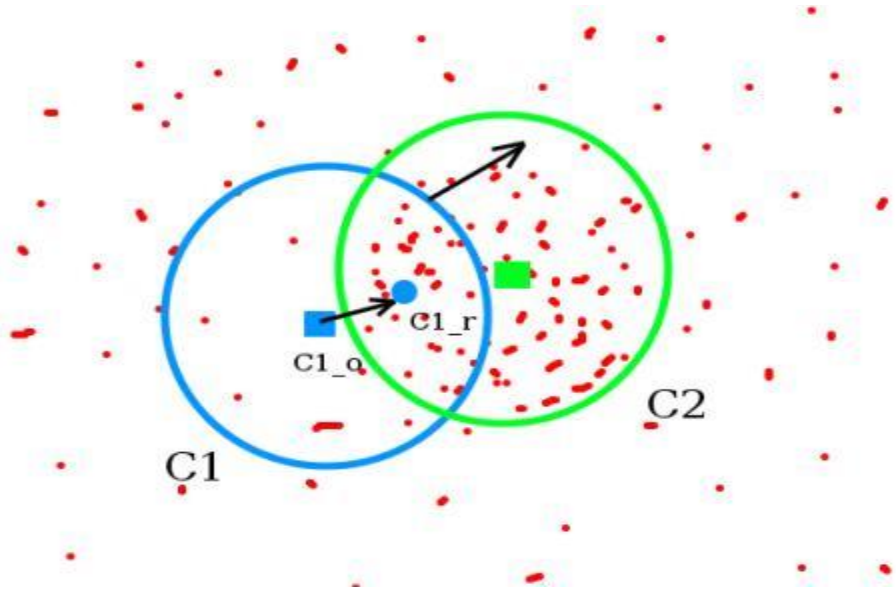
```
# if there are two reference points, then crop the region of interest
# from the image and display it
if len(refPt) == 2:
    roi = clone[refPt[0][1]:refPt[1][1], refPt[0][0]:refPt[1][0]]
    cv2.imshow('ROI', roi)
    cv2.waitKey(0)
```

- پس از حلقه، اگر دو نقطه مرجع وجود داشته باشد، ناحیه مورد نظر را از تصویر برش می‌زند و در یک پنجره جداگانه با نام 'ROI' نمایش می‌دهد.

این کد اساساً امکان برش تعاملی یک فریم ویدیو را فراهم می کند و گزینه هایی را برای تنظیم مجدد، ذخیره و خروج از برنامه ارائه می دهد.

10. MeanShift :

فرض کنید یک مجموعه از نقاط دارید. (می‌تواند توزیع پیکسلی مانند بازتاب هیستوگرام باشد). یک پنجره کوچک داده شده است (ممکن است یک دایره باشد) و شما باید این پنجره را به منطقه با بیشترین تراکم پیکسل (یا بیشترین تعداد نقاط) منتقل کنید. این در تصویر ساده زیر نشان داده شده است:



پنجره ابتدایی در دایره آبی با نام "C1" نمایش داده شده است. مرکز اصلی آن در مستطیل آبی با نام "C1_o" علامت گذاری شده است. اما اگر مرکزگرد نقاط در داخل آن پنجره را پیدا کنید، نقطه "C1_r" را (که در دایره آبی کوچک علامت گذاری شده است) که مرکز واقعی پنجره است، بدست می‌آورید. مطمئناً آن‌ها مطابقت ندارند. بنابراین پنجره را به گونه‌ای منتقل کنید که دایره پنجره جدید با مرکز قبلی همخوانی داشته باشد. دوباره مرکز جدید را بیابید. احتمالاً بیشتر اوقات مطابقت نخواهد داشت. بنابراین آن را دوباره حرکت دهید و ادامه دهید تا مرکز پنجره و مرکزگرد آن در یک مکان قرار گیرد (یا در داخل یک خطای کوچک مطلوب).

بنابراین در نهایت آنچه که بدست می‌آید، یک پنجره با توزیع بیشینه پیکسل است. این با دایره سبز علامت گذاری شده است و با نام "C2" نامیده می‌شود. همانطور که در تصویر مشاهده می‌شود، این پنجره بیشترین تعداد نقاط را دارد.

در کل، ما عموماً تصویر بازتاب هیستوگرام و مکان هدف ابتدایی را به الگوریتم MeanShift منتقل می‌کنیم. هنگامی که شیء حرکت می‌کند، بدیهی است که این حرکت در تصویر بازتاب هیستوگرام نمایان می‌شود. به عبارت دیگر، الگوریتم MeanShift پنجره را به موقعیت جدیدی با تراکم بیشینه منتقل می‌کند.

بیایید به ترتیب پیاده‌سازی مثال MeanShift خود بپردازیم:

1. مانند مثال ساده تفکیک پس‌زمینه‌ی ما، مثال MeanShift ما با گرفتن و حذف چندین فریم از دوربین شروع می‌شود تا خودکار اختصاصی تنظیمات را تنظیم کند:

```
# URL to the raw video file on GitHub
video_url = "https://github.com/ZahraEk/OpenCV-Course/raw/main/videos/car_racing.mp4"
# Download the video
video_filename = "car_racing.mp4"
urllib.request.urlretrieve(video_url, video_filename)
cap = cv2.VideoCapture(video_filename)
```

2. تا فریم بیستم، فرض می‌کنیم که تنظیمات نوردهی مناسب است؛ بنابراین، می‌توانیم یک هیستوگرام دقیق از یک ناحیه مورد نظر استخراج کنیم. کد زیر حدود ناحیه مورد نظر (ROI) را تعریف می‌کند:

```
# Capture 20 frames to allow the camera's autoexposure to adjust.
for i in range(20):
    success, frame = cap.read()
if not success:
    exit(1)
# Define an initial tracking window for a specific object.
frame_h, frame_w = frame.shape[:2]
w = frame_w // 3  # Adjust the width of the window
h = frame_h // 5  # Adjust the height of the window
x = 10  # Adjust the x-coordinate of the top-left corner of the window
y = 400  # Adjust the y-coordinate of the top-left corner of the window
track_window = (x, y, w, h)
```

این مراحل تا فریم بیستم از تصویر انجام می‌شوند. در مرحله بعدی، ما هیستوگرام را برای ناحیه مورد نظر محاسبه خواهیم کرد و با استفاده از الگوریتم جذب میانگین، شیء را ردیابی خواهیم کرد.

3. در این مرحله، کد زیر پیکسل‌های ناحیه مورد نظر (ROI) را انتخاب کرده و آن‌ها را به فضای رنگی HSV تبدیل می‌کند:

```
roi = frame[y:y+h, x:x+w]
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
```

4. سپس، ما هیستوگرام کانال Hue را برای ROI محاسبه می‌کنیم:

```
mask = None
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0, 180])
```

5. بعد از محاسبه هیستوگرام، مقادیر را به محدوده 0 تا 255 نرمال می‌کنیم:

```
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
```

6. به یاد داشته باشید که الگوریتم جذب میانگین تعدادی از تکرارها را قبل از دستیابی به همگرایی انجام می‌دهد؛ با این حال، همگرایی در اینجا تضمین نمی‌شود. بنابراین، OpenCV به ما اجازه می‌دهد تا شرایط پایانی معروف را مشخص کنیم. بیایید شرایط پایانی را به شکل زیر تعریف کنیم:

```
# Define the termination criteria:
# 10 iterations or convergence within 1-pixel radius.
term_crit = (cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS, 10, 1)
```

براساس این شرایط، الگوریتم جذب میانگین بعد از 10 تکرار (شرط تعداد) یا زمانی که شیفت دیگر بیشتر از 1 پیکسل نباشد (شرط اپسیلون) محاسبه را متوقف خواهد کرد. ترکیب پرچم‌ها (cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS) نشان می‌دهد که از هر دو این شرایط استفاده می‌کنیم.

7. حال که یک هیستوگرام محاسبه کردیم و شرایط پایانی جذب میانگین را تعریف کردیم، بیایید حلقه معمول خود را آغاز کنیم که در آن فریم‌ها را از دوربین گرفته و پردازش می‌کنیم. در هر فریم، اولین کاری که انجام می‌دهیم تبدیل آن به فضای رنگی HSV است:

```
success, frame = cap.read()
while success:
 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

8. حال که یک تصویر HSV داریم، می‌توانیم عملیات مورد انتظار هیستوگرام بازتاب را اجرا کنیم:

```
back_proj = cv2.calcBackProject([hsv], [0], h_hist, [0, 180], 1)
```

9. بازتاب، پنجره ردیابی و شرایط پایانی را می‌توان به cv2.meanShift منتقل کرد که اجرای الگوریتم MeanShift را در OpenCV انجام می‌دهد. در اینجا فراخوانی تابع است:

```
num_iters, track_window = cv2.meanShift(back_proj, track_window, term_crit)
```

10. لطفا توجه کنید که MeanShift تعداد تکرارهایی که اجرا شد را برمی‌گرداند، و همچنین پنجره جدید ردیابی را که پیدا کرده است. اختیاری است که ما می‌توانیم تعداد تکرارها را با شرایط پایانی خود مقایسه کنیم تا تصمیم بگیریم آیا نتیجه همگرا شده است یا خیر. (اگر تعداد واقعی تکرارها کمتر از حداکثر باشد، نتیجه باید همگرا شده باشد.)

در نهایت، مستطیل ردیابی به روز رسانی شده را رسم و نمایش می‌دهیم:

```
x, y, w, h = track_window
cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
cv2.imshow('back-projection', back_proj)
cv2.imshow('meanshift', frame)
```

نتیجه اجرای الگوریتم MeanShift :

11. CamShift:

در الگوریتم MeanShift یک مشکل وجود دارد. پنجره‌ی ما همیشه یک اندازه‌ی ثابت دارد، برخی اوقات حتی اگر ماشین بسیار دور یا بسیار نزدیک به دوربین باشد. این موضوع مطلوب نیست. ما نیاز داریم که اندازه پنجره با اندازه و چرخش هدف تطبیق پیدا کند. باز هم، راه‌حل از طریق آزمایشگاه‌های OpenCV پیدا شده و به CAMshift معانی متعدد به Adaptive (Continuously Meanshift) نامیده شده است که توسط گری بردسکی در مقاله‌اش با عنوان "ردیابی چهره در بینایی ماشین برای استفاده در یک رابط کاربری درکی" در سال 1998 منتشر شده است.

ابتدا این الگوریتم MeanShift را اجرا می‌کند. هنگامی که MeanShift همگرا می‌شود، اندازه پنجره را به روزرسانی می‌کند، به این صورت که $s = 2 \times \sqrt{\frac{M_{00}}{256}}$ که در آن M میانگین شیفت است .همچنین زاویه تطابق بهترین بیضی به آن را محاسبه می‌کند. دوباره MeanShiftرا با پنجره جستجوی جدید مقیاس‌شده و مکان قبلی پنجره اجرا می‌کند. این فرآیند تا زمانی که دقت مورد نیاز برآورده شود، ادامه می‌یابد

اگرچه الگوریتم CamShift پیچیده‌تر از MeanShift است، اما OpenCV رابط کاربری بسیار مشابهی برای دو الگوریتم ارائه می دهد. اصلی‌ترین تفاوت این است که فراخوانی cv2.CamShift یک مستطیل با چرخش خاصی را که چرخش شیء ردیابی شده را دنبال می‌کند، برمی‌گرداند. با چند تغییر کوچک در مثال MeanShift قبلی که در زیر آمده است، می‌توانیم به جای آن از CamShift استفاده کرده و یک مستطیل ردیابی چرخش‌دار رسم کنیم. تمام تغییرات لازم در متن زیر با مشخصات ضخیم مشخص شده‌اند:

```
import numpy as np
# Perform tracking with CamShift.
    rotated_rect, track_window = cv2.CamShift(back_proj, track_window, term_crit)
    # Draw the tracking window.
    box_points = cv2.boxPoints(rotated_rect)
    box_points = np.int0(box_points)
    cv2.polylines(frame, [box_points], True, (0, 0, 255), 2)
    # Display the resulting frame
    cv2.imshow('camshift', frame)
```

آرگومان‌های cv2.CamShift بدون تغییر هستند؛ به همان معانی و همان مقادیری که در مثال قبلی با cv2.meanShift داشتند.ما از تابع cv2.boxPoints برای پیدا کردن رئوس مستطیل ردیابی چرخشی استفاده می‌کنیم. سپس از تابع cv2.polylines برای رسم خطوطی که این رئوس را به یکدیگر متصل می‌کنند، استفاده می‌کنیم.

نتیجه اجرای الگوریتمCamShift :

12. Colorspace based tracking:

پیگیری مبتنی بر فضاهای رنگ در پردازش ویدیو، به کارگیری نمایش‌های مختلف رنگ (فضاهای رنگ) تصویر یا فریم‌های ویدیو برای جدا کردن و پیگیری رنگ‌ها یا اشیاء خاص مورد علاقه می‌پردازد. این یک تکنیک قدرتمند است که به طور معمول در وظایف دید کامپیوتری مورد استفاده قرار می‌گیرد.

در زیر خلاصه‌ای از نکات کلیدی مرتبط با پیگیری مبتنی بر فضاهای رنگ در پردازش ویدیو با استفاده از OpenCV آمده است:

1. **فضاهای رنگ:**

- نمایش‌های مختلف رنگ) مانندRGB ، HSV، YUVو غیره (رنگ‌ها را به شکل‌های مختلف نمایش می‌دهند و هرکدام برای برنامه‌های خاص خود مزایا دارند.

2. **فضای رنگHSV** :

- اغلب از فضای رنگ HSV (رنگ، اشباع، مقدار) برای پیگیری مبتنی بر رنگ استفاده می‌شود.
- رنگ (Hue) نوع رنگ را نمایش می‌دهد (مانند قرمز، سبز، آبی و غیره).
- اشباع (Saturation) شدت و توانایی رنگ را نمایش می‌دهد.
- مقدار (Value) روشنایی رنگ را نشان می‌دهد.

3. **آستانه‌گذاری رنگ:**

- در این مرحله، با تنظیم محدوده‌های خاص از مقادیر رنگ، رنگ‌ها یا اشیاء خاصی جدا می‌شوند.
- در فضای رنگی HSV ، تعیین محدوده‌های خاص از نمایانگی‌ها، اشباع‌ها و مقادیر برای فیلتر کردن رنگ‌های خاص بسیار آسان است.

4. **ترکیب کنترلی برای تنظیمات:**

- عناصر رابط کاربری گرافیکی مانند (trackbar) در OpenCV می‌توانند به طور پویا محدوده رنگی را برای پیگیری تنظیم کنند.
- این اجازه را می‌دهد که پارامترهای تشخیص رنگ به صورت زمان واقعی تنظیم شوند.

5. **ماسکینگ و عملیات بیتویس:**

- پس از تنظیم محدوده رنگ، یک ماسک ساخته می‌شود تا نواحی که رنگ مشخص شده وجود دارد را مشخص کند.
- عملیات بیتویس می‌تواند برای ترکیب ماسک با فریم اصلی برای جدا کردن رنگ مورد نظر استفاده شود.

6. **پیگیری در زمان واقعی:**

- الگوریتم تشخیص رنگ می‌تواند به صورت زمان واقعی در فریم‌های ویدیویی اعمال شود که اجازه پیگیری دینامیک رنگ مورد نظر را می‌دهد.

7. **نمونه‌های کاربرد:**

- پیگیری بر اساس رنگ به طور گسترده در برنامه‌های مختلفی مانند شناسایی اشیاء، تشخیص حرکات دست، رباتیک و واقعیت افزوده استفاده می‌شود.

8. **کارایی و انعطاف‌پذیری:**

- استفاده از پیگیری مبتنی بر فضاهای رنگ به یک رویکرد انعطاف‌پذیر برای هدفگذاری اشیاء یا رنگ‌های خاص بدون تحت تأثیر قرارگیری زیاد از شرایط نورپردازی می‌دهد.

کلیتاً، پیگیری مبتنی بر فضاهای رنگ در پردازش ویدیو یک تکنیک چندکاره است که اجازه تشخیص دقیق و پیگیری رنگ‌ها یا اشیاء خاص در یک جریان ویدیویی را می‌دهد. این یک ابزار اساسی در برنامه‌های دید کامپیوتری است که بر اطلاعات رنگ برای تجزیه و تحلیل و تصمیم‌گیری نیاز دارند.

این کد یک برنامه ساده برای تشخیص و نمایش رنگ‌ها در یک ویدیو است. این به وسیله تنظیم محدوده رنگ در فضای رنگی HSV انجام می‌شود. حال به توضیح اجزای این کد می‌پردازم:

1. :def on_change(x)
   - تابع on_change تعریف شده است که یک پارامتر x دریافت می‌کند، اما چیز خاصی انجام نمی‌دهد.

```
def on_change(x):
    pass
```

2. تعریف پنجره GUI و ایجاد trackbar ها برای تنظیم محدوده‌های رنگ:

```
# Create a GUI window with trackbars
cv2.namedWindow("Frame")
cv2.createTrackbar("Hue Lower", "Frame", 0, 255, on_change)
cv2.createTrackbar("Hue Upper", "Frame", 0, 255, on_change)
cv2.createTrackbar("Saturation Lower", "Frame", 0, 255, on_change)
cv2.createTrackbar("Saturation Upper", "Frame ", 0, 255, on_change)
cv2.createTrackbar("Value Lower", "Frame ", 0, 255, on_change)
cv2.createTrackbar("Value Upper", "Frame ", 0, 255, on_change)
```

   - این قسمت برای ایجاد پنجره گرافیکی و trackbar ها برای تنظیم محدوده‌های رنگ در فضای رنگی HSV است.

3. در حلقه while ، مقادیر اولیه trackbar ها گرفته می‌شوند:

```
while True:
    k = cv2.waitKey(1)
    if k == 27:
        break
h_lower = cv2.getTrackbarPos("Hue Lower", "Frame")
h_upper = cv2.getTrackbarPos("Hue Upper", "Frame")
s_lower = cv2.getTrackbarPos("Saturation Lower", "Frame")
s_upper = cv2.getTrackbarPos("Saturation Upper", "Frame ")
v_lower = cv2.getTrackbarPos("Value Lower", "Frame ")
v_upper = cv2.getTrackbarPos("Value Upper", "Frame ")
```

4. تعریف محدوده‌های پایین و بالای مقادیر HSV :

```
# Define the lower and upper range of HSV values
lower = np.array([h_lower, s_lower, v_lower])
upper = np.array([h_upper, s_upper, v_upper])
```

2. در حلقه while بعدی:

- فریم بعدی از ویدیو گرفته می‌شود.
- تبدیل فریم به فضای رنگی HSV انجام می‌شود.
- یک ماسک بر اساس محدوده مشخص شده ایجاد می‌شود.
- ماسک روی فریم اصلی اعمال می‌شود.
- فریم پردازش شده با فیلتر میانه‌ای بر روی آن اعمال می‌شود.
- فریم پردازش شده و فریم اصلی نمایش داده می‌شود.
- در صورتی که کلید 'Esc' فشرده شود، حلقه متوقف می‌شود.

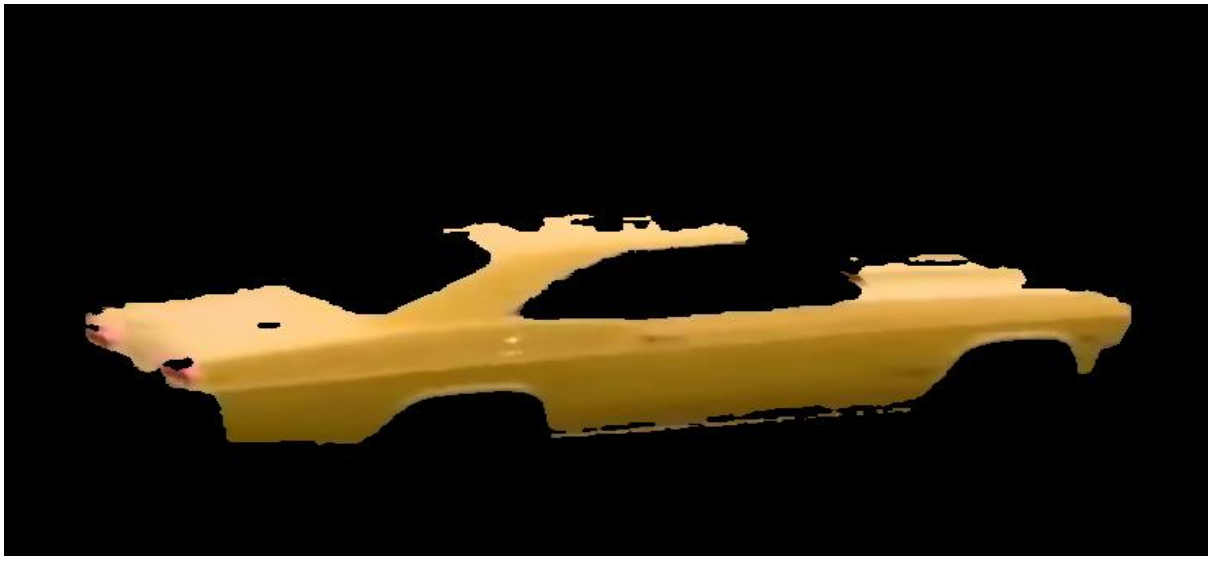
```
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Convert frame to HSV colorspace
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Create a mask using the specified range
    mask = cv2.inRange(hsv_frame, lower, upper)
    # Apply the mask to the original frame
    res = cv2.bitwise_and(frame, frame, mask=mask)
    res = cv2.medianBlur(res, ksize=3)
```

```
    # Display original frame and color-detected frame
    cv2.imshow('Frame', frame)
    cv2.imshow('Color Detector', res)
    k = cv2.waitKey(30)
    if k == 27:
        break
```

5. در انتها، ویدیو capture رها شده و پنجره‌های OpenCV بسته می‌شوند. محدوده‌های پایین و بالای مقادیر HSV نیز چاپ می‌شوند.

```
# Release video capture object and close OpenCV windows
cap.release()
cv2.destroyAllWindows()
print(f"Lower Range (H, S, V): {lower}")
    print(f"Upper Range (H, S, V): {upper}")
```

این کد به شما این امکان را می‌دهد تا با استفاده از trackbar ها، محدوده مشخصی از رنگ‌ها را در فضای رنگی HSV تعیین کنید و در زمان واقعی نمایش دهید.



13. Frame_differencing:

وقتی یک پخش زنده ویدیویی را در نظر می گیریم، تفاوت بین فریم های متوالی اطلاعات زیادی به ما می دهد. مفهوم نسبتاً ساده است! ما فقط تفاوت بین فریم های متوالی را می گیریم و تفاوت ها را نمایش می دهیم.

فقط قسمت های متحرک در ویدئو هایلایت می شوند. این یک نقطه شروع خوب به ما می دهد تا ببینیم چه مناطقی در ویدیو در حال حرکت هستند.

این کد یک برنامه ساده برای تشخیص و نمایش حرکت اشیاء در یک ویدیو است. در زیر توضیحاتی درباره اجزای کد آمده است:

1. تعریف متغیرهایی برای نگهداری فریم‌ها:
   - cur_frame: فریم فعلی
   - prev_frame: فریم قبلی
   - next_frame: فریم بعدی

```
cur_frame, prev_frame, next_frame = None, None, None
```

2. تابع frame_diff(prev_frame, cur_frame, next_frame):
   - این تابع تفاوت بین فریم فعلی و بعدی را محاسبه می‌کند.
   - ابتدا تفاوت مطلق بین فریم بعدی و فعلی (diff_frames1) و تفاوت مطلق بین فریم فعلی و قبلی (diff_frames2) محاسبه می‌شود.
   - سپس نتیجه عملیات AND بیتی بین دو تصویر حاصل برگشت داده می‌شود.

```
# Function to calculate frame difference
def frame_diff(prev_frame, cur_frame, next_frame):
 # Absolute difference between current frame and next frame
 diff_frames1 = cv2.absdiff(next_frame, cur_frame)
 # Absolute difference between current frame and previous frame
 diff_frames2 = cv2.absdiff(cur_frame, prev_frame)
 # Return the result of bitwise 'AND' between the above two resultant images
return cv2.bitwise_and(diff_frames1, diff_frames2)
```

- در حلقه while اصلی:
  - ابتدا بررسی می‌شود که فریم به درستی خوانده شده باشد یا خیر.
  - فریم‌ها به‌روزرسانی می‌شوند.
  - فریم‌ها به فضای رنگی خاکستری تبدیل می‌شوند.
- اگر فریم قبلی موجود باشد:
  - فریم با حرکت اشیاء نمایش داده می‌شود.

```
 while success:
 # Check if frame is read correctly (ret is True)
 if not success:
 print("Can't receive frame (stream end?). Exiting ...")
 break

 # Update frames
 prev_frame = cur_frame
 cur_frame = next_frame
 next_frame = cv2.cvtColor(frame, cv2.COLOR_RGB2GRAY)
 # Check if previous frame is available
 if prev_frame is not None:
 # Display the frame with object movement
```

```
    cv2.imshow('Frame', frame)
    cv2.imshow('Object Movement', frame_diff(prev_frame, cur_frame, next_frame))
```

این کد ویدیویی را باز می‌کند، هر فریم را از ویدیو می‌خواند و تفاوت‌های بین فریم‌ها را محاسبه می‌کند و نمایش می‌دهد. اگر کلید Escape فشرده شود یا ویدیو به اتمام برسد، برنامه خاتمه می‌یابد.

فریم با حرکت اشیاء نمایش داده می‌شود :