پروژه درس دید کامپیوتری

خلاصه سازی ویدیو

نگارش

مصطفی نوروزی

استاد درس

دکتر محمدزاده

فهرست مطالب

فهرست جدول‌ها

فهرست تصویرها

فصل ۱ مقدمه

امروزه تعداد بسیار زیادی دوربین در اماکنی همچون مراکز حمل و نقل، ATM، نظامی، تجاری، خانگی و آموزشی مورد استفاده قرار گرفته اند .تصاویر این دوربین ها به صورت ۲٤ ساعته در حال ضبط است و باعث استفاده از حجم عظیمی از حافظه شده است در حالی که تنها بخش بسیار اندکی از آنها توسط انسان مشاهده میشود. هنگامی که شخصی بخواهد در ویدیویی جستجو کند (مانند تعقیب یک شخص خاص با دوربین های نظارتی) بایستی مدت بسیار زیادی صرف کند تا شخص مورد نظر را بیابد و اگر اهمال ورزد ممکن است شخص مورد نظر را از دست بدهد. بدیهی است که این امر صرفا با استفاده از نیروی انسانی غیر قابل انجام است.

۱-۱ تعریف پروژه

در این پروژه با استفاده از روش های دید کامپیوتری، ویدیو های در نظر گرفته شده خلاصه می شوند. این ویدیو ها شامل تصاویری از ماشین های در حال حرکت است. همچنین با استفاده از روش های یادگیری ماشین رنگ ماشین ها تشخیص داده می شود و در قسمتی دیگر تعداد ماشین ها در حال حرکت ردیابی و شمارش می شوند.

فصل ۲ مبانی نظری

در این فصل سعی خواهد شد روشها و مفاهیمی که در این پروژه برای خلاصه سازی ویدیو استفاده شده است، معرفی و توضیح داده شوند.

۲-۱ به دست آوردن تصویر پس زمینه

در این قسمت ابتدا با استفاده از روش فیلتر میانه تصویر پس زمینه به دست آمده است. ابتدا ۲۵ فریم به صورت تصادفی از ویدیو انتخاب و در یک لیست ذخیره شده است. در ادامه برای هر یک از پیکسل ها میانه ان را به دست اورده شده و عکس پس زمینه نهایی به دست امده است. این روش با استفاده از تابع getBG پیاده شده است. در شکل ۱ و ۲ تصاویر پس زمینه به دست آمده از دو ویدیو مشاهده میشود.

شکل ۱- تصویر پس زمینه ویدیو ۱

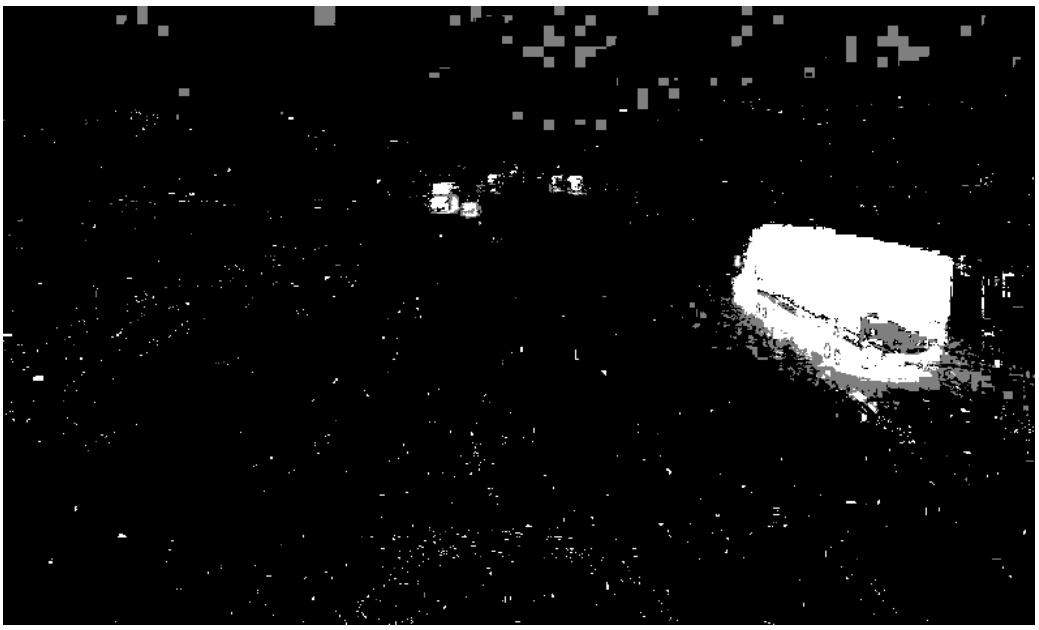
شکل ۲ – تصویر پس زمینه ویدیو ۲

۲-۲ حذف پس زمینه

یکی از مراحل انجام پروژه ، تشخیص اجسام متحرک و مکان آن ها است. برای این کار روش های مختلفی وجود دارد. با توجه به این که در این پروژه دوربین ثابت و در نتیجه تصویر پس زمینه ثابت است، یکی از ساده ترین روش ها، استفاده از روش MOG۲ است که تابع آماده آن در پایتون وجود دارد. این روش مبتنی بر روش mixture of guassian است.

ورودی های این تابع عبارت اند از :

- Varthreshold: آستانه مورد نظر برای حذف داده های پس زمینه
- History: تعداد فریم های استفاده شده برای حذف پس زمینه
- detectShadow : حذف یا عدم حذف سایه



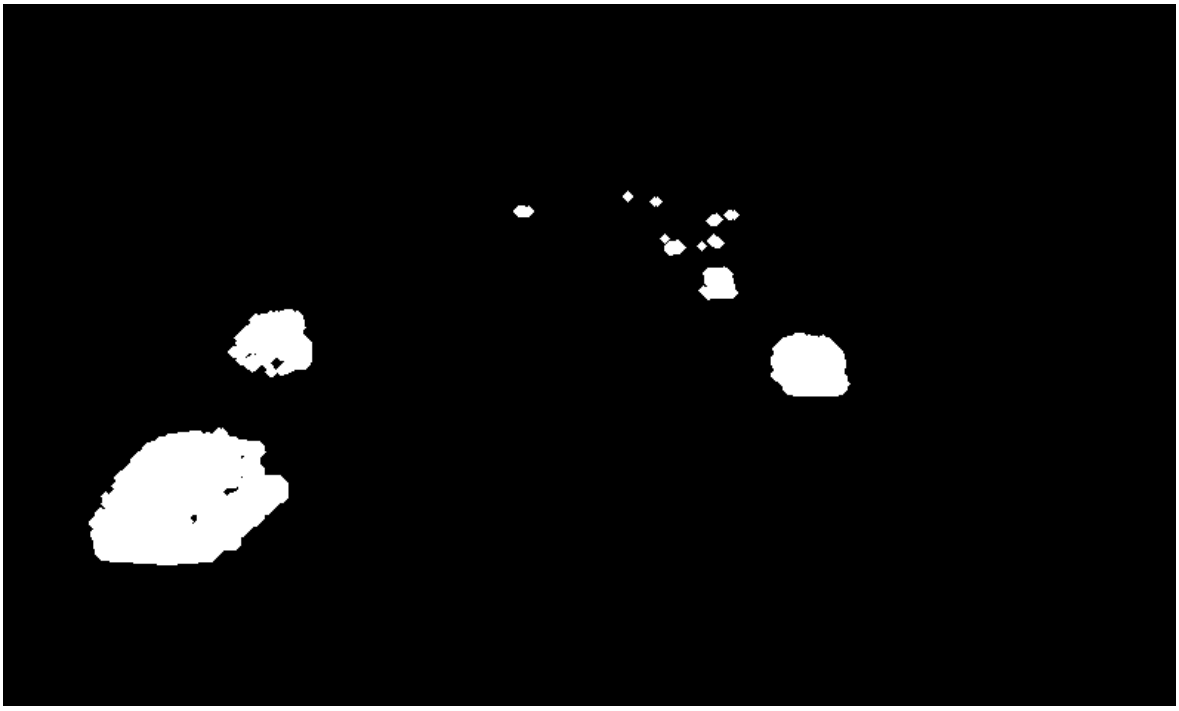
شکل ۳ – خروجی MOG۲

همانطور که در شکل ۳ مشاهده میشود پس از حذف پس زمینه از تصویر، هنوز مقداری نویز در تصویر وجود دارد. برای حذف نویز های باقی مانده در تصویر از فیلتر های erosionو dilation استفاده شده است. عمل Dilation تصویر باینری را بسط[۱] میدهد، و همینطور علاوه بر گسترش دادن تصویر ، حفره های موجود در آن را از بین می برد و لبه های خراب را ترمیم می کند. در مقابل erosion عکس عملیات Dilation می باشد به این صورت که تصویر باینری را تحلیل میدهد ، و در واقع تصویر را کوچک میکند و قسمت های اضافی را حذف می کند[۲]

با اعمال این دو فیلتر تقریبا تمام نویز ها حذف میشود و برای حذف نویز باقی مانده از یک ترشلد استفاده میشود.

[۱] Expand

[۲] Shrink



شکل ٤ – تصویر نهایی پس از اعمال انواع فیلتر ها

۲-۳ توابع و کلاس های استفاده شده

۲-۳-۱ کلاس tube

از این تابع برای ذخیره باکس های ماشین شناسایی شده در هر فریم استفاده می شود. با توجه به این که نیاز است برای هر باکس در هر فریم یک سری ویژگی ها ذخیره شود، برای راحتی کار از کلاس استفاده شده است.

جدول ۱ – ویژگی های کلاس tube

ویژگی های کلاس tube	
Cx	**مختصات x مرکز شی**
Cy	**مختصات y مرکز شی**
X	**مختصات x گوشه بالا سمت چپ باکس**
Y	**مختصات y گوشه بالا سمت چپ باکس**

W	**انداز عرض باکس**
h	**اندازه ارتفاع باکس**
T_sec	**زمان حضور شی در ویدیو بر حسب ثانیه**
target	**تصویر باکس به صورت** RGB

۲-۳-۲ کلاس MyCar

از این کلاس برای ذخیره مشخصات هر یک از ماشین های شناسایی شده، در طول ویدیو استفاده شده است. برای مثال در یک لیست تیوب های کلاس قبلی در آن ذخیره، و یا لیستی از مختصات مرکز باکس ها ذخیره شده است. ویژگی های مورد استفاده در این کلاس را در جدول ۲ مشاهده میشود.

جدول ۲ – ویژگی های کلاس MyCar

ویژگی های کلاس Car	
Cx	**مختصات x مرکز ماشین آخرین باکس**
Cy	**مختصات y مرکز ماشین آخرین باکس**
X	**مختصات x گوشه بالا سمت چپ اخرین باکس**
Y	**مختصات y گوشه بالا سمت چپ اخرین باکس**
W	**انداز عرض اخرین باکس**
h	**اندازه ارتفاع اخرین باکس**
tubes	**لیستی از باکس های شناسایی شده (کلاس** tube)
tracks	**لیستی از مختصات مرکز باکس های شناسایی شده**
R	**رنگ خودرو**
G	**رنگ خودرو**
B	**رنگ خودرو**
done	**شناسایی جهت حرکت ماشین انجام شده است یا نه**

State	**جهت حرکت ماشین مشخص می شود.**
dir	**جهت حرکت ماشین مشخص می شود**
empty	**آیا لیست tube ها خالی است یا پر**
endFrame	**فریم پایانی در ویدیو اصلی**
startFrame	**فریم شروع در ویدیو اصلی**
start	**آی دی شروع لیست tubes**
speed	**سرعت ماشین**

جدول ۳ – توابع کلاس MyCar

توابع کلاس MyCar	
getRGB	**رنگ ماشین را بر می گرداند**
begin	**اولین باکس شناسایی شده (قرار گرفته در لیست) را بر می گرداند**
lentube	**اندازه طول لیست باکس های شناسایی شده خودرو را بر می گرداند**
Pop_front	**اولین باکس شناسایی شده را از لیست خارج می شود**
getId	**آی دی مربوط به خودرو مورد نظر را بر میگرداند**
getX	**مختصات x گوشه بالا سمت چپ اخرین باکس شناسایی شده را بر می گرداند**
getY	**مختصات y گوشه بالاسمت چپ اخرین باکس شناسایی شده را بر می گرداند**
getCX	**مختصات x اخرین باکس شناسایی شده ماشین را بر می گرداند**
getCy	**مختصات y اخرین باکس شناسایی شده ماشین را بر می گرداند**
updateCoords	**باکس جدید شناسایی شده را به ماشین شناسایی شده مربوط به آن اضافه می کند**
setDone	**مقدار متغیر done را Trueقرار می دهد**
Going_UP	**مشحص می کند خودرو به سمت بالا حرکت می کند یا نه**
Going_DOWN	**مشخص می کند خودرو به سمت پایین حرکت می کند یا نه**

getSpeed	**اندازه سرعت خودرو رو بر می گرداند**

۲-٤ توابع ماژول Car

در ماژول Car یک سری توابع پیاده سازی شده است که در این قسمت به توضیح این توابع پرداخته می شود.

۲-٤-۱ تابع getBG

در این تابع با استفاده از روش میانه، تصویر پس زمینه ویدیو به دست می آید.

```
def getBG(path, randomFrameSize):

    # Open Video
    cap = cv2.VideoCapture(path)

    # Randomly select 25 frames
    frameIds = cap.get(cv2.CAP_PROP_FRAME_COUNT) * np.random.uniform(size=randomFrameSize)

    # Store selected frames in an array
    frames = []
    for fid in frameIds:
        cap.set(cv2.CAP_PROP_POS_FRAMES, fid)
        ret, frame = cap.read()
        if ret:
            frames.append(frame)

    # Calculate the median along the time axis
    img_b = np.median(frames, axis=0).astype(dtype=np.uint8)
    return img_b
```

شکل ۵ – تابع به دست آوردن تصویر پس زمینه

۲-٤-۲ تابع isOverLabRects

با استفاده از این تابع بررسی می شود آیا دو باکس با هم همپوشانی دارند یا نه. ورودی تابع دو متغیر از نوع tuple است که دارای چهار پارامتر x,y,w,h است.

```
def isOverLabRects(a, b):
    if (a[0] > b[0]):
        x = a
        a = b
        b = x
    l = a
    r = b
    if (l[2] < r[0] - l[0]):
        return False
    elif (l[1] < r[1] + 1 and l[3] >= r[2] - l[2]):
        return True
    elif (l[1] > r[1] and r[3] >= l[1] - r[1]):
        return True
    else:
        return False
```

شکل ٦ – ٢-٤-٢ تابع isOverLabRects

٢-٤-٣ تابع isOverLabCar

با استفاده از این تابع بررسی می شود که دو گروه شناسایی شده از خودرو ها، ایا با هم همپوشانی مکانی با توجه به زمانی که در ویدیو خلاصه شده قرار می گیرند دارند یا نه. ورودی تابع دو مقدار از نوع کلاس Car است که لیست باکس های شناسایی شده در طول ویدیو برای هر خودرو ذخیره شده است.

```
def isOverLabCar(car1, car2):
    if car2.startFrame > car1.lentube():
        return False

    for j in range(max(car2.startFrame, car1.startFrame), min(car1.endFrame, car2.endFrame)):
        a = (car2.tubes[j].x, car2.tubes[j].y, car2.tubes[j].w, car2.tubes[j].h)
        b = (car1.tubes[j].x, car1.tubes[j].y, car1.tubes[j].w, car1.tubes[j].h)
        if isOverLabRects(a, b):
            return True
    return False
```

شکل ٧ – تابع isOverLabCar

۲-٤-٤ تابع gradientline

با استفاده از این تابع یک خط با روشنایی متفاوت ، بر روی نقاط داده شده به عنوان ورودی و یک تصویر پس زمینه آن رسم می شود.

```
def gradientline(frame,pts):
    ans = pts.shape[0]
    for i in range(ans-1):
        cv2.polylines(frame, [pts[i:i+2]], False, (int(255/ans*i),int(255/ans*i),int(255/ans*i)),4)
    return np.array(frame)
```

شکل ۸ – تابع gradientline

۲-٤-٥ تابع setColor

با استفاده از این تابع رنگ خودرو شناسایی شده در هر فریم به دست می آید. جزییات روش کار در فصل بعد به صورت مفصل توضیح داده شده است.

۲-٤-٦ تابع keyPointSilmilarity

ورودی این تابع دو تصویر و خروجی ان میزان شباهت دو تابع با استفاده از نقاط کلیدی مشترک به دست آمده بر روی هر یک از تصاویر است.

فصل۳ روش انجام پروژه

در این فصل روش انجام پروژه با جزییات بیشتر توضیح داده می شود.

۳-۱ به دست آوردن تصویر پس زمینه

همانطور که در فصل ۲ اشاره شده ، برای به دست آوردن تصویر پس زمینه از روش میانه استفاده شده است. تصویر پس زمینه در انتهای پروژه برای قرار دادن اشیا تخیص داده شده بر روی آن استفاده شده است. برای این کار از تابع getBG در ماژول Car استفاده شده است.

۳-۲ به دست آوردن تصویر رو زمینه

برای تشخیص تصاویر متحرک از روش حذف تصویر پس زمینه استفاده شده است. همانطور که در فصل ۲ اشاره شد با استفاده از انواع فیلتر ها نویز تصویر پس از حذف پس زمینه گرفته شد. تصویر نهایی یک تصویر باینری از اشیا متحرک هب رنگ سفید است. در شکل نمونه ای از تصویر به دست آمده مشاهده میشود.

۳-۳ تشخیص اجسام متحرک

۳-۳-۱ شناسایی خودرو با استفاده از haar cascade

در مرحله اول برای شناسایی خودرو از روش haar cascade استفاده شد. این روش نسبت به روش های deep learning سرعت بسیار بالاتری دارد و امکان استفاده برخط از آن برای شناسایی وجود دارد. اما با پیاده سازی آن بر روی کد مشاهده شد تعداد False positive های آن زیاد بود و در

نتیجه در قسمت دسته بندی باکس های شناسایی برای خلاصه سازی ویدیو به مشکل برخورده شد. همچنین با این که سرعت این روش نسبت به روش های یادگیری عمیق بیشتر است اما باز هم به دلیل نیاز به ذخیره سازی باکس های شناسایی شده برای استفاده در قسمت های بعدی سرعت پایینی داشت. همچنین با توجه به این که خودرو ها به صورت شناسایی باکس دور آن ها شناسایی میشد زمانی که دو خودرو در نزدیکی هم حرکت کنند با هم تداخل دارند و در قسمت های بعدی برای خلاصه سازی ویدیو دچار خطای زیادی میشد.

۲-۳-۳ شناسایی خودرو با استفاده از روش پیدا کردن کانتور ها

با توجه به مشکلاتی که در قسمت قبلی به آن اشاره شد، از روش دیگری برای شناسایی خودرو استفاده شد. در این روش با حذف پس زمینه تصویر (در فصل ۲ به طور کامل توضیح داده شد) یک تصویر باینری از اشیا متحرک به دست می آید. در ادامه با استفاده از تابع آماده cv۲.findcountor مرز های پیوسته (کانتور) به دست آمده است :

cv۲.findContours(img,cv۲.RETR_TREE,cv۲.CHAIN_APPROX_SIMPLE)

آرگومان اول تصویر ورودی است، آرگومان دوم حالت کانتور است، آرگومان سوم روش یافتن کانتور است.

```
M = cv2.moments(cnt)
cx = int(M['m10'] / M['m00'])
cy = int(M['m01'] / M['m00'])
x, y, w, h = cv2.boundingRect(cnt)
```

شکل ۹ – به دست آوردن مرکز و مختصات باکس کانتور ها

درادامه با داشتن کانتور های هر یک از اشیا متحرک در تصویر با استفاده از دستور boundingRect۲cv. مختصات گوشه بالا سمت چپ و طول و عرض باکس شی مورد نظر به دست می آید. همچنین با استفاده از دستور moments۲cv. مختصات مرکز شی مورد نظر به دست می آید.

از جمله مزیت های این روش این است که با توجه به این که شناسایی خودرو با استفاده از مرز پیوسته دور اشیا به دست می آید، درصد خطا هنگامی که دو ماشین در نزدیکی هم در حال حرکت هستند را کم می کند. همچنین امکان به دست آوردن مرکز شی متحرک به صورت دقیق تر فراهم میشود.

۳-٤ گروه بندی مستطیل های شناسایی شده

در این مرحله مستطیل های شناسایی شده ردیابی[۱] میشوند. برای این منظور هر یک از ماشین هایی که ردیابی میشوند در یک کلاس Car مشخصات آن ها ذخیره می شود و هر کدام از ماشین هایی که ردیابی می شوند در یک لیست از کلاس های Car در لیست Cars ذخیره می شوند.

۳-٤-۱ ردیابی خودرو

همانطور که گفته شد یکی از مراحل گروه بندی هر یک از خودرو های شناسایی شده در خلاصه سازی ویدیو، ردیابی خودرو ها است. برای این منظور روش های مختلفی وجود دارد که در این قسمت هر یک از روش های انجام گرفته بررسی می شود.

۳-٤-۱-۱ استفاده از نقاط کلیدی

در این روش در هر فریم برای هر باکس جدید شناسایی شده ، نقاط کلیدی باکس جدید را به دست می آوریم و با اخرین باکس های ماشین های شناسایی شده تا این لحظه مقایسه می شود.(با نقاط کلیدی اخرین باکس های ماشین های شناسایی شده) در صورت داشتن نقاط کلیدی مشترک، باکس جدید را به آن ماشین اضافه می کند. برای پیاده سازی این روش از تابع keyPointSimilarity در ماژول Car استفاده شده است. در این تابع تعدد نقاط کلیدی مشترک بر تعداد نقاط کلیدی تصویر با نقاط کلیدی کمتر تقسیم میشود و خروجی رو بر می گرداند. با قرار دادن یک عدد مناسب به عنوان ترشلد شناسایی تشابه، نتیجه مورد نظر به دست می آید.

با پیاده سازی این روش مشاهده شد که نتایج به دست آمده از نظر دقت پایین هستند. زیرا هر مقدار که اندازه باکس شناسایی شده کمتر می شود دقت محاسبات نقاط کلیدی کمتر می شود. در نتیجه استفاده از این روش تنها برای باکس هایی که از نظر اندازه بزرگ هستند به نتایج مناسبی رسیده است. همچنین درمواردی که رنگ ماشین ها یکی هستند به دلیل شباهت ماشین های معمولی دچار خطا می شود. از طرفی از نظر زمانی، استفاده از نقاط کلیدی زمان اجرای برنامه را بیشتر کرده است.

[۱] Tacking

۳-۱-٤-۲ استفاده از نقاط کلیدی و همپوشانی

یکی از روش های دیگر پیاده سازی شده و امتحان شده استفاده از روش قبلی و همپوشانی باکس های جدید با اخرین باکس های خودرو های شناسایی شده است. در این روش تنها تشابه نقاط کلیدی اخرین باکس هایی که با باکس جدید همپوشانی مکانی دارند بررسی میشود. این کار باعث می شود اجرای برنامه بهبود یابد، اما همچنان خطاهایی که در روش نقاط کلیدی وجود دارد باقی مانده است.

۳-۱-٤-۳ استفاده از همپوشانی

با توجه به خطای زیاد روش نقاط کلیدی در باکس های کوچک برای شناسایی نقاط کلیدی، در این روش بدون استفاده از تشابه نقاط کلیدی، تنها با استفاده از همپوشانی باکس های جدید با اخرین باکس های شناسایی شده برای هر خودرو، ردیابی و گروه بندی انجام میشود. در این روش در هر فریم هر یک از مستطیل های شناسایی شده را با اخرین باکس هر یک از Carهای شناسایی شده در فریم های قبلی مقایسه می کنیم در صورتی که با اخرین باکس شناسایی شده از هر کدام از ماشین ها اشتراکی داشته باشند، مستطیل جدید را به آن Car اضافه می کنیم.

با بررسی نتایج مشاهده شد از نظر زمان اجرای برنامه بهبود قابل توجهی پیدا کرده است. همچنین با توجه به این که مسیر حرکت خودرو ها در یک خط راست است مشاهده شد تنها با استفاده از همپوشانی به نتایج قابل قبولی می توان رسید.

با بررسی خطاهای صورت گرفته در این روش مشاهده شد ممکن است زمان بررسی باکس جدید با اخرین باکس های ماشین های قبلی، با یک باکس که از نظر زمانی فاصله زیادی وجود دارد همپوشانی مکانی به وجود بیاید و به اشتباه در گروه آن ماشین قرار بگیرد. برای حل این مشکل از روش بعدی استفاده شده است.

۳-۱-٤-٤ استفاده همپوشانی و قید زمانی

در این روش علاوه بر بررسی همپوشانی، یک قید زمانی نیز استفاده شده است. به این صورت که برای هر باکس شناسایی شده یک متغیر به نام frameId در کلاس MyCar استفاده شده است. در این متغیر شماره فریم اخرین باکس شناسایی شده از یک ماشین ذخیره می شود. در هنگام بررسی همپوشانی باکس جدید با اخرین باکس های ماشین های شناسایی شده علاوه بر بررسی همپوشانی، در صورتی که اختلاف frameId باکس جدید با باکس مورد بررسی کمتر از یک مقدار مشخص باشد همپوشانی

بررسی می شود. این مقدار ۵ در نظر گرفته شده است.

برای این کار از الگوریتم زیر استفاده میکنیم :

در هر فریم هر یک از مستطیل های شناسایی شده را با اخرین باکس هر یک از Carهای شناسایی شده در فریم های قبلی مقایسه می کنیم در صورتی که با اخرین باکس شناسایی شده از هر کدام از ماشین ها اشتراکی داشته باشند، مستطیل جدید را به آن Car اضافه می کنیم.(همچنین باید اختلف فریمی که باکس جدید با اخرین باکس های شناسایی شده داشته باشد کمتر از ۵ باشد زیرا ممکن است اخرین باکس شناسایی شده برای یک باکس که قبلا از این مسیر عبور کرده است به اشتباه به عنوان گروه باکس جدید شناسایی شود.) در صورتی که هیچ اشتراکی با اخرین باکس های ماشین های شناسایی شده نداشته باشند به عنوان ماشین جدید شناسایی و یک شی Carجدید ایجاد کرده و به لیست Carsاضافه می شود.

۳-۵ خلاصه سازی ویدیو

در این مرحله یک لیست از Car های شناسایی شده وجود دارد که در هر یک از Car ها ، باکس های مربوط به آن ماشین به ترتیب زمانی ذخیره شده اند (به همراه یک سری ویژگی های دیگر که در لیست ویژگی های مربوط به Car ,tube در فصل ۲ مشخص شده است) در این مرحله نحوه قراردادن ماشین ها در کنار هم به طوری که همپوشانی مکانی و زمانی به حداقل مقدار برسد توضیح داده می شود.

۳-۵-۱ خلاصه سازی ویدیو بدون کمینه کردن همپوشانی بین tube ها

در این روش طول ویدیو برابر ماکزیمم طول tube های موجود در car ها است. در واقع روش این کار به این صورت است که از فریم صفر شروع کرده و فریم به طول ماکزیمم طول tubeهای موجود برای یک car شناسایی شده، باکس های شناسایی شده برای هر کدام از car ها را بر روی فریم پس زمینه می گذاریم. در این روش خودرو ها همپوشانی مکانی دارند. از نظر نتیجه به دست آمده با استفاده از این روش، نتیجه ها بسیار ضعیف هستند زیرا خودرو ها بر روی هم می افتند. به همین دلیل از یک روش با کمینه کردن همپوشانی بین تیوب ها در قسمت بعد استفاده شده است.

۳-۵-۲ خلاصه سازی ویدیو با کمینه کردن همپوشانی بین tube ها (امتیازی)

برای این منظور یک متغیر به نام startFrame در کلاس Car ایجاد شده است. در این متغیر زمان قرار گرفتن شی شناسایی شده در ویدیو اصلی ذخیره میشود. در ابتدا این عدد برای تمام اشیا شناسایی شده صفر در نظر گرفته شده است. در ادامه در یک حلقه به طول لیست Cars هر یک از هر یک از اشیا را همپوشانی آن با اشیا قبلی را بررسی می کنیم . در صورتی که که شی I ام با شی قبلی خود (I>J) همپوشانی داشته داشته باشد مقدار متغیر startFrame را به اندازه یک افزایش می دهیم. و دوباره همپوشانی را بررسی می کنیم و این کار را تا زمانی که هیچ همپوشانی بین این دو نباشد ادامه می دهیم. این فرایند را برای تمام اعضای لیست Cars ادامه می دهیم و در هر مرحله با ماشین های قبلی همپوشانی را بررسی می کنیم. با تمام شدن این فرایند مقدار همپوشانی مکانی و زمانی ماشین ها به حداقل مقدار خود می رسد.

۳-۶ تشخیص مسیر حرکت و ترسیم آن (امتیازی)

برای تشخیص و ترسیم مسیر حرکت خودرو ها هر یک از خودرو ها را ردیابی و دنبال کرده و مسیر حرکتی آن را رسم می کنیم. برای این کار باید هر یک از خودرو های شناسایی شده را ردیابی کرد. روش این کار در قسمت های قبلی به صور مفصل توضیح داده شده است. همانطور که در قسمت های قبلی اشاره شد با استفاده از دستور moments۲cv. مرکز کانتور شناسایی شده به دست می آید. در هر مرحله پس از شناسایی کانتور جدید و تشخیص گروه مربوط به آن، مختصات مرکز کانتور در لیستی به نام car.tracks ذخیره می شود. سپس با استفاده از تابع gradientline که در ماژول Car پیاده سازی شده است نقاط ذخیره شده خودرو تا آن لحظه و فریم پس زمینه به تابع داده میشود و مسیر حرکت آن ذخیره می شود.

```
def gradientline(frame, pts):
    ans = pts.shape[0]
    for i in range(ans - 1):
        cv2.polylines(frame, False, [pts[i:i + 2]], (int(255 / ans * i), int(255 / ans * i), int(255 / ans * i)), 4)
    return np.array(frame)
```

شکل ۱۰ – تابع gradientline

۳-۶-۱ ترسیم مسیر به صورت GradienLine

با توجه به این که امکان رسم خط به صورت gradeintline در opencv وجود ندارد، برای رسم آن از تابع gradientline استفاده شده است. ورودی تابع فریم مورد نظر و نقاط مربوط به یک خودرو که در لیست tracks ذخیره شده اند هستند. برای رسم خط از تابع cv۲.polylines استفاده شده است. به این صورت که با استفاده از یک حلقه در هر مرحله دو نقطه پشت سر هم را برای رسم به تابع cv۲.polylines داده می شود و رنگ خط نیز به صورت یک نواخت از ۰ تا ۲۵۵ تغییر می کند. به این صورت خط با تغییر روشنایی رسم می شود. نمونه ای از خطوط رسم شده در تصویر زیر مشاهده میشود.

شکل ۱۱ – نمونه ای از مسیر رسم شده

۳-۷ شمارش تعداد خودرو ها (امتیازی)

برای شمارش تعداد خودرو های عبوری در هر یک از جهت ها، از دو خط line_up و line_down استفاده شده است. روش کار به این صورت است که برای هر یک از خودرو های شناسایی شده، یک لیست به نام tracks استفاده میشود تا مسیر ردیابی شده ماشین در ان ذخیره شود. (یکی از متغیر های کلاس Car است). در هر فریم با شناسایی گروه مربوط به هر یک از ماشین های شناسایی شده جهت حرکت ماشین و تعداد آن نیز شمارش می شود. برای این منظور از دو تابع زیر استفاده شده است :

```
def going_UP(self,line_up,line_down):
    if len(self.tracks) >=2:
        if self.state == '0':
            if self.tracks[-1][1] < line_up <= self.tracks[0][1]:
                self.state = '1'
                self.dir = 'up'
                return True
        else:
            return False
    else:
        return False
```

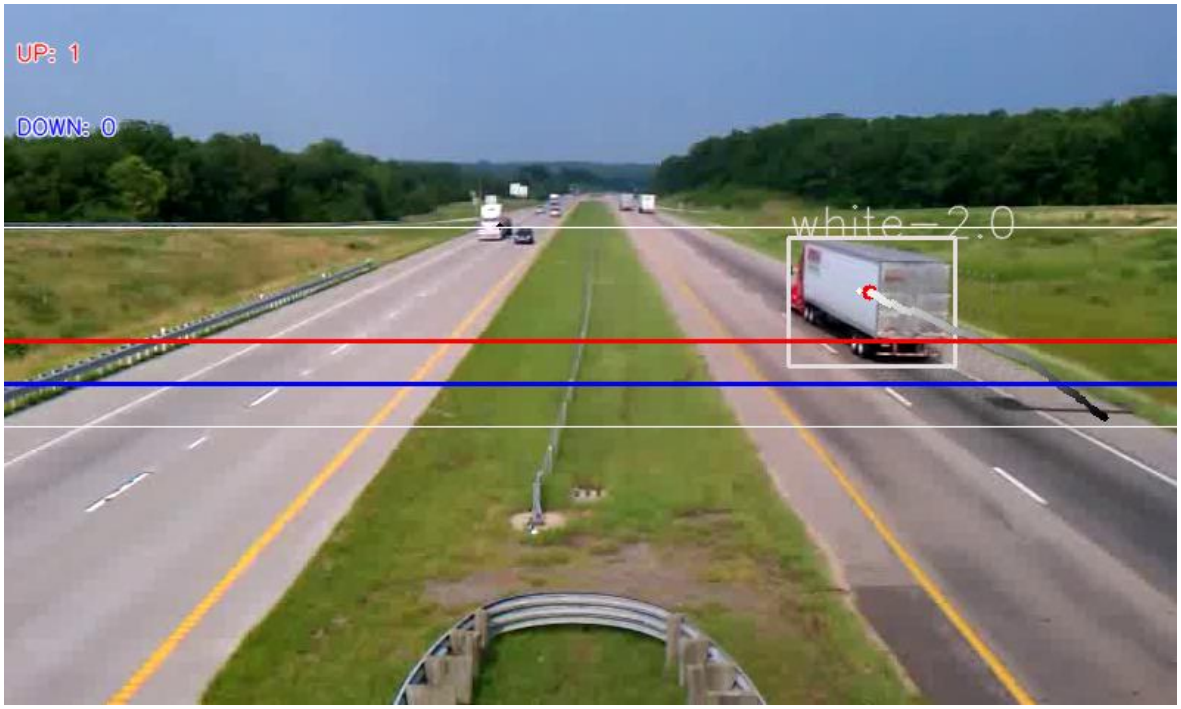
شکل ۱۲ – تابع شمارش خودرو going_up

برای شناسایی خودرو هایی که به سمت بالا حرکت می کنند، درهر فریم برای خودروهای شناسایی شده بررسی می شود در صورتی که اخرین مکان آن ها از خط line_up کمتر و اولین مکان آن ها از این خط بیشتر باشد در این صورت ماشین از از این خط عبور کرده و به تعداد ماشین ها یک عدد اضافه می شود.

```
def going_DOWN(self,line_up,line_down):
    if len(self.tracks) >= 3:
        if self.state == '0':
            if self.tracks[-1][1] > line_down >= self.tracks[0][1]:
                self.state = '1'
                self.dir = 'down'
                return True
        else:
            return False
    else:
        return False
```

شکل ۱۳ – تابع شمارش خودرو going_down

برای شناسایی خودرو هایی که به سمت بالا حرکت می کنندرهر فریم برای خودروهای شناسایی شده بررسی می شود در صورتی که اخرین مکان آن ها از خط line_down بیشتر و اولین مکان آن ها از این خط کمتر باشد در این صورت ماشین از از این خط عبور کرده و به تعداد ماشین ها یک عدد اضافه می شود.



شکل ١٤ – تصویری از شمارش خودرو ها در دو جهت در ویدیو١

همچنین تاریخ میلادی و ساعت عبور از خط های مشخص شده در هر جهت در خروجی چاپ می شود.

```
Area Threshold 512.0
Red line y: 259
Blue line y: 230
ID: 1 crossed going up at Sun Jul  5 10:46:30 2020
ID: 3 crossed going up at Sun Jul  5 10:46:37 2020
ID: 4 crossed going up at Sun Jul  5 10:46:38 2020
ID: 2 crossed going down at Sun Jul  5 10:46:40 2020
ID: 5 crossed going up at Sun Jul  5 10:46:41 2020
ID: 7 crossed going up at Sun Jul  5 10:46:45 2020
```

شکل ١٥ – نمونه ای از خروجی چاپ شده

٣-٨ تشخیص رنگ ماشین به روش K-MEANS (امتیازی)

K-means یک الگوریتم برای دسته بندی است. این الگوریتم، N داده را در K دسته مرتب می کند؛ بدین صورت که هر داده ای که به میانگین دسته ای نزدیک تر باشد، به آن دسته تعلق خواهد گرفت. برای تشخیص رنگ خودرو ها ابتدا تصاویر RGB موجود برای هر خودرو که شناسایی شده است و در ویژگی tube.target ذخیره شده است با استفاده از تابع setColor رنگ خودرو به وسیله الگوریتم K-

MEANS به دست می آید.

پارامتر های ورودی تابع **cv۲.kmeans()**

- samples : دیتا برای دسته بندی که نوع داده np.float۳۲ است و ویژگی های هر سمپل در یک ستون قرار می گیرد
- nclusters(K) : تعداد دسته ها برای دسته بندی داده ها
- criteria : شاخصی برای توقف الگوریتم. این شاخص از سه پارامتر تشکیل شده است
- Attempts : تعداد دفعاتی که الگوریتم با لیبل های اولیه متفاوت اجرا می شود.
- Flags : مشخص می کند که لیبل های اولیه به چه صورت انتخاب شوند.

۱-۸-۳ تشخیص رنگ خودرو

برای تشخیص رنگ هر باکس شناسایی شده تصاویر را با استفاده از الگوریتم K-means با پارامتر های مشخص شده در تصویر زیر دسته بندی می کنیم. با توجه به نوع مسیله ای که داریم تعداد دسته ها را ۲ در نظر می گیریم. زیرا تصویر به طور کلی از دورنگ پس زمینه و رنگ ماشین تشکیل شده است. قبل از دسته بدی هر یکی از تصاویر داده های مربوط به هر کدام از رنگ های RGB را در یک ستون قرار داده و دسته بندی رو انجام میشود. پس از انجام الگوریتم دسته بندی در هر لایه RGB رنگی که بیشترین تکرار را داشته باشد به عنوان رنگ نهایی آن لایه انتخاب می شود. در انتها با داشتن رنگ های سه لایه RGB باکس دور خودرو را به رنگ به دست آمده رسم شده است.

برای پیاده سازی الگوریتم تشخیص رنگ خودرو از تابع setColor استفاده شده است که ورودی آن از نوع کلاس tube است.

```python
def setColor(tube):
    slice_bg = np.array(tube.target)
    arr = np.float32(slice_bg)
    # reshaping the image to a linear form with 3-channels
    pixels = arr.reshape((-1, 3))

    # number of clusters
    n_colors = 2

    # number of iterations
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, .1)

    # initialising centroid
    flags = cv2.KMEANS_RANDOM_CENTERS

    # applying k-means to detect prominant color in the image
    _, labels, centroids = cv2.kmeans(pixels, n_colors, None, criteria, 10, flags)

    palette = np.uint8(centroids)
    quantized = palette[labels.flatten()]

    # detecting the centroid with densest cluster
    dominant_color = palette[np.argmax(itemfreq(labels)[:, -1])]
    tube.r = int(dominant_color[0])
    tube.g = int(dominant_color[1])
    tube.b = int(dominant_color[2])
```

شکل ۱۶ – تابع setColor برای تشخیص رنگ خودرو

شکل ۱۷ – نمونه ای از نتایج به دست آمده در تشخیص رنگ خودرو

فصل ٤ جمع بندی

در این پروژه سعی شد با استفاده از روش حذف تصویر پس زمینه و به دست آوردن یک تصویر باینری از اجسام متحرک، خودرو ها شناسایی و مسیر حرکت آن ها ردیابی شود و با ذخیره مسیر هر کدام از خودرو ها (گروه بندی خودرو ها) خلاصه سازی ویدیو انجام شود. برای ردیابی خودرو ها ٤ روش مختلف امتحان شد و در اخر از روش همپوشانی و در نظر گرفتن قید زمانی به دلیل نتایج بهتر و زمان اجرای کمتر استفاده شد.

در قسمت نهایی برای کنار هم قرار دادن خودرو ها برای کمینه کردن همپوشانی مکانی و زمانی از یک الگوریتم برای عدم همپوشانی خودرو ها در کنار هم استفاده شد. همانطور که در نتایج مشاهده میشود پارامتر های مختلفی برای fine tune کردن برای هر یک از ویدیو ها وجود دارد و می توان با تغییر این پارامتر ها نتایج متفاوتی به دست آورد.