3.1 车道界定与点位置判定

- 限定postprocess()中取得高可信度的boxes时，只取得car,bus,truck三类

```
selectedClassId = [2,5,7]
if confidence > confThreshold and classId in selectedClassId:
```

- 在 `parser` 中添加 `time_` 变量，用于指定视频处理所用的车道坐标信息。 `getCo()` 获取对应视频的车道坐标信息。

```
parser.add_argument('--time_', help='Time of the video.', default='day')
```

```
def getCo(videoTime):
    if videoTime == 'day':
        lane1 = [[(752.3653624856158, 567.9958079894789),
                  (660.6343087292455, 656.7677954956436),
                  ......]]
    elif videoTime == 'night':
        lane1 = [[(808.5876212395201, 502.896350484958),
                 (699.1021699819169, 606.4636692421502),
                 ......]]
    return [lane1, lane2, lane3, lane4, lane5]
```

- 在 `postprocess()` 中消除重叠的box后，对于最终的的bounding box，设定 `judge_x` 和 `judge_y` 为用于检测的目标坐标。需要说明的是，这里判断了目标大致在画面的左半边还是右半边，对应设定 `judge_x` 位置在车头或车尾的大约车牌位置，这样可以避免透视效果导致的判断失误。

```
    if x < framewidth/2:
        judge_x = (left + width/4)
    else:
        judge_x = (left + width*3/4)
    judge_y = top + height
```

- 新建 `point_in_poly.py` ,其中 `is_point_in_polygon()` 方法判定点是否在车道内，返回 `True` 或 `False`
- 在以上基础上定义 `inLane()` 函数。函数接收对象坐标和当前实时车辆数组 `inLaneArray` ，并调用 `is_point_in_polygon()` ，返回对象所在车道，以及更新后的各车道车数 `inLaneArray` 数组。在 `postprocess()` 和 `tracker()` 中调用 `inLane()` 以实现1、2、3目标的功能。

```
def inLane(judge_x, judge_y, laneArray, videoTime):
    inLaneArray=[]
    objInLane=999
    lane = getCo(videoTime)
    for i in range(5):
        res=poinpo.is_point_in_polygon((judge_x, judge_y), lane[i], False)
        inLaneArray.append(res)
    for i in inLaneArray:
        if i == 1:
            idx = inLaneArray.index(i)
            laneArray[idx] += 1
            objInLane = idx+1
    return laneArray,objInLane
```

- 将 `inLaneArray` 通过 `putText()` 方法实时输出显示到每帧，实现功能4。

3.2 短时、限定区域的简单目标跟踪与计数

- 尝试过对画面中所有目标进行跟踪来计数，效果不佳，很容易因为某帧识别不准确导致跟踪失败。cv库调用一些目标跟踪算法也不成功，可能是环境版本不匹配之类的问题。最后把跟踪区域限制到计数线附近一小块区域，只要在经过计数线前后识别为同一目标就不会技术失误。
- 先在 `postprocess()` 中，将所有y坐标在725到800之间的目标构成二维列表 `newObj` ，包含这些目标的class和坐标，并统一给予编号-1。

```
if judge_y>=725 and judge_y<=800:
    newObj.append([classIds[i], -1, judge_x, judge_y])
```

- 下一步调用 `tracker()` ，输入 `newObj` 和 `preObj` 进行比较,前者储存着当前帧的目标信息，后者储存上一帧的目标信息。

```
objInfo=tracker(preObj, newObj, laneArray, vieoTime)
```

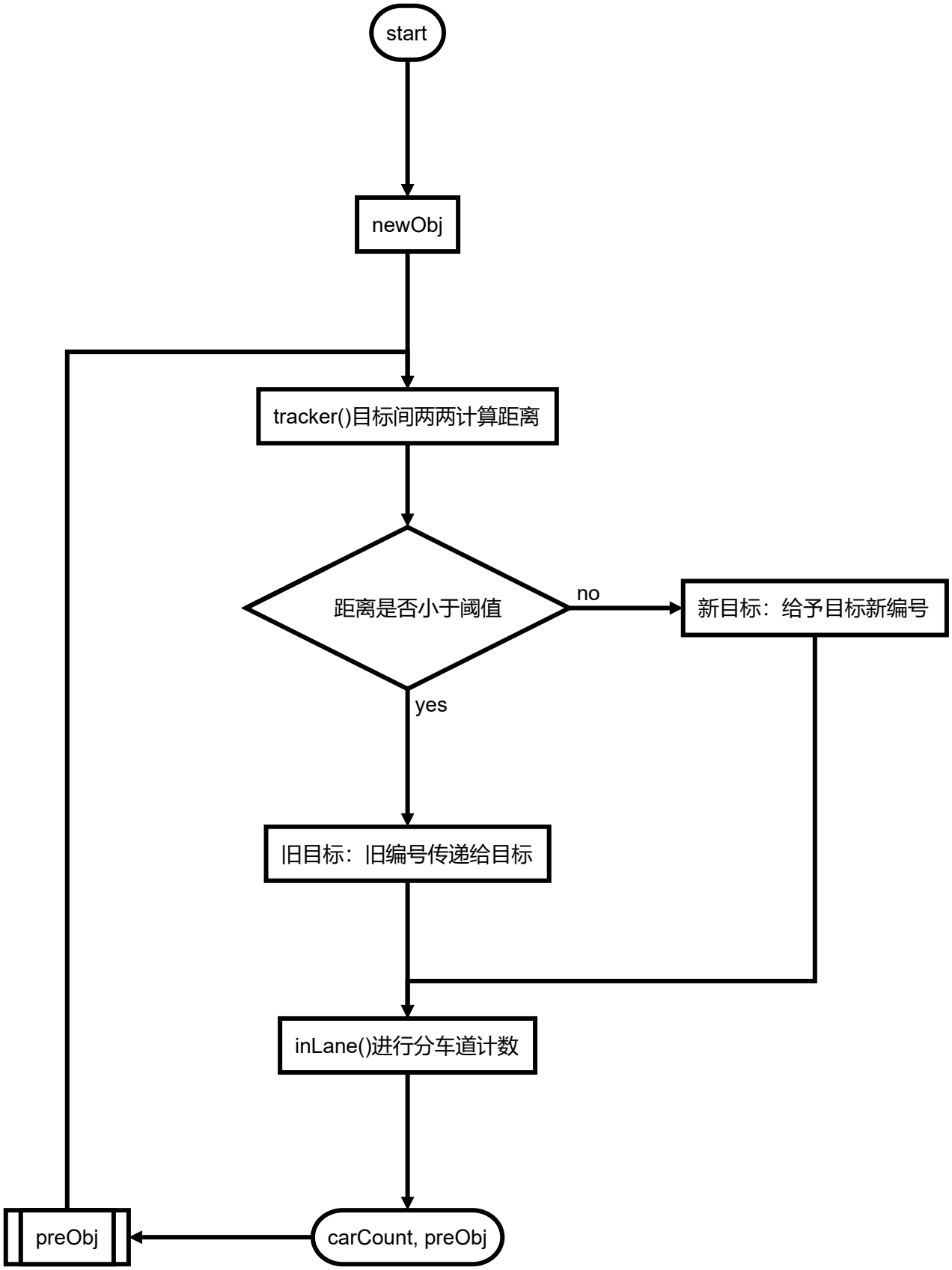
- 在 `tracker()` 中将前后帧目标进行两两比较，如果该帧的目标 `n` 和上一帧的目标 `p` 之间，坐标变化向量的F范数小于设定值（这里考虑到透视问题，设置判断阈值为y坐标的函数 `n[3]*65/1080` 则判断该目标为旧目标，把 `p` 的编号传递给 `n` ； 否则将 `n` 视为新目标，用计数器 `cnt` 给予新编号。

```
for n in newObj:
    idxN = newObj.index(n)
    for p in preObj:
        idxP = preObj.index(p)
        dis=torch.norm(torch.tensor([n[2]-p[2],n[3]-p[3]]))
        if dis<(n[3]*65/1080):
            n[1]=p[1]
            isNew = False
            break
        isNew = True
    if isNew == True:
        cnt+=1
        n[1]=cnt
```

- 设置计数线在y坐标775处，定义 `counted` 数组记录已记录的目标的编号避免重复计算，定义 `carCount` 数组记录各车道总计车辆数。 `newObj` 中目标信息更新完毕后，计算目标新旧y坐标是否在775两侧，并调用 `inLane()` 进行车道判定和累计计数，最后更新 `carCount` 和 `counted` ，并移除 `newObj` 和 `preObj` 中已统计的目标 实现目标5、6。

```
if n[1]==p[1] and ((p[3]-775)*(n[3]-775)) <= 0 and (n[1] not in counted):
    laneArray, objInLane = inLane(n[2], n[3], laneArray, videoTime)
    carCount[objInLane-1]+=1
    counted.append(n[1])
    newObj.remove(n)
    preObj.remove(p)
```

- 将 `carCount` 通过 `putText()` 方法实时输出显示到每帧，实现目标功能7。



3.3 输出csv文件

- 简单地将信息整理到列表中，视频处理完成后输出即可，实现目标8、9。

- 实时检测到的所有车辆信息

```
f1Cont.append([str(frameNum),idDic[classIds[i]],str(confidences[i]),'('+str(left)+','+str(to
```

- 被 `tracker()` 跟踪到的目标信息

```
f2Cont.append([str(frameNum),str(objNum),classId,x,y])
```

3.4 提高处理速度

- 车辆运动形式单一，可以通过降级目标识别的采样率来提高电脑的处理速度。
- 添加帧数计数器，可选择隔几帧处理一次，而不是每帧都处理。

```
while cv.waitKey(1) < 0:
    frameNum += 1
    if frameNum%sampling == 0:
```

- 在 `parser` 添加采样参数 `sampling` ， 指定每多少帧处理一次。修改后注意要将 `tracker()` 的跟踪阈值适当调大。

```
parser.add_argument('--sampling', help='Frequency of sampling.', default=1)
```