MEC设备的交通指标需求

|  |  |  |  |
| --- | --- | --- | --- |
| 时间 | 版本 | 人员 | 备注 |
| 2020-10-26 | V0.1 |  | 指标的定义和计算方法 |

目录

[1. 参考标准 4](#_Toc11914)

[2. 项目指标汇总 4](#_Toc19225)

[3. 项目数据格式 5](#_Toc25172)

[4. 计算流程 6](#_Toc7452)

[5. 车辆位置检测 6](#_Toc21964)

[5.1. 区域边界检测 6](#_Toc29009)

[5.2. 车道边界检测 9](#_Toc3055)

[5.3. 边界跨线检测 10](#_Toc31428)

[5.4. 输出格式 12](#_Toc8110)

[6. 交通指标计算 12](#_Toc5215)

[6.1. 流量 12](#_Toc9448)

[6.1.1. 定义 12](#_Toc7340)

[6.1.2. 计算方法 13](#_Toc6585)

[6.1.3. 输出格式 15](#_Toc5177)

[6.2. 速度 15](#_Toc13233)

[6.2.1. 定义 15](#_Toc15508)

[6.2.2. 计算方法 16](#_Toc28540)

[6.2.3. 输出格式 17](#_Toc9149)

[6.3. 车头时距 17](#_Toc16121)

[6.3.1. 定义 17](#_Toc10360)

[6.3.2. 计算方法 18](#_Toc20692)

[6.3.3. 输出格式 19](#_Toc9706)

[6.4. 车头间距 19](#_Toc1561)

[6.4.1. 定义 19](#_Toc10597)

[6.4.2. 计算方法 20](#_Toc3434)

[6.4.3. 输出格式 21](#_Toc27472)

[6.5. 密度 22](#_Toc6319)

[6.5.1. 定义 22](#_Toc19596)

[6.5.2. 计算方法 22](#_Toc2428)

[6.5.3. 输出格式 23](#_Toc25541)

[6.6. 通行能力 23](#_Toc4488)

[6.6.1. 定义 23](#_Toc17045)

[6.6.2. 计算方法 24](#_Toc28641)

[6.6.3. 输出格式 25](#_Toc6234)

[6.7. 停车次数 26](#_Toc30916)

[6.7.1. 定义 26](#_Toc18741)

[6.7.2. 计算方法 26](#_Toc17735)

[6.7.3. 输出格式 28](#_Toc398)

[6.8. 排队长度 29](#_Toc4404)

[6.8.1. 定义 29](#_Toc21251)

[6.8.2. 计算方法 29](#_Toc16576)

[6.8.3. 输出格式 34](#_Toc3521)

[7. 交通事件检测 35](#_Toc5383)

# 参考标准

**国家标准：**

GB/T 33171-2016 《城市交通运行状况评价规范》

GB/T 29192-2012 《城市交通流信息采集与存储》

GB/T 29017-2012 《道路交通信息服务 交通状况描述》

GB/T 29108-2012 《道路交通信息服务 术语》

**慧尔视的专利：**

201910480591.X 一种车辆动态排队长度检测方法

201910561411.0 交叉口车辆延误时间与停车次数的计算方法和装置

201710757530.4 基于微波的车辆排队长度检测方法及系统

**手册：**

Highway capacity manual 2000

VISSIM 5.20用户手册

全永燊 《城市交通控制》

# 项目指标汇总

表1 交通指标分类和汇总

|  |  |  |  |
| --- | --- | --- | --- |
|  | 交通指标 | 派生指标 | 毫米波雷达 |
| 基本指标 | 流量 | 车道流量、断面流量 | √ |
| 速度 | 平均区间速度 | √ |
| 车头时距 | 平均、饱和 | √ |
| 车头间距 | 平均 | √ |
| 密度 |  | √ |
| 特征指标 | **通行能力** |  | √ |
| **停车次数** |  | √ |
| **排队长度** | 最大排队长度、排队车辆数 | √ |

# 项目数据格式

提供的视频和毫米波雷达目标检测融合对应的字段结果，如下：

表2 数据说明

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 简写 | 数据类型 | 备注 |
| 时间秒 | sec | int |  |
| 时间毫秒 | msec | int |  |
| 目标ID | id | int |  |
| 车道ID | lane\_id | int |  |
| 车牌号码 | plate\_num | string | 可选；视频融合的结果，追踪不准 |
| 目标类别 | label | string | 可选；识别不准；分为汽车、非机动车、行人等 |
| 车辆类型 | cartype | string | 字段为空 |
| 长度 | dimension\_x | double | 可选，识别不准 |
| 宽度 | dimension\_y | double | 可选，识别不准 |
| 雷达x轴坐标 | radar\_px | double |  |
| 雷达y轴坐标 | radar\_py | double |  |
| 雷达z轴坐标 | radar\_pz | double |  |
| 雷达x轴速度 | radar\_vx | double |  |
| 雷达y轴速度 | radar\_vy | double |  |
| 雷达z轴速度 | radar\_vz | double |  |
| 目标经度 | longtitude | double |  |
| 目标纬度 | lattitude | double |  |
| 目标高度 | altitude | double |  |
| 雷达编号 | source | int |  |
| 目标左上角像素坐标x | pixel\_x | int | 可选；视频融合的结果 |
| 目标左上角像素坐标y | pixel\_y | int | 可选；视频融合的结果 |
| 目标像素宽度 | width | int | 可选；视频融合的结果 |

# 计算流程



# 车辆位置检测

车辆位置检测实现对车辆坐标的预处理，为车辆增加区域字段、车道字段和边界跨线字段等属性。

## 区域边界检测

假设条件：

1）毫米波雷达的检测方向正对着目标车辆行驶方向，检测车辆的车头正面；

2）已知所有的车道通过若干线段（即由道路边缘线和远近两个道路断面）合围而成。

已知道路中央分隔线的形状点集合表示为；道路外侧边缘线的形状点集合表示为；远离检测器的道路断面可选择配置，相应的形状点集合表示为；靠近检测器的道路断面，相应的形状点表示为。

为了便于计算，线段通过形状点表示为ax+by-c=0，

其中，a=，b=，c=。

通过射线法判断，目标车辆是否位于某一区域或者属于车道的位置。

1）沿毫米波雷达的Y轴方向引出一条射线，当与道路边缘线和有1个交叉点时，说明车辆在道路边缘线内部；

2）沿毫米波雷达的X轴方向引出一条射线，当与远近两个道路断面和有1个交叉点时，说明车辆在道路检测断面内。

综合上述两个条件，当同时满足时，车辆位于检测区域内部，返回TRUE。

**算法 zone\_border\_detection()**

**//用于判断目标车辆是否位于检测区域**

//输入：每一时刻车辆集合V，每辆车的坐标点，道路中央隔离线的集合，近处（停止线）道路断面线的集合

//输出：布尔结果z、区域内车辆信息

###代码块

veh  vehicle //初始化车辆结构体

for v0,1,…, do

ztrue

veh Vv

  veh.x\_pos, x\_index 0 //判断车辆v的所在的区间

search\_point(,0,n,,x\_index) //查找, logn计算量

slope  (Xcenter[x\_index+1][1]-Xcenter[x\_index][1])/

(Xcenter[x\_index+1][0]-Xcenter[x\_index][0])

if ( > slope \* (xv-Xcenter[x\_index][0]) + Xcenter[x\_index][1]) then

z1  true;

end if

if (z1 == false) then

z  faslse; break

end if

  veh.y\_pos, y\_index 0 ///判断车辆v的所在的区间

search\_point(,0,s,,y\_index) //查找, logs计算量

slope  (Xcenter[y\_index+1][0]-Xcenter[y\_index][0])/

(Xcenter[y\_index+1][1]-Xcenter[y\_index][1])

if ( > slope \* (yv-Xcenter[y\_index][1]) + Xcenter[y\_index][0]) then

z2  true;

end if

if (z2 == false) then

z  fasle

end if

return z //返回最终的布尔结果

end for

search\_point(float X[],int left,int right,float x,int index) { //二分查找算法，

i\_start  0, i\_end  n;

i\_middle  (i\_start + i\_end)/2;

if (x[index] <= x && x[index+1] < x ) then

return index;

end if

search\_point(i\_start,i\_middle,)；

search\_point(i\_middle+1,i\_end,)；

}

## 车道边界检测

假设条件：

1）毫米波雷达的检测方向正对着目标车辆行驶方向，检测车辆的车头正面；

2）已知所有的车道通过若干线段（即由道路边缘线和远近两个道路断面，其中远离检测器的道路断面可选）合围而成。

3）已知车道数为N，每个车道的分隔线由若干线段连接而成。

已知道路中央分隔线的形状点集合表示为；道路外侧边缘线的形状点集合表示为。

每个车道*l*分隔线的线段形状点表示为。

首先，通过射线法判断，目标车辆是否位于某一区域内。沿毫米波雷达的Y轴方向引出一条射线，当与道路边缘线和有1个交叉点时，说明车辆在道路边缘线内部；

然后，通过射线法判断车辆所在车道的问题。沿毫米波雷达的Y轴方向引出一条射线，统计与车道分隔线的交叉点个数，确定所在车道的位置。

**算法 lane\_num\_detection()**

**//用于判断目标车辆所在车道的编号**

//输入：每一时刻每一时刻车辆集合V，以及车辆的坐标点，道路中央隔离线的集合，道路外侧边缘线的集合，第l个车道分隔线的集合

//输出：车辆所在的车道编号、车辆信息

###代码块

for v0,1,…, do

cross\_num  0

for X {,Xlane\_1,Xlane\_2,…,Xleft} do

target  ,index 0, nX.size() //判断车辆v的所在的区间

search\_point(X,0,n,,index) //查找, logn计算量

slope  (X[index+1][1]-Xcenter[index][1])/

(Xcenter[index+1][0]-Xcenter[x\_index][0])

if ( > slope \* (xv-X[index][0]) + X[index][1]) then

cross\_num++;

end if

end for

return cross\_num

end for

## 边界跨线检测

假定条件：

1. 毫米波雷达的检测方向正对着目标车辆行驶方向，检测车辆的车头正面；
2. 已知道路断面通过连续的n个线段连接而成，线段的形状点集合表示为；第n个线段的区间和方向通过形状点、表示，一般是线段的起点和终点坐标。

为了便于计算，线段通过形状点表示为ax+by-c=0，

其中，a=，b=，c=。

1、首先通过欧式距离最近的原则，找到离目标车辆最近的线段；进而，目标车辆需要在相邻的两个时间戳内，是否通过线段的判断方法如下：



其中，和分别表示在t-1和t时刻的目标车辆位置。

2、目标车辆需要满足位置处于断面区间内，判断方法如下：



最终满足上述两个条件下，判断车辆正经过边界处，触发边界跨线检测。

**算法 cross\_line\_detection()**

**//用于判断车辆是否跨越检测线**

//输入：每一时刻车辆集合V，以及车辆上一时刻和当前的坐标点、，道路断面的形状点集合X

//输出：布尔结果z、跨线车辆的信息

###代码块

for v  0,1,…, do

z  false

target  yv, index 0 //判断车辆v的所在的区间

search\_point(X,0,n,yv,index) //查找, logn计算量

*yi* X[index+1][1], *yi-1* [index][1], *xi* X[index+1][0],

*xi-1*  X[index][0]

a，b，c

if 

z  true

xmid = (*xi+xi-1*)/2, ymid = (*yi+yi-1*)/2

end if

return z, xmid, ymid

end for

## 输出格式

以结构体的方式，增加四个车辆的位置信息、两个时间戳、一个坐标点，具体如下：

struct vehicle{

bool veh\_in\_zone; //车辆是否位于划线区域内部

int lane\_num; //车辆所在的车道编号

bool drive\_in\_zone; //车辆是否驶入划线区域

string time\_drive\_in; //车辆驶入划线区域的时间

bool drive\_out\_zone; //车辆是否驶出划线区域

string time\_drive\_out; //车辆驶出划线区域的时间

float x\_cross; //车辆驶出划线区域的X轴坐标

float y\_cross; //车辆驶出划线区域的Y轴坐标

}

# 交通指标计算

## 流量

### 定义

GB/T 29192-2012 《城市交通流信息采集与存储》

流量，是指观测时间内，通过道路指定断面或车道的车辆数。



式中：

——观测时间T内流量值，单位为标准小汽车量/观测间隔；

——采样间隔内标准小汽车量，单位为标准小汽车量/采样秒；

——车辆的类型总数；

——车辆类型对应的标准小汽车换算系数；

——采样间隔内某一车辆类型的车辆数，单位为（辆/采样秒）。

JTG B01-2014 《公路工程技术标准》

表2 车辆类型对应的标准小汽车换算系数

|  |  |  |
| --- | --- | --- |
| 车型 | 换算系数 | 说明 |
| 小客车 | 1.0 | 座位≤19座的客车和载重量≤2t的货车，**车长≤6m** |
| 中型车 | 1.5 | 座位＞19座的客车和2t＜载重量≤7t的货车，**6m＜车长≤9m**。 |
| 大型车 | 2.5 | 7t＜载重量≤20t的货车，**9m＜车长≤12m**。 |
| 汽车列车 | 4.0 | 载重量＞20t的货车，**车长＞12m**。 |

### 计算方法

#### 断面流量

断面流量统计的核心是车辆的边界跨线检测，大致步骤如下：

1. 获取当前时刻区域内的所有车辆集合；
2. 通过边界跨线检测，获取车辆是否驶离区域的字段值drive\_out\_zone；
3. 当drive\_out\_zone=True时，累加流量值，并记录车辆的类型。

**算法 section\_volume\_calculation()**

**//用于统计某一时间间隔的断面流量值**

//输入：车辆集合V、每一时刻车辆类型、是否驶离区域

//输出：车辆类型和流量值Volume

###伪代码块

volume[type]  0 //初始化不同车辆类型的流量

veh  vehicle //初始化车辆结构体

for v0,1,…, do

veh  Vv

if veh.drive\_out\_zone == true then

if (veh.type = type) then

volume[type]++

end if

end if

end for

return volume[]

#### 车道流量

车道流量的统计，是根据车道边界检测的结果，具体的步骤如下

1. 获取当前时刻区域内的所有车辆集合；
2. 通过车道边界检测，获取车辆所在的车道编号lane\_num；
3. 筛选指定车道的车辆集合，按车辆类型分别统计车型流量值。

**算法 lane\_volume\_calculation()**

**//用于统计某一时间间隔的车道流量值**

//输入：车辆集合V、每一时刻车辆类型、车道编号m

//输出：车辆类型和车道流量值Volume

###伪代码块

lane\_num  m

veh  vehicle //初始化车辆结构体

volume[type]  0 //初始化不同车辆类型的流量值

for v0,1,…, do

veh  Vv

if (veh.lane\_num == m) then

if (veh.type == type) then

volume[type]++

end if

end if

end for

return volume[]

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 断面或车道编号 |  |
| volume | 标准流量 |  |
| small\_volume | 小车流量 |  |
| medium\_volume | 中型车流量 |  |
| big\_volume | 大型车流量 |  |
| train\_volume | 铰链列车流量 |  |

## 速度

### 定义

GB/T 29108-2012 道路交通信息服务 术语

区间平均速度是指观测距离与车辆通过该距离所用平均行程时间之比，体现了交通流在特定路段空间上的运行状况。



式中：

——观测时间T内，区间平均速度，单位为千米每小时（km/h）；

——观测时间T内车辆数，单位为辆/观测时间；

——区间的长度，单位为米；

——采样时间内到达区间终点的车辆数，单位为辆/采样时间；

——某一车辆进入区间的时间；

——某一车辆驶离区间的时间；

### 计算方法

区间平均速度计算的核心是通过统计车辆在远近道路断面的的过车时间，计算出时间差，具体如下：

1. 获取每一时刻区域内的所有车辆信息；
2. 通过边界跨线检测，分别获取车辆是否驶入区域drive\_in\_zone、是否驶出区域drive\_out\_zone，以及对应的时间time\_drive\_in、time\_drive\_out；
3. 当drive\_in\_zone=true时，将键值对{车辆ID，时间time\_drive\_in}存储到映射Map：中；
4. 当drive\_out\_zone=true时，判断目标车辆ID是否包含在映射中。满足条件时，获取车辆的时间time\_drive\_out，则计算区间时间为

t = time\_drive\_out - time\_drive\_in

同时，从映射中剔除目标车辆ID，并累加区间时间的值和车辆数；

5、根据区间长度L、车辆数、累加的区间时间和，计算区间平均速度值。

**算法 space\_velocity\_calculation()**

**//用于统计观测时间内的区间平均速度值**

//输入：车辆集合V、每一时刻车辆是否驶入区域、驶入时间、是否驶出区域、驶出时间、区间长度L

//输出：区间平均速度

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

time\_sum 0

volume  0

for v0,1,…, do

veh Vv

if (veh.drive\_in\_zone == true) then

Mapveh  Mapveh union {veh.obj\_id,veh.time\_drive\_in} //加入集合

end if

if (veh.drive\_out\_zone == true) then

if (veh.obj\_idMapveh ) then

time\_sum  time\_sum + veh.time\_drive\_out - veh.time\_drive\_in

Mapveh  Mapveh -- {veh.obj\_id,veh.time\_drive\_in} //从集合删除

volume++

end if

end if

end for

return 3.6\*L\*volume/time\_sum

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 断面或区间编号 |  |
| volume | 车辆样本数 |  |
| velocity | 断面平均速度或区间平均速度 | 千米/小时 |

## 车头时距

### 定义

车头时距是指前后两辆车的前端通过同一地点的时间差。



式中：

——观测时间内，平均车头时距，单位为秒；

——车辆类型m对应的标准小汽车换算系数；

——采样时间内的车辆数，单位为辆/采样时间；

——同一车道内，前后相邻的两辆车经过断面的时间差，单位为秒。

### 计算方法

车头时距计算的核心是车道边界检测和边界跨线检测，具体步骤如下：

1. 获取当前时刻区域内所有车辆集合；
2. 通过车道检测和边界跨线检测，获取车辆的车道编号lane\_num和是否驶离区域driving\_out\_zone；
3. 当drving\_out\_zone=true时，将车辆ID、车辆类型、车道编号lane\_num、时间time\_driving\_out等，依次存储在以lane\_num为键的动态映射Map:veh\_map中；
4. 按照车道编号分类，筛选某一车道的子数组veh\_vectori，计算数组中相邻车辆的时间差，根据时间差和跟随车的车辆类型，计算平均车头时距。

**算法 time\_headway\_calculation()**

**//用于统计观测时间内的车头时距值**

//输入：每一时刻车辆类型、是否驶离区域、车道编号m

//输出：车头时距

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

veh\_ratio[type]  {ratioi |i{Ntype}} //初始化车辆换算系数

for v0,1,…, do

veh Vv

if (veh.driving\_out\_zone == true) then

//加入集合

Mapveh Mapveh union {veh.lane\_num,{veh.type,veh.time\_drive\_ out}}

end if

end for

time[]  Mapveh[m] //车道编号m对应的车辆集合

t\_sum  0,volume 0

for i 0,1,…, do

ratio  0

for j  0,1,…, do //判断车辆类型对应的换算系数

if (veh[i+1].type == type) then

ratio = veh\_ratio[type]

end if

end for

t\_sum  t\_sum +ratio\*(time[i+1]-time[i]) //计算车头时间差

volume  volume + ratio[type]

end for

return t\_sum/volume

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 路段编号或车道编号 |  |
| time\_headway | 平均车头时距 | 秒 |

## 车头间距

### 定义

车头间距是指一条车道上同向行驶的前后相邻车辆前端之间的间距。



式中：

——在观测时间T内，平均车头间距，单位为米；

——车辆类型m对应的标准小汽车换算系数；

——采样时间内，区域内的车辆数；

——同一车道上，前后相邻的两辆车间的距离，单位为米。

### 计算方法

车头时距计算的核心是车道边界检测，具体步骤如下：

1. 获取当前时刻区域内所有车辆集合；
2. 通过车道边界检测，获取车辆所在的车道编号lane\_num；
3. 以lane\_num为索引键，将车辆位置、车身类型存储在映射Map：中；
4. 根据车道编号，计算前后两辆车的距离，累加距离和车辆数；根据距离和车辆数，计算平均车头时距。

**算法 space\_headway\_calculation ()**

**//用于统计观测时间内的车头间距值**

//输入：车辆集合V、每一时刻车辆所在的车道编号m、车辆类型

//输出：车头间距

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

veh\_ratio[type]  {ratioi |i{Ntype}} //初始化车辆换算系数

for v0,1,…, do

veh Vv

if (veh.veh\_in\_zone == true) then

Mapveh Mapveh union {veh.lane\_num,{veh.type,veh.x\_pos,veh.y\_ pos}} //加入集合

end if

end for

veh\_pos[]  Mapveh[m] //车道编号m对应的车辆集合

veh\_head\_sum  0, volume 0

for i 0,1,…, do

ratio  0

for j  0,1,…, do //判断车辆类型对应的换算系数

if (veh[i+1].type == type) then

ratio = veh\_ratio[type]

end if

end for

volume  volume + ratio //累加换算的流量值

delta\_x  veh\_pos[i+1].x\_pos - veh\_pos[i].x\_pos

delta\_y  veh\_pos[i+1].y\_pos - veh\_pos[i].y\_pos

//累加车头时距值

veh\_head\_sum  veh\_head\_sum+sqrt(pow(delta\_x,2)+pow(delta\_y,2))

end for

return veh\_head\_sum/volume

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 路段编号或车道编号 |  |
| space\_headway | 平均车头间距 | 米 |

## 密度

### 定义

GB/T 29108-2012 道路交通信息服务 术语

密度是指在某一瞬间内，单位道路长度上存在的车辆数。



式中：

——某一瞬间的交通密度，单位为标准小汽车辆/千米；

——车辆类型i的数量；

——车辆类型i的标准小汽车换算系数；

——路段的车道数；

——道路长度，单位为米。

### 计算方法

与空间占有率计算类似，密度的核心是区域边界的检测，具体步骤如下：

1. 计算区域远近检测断面的距离L；
2. 获取当前时刻区域内所有车辆集合；
3. 通过区域边界检测，获取车辆是否位于区域内veh\_in\_zone；
4. 当veh\_in\_zone=true时，累加车辆数；

根据累加车辆数、路段距离、车道数等，计算密度。

**算法 density\_calculation()**

**//用于统计观测时间内的路段密度**

//输入：所有车辆集合V、车辆是否位于区域内、车道数、路段长度L

//输出：密度

###伪代码块

veh  vehicle //初始化车辆结构体

volume  0

for v0,1,…, do

veh Vv

if (veh.veh\_in\_zone == true) then

volume++

end if

end for

return volume/(N\*L)

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 路段编号 |  |
| density | 密度 | 标准小汽车辆/千米 |

## 通行能力

### 定义

基本通行能力是指道路与交通处于理想情况下，每一条车道（或每一条道路）在单位时间内能够通过的最大交通量。



式中：

——某一时段（通常一小时或者一天），车道的基本通行能力值，单位为标准小汽车量/小时；

——某一时段，车道的饱和车头时距，单位为秒；

——车辆类型m的车辆数；

——同一车道上，车头时距小于等于4的前后两辆车的车头时距值，单位为秒；

——车辆类型m的标准小汽车换算系数；

### 计算方法

通行能力计算的核心是车头时距，具体步骤如下：

1. 获取当前时刻区域内所有车辆集合；
2. 通过车道检测和边界跨线检测，获取车辆的车道编号lane\_num和是否驶离区域driving\_out\_zone；
3. 当drving\_out\_zone=true时，将车辆ID、车辆类型、车道编号lane\_num、时间time\_driving\_out等，依次存储在以lane\_num为键的动态映射Map:veh\_map中；
4. 按照车道编号分类，筛选某一车道的子数组veh\_vectori，计算数组中相邻车辆的时间差，筛选出时间差小于4秒的车头时间差，根据时间差和跟随车的车辆类型，计算平均车头时距；
5. 根据平均车头时距值，计算车道通行能力。

**算法 lane\_capacity\_calculation()**

**//用于统计观测时间内的车道通行能力**

//输入：车辆集合V、每一时刻车辆类型、是否驶离区域、车道编号

//输出：车道通行能力

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

veh\_ratio[type]  {ratioi |i{Ntype}} //初始化车辆换算系数

for v0,1,…, do

veh Vv

if (veh.driving\_out\_zone == true) then

//加入集合

Mapveh Mapveh union {veh.lane\_num,{veh.type,veh.time\_drive\_ out}}

end if

end for

time[]  Mapveh[m] //车道编号m对应的车辆集合

t\_sum  0,volume 0

for i 0,1,…, do

ratio  0

if (time[i+1]-time[i] <=4) then //满足车头时距值小于4秒

for j  0,1,…, do //判断车辆类型对应的换算系数

if (veh[i+1].type == type) then

ratio = veh\_ratio[type]

end if

end for

t\_sum  t\_sum +ratio\*(time[i+1]-time[i]) //计算车头时间差

volume  volume + ratio[type]

end if

end for

return t\_sum/volume

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 车道编号或路段 |  |
| capacity | 通行能力 | 标准小汽车量/小时 |

## 停车次数

### 定义

全永燊 《城市交通控制》 P59

停车次数是指观测时间内，构成一次完全停车和不完全停车的车辆数与车辆总数的比值。

GB/T 29107-2012 道路交通信息服务 交通状况描述

停车次数是指观测时间内，在通过交叉口停车线前车辆排队等候信号周期的个数。用于描述车辆在交叉口时间延误的指标之一。



式中：

——观测时间T内，平均停车次数；

——观测时间T内，车辆总数；

——观测时间T内，每辆车的停车次数；

### 计算方法

停车次数算法的核心是边界跨线检测，以及判断车辆轨迹中完全停车以及不完全停车的个数。具体的计算步骤如下：

1. 获取当前时刻区域内所有车辆集合；
2. 通过边界跨线检测，获取车辆是否驶入区域drive\_in\_zone以及驶出区域drive\_out\_zone；
3. 当drive\_in\_zone=true时，如果车辆速度speed≤5km/h，记初次停车时间veh\_stop\_start和停车持续时间veh\_stop\_end为当前时间戳、车是否停车in\_stop=false、停车次数n\_stop=0，则以车辆编号为索引键，将值{veh\_stop\_start、veh\_stop\_end、in\_stop、n\_stop、speed}存储到中；
4. 当speed>5km/h时，则in\_stop=false，并从中清空车辆编号对应的veh\_stop\_start、veh\_stop\_end、speed值；
5. 当veh\_stop\_end - veh\_stop\_start≥3s&&in\_stop=false时，执行如下判断计算停车次数：
6. 如果数组内存在速度等于0，则=+1；
7. 如果数组内速度大于0，则；

则为目标车辆的停车次数，同时in\_stop=true，并累加车辆数；

1. 当drive\_out\_zone=true时，从中删除车辆信息；
2. 跟据车辆数和停车次数，计算平均停车次数。

**算法 average\_stops\_calculation()**

**//用于统计观测时间内的车道通行能力**

//输入：车辆集合V、车辆速度值、是否位于区域、停车速度阈值vc、停车时间最小阈值tc

//输出：停车次数

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

vc  5，tc  3

volume  0

sum\_stop  0

for v 0,1,…,VN-1 do

veh Vv

if (veh.drive\_in\_zone == true ) then

in\_stop false

speed  sqrt(pow(veh.x\_speed,2) + pow(veh.y\_speed,2))

if (speed ≤ vc) then

//加入车辆结构体

MapvehMapveh union {veh.obj\_id,{ts,te,in\_stop,stops,{speed}}}

else

//移除速度值

MapvehMapveh union {veh.obj\_id,{ts,te,in\_stop,stops,{}}}

end if

vec\_speed  Mapveh[veh.obj\_id]

if ((te - ts) ≥3) && in\_stop == false) then

flag  false

for s  0,1,…,Nvec\_speed-1 do

if (vec\_speed[s] == 0) then

delta\_stop  1

flag  true

break

else

sum\_speed  sum\_speed + vec\_speed[s]

end if

if (flag == false) then

delta\_stop (vc - sum\_speed/Nvec\_speed)/vc

end if

Mapveh[veh.obj\_id].stopsMapveh[veh.obj\_id].stops +delta\_stop

end for

end if

if (veh.drive\_in\_zone == true) then

volume  volume+1

sum\_stop sum\_stop + Mapveh[veh.obj\_id].stops

//移除车辆结构体

MapvehMapveh -- {veh.obj\_id,{ts,te,in\_stop,stops,{speed}}}

end if

end for

return sum\_stop/volume

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 路段编号 |  |
| stops | 每车平均停车次数 |  |

## 排队长度

### 定义

Highway capacity manual 2000

排队长度是指从信号交叉口停止线到上游排队车辆末端之间的距离。即使先前停在队列前面的车辆开始移动，也应该计入排队长度。用来衡量交通拥挤程度。



式中：

——某一时刻，道路上的排队车辆长度，单位为米；

——排队车辆的集合中每辆车距离雷达原点的距离，单位为米；

——交叉口停止线距离雷达原点的距离，单位为米；

### 计算方法

排队长度算法的核心是动态车队组合以及判断区域车道内，处于缓行状态或者排队状态的车队中尾车位置以及头车位置。



1. 动态车队组合

根据车辆的速度（vc≤5km/h），将车流划分为独立的车队；同时，根据车头间距（hc≤20m）和标准车辆数（nc≥2），即临界车队密度为Kc=50pcu/km，动态合并和分裂车队。车队作为分析的单元，密度为k。

2、排队形成

根据车辆的速度，将车辆的运行分为三种状态s：畅通状态、缓行状态（5<v≤10）、排队状态（v≤5）。进而排队开始于，车队畅通状态切换为排队状态，终止于缓行状态切换为畅通状态。在交叉口车辆发生排队的场景有如下两个：

1. 红灯启动状态。从停止线开始，车队依次为排队状态1、缓行状态2、畅通状态3，车辆密度也依次递减；此时k1 > k2，排队长度为排队状态的尾车位置。
2. 绿灯启动状态。从停止线开始，车队依次可能为缓行状态1、排队状态2、缓行状态3、畅通状态4；此时k1 < k2，排队长度为缓行状态3的尾车位置。

根据上述两个场景，最大排队长度和排队车辆的计算步骤如下：

1. 获取当前时刻所有车辆集合；
2. 通过车道边界检测，获取车辆所在的车道编号lane\_num；
3. 以车道编号lane\_num为键，将车辆类型、车辆位置xy、速度v等存储在映射Map：中；
4. 选取特定车道编号m，从停止线处按照速度（vc=5km/h）将车辆依次划分为不同的车队fleeti，并计算每个车队的密度k以及运行状态s{1,0}。当k≥Kc时，将车队存储在队列Queue：fleet\_queue中；
5. 当fleet\_queue.size() ==1 && s==1，queuem等于最后一辆车的位置，否则queuem=0，vehs\_queue=0；
6. 当fleet\_queue.size() > 1，记queue\_initial=false，queue\_continue=false，车队的密度依次为k1、k2、…、kn以及车队尾车位置依次为d1、d2、…、dn，执行如下判断逻辑计算queuem：

6.a）： 当k1 > Kc > k2 && queue\_initial==false，则

queuem =d1，

vehs\_queue =N{xy≤d1}，

queue\_initial=true，queue\_continue=true；

6.b）： 当k1 < Kc < k2 &&queue\_continue==true，遍历后续车队中速度≤10的车辆最远位置d，则

queuem=d，

vehs\_queue = N{xy≤d}，

queue\_initial=false，queue\_continue=true；

6.c）：当k1 < Kc && v(k2,…,kn)>10，则

queuem=0，

vehs\_queue = 0，

queue\_initial=false，queue\_continue=false；

计算得到车道编号m对应的排队长度值queuem。

**算法 max\_queue\_\_caculation()**

**//用于统计观测时间内的排队长度值**

//输入：车辆集合V、每一时刻车辆坐标、车辆类型、车辆速度、车道编号m、排队起始速度阈值vc、排队消散速度阈值ve、车队车头间距阈值hc、车队最小规模阈值nc

//输出：排队长度

###伪代码块

Mapveh  {} //初始化车辆映射集合

veh  vehicle //初始化车辆结构体

veh\_ratio[type]  {ratioi |i{Ntype}} //初始化车辆换算系数

vc  5，ve 10， hc 20，nc 2

queues\_m  queue<vector<veh>>

queue\_initial false，queue\_continuefalse //初始化排队状态

for v0,1,…, do

veh Vv

if (veh.veh\_in\_zone == true) then

Mapveh Mapveh union {veh.lane\_num,{veh.type,veh.x\_pos,veh.y\_ pos,veh.x\_speed,y\_speed}} //加入集合

end if

end for

vehs\_m[]  Mapveh[m] //车道编号m对应的车辆集合

queues\_m  Create\_Queue(vehs\_m[], vc, hc,nc) //动态创建车队

queuem 0，vehs\_queue 0 //初始化排队长度、排队长度的输出

if (queues\_m.size() ==1 && s ==1) then

queuem 0，vehs\_queue 0

end if

if (fleet\_queue.size() > 1) then

if (k1 >= Kc > k2 && queue\_initial==false) then

queuem ，vehs\_queue = Caculate\_Queue(queues\_m,1,vc,hc)；

queue\_initial true，queue\_continuetrue

else if (k1 < Kc <= k2 &&queue\_continue==true) then

queuem ，vehs\_queue = Caculate\_Queue(queues\_m,n,ve,hc)；

queue\_initial true，queue\_continuetrue

else if (k1 < Kc && s ==0 &&queue\_continue==true) then

queuem 0，vehs\_queue  0，

queue\_initialfalse，queue\_continuefalse

end if

end if

//动态创建车队

queue<vector<veh>> Create\_Queue(vector<veh>vehs,float vc, float hc,int nc){

queues {}

temp[]  {}

volume  0

start  0

veh\_Status\_Now true

veh\_Status\_Before  true

for i 0,1,…,Nvehs -1 do

speed  sqrt(pow(vehs[i].x\_speed,2) + pow(vehs[i].y\_speed,2))

if ( temp.size() ==0) then

temp  temp union vehs[i] //加入车辆

volume volume + veh\_ratio[vehs[i].type] //标准小汽车流量

if (speed <= vc) then

veh\_Status\_Before = true

else

veh\_Status\_Before = false

endif

else

if (speed <= vc) then

veh\_Status\_Now = true

else

veh\_Status\_Now = false

endif

while (veh\_Status\_Now == veh\_Status\_Before) do

if (temp.size()<nc) then

temp  temp union vehs[i] //加入车辆

volumevolume + veh\_ratio[vehs[i].type] //标准小汽车流量

end if

if (temp.size() >=nc-1) then

dis = sqrt(pow(vehs[i].x\_pos,2) + pow(vehs[i].y\_pos,2))

k = volume/dis

if (k < 1000/hc) then

temp  temp union vehs[i] //加入车辆

else

if (temp.size() >= nc) then

queues  queues union temp //加入队列

endif

temp  {}

volume = 0

endif

end if

end if

end for

return queues

}

//计算排队长度，排队车辆数

struct{queuem ，vehs\_queue} Caculate\_Queue(queues,n,ve, hc){

i  0

veh  vehicle

flag true

while (i < n &&flag) do

temp[]  queues.front()

if (i > 0) then //当车头时距超过hc时，排队长度终止

h  sqrt(pow(temp[0].x\_pos,2) + pow(temp[0].y\_pos,2))-queuem

if (h > hc) then

break

end if

end if

if (stemp == 1) then

veh temp[Ntemp-1]

queuem  sqrt(pow(veh.x\_pos,2) + pow(veh.y\_pos,2))

vehs\_queue  temp.size()-1

else

for j  0,1,…,Ntemp-1 do //计算队尾的位置

speed  sqrt(pow(temp[j].x\_speed,2) + pow(temp[j].y\_speed,2))

if (speed < ve) then

jj+1

vehs\_queue vehs\_queue + 1

else if (j > 0) then

j j-1

queuemsqrt(pow(temp[j].x\_pos,2) + pow(temp[j].y\_pos,2))

end if

flag false

break

end if

end if

queues.pop() //移除当前车辆的队列

i i+1 //循环下一个车辆队列

end while

return struct{queuem ，vehs\_queue}

}

### 输出格式

|  |  |  |
| --- | --- | --- |
| 字段 | 名称 | 备注 |
| ID | 自增 |  |
| end\_time | 日期时间 |  |
| time\_interval | 统计间隔 |  |
| code | 路段编号 |  |
| max\_queue | 最大排队长度 | 米 |
| nums\_queue | 排队车辆数 | 辆 |

# 交通事件检测





