目录

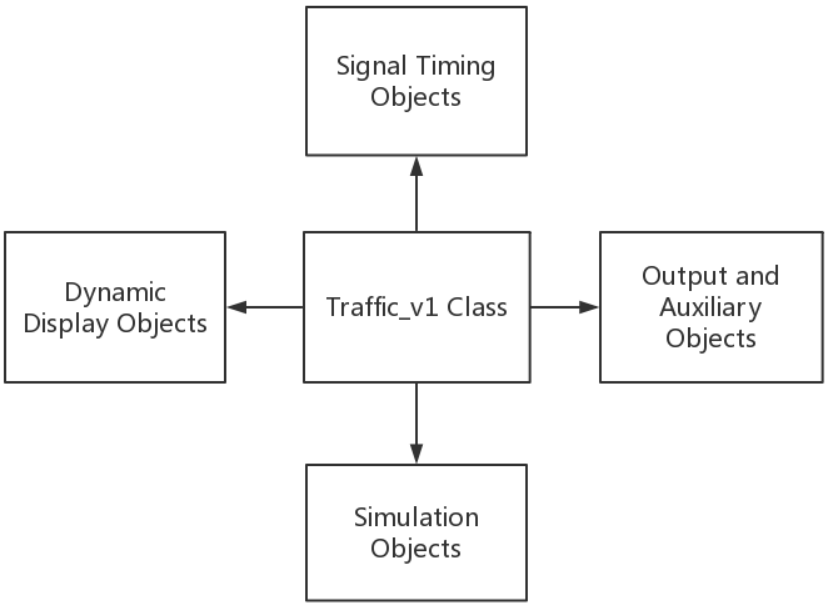

Fig. 1: Traffic_v1 类的架构

1 程序架构

我们采用C++/Qt来开发现行平台，平台的核心是一个Traffic_v1对象，如图1所示。

1.1 显示模块

这些模块用来动态显示仿真过程和控制仿真执行的流程，如设置仿真速度等。主要包括按键，标签，滑动条等对象。

当仿真每向前执行一步的时候，平台将刷新仿真模块来显示仿真过程中的具体情况，整个函数将完成如下几个步骤。

1. 绘制交通路口的基本要素

   根据信号配时模块给出的信号相位，显示模块将绘制红绿灯。同时，显示模块将绘制交通路口前的车道，目前设定绘制30m长，22.5m宽的车道

2. 绘制车辆

   绘制模块将绘制即将进入路口的车辆和驶离路口的车辆，在所有的车

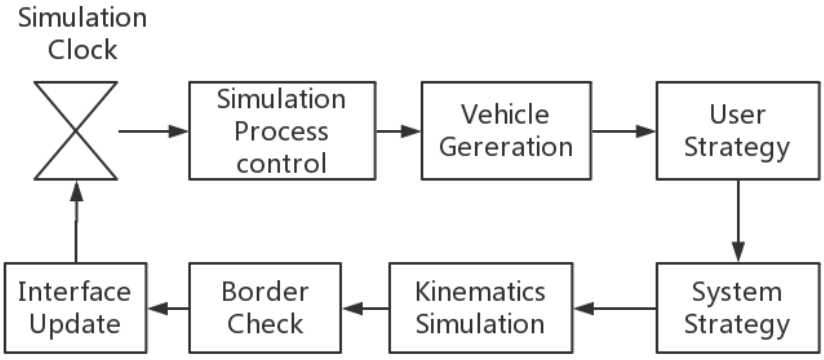

Fig. 2: 平台执行流程

辆中，即将驶入路口的车辆是接受控制的，而驶离路口的车辆是不接受控制的，平台将为他们补全整个的控制策略。

3. 绘制交通路口中的车辆轨迹线

   平台不处理车辆在交通路口内的行为，因此采用绘制车辆在路口中的轨迹线来表示车辆可能存在的位置。当车辆驶入交通路口时，我们假设左转车辆在交通路口的滞留时间是3s，而执行车辆在交通路口内的滞留时间是2s，右转车辆在交通路口内的滞留时间是1s，这些参数可以通过平台代码中的参数简单的修改。在滞留时间内的车辆会在交通路口内显示一道轨迹线，表示车辆可能在的位置。

   可以看出，轨迹线的交点多少可以表示交通路口的混乱程度，如果交通路口内存在着大量的轨迹线交点，交通效率将受到影响，同时交通危险性将显著提升。而这一部分应当是信号配时的主要工作。

1.2 仿真模块

这是整个平台的核心模块，通过设置的仿真速度，平台给出一个间隔为1ms（高速模式），10ms（中速模式），100ms（低速模式），1000ms（调试模式）的信号。

当触发信号被接受时，平台将按照如图2所示的行为进行相关的操作。

在执行过程中，进程控制模块根据用户设定的停止条件决定是否终止仿真，之后，每个道路的入口将根据车辆生成算法来生成新的车辆。之后，用户策略将根据车辆类型的不同分别和用户选择的模式应用在不同的车辆

上。同时系统策略负责离开交通路口的车辆的行为和诸如进入交通路口等边界情况的处理。

所有的策略均调整车辆的加速度，在运动学仿真模块，平台将通过车辆现有的位置，速度和加速度计算车辆下一时刻的位置和速度。

最后，平台将处理进入交通路口的车辆和即将离开研究路段的车辆的边界情况。

程序之后完成界面的重绘工作，并等待下一个触发信号的到来。

根据现行平台实现，在一般配置的笔记本电脑上，在10ms的仿真时钟间隔的情况（中速模式）下系统压力还是比较合适的，同时平台产生的仿真数据也是比较合理的。当仿真时钟间隔被调整到1ms（高速模式），在交通压力过大的情况下会出现时序问题。

1.3 输出和辅助模块

这部分模块的主要作用是在系统重绘时或一些特殊情况发生时输出仿真结果。

随着仿真的进行，系统在可执行文件所在目录根目录中建立文件夹”result”，在此文件夹内建立包含4张CSV表，一个子文件夹，这个子文件夹的命名规则是

’日期+时间+仿真模式选择’

四张CSV表中保存记录了不同类型的数据。

1. car.csv

   这张表保存记录了车辆的初始速度（init_velocity），理论到达时间（thoritical_time）和实际到达时间（act_time），理论到达时间被定义为车辆以最大加速度加速到最大速度之后进行匀速直线运动到达路口所花的时间。理论到达时间 thoritical_time 和实际到达时间 act_time之间的差异可以描述车辆经过这段区域所浪费的时间。

2. stop.csv

   这张表保存记录了每个车道上车辆的总停车次数和总共路口车辆的停车次数以及停车次数占整个车辆数量的比例。

3. road.csv

   这张表记录了每个时刻驶离某一车道的车辆数量和离开系统的车辆数量，和相对应的平均值，这张表往往被用来检查仿真的合理性。

4. stop_time.csv

这张表记录了每个车道上的总共的停车时间和整个系统内的通车时间以及分担到每个车辆上的停车时间。

总的来说，road.csv 被用来观察仿真过程是否正常执行而其他三个表用来评价整个所采用的驾驶策略的效率

另外，停车行为由车辆速度低于某个阈值决定，同时，当车辆停下时，停车时间不断累加，而停车次数不发生变化。这一点在长期停车时尤其明显。

1.4 信号配时模块

信号配时模块满足了用户自主设置信号周期和相位的要求，三个按键'set red behind'、 'set green behind'、 'set yellow behind' 被用来设置黑色时间线之后的信号相位。

同时，这个界面可以用来调整信号的周期。因此，通过调整黑色时间线或指定确定的调整时间，不断重复设置时间线之后的信号相位等工作，用户可以完全自主的完成信号的配时工作。

同时，左侧的界面将显示当前交通路口车辆的轨迹线，当前时间是黑色轨迹线所指示的时间。通过交通轨迹线的显示和对交叉情况的判断，系统可以在一定程度上辅助用户完成更好更合理的交通信号配时工作。

1.5 平台界面

仿真模块的界面和信号配时的界面如图3 和图4所示。

在仿真模块界面上点击'Edit Traffic Light'按钮可以激活信号配时模块的界面，如果用户关闭信号配时模块界面，他所作的变更将被保存同时仿真模块界面将被再次激活。

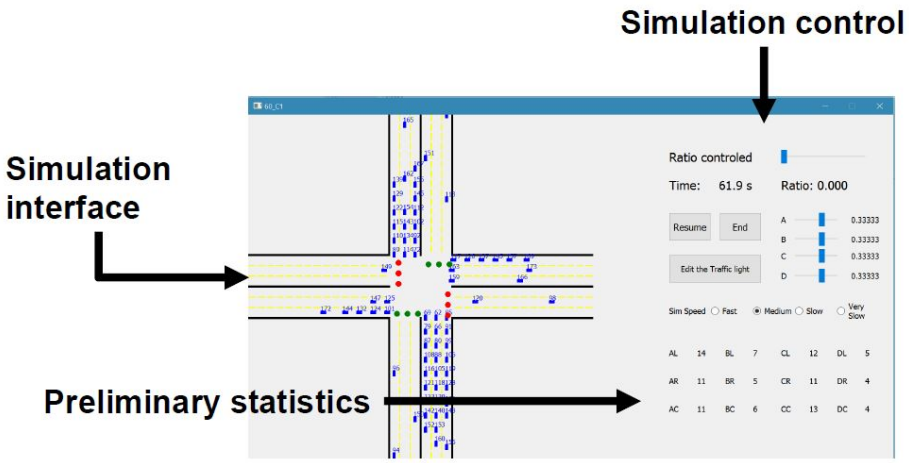

Fig. 3: 仿真模块界面

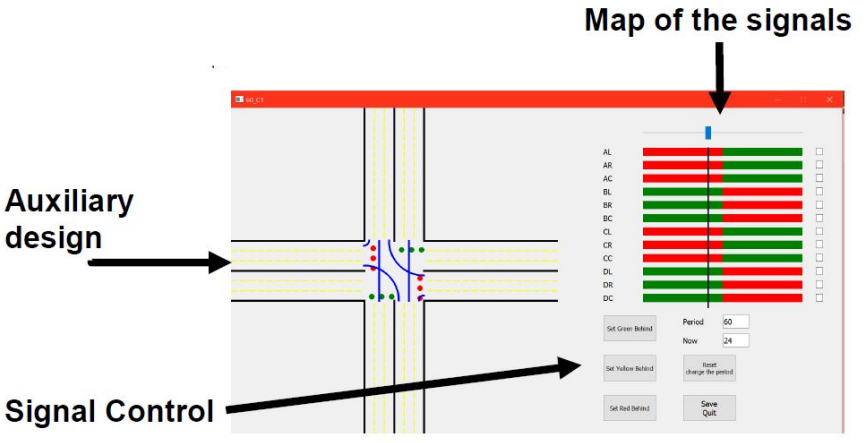

Fig. 4: 信号配时模块的界面

2 算法简介

2.1 基础假设和参数

首先，我们认为对车辆而言，车辆的加速度是可控的物理量，同时，其他的物理量（如速度和位置）由车辆的加速度和当前时刻的速度根据如下运动学公式确定：

$$v(t+\Delta_t) = v(t) + a\Delta_t$$

$$x(t+\Delta_t) = x(t) + v(t)\Delta_t + 0.5a\Delta_t^2$$

另外，由于车辆的加速度也不能被准确的控制，我们在策略给出加速度的准确值之后加入了一个正态分布的随机噪声，尤其是后文提到的人工驾驶的车辆模型，引入的正态分布随机噪声更大。

同时，平台对车辆的加速度和速度也有着限制，我们认为车辆加速度的绝对值的最大值是a_{max}，被用来限制车辆的加速过程和刹车过程。另外，车辆的最大速度被限制为v_{max}，车辆的最低速度被限制为v_{min}。

不论选择了那种驾驶策略，如果策略给出的加速度值大于最大加速度a_{max}或者小于$-a_{max}$，抑或是计算得到的速度大于最大速度或小于最小速度，程序将自动将这些值设定为对应的边界值。如果车辆受到红灯影响而被迫刹车停车，最小速度限制将被设定为0来描述可能的停车行为。但不论如何，倒车行为是永远不可接受的。

同时我们假设车辆期望的速度，v_{exp}，被设定为车辆最高限速的80%。

为了简化模型，我们认为在所研究路段进行驾驶的过程中，车辆不进行变道行为。这个假设在现实情况中也是可以被接受的，因为受到交通法规和规则的限制，车辆在交通路口前的变道行为往往是被禁止或限制的。车辆需要在进入交通路口之前完成相关的变道工作，因此我们的假设是合理的，同时仅仅从交通效率的角度来看，换道行为也只能降低交通效率，不利于对驾驶策略进行评估。因此，因为平台的评估主要以策略为主，就不考虑车辆的换道行为了。

因此，在一个车道进行驾驶的车辆可以被描述成一个队列结构，即Q_{in}。

2.2 车辆产生算法

我们假设车辆的到达过程是一个泊松过程，这个到达过程受到下面算法的限制。

我们假设泊松过程的强度是λ到达车辆而定时间间隔S_n, S_{n-1}可以被表述为

$$P((S_n - S_{n-1} \leq t) = 1 - e^{-\lambda t}$$

因为仿真过程的时间间隔被设定为0.1s，每小时的平均车流量即为

$$\bar{F} = 36000\lambda$$

对于一个方向的车辆，车辆的小时流量被仿真界面上的交通流控制滑块设定之后，这个方向的车流量将按照特定的泊松过程进行生成$\lambda = \frac{\bar{F}}{3600}$，并且被压入每个方向上的等待队列$Q_{pending}$。

在生成车辆之后，下一步是考虑车辆应当放入那个车道，我们认为车辆放入合理车道的可能性是相同的，合理车道被定义为车辆放入之后仍能安全行驶的车道。其具体定义可以被定义为，车辆生成位置（S_{start}）和最后一个车辆的距离X_{-1}之间的距离大于安全行驶距离S_{safe}即，

$$X_{-1} - S_{start} \leq S_{safe}$$

遍历所有符合条件的合理车道，并且将$Q_{pending}$中的车辆等可能的放入其中的一条车道上。新生成的车辆的位置为S_{start}，即车辆生成线的位置。

如果目前没有合适的放入车辆的车道，车辆将被保留在$Q_{pending}$队列中并等待下一个仿真循环。

车辆生成的算法在整个仿真流程的第二步进行，紧随仿真流程控制模块。

2.3 路口车辆排队的产生和消散

为了描述路口车辆排队的产生和消散过程，我们对每个驶入交通路口的车道引入队列Q_{block}。同时，根据现行信号相位和队列的长度，我们采用的所有控制策略几乎均采用如下提到的简单算法

1. 红灯

   如果队列Q_{in}中的第一辆车辆和队列Q_{block}中的最后一辆车辆之间的距离小于控制距离$S_{control}$，队列Q_{in}中的第一个车辆将刹车并期望停在队列Q_{block}最后一辆车的期望距离之后。这个两车之间的期望距离被定义为S_{stop}，用来描述在队列Q_{block}中的车辆间距。

   如果Q_{block}为空并且Q_{in}中的第一辆车辆和停车线之间的距离小于$S_{control}$，Q_{in}中的头车以停车线S_{end}为目标进行刹车。

   如果Q_{in}中的头车距离路口太远以至于上述两条均不满足，Q_{in}中的头车将自由驾驶。

同时，Q_{block}中的车辆应当停车等待直到信号灯变绿。

2. 绿灯在车辆驾驶策略的设计中，队列Q_{in}的第一辆车辆和队列Q_{block}中的最后一个车辆（如果他存在）之间的距离不应当小于安全距离S_{safe}。如果两车之间距离小于安全距离，后车将采取刹车行为。

综上所示，整个算法的简明思路可以用如下的伪代码表述

```
01 if (Q_block.empty()){
02  if (Light == Green)
03   Q_in.first().drive_freely();
04  else
05   Q_in.first().brake_to(S_end);
06 }
07 else{
08  if(Q_block.last().pos-Q_in.first().pos< S_control)
09   Q_in.first().brake_to
10   (Q_block.last().pos-S_stop);
11  else
12   Q_in.first().drive_freely();
13 }
```

代码中brake_to(desired_pos) 函数的意义是设置车辆的加速度使得车辆能够刹车到期望的位置上

另外，队列Q_{block}的产生和消散将遵循如下的模型。

1. 产生/增长

   当队列Q_{in}中的第一辆车辆距离距离Q_{block}中的最后一辆车辆（如果不存在，则设为停车线S_{end}）的距离小于停车距离S_{stop}时，这辆车将从Q_{in}中移除并被加入到Q_{block}中。

2. 消散

   当信号灯为绿灯时，队列Q_{block}中的所有车辆将按照消散速度v_{dis}向前移动，由于这个速度值设定得很小，使得加速过程和其他的相关过程可以忽略笔记。当车辆越过停车线时，将被从队列Q_{block}中移除。

以上所述均可以用下面的伪代码表示

```
01 if (light == green){
02  Q_block.all_move_forward(v_dis)
```

```
03 }
04 else{
05  if(!Q_block.empty()){
06   if(Q_block.last().pos-Q_in.first().pos
07    < S_stop){
08    Vehicle v;
09    v=Q_in.getfirst();
10    Q_block.push(v);
11   }
12  }
13  else{
14   if(S_end-Q_in.first().pos< S_stop){
15    Vehicle v;
16    v=Q_in.getfirst();
17    Q_block.push(v);
18   }
19  }
20
21 }
```

以上描述了路口队列的产生和消散过程和这个模型和其他道路要素的联动。经过仿真实验，由于模型设定的假设导致的差别是可以忽略的。由于这个模型的引入，我们可以简单的将车辆的驾驶和排队模型解耦，使得我们能够更简单的实现其他的驾驶模型。

3 驾驶模型概述

3.1 基本模型

在对人工驾驶模型和自主驾驶模型进行描述的过程中，车辆的跟驰模型和自由驾驶模型是经常被使用的模型。因此在讨论其他的驾驶模型和控制策略之前，我们首先讨论这两个模型

3.1.1 自由驾驶模型的实现

在不存在速度引导的情况下，车辆的行为可以大体上被分类为自由驾驶模型和人工驾驶模型两类，这两个模型将在本节和下一节讨论。控制距

离$S_{control}$被定义成前后两个车辆之间的距离。因此，上述两种驾驶模型将按照如下的规则进行选择。

1. 当和前车之间的距离小于预先设定的控制距离$S_{control}$时，我们选择跟驰模型描述车辆的行为

2. 当和前车之间的距离大于控制距离$S_{control}$时，我们选择自由驾驶模型来描述车辆的行为。自由驾驶模型中，车辆将通过一些加速度的调整来将自己的速度设定为期望速度v_{exp}，如果当前的速度值和v_{exp}相近，车辆所采取的加速度可能比较小，如果当前速度和v_{exp}相差比较远，车辆所采取的加速度可能比较大，这描述了车辆在出现可能的停车之后的起步加速直至正常行驶的过程。

   同样，由于有随机噪声的存在，如果车辆的速度大于最大速度v_{max}，车辆将采取适当的制动步骤。

3.1.2 车辆跟驰模型的实现

上文已经提到，当和前车的距离过近的时，车辆将采用跟驰模型来描述其行为，在我们使用的策略中，我们采用了Wiedemann生理——心理模型来描述车辆的行为，即：

$$a_n(t+\Delta_t) = \frac{[\Delta v_{n,n-1}(t)]^2}{2[\Delta x_{n,n-1}(t) - S_{exp}]} + a_{n-1}(t)$$

在上面的式子中，S_{exp}代表了期望的车辆之前的安全行驶距离，因为车辆在是将交通路口时速度变化十分剧烈，因此期望距离S_{exp}不应是一个常数。因此，根据交通安全法规和驾驶一般情况，我们使用了和前车速度线性相关的跟驰距离来描述期望的跟驰距离，因此，期望跟驰距离S_{exp}如下所示：

$$S_n(t) = \alpha v_{n-1}(t) + S_{safe}$$

其中S_{safe}和2.3提到的相同，而常数α可根据实际情况和经验设定。

一旦车辆之间的间距小于S_{exp}，亦即车辆之间的距离过近，车辆将采取紧急刹车制动,车辆采取的加速度值可能根据具体情形的不同而变化，在大多数情形下，车辆将采取的刹车加速度绝对值为$0.5a_{max}$

3.2 人工驾驶模型

平台提供了三种模型来验证平台的有效性，第一个模型是人工驾驶模型。

正如名称所示，人工驾驶模型描述的是车辆在人的驾驶操控下的行为，这个模型经常被用来提供一个交通效率的基本指标，而其他的控制策略至少应当取得比人工驾驶模型好的控制效果，否则没有意义。

我们首先讨论距离交通路口较远（距离大于S_{inter}）的车辆的行为。由于车辆距离路口太远，这些车辆将不受红绿灯相位的影响。因此，这些车辆可能采取建档的跟驰模型或自由驾驶模型。对于队列Q_{in}中的第一个车辆，如果它距离交通路口也较远的话，他应当采取自由驾驶模型。

对于那些距离交通路口较近的车辆，根据信号灯相位和具体场景的不同，不同的策略将被调用来描述车辆的行为，使得车辆的行为更加符合当时场景的要求。

1. 绿灯同时Q_{block}为空

   在这种情况下，车辆的行为和之前描述的行为基本相同

   另外，除之前描述的行为，车辆还需要考虑绿灯的剩余时间。如果剩余时间小于T_{safe}，我们认为车辆能够在绿灯时通过交通路口的可能性已经比较小了，因此车辆可能采取刹车减速，具体的行为和红灯时车辆的行为大致相同。

2. 绿灯同时Q_{block}为非空

   在这种情况下，车辆会刹车使得自己的速度大致为v_{dis}，并同时试图进入队列Q_{block}，一旦它进入队列Q_{block}，它将随着队列中的其他车辆一起向前做匀速直线运动。

   同样，对于还没有进入队列中的车辆，绿灯剩余时间也是要考虑的，如果剩余时间小于T_{safe}，车辆可能会采取制动并考虑在适当的位置停车。

3. 红灯在这种情况下，车辆会和之前在2.3一节中讨论的简单行为一致

另外，我们着重于研究队列Q_{in}头车的行为，因为剩下的车辆的行为可以被简单使用跟驰模型来描述。

将所有可能的情况纳入考虑之后，人工驾驶模型可以用如下的伪代码进行描述。

```
01 //the code below just consider the vehicle v
02 if(S_end-v.pos>S_inter)//far from intersection
03 {
04  //code block 1
```

```
05  if(car-following_requirement_met())
06   //the car-following requirement is   met
07   car_following();
08  else
09   //the head car of the queue or
10   //the car-following requirement isn't met
11   free_driving();
12 }
13 else //close to the intersection
14 {
15  if(light==red)
16   method1.3();//the method mentioned in 1.3
17  else
18   if(Q_block.empty()){
19    if(time_remain>T_safe)
20    //same as the code block 1
21    else
22     if(!pass_check())
23     //same as the red light case
24    }
25   else
26    if(time_remain>T_safe)
27    acc_to_speed(v_dis);
28    else
29     if(!pass_check())
30     //same as the red light case
31    }
32
33 }
```

平台将遍历队列Q_{in}，而在每次迭代中，车辆的加速度会通过上面的模型计算和设置。

3.3 单车车路协同模型

通过对跟驰模型进行一定改进，我们设计了只考虑车辆和信号灯之间互动的单车车路协同模型。在这个模型中，我们考虑了交通信号对车辆的影响

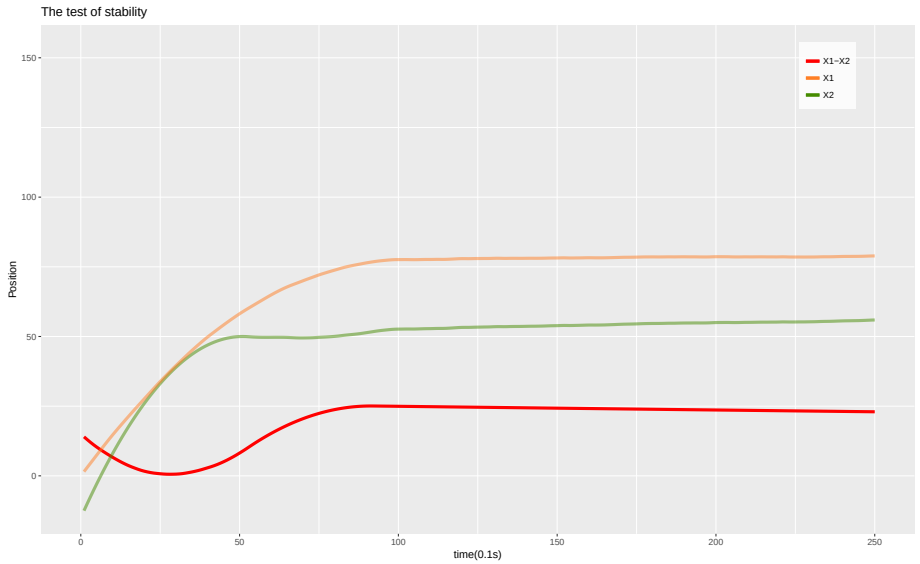

Fig. 5: The position of the vehicles and the distance between them

3.3.1 对跟驰模型的修改

首先，当车辆和前车的距离小于控制距离$S_{control}$时，我们仍然选择跟驰模型。作为对现有生理-心理跟驰模型的修改，我们设计了如下表述形式来作为自主驾驶车辆的策略控制函数

当$\Delta x_{n,n-1}(t) \leq S_{exp}$成立时，

$$a_n(t+\Delta_t) = -sgn(\Delta v_{n,n-1}(t)) \frac{[\Delta v_{n,n-1}(t)]^2}{2[\Delta x_{n,n-1}(t) - S_{exp}]} + a_{n-1}(t)$$

这表明，当后车速度低于前车速度时，控制策略会引导后车加速跟上前车的运动。

应用了这个小改动之后，我们通过一些实验研究得到修改过的跟驰模型工作良好并且能保持前后车辆之间的合理车距。

如图5所示，跟随的后车采用了经过修改的跟驰模型，而前车采取了一个急剧并且随机性很大的刹车，从图中我们可以看到两个车辆之间的间距最后会稳定在S_{exp}，并且基本不受到初始状态的影响。从另外的一些实验结果也可以看出随着前车的加速，后车也会加速并稳定在几乎同样的跟随距离中。

3.3.2 车路协同驾驶策略

除了对车辆的跟驰策略进行了修改，我们也将车辆的自由驾驶模型修正为车路协同的驾驶策略。

为方便后文叙述，我们先定义两个时间参数，最大时间花费T_{max}被定义为车辆按照可行的最大加速度a_{max}进行减速到最低限速v_{min}之后保持匀速直线运动到路口的时间，而最小时间花费T_{min}被定义为车辆以最大加速度a_{max}进行加速到最高限速v_{max}后进行匀速直线运动直到路口所花费的时间。显然，车辆在路中的实际运行时间在T_{min}和T_{max}之间，并且这两个时间参数之和车辆进入控制路段的初始速度和初始加速度有关。

之后，策略将为车辆在$(T_{min}, T_{max}]$区间内为车辆计算定出一个最早的绿灯时间T_G。如果上述区间内没有合理的T_G绿灯时间，车辆将采取自由驾驶策略。

如果在上述过程中可以定出一个合理的T_G，策略将指导车辆按照这个到达时间为目标调整自己的加速度，在这段时间中，如果车辆被前车影响，车辆可能需要重新进入跟驰模型

同时，车辆在交通路口前的表象和之前提到的人工驾驶模型等相同。

总的来说，这个策略是车辆跟驰模型版本的优化版，同时考虑了信号灯相位对于整个路段车辆的影响。

3.4 多车车路协同模型

这是一个简单的有车间通信的车路协同主动控制策略。整个策略中队列Q_{in}的第一辆车进行处理直到队列的Q_{in}的最后一辆车。

首先，对于队列中的Q_{in}中的第一辆车，策略使用3.3.2中的方法确定一个合适的到达时间T_G，为了方便后文表述，我们设定队列Q_{in}中的第n辆车辆的合理到达时间T_G是$T_G^{[n]}$，例如，第一辆车的到达时间就是$T_G^{[1]}$。

在上面确定了$T_G^{[1]}$之后，第一辆车辆将按照设定的合理到达时间$T_G^{[1]}$进行加速或者减速。如果$T_G^{[1]}$不存在，车辆将首先加速到期望速度v_{exp}，之后减速到停车线刹车。

之后，我们假定队列Q_{in}中的第$n+1$辆车辆能够获得队列Q_{in}中的第n辆车的合理到达时间$T_G^{[n]}$，包括其是否存在。

之后，队列中第$n+1$个车辆来根据他的最短到达时间T_{min}，最长到达时间T_{max}来确定一个合理到达时间$T_G^{[n+1]}$，并且确保式子$T_G^{[n+1]} - T_G^{[n]} \geq 1\text{second}$成立，这是因为后车的到达时间总是晚于前车。

如果前车的合理到达时间$T_G^{[n]}$不存在，则当前车辆的到达时间$T_G^{[n]+1}$可以相对自由的设置，只需要考虑最大到达时间T_{max}和最小到达时间T_{min}的限制即可。

之后和前面相同，第$n+1$个车辆根据设定的合理到达时间设置自己的加速度，之后策略开始处理后面的车辆。

由于车辆的合理到达时间是严格设计过的，保证了前车一定在后车之

前到达路口，即使是不存在合理到达时间的车辆也存在一个加速过程，尽可能的拉远与后车之间的距离。因此车辆之间发生车距过近的概率将显著降低，因此相关的避撞策略可以相对的简化，在本策略中，仅采用了比较简单的紧急情况（车辆间距过小的紧急刹车行为）加以处理。

3.5 策略执行顺序和混合驾驶策略

3.5.1 单控制策略仿真

平台将策略的执行过程处理为对车辆队列Q_{in}的迭代。整个迭代过程从队列Q_{in}的最后一辆车开始直到第一辆车，这是因为我们的策略主要以获得前车的速度和加速度为主，这种迭代方式能够更好的处理数据，对于对迭代有要求的多车车路协同策略，将按照策略的迭代方式进行处理

因此，如果仅选用了一个策略，程序的处理方式相当于是一个从后向前的对于队列Q_{in}的循环过程。

3.5.2 混合驾驶策略的实现

我们仅关心人工驾驶策略和自主驾驶——速度引导策略的混合结果，因此，平台提供了两种仿真方式，人工驾驶（在3.2中提到）和单车车路协同策略（在3.3中提到）的混合以及人工驾驶策略和多车车路协同策略（在3.4中提到）的混合。

在车辆的产生过程中，车辆的不同控制策略类型被按照在交互界面上用户设定的比例随机分配到每一个车辆上。

在生成了不同车辆的不同策略类型属性的之后，在策略的执行环节，平台按照车辆所采用的固定策略来处理车辆并设置车辆合理的加速度。我们假设当一个多车车路协同车辆跟随一个单车车路协同或人工驾驶车辆时，前面车辆的合理到达时间T_G被设置为不存在。因此，采用多车车路协同策略的车辆可以自由的设定自己的合理到达时间T_G。