本期我们将介绍 VISSIM 4.3 版本二次开发的基本指令，包括路网运行控制、仿真参数设置、检测参数获取及控制算法嵌入。同时将附上一个完整的匝道控制案例源代码（包含 inp 路网文件）。

1 前 言

自 VISSIM 二次开发系列前十一期文章发布以来，小编收到了很多反馈和问题，主要集中在以下几点：

(1) 许多读者朋友是用 matlab、vb 或者其他平台进行 VISSIM 二次开发，对 C#不熟悉；

(2) 有许多特定的需求，比如：无人驾驶场景模拟，驾驶行为参数标定、交叉口自适应控制等；

(3) 希望公开检测参数获取和匝道控制的源代码。

第一个问题，希望读者朋友们能够掌握开发的原理和方法，平台仅仅是工具；第二个问题，如果有好的案例或者 DEMO，希望能够拿出来分享，有问题也可以通过智联交通 QQ 群一起讨论；第三个问题，本期会介绍 4,3 版本开发的基本指令，并附上匝道控制案例的完整源代码。

2 匝道控制场景

本期以一个匝道控制场景的构建过程为例，总结 VISSIM 4.3 二次开发的基本指令。

如图 1 所示为实施匝道控制的快速路瓶颈区域，因其地理位置原因（该匝道是南环高架南段唯一的一处上匝道，毗邻商圈），上匝道需求较大，匝道汇入主线为 3 并 2，且汇入后的 2 车道宽度较窄，匝道车流汇入持续对主线交织区产生干扰，导致汇入点通行能力下降。，高峰时段主线排队部分时段超过 1 公里。为缓解拥堵，使上匝道车辆有序进入主线，管理部门拟在入口匝道处设计信号控制灯，在周边分流路口设置 VMS 信息板，并与地面信号控制相协调，限于篇幅，本文仅介绍匝道控制的一种方案——ALINEA 控制（相关资料参考第五期）。

(a) 匝道控制地理位置

(b) 高峰期快速路主线拥堵

图 1 快速路瓶颈区域

如图 2 所示为瓶颈区域早高峰车速及车流量变化，由匝道上下游车速、流量数据可见：匝道下游的主线通行能力保持稳定，在 4000pcu/h 左右，而上游主线流量在 2800~3200pcu/h，匝道流量达到 900~1200pcu/h 水平时，主线上游流量交通状态开始恶化。因此将匝道交通汇入量控制在 900pcu/h 以下能够保证快速路主线运行的通畅和通行能力的保持。

(a) 早高峰瓶颈区域各路段车速变化

(b) 早高峰瓶颈区各路段车流量变化

图 2 瓶颈段早高峰车速及流量变化

3 VISSIM 建模

首先在 VISSIM 软件中搭建好相应的快速路模型，设置好相应的检测器，本案例中一共设置了三类检测器，即：行程时间检测器（TravelTime）、交织区占有率检测器（DataCollection）、排队长度检测器（QueueCounter），各检测器布设位置如图 3(b)所示。

(a) 快速路模型（车速<10km/h 显示红色）

(b) 检测器布设

图 3 VISSIM 快速路建模

在进行检测参数读取时，为了简便起见，预先在 VISSIM 软件中激活各检测器并设置好要检测的参数（检测周期也需与代码中设置的一致，本文设置 100 仿真秒），如图 4 所示。

图 4 软件中完成相应检测设置

同时需要设置匝道信号灯，控制类型为周期控制，控制周期 120 仿真秒，如图 5 所示。需要说明的是，以上检测器和信号灯编号均为 1。

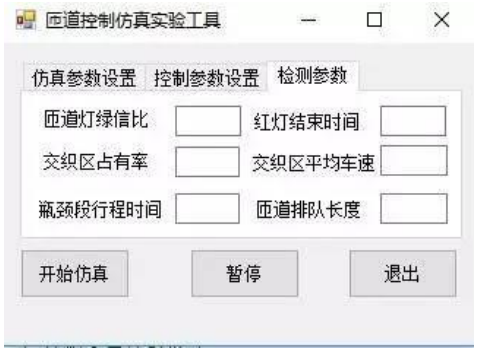
图 5 匝道控制信号灯

4 匝道控制 C#实现

本文中依据以构建一个匝道控制仿真实验工具为例进行说明，基本的 VISSIM 4.3 COM 接口库加载、C#开发环境设置、控件设置等内容参考第一期文章，不再敖述，实验工具界面如图 6 所示。

(a) 仿真参数设置

(b) 控制参数设置



(d) 检测参数设置

图 6 匝道控制仿真实验工具

整个系统的运行流程如图 7 所示：

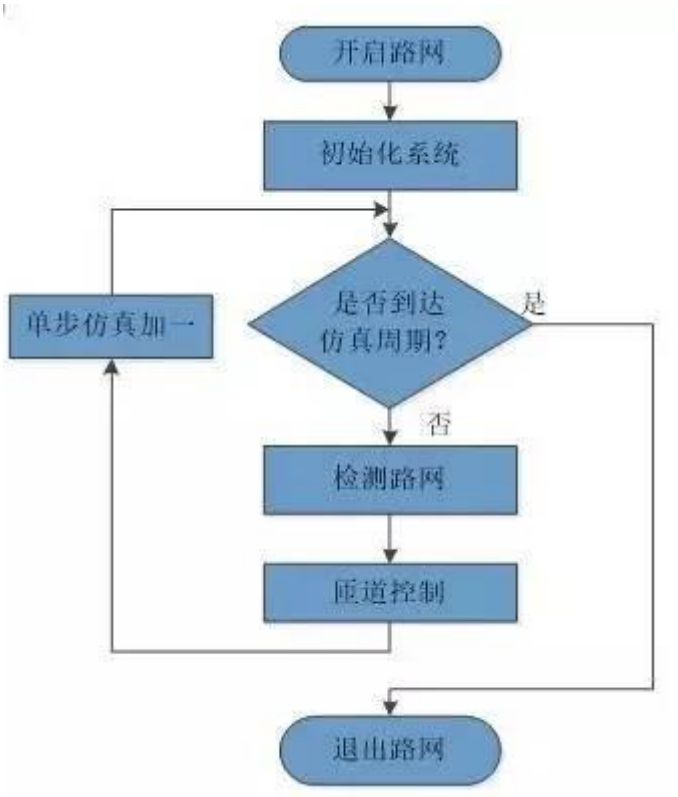


图 7 运行流程

启动路网，完成初始化后按照循环单步仿真的逻辑运行，根据设定的检测周期和控制周期调用相应的函数，到达仿真周期后退出路网并执行系统复位。相关函数如下：

① IntialSystem() 初始化路网对象、控件参数等；
② RsetSystem() 系统变量及控件参数复位
③ DetectNetwork() 获取路网各检测器参数
④ RampControl() 匝道控制函数

源代码如下，读者朋友们可以加入智联交通 QQ 群（365583964）进行下载：

① 全局变量设置

```
/*系统变量*/
string pathExeFile = System.Windows.Forms.Application.StartupPath; //存储当前.exe文件路径
bool startFlag; //仿真开始标志位
int itemDetectPeriod; //检测周期
int itemControlPeriod; //控制周期
int itemTimeStep;
/********VISSIM对象*****************/
Vissim vissim;
String vissimModelFilePath;
/*控制变量及参数*/
int itemOccupancyRate_Bottlencek; //交织区检测占有率
int itemSpeed_Bottleneck; //交织区检测车速
int itemTravelTime_Bottleneck; //瓶颈路段行程时间
int itemQueueLength_Bottlencek; //排队长度
```

② 初始化函数

```
public void InitialSystem()
{   vissim = new Vissim(); //定义VISSIM对象
    vissimModelFilePath = pathExeFile + @"\宛平南路上匝道VISSIM路网模型\WangPingRampControl.inp";
    vissim.LoadNet(vissimModelFilePath, 0); //开启路网
    vissim.Simulation.Resolution = 1;  //仿真精度设置为1
    vissim.Simulation.Period = Convert.ToInt32(this.textBox_SimPeriod.Text); //仿真周期
    vissim.Net.VehicleInputs.GetVehicleInputByNumber(1).set_AttValue("Volume", Convert.ToInt32(textBox_MainVolume.Text));// 主线流量设置
    vissim.Net.VehicleInputs.GetVehicleInputByNumber(2).set_AttValue("Volume", Convert.ToInt32(textBox_RampVolume.Text)); //上匝道流量设置
    vissim.Evaluation.set_AttValue("DATACOLLECTION", true); //软件激活检测器评价
    vissim.Evaluation.set_AttValue("TRAVELTIME", true);
    vissim.Evaluation.set_AttValue("QUEUECOUNTER", true);
    itemDetectPeriod = Convert.ToInt32(this.textBox_DetectPeriod.Text); //检测周期
    itemControlPeriod = Convert.ToInt32(this.textBox_ControlPeriod.Text); //控制周期
    this.labelSimTime.Text = ""; //仿真步数显示控件初始化
    this.progressBarSystem.Minimum = 0; //进度条显示控件最小值设置
    this.progressBarSystem.Maximum = Convert.ToInt32(vissim.Simulation.Period); //进度条显示控件最大值设置
}
```

注：进行了路网加载、主线及匝道流量输入及检测器激活。

③ 主流程函数

```
private void button_Start_Click(object sender, EventArgs e)
{
    InitialSystem();

    startFlag = true;

    for (itemTimeStep = 1; itemTimeStep <= vissim.Simulation.Period; itemTimeStep++) //“循环单步仿真”主体
    {
        System.Windows.Forms.Application.DoEvents(); //外部按键事件读取

        if (startFlag)
        {
            vissim.Simulation.RunSingleStep(); //单步仿真

            labelSimTime.Text = "仿真时长 " + Convert.ToString(itemTimeStep);

            this.progressBarSystem.Visible = true;

            this.progressBarSystem.Value++;
        }

        else return;

        if (itemTimeStep % itemDetectPeriod == 0 && itemTimeStep != 0)
        {
            DetectNetwork(); // 检测路网参数
        }
        if (itemTimeStep % itemControlPeriod == 0 && itemTimeStep != 0)
        {
            RampControl(); //实施匝道控制
        }
    } RestSystem();}
```

注：以“循环单步仿真”的方式运行路网，同时实施检测及控制，相关资料可参考第二期文章。

④ 路网检测函数

```
public void DetectNetwork()
{
    DataCollection dataCollection; //定义检测器对象，以下类推

    TravelTime travelTime;

    QueueCounter queueCounter;

    dataCollection = vissim.Net.DataCollections.GetDataCollectionByNumber(1); //获取检测器1的对象，以下类推

    travelTime = vissim.Net.TravelTimes.GetTravelTimeByNumber(1);

    queueCounter = vissim.Net.QueueCounters.GetQueueCounterByNumber(1);

    itemOccupancyRate_Bottlencek = Convert.ToInt32(dataCollection.GetResult("OCCUPANCYRATE", "SUM", 0)); //获取占有率参数

    textBox_LinkDensity.Text = Convert.ToString(itemOccupancyRate_Bottlencek);

    itemSpeed_Bottleneck = Convert.ToInt32(dataCollection.GetResult("SPEED", "MEAN", 0)); //获取速度参数

    textBox_LinkSpeed.Text = Convert.ToString(itemSpeed_Bottleneck);

    itemTravelTime_Bottleneck = Convert.ToInt32(travelTime.GetResult(itemTimeStep, "TRAVELTIME", "", 0)); //获取行程时间参数

    textBox_TravelTime.Text = Convert.ToString(itemTravelTime_Bottleneck);

    itemQueueLength_Bottlencek = Convert.ToInt32(queueCounter.GetResult(itemTimeStep, "MAX")); //获取排队长度参数

    textBox_RampQueue.Text = Convert.ToString(itemQueueLength_Bottlencek);
}
```

注：获取各检测器参数，注意所获取的参数需实现在软件中设定好，检测周期也需要与软件中一致，否则易报错或得不到正确检测结果。

⑤ 匝道控制

```
public void RampControl()
{
    SignalController signalController = vissim.Net.SignalControllers.GetSignalControllerByNumber(1);

    SignalGroup signalGroup = signalController.SignalGroups.GetSignalGroupByNumber(1); //匝道控制信号灯组

    double signalCycle = Convert.ToInt32(textBox_ControlPeriod.Text); //信号周期

    double rampControlParametr = Convert.ToInt32(textBox_ParametKp.Text); //匝道控制比例系数Kp

    double criticalOccupancyRate = Convert.ToInt32(textBox_DesiredOcc.Text); //最佳车道占有率

    double redEndCurrent = signalController.get_AttValue("REDEND"); //当前红灯结束时间

    double greenRate1 = (signalCycle - redEndCurrent) / signalCycle; //计算当前绿信比

    double greenRate2 = greenRate1 + rampControlParametr * (criticalOccupancyRate-itemOccupancyRate_Bottlencek )*0.01;

    greenRate2 = Math.Max(Math.Min(greenRate2, 1), 0); //将绿信比范围限定在0_1之间

    int redEndOptimal = Convert.ToInt32(signalCycle * (1 - greenRate2)); //优化后的红灯结束时间

    textBoxRampGreenRate.Text = Convert.ToString(greenRate2); // 显示绿信比

    textBoxRampRedEnd.Text = Convert.ToString(redEndOptimal); // 显示红灯结束时间

    if (greenRate2 >= 0.8)
    {
        signalGroup.set_AttValue("TYPE", 2);        //“TYPE”2=持续绿灯

        signalGroup.set_AttValue("REDEND", 1);
     }
     else if (greenRate2 <= 0.2)
    {
        signalGroup.set_AttValue("TYPE", 3);        //“TYPE”3=持续红灯

        signalGroup.set_AttValue("REDEND", Convert.ToInt32(signalCycle)-1);
    }
    else
    {
        signalGroup.set_AttValue("TYPE", 1);        //“TYPE”1=周期控制

        signalGroup.set_AttValue("REDEND", redEndOptimal);
    }
}
```

注：以 ALINEA 算法为主，通过控制上匝道车辆通过率，维持交织区车道占有率在最适值附近。

⑥ 复位函数

```
public void RestSystem()
{
    this.button_Pause.Text = "暂停";

    this.labelSimTime.Text = "";

    this.progressBarSystem.Visible = false;

    this.progressBarSystem.Value=0;

    vissim.Exit();
}
```

以上即为平台构建核心代码，编译运行后，效果如图 8 所示。

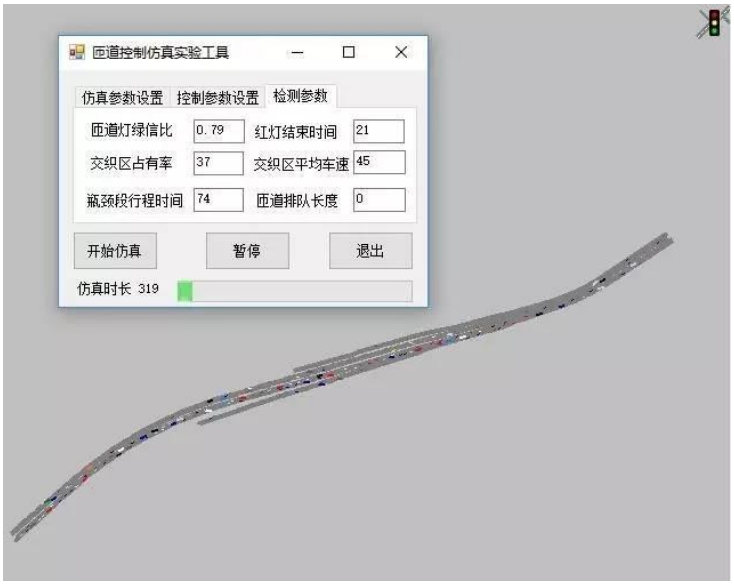


图 8 平台运行效果

5 总 结

本期所介绍的代码基本包括了 VISSIM 4.3 的常见操作，感谢您的支持！