目录

1. 实验简介

本实验是在 CyberTORCS 平台上进行的智能车控制仿真实验。仿真平台由杨明教授实验室成员黄炫圭、杨辰兮开发。用户可通过设计、编写自己的算法，利用车辆控制的输入输出变量，根据道路弯度、车辆偏航等因素改变车体状态，实现平台中车辆的自动行驶。本实验分为三部分：巡线竞速模式实验、巡线精度模式实验、泊车实验。编程平台为 Visual Studio。

巡线实验给出多条难度不同的赛道，要求编写一套算法，使车辆能沿着这些赛道跑完全程，考评分为精度模式测评和速度模式测评。

泊车实验给出多个泊车位的坐标，要求编写算法，使车辆能够以车尾或车头入库方式泊入车位，考评车位以泊车速度和精度（车辆四个顶点距离标准位置的偏差）作为评判标准，速度越快，精度越高，则算法越完善。

希望同学们在完成实验的同时，对智能车控制有更多了解，也欢迎大家加入智能车团队！

上海交通大學 实验报告

2. 巡线实验-速度模式

一、实验目的和要求
二、实验内容
三、实验工具
四、操作方法和实验步骤
五、算法框架与程序代码
六、实验结果与分析（必填）
七、讨论、心得

一、实验目的和要求

1. 学习、掌握 CyberTORCS 仿真平台的使用
2. 理解用于车辆控制的输入、输出变量的确切含义
3. 学会用 Microsoft visual studio 2017 编程实现车辆控制
4. 通过实验对 PID 等控制算法有更深入理解
5. 设计自己的巡线控制算法

二、实验内容

1. 参看软件说明书，熟悉 CyberTORCS 的各菜单
2. 选择 CyberOnHand 菜单，任意选择赛道，利用上下左右键手动控制赛车，对车辆、赛道有直观感受。
3. 用 Microsoft visual studio 2017 打开 driver_onhand.sln，通过改写 driver_onhand.cpp 中的代码输出自己感兴趣的参数，同时了解档位与发动机转速之间的关系。
4. 用 Microsoft visual studio 2017 打开 driver_cruise.sln，将说明书中的简单例程写入 driver_cruise.cpp 中，选择 CyberCruise 菜单，观看车辆运动状态
5. 通过改写 driver_cruise.cpp 中得代码，对难易不同的赛道进行巡线高速控制编程。

要求：最终能用一套程序跑完所有赛道，不能在程序中对赛道进行识别以使用不同代码。

三、实验工具

CyberTORCS 仿真平台、Microsoft visual studio 2017

四、操作方法和实验步骤

参见软件说明书

五、算法框架与程序代码

1. PID 转向控制

智能车领域经典的控制方法有开关控制、纯比例控制、PID 控制方法等。其中前两种方法虽然简单，易操作，但有着系统响应慢、不稳定、产生净差等明显缺点。PID 方法通过对偏差信号进行比例、积分和微分运算变换来消除偏差，对于小车可以起到更好的控制和调节作用。

PID 控制原理图如下：

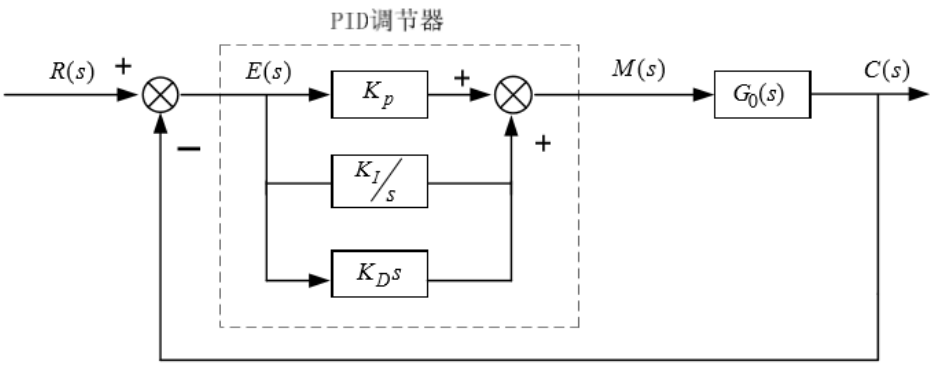


图 1 PID 调节

在调节 PID 参数时，我们主要实验试凑法进行调参，采用杨老师提供的 PID 整定口诀，对三个参数进行调节。同时，我们利用输出的 csv 文件存储行驶数据，利用 Python 可视化来观察车辆的行驶状况，尽可能使偏差接近 4:1 衰减的理想状态，并朝着减少震荡的方向努力。PID 整定口诀如下：

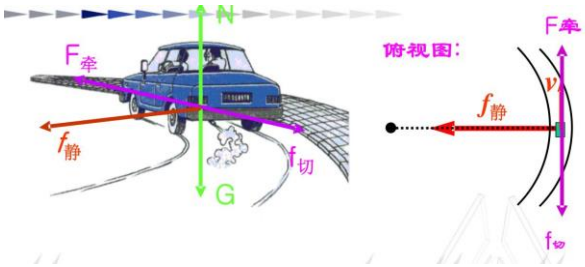
参数整定找最佳，从小到大顺序查。先是比例后积分，最后再把微分加。
曲线振荡很频繁，比例度盘要放大。曲线漂浮绕大湾，比例度盘往小扳。
曲线偏离回复慢，积分时间往下降。曲线波动周期长，积分时间再加长。
曲线振荡频率快，先把微分降下来。动差大来波动慢，微分时间应加长。
理想曲线两个波，前高后低四比一。一看二调多分析，调节质量不会低。

PID 转向控制的核心代码如下，如需查看竞速模式 driver_cruise 的完整代码，可前往 https://github.com/C-ZiHao/Introduction-to-Control-Course/blob/master/driver_cruise_SpeedMode.cpp 查看。

```
1.  //PID 转向控制核心代码
2.  D_err =-atan2(_midline[vpoint][0],_midline[vpoint][1]) //vpoint 是动态预瞄点
3.  //PID 参数
4.  kp_d = 6.025;
5.  ki_d = 0.0;
6.  kd_d = 52.329;
7.  D_errSum = D_errSum + D_err;//积分控制
8.  D_errDiff = D_err - Tmp;//微分控制
9.  Tmp = D_err;
10. *cmdSteer = constrain(-1.0,1.0, kp_d*D_err + ki_d*D_errSum+ kd_d*D_errDiff);
```

2. 速度控制

在竞速模式中，速度是控制的核心，速度越快成绩越好。但速度过快，又可能会导致转向调节不及时，易触边产生 Damage。我们组扫描前方一定路段测得最小曲率半径，根据该曲率半径设置目标速度。由于比赛道路有沥青路沙路结合的两种赛道，由于在沙路上路面凹凸不平且容易打滑，误差变化频繁且不稳定，我们通过调整 offset 对两种不同的道路，进行速度和加速度的调整。



针对车辆的转弯过程中，我们将车辆模型进行简化，利用动力学知识将车辆速度控制为道路曲率半径的根号关系。

图 2 速度控制

速度控制的核心代码如下：

```
startPoint = max(float(_speed * _speed * 0.00096), float(_speed * 0.48));
MIN = 501;
for (int i = 5; i < startPoint; i++)
{//寻找 startPoint 内最小曲率半径 MIN
  circle cirs = getR(_midline[i - 2][0], _midline[i - 2][1], _midline[i][0],
    _midline[i][1], _midline[i + 2][0], _midline[i + 2][1]);
  if (MIN > cirs.r){MIN = cirs.r;}
}
//根据曲率半径控制速度，使用根号关系控制
if (MIN < 30)  {expectedSpeed = constrain(60, 93, 22 * sqrt(MIN));}//转弯时
else if (MIN < 50)  {expectedSpeed = constrain(127, 140, 16 * sqrt(MIN));}
else if (MIN < 120)  {expectedSpeed = constrain(160, 240, 18.4 * sqrt(MIN));
}
else  { expectedSpeed = constrain(270, 300, MIN);}
curSpeedErr = expectedSpeed - _speed;
speedErrSum = 0.1 * speedErrSum + curSpeedErr;
if (curSpeedErr > 0)
{  //加速控制
   if (abs(*cmdSteer) < 0.6)
   {
      *cmdAcc=constrain(0.0,1.0, kp_s*curSpeedErr+ki_s*speedErrSum+offset);
      *cmdBrake = 0;
    }
   else if (abs(*cmdSteer) > 0.70)
   {
      *cmdAcc = 0.005 + offset;
      *cmdBrake = 0;
```

```
    }
    else
    {
        *cmdAcc = 0.11 + offset;
        *cmdBrake = 0;
    }
}
else if (curSpeedErr < 0)
{  //减速控制
     *cmdBrake = constrain(0.0, 0.8, -0.065 * curSpeedErr / 5 + offset / 3);
     *cmdAcc = 0;
}
```

3. 预瞄点贴内弯

竞速模式中，车辆行驶轨迹的控制是提高成绩的重要方法，经过不断的尝试，我们小组最后选用了通过动态调整预瞄点来达到贴内弯效果的方法。

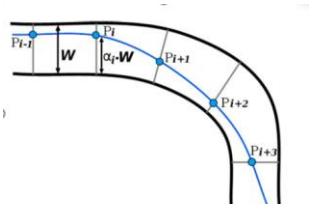


图 3 控制策略选择

车辆在入弯时，贴弯道的内弯，出弯时，贴外弯，此时车辆转弯半径较大，保证车辆速度可以较大。我们采取了根据道路曲率半径动态调整预瞄点 vpoint 的措施，同时在测试时我们也针对不同道路，引入了道路宽度_width 这一影响因子，使得预瞄点的控制更加稳定。

预瞄点控制公式如下

```
vpoint = min(23 + 0.0017 * MIN * MIN, 45) * _width / 10.5; //MIN 是曲率半径
```

六、实验结果与分析

不同赛道实验结果如下：

道路	成绩
Michigan	45.35
Street	92.36
CG Track3	72.38
Brondehach	95.00
E-Track4	119.85
E-Road	79.40
CG-Speedway	48.12
Dirt6	100.18
Etrack-5	38.446
Dirt4	116.988

表 1 竞速模式成绩

每条赛道都有最适合它的代码，但是为了适应不同的赛道，必须提高算法泛性，足以适应不同赛道。但从结果可以看出，算法还存在一定的改进空间。在同一路面的赛道，算法适配性尚可，但是在混合赛道 Mixed1，由于不能及时发现路面变化，车辆反应滞后，导致在路面变换时出现严重打滑等问题。

七、讨论、心得

巡线实验-速度模式是我们开始控制导论课程做的第一个比赛模式，所以开始着手最重要的 是了解编程规则，读懂初始代码。初始代码中有很多我们很容易忽略的参数和公式，这些参数的获取能帮助我们打开思路，找到更多的可能性。比如_yaw，在速度控制时，作为一个影响因子，提高车辆行驶的稳定和平滑；利用理论上转弯时摩擦力提供向心力，转弯半径与速度的二次方关系，得到理想速度的表达式。

其次，PID 控制是一个适用范围很广的经典控制方法，对模型的要求不高，适用于很多控制。kp 是比例系数，kp 越大，误差减小越快，但太大容易产生震荡；ki 作为积分系数，主要起到消除净差的作用，但 ki 在竞速模式起到的作用并不会不大；kd 是积分系数，在竞速模式下，因为速度快，需要提高车辆的预判能力，增大 kd 可以使车辆在转弯时表现更好。通过 PID 的调参过程我们也明白了不能一味暴力调参，需要通过分析出现的问题寻找可能的原因并探索解决方法，这样更高效，代码也更加泛性。

同时，在调参的过程中，也要学会利用输出并进行可视化，盲目根据直观感觉的判断可能会引起错误的判断，而将输出数据可视化后，可以直观的得到车辆运行的状况，并有针对的进行调参。

在车辆最佳行驶轨迹的探索中，我们组通过查阅文献资料，尝试了多种策略。我们尝试了靠近弯道-靠外弯-切内弯-出外弯的方法，但结果较不理想，之后也尝试着应用均值滤波等滤波算法对输出数据处理之后再进行控制，也未取得理想的结果。分析后，我们认为，有两个主要的原因，首先弯道情况过于复杂，双 S 型弯道、连续急弯等情况对策略的要求较为复杂，仅仅凭借前方 200m 的曲率半径情况，很难得出合适的弯道策略。其次，Wtorcs 赛道由多段道路拼接完成，曲率半径并不连续，对算法有一些影响。所以，我们最终依旧采用了通过动态调整预瞄点来达到切内弯效果的算法。

在竞速模式中，首先我们没有考虑混合赛道的存在，识别路况采用的是起步的识别，在 Mixed 赛道突然进入沙地后造成了打滑现象，所以在精度模式中的探索中，我们采取了另一种判断方法。在速度的分段考虑时也没有考虑极陡的弯道，如 Mixed1 的连续急弯曲率半径仅有 10，而我们限制了整个行驶过程最低下限为 60，速度太快难以通过赛道。考虑不周也导致我们未能完成 Mixed 赛道的测试，而 Mixed 赛道存在，也让我们意识到了算法泛化性的重要。

通过这个实验，我们也学会了理论联系实际的学习方法。车辆驾驶是我们生活中常见的事情，如何把实践的经验转化成编程语言是需要我们面临的问题。通过查阅资料，了解实际智能车及赛车比赛里的竞速策略、不同路面赛车的控制状态等知识，对竞速模式的成绩提高也有很重要的作用。

上海交通大學 实验报告

3. 巡线实验-精度模式

一、实验目的和要求
二、实验内容
三、实验工具
四、操作方法和实验步骤
五、算法框架与程序代码
六、实验结果与分析（必填）
七、讨论、心得

一、实验目的和要求

6. 学习、掌握 CyberTORCS 仿真平台的使用
7. 理解用于车辆控制的输入、输出变量的确切含义
8. 学会用 Microsoft visual studio 2017 编程实现车辆控制
9. 通过实验对 PID 等控制算法有更深入理解
10. 设计自己的巡线控制算法

二、实验内容

6. 参看软件说明书，熟悉 CyberTORCS 的各菜单
7. 选择 CyberOnHand 菜单，任意选择赛道，利用上下左右键手动控制赛车，对车辆、赛道有直观感受。
8. 用 Microsoft visual studio 2017 打开 driver_onhand.sln，通过改写 driver_onhand.cpp 中的代码输出自己感兴趣的参数，同时了解档位与发动机转速之间的关系。
9. 用 Microsoft visual studio 2017 打开 driver_cruise.sln，将说明书中的简单例程写入 driver_cruise.cpp 中，选择 CyberCruise 菜单，观看车辆运动状态
10. 通过改写 driver_cruise.cpp 中得代码，对难易不同的赛道进行巡线高精度控制编程。

要求：最终能用一套程序跑完所有赛道，不能在程序中对赛道进行识别以使用不同代码。

三、实验工具

CyberTORCS 仿真平台、Microsoft visual studio 2017

四、操作方法和实验步骤

参见软件说明书

五、算法框架与程序代码

1.核心控制算法

精度模式作为典型的轨迹跟踪问题，常见的有 PID 算法、Pure Pursuit 算法与 Stanley 算法，三种算法的特性如下表[1]。

算法	鲁棒性	路径要求	转弯内切	稳态误差	适用场景
PID	较差	无要求	不会	速度增加变大	路径曲率较低及低速的跟踪场景
Pure Pursuit	较好	无要求	速度增加变严重	速度增加变大	路径连续或不连续或低速的场景
Stanley	好	曲率连续	不会	速度增加变大	路径平滑的中低速跟踪场景

表 2 轨迹跟踪算法优缺点

在之前的竞速模式中，为了追求速度，我们采用的是 PID 算法。然而在精度模式中，PID 算法鲁棒性较差，并不能起到良好的效果。在尝试了 Pure Pursuit 算法和 Stanley 算法后，我们最终采用了 Stanley 算法与 PID 算法相结合的策略。

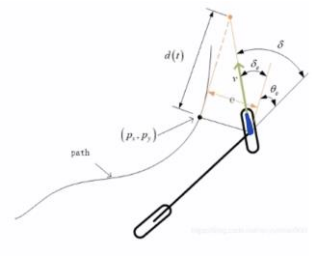


图 4 Stanley 算法示意图

采用 Stanley 算法公式为：

$$D\_err = \_yaw - atan((k_0 * \_midline[0][0]) / (\_speed+1))$$

其中，由于 wtorcs 的限制，我们用_midline[0][0]代替了前轮的横向偏差，这样虽然有一些误差，但对后续的影响并不大。

通过经验判断与不断尝试，我们最终将可调节参数 k_0 确定为 50。

精度模式核心的控制代码如下，如需查看精度模式 driver_cruise 的完整代码，可前往 https://github.com/C-ZiHao/Introduction-to-Control-Course/blob/master/driver_cruise_AccuracyMode.cpp

```
if (shaflag == 1)//较难行驶道路的 PID 参数
{
    kp_d = 12.5;
    ki_d = 0;  //将 ki 置为 0
    kd_d = 40;
    kp_s = 0.017;
    offset = 0.05;
}
else  //常规道路的 PID 参数
{
    kp_d = 7.6;
    ki_d = 0.4;
    kd_d = 40;
    offset = 0.06;
}
```

```
D_errSum = D_errSum + D_err;
k0 = 50;
//根据 Stanley 算法给出 D_err
D_err = _yaw - atan((k0 * (_midline[0][0])) / double(_speed + 1));
D_errDiff = D_err - Tmp;
Tmp = D_err;
//再结合 PID 算法控制方向
*cmdSteer = constrain(-1.0, 1.0, kp_d * D_err + ki_d * D_errSum + kd_d * D_errDiff);
```

2. 路况判断策略

我们将道路分为较难行驶的道路与常规道路两种，在常规道路上我们采用的是 Stanley 算法与 PID 算法相结合的策略，而在较难行驶的道路上，考虑滚阻的影响，我们将 PID 参数中的 ki 置为 0 避免震荡。同时，对于两种不同的路况，也采取不同的速度参数、PID 参数、加速度参数等，在较难行驶的道路上，使用较慢的速度、较小的加速度与较大的 PD 参数来保证较小的误差。

道路判断方面，我们借用其余小组的经验，使用滑移率方差进行判断，当滑移率方差大于指定阈值时，判断进入较难行驶的道路，将 shaflag 置为 1，并在此过程中不断迭代更新路况判断，及时修改进行 shaflag 的值，路况判断具体代码如下。

```
{
  testsum = 0;
  testf = 0;
  for (int n = 0; n < 199; n++)  //以过去 200 个点的滑移率为参照
  {
      test[n] = test[n + 1];
      testsum = testsum + test[n];
  }
  if (_gearbox == 1){ratio = 3.82;}  //利用 XML 文档查取的挡位转化关系
  else if (_gearbox == 2){ratio = 2.15;}
  else if (_gearbox==3){ratio = 1.56;}
  else  if(_gearbox==4){ratio = 1.21;}
  else if(_gearbox==5){ratio = 0.97;}
  else if (_gearbox == 6){ratio = 0.8;}
  test[199] = _rpm/_speed/ratio;  //基于挡位、速度、转速进行计算
  testsum = testsum + test[199];
  for (int n = 0; n < 200; n++)  //求取方差 testf
  {
      testf = testf + pow((test[n] - testsum / 200),2);
  }
  //利用 testf，进行判断
  if ((testf > testflag) || (dataOutCount < tmpflag + 3000))
  {
```

```
        if (testf > testflag)  //方差大于指定阈值，判断为较难行驶的道路
        {
                tmpflag = dataOutCount;
                shaflag = 1;
                testflag = 1;
        }
   }
   else
   {
       shaflag = 0;
       testflag = 10;
   }
}
```

3. 速度控制

速度控制方面，我们采用分段控制的方法，针对两种路况给出了两套控制函数，并对原有的油门刹车算法作出了一些调整。在较难行驶的道路上，采用更慢的速度来保证行驶的精度，同时，针对不同的道路，通过改变 offset 的值，来对油门与刹车做出动态的调整，从而达到更好的控制效果。

```
if (shaflag == 1)
{   //较难行驶道路的速度分段控制，MIN 是前方预瞄范围内的最小曲率半径
    if (MIN < 11)          {expectedSpeed = constrain(30, 30, MIN);}
    else if (MIN < 41)     {expectedSpeed = constrain(50, 50, MIN);}
    else if (MIN < 61)     {expectedSpeed = constrain(60, 60, MIN);}
    else if (MIN < 120)     {expectedSpeed = constrain(80, 80, MIN);}
    else if (MIN < 250)     {expectedSpeed = constrain(90, 90, MIN);}
    else if (MIN < 350)     {expectedSpeed = constrain(100, 100, MIN);}
    else                    {expectedSpeed = constrain(120, 120, MIN);}
}
else
{   //正常道路的分段控制
    if (MIN < 11)          {expectedSpeed = constrain(30, 30, MIN);}
    else if (MIN < 31)     {expectedSpeed = constrain(50, 50, MIN);}
    else if (MIN < 61)      {expectedSpeed = constrain(70, 70, MIN);}
    else if (MIN < 91)     {expectedSpeed = constrain(90, 90, MIN);}
    else if (MIN < 120)    {expectedSpeed = constrain(110, 110, MIN);}
    else if (MIN < 250)    {expectedSpeed = constrain(140, 140, MIN);}
    else if (MIN < 350)    {expectedSpeed = constrain(150, 150, MIN);}
    else                   {expectedSpeed = constrain(170, 170, MIN);}
}
curSpeedErr = expectedSpeed - _speed;
speedErrSum = 0.1 * speedErrSum + curSpeedErr;
if (curSpeedErr > 0)
```

```
{   //加速控制
    if (abs(*cmdSteer) < 0.1)
    {
        if (abs(_yawrate) > 0.2) //较大转向时限制油门
        {
          *cmdAcc=constrain(0,0.4,kp_s*curSpeedErr+ki_s*speedErrSum+offset);
        }
        else
        {
          *cmdAcc=constrain(0,1.0,kp_s*curSpeedErr+ki_s*speedErrSum+offset);
        }
        *cmdBrake = 0;
    }
    else if (abs(*cmdSteer) < 0.40)
    {
        if (abs(_yawrate)>0.2 && dataOutCount>500) //较大转向限制油门防打滑
        {
          *cmdAcc=constrain(0,0.4,kp_s*curSpeedErr+ki_s*speedErrSum+offset);
        }
        else
        {
         *cmdAcc=constrain(0,0.8,kp_s*curSpeedErr+ki_s*speedErrSum+offset);
        }
        *cmdBrake = 0;
    }
    else if (abs(*cmdSteer) > 0.70)
    {
        *cmdAcc = 0.11 + offset / 2;
        *cmdBrake = 0;
    }
    else
    {
        *cmdAcc = 0.3 + offset;
        *cmdBrake = 0;
    }
}
else if (curSpeedErr < 0)
{ //减速控制
    *cmdBrake = constrain(0.0, 0.7, -kp_s * curSpeedErr / 5 - offset / 3);
    *cmdAcc = 0;
}
updateGear(cmdGear); //挡位更新
```

六、实验结果与分析

利用以上算法，我们针对 wtorcs 中的不同的道路做出测试，测试结果如下表所示:

道路	成绩
Michigan	108.28
Street	122.85
CG Track3	91.31
Brondehach	121.12
E-Track4	152.00
E-Road	131.70
CG-Speedway	96.52
Dirt6	141.78
Mixed1	143.78
Dirt1	77.58

表 3 精度模式成绩

可以看出，该算法在不同道路下，鲁棒性较好，对于沙地、沥青路等不同地形也能较好的适用。在精度模式最终的比赛中，我们也利用这套算法获得了四分的满分。虽然在三条比赛道路中并没有进入前三的成绩，但我们的算法泛性较好，在三条赛道中都名列中上，也取得了较好的成绩。

七、讨论、心得

不同于竞速模式，精度模式的核心在于 Error 的控制，即控制的准确度，但也不能一味的为了准确而降低速度。在第一次提交后，我们下载第一名的 dll 并进行了分析，发现差距并不在于 Error，而在于所用的时间，所以在后续的调节中，我们也在尽可能保证精度的条件下提高速度。速度与精度的权衡，这也是精度模式最为困难的权衡难题。

在精度模式中，我们接触了 Pure Pursuit 算法与 Stanley 算法，我们组主要尝试了 Stanley 算法。在与其它组的交流中，我们发现 Stanley 算法在较急的弯道时，控制的精度远好于 Pure Pursuit 算法，但在直道或常规弯道时，最终的结果却不如 Pure Pursuit 算法。因此我们认为最好的控制策略应该是针对不同道路曲率半径，设定阈值，分段采取 Pure Pursuit 算法与 Stanley 算法，但由于最终时间的限制，这一思路最终作罢。

我们采用的路况判断算法，思路借鉴于第 10 小组的滑移率方差判断。之所以将它命名为对较难行驶的道路与常规道路的判断而不是沙地路与沥青路的判断，是因为我们认为不同道路最大的区别在于摩擦力、滑移率等因素的变化，而不是道路材质的判断。所以我们采取的算法对于某些地区（如 dirt4），并不会将它识别为较难行驶的道路，而针对一些较滑的沥青路反而会识别为较难行驶。路况的判断核心是对道路参数的判断，如果能实时的检测道路的参数，并利用道路参数函数化控制参数，相信这样一定会取得更好的泛化成果。在研究过程中，有组员提出在一段道路连续加速或减速来计算参数，但如此又对道路有一定的要求，最终，我们在这一方面也未取得较大的突破。

在实验中，合理利用可视化工具可以起到事倍功半的作用。使用可视化工具可以直观的、很方便的根据实际情况观察程序的输出，并根据输出状况进行参数的调节与整定、对一些特殊情况进行补偿与调整。在这里，再次感谢吴士一老师提供的可视化工具，为我们提供了很大的便利与帮助。

精度模式很重要的一点还在与细节的调整，如起步的调整。在起步时，由于车辆并非位于路中央，需要快速回到路中央。再比如速度的控制，在精度模式比赛后，通过听优秀小组的汇报，我们意识到了自己一个很大的不足，一些优秀小组针对不同曲率下的速度做了极为精细的调整，通过不断的测试来寻找曲率对应的速度阈值。相比之下，我们仅仅利用经验来对速度和曲率的关系做出设定，并未去寻找能通过特定曲率的弯道的最大速度，这也是我们小组的一个不足之处。

上海交通大學 实验报告

4. 泊车实验

一、实验目的和要求
二、实验内容
三、实验工具
四、操作方法和实验步骤
五、算法框架与程序代码
六、实验结果与分析（必填）
七、讨论、心得

一、实验目的和要求

1. 掌握 CyberTORCS 仿真平台的使用
2. 理解新增的输入变量的确切含义
3. 设计自己的泊车控制算法

二、实验内容

1. 打开 wtorcs_new.exe，选择 CyberParking 菜单，泊车赛道为 Road Tracks 中的 CG Track3，单击“New Race”，观察最小样例程序的泊车过程。
2. 根据泊车规则设计算法。泊车规则：将车泊入绝对坐标为(lotX, lotY)的车位，成绩计算：泊车时间*（1+10*偏差距离/车长），以下情况泊车失败：
   - 泊车时有 damage 值大于 1 的碰撞，
   - 没有到达规定里程（即开始计时的里程）就置位 bFinished，
   - 置位 bFinished 时车体速度大于 0.2 km/h
   - 最后车体的航向角和提供的 lotAngle 之差超过 10 度
3. 用 Microsoft visual studio 2017 打开 driver_parking.sln，在 driver_parking.cpp 中的适当位置编写自己的跟车程序。

三、实验工具

CyberTORCS 仿真平台、Microsoft visual studio 2017

四、操作方法和实验步骤

参见软件说明书

五、算法框架与程序代码

1. 泊车策略

我们主要尝试了双圆弧法和漂移入库法 2 种方法来完成泊车任务。

双圆弧法：通过反复调节转向的参数，保证车辆具有合适的转弯半径；通过积分的方法，测量车辆走过的长度，在到达一定的路程目标后自动切换为下一段轨迹，使车辆连续走过两个圆心角为 45 度的圆弧，达到车体转过 90 度的效果，圆弧半径与车辆相对于车库的位置有一定的几何关系，随后倒车入库，并微调车辆与车位之间的夹角为 0 度。待停稳后，置位泊车变量。置位完成则加速向右开出车位。具体示意图如下：

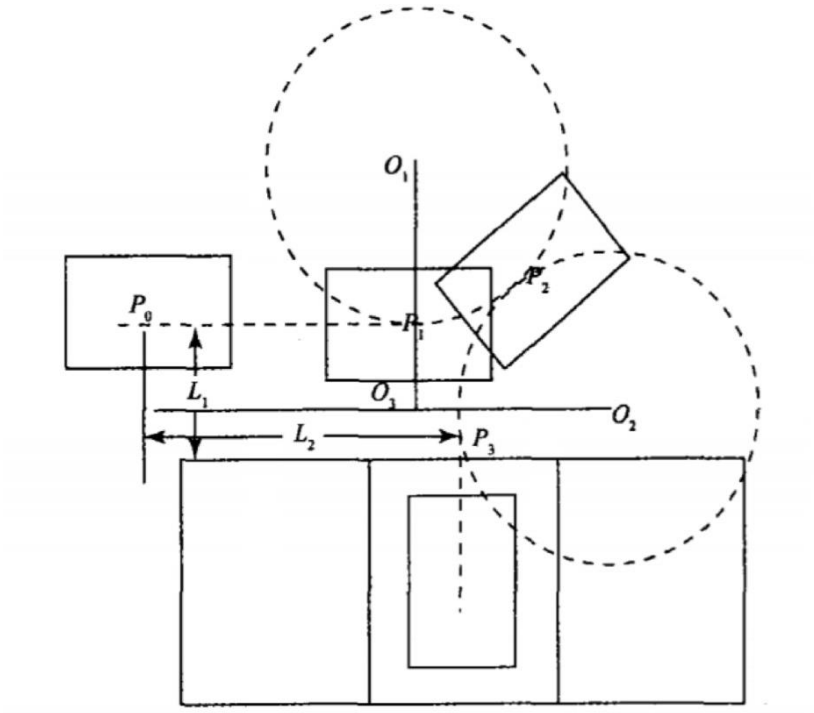


图 5 双圆弧法泊车示意图

漂移入库法：采用巡线-甩尾-倒车-刹车-泊车成功-加速驶离的方式完成入库泊车。具体步骤可分为：

① 在离车位较远处，程序采用巡线方式行驶。当检测到车辆与车位的距离在一预定阈值时，进入漂移阶段。

② 漂移阶段，首先打满方向并轻踩刹车；之后缓回方向，重踩刹车，使车身尽量平行于车位，完成漂移后进入倒车阶段。

③ 倒车阶段，采用分段策略：首先采用 PID 算法，以当前位置与车位中心点所成角度为航向偏差，以距车位中心线的距离为横向偏差进行倒车；待距离进一步拉近后，再对车体的绝对方向值进行比例控制，使之不断向车位的绝对方向靠近；接着调低控制参数并轻踩刹车，同时进行精确微调和初步减速；当速度很低时且误差很小时，对标志位进行置位。

④ 完成置位后，加速转向出库，快速通过终点。

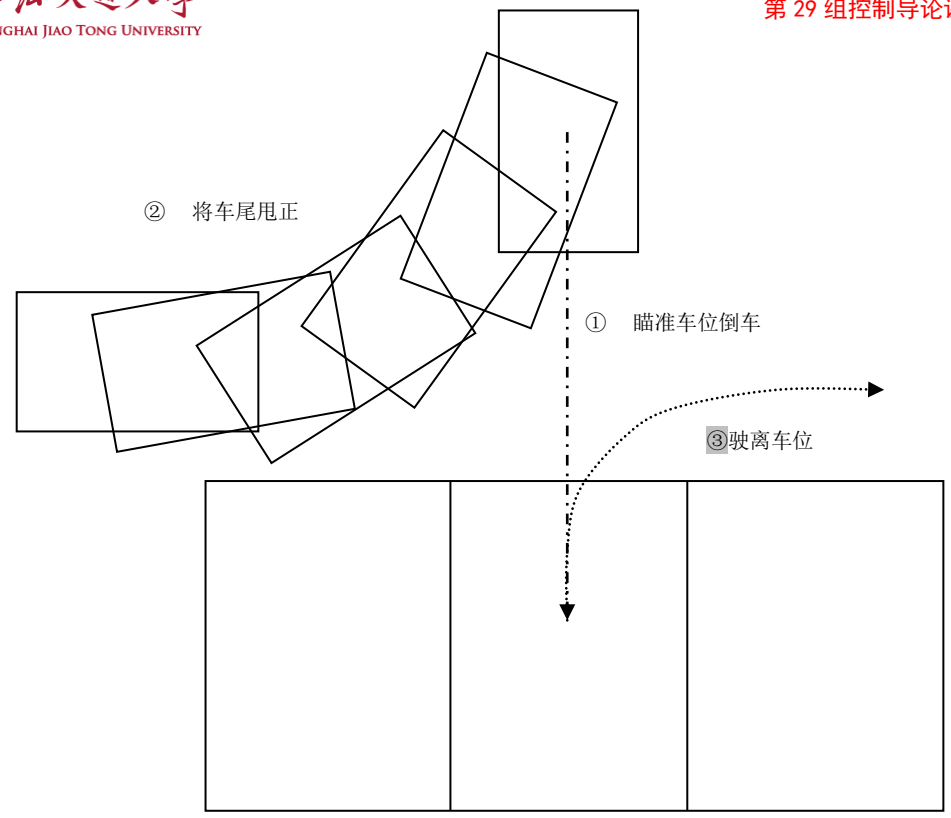


图 6 漂移泊车示意图

在第一次泊车实验中，我们采用了双圆弧法，但我们发现双圆弧法耗时较长，在以时间为基数的计分方式较为不利。所以在第二次泊车比赛中，我们利用了漂移入库法，后续代码的展示以漂移入库法为例。

2. 漂移泊车完整代码：

2.1 巡线行驶阶段

首先保持巡线行驶，在较靠近车位时，靠边行驶。当到达设定阈值时，进入漂移阶段。在这一阶段，我们发现开始漂移的距离与道路曲率有关，因此需要计算前方道路曲率，将开始执行漂移阶段的阈值设定为与道路曲率半径相关的函数。

泊车阶段的巡线代码如下，如需查看小组泊车模式 driver_parking 的完整代码，可前往 https://github.com/C-ZiHao/Introduction-to-Control-Course/blob/master/driver_parking.cpp

```
1. double distance = (_carX - _lotX) * (_carX - _lotX) + (_carY - _lotY) * (_ca
   rY - _lotY);//绝对距离
2. double dis = (cos(_lotAngle) * (_carY - _lotY) - sin(_lotAngle) * (_carX - _
   lotX));//到车位所在直线的距离
3. if (dotflag == 1)
4. {
5.     //根据道路曲率估算恰当的漂移距离
6.     Actionpoint = constrain(12.2, 14.4, 12.3 + (c.r - 90) * 0.032);
7.     //靠边行驶
8.     *cmdSteer = (_yaw-atan2(_midline[10][0] - _width/3, _midline[10][1]));
9.     if (abs(cos(_lotAngle) * (_carY - _lotY) - sin(_lotAngle) * (_carX - _lo
   tX)) < Actionpoint && dotflag < 3)
```

```
    {
        //进入下一阶段
        dotflag = 2;
        cnt = 0;
    }
}
else if (dotflag == 0)
{
    //其它路段按巡线方式行驶
    if (distance < 4000 && dotflag < 2 && successflag == 0)
    {
        dotflag = 1;
    }
    *cmdGear = 1;
    kp_d = 7.6;
    kd_d = 40;
    int k0 = 50;
    D_err = _yaw - atan((k0 * (_midline[0][0])) / double(_speed + 1));
    D_errDiff = D_err - Tmp;
    Tmp = D_err;
    *cmdSteer = constrain(-1.0, 1.0, kp_d * D_err + kd_d * D_errDiff);
}
```

2.2 漂移阶段

漂移阶段，首先右打满方向并轻踩刹车。在行进一定距离后，缓回方向，踩满刹车，达到漂移效果，在满足漂移完成判断条件后，进入下一阶段。

```
if (dotflag == 2)
{
    if (_speed < 0.05 || (abs(_lotAngle - _caryaw) < 0.01) || (abs(cos(_lotAngle) * (_carY - _lotY) - sin(_lotAngle) * (_carX - _lotX)) < 0.001) && dotflag < 3)
    { //漂移完成，进入下一阶段
        dotflag = 3;
        *cmdBrake = 0.05;
        cnt = 0;
    }
    if (err2 < PI / 6)
    { //漂移第二阶段
        *cmdBrake = 0.8;
        *cmdSteer = -0.6;
        *cmdAcc = 0;
    }
    else
```

```
    { //漂移第一阶段
        *cmdSteer = -1;
        *cmdBrake = 0.15;
        *cmdAcc = 0;
    }
}
```

2.3 倒车置位阶段

在使用漂移甩尾后，我们进入倒车置位阶段。在这一阶段（第 3 阶段）中，首先我们首先采用 PID 结合的算法，以当前位置与车位中心点所成角度为航向偏差，以距车位中心线的距离为横向偏差进行倒车，在完成调整后进入第 4 阶段。第 4 阶段和第 5 阶段对车辆速度进行限制，并对车位与车身的角度偏差进行比例控制，在足够接近车位中心后，进入第 6 阶段，在满足置位条件后，进行置位。

```
err = ((_caryaw) - atan2(_carY - _lotY, _carX - _lotX));//航向偏差
if (err > 3.14) err = -6.28 + err;
else if (err < -3.14) err = 6.28 + err;
err = err- dis / 20;  //引入横向偏差

float err2 = (_caryaw - _lotAngle);//细调角度偏差控制变量
if (err2 > 3.14) err2 = -6.28 + err2;
else if (err2 < -3.14) err2 = 6.28 + err2;

if (dotflag == 6) //最后的微调和置位、出车位
{
    if (successflag == 1) //成功出车位后，进入下一阶段
    {
        dotflag = 7;
    }
    else
    { //最后的微调
        *cmdSteer = err2 * 2;
        *cmdAcc = 0;
        *cmdBrake = 0.2;
        *cmdGear = 0;
        if ((abs(distance - dis * dis) < 0.01 * 0.01)||tf == 1) //置位、锁定
        {
            tf = 1;
            if (abs(_speed) < 0.2)//置位
                  { *bFinished = 1; successflag = 1;cnttmp = cnt;}
            else   { *cmdBrake = 1;}
        }
    }
}
```

```
else if (dotflag == 5)
{ //微调
    *cmdBrake = 0.11;
    *cmdGear = -1;
    *cmdAcc = 0;
    *cmdSteer = err2 * 6;  //err2 是与车位与车身的角度偏差
    if (distance - dis * dis < 0.2 * 0.2)
    {
        dotflag = 6;
        cnt = 0;
    }
}
else if (dotflag == 4)
{ //预瞄车位中心进行粗调
    *cmdSteer = 20 * err2; //err2 是与车位与车身的角度偏差
    *cmdGear = -1;
    *cmdAcc = 0.05;
    if (abs(_speed) > 8)   {*cmdBrake = 0.6;}
    if ((distance - dis * dis < 0.4 * 0.4))  {dotflag = 5;}
}
else if (dotflag == 8)
{
    //如果第一次倒车未成功，往前开后重新倒
    *cmdBrake = 0;
    *cmdGear = 1;
    *cmdAcc = 0.5;
    if (_speed < 0)  { *cmdBrake = 1;*cmdAcc = 0;}
    *cmdSteer = -dis / 3;
    if (distance > 8 * 8)  { dotflag = 3;}
}
else if (dotflag == 3)
{
    if (distance - dis * dis < 3.5 * 3.5) //满足距离要求，进入下一阶段
    {
        dotflag = 4;
        cnt = 0;
        if (abs(dis) > 0.6) { dotflag = 8;}
    }
    cnt = cnt + 1;//cnt 是计时器
    *cmdBrake = 0;
    if (_speed > 0){*cmdBrake = 1;}
    *cmdGear = -1; //开始倒车
    *cmdAcc = 0.25;
    kp = 20;  //控制参数
```

```
75.     ki = 10;
76.     if (abs(_speed) > 20) //速度控制
77.     {
78.         *cmdBrake = 0.2;
79.         *cmdAcc = 0.05;
80.     }
81.     if (abs(err) > 0.5) //不同条件下的分段 PID 倒车
82.     {
83.         *cmdSteer = constrain(-1, 1, 2 * kp * err + ki * errdipp);
84.     }
85.     else
86.     {
87.         *cmdSteer = constrain(-1, 1, kp * err + ki * errdipp);
88.     }
89. }
```

2.4 驶离车位阶段

置位后，首先由小到大增加油门量，左打方向，待到车体与道路中心线近乎平行后，加速通过终点，具体代码如下。

```
1.  if (dotflag == 7) //泊车完成后，驶离车位
2.  {
3.      *cmdSteer = _yaw;
4.      *cmdGear = 1;
5.      //逐步加油门，以防失控
6.      *cmdAcc = constrain(0.4, 0.8, (cnt - cnttmp - 30) / 100);
7.      *cmdBrake = 0;
8.      float D;
9.      if (_caryaw * _lotAngle< 0) {D = -abs(_caryaw) - abs(_lotAngle) + 6; }
10.     else  {D = abs(_caryaw - _lotAngle);}
11.     if (D > 1)  //当角度达到设定阈值，不再转向，加速直行
12.     {
13.         *cmdAcc = 1;
14.         *cmdSteer = 0;
15.         *cmdGear = 1;
16.         *cmdBrake = 0;
17.     }
18. }
```

六、实验结果与分析

各车位成绩如下：

车位	成绩
1	10.745
2	11.917
3	10.894
4	11.721
5	11.995
6	12.981
7	16.769
8	14.831
9	10.741
10	11.079
11	15.816

表 4 泊车模式成绩

所有给出的赛道都能完成停车，说明算法的可靠性较好。成绩平均值为 12.68，标准差为 2.156，说明成绩平均值比较好，泛用性在中上水平。

在泊车过程中发现，巡线、倒车、驶离车位的过程均可顺利进行，一般情况下，可以漂移到车位的中心线附近，但是在一些车位上还是出现了漂移位置不准的问题，需要靠后续的倒车环节纠正，这也是泊车误差的主要来源。

本算法对开始漂移位置的确定依赖于经验一次公式，如果可以通过动力学分析，找出漂移距离与道路条件的实际关系，可以进一步提高泊车精度。

七、讨论、心得

泊车实验是高度流程化的实验，为便于代码开发，分段解决问题的思想非常重要。对于这样一个任务，在事前就应当规划好具体流程，并分段进行开发调试。同时，任务一旦分段完成，段与段之间的连接就会变得十分重要。在实验中我们多次发现，由于一些条件语句的要求不合理，有时泊车程序会跳过一些任务段的执行，或卡在某一段中无法继续。经过小组成员的不断努力，反复地输出各类参数寻找特征，最终实现了在不同任务阶段之间的流畅切换。

在开始开发前，我们首先需要对数据接口的特征进行了解。在泊车实验中，对绝对方向角的表示范围是-180 度到 180 度，而不是我们一般熟知的 0 到 360 度。若不明情况就直接上手编程，在一些特殊角度，例如 0 度和 180 度附近，负反馈控制可能失效。这就需要我们在开发程序前，先详细了解各类参数的范围和定义，以免出现 Bug。

当对控制对象的动作无法清晰把握时，可以参考其它小组的 dll 进行借鉴。我们在开发的早期采用了点刹漂移的方法，取得的结果并不是很好。在参考其它小组的 dll 后，我们最终采用了先轻踩油门再重踩的策略，并适当调整参数，得到较佳的甩尾漂移效果。

多分析原理可以少调整参数。我们先使用平面几何的方法进行分析，采用了大量的平面几何计算过程，使用许多表达式代替固定参数。同时，我们发现系统给定坐标并不适于调节，通过计算得到了针对车位的相对坐标，并以这一坐标中的 x-y 距离进行阶段衡量。在计算正确的前提下，可以省去许多对代码的微调操作。

本次泊车，大家采取的策略其实相差并不大。我们进行反思后，认为我们的主要误差来自于漂移阶段的定位与倒车阶段的调节，在这两个阶段细节的处理并不到位，对最终的精度有一定影响。同时，倒车阶段我们使用的速度较慢于第一名的小组，在时间上也存在一定的差距。