

项目编号： IAP4043

上海交通大学

第四期“大学生创新实践计划” 项目研究论文

论文题目： 基于 TORCS 的智能车控制仿真

项目负责人： 黄炫圭 学院（系）： 电院计算机系

指导教师： 杨明 学院（系）： 电院自动化系

参与学生： 周丞 软件工程 郁翔 软件工程

项目执行时间： 2011 年 9 月 至 2012 年 9 月

摘要

智能车研究是当前智能交通系统ITS的热点之一，软件仿真可以节约大量的时间、人力、物力以及财力，对智能车的研究具有重要实际意义。TORCS是一款优秀的开源汽车仿真软件，内部已经集成了各种各样精确和逼真的车辆动力学模型及赛道，同时可以获得仿真环境下的所有车辆真实数据。本项目采用TORCS软件，实现单车巡线、车辆跟驰、自动泊车等自动驾驶、辅助驾驶功能。在此基础上，研究智能车评价指标，综合考虑速度、精度、安全性、节能等因素，制定评判规则，开发智能车仿真比赛裁判系统，定量评估智能车的综合性能。

关键词：智能车, 仿真, TORCS

ABSTRACT

Research on Intelligent Vehicles is one of the hottest topics in Intelligent Transportation Systems (ITS). Using software simulation to study Intelligent Vehicles, we can save a lot of resources, including our time, people resource and money, which has significant meaning to the study of Intelligent Vehicles. TORCS is an excellent open source car race simulator, which has fine vehicle dynamics models and track models built in. We use TORCS, to implement automated driving functionalities such as car cruising, car following and car parking. In addition, we study the gradients to evaluate the performance, considering its speed, accuracy, safety, fuel-consuming and so on, to make evaluation rules and construct a comprehensive Intelligent Vehicles Simulation Race Systems - CyberTORCS.

KEY WORDS: Intelligent Vehicles, simulation, TORCS

目录

第一章 绪论.....	5
1.1 智能车研究背景.....	5
1.1.1 智能车研究的目的与意义 ^[1]	5
1.1.2 发展历史 ^[1]	5
1.1.3 智能车的应用前景.....	5
1.2 软件仿真对于智能车研究的意义.....	6
1.3 智能车控制软件仿真对于自动控制教学的意义.....	6
1.4 论文组织结构.....	6
第二章 TORCS 架构分析.....	8
2.1 TORCS 简介.....	8
2.2 TORCS 平台主程序剖析.....	10
2.2.1 外层程序剖析.....	10
2.2.2 Race Manager 的读取、设置.....	12
2.2.3 游戏引擎自动机剖析.....	13
2.2.4 Race Manager 菜单剖析.....	15
2.2.5 Race Manager 配置文件剖析.....	15
2.2.6 游戏引擎主程序.....	18
2.3 车辆 robot 浅析.....	19
2.3.1 车辆 robot 配置文件浅析.....	19
2.3.2 车辆 robot 程序浅析.....	19
2.4 其他.....	20
第三章 程序界面设计.....	21
3.1 程序主界面改造.....	21
3.2 Race Manager 界面改造.....	21
3.3 对 TORCS 的简化.....	22
第四章 巡线实验设计.....	23
4.1 巡线实验内容设计.....	23
4.2 巡线实验接口设计.....	23
4.3 巡线实验接口实现.....	24
4.3 巡线实验界面改造方案.....	25
4.4 巡线实验评判规则设计.....	25
4.5 巡线实验裁判系统实现.....	25
第五章 车辆跟驰实验设计.....	27
5.1 车辆跟驰实验内容设计.....	27
5.2 车辆跟驰实验平台接口设计.....	27
5.3 车辆跟驰实验平台接口实现.....	27
5.4 车辆跟驰实验界面改造方案.....	27
5.5 车辆跟驰实验评判规则设计.....	28
5.6 车辆跟驰实验裁判系统实现.....	28
第六章 自动泊车实验设计.....	30
6.1 自动泊车实验内容设计.....	30

6.2 自动泊车实验平台接口设计.....	30
6.3 自动泊车实验平台接口实现.....	30
6.4 自动泊车实验界面改造方案.....	31
6.5 自动泊车实验评判规则设计.....	33
6.6 自动泊车实验裁判系统实现.....	33
第七章 总结与展望.....	35
7.1. 结论.....	35
7.1.1 总结.....	35
7.1.2 项目特色与创新点.....	35
7.2 展望.....	36
7.2.1 版本控制.....	36
7.2.2 完善辅助驾驶硬件仿真.....	36
参考文献.....	37
致谢.....	38

第一章 绪论

1.1 智能车研究背景

1.1.1 智能车研究的目的与意义^[1]

随着公路等级的不断提高，特别是高速公路的飞速发展，汽车的行驶速度越来越快，车流量越来越大，汽车碰撞交通事故也越来越多。专家、学者在分析城市交通事故的原因时，普遍认为事故原因主要由：人员素质、运输车辆、道路环境、管理法规等四个方面，而车辆性能的提高即研发高性能的智能汽车是其中很重要的一个环节。美国研究认为，包括智能汽车研究在内的智能运输系统对国家社会经济和交通运输的影响，可能会超过洲际高速公路。它的意义和价值在于：大幅度提高公路的通行能力，至少使现有高速公路的交通量增加1倍；大量减少公路交通堵塞、拥挤，降低汽车油耗，可使城市交通堵塞和拥挤造成的损失分别减少25%~40%左右，大大提高了公路交通的安全性。

1.1.2 发展历史^[1]

进入20世纪90年代以来，随着汽车市场竞争激烈程度的日益加剧和智能运输系统（ITS）研究的兴起，国际上对于智能汽车及其相关技术的研究成为热门，一批有实力、有远见卓识的汽车行业大公司、研究院所和高等院校也正展开智能汽车的研究。同时，美国俄亥俄州立大学和加州大学以及其他一些研究机构正在进行全自动车辆的研制与改进工作。世界各国著名大学也参与到智能汽车的开发中，如麻省理工学院、斯坦福大学、卡耐基—梅隆大学、剑桥大学、东京大学等。他们在人工智能、机器人视觉、自动驾驶和汽车自动导航等领域都有深入的研究。我国的相关研究也已展开。清华大学汽车研究所是国内最早成立的主要从事智能汽车及智能交通的研究单位之一，在汽车导航、主动避撞、车载微机等方面进行了广泛而深入的研究。上海市“智能汽车车内自主导行系统”的一种样车，2000年7月19日通过市科委鉴定，它标志着上海智能交通系统进入实质性实施阶段。国防科大成功试验了第四代无人驾驶汽车，它的最高时速达到了75.6公里，创国内最高纪录。西北工业大学空管所、吉林交通大学、重庆大学等都在展开相关研究。这一新兴学科吸引着越来越多的研究机构、学者加入到智能车相关技术开发研究中来。

1.1.3 智能车的应用前景

智能车系统有着极为广泛的应用前景。结合传感器技术和自动驾驶技术可以实现汽车的自适应巡航并把车开得又快又稳、安全可靠；汽车夜间行驶时，如果装上红外摄像头，就能实现夜晚汽车的安全辅助驾驶；他也可以工作在仓库、码头、工厂或危险、有毒、有害的工作环境中，此外他还能担当起无人值守的巡逻监视、物料的运输、消防灭火等任务。在普通家庭轿车消费中，

智能车的研发也是很有价值的，比如雾天能见度差，人工驾驶经常发生碰撞，如果用上这种设备，激光雷达会自动探测前方的障碍物，电脑会控制车辆自动停下来，撞车就不会发生了。

1.2 软件仿真对于智能车研究的意义

由于智能车的工作原理比较复杂，并且外界的各种干扰，可能导致使用硬件系统难以调试控制算法，而且系统中的硬件都有一定的工作寿命或者工作时间，使用过度或者过频繁可能导致硬件系统的损坏。但是软件方面则不同，软件仿真的时间一般较短，可以很方便地调试控制算法获得最优的参数设置，并且软件仿真使用条件较少，一般的 PC 机上都可以实现，而且没有次数限制，不受外界干扰。软件仿真出来的控制算法在硬件系统上一般能够运行，或者只需要在参数上做一些微小的调整即可。由此可见，软件仿真可以节约大量的时间、人力、物力以及财力，对智能车的研究具有重要意义。

1.3 智能车控制软件仿真对于自动控制教学的意义

智能车是一种典型的自动控制系统，是理想的自动控制教学实验设备。智能车控制实验知识点涉及到自动化控制原理与实践的方方面面，包括但不限于闭环控制、开环控制，系统稳定性，系统抗干扰能力，PID 控制，自适应控制，鲁棒控制。在国外（斯坦福大学）也有将智能车控制应用在教学实践的成功经验，但是其教学实践只限于研究生课堂。

实践证明，本项目所开发的 CyberTORCS 软件仿真平台投放到本科生自动控制实验教学课堂中，取得了非常好的效果。

1.4 论文组织结构

本论文组织结构如下：

第一章为绪论，主要介绍智能车技术的现状、研究方向以及软件仿真的意义。

第二章为 TORCS 程序分析，主要介绍原 TORCS 软件特点及平台，剖析源代码层次及架构，突出可以对其进行改造的地方。

第三章为程序界面设计，主要介绍如何针对 CyberTORCS 要求对 TORCS 程序主界面进行改造，简化、特化其各个结构。

第四章为巡线实验设计，主要介绍巡线实验内容设计，巡线实验平台接口设计、实现，巡线实验界面改造方案，巡线实验评判规则设计、裁判系统实现。

第五章为车辆跟驰实验设计，主要介绍车辆跟驰实验内容设计，车辆跟驰实验平台接口设计、实现，车辆跟驰实验界面改造方案，车辆跟驰实验评判规则设计、裁判系统实现。

第六章为自动泊车实验设计，主要介绍自动泊车实验内容设计，自动泊车实验平台接口设计、实现，自动泊车实验界面改造方案，自动泊车实验评判规则设计、裁判系统实现。

第七章为总结与展望，对整个设计进行总结，并针对 CyberTORCS 软件仿真平台存在的不足

提出改进方案，并对 CyberTORCS 在未来的发展进行展望。

第二章 TORCS 架构分析

2.1 TORCS 简介

TORCS 是一个具有高度可移植性的赛车模拟器。它可作为普通的赛车游戏，同样能做为赛车游戏和人工智能的研究平台。它可运行在 Linux(x86, AMD64, PPC), FreeBSD, Mac OS X 和 Windows 之上。



图 2-1 TORCS 启动界面



图 2-2 TORCS 游戏界面

“TORCS 通常被用来当作普通的赛车游戏，而把 AI racing 当作一个研发的平台。在平台支持的情况下可以使用手柄和转盘来掌舵。其图形有灯光、烟雾、刹车板等，同时也能模拟出物理破坏模型，如撞车、爆胎、车胎的软硬、空气动力学等。从简单的联赛环节到冠军赛都有不同的比

赛类型可供选择。在分屏模式下最多 4 个玩家参与比赛。TORCS 的主要特点如下：（1）它的内部已经集成了各种各样精确和逼真的车辆动力学模型及赛道，十分合理和逼真，一般仿真系统只是基于运动学仿真，不涉及高级的动力学系统，而车辆控制必须考虑动力学，否则没有实际意义。（2）它是一个开源的软件，可以通过 Microsoft Visual Studio 6.0 来进行软件的编译与修改，改成满足需要的模型和仿真环境，它内部的模块化程度比较高，尽管在 C++ 下，但是修改和添加功能是可行的。（3）它可以获得仿真环境下的所有车辆真实数据，这是个突出的优势，因为即使真车也很难实现这一功能，并且安全性难以保证。”^[2]

事实上，最新的 TORCS 已有 Microsoft Visual Studio 2010 的版本，我们用的也正式这个版本进行开发，而不是引文中的 Microsoft Visual Studio 6.0 版本。

TORCS 平台基本架构如下图：

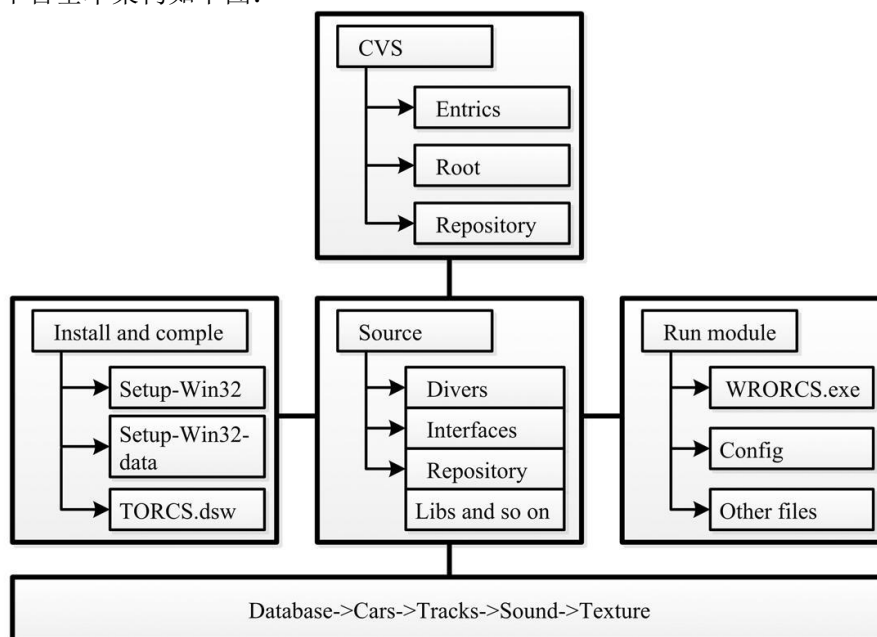


图 2-3 TORCS 平台基本架构图^[2]

2.2 TORCS 平台主程序剖析

2.2.1 外层程序剖析

```
int
main(int argc, char *argv[])
{
    init_args(argc, argv);

    WindowsSpecInit(); /* init specific windows functions */

    GfScrInit(argc, argv); /* init screen */

    TorcsEntry(); /* launch TORCS */

    glutMainLoop(); /* event loop of glut */

    return 0; /* just for the compiler, never reached */
}
```

图 2-4 TORCS 主程序

在点击运行wtorcs.exe后,main()函数先调用WindowsSpecInit()将系统相关的底层函数(例如后面经常会用到的dll挂载、卸载,文件夹读取,文件读取等操作的函数)设置为用windows API编写的对应函数;然后调用GfScrInit()初始化UI屏幕界面,算好屏幕界面参数;接着调用TorcsEntry()启动TORCS程序。

```
void
TorcsEntry(void)
{
    ssgInit();

    GfInitClient();

    TorcsMainMenuInit();

    SplashScreen();

    startMenuMusic();
}
```

图2-5 TorcsEntry() 函数

TorcsEntry()函数的主要部分便是TorcsMainMenuInit()函数,该函数位于mainmenu.cpp中,负责创建主程序主窗口中的标题、按钮等各种UI元素,即我们打开TORCS时看到的窗口。只要修改这个函数里面的元素相关代码,便可将主窗口修改为我们想要的样子。如果不需要音乐的话,可以将startMenuMusic()注释掉。

在完成TorcsEntry()函数的初始化之后,程序返回到main()函数中,运行glut的事件循环,即通过之前在TorcsMainMenuInit()函数中设置的各种UI元素的callback函数进行状态控制。

```
GfuiMenuButtonCreate(menuHandle,
    "Race", "Races Menu",
    ReSinglePlayerInit(menuHandle), GfuiScreenActivate);
```

图2-6 Race按钮代码

对于TorcsMainMenuInit()函数的修改将在下一章谈及,这里我们讨论程序下一步运行。当我们点击Race按钮(参考图)时,根据其设置代码,程序将调用ReSinglePlayerInit()函数,并传入一个包含了现在整个屏幕资源的句柄。该函数将会显示出选择Race种类的屏幕。



图2-7 选择Race种类界面

该函数内容如下:

```
/* Initialize the single player menu */
void *
ReSinglePlayerInit(void *precMenu)
{
    if (singlePlayerHandle) return singlePlayerHandle;

    singlePlayerHandle = GfuiScreenCreateEx((float*)NULL,
        NULL, singlePlayerMenuActivate,
        NULL, (tfuiCallback)NULL,
        1);
}
```

图2-8 ReSinglePlayerInit() 函数

```
GfuiTitleCreate(singlePlayerHandle, "SELECT RACE", 0);

GfuiScreenAddBgImg(singlePlayerHandle, "data/img/splash-single-player.png");

/* Display the raceman button selection */
ReAddRacemanListButton(singlePlayerHandle);

GfuiMenuDefaultKeysAdd(singlePlayerHandle);

ReStateInit(singlePlayerHandle);

GfuiMenuBackQuitButtonCreate(singlePlayerHandle,
    "Back", "Back to Main",
    precMenu, singlePlayerShutdown);

return singlePlayerHandle;
}
```

图 2-9 ReSinglePlayerInit() 函数（续）

其中最重要的便是ReAddRacemanListButton()函数，该函数负责从配置文件中读取各种Race的名字，并生成按钮供君选择。另外我们可以看到ReStateInit()函数，该函数把游戏引擎自动机的当前状态初始化，之后我们会解释这一自动机的作用。

另外，当菜单被激活时，singlePlayerMenuActivate()函数会被调用，该函数会调用ReInit()进行游戏引擎的初始化。ReInit()函数会读取游戏引擎配置文件，挂载并初始化赛道解析程序track.dll和3D图像引擎ssggraph.dll。

2.2.2 Race Manager 的读取、设置

TORCS中将不同种类的Race称为Raceman，为Race Manager的简称，其配置文件放置在\config\raceman下，并通过ReAddRacemanListButton()函数进行读取、设置。该函数位于raceinit.cpp文件中，从名字便可看出这是race的初始化相关程序。

```
/* Load race managers selection menu */
void ReAddRacemanListButton(void *menuHandle)
{
    tFList *racemanList;
    tFList *racemanCur;

    racemanList = GfDirGetListFiltered("config/raceman", "xml");
    if (!racemanList) {
        GfOut("No race manager available\n");
        return;
    }
}
```

图2-10 ReAddRacemanListButton() 函数

```
racemanCur = racemanList;
do {
    reRegisterRaceman(racemanCur);
    racemanCur = racemanCur->next;
} while (racemanCur != racemanList);

reSortRacemanList(&racemanList);

racemanCur = racemanList;
do {
    GfuiMenuButtonCreate(menuHandle,
        racemanCur->dispName,
        GfParmGetStr(racemanCur->userData, RM_SECT_HEADER, RM_ATTR_DESCR, ""),
        racemanCur->userData,
        reSelectRaceman);
    racemanCur = racemanCur->next;
} while (racemanCur != racemanList);

// The list contains at least one element, checked above.
tFList *rl = racemanList;
do {
    tFList *tmp = rl;
    rl = rl->next;
    // Do not free userData and dispName, is in use.
    freez(tmp->name);
    free(tmp);
} while (rl != racemanList);
}
```

图2-11 ReAddRacemanListButton() 函数 (续)

该函数先使用GfDirGetListFiltered()函数读取配置文件文件夹下的所有xml文件,生成一个列表;然后通过reRegisterRaceman()函数对所有的raceman进行注册,即读取其名字及相对应的xml文件句柄;然后通过reSortRacemanList()函数根据raceman配置文件中写明的priority进行排序;再通过一个while循环生成各个Race在Race Select窗口中对应的按钮, callback函数统一设置为reSelectRaceman(),传入callback函数的userData设置为之前读取的文件句柄。

当我们点击Race Select界面的任一个Race时, reSelectRaceman()函数被调用,该函数将会启动一个Race manager,进入Race的配置界面。该函数的实现,是在读取传入文件后,通过ReStateApply()函数将游戏引擎状态机的状态设置为RE_STATE_CONFIG,意为处于配置菜单状态,并转入游戏引擎自动机的控制。

2.2.3 游戏引擎自动机剖析

游戏引擎自动机,位于 racestate.cpp,是整个游戏最重要的部分,它控制着整个游戏接下来的全部状态。如果要在游戏中途改变状态,如退出 Race、显示结果,都需要通过改变该自动机状态并应用,以获得程序一致性,所以之后裁判系统的实现也正利用了这个自动机。

自动机的主程序,是ReStateManage(),它负责根据当前状态,执行相应程序,并做出状态跳转。自动机的默认运行方式,是RM_SYNC | RM_NEXT_STEP,即同步模式(自动机会连续运行)、

会自动跳转到下一步。自动机的各个状态及其功能如下：

状态名	剖析
RE_STATE_CONFIG	处于配置菜单状态，通过ReRacemanMenu()函数显示Race Manager菜单，在该菜单中可以开始Race，配置Players，读取之前的Race。在CyberTORCS中，后两个功能禁用，所以在该函数中将其注释掉。最后会将自动机的模式修改为RM_ASYNC，即异步模式，所以在跳转到RE_STATE_EVENT_INIT状态后会暂时跳出自动机的控制。接下来的情况请参看2.2.4小节。
RE_STATE_EVENT_INIT	游戏初始化，读取配置文件，用已经挂载的track.dll读取赛道配置文件，然后跳入下一状态。
RE_STATE_PRE_RACE	赛前准备，读取配置文件，初始化总圈数或距离，最大损坏值，Race种类：RACE（比赛）、QUALIF（排位赛）、PRACTICE（练习赛）。在CyberTORCS程序中，各个试验平台均是设置为RACE模式，综合测试平台则设为QUALIF模式。然后跳入下一状态。
RE_STATE_RACE_START	准备开始比赛。先调用ReRaceStart()函数，该函数读取配置文件，得到race中的全部车辆driver名单，并按照配置文件中设定的顺序（可以是按照出现的顺序、之前排位赛结果正序、之前排位赛结果倒序）对driver进行重新排序，然后调用reRaceRealStart()函数。该函数首先读取游戏引擎配置文件，挂载并初始化仿真引擎simu.dll；然后调用ReInitCars()，初始化全部车辆的名字等属性，挂载全部车辆的robot程序dll文件，根据车辆分类和车辆模型读取并融合车辆配置文件，调用initStartingGrid()读取配置文件算出车辆发车位置，调用initPits()初始化进站；再确定是否要显示比赛过程；接着，调用全部车辆的robot程序的rbNewRace接口通知其进行比赛前的初始化；先运行几步仿真引擎；如果是排位赛的话，调用ReUpdateQualifCurRes()更新结果；对3D图像引擎中的车辆进行初始化。跳入下一状态。
RE_STATE_RACE	比赛状态。调用ReUpdate()运行游戏，详细情况见2.2.6小节。如果比赛正常结束，跳到RE_STATE_RACE_END状态；被用户按Esc键暂停，跳到RE_STATE_RACE_STOP状态。
RE_STATE_RACE_STOP	显示暂停菜单。
RE_STATE_RACE_END	比赛结束。调用ReRaceEnd()函数，调用raceresults.cpp文件中的ReDisplayResults()函数保存并显示结果。对于CyberTORCS，裁判系统的设计使得结果显示的界面对于各个实验都不同，通过修改ReDisplayResult()调用的函数便可实现。跳到RE_STATE_POST_RACE状态。
RE_STATE_POST_RACE	比赛结束后续事项。调用RePostRace()函数，调用ReUpdateStandings()利用冒泡排序更改当前排名。在综合测试平台中由于需要修改排序依据，需要修改此函数。如果是排位赛，跳到RE_STATE_PRE_RACE继续进行下一车辆的比赛；否则跳到RE_STATE_EVENT_SHUTDOWN。

表2-1 游戏引擎自动机状态表

RE_STATE_EVENT_SHUTDOWN	停止各个引擎的事件，跳到RE_STATE_SHUTDOWN。
RE_STATE_SHUTDOWN RE_STATE_ERROR	比赛结束，回到RE_STATE_CONFIG状态，即回到比赛配置菜单。
RE_STATE_EXIT	游戏出错，直接退出。

表2-2 游戏引擎自动机状态表（续）

2.2.4 Race Manager 菜单剖析

通过ReRacemanMenu()函数显示Race Manager菜单，当我们点击New Race按钮时程序调用ReStartNewRace()函数，该函数先初始化记录比赛结果的结构体，再调用ReStateManage()。

```
GfuiMenuButtonCreate(racemanMenuHdle,
    "New Race", "Start a New Race",
    NULL, ReStartNewRace);

const char* name = GfParmGetStr(params, RM_SECT_HEADER, RM_ATTR_NAME, "CyberCruise");
if (!strcmp(name, "CyberCruise") || !strcmp(name, "CyberOnHand") || !strcmp(name, "CyberParking")) {
    GfuiMenuButtonCreate(racemanMenuHdle,
        "Configure Race", "Configure The Race",
        NULL, reConfigureMenu);
}
```

图2-12 ReRacemanMenu()函数部分代码

当我们点击Configure Race时，会调用reConfigureMenu()函数。在CyberTORCS中，只有巡线实验、自动泊车实验和手动试开模式可以配置比赛，所以可以在此处加入代码限制。

reConfigureMenu()函数会调用reConfigRunState()函数，根据配置文件的设置显示不同的配置选项，分别由Track Select选择赛道、Driver Select选择参与的车辆、Race Configuration配置比赛长度。在CyberTORCS平台中，我们要限制地图的使用，只需要更改配置文件中允许的配置选项即可。另外，在自动泊车实验中我们需要加入泊车位选择菜单，就可以在此处加入相应代码。

2.2.5 Race Manager 配置文件剖析

每一个Race Manager都对应一个配置文件，该文件记录了该种Race的重要信息。下面以我们编写的巡线程序CyberCruise的Race Manager配置文件为例。

```
<params name="CyberCruise">
  <section name="Header">
    <attstr name="name" val="CyberCruise"/>
    <attstr name="description" val="Cruise Experiment with CyberTORCS"/>
    <attnum name="priority" val="100"/>
    <attstr name="menu image" val="data/img/splash-qr.png"/>
    <attstr name="run image" val="data/img/splash-run-practice.png"/>
    <attstr name="start image" val="data/img/splash-dtm.png"/>
  </section>

  <section name="Tracks">
    <attnum name="maximum number" val="1"/>
    <section name="1">
      <attstr name="name" val="g-track-3"/>
      <attstr name="category" val="road"/>
    </section>
  </section>

  <section name="Races">
    <section name="1">
      <attstr name="name" val="CyberCruisePre"/>
    </section>
    <section name="2">
      <attstr name="name" val="CyberCruise"/>
    </section>
  </section>
</params>
```

图 2-13 Race Manager 配置文件示例

Header 节里面的 name、description 即为该种 Race 的名字、描述。priority 则为前面提到的对 raceman 进行排序的依据，越小越前。

Tracks 节里记录的是该种 Race 使用的赛道名、分类。

Racs 节里记录的是该种 Race 使用的内置 Race 类型的数量及各自名字。

```
<section name="CyberCruise">
  <attnum name="distance" unit="km" val="0"/>
  <attstr name="type" val="race"/>
  <attstr name="starting order" val="drivers list"/>
  <attstr name="restart" val="yes"/>
  <attnum name="laps" val="1"/>
  <section name="Starting Grid">
    <attnum name="rows" val="2"/>
    <attnum name="distance to start" val="25"/>
    <attnum name="distance between columns" val="20"/>
    <attnum name="offset within a column" val="10"/>
    <attnum name="initial speed" val="0"/>
    <attnum name="initial height" val="0.2"/>
  </section>
</section>
```

图 2-14 Race Manager 配置文件示例

之后根据对应 Race 的名字，有对其进行配置的节。type 即为其类型，如前述有排位赛、比赛、练习赛。starting order 即为 driver 初始排序方式，可以是按照出现的顺序、之前排位赛结果正序、之前排位赛结果倒序。restart 为是否允许用户重新开始比赛，laps 为圈数。另外，还有 maximum damage 参数，为最大损坏值，在禁止碰撞时可以用到。Starting Grid 小节用于计算发车位置，各项目意义自明。

```
<section name="Drivers">
  <attnum name="maximum number" val="40"/>
  <attstr name="focused module" val="cybercruise"/>
  <attnum name="focused idx" val="0"/>
  <section name="1">
    <attnum name="idx" val="0"/>
    <attstr name="module" val="cybercruise"/>
  </section>
</section>
```

图 2-15 Race Manager 配置文件示例

Drivers 节记录的是参与比赛的车辆 robot.module 即为该 robot 名字,idx 为其车队内编号。

```
<section name="Configuration">
  <attnum name="current configuration" val="2"/>
  <section name="1">
    <attstr name="type" val="track select"/>
  </section>
</section>
```

图 2-15 Race Manager 配置文件示例

Configuration 节记录的是上一小节所描述的允许的配置选项，可以有多个。通过更改这一节内容，便可控制程序中显示的 Configuration Race 菜单的内容。

2.2.6 游戏引擎主程序

ReUpdate() 函数位于 raceengine.cpp 文件，是游戏引擎主程序。该函数主要片段如下：

```
t = GfTimeClock();

START_PROFILE("ReOneStep*");
while (ReInfo->_reRunning && ((t - ReInfo->_reCurTime) > RCM_MAX_DT_SIMU)) {
    ReOneStep(RCM_MAX_DT_SIMU);
}
STOP_PROFILE("ReOneStep*");
```

图2-16 ReUpdate() 函数主要部分

该函数先获取当前时间，然后根据所设定的仿真时间比例关系，调用 ReOneStep() 函数进行游戏。所以，如果要更改游戏仿真时间间隔，更改 RCM_MAX_DT_SIMU 即可。ReOneStep() 故名思义即为进行一步仿真。在该函数一开始的地方有负责显示 Ready, Set, Go 的语句，如果要增加显示的内容的话只需要修改这里。然后是真实时间与仿真时间的计算：

```
ReInfo->_reCurTime += deltaTimeIncrement * ReInfo->_reTimeMult; /* "Real" time */
s->currentTime += deltaTimeIncrement; /* Simulated time */
```

图2-17 ReOneStep() 函数部分代码

所以，如果要修改游戏速度的话，只需要修改 ReInfo->_reTimeMult 相关的内容即可。在 CyberTORCS 中，为了方便调式，我们把快进和慢放的倍比上限都由 4 倍上升到了 16 倍，正是这样实现的。

接下来，ReOneStep() 函数调用各个车辆的 robot 的 rbDrive() 函数接口，得到了各个车辆 robot 的新的控制指令。然后，调用一次仿真引擎的 update 函数，计算新的世界参数。接着，调用 ReManage() 函数更新各个车辆的状态，包括停站、冲线等，记录成绩，在此处还可能在屏幕上显示名次信息，需要屏蔽。最后，调用 ReSortCars() 对车辆进行重新排序。

2.3 车辆 robot 浅析

TORCS 中的车辆都是由一定的 robot 程序定义的，并在运行时由程序动态加载其对应的 dll 文件。

2.3.1 车辆 robot 配置文件浅析

车辆 robot 配置文件为位于\drivers\robot 名字文件夹下的同名 xml 文件，其中记录了车辆的有用信息。

```
<params name="cybercruise" type="robotdef">
  <section name="Robots">
    <section name="index">
      <section name="0">
        <attstr name="name" val="CyberCruise"></attstr>
        <attstr name="desc" val="CyberCruise"></attstr>
        <attstr name="team" val="Cyber"></attstr>
        <attstr name="author" val="Xuanguai Huang"></attstr>
        <attstr name="car name" val="p406"></attstr>
        <attnum name="race number" val="1"></attnum>
        <attnum name="red" val="1.0"></attnum>
        <attnum name="green" val="0.0"></attnum>
        <attnum name="blue" val="1.0"></attnum>
        <attstr name="skill level" val="pro"/>
      </section>
    </section>
  </section>
</params>
```

图 2-18 CyberCruise 车辆 robot 配置文件

其中，index\0 小节所配置的车辆正是对应于 2.2.5 小节所展示的配置文件的 Drivers 节下的定义：cybercruise 车队中对内编号为 0 的车。在 CyberTORCS 中，车辆模型用的都是 p406，所以 car name 的值为 p406。skill level 设为最难的 pro，表示碰撞损坏值计算所用的常数取得最大。

2.3.2 车辆 robot 程序浅析

由于车辆robot是以dll的方式被挂载的，所以模块的进口需要按照tModInfo的定义编写。其中，InitFuncPt()函数即为模块初始化函数，该函数可以返回一个tRobotItf指针作为车辆robot程序接口。tRobotItf结构体包含了如下内容：

接口名	类型	浅析
rbNewTrack	tfRbNewTrack	给robot新的赛道信息
rbNewRace	tfRbNewRace	开始新的比赛。在CyberTORCS中，我们在此加载用户的dll，实现平台。
rbEndRace	tfRbEndRace	当前比赛结束。在CyberTORCS中，我们在此记录部分结果于特定xml文件中，实现裁判系统的功能。
rbDrive	tfRbDrive	车辆驾驶。在CyberTORCS中，我们在此对用户的接口进行调用并进行输入、输出参数转换，实现平台。

表 2-3 tRobotItf 结构体内容浅析

rbPitCmd	tfRbPitCmd	车辆停站
rbShutdown	tfRbShutdown	卸载该dll。在CyberTORCS中，我们在此卸载用户的dll。

表 2-4 tRobotItf 结构体内容浅析（续）

2.4 其他

在项目前期，我们深入的阅读了 TORCS 的源代码，包括但远远不限于上述部分。我们深深感到 TORCS 是一个用 C 编写的模块化程序，结构、层次分明。但是，由于篇幅关系，我们无法将全部的代码进行剖析，只能将 CyberTORCS 涉及到的重要的一部分，或者是有逻辑关联的一部分展示出来。剩下的源代码，诸如仿真引擎代码、图像代码、底层操作系统相关程序代码等，写的非常有参考价值，无法展示实属遗憾。

第三章 程序界面设计

3.1 程序主界面改造

根据2.2.1节，我们修改了TorcsMainMenuInit()函数的代码，屏蔽了部分内容，使其变成CyberTORCS的主界面。

```
GfuiTitleCreate(menuHandle, "CyberTORCS", 0);  
  
GfuiLabelCreate(menuHandle,  
    "SJTU Racing Car Simulator",  
    GFUI_FONT_LARGE,  
    320,  
    420,  
    GFUI_ALIGN_HC_VB,  
    0);
```

图3-1 TorcsMainMenuInit()函数代码部分修改

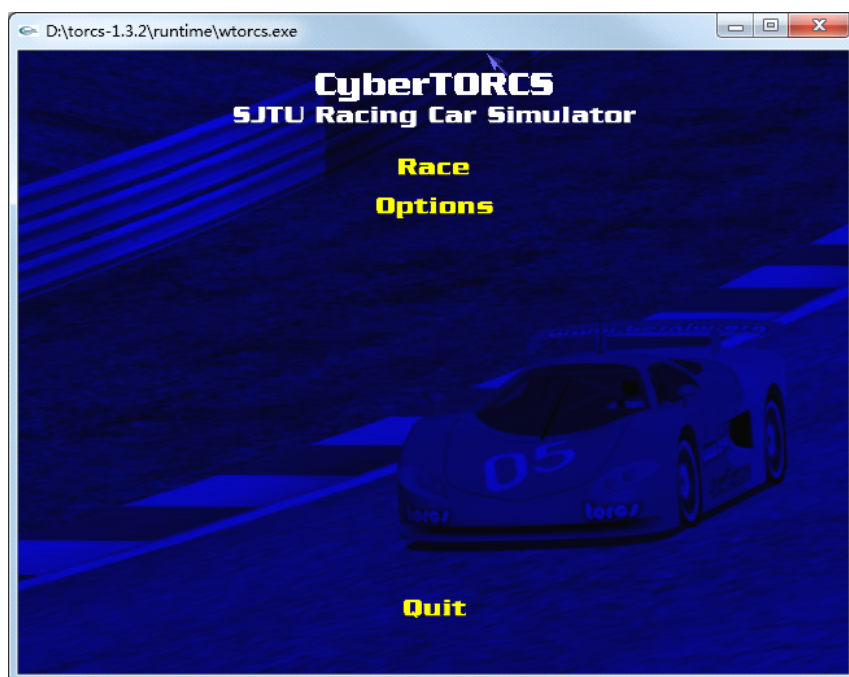


图3-2 修改效果图

3.2 Race Manager 界面改造

根据2.2.2节，只需要配置好config\raceman下面的xml文件，就可以有以下的界面：

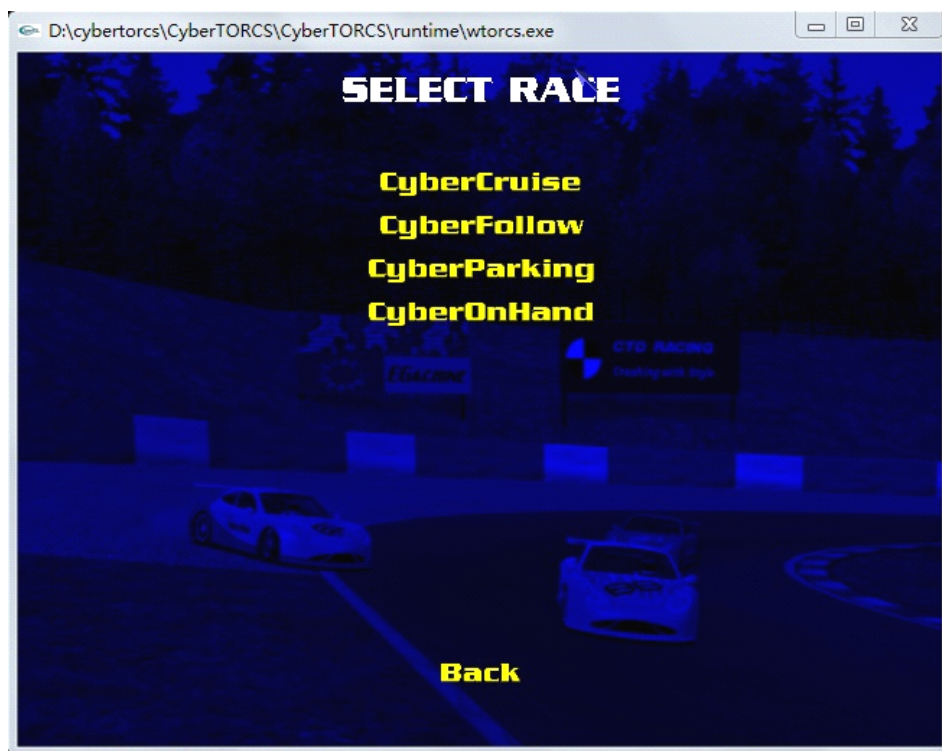


图3-3 CyberTORCS选择Race种类效果图

3.3 对 TORCS 的简化

由于我们限制了能使用的赛道，所以将其他赛道都一并删除。车辆模型全部限定为 p406，其他的车辆种类、车辆模型全部删除。原 TORCS 的安装包有接近 500M，在经过我们的简化之后，CyberTORCS 最终版本也只有 100M 多一点。

第四章 巡线实验设计

4.1 巡线实验内容设计

巡线实验给出 6 条难度不同的赛道, 要求编写一套算法, 使车辆能沿着这 6 条赛道跑完全程, 以 (完成时间+碰撞量/10) 作为评判标准, 速度越快, 碰撞量越小, 算法越完善。

4.2 巡线实验接口设计

输入参数:

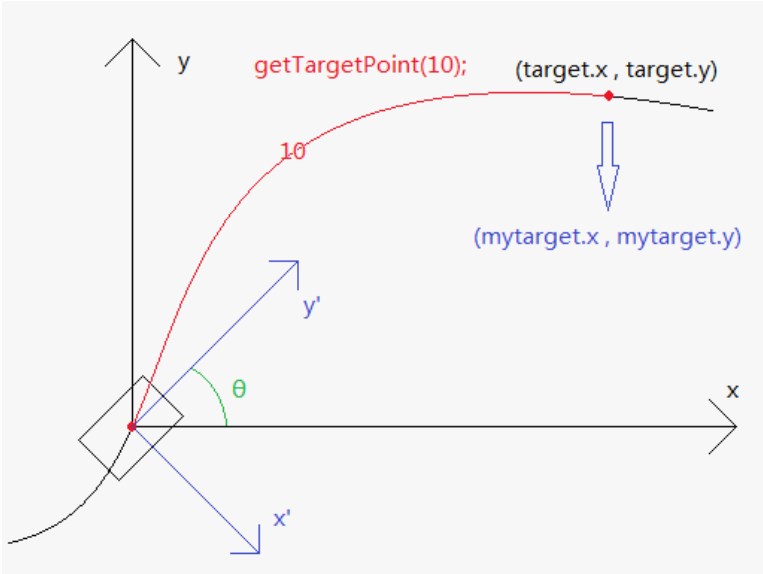
变量名	说明
float midline[k][2]	<p>预瞄点, 沿道路中线 k 米处的相对于当前车辆坐标系的 XY 坐标值, $k < 200$。如 midline[10][0] 表示沿道路中线前方 10 米处的点相对于当前车辆坐标系的 x 坐标, midline[10][1] 表示 y 坐标</p>  <p>图 4-1 预瞄点计算方式示意图</p>
float yaw	偏航角 (弧度)
float yawrate	角速度 (弧度/秒)
float speed	车速 (公里/小时)
float acc	加速度 (米/秒 ²)
float width	道路宽度 (米)
int gearbox	档位 (-1~5)
float rpm	发动机转速 (RPM)

表 4-1 输入参数表

输出参数:

变量名	说明
float* cmdAcc	油门命令, 范围[0.0, 1.0], 0 表示不踩油门
float* cmdBrake	刹车命令, 范围[0.0, 1.0], 0 表示不踩刹车
float* cmdSteer	转向命令, 范围[-1.0, 1.0], -1 表示方向盘向左打死
int* cmdGear	变速箱档位, 范围{r, 1, 2, 3, 4, 5, 6}

表 4-2 输出参数表

4.3 巡线实验接口实现

接口的在 CyberTORCS 中的定义位于 user1.h, 主要内容如下:

```
/* CyberCruise User Interface */
typedef void (*tfudGetParam) (float midline[200][2], float yaw, float yawrate, float speed, float acc, float width, int gearbox, float rpm);
typedef void (*tfudSetParam) (float* cmdAcc, float* cmdBrake, float* cmdSteer, int* cmdGear);

typedef struct {
    tfudGetParam userDriverGetParam;
    tfudSetParam userDriverSetParam;
} tUserItf;
```

图 4-2 巡线实验接口 C 定义文件

仿照 bt 工程, 新建一个 cybercruise 的车辆 robot 工程。如 2.3.1 节所述编写其 xml 配置文件。车辆 robot 工程文件结构如下:

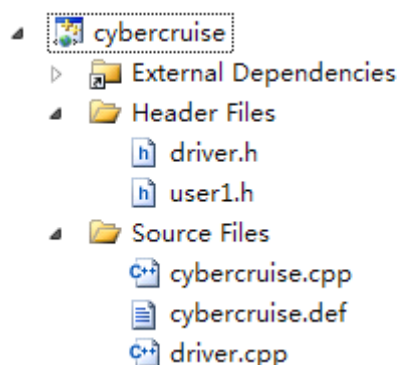


图 4-3 车辆工程文件结构

其中, cybercruise.cpp 即为按照 2.3.2 节所述规则编写的 robot 文件, driver.cpp 提供了 getTargetPoint() 函数。原有的 getTargetPoint() 函数得出的点坐标是绝对坐标, 在 CyberTORCS 中, 我们使用内置的向量运算函数将此转换为按照上图所示的相对坐标。

在 newRace() 函数中, 平台车辆 robot 使用 TORCS 内置底层程序挂载用户 dll 文件; 在 drive() 函数中, 通过以下的代码进行输入参数计算:

```
for (int i = 0; i < 200; ++i) {  
    tmp = getTargetPoint(car, (float)i);  
    midline[i][0] = tmp.x;  
    midline[i][1] = tmp.y;  
}  
yaw=RtTrackSideTgAngleL(&(car->_trkPos))-car->_yaw;  
NORM_PI_PI(yaw);  
yawrate=car->_yaw_rate;  
speed=sqrt(car->_speed_x*car->_speed_x+car->_speed_y*car->_speed_y)*3.6;  
if (car->_speed_x<0) speed=-speed;  
acc=sqrt(car->_accel_x*car->_accel_x+car->_accel_y*car->_accel_y);  
width= car->_trkPos.seg->width;  
gearbox=car->_gear;  
rpm=car->_enginerpm;  
  
userItf->userDriverGetParam(midline, yaw, yawrate, speed, acc, width, gearbox, rpm);  
userItf->userDriverSetParam(&cmdAcc, &cmdBrake, &cmdSteer, &cmdGear);  
  
car->_accelCmd = cmdAcc;  
car->_brakeCmd = cmdBrake;  
car->_steerCmd = cmdSteer;
```

图 4-4 drive() 函数参数计算及调用部分

其中红框所标出部分就是在计算出输入参数后调用已经加载的用户 d11 函数, 得到输出参数, 最后再将输出参数赋值给控制变量。

4.3 巡线实验界面改造方案

如 2.2.5 节所示, 在菜单部分只需要修改 Race Manager 配置文件即可。

另外, 在发车前会依次显示CyberCruise Ready、CyberCruise Set、CyberCruise Go!, 只需在ReOneStep() 函数前加入相应代码即可。

另外, 在界面上方还会显示user spend time, 意为用户d11执行所用时间, 其代码也是在ReOneStep() 函数中, 在调用robot->rbDrive(robot->index, s->cars[i], s)前后进行时间计算即可。

4.4 巡线实验评判规则设计

巡线实验的评判规则为完成时间+碰撞量/10。

4.5 巡线实验裁判系统实现

巡线实验裁判系统所记录的东西都已经是默认记录的东西, 所以只需要更改最后显示结果的界面, 使其能符合我们的要求。

在这里, 重要的是一开始先跑完后得到结果的页面, 是在排位赛模式下, 其对应的显示结果的函数是rmQualifResults() 函数, 我们只需要将其他多余部分删除, 剩下如下的标签:

```
GfuiLabelCreateEx(rmScrHdle, "Driver", fgcolor, GFUI_FONT_MEDIUM_C, x2+10, y, GFUI_ALIGN_HL_VB, 0);  
GfuiLabelCreateEx(rmScrHdle, "Time", fgcolor, GFUI_FONT_MEDIUM_C, x3, y, GFUI_ALIGN_HR_VB, 0);  
GfuiLabelCreateEx(rmScrHdle, "Damage", fgcolor, GFUI_FONT_MEDIUM_C, x4, y, GFUI_ALIGN_HR_VB, 0);
```

图4-5 显示结果页面部分代码

就能得到这样的显示结果的页面：



图4-6 显示结果页面效果图

第五章 车辆跟驰实验设计

5.1 车辆跟驰实验内容设计

跟车实验提供一辆行为随机的领航车，要求编写后车算法，使它能跟随领航车行驶，跟车过程中不能与领航车发生碰撞，也不能超过领航车，以两车的平均间距作为算法的评判标准。

5.2 车辆跟驰实验平台接口设计

输入参数增加下列内容：

float leaderXY[2]	leaderXY[0] 为前车中心在后车坐标系的 x 坐标值 leaderXY[1] 为前车中心在后车坐标系的 y 坐标值
-------------------	--

表 5-1 输入参数增加内容

输出参数不变。

5.3 车辆跟驰实验平台接口实现

接口定义方式与巡线实验类似。在输入参数计算的时候增加前车坐标点转换。

在newRace()函数中，需要根据当前的不同index值加载不同的dll文件：领航车或者用户车，在drive()函数中也需要做好区分。

5.4 车辆跟驰实验界面改造方案

在跟驰实验中，屏幕左上角会显示平均距离和到现在的控制周期数目。



图 5-1 跟驰实验效果图，注意左上角红字

这需要在 drive() 函数中加入相应的计算代码，并且用下面的代码将其显示出来：

```
snprintf(car->ctrl.msg[0], 32, "average distance: %.3f m", average_distance);
snprintf(car->ctrl.msg[1], 32, "run count: %lld", run_count);

memcpy(car->ctrl.msgColor, color, sizeof(car->ctrl.msgColor));
```

图 5-2 显示平均距离的代码

而在车辆跟随实验结束后的结果界面，则需要多方面的修改：首先，在 cyberfollow.cpp 文件的 shutdown() 函数处把 average_distance 记录到特定的 xml 文件中；然后，更改 rmRaceResults() 函数，从该 xml 文件中读取 average_distance，显示失败信息。

5.5 车辆跟驰实验评判规则设计

后车与前车的平均距离越近，成绩越好。但后车不得有任何区域超越领航车。如果违反了规则，即领航车的四个角的任意一角进入如下区域，则会强制停车，并输出错误信息。

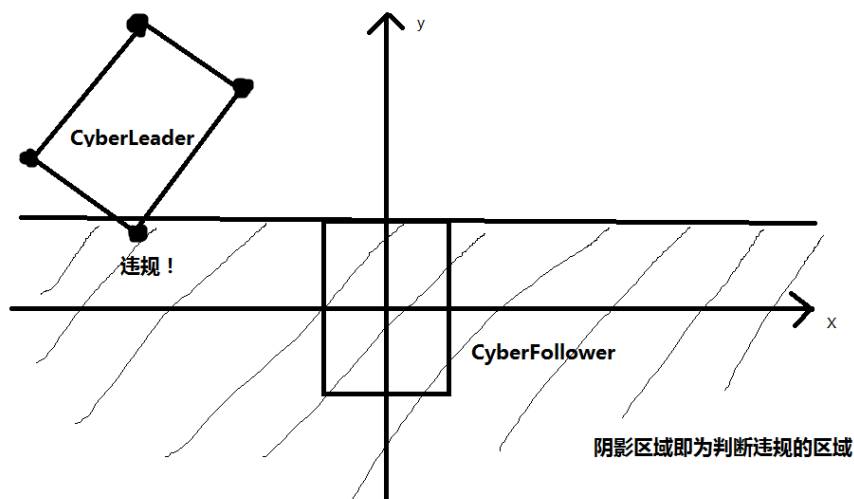


图 5-3 判断违规方式示意图

本实验中允许的最大 damage 值为 1，即不能有任何的碰撞。如果碰撞导致 damage 大于 1，车辆会飞出赛道。

5.6 车辆跟驰实验裁判系统实现

最大损坏值设置只需要在相应的 Race manager 配置文件中写即可。

检测是否违规的部分，放在 cyberfollow.cpp 中输入参数计算部分之前运行。用系统内置的向量转换函数，判断后车是否超越前车车尾。

```
if (index == 0) {  
    // check whether obey the rule  
    tmp = transform(car->_pos_X, car->_pos_Y, car->_yaw, s->cars[0]->_pos_X, s->cars[0]->_pos_Y);  
    if (tmp.len() < 100) {  
        for (int i = 0; i < 4; ++i) {  
            tmp = transform(car->_pos_X, car->_pos_Y, car->_yaw, s->cars[0]->_corner_x(i), s->cars[0]->_corner_y(i));  
            if (tmp.y < car->_dimension_x / 2.0f) {  
                failed = true;  
                break;  
            }  
        }  
    }  
}
```

图 5-4 判断违规代码

如果违规，则需要马上停止比赛，这就需要对游戏引擎自动机的修改。我们在 racestate.cpp 中增加了如下的函数，实现了由车辆 robot 停止比赛的功能。

```
static bool RE_SHUT_DOWN_BY_ROBOT = false;  
void ReRaceShutdownInit() {  
    RE_SHUT_DOWN_BY_ROBOT = false;  
}  
  
void ReShutdownbyRobot() {  
    RE_SHUT_DOWN_BY_ROBOT = true;  
}  
  
void ReCheckState(tRmInfo *ReInfo) {  
    if (RE_SHUT_DOWN_BY_ROBOT)  
        ReInfo->s->_raceState = RM_RACE_ENDED;  
}
```

图 5-5 终止比赛功能实现代码（部分）

这样，只要在发现违规的时候调用 ReShutdownbyRobot()，比赛就会自动结束，到达显示结果页面，显示失败信息。

第六章 自动泊车实验设计

6.1 自动泊车实验内容设计

泊车实验给出 109 个停车位的坐标，要求编写算法，使车辆能够以车头入库、车尾入库两种方式泊入车位，以泊车速度和精度（车辆中心与车位中心的偏差）作为评判标准，速度越快，精度越高，则算法越完善。

6.2 自动泊车实验平台接口设计

输入参数增加了以下内容：

lotX, lotY	车位中心点绝对坐标
lotAngle	车位绝对朝向
carX, carY	当前车辆绝对坐标
carYaw	当前车辆绝对朝向

表 6-1 增加的输入参数

输出参数增加以下内容：

bFinish	泊车结束时置 bFinish=1，告知系统泊车完毕 如果此时速度>0.2km/h，泊车失败
---------	--

表 6-2 增加的输出参数

6.3 自动泊车实验平台接口实现

泊车实验的实现程序是最复杂的，包含了两辆龙套车辆的实现程序，我们只将最重要的部分。首先，在newRace()函数里面，加载用户dll文件；然后，读取泊车实验的xml配置文件，获得停车位、泊车方向的信息。如果是随机停车位，还要获取停车位数目上下界，再进行随机。

```
snprintf(buf, 1024, "%sdrivers/cyberparking/config%s", GetLibDir (), PARAMEXT);
void *handle = GfParmReadFile(buf, GFARM_RMODE_CREAT);
StdPrkNum = (int)GfParmGetNum(handle, RM_PS_PARAM, RM_PS_PRKLOTNUM, NULL, 0);
int MaxPrkNum = (int)GfParmGetNum(handle, RM_PS_PARAM, RM_PS_MAXPRKLOT, NULL, RM_PS_D

const char *parkingDirectionOptionsList[2] = {RM_PS_PRKDRT_VER_FRONT, RM_PS_PRKDRT_VER_BAC
int NbPrkDrt = sizeof (parkingDirectionOptionsList)/ sizeof (parkingDirectionOptionsList[0]);
int CurPrkDrt = 0;
const char *optionName = GfParmGetStr(handle, RM_PS_PARAM, RM_PS_PRKDRT, parkingDirectionO
for (int i = 0; i < NbPrkDrt; i++) {
    if (strcmp(optionName, parkingDirectionOptionsList[i]) == 0) {
        CurPrkDrt = i;
        break;
    }
}
```

图6-1 读取泊车信息代码

之后，根据之前手工检测出的泊车位坐标点位置，查表得出泊车位坐标点。

在drive()函数中，需检测是否已把泊车完成标志位置位。

6.4 自动泊车实验界面改造方案

在自动泊车实验中，需要增加泊车位选择界面，首先需要在 racemanmenu.cpp 中加入如下代码：

```
////////////////////////////////////
// For CyberParking!
////////////////////////////////////
} else if (!strcmp(conf, RM_VAL_PRKLOTSEL)) {
    /* Parking lots select menu */
    ps.nextScreen = reConfigHookInit();
    if (curConf == 1) {
        ps.prevScreen = racemanMenuHdle;
    } else {
        ps.prevScreen = reConfigBackHookInit();
    }
    ps.param = ReInfo->params;
    RmParkingLotSelect(&ps);
    //////////////////////////////////////
```

图 6-2 增加泊车选择选项的代码

意思即为增加泊车位选择这个选项。泊车位选择界面的实现代码位于我们编写的 parkinglotselect.cpp，主要功能是显示相应的选项、最后记录结果的特定的 xml 文件。实现效果如图：

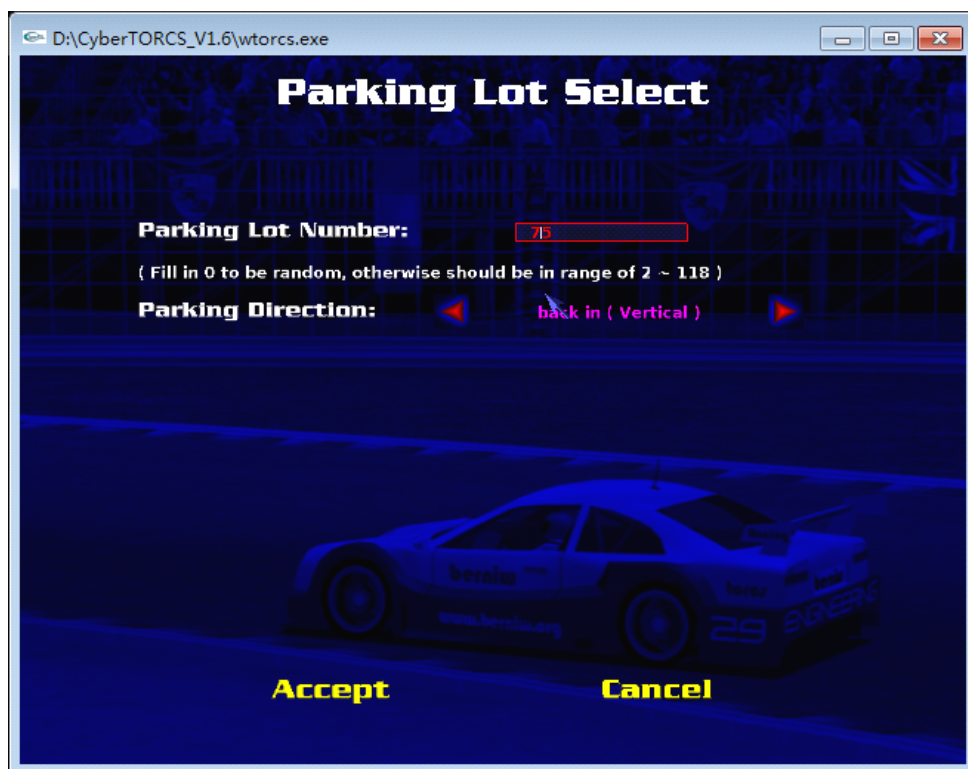


图 6-3 泊车选择页面效果图

在泊车实验进行的过程中，如果已经开始泊车，程序会在左上显示泊车时间，实现上需要检测当前时间并显示：

```
if (startTime != 0) {  
    sprintf(car->ctrl.msg[0], 32, "parking time: %.2f s", s->currentTime - startTime);  
    memcpy(car->ctrl.msgColor, color, sizeof(car->ctrl.msgColor));  
}  
else if (startTime == 0 && car->_laps == 1 && RtGetDistFromStart(car) > PrkDFS[PrkNum] - 20) {  
    startTime = s->currentTime;  
}  
}
```

图 6-4 计算并显示泊车时间的代码

自动泊车显示结果的页面，同时还会显示一些违规信息。违规信息的种类见下一小节。这就需要更改rmRaceResults()函数，当比赛名为CyberParking时运行我们自己的界面显示代码。该代码能从特定的xml文件读取并显示泊车时间、泊车距离、最后总分，当泊车失败时能显示违规信息。


```
} else if (failed_result == -1) {
    snprintf(buf, BUFSIZE, "Failed! Haven't starting parking! ");
    GfuiLabelCreate(rmScrHdle, buf, GFUI_FONT_MEDIUM_C,
        x3, y, GFUI_ALIGN_HC_VB, 0);

} else if (failed_result == -2) {
    snprintf(buf, BUFSIZE, "Failed! Your speed(%3f) is too fast when parking! ", time_result);
    GfuiLabelCreate(rmScrHdle, buf, GFUI_FONT_MEDIUM_C,
        x3, y, GFUI_ALIGN_HC_VB, 0);

} else if (failed_result == -3) {
    snprintf(buf, BUFSIZE, "Failed! Your car yaw (%3f) is not accurate enough! ", time_result);
    GfuiLabelCreate(rmScrHdle, buf, GFUI_FONT_MEDIUM_C,
        x3, y, GFUI_ALIGN_HC_VB, 0);

} else if (failed_result == -4) {
    snprintf(buf, BUFSIZE, "Failed! Your damage(%d) is too large! ",(int)(GfParmGetNum(results, path, RE_ATTR_DAMMAGES, NULL, 0)));
    GfuiLabelCreate(rmScrHdle, buf, GFUI_FONT_MEDIUM_C,
        x3, y, GFUI_ALIGN_HC_VB, 0);
}
```

图6-5 显示违规信息的代码

6.5 自动泊车实验评判规则设计

1. 将车泊入绝对坐标为(lotX, lotY)的车位，前面路段用巡线方法开
2. 成绩计算：泊车时间 * (1+偏差距离/车宽)。

偏差距离：车的中心与车位中心间的距离。

在经过某一固定历程（车位里程 - 15 m）时开始计算泊车时间， 同时显示在左上角。

违规情况：

1. 泊车时不得与旁边两辆车发生碰撞，也不得与栏杆碰撞，当 damage 值大于 1 时会直接退出显示 “Your damage is too large!”，龙套车不受该规则影响。
2. 没有置 bFinished 位比赛就结束或者没有到达规定里程（即开始计时的里程）就置位， 会显示 “Failed! Haven't starting parking!”
3. 置位 bFinished 时车体速度大于 0.2 km/h（注意单位）， 会显示 “Failed! Your speed is too fast when parking!”

最后车体的航向角和提供的 lotAngle 之差超过 10 度（注意单位）， 会显示 “Failed! Your car yaw is not accurate enough!”

6.6 自动泊车实验裁判系统实现

由于有众多的违规情况，计算结果的部分比较复杂：

```
if (bFinished) { // request for parking check
    if (startTime == 0) { // check whether starting counting time
        failed_result = -1;
    } else if (fabs(speed) > 0.2) { // check speed
        failed_result = -2;
        time_result = speed;
    } else {
        float angle1 = PrkAngle;
        NORM_PI_PI(angle1);
        float angle2 = car->_yaw;
        NORM_PI_PI(angle2);
        float angle = angle1 - angle2;
        NORM_PI_PI(angle);
        if (fabs(angle) > PI / 18) { // check for yaw
            failed_result = -3;
            time_result = angle;
        } else { // ok, record result
            failed_result = 0;
            time_result = s->currentTime - startTime;
            distance_result = sqrt((PrkLotX - car->_pos_X) * (PrkLotX - car->_pos_X) + (PrkLotY - car->_pos_Y) * (PrkLotY - car->_pos_Y));
        }
    }

    ReShutdownbyRobot();
    return;
}
```

图 6-6 检测违规并计算结果的代码

主要思想就是检测相应的违规，将违规标志位值为相应的值，然后再shutdown()函数中，将全部的结果都写到特定的dll文件中：

```
// Called before the module is unloaded.
static void shutdown(int index)
{
    GfModUnloadList(&userModList);
    if (index == 0) {
        GfOut("Writing results!\n");
        char buf[1024];
        sprintf(buf, 1024, "%sresults/cyberparking/results.xml", GetLibDir ());

        void *handle = GfParmReadFile(buf, GFARM_RMODE_CREATE);
        GfParmSetNum(handle, "result", "cyberparking/failed", (const char*) NULL, failed_result);
        GfParmSetNum(handle, "result", "cyberparking/time", (const char*) NULL, time_result);
        GfParmSetNum(handle, "result", "cyberparking/distance", (const char*) NULL, distance_result);
        GfParmSetNum(handle, "result", "cyberparking/carlength", (const char*) NULL, carlength);
        GfParmWriteFile(buf, handle, (const char*) NULL);
        GfParmReleaseHandle(handle);
    }
}
```

图6-7 记录结果的代码

第七章 总结与展望

7.1. 结论

7.1.1 总结

智能车研究是当前智能交通系统 ITS 的热点之一，软件仿真可以节约大量的时间、人力、物力以及财力，对智能车的研究具有重要实际意义。TORCS 是一款优秀的开源汽车仿真软件，内部已经集成了各种各样精确和逼真的车辆动力学模型及赛道，同时可以获得仿真环境下的所有车辆真实数据。本项目采用 TORCS 软件，实现单车巡线、车辆跟驰、自动泊车等自动驾驶、辅助驾驶功能。在此基础上，研究智能车评价指标，综合考虑速度、精度、安全性、节能等因素，制定评判规则，开发智能车仿真比赛裁判系统，定量评估智能车的综合性能。

首先，我们详细分析了 TORCS 工程文件，对其内核有了较深入理解。然后，通过基于 TORCS 的软件仿真，我们实现了单车巡线、车辆跟驰、自动泊车等典型自动驾驶功能。单车巡线即编程使车辆能沿着某条赛道自动行驶完全程；车辆跟驰即给出一辆行为随即的领航车，编程实现后车的自动跟踪行驶；自动泊车即给出位于路边的某一车位，编程实现车辆的自动泊入。

我们研究了各个驾驶功能中的重要指标，提出了一套评判规则。接着，通过对 TORCS 的再次开发，设计了开放接口，形成了完备的智能车仿真比赛裁判系统和综合测试平台。最后，我们将其投放在本科生实验教学环境中，取得了很好的效果。

7.1.2 项目特色与创新点

项目特色：

- 1、使用开源的 TORCS 进行再次开发，起点高
- 2、评判规则的设定综合考虑了各方面的因素
- 3、形成一个完整的留有开放接口的智能车仿真比赛裁判系统和一个综合测试演示平台，能为单车巡线、车辆跟驰、自动泊车进行打分评判。

项目创新点：

本项目成功地将汽车仿真软件与自动控制理论相结合，开发出了适合教学的 CyberTORCS 智能车仿真平台，降低了对实物系统的依赖，解决了传统仿真实验中动力学模型不完善的不足；同时创造性的开发了车辆跟驰、自动泊车平台及评判系统，使 CyberTORCS 能仿真更多的汽车自动驾驶、辅助驾驶功能。课堂实验结果表明，该平台使本科生真正理解了自动控制中一些抽象的知识概念，有助于进一步加强学生实践能力和培养创新能力。

7.2 展望

7.2.1 版本控制

TORCS 软件更新速度较快，我们当时基于最新的 TORCS1.3.2 版本开发了 CyberTORCS。但是到开发基本完成时，TORCS 原软件已经更新到了 1.3.4 版本，修正了部分 bug，增加了很多新的赛车和赛道。虽然其主要的更新就是增加赛车和赛道，物理引擎、软件架构等基本上没有修改，但是我们仍然期望将来的 CyberTORCS 平台能加入版本控制功能，能及时的和最新的 TORCS 进行版本控制上的 merge 合并操作，从而进行更新。

7.2.2 完善辅助驾驶硬件仿真



图 7-1 辅助驾驶硬件仿真效果示意图

最后，我们在辅助驾驶硬件仿真做的仍不足。我们期望能够借助游戏方向盘，探索辅助驾驶相关实验，实现半实物仿真。

参考文献

- [1] 胡海峰, 史忠科, 《智能汽车发展研究》, 《计算机应用研究》, 2004
- [2] 何 宁, 赵治国, 朱 阳, 《基于 TORCS 平台的虚拟车辆仿真系统开发》, 《中国制造业信息化》, 2010

致谢

感谢杨明老师，让立项人在大一就能接触智能车这么有趣的事物。

感谢杨辰兮学长对 CyberTORCS 程序的编写和报告的纂写的鼎力相助，没有您的努力，CyberTORCS 不可能顺利开发完成。

感谢顾霄琳学姐对于课堂实验情况的及时反馈，之前测试版本多有不足，因此而给您增加工作量，在此我们深感抱歉。

感谢参与智能车控制算法课的全部同学，正是你们的努力让我们感到我们的努力有价值，也让我们更有动力完善这个系统。