机器人足球赛程序设计实验指导书

实验 1 机器人足球比赛编程预备知识

1、实验目的

掌握 RoboCup 仿真机器人足球比赛相关知识点，具体内容如下：

（1）Linux 操作系统的熟悉及了解其基本操作。

（2）掌握 Linux 下如何进行 C++编程，了解 gcc 编译器以及一些简单编辑工具，如：vi、emacs、gedit、Anjuta、Kdevelope 等。

（3）启动 RoboCup 仿真（2D）足球队的比赛。

2、实验设备

硬件环境：PC 机

软件环境：操作系统 Linux

3、实验内容

（1）掌握 Linux 一些常用的命令

- 联机帮助

格式： man [命令名]

例如： man ls

屏幕上就会显示所有 ls 的用法。

- 文件或目录处理

格式：ls [-atFlgR][name]

第一项是一些语法加量。第二项是文件名。

常用的方法有：

ls 列出当前目录下的所有文件。

ls –a 列出包括以 . 开始的隐藏文件的所有文件名。

ls –t 依照文件最后修改时间的顺序列出文件名。

ls –F 列出当前目录下的文件名及其类型。以/结尾表示为目录名、以*结尾表示不可执行文件、以@结尾表示为符号连接。

ls –l 列出目录下所有文件的权限、所有者、文件大小、修改时间及名称。

ls –lg 同上，并显示出文件的所有者工作组名。

ls –R 显示出目录下以及其所有子目录的文件名。

- 改变工作目录

格式： cd [name]

name ：目录名、路径或目录缩写。

常用的方法有：

cd 改变目录位置至用户登录时的工作目录。

cd dirl 改变目录位置至 dirl 目录下。

cd ~user 改变目录位置至用户的工作目录。

cd .. 改变目录位置至。

cd ../user 改变目录位置至相对路径 user 的目录下。

cd /../.. 改变目录位置至绝对路径的目录位置下。

- 复制文件

格式：cp [-r] 源地址 目的地址

常用的方法有：

cp file1 file2 将文件 file1 复制成 file2

cp file1 dir1 将文件 file1 复制到目录 dir1 下，文件名仍为 file1。

cp /tmp/file1 ./ 将目录/tmp 下的文件 file1 复制到当前目录下，文件名仍为 file1。

cp /tmp/file1 file2 将目录/tmp 下的文件 file1 复制到当前目录下，文件名改为 file2。

cp -r dir1 dir2 复制整个目录。

- 移动或更改文件、目录名称

格式：mv 源地址 目的地址

常用的方法有：

mv file1 file2 将文件 file1 更名为 file2。

mv file1 dir1 将文件 file1 转移到目录 dir1 下，文件名仍为 file1。

mv dir1 dir2 将目录 dir1 更改为目录 dir2。

- 建立新目录

格式：mkdir 目录名

- 删除目录

格式：rmdir [目录名|文件名]

常用的方法有：

rm –r dir1 删除目录 dir1 及其子目录下的所有文件。

- 列出当前所在的目录位置

格式：pwd

- 查看文件内容

格式： cat 文件名

- 文件权限的设定

格式：chmod [-R] mode name

name：文件名或目录名。

mode：3 个或 8 个数字或 rwx 的组合。r-read(读权限)、w-write(写权限)、x-execute(执行)

常用的方法有：

chmod 777 file1 给所有用户 file1 全部的权限。

- 文件的打包和解压缩

格式：tar [option] [file] gzip[option] [file]

tar 主要来进行打包而不压缩，而 gzip 则进行压缩。

option 主要有：

-c — 创建一个新文档。

-f — 当与 -c 选项一起使用时，创建的 tar 文件使用该选项指定的文件名；当与-x 选项一起使用时，则解除该选项指定的归档。

-t — 显示包括在 tar 文件中的文件列表。

-v — 显示文件的文档进度。

-x — 从文档中抽取文件。

-z — 使用 gzip 来压缩 tar 文件。

如使用：tar -zcvf filename.tar /home/mine/work /home/mine/school

上面的命令把 /home/mine 目录下的 work 和 school 子目录内的所有文件都放入当前目录中一个叫做 filename.tar 的新文件里。

完成以下操作：

- 如何找到用户主目录的绝对路径名？在自己的系统上，用户主目录的绝对路径名是什么？
- 将当前工作目录从/home/UVA 转到/home/Tsinghua 需要使用什么命令？
- 如何显示当前目录？
- 如何在当前目录下建立子目录 RoboCup？

- 如何删除子目录 RoboCup？
- 如何查看当前目录下的内容？
- 如何将文件 start.sh 的权限设定为：start.sh 属于可读、可写、可执行?
- 如何将当前目录包括所有子目录全部做备份文件，备份文件名为 first.tar？
- 如何将目录/home 下每一个文件压缩成.gz 文件？
- 如何把上例中每个压缩的文件解压，并列出详细的信息？

（2）掌握 emacs、vi、gedit 等编辑工具以及 gcc、anjuta、Kdevelope 等编译工具的使用

emacs、vi、gedit 等工具是类似 Windows 操作系统下的 txt 文本、Word 等编辑器工具，完成对文本的编辑工作。以 emacs 为例介绍一下这类工具的使用情况：

emacs 启动:

直接打开 emacs, 如果有 X-windows 就会开视窗. 如果不想用 X-windows 的版本,就用 emacs -nw (No windows)起动.

注：以下 **C 表示 Ctrl 健，M 表示 Alt 健**。

上下移动 C-p 向上 (previous line) C-n 向下(next line)

左右移动 C-f 向右 (forward) C-b 向左 (backward)

翻页 下一页 C-v (view next screen) 上一页 M-v

移到行头 C-a **行尾** C-e **句首** M-a **句尾** M-e

删除游标目前指的/后面的字 C-d 前面的字 DEL (Delete 键)

M-DEL 往回删一个字(word)

M-d 往前删 (游标后面)

C-k 删至行尾 (kill)

M-k 删到一句子结尾(删到句点) (kill)

结束 emacs 使用：C-x C-c

gcc、anjuta 等是 Linux 下的源程序编译工具，类似 Tubro C/Turbo C++、C++builder、VC++等编译工具，以 gcc 为例介绍一下情况：

看下面的例子：test.c

```
#include<stdio.h>
main()
{ char *str="I like Linux! I advices you jion in the Linux World";
  printf("%s\n",str);
 exit(0);
}
```

使用 gcc 编译。输入 gcc -c test.c 得到目标文件，test.o.-c 命令表示对文件进行编译和汇编，但并不连接。如果再键入 gcc -o ../bin/test test.o,那么将得到名为 test 的可执行文件。其实这两个不可以一气呵成。输入 gcc ../bin/test test.c.如果程序没有错误就生成了可执行文件。也许你会觉得基于命令行的编译器比不上如 VC 之类的集成开发环境，的确 gcc 的界面要改进，但是你一旦熟练了就会感到，gcc 的效率如此之高。可以告诉大家的是 Linux 底下的 C/C++集成开发环境 Kdevelope 和 Vc 一样强大，使用了 Gcc 编译器。

gcc 的一般语法是：gcc [option | filename]...

g++ [option | filename]...

其中 option 为 gcc 使用时的选项，而 filename 为欲以 gcc 处理的文件

说明：

这 C 与 C++ 的 compiler 已将产生新程序的相关程序整合起来。产生一个新的程序需要经过四个阶段：预处理、编译、汇编、连结，而这两个编译器都能将输入的文件做不同阶段的处理。虽然原始程序的扩展名可用来分辨编写原始程序码所用的语言，但不同的 compiler，其预设的处理程序却各不相同：

gcc 预设经由预处理过(扩展名为.i)的文件为 C 语言，并于程式连结阶段以 C 的连结方式处理。

g++ 预设经由预处理过(扩展名为.i)的文件为 C++ 语言，并于程序连结阶段以 C++ 的连结方式处理。

源程序的扩展名指出所用编写程序所用的语言，以及相对应的处理方法：

- .c C 原始程序 预处理、编译、汇编
- .C C++ 原始程序 预处理、编译、汇编
- .cc C++ 原始程序 预处理、编译、汇编
- .cxx C++ 原始程序 预处理、编译、汇编
- .m Objective-C 原始程序 预处理、编译、汇编
- .i 已经过预处理之 C 原始程序 编译、汇编
- .ii 已经过预处理之 C++ 原始程序 编译、汇编
- .s 组合语言原始程序 汇编
- .S 组合语言原始程序 预处理、汇编
- .h 预处理文件(标头文件) (不常出现在指令行)
- 其他扩展名的文件是由连结程序来处理，通常有：
- .o Object file
- .a Archive file

除非编译过程出现错误，否则 "连结" 一定是产生一个新程序的最后阶段。然而你也可以通过 -c、-s 或 -E 等选项，将四个阶段中的其中一个停止。在连结阶段，所有与源码相对

应的 .o 文件、程序库、和其他无法自文件名辨明属性的文件(包括不以 .o 为扩展名的 object file 以及扩展名为 .a 的 archive file)都会交由连结程序来处理(在指令行将那些文件当作连结程序的参数传给连结程序)。

如果一个工程由几个源文件构成，就需要编写 Makefile 文件，makefile 的编写见讲义附录。

完成下面功能：

- 使用上述其中之一编辑环境，编写文件 counter.h、counter.cpp、appl.cpp:

//File counter.h

```
class counter
{
  private:
      unsigned int value;   //   私有数据成员
  public:
      counter(void);
      void increment(void);
      void decrement(void);
      unsigned int getvalue(void);
  };
```

//File counter.cpp

```
#include "counter.h"
counter::counter(void) {value=0;}     // 构造函数(Constructor)
void counter::increment(void) { if (value<9999) value++;}
void counter::decrement(void) { if (value>0) value--;}
unsigned int counter::getvalue(void) { return value;}
```

//File appl.cpp

```
#include <iostream.h>
#include "counter.h"
int main()
{   counter c1,c2;
     for(int i=1; i<5; i++)
     {   c1.increment();
          cout<<"c1="<<c1.getvalue()<<"\n";
```

```
        c2.increment();
    }
    c2.decrement();
    cout<<"The final value of c1="<<c1.getvalue()<<"\n";
    cout<<"The final value of c2="<<c2.getvalue()<<"\n";
    return 0;
}
```

- **使用 g++或 gcc 实现生成可执行程序 appl.exe。**

（3）启动球队上场比赛

完成以下操作：

- 首先完成 Server 的安装：
  - 解压 tar zvf sserver-*.tar.gz
  - 编译./configure
  - 生成可执行文件make
- 同时启动Server和Monitor/分别启动Server和Monitor

  第一种方式：

  rcsoccersim

  第二种方式：

  /cd rcssserver ./rcssserver

  cd rcssmonitor ./rcssmonitor
- 启动比赛的2支球队，详细见每个球队的readme。
- 只启动部分球员
- 有兴趣的同学可以熟悉一下shell脚本文件的语法及脚本编写。

实验 2 Demeer5（kick 和 dash）

1、试验目的：

（1）了解 Demeer5 的工作原理
（2）会对 Demeer5 进行简单的修改

2、试验设备

硬件环境：PC
软件环境：Linux

3、试验内容

（1）Demeer5 的工作原理：

Demeer5 函数是整个球队的核心，它最终返回一个可以执行的动作，底层的模块负责将此动作发送给 Server，然后由 Server 执行。可以说，Demeer5 就是我们想法的体现，是一支球队的大脑。在 Demeer5 中有一系列的判断来决定每个周期的动作。下面对 Demeer5 进行简要的分析。

Demeer5()是一个决策函数，在策略上使用的是下面这个简单的策略：

如果球可踢，则用最大力量踢球；

如果球不可踢且我是队友中最快到达球的队员，则去截球；

其他情况按战略点跑位。

我们现在只要看一下球可踢时的代码：

```
else if( WM->isBallKickable())                    // 如果球可踢
{
    //确定踢向的点
    VecPosition posGoal( PITCH_LENGTH/2.0,
            (-1+2*(WM->getCurrentCycle()%2)) * 0.4 * SS->getGoalWidth() );
    //调用踢球的动作
    soc = kickTo( posGoal, SS->getBallSpeedMax() ); // kick maximal
     //将动作放入命令队列中
    ACT->putCommandInQueue( soc );
    //将脖子转向球
    ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
    //记录调试信息
    Log.log( 100, "kick ball" );
}
```

这一小段函数决定了当球在球员智能体的可踢范围之内时应当做的动作，这里是一个简单的把球向前踢而不考虑任何其他情况的方法。该程序段的条理是很清晰的。

（2）对 Demeer5 进行简单的修改：

现在我们对 Demeer5 进行简单的修改，让它在球可踢的时候进行带球的动作。带球就是 kick 和 dash 动作序列的结合。带球的函数在 BasicPlayer 中，函数为 dribble（）. 它接收两个参数，第一个参数为带球的方向，第二个参数为带球的类型。

带球类型解释如下：

DRIBBLE_FAST：快速带球；

DRIBBLE_SLOW：慢速带球；

DRIBBLE_WITHBALL：安全带球；

所以，对 dribble 的一种调用形式为：dribble（ang，DRIBBLE_FAST ）

其中 ang 为 AngDeg（是一个 double）类型

该函数的返回值是一个 SoccerCommand 类型。

知道了如何调用 dribble，我们来对 Demeer5 进行修改：

```
else if( WM->isBallKickable())
{
    AngDeg ang = 0.0;
    soc = dribble（ang，DRIBBLE_FAST ）; // 进行带球
```

```
        //将动作放入命令队列中
        ACT->putCommandInQueue( soc );
        //将脖子转向球
        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
        //记录调试信息
        Log.log( 100, "kick ball" );
}
```

这样，在球可踢的时候，球员智能体将把球带向前方，这里取的是 0 度角，即沿 x 轴一直向前快速带球。

我们再次对 Demeer5 函数进行修改，这次是让球员 Agent 将球踢向各个不同的地方。 这将调用 kickTo（）函数来完成。下面简要说明一下 kickTo（）函数的使用方法：这个函数有两个参数，第一个参数是目标点的坐标，第二个参数是球到达目标点时的速度，返回一个踢球的动作。可以使用下面的形式来调用：

```
VecPosition pos( x, y );
double speed = 1.0;
kickTo( pos , speed );
```

kickTo()函数在其内部将会决定踢球时所用力量的大小，并且会判断是否能够将球踢到该点，并作出相应的调整。比如，将球踢向（ 0， 0）点, 1.0 的末速度：

```
else if( WM->isBallKickable())
{
        soc = kickTo( VecPosition( 0, 0 ),1.0);
        //将动作放入命令队列中
        ACT->putCommandInQueue( soc );
        //将脖子转向球
        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
        //记录调试信息
        Log.log( 100, "kick ball" );
}
```

- **根据上述操作，完成以下踢球操作：**
  - 将球踢向对方的球门。
  - 将球踢向距离自己最近的队友。
  - 尝试不同的踢球点。

以下是一些可能用到的函数：

1．得到对方球门的坐标

VecPosition getPosOpponentGoal();

调用：WM-> getPosOpponentGoal ();

2．得到距离自己最近的队员

ObjectT getClosestInSetTo( );

调用：WM->getClosestInSetTo(OBJECT_SET_TEAMMATES，posAgent)

3 得到一个对象的坐标

VecPosition getGlobalPosition(ObjectT o)

调用：WM->getGlobalPosition(o);

更多的函数请查找教材，或者查看源程序的 WorldModel.cpp, WorldModelHighLevel.cpp,

WorldModelPredict.cpp 等文件。

- **根据上述内容，完成以下带球的操作：**
  - 用不同的带球模式进行带球，并观察效果，比较异同。
  - 将球向对方的球门方向带。
  - 尝试不同的带球组合。

- **根据上述内容，完成以下综合练习：**

带球与踢球的结合：

- 让球员智能体一直向对方球门的方向带球，在进入对方禁区后以最大力量踢向球门(末速度最大为 2.7)。涉及到的具体函数请查看教材。

实验 3　　Demeer5 的基本动作

1.实验目的

熟悉demeer5并学会demeer5的基本使用方法，具体内容如下：

（1）能理解UVA程序中原来的demeer5中的内容

（2）能通过修改demeer5中的具体函数内容实现对场上球员的控制

（3）能通过底层动作的简单组合控制场上队员做出一些复杂动作

2、实验设备

硬件环境：PC机

软件环境：操作系统Linux

3.实验内容及步骤

（1）在球队程序中找到player.c并打开（03版本的程序，打开playerTeams.c）；

（2）在player.c中找到demmer5函数;

（3）阅读此段程序，并结合Monitor观察球员的具体行为（你将发现可以踢到球的球员会将球朝球门的方向踢去，而不能踢到球的队员中如果是离球最近的队员就去截球，否者则按阵型跑位）；

（4）修改demmer5函数改变队员的行为具体步骤如下：

① 在demeer5函数中找到

```
if( WM->isBallKickable())
                                    // isBallKickable()函数用来判断球是否可踢
{
  VecPosition posGoal( PITCH_LENGTH/2.0,
          (-1 + 2*(WM->getCurrentCycle()%2)) * 0.4 * SS->getGoalWidth() );
       //设定射门位置坐标
  soc = kickTo( posGoal, SS->getBallSpeedMax() ); //  朝球门方向将球以最大力度踢出
  ACT->putCommandInQueue( soc ); //只有把命令放入命令队列动作才会执行
  ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
```

//做动作的同时改变脖子的方向

}

② 『控球』将此函数修改为

```
if( WM->isBallKickable())
{
  soc = kickBallCloseToBody(45);
  ACT->putCommandInQueue( soc );
  ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
}
```

然后编译运行程序，观察球员的行为我们会发现当球可踢时，球员不再朝着球门的方向踢了，而是将球绕自己身体转动（UVA的这个底层动作经常把球转丢!）

③ 『带球』将此函数修改为

```
if( WM->isBallKickable())
{
  soc = dribble(0.0,DRIBBLE_SLOW);//其中dribble函数中第一个参数表示带球的方向可以是-180~180之间，不一定是0.0
  ACT->putCommandInQueue( soc );
  ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
}
```

然后编译运行程序，观察球员的行为我们会发现当球可踢时，球员不再朝着球门的方向踢了，而是朝我们指定的方向执行带球

④ 『传球』将此函数修改为

```
if( WM->isBallKickable())
{
  soc = leadingPass(OBJECT_TEAMMATE_9，1);
  //其中leadingPass中第一个参数表示传球的对象,本实验中我们将球直接传给指定号码（1~11）的球员,不一定是OBJECT_TEAMMATE_9

  ACT->putCommandInQueue( soc );
  ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
}
```

然后编译运行程序，观察球员的行为我们会发现当球可踢时，球员不再朝着球门的方向踢了，而是将球传给我们指定号码的队员

⑤ 『配合』将此函数修改为

```
if( WM->isBallKickable())
{
```

```
  if(WM->getAgentObjectType()==OBJECT_TEAMMATE_9)
    soc = dribble(0.0,DRIBBLE_SLOW);        //带球
    else
    soc = leadingPass(OBJECT_TEAMMATE_9，1); //传球
    ACT->putCommandInQueue( soc );
    ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );
}
```

编译程序，观察球员行为，我们会发现，当9号队员得到球后会朝前方带球，其他队员得到球后会将球传给9号（不管9号是不是越位）.

根据以上描述完成练习：通过基本动作的组合实现球员的以下行为（1，2题希望大家仔细思考一下，第3题有兴趣的同学可以思考一下，更多基本动作请查阅BasicPlayer.C函数,或其他有关书籍,也希望大家能做出更多的动作）

- 如果在对方禁区内就射门，否则，如果是7，8，9号队员就朝前带球，其他队员将球传给9号（用WM->isInTheirPenaltyArea(WM->getBallPos())来判断球是否在对方禁区）
- 如果队员的位置在自己半场就将球朝对方球门踢去，否者就朝前方带球（用WM->getBallPos().getX()来得到球的x坐标）
- 当有人来抢球时（离自己很近），就将球传给离自己最近的队友，否则就自己带球（调用WM->getClosestRelativeInSet函数来得到离自己最近的己方或对方球员，通过pos1.getDistanceTo(pos2)来得到两位置之间的距离）。

实验 4　　复杂的动作决策

1.实验目的

进一步了解demeer5并能熟练的修改demeer5的内容以达到对场上球员的控制：

（1）能理解UVA程序中原来的demeer5中的全部内容

（2）能通过修改demeer5中的具体函数内容实现对场上球员的控制

（3）能通过底层动作的简单组合控制场上队员做出一些复杂动作决策

（4）对WorldModel有初步的认识，学会在WorldModel,basicplayer里添加新函数

2、实验设备

硬件环境：PC机
软件环境：操作系统Linux

3.实验内容及步骤

（1）在球队程序中找到player.c并打开;

（2）在player.c中找到demmer5函数;

（3）阅读此段程序，并结合monitor观察球员的具体行为（你将发现可以踢到球的球员会

将球朝球门的方向踢去，而不能踢到球的队员中如果是离球最近的队员就去截球，否则就按阵型跑位）；

（4）修改demmer5函数改变队员的行为具体步骤如下：

① 在demeer5函数中找到

```
else if( WM->getFastestInSetTo( OBJECT_SET_TEAMMATES, OBJECT_BALL, &iTmp )
                == WM->getAgentObjectType()    && !WM->isDeadBallThem() )
    {                                                          // 如果是最快到达球的队员
      Log.log( 100, "I am fastest to ball; can get there in %d cycles", iTmp );
      soc = intercept( false );                                // 截球

      if( soc.commandType == CMD_DASH &&                       // 当体力低时
          WM->getAgentStamina().getStamina() <
              SS->getRecoverDecThr()*SS->getStaminaMax()+200 )
      {
        soc.dPower = 30.0 * WM->getAgentStamina().getRecovery(); //  慢速移动
        ACT->putCommandInQueue( soc );
        ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) ); //脖子转向
球
      }
      else                                                     // 当体力高时
      {
        ACT->putCommandInQueue( soc );                         //正常移动速度
       ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );//脖子转向球
      }
     }
```

此函数的内容是，当球不可踢时，如果是离球最近的队员就执行截球命令（截球函数intercept()在BasicPlayer.c中定义）观察在队员体力值小于多少时，带球速度会变慢。

② 现在我们通过修改函数来改变非持球队员的决策，在以上函数之前加上此段代码：

```
else if(WM->getAgentObjectType()==OBJECT_TEAMMATE_9) //如果是9号队员
{
    soc = SoccerCommand(CMD_TURN,60);        //转身体
    ACT->putCommandInQueue( soc );
    ACT->putCommandInQueue( alignNeckWithBody( ) );   //脖子随着身体一起转动
}
```

然后编译运行程序，观察球员的行为我们会发现当9号不持球时，身体在一直的转动（此动作可用来找球）

(5)在WorldMoled里添加状态函数

① .打开WorldModel.h，在里面预定义函数，即写入

bool　　isOpponentAtAngleEx(AngDeg angA , AngDeg angB ,double　　dDist)；该函数用来判

断当前球员角度在angA~angB之间距离小于dDist的范围内是否有对方队员

② .找到并打开WorldModel.c在里面添加一个新函数

```
bool      WorldModel::isOpponentAtAngleEx( AngDeg angA , AngDeg angB ,double      dDist )
{
  VecPosition posAgent     = getAgentGlobalPosition();
  VecPosition posOpp;
  AngDeg          angOpp;
  int               iIndex;

  for( ObjectT o = iterateObjectStart( iIndex, OBJECT_SET_OPPONENTS );
       o != OBJECT_ILLEGAL;
       o = iterateObjectNext ( iIndex, OBJECT_SET_OPPONENTS ) )
  {
    posOpp       = getGlobalPosition( o );
    angOpp       = ( posOpp - posAgent ).getDirection() ;
    if( angA<=angOpp && angOpp <=angB && posAgent.getDistanceTo( posOpp ) < dDist )
      return true;
  }
  iterateObjectDone( iIndex );
  return false;
}
```

③ .将 if(WM->isBallKickable())内的内容修改为：

```
if( WM->isBallKickable())
{
    double ang = (VecPosition(52.5,0)-posAgent).getDirection();
          if ( WM->isOpponentAtAngleEx(ang-45, ang, 6) )
                ang+=45;
         else if ( WM->isOpponentAtAngleEx(ang,ang+45,6) )
                ang-=45;
    SoccerCommand soc = dribble ( ang , DRIBBLE_SLOW    );
    ACT->putCommandInQueue( soc );
    ACT->putCommandInQueue( turnNeckToObject( OBJECT_BALL, soc ) );

}
```

然后编译运行程序，观察球员的行为，试分析球员的行为

到此为止，demeer5的基本内容已经向大家介绍完毕，希望大家能够熟练的应用demeer5函数。

根据以上描述完成练习：通过基本动作的组合实现球员的以下行为（要求至少做出1~2个题目，更多基本动作请查阅BasicPlayer.c函数,或其他有关书籍,也希望大家能做出更多的动作来）

- 判断守门员的位置，朝球门空隙较大的一方射门。（通过在WorldModel里建立新状态来判断，球门哪一方空隙较大，守门员的位置为VecPosition posGoalie = WM->getGlobalPosition(WM->getOppGoalieType());球门位置坐标为（52.5，0），可尝试朝（52.5，6.5）（52.5，-6.5）两点射门）
- 在 BasicPlay里添加一个带球函数，要求如果无人阻挡（带球将要经过的路线附近没有对方球员）就朝球门方向带球，否则想办法避开对方球员带球前进（要求只要作出闪避的动作即可，不要求效果）。
- 尝试修改视觉函数使得球员能更多的获得场上信息（要求不影响球员的动作）。

实验 5 特殊比赛模式的设计

1、实验目的

（1）掌握 Robocup 仿真机器人足球比赛中特殊比赛模式发生的条件；
（2）掌握 Robocup 仿真机器人足球比赛特殊比赛模式的规则要求；
（3）了解 Robocup 仿真机器人足球比赛特殊比赛模式的战术设计思想；
（4）进一步熟悉 WorldModel 类。

2、实验设备

硬件环境：PC 机
软件环境：操作系统 Linux

3、实验内容

（1）概述

Robocup 仿真机器人足球比赛特殊比赛模式包括**角球（corner_kick）**、**界外球（kick_in）**、**定位球/任意球（free_kick）**以及**球门球（goal_kick）**。

（2）角球（corner_kick）

当防守方球员将球踢出底线时，由进攻方开角球。Server 一旦接收到发球队员发出的 kick 命令后，就将比赛模式设为正常的 play_on 模式。注意，和国际足联的规则类似，发球队员在其他球员接触球之前不能再触球，否则判犯规。此时，另一方球员须在一定时间（根据 Manual 的规定为 300 周期）内将球开出，否则算发球失误，由对方发定位球。

在 UVA_trilearn 的原代码中，球员是不管这些特殊比赛模式的，统统是最近的球员跑向球，一脚踢向球门。这就需要我们手工编码完成这些细节地方。一种简单的设计思路是：（1）如果比赛进入角球模式，则离球最近的球员 A 跑向球，而次近的球员 B 跑到某个接应点，等待 A 将球传过来；（2）A 跑到球跟前（即进入可踢范围）时，不必立即将球开出，可以先看看场上环境，等 B 跑到预定位置并且体力恢复得差不多时再开球。

以上设计思路同学们在实验过程中只须完成（1）即可，对于学有余力的同学可以考虑（2）

的实现。下面列出几个可能用到的函数：

PlayModeT WorldModel::getPlayMode() const 返回比赛模式（或者直接用 bool WorldModel::isCornerKickUs() 判断是不是我方开角球）；

ObjectT WorldModel::getClosestInSetTo(ObjectSetT objectSet, ObjectT o, double *dDist=NULL, double dConfThr=-1.0) 返回在对象集合 objectSet 中距离对象 o **最近**的对象，只有当对象的可信度高于给定的阈值才被考虑，如果没有给出阈值则使用 PlayerSettings 中定义的阈值，同时 dDist 返回距离；

ObjectT WorldModel::getSecondClosestInSetTo(ObjextSetT objectSet, ObjectT o, double *dDist=NULL, double dConfThr=-1.0) 返回在对象集合 objectSet 中距离对象 o **次近**的对象，只有当对象的可信度高于给定的阈值才被考虑，如果没有给出阈值则使用 PlayerSettings 中定义的阈值，同时 dDist 返回距离。

下面具体描述一下实现过程：

首先，我们知道 demeer5()主要是围绕三句话展开的，即（1）如果球可踢，则用最大力量踢球；（2）如果球不可踢且我是队友中最快到球的队员，则去截球；（3）其他队友则按战略点跑位。我们可以围绕这三句话来实现角球策略。即在角球模式下（1）如果球可踢，则传球给接应球员；（2）如果球不可踢且我是队友中最快到球的队员，则去发球；（3）其他情况下，如果我是离球次近的队员，则跑到预定位置接应。

接着，有了这个基本思想后，我们开始编写代码。在 demeer5() 中找到

Else if (WM->isBallKickable()){ // if kickable

在里面加入我们的角球代码：

```
If ( WM-> isCornerKickUs() ){
    ACT->putCommandInQueue(kickTo(pointToKickTo(),SS->getBallSpeedMax()*0.8 ));
}
```

这一部分代码是完成开球动作。函数 pointToKickTo()是 Player 类的成员函数，返回一个 VecPosition 类型的参数，代表要将球踢向的坐标。下面给出一段参考代码：

```
VecPosition Player::pointToKickTo(){
    VecPosition pos,temp;
    double x,y;
    pos=WM->getBallPos();
    x=pos.getX();
    y=pos.getY();
    temp.setX(-x/fabs(x)*5+x);
    temp.setY(-y/fabs(y)*12+y);
    return temp;
```

```
    }
```

再在 demeer5()中找到

```
else if( posAgent.getDistanceTo(WM->getStrategicPosition()) >1.5 + fabs( posAgent.getX() -
posBall.getX() ) / 10.0)                    // if not near strategic pos
```

在里面加入如下代码：

```
If   ( WM-> isCornerKickUs() ) {
    ObjectT o;
    SoccerCommand sctemp;
    o = WM->getSecondClosestInSetTo(OBJECT_SET_TEAMMATES, OBJECT_BALL);
    if ( o = WM->getAgentObjectType() ){
        sctemp = ACT->putCommandInQueue( moveToPos( pointToKickTo(),
PS->getPlayWhenToTurnAngle() );
    }
    ACT = putCommandInQueue( turnNeckToObject( OBJECT_BALL, sctemp ));
}
```

这段代码是实现接应球员的跑位。

这样，一个简单的角球策略就实现了。同学们可以编译运行看看效果。

（3）界外球（kick_in）

当一方球员将球踢出边线时，比赛即进入界外球模式。

比赛的规则要求基本同角球。

简单的设计思路和实验内容与角球也差不多，在此不再赘述。

- 要求同学们编写一段程序实现界外球策略，要求能够根据发球点的位置给出合适的接应球员的接应位置。

可能用到的函数：

bool WorldModel::isKickInUs() 返回是否是我方开界外球。

（4）定位球/任意球（free_kick）

当一方球员犯规或违例时，由对方开任意球。跟国际足联的规定略有不同的是，Robocup 的任意球没有直接任意球跟间接任意球之分。

比赛的规则要求基本同界外球。

简单的设计思路和实验内容与界外球也差不多，在此不再赘述。略有不同的是，如果条件允许，开球队员可以选择直接射门。另外，有能力的同学可以在这里考虑实现开球队员等待一段时间再开球的策略（提示，要用到锁定机制）。

- 要求完成一种任意球的开球动作。

（5）球门球（goal_kick）（选做）

当对方进攻球员将球踢出底线或我方守门员截住对方射门的球时，由我方守门员开球门

球。注意，如果守门员截住的是己方后卫的回传球，那么会由对方球员在离截球点最近的禁区角点发任意球，因为这是违例行为。

如果守门员在 300 周期内不能将球开出，那么由对方球员在离开球点最近的禁区角点发任意球。

在 UVA_trilearn 的源代码中，守门员开球也是直接将球踢向对方球门。而开门球是一个比较复杂且非常重要的细节，许多强队，诸如清华、科大，都有不错的开门球策略。我们这次实验只完成一个简单的策略，即守门员沿某一直线开球，该直线是由最近对方球员、守门员、次近对方球员三点所构成角的角平分线。

附：角球伪代码：

```
Demeer5() {
    …
    If ( WM->isBallKickable() ) {
        …
        If ( WM->isCornerKickUs() ) {
            If ( not ready ) {
                Wait;
            } else {
                Kick to  接应球员;
            }
        }
        …
    }Else if ( I am fastest to the ball ) {
        Dash to the ball;
    }Else {
        …
        If ( WM->isCornerKickUs() ) {
            If ( I am second fastest to the ball ) {          // I am  接应球员
                Dash to  接应点;
            }
        }
        …
    }
}
```

注：

代码是分解穿插写在 demeer5() 中的，这样虽然降低了程序的可读性，但却大大简化了程序设计的复杂性。有能力的同学可以试着写一个函数来完成整个角球功能。界外球和任意球的设计可仿照该模板完成。

实验 6 数据采集及个体技术训练

1、实验目的

掌握 RoboCup 仿真程序（2D）中是如何进行训练或比赛数据的采集和实现球员智能体的传球、射门、截球等个体技术的。

（1） Trainer 基本使用方法;

（2） 带球数据的采集;

（3） 射门数据的采集;

（4） 截球数据的采集;

（5） 传球数据的采集;

（6） 一般机器学习方法在个体技术训练的使用。

2、实验设备

硬件环境：PC 机
软件环境：操作系统 Linux

3、实验内容

（1）Trainer的功能简介

Trainer 通过与 server 的功能消息传递来实现其功能，其功能主要有以下几点：

- 可以控制比赛模式；
- 可以向球员广播自定义格式和内容的消息；
- 可以把球员和足球放置到任意的球场位置，并且可以设置它们的方向和速度；
- 可得到无噪音的移动对象信息；
- 更换异构球员

与标准的 trilearn2002 源码对比，删去的文件为 BasicPlayer.h(C)和 Player.h(C)，添加的文件为 BasicCoach.h(C)和 Trainer.h(C) 然后根据实际需要编写相应的训练程序进行相关技术的训练。

(2) Trainer 中的功能函数

Trainer 的功能函数是对功能消息发送的封装，主要有以下几种：

changePlayMode(PlayModeT mode)

设置比赛模式.

moveBall(double X, double Y, double velx = 0.0, double vely = 0.0)

移动足球按指定的坐标和速度.

movePlayer(ObjectT player, double X, double Y, double vdir = 0, double velx = 0.0, double vely = 0.0)

移动球员按指定的坐标、朝向和速度.

recover()

恢复球员的身体状态.

start()

启动 Server.

substitutePlayer(int iPlayer, int iPlayerType)

更换异构球员，iPlayer 为被更换的球员号码，iPlayerType 为更换的异构球员类型.

(3) Trainer 的使用方法

① 修改 Server 的配置文件;

在 root 目录下键入 vi .rcssserver-server.conf;

将 half_time:300 改为 half_time:具体的数据,这样半场的时间不再是 300 周期,而是根据训练模块具体要求设置时间;

将#coach_w_referee 改为 coach_w_referee,开启训练器端口.

② 修改 trainer 的配置文件;

在 trainer 目录下键入 vi trainer.conf;

开启具体的训练模块,例如 TMShoot(射门模块),TMDirbble(带球模块),开启的方法很简单,只要去掉该模块前面的"#"即可.

③ 启动 server 和 monitor;

④ 根据训练模块的具体要求上相应数目的球员,例如截球模块就需要上两个球员.

⑤ 键入./trainer 开始训练.

当然,为了避免每次训练都重复调用上述命令,我们也可以编写训练脚本来简化工作.下面给出一个训练脚本的例子。

```
#!/bin/tcsh
set TeamA="./start.sh 9"          //设置变量 TeamA
set TeamB="./start.sh 9"          //设置变量 TeanB
set Trainer="./trainer"           //设置变量 Trainer
echo "Running the training script"
sleep 1                           //延时
rcssserver &                      //启动 server
set PID = $!                      //得到 server 的进程号
echo the PID of server is $PID
sleep 2
cd /root/Projects/trilearn_2002_source_opp  //进入我方球队所在的目录
$TeamA                                      //上一个我方球员
sleep 1
cd /root/trilearn_2002_binary               //进入对方球队所在的目录
$TeamB                                      //上一个对方球员
sleep 1
echo start trainer....
cd /root/trainer                            //进入 trainer 所在的目录
$Trainer                                    //启动 trainer 开始训练
sleep 1
kill $PID                                   //关闭 server
```

(4) 带球数据的采集

下面通过带球模块来讲述如何通过 Trainer 来采集数据.

① 在 trainer 程序中找到 TMDribble.c 并打开;

② 在 TMDribble.c 中找到 mainloop 函数,该函数为每个训练模块的主循环,我们分析一下程序代码.

```
while (bContLoop && (!isTrainOver()))
                                //isTrainOver()函数用来判断训练是否结束
{
  setNewScenario();             //设置新场景
  if (training() == true)       //training()为训练函数,具体代码稍后分析
  {
    logPerformance();           //用于记录训练中采集的数据
  }
}
```

③ 在 TMDribble.c 中找到 training 函数,该函数为训练函数,抽取主要代码进行

行分析.

```
……
switch (playMode)
{
  case PM_PLAY_ON:
  posBall = WM->getBallPos();          //得到球的位置
  posOur   = WM->getGlobalPosition(OBJECT_TEAMMATE_1);
                                       //得到我方一号球员的位置
  posOpp   = WM->getGlobalPosition(OBJECT_OPPONENT_1);
                                       //得到对方一号球员的位置
  if (posBall.getDistanceTo(posStart) > TS->getDribbleRadius())
                                       //检查我方球员是否将球带出特定的区域
{
      m_bSucceed =true;                //设置带球成功标志
      m_dDribbleDist = TS->getDribbleRadius();//记录带球的距离
      m_iCurrentEpoch++;                        //带球成功,训练步数加 1
      cout << "dribble out of the area successfully" << endl;
      return true;
  }
      if ( fabs((posOpp - posBall).getMagnitude())
        <= SS->getMaximalKickDist())
                                                //检查对手是否截到球
  {
     m_bSucceed = false;                      //带球失败
     m_dDribbleDist = posBall.getDistanceTo(posStart);
                                                //记录带球的距离
     m_iCurrentEpoch++;                       //带球失败,训练步数加 1
     cout << "opponent can kick the ball" << endl;
     changePlayMode(PM_TIME_OVER);
     poll(0,0,2000);
     return true;
  }
  currentStateInfo.relDist = (posOpp - posBall).getMagnitude();
  currentStateInfo.relAngle = posBall.getDirectionTo(posOpp);
  currentStateInfo.ballSpeed = WM->getBallSpeed();
  //以上三条语句分别记录对手与球的相对距离,相对角度,以及球的速度.
    ……
```

上面我们详述了带球训练的主要程序代码,如果你想了解一次训练的来龙去脉,你可以继续阅读该类中的其他几个函数, setNewScenario(),logHeader()及 logPerformance(),你会发现写训练程序是如此的简单.当然,如果训练球员某个具体动作应该修改 demeer5 的相关函数,这在前面的实验中已经介绍.

根据上面描述完成如下练习：

① 将我方球员放在对方禁区内；

② 将球放在我方球员的可控范围内；

③ 将带球区域的半径设为 18 米(修改 trainer.conf 中的 dribble_area_radius)；

④ 将一次训练的总训练步数设为 100,每个训练步为 20 周期(修改 trainer.conf 中的 epoch_length 和 epoch_time_out_cycles)；

⑤ 完成 100 步的带球训练,采集每一个场景的数据,将其保存在文件中(按**(3)**中所描述的步骤进行操作,注意我们应开启的是 TMDribble 模块)。

(5) 射门数据的采集

一般认为进攻禁区内为球员智能体的射门区域，在此区域内的任意点均为可射位置。对于射门训练我们要求设置这样一个场景,将我方球员放在对方禁区内的任意位置,对方守门员放在相应的位置.要求编写一个射门训练模块,记录下每次射门成功时我方球员和对方守门员所在的位置，射门路线。

(6) 截球数据的采集

截球问题的描述见讲义 7.2.1.要求编写一个截球训练模块,将我方球员及球放在场上任意位置,记录下每次截球成功时截球队员与球的相对距离,截球队员与球的相对方向以及球的速度。

(7) 传球数据的采集

传球问题的描述见讲义 7.2.2.要求编写一个传球训练模块,参照讲义将传球队员,接球队员及球放在合适的位置,记录下每次传球成功时传球队员与接球队员之间的相对距离,接球队员与球之间的相对距离,相对方向以及球的速度.

(8) 机器学习方法在动作训练中的应用

如上所述,我们通过训练器采集到大量的数据,有了这些数据我们就可以用机器学习的方法进行处理.常用的机器学习方法有决策树算法,神经网络,贝叶斯网络及强化学习等等.

举个简单的例子,根据射门训练模块采集到的数据,我们可以用 BP 网络进行学习,网络的输入是我方球员位置和对方守门员的位置,输出是进球与否.Matlab 提供了神经网络工具箱供我们训练使用,请参考相关文献.

机器学习的方法对于提高球员的个体技术非常有效,对于具体的问题可以尝试采用不同的方法,有兴趣的同学可以查阅相关文献,对采集到的数据进行训练以优化球员的个体技术.

实验 7 机器人足球队的完整设计

1、实验目的

根据以上 7 个实验，大家应该可以设计出一支具有一定思路的机器人足球队，本实验就是要求大家按照小组的方法完成一支仿真足球队伍。

2、实验设备

硬件环境：PC 机
软件环境：操作系统 Linux

3、实验内容

完成一支完整的仿真机器人足球队。