**实验七 机器人足球队的完整设计**

**一、实验目的**

根据以上6个实验,大家应该可以设计出一支具有一定战术策略的机器人足球队，本实验就是要求大家按照小组的方法完或一支仿真足球队伍。

**二、实验设备**

硬件环境：PC机；

软件环境：操作系统linux。

**三、实验内容**

**完成一支完整的仿真机器人足球队。**

**分工：**

关于分工，敬成超，屈天奇，周诗奥都贡献了自己的力量。一开始，由周诗奥上网搜集资料。屈天奇，敬成超负责实现具体的要求细节。明确了修改方向，鉴于我们水平初级，便想在一些细节入手，让踢球更人性化，更有竞争力。敬成超负责整合意见，撰写报告。

于是，我们就开始实践了。

屈天奇同学负责修改核心战术逻辑和比赛策略的设计，如开球时如何决定谁开球、比赛时的不同球员行为逻辑（如进攻、防守、抢球、回位等）。

在代码中修改了比赛关键情景的逻辑，例如判断球是否可踢（isBallKickable()）、球员距离球的相对位置（getFastestInSetTo()），以及射门和防守的具体策略。

提供了整体的阵型策略（setFormation(FT\_433\_OFFENSIVE)），并结合球场布局计算最佳站位。

敬成超同学修改核心代码的优化和功能扩展，引入新的指令集、扩展球员行为选项，以及处理更多比赛场景。

在代码中加入了新的运动方向（如CMD\_DASH的角度设定）和能量管理功能，增强了球员的动态表现。

对代码细节进行了优化，例如处理低体力的球员行为（getAgentStamina()与getRecovery()），使其在比赛中更加真实和智能化。

添加了新功能，如守门员的防守区域计算逻辑，通过几何方法（Line::makeLineFromTwoPoints）计算最佳拦截点。

以下是他的修改代码，及见解。

周诗奥同学负责代码的调试与注释工作，确保每一部分逻辑清晰明了，同时记录代码的关键改动，便于团队沟通和后续维护。

在代码中添加了大量详尽的注释和日志信息（如Log.log()），帮助解释代码逻辑，并为未来的调试提供支持。

增加了注释，说明每个重要函数的作用，例如deMeer5函数中的策略说明和逻辑起点标记。

添加了友好的日志输出（例如“kick ball”或“I am fastest to ball”），以便分析比赛行为是否符合预期。

我们在策略上做了功夫，在Player.cpp中增加了功能，细节。

例如

SoccerCommand Player::deMeer5( )

{

SoccerCommand soc(CMD\_ILLEGAL);// 定义一个命令对象，后面会频繁用它来构造一个球员动作命令

VecPosition posAgent = WM->getAgentGlobalPosition();// 一个点对象，表示Agent(可以理解成当前在执行这个代码的球员它自己)的位置

///下面我们把“我“就理解成当前执行这个代码的Agent，因为有11个球员在执行这个代码，所以对于每个球员 肯定有一个“我”存在。

VecPosition posBall = WM->getBallPos();// 球的位置

int iTmp;

if ( WM->isBeforeKickOff( ) ) /// 如果还没有开球 注意！ 这里只是还没有开球的情况！ 要做开球后的决策，向下面继续找另外一个 WM->isBeforeKickOff( )

{

if ( WM->isKickOffUs( ) && WM->getPlayerNumber() == 9 ) // 9 takes kick //就找9号去开球（判断了是不是我们开球，和我"Agent"是不是9号）

{

if ( WM->isBallKickable() )// 当球对于我来说是不是可踢！

{

VecPosition posGoal( PITCH\_LENGTH/2.0,

(-1 + 2\*(WM->getCurrentCycle()%2)) \*

0.4 \* SS->getGoalWidth() );// 计算一个射门点

soc = kickTo( posGoal, SS->getBallSpeedMax() ); // kick maximal 以最大速度把球踢向射门点！

Log.log( 100, "take kick off" );

}

else //如果球对于我不可踢，那我就去抢球！

{

soc = intercept( false );

Log.log( 100, "move to ball to take kick-off" );

}

ACT->putCommandInQueue( soc );

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) );

return soc;

}

开球后的决策，屈天奇同学丰富了对象的细节

else

{

formations->setFormation( FT\_433\_OFFENSIVE );//设置球队出场阵形！

soc.commandType = CMD\_ILLEGAL;//初始化soc命令对象

///test SoccerCommand(CMD\_DASH, 80, 45) for v14 ... fixed by misol.gao

/\* if ( WM->getPlayerNumber() == 8 )

{

soc = SoccerCommand(CMD\_DASH, 80, 45);

ACT->putCommandInQueue( soc ); // 放入命令队列

ACT->putCommandInQueue( alignNeckWithBody() );

return soc;

}\*/

if ( WM->getConfidence( OBJECT\_BALL ) < PS->getBallConfThr() )//判断对球的可信度，如果小于某个阈值，则...也就是说，如果很多周期没有看到球在哪里了

{

ACT->putCommandInQueue( soc = searchBall() ); // if ball pos unknown //执行找球动作！并放入命令队列

ACT->putCommandInQueue( alignNeckWithBody( ) ); // search for it //同时把脖子随身体一起转

}

敬成超同学采用了直截了当方法直接踢球

这里开始踢球

else if ( WM->isBallKickable()) // if kickable // 如果球已知，而且当前球在我脚下(可踢)

{

VecPosition posGoal( PITCH\_LENGTH/2.0,

(-1 + 2\*(WM->getCurrentCycle()%2)) \* 0.4 \* SS->getGoalWidth() ); //定义一个射门点 PITCH\_LENGTH是球场的长度。这个点就是按照周期看，交替的打球门的两个死角。

soc = kickTo( posGoal, SS->getBallSpeedMax() ); // kick maximal // 把球以最大速度踢向那个射门点

ACT->putCommandInQueue( soc ); // 放入命令队列

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) ); // 把脖子转向球，也就是一直看着球

Log.log( 100, "kick ball" );

}周诗奥同学提出了想法，球不在我手里那么就抢不能傻楞这

else if ( WM->getFastestInSetTo( OBJECT\_SET\_TEAMMATES, OBJECT\_BALL, &iTmp )

== WM->getAgentObjectType() && !WM->isDeadBallThem() ) // 如果球不在我的控制范围下，但是当前能最快抢到球的是我，那我就去执行抢球动作

{ // if fastest to ball

Log.log( 100, "I am fastest to ball; can get there in %d cycles", iTmp );

soc = intercept( false ); // intercept the ball

}

else if ( posAgent.getDistanceTo(WM->getStrategicPosition()) >

1.5 + fabs(posAgent.getX()-posBall.getX())/10.0) // 到了这里就是其他距离球相对远一点的人了，如果离自己的阵形点太远，就跑回 自己的阵形点去。

// if not near strategic pos

{

if ( WM->getAgentStamina().getStamina() > // if stamina high

SS->getRecoverDecThr()\*SS->getStaminaMax()+800 )

{

soc = moveToPos(WM->getStrategicPosition(),

PS->getPlayerWhenToTurnAngle());

ACT->putCommandInQueue( soc ); // move to strategic pos

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) );

}

else if ( fabs( WM->getRelativeAngle( OBJECT\_BALL ) ) > 1.0 ) // watch ball //其他剩下的球员呢，就看球。！在这里屈天奇对其他球员的行为做了修改，增加了转向移动断球功能

{///这里就是无球队员的跑位决策

ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT\_BALL ) );

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) );

}

else // nothing to do

ACT->putCommandInQueue( SoccerCommand(CMD\_TURNNECK,0.0) );

}

return soc;

}

if ( WM->getConfidence( OBJECT\_BALL ) < PS->getBallConfThr() )

{ // confidence ball too low

ACT->putCommandInQueue( searchBall() ); // search ball

ACT->putCommandInQueue( alignNeckWithBody( ) );

}

else if ( WM->getPlayMode() == PM\_PLAY\_ON || WM->isFreeKickThem() ||

WM->isCornerKickThem() )

{

posIntersect = lineRight.getIntersection( lineBall );

else if (posIntersect.isLeftOf( posLeftTop ) )

posIntersect = lineLeft.getIntersection( lineBall );

if ( posIntersect.getX() < -49.0 )

posIntersect.setX( -49.0 );

// and move to this position

if ( posIntersect.getDistanceTo( WM->getAgentGlobalPosition() ) > 0.5 )

{

soc = moveToPos( posIntersect, PS->getPlayerWhenToTurnAngle() );

ACT->putCommandInQueue( soc );

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) );

}

else

{

ACT->putCommandInQueue( soc = turnBodyToObject( OBJECT\_BALL ) );

ACT->putCommandInQueue( turnNeckToObject( OBJECT\_BALL, soc ) );

}

}

}周诗奥对该模式下的情况做出判断并采取行动

else if ( WM->isFreeKickUs() == true || WM->isGoalKickUs() == true )

{

if ( WM->isBallKickable() )

{

if ( WM->getTimeSinceLastCatch() == 25 && WM->isFreeKickUs() )

{

// move to position with lesser opponents.

if ( WM->getNrInSetInCircle( OBJECT\_SET\_OPPONENTS,

Circle(posRightTop, 15.0 )) <

WM->getNrInSetInCircle( OBJECT\_SET\_OPPONENTS,

Circle(posLeftTop, 15.0 )) )

soc.makeCommand( CMD\_MOVE,posRightTop.getX(),posRightTop.getY(),0.0);

else

soc.makeCommand( CMD\_MOVE,posLeftTop.getX(), posLeftTop.getY(), 0.0);

ACT->putCommandInQueue( soc );

}

敬成超在basicplayer.cpp里增加了一些函数，让球员的动作更多样化，抉择更灵活。

以下是他的修改代码，及见解。

*// 朝较大的夹角射门*

根据与守门员的角度关系，选择一个适当的位置进行射门。

如果角度差异较大，选择向一个侧边射门，从而可能提高得分的概率。

提高了机器人射门时的智能化，能够更灵活地选择最佳的射门方向，尤其在面对强手时，能避免守门员拦截。

SoccerCommand BasicPlayer::maxAngShoot( VecPosition posAgent )

{

SoccerCommand soc;

VecPosition posGoalie;

AngDeg ang\_goalie, angup, angdown;

posGoalie = WM->getGlobalPosition(WM->getOppGoalieType());

ang\_goalie = (posGoalie - posAgent).getDirection();

angup = (VecPosition(52.5,6.0) - posAgent).getDirection();

angdown = (VecPosition(52.5,-6.0) - posAgent).getDirection();

if(fabs(angup - ang\_goalie) > fabs(angdown - ang\_goalie))

{

soc = kickTo(VecPosition(52.5, -6.5),2.7);

}

else

{

soc = kickTo(VecPosition(52.5, 6.5),2.7);

}

return soc;

}

*// 朝球门方向将球以最大力度踢球*

：

保证踢球的力度达到最大，使得球更难被守门员拦截，增强了直接得分的机会。

SoccerCommand BasicPlayer::maxAngShoot1( VecPosition posAgent )

{

SoccerCommand soc;

VecPosition posGoal(PITCH\_LENGTH/2.0,

(-1 + 2\*(WM->getCurrentCycle()%2)) \*

0.4 \* SS->getGoalWidth());

soc=kickTo( posGoal,SS->getBallSpeedMax());*// 朝球门方向将球以最大力度*

return soc;

}

*// 闪避带球*

处理带球时的闪避逻辑。

计算现在的运动方向，再根据敌方位置判断是否需要调整运动方向来躲避对手。

让机器人在带球时更具反应能力，能够在对抗中灵活应对，降低被对手拦截的风险

SoccerCommand BasicPlayer::dribble\_Dodge(VecPosition tar, VecPosition posAgent)

{

SoccerCommand soc;

AngDeg ang = ( tar - posAgent ).getDirection();

soc = dribble(ang, DRIBBLE\_FAST );

if( WM->isOpponentAtAngleEx( ang - 45, ang , 7 ) )

{

ang += 45;

soc = dribble( ang , DRIBBLE\_FAST );

}

else if( WM->isOpponentAtAngleEx( ang, ang + 45 , 7 ) )

{

ang -= 45;

soc = dribble( ang , DRIBBLE\_FAST );

}

return soc;

}

*// 计算角球应该踢向的点*

**4.** 计算角球应该踢向的目标位置。

通过当前球的位置，计算并返回最佳的踢向目标位置，以增加进攻时的灵活性。：

提供了明确的踢向点，有助于在比赛中进行更精准的策略执行，提升了团队配合的效率

VecPosition BasicPlayer::pointKickToC()

{

VecPosition posBall, posKickTo;

double x, y;

posBall = WM->getBallPos();

x = posBall.getX();

y = posBall.getY();

posKickTo.setX(- x / fabs(x) \* 5 + x );

posKickTo.setY(- y / fabs(y) \* 12 + y );

return posKickTo;

}

*// 计算界外球应该踢向的点*

通过当前球的位置，计算并返回非被防守位置的最佳界外球踢向目标。

确保机器人在界外球时能够有效地给队友创造机会，增强比赛中的策略灵活性。

VecPosition BasicPlayer::pointKickToK()

{

VecPosition posBall, posKickTo;

double x, y;

posBall = WM->getBallPos();

x = posBall.getX();

y = posBall.getY();

posKickTo.setX( x + 7 );

posKickTo.setY(- y / fabs(y) \* 12 + y );

return posKickTo;

}

这些修改和新增的功能整体上增强了机器人的实战能力，使得它们可以在动态环境下更好地做出决策。通过智能化的射门、强劲的踢球、灵活的闪避、清晰的传球点计算，机器人在比赛中的表现得以提升。这样的战术优化使得机器人在复杂场景下仍能做出有效的反应，并保持游戏的竞争性。而通过这些代码的方法，团队合作和策略执行也变得更加协调和有效。

周诗奥同学的代码，在WorldModel.cpp里做了修改，

void WorldModel::setSynchSeeMode()

{

bSynchSeeMode = true;

}

bool WorldModel::isSynchSeeMode()

{

return bSynchSeeMode;

}

第一段代码新增了setSynchSeeMode和isSynchSeeMode函数，用于设置和检查是否处于同步视觉模式。

bool WorldModel::ifCanPassBall(Robot robot)

{

*// ... 省略部分代码 ...*

*//搜索是否存在传球路径*

for(int i=1;i!=5;++i)

{

*// ... 省略部分代码 ...*

if(ifOpsCannotIntercept(preBall[countBall].getPosition()))

{

if(ballTime >= ownTime)

{

*//把截球相关信息存入结构体*

tempInter.pos = preBall[countBall].getPosition();

tempInter.robotNo = i;

tempInter.time = ballTime;

tempInter.ang = ang;

tempInter.value = 0;

pathVector.push\_back(tempInter);

DoLog(CT\_TEST, "执行了信息压入！");

}

else

{

continue;

}

}

else

{

continue;

}

}

*//判断是否存在传球路径*

if(pathVector.size() > 0)

{

return true;

}

else

{

return false;

}

}

ifCanPassBall函数的逻辑有所修改，特别是在判断是否可以传球时，考虑了对方是否无法拦截球的情况。

void WorldModel::DoorKeeper(void)

{

*// ... 省略部分代码 ...*

if(ifCanCatch())

{

cmd.CommandType = Com\_Catch;

ownRobots[0].setCommand(cmd);

}

else if(ownRobots[0].getPosition().getDistanceTo(ball.getPosition()) < CKICKABLEDIST)

{

if(ifCanPassBall(ownRobots[0]))

{

passBall(ownRobots[0], doorkeeperCmd);

ownRobots[0].setCommand(doorkeeperCmd);

DoLog(CT\_TEST, "踢球方向 %f, 踢球力度 %f", doorkeeperCmd.ComParam0, doorkeeperCmd.ComParam1);

}

else

{

kickOutOfArea(ownRobots[0],doorkeeperCmd);

ownRobots[0].setCommand(doorkeeperCmd);

}

}

else

{

*// ... 省略部分代码 ...*

}

}

第一段代码中，DoorKeeper函数的逻辑有所修改，特别是在守门员决策时，考虑了是否可以传球和是否可以将球踢出禁区的情况。

总结：在机器人足球实验的代码修改和功能完善过程中，团队通常会获得很多方面的收获和体验，包括技术能力的提升、团队协作的改善、以及对机器人的控制和决策系统更深入的理解。以下是一些具体的收获：

**1. 技术能力提升**

**编码能力**：通过修改代码和添加新功能，程序员的编码技巧和问题解决能力得到了提高。

**算法与数据结构**：在实现复杂的足球策略时，团队需要运用各种算法（如路径规划、预测模型）和数据结构（如队列、树、图）来优化机器人的行为。

* **使用标准库**：对 Linux 工具和库（如 tar, make, gcc 等）的熟悉程度提高，能够在实际应用中更加熟练地使用这些工具。

**2. 对机器人控制理解加深**

**运动控制**：通过修改运动控制的算法，团队对机器人如何在动态环境中平衡和移动有了更深刻的理解。

**感知与决策**：通过增加决策逻辑和预测能力，团队学习如何使机器人根据周围环境做出更智能的决策。

**3. 团队协作与项目管理**

**协作开发**：在团队项目中，成员分工合作进行代码的编写和修改，有助于提高沟通能力和团队协作能力。

**版本控制**：使用版本控制系统（如 Git）进行代码管理，提高了代码的可维护性与历史版本的追踪能力。

**文档和注释**：意识到良好文档和注释的重要性，确保其他团队成员能够迅速理解做出的更改和代码的意图。

**4. 实验与测试**

**实验精神**：在实践中，通过不断的实验与测试，发现并解决了许多潜在的问题，这对代码质量和功能实现有重要影响。

**反馈与迭代**：通过持续地获取来自仿真测试的反馈，团队能够迭代和优化代码，确保最终系统的有效性和鲁棒性。

**5. 解决问题的技能提高**

**调试能力**：修改后的代码往往会引入新的问题，团队成员在调试过程中提升了分析和解决问题的能力。

**处理异常情况**：在实际应用中，添加了对各种边界条件和异常情况的处理逻辑，例如如何在对手接近时做出反应。

**结论**

通过对代码的修改和功能的完善，团队在技术能力、项目管理、团队合作等各个方面都获得了重要的经验和成长。在机器人足球这样的动态环境中，有效的合作和技术创新是成功的关键。这样的经历不仅有助于提高他们在机器人工程方面的专业能力，同时也为未来的项目打下了良好的基础。