摘　要

随着计算机技术的发展，分布式人工智能中多智能体系统(MAS：Multi-agent System)的理论及应用研究已经成为人工智能研究的热点。RoboCup（Robot World Cup），即机器人世界杯足球锦标赛，是一个在实时异步，有噪声的对抗环境下，研究多智能体的决策和合作的问题。要实现多智能体间的良好的合作必须克服诸如通讯带宽低，通讯不可靠，干扰因素多，每个智能体获得的感知信息是不完整的，智能体的执行动作也存在误差等问题。作为国际上一项为提高相关领域的教育和研究水平而举行的大型比赛和学术活动，通过提供一个标准任务来促进分布式人工智能、智能机器人技术、及其相关领域的研究与发展。

本篇论文详细介绍了计算机仿真足球机器人的设计与实现。首先介绍了RoboCup 的研究背景，然后介绍了 SoccerServer 的仿真环境。从第三章开始介绍了具体的足球机器人 Agent 的设计和实现。Agent 的框架部分说明了开发环境，程序的基本框架。通讯层主要分析智能体获得信息后，怎么处理这些信息，更新智能体内部维持的世界模型。 Agent 的个人技术部分则侧重阐述了程序设计中的难点。最后分析了 Agent 的顶层策略，讲述了球队上层结构的设计，提出了 Agent 的智能化设计。例如基于场上形势的动态站位和角色变换，智能性的观察和通讯机制。

关键词：

RoboCup，SoccerServer，多智能体系统，分布式人工智能

Abstract

With the development of the computer technology, research on the theory and application of Multi-agent system (MAS) has become a hotspot of Artificial Intelligence. RoboCup，the Robot World Cup，is a fully distributed, multi-agent domain with both teammates and adversaries under the condition of real-time asynchronous and noisy conflict. In order to achieve a good co-operation between the agents, some problems must be resolved such as low bandwidth, unstable communication and too much interference. As an academic initiative of the education and research in such fields, RoboCup provides a standard project for development of DAI (Distributed Artificial Intelligence), Robotic, etc.

The paper presents a Design and Implement of the Soccer Simulator Robot. First, the background of RoboCup and the simulator model of the Soccerserver are introduced. Then the design and implementation of Soccer Agent is described. In chapter 3, the main framework of the Agent is suggested. Simultaneously, how to maintain the world state when the agent receives sense information from server is analyzed. Finally the top-level strategy such as Situation Based Dynamic Positioning and Role Exchange, Intelligent Perception and Communication is discussed in detail.

Keywords:

RoboCup, SoccerServer, Multi-agent system,Distributed Artificial Intelligence

目　　录

第一章　RoboCup 研究背景

1.1　引言

RoboCup（Robot World Cup），即机器人世界杯足球锦标赛。它是国际上一项为提高相关领域的教育和研究水平而举行的大型比赛和学术活动，通过提供一个标准任务来促进分布式人工智能、智能机器人技术、及其相关领域的研究与发展。

训练和制造机器人进行足球赛，是当前人工智能和机器人领域的研究热点之一。机器人足球比赛的设想首先是由加拿大不列颠哥伦比亚大学的教授 *Alan Mackworth* 在 1992 年的论文《*On Seeing Robots*》中提出的。同年 10 月，在东京举行的关于人工智能领域重大挑战的研讨会中，与会的研究人员讨论了人工智能领域中具有挑战性的问题，同时对制造和训练机器人进行足球比赛以促进相关领域研究的问题进行了探讨。在一些学者的积极倡导（如美国 CMU 的 *Manuela M. Veloso* 教授等），以及 SONY 公司的全力支持下，RoboCup 世界联合会于 1996 年宣告成立，并于同年在日本举行了一次表演赛，获得了很大地成功。比赛每年举办一届。第一届 RoboCup 比赛和会议于 1997 年在日本的名古屋举行，大约有 40 支机器人球队（包括美国、日本和欧洲的主要大学及研究机构）和超过 5000 名观众参加此次盛会。1998 年 7 月，正当第 16 届世界杯足球赛渐入佳境之时，第二届机器人足球世界杯赛在巴黎隆重举行，60 多支球队参加了比赛。1999 年 7 月，第三届机器人足球世界杯赛在瑞典斯德哥尔摩举行，参赛队多达 90 余支。一些著名的大学（如美国卡内基－梅隆大学、康奈尔大学）、国立研究机构（如美国 NASA[国家航空和宇宙航行局]）和大公司（如日本 SONY 公司）均参与了相关的活动和比赛。第四届 RoboCup 机器人足球世界杯赛在澳大利亚墨尔本举行，中国有了第一支进入 RoboCup 世界杯赛仿真组比赛的球队科大蓝鹰队，并获得第九名。今年 8 月将在美国西雅图举行第五届机器人足球世界杯赛。

1997 年可以说是人工智能发展史上重要的一年。1997 年 5 月，IBM 公司的深蓝计算机在国际象棋比赛中击败了人类的国际象棋冠军卡斯帕洛夫。1997 年 7 月 4 日，NASA 的自治机器人系统——寄居者，成功登陆火星，并在火星表面

开展活动。1997 年 8 月 23 日到 29 日，第一届 RoboCup 比赛和会议成功举行，从而为实现机器人足球队在未来击败人类足球冠军队的梦想迈出了坚实的第一步。

举办机器人世界杯足球赛的目的是为了促进分布式人工智能研究与教育的发展。通过提供一个标准任务，使得研究人员利用各种技术，获得更好的解决方案，从而有效促进各领域的发展。为了让一个机器人组成的足球队更接近一个人类的足球队，各种技术必须完美的结合在一起。涉及的研究领域包括：智能机器人系统、多智能体系统、实时模式识别与行为系统、智能体结构设计、实时规划和推理、基于网络的三维图形交互、传感器技术等。研究目标是计划经过五十年左右的研究，使机器人足球队能战胜人类足球冠军队。

在棋类比赛中人工智能已经取得了很大的成功。比较一下 Chess 和 RoboCup 的区别(表一)，可以明显看出 RoboCup 有如下特点：动态、实时、信息不完全、非符号化，和分布性。这就对人工智能的研究提出了更高的要求。

	Chess	Robocup
Environment	Static	Dynamic
State Change	Turn Taking	RealTime
Info.accessibility	Complete	Incomplete
Sensor Readings	Symbolic	Non-Symbolic
Control	Central	Distributed

表一 棋类比赛和 RoboCup 的对比

目前，RoboCup 活动包括如下几个方面：

1. 技术会议
2. 机器人足球赛
3. 机器人足球挑战赛
4. 教育程序
5. 基础结构发展

当然，机器人足球赛是整个活动的主要部分，研究者们会聚一堂，评价研究的进展情况。机器人足球赛锦标包括：

1. 仿真组比赛

2. 小型机器人组比赛
3. 小型机器人组标准比赛（每队 11 人）
4. 中型机器人组比赛
5. Sony 有腿机器人比赛 (Sony 公司发起)
6. 类人机器人组比赛 (计划从 2002 年开始)
7. 遥操作比赛(即将宣布)
8. RoboCup 展示会

RoboCup 仿真组比赛是各种比赛中参赛队数目最多的一种。由于仿真环境与人类足球比赛的环境相似，比赛队员的仿真模型与实际队员也很接近，这里可以实现机器人比赛中由于机器人硬件的不足而放弃的规则，故其对于分布式人工智能理论的研究具有重要意义。

1.2 中国 RoboCup 研究现状

1999 年 6 月，经国际 RoboCup 联合会主席北野宏明授权，由清华大学和中国科大共同发起，成立了国际 RoboCup 联合会中国分会，即 China RoboCup Association（CRA)。并于当年 10 月在重庆举办第一届中国 RoboCup 仿真机器人足球赛。2000 年 6 月在安徽合肥中国科学技术大学举行的全球智能控制大会上同时进行了中国 RoboCup2000 机器人足球锦标赛。中国 RoboCup-2001 机器人足球锦标赛将于今年 8 月 13 日至 16 日在昆明举行，参赛队伍有浙江大学、中国科技大学、清华大学、北京理工大学等院校。中国科技大学已经参加了 2000 年 RoboCup 世界杯赛仿真组比赛，还将参加今年八月在美国西雅图举行的第五届机器人足球世界杯仿真组比赛和有腿组比赛；参加今年仿真组比赛的中国队伍还有清华大学队。

第二章　SoccerServer 仿真环境

2.1　RoboCup 仿真比赛环境

RoboCup 仿真组比赛是各种比赛中参赛队数目最多的一种。仿真比赛是在一个标准的计算机环境内进行的，比赛规则与国际足球联合会的比赛规则相似，只是在某些方面有改动。本节简要的介绍仿真比赛主要部分。

比赛采用 Client/Server 方式，由 RoboCup 联合会提供标准的 SoccerServer 系统，参赛队编写各自的 Client 程序，模拟实际足球队员进行比赛。

SoccerServer 是一个允许竞赛者使用各种程序语言进行仿真足球比赛的系统。它提供了一个虚拟场地，并对比赛双方的全部队员和足球的状态进行仿真。Client，相当于球员的大脑，指挥球员的运动。由于比赛是以 Client/Server 方式进行的，所以对球队的开发编译没有任何限制。仅要求球队的开发工具提供通过 UDP/IP 连接的 Client/Server 支持，这是因为 Server 和每个 Client 之间的通讯都是通过 UDP/IP 端口实现的。每个 Client 都是独立的进程，通过给定的端口和 Server 连接。一支球队可以有最多 11 个 Client（或者说是球员），也就是说竞赛者同时运行与比赛球员数目相等的 Client。当球员和 Server 连接上后，所有的信息都通过这个端口传递。球员发送他们下一步要做的动作请求给 Server（如踢球，转身等）。Server 接收到这些消息后，执行对应指令，并相应的更新

图 1 SoccerMonitor

环境。另一方面，Server 向所有的球员提供感知信息(如：关于足球，球门和其他球员位置的可视信息)。每个 Client 模块只允许控制一名球员，Client 之间不允许直接进行通信，Client 之间的通讯必须通过 SoccerServer 来进行，因此通讯带宽受到一定的限制。SoccerServer 包含两个程序：SoccerServer 和 SoccerMonitor。

SoccerServer 的工作是仿真足球和队员的状态、与 Client 进行通信、按照一定的规则控制游戏的进程（裁判功能）。SoccerMonitor 是一个可视化的工具，负责利用 Windows（X window 或 Windows 9X/2K）系统显示虚拟场地，如图 1 所示。在 SoccerMonitor 上显示的信息包括比分，球队名字，所有球员和足球的位置。SoccerMonitor 也提供了一个很简单的 Server 接口。如：当两支球队都连接上后，可以点击在 SoccerMonitor 上的“Kick－Off”按钮开始比赛。当然，在 Server 上进行比赛，SoccerMonitor 并不是必需的。然而，如果有需要的话，可以同时与多个 SoccerMonitor 相连，在多个显示器上同时显示比赛的情况。

2.2 SoccerServer 仿真模型

2.2.1 场地和球员

仿真环境中足球场和其中的全部对象都是二维的。任何对象都没有高度的概念。比赛场地的尺寸为 105×68（单位没有意义），球门宽度为 14.64，是实际比例的两倍。实验证明，对于正常的宽度比例是很难进球的。

球员和球都使用圆圈来表示。动作模型是离散的(在一个仿真周期结束时全部的动作被执行一次)。每个仿真周期时间的长短是由参数 *simulator_step*[*] 决定的。在每个仿真周期结束前，SoccerServer 接收所有 Client 的命令，并执行命令，利用当前场上对象(球员和球)的位置和速度信息计算出全部对象新的位置和速度信息。

在仿真周期内，对象的移动按如下公式进行计算：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) : accelerate$$

$$(p_x^{t+1}, p_y^{t+1}) = (p_x^t, p_y^t) + (u_x^{t+1}, u_y^{t+1}) : move$$

$$(v_x^{t+1}, v_y^{t+1}) = decay \times (u_x^{t+1}, u_y^{t+1}) : decayspeed$$

[*] 斜体表示 SoccerServer 的参数。

(a_x^{t+1}, a_y^{t+1}) = (0,0)：*resetacceleration*

其中，(p_x^t, p_y^t) 和 (v_x^t, v_y^t) 分别表示 *t* 时刻物体的位置和速度。*Decay* 是一个参数，分别由 *ball-decay* 和 *player-decay* 控制。(a_x^t, a_y^t) 表示对象的加速度，可以通过 ***dash****（针对队员）和 ***kick***（针对足球）的 *Power* 参数计算得到：

$$(a_x^t, a_y^t) = Power \times power\_rate \times (\cos(\theta^t), \sin(\theta^t))$$

其中 θ^t 表示对象在 *t* 时刻的前进方向，*power_rate* 就是 *dash_power_rate* 或者是 *kick_power_rate*。如果对象为球员，它的方向就是球员脸朝向的方向。

对于足球，其方向的计算方法是：

$$\theta_{ball}^t = \theta_{kick}^t + Direction$$

其中 θ_{ball}^t 和 θ_{kick}^t 表示球和踢球队员当前的方向，而 *Direction* 是 ***kick*** 命令中第二个参数。

2.2.2 环境干扰

为了反映出实际比赛中球及球员运动的不确定性，SoccerServer 在球及球员的移动和转身过程中加入了一定的干扰因素。

首先考虑移动，干扰是以如下方式加入的：

$$(u_x^{t+1}, u_y^{t+1}) = (v_x^t, v_y^t) + (a_x^t, a_y^t) + (\tilde{r}_{r\max}, \tilde{r}_{r\max})$$

$\tilde{r}_{r\max}$ 为属于 $[-r\max, r\max]$ 的随机数。*rmax* 的定义为：

$$rmax = rand \cdot \left|(v_x^t, v_y^t)\right|$$

rand 是 *player-rand* 和 *ball-rand* 的参数。

命令中的 *Moment* 和 *Power* 参数的干扰为：（用 argument 表示）

$$\arg ument = (1 + \tilde{r}_{rand}) \cdot \arg ument$$

2.2.3 球员的体力模型

每个球员都有自己的体力值。SoccerServer 通过限制球员的体力来阻止队员始终以最大速度（*player_sp_max*）跑动。球员的体力模型包含三个方面：

1) *stamina* (∈ [0，*stamina_max*])，表示球员的体力，它限制 ***dash*** 命令的 *Power* 参数。
2) *effort* (∈ [*effort_min*，1.0]) 表示了球员体力的使用效率。
3) *recovery* (∈ [*recovery_min*，1.0]) 控制体力的恢复速率。

*黑色斜体表示 SoccerServer 定义的球员命令。

具体的计算方法为：

- 当一个球员使用(***dash*** *Power*)命令时，他的 *Power* 参数要受到 *stamina* 和 *effort* 的影响：

  effective_dash_power = Min(stamina, Power) × effort

  stamina = stamina – effective_dash_power

  由上式可知，*effort* 的大小决定了 *stamina* 中可以有效使用的部分。

- 在每个循环周期内，如果 *stamina* 低于 *effort_dec_thr* 时，*effort* 减少，*stamina* 高于 *effort_inc_thr* 时，*effort* 增加：

  if *stamina* ≤ *effort_dec_thr* × *stamina_max* and *effort* >*effort_min* then

  effort = effort - effort_dec

  if stamina ≥ effort_inc_thr × stamina_max and effort < 1.0 then

  effort = effort + effort_inc

  此公式表示体力的使用效率正比于体力值的变化。

- 在每个循环周期内，如果 stamina 低于阈值 recover_dec_thr 时，recovery 减少：

  if stamina ≤ recover_dec_thr × stamina_max and recovery > recover_min then

  recovery = recovery - recover_dec

- 在每个循环周期内，stamina 会得到一定程度的恢复。恢复时，依 recovery 的当前值进行：

  stamina = stamina + recovery × stamina_inc

  if stamina > stamina_max, then stamina = stamina_max

2.2.4 球员的感知信息

球员从 SoccerServer 接受的感知信息包括听觉和视觉信息。对于球员来说，这些信息是非常重要的。本节详细论述感知信息的格式和使用方法。

1．听觉信息

当某球员或裁判（referee）发送消息（***say*** *Message*）时，附近的其他球员包括对方球员可以立即听到消息，没有延迟。他们从 Server 接收到（***hear*** *Time*

Sender Message）格式的消息。其中：

1. *Time*：当前的仿真周期。
2. *Sender* 如果是其他球员发送的消息，那么是发送者的相对方向，否则就是下面的可能选项：
   self：发送者是球员本人。
   referee：发送者是裁判。
   online_coach_left 或者 *online_coach_ringt*：发送者是在线教练。
3. 如是裁判发的消息，*Sender* 为“ *referee* ”，且可能的 *Message* 为：*before_kick_off*、*kick_off_l*、*kick_off_r*、*kick_in_l*、*kick_in_r*、*corner_kick_l*、*corner_kick_r*、*goal_kick_l*、*goal_kick_r*、*free_kick_l*、*free_kik_r*、*offside_l*、*offside_r*、*play_on*、*half_time*、*time_up*、*extend*、*foul_Side_Unum*、*goal_Side_Point*。
4. *Message* 代表消息的内容，最长 *say_msg_size* 个字节。

队员仅有有限的通讯能力，只能听到一定距离之内的声音，此距离由 SoccerServer 参数 *audio_cut_off_dist* 决定。同时队员在 *hear_decay* 个循环周期内只能听到 *hear_inc* 条消息。一般情况下，在 2 个循环周期内，当多名队员同时发送多个某消息时，一名队员只能接收一条，而丢失了其它的消息，被接收到的消息是不确定的(现在是根据消息到达的顺序来选择)。裁判（referee）所发的消息具有最高的优先级，可以被全部队员接收到。为了避免球队阻塞信道使对手的交流失效，我们将球员的两支球队的听觉能力分离，也就是说，每个球员在每两个仿真周期内能从每支球队接收到一条消息。

2．视觉信息

球员定期从 SoccerServer 得到视觉信息，视觉信息按如下格式定义：

（***see*** *Time ObjInfo ObjInfo* **…**）

Time 指示当前时间。

ObjInfo 表示了可视对象的信息。其格式为：

（*ObjName Distance Direction DistChng DirChng BodyDir HeadDir*）

ObjName = （*p Teamname UniformNumber goalier*）
| （*g* [*l*|*r*]）
| （*b*）
| （*f c*）

| (*f* [*l* | *c* | *r*] [*t* | *b*])
| (*f p* [*l* | *r*] [*t* | *c* | *b*])
| (*f g* [*l* | *r*] [*t* | *b*])
| (*f* [*t* | *b*] [*l* | *r*] [10 | 20 | 30 | 40 |50])
| (*f* [*l* | *r*] [*t* | *b*] [10 | 20 | 30])
| (*l* [*l* | *r* | *t* | *b*])

Distance，*Direction* 表示目标的相对距离和相对方向。*DistChng* 和 *DirChng* 分别表示目标距离和方向的相对变化，*DistChng* 和 *DirChng* 不是精确值，只是一个粗略值。当被观察的目标是其他队员时，参数中增加 *BodyDir* 和 *HeadDir*，分别是被观察球员相对观察者的身体和头部的相对角度。如果两个球员的身体都是相同的角度，那么 *BodyDir* 就等于零。*HeadDir* 也一样。字母：“ *l r c t b* ”分别表示了左，右，中心，上，下。“ *p* ”表示罚球区。详见下图。

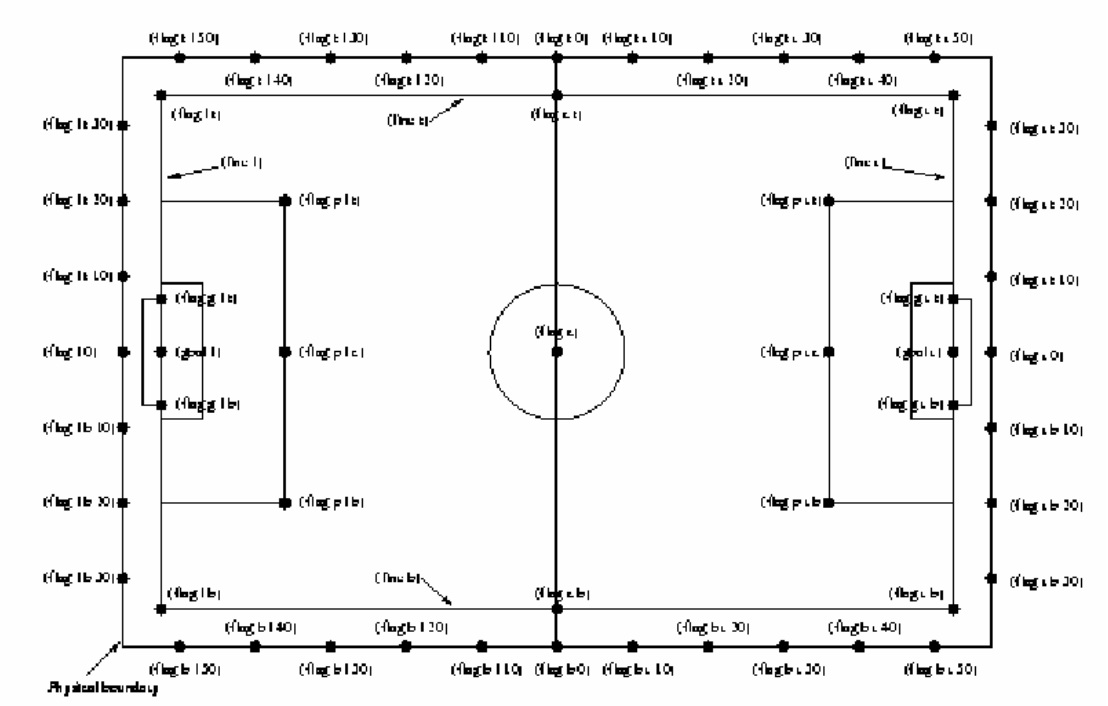


图 2 比赛场地标志

Distance，*Direction*，*DistChng* 和 *DirChng* 的计算方法为：

$$P_{rx} = P_{xt} - P_{x0} \qquad P_{ry} = P_{yt} - P_{y0}$$

$$v_{rx} = v_{xt} - P_{x0} \qquad v_{ry} = v_{yt} - P_{y0}$$

$$Distance = \sqrt{P_{rx}^2 + P_{ry}^2}$$

$$Direction = \arctan(P_{ry} / P_{rx}) - a_0$$

$$e_{rx} = P_{rx} / Distance \qquad e_{ry} = P_{ry} / \text{Distance}$$

$$DistChng = (v_{rx} * e_{rx}) + (v_{ry} * e_{ry})$$

$$DirChng = [(-(v_{rx} * e_{rx}) + (v_{ry} * e_{ry})) / Distance] * (180 / \pi)$$

其中(P_{xt}, P_{yt})是目标的绝对位置坐标，(P_{x0}, P_{y0})是接收视觉信息的队员自己本身的绝对坐标，(v_{xt}, v_{yt})是目标的绝对速度，(v_{x0}, v_{y0})队员自己的绝对速度。a_0是队员所面向的绝对方向。另外(P_{rx}, P_{ry})和(v_{rx}, v_{ry})表示目标的相对位置和相对速度。(e_{rx}, e_{ry})表示平行于相对位置向量的单位向量。

球员的视觉信息由下面几个方面决定：

- 视野宽度。正常模式为[-45，45]，宽模式为[-90，90]，窄模式为[-22.5，22.5]。
- 邻域。距离自己 3 米之内。当某个对象在球员的邻域内但在视野之外时，球员只能知道对象的类型（球，其他队员，球门或标志），不知道对象的准确名字。
- 远处目标信息的不确定性。无论远处的目标是球还是球员，目标的距离值按如下方式进行量化：

$$d' = Quantize(\exp(Quantize(\log(d), quantize\_step)), 0.1)$$

其中d, d'分别表示精确距离和相应的量化距离。且：

$$Quantize(V, Q) = \text{ceiling}\ (V / Q)\ * Q$$

这表示队员是不能知道远处物体的精确位置的。例如：当距离为 100.0 时，最大噪声可达到 10.0，但当距离在 10.0 之内时，噪声小于 1.0。

对于远处目标是旗或线的情况，距离值按如下公式量化：

$$d' = Quantize(\exp(Quantize(\log(d), quantize\_stepp\_l)), 0.1)$$

观察模式决定了观察频率和从 SoccerServer 所接受视觉信息的细节。观察模式的包含两个参数：*ANGLE_WIDTH* 和 *QUALITY*。*ANGLE_WIDTH* 为 *wide*（=180）、*normal*（=90）和 *narrow*（45）之一。*QUALITY* 为 *high* 或 *low*。当 *QUALITY* 设置为 *high* 时，SoccerServer 为观察者发送详细的目标位置信息。当 *QUALITY* 设置为 *low* 时，Server 为观察者发送简化的目标信息（只有目标的方向）。另一方面，Server 为队员发送视觉信息的频率随着 *ANGLE_WIDTH* 和 *QUALITY* 而变化。缺省时间间隔为 150 毫秒（由 Server 参数 *send_step* 控制）。当观察角变宽时，频率加倍，反之减半。

请参考图 3 的标志，球员所获得的视觉信息和距离的相关程度很大。对于近距离球员，它既可以看到它所属的球队同时也可以看到它的球员号码。然而，随着距离的增加，首先消失的是球员的号码。然后距离的增加还会导致球员的队别也分不清楚了。在服务器端假定这些距离是

unum_far_length ≤ unum_too_far_length ≤ team_far_length ≤ team_too_far_length

这里假定 dist 是球员和自己的距离，那么：

- 如果 $\text{dist} \le unum\_far\_length$,那么球员号码和球队名称都可见
- 如果 $\text{unum\_far\_length} < \text{dist} \le unum\_too\_far\_length$，那么队名是可见的，但是队员号码有一定的概率看不到。这个概率根据 dist 是线性的从 1 到 0 减少的
- 如果 $\text{dist} \ge unum\_too\_far\_length$,那么球员号码是不可见的。
- 如果 $\text{team\_far\_length} < \text{dist} \le team\_too\_far\_length$，那么队名也存在一定的概率不可见，概率是随着 dist 的减少从 1 到 0 线性的递减的。
- 如果 $\text{dist} \ge \text{team\_too\_far\_length}$,那么队名是不可见的

举例说明：根据图 3，假想根据所有的球员周围的环，对于球员 c 可以识别球队名称和号码，球员 d 可以识别队名，而且有大概 50%的机会识别出号码；球员 e 则有 25%的机会识别出队名。对于球员 f 恐怕只能识别为一个匿名的球员了。

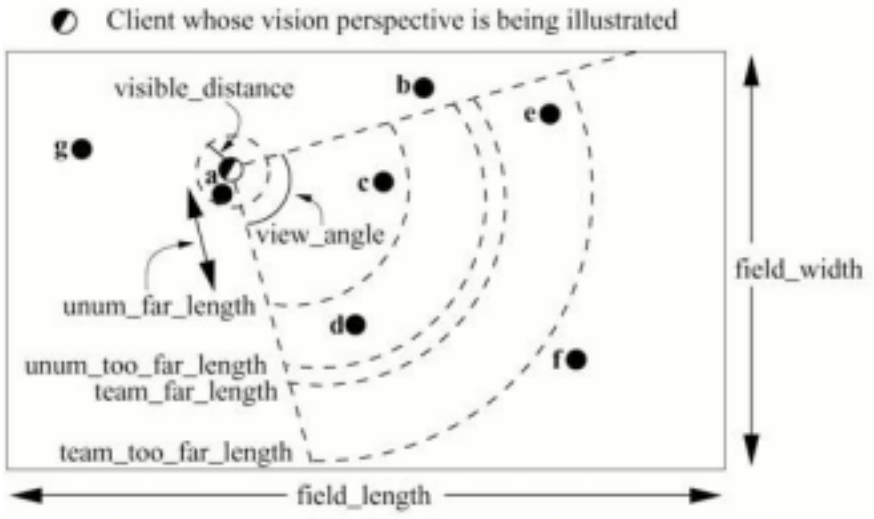


图 3 球员视觉模型

2.3 特点

仿真比赛环境的特点主要有：

1 动态实时系统。比赛每个仿真周期为 100ms，要求每个 Agent 在此时间内完成全部计算并将要执行的命令发送给 SoccerServer，否则将失去本次执行的机会。

2 环境干扰。由于 SoccerServer 的影响，每个 Agent 不能准确地感知环境，同时不能精确地改变环境。

3 合作与协调。全部 Agent 具有一个共同的目标，是一个分布式的多智能体系统。

4 受限的通讯带宽。系统不允许 Agent 之间直接进行信息交换，全部通讯必须由 SoccerServer 控制。在一定的仿真周期内，只有有限的消息得到传递。

2.4 小结

本章详细介绍了 RoboCup 软件仿真比赛的 SoccerServer 仿真环境。主要包括：SoccerServer 的仿真速度模型、视觉模型、物体模型、听觉模型、体力模型等。总结了 SoccerServer 环境的特点：动态实时系统、环境干扰、合作与协调、受限的通讯带宽。

第三章　Agent 框架概述

3.1　开发环境

如前所述，仿真比赛环境中 Server 和 Client 是通过 UDP/IP 协议进行通信的。SoccerServer 并没有对 Client 的开发和运行环境提出任何限制，只要支持 UDP/IP 协议即可，因此在开发环境和使用语言上可以有多种选择。由于 Linux 的网络功能和实时处理功能相当强大，我们选择 Linux 平台作为我们的 Client 运行平台，开发工具使用 KDevelop。我们用一台 Linux(RedHat 7.1)服务器运行 Server 端程序。开发的网络环境是自己组建的一个小型的局域网，机器连在同一个集线器上。

3.2　程序框架

可以按照 Agent 的智能层次划分，也可以按照程序的结构划分框架。

3.2.1　根据智能层次划分

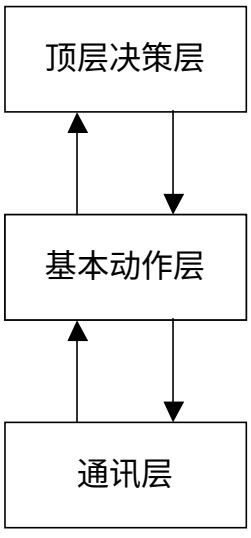


图 4 Agent 层结构

按照智能层次的划分，Agent 结构可以分为三层：通讯层、基本动作层、顶层决策层。如图 4 所示。

通讯层是整个结构的最低层，直接负责与 SoccerServer 进行通讯，发送动作指令到 SoccerServer，并从 Server 得到视觉信息、身体感知信息和听觉信息，并且处理这些信息更新智能体内部维护的世界模型。基本动作层定义了全部 Client 的个人动作，这些动作属于复合动作，即需要多个周期才能作完。顶层决策层是智能水平最高的，负责根据比赛情况进行实时动作评价并做出决策。在基本动作层与顶层决策层之间还缺少一个模式层，来定义一些有效的配合模式，由于时间的关系，目前没有实现。

3.2.1.1 通讯层

通讯层的主要技术难点包括时间片的控制、自身时钟与 SoccerServer 时钟的同步、听觉信息以及视觉信息的处理。

- 同步问题的处理

client 和 server 之间的同步是非常重要的，也非常复杂。server 每周期结束之前，都要求 client 发送本周期末要执行的命令，如果没有收到，则 client 就失去了本次机会。所以如果 client 时钟与 server 时钟不同步，必将导致 client 不能在合适的时间将执行命令发送到 server，影响动作执行。由于 server 按照自己的时钟运行，而 client 按照自己的时钟运行，并不知道 server 的情况，只有 server 给 client 发送消息时，会包含自身当前的周期数。所以 client 必须根据这些有限的信息与 server 进行同步。

- server 信息的不协调

server 并不保证 client 接到信息的每个瞬间，此信息是和该瞬间匹配的。而只能保证在每个周期结束时该信息的可信度。例如，身体动作(turn，dash，kick，catch)在每个模拟周期末尾产生作用。

这就导致了很多问题，其中包括球员的角度问题。比赛过程中希望在球员转身（发出 turn 的命令）的同时更新球员的视觉信息。这样即使没有收到 server 的视觉信息，我们仍然可以知道周围的环境。这就导致了很多问题，其中包括球员的角度问题。比赛过程中希望在球员转身（发出 turn 的命令）的同时更新球员的视觉信息。这样即使没有收到 server 的视觉信息，我们仍然可以知道周围的环境。然而，需要小心的是接下去从 server 接受到的视觉信息是否已经包含了这个 turn 指令。否则，球员可能错误计算。判断 server 是否执行了 turn 指令只能通过判断当前 server 传送的视觉信息和 turn 指令发送之前的相似还是和 turn 执行之后的环境信息相似。

- 跟踪丢失的命令

由于网络环境和运行环境的原因，server 有时候会丢失指令。client 可以通过比较 server 发出的信息和本身记录的信息得知，即 server 在 sense_body 信息中给出的已经接收、执行的 dashes，catches，turns，kicks 和 client 发出的数目不同。在解决了前面的同步问题之后，我们发现确实还存在其他的因素导致丢失指令——网络问题。我们只是在一个很小的局域网内开发，100M 的以太网，程序运行的时候网络内并无其他流量，但是当两边各开 22 个球员的时候仍

然造成网络阻塞。如果 client 按照规定编写，同时符合 RoboCup 的君子约定(其中一条是不故意阻塞 server)，那么局域网应该是可以承受的，满负荷大概有接近 2M 的信息。我们分析产生这个问题的主要原因与 SoccerServer 的 I/O 处理能力不强有关系。解决方案是用网线直连我们的 client 和 server 计算机，然后把 I/O 开销比较大的对手(如 CMUnited-99，1999 年 RoboCup 仿真组的世界冠军)程序运行在 server 的机器上，不通过 I/O，而通过 Linux 的 loopback 设备直接与 server 通讯。这样只有我们的 client 在网络上传输数据。通过测试效果比较好。server 基本不丢失命令。

- 记录视觉信息

因为 client 的视角有限，场上的物体只能被某些 client 看到。虽然可移动物体在离开 client 视野范围内的时候往往会改变位置，但是他们在每个周期内的移动距离是有限制的。因此，agents 需要“记忆”这些移动物体的位置，但是随着时间的增长，他们的位置信息的可信度不断的降低。例如球的位置置信度，在看到球的时候为 1，然后每周期衰减 0.05。根据球当时的速度同时更新球的位置。每个移动物体都有这个参数，首先设定初值（初始值亦随距离的增加而递减），然后随时间递减。当这个变量达到某极限的时候我们认为这个物体位置已经不可信了，例如 0.7。

因为球的速度比球员快得多，所以球的位置置信度的衰减速度也比球员的大。然而，当球员转向球的相反方向并移动到其他位置的时候，他也不能完全忘记球的位置。

- “幻影物体”

因为我们对于物体的记忆效应，所以导致一个经常所犯的错误，即错误认为某物体在某点。我们称之为“幻影物体”。显然，产生问题的原因是当我们每次把视觉信息更新的时候，那些没有看到的物体仍然因为位置置信度保留在记忆中。但是可能已经产生了某些突变，那个物体已经远离改点。因为球速最快，这种情况经常发生在球身上。例如，球员的记忆中，球的位置是在前方的，但是他并没有看到球。这里其实应该更新的是把球的置信度降为零，但是事实上很难做到这点，这需要大量的计算。这时候可能球已经在球员的侧面或者后面了。球员仍然认为球在那个错误的位置，并且可能跑向那里。所应该做的是根据视觉信息更新那些显然的错误，我们对足球等重要的物体进行判断，省略对其他一些物体的判断。

- 听觉信息的处理

队员仅有有限的通讯能力，只能听到一定距离之内的声音，此距离由 SoccerServer 参数 *audio_cut_off_dist* 决定。同时队员在 *hear_decay* 个循环周期内只能听到 *hear_inc* 条消息。为了避免球队阻塞信道使对手的交流失效，我们将球员的两支球队的听觉能力分离。一般情况下，在 2 个循环周期内，当多名队员同时发送多个消息时，可以从两支球队分别接收一条，而丢失了其它的消息。裁判（referee）所发的消息具有最高的优先级，可以被全部队员接收到。这样就可能出现一种情况，你所说的话被对方球员恶意记录，并在几个周期后重复，来扰乱我方决策。针对这个情况，我们设想对所说的话根据周期数进行加密。当收到听觉信息以后首先进行解密，如果和周期数不符合，则直接丢弃。由于时间关系，我们本次还没有实现。

3.2.1.2 基本动作层

基本动作层，也可以说是基本技术层。这里实现了球员的全部个人技术。球队的一切都是在这个基础上实现的。主要包括跑位、射门、转球、带球、守门员技术、断球、防守、破坏、盯人等个人技术。

3.2.1.3 顶层决策层

顶层决策层负责进行球队的整体策略。阵型的确定、攻防的转换、球员的跑位与球员之间的配合。基于场上形势的动态策略站位，基于场上形势的动作选择，射门还是传球？破坏还是转移？带球突破还是回传等待时机？这些问题都在顶层决策层解决。

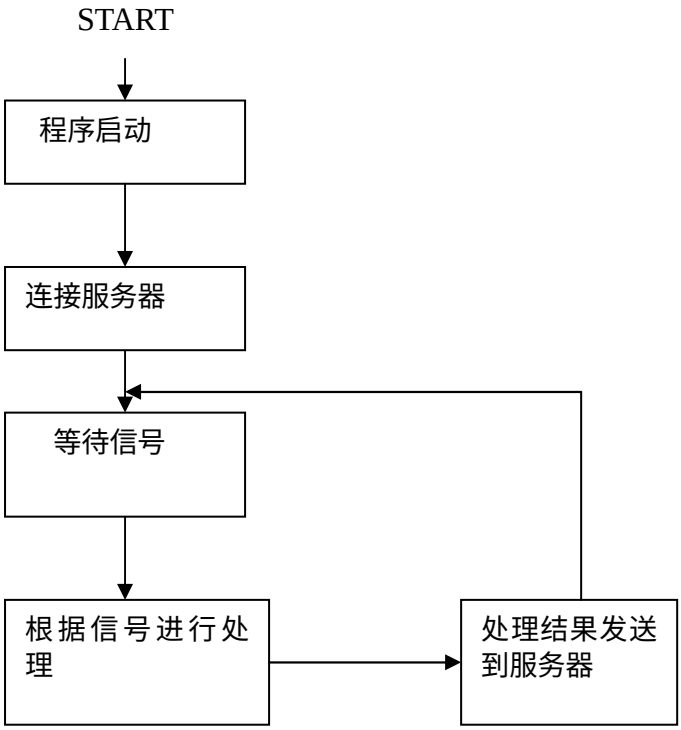
3.2.2 根据程序结构划分程序框架

程序结构上我们采用了和 CMUnited-99（99 年软件仿真组的世界冠军）相似的程序结构。

程序的主体采用时钟触发控制，主要有两个处理函数：**Information Procedure** 和 **AI Procedure**，分别处理从 Server 端来的消息以及进行 AI 计算。他们分别以间隔 InfInterval 和 AIInterval 被调用。

一、程序启动部分

1. GetOption 从命令行读取参数，并且支持从文件中读取 Options，只要



程序流程图

在命令行中写上 –file *filename* 就可以了。具体的参数格式见源码函数。

2. Intialize 完成程序运行前一些必要的初始化工作，并且开启时钟触发函数。

二、循环部分

Information Procedure 和 AI Procedure 被触发后就与主程序并行执行了，主程序进入等待状态，等待结束信号。目前用的两个信号为 ServerAlive(是否还和 Server 保持联系)，和 breakoutside()，是否有外部的终止信号。因为程序在 server 结束后会自动终止，所以没有实现外部的终止信号。

三、程序结束

destruction 作必要析构工作。

这里说明一下 Information Procedure 和 AI Procedure。程序中开两个线程分别运行。如果 Information Procedure 收到 Server 的信息，它将更新自己的信息库。同时，AI Procedure 做动作以后，他也要更新 Information 的信息库。AI Procedure 可以有两种情况激活：Information 收到 Server 的 Sense_body 信息或者时钟激发。

因为 server 的 sense_body 是固定的美周期发送一次。如果 sense_body 激发了 AI Procedure，则重置时钟。对于 AI Procedure 的信息库的更新，因为 2 个线程可能都要对它进行操作，所以需要类似原语的功能进行加锁互斥。即 AI Procedure 访问的时候 Information Procedure 不能对其进行写操作，反之亦然。

针对 server 的模型，我们定义了一系列类的结构，用以反映模型中所定义的各种物体。设计思想：所有的物体继承一个 Object 类。包括球，球员，场上固定的物体等等。具体的结构见图 5。

这种类结构的设计使得很多代码可以重复利用。是一种很好的结构化设计。每级类有自己的属性，高级类继承低级类的属性。Object 的属性有简单的位置，与另一 Object 的距离，位置是否合法等等。Stationary Object 是指场上为了识别队员自己的位置而人为设定的标志（FLAG）。它们的属性包含了初始化函数以及一些为识别方便设定的角度信息。Mobile Object 指场上的一切移动物体。

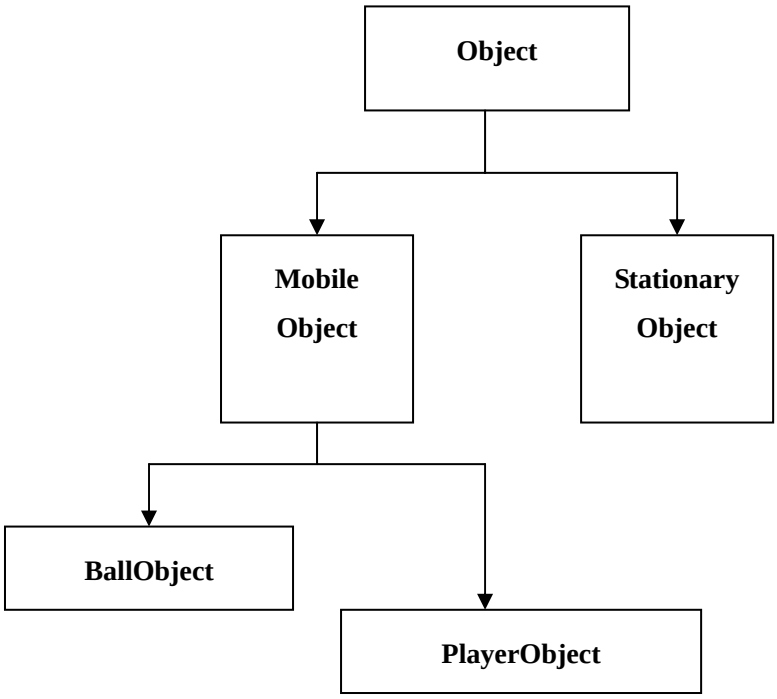


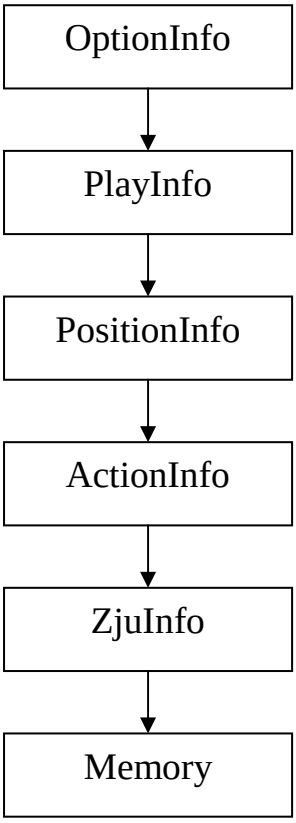
图 5 .Object 及其派生类

例如球和球员。其中主要设置了关于速度的各个功能。例如取得相对速度，绝对速度等等属性。具体到 BallObject 设定的是关于场上球与球员有关的信息了，例如 kickable()，catchable()，moving()等等。PlayerObject 则包含具体球员的作为 Object 信息速度，方向。信息的来源是每个周期 server 发送的信息，例如

recv 52237 : (see 0 ((g r) 70.8 20) ((f c t) 16.9 –35 0 0) ((f r t) 67.4 –8) ((f r b) 88.2 41)……

同时设计了一些公用的关于几何计算的类。定义了 Line，Vector，Rectangle 等类。包括一些向量、点、线、矩形的关系的几何计算。例如 PointOnLine()，LineFromToPoints()，LineFromRay()等等。

基于知识的表述，我们设定了 Info 系列类。



OptionInfo 类包括了很多参数初始全局信息，主要有 SP（Server Params 如禁区长度） CP（Client Params 如我方队名） FP（Formation Params 如刚开始采用的阵型） VP（Version Params 版本测试用） IP（Initialize Params，如初始得分等）。PlayerInfo 类继承了 OptionInfo ，还包含很多与 Client 的位置、速度、耐力值等有关的参数和函数。PositionInfo 类继承了 PlayerInfo，并且给出了队友、对手和球的信息参数和函数。ActionInfo 类继承了 PositionInfo，增加了有关断球的信息及函数。ZjuInfo 类继承了 ActionInfo，高层决策的函数大部分都在这个类中实现。Memory 类则继承了 ZjuInfo，包含了整个信息库并

且在中间调用了场上对象（Object 系列类），作为一个完整的场上知识的表述。

基于足球场上的阵型，场地情况，比赛中的中断，程序中还包括场地（Field），阵型（Formation），任意球（Set_Play）。

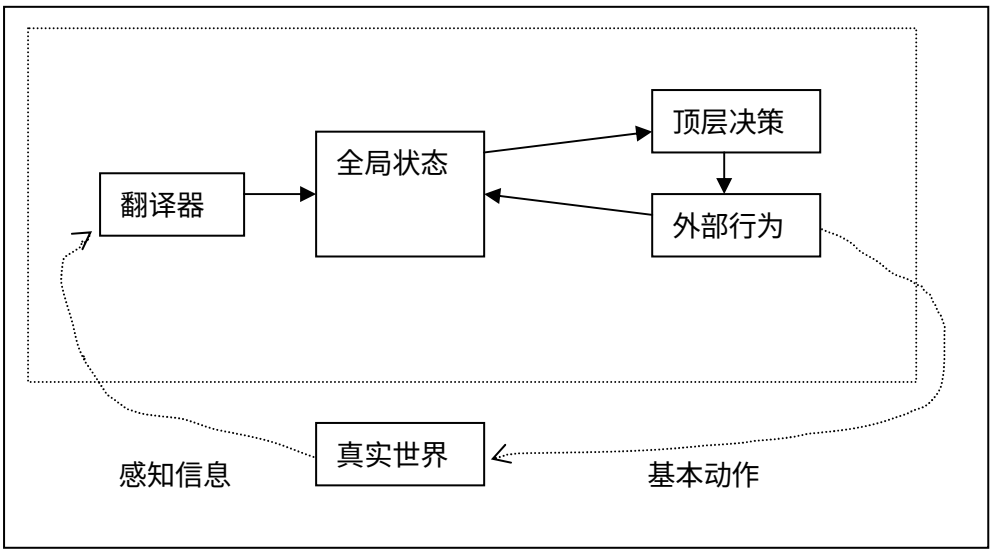
3.3 小结

本章介绍了仿真足球机器人的开发环境和程序框架。程序框架部分介绍了通讯层，基本动作层，和决策层。同时介绍了程序的结构框架和具体的设计。

第四章　通讯及信息处理

前面已经简单的介绍了通讯层部分，主要集中在 Server 和 Client 之间的信息同步匹配中，同时也分析了由于网络本身的物理原因造成的信息丢失问题，并且提出了相应的解决方法。本章主要分析智能体获得信息后，怎么处理这些信息，更新智能体内部维持的世界模型。

4.1　球员体系结构



球员体系结构图

全局状态反应了智能体对真实世界的知识表述，通过它的感知和动作预测结果构成。它由感知信息的处理结果更新，也可由预测外部行为模块所选择的动作结果更新。外部行为参考顶层决策向执行器发送基本动作命令。这个动作作用于真实世界，从而改变智能体将来的感知。

4.2　智能体基本循环

仿真平台(Server)在每个仿真周期的末尾更新当前全局状态，因此一个智能体可以通过确定上一周期结束时的全局状态来选择本周期应该采取的动作。

因此，在仿真周期 t，基本的智能体循环如下：

1. 假设智能体已有在 t-2 周期结束时的全局状态信息，并且已经在 t-1 周期时发送了一个动作。
2. 当服务器仍在 t-1 周期时，接收服务器所发送的感知信息（see, hear, sense_body），将这些新的信息存放在一个临时结构中。并不更新当

前的状态。

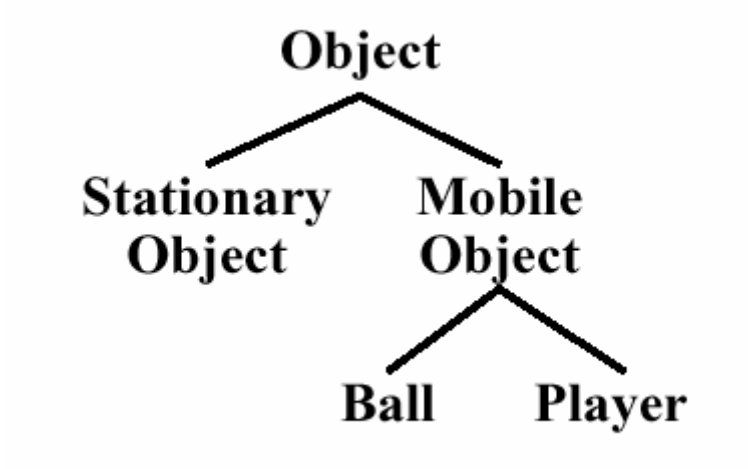
3. 当服务器进入周期 t（通过所运行的时钟确定或通过所接收到的感知信息中所包含的时间信息 time stamp），通过所有有用信息（临时结构中的信息及所预测的上一次动作的结果）来**更新全局模型**以匹配服务器端 t-1 周期结束时的全局状态。然后为周期 t **选择一个动作，发送**。
4. t+1 周期重复

4.3 预言性记忆

球员发送动作的选择是由顶层决策确定的，而顶层决策则完全是由球员内部维持的全局状态决定。当我们需要裁决采用哪个动作时，很有必要获取其时的比较精确的全局状态。针对 RoboCup 获取信息不完整的特点，我们使用了预言性的世界模型。当所获得的感知信息中缺少某些对象的信息时，根据以前的观察结果和行为预测来确定当前对象信息。

4.3.1 对象描述

球场上的对象分为静止对象和运动对象。静止对象包括球门和场上的标记，可以用来定位。运动对象则是智能体自身、足球、和其他 21 个球员（10 个队友和 11 个对方球员）。这些对象的层次类型如下图所示。



对象层次结构

每个智能体的世界模型中都存储着场上所有静止和运动对象的实例。包括所有的球门、边线、场地标志；球的实例，21 个球员对象（这些对象可应因为视觉的原因没有标明所属队或/和球员号）。

因为场上的对象大部分情况下都是不可见的，没有一个对象能够被连续的看到，所以对移动对象设置了一个可信度，取值范围[0, 1]，标识他们所处位置的

可信度、速度的可信度。在没有看到对象时，要随时间的增加相应的降低该对象的可信度，并且更加可信度的大小来确定是否要使用这个位置或速度值。

在所获取的感知信息和所发送的动作指令中，都使以该球员当前所处位置为原点的相对坐标。但在全局模型中，所有的信息以全局的绝对坐标存储。这样，当智能体移动时，固定对象可以不做修改。虽然相对坐标发生了变化，但是只要知道本身的位置，很容易就可以在相对坐标和绝对坐标之间进行转换。

对象的相关变量见下所示：

Object:

全局位置坐标(x , y)
位置的可信度
固定对象 Stationary Object：无其他变量增加
移动对象 Mobile Object：
全局速度坐标(dx , dy)
速度的可信度
球 Ball：无其他变量增加
球员 Player：
Team
Uniform Number
身体的角度
该角度的可信度
头部的角度
该角度的可信度

4.3.2 智能体世界模型的更新

更新世界模型的信息来自：

———视觉信息

———听觉信息

———自身感知信息

———上一个动作的预测结果

视觉信息以相对距离和相对角度的形式出现。听觉信息还可以包括队友发出的对象的绝对坐标信息。自身感知信息则是球员本身的状态，包括了体力、视野模式和速度等。

一旦有新的信息到达时，它与时间戳和置信度一起首先被存储在个临时结构里面。视觉信息的可信度最大为 1，随相对距离的增加而递减，听觉信息的可信

度相对要小。视觉信息将首先以相对坐标的形式存储，直到本智能体的确切位置确定后再转换成绝对坐标。

当进入周期 t 时，利用所有可用信息来确定 t-1 周期结束时服务器端的状态。如果对于指定的对象没有收到新的信息，则通过速度和所执行的动作来预测对象的新位置，对象的位置和速度的可信度都将降低。

当更新智能体的全局模型时，首先更新智能体本身的位置，然后更新球和其他球员。

■ 智能体本身

由于视觉信息是以相对坐标表示的，那么首先要确定对应视觉信息时刻的智能体的绝对坐标。在我们将智能体世界模型更新到仿真周期 t－1 末尾时，可能会有时间戳是 t－1 和（或）时间戳是 t 的视觉信息（如果有更早的视觉信息，则不予采用，可以认为这些因素已经在上一次的模型更新中被包括了）。

如果最近的视觉信息的时间戳是 t-1，则智能体本身的位置不做更新，直到其他对象更新完毕（因为他们的坐标是相对于本智能体原来的位置）。另一方面，如果最近的视觉信息的时间戳是 t，或自上一次更新后没有收到新的视觉信息，则智能体本身的状态可以立即更新。

在以上两种情况，更新智能体状态的过程如下：

（1） 如果收到新的视觉信息

智能体的位置可以通过他的最近的可见边线和最近的固定对象的相对坐标来精确地确定。

（2） 如果没有收到新的视觉信息

根据上一周期的动作（dash）进行预测，更新速度

利用原来的位置和速度，来计算新的位置和速度

（3） 如果可以，根据自身感知信息重新设置智能体的速度。假设速度方向为头部的绝对方向

（4） 根据自身感知信息或动作结果预测更新体力模型

■ 足球

在智能体的决策过程中，足球的位置和速度对动作的选择和一个相当关键的因素，占了较大的比重。因此，要尽可能的获得关于足球精确的及时的信息。

足球的更新过程如下：

（1） 如果有新的视觉信息，利用上述步骤确定的智能体的绝对位置和球相对于智能体的相对位置来确定球的绝对位置。

（2） 如果同时也给出了速度信息，则更新球的速度。否则，通过比较当前球的位置和所预测的球的位置来检查原来的速度是否正确。

（3） 如果没有收到新的视觉信息或所收到的视觉信息是周期 t-1 的，利用 t-1 的信息来估计周期 t 时球的位置和速度。如果在上一周期智能体踢球，将考虑足球的运动

（4） 如果足球应该在视野范围内（预测的位置在球员视线区内），而实际没有（所收到的视觉信息中没有包括球的信息），将足球的位置可信度设置为 0。

（5） 有关球的信息也可能来自队友。如果有听觉信息可以增加当前球的位置或速度的可信度，那么就使用这些更加正确的信息。队友信息的可信度将取决于时间信息（队友看到球的时间是否更近）和队友离球的距离（因为球员离足球越近，他得到的信息也越精确）。

当决定是否要截球，以及是否要踢球时，球的速度是非常重要的。而在视觉信息中通常不给出球的速度。因此当需要得到足球速度值时，可以通过原来的位置和当前的位置来计算球的速度。

■ 其他球员

由于在视觉信息中经常会因为距离原因不标明所看到的球员，因此很难跟踪一个球员的位置。有一种意见是忽略所有不确定的球员，然而，在进行策略规划时，维持球员位置的全景图是非常有必要的。

通常情况下，球员的位置和速度的确定和球所采用的方法一样。所额外增加的是视觉信息中所提供的球员的面向。

在不能获知球员的全部信息时，可用该球员的原先位置来消除这种不确定性。由于已知一个球员在一个周期内所能够移动的最大距离，因此当所得到的一个球员信息没有标识时，可以通过原来球员的位置来确定它的标识。

因为不同的球员可以清楚地看到球场的不同区域，因此可以充分利用队友之间关于球员位置信息的通讯来实现信息的精确性。

通过所有的球员位置信息，智能体可以确定防守和进攻的越位线。为了避免越位，前锋要在对方防守队员前方活动。为了提高对方防守队员的位置信息精确度，前锋需要周期性的看对方防守队员。

第五章　底层技术

Agent 的个人技术是仿真足球比赛的基础。基本动作层，也可以说是基本技术层。这里实现了球员的个人技术。球队的一切都是在此基础上实现的。重要包括跑位、踢球、转球、带球、守门员技术、断球、防守、破坏等个人技术。这些技术是最基本的，是实现高层的复杂策略所必不可少的。这些动作都是复合动作，每个动作的实现都需要多个周期和多个基本命令（dash, kick）组成。

5.1　PLOS

参考了 CMUnited-99 的特点，我们也设计并实现了智能化的个人技术。各种技术之间的最基本特点是相同的，即“ 预测性的局部最优技术 ”。也就是 PLOS(predictive，locally optimal skills)。 这些动作在时间和空间上都详细分析做动作以后几个周期外部世界环境的变化，并且预计未来几个周期可能的动作，根据分析结果对当前要做的动作进行优化处理。同时根据自身的情况再做一次处理，最后才得到这次真正所要做的最基本动作。一个最简单的例子：智能体的体力 stamina 管理。在服务器端调整球员体力模型时有补偿和不补偿的区别。当球员的体力低于一定的阀值，就只降低体力不再补偿了。每个球员检测自己的体力，确保体力不会低于补偿阀值。当当前所执行的动作需要较大的速度，会使所消耗的体力太大时，它将降低速度而确保体力不会低于补偿阀值。这种行为对于整个队的目标来讲可能不是最优的，但是对于智能体当前的疲惫状态来讲却是局部最优的。

虽然动作是可预计的，Agent 每个周期仍然只提交一个动作给底层机构。当下一个动作周期到来的时候，所有的情况完全重新进行计算。如果周围环境和上周期所预测的十分接近，那么 Agent 将保持上个周期的行为模式，并且继续执行上个周期的技能。然而，如果周围环境和预测有本质的不同，Agent 将重新计算并提交新的动作序列，而不是继续执行上一次计划中的指令。同样，在新的序列中，执行机构仍然只执行第一个动作。

5.2　几何计算和神经网络

对智能体个人技术的确定有两种方法，一是采用几何方法进行准确的计算，还有就是使用前馈神经网络（BP 网）来训练个人技术。采用几何方法计算简单，属于平面几何的处理，而且得到的结果准确，但是由于 RoboCup 是一个不确定的多干扰系统，准确的计算结果在此无用武之地，在刚刚开始 RoboCup 时往往

还采用几何方法，现在则向神经网络训练方向发展。虽然神经网络在处理噪声时相当成功，但是在层和节点的选取时往往依赖于经验，没有一个统一的选取方案。而且，样本空间的选取，也就是神经网络的输入输出变量的确定也需要进一步的研究。总的来说，在积累以往经验的基础上，更倾向于选择神经网络进行训练智能体的个人技术，乃至训练上层决策。

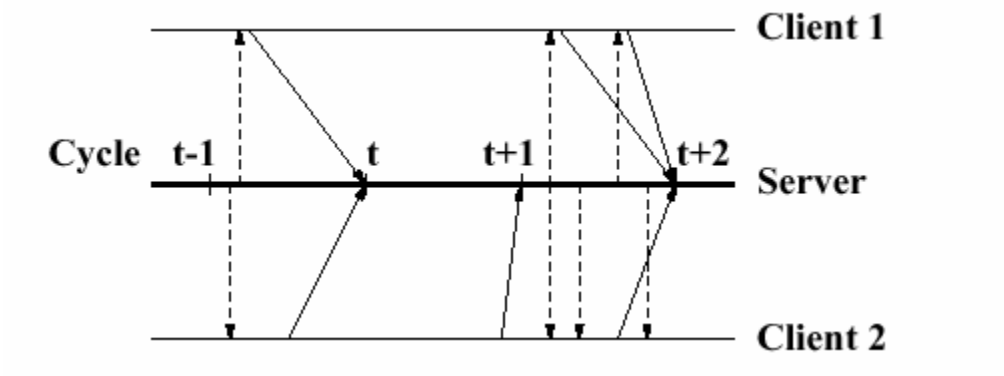
5.3 底层动作的发送策略

目前底层动作方面的研究已经比较成熟，RoboCup 的研究主要集中在上层决策，多智能体之间的协调合作上。本章不再介绍每个底层动作的具体实现，侧重分析发送动作时的时机选择。

下图描述了在三个仿真周期 t－1，t，t＋1 中，Server 和两个 Client 之间的消息交互。中间的粗实线代表 server 维持的时间轴，上下两条水平线分别代表两个独立 client 的时间轴线。由于他们是独立的进程，他们不知道 server 什么时候增加了仿真周期数。从 server 指向 client 的虚线代表该 client 能够获知的感知信息。从 client 指向 server 的实线代表 client 发送的动作指令。在 server 执行指令（server 在仿真周期末执行指令）前，这些动作命令就要被 server 接收到，也就是说在动作发送所处的周期末尾，这些动作指令一定要被 server 接收到。

Client 1 采用的策略是在接收到感知信息后再发送动作指令。在周期 t－1，这种策略工作的很好；但是在周期 t，它没有接收到任何感知信息，那么就浪费了动作机会；在周期 t＋1，发送了两个动作指令，但是只会被执行一个动作。

Client 2 能够在每个周期都发送一个动作指令。在周期 t，必须在没有任何感知信息的情况下发送动作指令；而在周期 t＋1，它又会在发送动作指令之前依次接收到两次感知信息。在理想情况下，client 应该在分析了所有三种感知信息（指视觉感知信息、听觉感知信息和自身感知信息）后，然后才是发送动作指令的最佳时机。但是，client 不可能精确的知道什么时候 server 改变仿真周期数，也不会提前知道什么时候会接收到 server 发送的感知信息。因此，为了在每个仿真周期都可以发送一个动作指令，client 往往在发送动作指令前是没有得到信息量的最大值的。然而，因为每个仿真周期费时很少（*simulator_step* 目前取 100 毫秒），在两个周期之间，世界模型不会出现很大的变化，所以这种方法是可行的。



异步的感知和动作，特别是感知以不确定的间隔发生，对 agent 来说是一个非常有挑战性的领域。既有周期性的发送动作的必要，也需要尽可能快的收集足够的信息，agent 必须找到这两个需要之间的平衡点。

一方面，client 需要在每个周期都发送一个动作指令，在程序中通过定时器进行触发。另一方面，client 在本周期发送的动作指令要能在周期末被 server 执行，以免浪费本周期的动作执行机会。由于对发送动作有了时间上的限制，那么也就间接的限制了智能体进行决策所需要的时间，因为 client 在每个周期内需要做的依次是更新世界模型、高层决策、发送底层动作。

5.4 小结

本章没有介绍 Agent 底层技术的具体实现，主要介绍了程序设计中需要澄清的思想概念。介绍了底层技术设计的原则 PLOS，分析比较了几何方法和神经网络方法来训练个人技术，并且详细的给出了 client 和 server 的 I/O 界面的接口以及他们的时序关系。

第六章　顶层策略

Agent 的顶层策略也就是整个球队的策略。控制全队的进攻、防守、攻防转换等。我们在程序中使用了两棵决策树进行顶层策略，第一棵用来选择阵型、阵线和基本角色的确定；第二棵是决策的关键，即基于场上的动态形势进行动态站位和角色变换。为了提高世界模型的准确性，本章最后介绍了智能观察和智能通讯策略。

6.1　阵型和角色

首先为了设定一个衡量双方球队性能的指标，我们提出了控球度的概念。比较精确的计算方法应该是对应于两支球队的控球时间。但是在 RoboCup 中，这样是较难实现的，这是由球员的视觉模型确定的，球员的视野范围不能覆盖球场的长度，如果足球在对方球门附近，那么我方后卫是难以得到足球相关信息，从而得知是哪支球队控球。为了简化这个问题，我们以足球在敌我半场的位置来确定。只要简单的统计足球的 x 坐标的正负值的时间总数（x<0 说明在己方半场，x>0 说明在对方半场），从而设定控球度。控球度 flag 取三个值，-1，0，1。初始情况下 flag＝0，如果我方性能弱（目前设置为控球时间占小于等于 30%），则 flag＝1；如果对方性能较弱（目前取控球时间占大于等于 70%），则 flag＝－1；如果双方球队性能相差不大（我方控球时间介于 40%到 60%之间），flag＝0；其余 flag 不变。

球场上每支球队都有阵型，每个球员都有自己的角色，RoboCup 也不例外。我们采用了三种阵型，433、334 和 541。比赛刚开始时，采用的阵型是 433 打法，在比赛期间，球队根据比分、剩余时间和球队控球度 flag 来转换阵型。

- 在比赛的大部分时间内，我们主要根据 flag 来转换阵型。
  （1） 如果 flag＝0，阵型不变；
  （2） 如果 flag＝1，切换阵型到 541，采用防守反击打法。
  （3） 如果 flag＝－1，切换阵型到 334，用以加强进攻。
- 在比赛临近结束时，则根据场上比分来确定阵线
  （1） 如果我方落后，切换阵型到 334，意图通过加强进攻，反败为胜。
  （2） 如果我方领先，切换阵型到 541，确保当前的胜利。

角色只有三种，前锋、中场、后卫（对于守门员，有专门的守门员策略）。我们对基本角色的划分非常粗略，角色的动态变换依赖于第二棵决策树的基于场

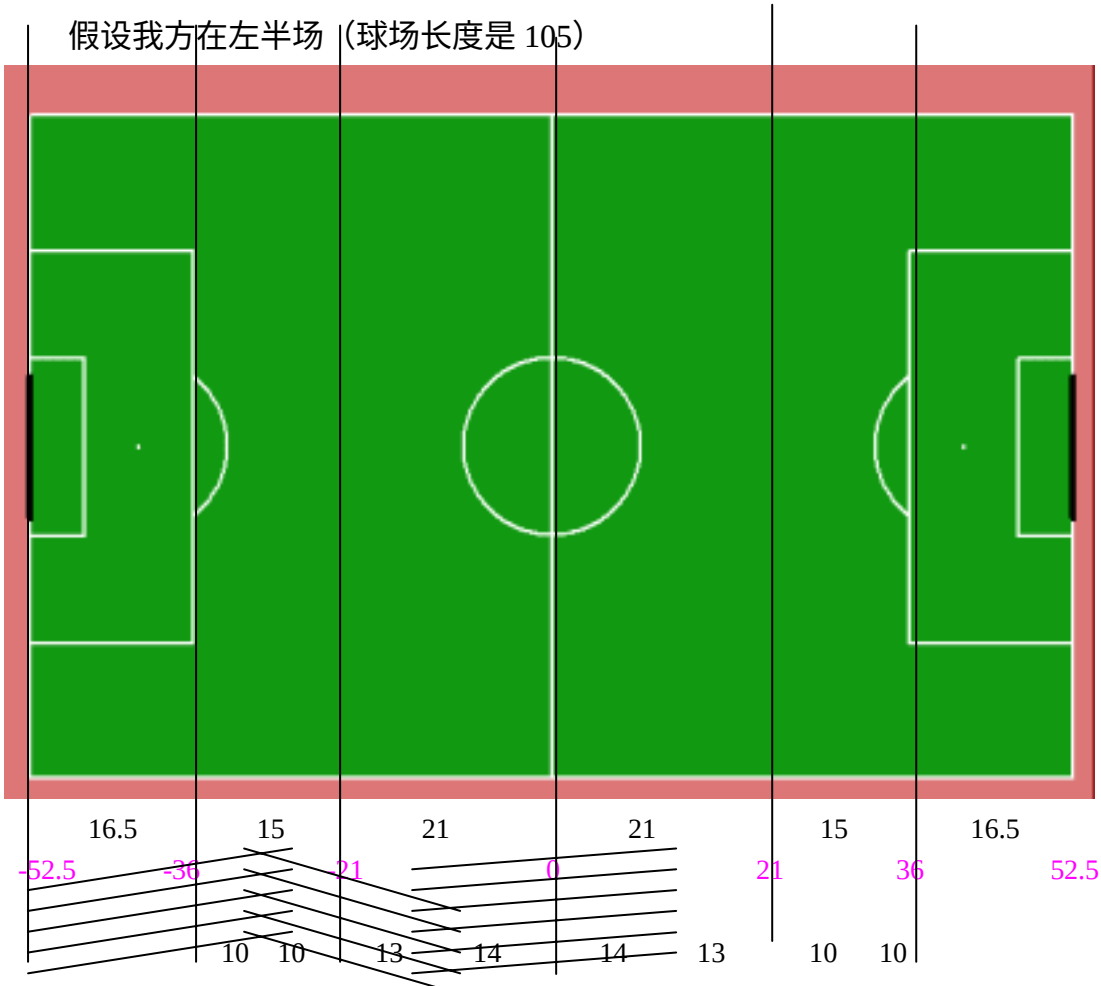
上动态形势的分析。当前只是简单的以空间位置来划分球员角色，并且指明在对应阵型下的球员应该在的策略站位点。对于已经确定的阵型，简单的划分位置靠前的相应几个球员为前锋，位置靠后的相应几个球员为后卫，中间剩下的就是中场。为了保证前锋线、中场线、后卫线这三条线的协调配合，我们还要确定这三线（也就是阵线）的大致位置范围。

6.2 阵线位置的确定

阵线位置直接关系到球员个体的场上站位。如果能够找到一种算法，能够将千变万化的场上形势抽象出一个场上模型出来，那么我们可以很轻松的根据这个场上模型来进行各种决策。到目前为止，还没有出现一种行之有效的算法。我们考虑的是和阵线有关的两个因素：足球位置和对方前锋位置。我方前锋的位置主要是由足球位置确定，而我方后卫则是考虑对方前锋而定，中场位置用来保证三条线之间的衔接。

为了更好的确定量度足球和对方前锋的位置，我们首先将球场模糊划分。

假设我方在左半场（球场长度是 105）



球场分成五个区域，相邻两个区域是重叠在一起的。如果不进行模糊划分，

相邻区域间有明确的分界线，那么一旦对象在分界线附近左右移动，会造成较大的振荡，影响球队性能。

- 足球在三区，即球场中央位置
  - ✧ 前锋线在四区活动，中场线在三区活动。
  - ✧ 如果在一区活动的对方球员人数超过了一个，那么后卫线在一区活动；否则，考虑造越位情况，后卫线于二区活动。
  - ✧ 如果足球在三区的停留时间超过一定限度，则前锋线后移，向三区靠拢。
- 足球在二区，即靠近我方球门的一边
  - ✧ 前锋线于三区活动，中场线于二区活动，后卫线在一区。
  - ✧ 如果足球在二区停留时间超过一定限度，则前锋线后移，向二区靠拢。
- 足球位于四区，即靠近对方球门处
  - ✧ 前锋线为五区，中场线为四区。
  - ✧ 如果在一区活动的对方球员人数超过了一个，那么后卫线在一区活动；否则，考虑造越位情况，后卫线于二区活动。
  - ✧ 如果足球在四区的停留时间超过一定限度，则后卫线前移。
- 足球位于一区，即我方罚球区附近
  - ✧ 前锋位于三区，中场位于二区，后卫位于一区。
  - ✧ 如果足球在一区的停留时间超过一定限度，则中场退后支援，同时前锋稍微退后，保证中场和前锋的衔接。
  - ✧ 如果足球在一区的停留时间更长，则中场线融入后卫线，前锋线退后支援。
- 足球位于五区，即对方罚球区附近
  - ✧ 前锋位于五区，中场位于四区。
  - ✧ 对方球员在一区二区的球员人数如果超过一个，则后卫在二区活动；否则后卫在三区活动。
  - ✧ 如果足球在五区的停留时间超过一定限度，则中场前移支援，同时后卫稍微前移，保证中场和后卫的衔接。
  - ✧ 如果足球在五区的停留时间更长，则中场线融入前锋线，后卫线前移支援。

6.3 攻防状态的切换

比赛状态是由球被何队所控制决定的，球进入我方队员脚下，则状态进入进攻状态，一旦进入对方控制，立即切换到防守状态；具体的所谓控制首先是判断控球队员，根据控球队员的队别来区分攻防状态；如果由于世界模型的原因，无

法获知控球队员，或者足球不为球员所控制，那么根据球队的控球度 flag 来决定攻防状态，flag＝1，切换到防守状态；而若 flag＝－1，则切换到进攻状态。下面描述攻防转换的一些情况和简单策略。

- 防守状态可由四种情况进入：
  - ✧ 队员在敌方后场丢球；
    辅助防守队员与主动防守队员全力回防，而前锋则全力进行回抢；
  - ✧ 队员在中场丢球；
    前锋回防中场，辅助防守负责前去抢球；主动防守队员回防；
  - ✧ 队员在我方后场丢球；
    辅助防守队员与主动防守队员全力拼抢；
- 进攻状态可由四种情况进入：
  - ✧ 我方开球、任意球、球门球、界外球、角球等；
    按照已经定义的来进行；
  - ✧ 队员在后防区域内截敌人的传球或断对方的带球；
    如果在此区域内的被对方队员再截回去的可能性大，或传球的成功率低，则将球踢出界外；带球基本不予考虑；
  - ✧ 队员在中场范围内得球；
    在敌方的后防区域断他们的失误球和回传球；

6.4 基于场上形势的动态站位和角色变换

这是顶层决策的关键所在。球员不仅能够改变它们的站位（由阵型决定），而且能在当前阵型下改变球员角色。站位变换仅在此变换效用是利于球队时进行。效用是根据以下因素确定的，球员位置到它们的战略位置的距离，每个站位的重要性和在当前形势下每个站位是否恰当。

站位的重要性是由阵型和足球位置决定的。如：一个只有两种策略（进攻和防守）的球队策略，会在防守状态下赋予中卫很高的站位重要性，而在进攻状态下，中锋的重要性就增加了。虽然空间上的协调能保证成功的进行站位变换，我们还通过通讯得到加强。成功的动态站位和角色变换可以覆盖场上更多的区域。因为场上一共才 11 个球员，不可避免的会出现空当、防守盲区。通过球员间的角色变换，场上跑位，可以覆盖场上更多的区域范围，同时也可以带动对方球员，拉开对方防线，为我方提供进攻机会。

基于场上形势的动态站位和角色变换是相当复杂也是非常有效的研究课题，本文并不打算详细讨论，仅仅是提出这个问题供大家探讨。

6.5 智能感知和通讯

在前面讨论的世界状态模型中，这个模型维持着场上对象的位置和速度值，每个数值对应一个可信度。当位置是被看到的，可信度还和球员得到的精确性有关，视觉可信度最大（在球员精确的看到对象情况下）就置为最大值（1.0），其他的则赋予一个随两者距离增加而减少的位置可信度，然后随着时间（以周期为单位）而降低。SoccerServer 的可视信息精度随间距而改变，由此反映在位置的可信度上。速度可信度也使用类似的推理方法。当球员要通过通讯共享它们的世界状态知识时，这种可信度方法是很有用的。

6.5.1 智能通讯机制

由于通讯是单声道，低带宽而且不可靠的，所以只有在从球队整体角度看有必要时才使用通讯。主要的问题在于什么时候传输什么通讯信息。通讯交流的目的是为了：

- 在适当的时刻通过共享个体的世界状态，来维持智能体的世界状态的更新和精确性；
- 通过交流部分决策来提高球队的协调性（如传球或站位更换）。

在通讯时，球员交流它们的完整的世界状态，以及一些高层决策事件。我们通过评价通讯的效用决定是否需要通讯。当一个球员看到一个对球队有用的事件后，他就考虑自己是否处于最利于发送此事件的位置，或者其他球员能更精确的描述这个事件。如果球员相信他的通讯比队友更加精确重要时，就执行通讯。如果没有什么有用的事件（如抛球，球速改变，传球等）被觉察到，球员就会关心他们世界状态的有用信息是否也被队友所知道。为了实现这种功能，智能体维持一个“通讯世界状态模型”。这个世界状态更新的信息来源仅仅使用听觉信息，而不使用任何感知或动作预测信息。球员使用这种通讯世界状态来作为是否需要与他们的队友进行知识交流的指示。

6.5.2 智能观察机制

在足球比赛这样的复杂领域中，必须使用一种智能的方法来使智能体维持一个精确的世界状态。然而，在不同的形势下，智能体对世界状态模型各部分的兴趣是不一样的。如果球员带球到对方球门边，那么对方守门员的位置和速度是世界状态中的一个很重要的部分，而如果球员在中场附近活动的话，对方守门员的情况就不那么重要了。

基于球员类型，阵型，球员位置，以及场上其他对象的位置和速度及可信度，智能化感知机制决定了智能体的观察角度。对于每个可能的观测方向，使用一个效用计算函数来估计智能体朝向的有效性。这个有效性依赖于场上形势，足球位置和自身的位置，球员对每个对象赋予不同的权重。然后选择一个最好的观测方

向，发送一个适当的转头指令。

6.6 小结

本章介绍了 Agent 的顶层策略。主要分析了阵型、角色和阵线，攻防状态的切换，基于场上形势的动态站位和角色变换、以及智能化的观察和通讯机制。

第七章　结论和展望

本篇论文详细介绍了计算机仿真足球机器人的设计与实现。首先介绍了 RoboCup 的研究背景，然后介绍了 SoccerServer 的仿真环境。从第三章开始介绍了具体的足球机器人 Agent 的设计和实现。Agent 的框架部分说明了开发环境，程序的基本框架。通讯层主要分析智能体获得信息后，怎么处理这些信息，更新智能体内部维持的世界模型。Agent 的个人技术部分则侧重阐述了程序设计中的难点。第六章分析了 Agent 的顶层策略，讲述了球队上层结构的设计，提出了 Agent 的智能化设计。例如基于场上形势的动态站位和角色变换，智能性的观察和通讯机制。

我们的球队命名为浙大闪电，他将参加今年 8 月份在云南昆明进行的中国 RoboCup2001 机器人足球锦标赛。希望他能获得好成绩，为校、系争光。

目前还有一些策略没有很好的实现，我们将在最后的几天时间内把整体球队的效果调整到最佳状态。同时我们希望能打败 2000 年的世界冠军 FC Portugal V2.0。今年 8 月份在西雅图举行世界 RoboCup 锦标赛，比赛结束后仍将发布冠军的程序，我们也将和它进行比赛测试自己的水平。最后希望浙大能参加明年的国际 RoboCup 锦标赛，为国争光。

参考文献

- ✧ Peter Stone，Layered Learning in Multi-Agent Systems，Carnegie Mellon University，1998。
- ✧ 李实、徐旭明、叶榛、孙增圻编著，《综述：国际机器人足球比赛及其相关技术》，清华大学计算机系国家智能技术与系统重点实验室，1999 年。
- ✧ 孙增圻、李实编著，《RoboCup 与智能自动化》，清华大学计算机科学与技术系，1999 年。
- ✧ Ehsan Foroughi & Fredrik Heintz & Spiros Kapetanakis & Kostas Kostiadis & Johan Kummeneje & Itsuki Noda & Oliver Obst & Pat Riley & Timo Ste_ens & USTC9811 Group，Users Manual RoboCup Soccer Server，2001。
- ✧ Luís Paulo Reis & Nuno Lau，FC Portugal Team Description: RoboCup 2000 Simulation League Champion，http://www.ieeta.pt/robocup，2000。
- ✧ Peter Stone & Manuela Veloso & Patrick Riley，The CMUnited-98 Champion Simulator Team，http://www.andrew.cmu.edu/~priley，1998。

致　谢

就要毕业了。

本科毕业设计的时间只是短短的半年；在这半年里，以及在此之前在浙大度过的四年美好时光里，难以忘怀的是师长、同学和亲友们无尽的关心、鼓励、支持和帮助。尤其是实验室的老师和同学在毕业设计期间对我的帮助是我完成毕业设计所必不可少的。特别感谢熊蓉老师和杨晗、彭司华、于江涛博士，沈华品、顾尔丹硕士。

在此唯有深表谢意。